ROGÉRIO APARECIDO GONÇALVES

PowerSMT: FERRAMENTA PARA ANÁLISE DE CONSUMO DE POTÊNCIA EM ARQUITETURAS SMT

MARINGÁ

2008

Livros Grátis

http://www.livrosgratis.com.br

Milhares de livros grátis para download.

ROGÉRIO APARECIDO GONÇALVES

PowerSMT: FERRAMENTA PARA ANÁLISE DE CONSUMO DE POTÊNCIA EM ARQUITETURAS SMT

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual de Maringá, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Ronaldo Augusto de Lara Gonçalves

MARINGÁ

2008

Dados Internacionais de Catalogação-na-Publicação (CIP)
(Biblioteca Central - UEM, Maringá – PR., Brasil)

G635p

Gonçalves, Rogério Aparecido
PowerSMT : ferramenta para análise de consumo de potência em arquiteturas SMT / Rogério Aparecido Gonçalves. -- Maringá : [s.n.], 2008.
158 p. : il., figs., tabs.

Orientador : Prof. Dr. Ronaldo Augusto de Lara Gonçalves.
Dissertação (mestrado) - Universidade Estadual de Maringá, Programa de Pós-graduação em Ciência da Computação, 2008.

1. Arquiteturas de Múltiplas Tarefas Simultâneas. 2. Consumo de potência. 3. SimpleScalar. 4. Wattch. 5. CACTI. I. Universidade Estadual de Maringá, Programa de Pós-graduação em Ciência da Computação. II. Título.

CDD 21.ed. 004.2

ROGÉRIO APARECIDO GONÇALVES

PowerSMT: FERRAMENTA PARA ANÁLISE DE CONSUMO DE POTÊNCIA EM ARQUITETURAS SMT

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual de Maringá, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Aprovado em 14/07/2008.

BANCA EXAMINADORA

Prof. Dr. Ronaldo Augusto de Lara Gonçalves
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. João Angelo Martini
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Márcio Seiji Oyamada
Universidade Estadual do Oeste do Paraná – INF/UNIOESTE

“Faça somente o que somente você pode fazer”
-- DIJKSTRA

À minha mãe
Maria pelo incentivo ininterrupto e por sempre
acreditar no meu esforço e em minha capacidade.

Agradecimentos

Agradeço primeiramente a Deus, pois Ele é quem nos permite desenvolver qualquer ponto de nossas vidas, iluminando os caminhos e as mentes, possibilitando as escolhas mais certas.

Agradeço a todos os professores do Programa de Pós-Graduação em Ciência da Computação (PCC) por terem colaborado para o cumprimento de mais esta etapa em minha vida.

Registro aqui o meu agradecimento especial e minha gratidão ao Professor Ronaldo Augusto de Lara Gonçalves pela orientação, pelas conversas e por sua amizade. Da mesma forma agradeço ao Professor João Angelo Martini, pelas contribuições, pelas discussões esclarecedoras e pela disposição de sempre ajudar.

Agradeço aos Amigos da turma do Mestrado, pela amizade que conquistamos juntos, e pelo incentivo que cada um sempre demonstrou.

Agradeço ainda ao Grupo HPPCA e à direção dos laboratórios LECAD/LEAL pelo suporte e apoio, e aos Amigos e companheiros de trabalho no laboratório pelas conversas e contribuições.

Em especial agradeço à Maria Inês Davanço, a Inês secretária do PCC, pela competência, pela dedicação, pela atenção e pela amizade com que sempre atendeu a todos os alunos do Programa.

E por fim agradeço a CAPES pelo suporte financeiro concedido, e a todas as pessoas que direta ou indiretamente contribuíram para a realização deste trabalho.

Sumário

Listas de Ilustrações

Lista de Tabelas

Lista de Quadros

Lista de Símbolos e Abreviaturas

CACTI	*Cache Timing, Power, and Area Model*
CMOS	Complementary Metal Oxide Semiconductor
DVFS	*Dynamic Voltage and Frequency Scaling* – Escala Dinâmica de Voltagem e Freqüência
HDL	*Hardware Description Language* – Linguagem de Descrição de Hardware
HPPCA	*High Performance and Parallel Computer Architecture* – Arquiteturas de Computadores Paralelos e de Alto Desempenho
ILP	*Instruction Level Parallelism* – Paralelismo no nível de instruções
LEAL	Laboratório de Engenharia de Algoritmos
LECAD	Laboratório Experimental de Computação de Alto Desempenho
LSQ	*Load/Store Queue* – Fila de Load/Store
PCC	Programa de Pós Graduação em Ciência da Computação
RAW	*Read after write* - Leitura após escrita
RUU	*Register Update Unit* – Unidade de Atualização de Registradores
SCM	*Software Code Morphing*
SMT	*Simultaneous Multithreading* – Múltiplas Tarefas Simultâneas
TLP	*Thread Level Parallelism* – Paralelismo no nível de tarefas
WAR	*Write after Read* - Escrita após leitura
WAW	*Write after write* - Escrita após escrita

Resumo

A evolução das aplicações tem exigido um processamento cada vez mais intenso, o qual tem sido obtido por meio de melhorias arquiteturais, tais como o uso de processamento pipeline, superescalar e execução simultânea de tarefas (SMT). A crescente complexidade das aplicações leva naturalmente à necessidade do aumento da complexidade do hardware, para que este suporte a execução e garanta o melhor desempenho possível. As estruturas necessárias ao processamento e toda a parte de controle exigida ocupam espaço no chip e proporcionam um aumento no consumo de potência, que nos processadores atuais chega a níveis elevados e acarreta problemas. Neste contexto, surgem desafios relacionados ao controle do consumo, que levam ao desenvolvimento de melhorias nos circuitos dos processadores de forma a consumirem potência eficientemente. Para a avaliação de desempenho das arquiteturas de processadores, ferramentas baseadas no SimpleScalar têm sido usadas satisfatoriamente. Para o estudo do consumo de potência nos diversos componentes arquiteturais do processador as ferramentas Wattch e CACTI assumem um papel em igual escala de importância. A proposta trazida por este trabalho está inserida neste contexto, tendo como objetivo a análise do consumo de potência em arquiteturas SMT, haja vista a importância comercial de tais arquiteturas na atualidade. A maior intensidade de utilização de recursos de hardware nesse tipo de arquitetura sugere um consumo maior que precisa ser estudado, demandando a elaboração de estratégias de controle e otimização para a utilização e o compartilhamento de recursos visando a redução do consumo de potência sem prejuízos ao desempenho. Para satisfazer os objetivos de simulação e estudo e por meio deles proporcionar auxílio ao desenvolvimento de técnicas de controle, um simulador chamado PowerSMT foi desenvolvido sobre as plataformas SimpleScalar, Wattch e CACTI. O mesmo foi utilizado na análise do consumo de potência em diferentes aspectos das arquiteturas SMT e mostrou ser uma ferramenta de fundamental importância naquilo que se propõe.

Palavras-chave: Arquiteturas de Múltiplas Tarefas Simultâneas, Consumo de Potência, SimpleScalar, Wattch, CACTI.

Abstract

The evolution of the applications has required an intense processing and it has been obtained though architectural improvements such as the use of pipeline architecture, superescalars and simultaneous multithreading (SMT). The increasing complexity of the applications naturally leads to the need of increasing the complexity of the hardware. In this way, it'll be able to support the implementation and ensure the best performance possible. The structures necessary for the processing and all the controls required take up space in the chip and generate an increase in the power consumption. In current processors it reaches high levels causing serious problems. In this context, there are some challenges related to the control of the consumption that lead to the development of improvements in the circuits of the processors in order to use power efficiently. For the evaluation of the performance of the processor architectures, some tools, based on SimpleScalar plataform, has been used satisfactorily. For the power consumption study in the several architectural processor components, the tools Wattch and CACTI have the same scale of importance. Therefore, the present work aims to analyze the power consumption in SMT architectures, since there is a commercial importance of such architectures in the present. The highest intensity of hardware resources used in this sort of architecture suggests that a higher consumption needs to be studied and it demands the development of strategies of control and optimization to the use and sharing of the resources aiming the reduction of the power consumption without damaging the performance. To reach this simulation and study goals and through them provide aid to the development of control techniques, a simulator called PowerSMT was developed on the SimpleScalar, Wattch and CACTI platforms. It was used in the analysis of power consumption in different aspects of SMT architectures and it proved itself to be a tool of fundamental importance in what it is supposed to do.

Keywords: Simultaneous Multithreading Architecture, Power Consumption, SimpleScalar, Wattch, CACTI.

Capítulo 1
Introdução

A evolução das aplicações promove o desenvolvimento de melhorias nos processadores de forma que estes possam responder à carga de trabalho imposta pelos novos contextos e aplicações. O provimento de um processamento mais eficiente é necessário para atender a essa demanda computacional criada e intensificar a utilização de recursos existentes, sendo assim, a proposição de novos modelos e estratégias tem sido alvo de intensas pesquisas. A história registra os principais modelos arquiteturais que levaram à obtenção de melhorias no desempenho, entre eles, a arquitetura *pipeline*, arquiteturas superescalares, arquiteturas de múltiplas tarefas simultâneas (SMT), arquiteturas de múltiplos núcleos e de múltiplos núcleos com suporte a SMT, sendo estas três últimas exploradas correntemente e com perspectivas de crescimento.

Com a utilização da técnica de *pipeline* é possível decompor o processamento das instruções em estágios, para a obtenção de paralelismo parcial de instruções nas fases de busca, decodificação e despacho de instruções, conforme a profundidade do *pipeline*. Já as arquiteturas superescalares expandiram a capacidade em largura através de *pipelines* distintos para cada unidade funcional, fazendo com que múltiplas instruções sejam buscadas, decodificadas e despachadas para a execução fisicamente paralela. Mesmo as arquiteturas superescalares, tendo como objetivo principal a exploração do paralelismo no nível de instruções, ainda não atingem níveis de paralelismo desejáveis, devido às características das aplicações que levam a dependências de dados e de recursos. Com isso, as unidades funcionais e outros recursos arquiteturais são utilizados ineficientemente.

Diante da necessidade de suprir a demanda por mais desempenho e de utilizar mais eficientemente os recursos, surge a idéia de execução simultânea de instruções de tarefas distintas

com o objetivo de maximizar a utilização de recursos, uma vez que os recursos não explorados por uma tarefa podem ser explorados por outras. Este tipo de arquitetura foi chamada de SMT (*Simultaneous Multithreading* – Múltiplas Tarefas Simultâneas) e foi implementada em processadores comerciais. Entretanto, sua existência foi inicialmente sufocada pelas arquiteturas de múltiplos núcleos físicos, que passaram a ter maior destaque. Com o avanço das pesquisas e da tecnologia de fabricação, a abordagem SMT ressurgiu associada à multiplicidade de núcleos físicos. Neste caso, cada núcleo físico é visto como dois ou mais núcleos lógicos, aumentando a capacidade de processamento através da exploração do paralelismo no nível das tarefas executadas em cada núcleo em particular.

Infelizmente, essa evolução arquitetural está diretamente relacionada ao aumento da complexidade do *hardware* necessário para suportar a demanda da computação exigida. Os recursos de armazenamento temporário e de lógica de controle, exigidos para a manutenção e gerenciamento dos diversos contextos, ocupam maior espaço no *chip* e proporcionam um aumento no consumo de potência, que pode atingir níveis prejudiciais aos componentes internos.

Seja para aumentar a autonomia de dispositivos móveis ou embarcados, reduzir o aquecimento, disponibilizar maior área do *chip*, prover a estabilidade dos componentes ou ainda para melhorar o desempenho, o controle do consumo de potência passa a ter uma importância fundamental nos projetos de novos processadores. É preciso alavancar as pesquisas nessa área e formular propostas que enfoquem a redução do consumo de potência em todos os níveis sistêmicos, envolvendo o hardware, sistema operacional, aplicações e demais componentes que integram os sistemas computacionais.

Neste contexto, surgem ferramentas relacionadas à avaliação de consumo de potência, tais como CACTI (JOUPPI e WILTON, 1994), Sim-Wattch (BROOKS *et al*, 2000), PowerTimer (BROOKS *et al*, 2003) (HU *et al*, 2003) (BROOKS *et al*, 2004), PTSMT (KANNAN *et al*, 2008) e PowerSMT, desenvolvida neste trabalho, as quais permitem prever consumo em tempo de projeto, de forma rápida e eficiente, possibilitando desenvolver circuitos com melhor custo/benefício entre consumo e desempenho. Essas ferramentas de simulação são comercialmente fundamentais, pois o processo de concepção de circuitos até a fabricação do *chip* é caro e as avaliações pós-fabricação não são viáveis.

O simulador Sim-Wattch (BROOKS *et al*, 2000) realiza medições por meio de contagem de acessos às diversas estruturas do processador, ciclo-a-ciclo, acumulando os consumos de potência para esses componentes arquiteturais. Este simulador foi desenvolvido para avaliação de desempenho dos componentes arquiteturais sobre a plataforma do SimpleScalar (BURGER e

AUSTIN, 1997), que fornece um conjunto de ferramentas que possibilitam simular e avaliar as arquiteturas superescalares, além disso, tem sido utilizado como base para extensões que dão origem a novas ferramentas.

Como a densidade das áreas ocupadas nos *chips* tende a aumentar, a dissipação de potência é prevista como um fator chave limitante no desempenho de processadores. Os processadores estão alcançando os limites convencionais de técnicas de resfriamento, em adição às dificuldades de alcance do sinal de *clock* e de alimentação sobre o *chip* de área extensa.

Este trabalho está inserido neste contexto, tendo como foco a investigação do consumo de potência em arquiteturas SMT. A preocupação está no fato das arquiteturas SMT terem uma utilização mais intensiva dos recursos e isto sugere um consumo de potência maior que precisa ser medido, estudado e avaliado. Com a investigação e estudo, será possível elaborar estratégias de controle e otimização para a utilização, a distribuição e o compartilhamento de recursos visando a redução do consumo de potência sem prejuízos ao desempenho desejado. Para prover meios de realizar esta investigação, o simulador PowerSMT foi desenvolvido, experimentado e é aqui apresentado, cujos resultados mostram a sua flexibilidade de configuração e pontos arquiteturais que podem ser avaliados e sua importância enquanto ferramenta de simulação e avaliação de desempenho e consumo.

A presente dissertação está organizada da seguinte forma. No capítulo 2 são apresentados os conceitos e as características relacionadas às arquiteturas SMT, contextualizando o estado da arte e exemplificando com um caso real. Questões e técnicas visando a redução do consumo de potência são apresentadas no capítulo 3. No capítulo 4 são apresentadas as ferramentas relacionadas com o tema deste trabalho, incluindo aquelas que serviram como base para o desenvolvimento do PowerSMT. O capítulo 5 apresenta o PowerSMT, suas características e questões sobre o seu desenvolvimento. No capítulo 6 são descritos os experimentos realizados e os resultados obtidos. No capítulo final são apresentadas as conclusões, e por fim as referências bibliográficas utilizadas. Os anexos incluem informações adicionais e detalhadas, pertinentes ao desenvolvimento deste trabalho.

Capítulo 2

Arquiteturas de Múltiplas Tarefas Simultâneas

A crescente aplicabilidade dos computadores e a multiplicidade dos contextos aos quais estas aplicações estão sediadas, garantem a manutenção da necessidade por melhorias nas condições e estratégias de processamento, bem como no desempenho alcançado pela utilização das mesmas. Ficando a cargo das máquinas a obrigação de responderem à carga de trabalho imposta pelo volume crescente de informações e a exigência de intensificação do processamento. A busca pelo desempenho passou historicamente por alguns modelos que incorporaram melhorias nas arquiteturas, sendo as arquiteturas superescalares um dos modelos principais.

As arquiteturas superescalares expandiram a capacidade em largura através de *pipelines* distintos para cada unidade funcional, fazendo com que múltiplas instruções sejam buscadas, decodificadas e despachadas para a execução. Mesmo as arquiteturas superescalares, ainda não atingem níveis de paralelismo desejados e acabam sendo subutilizadas ou utilizadas ineficientemente.

Diante da necessidade de melhoria no desempenho e da utilização mais eficiente dos recursos, surgiu a idéia de remessa e execução de instruções de tarefas distintas, caracterizando a execução simultânea de tarefas com o objetivo de intensificar a utilização de recursos e explorar o grau de paralelismo existente entre essas instruções e ainda o grau de paralelismo existente entre tarefas. As seções seguintes detalham os conceitos relacionados às arquiteturas superescalares e às arquiteturas de múltiplas tarefas simultâneas (SMT).

2.1 Arquiteturas Superescalares

O conceito de superescalar é implementado na maioria dos processadores modernos, dando-lhes a

capacidade de explorar o paralelismo no nível de instruções (ILP). Em uma arquitetura superescalar típica, o processador busca e decodifica várias instruções tentando garantir um fluxo contínuo de instruções prevendo antecipadamente os resultados de instruções de desvio. As instruções são analisadas e as dependências de dados e de controle são reduzidas, e instruções são despachadas e distribuídas para as estações de reserva associadas e na seqüência são emitidas para a execução em suas respectivas unidades funcionais (SMITH e SOHI, 1995). A Figura 2.1, apresenta uma possível microarquitetura de um processador superescalar.

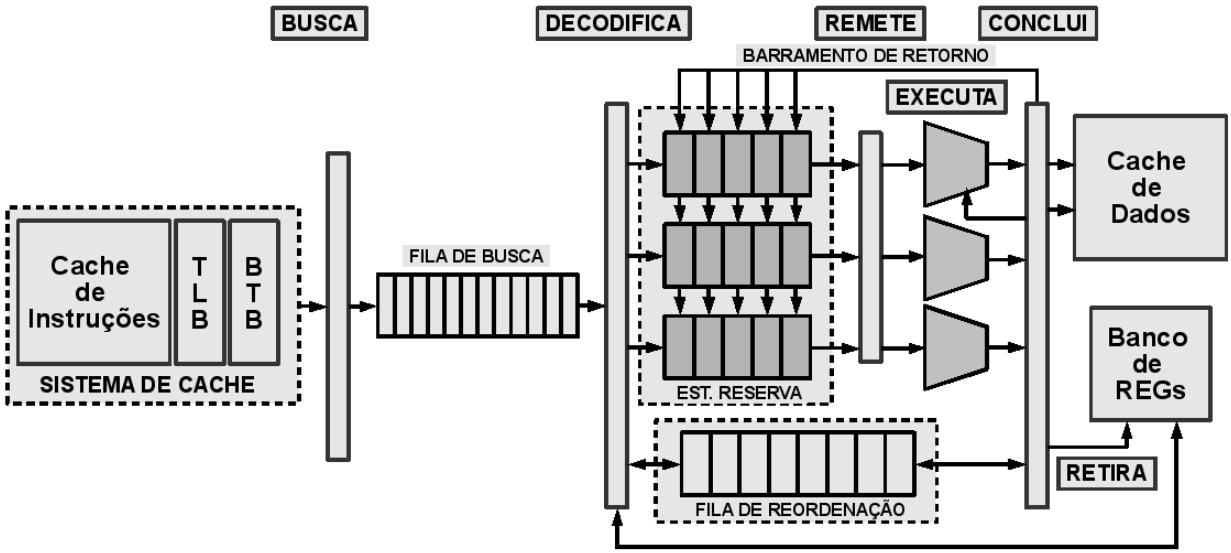

Figura 2.1: Possível Organização Superescalar

A execução em paralelo baseia-se na disponibilidade de operandos ao invés de restringir-se à seqüência original do programa, proporcionando um escalonamento dinâmico de instruções, no qual os operandos pendentes em outras instruções são atualizados à medida em que são calculados, permitindo que a execução das instruções que aguardam esses operandos possa então prosseguir. A arquitetura utiliza-se de uma estrutura de reordenação, um *buffer* ou uma fila, para que após a execução e obtenção dos resultados, o estado do programa seja finalizado na mesma ordem estabelecida pelo código, mantendo assim a semântica do código original.

A execução fora de ordem é obtida sobre as instruções que estão com os operandos prontos, não respeitando a ordem original do código, mas sim o fluxo de dados. Esse paralelismo é implícito no código e ocorre de forma transparente ao usuário, não sendo necessário recompilar as aplicações existentes para a arquitetura alvo.

A execução de instruções em paralelo envolve questões relacionadas à dependência de controle e de dados entre as instruções, ao hardware para suportar a execução de múltiplas instruções em paralelo, às estratégias para determinar quando uma instrução está pronta e à

passagem de valores entre instruções que estão em execução.

Instruções de desvio condicional podem levar o fluxo de execução a um caminho tomado pela decisão das técnicas de previsão de desvios, de forma que as instruções executadas desse caminho são executadas em um contexto especulativo, podendo mais tarde serem descartadas ou efetivadas, dependendo da conformidade da previsão. As operações realizadas especulativamente são efetivadas no estado real da máquina somente quando as previsões de desvios são confirmadas. O controle da visibilidade desse novo estado e da manutenção da aparência de execução seqüencial sem prejuízos para a semântica do código original é uma tarefa complexa, exigindo para isso um hardware também complexo.

As técnicas de previsão de desvio tentam prever o caminho a ser seguido pelo fluxo de execução, procurando minimizar os erros de forma que sempre instruções de um caminho correto sejam buscadas, decodificadas e executadas. Envolvem algoritmos utilizados para predizer qual será o caminho tomado por uma instrução de desvio de forma que a taxa de acerto seja a melhor possível, pois o erro na predição leva a situações tais como o esvaziamento do *pipeline* para buscar as instruções do caminho correto e a má composição da cache de instruções por instruções que nem sequer deveriam ter sido buscadas. Este esvaziamento e o reinício da busca no novo caminho tem um custo de consumo de potência e uma penalidade ao desempenho, pois todo acesso aos blocos do sistema de hierarquia de memória para o caminho incorreto é desnecessário.

A dependência de dados existe entre as instruções que fazem operações de leitura ou escrita de uma localização de armazenamento, seja registrador ou memória. Essa referência a um mesmo local por múltiplas instruções exige que a ordem entre a execução das operações seja preservada para que resultados incorretos não sejam lidos ou produzidos. Essa ordem obrigatória entre a execução dessas instruções caracteriza uma dependência verdadeira entre elas, que deve ser controlada. Essas dependências podem ser de leitura após escrita (RAW), escrita após leitura (WAR) e escrita após escrita (WAW).

Para realizar o controle da ordem de execução das instruções, os dados que resultam das execuções não podem ser refletidos imediatamente no estado arquitetural. Esses dados são armazenados como um estado temporário e após a confirmação de que o resultado esteja correto, o estado temporário é efetivado para os registradores físicos.

2.2 Arquiteturas SMT

A denominação *Simultaneous Multithreading* (SMT) teve origem na publicação de (TULLSEN *et al*, 1995), embora o conceito tenha sido apresentado por idéias semelhantes em trabalhos anteriores

como a arquitetura *Simultaneous Instruction Issuing* (HIRATA et al, 1992) e a arquitetura chamada *Multistreamed Superscalar* (YAMAMOTO e NEMIROVSKY, 1994), ambos com a capacidade de executar instruções provenientes de diferentes fluxos simultaneamente.

As arquiteturas SMT têm como objetivo a execução simultânea de tarefas, remetendo à execução instruções de diferentes tarefas durante o mesmo ciclo do processador. Cada tarefa detém, durante a execução, o uso de alguns recursos que constituem o seu contexto de execução, compartilhando diversos outros com outras tarefas em execução. O contexto de cada tarefa é formado pelo conteúdo dos registradores que estão sendo utilizados, pelo seu contador de programa e pelos recursos que estão sob seu domínio.

Através do embaralhamento das instruções das diversas tarefas em execução simultânea, a arquitetura explora o paralelismo em nível de tarefas (TLP), em adição ao ILP, o qual é típico das arquiteturas superescalares convencionais. Esta característica proporciona melhor aproveitamento dos recursos existentes, pois aqueles que estariam ociosos com a execução de uma única tarefa poderão ser aproveitados pela execução simultânea de outras tarefas.

A execução de tarefas simultâneas garante um paralelismo maior em relação à execução convencional, permitindo uma exploração maior dos recursos do *hardware*. A unidade de busca, entre outros recursos é compartilhada pelas diversas tarefas, permite que as instruções das tarefas sejam buscadas e inseridas no caminho de execução, onde a coexistência de vários contextos permite aproveitar melhor as oportunidades de execução enquanto as tarefas concorrentes alternam seus momentos de *cpu-bound* e *io-bound*.

O compartilhamento de recursos e estruturas internas do processador pode ser feito de diversas maneiras e da forma que melhor se adequar ao projeto arquitetural. Por exemplo, a implementação da Intel, denominada tecnologia *Hyper-Threading*, implementa um *pipeline* para a execução simultânea de duas tarefas através da duplicação do estado arquitetural, como sendo dois processadores lógicos, porém a execução das instruções é simultânea sobre os recursos compartilhados do mesmo processador físico (MARR *et al*, 2002), sendo vistos pelo SO como sendo dois processadores físicos. A Figura 2.2 apresenta a idéia geral da utilização de recursos na execução em um núcleo SMT, fazendo um comparativo com a utilização dos recursos na execução em um processador de núcleo simples e em um processador de núcleo duplo. Os recursos são representados pelos quadrados, as linhas representam os recursos que podem ser utilizados em paralelo e as colunas mostram a utilização efetiva de cada recurso no decorrer do tempo.

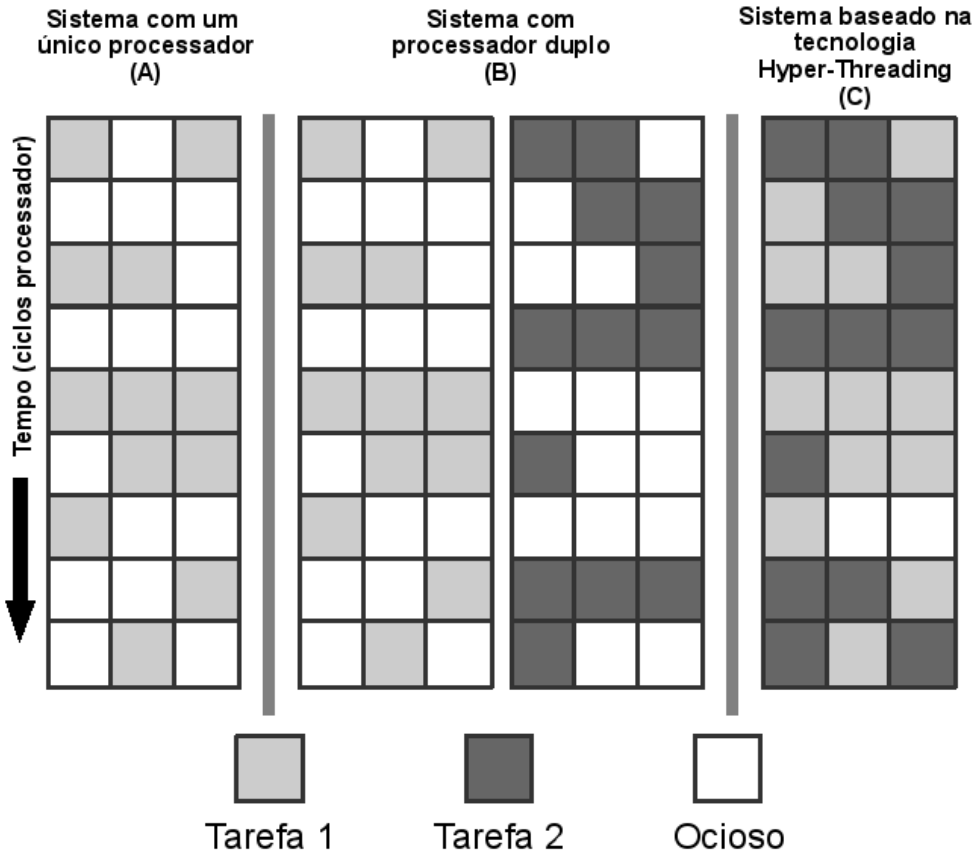

Figura 2.2: Comparativo de execução por modelos arquiteturais

Percebe-se na figura 2.2 a idéia da sub-utilização de recursos, característica dos núcleos de execução de tarefa única, e a forma de aproveitamento desses recursos em uma utilização intensificada pela exploração do paralelismo em nível de tarefas, além do paralelismo em nível de instruções. De acordo com (TULLSEN *et al*, 1995) o desperdício de recursos nos *slots,* causado pela não utilização deles na execução, pode ser classificado de duas formas, sendo elas: o desperdício (*waste*) horizontal (quando nem todos os recursos são utilizados) e desperdício vertical (quando todos os recursos no *slot* estão ociosos). Uma representação desses tipos de desperdício é apresentada na figura 2.3.

A tecnologia *Hyper-Threading* (HT) executa duas tarefas simultaneamente, enquanto que na proposta inicial do conceito SMT (TULLSEN *et al*, 1995) foram realizados experimentos com até 8 tarefas, o que leva a crer que é alta a complexidade envolvida no controle e no compartilhamento de recursos.

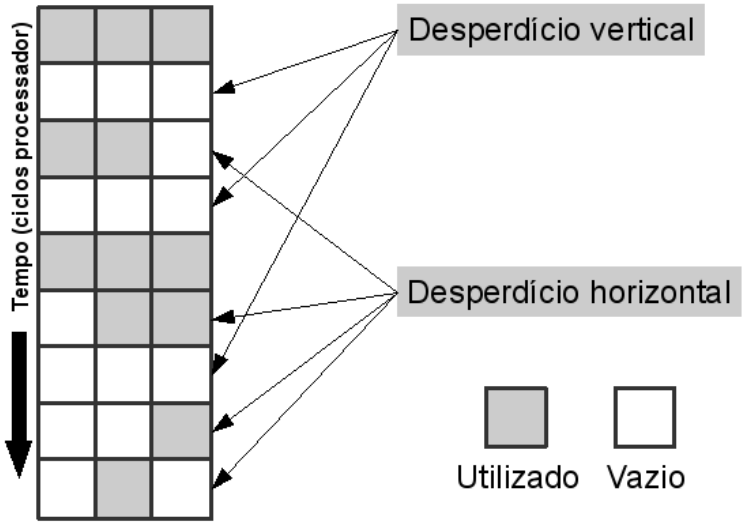

Figura 2.3: Tipos de desperdício na ocupação de recursos

A Figura 2.4 apresenta alguns estilos de implementações para arquiteturas SMT. Pode-se notar a variabilidade de configurações relacionadas à organização de cada estágio.

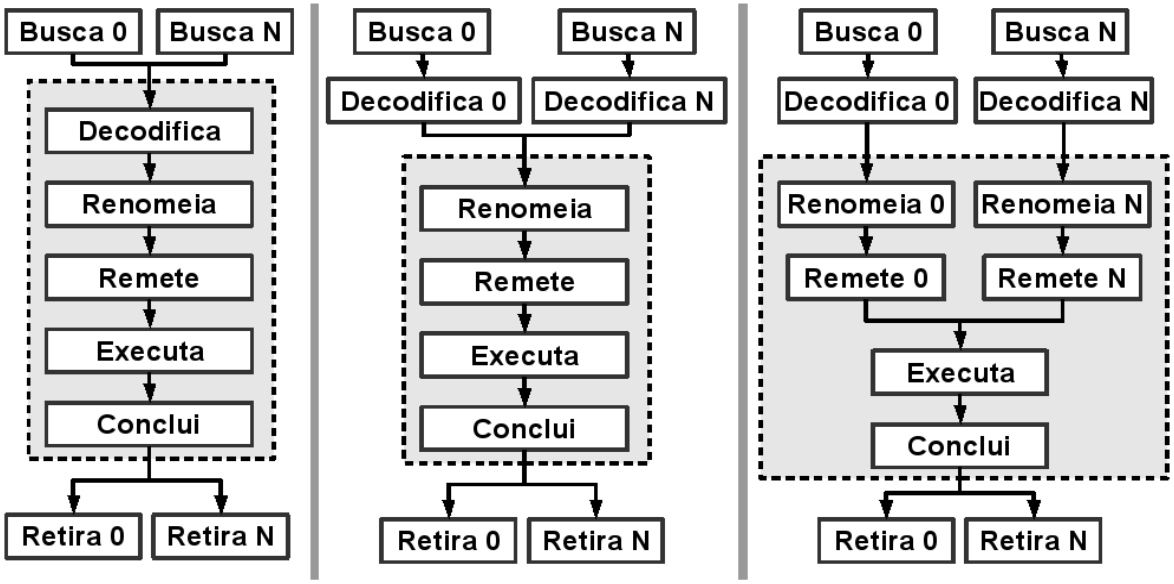

Figura 2.4: Estilos de implementações de arquiteturas SMT

Na Figura 2.4 pode-se notar que as fases de busca e de retirada de instruções são separadas para garantir a privacidade das tarefas, pois são as fases nas quais são acessados os respectivos códigos e estruturas contextuais particulares, tais como registradores e área de dados. As outras fases podem ser compartilhadas ou distrituídas conforme as necessidades do projeto arquitetural, mas existe uma gama bem maior de possibilidades que as apresentadas na figura 2.4.

2.2.1 Particionamento e distribuição de recursos

Em um ambiente de compartilhamento de recursos faz-se necessário políticas para a alocação de

tais recursos pelas tarefas em execução. A alocação e a distribuição estática dos recursos disponíveis podem ser vistas como uma divisão simples do total de recursos disponível pelo total de tarefas em execução. Desta forma cada tarefa teria a mesma quantidade de recursos independente do seu tamanho ou da carga de trabalho exigida para a sua execução. Esta seria a idéia mais básica e natural dentro do contexto inicial sobre o projeto de arquiteturas SMT.

Com a inserção, em um segundo momento, do conceito de compartilhamento dos recursos, a alocação e a distribuição dinâmica desses recursos surgem neste contexto como sendo possibilidades de melhoria de desempenho.

O modelo arquitetural proposto por (GONÇALVES, 2000) utiliza-se fundamentalmente do modelo estático de distribuição dos recursos, porém algumas estruturas podem ser compartilhadas dinamicamente. Os recursos são organizados em *slots*, sendo que cada *slot* possui a mesma quantidade de recursos, e estes recursos serão privativos à tarefa que ocupar o *slot*. Os recursos compartilhados são controlados de forma que cada tarefa possa acessá-los de maneira igualitária e de acordo com a disponibilidade. Cada tarefa em execução possui um conjunto de recursos em seu *slot* que detém durante toda a sua execução. Este modelo arquitetural pode ser simulado na ferramenta SS_SMT (GONÇALVES *et al,* 2000) (GONÇALVES *et al,* 2001), que será apresentada no capítulo 4, Ferramentas Relacionadas.

A organização em *slots* impede que os recursos de uma tarefa sejam acessados por outra, e o controle sobre os recursos compartilhados impede que uma tarefa domine a maior parte dos recursos, fazendo com que as outras fiquem esperando pela liberação. A distribuição não considera as características das tarefas, tal como a carga de processamento exigida por cada uma, não considera o tamanho ou quaisquer outras métricas, fazendo sempre com que os subconjuntos sejam idênticos independente das características das tarefas, possibilitando a ocorrência de casos nos quais tarefas mais leves detenham recursos que ficam ociosos enquanto outras tarefas enfrentam a escassez de recursos dentro do seu sub-conjunto.

O particionamento estático dos recursos ocorre também na tecnologia HT (MARR *et al*, 2002), a qual possibilita a execução de duas tarefas, cada uma detém desde o início um conjunto de recursos que formam seus respectivos contextos.

Uma alternativa ao particionamento estático de recursos foi apresentada por (CAZORLA *et al*, 2004), sugerindo a existência de um *pool* de recursos comum a todas as tarefas, ficando a critério da política assumida pelo estágio de busca a determinação de quais recursos serão compartilhados, quais tarefas entram ou saem do caminho de execução.

Em (CAZORLA *et al*, 2004) a Alocação de Recursos Controlada Dinamicamente (DCRA) classifica as tarefas em duas categorias (rápidas e lentas) de acordo com a quantidade de recursos que elas necessitam para serem executadas. Inicialmente todas as tarefas recebem a mesma quantidade de recursos, porém na dinâmica da execução as tarefas rápidas emprestam os recursos excedentes à sua carga de trabalho, que ficariam ociosos, para as tarefas lentas. O que possibilita o empréstimo é o fato de que as tarefas rápidas, mesmo com menos recursos, mantêm o desempenho individual de quando com a quantidade atribuída inicialmente e as tarefas lentas com os recursos adicionais melhoram o seu desempenho.

2.2.2 Políticas de Busca

O funcionamento da unidade de busca pode ser feito de várias formas, podendo-se citar os algoritmos avaliados por (TULLSEN *et al*, 1996). O algoritmo de prioridade *round-robin,* que pode ser considerado a forma mais básica, é utilizado na implementação da ferramenta SS_SMT de (GONÇALVES *et al,* 2000), permite que as instruções sejam buscadas das tarefas alternadamente na forma de rodízio.

Outros algoritmos baseados em informações particulares da própria tarefa foram avaliados por (TULLSEN *et al*, 1996) permitindo que a busca de instruções utilize um política que identifique qual a tarefa que pode tirar o melhor aproveitamento dos recursos e prover o melhor desempenho ao sistema. Informações como o número de instruções (*ICount)*, o número de desvios (*BRCount*) e a taxa de falta das caches (*MISSCount*) podem ser usadas.

O algoritmo *ICount* prioriza as tarefas com poucas instruções, apresentando bons resultados para tarefas com ILP alto. Em um cenário com tarefas que apresentam taxas de falta na *cache* de nível 2, recursos compartilhados podem ser monopolizados pelo fato do *ICount* não perceber que tarefas podem estar bloqueadas por muitos ciclos sem prosseguir no fluxo de execução. Segundo o autor isso ocorre mais gravemente quando se busca instruções de até duas tarefas por ciclo.

O algoritmo *BrCount* dá prioridade para as tarefas com menor número de instruções de desvios, o algoritmo avalia as instruções que estão da fila de instruções e na renomeação, favorecendo as tarefas com menor possibilidade de estarem executando instruções de um caminho tomado por uma decisão de previsão errada sobre qual caminho seguir. Esse algoritmo evita ainda que a largura de busca e os recursos de busca sejam consumidos na aquisição de instruções de um caminho errado.

O *MissCount* é uma política que prioriza as tarefas que têm as menores taxas de falta na cache de dados, evitando que os recursos da arquitetura fiquem ocupados por instruções que

aguardam a conclusão de um *load* da memória.

Entre as políticas avaliadas, a que produziu melhor resultado foi a *Icount*. As políticas não proveram mais que 4% de melhora em relação ao algoritmo *round-robin*.

Outras políticas denominadas *Stall* e *Flush*, baseadas na quantidade de faltas na *cache* de nível 2 também são apresentadas em (TULLSEN e BROWN, 2001). A política *Stall* detecta algum *miss* pendente na *cache* de nível 2 e impede que mais instruções da tarefa sejam buscadas, essa detecção pode ocorrer tardiamente não evitando a ocupação dos recursos pela tarefa. A política *Flush,* é uma melhoria para a *Stall* detectando a ocorrência de falta na cache de nível 2 e desalocando os recursos para que possam ser utilizados por outras tarefas.

Tanto na *Stall* quanto na *Flush* é possível que ocorra a subutilização de recursos. Na *Stall* os recursos detidos pela tarefa que está sendo impedida de continuar a execução, podem não ser requeridos por outras tarefas ficando subutilizados, pois poderiam ser aproveitados na continuação da execução da tarefa impedida. Na *Flush* é possível que tarefas sejam punidas sem razão, sendo os recursos também subutilizados e ainda todas as instruções da tarefa retiradas, tendo portanto um custo adicional em um futuro próximo, pois as instruções da tarefa penalizada terão de ser buscadas novamente para que o trabalho seja refeito.

Combinando as características das políticas *Stall* e *Flush,* (CAZORLA *et al*, 2003) propuseram a política *Flush++,* considerando que a *Stall* trabalha melhor em cenários com poucas tarefas que apresentam uma alta taxa de falta na *cache* de nível 2 e com uma disputa por recursos menor, e que a *Flush* tem melhor desempenho em cenários que possuem tarefas com freqüentes taxas de falta na *cache* de nível 2, porém com uma disputa por recursos mais acirrada. A idéia é usar o comportamento da *cache* para alternar entre as políticas combinadas.

2.3 Tecnologia Hyper-Threading

A tecnologia *Hyper-Threading* (HT) da Intel (MARR *et al*, 2002) faz com que um único núcleo físico seja visto como múltiplos núcleos lógicos, pois há uma cópia do estado arquitetural para cada processador lógico que compartilha os mesmos recursos físicos durante a execução. Esses núcleos lógicos são percebidos pelo Sistema Operacional (SO) como se fossem núcleos físicos, podendo escalonar processos ou tarefas para eles como se estivessem em um sistema multiprocessado.

A primeira implementação da HT foi feita para a família de processadores Intel *Xeon*, com dois processadores lógicos por processador físico. Sua implementação custou menos de 5% no aumento na área do *chip* e nos requisitos de consumo de potência, fornecendo uma melhoria considerável no desempenho.

O estado arquitetural mantido por cada processador lógico é formado pelos registradores de propósito geral e de controle, por registradores relacionados ao controlador de interrupção e alguns registradores de estado. Todos os outros recursos do núcleo físico são compartilhados, tais como *caches*, unidades de execução, previsores de desvios, lógica de controle e barramentos.

Cada processador lógico tem seu próprio controlador de interrupção (APIC), favorecendo a idéia de que para o SO é como se fosse um sistema com dois processadores físicos. Assim, interrupções enviadas via software (*trap*) ou via hardware são tratadas somente pelo processador lógico ao qual são endereçadas.

Um processador lógico pode ter seu funcionamento suspenso temporariamente enquanto falhas na *cache* e erros de previsão de desvios são tratados ou enquanto aguarda por resultados de instruções executadas anteriormente. Porém, devido ao particionamento dos recursos ser estático, cada tarefa tem seu conjunto particular de recursos. Em um cenário de execução com uma única tarefa, os recursos que seriam compartilhados, provavelmente serão subutilizados momentaneamente, mas este fato não poderá prejudicar as demais tarefas que vierem a ser executadas simultaneamente. Um dos objetivos de SMT é que contextos com uma única tarefa ativa tenham o mesmo desempenho com a tecnologia HT, como teriam em um processador sem essa tecnologia.

As instruções buscadas são decodificadas e depositadas em uma *trace cache,* passando a ser denominadas micro-operações (uops). O acesso de cada processador lógico à *trace cache* é alternado em cada ciclo. Se um processador por algum motivo não pode acessar a *trace cache* na sua vez, o outro acessa utilizando toda a largura de banda da *trace cache* em todos os ciclos.

A *trace cache* tem associatividade 8, utiliza a política de substituição LRU e sua ocupação é feita mediante demanda e disputa entre as tarefas, suas entradas são alocadas dinamicamente, e são identificadas por um rótulo (*tag*) que diz a qual tarefa pertencem.

A lógica de alocação (*allocator*) consome as micro-operações da fila e aloca os recursos necessários para sua execução, entre 126 entradas do *buffer* de reordenação, 128 registradores de inteiro e 128 de ponto flutuante, 48 entradas de *load* e 24 entradas de *store*. Alguns desses recursos são particionados de forma que cada processador lógico use a metade.

Caso sejam esgotados os recursos disponíveis para um processador, um sinal é enviado para que os recursos continuem sendo atribuídos somente para o outro processador, até que as entradas ocupadas pelo primeiro sejam liberadas e outras micro-operações possam ser consumidas da estrutura e a execução da tarefa possa continuar.

2.4 Arquiteturas de Múltiplos Núcleos com SMT

Com o surgimento de arquiteturas de múltiplos núcleos em um único processador físico ou *Chip Multiprocessors* (CMP) (OLUKOTUN *et al*, 1996), houve certa redução nas pesquisas em SMT. Entretanto, devido aos limites de desempenho alcançado por esta nova abordagem, o desejo atual é que os processadores passem a combinar as duas estratégias em uma única arquitetura. Com isso, surgem as arquiteturas *Chip Multi-Threaded* (CMT) (SPRACKLEN e ABRAHAM, 2005) como sendo processadores que suportam a execução simultânea de muitas tarefas via combinação do suporte a múltiplos núcleos com SMT. A aplicação de SMT em cada núcleo é feita para os mesmos fins que em processadores de núcleo único, melhor aproveitando os recursos e as oportunidades de execução durante a latência de operações de memória.

Alguns processadores podem ser classificados dentro desse novo conceito. O SPARC64 VI (MARUYAMA, 2003) da Fujitsu, possui dois núcleos com capacidade de executar duas tarefas simultâneas em cada um. O POWER-5 da IBM (KALLA *et al*, 2004), que surgiu como uma melhoria ao *dual-core* POWER-4 lançado em 2001, introduziu a capacidade de execução de tarefas simultâneas em dois núcleos.

O Montecito da Intel (MCNAIRY e BHATIA, 2005) é também um processador com 2 núcleos que suporta a execução de 2 tarefas simultâneas. O CMT SPARC, UltraSPARC T1, codinome Niagara (KONGETIRA *et al*, 2005) da Sun, tem 8 núcleos com capacidade de executar 4 tarefas simultâneas cada, totalizando 32 tarefas. O Niagara-2 (GROHOSKI, 2006) traz 8 núcleos físicos com a capacidade de execução de 8 tarefas por núcleo, totalizando 64 tarefas em execução simultânea.

Esse cenário sugere a continuidade das pesquisas em arquiteturas SMT, haja vista que a tendência das pesquisas e do mercado são os processadores de múltiplos núcleos com a execução de múltiplas tarefas simultâneas. Desta forma, a preocupação com o consumo de potência deste tipo de processador é real e exige esforços de pesquisa.

2.5 Considerações Finais

Neste capítulo foi apresentado o contexto ao qual este trabalho está inserido, mostrou-se algumas técnicas de melhorias de desempenho, com ênfase na subutilização de recursos que pode ocorrer nas arquiteturas superescalares, sugerindo um espaço de exploração do TLP, em adição ao ILP, pelas as arquiteturas SMT. Essas arquiteturas tanto pela implementação quanto pela utilização intensificada de recursos sugerem um consumo de potência maior, justificando o estudo, a análise e o desenvolvimento deste trabalho.

Capítulo 3
Consumo de Potência

Na área conhecida como Computação com Consumo Eficiente de Potência (*Power-eficient Computing*), usam-se termos tais como *power-aware*, *power consumption* e *low-energy* nas pesquisas que se ocupam com as questões relacionadas ao consumo de potência e energia, tanto nos componentes do nível de hardware quanto nos componentes da camada de software (nível sistema). O consumo de potência e as questões térmicas são cada vez mais preocupantes nos processadores modernos. Torna-se importante que informações referentes à potência e ao desempenho sejam visíveis aos projetistas e desenvolvedores em tempo de projeto, pois assim o equilíbrio entre o desempenho esperado e o consumo desejado pode ser equacionado. Estas questões têm se tornado alvo do trabalho de muitos grupos de pesquisas.

Há pouco tempo atrás quando os grandes fabricantes de processadores se mantinham na corrida pela alta freqüência de seus processadores, estas questões não eram tão preocupantes em tempo de projeto talvez pelo fato de questões mercadológicas estarem envolvidas. Pela lei de Moore (MOORE, 1965), que diz que a quantidade de transistores dobra a cada 18 meses, pode-se ter uma idéia da escala redutiva do tamanho dos transistores empregados nos componentes e do aumento da densidade dos *chips*, pois componentes menores aliados à disponibilidade de espaço deram margem a novos arranjos na planta baixa do circuito dos processadores possibilitando melhorias e a adição de mais componentes.

Com a necessidade de processamento cada vez maior, o processador passou a trabalhar com freqüências maiores para que seus circuitos ficassem mais rápidos e melhorassem seu desempenho. Esse fato associado às expectativas do mercado consumidor disparou o início da disputa dos fabricantes pela busca por processadores mais rápidos, busca esta que culminou com o alcance das

limitações físicas dos circuitos. Uma série de problemas que não emergiam nos contextos dos projetos de densidade menor passaram a ser um grande impecílio, transformando o consumo de potência no maior fator limitante para o desempenho de computadores (VENKATACHALAM e FRANZ, 2005).

3.1 Conceito de Potência e Energia

Conceitos que a princípio parecem ser sinônimos, na realidade podem assumir diferentes significados conforme o contexto. Energia é comumente definida como a propriedade de um sistema capaz de realizar trabalho. Nesse sentido, energia e trabalho se associam a uma mesma grandeza. Como o trabalho é medido em *joules*, a energia também pode ser definida em *joules*, mas comumente é medida em *Wh* (Watts*hora).

Potência é a taxa de trabalho realizado ou de energia consumida por unidade de tempo, a qual é medida em *Watts*. Na computação, o trabalho envolve atividades associadas com a execução de programas (adição, subtração, operações de memória), que implica no fluxo de elétrons para o funcionamento do hardware. Nesse contexto, a potência é a taxa na qual o processador consome energia elétrica ou dissipa na forma de calor na realização dessas atividades (VENKATACHALAM e FRANZ, 2005). Formalmente, tem-se que:

$$P = W / T \quad (1)$$

$$E = P * T \quad (2)$$

onde:

- **P**: Potência, medida em *watts*;
- **E**: Energia, medida em *joules*;
- **T**: Um intervalo de tempo; e
- **W**: Total de trabalho realizado no intervalo de tempo.

Esta distinção entre os termos potência e energia é importante, pois as técnicas que reduzem o consumo de potência não necessariamente reduzem o consumo de energia no contexto global. A potência consumida pode ser reduzida pela alteração da freqüência de *clock*, mas se por conta disso o computador aumentar o tempo de execução dos mesmos programas, então o total de energia consumida será maior. Assim, a decisão entre reduzir potência ou energia dependente do contexto.

3.2 Técnicas e Abordagens

Diversas técnicas são aplicadas nos diferentes níveis do sistema computacional, envolvendo componentes, arquitetura e aplicações, caracterizando o gerenciamento de potência como uma

disciplina de grande abrangência e em expansão contínua (VENKATACHALAM e FRANZ, 2005).

3.2.1 Escala Dinâmica de Voltagem e Freqüência (DVFS)

Uma das técnicas que tem sido citada em vários trabalhos é a Escala Dinâmica de Voltagem e Freqüência ou do inglês, *Dynamic Voltage and Frequency Scaling* (DVFS) (FLAUTNER e MUDGE, 2002A), que explora o fato da freqüência ser proporcional à voltagem, enquanto a quantidade de energia dinâmica requerida para dada carga de trabalho é proporcional ao quadrado da voltagem. Assim, reduzindo-se a voltagem, reduz-se a freqüência que conseqüentemente implicará em um aumento no tempo de execução. Sendo considerado ainda a tecnologia utilizada, pois transistores menores necessitam de uma voltagem menor para seu funcionamento, caso contrário são danificados.

Alguns processadores adotam esta técnica na implementação de suas tecnologias, tais como a *LongRun* do Transmeta Crusoe (FLEISCHMANN, 2001), *PowerNow* nos processadores AMD (AMD, 2006) e *XScale* em processadores da Intel (INTEL, 2003), considerando o fato da potência variar linearmente em relação à freqüência de *clock* e ao quadrado da voltagem, ajustando ambos pode-se reduzir cubicamente o consumo de potência.

De acordo com (FLEISCHMANN, 2001), os processadores da família do Transmeta Crusoe possuem gerenciamento adaptativo de temperatura e potência reduzindo dinamicamente o consumo de potência no núcleo para níveis quase ótimos em relação à carga de trabalho requerida pela execução da aplicação, seu gerenciamento térmico adapta a operação do processador ao ambiente do sistema.

O núcleo VLIW (*Very Long Instruction Word*) em conjunto com o *Code Morphing Sofware* (CMS) (DEHNERT *et al*, 2003) que traduz as instruções x86 para instruções VLIW, suporta o padrão industrial criado pela Microsoft, Intel e Toshiba, o ACPI (*Advanced Configuration and Power Interface)* para os modos de gerenciamento de potência. Esse núcleo VLIW possui 6 estados distintos de potência: *Normal*, *Auto Halt*, *QuickStart*, *Deep Sleep*, *DSX* e *Off*, que podem ser usados para reduzir a potência de operação durante os estados do sistema que requerem menor ou nenhuma atividade do processador. Na Tabela 3.1, são apresentados os estados do ACPI e os estados correspondentes implementados pelo Crusoe.

A maioria dos processadores x86 convencionais utiliza as políticas ACPI para regular o consumo de potência, com a capacidade de reduzir a freqüência e alternar rapidamente entre um estado de alto consumo e o de quase desligamento por completo, essa capacidade é denominada de *clock-throttling ou cpu- throttling.* Um dos problemas que pode acontecer é o processador entrar

nesse modo de inatividade no momento em que ocorre uma situação que exige sua capacidade total.

Tabela 3.1: Padrão ACPI

Estados definidos pelo padrão ACPI		Estado do Processador	SDRAM	Gerador de Clock	Consumo
G0 / S0 / C0	Em atividade	Normal	Normal	Executando	
G0 / S0 / C1	Auto Halt	Auto Halt	Normal/SelfRefresh	Executando	0,35 W
G0 / S0 / C2	Quick Start	Quick Start	Self Refresh	Executando	0,30 W
G0 / S0 / C3	Deep Sleep	Deep Sleep DSX	Self Refresh	Parado	0,15 W
G1 / S1	Desativado	Deep Sleep DSX	Self Refresh	PLL Shutdown	0,10 W
G1 / S3	Suspenso para RAM	Off	Self Refresh	PLL Shutdown	
G1 / S4	Suspenso para disco	Off	Off	Off	
G2 / S5	Soft Off	Off	Off	Off	
G3	Off mecânico	Off	Off	Off	

Os mecanismos empregados no gerenciamento de potência do Crusoe selecionam dinamicamente a melhor freqüência de *clock* e a voltagem necessária para a execução da aplicação, proporcionando um consumo eficiente. De acordo com (FLEISCHMANN, 2001) o núcleo VLIW pode se auto configurar ajustando a freqüência em intervalos de 33*MHz*, a voltagem em intervalos de 25*mV* de acordo com a freqüência, isto é quanto mais baixa a freqüência menor é a voltagem empregada. Esses ajustes são rápidos.

Trabalhando com escalas de voltagem e freqüência, ao atingir a escala mínima de um ponto de operação (uma freqüência e voltagem específica), o processador efetua suas trocas de estado no modelo tradicional ACPI, permitindo políticas tais como as do padrão ACPI para tratar o gerenciamento nos pontos de mais baixo consumo. O gerenciamento térmico é feito de forma integrada com o de potência através das políticas que gerenciam o ambiente térmico usando freqüência e a voltagem entre os pontos da escala, na tentativa de maximizar o desempenho e minimizar a temperatura.

3.2.2 *Gating* de Circuitos e de *Clock*

A técnica de *gating* de circuitos (MANNE *et al*, 1998) tem sido aplicada aos estágios do *pipeline,* permitindo que as partes do *pipeline* sejam ativadas de forma independente, restringindo a produção de atividades desnecessárias no *chip*. Esta técnica é tratada em alguns casos como *banking* (PARIKH *et al*, 2002). A redução das atividades de busca, decodificação, remessa e execução,

proporcionada pelo *gating* do *pipeline*, pode ser usada na redução do consumo.

O *gating* de *clock* é o controle da distribuição do sinal do *clock* pelas diversas regiões do *chip*, podendo reduzir tanto quanto for necessário até o impedimento por completo da passagem do sinal de *clock* para poupar o consumo relacionado a ele, nos componentes da arquitetura que não estão sendo utilizados no momento.

A técnica de *gating* (HOMAYOUN *et al*, 2005) é utilizada para reduzir o consumo de potência nas unidades funcionais, sendo utilizado um circuito para detectar as condições e decidir quando é apropriado bloquear o fornecimento de energia e um único transistor para fazer o bloqueio efetivo.

3.2.3 Alterações arquiteturais

Em aplicações móveis, a redução de energia é freqüentemente mais importante para o aumento do tempo de vida da bateria, dando uma autonomia maior as dispositivos que dela dependem. Já para sistemas tais como servidores a manutenção da temperatura dentro de limites aceitáveis é mais importante, sendo necessário reduzir a potência instantânea apesar do impacto na energia total. De acordo com (GIVARGIS *et al*, 2001), o subsistema formado pela CPU, caches, memória principal e barramentos compreende a região de tráfego mais intenso e normalmente é o responsável pela maior parte do consumo de potência.

A proposta de (GIVARGIS *et al*, 2001) para a redução de potência no barramento foi a codificação de dados pelo método barramento invertido (*bus-invert*) o qual utiliza a distância de Hamming entre duas palavras de dados. Se a distância é maior que a metade do tamanho da palavra, os dados são enviados invertidos e o sinal de controle é ajustado para que o dado seja decodificado corretamente no destino. Ainda de acordo com (GIVARGIS *et al*, 2001) a técnica barramento invertido economiza em média 25% da potência teórica e é possível alcançar taxas de até 50%, porém esta técnica pode aumentar o tempo do ciclo do barramento, penalizando assim o desempenho.

Outros trabalhos enfocam o combate ao uso inútil de potência (MUSOLL, 1999), que se caracteriza pelas seguintes situações: quando um desvio é tomado para o caminho errado e quando são feitos acessos desnecessários aos blocos arquiteturais na execução de instruções. Na primeira, a potência consumida na execução das instruções desse caminho e no tratamento da penalidade após a descoberta do erro é considerada um consumo inútil, bem como o consumo proporcionado por cada acesso na segunda situação.

Em sistemas formados por vários blocos, tais como caches de níveis 1 e 2 (L1 e L2), para

reduzir-se o tempo de acesso à L2 os acessos são simultâneos independente de ocorrer uma falta em L1. Porém, nesta estratégia quando o dado é encontrado em L1 o acesso à L2 se torna desnecessário e isso leva a um consumo inútil de potência. Uma alternativa para isso seria desabilitar L2 quando se está acessando L1 e somente acessá-la se ocorrer uma falta em L1. O total de acessos modelado por (MUSOLL, 1999) é dado pela soma dos termos:

$$A=(A_{cc}^{n})+(A_{cc}^{nn})+(A_{ce}^{n})+(A_{ce}^{nn}) \quad (3)$$

Onde:

- **n:** necessário;
- **nn:** não necessário;
- **cc:** caminho correto; e
- **ce:** caminho errado.

O termo A_{cc}^{n} se caracteriza como os acessos úteis e os termos restantes como acessos nos quais ocorre consumo de potência inutilmente. Em ordem da esquerda para a direita temos as parcelas correspondentes aos acessos necessários dentro de um caminho correto, aos acessos desnecessários dentro de um caminho correto e aos acessos necessários ou desnecessários dentro de um caminho tomado por uma decisão de desvio incorreta. Existem duas estratégias para reduzir os acessos inúteis: a primeira aconselha a redução de todo o número de instruções executadas dentro de um caminho errado, reduzindo os acessos sendo eles necessários ou não; a segunda defende a redução de acessos inúteis sendo gerados por instruções de caminhos corretos ou incorretos.

Com a melhoria dos previsores de desvios a primeira seria satisfeita, pois quanto maior a precisão de acerto menos caminhos errados farão parte da execução. A segunda (MUSOLL, 1999) consistiria em reduzir os acessos desnecessários, tentando predizer se o acesso seguinte a um bloco será necessário ou não, e sendo desnecessário o bloco poderia ser desabilitado para que a potência relacionada não seja consumida. Porém, se tal previsor falhar acusando que um acesso é desnecessário quando ele for realmente necessário, a habilitação do bloco desativado e o tempo maior decorrido para a obtenção do resultado ocasionarão um custo adicional.

Decisões no projeto da arquitetura contribuiriam para a redução global de consumo, como por exemplo, operações com operandos de valores em zero que resultam em zero poderiam ser impedidas de serem computadas por suas respectivas unidades funcionais, esse filtro evitaria o consumo desnecessário para a manutenção da atividade dessas unidades na execução desse tipo de

operação. Outras questões importantes, relacionadas ao tamanho e organização da estruturas que compõem o conjunto arquitetural, podem trazer benefícios em tempo de projeto, considerando-se a relação entre o consumo dessas estruturas e o impacto no desempenho que elas proporcionam.

3.2.4 Redução do consumo sob a perspectiva da área de Otimização

O objetivo de todas as técnicas é a redução de consumo de potência e energia sem causar grandes impactos no desempenho dos processadores. Pela perspectiva da área de Otimização, a redução do consumo pode perfeitamente ser tratada como um problema de minimização, sendo o consumo relacionado a um custo que dependendo do ponto em que se está no ciclo que vai desde o projeto até o efetivo funcionamento, pode ser associado a diferentes objetivos conforme a fase a qual a técnica está sendo aplicada.

Por exemplo, um projetista pode empregar tais técnicas para a exploração do espaço de solução no conjunto de todas as combinações de configurações, organizações de um projeto arquitetural, visando um projeto de consumo mínimo. Em seu trabalho (CONTE *et al*, 2000) utilizaram a técnica conhecida como *Simulated Annealing* ou Têmpera Simulada (KIRKPATRICK *et al*, 1983), que é inspirada em um fenômeno da termodinâmica do resfriamento de cristais, para a otimização do desempenho em função do consumo. Uma variante do algoritmo foi utilizada para a busca no espaço de projetos, tendo como ponto de partida um processador de alto desempenho com unidades funcionais duplicadas, sendo que cada uma dessas unidades poderia ser duplicada por muitas vezes, respeitando-se obviamente as restrições de um valor máximo de consumo de potência.

O modelo comportamental desenvolvido em (CONTE *et al*, 2000) para ser aplicado no nível de sistema tem enfoque no projeto de processadores superescalares de alto desempenho. O modelo é uma função de custo separável, que pode ser utilizada para otimizar tais arquiteturas, a função custo é separada dentro dos componentes organizacionais e tecnológicos. O componente organizacional é medido a partir de uma simulação no nível comportamental do tipo usado para o projeto de alto nível e o tecnológico depende da implementação da tecnologia, e a combinação desses componentes permite a obtenção das estimativas de consumo.

O simulador desenvolvido por (CONTE *et al*, 2000) estima também o total de tempo de execução do *benchmark* e combina essas informações com os parâmetros da tecnologia para encontrar a potência dinâmica do componente usado. Por fim, a potência total é estimada somando-se os subtotais dos componentes dinâmico e estático para determinar projetos que alcancem baixo consumo sem sacrificar o desempenho. O modelo de processador utilizado para o estudo foi um

núcleo superescalar com o escalonamento completo de Tomasulo e com *pipeline* em suas unidades funcionais. As unidades de cálculo inteiro e de ponto-flutuante foram duplicadas e suas latências variadas para se obter um paralelismo maior.

Os consumos de cada estágio ativo do *pipeline* associado à cada unidade funcional são obtidos por simulações que acumulam os valores da freqüência de uso dos estágios do *pipeline* para que seja possível o cálculo do consumo total de potência.

3.2.5 Simulação Experimental

Outras formas que abrangem o estudo do consumo de energia e potência estão relacionadas com o emprego de ferramentas computacionais para o projeto e testes de circuitos, e para simulação comportamental e de desempenho. Simuladores arquiteturais possibilitam a avaliação do desempenho e medidas relacionadas ao consumo de potência. Algumas dessas ferramentas que têm relação direta com este trabalho são apresentadas no capítulo intitulado "Ferramentas Relacionadas", sendo elas SimpleScalar (BURGER e AUSTIN, 1997), CACTI (JOUPPI e WILTON, 1994), Sim-Wattch (BROOKS *et al*, 2000), SS_SMT (GONÇALVES *et al*, 2000).

Outros simuladores também são usados para a análise do consumo de potência, tais como o SimplePower (YE *et al*, 2000), PowerTimer (BROOKS *et al*, 2003) (HU *et al*, 2003) (BROOKS *et al*, 2004) e o SoftExplorer (LAURENT *et al*, 2004), mas não interagem com o presente trabalho.

O PowerTimer usa uma metodologia parecida com a do Wattch, porém seu modelo de consumo usa equações baseadas nos parâmetros tecnológicos e de dados empíricos obtidos em simulações e dados reais do processador (BROOKS *et al*, 2004). Utiliza como núcleo arquitetural o simulador Turandot (MOUDGILL *et al*, 1999), simulador da arquitetura do PowerPC da IBM.

Outras ferramentas foram desenvolvidas baseando-se também em ferramentas existentes. Pode-se citar o PTSMT (KANNAN *et al*, 2008) , que é um simulador para a avaliação de consumo de potência, temperatura e desempenho sobre arquiteturas SMT. Tem como núcleo arquitetural o simulador SSMT (MADON *et al*, 1999), como modelo de temperatura o HotSpot (HUANG *et al*, 2005) e como modelo de consumo de potência também utiliza o Wattch (BROOKS *et al*, 2000).

A maioria das ferramentas de análise de potência alcança uma precisão alta calculando as estimativas de potência para projetos, porém consegue esta precisão somente depois que o *layout* e a planta baixa do circuito estão prontos. Ocorre que essas informações são disponibilizadas tardiamente no processo do projeto, além dessas ferramentas serem lentas, o que dificulta a sua execução para um espaço maior de possibilidades de projetos. Neste contexto, ferramentas que proporcionam estimativas do desempenho e do consumo no nível arquitetural ganham espaço e

auxiliam na tomada de decisões inerentes ao projeto arquitetural, antes mesmo de um detalhamento das estruturas e dos circuitos em um nível mais específico.

3.3 Considerações Finais

Pode-se perceber neste capítulo a extensão da área de computação com consumo eficiente de potência, foram apresentados conceitos, algumas técnicas de redução do consumo por modificações arquiteturais, mencionou-se o uso de algoritmos de otimização para a exploração do espaço de configurações e a técnica de simulação experimental, com a qual este trabalho e a ferramenta PowerSMT estão diretamente relacionados.

Capítulo 4

Ferramentas Relacionadas

A ferramenta SimpleScalar (BURGER e AUSTIN, 1997) vem sendo amplamente utilizada por diversos grupos de pesquisa no projeto e avaliação de novas propostas de arquiteturas e de suas características. Para investigar arquiteturas SMT, foi desenvolvido o simulador SS_SMT (GONÇALVES *et al*, 2000) tendo como base o *sim-outorder* do SimpleScalar. Em se tratando especificamente do consumo de potência, o simulador Sim-Wattch (BROOKS *et al*, 2000), também desenvolvido sobre a plataforma SimpleScalar, é uma das principais ferramentas de simulação em nível arquitetural que faz estimativas de consumo de potência. Nas seções seguintes conceitos sobre essas ferramentas são detalhados e uma visão do histórico envolvendo o desenvolvimento delas é apresentada na Figura 4.1, situando essas ferramentas citadas e a desenvolvida neste trabalho, a qual denominamos PowerSMT.

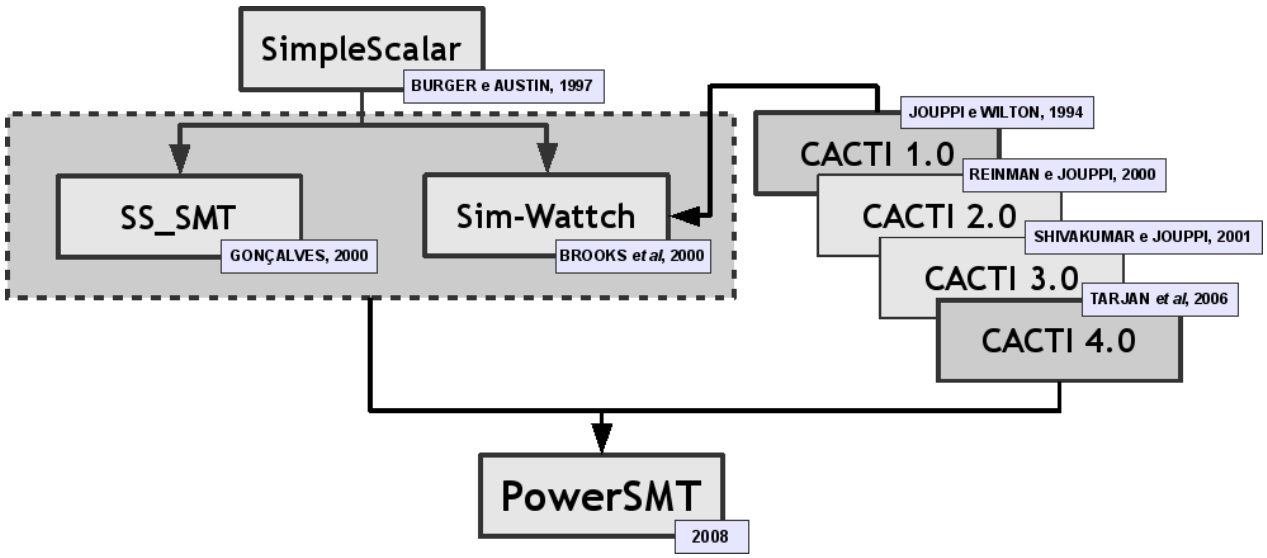

Figura 4.1: Cronologia das Ferramentas

4.1 SimpleScalar

O SimpleScalar (BURGER e AUSTIN, 1997) é um pacote que agrupa um conjunto de simuladores de processadores superescalares baseados no MIPS, compiladores e depuradores, permitindo configurar as principais características arquiteturais. Foi desenvolvida na Universidade de Wisconsin (Madison) como parte do projeto MultiScalar, tendo grande popularidade e é usada como ferramenta de apoio a simulações e no desenvolvimento de simuladores arquiteturais dirigidos por execução (*execution driven*), capazes de executar binários que estão em um formato específico "ss" (subconjunto MIPS IV) e em suas outras variações suporta também a execução de binários compilados para Alpha e ARM.

Dentre os simuladores do pacote, o *sim-outorder* é o mais completo para arquiteturas superescalares, incorporando previsão de desvios, renomeação de registradores e execução fora-de-ordem, em um *pipeline* de 6 estágios: busca, decodificação, remessa, execução, retorno e conclusão. Permitindo a configuração de praticamente todos os recursos do *pipeline* superescalar, incluindo os tamanhos das estruturas, tais como tamanho da fila de busca de instruções, a largura de decodificação, o número de estações de reserva da *Register Update Unit* (RUU), a largura da remessa, fila de *load/store*, a quantidade de unidades funcionais, as caches de instruções e de dados dos níveis 1 e 2.

Traz em seu ferramental ainda um compilador GNU GCC modificado para compilar códigos fonte FORTRAN ou C do *benchmarks* gerando códigos binários executáveis para sua arquitetura específica, que é derivada da arquitetura MIPS-IV ISA suportando executáveis *big-endian* e *little-endian.*Todas as instruções possuem tamanho de 64 *bits* podendo assumir três formatos de codificação: registrador, imediato e salto. O formato registrador é usado em instruções computacionais, o formato imediato suporta a inclusão de uma constante de 16 *bits* e o formato salto especifica alvos de 24 *bits* das instruções de desvio.

Nos formatos de instrução os campos correspondentes a registradores são todos de 8 *bits* suportando a extensão de 256 registradores de inteiro e de ponto flutuante. Seu espaço de endereçamento é de 31 *bits*. As chamadas ao sistema que são feitas pelos binários executados em simulação são interceptadas por um tratador que decodifica a chamada e faz a chamada correspondente ao sistema operacional da máquina hospedeira com os parâmetros interceptados e quando recebe o retorno copia os resultados da chamada para a memória do programa simulado.

O sistema de memória do simulador trabalha com uma fila de *load/store* na qual os valores a serem armazenados são colocados na fila de forma especulativa. Os *loads* são despachados para a memória quando o endereço de todos os *stores* anteriores são conhecidos e podem ser satisfeitos

pelo sistema de memória ou por um valor armazenado anteriormente na fila por uma operação de *store*. Esses *loads* especulativos podem gerar faltas na *cache*, mas faltas especulativas na TLB permanecem no *pipeline* até que o resultado da condição de desvio seja conhecida.

A execução fora-de-ordem consome as instruções que estão armazenadas na RUU, cada instrução dessa estrutura somente é remetida à execução quando tem suas dependências de entrada satisfeitas. O resultado da execução dessas instruções retorna via escrita de resultados, alimentando as dependências de saída dessa instrução, que são as estações de reserva que possuem a instrução como sua dependência de entrada.

4.2 SS_SMT

Como uma modificação do simulador *sim-outorder* da plataforma SimpleScalar, o simulador SS_SMT (GONÇALVES *et al,* 2000) foi desenvolvido para simular e avaliar arquiteturas SMT. O SS_SMT permite executar simultaneamente um conjunto de tarefas sobre um mesmo *pipeline* SMT contendo recursos privados e compartilhados. Foram modificados o sistema de cache, o controle para possibilitar a manipulação dos contextos denominados *slots*. Todas as facilidades de configuração e execução, existentes no SimpleScalar, foram herdadas pelo SS_SMT.

Para o desenvolvimento do SS_SMT primeiramente foi desenvolvido um simulador de multi-processador por meio da replicação das estruturas encontradas no *sim-outorder*, passando cada réplica de processador a ser indexada por um número. De acordo com (GONÇALVES *et al,* 2000) as variáveis simples foram transformadas em variáveis vetoriais e as vetoriais por sua vez em matriciais, sendo que todas as funções foram redefinidas e preparadas para aceitarem como parâmetro o identificador do processador a ser manipulado. Desta forma, o simulador de multiprocessador permitia a execução simultânea de *n* programas de forma independente.

Posteriormente, todos os estágios do *pipeline* foram unificados e muitos recursos passaram a ser compartilhados, surgindo assim o simulador SS_SMT, no qual cada conjunto de recursos usado para manter o contexto de uma tarefa é chamado *slot*. Novas características foram adicionadas às existentes no *sim-outorder*, como por exemplo, a capacidade de utilização de diferentes organizações (distribuída ou compartilhada) para as estações de reserva (RUU) e para a fila de *load-store* (LSQ) entre os *slots*, a configuração da profundidade de decodificação, a definição de diferentes hierarquias de memória, possibilidade de controle da taxa de acerto da previsão de desvios (forçar uma taxa de acerto) e a possibilidade de diferentes organizações para as unidades funcionais (heterogênea, homogênea e compacta).

Para as estações de reserva na organização distribuída, cada tarefa acessa o recurso

disponível em seu *slot* e na compartilhada, todas as tarefas acessam a mesma estrutura, porém o acesso que cada *slot* faz é sobre um subconjunto das entradas da estrutura.

Quanto às organizações das unidades funcionais, a heterogênea é a mesma organização encontrada no SimpleScalar, a homogênea fornece um único tipo de unidade funcional que trata todas as classes de instruções e a compacta é organizada para permitir a execução de alguns agrupamentos de classes de instruções.

4.2.1 Modelo Arquitetural SS_SMT

O simulador SS_SMT possibilita, por meio de configuração e parametrização, simular o uso de estruturas compartilhadas ou privativas entre as tarefas, tais como caches, estações de reserva e unidades funcionais, possibilitando a análise de vantagens e desvantagens de cada caso. O simulador prevê os mesmos estágios do *pipeline* superescalar.

O estágio de busca de instruções trabalha segundo estilo *round-robin*, buscando a cada ciclo instruções de todos os módulos paralelamente, mas de apenas uma das tarefas que estão dentro de cada módulo de cache, de forma a usar todo o barramento do respectivo módulo. As instruções são colocadas em filas privativas das tarefas em execução. Por exemplo, em uma execução de 4 tarefas simultâneas são possíveis organizações para os 4 bancos do sistema de cache: 1 módulo com os 4 bancos, 2 módulos com 2 bancos cada ou 4 módulos com 1 banco cada. Os acessos são feitos em paralelo para cada módulo e neste o acesso exclusivo a 1 banco. Assim sendo na configuração de 1 módulo, o acesso ocorre a um banco por vez, instruções de uma única tarefa são buscadas por ciclo, enquanto para a configuração de 4 módulos, sendo os quatro módulos acessíveis em paralelo, instruções das quatro tarefas podem ser buscadas no mesmo ciclo de busca.

O estágio de decodificação verifica as extremidades de todas as filas de instruções e decodifica um *mix* de instruções de todas as tarefas simultaneamente, cujo número depende do tamanho do barramento de decodificação. As instruções decodificadas têm suas dependências tratadas e são despachadas às estações de reserva junto as unidades funcionais.

O estágio de remessa seleciona as instruções que estão prontas nas estações de reserva e as remete às unidades funcionais especializadas que estão livres e de acordo com o tipo de instrução. As instruções são executadas pelas unidades funcionais e os resultados alimentam as instruções que aguardam operandos nas estações de reserva para se tornarem prontas para execução. Uma fotografia do *pipeline* em funcionamento mostrará que a janela de instruções deste processador contém um *mix* de instruções de todas as tarefas que estão sendo executadas simultaneamente.

4.3 CACTI

A CACTI é uma biblioteca que define e implementa um modelo baseado em formulações para a medição do consumo de potência. A versão inicial da CACTI 1.0 (JOUPPI e WILTON, 1994) (WILTON e JOUPPI, 1996) foi desenvolvida com o intuito de possibilitar a modelagem de caches SRAM, considerando que o consumo é calculado em função do tempo de acesso às estruturas da cache. O *Time Model* que otimiza os parâmetros quebrando a estrutura em *subarrays*, foi o primeiro modelo definido estando presente desde a versão inicial até as versões mais recentes. A Figura 4.2 apresenta o modelo estrutural da cache implementada pela biblioteca CACTI.

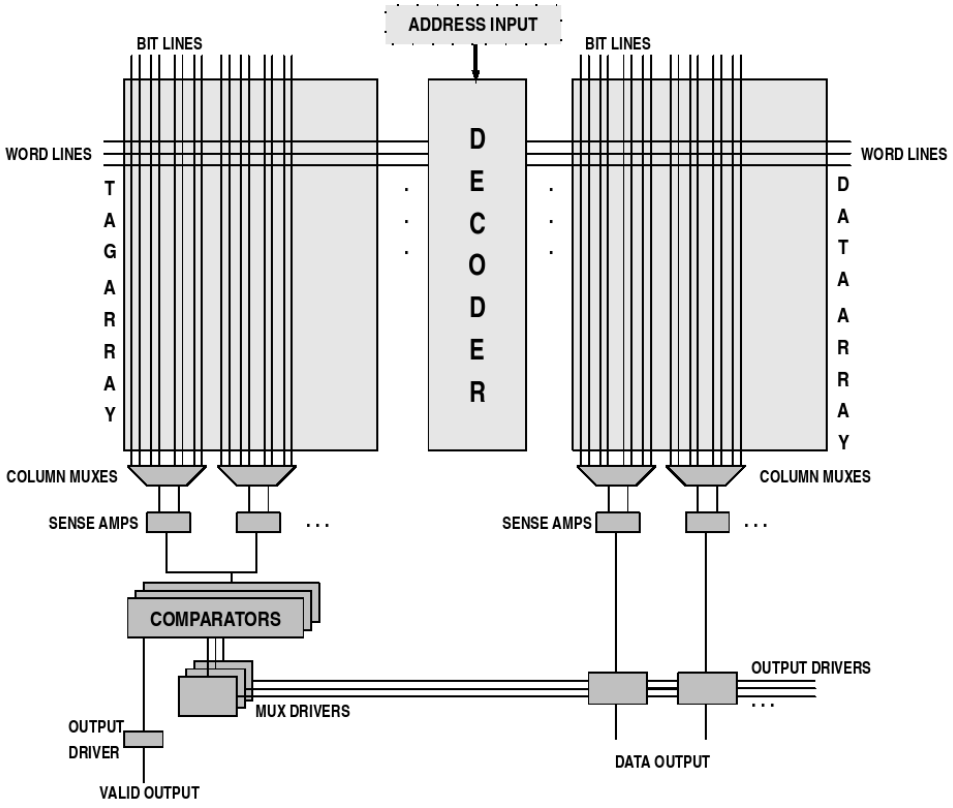

Figura 4.2: Modelo de cache implementado pela CACTI

Duas novas versões, a CACTI 2.0 (REINMAN e JOUPPI, 2000) e CACTI 3.0 (SHIVAKUMAR e JOUPPI, 2001) adicionaram o modelo de área (*Area Model*) e o modelo de potência (*Power Model*) que são utilizados na otimização dos parâmetros, em adição ao *Time Model* que já era utilizado na versão anterior. A versão 4.0 (TARJAN *et al*, 2006) adicionou ao conjunto anterior o modelo de potência estática (*leakage power*) baseando-se nas modificações da versão 3.0 propostas por outras implementações, tais como eCACTI (MAMIDIPAKA e DUTT, 2004) e Hotleakage (ZHANG *et al*, 2003), atualizando ainda a estrutura de circuitos básicos e parâmetros dos dispositivos para melhor refletir os avanços da tecnologia dos semicondutores. O objetivo do *Leakage Model* é fornecer medidas do consumo estático, já que o consumo dinâmico já

é contemplado desde a versão inicial.

A versão 5.0 (THOZIYOOR *et al*, 2007), é a versão principal da biblioteca atualmente e apresenta melhorias em relação a versão 4.0, modificando a forma de ser utilizada. Permite a simulação de caches, estruturas SRAM como era possível já na 4.0, mas também de outros tipos de memória, como por exemplo a DRAM. A versão 5.1 (THOZIYOOR *et al*, 2008) corrige alguns pequenos *bugs* da versão 5.0. A versão 5.2 é a última versão disponível trazendo algumas melhorias para a versão 5.1 quanto ao suporte a DRAM.

4.4 Sim-Wattch

Este simulador é uma extensão do *sim-outorder*, feita por (BROOKS *et al*, 2000) com o *framework* Wattch (BROOKS *et al*, 2000) que utiliza a biblioteca CACTI 1.0 (JOUPPI e WILTON, 1994) como parte do modelo de consumo para estimar o consumo de potência. É uma ferramenta de simulação que faz estimativas de consumo de potência do processador em nível arquitetural, desenvolvida na Universidade de Princeton. Segundo resultados alcançados por (BROOKS *et al*, 2000) é cerca de 1000 vezes mais rápida que as ferramentas existentes para medir potência em nível de *layout*, mantendo a precisão de suas estimativas dentro da taxa de 10% conforme verificação realizada utilizando resultados da indústria, no caso a Intel.

O funcionamento do Sim-Wattch é baseado em ciclos e na freqüência de acesso às estruturas, inicialmente valores de referência para o consumo de cada estrutura são calculados. A cada ciclo os contadores de acesso são reinicializados antes da passagem pelo ciclo arquitetural, e ao final uma função é chamada para atualizar os valores acumulados para o consumo baseando-se no número de acessos que cada estrutura teve. O relatório emitido pela ferramenta traz informações detalhadas sobre a potência consumida em cada estrutura do processador no mesmo estilo das estatísticas geradas pelas ferramentas do SimpleScalar. O Sim-Wattch permite aos arquitetos explorarem e selecionarem o espaço de projeto antecipadamente, usando uma ferramenta rápida e de alto nível. Abre o campo de consumo eficiente de potência para pesquisadores de Ciência da Computação fornecendo uma metodologia de avaliação de potência dentro do SimpleScalar, que é bem conhecido e amplamente utilizado.

O Wattch (BROOKS *et al*, 2000) pode ser usado de pelo menos 3 formas diferentes. Na primeira, o projetista está interessado em entender o comportamento do *hardware* e assim, varia os parâmetros internos do processador e analisa os resultados sobre um mesmo *benchmark*. Na segunda, o projetista está interessado em entender o comportamento do software e assim, mantém fixados os parâmetros internos do processador e analisa os resultados sobre diferentes *benchmarks*.

Na terceira forma o projetista está interessado em propor novas técnicas de otimização de hardware e assim, adiciona novas estruturas ao processador e analisa os resultados.

De fato, os projetistas podem combinar estas formas a fim de obter resultados mais abrangentes, o Wattch permite medir o consumo de potência considerando 3 tipos diferentes de estilos de *clock gating*, denominados CC1, CC2 e CC3. O CC1 considera que qualquer acesso a um bloco arquitetural consome a potência do bloco todo, mesmo quando o bloco permite mais do que um acesso simultâneo. O estilo CC2 considera que o consumo de potência em um bloco arquitetural que permite mais do que um acesso simultâneo é proporcional ao número de acessos. E o CC3 é uma variante do CC2, considerando um consumo mínimo de 10%, a título de manutenção, mesmo quando um bloco arquitetural não é acessado. Os autores afirmam que o tipo CC3 produz as estimativas mais próximas dos valores obtidos em situações reais. Esses estilos foram herdados juntamente com o modelo de consumo pelo PowerSMT.

A Figura 4.3 apresenta o esquema de funcionamento do Sim-Wattch o qual utiliza o *sim-outorder* como simulador arquitetural, para obter a freqüência dos acessos aos blocos e recursos arquiteturais estimando o consumo de potência através do mapeamento de estruturas como *caches*, registradores e janela de instruções com um modelo de consumo definido para o Wattch utilizando funcionalidades fornecidas pela biblioteca CACTI 1.0 (JOUPPI e WILTON, 1994).

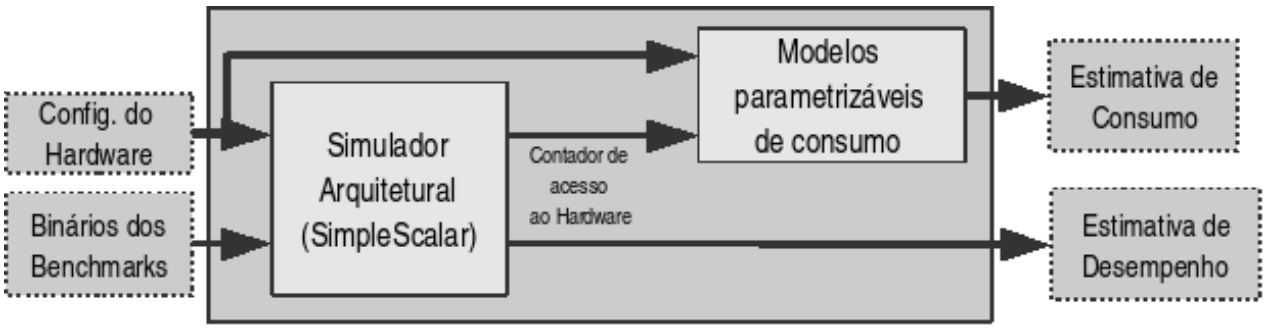

Figura 4.3: Arquitetura do funcionamento do Sim-Wattch

4.4.1 Modelo de Consumo do Sim-Wattch

O Wattch (BROOKS *et al*, 2000) tem como objetivo expor as bases do modelo de consumo de potência em alto nível para arquitetos e desenvolvedores, pois ferramentas de baixo nível operam sobre os projetos dos circuitos completos ou de suas especificações em HDL alcançando alta precisão, porém não são de uso no nível arquitetural para a tomada de decisão em tempo de planejamento do projeto.

De acordo com (BROOKS *et al*, 2000) os fundamentos para a infra-estrutura de modelagem

proposta pelo Wattch são modelos parametrizados de potência de estruturas comuns presentes em microprocessadores modernos. Estes modelos podem ser integrados com simuladores arquiteturais para fornecer estimativas de potência, não precisando, necessariamente, ser o SimpleScalar. As principais unidades que foram modeladas utilizando-se a biblioteca CACTI 1.0 (JOUPPI e WILTON, 1994) são:

- **Estruturas no formato de *array*:** utilizadas para o mapeamento das *caches*, dos *arrays* de *tags*, todos os registradores, tabela de renomeação de registradores, previsores de desvio, grandes porções da janela de instruções e fila *load/store*.

- **Memórias endereçáveis pelo conteúdo totalmente associativo:** janela de instruções, lógica do *buffer* de reordenação, verificação de ordem de *load/store* e TLBs.

- **Lógica Combinacional e fios:** unidades funcionais, lógica de seleção da janela de instruções, lógica de verificação de dependência e barramento de resultados.

- ***Clocking:*** *buffers* de *clock*, *clock wires* e *capacitive loads*.

Em microprocessadores baseados na tecnologia CMOS, o consumo dinâmico é a principal fonte de consumo de potência e é definido como:

$$P_d = C V_{dd}^2 a f \tag{4}$$

Onde:

- C: a carga de capacitância;
- V_{dd}: voltagem fornecida;
- f: a frequência de *clock*; e
- a: fator de atividade.

O fator de atividade *a*, é uma fração entre 0 e 1 indicando com qual freqüência o *clock* pulsa propiciando a troca de atividade na média. O modelo proposto (BROOKS *et al*, 2000) estima *C* baseando-se no circuito e nos tamanhos dos transistores, sendo que *Vdd* e *f* dependem da tecnologia assumida. No caso do Sim-Wattch segundo (BROOKS *et al*, 2000) utilizaram a tecnologia 0,35µm como parâmetro tecnológico. O fator de atividade (*a*) está relacionado ao *benchmark* executado, sendo que para circuitos que pré-carregam e descarregam em todo ciclo é usado o valor 1 para *a*. Os fatores de atividade para certos sub-circuitos críticos são mensurados a partir dos *benchmarks* usando o simulador arquitetural.

O Sim-Wattch faz uma estimativa do consumo, e os valores obtidos conforme a

configuração arquitetural podem ser bem altos, o que coloca em questionamento a precisão com a qual esses valores são calculados, porém mesmo que em termos tecnológicos os consumos sejam calculados em função de uma tecnologia considerada desatualizada, ainda assim para avaliação arquitetural do consumo e do desempenho, esses resultados são eficazes quando vistos comparativamente e não encarados como valores absolutos.

Se os valores dos consumos não podem ser considerados como de precisão absoluta, pode-se confiar no consumo relativo que eles representam. Satisfazendo a necessidade de se saber se uma determinada alteração arquitetural irá produzir impactos positivos ou negativos no desempenho e no consumo, pois um percentual de ganho de uma configuração em relação a outra, considerando os valores como sendo relativos, também será transmitido para a implementação final. Sendo portanto, as estimativas de consumo dadas pelo Sim-Wattch confiáveis enquanto valores relativos, servindo como fonte de informação para a tomada de decisão em tempo de projeto.

4.5 Considerações Finais

Neste capítulo foram apresentadas as ferramentas que serviram como base para o desenvolvimento da ferramenta proposta por este trabalho, o PowerSMT. Apresentou-se simuladores como o SimpleScalar que é amplamente utilizado em pesquisas, o SS_SMT que foi utilizado como núcleo arquitetural para o PowerSMT e o Sim-Wattch, do qual o modelo de consumo foi extraído, modificado e aplicado ao núcleo arquitetural do PowerSMT.

Capítulo 5
Simulador PowerSMT

Sabe-se que na execução de tarefas simultâneas, ocorre uma exploração maior dos recursos arquiteturais, conseqüentemente, acarretando um consumo maior de potência. Desta forma, a investigação a respeito da organização SMT, considerando que os níveis de compartilhamento possam ser otimizados a fim de se atingir uma possível configuração ideal de arquitetura orientada à minimização do consumo de potência sem prejuízos ao desempenho, é de fundamental importância.

O PowerSMT foi desenvolvido sobre o simulador SS_SMT (GONÇALVES *et al*, 2000) por meio da adição do modelo de consumo de potência utilizado no Sim-Wattch (BROOKS *et al*, 2000), modelo esse que é baseado originalmente no CACTI 1.0 (JOUPPI e WILTON, 1994). Desta forma, as características relacionadas ao núcleo arquitetural são as mesmas do simulador SS_SMT nessa versão inicial, quanto às características relacionadas ao modelo de consumo de potência seguem o padrão de implementação do modelo de consumo do Sim-Wattch, porém esse modelo sofreu alguns ajustes e modificações para se adequar ao novo simulador e a biblioteca CACTI foi atualizada para a versão 4.0.

O modelo arquitetural implementado pelo simulador SS_SMT permite a execução de *benchmarks* codificados para o conjunto de instruções do SimpleScalar (subconjunto do MIPS). Assim, no contexto do PowerSMT, as tarefas são de fato programas independentes. Várias tarefas podem ser colocadas para a execução simultânea. Cada tarefa terá o seu próprio contexto de execução e produzirá os seus próprios resultados, muito embora todos estejam sujeitos às mesmas configurações arquiteturais. Ao final da execução de cada tarefa o resultado correto e a semântica do código original de cada tarefa são preservados.

A Figura 5.1 apresenta as possibilidades vislumbradas inicialmente, sendo possível por

parametrização especificar ao simulador os compartilhamentos e o comportamento desejado para a arquitetura, possibilitando comparar o desempenho e o consumo para diversas organizações da execução SMT. É perceptível a variabilidade de abordagens de avaliação vinculadas a essas opções de compartilhamento, o isolamento de determinados recursos e a necessidade de controle e manutenção dos contextos das tarefas em execução, garantindo ao final da execução de cada tarefa o resultado correto, e que a semântica do código original tenha sido mantida.

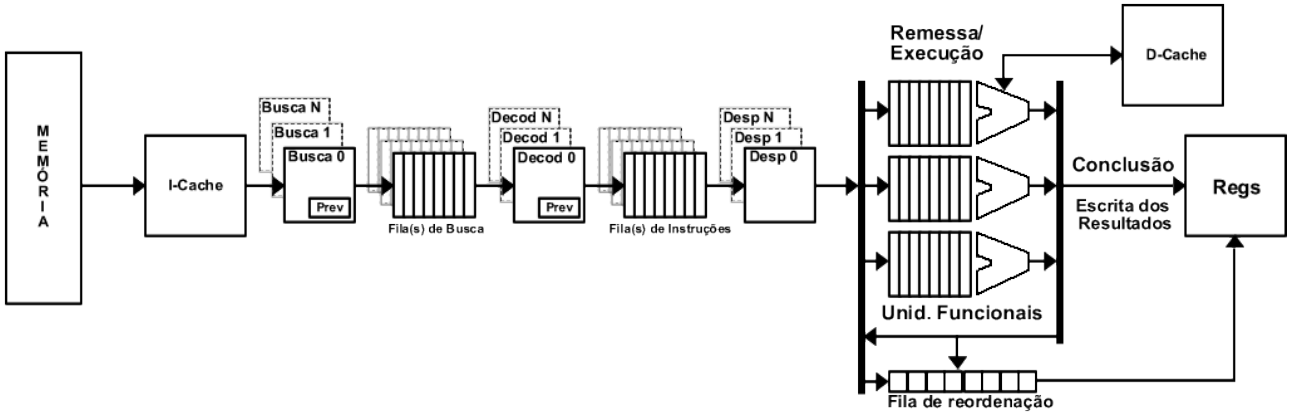

Figura 5.1: Organização da arquitetura SMT com suporte para N contextos

O PowerSMT é uma ferramenta de código aberto, respeitando os termos da licença das ferramentas que serviram como base para seu desenvolvimento, tendo como objetivo principal a investigação do consumo de potência em arquiteturas SMT. Como objetivo secundário, porém necessário à validação das idéias e proposições desencadeadas a partir dessa análise, tem-se o desenvolvimento de uma ferramenta capaz de fornecer resultados através de simulações, resultados esses relacionados ao desempenho e ao consumo de potência, possibilitando aos seus usuários uma avaliação rápida tal como é possível realizar hoje com o SimpleScalar e com o Sim-Wattch para arquiteturas superescalares.

5.1 Desenvolvimento

O processo de desenvolvimento do PowerSMT deu-se por meio do estudo e mapeamento funcional das ferramentas que serviram de base para sua implementação. No intuito de identificar as estruturas relacionadas ao conceito superescalar no SimpleScalar, das estruturas relacionadas à implementação da execução de tarefas simultâneas no SS_SMT, e quanto a forma de mensurar o consumo pelo Sim-Wattch, bem como a integração do simulador arquitetural *sim-outorder* com o modelo de consumo e do uso da biblioteca CACTI 1.0 por esse modelo.

Após o estudo dos códigos das ferramentas o simulador SS_SMT foi tomado como base principal para a implementação, sendo utilizado como núcleo arquitetural SMT, dele foi herdada toda a estrutura relacionada às possibilidades de compartilhamento de recursos e controle de

contextos, optou-se por utilizar o modelo de consumo do Sim-Wattch por ser um trabalho conhecido e por seguir a mesma linha de desenvolvimento da plataforma SimpleScalar.

O PowerSMT foi desenvolvido para o ambiente GNU/Linux seguindo o mesmo padrão de entradas de parâmetros e de geração de resultados do SimpleScalar. Utilizou-se como IDE de programação o Eclipse, uma ferramenta de código aberto que além de suportar o desenvolvimento em Java, possui também suporte à programação em C/C++, que contribuiu e facilitou em muito o trabalho, desde o rastreamento e mapeamento das funcionalidades durante o estudo do código das ferramentas, até no desenvolvimento do simulador. Mantendo a compatibilidade com a estrutura de projeto advinda do SimpleScalar, que prevê somente a utilização de um arquivo *Makefile*.

A versão inicial do PowerSMT foi obtida pela implantação do modelo de consumo de potência herdado do Sim-Wattch, a Figura 5.2 representa a idéia inicial de integração entre o modelo de consumo e o simulador arquitetural SS_SMT.

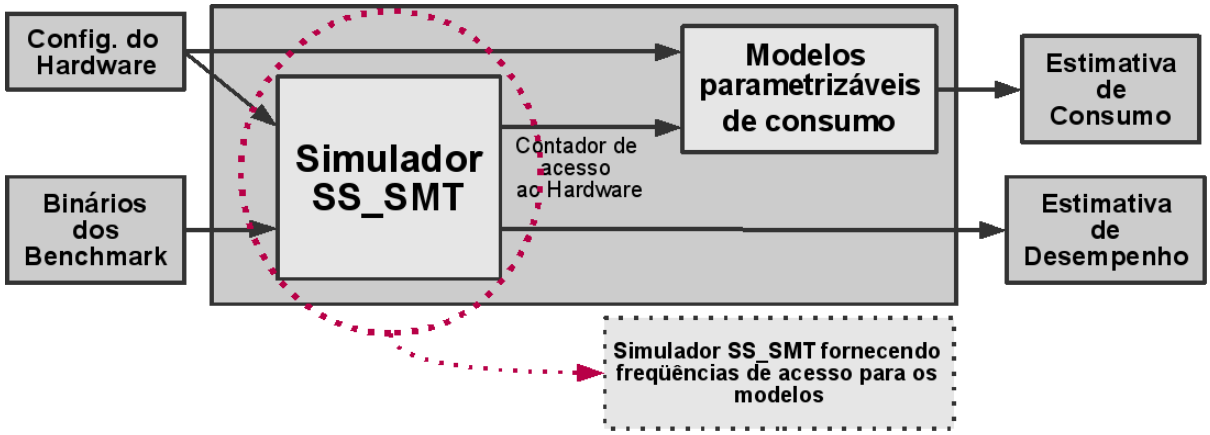

Figura 5.2: Versão inicial do PowerSMT, implantação do modelo de consumo no SS_SMT

Inicialmente o modelo de consumo do Sim-Wattch original foi transformado para o padrão proposto pelo SS_SMT, as funções e estruturas do modelo de consumo foram modificadas para suportarem um identificador, o número do *slot*, tal como é utilizado nas funcionalidades existentes no SS_SMT, essa variação de implementação foi denominada *sim-wattch-smt* devido à natureza da modificações.

Durante a fase de validação da ferramenta PowerSMT, a mesma foi experimentada exaustivamente e os resultados da simulação foram comparados com aqueles obtidos nas ferramentas originais de base. Realizou-se uma verificação de conformidade entre os resultados do Sim-Wattch original e os resultados do *sim-wattch-smt* através de execuções dos mesmos *benchmarks*, fixando o número de tarefas a serem executadas pelo *sim-wattch-smt* em um, desta forma a execução de uma única tarefa produziu os mesmos resultados que uma execução comum no *Sim-Wattch* original. Esses testes garantiram que erros não foram inseridos no processo de

modificação para o padrão do simulador SS_SMT.

Obtendo resultados positivos na verificação realizada, a passo seguinte foi implantar esse modelo de consumo modificado na estrutura arquitetural herdada do SS_SMT, surgindo assim a ferramenta alvo da implementação deste trabalho, o simulador PowerSMT. Após a implantação do modelo de consumo, foram realizados novos testes de verificação de conformidade entre os resultados do *Sim-Wattch* original, do SS_SMT e do PowerSMT, os testes e os resultados são descritos na seção "Avaliação e validação do modelo de consumo" neste capítulo.

Em um momento posterior, outras modificações foram feitas para adequar as estruturas herdadas do SS_SMT que em sua implementação foram modificadas, tais como o sistema de *cache*, que trabalha com módulos e bancos, diferentemente do sistema de *cache* original. Foram considerados nesta etapa as possíveis modificações no modelo de consumo para que se adequasse ao modelo arquitetural herdado do SS_SMT com estruturas nas quais estão previstos os compartilhamentos, as dimensões e formas de acessos ao sistema de *cache* e organizações das unidades funcionais.

Pode-se dizer que o simulador PowerSMT é uma integração entre o modelo de consumo proposto pelo Sim-Wattch e o modelo arquitetural herdado do SS_SMT, sendo adicionadas as alterações e adequações do modelo de consumo ao modelo arquitetural, conforme representa a Figura 5.3.

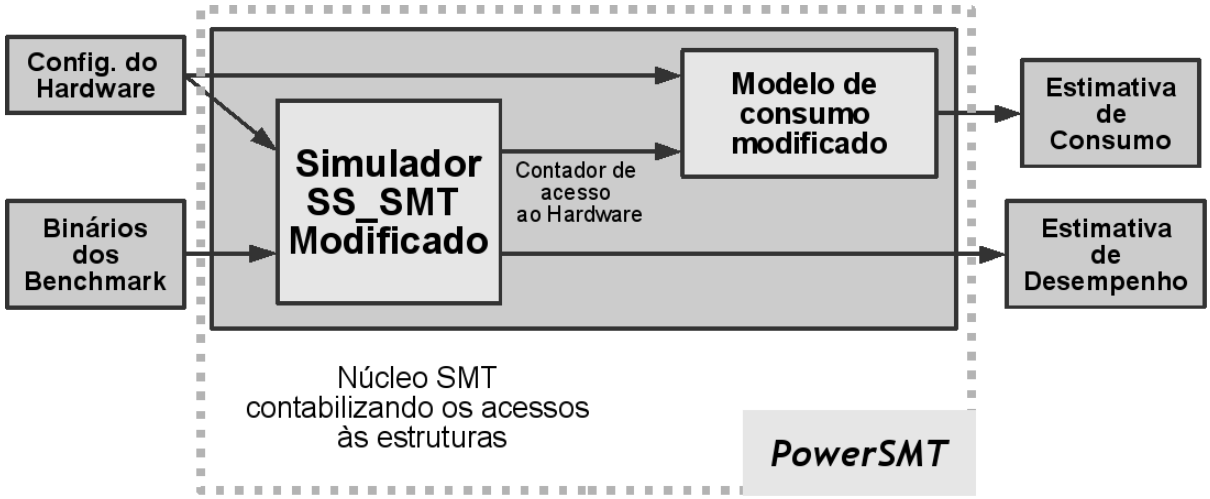

Figura 5.3: Simulador PowerSMT

5.2 Avaliação e validação do modelo de consumo

Como método de avaliação utilizou-se durante os testes e experimentos de simulação a comparação dos resultados. A conformidade dos resultados obtidos pelo PowerSMT foi verificada através de testes que objetivaram confrontar os resultados obtidos com os resultados das ferramentas originais.

Estes testes foram realizados com a versão inicial do PowerSMT por estar mais próxima das versões das ferramentas originais, sendo ainda parametrizado de forma a comportar-se como uma arquitetura livre de compartilhamentos e em modo de execução de tarefa única, possibilitando a comparação com as execuções feitas no Sim-Wattch original.

Foram tomados os devidos cuidados para que as estruturas em um simulador fossem equivalentes às estruturas representadas no outro simulador, para que as diferenças estruturais fossem mapeadas de forma a serem equivalentes entre as ferramentas comparadas. É importante ressaltar isso, pois já mencionou-se neste trabalho as diferenças quanto ao sistema de *caches*, organizações das unidades funcionais e os possíveis compartilhamentos de recursos entre os *slots*. As simulações foram disparadas através de *scripts* para *benchmarks* de aritmética inteira e de ponto flutuante.

5.3 Atualização da Biblioteca CACTI

A versão da biblioteca CACTI utilizada no Wattch é a CACTI 1.0. Sendo assim, na modelagem feita no Sim-Wattch é considerada somente a otimização de parâmetros orientada ao tempo de acesso, pois como mencionado, esta versão inicial da CACTI somente contém o T*ime Model*. Os parâmetros repassados à CACTI para o cálculo do consumo de potência, referentes a tecnologia do processo de fabricação, são fixos no código na CACTI 1.0. Este fato pode confundir os usuários quanto à inicialização desses parâmetros nos arquivos relacionados e a utilização efetiva desses parâmetros pela biblioteca, pois testes realizados mostraram que eles eram ignorados pela biblioteca. Diante dessa restrição e da possível melhoria na precisão dos resultados considerando os demais modelos e não somente o *Time Model*, surgiu a motivação para a atualização da biblioteca CACTI utilizada no PowerSMT para a versão 4.0.

Na versão CACTI 4.0, toda a interação com a biblioteca ocorre por meio de uma única chamada de função, que inclui diversos parâmetros, inclusive o da tecnologia do processo de fabricação que era fixo na versão 1.0. Além disso, muitos outros parâmetros que eram fixos na CACTI 1.0 são calculados dinamicamente na CACTI 4.0 em função do parâmetro de tecnologia do processo de fabricação. As duas versões foram depuradas e os modelos de área, de *leakage*, de tempo e de potência foram comparados para maior certificação de compatibilidade durante os cálculos feitos em tempo de simulação.

A possível confusão entre os reais valores utilizados nos cálculos pode ocorrer com a utilização do Sim-Wattch e de outras ferramentas que utilizam o seu modelo de consumo, tal como HotLeakage (ZHANG *et al*, 2003). Alguns parâmetros, tais como o FUDGEFACTOR, utilizado

pelo HotLeakage como 4,4 para a escala da tecnologia 0,18μm, não são considerados pela biblioteca CACTI 1.0 que se mantém utilizando 0,8μm.

Foram feitos testes compilando a biblioteca com mensagens de depuração e o valor considerado para o FUDGEFACTOR continuou como 1,0 instituído para a tecnologia 0,8μm. Considerando ainda que o problema pudesse estar no fato de compilar a biblioteca novamente, em outros testes utilizou-se a biblioteca original e os resultados foram idênticos aos obtidos com a biblioteca utilizando o FUDGEFACTOR com valor igual a 1,0. Assim, decidiu-se pela tentativa de atualizar a biblioteca para uma versão mais recente, a versão 4.0, que apresenta algumas facilidades para ser usada como componente de outras ferramentas.

Pelos arquivos *def.h*, *area.h* e outros arquivos de *header* da CACTI 4.0, nota-se que os parâmetros antes definidos através da diretiva **#define**, portanto fixos na versão inicial, agora estão sendo calculados dinamicamente pela biblioteca em função do parâmetro de tecnologia e a aplicação destes aos modelos de tempo, área, potência dinâmica e potência de *leakage*. Isto evita a dispersão dos lugares contendo parâmetros a serem configurados na utilização da biblioteca e possíveis desencontros entre as configurações do modelo de consumo e as assumidas pela biblioteca CACTI.

5.4 Adequação do Modelo de Consumo

A dinâmica de execução que foi herdada pelo PowerSMT obrigou a ajustes do modelo de consumo de potência e algumas adaptações e modificações, principalmente nos cálculos dos valores de referência por meio da função *calculate_power()*, para estimar cálculos corretos do consumo de potência e para que a atualização por meio da função *update_power(),* e a geração das estatísticas de execução pudessem ser feitas e reportadas também corretamente, no relatório de execução, para estruturas novas e para as que foram modificadas.

A modificação usou a própria padronização do modelo de consumo no formato trazido ao projeto pela estrutura do SS_SMT, a transformação de variáveis comuns em vetoriais, de vetoriais em matriciais e assim por diante que foi aplicada ao modelo de consumo garantindo que as estruturas que eram comuns ao *sim-outorder* original e a cada *slot* do SS_SMT tivessem as mesmas formas para os cálculos dos valores de referência e as mesmas formas de atualização, preservando o modelo de consumo original.

As diferenças entre as duas ferramentas Sim-Wattch e SS_SMT são consideráveis, e conseqüentemente o PowerSMT por herdar as características arquiteturais do SS_SMT também as apresenta. O PowerSMT recebeu um sistema de *cache* diferente daquele usado no SimpleScalar

original e contido no Sim-Wattch. A organização das unidades funcionais também difere, existindo duas classes a mais que no SimpleScalar (heterogênea, homogênea e compacta). A organização da RUU e da LSQ, que no Sim-Wattch é única, pode ser configurada como distribuída ou compartilhada no PowerSMT. Nas seções seguintes, estas características são explicadas.

5.4.1 Sistema de Caches

O sistema de cache no PowerSMT é estruturado por um conjunto de módulos que são acessados paralelamente, sendo que cada módulo possui um barramento separado e conjunto de bancos internos que são acessados de forma exclusiva entre si, conforme mostra a Figura 5.4. Uma ou mais tarefas podem compartilhar o mesmo banco, mas experimentos prévios mostram que o recomendável é que as tarefas estejam em bancos separados para reduzir conflitos significativamente. Esta estrutura é bem distinta da original sendo necessário a adequação do modelo de consumo para considerar consumos relacionados à lógica envolvida para a seleção do módulo ou módulos que estão sendo acessados, e nestes módulos os respectivos bancos.

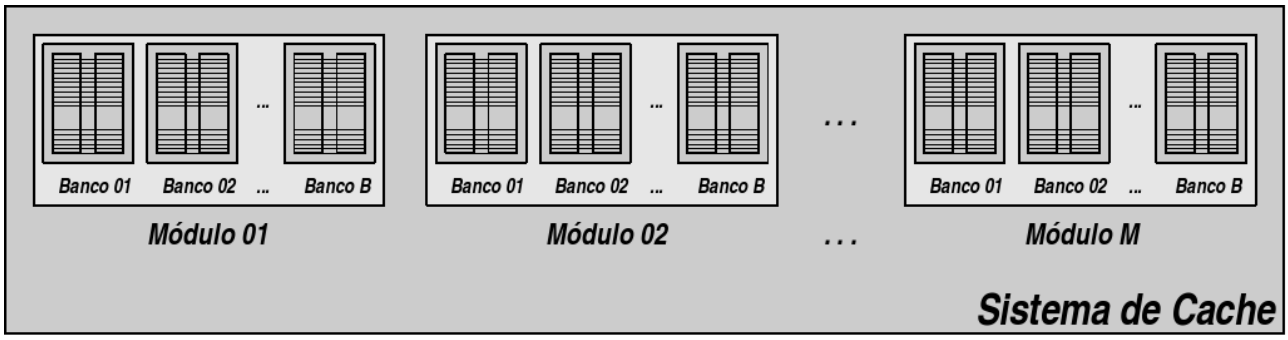

Figura 5.4: Visão Lógica do Sistema de Cache do PowerSMT

A estrutura lógica formada pelos módulos é restritiva à forma de acesso, pois permite que todos os módulos possam ser acessados simultaneamente dependendo da configuração, porém, dentro de um mesmo módulo somente um banco pode ser acessado por vez. Desta forma, cada tarefa acessa seu respectivo banco utilizando o sistema de *cache* tal como mostrado na Figura 5.5. O exemplo desta figura mostra que a distribuição dos bancos pelos módulos é dada por T/M, isto é, o número de tarefas dividido pelo número de módulos, possibilitando que um banco de cada módulo possa ser acessado simultaneamente no mesmo ciclo de busca.

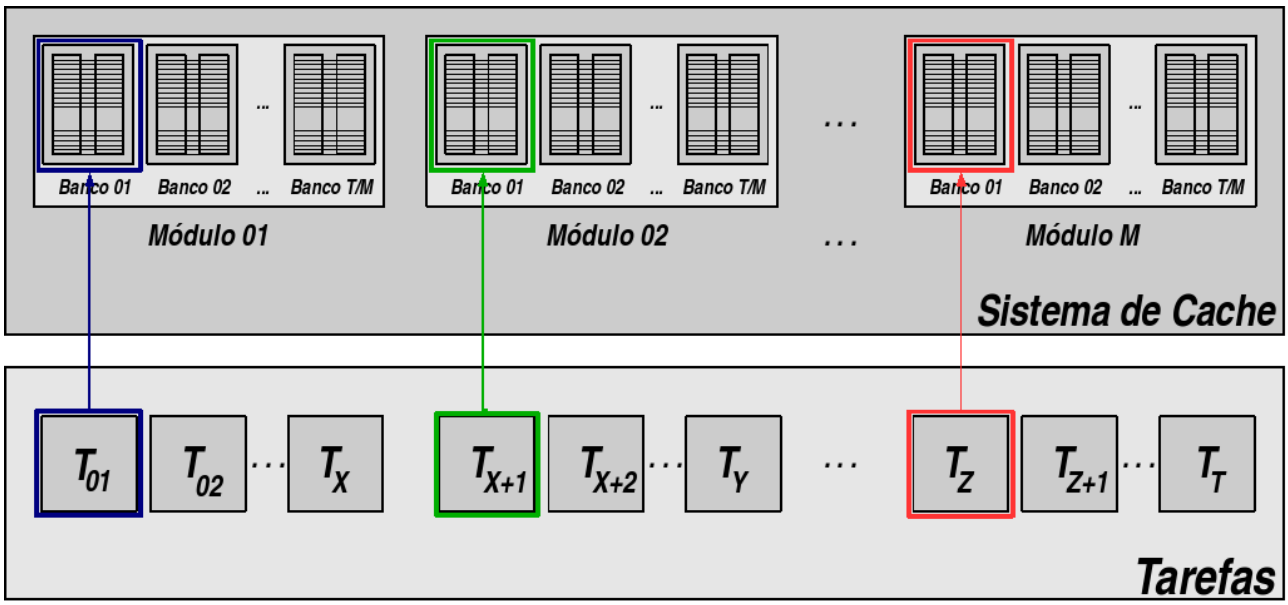

Figura 5.5: Uso dos bancos pelas tarefas

A adequação do modelo de consumo fez surgir algumas idéias e possibilidades para a medida do consumo no sistema de *cache* do PowerSMT, mapeando-se a idéia original do Sim-Wattch e considerando a necessidade de computar o consumo relacionado à lógica agora envolvida no sistema de *cache*.

A primeira idéia considera o consumo de todo o sistema a cada acesso independente do qual módulo e de qual banco estiver sendo acessado. Uma segunda abordagem considera o consumo total do módulo cujo banco está sendo acessado mais o consumo imposto pela lógica de decisão sobre o conjunto dos módulos. A terceira abordagem computa somente o consumo do banco acessado, considerando também o consumo gasto com decodificação na lógica de acesso. A última abordagem, pertencente ao nível hierárquico mais baixo, vislumbra a possibilidade de desligamento por completo ou uma redução do consumo para as linhas que não estiverem sendo acessadas, considerando obviamente as questões de preservação ou não da informação contida no conjunto (linha) da *cache*.

Esta última abordagem motivou a realização dos testes de freqüência de acesso aos conjuntos da *cache*, da distância e freqüência dos acessos aos conjuntos e o teste do impacto do desligamento por completo e do modo de baixo consumo das linhas acessadas com pouca freqüência. Esta técnica é denominada por *Drowsy Caches* (FLAUTNER *et al*, 2002B). Nela, as linhas menos acessadas são colocadas em *drowsy mode* (modo baixo consumo), que nos testes denominamos de estado de inatividade.

Alguns testes foram realizados, inclusive a implementação de um algoritmo, que foi chamado *Algoritmo para a Redução do Consumo nas Linhas (ARCL)* cuja a idéia se baseava em

poupar energia colocando as linhas da cache não pertencentes ao conjunto de trabalho imediato em modo de inatividade. Eram deixadas ativas somente as linhas corrente e sua próxima adjacente, porém, devido à parcela de consumo imposto pelas *wordlines* não ser muito significativa os ganhos obtidos com as linhas colocadas em estado de inatividade eram facilmente gastos na ocorrência de uma penalidade.

5.4.2 Definição do Algoritmo para a Redução do Consumo nos Bancos (ARCB)

Não obtendo resultados significativos com o ARCL cogitou-se a sua adaptação e aplicação aos bancos dos módulos do sistema de *cache*, chamando-o de ARCB. A idéia do algoritmo baseia-se em um mecanismo bem simples de maximização de disponibilidade e minimização de consumo. Neste caso pensando-se na organização dos módulos e na forma como os acessos são feitos aos seus bancos por meio da política *round robin*, tem-se que a necessidade básica é que o banco da tarefa corrente esteja disponível no momento em que ocorra o acesso. Isto nos sugere que enquanto os demais bancos não estiverem sendo acessados podem permanecer no modo de baixo consumo, o que denominamos de modo de inatividade, no qual o consumo é reduzido de forma a não perder as informações e que algum ganho em relação ao consumo seja obtido devido à redução de atividade.

Para adaptar o ARCL para bancos, e propor o ARCB como mencionado, modificou-se o modelo para a lógica envolvida na decisão entre módulos e bancos já que a seqüência de acessos aos bancos é totalmente previsível. O Quadro 5.1 apresenta a idéia do ARCB em pseudo-código.

Quadro 5.1: Pseudo-código do ARCB

```
ARCB(){
  Inicialização {
      Colocar todas os bancos de cada módulo em Modo de Inatividade;
  }
  Execução {
      Colocar o banco anterior em modo de inatividade;
      Ativar o banco corrente e o próximo adjacente no acesso (se
necessário, pois pode ser que já estejam ativados);
      Penalizar quando o banco corrente não estiver ativo na tentativa de
acesso;
  }
}
```

No ARCB o próximo banco a ser ativado será considerado o seguinte, o próximo adjacente, pois como mencionado a seqüência de acesso é totalmente previsível, sendo ela linear. Como não existem saltos na seqüência de acesso aos bancos, utilizando o ARCB não ocorre o problema que ocorreria com o ARCL, o problema de em um determinado momento se ter um número de linhas ativadas como sendo as próximas adjacentes e que não foram colocadas novamente em modo de inatividade no acesso às suas próximas, pois devido aos saltos as próximas não eram as adjacentes

conforme determinado pelo algoritmo.

O ARCB deve garantir que sempre se terá apenas dois bancos ativos independente do número de módulos, poderia-se pensar em se manter um banco ativo em cada módulo, mas analisando-se a seqüência de acesso, a forma como ocorre, basta mantermos o banco acessado pela tarefa corrente e o próximo banco adjacente na seqüência de acessos. Desta forma, o cálculo do consumo dos bancos fica determinístico, pois o algoritmo impõe que o conjunto de bancos ativos, independente de a quais módulos pertencem, terá sempre a cardinalidade dois.

Para a obtenção de uma estimativa do consumo dos bancos pode-se matematicamente calcular a redução do consumo propiciado pela aplicação do ARCB, utilizando-se o valor do consumo médio de cada banco da *cache* de instruções e o número de ciclos é possível calcular o consumo considerando que em cada ciclo somente dois bancos estariam ativos.

5.4.3 Unidades Funcionais

Outra necessidade de alteração no modelo de consumo foi a inclusão do cálculo de valores de referência e a atualização deles para os tipos variados de unidades funcionais do PowerSMT herdadas do SS_SMT. Os tipos de unidades funcionais podem ser: heterogênea (*hetero*), homogênea (*homo*) ou compacta (*compact*). Os recursos aqui são as unidades funcionais e seguem a definição estabelecida pela estrutura mostrada no Quadro 5.2.

Quadro 5.2: Descritor de recursos

```
/* resource descriptor */
struct res_desc {
  char *name;                 /* name of functional unit */
  int quantity;               /* total instances of this unit */
  int busy;                   /* non-zero if this unit is busy */
  struct res_template {
    int class;                /* matching resource class: insts with this
                                    resource class will be able to
execute
                                 on this unit */
    int oplat;                /* operation latency: cycles until result
                                 is ready for use */
    int issuelat;                /* issue latency: number of cycles
before
                                 another operation can be issued on this
                                  resource */
    struct res_desc *master; /* master resource record */
  } x[MAX_RES_CLASSES];
};
```

Cada unidade funcional recebe um nome, uma quantidade e através de uma variável de controle é indicado seu estado ocupado (*busy*) ou disponível. Em cada definição de unidade

funcional, existe uma sub-estrutura na qual são especificadas as classes de instruções que serão tratadas ou executadas pela unidade funcional, bem como a latência da operação (número de ciclos até que o resultado esteja pronto para o uso) e latência de *issue* (número de ciclos antes que outra operação possa ser emitida para tal unidade funcional).

Usando-se a definição da estrutura de recursos são criadas as estruturas organizacionais *default* para os tipos de organização heterogênea, homogênea e compacta. Sendo elas as seguintes:

- ***Organização Heterogênea***

O Quadro 5.3 apresenta a definição da organização para as unidades funcionais do tipo heterogênea. Essa é a definição padrão encontrada também no SimpleScalar.

Quadro 5.3: Definição da Organização Heterogênea

```
struct res_desc fu_hetero_config[5] =
{ { "integer-ALU", 4, 0,
      {
            { IntALU, 1,1 }
      }
  },
  { "integer-MULT/DIV", 1, 0,
      {
            { IntMULT, 3, 1 },
            { IntDIV, 20, 19 }
      }
  },
  { "memory-port", 2, 0,
      {
            { RdPort, 1, 1 },
            { WrPort, 1, 1 }
      }
  },
  { "FP-ALU", 4, 0,
      {
            { FloatADD, 2, 1 },
            { FloatCMP,2, 1 },
            { FloatCVT, 2, 1 }
      }
  },
  { "FP-MULT/DIV", 1, 0,
      {
            { FloatMULT, 4, 1 },
            { FloatDIV, 12, 12 },
            { FloatSQRT, 24, 24 }
      }
  },
};
```

Um resumo da definição da organização heterogênea apresentada no Quadro 5.3 pode ser resumida na Tabela 5.1. Tem-se a identificação da unidade funcional, a quantidade disponível e as classes de instruções que são atendidas pelo tipo de unidade.

Tabela 5.1: Organização Heterogênea Padrão

Unidade Funcional	Quantidade	Classes de Instruções Atendidas
Integer-ALU	4	IntALU
integer-MULT/DIV	1	IntMULTI, IntDIV
Memory-port	2	RdPort, WrPort
FP-ALU	4	FloatADD, FloatCMP, FloatCVT
FP-MULTI/DIV	1	FloatMULT, FloatDIV, FloatSQRT

A Figura 5.6 pode ser vista como uma representação visual da organização heterogênea para as unidades funcionais.

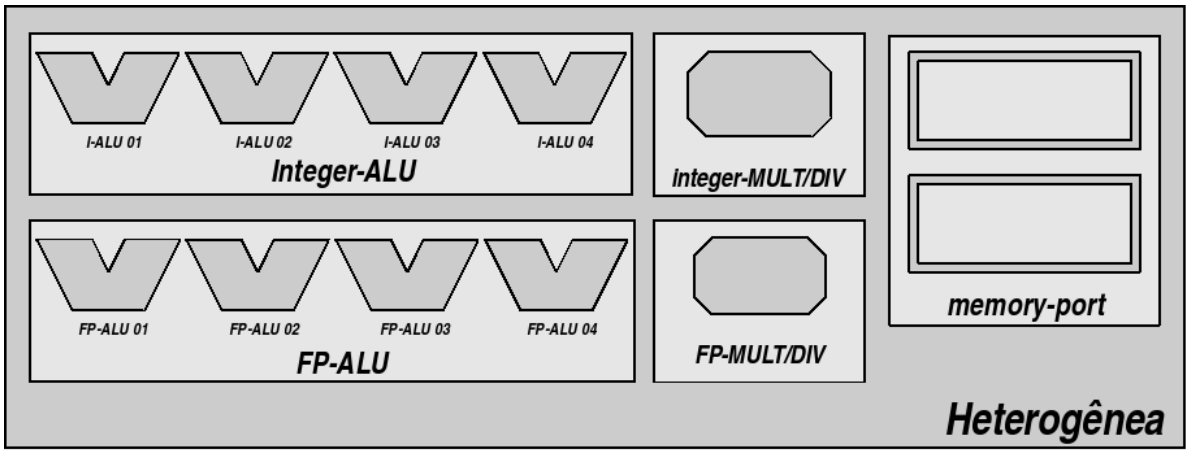

Figura 5.6: Representação da Organização Heterogênea

- ***Organização Homogênea***

A organização homogênea se caracteriza por tratar todas as classes de instruções em um único tipo de unidade funcional. A definição da organização homogênea para as unidades funcionais pode ser vista no Quadro 5.4.

Quadro 5.4: Definição da Organização Homogênea

```
struct res_desc fu_homo_config[1] =
{ { "homogeneous", 4, 0,
      {
            { IntALU, 1, 1 },
            { IntMULT, 3, 1 },
            { IntDIV, 20, 19 },
            { FloatADD, 2, 1 },
            { FloatCMP, 2, 1 },
            { FloatCVT, 2, 1 },
            { FloatMULT, 4, 1 },
            { FloatDIV, 12, 12 },
            { FloatSQRT, 24, 24 },
            { RdPort, 1, 1 },
```

```
            { WrPort, 1, 1 }
        },
    }
};
```

A definição apresentada no Quadro 5.4, pode ser resumida na Tabela 5.2, na qual tem-se a quantidade padrão de unidades para a organização homogênea e a lista de classes de instruções que são tratadas por ela.

Tabela 5.2: Organização Homogênea

Unidade Funcional	Quantidade	Classes de Instruções Atendidas
Homogeneous	4	IntALU, IntMULT, IntDIV, FloatADD, FloatCMP, FloatCVT, FloatMULT, FloatDIV, FloatSQRT, RdPort, WrPort.

A organização homogênea das unidades funcionais pode ser entendida conforme a representação apresentada na Figura 5.7.

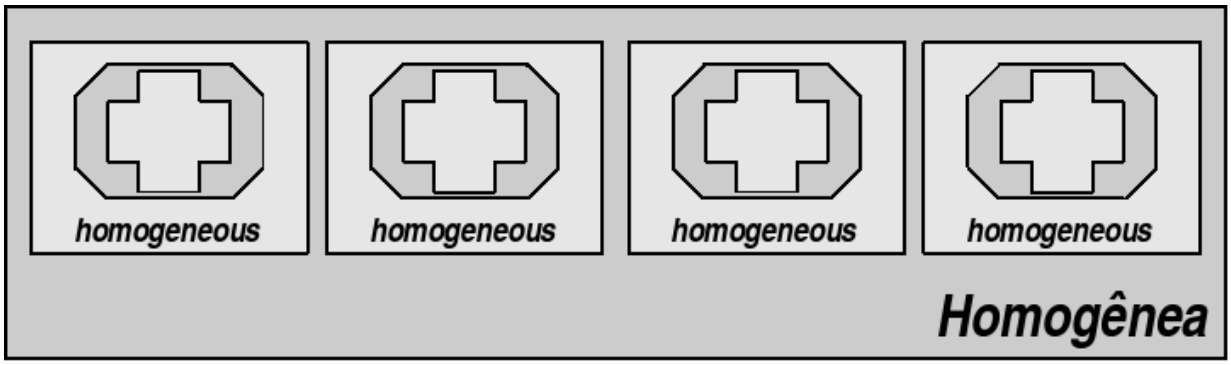

Figura 5.7: Representação da Organização Homogênea

- ***Organização Compacta***

A organização compacta faz alguns agrupamentos entre classes de instruções que podem ser atendidas por algumas unidades funcionais, possibilitando que instruções de aritmética inteira possam tanto ser atendidas pela unidade específica de aritmética inteira quanto pela unidade que faz divisão e multiplicação. O Quadro 5.5 apresenta a definição da organização compacta.

Quadro 5.5: Definição da Organização Compacta

```
struct res_desc fu_compact_config[4] =
{ { "Integer-ALU", 1, 0,
        {
            { IntALU, 1, 1 }
        }
    },
    { "Div/Mult", 1, 0,
        {
            { IntALU, 1, 1 },
            { FloatADD, 2, 1 },
```

```
                { FloatCMP, 2, 1 },
                { FloatCVT, 2, 1 },
                { IntMULT, 3, 1 },
                { IntDIV, 20, 19 },
                { FloatMULT, 4, 1 },
                { FloatDIV, 12, 12 },
                { FloatSQRT, 24, 24 }
        }
    },
    { "Memory-Port", 1, 0,
        {
                { RdPort, 1, 1 },
                { WrPort, 1, 1 }
        }
    },
    { "FP-ALU", 1, 0,
        {
                { FloatADD, 2, 1 },
                { FloatCMP, 2, 1 },
                { FloatCVT, 2, 1 }
        }
    },
};
```

A Tabela 5.3 apresenta os valores padrões para as quantidades e as classes de instruções que são atendidas pela organização compacta.

Tabela 5.3: Organização Compacta

Unidade Funcional	Quantidade	Classes de Instruções Atendidas
Integer-ALU	1	IntALU
Div/Mult	1	IntALU, FloatADD, FloatCMP, FloatCVT, IntMULT, IntDIV, FloatMULT, FloatDIV, FloatSQRT.
Memory-port	1	RdPort, WrPort
FP-ALU	1	FloatADD, FloatCMP, FloatCVT

A Figura 5.8 representa a organização compacta para as unidades funcionais.

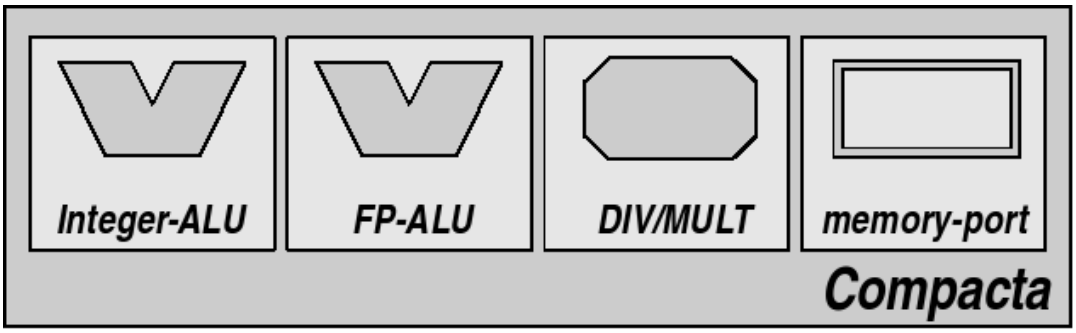

Figura 5.8: Representação da Organização Compacta

- ***Organização Original no SimpleScalar***

O modelo de consumo original prevê somente o consumo para a organização das unidades funcionais existente no SimpleScalar, a definição da organização original para as unidades funcionais é apresentada no Quadro 5.6.

Quadro 5.6: Definição da Organização Original

```
struct res_desc fu_config[] =
{
  { "integer-ALU", 4, 0,
    {
      { IntALU, 1, 1 }
    }
  },
  { "integer-MULT/DIV", 1, 0,
    {
      { IntMULT, 3, 1 },
      { IntDIV, 20, 19 }
    }
  },
  { "memory-port", 2, 0,
    {
      { RdPort, 1, 1 },
      { WrPort, 1, 1 }
    }
  },
  { "FP-adder", 4, 0,
    {
      { FloatADD, 2, 1 },
      { FloatCMP, 2, 1 },
      { FloatCVT, 2, 1 }
    }
  },
  { "FP-MULT/DIV", 1, 0,
    {
      { FloatMULT, 4, 1 },
      { FloatDIV, 12, 12 },
      { FloatSQRT, 24, 24 }
    }
  },
};
```

A organização padrão original é a mesma definida para a organização heterogênea no PowerSMT. A relação entre unidade funcional (FU) e as classes de instruções tratadas (CIT) é apresentada na Tabela 5.4.

Tabela 5.4: Relação entre Unidades Funcionais e Classes de Instruções Tratadas

FU/ CIT	Int ALU	Int MULT	Int DIV	Float ADD	Float CMP	Float CVT	Float MULT	Float DIV	Float SQRT	Rd Port	Wr Port
integer-ALU	X										
integer-MULT/DIV		X	X								

FU/ CIT	Int ALU	Int MULT	Int DIV	Float ADD	Float CMP	Float CVT	Float MULT	Float DIV	Float SQRT	Rd Port	Wr Port
memory-port										X	X
FP-ALU				X	X	X					
FP-MULT/DIV							X	X	X		
DIV/MULT	X	X	X	X	X	X	X	X	X		
homogeneous	X	X	X	X	X	X	X	X	X	X	X

Pode-se observar ainda a composição entre as unidades funcionais. A homogênea trata todas as classes de instruções. Considerando que o pior caso nesta composição seria a replicação dos componentes que realizam cada operação, neste caso a homogênea teria os circuitos responsáveis pelas outras classes de instruções, seria como a homogênea ter cada uma das outras unidades funcionais em sua composição. Com base nestes dados pode-se compor também os consumos dessas unidades funcionais, desde a mais básica nessa escala de composição até a mais complexa que teria todas as outras unidades funcionais na escala de composição. As composições são apresentadas na Tabela 5.5.

Tabela 5.5: Composição das Organizações das Unidades Funcionais

FU/ CIT	integer-ALU	integer-MULT/ DIV	memory-port	FP-ALU	FP-MULT/DIV	DIV/ MULT
integer-ALU	X					
integer-MULT/DIV		X				
memory-port			X			
FP-ALU				X		
FP-MULT/DIV					X	
DIV/MULT	X	X		X	X	X
homogeneous	X	X	X	X	X	

Na Tabela 5.6 são apresentadas as unidades funcionais possíveis em relação à cada organização, sendo mostradas quais estão presentes em cada organização para ter-se uma idéia de como é a composição desses recursos.

Tabela 5.6: Unidades Funcionais por Organização

Heterogênea	Homogênea	Compacta
integer-ALU •IntALU		**integer-ALU** •IntALU
integer-MULT/DIV •IntMULT •IntDIV		
memory-port •RdPort •WrPort		**memory-port** •RdPort •WrPort

Heterogênea	Homogênea	Compacta
FP-ALU •FloatADD •FloatCMP •FloatCVT		**FP-ALU** •FloatADD •FloatCMP •FloatCVT
FP-MULT/DIV •FloatMULT •FloatDIV •FloatSQRT		
	homogeneous •IntALU •IntMULT •IntDIV •FloatADD •FloatCMP •FloatCVT •FloatMULT •FloatDIV •FloatSQRT •RdPort •WrPort	
		Div/Mult •IntALU •IntMULT •IntDIV •FloatADD •FloatCMP •FloatCVT •FloatMULT •FloatDIV •FloatSQRT

Cada instrução é identificada pelos *flags* do seu *opcode,* como pode ser visto na contagem dos acessos às unidades funcionais do modelo de consumo original, apresentado no Quadro 5.7.

Quadro 5.7: Contagem original dos acessoas às unidades funcionais

```
/* Wattch -- ALU access Wattch-FIXME (different op types) also spread out
power of multi-cycle ops */
alu_access[sn]++;
if((MD_OP_FLAGS(rs->op) & (F_FCOMP)) == (F_FCOMP))
      falu_access[sn]++;
```

```
else
      ialu_access[sn]++;
```

Assim cada instrução pode ser identificada e por conseguinte identifica-se também qual unidade funcional atende ou executa a classe de instruções a que ela pertence. Os *flags* que identificam o tipo de instrução e que podem ser encontrados no *opcodes* são apresentados no Quadro 5.8.

Quadro 5.8: *Flags* possíveis para os *opcodes*

```
/* instruction flags */
#define F_ICOMP         0x00000001 /* integer computation */
#define F_FCOMP         0x00000002 /* FP computation */
#define F_CTRL          0x00000004 /* control inst */
#define F_UNCOND        0x00000008 /* unconditional change */
#define F_COND          0x00000010 /* conditional change */
#define F_MEM           0x00000020 /* memory access inst */
#define F_LOAD          0x00000040 /* load inst */
#define F_STORE         0x00000080 /* store inst */
#define F_DISP          0x00000100 /* displaced (R+C) addr mode */
#define F_RR            0x00000200 /* R+R addr mode */
#define F_DIRECT        0x00000400 /* direct addressing mode */
#define F_TRAP          0x00000800 /* traping inst */
#define F_LONGLAT       0x00001000 /* long latency inst (for sched) */
#define F_DIRJMP        0x00002000 /* direct jump */
#define F_INDIRJMP      0x00004000 /* indirect jump */
#define F_CALL          0x00008000 /* function call */
#define F_FPCOND        0x00010000 /* FP conditional branch */
#define F_IMM           0x00020000 /* instruction has immediate operand */
```

Utilizando-se a macro *MD_OP_FLAGS(rs->op)* e comparando-se os *flags* presentes nos *opcodes* com os *flags* previstos, determina-se qual a classe da instrução e pela organização das unidades funcionais qual a unidade funcional que trata a classe de instrução recuperada.

O mapeamento do *opcode* em unidade funcional é feito pela estrutura de enumeração criada com a redefinição da macro *DEFINST(...)* e pela injeção das chamadas à macro, que é também utilizadas em outros locais do código com outros propósitos, tais como decodificação. O Quadro 5.9 mostra como a estrutura de mapeamento é inicializada com a injeção do código presente no arquivo *machine.def*.

Quadro 5.9: Mapeamento de *opcodes* em unidades funcionais

```
/* enum md_opcode -> enum md_fu_class, used by performance simulators */
enum md_fu_class md_op2fu[OP_MAX] = {
  FUClass_NA, /* NA */
#define DEFINST(OP,MSK,NAME,OPFORM,RES,FLAGS,O1,O2,I1,I2,I3) RES,
#define DEFLINK(OP,MSK,NAME,MASK,SHIFT) FUClass_NA,
#define CONNECT(OP)
#include "machine.def"
};
```

O quadro 5.10 apresenta como o mapeamento é feito para algumas instruções, baseando-se na utilização da macro *DEFINST* combinada com a injeção das chamadas feita a ela pelo arquivo *machine.def*, sendo neste caso suprimida a definição da implementação da instrução, útil quando o arquivo *machine.def* é utilizado para a montagem da máquina de execução.

Quadro 5.10: Exemplos do pré-processamento das chamadas à macro DEFINST

```
DEFINST( OP,  MSK,  NAME, OPFORM,    RES,  FLAGS,O1,O2,I1,I2,I3) RES
DEFINST(NOP, 0x00, "nop",     "",  IntALU,F_ICOMP,DNA,DNA,DNA,DNA, DNA)

DEFINST(ADD, 0x40, "add","d,s,t",  IntALU, F_ICOMP, DGPR(RD), DNA,
DGPR(RS), DGPR(RT), DNA)

DEFINST(MULT,0x46, "mult", "s,t", IntMULT, F_ICOMP|F_LONGLAT, DHI,  DLO,
DGPR(RT), DGPR(RS), DNA)

DEFINST(DIV, 0x48,  "div", "s,t",  IntDIV, F_ICOMP|F_LONGLAT,DHI, DLO,
DGPR(RT), DGPR(RS), DNA)

DEFINST(AND_,0x4e,  "and","d,s,t", IntALU, F_ICOMP, DGPR(RD), DNA,
DGPR(RS), DGPR(RT), DNA)
```

Com a injeção do conteúdo do arquivo *machine.def* por meio da diretiva *include*, a estrutura de mapeamento de *opcodes* em unidades funcionais é montada, como pode ser visto no trecho de código apresentado no quadro 5.10, o código resultante para as chamadas à macro *DEFINST(...)* para esse caso resultam no conteúdo *RES* que nas chamadas traz a classe a qual a instrução pertence. Assim os *opcodes* das instruções NOP, ADD, AND serão mapeados para a classe IntALU, o da MULT será mapeado para IntMULT e da DIV para IntDIV. Neste trecho de código do quadro 5.10 podem ser observados ainda, os *FLAGS* pertencentes aos *opcodes* e que podem ser utilizados na identificação de cada instrução. As classes de unidades funcionais são definidas no Quadro 5.11.

Quadro 5.11: Classes de Unidades Funcionais

```
/* function unit classes, update md_fu2name if you update this definition
*/
enum md_fu_class {
  FUClass_NA = 0,/* inst does not use a functional unit */
  IntALU,        /* integer ALU */
  IntMULT,       /* integer multiplier */
  IntDIV,        /* integer divider */
  FloatADD,      /* floating point adder/subtractor */
  FloatCMP,      /* floating point comparator */
  FloatCVT,      /* floating point<->integer converter */
  FloatMULT,     /* floating point multiplier */
  FloatDIV,      /* floating point divider */
  FloatSQRT,     /* floating point square root */
  RdPort,        /* memory read port */
  WrPort,        /* memory write port */
  NUM_FU_CLASSES /* total functional unit classes */
};
```

Para o mapeamento entre *opcode* e classe de unidade funcional que executa a instrução, foi previamente definida, em *machine.def,* a macro *MD_OP_FUCLASS(OP)* que retorna o elemento da estrutura md_op2fu na posição OP (md_op2fu[OP]) como mostrado no Quadro 5.12.

Quadro 5.12: Macro MD_OP_FUCLASS

```
/* enum md_opcode -> enum md_fu_class, used by performance simulators */
#define MD_OP_FUCLASS(OP)   (md_op2fu[OP])
extern enum md_fu_class md_op2fu[];
```

Está prevista também outras macros tais como *MD_FU_NAME(FU)* que retorna uma descrição da unidade funcional previamente armazenada na estrutura md_fu2name[FU]. Conforme o código apresentado no Quadro 5.13.

Quadro 5.13: Macro MD_FU_NAME

```
/* enum md_fu_class -> description string */
#define MD_FU_NAME(FU)      (md_fu2name[FU])
extern char *md_fu2name[];
```

O mapeamento do *opcode* nos *flags*, como apresentado no Quadro 5.14, é também feito através de uma estrutura, a md_op2flags, a qual é acessada pelo subscrito OP passado através da macro MD_OP_FLAGS(OP) que é o resultado de md_op2flags[OP].

Quadro 5.14: Macro MD_OP_FLAGS

```
/* enum md_opcode -> opcode flags, used by simulators */
#define MD_OP_FLAGS(OP)          (md_op2flags[OP])
extern unsigned int md_op2flags[];
```

Para os cálculos do consumo é necessária a contabilização do número de acessos às unidades funcionais, não sendo necessário uma comparação por instrução, recuperando seus *flags* para tal verificação. Assim, ao invés de comparar o tipo de instrução através dos *flags* como é feito via código, pode-se acumular os acessos à classe de unidade funcional que atende ou executa a instrução, através do mapeamento ***opcode-fu*** (md_op2fu[OP]) considerando que a quantidade de classes é bem menor e pelo número de instruções (cerca de 123) o código gerado terá uma legibilidade maior utilizando-se as classes ao invés das instruções. Definiu-se uma matriz de contadores fus_access[process_num][NUM_FU_CLASSES], cuja atualização ficou bem simples como pode ser visto no Quadro 5.15.

Quadro 5.15: Contagem de acessos às unidades funcionais por classes

```
fus_access[sn][MD_OP_FUCLASS(rs->op)]++;
```

Os acessos a cada classe de unidade funcional é contabilizado em separado, o consumo de cada unidade funcional deve agrupar os números de acessos registrados para as classes de

instruções que a unidade funcional executa. Os cálculos ficam distribuídos primariamente por unidade funcional e não por organização (hetero, homo ou compacta). Por exemplo, o consumo das unidades funcionais integer-MULT/DIV deve considerar os acessos realizados pelas instruções das classes IntMULT e IntDIV, desta forma o número de acessos a esse tipo de unidade é composto por fus_access[sn][***IntMULT***] + fus_access[sn][***IntDIV***]. Estendendo-se a idéia para todos os outros tipos de unidades funcionais temos a composição do número de acessos a cada unidade funcional, conforme a classe de instruções por ela tratada.

No modelo de consumo original eram calculados somente os consumos para as unidades funcionais de lógica e aritmética de inteiro (ialu) e de ponto flutuante (fpalu). Sendo a organização das unidades funcionais herdada do SS_SMT diferente, foram mapeados os demais cálculos para as outras unidades funcionais conforme a organização que assumem e as instruções que tratam. O Quadro 5.16 apresenta a inicialização dos valores de referência para as unidades funcionais antes do seu cálculo efetivo.

Quadro 5.16: Inicialização dos valores de referência antes do cálculo

```
/* PowerSMT Added */
/* Initialize fu power values */
power->ialu_power            = 0.0;
power->imult_div_power       = 0.0;
power->mem_port_power         = 0.0;
power->fpalu_power           = 0.0;
power->fpmult_div_power      = 0.0;
power->divmult_power          = 0.0;
power->homo_power            = 0.0;
```

O cálculo dos valores de referência para as unidades funcionais de acordo com a organização é apresentado no Quadro 5.17 para a organização heterogênea.

Quadro 5.17: Cálculo dos valores de referência para organização heterogênea

```
/* Calculate the values according fu organization */
if (!mystricmp(fu_pool_type, "hetero")){
   power->ialu_power = res_ialu * I_ADD;/* integer-ALU, instructions
classes treated (ICT): IntALU */
   // res_imult -> integer-MULT/DIV.quantify.
   power->imult_div_power = res_imult * MAX(I_MULT, I_DIV);/* integer-
MULT/DIV, integer multiplier/divider, ICT: IntMULT, IntDIV */
   power->mem_port_power = res_memport * (RD_PORT + WR_PORT);/* memory-
port, ICT: RdPort, WrPort */
   power->fpalu_power = res_fpalu * F_ADD;    /* FP-ALU, floating point
adder/subtractor, ITC: FloatADD, FloatCMP, FloatCVT */
   power->fpmult_div_power = res_fpmult * (MAX(F_MULT, F_DIV) + F_SQRT);
/* FP-MULT/DIV, floating point multiplier/divider, ICT: FloatMULT,
FloatDIV, FloatSQRT */
}
```

O cálculo para os valores de referência para as unidades funcionais quando assumida a organização compacta é apresentado no Quadro 5.18.

Quadro 5.18: Cálculo dos valores de referência para organização compacta

```
else {
 if (!mystricmp(fu_pool_type, "compact")){
    power->ialu_power = res_ialu * I_ADD; /* integer-ALU, instructions
classes treated (ICT): IntALU */
    power->divmult_power    = res_divmult * (MAX(F_DIV_MULT,
MAX(MAX(I_MULT, F_MULT), MAX(I_DIV, F_DIV))) + MAX(I_ADD, F_ADD) + F_CMP
+ F_CVT  + F_SQRT); /* DIV/MULT, divider/multiplier, ICT: IntALU,
FloatADD, FloatCMP, FloatCVT, IntMULT, IntDIV, FloatMULT, FloatDIV,
FloatSQRT */
    power->mem_port_power = res_memport * (RD_PORT + WR_PORT); /* memory-
port, ICT: RdPort, WrPort */
    power->fpalu_power = res_fpalu * F_ADD; /* FP-ALU, floating point
adder/subtractor, ITC: FloatADD, FloatCMP, FloatCVT */
 }
```

Para organização homogênea, os cálculos são realizados tal como apresentado no Quadro 5.19. Entre componentes que realizam a mesma operação aritmética, mas modalidades inteira ou de ponto flutuante foi colocado o consumo do componente como sendo o máximo entre o consumo para realizar uma operação inteira e uma de ponto flutuante. Assumiu-se por exemplo, que em um somador que realiza adições de ponto flutuante, o circuito que trata a parte inteira possa fazer adições na aritmética inteira, sendo o consumo assumido para o componente como o máximo entre realizar as duas operações.

Quadro 5.19: Cálculo dos valores de referência para organização homogênea

```
else { /* homogeneous fus */
    power->homo_power = res_homo * (MAX(I_ADD, F_ADD) + F_CMP + F_CVT +
MAX(I_MULT, F_MULT) + MAX(I_DIV, F_DIV) + F_SQRT + RD_PORT + WR_PORT);
     /* homogeneous: ICT: IntALU, IntMULT, IntDIV, FloatADD, FloatCMP,
FloatCVT,IntMULT, IntDIV, FloatMULT, FloatDIV, FloatSQRT, RdPort, WrPort
*/
     }
}
```

As constantes utilizadas nos cálculos foram incluídas junto às existentes herdadas do modelo de consumo original, e alguns valores de consumo sugeridos baseados nos valores existentes.

Quadro 5.20: Consumos das unidades funcionais

```
POWER_SCALE  = (GEN_POWER_SCALE * NORMALIZE_SCALE * Powerfactor) ;
I_ADD        = ((.37 - .091) * POWER_SCALE) ;
I_ADD32      = (((.37 - .091)/2)  *POWER_SCALE) ;
I_MULT16     = ((.31-.095)*POWER_SCALE) ;
I_SHIFT      = ((.21-.089)*POWER_SCALE) ;
I_LOGIC      = ((.04-.015)*POWER_SCALE) ;
F_ADD        = ((1.307-.452)*POWER_SCALE) ;
```

```
F_MULT        = ((1.307-.452)*POWER_SCALE) ;
```

Ao conjunto de valores básicos fornecidos pelo modelo de consumo foram adicionados novos valores para outras unidades funcionais não previstas no modelo original, que já existiam no SS_SMT e foram herdadas pelo PowerSMT. O Quadro 5.21 mostra os novos valores sendo inicializados com base em valores já existentes no modelo.

Quadro 5.21: Consumos adicionados para as unidades funcionais não previstas no modelo

```
/* Added by PowerSMT */
/* FIXME: I don't have details about this values */
I_MULT       = (I_MULT16);
I_DIV        = (I_MULT);
F_DIV        = (F_MULT);
F_DIV_MULT   = (F_DIV + F_MULT);

/* FIXME: I don't have details about this values */
RD_PORT      = (0.0);
WR_PORT      = (0.0);
F_CMP        = (0.0);
F_CVT        = (0.0);
F_SQRT       = (0.0);
/* Added by PowerSMT */
```

Ainda no Quadro 5.21, na inicialização dos valores para as outras unidades funcionais presentes no PowerSMT, nota-se que foram feitas algumas aproximações, considerando para as novas unidades funcionais alguns valores já existentes ou a combinação deles. Por exemplo, para a unidade de divisão inteira (I_DIV) considerou-se o valor aproximado de consumo equivalente a uma de multiplicação inteira (I_MULT), para a divisão de ponto flutuante (F_DIV) a mesma idéia foi seguida, utilizando-se o mesmo valor da multiplicação (F_MULT) e a unidade que realiza divisão e multiplicação em ponto flutuante (F_DIV_MULT) foi inicializada com a soma dos valores das unidades que realizam essas operações individualmente.

O consumo relacionado ao *clock* também é calculado, o modelo já utilizava os valores apresentados no Quadro 5.22 para fazer o cálculo pela função ***total_clockpower****(die_length)*.

Quadro 5.22: Inicialização dos valores de *clock* para cada tipo de unidade funcional

```
I_ADD_CLOCK      = (.091*POWER_SCALE);
I_MULT_CLOCK     = (.095*POWER_SCALE);
I_SHIFT_CLOCK    = (.089*POWER_SCALE);
I_LOGIC_CLOCK    = (.015*POWER_SCALE);
F_ADD_CLOCK      = (.452*POWER_SCALE);
F_MULT_CLOCK     = (.452*POWER_SCALE);
```

Foram acrescentados os consumos relacionados ao *clock* para as outras unidades funcionais não previstas no modelo de consumo original, os valores apresentados no Quadro 5.23 também são

considerados no cálculo do consumo do *clock*.

Quadro 5.23: Inicialização dos valores de *clock* para cada tipo de unidade funcional não previsto no modelo

```
/* Added by PowerSMT */
/* FIX ME, I don't have details about this values */
I_DIV_CLOCK      = (I_MULT_CLOCK);
F_DIV_CLOCK      = (F_MULT_CLOCK);
/* Considerando que uma unidade que faca divisao e multiplicacao, o
consumo sendo o maximo entre os consumos de cada unidade individualmente
*/
F_DIV_MULT_CLOCK = MAX(F_DIV_CLOCK, F_MULT_CLOCK);

/* FIX ME, I don't have details about this values */
RD_PORT_CLOCK    =     (0.0);
WR_PORT_CLOCK    =     (0.0);
F_CMP_CLOCK      =     (0.0);
F_CVT_CLOCK      =     (0.0);
F_SQRT_CLOCK     =     (0.0);
/* Added by PowerSMT */
```

5.4.4 RUU e LSQ

As duas formas de organização permitidas pelo PowerSMT para as estações de reserva (RUU/LSQ) são: distribuídas ou compartilhadas. Podendo essa escolha ser feita através do parâmetro *ruulsq:type*, apresentado no Quadro 5.24.

Quadro 5.24: Parâmetro para o tipo de *ruu/lsq*

```
-ruulsq:type: it defines <distributed> or <shared> ruu/lsq for
the slots
```

A verificação foi feita e quanto ao mapeamento da RUU e da LSQ não há problema pois o consumo dessas estruturas é calculado corretamente nas duas abordagens. Em ambos os casos cada *slot* registra seu consumo baseado nos acessos que realizou, sendo o consumo global a soma de todos os consumos individuais dos *slots*.

Por exemplo, caso tenhamos estruturas individuais de 16 entradas para cada *slot*, o consumo é calculado para cada slot sobre os acessos que faz à essa estrutura, e quando a estrutura é compartilhada, mesmo que cada slot acesse 4 entradas (em um cenário de 4 slots), o acesso estará sendo feito às suas respectivas 4 entradas, mas ainda a uma estrutura de 16 entradas. Desta forma, a modificação organizacional da RUU e da LSQ não implicou em alterações no modelo de consumo.

5.4.5 Fila de Busca - *Instruction Fetch Queue* (IFQ)

No modelo de consumo do Sim-Wattch não estava previsto o cálculo para a fila de instruções da busca (IFQ), então foi feito um mapeamento para essa estrutura, baseando na forma como foi feito o

mapeamento para as outras estruturas.

O cálculo do consumo da IFQ foi baseado no cálculo para o banco de registradores, feito as devidas alterações e adequações para o cenário da IFQ. O tamanho considerado para a estrutura é o dado pelas variáveis *ruu_ifq_size* representando o número de linhas e *data_width* representando o número de colunas em bits, assumindo portanto que cada linha possui o tamanho de 64 bits, o tamanho de uma instrução.

Quadro 5.25: Extensão do Modelo com cálculo adicionado para a Fila de Instruções

```
/* PowerSMT Added */
if (verbose)
 fprintf(stderr,"Instruction Fetch Queue (ifq-fetch_data) power
stats\n");

predeclength = ruu_ifq_size * (RegCellHeight + 3 * ruu_fetch_width *
WordlineSpacing);

wordlinelength = data_width *(RegCellWidth + 6 * ruu_fetch_width *
BitlineSpacing);

bitlinelength = ruu_ifq_size * (RegCellHeight + 3 * ruu_fetch_width *
WordlineSpacing);

// array_decoder_power(rows, cols, predeclength, rports, wports, cache)
// read ports: assumed decode_width, because decode stage that read
instructions of fetch_data.
// write ports: assumed fetch_width, because fetch stage that write
instructions on fetch_data.
power->ifq_decoder = array_decoder_power(ruu_ifq_size,
                                         data_width,
                                        predeclength,
                                         ruu_decode_width,
                                         ruu_fetch_width, cache);
power->ifq_wordline = array_wordline_power(ruu_ifq_size,
                                          data_width,
                                          predeclength,
                                          ruu_decode_width,
                                          ruu_fetch_width,cache);
power->ifq_bitline =  array_bitline_power(ruu_ifq_size,
                                         data_width,
                                         predeclength,
                                         ruu_decode_width,
                                         ruu_fetch_width, cache);
power->ifq_senseamp = 0;
```

5.4.6 Buffer de Reordenação - *Reorder Buffer* (ROB)

Para o *Buffer* de Reordenação (ROB), o modelo de consumo do Sim-Wattch apresenta as variáveis para conter os valores de referência, porém seus valores não eram calculados. Talvez até mesmo motivado pelo fato do SimpleScalar utilizar uma única estrutura que representa a Unidade de Atualização de Registradores (RUU), as estações de reserva e o ROB.

Optou-se por calcular o valor para o consumo do ROB, utilizando-se uma estrutura com a mesma quantidade de linhas da RUU, porém assumiu-se que as entradas do ROB são de 32 *bits* representando possíveis resultados que possam ser armazenados nessa estrutura.

Quadro 5.26: Extensão do Modelo com cálculo adicionado para o ROB

```
if (verbose)
      fprintf(stderr,"Reorder Buffer power stats\n");

predeclength = RUU_size * (RegCellHeight + 3 * ruu_commit_width *
WordlineSpacing);
// data_width is 64, assuming elements in reorder buffer are 32 bits.
wordlinelength =
(data_width / 2) * (RegCellWidth + 6 * ruu_commit_width *
BitlineSpacing);
bitlinelength =
RUU_size * (RegCellHeight + 3 * ruu_commit_width * WordlineSpacing);

power->reorder_decoder = array_decoder_power(RUU_size, (data_width / 2),
predeclength, 2*ruu_commit_width, ruu_commit_width, cache);
power->reorder_wordline = array_wordline_power(RUU_size, (data_width /
2), wordlinelength, 2*ruu_commit_width, ruu_commit_width, cache);
power->reorder_bitline = array_bitline_power(RUU_size, (data_width / 2),
bitlinelength, 2*ruu_commit_width, ruu_commit_width, cache);
/* no senseamps in reg file structures (only caches) */
power->reorder_senseamp =0;
```

5.4.7 Seletores

Pelo fato da arquitetura SMT ter recursos privativos para cada tarefa, sendo que em várias partes da arquitetura é necessário selecionar qual estrutura utilizar, o cômputo do consumo dessas lógicas de seleção (seletores) para os recursos do *slot* também foram considerados. Mapeou-se essas estruturas como decodificadores utilizados no acesso de cada recurso que não é compartilhado pelos *slots*.

Esse decodificadores foram inseridos para selecionarem o banco do sistema de cache que está sendo acessado pelo *slot*, para selecionar a fila de busca (*ifq*) na qual as instruções pertencentes a um determinado *slot* devem ficar, para selecionarem o banco de registradores, a Unidade de Atualização de Registradores ou *Register Update Unit* (RUU), tabela de renomeação ou *Register Alias Table* (RAT) e o *buffer* de reordenação ou *Reorder Buffer* (ROB). Os consumos relacionados a cada seletor foram adicionados aos consumos das estruturas às quais servirão de seletor.

5.5 Organização Final da Ferramenta

O esquema de funcionamento arquitetural do PowerSMT pode ser visualizado na Figura 5.9, a qual mostra a organização do sistema de cache em instruções em bancos. A Figura 5.9 mostra o sistema de *cache*, que aparece como sendo apenas um módulo com vários bancos, porém outras organizações são possíveis, como por exemplo a definição de vários módulos contendo um banco

cada. Outra observação a ser feita com relação ao esquema de funcionamento é o sobre as estações de reserva (RUU) que podem ser compartilhadas ou distribuídas.

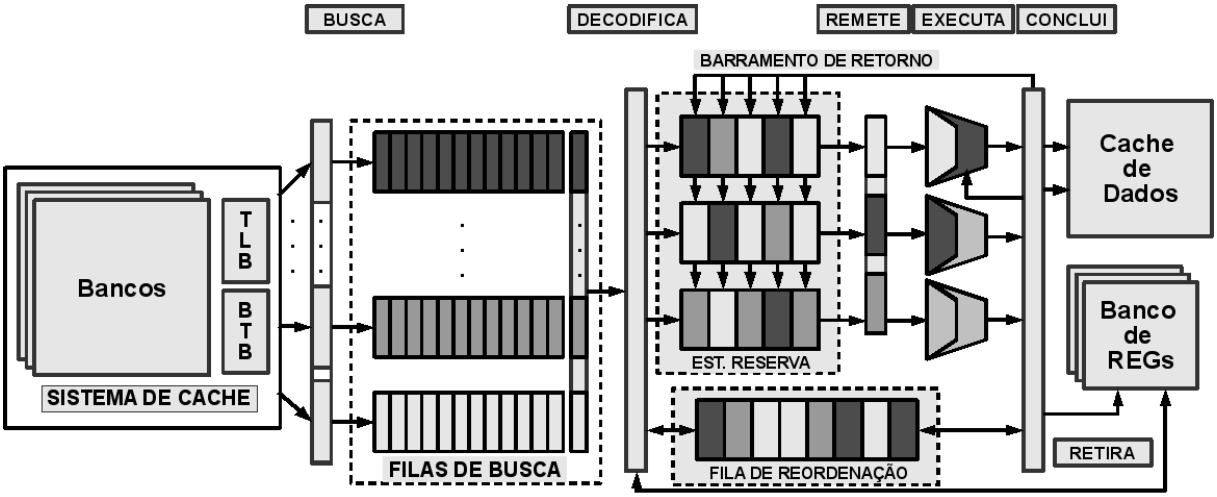

Figura 5.9: Uma possível Organização SMT para o PowerSMT

A integração entre PowerSMT e o modelo de consumo de potência é feita por chamadas de funções na inicialização (sim_init) e no bloco principal do simulador (sim_main) conforme apresentado na Figura 5.10. Na inicialização, os parâmetros e as variáveis do modelo de consumo são inicializados pelas funções *power_init()* e *init_all_variables_power_model()*.

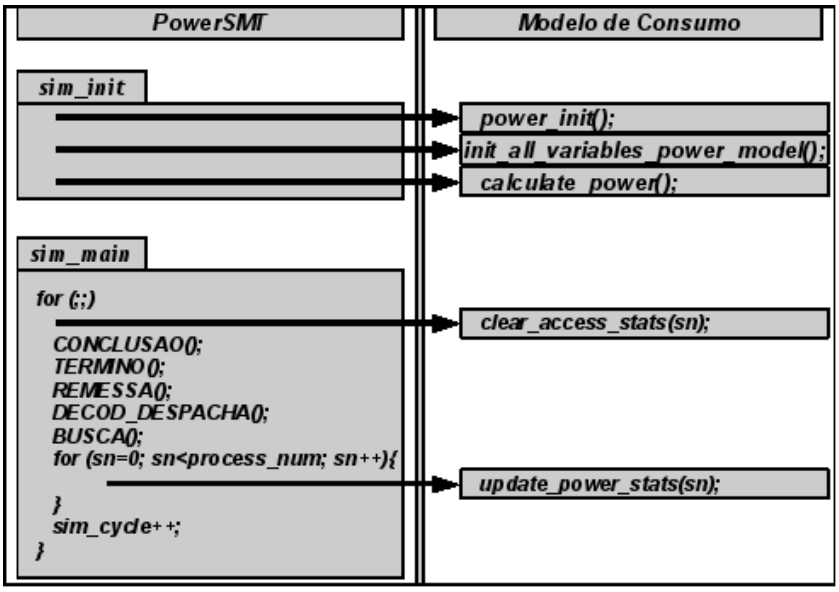

Figura 5.10: Integração PowerSMT com o Modelo de Consumo

Em seguida os valores da potência estática para as estruturas são calculados na função *calculate_power()*, esses valores por serem utilizados nas atualizações dos consumos de cada estrutura, foram chamados de valores de referência de consumo. Ainda na figura 5.10 pode-se notar que no início de cada ciclo arquitetural, controlado pelo bloco principal do programa, os contadores de acessos das estruturas são reinicializados pela função *clear_access_stats()* e as funções que representam os estágios arquiteturais (Conclusão, Término, Remessa, Decodificação e Despacho e

Busca) são executadas, fazendo com que as estruturas sejam acessadas conforme o fluxo de execução de cada tarefa, e com que os contadores de acessos registrem esses acessos. Ao final do ciclo arquitetural, a função *update_power_stats()* é chamada para que os consumos de cada tarefa sejam atualizados conforme os acessos feitos às estruturas arquiteturais.

5.6 Utilização da Ferramenta

O PowerSMT segue o mesmo padrão de entrada de parâmetros do SimpleScalar, foi modificada a forma de passagem de alguns parâmetros, alguns parâmetros foram acrescentados desde o SS_SMT, e outros parâmetros foram acrescentados pelo PowerSMT ao conjunto de parâmetros originais do SimpleScalar. A configuração de *caches*, que ao invés de passar o número de conjuntos, desde o SS_SMT, passa o tamanho da cache em Kbytes. É possível ainda, por parâmetro, forçar a taxa de acerto do previsor de desvios, e escolher a tecnologia do processo de desenvolvimento para que o cálculo do consumo das caches pela biblioteca CACTI seja conforme a tecnologia desejada, característica essa baseada na forma de entrada de parâmetros do HotLeakage que também utilizou o modelo de consumo do Sim-Wattch.

A execução da ferramenta pode ser disparada através de linhas de comando semelhantes à apresentada no Quadro 5.27.

Quadro 5.27: Linha de comando genérica para o PowerSMT

```
powerSMT <Ti> <Tf> -config <ArquivoConfig> <ArquivoDefinicaoTarefas>
```

A linha de comando apresentada no Quadro 5.27 representa a linha de comando genérica para a execução de *benchmarks* como tarefas simultâneas no PowerSMT, padrão herdado do SS_SMT. Os parâmetros podem ser entendidos da seguinte forma:

- ***T_i e T_f:*** Definem o intervalo de *benchmarks,* número inicial e número final dentro do ArquivoDefinicaoTarefas.

- ***-config:*** Parâmetro original aceito pelo SimpleScalar, o qual espera um arquivo com todos os parâmetros de configuração, utilizado em substituição à opção de todos os parâmetros serem passados na mesma linha de comando. Esse parâmetro espera um arquivo de configuração, neste caso um nome de arquivo qualquer, aqui representado por <*ArquivoConfig*>.

- ***<ArquivoDefinicaoTarefas>:*** É um arquivo que contém as linhas de comandos dos *benchmarks* que serão indexadas pelo intervalo definido nos parâmetros T_i e T_f. No anexo tem um exemplo de arquivo de definição de tarefas para os *benchmarks* do SPEC2000.

A opção *-config* pode ser substituída na linha de comando por todas as outras opções que estariam dentro do arquivo de configuração, alguns dos parâmetros aceitos pelas linhas de comando ou em arquivos de configurações que foram adicionados pelo SS_SMT e por fim pelo PowerSMT em adição aos parâmetros existentes no Sim-Wattch são listados na Tabela 5.7, os demais parâmetros aceitos como entrada pelo PowerSMT e um exemplo de resultado de execução são apresentados no anexo.

Tabela 5.7: Parâmetros do PowerSMT

Parâmetro	Descrição
-fu:type <string>	Parâmetro para a seleção do tipo de organização das unidades funcionais, podendo assumir os valores: hetero, homo:n ou compact. Valor padrão: hetero.
-imodules:num <uint>	Número de módulos para a cache de instruções.
-ruulsq:type <string>	Organização da RUU e LSQ, podendo ser compartilhadas (*shared*) ou distribuídas (*distributed*).
-res:homo <int>	Número de unidades funcionais do tipo homogênea, usado quando o parâmetro -fu:type possui o valor homo.
-res:divmult <int>	Número de unidades funcionais que fazem divisão e multiplicação. Tendo sentido somente para quando a organização das unidades funcionais for compacta.
-technology <string>	Parâmetro do processo tecnológico, valores complementares ao modelo de consumo original foram obtidos na ferramenta HotLeakage. Os valores assumidos por esse parâmetro podem ser TECH_800, TECH_400, TECH_350, TECH_250, TECH_180, TECH_130, TECH_100, TECH_070 ou nenhum quando a ferramenta assume o valor padrão.

5.7 Considerações Finais

Neste capítulo apresentou-se o PowerSMT, ferramenta desenvolvida para a avaliação de desempenho e do consumo de potência em arquiteturas SMT. Foram apresentadas questões relacionadas ao desenvolvimento da ferramenta partindo-se da integração inicial do núcleo arquitetural herdado do SS_SMT com o modelo de consumo extraído do Sim-Wattch, passando pela validação inicial, até a atualização da CACTI e adequação do modelo de consumo para suportar as diferenças apresentadas pelo novo núcleo arquitetural, tais como sistema de caches, organização das unidades funcionais e da RUU/LSQ. Nas seções finais, são apresentadas a organização final do PowerSMT e algumas questões relacionadas à forma de utilização da ferramenta, tais como os parâmetros de entrada que são aceitos.

Capítulo 6

Experimentos e Resultados

Neste capítulo são apresentados os experimentos que foram realizados com o PowerSMT com o intuito de mostrar a variabilidade de experimentos que a ferramenta possibilita, bem como a avaliação de algumas das principais características arquiteturais simuladas pela ferramenta. A tecnologia adotada nestes experimentos foi a de 0,07μm, sendo esse parâmetro passado para a biblioteca CACTI 4.0 e considerado nas otimizações das estruturas das caches feitas por seus modelos. Nos experimentos foram utilizados *benchmarks* do SPEC2000 (SPEC, 2000) para a composição dos grupos de execução denominados SMT-X, sendo que X indica o número de tarefas executadas simultaneamente, podendo ser 1, 2, 4, 8 e 16. Os *benchmarks* são apresentados na Tabela 6.1, sendo 8 deles de aritmética de números inteiros e 8 de aritmética de números de ponto-flutuante. Nos experimentos foram estabelecidos os seguintes valores para número de instruções executadas como aquecimento da arquitetura e número máximo de instruções executadas por simulação de 50 milhões e 300 milhões, respectivamente.

Tabela 6.1: Lista dos *benchmarks* utilizados

Número	Benchmark	Inteiro/Ponto Flutuante
01	gzip.ss-little.O3	Inteiro
02	vpr.ss-little.O3	Inteiro
03	cc1.ss-little.O3	Inteiro
04	mcf.ss-little.O3	Inteiro
05	parser.ss-little.O3	Inteiro
06	vortex.ss-little.O3	Inteiro
07	bzip2.ss-little.O3	Inteiro
08	twolf.ss-little.O3	Inteiro

Número	Benchmark	Inteiro/Ponto Flutuante
09	wupwise.ss-little.O3	Ponto Flutuante
10	swim.ss-little.O3	Ponto Flutuante
11	mgrid.ss-little.O3	Ponto Flutuante
12	applu.ss-little.O3	Ponto Flutuante
13	mesa.ss-little.O3	Ponto Flutuante
14	art.ss-little.O3	Ponto Flutuante
15	equake.ss-little.O3	Ponto Flutuante
16	ammp.ss-little.O3	Ponto Flutuante

Muito embora o PowerSMT permita que cada *benchmark* possa ser avaliado individualmente, considerou-se neste trabalho a avaliação de cada grupo como um todo. Além disso, várias combinações foram testadas para o mesmo grupo, para diluir coincidências e situações oportunistas. Por isso, os resultados de cada grupo foram construídos como a média entre as todas as combinações simuladas para o respectivo grupo. Por exemplo, a arquitetura SMT-4 foi testada com 4 combinações diferentes de 4 *benchmarks*. Cada combinação foi executada e obteve um resultado. Então, a média aritmética simples dos resultados obtidos com todas as 4 combinações foi utilizada como resultado do grupo SMT-4.

Com exceção do primeiro experimento, que foi realizado com um configuração básica, nos outros experimentos variou-se os parâmetros relacionados à estrutura foco do teste e o restante do *pipeline* foi configurado em grandes proporções a ponto de não ser ele o motivo de um possível baixo desempenho. Nestes experimentos a idéia foi a de avaliar uma arquitetura ampla, restringindo e monitorando as configurações dos pontos específicos que são os alvos de cada teste experimental. Os resultados em termos de consumo de potência são apresentados segundo os estilos de *clock* do modelo de consumo herdado do Sim-Wattch (CC1, CC2 e CC3).

A idéia é validar a ferramenta, mostrar a sua importância e a variabilidade de experimentos e pontos arquiteturais que permite analisar. Nas seções seguintes são apresentados os experimentos realizados e os resultados obtidos, o primeiro experimento é uma execução dos diversos modos SMT-X sobre uma arquitetura básica, o segundo é relacionado à *cache* de instruções, o terceiro é sobre as unidades funcionais na organização heterogênea e o quarto avalia o tamanho da janela de instruções (RUU/LSQ). Em todos procurou-se confrontar a configuração das estruturas arquiteturais com o desempenho e consumo.

6.1 *Experimento 1:* Avaliação sobre uma arquitetura básica

Nesse experimento elaborou-se uma configuração arquitetural básica no intuito de avaliar o desempenho e o consumo de potência para essa arquitetura, além de avaliar a própria a ferramenta frente a uma situação mais realística, assim tomou-se como base a arquitetura do MIPS R10k.

6.1.1 Descrição do Experimento

O PowerSMT executou todos os modos de execução SMT, com 1, 2, 4, 8 e 16 tarefas sobre a arquitetura proposta. A definição da arquitetura básica utilizada neste experimento é apresentada na Tabela 6.2.

Tabela 6.2: Definição de uma arquitetura básica

Configuração arquitetural básica		
Busca (*Fetch Stage*)	IFQ de 32 entradas Largura de busca: 4 instruções por ciclo	-fetch:width 4 -fetch:ifqsize 32
Decodificação *(Decode Stage)*	Largura de Decodificação: 4 instruções por ciclo	-decode:width 4
Remessa *(Issue)*	Largura de remessa: 4 instruções por ciclo	-issue:inorder FALSE -issue:width 4
Commit	Largura do *commit*: 4 instruções por ciclo	-commit:width 4
Janela de Instruções	32 entradas na RUU 16 entradas na LSQ Organização compartilhada	-ruulsq:type shared -ruu:size 32 -lsq:size 16
Registradores Físicos	32-Int e 32-FP	*hardcoded*
Unidades Funcionais	Organização heterogênea ALU Inteiro: 3 ALU FP: 3 Mult. Int.: 1 Mult. FP: 1 Memory Port: 2	-fu:type hetero -res:ialu 3 -res:imult 1 -res:fpalu 3 -res:fpmult 1 -res:memport 2
Previsor de Desvios	Gshare (2 níveis): 1 entrada no primeiro e 512 entradas no segundo de 2 *bits* de histórico (2 *bits* é o padrão) Registrador de *Shift* : 9 *bits* XOR: 1 BTB: 32 conjuntos, assoc. 1 Tamanho da pilha de endereço de retorno (*Return address stack*): 0 (zero para não ter pilha)	-bpred:type 2lev -bpred:2lev 1 512 9 1 -bpred:btb 32 1 -bpred:ras 0
Sistema de *Caches*	**Número de módulos:** igual ao número de tarefas. ***I-Cache* de nível 1:** 32KB, assoc. 2,	-imodules:num <NumTarefas> -cache:il1 il1:1:32:16:2:l

Configuração arquitetural básica		
	16-*byte block size*, LRU *repl. policy.* ***D-Cache* de nível 1:** 32KB, assoc. 2, 8-*byte block size,* LRU *repl. policy.* ***Caches* de instrução e dado nível 2 unificada:** 512KB, assoc. 2, 32-*byte block size,* LRU *repl. policy.* **TLBS:** 16 entradas, tamanho de página igual a 64, totalmente associativa, LRU *repl. policy.*	-cache:dl1 dl1:1:32:8:2:l -cache:il2 dl2 -cache:dl2 ul2:1:512:32:2:l -tlb:itlb itlb:16:64:1:l -tlb:dtlb dtlb:16:64:1:l
Memória	*64-bit data*	-mem:width 8
Tecnologia	0,07µm	-technology TECH_070

6.1.2 Resultados

Para esse experimento são apresentados o consumo de potência estática para a arquitetura básica, que são os valores calculados no início da execução para cada uma das estruturas, e que são utilizados como valores de referência nos cálculos do consumo dinâmico pelo modelo de consumo para os estilos de *clock* CC1, CC2 e CC3. Os resultados do consumo de potência dinâmica são apresentados somente para o estilo de *clock* CC3, o que mais se aproxima de situações reais como já mencionado. Os valores do consumo de potência estática por estrutura da arquitetura básica são apresentados na Figura 6.1.

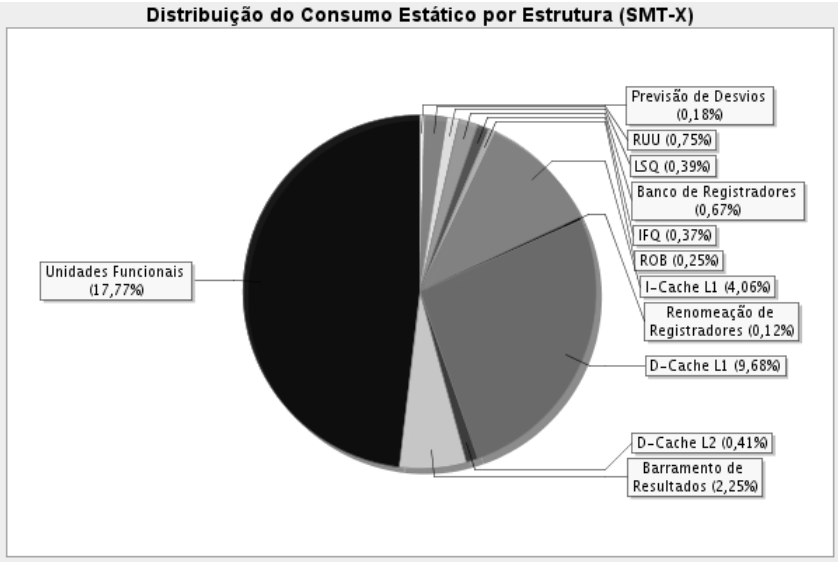

Figura 6.1: Distribuição do consumo estático por estrutura

Os valores de potência estática mostrados no gráfico da Figura 6.1 são para a arquitetura básica, ficando em torno de 72 *Watts*, incluindo o consumo do *clock* que atinge 63,4% desse valor. Considerando o aumento de recursos para a execução de mais tarefas, proporcionado pelas estruturas que são individuais às tarefas, foram feitos os cálculos dos consumos para esse aumento de recursos. Exceto a RUU e a LSQ, que foram utilizadas na organização compartilhada e as

unidades funcionais que são compartilhadas no esquema de *pool* de recursos que seguem portanto a mesma configuração em todos os modos de execução SMT, as outras estruturas são individuais por tarefa e são replicadas nos modos com mais de uma tarefa em execução simultânea.

O consumo estático para cada estrutura nos modos SMT são apresentados na Tabela 6.3, o consumo total apresentado não considera o consumo do *clock*, mostrando somente o percentual do consumo que cada estrutura apresenta na replicação pelo número de tarefas quando necessário.

Tabela 6.3: Previsão de consumo estático por modo SMT

SMT	Prev. Desvios (%)	RUU (%)	LSQ (%)	Banco de REGS (%)	IFQ (%)	ROB (%)	I-Cache L1 (%)	Renomeação (%)	D-Cache L1 (%)	D-Cache L2 (%)	Bar. Resultados (%)	FU's (%)	Total (W)
1	0,18	0,75	0,39	0,67	0,37	0,25	4,05	0,12	9,65	0,41	2,25	17,72	26,51
2	0,18	0,38	0,20	0,67	0,37	0,25	4,07	0,12	9,70	0,41	2,26	17,82	52,20
4	0,18	0,19	0,10	0,67	0,37	0,25	4,08	0,12	9,73	0,41	2,27	17,87	103,58
8	0,18	0,09	0,05	0,67	0,37	0,25	4,09	0,12	9,74	0,41	2,27	17,90	206,33
16	0,18	0,05	0,02	0,67	0,37	0,25	4,09	0,12	9,75	0,41	2,27	17,91	411,83

A Figura 6.2 apresenta a distribuição do consumo dinâmico no modo de execução SMT-1 pelas estruturas que compõem a arquitetura. Pode-se perceber que os maiores consumos estão relacionados às unidades funcionais, às *caches* de nível 1 e ao barramento de resultados, atingindo 12,54% para as unidades funcionais, 8,16% para I-Cache L1, 9,50% para a D-Cache L1 e 2,99% para o barramento de resultados.

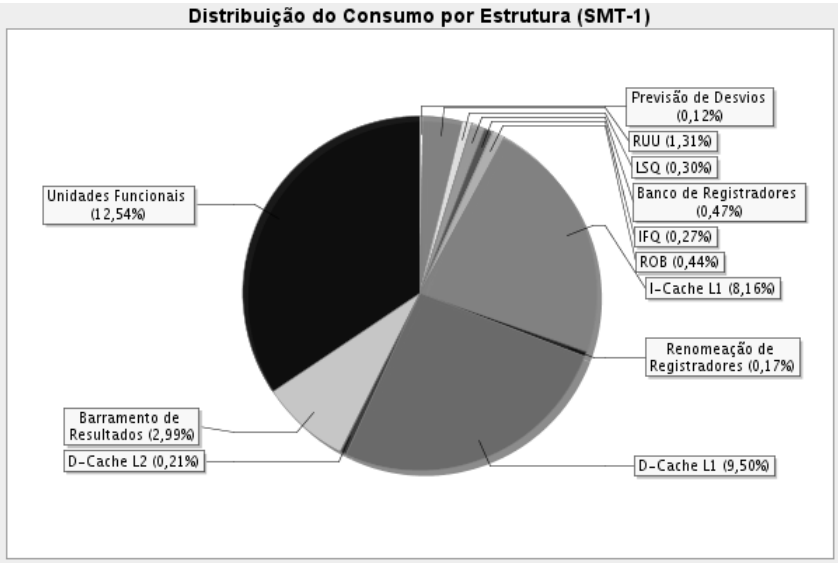

Figura 6.2: Distribuição do consumo por estrutura no estilo CC3 para o modo SMT-1

Na Figura 6.3 são apresentados os valores de consumo dinâmico no modo de execução SMT-2.

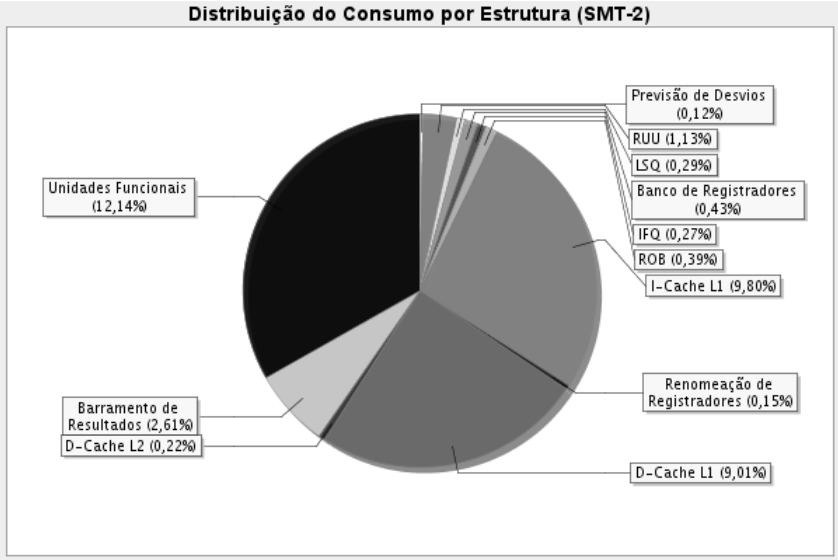

Figura 6.3: Distribuição do consumo por estrutura no estilo CC3 para o modo SMT-2

Na Figura 6.4 são apresentados os valores de consumo dinâmico no modo de execução SMT-4.

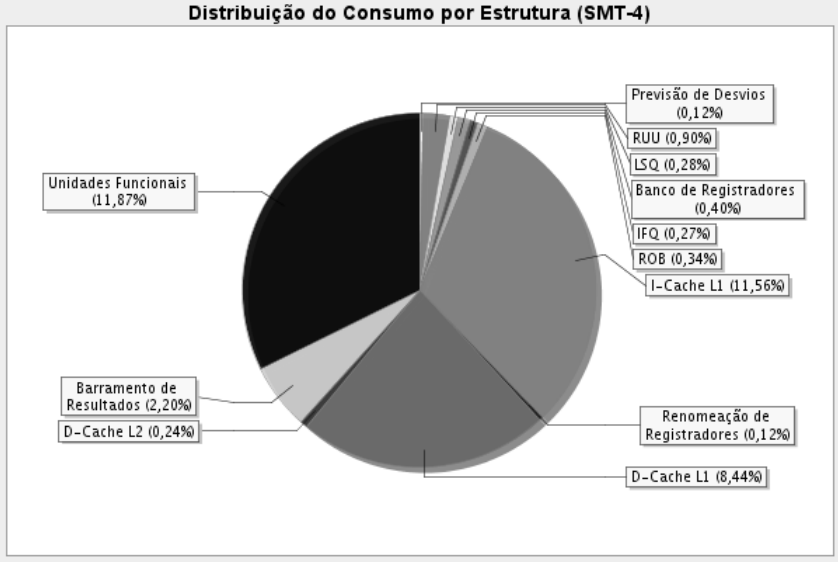

Figura 6.4: Distribuição do consumo por estrutura no estilo CC3 para o modo SMT-4

Na Figura 6.5 são apresentados os valores de consumo dinâmico no modo de execução SMT-8.

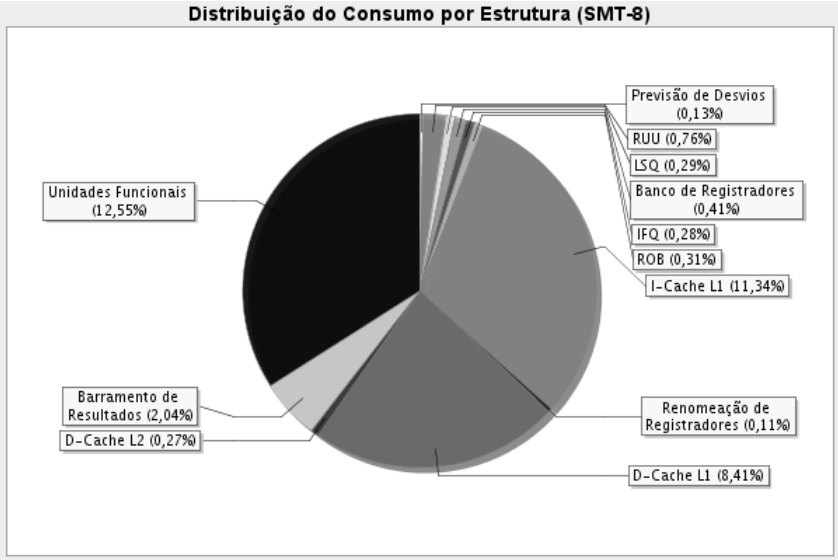

Figura 6.5: Distribuição do consumo por estrutura no estilo CC3 para o modo SMT-8

Na Figura 6.6 são apresentados os valores de consumo dinâmico no modo de execução SMT-16.

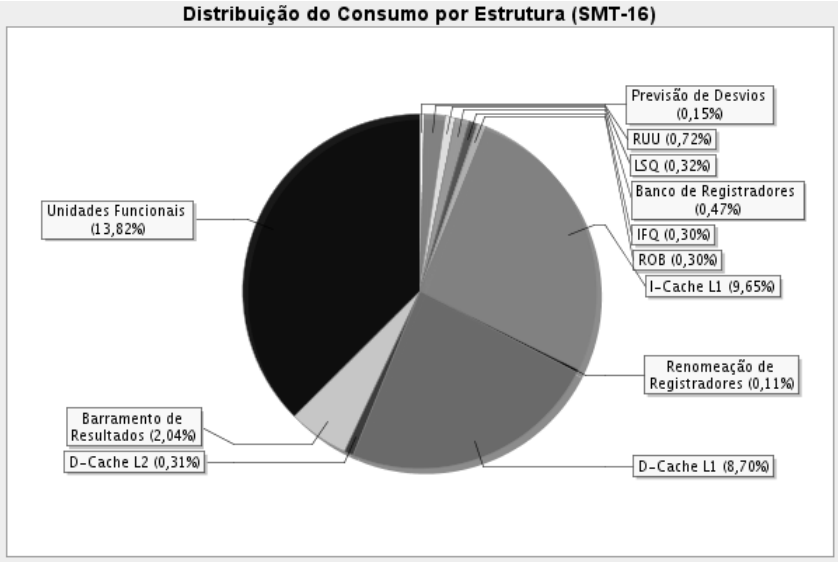

Figura 6.6: Distribuição do consumo por estrutura no estilo CC3 para o modo SMT-16

O consumo por estágios apresentado pelo PowerSMT é formado pela soma dos consumos das estruturas que foram mapeadas como componentes de cada estágio no modelo de consumo. A Tabela 6.4 apresenta os agrupamentos que formam o consumo de cada estágio de acordo com o mapeamento no modelo de consumo.

Tabela 6.4: Agrupamento de consumos por estágio

Estágio	Consumos acumulados
Fetch	Consumo da *cache* de instruções e o consumo com previsão de desvios.
Dispatch	Basicamente o consumo da Renomeação de Registradores.
Issue	Consumo do barramento de resultados, das unidades funcionais, das

Estágio	Consumos acumulados
	caches de dados de nível 1 e 2, da janela de instruções (RUU + lógica de seleção e *wake up*) e consumo da fila de LSQ.

O desempenho das execuções nos modos SMT-X sobre a arquitetura básica, é apresentado na Figura 6.7 juntamente com o ganho do IPC de uma configuração em relação à anterior. Pode-se notar o aumento do IPC com o aumento do número de tarefas. Embora haja um ganho para os modos de execução SMT apresentados na Figura 6.7, a tendência é que esse ganho diminua com o aumento do número de tarefas, chegando ao ponto no qual é necessário aumentar os recursos para que se tenha alguma melhoria no desempenho.

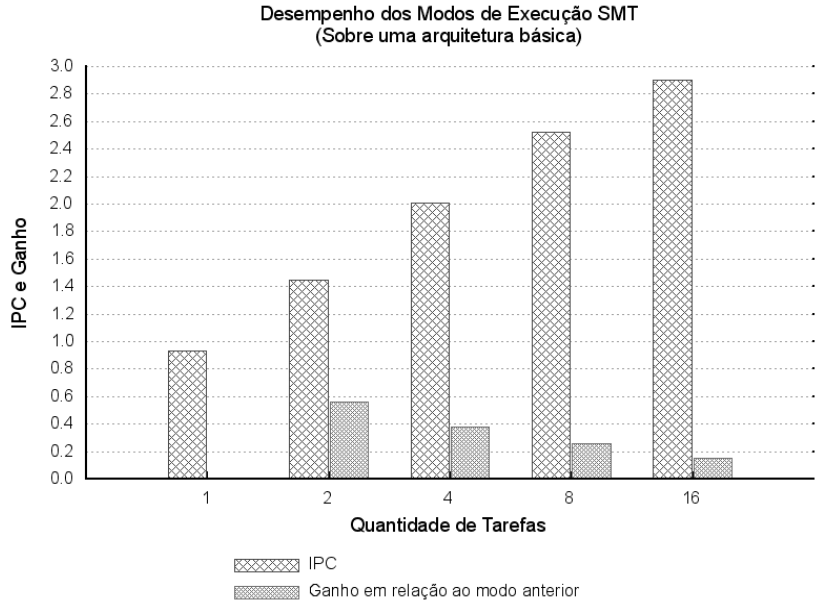

Figura 6.7: Desempenho dos modos SMT-X na execução sobre a arquitetura básica

A Figura 6.8 apresenta a média do tempo de execução em horas para cada modo de execução SMT-X sobre a arquitetura básica definida. É perceptível o aumento do tempo de execução com o aumento do número de tarefas sendo executadas, indo de um pouco mais de 18 minutos para os modos SMT-1 (uma tarefa) até 10 horas, 57 minutos e 49 segundos na execução do modo SMT-16 (dezesseis tarefas). Esse aumento do tempo de execução deve-se à limitação de recursos imposta pela arquitetura básica que foi definida para o experimento. A escassez de recursos, a disputa pela ocupação deles, à medida que ocorre o aumento do número de tarefas em execução, ocasiona essa necessidade de um tempo de execução maior para que as instruções de todas as tarefas sejam executadas.

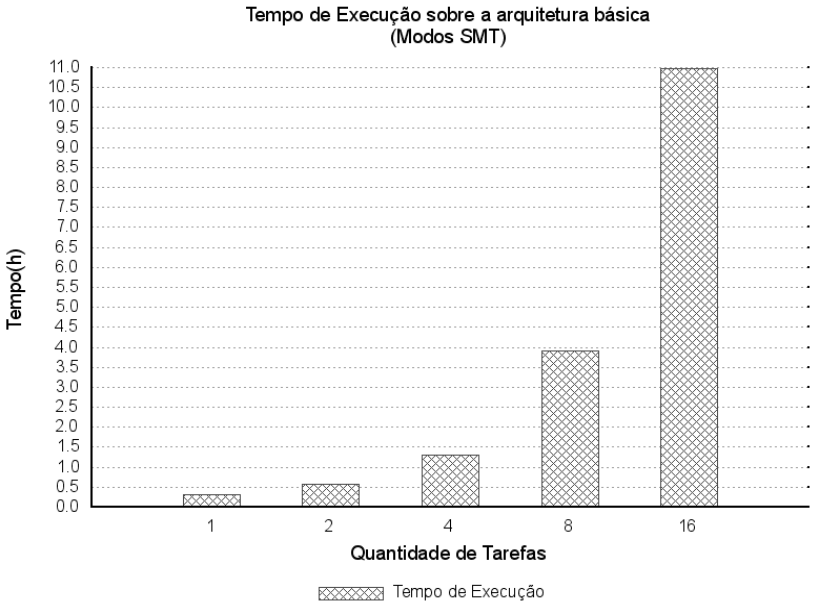

Figura 6.8: Tempo de execução de cada modo SMT-X

6.1.3 Conclusões do Experimento 1

A realização do experimento com uma configuração arquitetural básica mostra como é possível utilizar o PowerSMT em avaliações da arquitetura como um todo, do consumo dinâmico apresentado como seus resultados e relacioná-los com o consumo estático calculado para cada uma das estruturas. É possível ainda que a análise do consumo seja feita por estágios, os quais acumulam o consumo das estruturas que os pertencem conforme mapeamento do modelo de consumo.

Em todos os modos de execução SMT-X, percebe-se que os maiores consumos estão relacionados às unidades funcionais, às *caches* de nível 1 e ao barramento de resultados, e que devido à escassez de recursos proporcionada pelo aumento do número de tarefas em execução em cada modo, o tempo de execução aumenta, como se esperava que acontecesse.

Não é que o modo de execução SMT-16 não compensa, é que a escassez de recursos na configuração básica restringe a exploração para as 16 tarefas. O ganho do desempenho diminui com o aumento do número de tarefas, porém não se pode afirmar em que momento ou com qual configuração SMT acima de 16 tarefas esse ganho chegará a zero, implicando na necessidade de aumentar os recursos para melhorar o desempenho. Nessa versão inicial do PowerSMT, o número máximo de *slots* suportados é igual a 16, possibilitando a execução de no máximo dezesseis tarefas simultâneas, não sendo possível verificar com quantas tarefas o aumento de recurso para essa arquitetura básica seria necessário.

6.2 *Experimento 2:* Avaliação do Sistema de Cache e os impactos no consumo

Sabe-se da importância que as *caches* assumem frente à arquitetura, tornando a avaliação do desempenho e do consumo de potência associado a essas estruturas uma questão importante. Para avaliar o sistema de cache foi feito um experimento relacionado à cache de instruções de nível 1 (*I-Cache*), no qual variou-se o tamanho dos bancos da *cache* e a organização desses bancos em módulos.

6.2.1 Descrição do Experimento

Foram executadas simulações com os modos SMT-1, SMT-2, SMT-4, SMT-8 e SMT-16, nas quais o tamanho dos bancos foi variado de 4KB até 4096KB (4MB). A configuração arquitetural utilizada para esse experimento é apresentada na Tabela 6.5, a qual é uma configuração ampla para os outros pontos arquiteturais, variando somente os parâmetros da *I-Cache.*

Tabela 6.5: Definição da arquitetura ampla utilizada no experimento 2

Configuração arquitetural para o experimento 2		
Busca (*Fetch Stage*)	IFQ de 256 entradas Largura de busca: 256 instruções por ciclo	-fetch:width 256 -fetch:ifqsize 256
Decodificação *(Decode Stage)*	Largura de Decodificação: 256 instruções por ciclo	-decode:width 256
Remessa *(Issue)*	Largura de remessa: 256 instruções por ciclo	-issue:inorder FALSE -issue:width 256
Commit	Largura do *commit*: 256 instruções por ciclo	-commit:width 256
Janela de Instruções	256 entradas na RUU 128 entradas na LSQ Organização distribuída	-ruulsq:type distributed -ruu:size 256 -lsq:size 128
Registradores Físicos	32-Int e 32-FP	*hardcoded*
Unidades Funcionais	Organização heterogênea ALU Inteiro: 32 ALU FP: 32 Mult. Int.: 32 Mult. FP: 32 Memory Port: 32	-fu:type hetero -res:ialu 32 -res:imult 32 -res:fpalu 32 -res:fpmult 32 -res:memport 32
Previsor de Desvios	Previsão em 2 níveis: 1 entrada no primeiro e 4096 entradas no segundo de 2 *bits* de histórico (2 *bits* é o padrão) Registrador de *Shift* : 16 *bits* XOR: 0 BTB: 512 conjuntos, assoc. 4 Tamanho da pilha de endereço de	-bpred:type 2lev -bpred:2lev 1 4096 16 0 -bpred:btb 512 4 -bpred:ras 8

Configuração arquitetural para o experimento 2		
	retorno (*Return address stack*): 8	
Sistema de *Caches*	**Número de módulos:** igual ao número de tarefas. ***I-Cache* de nível 1:** Tamanho variando de 4KB a 4096KB, assoc. 1, 64-*byte block size*, LRU *repl. policy.* ***D-Cache* de nível 1:** 64KB, assoc. 4, 64-*byte block size,* LRU *repl. policy.* ***Caches* de instrução e dado nível 2 unificada:** 1024KB, assoc. 8, 128-*byte block size,* LRU *repl. policy.* **TLBS:** 1024 entradas, tamanho de página 4096, assoc. 4, LRU *repl. policy.*	-imodules:num <número de Tarefas> -cache:il1 il1:1:**KB**:64:1:l -cache:il2 dl2 -cache:dl1 dl1:1:64:64:4:l -cache:dl2 ul2:1:1024:128:8:l -tlb:itlb itlb:1024:4096:4:l -tlb:dtlb dtlb:1024:4096:4:l
Memória	*64-bit data*	-mem:width 8
Tecnologia	0,07µm	-technology TECH_070

6.2.2 Resultados

Os resultados obtidos para o consumo da *I-Cache* são apresentados graficamente em termos de consumo médio de potência em *Watts*. As médias são calculadas sobre todas as simulações com grupos de *benchmarks* diferentes e sobre todas as possíveis configurações usando diferentes números de módulos para o sistema de *cache*. Todos os 3 estilos CC1, CC2 e CC3 foram experimentados. As Figuras 6.9, 6.11 e 6.13 apresentam o consumo para cada um deles. Os mesmos resultados sob a forma de linhas representando as respectivas curvas de crescimento são apresentados nas Figuras 6.10, 6.12 e 6.14. Em todas as figuras é perceptível a variação proporcional do consumo obtido em relação aos tamanhos da *cache*, além do comportamento semelhante entre todas.

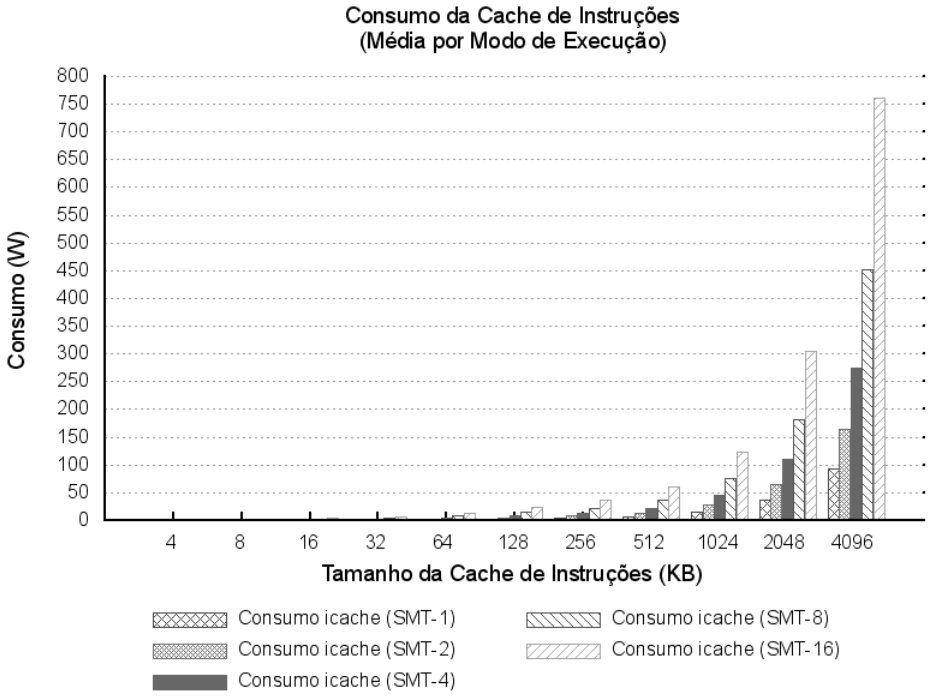

Figura 6.9: Consumo da *I-Cache* no estilo CC1

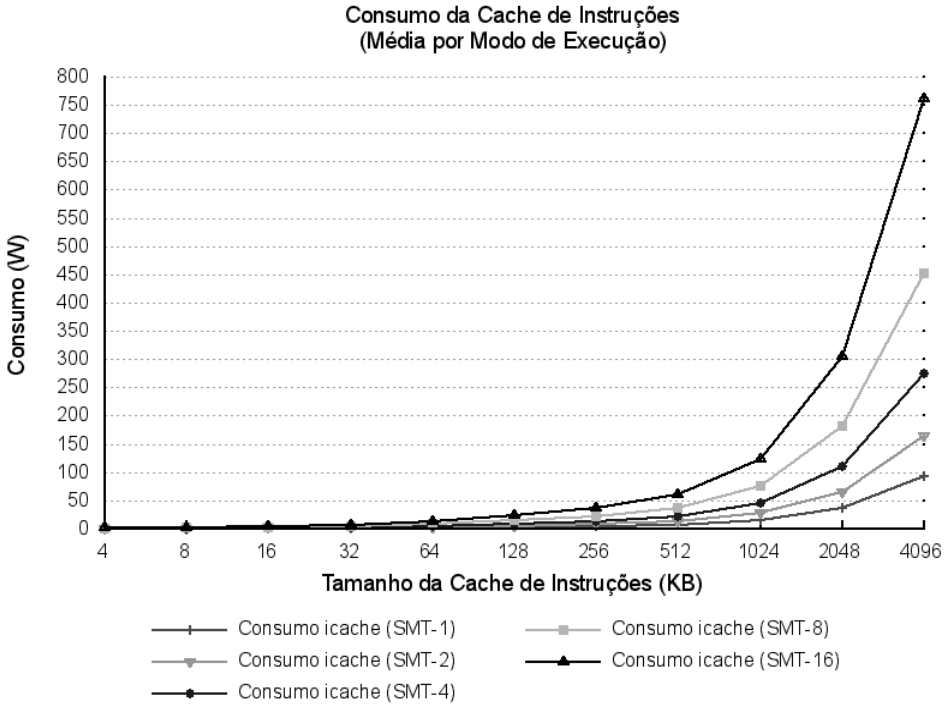

Figura 6.10: Curva do consumo da *I-Cache* no estilo CC1

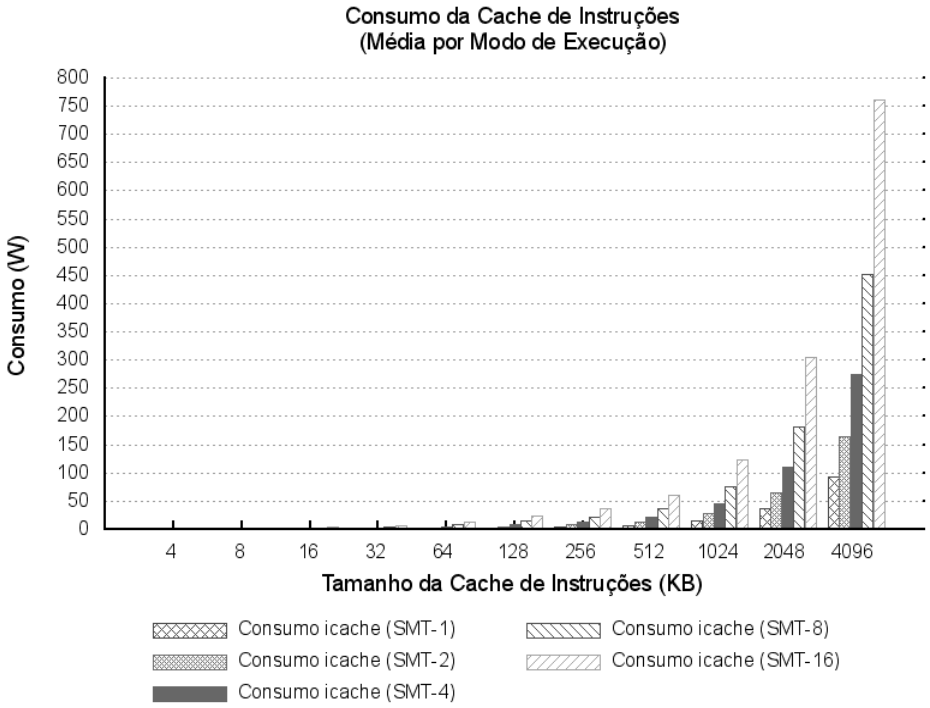

Figura 6.11: Consumo da *I-Cache* no estilo CC2

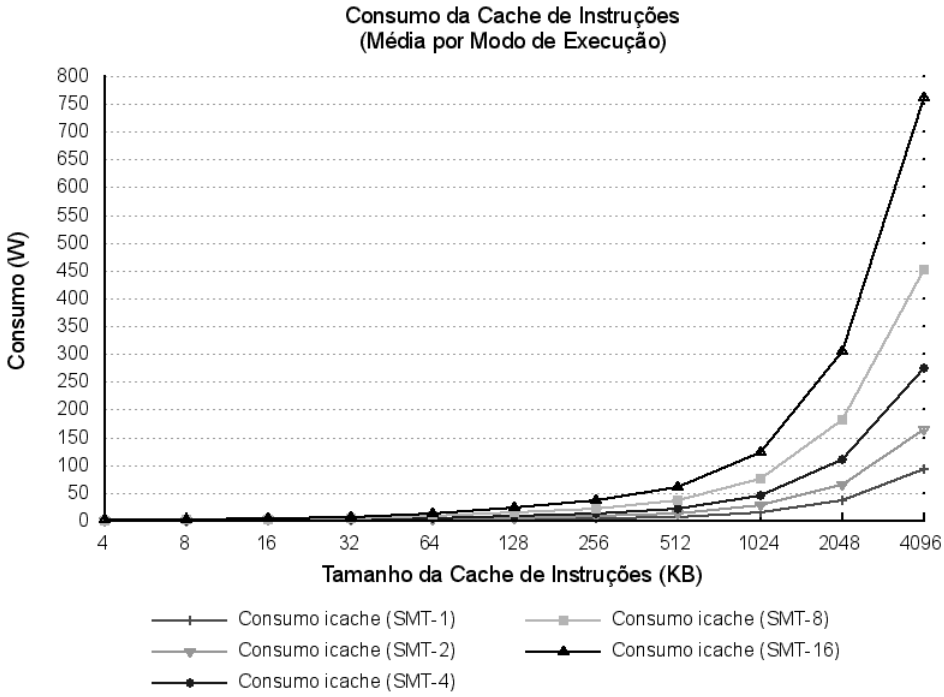

Figura 6.12: Curva do consumo da I-Cache no estilo CC2

O estilo CC3 é de acordo com (BROOKS *et al*, 2000) o que mais se aproxima das situações reais. Por este motivo, apesar de haver uma semelhança no comportamento dos três estilos, embora para valores diferentes de consumo, o estilo CC3 será o foco das discussões. A Figura 6.13 apresenta os resultados obtidos para o consumo da cache de instruções no estilo de *clock* CC3.

Pode-se observar nesta figura que o consumo de potência se eleva menos do que o dobro na medida em que dobra-se o tamanho da *cache*. De fato, olhando os arquivos de *log* das simulações, este aumento é em média 87%, ou seja, na medida em que o tamanho da cache de instruções dobra, o consumo aumenta 87% na média de todas as configurações avaliadas. O maior pico que observou-se é de 149% de aumento.

Sabe-se que existem várias questões associadas ao consumo da *cache* que não dependem somente do número de entradas. A lógica de decodificação de endereçamento tem um crescimento não linear. Por exemplo, decodificar 8 posições requer um único *bit* a mais no endereçamento do que decodificar 4 posições. É interessante notar que com o aumento do número de tarefas, ainda que usando o mesmo tamanho de cache, ocorre também um aumento no consumo de potência bastante significativo.

Olhando os arquivos de *log* destas simulações este aumento atinge de 71%, quando dobramos de 8 para 16 o número de tarefas, a 81%, quando dobramos de 1 para 2 o número de tarefas, na média. A justificativa para este fato é que a técnica de cálculo do consumo de potência CC3, derivado do modelo do Wattch, considera o consumo mediante o acesso às estruturas que estão sendo avaliadas. Assim, este aumento está diretamente relacionado ao número de acessos à *cache* quando aumentamos o número de tarefas compartilhando o sistema de *cache*.

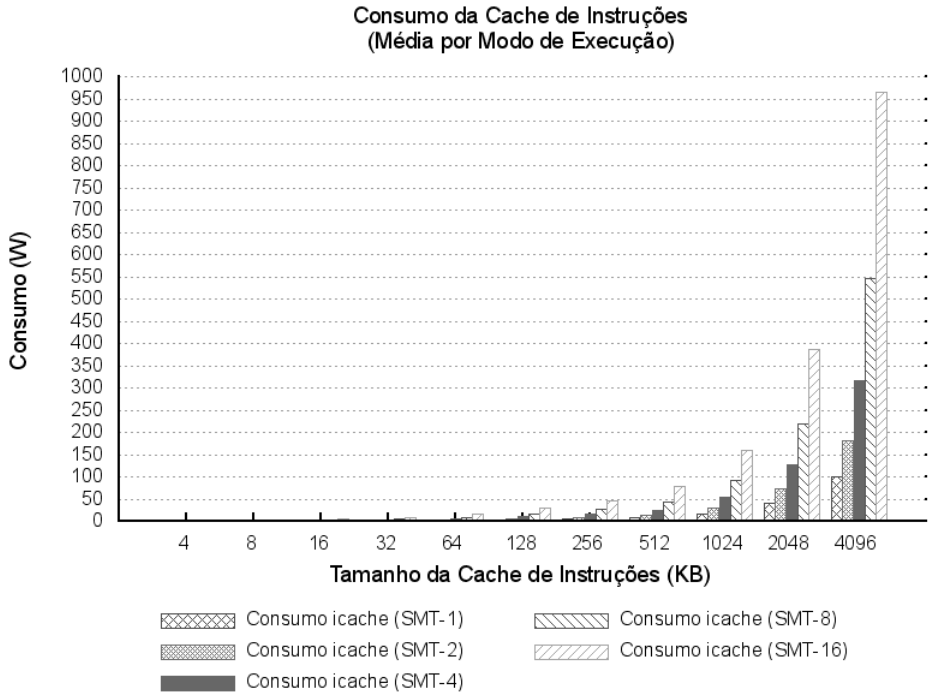

Figura 6.13: Consumo da *I-Cache* no estilo CC3

A Figura 6.14 traz o consumo da cache de instruções para o estilo CC3 representando as curvas de crescimentos dos valores desse consumo para a variação do tamanho da cache.

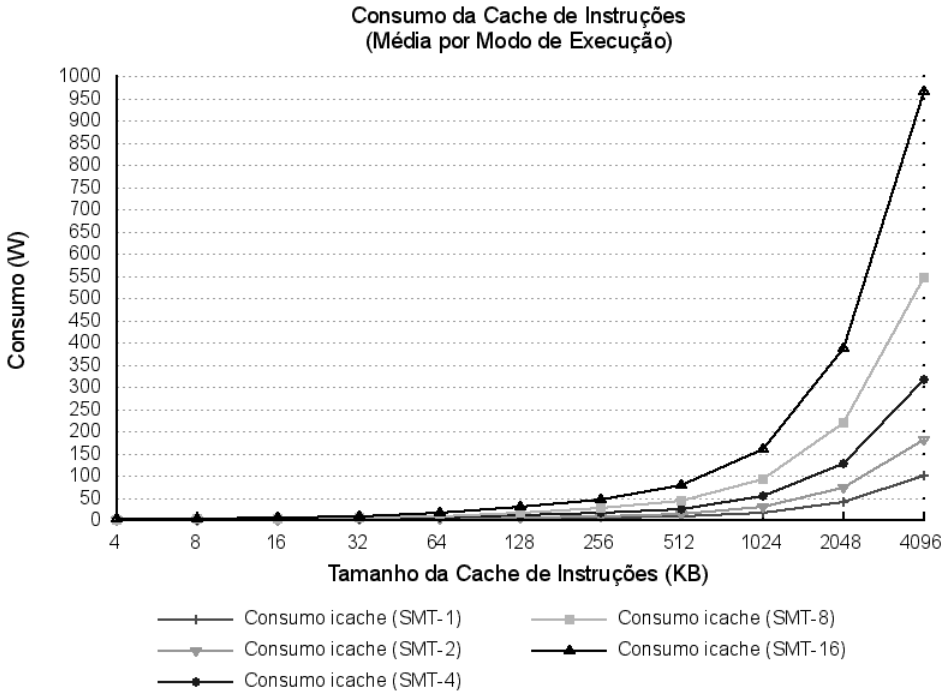

Figura 6.14: Curva do consumo da *I-Cache* no estilo CC3

Para que a apresentação dos consumos de caches com tamanhos abaixo de 64KB fosse adequada, os dados apresentados nas figuras 6.13 e 6.14 foram subdivididos em outros três gráficos apresentados nas figuras 6.15, 6.16 e 6.17, dando destaque à visualização de alguns intervalos.

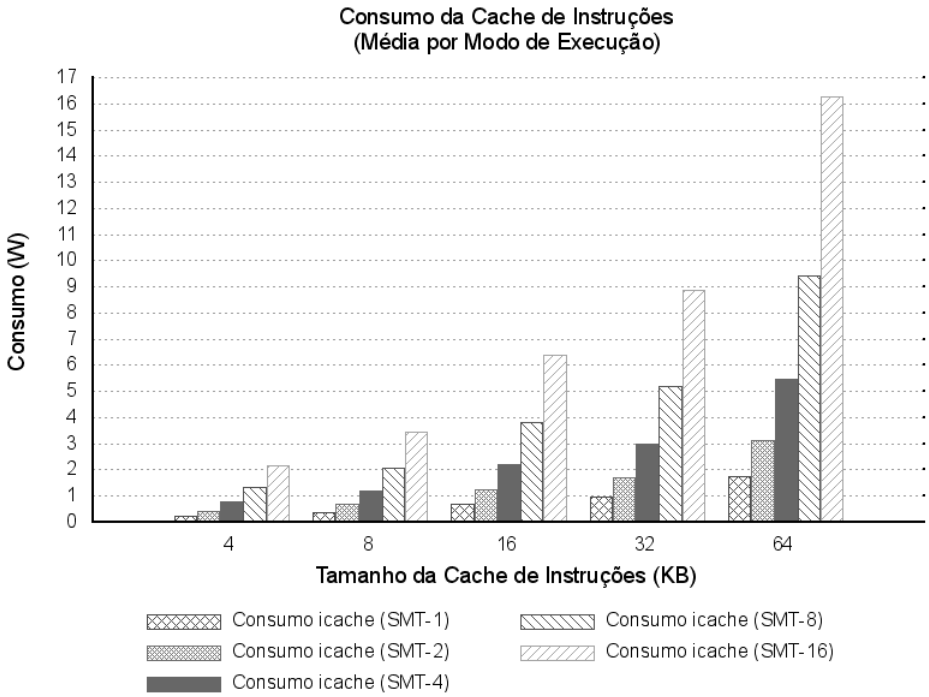

Figura 6.15: Consumo da *I-Cache* no estilo CC3 para os tamanhos de 4KB até 64KB

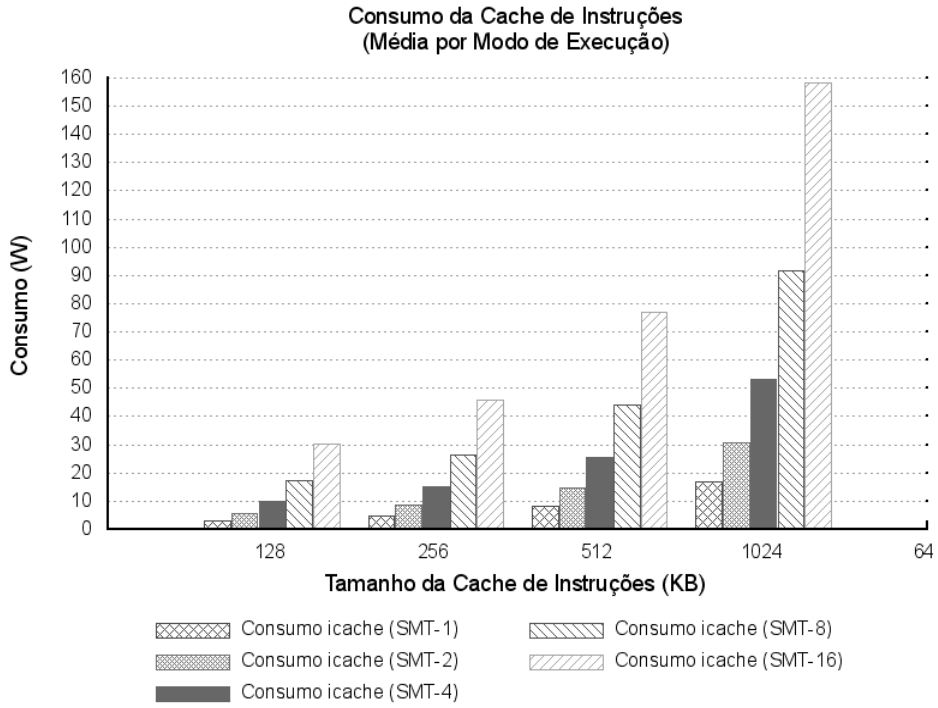

Figura 6.16: Consumo da *I-Cache* no estilo CC3 para os tamanhos de 128KB até 1024KB

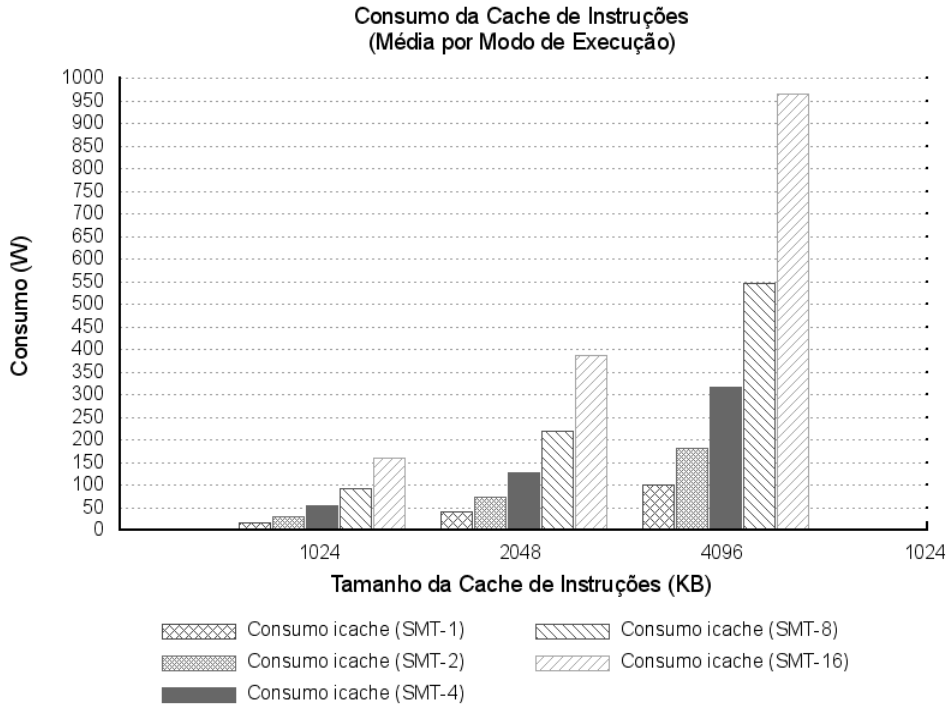

Figura 6.17: Consumo da *I-Cache* no estilo CC3 para os tamanhos de 1024KB até 4096KB

A Figura 6.18 apresenta a variação do IPC (número de instruções executadas por ciclo), medida de desempenho para a execução com os diversos tamanhos de *cache* de instruções no modos de execução SMT.

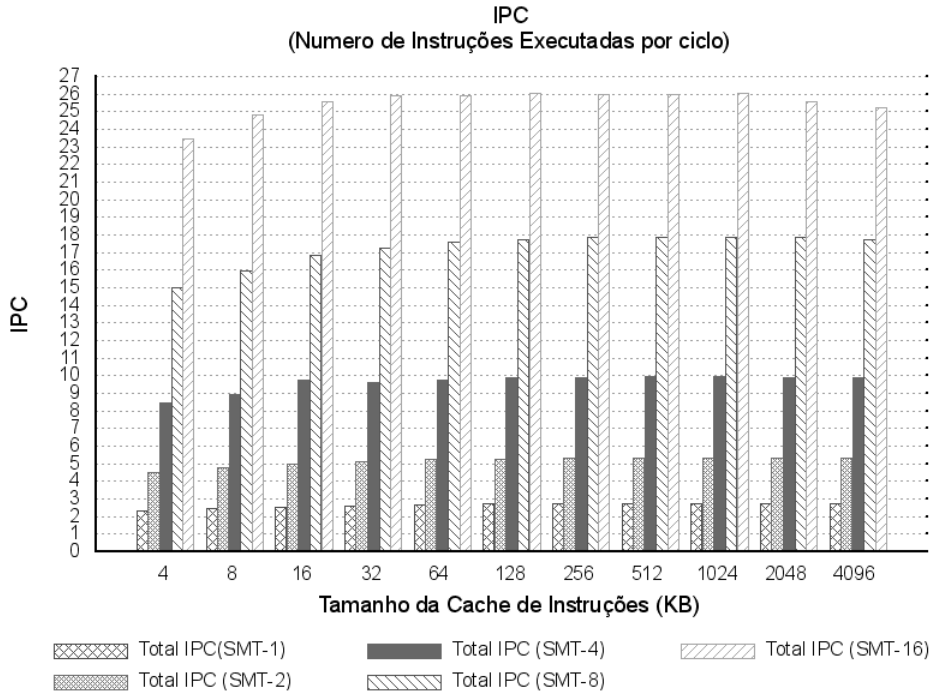

Figura 6.18: Desempenho relacionado a cada modo de execução SMT-X

O gráfico da Figura 6.18 possibilita a visualização do desempenho e ajuda a encontrar um ponto de saturação em seu aumento, pois o desempenho tem pequeno aumento quando dobramos o tamanho da cache, alcançando a média de 6% apenas nas 2 caches menores e nas demais *caches* não sofre nenhum incremento considerável. Agora, quando dobramos o número de tarefas simultâneas, o desempenho aumenta significativamente independente do tamanho da *cache* de instruções nível 1, alcançando a média de 78% no aumento do IPC.

6.2.3 Conclusões do Experimento 2

Nesse experimento pode-se notar que tamanho da *cache* em 16KB tornou-se o ponto de saturação do desempenho em todos os modos de execução, a partir desse tamanho o ganho no desempenho começa a diminuir, chegando ao ponto de começar a sofrer um decréscimo a partir do tamanho de 1024KB. Foram verificadas algumas possibilidades de explicação para isso, como um possível aumento no taxa de erros na *cache* (*miss rate*), mas isso não firmou-se como a causa dessa queda no desempenho, pois os dados foram verificados e à medida que o tamanho da *cache* aumenta, o número de faltas tende a diminuir. Dependendo do tamanho do *benchmark*, o aumento no tamanho da *cache* favorece a disponibilidade das instruções podendo em alguns casos o código necessário à execução estar todo na *cache.*

Outra possibilidade verificada para a queda no desempenho das *caches* com tamanho a partir de 1024KB, foi o fato dos resultados serem formados pela média das execuções com a mesma quantidade de tarefas, diferindo somente na quantidade de módulos do sistema de *cache*, mas

verificando-se os dados para cada configuração de número de módulos, constatou-se que em todas as configurações individualmente, há essa queda no desempenho.

6.3 *Experimento 3:* Avaliação da Organização das Unidades Funcionais e os impactos no consumo

As unidades funcionais e a organização assumida por elas influenciam no desempenho arquitetural por estarem diretamente ligadas às classes de instruções que são executadas. A quantidade dessas unidades funcionais podem exceder a demanda de execução não impactando no desempenho, porém acumulando um consumo desnecessário. Diante disso, esse experimento foi proposto e está relacionado à organização heterogênea das unidades funcionais, sendo baseado na variação da quantidade de unidades funcionais com o intuito de verificar a relação entre a quantidade de unidades funcionais, o desempenho e o consumo de potência alcançado.

6.3.1 Descrição do Experimento

A quantidade de unidades funcionais foi variada de 1 até 10, para cada um dos 4 tipos contabilizados (IALU, IMultDiv, FPALU e FPMultDiv) simultaneamente. As unidades funcionais de acesso à memória não foram contabilizadas e as configurações arquiteturais foram mantidas em uma formatação ampla para que não houvesse interferência no desempenho conforme apresentado na Tabela 6.6.

Tabela 6.6: Definição da arquitetura ampla utilizada no experimento 3

Configuração arquitetural para o experimento 3		
Busca (*Fetch Stage*)	IFQ de 256 entradas Largura de busca: 256 instruções por ciclo	-fetch:width 256 -fetch:ifqsize 256
Decodificação *(Decode Stage)*	Largura de Decodificação: 256 instruções por ciclo	-decode:width 256
Remessa *(Issue)*	Largura de remessa: 256 instruções por ciclo	-issue:inorder FALSE -issue:width 256
Commit	Largura do *commit*: 256 instruções por ciclo	-commit:width 256
Janela de Instruções	256 entradas na RUU 128 entradas na LSQ Organização distribuída	-ruulsq:type distributed -ruu:size 256 -lsq:size 128
Registradores Físicos	32-Int e 32-FP	*hardcoded*
Unidades Funcionais	Organização heterogênea Quantidade X variando de 1 a 10.	-fu:type hetero -res:ialu X -res:imult X -res:fpalu X

Configuração arquitetural para o experimento 3		
		-res:fpmult X -res:memport X
Previsor de Desvios	Previsão em 2 níveis: 1 entrada no primeiro e 4096 entradas no segundo de 2 *bits* de histórico (2 *bits* é o padrão) Registrador de *Shift* : 16 *bits* XOR: 0 BTB: 512 conjuntos, assoc. 4 Tamanho da pilha de endereço de retorno (*Return address stack*): 8	-bpred:type 2lev -bpred:2lev 1 4096 16 0 -bpred:btb 512 4 -bpred:ras 8
Sistema de *Caches*	**Número de módulos:** igual ao número de tarefas. ***I-Cache* de nível 1:** 4096KB, assoc. 1, 64-*byte block size*, LRU *repl. policy.* ***D-Cache* de nível 1:** 64KB, assoc. 4, 64-*byte block size,* LRU *repl. policy.* ***Caches* de instrução e dado nível 2 unificada:** 1024KB, assoc. 8, 128-*byte block size,* LRU *repl. policy.* **TLBS:** 1024 entradas, tamanho da página igual 4096, assoc. 4, LRU *repl. policy.*	-imodules:num <número de Tarefas> -cache:il1 il1:1:4096:64:1:l -cache:dl1 dl1:1:64:64:4:l -cache:il2 dl2 -cache:dl2 ul2:1:1024:128:8:l -tlb:itlb itlb:1024:4096:4:l -tlb:dtlb dtlb:1024:4096:4:l
Memória	*64-bit data*	-mem:width 8
Tecnologia	0,07µm	-technology TECH_070

6.3.2 Resultados

Os resultados obtidos no experimento relacionado às unidades funcionais são apresentados para cada cenário SMT-X. Nos gráficos para esse experimento as colunas indicam o consumo de potência em *Watts* por tipo de unidade funcional e a curva indica o IPC global da arquitetura, ambos valores médios entre todas as possíveis configurações. Observe que o consumo total para dada quantidade de unidades funcionais é obtido pela somatória dos valores das respectivas colunas. Note também que a curva do IPC está na mesma escala do consumo em todos os gráficos, pois os resultados obtidos apresentaram uma proximidade coincidente entre os valores do IPC e do consumo de potência. Porém entre os gráficos as escalas são diferentes, pois seria difícil visualizá-los em uma mesma escala.

Os resultados apresentados são médias das execuções individuais dos *benchmarks* apresentados na Tabela 6.1 que, como mencionado anteriormente, incluem tanto *benchmarks* de aritmética inteira, quanto de ponto flutuante. A Figura 6.19 mostra o consumo para as unidades funcionais no estilo de *clock* CC3 pela variação da quantidade de unidades funcionais no modo de

execução SMT-1, e ainda mostra a curva do desempenho (IPC) alcançado. Na execução SMT-1 o desempenho alcançado com o aumento da quantidade de unidades funcionais iniciou em 0,9289 e atingiu o máximo de 2,7634. Enquanto os consumos máximos para cada tipo de unidade funcional atingiu 2.7540W (IALU), 0.5966W (ImultDiv), 2.5471W (FPALU) e 2.4796W (FPMultDiv).

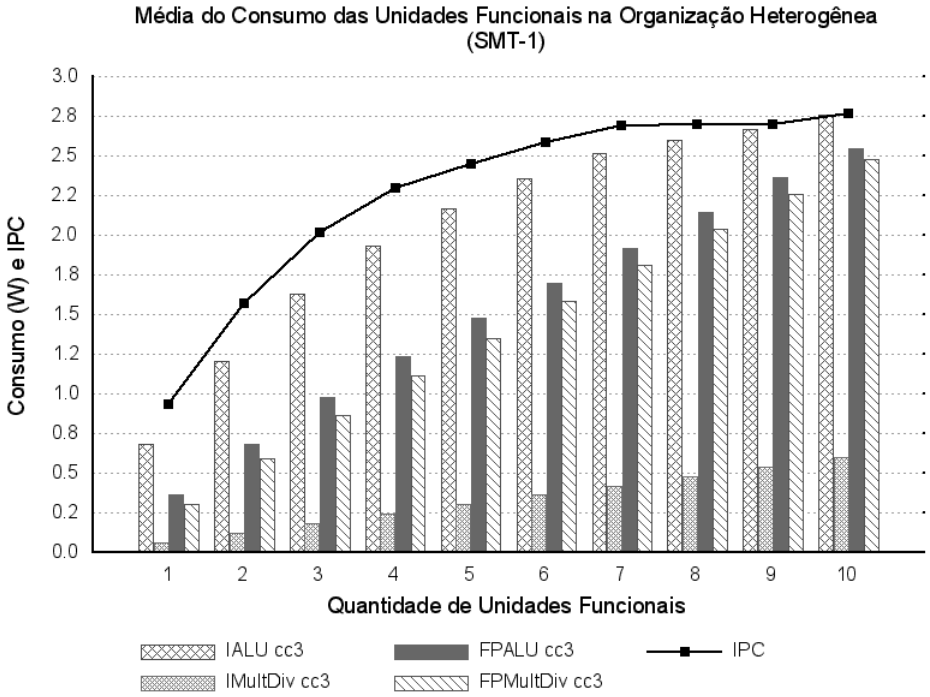

Figura 6.19: Consumo para as unidades funcionais no modo SMT-1

O desempenho foi de 0,9223 com uma unidade funcional de cada, e atingiu 5,1216 com a quantidade de unidades funcionais igual a 10. Em todas as configurações para SMT-2, o desempenho alcançado não chega a ser o dobro do obtido com SMT-1, porém o consumo para algumas unidades funcionais com quantidade acima de 5 chega a ser mais que o dobro. O consumo nesse caso é influenciado pela quantidade de acessos a essas unidades que foi aumentado com a execução de mais tarefas e pela própria quantidade de unidades que participam da composição do consumo.

A Figura 6.20 apresenta os consumos das unidades funcionais e a curva de desempenho para cada configuração de quantidade no modo de execução SMT-2.

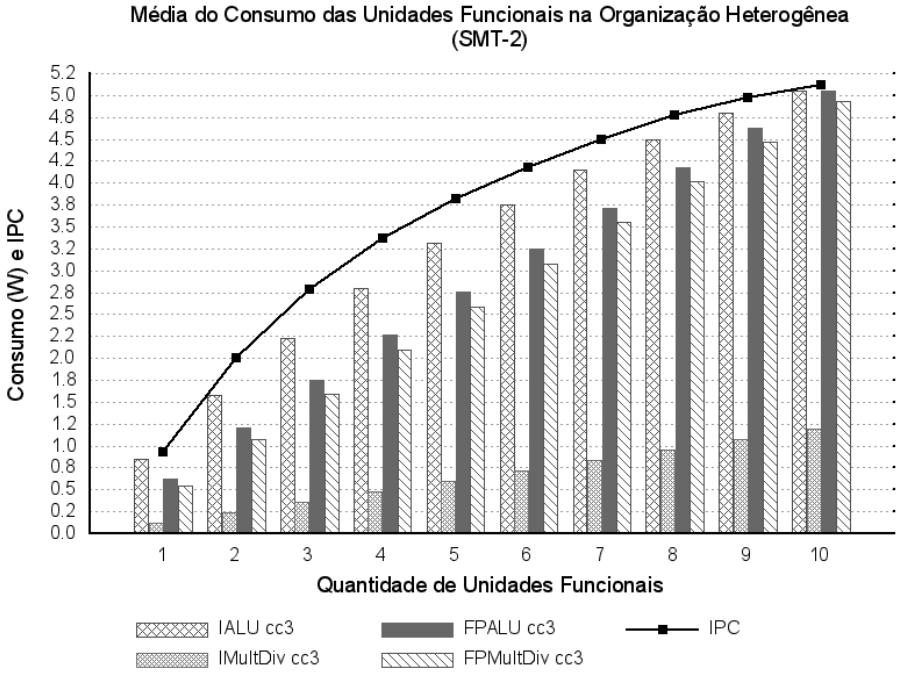

Figura 6.20: Consumo para as unidades funcionais no modo SMT-2

A Figura 6.21 apresenta os consumos das unidades funcionais e a curva de desempenho para cada quantidade de unidades funcionais no modo de execução SMT-4.

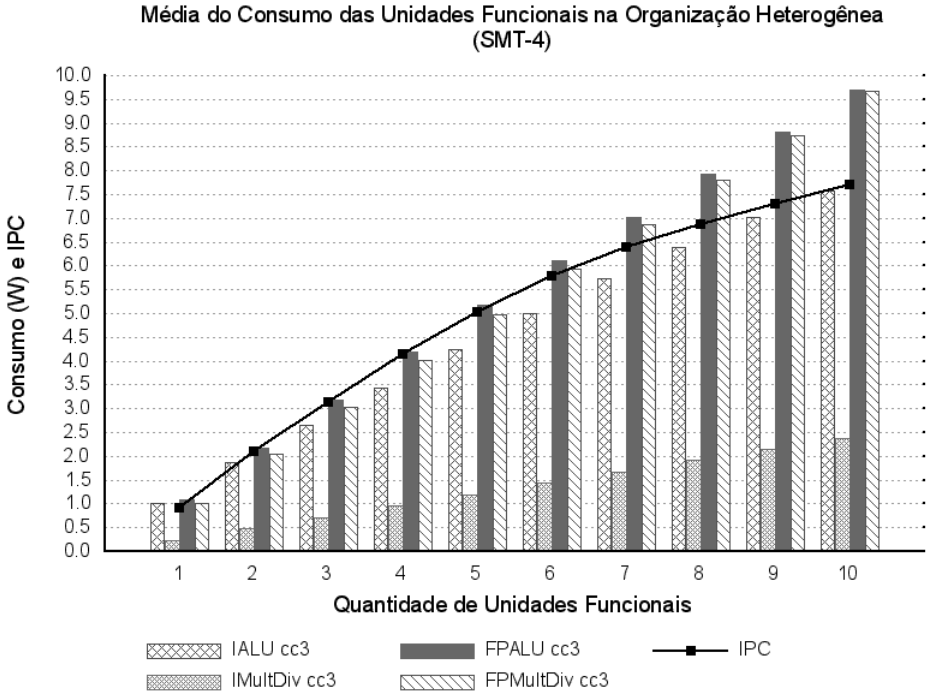

Figura 6.21: Consumo para as unidades funcionais no modo SMT-4

A Figura 6.22 apresenta os consumos das unidades funcionais e a curva de desempenho para cada configuração de quantidade no modo de execução SMT-8, no qual o desempenho atingiu o valor de 10,2591 com 10 unidades funcionais de cada tipo e o consumo seguiu um comportamento linear, sendo seu crescimento proporcional à quantidade de unidades.

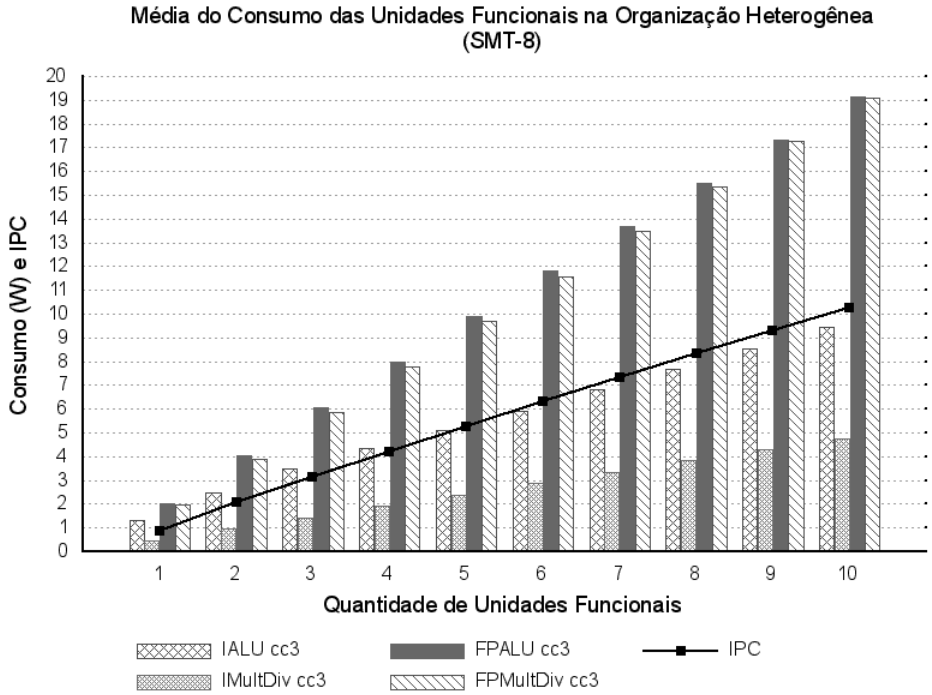

Figura 6.22: Consumo para as unidades funcionais no modo SMT-8

A Figura 6.23 apresenta os consumos das unidades funcionais e a curva de desempenho para cada configuração de quantidade no modo de execução SMT-16.

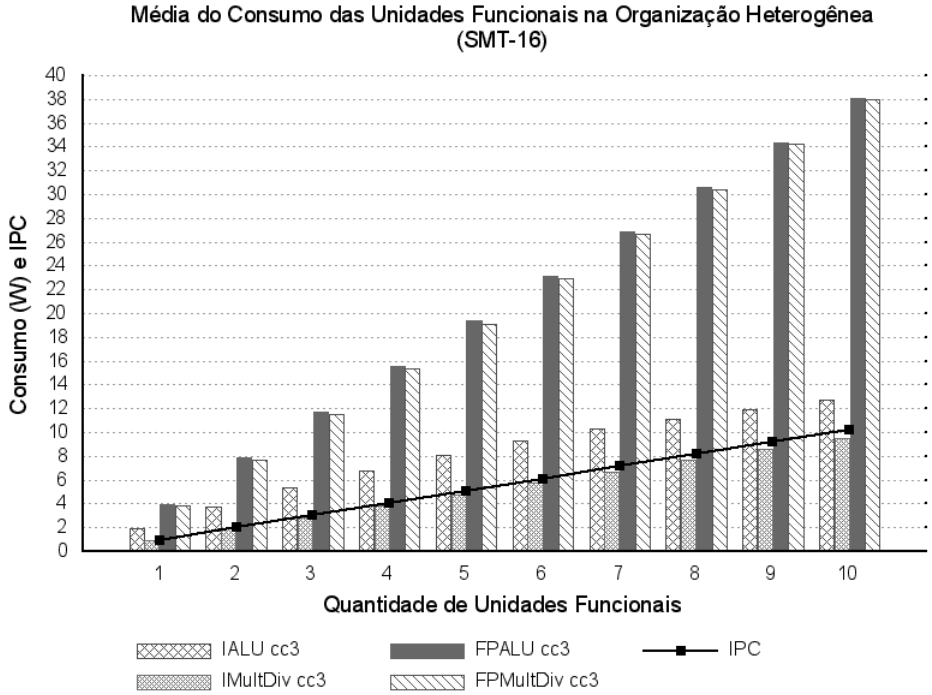

Figura 6.23: Consumo para as unidades funcionais no modo SMT-16

O desempenho nesse modo de execução atingiu 10,2309, mostrando que o desempenho se manteve praticamente o mesmo que o da execução com oito tarefas. O consumo neste caso foi um pouco maior atingindo para cada tipo de unidade funcional os seguintes valores 12,7046W (IALU),

9,5419W (ImultiDiv), 38,0481W (FPALU) e 37,9761W (FPMultDiv).

6.3.3 Conclusões do Experimento 3

De uma forma geral, percebe-se em todas as situações que o consumo de potência possui um crescimento razoavelmente linear para os 4 tipos de unidades funcionais, com alguma distorção no caso da IALU na SMT-1. Percebe-se também que o desempenho obtido cresce mais próximo ao consumo de potência quanto menor for o número de tarefas. Nas arquiteturas com maior número de tarefas, por exemplo SMT-16, o desempenho possui uma taxa de crescimento menor do que a taxa de crescimento do consumo de potência. Obviamente, o aumento do *hardware* certamente aumenta o consumo de potência, mas não necessariamente aumenta o desempenho, haja vista a possibilidade de saturação dos demais recursos.

Este crescimento indesejável do consumo de potência nas arquiteturas com muitas tarefas pode ser explicado, em parte, pela forma como o experimento foi realizado, que fixou para todos os tipos de unidades funcionais a mesma quantidade. Entretanto, existem tipos de unidades funcionais que não precisam ter uma quantidade tão grande, até mesmo pelos tipos de instruções que executam e pela quantidade de instruções desses tipos presentes nos códigos dos *benchmarks*. Os valores do consumo poderiam ser melhor ajustados com um número menor de unidades funcionais para os tipos de instruções com menor ocorrência, já que não influenciam tanto para a melhoria do desempenho, acarretando somente um aumento no consumo.

Por terem a mesma quantidade em cada configuração, os consumos apresentados para as unidades funcionais dependem do número de acessos que cada unidade funcional recebeu, esses acessos são feitos conforme as classes das instruções que estão sendo executadas. Por exemplo, na Figura 6.19, que apresenta a média do consumo e do desempenho para todas as execução realizadas como uma única tarefa, é perceptível que as instruções que ocorrem com maior freqüência pertencem às classes que são tratadas pela Unidade Lógica e Aritmética de números inteiros, pois esta apresenta o maior consumo em cada umas das configurações. O consumo da outras unidades funcionais se mantem proporcional à medida que o número de unidades aumenta.

6.4 *Experimento 4:* Avaliação da Janela de instruções e os impactos no consumo

Este experimento avalia o tamanho da Janela de Instruções (RUU/LSQ) e os impactos produzidos no desempenho e no consumo de potência.

6.4.1 Descrição do Experimento

O número total de entradas na RUU foi variado de 16 a 256, sendo que o tamanho da LSQ foi

sempre a metade do tamanho assumido pela RUU. Foram exploradas a organização distribuída (*distributed*) e a compartilhada (*shared*) para a RUU/LSQ. Da mesma forma que nos experimentos anteriores, todos os modos de execução SMT-X foram executados e as arquiteturas foram configuradas com recursos amplos para que o desempenho dependesse somente das estruturas analisadas. A configuração definida para esse experimento é apresentada na Tabela 6.7.

Tabela 6.7: Definição da arquitetura ampla utilizada no experimento 4

Configuração arquitetural para o experimento 4		
Busca (*Fetch Stage*)	IFQ de 256 entradas Largura de busca: 256 instruções por ciclo	-fetch:width 256 -fetch:ifqsize 256
Decodificação *(Decode Stage)*	Largura de Decodificação: 256 instruções por ciclo	-decode:width 256
Remessa *(Issue)*	Largura de remessa: 256 instruções por ciclo	-issue:inorder FALSE -issue:width 256
Commit	Largura do *commit*: 256 instruções por ciclo	-commit:width 256
Janela de Instruções	Organização compartilhada ou distribuída. Número de entradas na RUU variando de 16 a 256. Número de entradas na LSQ igual à metade do número de entradas da RUU. Organização distribuída	-ruulsq:type <distributed ou shared> -ruu:size X -lsq:size X/2
Registradores Físicos	32-Int e 32-FP	*hardcoded*
Unidades Funcionais	Organização heterogênea ALU Inteiro: 32 ALU FP: 32 Mult. Int.: 32 Mult. FP: 32 Memory Port: 32	-fu:type hetero -res:ialu 32 -res:imult 32 -res:fpalu 32 -res:fpmult 32 -res:memport 32
Previsor de Desvios	Previsão em 2 níveis: 1 entrada no primeiro e 4096 entradas no segundo de 2 *bits* de histórico (2 *bits* é o padrão) Registrador de *Shift* : 16 *bits* XOR: 0 BTB: 512 conjuntos, assoc. 4 Tamanho da pilha de endereço de retorno (*Return address stack*): 8	-bpred:type 2lev -bpred:2lev 1 4096 16 0 -bpred:btb 512 4 -bpred:ras 8
Sistema de *Caches*	**Número de módulos:** igual ao número de tarefas. ***I-Cache* de nível 1:** 4096KB, assoc. 1, 64-*byte block size*, LRU *repl. policy.*	-imodules:num <número de Tarefas> -cache:il1 il1:1:4096:64:1:l -cache:dl1 dl1:1:64:64:4:l

Configuração arquitetural para o experimento 4		
	***D-Cache* de nível 1:** 64KB, assoc. 4, 64-*byte block size,* LRU *repl. policy.* ***Caches* de instrução e dado nível 2 unificada:** 1024KB, assoc. 8, 128-*byte block size,* LRU *repl. policy.* **TLBS:** 1024 entradas, tamanho da página igual 4096, assoc. 4, LRU *repl. policy.*	-cache:il2 dl2 -cache:dl2 ul2:1:1024:128:8:l -tlb:itlb itlb:1024:4096:4:l -tlb:dtlb dtlb:1024:4096:4:l
Memória	*64-bit data*	-mem:width 8
Tecnologia	0,07µm	-technology TECH_070

6.4.2 Resultados

Os resultados obtidos para o consumo das estruturas relacionadas à janela de instruções são apresentados graficamente, também em termos de consumo médio de potência dado em *Watts*. As médias foram calculadas sobre todos os modos de execução SMT-X e para as organizações distribuída e compartilhada, possibilitando a comparação entre o consumo e desempenho da mesma quantidade de entradas nessas estruturas organizadas de forma diferente.

Para a análise dos dados apresentados nos gráficos que seguem é importante ressaltar a política adotada na simulação desse experimento para os tamanhos da RUU e da LSQ. A Tabela 6.8 mostra a configuração da organização compartilhada, na qual para todos os modos de execução, a RUU e a LSQ foram configuradas de uma única forma.

Tabela 6.8: Tamanhos da RUU/LSQ na organização compartilhada

SMT-X	
RUU	**LSQ**
16	8
32	16
64	32
128	64
256	128

Na organização distribuída, a distribuição do número de entradas proposto pelo experimento ocorre tal como é apresentada pela Tabela 6.9, na qual pode-se notar que a quantidade total de entradas para a RUU de uma forma global é distribuída pelo número de tarefas, sendo que cada tarefa terá em seu contexto uma RUU de tamanho proporcional ao total, já que trata-se da organização distribuída. A LSQ continua sendo tratada da mesma maneira, tendo a metade do tamanho da RUU.

Esse tratamento foi dado no intuito de preservar a equivalência entre os ambientes de execução nas duas organizações, para que pudessem ser comparadas corretamente. Por exemplo, a SMT-16 para 64 entradas, deve comparar sua a configuração (RUU, LSQ) = (4, 2) com as configurações equivalentes nos modos SMT-8 (RUU, LSQ) = (8, 4), SMT-4 (RUU,LSQ) = (16, 8), SMT-2 (RUU, LSQ) = (32,16) e SMT-1 (RUU, LSQ) = (64, 32).

Tabela 6.9: Tamanhos da RUU/LSQ na organização distribuída

Número de Entradas	SMT-1		SMT-2		SMT-4		SMT-8		SMT-16	
	RUU	LSQ	RUU	LSQ	RUU	LSQ	RUU	LSQ	RUU	LSQ
16	16	8	8	4	4	2	-	-	-	-
32	32	16	16	8	8	4	4	2	-	-
64	64	32	32	16	16	8	8	4	4	2
128	128	64	64	32	32	16	16	8	8	4
256	256	128	128	64	64	32	32	16	16	8

As figuras 6.24 e 6.25 possibilitam um comparativo entre as médias de consumo para o modo de execução SMT-1, tem-se o resultado do consumo para a organização distribuída na Figura 6.24 e para a compartilhada na Figura 6.25, para os diversos tamanhos da RUU.

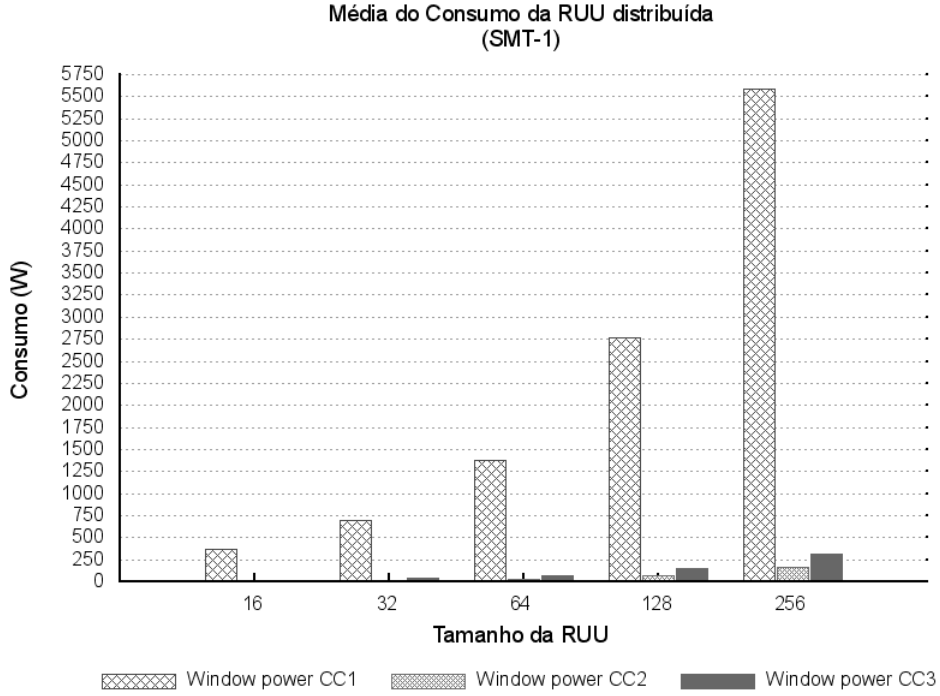

Figura 6.24: Consumo da RUU distribuída no modo SMT-1

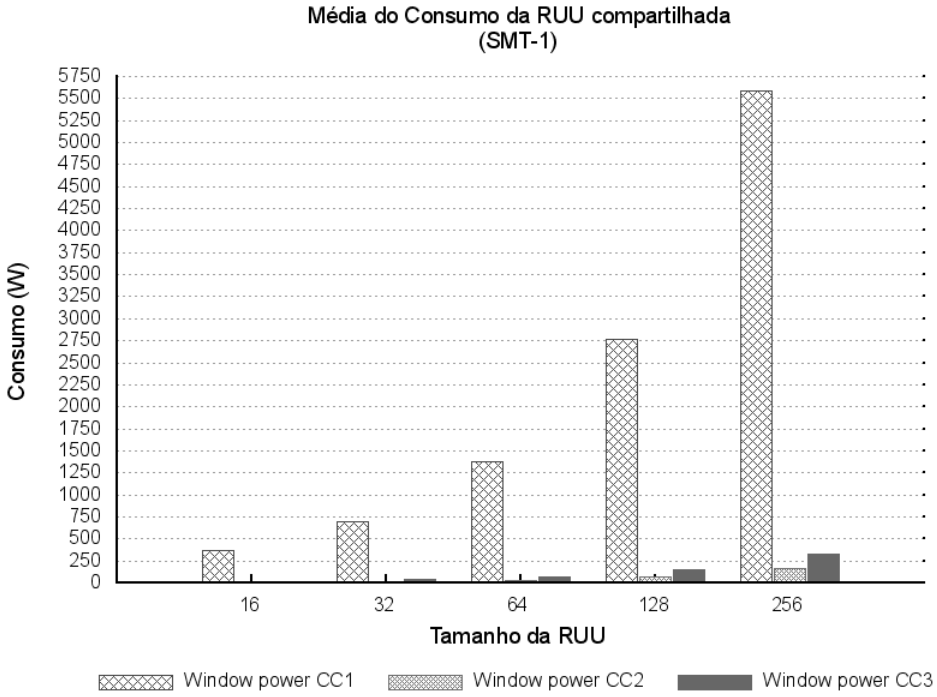

Figura 6.25: Consumo da RUU compartilhada no modo SMT-1

Tanto na Figura 6.24 quanto na Figura 6.25, é perceptível que à medida que o tamanho da RUU é dobrado o consumo segue a taxa de crescimento de 97% para os valores no estilo CC1, não sendo uma simples duplicação dos valores, o que intuitivamente poderia ser esperado. Para o estilo CC2 a taxa de crescimento nas duas organizações fica em torno dos 140%, já os valores no estilo CC3, voltam à casa dos 105%, sendo um aumento no consumo de mais que o dobro para os dois últimos estilos.

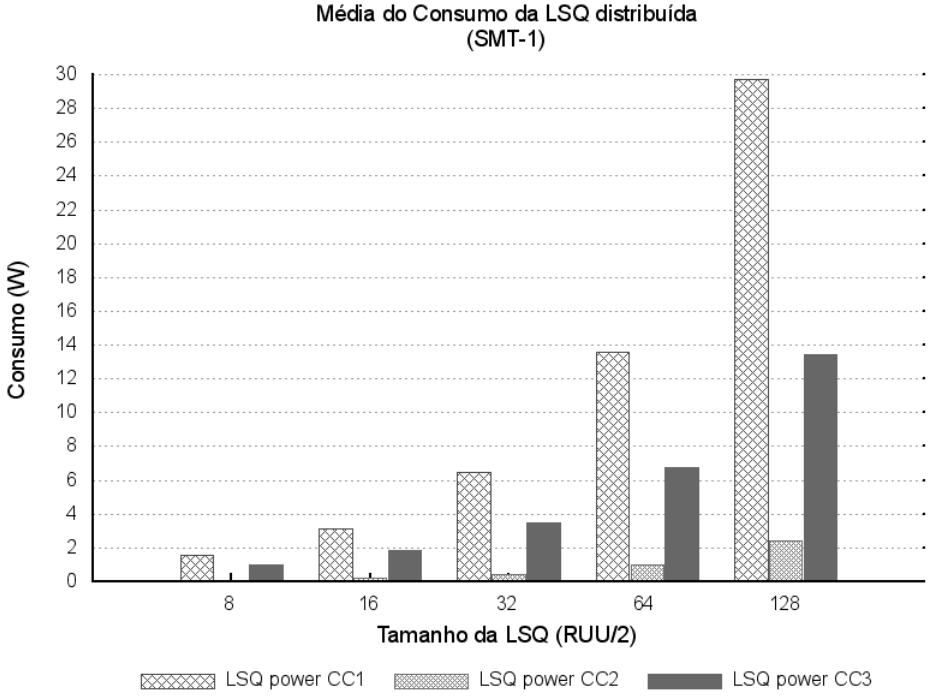

Figura 6.26: Consumo da LSQ distribuída no modo SMT-1

No modo SMT-1, as taxas de crescimento individual de cada organização para cada estilo de *clock* são bem próximas quando não idênticas, não diferindo também entre as organizações distribuída e compartilhada, pois na execução de uma tarefa com a organização distribuída o número de entradas é dividido pelo número de tarefas, o que resulta na mesma quantidade com a estrutura compartilhada. Desta forma tanto faz se a RUU/LSQ for distribuída ou compartilhada no modo SMT-1, pois os valores serão os mesmos.

O mesmo comportamento se repete para a LSQ. O consumo de potência para esta estrutura no modo SMT-1, na organização distribuída é apresentado na Figura 6.26 e para a compartilhada na Figura 6.27. A taxa de crescimento dos consumos da LSQ para o estilo CC1 fica, na média, por volta de 110%. Para CC2 este valor é de 137% e para o CC3 a taxa de crescimento atinge cerca de 91%. A justificativa para a RUU serve também para a LSQ, não havendo ganhos entre a utilização de uma em relação à outra organização.

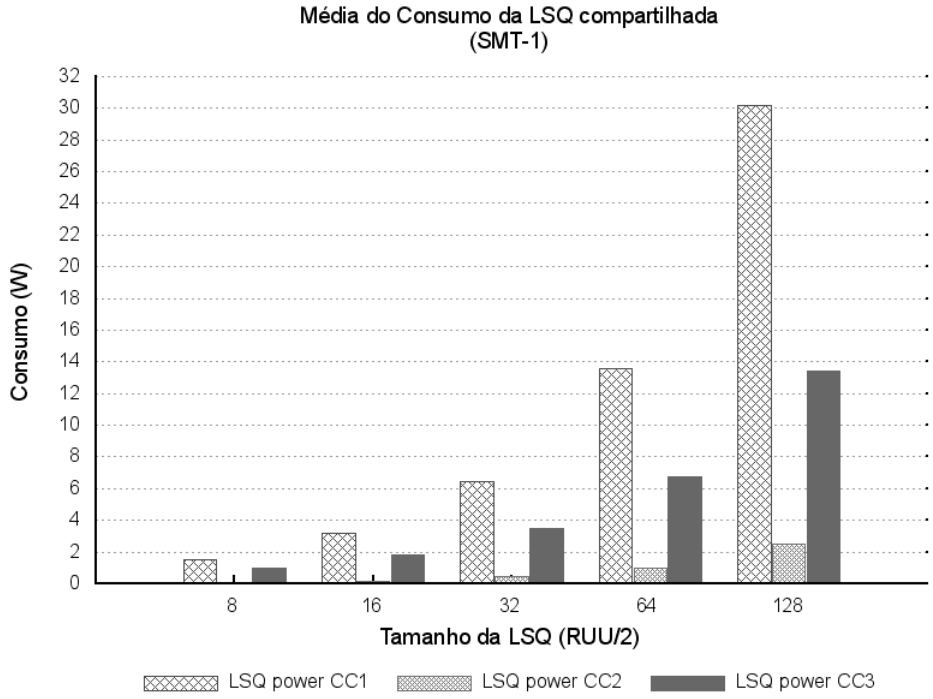

Figura 6.27: Consumo da LSQ compartilhada no modo SMT-1

Observando-se as figuras 6.24 e 6.25 é claramente perceptível a diferença entre os consumos dos estilos de *clock* (CC1, CC2 e CC3). O CC1 apresenta valores maiores que os outros dois estilos devido ao fato de acumular o consumo total da estrutura mediante a cada acesso. Para uma visualização dos valores do consumo nos estilos CC2 e CC3 isolada do CC1 detalhou-se esses consumos nas figuras 6.28 e 6.29.

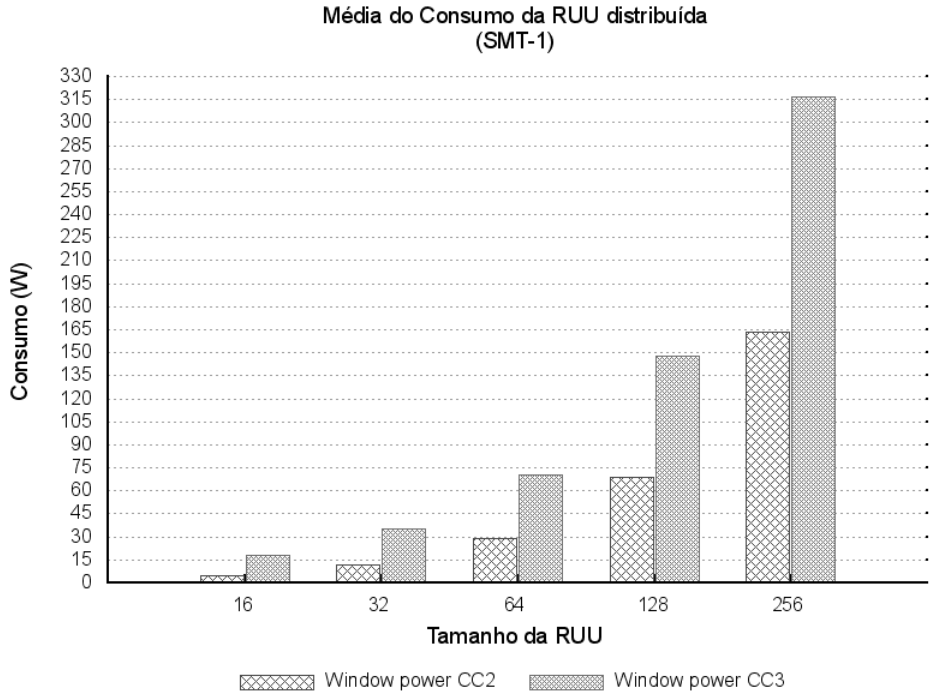

Figura 6.28: Consumo da RUU distribuída para SMT-1 nos estilos CC2 e CC3

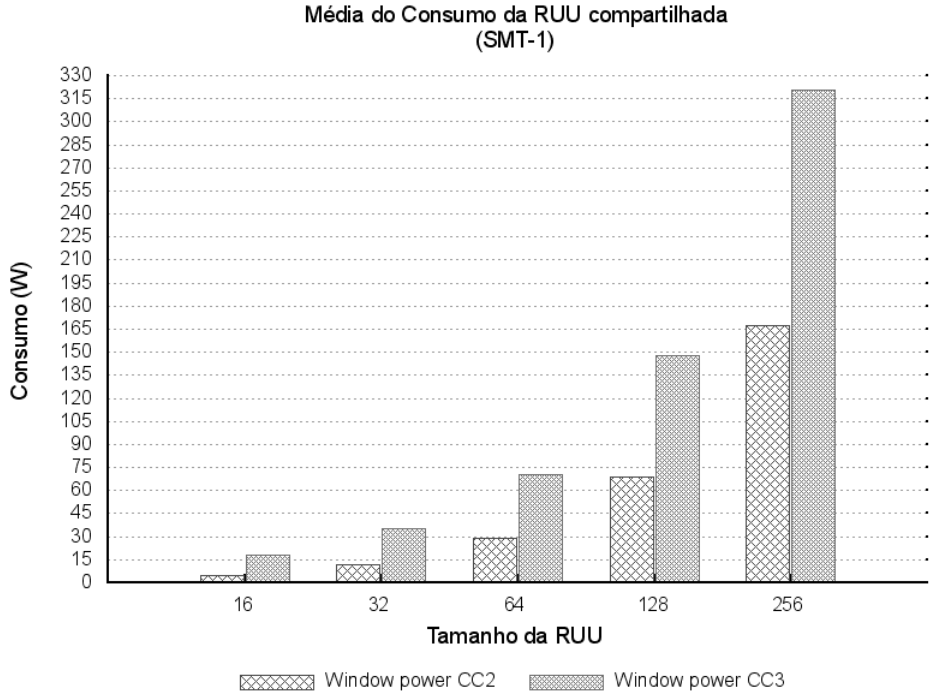

Figura 6.29: Consumo da RUU compartilhada para SMT-1 nos estilos CC2 e CC3

Para os consumos no modo de execução SMT-2 mostrados pelas figuras 6.30 e 6.31, tem-se que à medida que o tamanho da RUU é duplicado, o crescimento individual dos valores em cada estilo de *clock* seguem as taxas, 91,17% e 98,84% para o CC1 nas organizações distribuída e compartilhada, respectivamente.

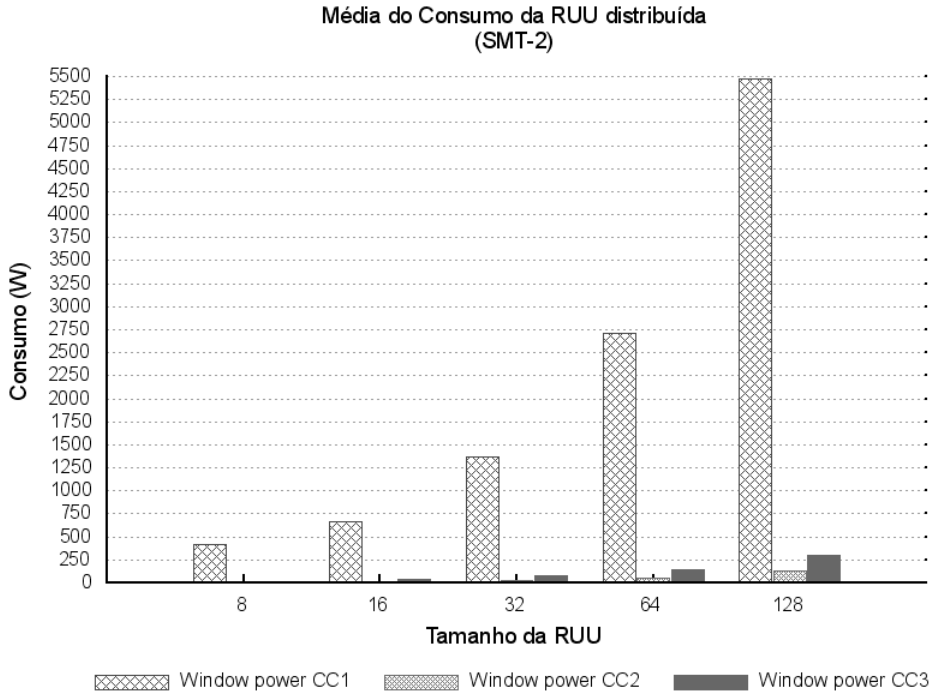

Figura 6.30: Consumo da RUU distribuída no modo SMT-2

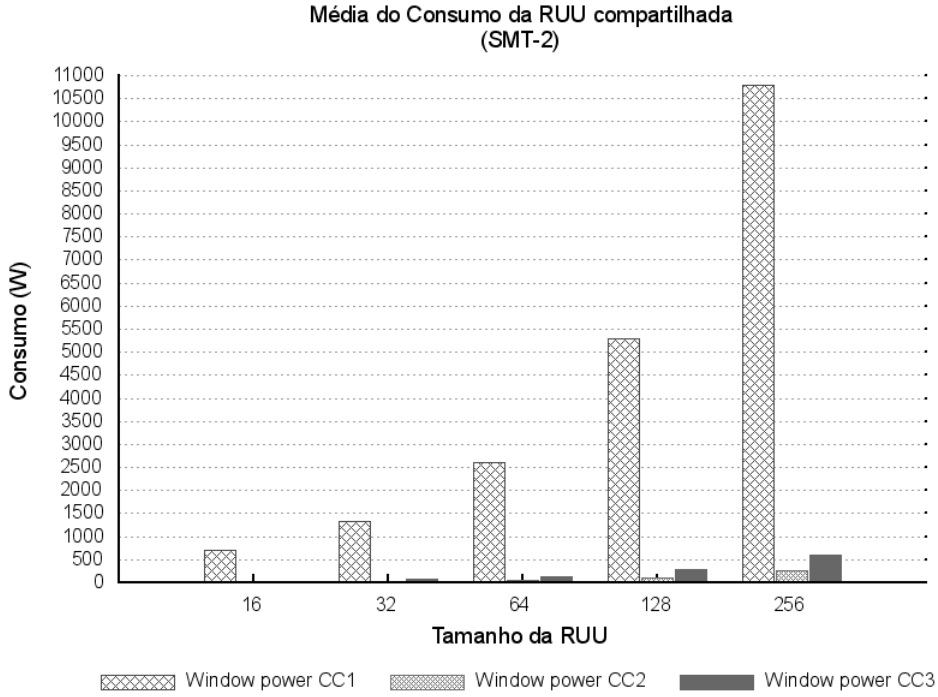

Figura 6.31: Consumo da RUU compartilhada no modo SMT-2

Para o CC2, as taxas de crescimento individual para cada organização são de 142,31% (distribuída) e 152,17% (compartilhada) em relação ao crescimento do tamanho da RUU. No estilo CC3 o crescimento individual seguiu as seguintes taxas 98,88% e 101,69% para todas as configurações. Como pode-se notar pela diferença das taxas de crescimento dos valores de consumo, organização compartilhada consome mais que a distribuída usando as configurações

equivalentes, conforme os valores apresentados nas figuras 6.30 e 6.31.

A Figura 6.32 mostra o consumo para a LSQ relacionada à RUU apresentada na Figura 6.30 e a Figura 6.33 para os valores da RUU apresentada na Figura 6.31. A LSQ segue o mesmo padrão de comportamento da RUU, porém apresenta taxas de crescimento de consumo menores, guardadas as devidas proporções. Com a duplicação do tamanho da LSQ tem-se para o estilo CC1 a taxa em torno de 104,62% para a distribuída e, 115,6% para a compartilhada, isto é, o consumo é mais que o dobro.

No estilo CC2 as taxas de crescimento individual por organização são de 135,91% para distribuída e de 148,87% para a compartilhada, sendo a taxa de crescimento na média 142,39% para as duas organizações. Já para o estilo CC3, as taxas ficam em torno de 82,92% e de 90,22% para ambas as organizações. Neste caso, o consumo maior continua sendo para CC1, porém a maior taxa de crescimento pertence ao estilo CC2 e o consumo de CC3 é o único que não dobrou com a duplicação do tamanho da LSQ ficando entre 80% e 90%.

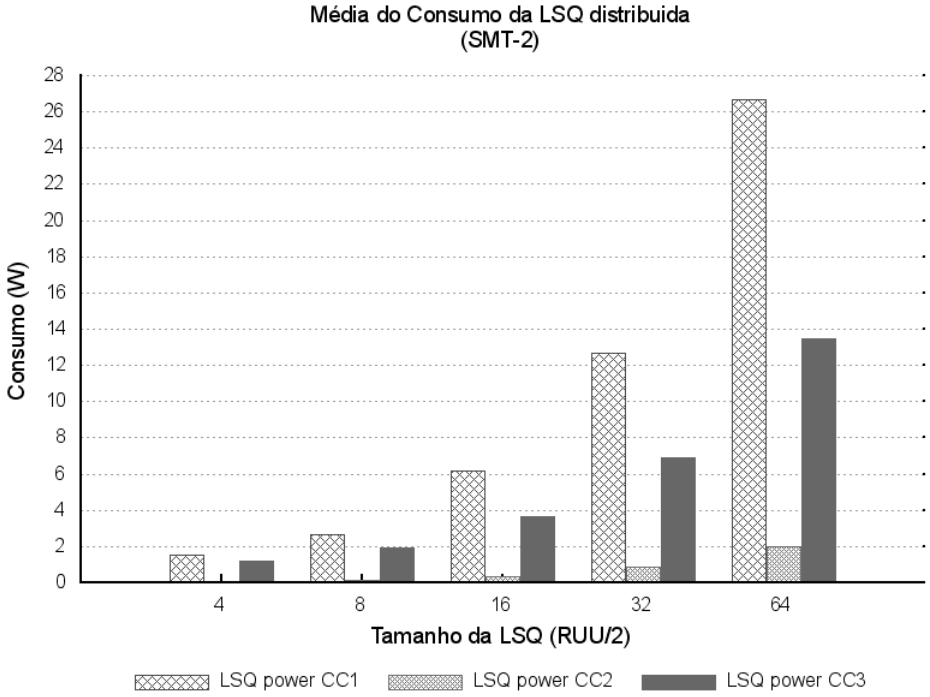

Figura 6.32: Consumo da LSQ distribuída no modo SMT-2

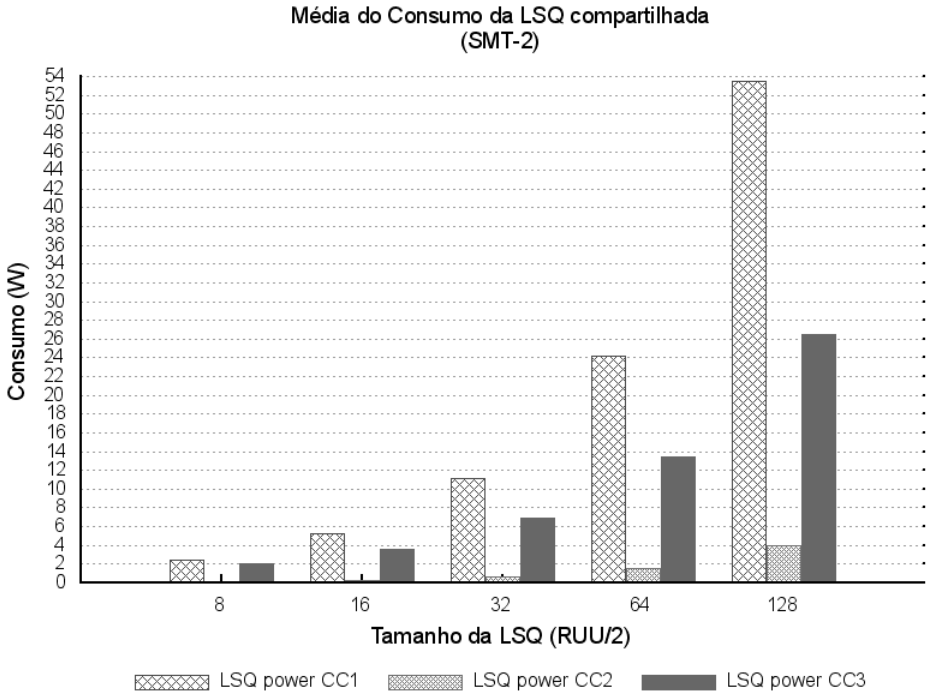

Figura 6.33: Consumo da LSQ compartilhada no modo SMT-2

As figuras 6.34 e 6.35 apresentam os consumos para a RUU, nos estilos CC2 e CC3 isolados, que se apresentam proporcionalmente mais ajustados.

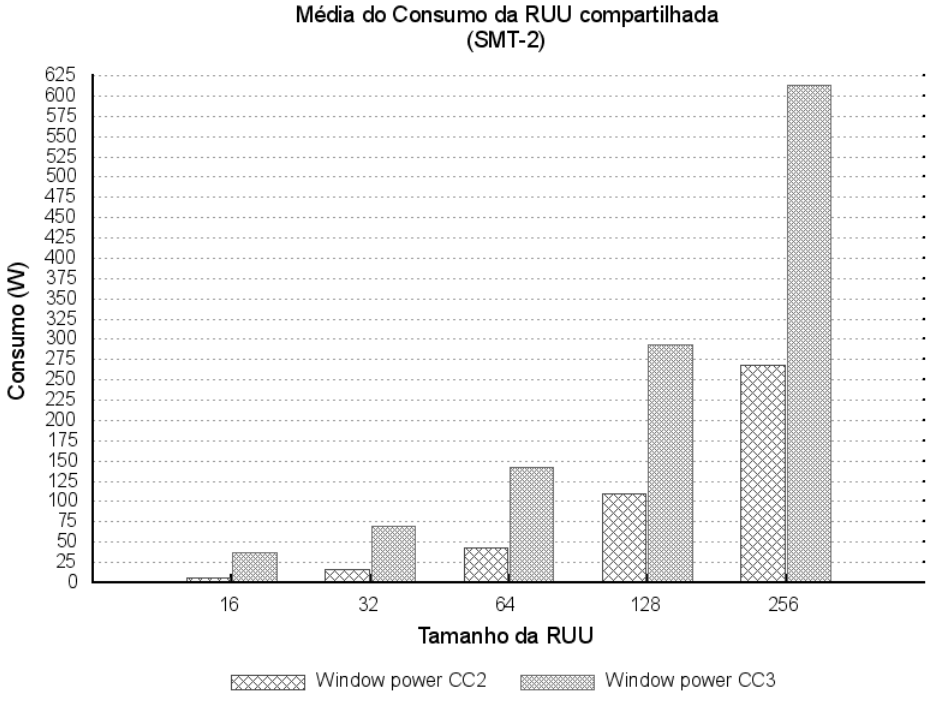

Figura 6.35: Consumo da RUU compartilhada para SMT-2 nos estilos CC2 e CC3

Nas figuras 6.36 e 6.37 são apresentados os valores da média de consumo para a RUU no modo de execução SMT-4. As taxas de crescimento individual para os consumos são de 75,11% e 96% para o estilo CC1, 134,73% e 162,53% para o CC2 e para o CC3 são de 106,2% e 105,51%

para a organização distribuída e para a compartilhada, respectivamente. Embora a taxa de crescimento para a distribuída seja superior no estilo CC3, o consumo da compartilhada continua a ser maior, como por exemplo para o número de entradas igual a 16, tem-se o consumo de 76.0833W para a compartilhada e de 16.5287W para a distribuída.

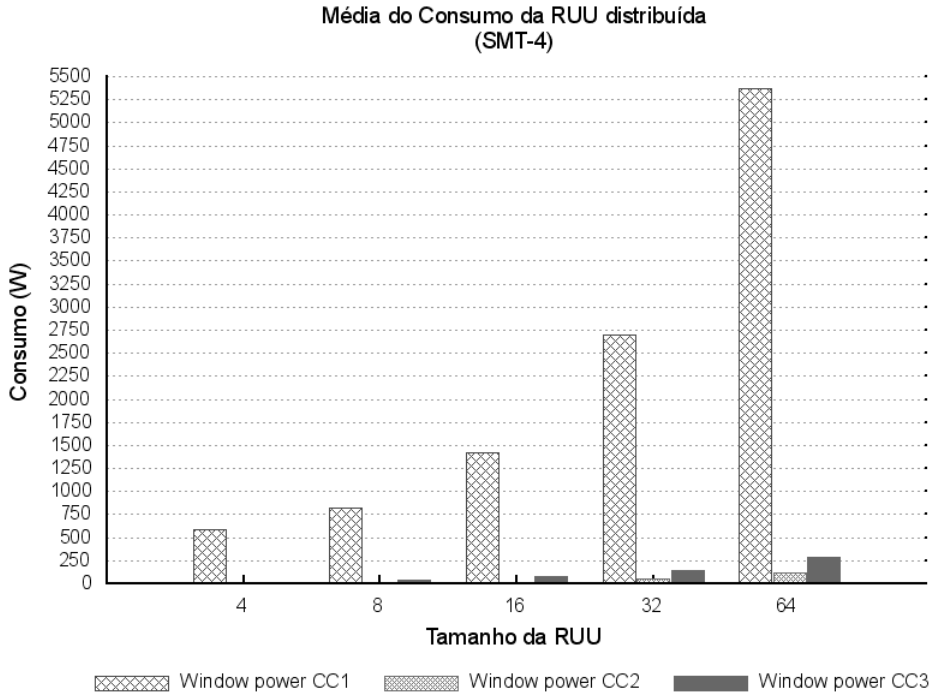

Figura 6.36: Consumo da RUU distribuída no modo SMT-4

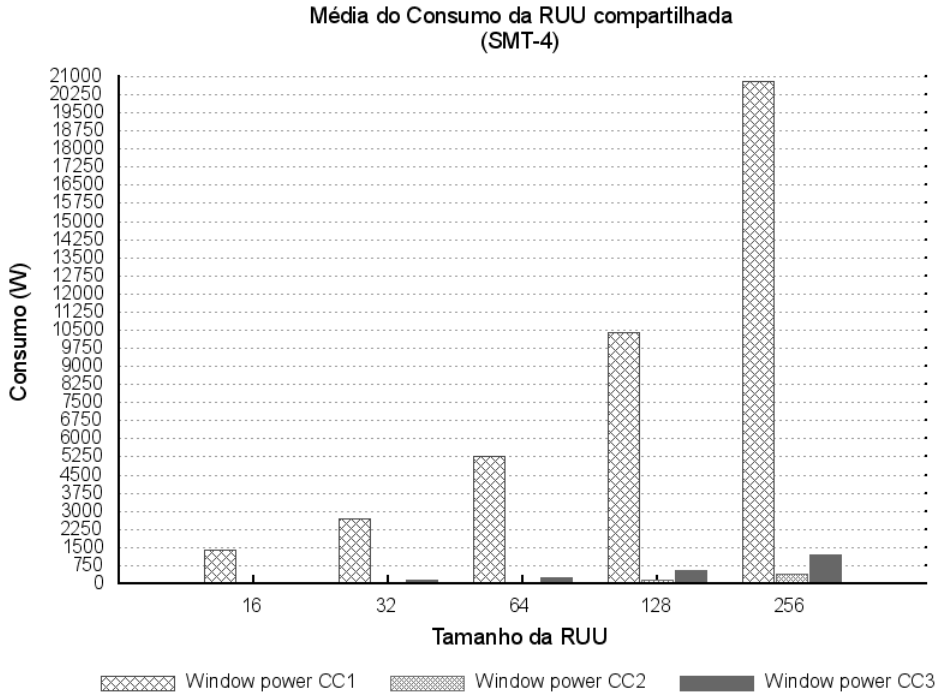

Figura 6.37: Consumo da RUU compartilhada no modo SMT-4

Para os consumos da LSQ, as figuras 6.38 e 6.39 podem ser utilizadas como comparativo dos valores do consumo de cada tamanho de RUU para os três estilos de *clock*. As taxas de crescimento individual do consumo para cada estilo são de 97,24% para a distribuída e 122,18% para a compartilhada no CC1, de 131,7% e 157,48% para CC2 e de 72,72% e 89,26% para CC3 para as configurações distribuída e compartilhada, respectivamente.

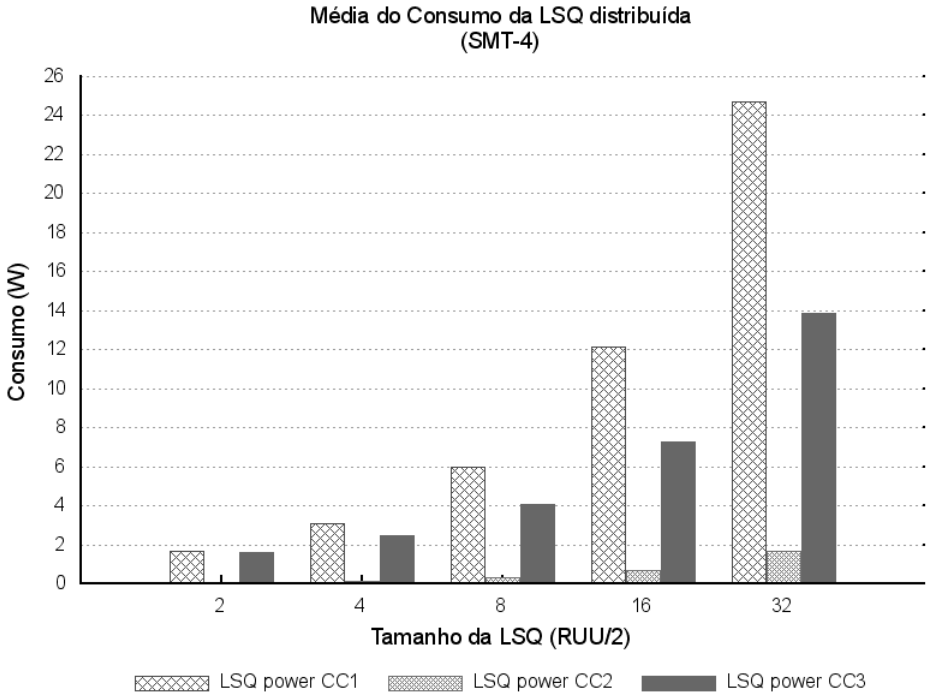

Figura 6.38: Consumo da LSQ distribuída no modo SMT-4

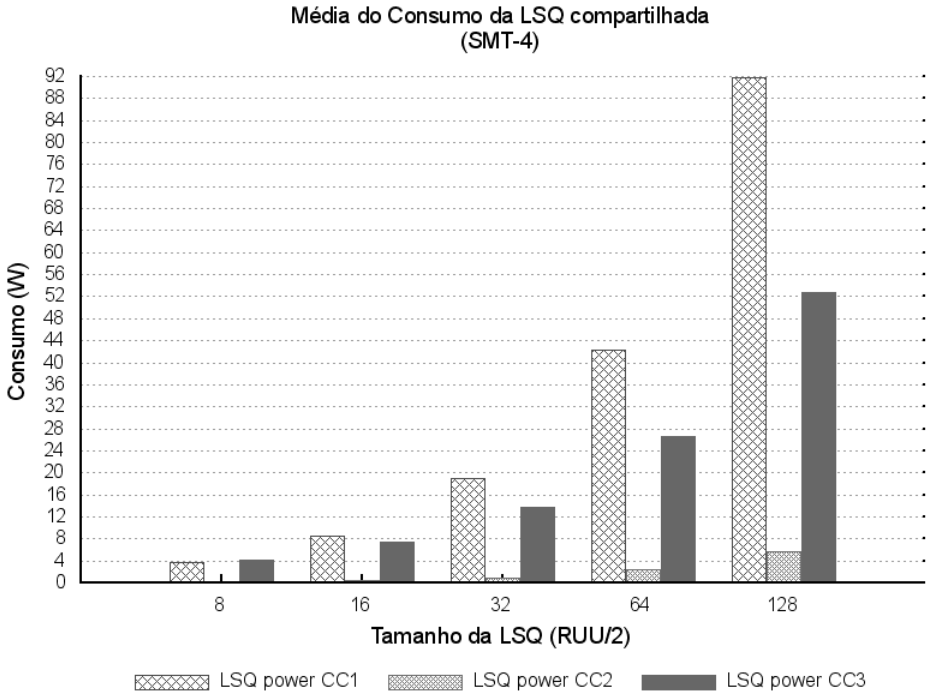

Figura 6.39: Consumo da LSQ compartilhada no modo SMT-4

As figuras 6.40 e 6.41 apresentam os consumos para a RUU, nos estilos CC2 e CC3 isolados do CC1 para que seja possível a visualização mais ajustada dos seus valores para as duas organizações.

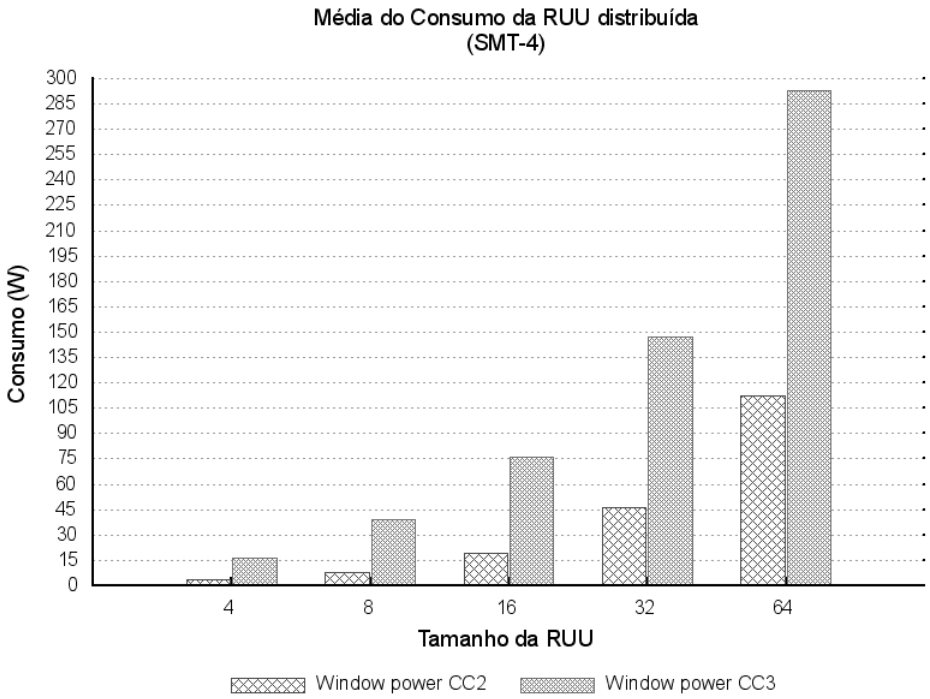

Figura 6.40: Consumo da RUU distribuída para SMT-4 nos estilos CC2 e CC3

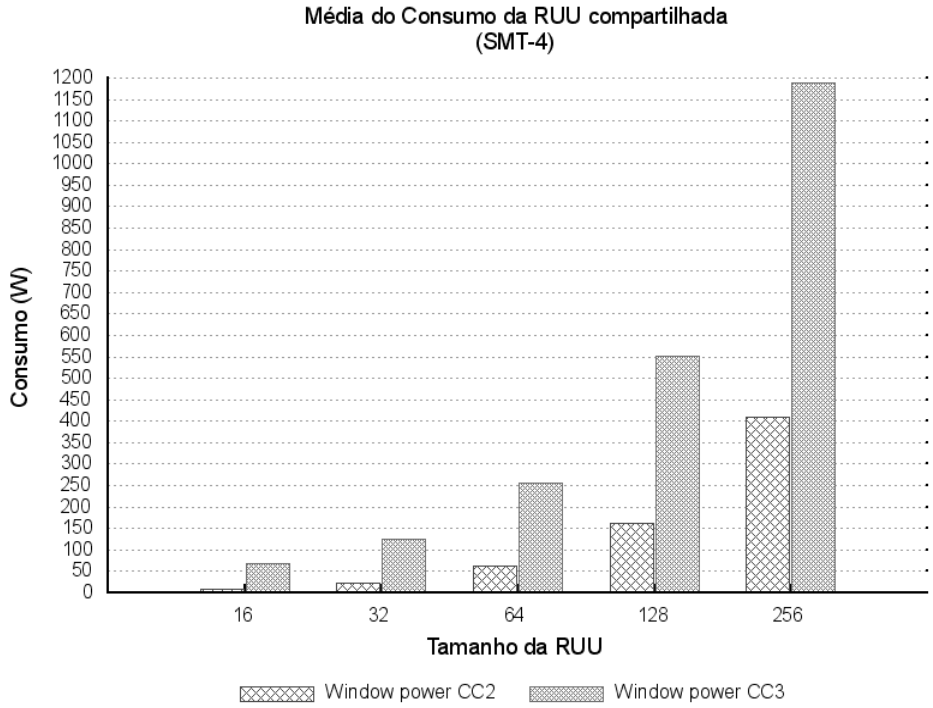

Figura 6.41: Consumo da RUU compartilhada para SMT-4 nos estilos CC2 e CC3

Os consumos para a RUU no modo de execução SMT-8 são apresentados nas figuras 6.42 e 6.43. Estes consumos são apresentados por configuração e por tamanho da RUU, seguindo as taxas de crescimento individual para cada um dos estilos de *clock*, sendo elas 70,21% e 100,25% para o CC1, de 139,27% e 169,76% para o CC2 e de 93,89% e 94,15% para o CC3 nas organizações distribuída e compartilhada, respectivamente.

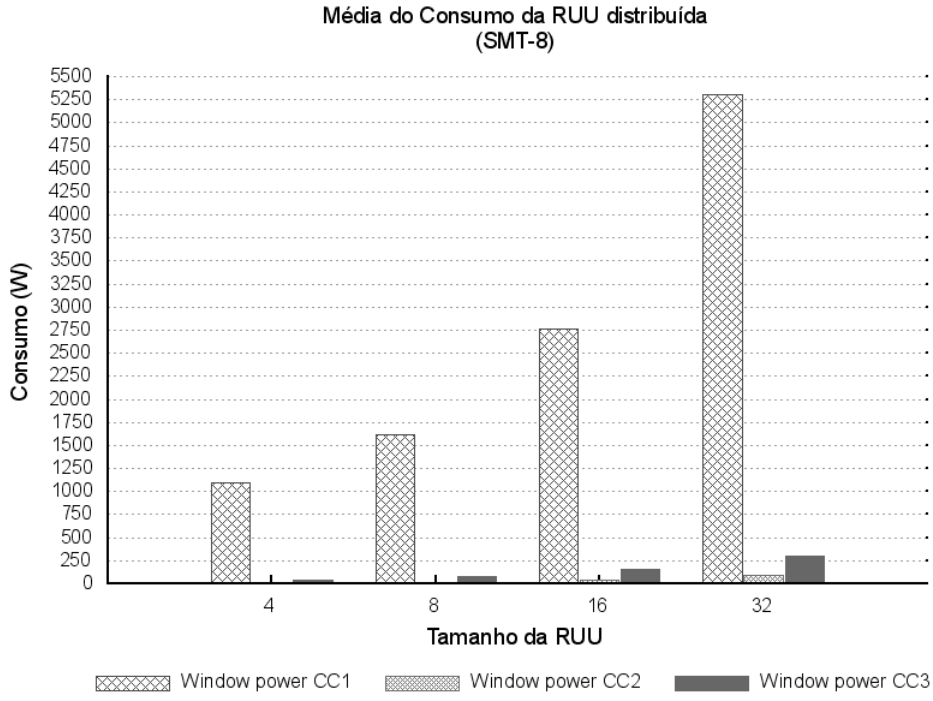

Figura 6.42: Consumo da RUU distribuída no modo SMT-8

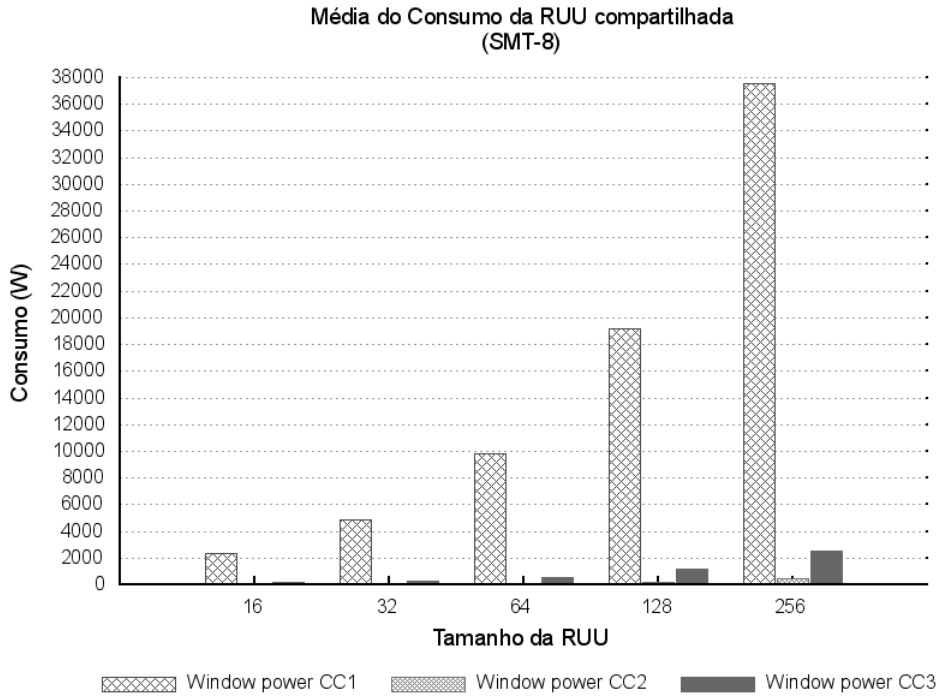

Figura 6.43: Consumo da RUU compartilhada no modo SMT-8

Para a LSQ, os consumos apresentados nas figuras 6.44 e 6.45, seguem as taxas de crescimento individual, 97,40% e 135,98% para o CC1, 132,75% e 165,59% para o CC2, e 66,71% e 88,39% para o CC3. Entre as duas organizações, o consumo é maior para a organização compartilhada.

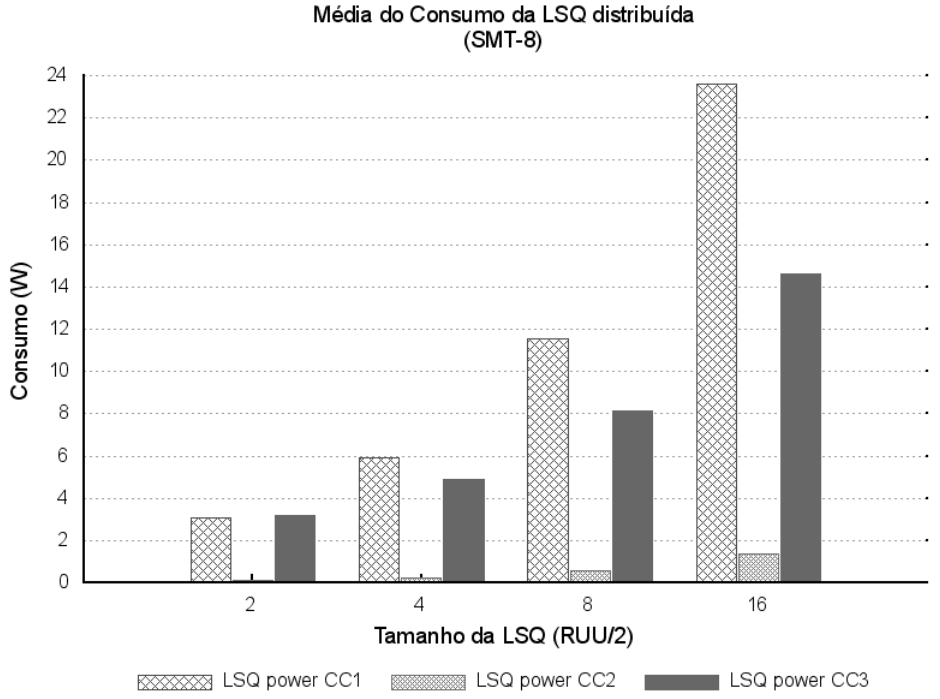

Figura 6.44: Consumo da LSQ distribuída no modo SMT-8

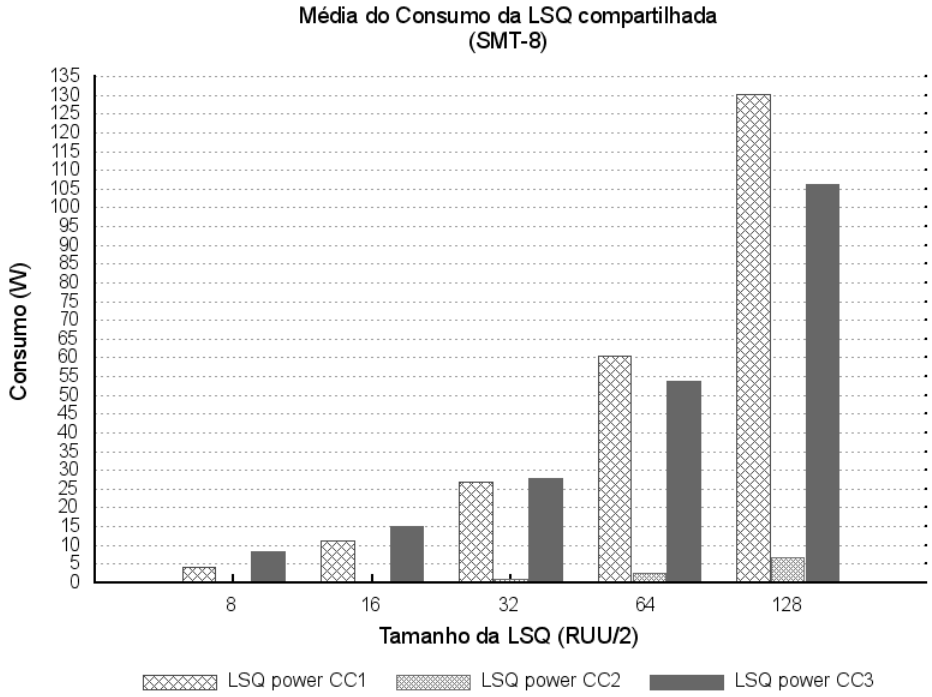

Figura 6.45: Consumo da LSQ compartilhada no modo SMT-8

As figuras 6.46 e 6.47 apresentam os consumos da RUU no modo SMT-8 nos estilos CC2 e CC3 mostrados nas figuras 6.42 e 6.43, porém isolando-os do consumo do estilo CC1 para melhorar a visualização desses dois outros estilos, que foi prejudicada pelos valores de CC1 serem superiores.

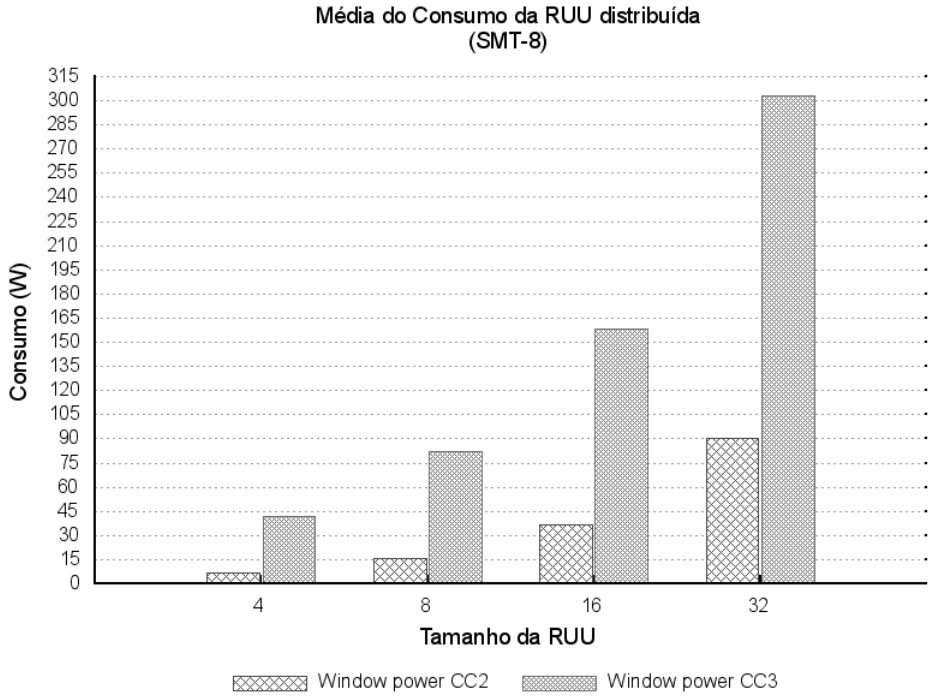

Figura 6.46: Consumo da RUU distribuída para SMT-8 nos estilos CC2 e CC3

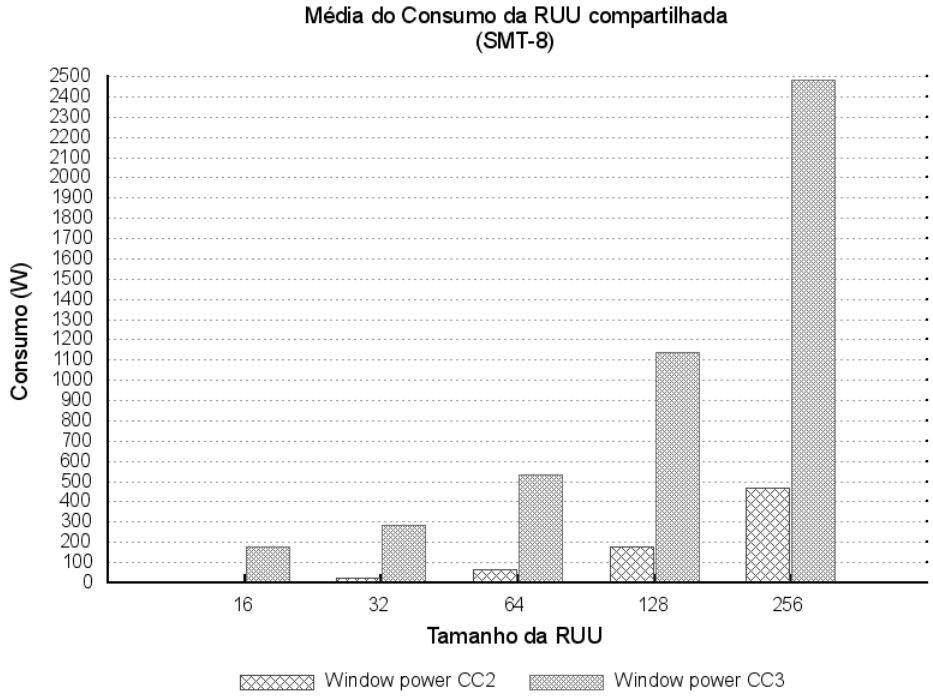

Figura 6.47: Consumo da RUU compartilhada para SMT-8 nos estilos CC2 e CC3

Os consumos para a RUU no modo SMT-16 são apresentados na Figura 6.48 para a organização distribuída e na Figura 6.49 para a organização compartilhada. Nesse modo de execução com 16 tarefas somente é possível comparar as configurações com tamanho de 64, 128 e 256 na compartilha com os equivalentes 4, 8 e 16 na distribuída. Embora os gráficos se apresentem em escalas diferentes é possível observar que o consumo é bem maior para a organização compartilhada. Sendo que esses valores seguem as taxas de crescimento individual em cada organização, sendo elas 57,86% (CC1), 131,43% (CC2) e 97,48% (CC3) na distribuída e 128,97% (CC1), 180,36% (CC2) e 74,20% (CC3) para a compartilhada.

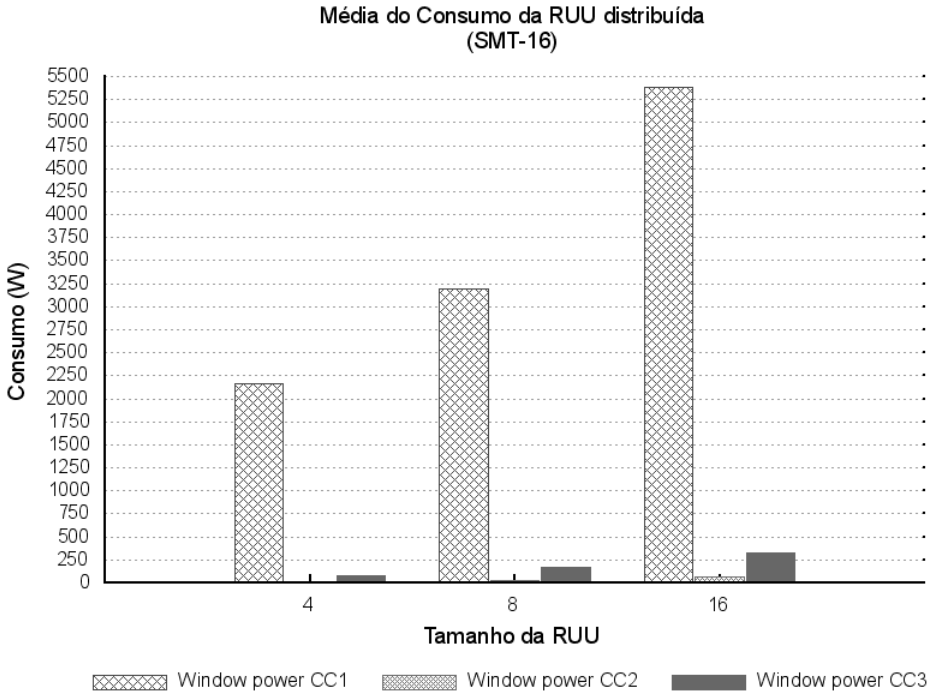

Figura 6.48: Consumo da RUU distribuída no modo SMT-16

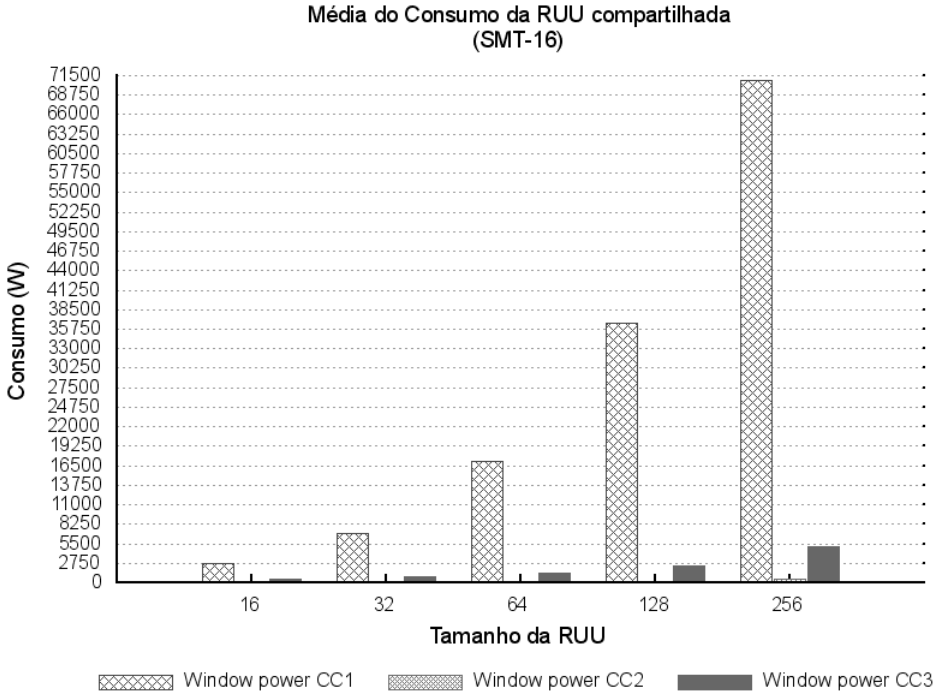

Figura 6.49: Consumo da RUU compartilhada no modo SMT-16

Para os consumos da LSQ os valores que devem ser comparados são 32, 64 e 128 na organização compartilhada com os valores de 2, 4, e 8 na distribuída, pois são as configurações equivalentes. Novamente pode-se notar pelos valores apresentados nas figuras 6.50 e 6.51 que o consumo da compartilhada é maior que o da distribuída.

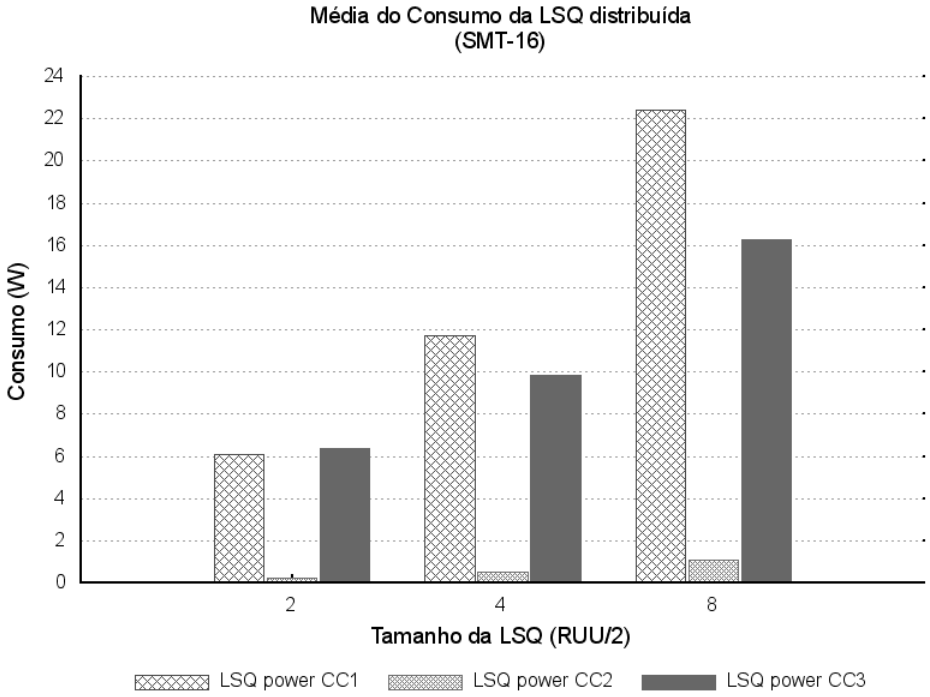

Figura 6.50: Consumo da LSQ distribuída no modo SMT-16

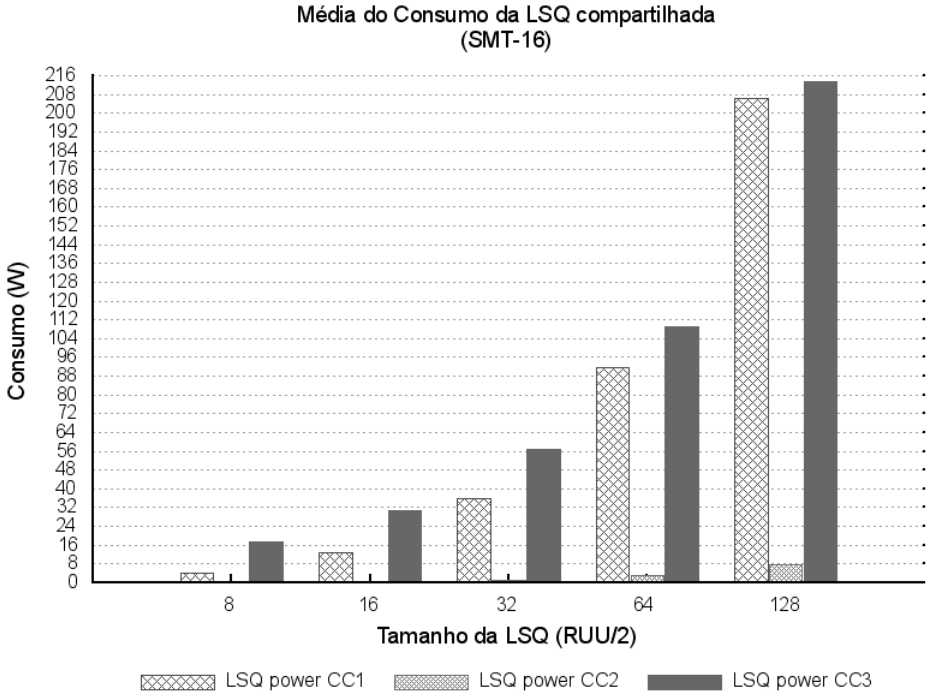

Figura 6.51: Consumo da LSQ compartilhada no modo SMT-16

As figuras 6.52 e 6.53 apresentam os consumos da RUU no modo SMT-16 para os estilos CC2 e CC3, de forma isolada do consumo para o estilo CC1, no intuito de possibilitar uma visualização mais clara em relação com o gráfico que apresenta os três estilos juntos.

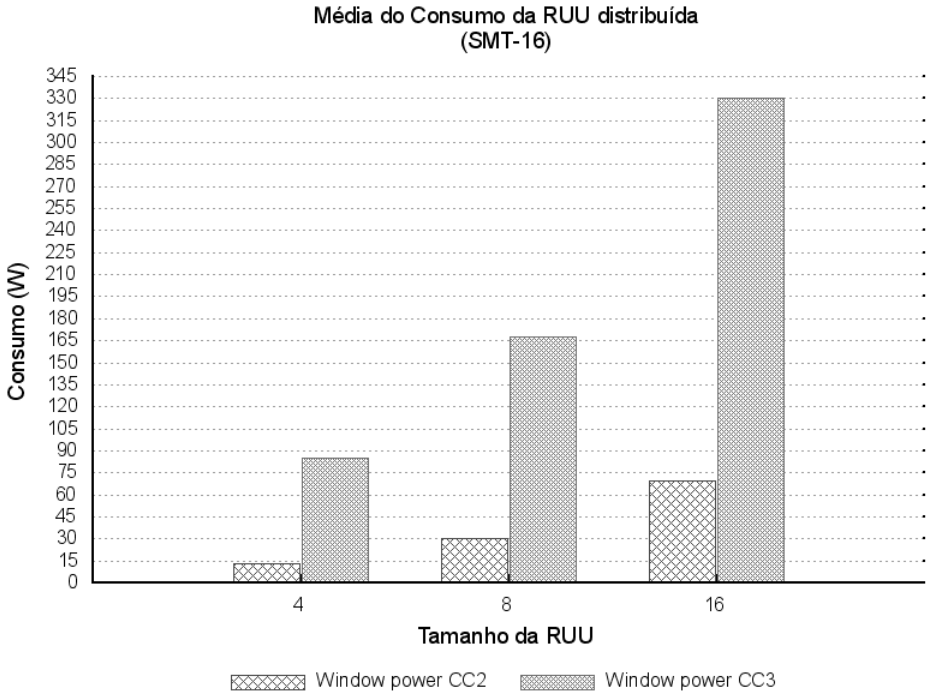

Figura 6.52: Consumo da RUU distribuída para SMT-16 nos estilos CC2 e CC3

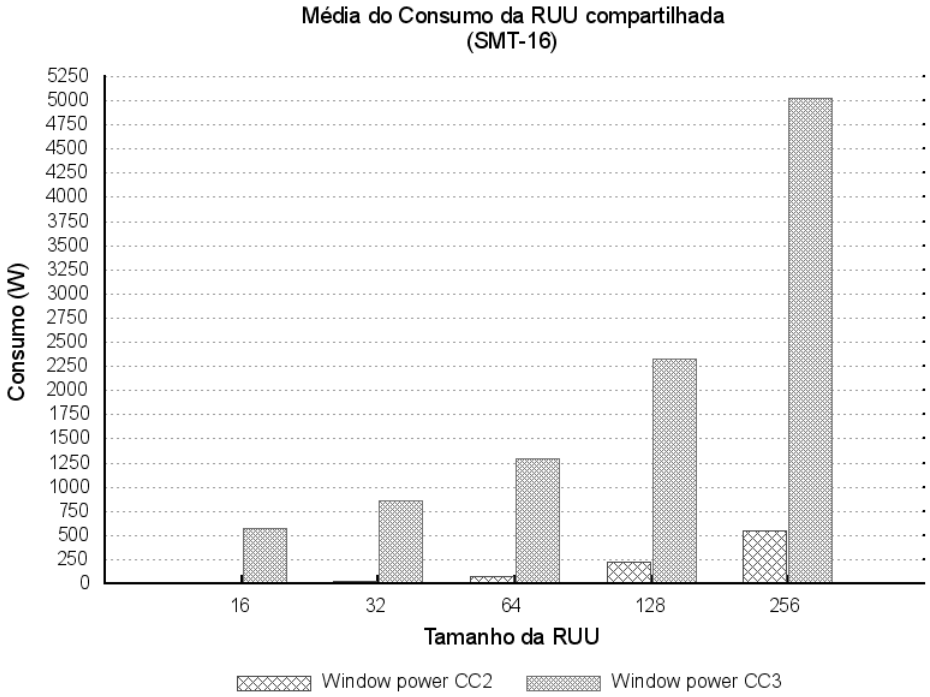

Figura 6.53: Consumo da RUU compartilhada para SMT-16 nos estilos CC2 e CC3

Avaliando-se as figuras 6.54 e 6.55 é possível comparar o desempenho alcançado para cada tamanho de RUU/LSQ na organização distribuída e na compartilhada individualmente para cada um dos modos de execução ou ainda comparar os resultados entre as organizações. Como por exemplo, comparar os resultados de uma RUU de tamanho 16 na organização distribuída com a equivalente

na compartilhada.

A análise desses resultados deve ser feita considerando-se as configurações que são possíveis tanto em uma organização quanto em outra. Pois dependendo do modo de execução, isto é, do número de tarefas executadas, somente um dos tipos de organização é possível. A divisão do número de entradas pela quantidade de tarefas sendo executadas simultaneamente, que deve ser maior que 2, assim sendo a configuração que está presente em ambas as organizações é a RUU de tamanho 16.

No modo de execução SMT-1 a organização compartilhada com RUU de tamanho 256 é 3% melhor que a equivalente distribuída, para os outros tamanhos menores o desempenho foi o mesmo em ambas as organizações para as configurações possíveis para o modo, na média para essas configurações possíveis nas duas organizações a compartilhada é 1% melhor que a distribuída. Porém pelo modo de execução, verifica-se que ter uma organização distribuída ou compartilhada para o modo de execução com apenas uma tarefa no fim acaba sendo a mesma coisa, pois na distribuída a quantidade de entradas é dividida pelo número de tarefas, no caso do modo SMT-1 seria uma divisão por um, sendo portanto estruturas equivalentes, o que justifica a concordância dos valores apresentados pelas duas organizações.

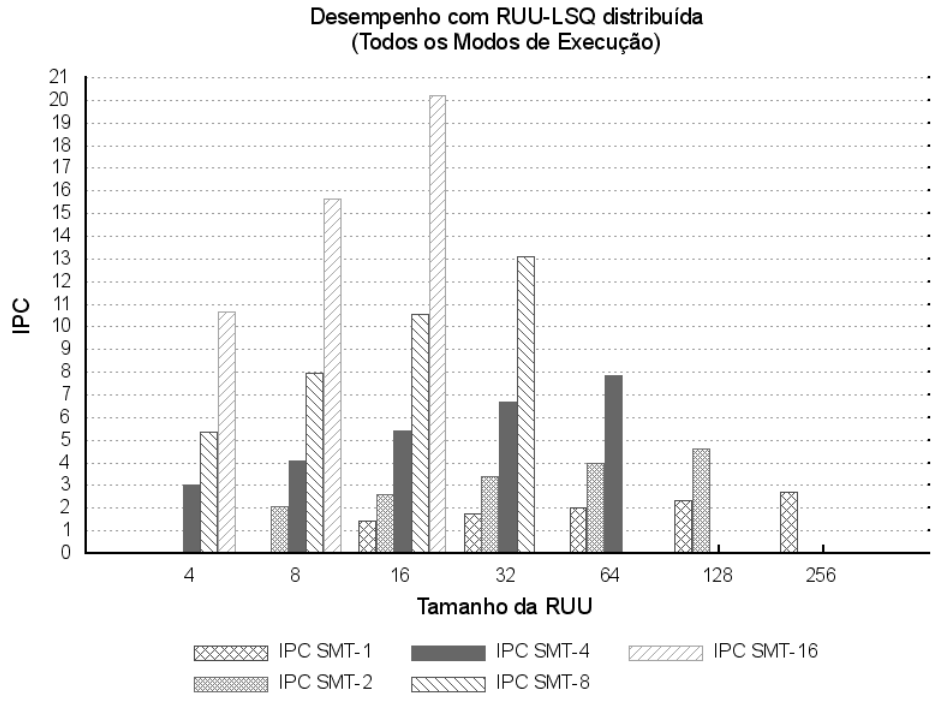

Figura 6.54: Desempenho com RUU-LSQ distribuída

A organização distribuída apresenta ganhos médios em relação à compartilhada de 30% para o modo SMT-2, de 89% no modo SMT-4, de 267% no SMT-8 e de 674% para o modo SMT-16.

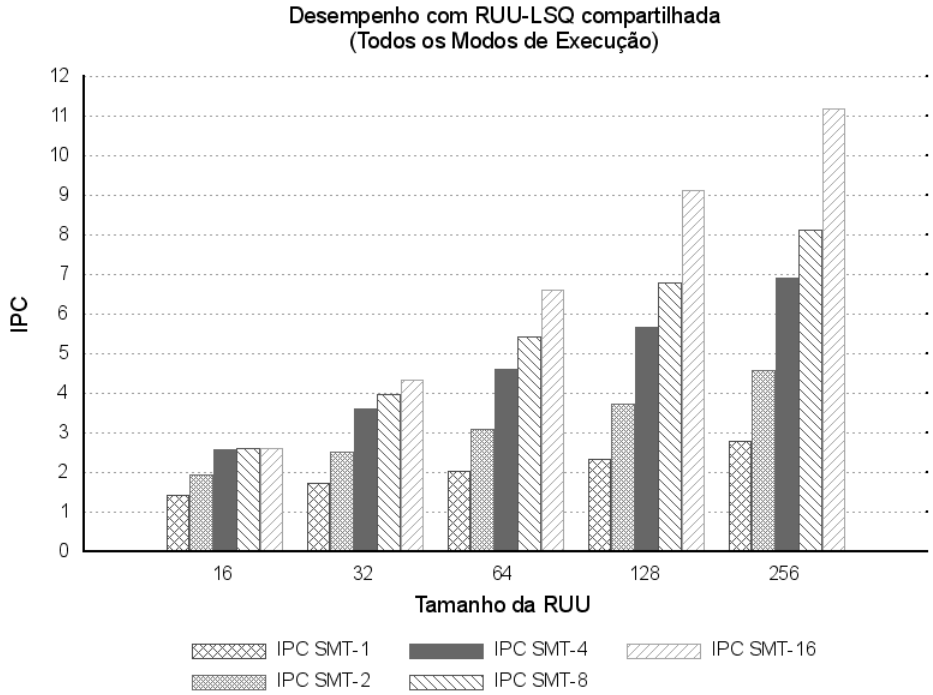

Figura 6.55: Desempenho com RUU-LSQ compartilhada

Um comparativo do desempenho entre as organizações distribuída e compartilhada pode ser feito por meio das figuras 6.56 e 6.57, as quais apresentam o IPC em cada modo de execução SMT. Percebe-se o ganho de desempenho em crescimento à medida que o número de tarefas também aumenta, sendo que na organização distribuída o ganho é maior que na compartilhada. Isto se dá pelo mesmo fato detectado na análise do consumo, nesse caso devido ao compartilhamento, o fluxo de instruções de cada tarefa individualmente é mais lento conforme já dito. Para que os fluxos de instruções sejam concluídos, isto é, para que todas as instruções de todas as tarefas passem pela estrutura, é consumido um número de ciclos bem maior que na distribuída, cujas estruturas são dedicadas à cada tarefa em particular. Isso ocorre na RUU compartilhada impactando no número de instruções executadas por ciclo (IPC), além da questão do consumo já mencionado.

O ganho de desempenho é perceptível na execução nos modos SMT-X à medida que o número de tarefas aumenta. O ganho com a organização distribuída em relação aos valores apresentados pela compartilhada pode ser visto pelo percentual da diferença entre seus valores: 0,00% para SMT-1, 5,24% para SMT-2, 15,77% para SMT-4, 37,27% para SMT-8 e 37,48% para SMT-16. No modo SMT-1 não houve ganho, pois as estruturas para a execução de uma tarefa apenas, torna-se a mesma, o ganho inicia a partir da execução de 2 tarefas, aumentando à medida que o número de tarefas aumenta.

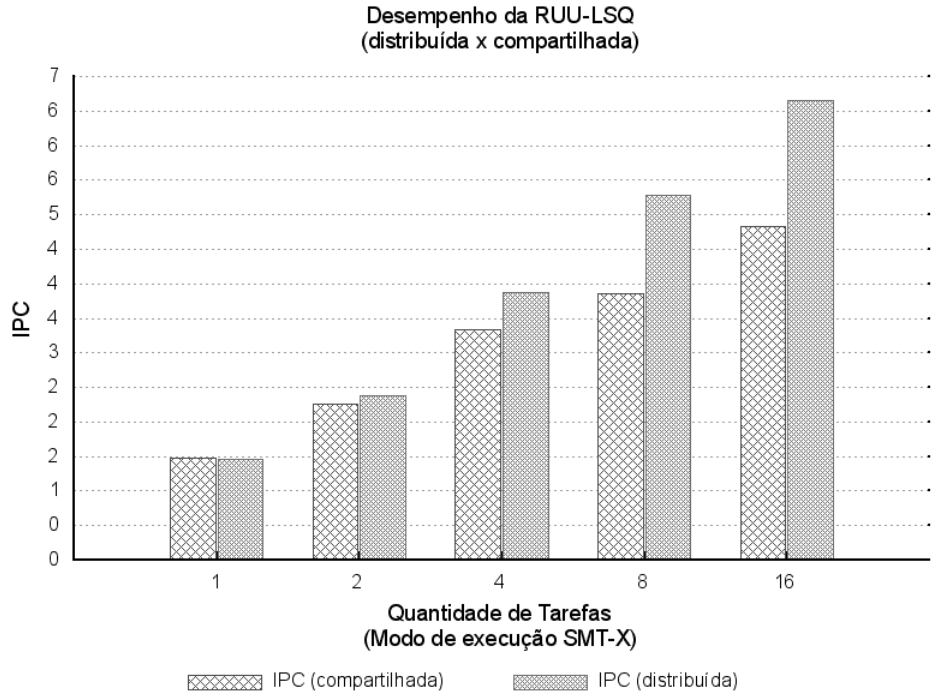

Figura 6.56: Comparativo do desempenho entre as organizações da RUU/LSQ

A Figura 6.57 apresenta a idéia do comportamento da função de crescimento do desempenho entre os modos de execução SMT-X.

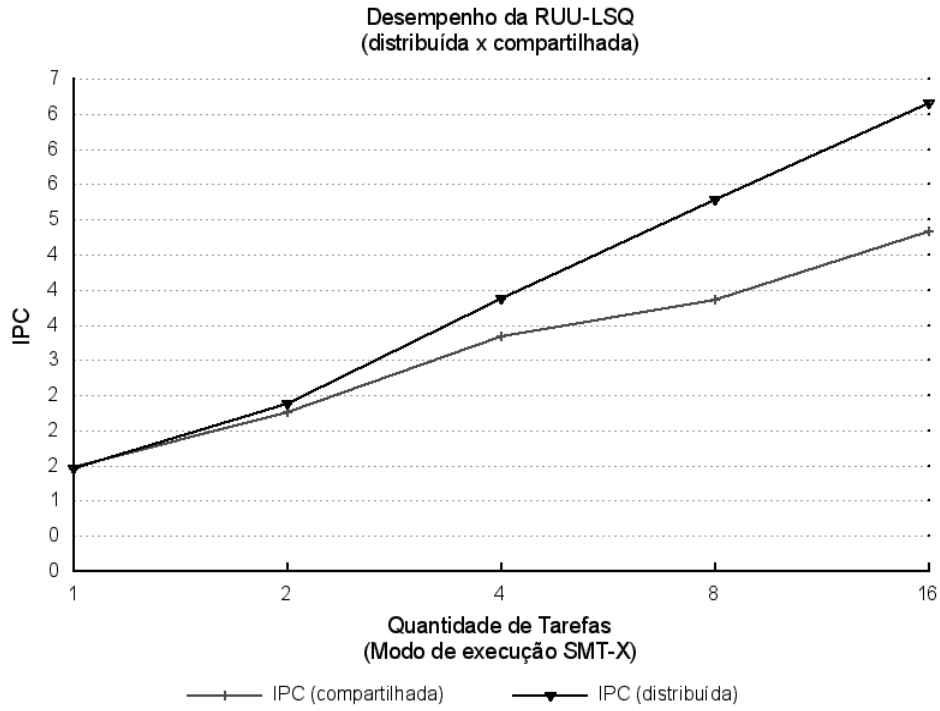

Figura 6.57: Curva de crescimento do desempenho entre as organizações da RUU/LSQ

As figuras 6.58 e 6.59 apresentam o desempenho individual das organizações entre as configurações equivalentes da RUU, considerando a idéia do experimento de variar a quantidade de entradas da RUU, estabelecendo uma quantidade total de entradas independente da organização

assumida. Sabe-se que o tamanho assumido pelas estruturas RUU e LSQ, são diferentes nas duas organizações conforme estão apresentados na Tabela 6.9. Na Figura 6.58, o gráfico apresenta os valores para a organização distribuída, os valores do eixo X são as somas de todos os tamanhos individuais de cada *slot* (totais da RUU em cada configuração), que foram organizados para serem comparados com a organização compartilhada, cujos resultados são apresentados pela Figura 6.59.

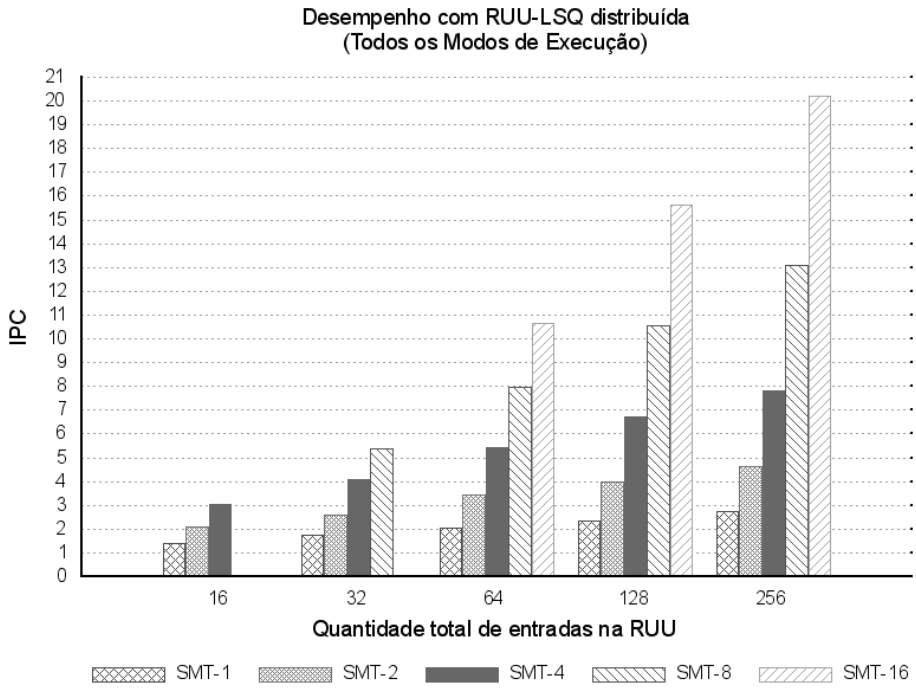

Figura 6.58: Desempenho individual das organização distribuída

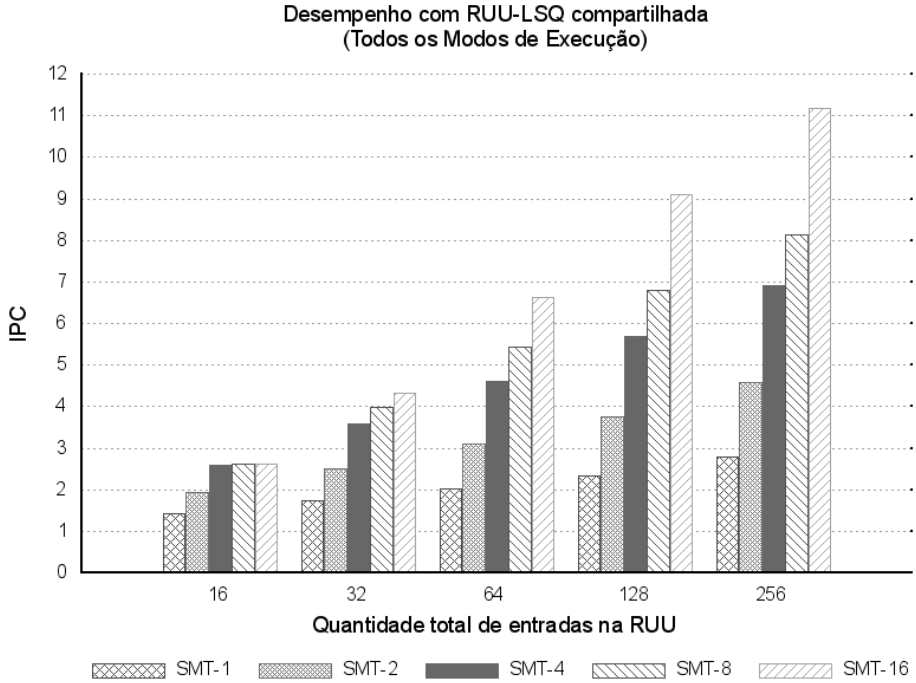

Figura 6.59: Desempenho individual das organização compartilhada

6.4.3 Conclusões do Experimento 4

Como pode-se notar entre as organizações, a compartilhada apresenta um consumo maior em relação à configuração equivalente na distribuída. Isso deve-se ao fato de que na compartilhada todas as tarefas estarão concentrando seus acessos a uma única estrutura (RUU/LSQ). Embora o número de acessos necessários para a execução do código da tarefa seja o mesmo nas duas organizações, na compartilhada o fluxo de instruções pela estrutura torna-se um pouco mais lento para cada tarefa individualmente, já que a estrutura é compartilhada por todas as tarefas que concorrem à ocupação de suas entradas. Considerando a ocupação mais lenta das entradas da estrutura compartilhada e o fato dos consumos serem computados por ciclo arquitetural, naturalmente o consumo da compartilhada será maior, pois a estrutura ficará ativa por mais ciclos.

6.5 Considerações Finais

Pelos experimentos apresentados é possível ter-se uma idéia da variabilidade de configurações possíveis de serem verificadas e da utilidade da ferramenta PowerSMT. O primeiro experimento mostrou a possibilidade de verificação do desempenho e do consumo da arquitetura como um todo, por meio do relacionamento entre as estruturas arquiteturais. Os outros experimentos, avaliaram partes específicas da arquitetura, o sistema de *cache*, as unidades funcionais e a janela de instruções (RUU/LSQ). Os experimentos realizados, de uma forma geral, mostraram as possibilidades de análise do desempenho e do consumo em um contexto global e em focos particulares utilizando o PowerSMT, atestando sua utilidade em pesquisas na área.

Capítulo 7

Conclusões e Trabalhos Futuros

O crescimento da área de computação com consumo eficiente de potência tem mostrado o papel importante que ferramentas de simulação podem ocupar no auxílio e na obtenção de resultados rápidos em tempo de projeto de novas arquiteturas. Os simuladores voltados ao projeto e avaliação de desempenho arquitetural e, mais recentemente, à avaliação de consumo de potência, têm a favor de seu desenvolvimento uma área em expansão.

A ferramenta PowerSMT permite estudar e avaliar as arquiteturas SMT, ainda na fase de projeto, considerando estimativas de desempenho e de consumo de potência. A ferramenta permite avaliar diferentes modelos arquiteturais e assim buscar maximizar a relação custo/benefício, proporcionando a obtenção de resultados de forma rápida e eficiente para arquiteturas SMT, atendendo assim a demanda por ferramentas que auxiliem pesquisas na área.

Os resultados experimentais apresentados neste trabalho e a análise feita sobre os mesmos mostram a utilidade e importância da ferramenta PowerSMT. Os próximos passos objetivam a inclusão de melhorias como o compartilhamento de estruturas que atualmente são privativas às tarefas. Além disso, a *clusterização* do *pipeline* de forma a permitir o uso de técnicas de desligamento e redução do consumo de forma automática em áreas não usadas é de grande interesse na continuidade deste trabalho.

Os experimentos apresentados no capítulo 6 abordam apenas algumas das muitas questões relacionadas às arquiteturas SMT e assim os resultados são parciais. Mais experimentos são necessários para levar a conclusões mais efetivas sobre o destino das arquiteturas SMT no referente ao consumo eficiente de energia. Acredita-se que a ferramenta desenvolvida nesse nível é útil à análise de consumo de potência, contribuindo para o estudo, projeto e avaliação de arquiteturas

SMT orientados à redução de consumo de potência, propiciando aos seus usuários resultados de forma rápida para auxílio em tempo de projeto.

Como continuidade para este trabalho pretende-se o desenvolvimento de melhorias no núcleo arquitetural, para que seja possível, por parametrização, a escolha de quais estruturas devem ser compartilhadas ou distribuídas, em adição às possibilidades existentes. Pretende-se ainda alterar a atual forma de distribuição dos recursos, de maneira que seja possível com uma quantidade específica de recursos executar os cenários SMT-X, sem a necessidade de que cada tarefa tenha o seu próprio *slot*, permitindo que o número de *slots* seja menor que a quantidade de tarefas.

O PowerSMT permite a simulação do sistema de cache organizado em módulos e bancos, sendo possível diversas organizações conforme parametrização. Os modos de execução SMT suportados vão de 1 tarefa até 16 tarefas nessa versão inicial, porém com as modificações que se pretende para o núcleo arquitetural esse número se tornará variável, sendo possível uma quantidade maior de tarefas.

Pretende-se ainda para facilitar a utilização da ferramenta a implementação de uma interface, na qual seja possível a escolha dos parâmetros de execução e o disparo de simulações, bem como a visualização dos resultados em forma de gráficos e tabelas. Não sendo necessariamente uma interface local, podendo ser uma interface de um sistema *web* no qual os usuários possam se cadastrar e submeter suas simulações para execução no PowerSMT e em outras ferramentas.

Portanto, a implementação do PowerSMT e a investigação, ambos componentes desta dissertação apresentam-se como uma contribuição importante para a área, justificando assim o estudo e o desenvolvimento deste projeto.

Referências

AMD: Advanced Micro Devices. PowerNow with optimized power management. http://www.amd.com/usen/0,,3715 12353,00.html, Jan. 2006.

BROOKS, D., TIWARI, V.; e MARTONOSI, M., Wattch: A framework for architectural-level power analysis and optimizations. In: 27th Annual International Symposium on Computer Architecture (ISCA), Proceedings..., pages 83--94. IEEE Computer Society Process, Los Alamitos, California, USA, 2000.

BROOKS, D., BOSE, P., SRINIVASAN, V., GSCHWIND, M., EMMA, P., ROSENFIELD, M., "Microarchitectural-Level Power-Performance Analysis: The PowerTimer Approach," IBM Journal of Research and Development, Oct/Nov 2003.

BROOKS, D., BOSE, P., and MARTONOSI, M., Power-Performance Simulation: Design and Validation Strategies. In: SIGMETRICS Perform. Eval. Rev. 31, 4 (Mar. 2004), pages 13-18, 2004.

BURGER, Doug, AUSTIN, Todd M.. The SimpleScalar Tool Set, Version 2.0. University of Wisconsin-Madison Computer Sciences Department. Technical Report #1342, Jun. 1997.

CAZORLA, F.J., FERNANDEZ, E. , RAMIREZ, A., e VALERO, M.. Improvingmemory Latency Aware Fetch Policies for SMT Processors. In: ISHPC-5, Proceedings..., pages 70–85, October 2003.

CAZORLA, F. J., RAMIREZ, A., VALERO, M., and FERNANDEZ, E.. Dynamically Controlled Resource Allocation in SMT Processors. In: 37th Annual IEEE/ACM International Symposium on Microarchitecture (Portland, Oregon, December 04 - 08, 2004), Proceeding..., International Symposium on Microarchitecture. IEEE Computer Society, Washington, DC, 171-182. 2004.

CONTE, T. M., TOBUREN, M. C., MENEZES, K. N., and SATHAYE, S. W., "System-level power consumption modeling and tradeoff analysis techniques for superscalar processor design", In: IEEE Transactions on Very Large Scale Integration (VLSI) Systems , vol.8, no.2, pp.129-137, Apr 2000.

DEHNERT, J. C.; GRANT, B. K.; BANNING, J. P.; JOHNSON, R.; KISTLER, T.; KLAIBER A.; "The Transmeta Code Morphing™ Software: Using Speculation, Recovery and Adaptative Retranslation to Address Real-Life Challenges", In: International Symposium on Code Generation and Optimization (CGO'03), Proceedings..., 2003.

FLAUTNER K., e MUDGE, T. Vertigo: automatic performance-setting for Linux. SIGOPS Oper. Syst. Rev. 36, SI (DEC. 2002), pages 105-116, New York, NY, USA, 2002A.

FLAUTNER, K.; KIM, Nam Sung; MARTIN, S.; BLAAUW, D.; MUDGE, T., "Drowsy caches: simple techniques for reducing leakage power," In: 29th Annual International Symposium on Computer Architecture, Proceedings..., vol., no., pp.148-157, 2002B.

FLEISCHMANN, M.; "LongRun™ Power Management. Dynamic Power Management for Crusoe™ Processors". White Paper. Transmeta Corporation, 17 de janeiro de 2001.

GIVARGIS, T. D.; VAHID, F. e HENKEL, J.. Evaluating Power Consumption of Parameterized Cache and Bus Architectures in System-on-a-Chip Designs. IEEE TRANS. VLSI SYSTEMS, vol. 9, pp. 500--508, Aug. 2001.

GONÇALVES, R. A. L. Arquiteturas Multi-Tarefas Simultâneas: SEMPRE – Arquitetura SMT com Capacidade de Execução e Escalonamento de Processos / por Ronaldo Augusto de Lara Gonçalves. – Porto Alegre : PPGC do II/UFRGS, 2000. Tese de doutorado.

GONÇALVES, R. A. L.; AYGUADÉ, E. P.; VALERO, M.; NAVAUX, P. O. A. A Simulator for SMT Architecture: Evaluating Instruction Cache Topologies, In: 12th Symposium on Computer Architecture and High Performance Computing, SBAC-PAD'00, 12., 2000, São Pedro, Brasil. Proceedings..., São Carlos-SP: SBC/UFSCar, Oct. 2000, pages 279–286.

GONÇALVES, R. A. L.; AYGUADÉ, E. P.; VALERO, M.; NAVAUX, P. O. A. Performance evaluation of decoding and dispatching stages in simultaneous multithreaded architectures. In: Computer Architecture and High Performance Computing, Proceedings..., 2001.

GROHOSKI, G., Niagara-2: A Highly Threaded Server-on-a-Chip, In: 18th Hot Chips Symposium, Palo Alto, CA, Aug, 2006.

HIRATA, H.; KIMURA, S.; NAGAMINE, S.; MOCHIZUKI, Y.; NISHIMURA, A.; NAKASE, Y.; NISHIZAWA, T., "An Elementary Processor Architecture with Simultaneous Instruction Issuing from Multiple Threads," In: 19th Annual International Symposium on Computer Architecture (19th ISCA), 1992. Proceedings..., vol., no., pp.136-145, 1992

HOMAYOUN, H.; LI, K.F.; RAFATIRAD, S., "Functional units power gating in SMT processors", In: IEEE Pacific Rim Conference on Computers and signal Processing, Communications (PACRIM 2005), vol., no., pp. 125-128, 24-26 Aug. 2005.

HU, Z., BROOKS, D., ZYUBAN, V., and BOSE, P.. Microarchitecture-level power-performance simulators: Modeling, validation, and impact on design, In: 36th Annual IEEE/ACM International Symposium on Microarchitecture, Tutorial, Dec. 2003.

HUANG, W., GHOSH, S., SANKARANARAYANAN, K., SKADRON, K., and STAN, M. R.. "HotSpot: Thermal Modeling for CMOS VLSI Systems." IEEE Transactions on Component Packaging and Manufacturing Technology. 2005.

Intel XScale Microarchitecture for the PXA255 Processor: User's Manual Intel Corporation, March 2003. Order No. 278796.

JOUPPI, N. and WILTON, S.. An enhanced access and cycle time model for on-chip caches. Technical Report TR-93-5, Compaq WRL, July 1994.

KALLA, R.; SINHAROY, B.; TENDLER, J.M., "IBM Power5 chip: a dual-core multithreaded processor," Micro, IEEE , vol.24, no.2, pp. 40-47, Mar-Apr 2004.

KANNAN, D.; GUPTA, A.; SHRIVASTAVA, A.; DUTT, N. D.; KURDAHI, F. J., "PTSMT: A Tool for Cross-Level Power, Performance, and Thermal Exploration of SMT Processors," In: 21st International Conference on VLSI Design (VLSID 2008), vol., no., pp.421-427, 4-8 Jan. 2008.

KIRKPATRICK, S.; C. D. GELATT, Jr.; and M. P., VECCHI, "Optimization by simulated anneling", Science, vol. 220, pp. 671-681, 1983.

KONGETIRA, P.; AINGARAN, K.; OLUKOTUN, K., "Niagara: a 32-way multithreaded Sparc processor", Micro, IEEE, vol.25, no.2, pp. 21-29, March-April 2005.

LAURENT, J.; JULIEN, N.; SENN, E.; MARTIN, E., "Functional level power analysis: an efficient approach for modeling the power consumption of complex processors," Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings..., vol.1, no., pp. 666-667 Vol.1, 16-20 Feb. 2004.

MADON, D., SANCHEZ, E., e MONNIER, S.. A Study of a Simultaneous Multithreaded Processor Implementation. In Euro-Par '99 Parallel Processing, 5th International Euro-Par Conference, Lecture Notes in Computer Science, vol. 1685, pages: 716-726, Springer, Toulouse, France, August 31 - September 3, 1999.

MCNAIRY, C.; BHATIA, R., "Montecito: a dual-core, dual-thread Itanium processor," Micro, IEEE , vol.25, no.2, pp. 10-20, March-April 2005.

MANNE, S.; KLAUSER, A.; GRUNWALD, D.. Pipeline gating: Speculation control for energy reduction. In: 25th Annual International Symposium on Computer Architecture, pages 132-141, Barcelona, Spain, June 1998.

MARUYAMA, T., "SPARC64 VI: Fujitsu's Next Generation Processor," In: Microprocessor Forum 2003, 2003.

MAMIDIPAKA, Mahesh and DUTT, Nikil. eCACTI: An Enhanced Power Estimation Model for On-chip Caches. University of California Irvine Center for Embedded Computer Systems Technical Report TR-04-28, Sept. 2004.

MARR, D. T.; BINNS, F.; HILL, D. L.; HINTON, G.; KOUFATY, D. A.; MILLER, J. A.; and UPTON, M.. Hyper-Threading Technology Architecture and Microarchitecture. Intel Technology Journal, 6(1), pages: 4-15, February 2002.

MOORE, G. E. Cramming more components onto integrated circuits," Electronics, vol. 38, no. 8, pp. 114-117, April 19, 1965.

MOUDGILL M., BOSE, P. and MORENO, J. H., "Validation of Turandot, a fast processor model for microarchitecture exploration" In: IEEE International Performance Computing and Communications Conference (IPCCC '99), vol., no., pp.451-457, 10-12 Feb 1999.

MUSOLL, Enric. Micro-architecture estimation of the useless power consumption of a high-performance processor. In: XII Symposium on Integrated Circuits and Systems Design, Proceedings..., pages: 4-7, 29 Sept.-2 Oct. 1999.

OLUKOTUN, K., NAYFEH, B. A., HAMMOND, L., WILSON, K., and CHANG, K.. The case for a single-chip multiprocessor. SIGPLAN Not. 31, 9 (Sep. 1996), pages 2-11, 1996.

PARIKH, D.; SKADRON, K.; YAN ZHANG; BARCELLA, M.; STAN, M.R.. Power issues related to branch prediction. In: 8th International Symposium on High-Performance Computer Architecture, Proceedings..., vol., no., pp. 233-244, 2-6 Feb. 2002

REINMAN, G., and JOUPPI, N.. "CACTI 2.0: An Integrated Cache Timing and Power Model," WRL Research Report 2000/7, Feb.2000.

SHIVAKUMAR, P. e JOUPPI, N. P.. CACTI 3.0: An Integrated Cache Timing, Power, and Area Model., Technical report, Compaq Computer Corporation August, 2001.

SMITH, J.E.; SOHI, G. S.. The microarchitecture of superscalar processors. Proceedings of the IEEE, Volume 83, Issue 12, Page(s):1609 – 1624, Dec. 1995.

SPEC CPU 2000. Standard Performance Evaluation Corporation. Site www.spec.org.

SPRACKLEN, L.; ABRAHAM, S.G., "Chip multithreading: opportunities and challenges", In: 11th International Symposium on High-Performance Computer Architecture (HPCA-11), vol., no., pp. 248-252, 12-16 Feb. 2005.

TARJAN, David, Shyamkumar THOZIYOOR, Norman P. JOUPPI. CACTI 4.0, HP Laboratories Palo Alto, HPL-2006-86, June 2, 2006.

THOZIYOOR, Shyamkumar, MURALIMANOHAR, Naveen, and JOUPPI, Norman P.. CACTI 5.0, Advanced Architecture Laboratory, HP Laboratories, HPL-2007-167, October 19, 2007.

THOZIYOOR, Shyamkumar, MURALIMANOHAR, Naveen, AHN, Jung Ho, and JOUPPI, Norman P.. CACTI 5.1. HP Laboratories, Palo Alto, HPL-2008-20, April 2, 2008.

TULLSEN, D. M.; EGGERS, S. J.; LEVY, H. M., "Simultaneous Multithreading: Maximizing on-chip Parallelism," In: 22th Annual International Symposium on Computer Architecture, Proceedings..., vol., no., pp. 392-403, 22-24, Jun 1995.

TULLSEN, D. M., EGGERS, S. J., EMER, J. S., LEVY, H. M., LO, J. L., and STAMM, R. L. 1996. Exploiting choice: instruction fetch and issue on an implementable simultaneous multithreading processor. In Proceedings of the 23rd Annual International Symposium on Computer Architecture (Philadelphia, Pennsylvania, United States, May 22 - 24, 1996). ISCA '96. ACM, New York, NY, 191-202.

TULLSEN, D. e BROWN, J.. Handling long-latency loads in a simultaneous multithreaded processor. In Proc. MICRO-34, pages 318–327, 2001.

VENKATACHALAM, V. e FRANZ, M. 2005. Power reduction techniques for microprocessor systems. ACM Comput. Surv. 37, 3 (Sep. 2005), 195-237.

YAMAMOTO, W.; NEMIROVSKY, M.. Performance Estimation of Multistreamed, Superscalar Processors, In: Hawaii International Conference on System Sciences, Proceedings..., Janeiro. 1994.

YE, W., VIJAYKRISHNAN, N., KANDEMIR, M., and IRWIN, M. J. 2000. The design and use of simplepower: a cycle-accurate energy estimation tool. In: 37th Conference on Design Automation (Los Angeles, California, United States, June 05 - 09, 2000). DAC '00, Proceedings..., ACM, New York, NY, 340-345.

WILTON, S.J.E.; JOUPPI, N.P., "CACTI: An enhanced cache access and cycle time model," Solid-State Circuits, IEEE Journal of , vol.31, no.5, pp.677-688, May 1996.

ZHANG, Y., PARIKH, D., SANKARANARAYANAN, K, SKADRON, K., and STAN, M.. HotLeakage: A temperatureaware Model of Subthreshold and Gate Leakage for Architects. TR-CS-2003-05, University of Virginia, Dept of Computer Science, March 2003.

Anexos

Arquivos por versão da biblioteca CACTI.

No estudo e mapeamento funcional feito do código da biblioteca CACTI, comparou-se as implementações das versões desde a utilizada pelo modelo de consumo do sim-wattch até as versões mais recentes.

A Tabela 1 apresenta os arquivos da CATI 1.0 a utilizada pelo modelo de consumo, tendo somente a implementação do *Time Model*. A Tabela 2 apresenta os arquivos da CACTI 2.0, a Tabela 3 lista os arquivos da CACTI 3.0, tendo a adição do *Area Model.*

A Tabela 4 mostra os arquivos da eCACTI com definições de *leakage power*, que depois foi incorporado pela implementação da CACTI 4.0 no *Leakage Model* como pode ser visto na Tabela 5 e por fim a Tabela 6 apresenta uma lista dos arquivos da versão corrente da biblioteca CACTI, que atualmente está com a versão 5.0 como sendo a principal.

Tabela 1: Arquivos da CACTI 1.0

Arquivo	Descrição
main.c	Programa principal que chama as outras funcionalidades.
io.c	Funções *input/output*.
def.h	Define os parâmetros específicos da tecnologia utilizados pela biblioteca, tais como largura do transistor, limiares de voltagem e etc.
time.c	Implementação do *Time Model.*

Tabela 2: Arquivos da CACTI 2.0

Arquivo	Descrição
main.c	Programa principal que chama as outras funcionalidades.
io.c	Funções *input/output*.
def.h	Define os parâmetros específicos da tecnologia utilizados pela

Arquivo	Descrição
	biblioteca, tais como largura do transistor, limiares de voltagem e etc.
time.c	Implementação do *Time Model.*
cache_params.aux	Após a execução da CACTI, conterá a configuração descoberta, para ser utilizado no *spice cache model*.

Tabela 3: Arquivos da CACTI 3.0

Arquivo	Descrição
main.c	Programa principal que chama as outras funcionalidades.
io.c	Funções *input/output*.
def.h	Define os parâmetros específicos da tecnologia utilizados pela biblioteca, tais como largura do transistor, limiares de voltagem e etc.
time.c	Implementação do *Time Model.*
areadef.h	Definições do *Area Model.*
area.c	implementação do *Area Model.*

Tabela 4: Arquivos da eCACTI

Arquivo	Descrição
main.c	Programa principal que chama as outras funcionalidades.
io.c	Funções *input/output*.
def.h	Define os parâmetros específicos da tecnologia utilizados pela biblioteca, tais como largura do transistor, limiares de voltagem e etc.
time.c	Implementação do *Time Model.*
areadef.h	Definições do *Area Model.*
area.c	Implementação do *Area Model.*
leakage.h	Definições para *Leakage Power.*
leakage.c	Implementação para *Leakage Power.*

Tabela 5: Arquivos da CACTI 4.0

Arquivo	Descrição
main.c	Programa principal para a utilização da cacti como aplicação.
io.h	Define as assinaturas das funções relacionadas à entrada e saída de parâmetros e resultados e uma função de inicialização dos parâmetros *default* da tecnologia (init_tech_params_default_process).
io.c	Implementação das funções definidas no io.h, é neste arquivo que está implementada a função `total_result_type` **`cacti_interface`** que está definida no arquivo cacti_interface.h. Esta função é a que faz a interface da cacti para que possa ser utilizada como biblioteca por outras aplicações, com uma chamada, os parâmetros são inicializados e o resultado retornado.
def.h	Define os parâmetros específicos da tecnologia utilizados pela biblioteca, tais como largura do transistor, limiares de voltagem e etc.
time.h	Implementação do *Time Model.*
time.c	Definições do *Area Model.*
areadef.h	Implementação do *Area Model.*
area.c	Define os parâmetros específicos da tecnologia utilizados pela biblioteca, tais como largura do transistor, limiares de voltagem e etc.
basic_circuit.h	Define as assinaturas das funções de relacionadas aos circuitos básicos.
basic_circuit.c	Implementação das funções relacionadas aos circuitos básicos definidas em basic_circuit.h. Funções estas que nas versões iniciais da CACTI eram implementadas no arquivo time.c.
leakage.h	Definições para *Leakage Power.*
leakage.c	Implementação para *Leakage Power.*
cacti_interface.h	Definições dos tipos de dados, funções da interface.
cacti_wrap.c	Arquivo gerado pelo comando *swig –python cacti.i* (swig é um comando que gera interfaces de códigos C/C++ para diversas linguagens (python, perl, Java, php...). O cacti_wrap.c pode ser compilada gerando uma biblioteca _cacti.so/_cacti.dll. A convenção do nome é necessária para que o *runtime* do python reconheça como sendo uma

Arquivo	Descrição
	biblioteca do python para que possa ser importada e utilizada.

Tabela 6: Arquivos da CACTI 5.0

Arquivo	Descrição
main.c	Programa principal para a utilização da cacti como aplicação.
io.h	Define as assinaturas das funções relacionadas à entrada e saída de parâmetros e resultados e uma função de inicialização dos parâmetros *default* da tecnologia (init_tech_params_default_process).
io.c	Implementação das funções definidas no io.h, é neste arquivo que está implementada a função `total_result_type cacti_interface` definida no arquivo cacti_interface.h. Esta função é a que faz a interface da cacti para que possa ser utilizada como biblioteca por outras aplicações, com uma chamada, os parâmetros são inicializados e o resultado retornado.
def.h	Define os parâmetros específicos da tecnologia utilizados pela biblioteca, tais como largura do transistor, limiares de voltagem e etc.
time.h	Implementação do *Time Model.*
time.c	Definições do *Area Model.*
areadef.h	Implementação do *Area Model.*
area.c	Define os parâmetros específicos da tecnologia utilizados pela biblioteca, tais como largura do transistor, limiares de voltagem e etc.
basic_circuit.h	Define as assinaturas das funções de relacionadas aos circuitos básicos.
basic_circuit.c	Implementação das funções relacionadas aos circuitos básicos definidas em basic_circuit.h. Funções estas que nas versões iniciais da CACTI eram implementadas no arquivo time.c.
leakage.h	Definições para *Leakage Power.*
leakage.c	Implementação para *Leakage Power.*
cacti_interface.h	Definições dos tipos de dados, funções da interface.

Arquivo	Descrição
cacti_wrap.c	Arquivo gerado pelo comando *swig –python cacti.i* (swig é um comando que gera interfaces de códigos C/C++ para diversas linguagens (python, perl, Java, php...). O cacti_wrap.c pode ser compilada gerando uma biblioteca _cacti.so/_cacti.dll. A convenção do nome é necessária para que o *runtime* do python reconheça como sendo uma biblioteca do python para que possa ser importada e utilizada.

Sobre a utilização do PowerSMT

O Quadro 5.27 apresenta um exemplo de arquivo de definição de *benchmarks* que serão utilizados como tarefas na execução do simulador PowerSMT.

Quadro 1: Exemplo de Arquivo de Definição de Tarefas

```
# File generated by Script (./ss_smtCommandLines.sh) at
08/11/2007-14:35:12.
# author: Rogerio A. Goncalves (rogerio.rag@gmail.com)
# Run SPEC2000 benchmarks.
# This script must be executed in $SPEC2000_INPUTS, because the inputs of
benchmarks inputs.
#1 Integer
$BENCH_BIN_PATH/gzip.ss-little.O3 $SPEC2000_INPUTS/input.source 60
#2 Integer
$BENCH_BIN_PATH/vpr.ss-little.O3 $SPEC2000_INPUTS/net.in
$SPEC2000_INPUTS/arch.in $SPEC2000_INPUTS/place.out
$SPEC2000_INPUTS/dum.out -nodisp -place_only -init_t 5 -exit_t 0.005
-alpha_t 0.9412 -inner_num 2
#3 Integer
$BENCH_BIN_PATH/cc1.ss-little.O3 $SPEC2000_INPUTS/200.i -o ./cc1_200.s
#4 Integer
$BENCH_BIN_PATH/mcf.ss-little.O3 $SPEC2000_INPUTS/mcf_inp.in
#5 Integer
$BENCH_BIN_PATH/parser.ss-little.O3 $SPEC2000_INPUTS/2.1.dict -batch <
$SPEC2000_INPUTS/ref.in
#6 Integer
$BENCH_BIN_PATH/vortex.ss-little.O3 $SPEC2000_INPUTS/lendian1.raw
#7 Integer
$BENCH_BIN_PATH/bzip2.ss-little.O3 $SPEC2000_INPUTS/input.source 58
#8 Integer
$BENCH_BIN_PATH/twolf.ss-little.O3 $SPEC2000_INPUTS/twolf.ref
#9 float Point
$BENCH_BIN_PATH/wupwise.ss-little.O3
#10 float Point
$BENCH_BIN_PATH/swim.ss-little.O3 < $SPEC2000_INPUTS/swim.in
#11 float Point
$BENCH_BIN_PATH/mgrid.ss-little.O3 < $SPEC2000_INPUTS/mgrid.in
#12 float Point
$BENCH_BIN_PATH/applu.ss-little.O3 < $SPEC2000_INPUTS/applu.in
#13 float Point
$BENCH_BIN_PATH/mesa.ss-little.O3 -frames 1000 -meshfile
```

```
$SPEC2000_INPUTS/mesa.in -ppmfile $SPEC2000_INPUTS/mesa.ppm
#14 float Point
$BENCH_BIN_PATH/art.ss-little.O3 -scanfile $SPEC2000_INPUTS/c756hel.in
-trainfile1 $SPEC2000_INPUTS/a10.img -trainfile2 $SPEC2000_INPUTS/hc.img
-stride 2 -startx 110 -starty 200 -endx 160 -endy 240 -objects 10
#15 float Point
$BENCH_BIN_PATH/equake.ss-little.O3 < $SPEC2000_INPUTS/equake_inp.in
#16 float Point
$BENCH_BIN_PATH/ammp.ss-little.O3 < $SPEC2000_INPUTS/ammp.in
#17 float Point
$BENCH_BIN_PATH/sixtrack.ss-little.O3 < $SPEC2000_INPUTS/sixtrack_inp.in
#18 float Point
$BENCH_BIN_PATH/apsi.ss-little.O3
#19 Integer
$BENCH_BIN_PATH/gzip.ss-little.O3 $SPEC2000_INPUTS/input.graphic 60
#20 Integer
$BENCH_BIN_PATH/gzip.ss-little.O3 $SPEC2000_INPUTS/input.random 60
#21 Integer
$BENCH_BIN_PATH/gzip.ss-little.O3 $SPEC2000_INPUTS/input.program 60
#22 Integer
$BENCH_BIN_PATH/gzip.ss-little.O3 $SPEC2000_INPUTS/input.log 60
#23 Integer
$BENCH_BIN_PATH/vpr.ss-little.O3 $SPEC2000_INPUTS/net.in
$SPEC2000_INPUTS/arch.in $SPEC2000_INPUTS/place.in
$SPEC2000_INPUTS/route.out -nodisp -route_only -route_chan_width 15
-pres_fac_mult 2 -acc_fac 1 -first_iter_pres_fac 4 -initial_pres_fac 8
#24 Integer
$BENCH_BIN_PATH/cc1.ss-little.O3 $SPEC2000_INPUTS/integrate.i -o
./cc1_integrate.s
#25 Integer
$BENCH_BIN_PATH/cc1.ss-little.O3 $SPEC2000_INPUTS/166.i -o ./cc1_166.s
#26 Integer
$BENCH_BIN_PATH/cc1.ss-little.O3 $SPEC2000_INPUTS/scilab.i -o
./cc1_scilab.s
#27 Integer
$BENCH_BIN_PATH/cc1.ss-little.O3 $SPEC2000_INPUTS/expr.i -o ./cc1_expr.s
#28 Integer
$BENCH_BIN_PATH/vortex.ss-little.O3 $SPEC2000_INPUTS/lendian2.raw
#29 Integer
$BENCH_BIN_PATH/vortex.ss-little.O3 $SPEC2000_INPUTS/lendian3.raw
#30 Integer
$BENCH_BIN_PATH/bzip2.ss-little.O3 $SPEC2000_INPUTS/input.graphic 58
#31 Integer
$BENCH_BIN_PATH/bzip2.ss-little.O3 $SPEC2000_INPUTS/input.program 58
#32 float Point
$BENCH_BIN_PATH/art.ss-little.O3 -scanfile $SPEC2000_INPUTS/c756hel.in
-trainfile1 $SPEC2000_INPUTS/a10.img -trainfile2 $SPEC2000_INPUTS/hc.img
-stride 2 -startx 470 -starty 140 -endx 520 -endy 180 -objects 10
```

Quadro 2: Exemplo de Execução com a lista de parâmetros aceita pelo PowerSMT

```
PowerSMT: SMT Power Consumption Simulator by Rogerio A. Goncalves
Version: 1.0 of July, 2008
-[Using]-----------------------------------------------------------------
 |+ SS-SMT: SMT Simulator by Ronaldo A. L. Goncalves Version: 1.0 of 2000.
All Rights Reserved
    |+ Sim-Outorder/SimpleScalar/PISA Tool Set version 3.0 of September,
1998. Copyright (c) 1994-1998 by Todd M. Austin.  All Rights Reserved.
 |+ Sim-Wattch: Wattch Simulator by David Brooks Version: 1.2 of 2000. All
Rights Reserved.
    |+ Sim-Outorder/SimpleScalar/PISA Tool Set version 3.0 of September,
1998.
    |+ CACTI Version: 1.0 of 1994
       Copyright 1994 Digital Equipment Corporation and Steve Wilton. All
Rights Reserved
 |+ CACTI Version: 4.0 of 2005
    Copyright 2005 Hewlett-Packard Development Corporation. All Rights
Reserved
-------------------------------------------------------------------------
*** Calculating dl1 nsets parameter: dl1->nsets = 128
*** Calculating dl2 nsets parameter: dl2->nsets = 1024
*** Calculating il1 nsets parameter: il1->nsets = 512
*** Calculating il2 nsets parameter: il2 nsets = dl2 nsets
*** Calculating itlb nsets parameter: itlb->nsets = 16
*** Calculating dtlb nsets parameter: dtlb->nsets = 32

sim: command line: ./powerSMT 1 2 -imodules:num 2 ./batch/RUN_TESTE

*** Technology process considered: TECH_070 (0.07)

*** Calculating btb consumption parameters
*** Calculating il1 consumption parameters
*** Calculating dl1 consumption parameters
*** Calculating dl2 consumption parameters

Processor Parameters:
Issue Width: 4
Window Size: 16
Number of Virtual Registers: 32
Number of Physical Registers: 16
Datapath Width: 64
Total Power Consumption: 70.1657
Branch Predictor Power Consumption: 0.591816  (0.868%)
 branch target buffer power (W): 0.490903
      btb selector power (W): 0.000520306
 local predict power (W): 0.0247262
 global predict power (W): 0.0274997
 chooser power (W): 0.0197892
 RAS power (W): 0.0288974
Rename Logic Power Consumption: 0.0759256  (0.111%)
 Instruction Decode Power (W): 0.00401954
 RAT decode_power (W): 0.031571474
 RAT wordline_power (W): 0.00717952
 RAT bitline_power (W): 0.0308622
 DCL Comparators (W): 0.00229286
Instruction Window Power Consumption: 0.304957  (0.447%)
 tagdrive (W): 0.0195795
 tagmatch (W): 0.00697781
 Selection Logic (W): 0.00303868
```

```
 decode_power (W): 0.0154453
 wordline_power (W): 0.0204961
 bitline_power (W): 0.23942
 RUU selector_power (W): 0
Load/Store Queue Power Consumption: 0.178786  (0.262%)
 tagdrive (W): 0.100931
 tagmatch (W): 0.0217196
 decode_power (W): 0.00200977
 wordline_power (W): 0.00314357
 bitline_power (W): 0.0509818
Arch. Register File Power Consumption: 0.464967  (0.682%)
 decode_power (W): 0.0315715
 wordline_power (W): 0.0204961
 bitline_power (W): 0.41238
 selector_power (W): 0.000520306
Instruction Fetch Queue Power Consumption: 0.0408683  (0.06%)
 decode_power (W): 0.00100658
 wordline_power (W): 0.000496026
 bitline_power (W): 0.0392677
 ifq selector_power (W): 9.80408e-05
Result Bus Power Consumption: 1.69948  (2.49%)
Total Clock Power: 45.9089  (67.3%)
Int ALU Power: 3.09557  (4.54%)
Int Mult/DIV Power: 0.596369  (0.875%)
Memory Port Power: 0  (0%)
FP ALU Power: 9.48643  (13.9%)
FP Mult/Div Power: 2.37161  (3.48%)
Div/Mult Power: 0  (0%)
Homo Power: 0  (0%)
Instruction Cache Power Consumption: 0.897648  (1.32%)
 decode_power (W): 0.0960271
 wordline_power (W): 0.0246049
 bitline_power (W): 0.30609
 senseamp_power (W): 0.1536
 tagarray_power (W): 0.317228
 icache_selector_power (W): 9.80408e-05
Itlb_power (W): 0.0748965 (0.11%)
 itlb_selector_power (W): 0.000520306
Data Cache Power Consumption: 1.95283  (2.86%)
 decode_power (W): 0.193844
 wordline_power (W): 0.0493624
 bitline_power (W): 0.61218
 senseamp_power (W): 0.3072
 tagarray_power (W): 0.790143
Dtlb_power (W): 0.252031 (0.37%)
 dtlb_selector_power (W): 0.000520306
Level 2 Cache Power Consumption: 0.172619 (0.253%)
 decode_power (W): 0.0141419
 wordline_power (W): 0.005535
 bitline_power (W): 0.0765225
 senseamp_power (W): 0.0384
 tagarray_power (W): 0.0379215
 cachel2_bank_selector (W): 9.80408e-05

Starting sim_load_prog...
sim: program 0: /powerSMT/tests/little.bin/test-math
sim: program 1: /powerSMT/tests/little.bin/test-printf
Finished sim_load_prog...
```

```
Starting sim_reg_stats...
Starting power_reg_stats...
Finished sim_reg_stats...

sim: simulation started @ Thu Jul 17 11:11:12 2008, options follow:

PowerSMT: This simulator implements evaluating Power Consumption in SMT
Architectures
Developed using:
SS_SMT: This simulator implements a SMT processor architecture that
has many pipelines totally independents. Each pipeline is a very detailed
out-of-order issue superscalar processor with a two-level memory system
and speculative execution support. This simulator is a performance simulator
and was developed from SimpleScalar 3.0 Tool.
Sim-wattch 1.02 (Power Model modified)
CACTI 4.0

# -config                     # load configuration from a file
# -dumpconfig                 # dump configuration to a file
# -h                    false # print help message
# -v                    false # verbose operation
# -d                    false # enable debug message
# -Dinit                false # start in Dlite debugger
-seed                       1 # random number generator seed (0 for timer
seed)
# -q                    false # initialize and terminate immediately
# -chkpt               <null> # restore EIO trace execution from <fname>
# -redir:sim           <null> # redirect simulator output to file (non-
interactive only)
# -redir:prog          <null> # redirect simulated program output to file
-nice                       0 # simulator scheduling priority
-fu:type               hetero # type of functional units: <hetero|homo:n|
compact>
-max:inst                   0 # maximum number of inst's to execute
-max:cycles                 0 # maximum number of cycles to execute
-imodules:num               2 # number of i-cache modules
-il1banks:num               1 # number of i-cache internal banks
-dl1banks:num               1 # number of d-cache banks
-ul2banks:num               1 # number of unified l2-cache banks
-decode:depth               4 # depth for decode
-fastfwd                    0 # number of insts skipped before timing starts
# -ptrace              <null> # generate pipetrace, i.e., <fname|stdout|
stderr> <range>
-fetch:ifqsize              4 # instruction fetch queue size (in insts)
-fetch:mplat                3 # extra branch mis-prediction latency
-fetch:speed                1 # speed of front-end of machine relative to
execution core
-bpred:type             bimod # branch predictor type <nottaken|taken|
perfect|bimod|2lev|comb|forced:value>}
-bpred:bimod             2048 # bimodal predictor config (<table size>)
-bpred:2lev        1 1024 8 0 # 2-level predictor config (<l1size> <l2size>
<hist_size> <xor>)
-bpred:comb              1024 # combining predictor config
(<meta_table_size>)
-bpred:ras                  8 # return address stack size (0 for no return
stack)
-bpred:btb               512 4 # BTB config (<num_sets> <associativity>)
# -bpred:spec_update   <null> # speculative predictors update in {ID|WB}
(default non-spec)
```

```
-fetch:width                0 # instruction fetch Width (insts/cycle)
-decode:width               4 # instruction decode B/W (insts/cycle)
-issue:width                4 # instruction issue B/W (insts/cycle)
-issue:inorder          false # run pipeline with in-order issue
-issue:wrongpath         true # issue instructions down wrong execution
paths
-commit:width               4 # instruction commit B/W (insts/cycle)
-ruu:size                  16 # register update unit (RUU) size
-lsq:size                   8 # load/store queue (LSQ) size
-ruulsq:type           shared # it defines <distributed> or <shared> ruu/lsq
for the slots
-cache:dl1    dl1:1:16:32:4:l # l1 data cache config, i.e., {<config>|none}
-cache:dl1lat               1 # l1 data cache hit latency (in cycles)
-cache:dl2   ul2:1:256:64:4:l # l2 data cache config, i.e., {<config>|none}
-cache:dl2lat               6 # l2 data cache hit latency (in cycles)
-cache:il1    il1:1:16:32:1:l # l1 inst cache config, i.e., {<config>|none}
-cache:il1lat               1 # l1 instruction cache hit latency (in cycles)
-cache:il2                dl2 # l2 instruction cache config, i.e.,
{<config>|dl2|none}
-cache:il2lat               6 # l2 instruction cache hit latency (in cycles)
-l1bus:opt                  1 # l1 bus: <0>-individual for dl1 & il1 <1>-
shared for them
-cache:flush            false # flush caches on system calls
-cache:icompress        false # convert 64-bit inst addresses to 32-bit inst
equivalents
-mem:lat                 18 2 # memory access latency (<first_chunk>
<inter_chunk>)
-mem:width                  8 # memory access bus width (in bytes)
-tlb:itlb   itlb:256:4096:4:l # instruction TLB config, i.e., {<config>|
none}
-tlb:dtlb   dtlb:512:4096:4:l # data TLB config, i.e., {<config>|none}
-tlb:lat                   30 # inst/data TLB miss latency (in cycles)
-res:ialu                   4 # total number of integer ALU's available
(valid when -fu:type is "hetero" or "compact")
-res:imult                  1 # total number of integer multiplier/dividers
available (valid when fu:type is "hetero")
-res:memport                2 # total number of memory system ports
available (to CPU) (valid when fu:type is "hetero" or "compact")
-res:fpalu                  4 # total number of floating point ALU's
available (valid when fu:type is "hetero" or "compact")
-res:fpmult                 1 # total number of floating point
multiplier/dividers available (valid when fu:type is "hetero")
-res:homo                   4 # total number of homogeneous functional units
available (valid when fu:type is "homo")
-res:divmult                1 # total number of div/mult functional unit
available (valid when fu:type is "compact")
# -pcstat              <null> # profile stat(s) against text addr's (mult
uses ok)
-bugcompat              false # operate in backward-compatible bugs mode
(for testing only)
-cache:il1_output_width    64 # instruction cache level 1 data output width
for CACTI
-cache:dl1_output_width    64 # data cache level 1 data output width for
CACTI
-cache:dl2_output_width    64 # data cache level 2 data output width for
CACTI
-technology          TECH_070 # technology length (nm) i.e., {TECH_800,
TECH_400, TECH_350, TECH_250, TECH_180, TECH_130, TECH_100, TECH_070, |none}
```

```
  Atention: The -fu:type parameter indicates which -res:{parameter} are
valids according the Functional Units organization:
             *hetero: -res:ialu, -res:memport, -res:fpalu, -res:imult,
-res:fpmult.
             *compact: -res:ialu, -res:memport, -res:fpalu, -res:divmult.
             *homo: -res:homo.

  Pipetrace range arguments are formatted as follows:

    {{@|#}<start>}:{{@|#|+}<end>}

  Both ends of the range are optional, if neither are specified, the entire
  execution is traced.  Ranges that start with a `@' designate an address
  range to be traced, those that start with an `#' designate a cycle count
  range.  All other range values represent an instruction count range.  The
  second argument, if specified with a `+', indicates a value relative
  to the first argument, e.g., 1000:+100 == 1000:1100.  Program symbols may
  be used in all contexts.

    Examples:   -ptrace FOO.trc #0:#1000
                -ptrace BAR.trc @2000:
                -ptrace BLAH.trc :1500
                -ptrace UXXE.trc :
                -ptrace FOOBAR.trc @main:+278

  Branch predictor configuration examples for 2-level predictor:
    Configurations:   N, M, W, X
      N   # entries in first level (# of shift register(s))
      W   width of shift register(s)
      M   # entries in 2nd level (# of counters, or other FSM)
      X   (yes-1/no-0) xor history and address for 2nd level index
    Sample predictors:
      GAg     : 1, W, 2^W, 0
      GAp     : 1, W, M (M > 2^W), 0
      PAg     : N, W, 2^W, 0
      PAp     : N, W, M (M == 2^(N+W)), 0
      gshare  : 1, W, 2^W, 1
  Predictor `comb' combines a bimodal and a 2-level predictor.

  The cache config parameter <config> has the following format:

    <name>:<nbanks>:<csize>:<bsize>:<assoc>:<repl>

    <name>   - name of the cache being defined
    <nbanks> - number of banks of this cache
    <csize>  - size of each bank in K bytes
    <bsize>  - block size of each bank
    <assoc>  - associativity of each bank
    <repl>   - block replacement strategy, 'l'-LRU, 'f'-FIFO, 'r'-random

    Example:   -cache:dl1 dl1:1:64:32:1:l

    The number of sets will be like:
             (<csize>*1024)/(<bsize>*<assoc>)
    The total size will be: <nbanks>*<csize>

    CAUTION! for tlb caches don't use <nbanks>!!!

    Example:   -dtlb dtlb:16:2048:2:r
```

```
  Second level cache can be unified by pointing a second level of the
  instruction cache to second level data cache using the "dl2" cache
  configuration arguments.  Most sensible combinations are supported, e.g.,

    A unified l2 cache (il2 is pointed at dl2):
      -cache:il1 il1:1:16:64:1:l -cache:il2 dl2
      -cache:dl1 dl1:1:16:32:1:l -cache:dl2 ul2:1:256:64:2:l

Starting sim_main ....
il1_banks_num = 1 dl1_banks_num = 1 l2_banks_num = 1
IL1WIDTH = 1 IL2WIDTH = 2  DL1WIDTH = 2 DL2WIDTH = 1

SS_SMT: 0: ** TEST DONE: starting performance simulation **
SS_SMT: 1: ** TEST DONE: starting performance simulation **
```

Quadro 3: Exemplo da saída gerada pelo PowerSMT

```
sim: ** simulation statistics **

sim_elapsed_time                   1 # total simulation time in seconds
exited_slot                        0 # Slot that has exited firstly
max_delayed_slot                   1 # slot that had the maximum delay
max_slot_delay                    32 # maximal delay for a slot
max_delayed_bank                   1 # bank that had the maximum delay
max_bank_delay                     2 # maximal delay for a i cache bank
max_delayed_bank_cycle          7559 # cycle which has the maximal delay for
a i cache bank
sim_num_insn_00               189848 # total number of instructions
committed
sim_num_insn_01               192221 # total number of instructions
committed sim_num_refs_00       47660 # total number of loads and stores
committed
sim_num_refs_01                47289 # total number of loads and stores
committed
sim_num_loads_00               27132 # total number of loads committed
sim_num_loads_01               27056 # total number of loads committed
sim_num_stores_00         20528.0000 # total number of stores committed
sim_num_stores_01         20233.0000 # total number of stores committed
sim_num_branches_00            32274 # total number of branches committed
sim_num_branches_01            35111 # total number of branches committed
sim_num_good_pred_00           28483 # total number of branches committed
and predicted correctly
sim_num_good_pred_01           31259 # total number of branches committed
and predicted correctly
sim_inst_rate_00          189848.0000 # simulation speed (in insts/sec)
sim_inst_rate_01          192221.0000 # simulation speed (in insts/sec)
sim_total_insn_00              203737 # total number of instructions executed
sim_total_insn_01              208420 # total number of instructions executed
prev_sim_total_insn_00         203737 # total prev number of instructions
executed
prev_sim_total_insn_01         208420 # total prev number of instructions
executed
sim_total_nops_insn_00            967 # total number of NOPs instructions in
the IFQ
sim_total_nops_insn_01            849 # total number of NOPs instructions in
the IFQ
sim_total_refs_00              51234 # total number of loads and stores
```

```
executed
sim_total_refs_01              51250 # total number of loads and stores
executed
sim_total_loads_00             29571 # total number of loads executed
sim_total_loads_01             29886 # total number of loads executed
sim_total_stores_00       21663.0000 # total number of stores executed
sim_total_stores_01       21364.0000 # total number of stores executed
sim_total_branches_00          35256 # total number of branches executed
sim_total_branches_01          38510 # total number of branches executed
sim_cycle                     235305 # total simulation time in cycles
sim_IPC_00                    0.8068 # instructions per cycle
sim_IPC_01                    0.8169 # instructions per cycle
total_IPC                     1.6237 # Total IPC
sim_CPI_00                    1.2394 # cycles per instruction
sim_CPI_01                    1.2241 # cycles per instruction
sim_exec_BW_00                0.8658 # total instructions (mis-spec +
committed) per cycle
sim_exec_BW_01                0.8857 # total instructions (mis-spec +
committed) per cycle
sim_IPB_00                    5.8824 # instruction per branch
sim_IPB_01                    5.4747 # instruction per branch
sim_goodpred_00               0.8825 # good prediction rate
sim_goodpred_01               0.8903 # good prediction rate
IFQ_count_00                  480042 # cumulative IFQ occupancy
IFQ_count_01                  490752 # cumulative IFQ occupancy
IFQ_fcount_00                 104909 # cumulative IFQ full count
IFQ_fcount_01                 106845 # cumulative IFQ full count
ifq_occupancy_00              2.0401 # avg IFQ occupancy (insn's)
ifq_occupancy_01              2.0856 # avg IFQ occupancy (insn's)
ifq_occupancy_global_avg      2.0628 # global avg of IFQ Occupancy (insn's)
ifq_rate_00                   0.8658 # avg IFQ dispatch rate (insn/cycle)
ifq_rate_01                   0.8857 # avg IFQ dispatch rate (insn/cycle)
ifq_latency_00                2.3562 # avg IFQ occupant latency (cycle's)
ifq_latency_01                2.3546 # avg IFQ occupant latency (cycle's)
ifq_full_00                   0.4458 # fraction of time (cycle's) IFQ was
full
ifq_full_01                   0.4541 # fraction of time (cycle's) IFQ was
full
RUU_count_00                 1234861 # cumulative RUU occupancy
RUU_count_01                 1366783 # cumulative RUU occupancy
RUU_fcount_00                  14371 # cumulative RUU full count
RUU_fcount_01                  18015 # cumulative RUU full count
ruu_occupancy_00              5.2479 # avg RUU occupancy (insn's)
ruu_occupancy_01              5.8086 # avg RUU occupancy (insn's)
ruu_occupancy_global_avg      5.5282 # global avg of RUU Occupancy (insn's)
ruu_rate_00                   0.8658 # avg RUU dispatch rate (insn/cycle)
ruu_rate_01                   0.8857 # avg RUU dispatch rate (insn/cycle)
ruu_latency_00                6.0611 # avg RUU occupant latency (cycle's)
ruu_latency_01                6.5578 # avg RUU occupant latency (cycle's)
ruu_full_00                   0.0611 # fraction of time (cycle's) RUU was
full
ruu_full_01                   0.0766 # fraction of time (cycle's) RUU was
full
LSQ_count_00                  254893 # cumulative LSQ occupancy
LSQ_count_01                  257466 # cumulative LSQ occupancy
LSQ_fcount_00                   1593 # cumulative LSQ full count
LSQ_fcount_01                   2301 # cumulative LSQ full count
lsq_occupancy_00              1.0832 # avg LSQ occupancy (insn's)
lsq_occupancy_01              1.0942 # avg LSQ occupancy (insn's)
```

```
lsq_occupancy_global_avg      1.0887 # global avg of LSQ Occupancy (insn's)
lsq_rate_00                   0.8658 # avg LSQ dispatch rate (insn/cycle)
lsq_rate_01                   0.8857 # avg LSQ dispatch rate (insn/cycle)
lsq_latency_00                1.2511 # avg LSQ occupant latency (cycle's)
lsq_latency_01                1.2353 # avg LSQ occupant latency (cycle's)
lsq_full_00                   0.0068 # fraction of time (cycle's) LSQ was
full
lsq_full_01                   0.0098 # fraction of time (cycle's) LSQ was
full
bpred_bimod.lookups_00         36937 # total number of bpred lookups
bpred_bimod.lookups_01         40482 # total number of bpred lookups
bpred_bimod.updates_00         32274 # total number of updates
bpred_bimod.updates_01         35111 # total number of updates
bpred_bimod.addr_hits_00       28483 # total number of address-predicted
hits
bpred_bimod.addr_hits_01       31259 # total number of address-predicted
hits
bpred_bimod.dir_hits_00        29064 # total number of direction-predicted
hits (includes addr-hits)
bpred_bimod.dir_hits_01        31827 # total number of direction-predicted
hits (includes addr-hits)
bpred_bimod.misses_00           3210 # total number of misses
bpred_bimod.misses_01           3284 # total number of misses
bpred_bimod.jr_hits_00          3017 # total number of address-predicted
hits for JR's
bpred_bimod.jr_hits_01          2752 # total number of address-predicted
hits for JR's
bpred_bimod.jr_seen_00          3140 # total number of JR's seen
bpred_bimod.jr_seen_01          2894 # total number of JR's seen
bpred_bimod.jr_non_ras_hits.PP_00 28 # total number of address-predicted
hits for non-RAS JR's
bpred_bimod.jr_non_ras_hits.PP_01 60 # total number of address-predicted
hits for non-RAS JR's
bpred_bimod.jr_non_ras_seen.PP_00 41 # total number of non-RAS JR's seen
bpred_bimod.jr_non_ras_seen.PP_01 92 # total number of non-RAS JR's seen
bpred_bimod.bpred_addr_rate_00 0.8825 # branch address-prediction rate
(i.e., addr-hits/updates)
bpred_bimod.bpred_addr_rate_01 0.8903 # branch address-prediction rate
(i.e., addr-hits/updates)
bpred_bimod.bpred_dir_rate_00 0.9005 # branch direction-prediction rate
(i.e., all-hits/updates)
bpred_bimod.bpred_dir_rate_01 0.9065 # branch direction-prediction rate
(i.e., all-hits/updates)
bpred_bimod.bpred_jr_rate_00  0.9608 # JR address-prediction rate (i.e., JR
addr-hits/JRs seen)
bpred_bimod.bpred_jr_rate_01  0.9509 # JR address-prediction rate (i.e., JR
addr-hits/JRs seen)
bpred_bimod.bpred_jr_non_ras_rate.PP_00 0.6829 # non-RAS JR addr-pred rate
(ie, non-RAS JR hits/JRs seen)
bpred_bimod.bpred_jr_non_ras_rate.PP_01 0.6522 # non-RAS JR addr-pred rate
(ie, non-RAS JR hits/JRs seen)
bpred_bimod.retstack_pushes_00  3384 # total number of address pushed onto
ret-addr stack
bpred_bimod.retstack_pushes_01  3090 # total number of address pushed onto
ret-addr stack
bpred_bimod.retstack_pops_00    4081 # total number of address popped off of
ret-addr stack
bpred_bimod.retstack_pops_01    3458 # total number of address popped off of
ret-addr stack
```

```
bpred_bimod.used_ras.PP_00      3099 # total number of RAS predictions used
bpred_bimod.used_ras.PP_01      2802 # total number of RAS predictions used
bpred_bimod.ras_hits.PP_00      2989 # total number of RAS hits
bpred_bimod.ras_hits.PP_01      2692 # total number of RAS hits
bpred_bimod.ras_rate.PP_00    0.9645 # RAS prediction rate (i.e., RAS
hits/used RAS)
bpred_bimod.ras_rate.PP_01    0.9607 # RAS prediction rate (i.e., RAS
hits/used RAS)
il1_M00_B00.accesses          225674 # total number of accesses
il1_M00_B00.hits              211693 # total number of hits
il1_M00_B00.misses             13981 # total number of misses
il1_M00_B00.replacements       13471 # total number of replacements
il1_M00_B00.writebacks             0 # total number of writebacks
il1_M00_B00.invalidations          0 # total number of invalidations
il1_M00_B00.miss_rate         0.0620 # miss rate (i.e., misses/ref)
il1_M00_B00.repl_rate         0.0597 # replacement rate (i.e., repls/ref)
il1_M00_B00.wb_rate           0.0000 # writeback rate (i.e., wrbks/ref)
il1_M00_B00.inv_rate          0.0000 # invalidation rate (i.e., invs/ref)
il1_M01_B00.accesses          230635 # total number of accesses
il1_M01_B00.hits              216499 # total number of hits
il1_M01_B00.misses             14136 # total number of misses
il1_M01_B00.replacements       13628 # total number of replacements
il1_M01_B00.writebacks             0 # total number of writebacks
il1_M01_B00.invalidations          0 # total number of invalidations
il1_M01_B00.miss_rate         0.0613 # miss rate (i.e., misses/ref)
il1_M01_B00.repl_rate         0.0591 # replacement rate (i.e., repls/ref)
il1_M01_B00.wb_rate           0.0000 # writeback rate (i.e., wrbks/ref)
il1_M01_B00.inv_rate          0.0000 # invalidation rate (i.e., invs/ref)
ul2_00.accesses                29939 # total number of accesses
ul2_00.hits                    27587 # total number of hits
ul2_00.misses                   2352 # total number of misses
ul2_00.replacements               34 # total number of replacements
ul2_00.writebacks                 10 # total number of writebacks
ul2_00.invalidations               0 # total number of invalidations
ul2_00.miss_rate              0.0786 # miss rate (i.e., misses/ref)
ul2_00.repl_rate              0.0011 # replacement rate (i.e., repls/ref)
ul2_00.wb_rate                0.0003 # writeback rate (i.e., wrbks/ref)
ul2_00.inv_rate               0.0000 # invalidation rate (i.e., invs/ref)
dl1_00.accesses                95025 # total number of accesses
dl1_00.hits                    93839 # total number of hits
dl1_00.misses                   1186 # total number of misses
dl1_00.replacements              674 # total number of replacements
dl1_00.writebacks                636 # total number of writebacks
dl1_00.invalidations               0 # total number of invalidations
dl1_00.miss_rate              0.0125 # miss rate (i.e., misses/ref)
dl1_00.repl_rate              0.0071 # replacement rate (i.e., repls/ref)
dl1_00.wb_rate                0.0067 # writeback rate (i.e., wrbks/ref)
dl1_00.inv_rate               0.0000 # invalidation rate (i.e., invs/ref)
itlb_00.accesses_00           225674 # total number of accesses
itlb_00.accesses_01           230635 # total number of accesses
itlb_00.hits_00               225651 # total number of hits
itlb_00.hits_01               230616 # total number of hits
itlb_00.misses_00                 23 # total number of misses
itlb_00.misses_01                 19 # total number of misses
itlb_00.replacements_00            0 # total number of replacements
itlb_00.replacements_01            0 # total number of replacements
itlb_00.writebacks_00              0 # total number of writebacks
itlb_00.writebacks_01              0 # total number of writebacks
itlb_00.invalidations_00           0 # total number of invalidations
```

```
itlb_00.invalidations_01           0 # total number of invalidations
itlb_00.miss_rate_00          0.0001 # miss rate (i.e., misses/ref)
itlb_00.miss_rate_01          0.0001 # miss rate (i.e., misses/ref)
itlb_00.repl_rate_00          0.0000 # replacement rate (i.e., repls/ref)
itlb_00.repl_rate_01          0.0000 # replacement rate (i.e., repls/ref)
itlb_00.wb_rate_00            0.0000 # writeback rate (i.e., wrbks/ref)
itlb_00.wb_rate_01            0.0000 # writeback rate (i.e., wrbks/ref)
itlb_00.inv_rate_00           0.0000 # invalidation rate (i.e., invs/ref)
itlb_00.inv_rate_01           0.0000 # invalidation rate (i.e., invs/ref)
dtlb_00.accesses_00            47988 # total number of accesses
dtlb_00.accesses_01            47870 # total number of accesses
dtlb_00.hits_00                47978 # total number of hits
dtlb_00.hits_01                47861 # total number of hits
dtlb_00.misses_00                 10 # total number of misses
dtlb_00.misses_01                  9 # total number of misses
dtlb_00.replacements_00            0 # total number of replacements
dtlb_00.replacements_01            0 # total number of replacements
dtlb_00.writebacks_00              0 # total number of writebacks
dtlb_00.writebacks_01              0 # total number of writebacks
dtlb_00.invalidations_00           0 # total number of invalidations
dtlb_00.invalidations_01           0 # total number of invalidations
dtlb_00.miss_rate_00          0.0002 # miss rate (i.e., misses/ref)
dtlb_00.miss_rate_01          0.0002 # miss rate (i.e., misses/ref)
dtlb_00.repl_rate_00          0.0000 # replacement rate (i.e., repls/ref)
dtlb_00.repl_rate_01          0.0000 # replacement rate (i.e., repls/ref)
dtlb_00.wb_rate_00            0.0000 # writeback rate (i.e., wrbks/ref)
dtlb_00.wb_rate_01            0.0000 # writeback rate (i.e., wrbks/ref)
dtlb_00.inv_rate_00           0.0000 # invalidation rate (i.e., invs/ref)
dtlb_00.inv_rate_01           0.0000 # invalidation rate (i.e., invs/ref)
rename_power_00           17865.6810 # total power usage of rename unit
rename_power_01           17865.6810 # total power usage of rename unit
rename_power_sum          35731.3620 # global sum of rename_power
bpred_power_00           139257.1954 # total power usage of bpred unit
bpred_power_01           139257.1954 # total power usage of bpred unit
bpred_power_sum          278514.3909 # global sum of bpred_power
window_power_00           71757.8807 # total power usage of instruction
window
window_power_01           71757.8807 # total power usage of instruction
window
window_power_sum         143515.7614 # global sum of window_power
lsq_power_00              42069.2201 # total power usage of load/store queue
lsq_power_01              42069.2201 # total power usage of load/store queue
lsq_power_sum             84138.4402 # global sum of lsq_power
regfile_power_00         109409.1409 # total power usage of arch. regfile
regfile_power_01         109409.1409 # total power usage of arch. regfile
regfile_power_sum        218818.2818 # global sum of regfile_power
ifq_power_00               9616.5148 # total power usage of instruction
fetch queue
ifq_power_01               9616.5148 # total power usage of instruction
fetch queue
ifq_power_sum             19233.0296 # global sum of ifq_power
reorder_power_0           34581.8119 # total power usage of reorder buffer
reorder_power_0           34581.8119 # total power usage of reorder buffer
reorder_power_sum         69163.6239 # global sum of reorder_power
icache_power_00          228844.5007 # total power usage of icache
icache_power_01          228844.5007 # total power usage of icache
icache_power_sum         457689.0013 # global sum of icache_power
dcache_power_00          518814.1006 # total power usage of dcache
dcache_power_01          518814.1006 # total power usage of dcache
```

```
dcache_power_sum        1037628.2012 # global sum of dcache_power
dcache2_power_00          40618.0932 # total power usage of dcache2
dcache2_power_01          40618.0932 # total power usage of dcache2
dcache2_power_sum         81236.1864 # global sum of dcache2_power
ialu_power_00            728403.6317 # total power usage of alu
ialu_power_01            728403.6317 # total power usage of alu
ialu_power_sum          1456807.2634 # global sum of ialu_power
imult_div_power_00       140328.6566 # total power usage of imult_div FU
imult_div_power_01       140328.6566 # total power usage of imult_div FU
imult_div_power_sum      280657.3133 # global sum of imult_div_power
mem_port_power_00             0.0000 # total power usage of memory port FU
mem_port_power_01             0.0000 # total power usage of memory port FU
mem_port_power_sum            0.0000 # global sum of mem_port_power
fpalu_power_00          2232204.6778 # total power usage of fpalu
fpalu_power_01          2232204.6778 # total power usage of fpalu
fpalu_power_sum         4464409.3555 # global sum of fpalu_power
fpmult_div_power_00      558051.1694 # total power usage of fpmult_div
fpmult_div_power_01      558051.1694 # total power usage of fpmult_div
fpmult_div_power_sum    1116102.3389 # global sum of fpmult_div_power
divmult_power_00              0.0000 # total power usage of divmult
divmult_power_01              0.0000 # total power usage of divmult
divmult_power_sum             0.0000 # global sum of divmult_power
homo_power_00                 0.0000 # total power usage of homo FU
homo_power_01                 0.0000 # total power usage of homo FU
homo_power_sum                0.0000 # global sum of homo_power
resultbus_power_00       399895.9138 # total power usage of resultbus
resultbus_power_01       399895.9138 # total power usage of resultbus
resultbus_power_sum      799791.8276 # global sum of resultbus_power
clock_power_00         10802592.6555 # total power usage of clock
clock_power_01         10802592.6555 # total power usage of clock
clock_power_sum        21605185.3109 # global sum of clock_power
avg_rename_power_00           0.0759 # avg power usage of rename unit
avg_rename_power_01           0.0759 # avg power usage of rename unit
avg_rename_power_sum          0.1519 # global sum of avg_rename_power
avg_bpred_power_00            0.5918 # avg power usage of bpred unit
avg_bpred_power_01            0.5918 # avg power usage of bpred unit
avg_bpred_power_sum           1.1836 # global sum of avg_bpred_power
avg_window_power_00           0.3050 # avg power usage of instruction window
avg_window_power_01           0.3050 # avg power usage of instruction window
avg_window_power_sum          0.6099 # global sum of avg_window_power
avg_lsq_power_00              0.1788 # avg power usage of lsq
avg_lsq_power_01              0.1788 # avg power usage of lsq
avg_lsq_power_sum             0.3576 # global sum of avg_lsq_power
avg_regfile_power_00          0.4650 # avg power usage of arch. regfile
avg_regfile_power_01          0.4650 # avg power usage of arch. regfile
avg_regfile_power_sum         0.9299 # global sum of avg_regfile_power
avg_ifq_power_00              0.0409 # avg power usage of instruction fetch
queue
avg_ifq_power_01              0.0409 # avg power usage of instruction fetch
queue
avg_ifq_power_sum             0.0817 # global sum of avg_ifq_power
avg_reorder_power_00          0.1470 # avg power usage of reorder buffer
avg_reorder_power_01          0.1470 # avg power usage of reorder buffer
avg_reorder_power_sum         0.2939 # global sum of avg_reorder_power
avg_icache_power_00           0.9725 # avg power usage of icache
avg_icache_power_01           0.9725 # avg power usage of icache
avg_icache_power_sum          1.9451 # global sum of avg_icache_power
avg_dcache_power_00           2.2049 # avg power usage of dcache
avg_dcache_power_01           2.2049 # avg power usage of dcache
```

```
avg_dcache_power_sum          4.4097 # global sum of avg_dcache_power
avg_dcache2_power_00          0.1726 # avg power usage of dcache2
avg_dcache2_power_01          0.1726 # avg power usage of dcache2
avg_dcache2_power_sum         0.3452 # global sum of avg_dcache2_power
avg_ialu_power_00             3.0956 # avg power usage of ialu FU
avg_ialu_power_01             3.0956 # avg power usage of ialu FU
avg_ialu_power_sum            6.1911 # global sum of avg_ialu_power
avg_fpalu_power_00            9.4864 # avg power usage of fpalu FU
avg_fpalu_power_01            9.4864 # avg power usage of fpalu FU
avg_fpalu_power_sum          18.9729 # global sum of avg_fpalu_power
avg_imult_div_power_00        0.5964 # avg power usage of imult_div FU
avg_imult_div_power_01        0.5964 # avg power usage of imult_div FU
avg_imult_div_power_sum       1.1927 # global sum of avg_imult_div_power
avg_mem_port_power_00         0.0000 # avg power usage of mem_port FU
avg_mem_port_power_01         0.0000 # avg power usage of mem_port FU
avg_mem_port_power_sum        0.0000 # global sum of avg_mem_port_power
avg_fpmult_div_power_00       2.3716 # avg power usage of fpmult_div FU
avg_fpmult_div_power_01       2.3716 # avg power usage of fpmult_div FU
avg_fpmult_div_power_sum      4.7432 # global sum of avg_fpmult_div_power
avg_divmult_power_00          0.0000 # avg power usage of divmult FU
avg_divmult_power_01          0.0000 # avg power usage of divmult FU
avg_divmult_power_sum         0.0000 # global sum of avg_divmult_power
avg_homo_power_00             0.0000 # avg power usage of homo FU
avg_homo_power_01             0.0000 # avg power usage of homo FU
avg_homo_power_sum            0.0000 # global sum of avg_homo_power
avg_resultbus_power_00        1.6995 # avg power usage of resultbus
avg_resultbus_power_01        1.6995 # avg power usage of resultbus
avg_resultbus_power_sum       3.3990 # global sum of avg_resultbus_power
avg_clock_power_00           45.9089 # avg power usage of clock
avg_clock_power_01           45.9089 # avg power usage of clock
avg_clock_power_sum          91.8178 # global sum of avg_clock_power
fetch_stage_power_00     368101.6961 # total power usage of fetch stage
fetch_stage_power_01     368101.6961 # total power usage of fetch stage
fetch_stage_power_sum    736203.3922 # global sum of fetch_stage_power
dispatch_stage_power_00   17865.6810 # total power usage of dispatch stage
dispatch_stage_power_01   17865.6810 # total power usage of dispatch stage
dispatch_stage_power_sum  35731.3620 # global sum of dispatch_stage_power
issue_stage_power_00    4732143.3440 # total power usage of issue stage
issue_stage_power_01    4732143.3440 # total power usage of issue stage
issue_stage_power_sum   9464286.6879 # global sum of issue_stage_power
avg_fetch_power_00            1.5644 # average power of fetch unit per cycle
avg_fetch_power_01            1.5644 # average power of fetch unit per cycle
avg_fetch_power_sum           3.1287 # global sum of avg_fetch_power
avg_dispatch_power_00         0.0759 # average power of dispatch unit per
cycle
avg_dispatch_power_01         0.0759 # average power of dispatch unit per
cycle
avg_dispatch_power_sum        0.1519 # global sum of avg_dispatch_power
avg_issue_power_00           20.1107 # average power of issue unit per cycle
avg_issue_power_01           20.1107 # average power of issue unit per cycle
avg_issue_power_sum          40.2214 # global sum of avg_issue_power
total_power_00          16074310.8441 # total power per cycle
total_power_01          16074310.8441 # total power per cycle
total_power_sum         32148621.6883 # global sum of total_power
avg_total_power_cycle_00     68.3127 # average total power per cycle
avg_total_power_cycle_01     68.3127 # average total power per cycle
avg_total_power_cycle_sum   136.6253 # global sum of avg_total_power_cycle
avg_total_power_cycle_nofp_nod2_00 49.1672 # average total power per cycle
avg_total_power_cycle_nofp_nod2_01 49.1672 # average total power per cycle
```

```
avg_total_power_cycle_nofp_nod2_sum 98.3344 # global sum of
avg_total_power_cycle_nofp_nod2
avg_total_power_insn_00      78.8974 # average total power per insn
avg_total_power_insn_01      77.1246 # average total power per insn
avg_total_power_insn_sum    156.0220 # global sum of
avg_total_power_insn_power
avg_total_power_insn_nofp_nod2_00 56.7854 # average total power per insn
avg_total_power_insn_nofp_nod2_01 55.5095 # average total power per insn
avg_total_power_insn_nofp_nod2_sum 112.2949 # global sum of
avg_total_power_insn_nofp_nod2 rename_power_cc1_00 7277.3960 # total power
usage of rename unit_cc1
rename_power_cc1_01        7417.0232 # total power usage of rename unit_cc1
rename_power_cc1_sum      14694.4192 # global sum of rename_power_cc1
bpred_power_cc1_00        17516.8573 # total power usage of bpred unit_cc1
bpred_power_cc1_01        18799.0260 # total power usage of bpred unit_cc1
bpred_power_cc1_sum       36315.8833 # global sum of bpred_power_cc1
window_power_cc1_00       40398.4083 # total power usage of instruction
window_cc1
window_power_cc1_01       41333.5524 # total power usage of instruction
window_cc1
window_power_cc1_sum      81731.9607 # global sum of window_power_cc1
lsq_power_cc1_00           3559.3385 # total power usage of lsq_cc1
lsq_power_cc1_01           3549.4122 # total power usage of lsq_cc1
lsq_power_cc1_sum          7108.7507 # global sum of lsq_power_cc1
regfile_power_cc1_00      49466.4826 # total power usage of arch.
regfile_cc1
regfile_power_cc1_01      51183.8768 # total power usage of arch.
regfile_cc1
regfile_power_cc1_sum    100650.3595 # global sum of regfile_power_cc1
ifq_power_cc1_00           3789.3086 # total power usage of ifq_cc1
ifq_power_cc1_01           3856.0465 # total power usage of ifq_cc1
ifq_power_cc1_sum          7645.3551 # global sum of ifq_power_cc1
reorder_power_cc1_00      26099.6748 # total power usage of reorder_cc1
reorder_power_cc1_01      26713.4044 # total power usage of reorder_cc1
reorder_power_cc1_sum     52813.0793 # global sum of reorder_power_cc1
icache_power_cc1_00      100676.7976 # total power usage of icache_cc1
icache_power_cc1_01      102015.9909 # total power usage of icache_cc1
icache_power_cc1_sum     202692.7885 # global sum of icache_power_cc1
dcache_power_cc1_00       84740.4081 # total power usage of dcache_cc1
dcache_power_cc1_01       84985.1474 # total power usage of dcache_cc1
dcache_power_cc1_sum     169725.5555 # global sum of dcache_power_cc1
dcache2_power_cc1_00       2515.0576 # total power usage of dcache2_cc1
dcache2_power_cc1_01       2542.4177 # total power usage of dcache2_cc1
dcache2_power_cc1_sum      5057.4752 # global sum of dcache2_power_cc1
ialu_power_cc1_00        320466.0231 # total power usage of alu_cc1
ialu_power_cc1_01        325725.4004 # total power usage of alu_cc1
ialu_power_cc1_sum       646191.4235 # global sum of ialu_power_cc1
imult_div_power_cc1_00     2257.8538 # total power usage of imult_div_cc1 FU
imult_div_power_cc1_01     2295.4251 # total power usage of imult_div_cc1 FU
imult_div_power_cc1_sum    4553.2789 # global sum of imult_div_power_cc1
mem_port_power_cc1_00         0.0000 # total power usage of memory-port_cc1
FU
mem_port_power_cc1_01         0.0000 # total power usage of memory-port_cc1
FU
mem_port_power_cc1_sum        0.0000 # global sum of mem_port_power_cc1
fpalu_power_cc1_00        13148.1936 # total power usage of fpalu_cc1
fpalu_power_cc1_01         1280.6682 # total power usage of fpalu_cc1
fpalu_power_cc1_sum       14428.8618 # global sum of fpalu_power_cc1
fpmult_div_power_cc1_00    1071.9667 # total power usage of fpmult_div_cc1
```

```
fpmult_div_power_cc1_01      87.7495 # total power usage of fpmult_div_cc1
fpmult_div_power_cc1_sum   1159.7162 # global sum of fpmult_div_power_cc1
divmult_power_cc1_00          0.0000 # total power usage of divmult_cc1
divmult_power_cc1_01          0.0000 # total power usage of divmult_cc1
divmult_power_cc1_sum         0.0000 # global sum of divmult_power_cc1
homo_power_cc1_00             0.0000 # total power usage of homo_cc1 FU
homo_power_cc1_01             0.0000 # total power usage of homo_cc1 FU
homo_power_cc1_sum            0.0000 # global sum of homo_power_cc1
resultbus_power_cc1_00   162539.7060 # total power usage of resultbus_cc1
resultbus_power_cc1_01   167952.9583 # total power usage of resultbus_cc1
resultbus_power_cc1_sum  330492.6643 # global sum of resultbus_power_cc1
clock_power_cc1_00      1518028.5401 # total power usage of clock_cc1
clock_power_cc1_01      1644365.4416 # total power usage of clock_cc1
clock_power_cc1_sum     3162393.9817 # global sum of clock_power_cc1
avg_rename_power_cc1_00       0.0309 # avg power usage of rename unit_cc1
avg_rename_power_cc1_01       0.0315 # avg power usage of rename unit_cc1
avg_rename_power_cc1_sum      0.0624 # global sum of avg_rename_power_cc1
avg_bpred_power_cc1_00        0.0744 # avg power usage of bpred unit_cc1
avg_bpred_power_cc1_01        0.0799 # avg power usage of bpred unit_cc1
avg_bpred_power_cc1_sum       0.1543 # global sum of avg_bpred_power_cc1
avg_window_power_cc1_00       0.1717 # avg power usage of instruction
window_cc1
avg_window_power_cc1_01       0.1757 # avg power usage of instruction
window_cc1
avg_window_power_cc1_sum      0.3473 # global sum of avg_window_power_cc1
avg_lsq_power_cc1_00          0.0151 # avg power usage of lsq_cc1
avg_lsq_power_cc1_01          0.0151 # avg power usage of lsq_cc1
avg_lsq_power_cc1_sum         0.0302 # global sum of avg_lsq_power_cc1
avg_regfile_power_cc1_00      0.2102 # avg power usage of arch. regfile_cc1
avg_regfile_power_cc1_01      0.2175 # avg power usage of arch. regfile_cc1
avg_regfile_power_cc1_sum     0.4277 # global sum of avg_regfile_power_cc1
avg_ifq_power_cc1_00          0.0161 # avg power usage of ifq_cc1
avg_ifq_power_cc1_01          0.0164 # avg power usage of ifq_cc1
avg_ifq_power_cc1_sum         0.0325 # global sum of avg_ifq_power_cc1
avg_reorder_power_cc1_00      0.1109 # avg power usage of reorder_cc1
avg_reorder_power_cc1_01      0.1135 # avg power usage of reorder_cc1
avg_reorder_power_cc1_sum     0.2244 # global sum of avg_reorder_power_cc1
avg_icache_power_cc1_00       0.4279 # avg power usage of icache_cc1
avg_icache_power_cc1_01       0.4335 # avg power usage of icache_cc1
avg_icache_power_cc1_sum      0.8614 # global sum of avg_icache_power_cc1
avg_dcache_power_cc1_00       0.3601 # avg power usage of dcache_cc1
avg_dcache_power_cc1_01       0.3612 # avg power usage of dcache_cc1
avg_dcache_power_cc1_sum      0.7213 # global sum of avg_dcache_power_cc1
avg_dcache2_power_cc1_00      0.0107 # avg power usage of dcache2_cc1
avg_dcache2_power_cc1_01      0.0108 # avg power usage of dcache2_cc1
avg_dcache2_power_cc1_sum     0.0215 # global sum of avg_dcache2_power_cc1
avg_ialu_power_cc1_00         1.3619 # avg power usage of ialu_cc1
avg_ialu_power_cc1_01         1.3843 # avg power usage of ialu_cc1
avg_ialu_power_cc1_sum        2.7462 # global sum of avg_ialu_power_cc1
avg_fpalu_power_cc1_00        0.0559 # avg power usage of fpalu_cc1 FU
avg_fpalu_power_cc1_01        0.0054 # avg power usage of fpalu_cc1 FU
avg_fpalu_power_cc1_sum       0.0613 # global sum of avg_fpalu_power_cc1
avg_imult_div_power_cc1_00    0.0096 # avg power usage of imult_div_cc1 FU
avg_imult_div_power_cc1_01    0.0098 # avg power usage of imult_div_cc1 FU
avg_imult_div_power_cc1_sum   0.0194 # global sum of avg_imult_div_power_cc1
avg_mem_port_power_cc1_00     0.0000 # avg power usage of mem_port_cc1 FU
avg_mem_port_power_cc1_01     0.0000 # avg power usage of mem_port_cc1 FU
avg_mem_port_power_cc1_sum    0.0000 # global sum of avg_mem_port_power_cc1
avg_fpmult_div_power_cc1_00   0.0046 # avg power usage of fpmult_div_cc1 FU
```

```
avg_fpmult_div_power_cc1_01   0.0004 # avg power usage of fpmult_div_cc1 FU
avg_fpmult_div_power_cc1_sum  0.0049 # global sum of avg_fpmult_div_power
_cc1
avg_divmult_power_cc1_00      0.0000 # avg power usage of divmult_cc1 FU
avg_divmult_power_cc1_01      0.0000 # avg power usage of divmult_cc1 FU
avg_divmult_power_cc1_sum     0.0000 # global sum of avg_divmult_power_cc1
avg_homo_power_cc1_00         0.0000 # avg power usage of homo_cc1 FU
avg_homo_power_cc1_01         0.0000 # avg power usage of homo_cc1 FU
avg_homo_power_cc1_sum        0.0000 # global sum of avg_homo_power_cc1
avg_resultbus_power_cc1_00    0.6908 # avg power usage of resultbus_cc1
avg_resultbus_power_cc1_01    0.7138 # avg power usage of resultbus_cc1
avg_resultbus_power_cc1_sum   1.4045 # global sum of avg_resultbus_power_cc1
avg_clock_power_cc1_00        6.4513 # avg power usage of clock_cc1
avg_clock_power_cc1_01        6.9882 # avg power usage of clock_cc1
avg_clock_power_cc1_sum      13.4396 # global sum of avg_clock_power_cc1
fetch_stage_power_cc1_00 121982.9635 # total power usage of fetch stage_cc1
fetch_stage_power_cc1_01 124671.0634 # total power usage of fetch stage_cc1
fetch_stage_power_cc1_sum 246654.0269 # global sum of fetch_stage_power_cc1
dispatch_stage_power_cc1_00 7277.3960 # total power usage of dispatch
stage_cc1
dispatch_stage_power_cc1_01 7417.0232 # total power usage of dispatch
stage_cc1
dispatch_stage_power_cc1_sum 14694.4192 # global sum of
dispatch_stage_power_cc1
issue_stage_power_cc1_00 630696.9557 # total power usage of issue stage_cc1
issue_stage_power_cc1_01 629752.7310 # total power usage of issue stage_cc1
issue_stage_power_cc1_sum 1260449.6867 # global sum of issue_stage_power_cc1
avg_fetch_power_cc1_00        0.5184 # average power of fetch unit per
cycle_cc1
avg_fetch_power_cc1_01        0.5298 # average power of fetch unit per
cycle_cc1
avg_fetch_power_cc1_sum       1.0482 # global sum of avg_fetch_power_cc1
avg_dispatch_power_cc1_00     0.0309 # average power of dispatch unit per
cycle_cc1
avg_dispatch_power_cc1_01     0.0315 # average power of dispatch unit per
cycle_cc1
avg_dispatch_power_cc1_sum    0.0624 # global sum of avg_dispatch_power_cc1
avg_issue_power_cc1_00       16.7984 # average power of issue unit per
cycle_cc1
avg_issue_power_cc1_01       16.8265 # average power of issue unit per
cycle_cc1
avg_issue_power_cc1_sum      33.6248 # global sum of avg_issue_power_cc1
total_power_cycle_cc1_00 5649496.4363 # total power per cycle_cc1
total_power_cycle_cc1_01 5786989.0284 # total power per cycle_cc1
total_power_cycle_cc1_sum 11436485.4647 # global sum of total_power_cycle
_cc1
avg_total_power_cycle_cc1_00 24.1202 # average total power per cycle_cc1
avg_total_power_cycle_cc1_01 24.7071 # average total power per cycle_cc1
avg_total_power_cycle_cc1_sum 48.8273 # global sum of avg_total_power_cycle
_cc1
avg_total_power_insn_cc1_00  27.8575 # average total power per insn_cc1
avg_total_power_insn_cc1_01  27.8942 # average total power per insn_cc1
avg_total_power_insn_cc1_sum 55.7516 # global sum of avg_total_power_insn
_cc1
rename_power_cc2_00        3859.9832 # total power usage of rename unit_cc2
rename_power_cc2_01        3947.3926 # total power usage of rename unit_cc2
rename_power_cc2_sum       7807.3759 # global sum of rename_power_cc2
bpred_power_cc2_00         9550.1301 # total power usage of bpred unit_cc2
bpred_power_cc2_01        10389.6207 # total power usage of bpred unit_cc2
```

```
bpred_power_cc2_sum       19939.7508 # global sum of bpred_power_cc2
window_power_cc2_00       20173.2452 # total power usage of instruction
window_cc2
window_power_cc2_01       20830.7306 # total power usage of instruction
window_cc2
window_power_cc2_sum      41003.9759 # global sum of window_power_cc2
lsq_power_cc2_00           2113.2169 # total power usage of lsq_cc2
lsq_power_cc2_01           2125.2497 # total power usage of lsq_cc2
lsq_power_cc2_sum          4238.4666 # global sum of lsq_power_cc2
regfile_power_cc2_00       9723.0807 # total power usage of arch.
regfile_cc2
regfile_power_cc2_01       9843.1876 # total power usage of arch.
regfile_cc2
regfile_power_cc2_sum     19566.2683 # global sum of regfile_power_cc2
ifq_power_cc2_00            724.1352 # total power usage of ifq_cc2
ifq_power_cc2_01            740.0057 # total power usage of ifq_cc2
ifq_power_cc2_sum          1464.1408 # global sum of ifq_power_cc2
reorder_power_cc2_00       7244.0351 # total power usage of reorder_cc2
reorder_power_cc2_01       7365.7474 # total power usage of reorder_cc2
reorder_power_cc2_sum     14609.7825 # global sum of reorder_power_cc2
icache_power_cc2_00      100676.7976 # total power usage of icache_cc2
icache_power_cc2_01      102015.9909 # total power usage of icache_cc2
icache_power_cc2_sum     202692.7885 # global sum of icache_power_cc2
dcache_power_cc2_00       52397.3468 # total power usage of dcache_cc2
dcache_power_cc2_01       52360.9667 # total power usage of dcache_cc2
dcache_power_cc2_sum     104758.3135 # global sum of dcache_power_cc2
dcache2_power_cc2_00       1286.2698 # total power usage of dcache2_cc2
dcache2_power_cc2_01       1297.7490 # total power usage of dcache2_cc2
dcache2_power_cc2_sum      2584.0188 # global sum of dcache2_power_cc2
ialu_power_cc2_00        141896.3892 # total power usage of alu_cc2
ialu_power_cc2_01        145300.7448 # total power usage of alu_cc2
ialu_power_cc2_sum       287197.1341 # global sum of ialu_power_cc2
imult_div_power_cc2_00     2257.8538 # total power usage of imult_div_cc2 FU
imult_div_power_cc2_01     2295.4251 # total power usage of imult_div_cc2 FU
imult_div_power_cc2_sum    4553.2789 # global sum of imult_div_power_cc2
mem_port_power_cc2_00         0.0000 # total power usage of memory-port_cc2 FU
mem_port_power_cc2_01         0.0000 # total power usage of memory-port_cc2 FU
mem_port_power_cc2_sum        0.0000 # global sum of mem_port_power_cc2
fpalu_power_cc2_00         3460.1758 # total power usage of fpalu_cc2
fpalu_power_cc2_01          320.1671 # total power usage of fpalu_cc2
fpalu_power_cc2_sum        3780.3428 # global sum of fpalu_power_cc2
fpmult_div_power_cc2_00    1071.9667 # total power usage of fpmult_div_cc2
fpmult_div_power_cc2_01      87.7495 # total power usage of fpmult_div_cc2
fpmult_div_power_cc2_sum   1159.7162 # global sum of fpmult_div_power_cc2
divmult_power_cc2_00          0.0000 # total power usage of divmult_cc2
divmult_power_cc2_01          0.0000 # total power usage of divmult_cc2
divmult_power_cc2_sum         0.0000 # global sum of divmult_power_cc2
homo_power_cc2_00             0.0000 # total power usage of homo_cc2 FU
homo_power_cc2_01             0.0000 # total power usage of homo_cc2 FU
homo_power_cc2_sum            0.0000 # global sum of homo_power_cc2
resultbus_power_cc2_00    70625.1439 # total power usage of resultbus_cc2
resultbus_power_cc2_01    73737.5606 # total power usage of resultbus_cc2
resultbus_power_cc2_sum  144362.7045 # global sum of resultbus_power_cc2
clock_power_cc2_00       754589.8239 # total power usage of clock_cc2
clock_power_cc2_01       874364.4305 # total power usage of clock_cc2
clock_power_cc2_sum     1628954.2544 # global sum of clock_power_cc2
avg_rename_power_cc2_00       0.0164 # avg power usage of rename unit_cc2
avg_rename_power_cc2_01       0.0168 # avg power usage of rename unit_cc2
avg_rename_power_cc2_sum      0.0332 # global sum of avg_rename_power_cc2
```

```
avg_bpred_power_cc2_00          0.0406 # avg power usage of bpred unit_cc2
avg_bpred_power_cc2_01          0.0442 # avg power usage of bpred unit_cc2
avg_bpred_power_cc2_sum         0.0847 # global sum of avg_bpred_power_cc2
avg_window_power_cc2_00         0.0857 # avg power usage of instruction
window_cc2
avg_window_power_cc2_01         0.0885 # avg power usage of instruction
window_cc2
avg_window_power_cc2_sum        0.1743 # global sum of avg_window_power_cc2
avg_lsq_power_cc2_00            0.0090 # avg power usage of instruction
lsq_cc2
avg_lsq_power_cc2_01            0.0090 # avg power usage of instruction
lsq_cc2
avg_lsq_power_cc2_sum           0.0180 # global sum of avg_lsq_power_cc2
avg_regfile_power_cc2_00        0.0413 # avg power usage of arch. regfile_cc2
avg_regfile_power_cc2_01        0.0418 # avg power usage of arch. regfile_cc2
avg_regfile_power_cc2_sum       0.0832 # global sum of avg_regfile_power_cc2
avg_ifq_power_cc2_00            0.0031 # avg power usage of ifq_cc2
avg_ifq_power_cc2_01            0.0031 # avg power usage of ifq_cc2
avg_ifq_power_cc2_sum           0.0062 # global sum of avg_ifq_power_cc2
avg_reorder_power_cc2_00        0.0308 # avg power usage of reorder_cc2
avg_reorder_power_cc2_01        0.0313 # avg power usage of reorder_cc2
avg_reorder_power_cc2_sum       0.0621 # global sum of avg_reorder_power_cc2
avg_icache_power_cc2_00         0.4279 # avg power usage of icache_cc2
avg_icache_power_cc2_01         0.4335 # avg power usage of icache_cc2
avg_icache_power_cc2_sum        0.8614 # global sum of avg_icache_power_cc2
avg_dcache_power_cc2_00         0.2227 # avg power usage of dcache_cc2
avg_dcache_power_cc2_01         0.2225 # avg power usage of dcache_cc2
avg_dcache_power_cc2_sum        0.4452 # global sum of avg_dcache_power_cc2
avg_dcache2_power_cc2_00        0.0055 # avg power usage of dcache2_cc2
avg_dcache2_power_cc2_01        0.0055 # avg power usage of dcache2_cc2
avg_dcache2_power_cc2_sum       0.0110 # global sum of avg_dcache2_power_cc2
avg_ialu_power_cc2_00           0.6030 # avg power usage of ialu_cc2
avg_ialu_power_cc2_01           0.6175 # avg power usage of ialu_cc2
avg_ialu_power_cc2_sum          1.2205 # global sum of avg_ialu_power_cc2
avg_fpalu_power_cc2_00          0.0147 # avg power usage of fpalu_cc2 FU
avg_fpalu_power_cc2_01          0.0014 # avg power usage of fpalu_cc2 FU
avg_fpalu_power_cc2_sum         0.0161 # global sum of avg_fpalu_power_cc2
avg_imult_div_power_cc2_00      0.0096 # avg power usage of imult_div_cc2 FU
avg_imult_div_power_cc2_01      0.0098 # avg power usage of imult_div_cc2 FU
avg_imult_div_power_cc2_sum     0.0194 # global sum of avg_imult_div_power_cc2
avg_mem_port_power_cc2_00       0.0000 # avg power usage of mem_port_cc2 FU
avg_mem_port_power_cc2_01       0.0000 # avg power usage of mem_port_cc2 FU
avg_mem_port_power_cc2_sum      0.0000 # global sum of avg_mem_port_power_cc2
avg_fpmult_div_power_cc2_00     0.0046 # avg power usage of fpmult_div_cc2 FU
avg_fpmult_div_power_cc2_01     0.0004 # avg power usage of fpmult_div_cc2 FU
avg_fpmult_div_power_cc2_sum    0.0049 # global sum of avg_fpmult_div_power
_cc2
avg_divmult_power_cc2_00        0.0000 # avg power usage of divmult_cc2 FU
avg_divmult_power_cc2_01        0.0000 # avg power usage of divmult_cc2 FU
avg_divmult_power_cc2_sum       0.0000 # global sum of avg_divmult_power_cc2
avg_homo_power_cc2_00           0.0000 # avg power usage of homo_cc2 FU
avg_homo_power_cc2_01           0.0000 # avg power usage of homo_cc2 FU
avg_homo_power_cc2_sum          0.0000 # global sum of avg_homo_power_cc2
avg_resultbus_power_cc2_00      0.3001 # avg power usage of resultbus_cc2
avg_resultbus_power_cc2_01      0.3134 # avg power usage of resultbus_cc2
avg_resultbus_power_cc2_sum     0.6135 # global sum of avg_resultbus_power_cc2
avg_clock_power_cc2_00          3.2069 # avg power usage of clock_cc2
avg_clock_power_cc2_01          3.7159 # avg power usage of clock_cc2
avg_clock_power_cc2_sum         6.9227 # global sum of avg_clock_power_cc2
```

```
fetch_stage_power_cc2_00 110951.0629 # total power usage of fetch stage_cc2
fetch_stage_power_cc2_01 113145.6173 # total power usage of fetch stage_cc2
fetch_stage_power_cc2_sum 224096.6801 # global sum of fetch_stage_power_cc2
dispatch_stage_power_cc2_00 3859.9832 # total power usage of dispatch
stage_cc2
dispatch_stage_power_cc2_01 3947.3926 # total power usage of dispatch
stage_cc2
dispatch_stage_power_cc2_sum 7807.3759 # global sum of dispatch_stage_power
_cc2
issue_stage_power_cc2_00 295281.6083 # total power usage of issue stage_cc2
issue_stage_power_cc2_01 298356.3430 # total power usage of issue stage_cc2
issue_stage_power_cc2_sum 593637.9513 # global sum of issue_stage_power_cc2
avg_fetch_power_cc2_00        0.4715 # average power of fetch unit per
cycle_cc2
avg_fetch_power_cc2_01        0.4808 # average power of fetch unit per
cycle_cc2
avg_fetch_power_cc2_sum       0.9524 # global sum of avg_fetch_power_cc2
avg_dispatch_power_cc2_00     0.0164 # average power of dispatch unit per
cycle_cc2
avg_dispatch_power_cc2_01     0.0168 # average power of dispatch unit per
cycle_cc2
avg_dispatch_power_cc2_sum    0.0332 # global sum of avg_dispatch_power_cc2
avg_issue_power_cc2_00        1.2549 # average power of issue unit per
cycle_cc2
avg_issue_power_cc2_01        1.2680 # average power of issue unit per
cycle_cc2
avg_issue_power_cc2_sum       2.5228 # global sum of avg_issue_power_cc2
total_power_cycle_cc2_00 1181649.5941 # total power per cycle_cc2
total_power_cycle_cc2_01 1307022.7184 # total power per cycle_cc2
total_power_cycle_cc2_sum 2488672.3124 # global sum of total_power_cycle_cc2
avg_total_power_cycle_cc2_00  5.0218 # average total power per cycle_cc2
avg_total_power_cycle_cc2_01  5.5546 # average total power per cycle_cc2
avg_total_power_cycle_cc2_sum 10.5764 # global sum of
avg_total_power_cycle_cc2
avg_total_power_insn_cc2_00   5.7999 # average total power per insn_cc2
avg_total_power_insn_cc2_01   6.2711 # average total power per insn_cc2
avg_total_power_insn_cc2_sum 12.0710 # global sum of avg_total_power_insn
_cc2
rename_power_cc3_00        4918.8117 # total power usage of rename unit_cc3
rename_power_cc3_01        4992.2584 # total power usage of rename unit_cc3
rename_power_cc3_sum       9911.0702 # global sum of rename_power_cc3
bpred_power_cc3_00        21727.3893 # total power usage of bpred unit_cc3
bpred_power_cc3_01        22441.7701 # total power usage of bpred unit_cc3
bpred_power_cc3_sum       44169.1594 # global sum of bpred_power_cc3
window_power_cc3_00       22891.3731 # total power usage of instruction
window_cc3
window_power_cc3_01       23535.1979 # total power usage of instruction
window_cc3
window_power_cc3_sum      46426.5710 # global sum of window_power_cc3
lsq_power_cc3_00           5936.9731 # total power usage of lsq_cc3
lsq_power_cc3_01           5953.1520 # total power usage of lsq_cc3
lsq_power_cc3_sum         11890.1251 # global sum of lsq_power_cc3
regfile_power_cc3_00      14965.8665 # total power usage of arch. regfile
_cc3
regfile_power_cc3_01      15040.0346 # total power usage of arch. regfile
_cc3
regfile_power_cc3_sum     30005.9011 # global sum of regfile_power_cc3
ifq_power_cc3_00           1306.8558 # total power usage of ifq_cc3
ifq_power_cc3_01           1316.0525 # total power usage of ifq_cc3
```

```
ifq_power_cc3_sum          2622.9083 # global sum of ifq_power_cc3
reorder_power_cc3_00       8092.2488 # total power usage of reorder_cc3
reorder_power_cc3_01       8152.5881 # total power usage of reorder_cc3
reorder_power_cc3_sum     16244.8370 # global sum of reorder_power_cc3
icache_power_cc3_00      113493.5679 # total power usage of icache_cc3
icache_power_cc3_01      114698.8419 # total power usage of icache_cc3
icache_power_cc3_sum     228192.4098 # global sum of icache_power_cc3
dcache_power_cc3_00       95867.2238 # total power usage of dcache_cc3
dcache_power_cc3_01       95826.8749 # total power usage of dcache_cc3
dcache_power_cc3_sum     191694.0987 # global sum of dcache_power_cc3
dcache2_power_cc3_00       5096.6597 # total power usage of dcache2_cc3
dcache2_power_cc3_01       5105.3770 # total power usage of dcache2_cc3
dcache2_power_cc3_sum     10202.0367 # global sum of dcache2_power_cc3
ialu_power_cc3_00        182690.1501 # total power usage of alu_cc3
ialu_power_cc3_01        185568.5680 # total power usage of alu_cc3
ialu_power_cc3_sum       368258.7181 # global sum of ialu_power_cc3
imult_div_power_cc3_00    16064.9341 # total power usage of imult_div_cc3 FU
imult_div_power_cc3_01    16098.7482 # total power usage of imult_div_cc3 FU
imult_div_power_cc3_sum   32163.6823 # global sum of imult_div_power_cc3
mem_port_power_cc3_00       0.0000 # total power usage of memory-port_cc3 FU
mem_port_power_cc3_01       0.0000 # total power usage of memory-port_cc3 FU
mem_port_power_cc3_sum        0.0000 # global sum of mem_port_power_cc3
fpalu_power_cc3_00       225365.8242 # total power usage of fpalu_cc3
fpalu_power_cc3_01       223412.5680 # total power usage of fpalu_cc3
fpalu_power_cc3_sum      448778.3922 # global sum of fpalu_power_cc3
fpmult_div_power_cc3_00   56769.8870 # total power usage of fpmult_div_cc3
fpmult_div_power_cc3_01   55884.0915 # total power usage of fpmult_div_cc3
fpmult_div_power_cc3_sum 112653.9785 # global sum of fpmult_div_power_cc3
divmult_power_cc3_00          0.0000 # total power usage of divmult_cc3
divmult_power_cc3_01          0.0000 # total power usage of divmult_cc3
divmult_power_cc3_sum         0.0000 # global sum of divmult_power_cc3
homo_power_cc3_00             0.0000 # total power usage of homo_cc3 FU
homo_power_cc3_01             0.0000 # total power usage of homo_cc3 FU
homo_power_cc3_sum            0.0000 # global sum of homo_power_cc3
resultbus_power_cc3_00    92232.3204 # total power usage of resultbus_cc3
resultbus_power_cc3_01    95477.2963 # total power usage of resultbus_cc3
resultbus_power_cc3_sum  187709.6167 # global sum of resultbus_power_cc3
clock_power_cc3_00      1677128.1686 # total power usage of clock_cc3
clock_power_cc3_01      1786145.9690 # total power usage of clock_cc3
clock_power_cc3_sum     3463274.1376 # global sum of clock_power_cc3
avg_rename_power_cc3_00       0.0209 # avg power usage of rename unit_cc3
avg_rename_power_cc3_01       0.0212 # avg power usage of rename unit_cc3
avg_rename_power_cc3_sum      0.0421 # global sum of avg_rename_power_cc3
avg_bpred_power_cc3_00        0.0923 # avg power usage of bpred unit_cc3
avg_bpred_power_cc3_01        0.0954 # avg power usage of bpred unit_cc3
avg_bpred_power_cc3_sum       0.1877 # global sum of avg_bpred_power_cc3
avg_window_power_cc3_00   0.0973 # avg power usage of instruction window_cc3
avg_window_power_cc3_01   0.1000 # avg power usage of instruction window_cc3
avg_window_power_cc3_sum      0.1973 # global sum of avg_window_power_cc3
avg_lsq_power_cc3_00         0.0252 # avg power usage of instruction lsq_cc3
avg_lsq_power_cc3_01         0.0253 # avg power usage of instruction lsq_cc3
avg_lsq_power_cc3_sum         0.0505 # global sum of avg_lsq_power_cc3
avg_regfile_power_cc3_00      0.0636 # avg power usage of arch. regfile_cc3
avg_regfile_power_cc3_01      0.0639 # avg power usage of arch. regfile_cc3
avg_regfile_power_cc3_sum     0.1275 # global sum of avg_regfile_power_cc3
avg_ifq_power_cc3_00          0.0056 # avg power usage of ifq_cc3
avg_ifq_power_cc3_01          0.0056 # avg power usage of ifq_cc3
avg_ifq_power_cc3_sum         0.0111 # global sum of avg_ifq_power_cc3
avg_reorder_power_cc3_00      0.0344 # avg power usage of reorder_cc3
```

```
avg_reorder_power_cc3_01      0.0346 # avg power usage of reorder_cc3
avg_reorder_power_cc3_sum     0.0690 # global sum of avg_reorder_power_cc3
avg_icache_power_cc3_00       0.4823 # avg power usage of icache_cc3
avg_icache_power_cc3_01       0.4874 # avg power usage of icache_cc3
avg_icache_power_cc3_sum      0.9698 # global sum of avg_icache_power_cc3
avg_dcache_power_cc3_00       0.4074 # avg power usage of dcache_cc3
avg_dcache_power_cc3_01       0.4072 # avg power usage of dcache_cc3
avg_dcache_power_cc3_sum      0.8147 # global sum of avg_dcache_power_cc3
avg_dcache2_power_cc3_00      0.0217 # avg power usage of dcache2_cc3
avg_dcache2_power_cc3_01      0.0217 # avg power usage of dcache2_cc3
avg_dcache2_power_cc3_sum     0.0434 # global sum of avg_dcache2_power_cc3
avg_ialu_power_cc3_00         0.7764 # avg power usage of ialu_cc3
avg_ialu_power_cc3_01         0.7886 # avg power usage of ialu_cc3
avg_ialu_power_cc3_sum        1.5650 # global sum of avg_ialu_power_cc3
avg_fpalu_power_cc3_00        0.9578 # avg power usage of fpalu_cc3 FU
avg_fpalu_power_cc3_01        0.9495 # avg power usage of fpalu_cc3 FU
avg_fpalu_power_cc3_sum       1.9072 # global sum of avg_fpalu_power_cc3
avg_imult_div_power_cc3_00    0.0683 # avg power usage of imult_div_cc3 FU
avg_imult_div_power_cc3_01    0.0684 # avg power usage of imult_div_cc3 FU
avg_imult_div_power_cc3_sum   0.1367 # global sum of avg_imult_div_power_cc3
avg_mem_port_power_cc3_00     0.0000 # avg power usage of mem_port_cc3 FU
avg_mem_port_power_cc3_01     0.0000 # avg power usage of mem_port_cc3 FU
avg_mem_port_power_cc3_sum    0.0000 # global sum of avg_mem_port_power_cc3
avg_fpmult_div_power_cc3_00   0.2413 # avg power usage of fpmult_div_cc3 FU
avg_fpmult_div_power_cc3_01   0.2375 # avg power usage of fpmult_div_cc3 FU
avg_fpmult_div_power_cc3_sum  0.4788 # global sum of avg_fpmult_div_power
_cc3
avg_divmult_power_cc3_00      0.0000 # avg power usage of divmult_cc3 FU
avg_divmult_power_cc3_01      0.0000 # avg power usage of divmult_cc3 FU
avg_divmult_power_cc3_sum     0.0000 # global sum of avg_divmult_power_cc3
avg_homo_power_cc3_00         0.0000 # avg power usage of homo_cc3 FU
avg_homo_power_cc3_01         0.0000 # avg power usage of homo_cc3 FU
avg_homo_power_cc3_sum        0.0000 # global sum of avg_homo_power_cc3
avg_resultbus_power_cc3_00    0.3920 # avg power usage of resultbus_cc3
avg_resultbus_power_cc3_01    0.4058 # avg power usage of resultbus_cc3
avg_resultbus_power_cc3_sum   0.7977 # global sum of avg_resultbus_power_cc3
avg_clock_power_cc3_00        7.1275 # avg power usage of clock_cc3
avg_clock_power_cc3_01        7.5908 # avg power usage of clock_cc3
avg_clock_power_cc3_sum      14.7182 # global sum of avg_clock_power_cc3
fetch_stage_power_cc3_00 136527.8130 # total power usage of fetch stage_cc3
fetch_stage_power_cc3_01 138456.6644 # total power usage of fetch stage_cc3
fetch_stage_power_cc3_sum 274984.4774 # global sum of fetch_stage_power_cc3
dispatch_stage_power_cc3_00 4918.8117 # total power usage of dispatch stage
_cc3
dispatch_stage_power_cc3_01 4992.2584 # total power usage of dispatch
stage_cc3
dispatch_stage_power_cc3_sum 9911.0702 # global sum of dispatch_stage_power
_cc3
issue_stage_power_cc3_00 702915.3454 # total power usage of issue stage_cc3
issue_stage_power_cc3_01 706861.8738 # total power usage of issue stage_cc3
issue_stage_power_cc3_sum 1409777.2192 # global sum of issue_stage_power_cc3
avg_fetch_power_cc3_00    0.5802 # average power of fetch unit per cycle_cc3
avg_fetch_power_cc3_01    0.5884 # average power of fetch unit per cycle_cc3
avg_fetch_power_cc3_sum       1.1686 # global sum of avg_fetch_power_cc3
avg_dispatch_power_cc3_00     0.0209 # average power of dispatch unit per
cycle_cc3
avg_dispatch_power_cc3_01     0.0212 # average power of dispatch unit per
cycle_cc3
avg_dispatch_power_cc3_sum    0.0421 # global sum of avg_dispatch_power_cc3
```

```
avg_issue_power_cc3_00        2.9873 # average power of issue unit per
cycle_cc3
avg_issue_power_cc3_01        3.0040 # average power of issue unit per
cycle_cc3
avg_issue_power_cc3_sum       5.9913 # global sum of avg_issue_power_cc3
total_power_cycle_cc3_00 2544548.2541 # total power per cycle_cc3
total_power_cycle_cc3_01 2659649.3884 # total power per cycle_cc3
total_power_cycle_cc3_sum 5204197.6425 # global sum of total_power_cycle_cc3
avg_total_power_cycle_cc3_00 10.8138 # average total power per cycle_cc3
avg_total_power_cycle_cc3_01 11.3030 # average total power per cycle_cc3
avg_total_power_cycle_cc3_sum 22.1168 # global sum of
avg_total_power_cycle_cc3
avg_total_power_insn_cc3_00  12.4894 # average total power per insn_cc3
avg_total_power_insn_cc3_01  12.7610 # average total power per insn_cc3
avg_total_power_insn_cc3_sum 25.2504 # global sum of total_power_insn_cc3
total_rename_access_00        203356 # total number accesses of rename unit
total_rename_access_01        207961 # total number accesses of rename unit
total_rename_access_sum  411317.0000 # global sum of total_rename_access
total_bpred_access_00          32274 # total number accesses of bpred unit
total_bpred_access_01          35111 # total number accesses of bpred unit
total_bpred_access_sum    67385.0000 # global sum of total_bpred_access
total_window_access_00        713774 # total number accesses of instruction
window
total_window_access_01        718985 # total number accesses of instruction
window
total_window_access_sum 1432759.0000 # global sum of total_window_access
total_lsq_access_00        48647 # total number accesses of load/store queue
total_lsq_access_01        48446 # total number accesses of load/store queue
total_lsq_access_sum      97093.0000 # global sum of total_lsq_access
total_regfile_access_00      293293 # total number accesses of arch. regfile
total_regfile_access_01      288167 # total number accesses of arch. regfile
total_regfile_access_sum 581460.0000 # global sum of total_regfile_access
total_ifq_access_00           212625 # total number accesses of ifq
total_ifq_access_01           217285 # total number accesses of ifq
total_ifq_access_sum     429910.0000 # global sum of total_ifq_access
total_reorder_access_00       591487 # total number accesses of reorder
total_reorder_access_01       601425 # total number accesses of reorder
total_reorder_access_sum 1192912.0000 # global sum of total_reorder_access
total_icache_access_00        226641 # total number accesses of icache
total_icache_access_01        231484 # total number accesses of icache
total_icache_access_sum  458125.0000 # global sum of total_icache_access
total_dcache_access_00         47529 # total number accesses of dcache
total_dcache_access_01         47496 # total number accesses of dcache
total_dcache_access_sum   95025.0000 # global sum of total_dcache_access
total_dcache2_access_00        14903 # total number accesses of dcache2
total_dcache2_access_01        15036 # total number accesses of dcache2
total_dcache2_access_sum  29939.0000 # global sum of total_dcache2_access
total_fus_access_00       189051 # total number accesses of Functional Units
total_fus_access_01       191774 # total number accesses of Functional Units
total_fus_access_sum     380825.0000 # global sum of total_fus_access
total_resultbus_access_00     188488 # total number accesses of resultbus
total_resultbus_access_01     189100 # total number accesses of resultbus
total_resultbus_access_sum 377588.0000 # global sum of total_resultbus
_access
avg_rename_access_00          0.8642 # avg number accesses of rename unit
avg_rename_access_01          0.8838 # avg number accesses of rename unit
avg_rename_access_sum         1.7480 # global sum of avg_rename_access
avg_bpred_access_00           0.1372 # avg number accesses of bpred unit
avg_bpred_access_01           0.1492 # avg number accesses of bpred unit
```

```
avg_bpred_access_sum          0.2864 # global sum of avg_bpred_access
avg_window_access_00          3.0334 # avg number accesses of instruction
window
avg_window_access_01          3.0555 # avg number accesses of instruction
window
avg_window_access_sum         6.0889 # global sum of avg_window_access
avg_lsq_access_00             0.2067 # avg number accesses of lsq
avg_lsq_access_01             0.2059 # avg number accesses of lsq
avg_lsq_access_sum            0.4126 # global sum of avg_lsq_access
avg_regfile_access_00         1.2464 # avg number accesses of arch. regfile
avg_regfile_access_01         1.2247 # avg number accesses of arch. regfile
avg_regfile_access_sum        2.4711 # global sum of avg_regfile_access
avg_ifq_access_00             0.9036 # avg number accesses of ifq
avg_ifq_access_01             0.9234 # avg number accesses of ifq
avg_ifq_access_sum            1.8270 # global sum of avg_ifq_access
avg_reorder_access_00         2.5137 # avg number accesses of reorder
avg_reorder_access_01         2.5559 # avg number accesses of reorder
avg_reorder_access_sum        5.0696 # global sum of avg_reorder_access
avg_icache_access_00          0.9632 # avg number accesses of icache
avg_icache_access_01          0.9838 # avg number accesses of icache
avg_icache_access_sum         1.9469 # global sum of avg_icache_access
avg_dcache_access_00          0.2020 # avg number accesses of dcache
avg_dcache_access_01          0.2018 # avg number accesses of dcache
avg_dcache_access_sum         0.4038 # global sum of avg_dcache_access
avg_dcache2_access_00         0.0633 # avg number accesses of dcache2
avg_dcache2_access_01         0.0639 # avg number accesses of dcache2
avg_dcache2_access_sum        0.1272 # global sum of avg_dcache2_access
avg_fus_access_00           0.8034 # avg number accesses of functional units
avg_fus_access_01           0.8150 # avg number accesses of functional units
avg_fus_access_sum            1.6184 # global sum of avg_fus_access
avg_resultbus_access_00       0.8010 # avg number accesses of resultbus
avg_resultbus_access_01       0.8036 # avg number accesses of resultbus
avg_resultbus_access_sum      1.6047 # global sum of avg_resultbus_access
max_rename_access_00               4 # max number accesses of rename unit
max_rename_access_01               4 # max number accesses of rename unit
max_bpred_access_00                4 # max number accesses of bpred unit
max_bpred_access_01                4 # max number accesses of bpred unit
max_window_access_00              16 # max number accesses of instruction window
max_window_access_01              16 # max number accesses of instruction window
max_lsq_access_00                  6 # max number accesses of load/store queue
max_lsq_access_01                  5 # max number accesses of load/store queue
max_regfile_access_00             12 # max number accesses of arch. regfile
max_regfile_access_01             12 # max number accesses of arch. regfile
max_ifq_access_00                  4 # max number accesses of ifq
max_ifq_access_01                  4 # max number accesses of ifq
max_reorder_access_00             12 # max number accesses of reorder
max_reorder_access_01             12 # max number accesses of reorder
max_icache_access_00               4 # max number accesses of icache
max_icache_access_01               4 # max number accesses of icache
max_dcache_access_00               4 # max number accesses of dcache
max_dcache_access_01               4 # max number accesses of dcache
max_dcache2_access_00              4 # max number accesses of dcache2
max_dcache2_access_01              4 # max number accesses of dcache2
max_fus_access_00                  4 # max number accesses of Functional Units
max_fus_access_01                  4 # max number accesses of Functional Units
max_resultbus_access_00            7 # max number accesses of resultbus
max_resultbus_access_01            7 # max number accesses of resultbus
max_cycle_power_cc1_00        27.5305 # maximum cycle power usage of cc1
max_cycle_power_cc1_01        26.4384 # maximum cycle power usage of cc1
```

```
max_cycle_power_cc2_00       13.6236 # maximum cycle power usage of cc2
max_cycle_power_cc2_01       13.2342 # maximum cycle power usage of cc2
max_cycle_power_cc3_00       18.7221 # maximum cycle power usage of cc3
max_cycle_power_cc3_01       18.3890 # maximum cycle power usage of cc3
sim_invalid_addrs_00               0 # total non-speculative bogus addresses
seen (debug var)
sim_invalid_addrs_01               0 # total non-speculative bogus addresses
seen (debug var)
ld_text_base_00           0x00400000 # program text (code) segment base
ld_text_base_01           0x00400000 # program text (code) segment base
ld_text_size_00                91744 # program text (code) size in bytes
ld_text_size_01                74640 # program text (code) size in bytes
ld_data_base_00           0x10000000 # program initialized data segment base
ld_data_base_01           0x10000000 # program initialized data segment base
ld_data_size_00                13028 # program init'ed `.data' and uninit'ed
`.bss' size in bytes
ld_data_size_01                13636 # program init'ed `.data' and uninit'ed
`.bss' size in bytes
ld_stack_base_00          0x7fffc000 # program stack segment base (highest
address in stack)
ld_stack_base_01          0x7fffc000 # program stack segment base (highest
address in stack)
ld_stack_size_00               16384 # program initial stack size
ld_stack_size_01               16384 # program initial stack size
ld_prog_entry_00          0x00400140 # program entry point (initial PC)
ld_prog_entry_01          0x00400140 # program entry point (initial PC)
ld_environ_base_00     0x7fff8000 # program environment base address address
ld_environ_base_01     0x7fff8000 # program environment base address address
ld_target_big_endian_00            0 # target executable endian-ness, non-
zero if big endian
ld_target_big_endian_01            0 # target executable endian-ness, non-
zero if big endian
mem_00.page_count_00              33 # total number of pages allocated
mem_01.page_count_01              28 # total number of pages allocated
mem_00.page_mem_00              132k # total size of memory pages allocated
mem_01.page_mem_01              112k # total size of memory pages allocated
mem_00.ptab_misses_00             34 # total first level page table misses
mem_01.ptab_misses_01             29 # total first level page table misses
mem_00.ptab_accesses_00       802911 # total page table accesses
mem_01.ptab_accesses_01       762719 # total page table accesses
mem_00.ptab_miss_rate_00      0.0000 # first level page table miss rate
mem_01.ptab_miss_rate_01      0.0000 # first level page table miss rate
```

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo