SOLUÇÃO PARALELA PARA SISTEMAS DE BALANÇO NÃO–LINEARES

Gustavo Hime

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO LABORATÓRIO NACIONAL DE COMPUTAÇÃO CIENTÍFICA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM MODELAGEM COMPUTACIONAL.

Aprovada por:

Prof. Bruno Richard Schulze, D. Sc.
(Presidente)

Prof. Eduardo Lúcio Mendes Garcia, D. Sc.

Prof. Dan Marchesin, Ph. D.

Prof. Eugene Francis Vinod Rebello, Ph. D.

RIO DE JANEIRO, BRASIL
SETEMBRO DE 2007

H657s

HIME, GUSTAVO

Solução paralela para sistemas de balanço não–lineares / Gustavo Hime, Rio de Janeiro, 2007.

V, 80 p. 29 cm (MCT/LNCC, M.Sc., Modelagem Computacional, 2007)

Dissertação - Laboratório Nacional de Computação Científica, LNCC

1. Algoritmos paralelos 2. Sistemas bloco tridiagonais 3. Análise de desempenho

I. Título II. MCT/LNCC

Aos que me ensinaram o valor daquilo que não é dito em palavras:

aos meus pais, por jamais haverem–me dito que seria fácil;

aos meus professores, por jamais haverem–me dito que era o bastante;

aos meus amigos, por jamais haverem–me dito meias verdades;

ao meu cachorro, por jamais haver dito nada.

A validação experimental dos resultados teóricos desta tese foi viável graças ao acesso a dois clusters de computadores:

o computador paralelo do Laboratório de Dinâmica dos Fluidos (IMPA), sob coordenação do Prof. Dan Marchesin, adquirido com recursos do convênio IMPA/FINEP número 01.04.0303.00,

um dos ambientes paralelos do National Center for Supercomputing Applications, que firmou um $Memorandum\ of\ Agreement$ com o LNCC em 2004 para cooperação acadêmica, cujo representante em assuntos de computação por parte do LNCC é o Prof. Bruno Schulze, orientador deste trabalho.

Resumo da Dissertação apresentada ao MCT/LNCC como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M. Sc.)

SOLUÇÃO PARALELA PARA SISTEMAS DE BALANÇO NÃO–LINEARES

Gustavo Hime

Setembro/2007

Orientador: Bruno Schulze

Modelos para diversos fenômenos baseiam–se em equações de balanço ou conservação. Dependendo do fenômeno e do que é admitido pelo modelo, as equações são simplificadas e resolvidas de diferentes modos. O problema de injeção em um meio poroso de um fluido bifásico cujo equilíbrio depende da temperatura, por exemplo, pode ser modelado por uma equação de conservação de massa que inclui um termo difusivo; esta equação, por sua vez, pode ser discretizada por diferenças finitas tanto no tempo quanto no espaço e resolvida numericamente.

O estudo estritamente analítico destes modelos é muito limitado. Uma compreensão mais detalhada do comportamento do modelo só pode ser obtida através de simulações numéricas e do estudo qualitativo de seus resultados. Os resultados de uma simulação só podem ser visualizados uma vez que esta tenha sido concluída: mas simulações de alta qualidade requerem simulações em malhas mais finas, que necessitam de mais tempo computacional. Mesmo para fluxos unidimensionais, o ciclo interativo de especificar os parâmetros para uma nova simulação com base nas conclusões tiradas de simulações prévias necessariamente inclui um tempo de espera indesejável. Sistemas capazes de resolver esta classe de problemas numéricos rápida e eficientemente são portanto o objetivo principal deste trabalho.

Para obter alto desempenho no cálculo destas soluções, muitos fatores precisam ser levados em consideração: o custo computacional inerente às equações constitutivas usadas no modelo, o tipo específico de sistema linear resultante da discretização do problema, as diferentes alternativas quanto ao algoritmo de solução do sistema — e suas implementações — e os pontos fortes e limitações impostas por cada ambiente computacional que se deseja explorar. Como resultado do teste de diversas abordagens em diferentes máquinas, nós obtemos não somente um motor numérico eficiente para os casos de estudo apresentados neste trabalho, mas também um guia para a aplicação destas técnicas a problemas similares.

Abstract of the Dissertation presented to MCT/LNCC as a partial fullfillment of the requirements for the degree of Master of Science (M. Sc.)

PARALLEL SOLUTION OF NONLINEAR BALANCE SYSTEMS

Gustavo Hime

September/2007

Advisor: Bruno Schulze

Models for many physical phenomena are based on balance or conservation equations. Depending on the phenomenon and the model assumptions, the equations are simplified and solved in different ways. The problem of injection into porous media of a two–phase fluid subject to temperature dependent equilibria, for instance, can be modeled with a mass conservation equation that includes a diffusion term; this equation, in turn, can be discretized via finite differences both in space and time and solved numerically.

The strictly analytical study of these models has severe limitations. Further understanding of a model's behavior can be obtained only through numerical simulations and qualitative study of the results. The results of a simulation can be visualized only once it is concluded: but high quality simulations require higher discretization resolutions, and therefore longer computation times. Even for one dimensional flows, the interactive cycle of setting up new simulations based on the conclusions drawn from previous ones necessarily includes an undesirable waiting time. Fast and efficient solvers for this particular class of problems are the main objective in this work.

In order to achieve high performance while computing these solutions, many factors must be taken into account: the computational costs inherent to the constitutive equations used in the model, the specific type of linear system resulting from the linearization of the discrete problem, the different algorithmic and implementational alternatives for the numerical solution, and the computational environment strengths and limitations. As a result of testing different approaches on different machines, we obtain not only a highly efficient numerical engine for the case studies presented in this work, but also a roadmap for applying these techniques to similar problems.

Sumário

CAPíTULO 1

Introdução

Diversos fenômenos físicos são estudados usando modelos baseados em equações que expressam a conservação de massa ou de outras grandezas ao longo do tempo. Em particular, problemas relativos a escoamentos multifásicos vêm sendo objeto de intensa investigação nas últimas décadas, estando relacionados a aplicações economica e socialmente críticas como a recuperação de petróleo e a descontaminação do solo e de aquíferos. A exploração eficiente destes recursos naturais limitados exige uma boa compreensão dos impactos resultantes de diferentes abordagens à exploração.

Modelos coerentes e consistentes para estes e outros fenômenos já existem e foram exaustivamente estudados com as técnicas disponíveis até o presente momento. Quando de sua concepção, numa época em que computação numérica ainda não era algo disponível ou economicamente viável, a maioria dos estudos restringiu–se a aspectos puramente teóricos. As soluções analíticas, entretanto, costumam ser de difícil obtenção, quando obteníveis de todo. Além disso, estas soluções nem sempre esclarecem o comportamento do modelo em contextos realistas, servindo apenas para situações altamente idealizadas no processo de modelagem por simplificações e premissas restritivas. Relaxar estas premissas pode tornar a solução analítica inadmissível ou inválida, e requerer que o modelo seja estudado através de simulações numéricas.

Nos anos 60, o uso de computadores passou de uma mera possibilidade para uma realidade no meio acadêmico. Procedimentos numéricos que até então jamais haviam sido considerados foram pela primeira vez concebidos e implementados com sucesso. Esta tendência não mudou de lá para cá: conforme os computadores evoluem e provêem novas possibilidades, os cientistas buscam novas formas de tirar vantagem destas possibilidades na investigação de problemas, tanto antigos quanto novos. Com o apoio da computação digital, o que antes

era rotulado “impossível” pode agora ser dito “impraticável” — com a tecnologia disponível no momento.

A disponibilidade de computadores não basta: uma ferramenta é tão boa quanto o uso que se faz dela. As últimas três décadas trouxeram muitas mudanças qualitativas aos paradigmas de todos os níveis da ciência da computação e da tecnologia a ela associada, desde a arquitetura de hardware no nível mais baixo até a engenharia de software no mais alto. Os primeiros simuladores numéricos — por exemplo, aqueles escritos para os primeiros computadores digitais por cientistas a serviço dos militares, no final da Segunda Guerra, ou seja, em meados dos anos 40 — foram concebidos num contexto tecnólogico no qual a separação entre hardware e software não era clara. Trinta anos mais tarde, os primeiros simuladores de escoamento multifásico escritos num ambiente puramente acadêmico já se apoiavam em subestruturas deveras sofisticadas como linguagens compiladas (por exemplo, FORTRAN firmou–se no início dos anos 60) e sistemas operacionais (os precursores do UNIX datam do início dos anos 70). Mais trinta anos se passaram, e observa–se que o melhoramento dos simuladores numéricos não acompanhou a evolução dos computadores. Código escrito há trinta anos, como o pacote BLAS, é usado ainda hoje, praticamente inalterado.

Não se deve encarar esta realidade como um problema a ser resolvido, pois não há razão para descartar a maior parte deste software numérico. Os fundamentos da álgebra linear e os algoritmos correspondentes, por exemplo, que são os blocos elementares nos quais a maioria dos simuladores numéricos se apóia, já eram bem entendidos e foram bem implementados há muito tempo (por exemplo, o pacote BLAS, amplamente difundido, foi escrito no fim dos anos 70). Mas os níveis mais altos da hierarquia de software dos simuladores numéricos poderiam ter–se beneficiado de modernizações radicais, o que muitas vezes não ocorreu. Tentativas para modernizar tais simuladores esbarraram em diversas dificuldades, como a identificação de quais partes modernizar, quais os custos e os benefícios trazidos pela adoção de novas abordagens em detrimento das mais antigas, já conhecidas. Os computadores tornaram–se capazes de fazer muito mais do que era necessário aos simuladores originais, e logo tornou–se comum adicionar funcionalidades supérfluas mas cômodas a programas

antes simples, como interatividade e mecanismos de visualização. Os ganhos trazidos por estes acréscimos nem sempre compensavam, e algumas ferramentas científicas tornaram–se ineficientes e rebuscadas, pois o foco deixou de ser exclusivamente a computação numérica, e a gama de conhecimento técnico necessária para a manutenção destes aplicativos mais complexos mostrou–se muito mais difícil de agregar em ambientes acadêmicos.

Os simuladores antigos certamente tiraram vantagem do aumento geral no desempenho dos computadores, pois processadores mais rápidos executam seus programas em menos tempo e mais memória permite a solução de problemas maiores. Entretanto, o salto qualitativo trazido por novos paradigmas como computação paralela não pode ser feito sem uma mudança correspondente na arquitetura do simulador. Ao mesmo tempo, a disponibilidade de mais poder computacional permite a execução de simulações em duas e três dimensões espaciais, as quais passaram a ser prioridade por razões óbvias. Esta mudança de foco fez com que o estudo de simulações de alta qualidade de escoamentos em uma dimensão espacial fosse negligenciado, juntamente com questões científicas básicas que podem ser mais bem entendidas neste contexto mais simples. O objetivo deste trabalho é a exploração de ambientes paralelos, ao máximo, para realizar simulações numéricas unidimensionais de modelos baseados em sistemas de balanço. Para tanto, nós estudamos no Capítulo 1 o modelo geral para a classe de problemas a ser tratada, sua discretização, e os problemas de álgebra linear a ela relativos. No Capítulo 2, estudamos e comparamos diferentes algoritmos para resolver estes problemas de álgebra linear num ambiente paralelo de computação. No Capítulo 3, nós aplicamos o conhecimento adquirido para obter resultados de simulações, e no Capítulo 4 sumarizamos nossas conclusões.

1. Motivação

O problema geral cuja solução buscamos pode ser mais bem ilustrado como denominador comum de diversos casos particulares que surgem no contexto de escoamento multifásico em uma dimensão. Um primeiro exemplo é o escoamento trifásico de água, óleo e gás num meio

poroso, que aparece no contexto de recuperação secundária de petróleo, conforme apresentado em [**24**] e suas referências. O escoamento é descrito por uma equação de pressão, que expressa a lei de Darcy para força, acoplada a duas equações de saturação, que generalizam a equação clássica de Buckley–Leverett para escoamento bifásico e expressam a conservação das três grandezas — as massas de água, óleo e gás. A lei de Darcy para escoamentos unidimensionais incompressíveis implica na velocidade total do fluido ser independente da posição, logo se esta for dada como uma condição de contorno independente do tempo então necessariamente é constante em todo o domínio, e o sistema adimensional pode ser escrito como

$$\frac{\partial s_w}{\partial t} + \frac{\partial}{\partial x} f_w(s_w, s_g) = D_w \quad \text{e} \quad \frac{\partial s_g}{\partial t} + \frac{\partial}{\partial x} f_g(s_w, s_g) = D_g, \tag{1.1}$$

onde s_w e s_g denotam as saturações de água e gás, e como a saturação de óleo é $s_o = 1 - s_w - s_g$ o estado do fluido é definido por $\mathbf{u} = (s_w, s_g)$. Na generalização do modelo clássico introduzido por Buckley–Leverett em [**7**] surgem as funções de fluxo fracional f_w e f_g que são dadas em termos das viscosidades fixas dos fluidos, μ_w, μ_g e μ_o, e das funções de permeabilidade relativa, k_w, k_g e k_o, ou seja,

$$f_w = \frac{k_w/\mu_w}{k_w/\mu_w + k_g/\mu_g + k_o/\mu_w} \quad \text{e} \quad f_g = \frac{k_g/\mu_g}{k_w/\mu_w + k_g/\mu_g + k_o/\mu_w}, \tag{1.2}$$

Segundo [**3**], os termos difusivos devidos aos efeitos da pressão capilar

$$D_w = \frac{\partial}{\partial x}\left(B_{ww}\frac{\partial s_w}{\partial x}\right) + \frac{\partial}{\partial x}\left(B_{wg}\frac{\partial s_g}{\partial x}\right) \quad \text{e} \quad D_g = \frac{\partial}{\partial x}\left(B_{gw}\frac{\partial s_w}{\partial x}\right) + \frac{\partial}{\partial x}\left(B_{gg}\frac{\partial s_g}{\partial x}\right) \tag{1.3}$$

representam o efeito das diferenças das pressões capilares entre os fluidos, com os quatro valores de B tirados da matriz

$$\mathbf{B} = \begin{bmatrix} B_{ww} & B_{wg} \\ B_{gw} & B_{gg} \end{bmatrix} = \begin{bmatrix} \dfrac{k_w}{\mu_w}(1 - f_w) & -\dfrac{k_w}{\mu_w} f_g \\ \dfrac{k_g}{\mu_g} f_w & \dfrac{k_g}{\mu_g}(1 - f_g) \end{bmatrix} \begin{bmatrix} \dfrac{\partial p_{c,wo}}{\partial s_w} & \dfrac{\partial p_{c,wo}}{\partial s_g} \\ \dfrac{\partial p_{c,go}}{\partial s_w} & \dfrac{\partial p_{c,go}}{\partial s_g} \end{bmatrix},$$

que depende também das pressões capilares

$$p_{c,wo} = p_w - p_o \quad \text{e} \quad p_{c,go} = p_c - p_o.$$

O modelo costuma ser simplificado ainda mais ([**12**]) admitindo que as funções de permeabilidade relativas k dependem das saturações de ambos os fluidos e nada mais. As diferenças

de pressão capilar $p_{c,wo}$ e $p_{c,go}$ dependem apenas de s_w e s_g respectivamente. Em síntese, a equação (1.1) pode ser escrita como

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{f}(\mathbf{u})}{\partial x} = \frac{\partial}{\partial x}\left(\mathbf{g}(\mathbf{u})\frac{\partial \mathbf{u}}{\partial x}\right), \tag{1.4}$$

com o vetor de estado $\mathbf{u} = (s_w, s_g)$, a função vetorial $\mathbf{f}$ dada por (1.2) e a função de difusão $\mathbf{g}$ dada por (1.3).

Usaremos este modelo simples como caso de estudo no Capítulo 3; mas fenômenos mais complexos podem ser modelados usando equações semelhantes. Por exemplo, outra situação é a injeção de água, vapor e nitrogênio ou outro gás não–reativo num meio poroso saturado com água, conforme apresentado no contexto de remoção de poluentes do solo em [**23**]. Um modelo representativo para este fenômeno requer não apenas o balanço de massa como no caso anterior, mas também o balanço de energia do processo termodinâmico envolvido. Usando definições semelhantes para fluxo fracional e saturação, a conservação de massa para este escoamento bifásico pode ser escrita como

$$\begin{aligned}
\frac{\partial}{\partial t}(\varphi \varrho_w s_w) + \frac{\partial}{\partial x}(v \varrho_w f_w) &= q_{g\to a,w}, \\
\frac{\partial}{\partial t}(\varphi \rho_{gw} s_g) + \frac{\partial}{\partial x}(v \rho_{gw} f_g) &= -q_{g\to a,w}, \\
\frac{\partial}{\partial t}(\varphi \rho_{gn} s_g) + \frac{\partial}{\partial x}(v \rho_{gn} f_g) &= 0,
\end{aligned} \tag{1.5}$$

onde as saturações de água e de gás total são s_w e s_g, as concentrações de nitrogênio e vapor na fase gasosa são ρ_{gn} e ρ_{gw}, ϱ_w é a densidade da água e a velocidade total de Darcy é $v = v_g + v_w$, que por sua vez são dadas por $v_g = v f_g$ e $v_w = v f_w$. A taxa de condensação da água é denotada por $q_{g\to a,w}$. Quanto ao balanço de energia, a conservação de entalpia é dada por uma equação semelhante,

$$\frac{\partial}{\partial t}\big(H_r + \varphi s_w H_w + \varphi s_g (H_{gw} + H_{gn})\big) + \frac{\partial}{\partial x}\big(v_w H_w + v_g (H_{gw} + H_{gn})\big) = 0, \tag{1.6}$$

onde H_r é a entalpia da rocha, H_w é a entalpia da água na fase líquida, e H_{gw} e H_{gn} são as entalpias do vapor e do nitrogênio na fase gasosa. Estas são densidades de entalpia por volume unitário, ou seja, são o produto das entalpias específicas h pelas respectivas

densidades ϱ. O sistema completo dado por (1.5) e (1.6) pode ser escrito na forma

$$\frac{\partial \mathbf{h}(\mathbf{u})}{\partial t} + \frac{\partial \big(v\mathbf{f}(\mathbf{u})\big)}{\partial x} = \mathbf{q}(\mathbf{u}); \tag{1.7}$$

após manipulações consideravelmente extensas no termo de fluxo, podemos inclur v numa nova definição do vetor de estado $\mathbf{u}$ e mudar $\mathbf{f}$ de acordo, ganhando com isso um termo de segunda ordem, levando a uma equação da forma

$$\frac{\partial \mathbf{h}(\mathbf{u})}{\partial t} + \frac{\partial \mathbf{f}(\mathbf{u})}{\partial x} = \frac{\partial}{\partial x}\left(\mathbf{g}(\mathbf{u})\frac{\partial \mathbf{u}}{\partial x}\right) + \mathbf{q}(\mathbf{u}),$$

que é semelhante à equação (1.4), mas com uma não–linearidade no termo de acumulação e o acréscimo de um termo de fonte. Esta é a equação geral de convecção–difusão–reação, cujo estudo numérico é nossa motivação principal.

Muitos outros fenômenos físicos podem ser modelados de forma a encaixarem–se nesta forma geral. O objetivo deste trabalho é o desenvolvimento de algoritmos paralelos para resolver estes sistemas nesta forma, e assim prover um mecanismo genérico para pesquisa em diversos campos de aplicação.

2. Formulação do problema

Reproduzimos aqui a formulação do problema discreto tal como encontrado na literatura ([**24**]). Buscamos a solução numérica de um problema de valor inicial/condição de contorno em uma dimensão governado por um sistema de equações diferenciais parciais de convecção–difusão–reação em $\mathbf{u}(x,t)$ da forma

$$\frac{\partial \mathbf{h}(\mathbf{u})}{\partial t} + \frac{\partial \mathbf{f}(\mathbf{u})}{\partial x} = \frac{\partial}{\partial x}\left(\mathbf{g}(\mathbf{u})\frac{\partial \mathbf{u}}{\partial x}\right) + \mathbf{q}(\mathbf{u}), \tag{1.8}$$

onde $\mathbf{u} \in \mathbb{R}^M$, $\mathbf{h}(\mathbf{u}), \mathbf{f}(\mathbf{u}), \mathbf{q}(\mathbf{u}) : \mathbb{R}^M \to \mathbb{R}^M$ e $\mathbf{g}(\mathbf{u}) : \mathbb{R}^M \to \mathbb{R}^{M\times M}$ são diferenciáveis, o domínio físico é $x \in [a,b]$ e $t \geq 0$, e os quatro termos da equação (1.8) correspondem a acumulação, convecção, difusão e reação, da esquerda para a direita. As condições de contorno e iniciais podem variar. Um caso comum ao qual iremos nos restringir corresponde ao problema de Dirichlet:

$$\mathbf{u}(a,t) = \mathbf{u}_a, \quad \mathbf{u}(b,t) = \mathbf{u}_b, \quad \text{e} \quad \mathbf{u}(x,0) = \mathbf{u}^0(x), a \leq x \leq b. \tag{1.9}$$

Discretizamos o domínio $x \in [a,b]$ em $N-1$ subintervalos de comprimento $h = (b-a)/(N-1)$, delimitados pelos pontos x_i, $i = 1, \cdots, N$, com $x_1 = a$ e $x_N = b$; analogamente, adotamos intervalos discretos de comprimento k no tempo t, ou seja, $t^n - t^{n-1} = k$, e indexamos os tempos t^n, $n \geq 1$, com $t^1 \equiv 0$. Para descarregar a notação, denotamos $\mathbf{u}(x_i, t^n) \equiv \mathbf{u}_i^n$, e por extensão $\mathbf{f}(\mathbf{u}(x_i,t^n)) = \mathbf{f}_i^n$, $\mathbf{h}(\mathbf{u}(x_i,t^n)) = \mathbf{h}_i^n$, $\mathbf{g}(\mathbf{u}(x_i,t^n)) = \mathbf{g}_i^n$ e $\mathbf{q}(\mathbf{u}(x_i,t^n)) = \mathbf{q}_i^n$.

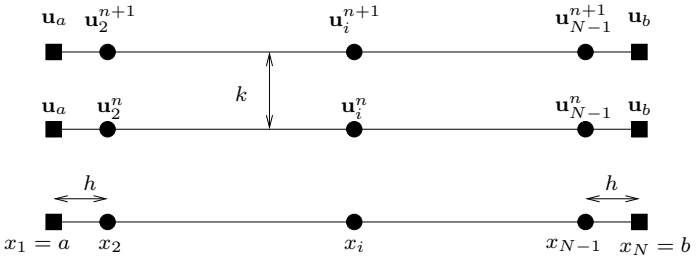


FIGURA 1.1. Malha unidimensional uniforme.

Usando diferença central e médias no tempo, ou seja,

$$\frac{\partial}{\partial t}\mathbf{h}^{n+1/2} = \frac{\mathbf{h}^{n+1} - \mathbf{h}^n}{k} + \mathcal{O}(k^2) \quad \text{e} \quad \mathbf{f}^{n+1/2} = \frac{\mathbf{f}^{n+1} + \mathbf{f}^n}{2} + \mathcal{O}(k^2), \tag{1.10}$$

a equação (1.8) pode ser aproximada no tempo pelo esquema Crank–Nicolson [**30**] por

$$\frac{\mathbf{h}^{n+1} - \mathbf{h}^n}{k} + \frac{1}{2}\left[\left(\frac{\partial \mathbf{f}(\mathbf{u})}{\partial x}\right)^{n+1} + \left(\frac{\partial \mathbf{f}(\mathbf{u})}{\partial x}\right)^{n}\right] = \frac{1}{2}\frac{\partial}{\partial x}\left[\left(\mathbf{g}(\mathbf{u})\frac{\partial \mathbf{u}}{\partial x}\right)^{n+1} + \left(\mathbf{g}(\mathbf{u})\frac{\partial \mathbf{u}}{\partial x}\right)^{n}\right] + \mathbf{q}^{n+1/2} + \mathcal{O}(k^2). \tag{1.11}$$

Aplicando o operador de diferença central para a primeira derivada no espaço dado por

$$\frac{\partial}{\partial x}\mathbf{f}_i = \frac{\mathbf{f}_{i+1} - \mathbf{f}_{i-1}}{2h} + \mathcal{O}(h^2)$$

na equação (1.11) nos dá, para o lado esquerdo,

$$\frac{\mathbf{h}_i^{n+1} - \mathbf{h}_i^n}{k} + \frac{\mathbf{f}_{i+1}^{n+1} - \mathbf{f}_{i-1}^{n+1} + \mathbf{f}_{i+1}^n - \mathbf{f}_{i-1}^n}{4h} + \mathcal{O}(h^2); \tag{1.12}$$

para o lado direito, usamos os pontos médios virtuais das células da malha discreta em x, denotados pelos índices $i \pm 1/2$, e obtemos

$$\begin{aligned} \frac{\partial}{\partial x}\left(\mathbf{g}(\mathbf{u})\frac{\partial \mathbf{u}}{\partial x}\right) &= \frac{1}{h}\left(\mathbf{g}_{i+1/2}\frac{\partial \mathbf{u}}{\partial x}\bigg|_{x=x_{i+1/2}} - \mathbf{g}_{i-1/2}\frac{\partial \mathbf{u}}{\partial x}\bigg|_{x=x_{i-1/2}}\right) + \mathcal{O}(h^2) \\ &= \frac{1}{h}\left(\mathbf{g}_{i+1/2}\frac{\mathbf{u}_{i+1} - \mathbf{u}_i}{h} - \mathbf{g}_{i-1/2}\frac{\mathbf{u}_i - \mathbf{u}_{i-1}}{h}\right) + \mathcal{O}(h^2), \end{aligned} \tag{1.13}$$

onde o valor de $\mathbf{g}_{i\pm1/2}$ pode ser aproximado por

$$\mathbf{g}_{i\pm1/2} = \frac{\mathbf{g}_{i\pm1} + \mathbf{g}_i}{2} + \mathcal{O}(h^2),$$

com o que a forma final da expressão (1.13) é

$$\begin{aligned}\frac{\partial}{\partial x}\left(\mathbf{g}(\mathbf{u})\frac{\partial \mathbf{u}}{\partial x}\right) &= \frac{1}{h^2}\left(\frac{\mathbf{g}_{i+1}+\mathbf{g}_i}{2}(\mathbf{u}_{i+1}-\mathbf{u}_i)+\frac{\mathbf{g}_{i-1}+\mathbf{g}_i}{2}(\mathbf{u}_i-\mathbf{u}_{i-1})\right)+\mathcal{O}(h^2)\\ &= \frac{1}{2h^2}\big((\mathbf{g}_{i+1}+\mathbf{g}_i)\mathbf{u}_{i+1}-(\mathbf{g}_{i+1}+2\mathbf{g}_i+\mathbf{g}_{i-1})\mathbf{u}_i+(\mathbf{g}_i+\mathbf{g}_{i-1})\mathbf{u}_{i-1}\big)+\mathcal{O}(h^2).\end{aligned} \tag{1.14}$$

Compondo as expressões (1.10) a (1.14), chegamos à equação totalmente discreta

$$\begin{aligned}&\frac{\mathbf{h}_i^{n+1}-\mathbf{h}_i^n}{k}+\frac{\mathbf{f}_{i+1}^{n+1}-\mathbf{f}_{i-1}^{n+1}+\mathbf{f}_{i+1}^{n}-\mathbf{f}_{i-1}^{n}}{4h}=\\ &\quad\frac{1}{4h^2}\big((\mathbf{g}_{i+1}^{n+1}+\mathbf{g}_i^{n+1})\mathbf{u}_{i+1}^{n+1}-(\mathbf{g}_{i+1}^{n+1}+2\mathbf{g}_i^{n+1}+\mathbf{g}_{i-1}^{n+1})\mathbf{u}_i^{n+1}+(\mathbf{g}_i^{n+1}+\mathbf{g}_{i-1}^{n+1})\mathbf{u}_{i-1}^{n+1}+\\ &\qquad(\mathbf{g}_{i+1}^{n}+\mathbf{g}_i^{n})\mathbf{u}_{i+1}^{n}-(\mathbf{g}_{i+1}^{n}+2\mathbf{g}_i^{n}+\mathbf{g}_{i-1}^{n})\mathbf{u}_i^{n}+(\mathbf{g}_i^{n}+\mathbf{g}_{i-1}^{n})\mathbf{u}_{i-1}^{n}\big)\\ &\quad+\frac{\mathbf{q}_i^{n+1}+\mathbf{q}_i^n}{2}+\mathcal{O}(k^2+h^2).\end{aligned} \tag{1.15}$$

Da equação (1.15) obtém-se um esquema numérico para integração no tempo: separando os termos que envolvem as variáveis a serem determinadas no tempo $n+1$ do lado esquerdo, obtemos

$$\begin{aligned}&\frac{\mathbf{h}_i^{n+1}}{k}+\frac{\mathbf{f}_{i+1}^{n+1}-\mathbf{f}_{i-1}^{n+1}}{4h}-\\ &\frac{1}{4h^2}\big((\mathbf{g}_{i+1}^{n+1}+\mathbf{g}_i^{n+1})\mathbf{u}_{i+1}^{n+1}-(\mathbf{g}_{i+1}^{n+1}+2\mathbf{g}_i^{n+1}+\mathbf{g}_{i-1}^{n+1})\mathbf{u}_i^{n+1}+(\mathbf{g}_i^{n+1}+\mathbf{g}_{i-1}^{n+1})\mathbf{u}_{i-1}^{n+1}\big)-\frac{\mathbf{q}_i^{n+1}}{2}=\end{aligned} \tag{1.16}$$

$$\frac{\mathbf{h}_i^n}{k}-\frac{\mathbf{f}_{i+1}^n-\mathbf{f}_{i-1}^n}{4h}+\frac{1}{4h^2}\big((\mathbf{g}_{i+1}^{n}+\mathbf{g}_i^{n})\mathbf{u}_{i+1}^{n}-(\mathbf{g}_{i+1}^{n}+2\mathbf{g}_i^{n}+\mathbf{g}_{i-1}^{n})\mathbf{u}_i^{n}+(\mathbf{g}_i^{n}+\mathbf{g}_{i-1}^{n})\mathbf{u}_{i-1}^{n}\big)+\frac{\mathbf{q}_i^n}{2}, \tag{1.17}$$

com um erro de ordem $\mathcal{O}(k^2+h^2)$.

Existem diversas condições de contorno de interesse, válidas sob diferentes premissas: sua discussão está além do escopo deste trabalho. Aqui, restringimo–nos às condições mais simples de Dirichlet (1.9), para as quais $\mathbf{u}_1^n$ e $\mathbf{u}_N^n$ são dados: se os valores de $\mathbf{u}_i^n$, $i=2,\ldots,N-1$ são conhecidos, a equação (1.16)–(1.17) define um sistema de $N-2$ equações não–lineares e $N-2$ incógnitas nas variáveis $\mathbf{u}_i^{n+1}$, ou seja, o lado direito (1.17) pode ser computado a partir dos valores conhecidos para t^n, e o lado esquerdo (1.16) contém as incógnitas no tempo t^{n+1}.

Nas médias temporais feitas nas equações (1.10) e (1.11), optamos por dar pesos iguais aos dois níveis de tempo envolvidos de forma a termos convergência de segunda ordem no

tempo. Entretanto, poderíamos ter usado quaisquer dois valores $w^+ + w^- = 1$ nesta média ponderada. Se $w^+ = w^- = 1/2$, obtém–se o esquema Crank-Nicolson desenvolvido acima. Valores mais altos para w^+ resultam em esquemas numéricos mais difusivos e estáveis, mas de primeira ordem no tempo. Em particular, se $w^+ = 1$ nós obtemos o método de Euler para trás. Neste trabalho nós consideramos apenas o valor fixo de $1/2$ para estes pesos, mas toda a família de esquemas pode ser obtida com pequenas modificações à dedução apresentada.

O problema numérico a ser resolvido tem tamanho $\{N, M\}$, onde N é o número de pontos da discretização e M é o número de leis de conservação do modelo. Experiência prévia na análise de tais modelos sugere um intervalo de interesse particular para estes dois parâmetros: $N \in [10^2, 10^9]$ and $M \in [2, 12]$.

3. Solução iterativa

Num dado instante t^n, os valores de $\mathbf{u}_i^n$ são conhecidos e queremos determinar os valores de $\mathbf{u}_i^{n+1}$. Nós suprimimos o índice n nesta seção, uma vez que o problema está confinado aos dados num instante particular t^n e incógnitas em t^{n+1}. A equação (1.16)–(1.17) pode ser reescrita sinteticamente, com $\mathbf{u}$ denotando $\mathbf{u}^{n+1}$, como

$$\mathbf{G}(\mathbf{u}) = \mathbf{F}(\mathbf{u}) - \mathbf{y} = 0, \tag{1.18}$$

onde $\mathbf{F}(\mathbf{u}) : \mathbb{R}^{M(N-2)} \to \mathbb{R}^{M(N-2)}$ é o lado esquerdo (1.16), que é uma função não–linear das incógnitas, e $\mathbf{y} \in \mathbb{R}^{M(N-2)}$ é o lado direito (1.17), que para um n fixo é conhecido. A mesma abreviação de notação usada para as funções do vetor de estado na página 7 é aplicável, então denotamos $\mathbf{G}(\mathbf{u}_i) \equiv \mathbf{G}_i$.

Aplicamos o método de Newton para encontrar a raiz $\mathbf{u}$ de $\mathbf{G}(\mathbf{u}) : \mathbb{R}^{M(N-2)} \to \mathbb{R}^{M(N-2)}$, isto é, de uma aproximação inicial $\mathbf{u}^{(0)}$, procedemos pela iteração de Newton

$$\mathbf{u}^{(l+1)} = \mathbf{u}^{(l)} + \delta\mathbf{u}, \quad \text{onde} \quad \left[\frac{\partial \mathbf{G}(\mathbf{u})}{\partial \mathbf{u}}\right]_{\mathbf{u}=\mathbf{u}^{(l)}} \delta\mathbf{u} = -\mathbf{G}(\mathbf{u}^{(l)}) \tag{1.19}$$

até que $|\mathbf{G}(\mathbf{u}^{(l)})|/|\mathbf{G}_{\text{ref}}| < \epsilon_1$ ou $|\delta\mathbf{u}|/|\mathbf{u}_{\text{ref}}| < \epsilon_2$ para valores apropriados de ϵ e valores de referência $\mathbf{u}_{\text{ref}}$ e $\mathbf{G}_{\text{ref}}$.

A equação (1.16) mostra que as únicas células não–nulas da matrix Jacobiana $\dfrac{\partial \mathbf{G}}{\partial \mathbf{u}}$ são

$$\begin{aligned}
\mathbf{C}_i &= \frac{\partial \mathbf{G}_i}{\partial \mathbf{u}_{i+1}} &&= \frac{\mathbf{f}'_{i+1}}{4h} - \frac{\mathbf{g}'_{i+1}(\mathbf{u}_{i+1} - \mathbf{u}_i) + \mathbf{g}_{i+1} + \mathbf{g}_i}{4h^2};\\
\mathbf{A}_i &= \frac{\partial \mathbf{G}_i}{\partial \mathbf{u}_i} &&= \frac{\mathbf{h}'_i}{k} - \frac{\mathbf{g}'_i(\mathbf{u}_{i+1} - 2\mathbf{u}_i + \mathbf{u}_{i-1}) - \mathbf{g}_{i+1} - 2\mathbf{g}_i - \mathbf{g}_{i-1}}{4h^2} - \frac{\mathbf{q}'_i}{2};\\
\mathbf{B}_i &= \frac{\partial \mathbf{G}_i}{\partial \mathbf{u}_{i-1}} &&= -\frac{\mathbf{f}'_{i-1}}{4h} - \frac{\mathbf{g}'_{i-1}(\mathbf{u}_{i-1} - \mathbf{u}_i) + \mathbf{g}_i + \mathbf{g}_{i-1}}{4h^2}.
\end{aligned} \tag{1.20}$$

Se $\mathbf{h}(\mathbf{u})$, $\mathbf{f}(\mathbf{u})$, $\mathbf{g}(\mathbf{u})$, $\mathbf{h}'(\mathbf{u})$, $\mathbf{f}'(\mathbf{u})$ e $\mathbf{g}'(\mathbf{u})$ são conhecidos, a matrix de coeficientes dada por (1.20) e o lado direito $\mathbf{G}(\mathbf{u})$ podem ser calculados, de forma que $\delta\mathbf{u}$ pode ser determinado resolvendo o sistema

$$\begin{bmatrix}
\mathbf{A}_2 & \mathbf{C}_2 & 0 & 0 & 0 & \cdots & & & \\
\mathbf{B}_3 & \mathbf{A}_3 & \mathbf{C}_3 & 0 & 0 & 0 & \cdots & & \\
0 & \mathbf{B}_4 & \mathbf{A}_4 & \mathbf{C}_4 & 0 & 0 & 0 & \cdots & \\
0 & 0 & \mathbf{B}_5 & \mathbf{A}_5 & \mathbf{C}_5 & 0 & 0 & 0 & \cdots \\
 & & & \ddots & \ddots & \ddots & & & \\
\cdots & 0 & 0 & 0 & \mathbf{B}_{N-4} & \mathbf{A}_{N-4} & \mathbf{C}_{N-4} & 0 & 0 \\
 & \cdots & 0 & 0 & 0 & \mathbf{B}_{N-3} & \mathbf{A}_{N-3} & \mathbf{C}_{N-3} & 0 \\
 & & \cdots & 0 & 0 & 0 & \mathbf{B}_{N-1} & \mathbf{A}_{N-2} & \mathbf{C}_{N-2} \\
 & & & \cdots & 0 & 0 & 0 & \mathbf{B}_{N-1} & \mathbf{A}_{N-1}
\end{bmatrix}
\begin{bmatrix}
\delta\mathbf{u}_2 \\ \delta\mathbf{u}_3 \\ \delta\mathbf{u}_4 \\ \delta\mathbf{u}_5 \\ \vdots \\ \delta\mathbf{u}_{N-4} \\ \delta\mathbf{u}_{N-3} \\ \delta\mathbf{u}_{N-2} \\ \delta\mathbf{u}_{N-1}
\end{bmatrix}
=
\begin{bmatrix}
-\mathbf{G}_2 \\ -\mathbf{G}_3 \\ -\mathbf{G}_4 \\ -\mathbf{G}_5 \\ \vdots \\ -\mathbf{G}_{N-4} \\ -\mathbf{G}_{N-3} \\ -\mathbf{G}_{N-2} \\ -\mathbf{G}_{N-1}
\end{bmatrix}, \tag{1.21}$$

que é *bloco tridiagonal*. A estimativa inicial $\mathbf{u}^{(0)}$ para a raiz $\mathbf{u}^{n+1}$ pode ser o valor $\mathbf{u}^n$ do passo de tempo anterior, ou a condição inicial $\mathbf{u}^0$ no caso do primeiro passo.

Para escolhas apropriadas de k e h, a matrix da equação (1.21) é diagonalmente dominante. Embora esta condição seja suficiente para garantir invertibilidade, ela é excessivamente restritiva e desnecessária. Limitar o valor de k em relação ao de h para garantir a dominância diagonal neste método implícito é equivalente à restrição de CFL dos métodos explícitos, que limita o valor de k em proporção direta ao de h para garantir convergência: e a principal vantagem de usar um método implícito é a possibilidade de empregar valores maiores para k. Felizmente, muitos problemas levam a matrizes que são invertíveis mesmo para escolhas de k e h que não garantem dominância diagonal. Vale mencionar também que em muitas aplicações práticas basta realizar uma única atualização a partir da estimativa inicial, ou seja, as operações descritas na equação (1.19) são realizadas uma única vez por passo de tempo. O algoritmo não–iterativo resultante é estudado em [**4**], e é aplicável a modelos cuja não–linearidade é fraca.

CAPíTULO 2

Sistemas bloco tridiagonais

Os sistemas lineares obtidos no primeiro capítulo têm uma estrutura particular, bloco tridiagonal, que sugere estratégias especializadas de solução. Se um alto desempenho paralelo também for requerido, os algoritmos de solução ficam restritos a um subconjunto ainda menor. Neste capítulo, estudamos algoritmos para a solução em paralelo desta classe de sistemas lineares em particular.

Na Seção 1 revisamos os trabalhos relacionados na literatura técnica e científica. Na Seção 2, apresentamos o algoritmo básico para a solução de sistemas tridiagonais, que é uma versão particular da fatoração LU. Começamos formalizando o caso escalar, depois o estendemos para o caso de blocos. Este algoritmo não se presta à paralelização: nas Seções 3 e 4, nós apresentamos dois algoritmos mais sofisticados que podem ser paralelizados. Na Seção 5, realizamos uma detalhada análise de complexidade dos três algoritmos; os valores teóricos do desempenho relativo dos três algoritmos são comparados a resultados experimentais na Seção 6. Discutimos os resultados na Seção 7.

1. Trabalhos relacionados

A solução em paralelo de sistemas lineares foi amplamente estudada nas últimas décadas, e a literatura relacionada é extensa o bastante para justificar resenhas ocasionais para sumarizá–la ([**1, 2, 15, 20**]). Muito esforço de pesquisa foi investido na solução de sistemas lineares com estruturas particulares, e algoritmos específicos foram propostos para sistemas em banda ([**2, 20**]), banda estreita ([**1**]), tridiagonais ([**5, 8, 22**]) e bloco tridiagonais([**9, 18**]). Muitos algoritmos apóiam-se ou aplicam-se apenas a matrizes com propriedades especiais como simetria ou dominância diagonal ([**2, 5, 9, 11**]), propriedades estas que as matrizes advindas do problema geral apresentado no Capítulo 1 podem não apresentar. Assim, os

algoritmos que estudamos neste trabalho derivam daqueles apresentados em [**18, 19, 27**], que não requerem tais propriedades.

Juntamente com o estudo teórico, bastante investigação experimental foi conduzida para avaliar o desempenho em paralelo dos diversos algoritmos propostos. Estes experimentos refletem a combinação de um algoritmo, uma implementação e um ambiente computacional conjuntamente, e são portanto muito difíceis de comparar entre si. Ambientes paralelos diferentes impõem restrições também diferentes na escolha de algoritmo e nas decisões quanto à sua implementação. Resultados de testes de desempenho em diversos ambientes de memória distribuída ([**1, 2, 11, 18**]), máquinas vetoriais ([**9, 31, 32**]) e arquiteturas híbridas ([**5**]) constam da literatura técnica. Em particular, sistemas de memória compartilhada de alto desempenho foram usados nos testes mais recentes, como os supercomputadores da Intel iPSC ([**18**]) e Paragon ([**1, 2**]), ou o IBM SP/2 ([**1, 11**]), todos com trinta e dois ou mais processadores. Por outro lado, a maior parte do trabalho experimental com clusters pequenos foi realizada há mais de quinze anos ([**5, 13, 14**]), e está tecnologicamente ultrapassada: estes trabalhos datam de um momento em que os clusters considerados pequenos pelos padrões de hoje eram os maiores disponíveis.

Os vários algoritmos apresentados na literatura são todos equivalentes em algum sentido ([**1**]), mas suas formulações são bastante diferentes. Formulações diferentes levam a diferentes obstáculos à implementação e conseqüentemente a diferentes desempenhos; alguns algoritmos exibem melhor desempenho que outros para certos problemas ou certas arquiteturas de computadores ([**1, 2, 15**]). Um algoritmo bem difundido e advogado ([**15, 17**]) para a solução paralela de sistemas bloco tridiagonais, conhecido como *redução cíclica*, foi apresentado pela primeira vez no contexto de computação paralela por Buzbee et alii em 1970 ([**8**]). A técnica foi explorada com sucesso para a criação de códigos eficientes em máquinas vetorias cuja velocidade vetorial era significativamente maior que a escalar ([**15**]). O algoritmo permite, entretanto, diferentes estratégias de paralelização, e aqui estudamos seu desempenho em clusters de computadores de baixo custo.

Muitos outros algoritmos foram desenvolvidos independentemente baseados na abordagem conceitualmente simples para a solução de problemas grandes conhecida por *divisão–e–conquista*. Se um problema grande é considerado difícil apenas por causa de seu tamanho, então quebra–se o problema em outros menores e portanto mais fáceis, e em seguida usa–se a solução dos subproblemas para resolver o todo original. O advento da computação paralela tornou esta abordagem atraente sob outro ponto de vista, pois estes subproblemas podem obviamente ser resolvidos em paralelo. Todo um campo da teoria formal de algoritmos desenvolveu–se em torno deste conceito: uma visão geral pode ser encontrada em [**6**]. Em problemas numéricos, a aplicação do conceito pode ser identificada ainda nos primórdios da computação ([**21**]), e alastrou–se pelas áreas de estudo envolvendo simulações numéricas de grandes dimensões ([**16, 28**]).

Indivíduos de diferentes formações técnicas reconhecerão o algoritmo de divisão–e–conquista apresentado na Seção 4 como uma variante de *nested dissection* ou *decomposição de domínio*. O algoritmo apresentado aqui assemelha–se ao algoritmo mais geral sugerido por Mehrmann em [**27**], onde sua relação com decomposição de domínio é discutida. A analogia com nested dissection é clara em [**20**], onde as mesmas idéias são formalizadas no contexto de teoria de grafos, de forma que o algoritmo pode ser encaixado também nesta classe. Para formulações semelhantes à apresentada Seção 4, ver [**11**], que enfatiza permutações de linhas e colunas, e [**20**], que enfatiza bipartição recursiva. Nosso algoritmo é uma construção simples de dois níveis apoiada em qualquer outro método para a solução de sistemas tridiagonais, assim como o algoritmo em [**27**].

Pode-se alegar que qualquer abordagem à solução em paralelo de um sistema linear é uma abordagem do tipo divisão–e–conquista: uma vez que a solução requer trocas globais de informações em algum momento, qualquer algoritmo que encontre (parte de) a solução em paralelo irá essencialmente "dividir" o problema por um dado número de elementos de processamento e agregar estas soluções parciais para "conquistar" o desafio original. Em particular, o algoritmo de redução cíclica apresentado na Seção 3 divide recursivamente o problema em dois problemas diferentes: um problema é idêntico ao original, com metade do

tamanho, e o outro é resolvido em parte durante a fase de "divisão" — o passo de redução, com a construção da fatoração LU parcial — e em parte durante a fase de "conquista" — o passo de retro–substituição, depois que a solução do sistema com as equações pares tiver sido obtida. Uma vez que esta taxonomia é encontrada na literatura a respeito da redução cíclica ([**1**]), nós enfatizamos a definição de divisão–e–conquista dada na página 105 de [**6**]:

> Divide-and-conquer is a technique for designing algorithms that consists of decomposing the instance to be solved into a number of smaller subinstances of the same problem, solving successively and independently each of these subinstances, and then combining the subsolutions thus obtained in such a way as to obtain the solution of the original instance.
>
> *[Divisão–e–conquista é uma técnica para concepção de algoritmos que consiste na decomposição da instância a ser resolvida num número de subinstâncias menores do mesmo problema, na resolução sucessiva e independente de cada uma destas subinstâncias, e por fim na combinação das subsoluções assim obtidas de forma a obter a solução da instância original.]*

Tal definição não engloba o algoritmo de redução cíclica na forma como é apresentado aqui e em [**18**]. Em vez disso, divisão–e–conquista é uma abordagem à concepção de um algoritmo, que pode ou não ser recursivo, mas no qual todos os subproblemas são similares àquele que os originou. Note-se que a ressalva quanto às subinstâncias serem resolvidas "sucessivamente", conforme na definição acima, só é relevante num contexto não–paralelo, que é o contexto da referência [**6**].

A matriz de permutação usada no algoritmo de redução cíclica pode, entretanto, ser usada como ponto de partida para um algoritmo de divisão–e–conquista, uma vez que leva à bipartição de um sistema em dois subsistemas que podem ser resolvidos independentemente ([**17**]). Esta bipartição requer a computação explícita dos complementos de Schur e introduz bastante computação desnecessária.

Estes são os dois algoritmos mais difundidos para a solução de sistemas tridiagonais em paralelo. Surpreendentemente, houve pouco interesse em comparar seu desempenho

nos últimos anos. Embora talvez seja verdade que toda comparação teórica já tenha sido feita e publicada, tivemos dificuldade em usar a literatura disponível para orientar-nos na tarefa supostamente simples de escolher qual algoritmo empregar na solução dos sistemas apresentados no Capítulo 1.

2. Sistemas tridiagonais

Considere um sistema linear de N equações com matriz de coeficientes $\mathbf{A}$, vetor de incógnitas $\mathbf{x}$ e lado direito conhecido $\mathbf{y}$, ou seja,

$$\mathbf{A}\mathbf{x} = \mathbf{y}, \quad \mathbf{A} \in \mathbb{R}^{N\times N}, \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^N, \tag{2.1}$$

no qual as únicas células não–nulas da matriz $\mathbf{A}$ são as da diagonal principal e das duas diagonais vizinhas. Seguindo a notação de [**19**], a matriz $\mathbf{A}$ é da forma

$$\mathbf{A} = \begin{bmatrix} a_1 & c_1 & & & \\ b_2 & a_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{N-1} & a_{N-1} & c_{N-1} \\ & & & b_N & a_N \end{bmatrix}. \tag{2.2}$$

A fatoração LU, quando aplicada a este tipo de matriz, fica bastante simplificada: torna–se um algoritmo linear, quando é quadrático para uma matriz qualquer. Trivialmente, obtém–se

$$\mathbf{A} = \mathbf{L}\mathbf{U} = \begin{bmatrix} \overline{a}_1 & & & & \\ b_2 & \overline{a}_2 & & & \\ & \ddots & \ddots & & \\ & & b_{N-1} & \overline{a}_{N-1} & \\ & & & b_N & \overline{a}_N \end{bmatrix} \begin{bmatrix} 1 & \overline{c}_1 & & & \\ & 1 & \overline{c}_2 & & \\ & & \ddots & \ddots & \\ & & & 1 & \overline{c}_{N-1} \\ & & & & 1 \end{bmatrix}. \tag{2.3}$$

com

$$\begin{array}{l} \overline{a}_1 = a_1,\ \overline{c}_1 = \dfrac{1}{\overline{a}_1}\, c_1, \\ \left\{ \begin{array}{l} \overline{a}_i = a_i - b_i \overline{c}_{i-1} \\ \overline{c}_i = \dfrac{1}{\overline{a}_i}\, c_i \end{array} \right. , \qquad i = 2, \ldots, N-1, \\ \overline{a}_N = a_N - b_N \overline{c}_{N-1}. \end{array} \tag{2.4}$$

A solução do sistema (2.1) pode ser computada por retro–substituição padrão, resolvendo

$$\mathbf{L}\mathbf{z} = \mathbf{y} \quad \text{e depois} \quad \mathbf{U}\mathbf{x} = \mathbf{z},$$

ou mais especificamente

$$\begin{array}{rcll} z_1 &=& \dfrac{y_1}{\overline{a}_1} & \\ z_i &=& \dfrac{y_i - b_i z_{i-1}}{\overline{a}_i}, & i = 2, \ldots, N, \end{array}$$

e

$$\begin{array}{rcll} x_N &=& z_N & \\ x_i &=& z_i - \overline{c}_i x_{i+1}, & i = N-1, \ldots, 1. \end{array} \tag{2.5}$$

2.1. Sistemas bloco tridiagonais. Se as células de $\mathbf{A}$ forem elas próprias matrizes $M \times M$, então o sistema tem tamanho NM, ou seja,

$$\mathbf{A}\mathbf{x} = \mathbf{y}, \quad \mathbf{A} \in \mathbb{R}^{NM \times NM}, \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^{NM}, \tag{2.6}$$

e pode ser visto como *bloco* tridiagonal, ou seja, ao coeficientes $\mathbf{A}_{ij}$ da matriz $\mathbf{A} = [\mathbf{A}_{ij}]$ são eles mesmos matrizes $M \times M$, logo a matriz da equação (1.21) tem a forma

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{C}_1 & & & \\ \mathbf{B}_2 & \mathbf{A}_2 & \mathbf{C}_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \mathbf{B}_{N-1} & \mathbf{A}_{N-1} & \mathbf{C}_{N-1} \\ & & & \mathbf{B}_N & \mathbf{A}_N \end{bmatrix}. \tag{2.7}$$

O mesmo procedimento da subseção anterior pode ser imediatamente aplicado à sua solução: a matriz $\mathbf{A}$ admite uma fatoração análoga à dada por (2.3)

$$\mathbf{A} = \mathbf{L}\mathbf{U} = \begin{bmatrix} \overline{\mathbf{A}}_1 & & & & \\ \mathbf{B}_2 & \overline{\mathbf{A}}_2 & & & \\ & \ddots & \ddots & & \\ & & \mathbf{B}_{N-1} & \overline{\mathbf{A}}_{N-1} & \\ & & & \mathbf{B}_N & \overline{\mathbf{A}}_N \end{bmatrix} \begin{bmatrix} \mathbf{I} & \overline{\mathbf{C}}_1 & & & \\ & \mathbf{I} & \overline{\mathbf{C}}_2 & & \\ & & \ddots & \ddots & \\ & & & \mathbf{I} & \overline{\mathbf{C}}_{N-1} \\ & & & & \mathbf{I} \end{bmatrix},$$

com

$$\begin{array}{l} \overline{\mathbf{A}}_1 = \mathbf{A}_1,\ \overline{\mathbf{C}}_1 = \overline{\mathbf{A}}_1^{-1}\mathbf{C}_1, \\ \left\{ \begin{array}{l} \overline{\mathbf{A}}_i = \mathbf{A}_i - \mathbf{B}_i\overline{\mathbf{C}}_{i-1} \\ \overline{\mathbf{C}}_i = \overline{\mathbf{A}}_i^{-1}\mathbf{C}_i \end{array} \right., \qquad i = 2, \ldots, N-1, \\ \overline{\mathbf{A}}_N = \mathbf{A}_N - \mathbf{B}_N\overline{\mathbf{C}}_{N-1}. \end{array} \tag{2.8}$$

A solução pode ser obtida de forma semelhante à dada em (2.5), usando

$$\begin{array}{rcll} \mathbf{z}_1 &=& \overline{\mathbf{A}}_1^{-1}\mathbf{y}_1 & \\ \mathbf{z}_i &=& \overline{\mathbf{A}}_i^{-1}(\mathbf{y}_i - \mathbf{B}_i\mathbf{z}_{i-1}), & i = 2, \ldots, N, \end{array}$$

e

$$\begin{array}{rcll} \mathbf{x}_N &=& \mathbf{z}_N & \\ \mathbf{x}_i &=& \mathbf{z}_i - \overline{\mathbf{C}}_i\mathbf{x}_{i+1}, & i = N-1, \ldots, 1, \end{array} \tag{2.9}$$

onde $\mathbf{x}_i$, $\mathbf{y}_i$ e $\mathbf{z}_i$, com $i = 1, \ldots, N$, são vetores em $\mathbb{R}^M$ que correspondem a cada linha de blocos do sistema completo.

Um sistema bloco tridiagonal com blocos de tamanho M pode ser visto como um sistema em banda com bandas direita e esquerda

$$k_r = k_l = 2M - 1; \tag{2.10}$$

da mesma forma, qualquer sistema em banda tal que $\max(k_r, k_l) = K$ pode ser visto como um sistema bloco tridiagonal com blocos de tamanho $M = \lceil (K+1)/2 \rceil$. A aplicação de um algoritmo que resolve sistemas em banda quaisquer a um sistema bloco tridiagonal é possível, mas menos eficiente, pois tal algoritmo não tira tanta vantagem do padrão bem definido de zeros na matriz de coeficientes. Para N grande, o algoritmo específico aqui apresentado executa aproximadamente 40% menos operações de ponto flutuante quando comparado ao algoritmo para sistemas em banda quaisquer ([**1**]). Além disso, a aplicação de algoritmos para solução de sistemas em banda gerais pode levar a pivoteamento desnecessário e problemas numéricos: por exemplo, algoritmos que não levam em consideração a estrutura em blocos podem pivotear as equações escalares que compõem cada equação em bloco.

2.2. Paralelizando a solução. Um rápido exame das equações (2.8) e (2.9) mostra que este algoritmo não é paralelizável, pois cada operação matricial depende do resultado obtido da operação imediatamente anterior. Isto vale para eliminação Gaussiana de sistemas em banda, e portanto para fatoração LU também.

Para distribuir o procedimento de solução entre elementos de processamento separados, é necessário introduzir cálculos redundantes. Esta *redundância* não é uma mera replicação de cálculos em cada elemento de processamento: é, em vez disso, um trabalho adicional realizado por uma abordagem diferente, mais custosa, para a solução do mesmo problema. Este trabalho adicional é justificado se o paralelismo por ele viabilizado for suficiente para reduzir o tempo total de computação. O *fator de redundância* de um algoritmo paralelo é calculado em relação a um algoritmo serial, geralmente o mais econômico, pela expressão

$$R = \frac{O_p}{O_s}, \tag{2.11}$$

onde O_p é o número de operações necessárias para resolver o problema em paralelo, e O_s é o número de operações necessárias para resolver o problema seqüencialmente [**29**]. Nos cálculos de fatores de redundância das seções seguintes, tomaremos O_s como a contagem de operações do algoritmo que foi apresentado nesta seção, ao qual nos referiremos como algoritmo *serial* nas discussões futuras.

Sistemas lineares podem ser resolvidos em um ou dois estágios. A abordagem de um estágio recebe ambos a matriz $\mathbf{A}$ e o lado direito $\mathbf{y}$ como dados de entrada, e gera a solução $\mathbf{x}$ como saída; se o sistema precisar ser resolvido para um novo $\mathbf{y}$, pode ser necessário repetir o processo inteiro com ambos $\mathbf{A}$ original e este novo $\mathbf{y}$. A abordagem de dois estágios tem apenas a matriz $\mathbf{A}$ como entrada no primeiro estágio, e produz uma fatoração que pode ser usada posteriormente para qualquer $\mathbf{y}$ dado. O segundo estágio, que é computacionalmente muito menos custoso, usa esta fatoração e $\mathbf{y}$ para encontrar a solução. Entretanto, num ambiente paralelo de memória distribuída a abordagem em dois estágios requer duas comunicações globais (agregação e broadcast), uma para a fatoração e uma para a retro–substituição. A aplicação que temos em vista, a solução de sistemas não–lineares pelo método de Newton conforme a equação (1.19), faz da abordagem com um único estágio a escolha inequívoca: cada sistema terá ambos sua matriz de coeficientes e seu lado direito computados (ou seja, disponíveis) no mesmo ponto da linha de execução, e será resolvido uma única vez. Nós abordaremos os dois algoritmos paralelizáveis nas próximas seções como algoritmos de um estágio.

O desempenho de um algoritmo paralelo depende do número de processadores pelos quais ele é distribuído. Para avaliar o desempenho de um algoritmo paralelo que leva $T_{\text{par}}(\mathcal{P}, P)$ unidades de tempo para executar $\mathcal{P}$ processos em P processadores, podemos comparar este valor ao tempo $T_{\text{par}}(\mathcal{P}, 1)$ consumido pelo mesmo algoritmo paralelo executando os mesmos $\mathcal{P}$ processos num único processador, ou ao tempo T_{ser} consumido pelo algoritmo serial para resolver o mesmo problema. As razões $T_{\text{par}}(\mathcal{P}, 1)/T_{\text{par}}(\mathcal{P}, P)$ e $T_{\text{ser}}/T_{\text{par}}(\mathcal{P}, P)$ são conhecidas como *speedup relativo* e *absoluto*, respectivamente.

O speedup relativo reflete apenas a escalabilidade de um algoritmo paralelo, ou seja, sua capacidade de diminuir o tempo total de execução em proporção direta ao número de processadores empregado. Entretanto, a redundância inerente ao algoritmo paralelo não é levada em consideração. O speedup absoluto reflete o ganho real do algoritmo paralelo em comparação ao algoritmo serial: ele é igual ao speedup relativo dividido pelo fator de redundância. Costuma ocorrer do speedup absoluto ser menor que um para um número pequeno de processadores, ou seja, o algoritmo paralelo possui tanta redundância que torna–se mais lento que o algoritmo serial a menos que muitos processadores sejam empregados. Isto não significa que o algoritmo paralelo deva ser descartado: ele pode ser muito útil se o problema for grande o bastante e houver processadores suficientes para atingir um speedup absoluto que compense.

Ambos o speedup e o fator de redundância podem ser inferidos teoricamente, mas se por um lado a redundância teórica pode ser comparada a tomadas de tempo de execução com relativa facilidade, os speedups teóricos podem quando muito ser encarados como limites superiores, pois eles requerem a modelagem do tempo de comunicação entre as unidades de processamento. Na Seção 5 analisamos a complexidade computacional dos algoritmos apresentados neste capítulo de forma a comparar suas redundâncias com os respectivos tempos de execução na Seção 6; como não incluímos um modelo teórico para os tempos de comunicação, apresentamos apenas os speedups aferidos experimentalmente.

3. Redução cíclica

Apresentamos este algoritmo segundo a forma mais procedural dada nas referências [**17**] e [**18**], e seguimos a notação de [**19**]. Como na seção anterior, primeiro introduzimos a versão escalar do algoritmo, isto é, o algoritmo para resolver sistemas tridiagonais escalares; depois estendemo–lo para a solução de sistemas bloco tridiagonais, e finalmente discutimos questões de paralelização.

3.1. Redução cíclica para sistemas escalares. Considere novamente um sistema tridiagonal como o da equação (2.1) que pode ser resolvido por eliminação Gaussiana sem pivoteamento. Esta propriedade se mantém quando aplicamos uma matriz de permutação $\mathbf{P}$ de forma a obter um sistema equivalente

$$\mathbf{PAP}^{\mathrm{T}}\mathbf{Px} = \mathbf{Py}.$$

A matriz de permutação a ser usada é dada por

$$\mathbf{P} = [p_{ij}]; \quad p_{ij} = \begin{cases} 1 & \text{para} \quad i = \lceil j/2 \rceil, \ \text{para } j \text{ ímpar}, \\ 1 & \text{para} \quad i = \left\lceil \dfrac{j+N}{2} \right\rceil, \ \text{para } j \text{ par, e} \\ 0 & \text{em todo o resto.} \end{cases}$$

onde $\lceil x \rceil$ indica o menor inteiro que é maior ou igual a x. Para fixar idéias, supomos que N é par e apresentamos uma matriz–exemplo $\mathbf{A}$ de tamanho 8×8 com sua correspondente $\mathbf{PAP}^{\mathrm{T}}$:

$$\mathbf{A} = \begin{bmatrix} a_1 & c_1 & & & & & & \\ b_2 & a_2 & c_2 & & & & & \\ & b_3 & a_3 & c_3 & & & & \\ & & b_4 & a_4 & c_4 & & & \\ & & & b_5 & a_5 & c_5 & & \\ & & & & b_6 & a_6 & c_6 & \\ & & & & & b_7 & a_7 & c_7 \\ & & & & & & b_8 & a_8 \end{bmatrix}; \quad \mathbf{PAP}^{\mathrm{T}} = \left[\begin{array}{cccc:cccc} a_1 & & & & c_1 & & & \\ & a_3 & & & b_3 & c_3 & & \\ & & a_5 & & & b_5 & c_5 & \\ & & & a_7 & & & b_7 & c_7 \\ \hdashline b_2 & c_2 & & & a_2 & & & \\ & b_4 & c_4 & & & a_4 & & \\ & & b_6 & c_6 & & & a_6 & \\ & & & b_8 & & & & a_8 \end{array}\right]. \tag{2.12}$$

Repare na separação da diagonal principal em blocos par e ímpar: o efeito final de aplicar $\mathbf{P}$ à esquerda de $\mathbf{A}$ é mover todas as linhas pares para baixo das linhas ímpares; da mesma forma, aplicar $\mathbf{P}^{\mathrm{T}}$ à esquerda de $\mathbf{A}$ move todas as colunas pares para a direita das colunas ímpares. Podemos computar uma fatoração LU parcial da matriz $\mathbf{PAP}^{\mathrm{T}}$, obtendo

$$\mathbf{PAP}^{\mathrm{T}} = \left[\begin{array}{cccc:cccc} 1 & & & & 0 & & & \\ & 1 & & & & 0 & & \\ & & 1 & & & & 0 & \\ & & & 1 & & & & 0 \\ \hdashline \overline{b}_2 & \overline{c}_2 & & & 1 & & & \\ & \overline{b}_4 & \overline{c}_4 & & & 1 & & \\ & & \overline{b}_6 & \overline{c}_6 & & & 1 & \\ & & & \overline{b}_8 & & & & 1 \end{array}\right] \left[\begin{array}{cccc:cccc} a_1 & & & & c_1 & & & \\ & a_3 & & & b_3 & c_3 & & \\ & & a_5 & & & b_5 & c_5 & \\ & & & a_7 & & & b_7 & c_7 \\ \hdashline 0 & & & & \overline{a}_2 & \overline{e}_2 & & \\ & 0 & & & \overline{d}_4 & \overline{a}_4 & \overline{e}_4 & \\ & & 0 & & & \overline{d}_6 & \overline{a}_6 & \overline{e}_6 \\ & & & 0 & & & \overline{d}_8 & \overline{a}_8 \end{array}\right],$$

ou mais sinteticamente

$$\mathbf{PAP}^{\mathrm{T}} = \begin{bmatrix} \mathbf{A}_o & \mathbf{C} \\ \mathbf{B} & \mathbf{A}_e \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{W} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A}_o & \mathbf{V} \\ \mathbf{0} & \mathbf{A}_t \end{bmatrix}, \tag{2.13}$$

onde $\mathbf{V} \equiv \mathbf{C}$; os coeficientes de $\mathbf{W} = \mathbf{B}\mathbf{A}_o^{-1}$, dados por

$$\begin{aligned} \overline{b}_i &= b_i / a_{i-1}, \qquad i = 2, 4, \ldots, N, \\ \overline{c}_i &= c_i / a_{i+1}, \qquad i = 2, 4, \ldots, N-2, \end{aligned} \tag{2.14}$$

são usados para calcular os coeficientes de $\mathbf{A}_t$, dados por

$$\begin{aligned} \overline{d}_i &= -\overline{b}_i b_{i-1}, \qquad i = 4, 6, \ldots, N, \\ \overline{e}_i &= -\overline{c}_i c_{i+1}, \qquad i = 2, 4, \ldots, N-2, \\ \overline{a}_i &= a_i - \overline{c}_i b_{i+1} - \overline{b}_i c_{i-1}, \qquad i = 2, 4, \ldots, N-2, \\ \overline{a}_N &= a_N - \overline{b}_N c_{N-1}. \end{aligned} \tag{2.15}$$

A decomposição LU dada em (2.13) é dita parcial porque a matriz $\mathbf{A}_t$ em si não é triangular superior; entretanto, se levarmos em consideração apenas a estrutura de blocos, a decomposição é efetivamente LU. Reescrevendo a permutação do problema original $\mathbf{Ax} = \mathbf{y}$ na notação de blocos apresentada em (2.13), nós temos

$$\mathbf{PAP}^{\mathrm{T}}\mathbf{Px} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{W} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A}_o & \mathbf{V} \\ \mathbf{0} & \mathbf{A}_t \end{bmatrix} \begin{bmatrix} \mathbf{x}_o \\ \mathbf{x}_e \end{bmatrix} = \begin{bmatrix} \mathbf{y}_o \\ \mathbf{y}_e \end{bmatrix}. \tag{2.16}$$

Admitamos que o sistema $\mathbf{A}_t\mathbf{x}_e = \mathbf{y}_e$ possa ser resolvido, o que é válido se o sistema completo $\mathbf{Ax} = \mathbf{y}$ puder ([**17, 18**]); então a solução do sistema completo pode ser obtida aplicando retro–substituição padrão no nível de blocos, computando

$$\overline{\mathbf{y}}_o \equiv \mathbf{y}_o \quad \text{e} \quad \overline{\mathbf{y}}_e = \mathbf{y}_e - \mathbf{W}\overline{\mathbf{y}}_o, \tag{2.17}$$

e em seguida resolvendo os sistemas lineares

$$\mathbf{A}_t\mathbf{x}_e = \overline{\mathbf{y}}_e \quad \text{e} \quad \mathbf{A}_o\mathbf{x}_o = \overline{\mathbf{y}}_o - \mathbf{V}\mathbf{x}_e. \tag{2.18}$$

Agora o nome *redução cíclica* pode ser justificado. O procedimento descrito acima reduz o tamanho do sistema original à metade, eliminando todas as incógnitas de índice ímpar e gerando um novo sistema contendo as incóginitas de índice par. Este novo sistema é semelhante ao original, uma vez que também é tridiagonal e herda todas as propriedades como dominância diagonal e não-singularidade que este último possa ter ([**17**]): assim, o procedimento pode ser aplicado recursivamente ao novo sistema, e cada ciclo de redução reduz o tamanho do sistema pela metade. O procedimento recursivo é interrompido quando

o sistema dele resultante tem uma solução trivial. Neste caso simples, escalar, podemos considerar que a solução de um sistema 1×1 é trivial: se $N = 2^n$, então o algoritmo passa por exatamente n ciclos.

É evidente das equações precedentes que a redução cíclica requer mais esforço computacional que a fatoração LU direta apresentada anteriormente. O algoritmo só é atraente porque a computação da fatoração LU parcial, a cada ciclo de redução, pode ser paralelizada. Antes de discutir como isto é feito, nós generalizamos o procedimento para sistemas bloco tridiagonais.

3.2. Redução cíclica de sistemas bloco tridiagonais. Consideremos agora um sistema bloco tridiagonal como o da equação (2.6), que pode ser resolvido por eliminação Gaussiana sem pivoteamento. A matriz de permutação, a estrutura em blocos e fatoração parcial LU resultantes são semelhantes àquelas obtidas na subseção anterior: $\mathbf{P}$ é dada por

$$\mathbf{P} = [\mathbf{P}_{ij}]; \quad \mathbf{P}_{ij} = \begin{cases} \mathbf{I} & \text{para} \quad i = \lceil j/2 \rceil, \ \text{para } j \text{ ímpar}, \\ \mathbf{I} & \text{para} \quad i = \left\lceil \dfrac{j+N}{2} \right\rceil, \ \text{para } j \text{ par, e} \\ \mathbf{0} & \text{em todo o resto}, \end{cases}$$

de forma que

$$\mathbf{PAP}^{\mathrm{T}} = \left[\begin{array}{cccc:cccc} \mathbf{A}_1 & & & & \mathbf{C}_1 & & & \\ & \mathbf{A}_3 & & & \mathbf{B}_3 & \mathbf{C}_3 & & \\ & & \mathbf{A}_5 & & & \mathbf{B}_5 & \mathbf{C}_5 & \\ & & & \mathbf{A}_7 & & & \mathbf{B}_7 & \mathbf{C}_7 \\ \hdashline \mathbf{B}_2 & \mathbf{C}_2 & & & \mathbf{A}_2 & & & \\ & \mathbf{B}_4 & \mathbf{C}_4 & & & \mathbf{A}_4 & & \\ & & \mathbf{B}_6 & \mathbf{C}_6 & & & \mathbf{A}_6 & \\ & & & \mathbf{B}_8 & & & & \mathbf{A}_8 \end{array}\right],$$

e a fatoração parcial é dada por

$$\mathbf{PAP}^{\mathrm{T}} = \begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \mathbf{W} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{U} & \mathbf{V} \\ \mathbf{0} & \mathbf{A}_t \end{bmatrix} \tag{2.19}$$

$$= \left[\begin{array}{cccc:cccc} \mathbf{L}_1 & & & & \mathbf{0} & & & \\ & \mathbf{L}_3 & & & & \mathbf{0} & & \\ & & \mathbf{L}_5 & & & & \mathbf{0} & \\ & & & \mathbf{L}_7 & & & & \mathbf{0} \\ \hdashline \overline{\mathbf{B}}_2 & \overline{\mathbf{C}}_2 & & & \mathbf{I} & & & \\ & \overline{\mathbf{B}}_4 & \overline{\mathbf{C}}_4 & & & \mathbf{I} & & \\ & & \overline{\mathbf{B}}_6 & \overline{\mathbf{C}}_6 & & & \mathbf{I} & \\ & & & \overline{\mathbf{B}}_8 & & & & \mathbf{I} \end{array}\right] \left[\begin{array}{cccc:ccc} \mathbf{U}_1 & & & & \overline{\mathbf{C}}_1 & & \\ & \mathbf{U}_3 & & & \overline{\mathbf{B}}_3 & \overline{\mathbf{C}}_3 & \\ & & \mathbf{U}_5 & & & \overline{\mathbf{B}}_5 & \overline{\mathbf{C}}_5 \\ & & & \mathbf{U}_7 & & & \overline{\mathbf{B}}_7 \\ \hdashline \mathbf{0} & & & & \overline{\mathbf{A}}_2 & \overline{\mathbf{E}}_2 & \\ & \mathbf{0} & & & \overline{\mathbf{D}}_4 & \overline{\mathbf{A}}_4 & \overline{\mathbf{E}}_4 \\ & & \mathbf{0} & & & \overline{\mathbf{D}}_6 & \overline{\mathbf{A}}_6 \\ & & & \mathbf{0} & & & \overline{\mathbf{D}}_8 \end{array}\right]$$

onde $\mathbf{L}_i$ e $\mathbf{U}_i$ são os fatores LU de $\mathbf{A}_i$ para i ímpar, ou seja, $\mathbf{A}_i = \mathbf{L}_i\mathbf{U}_i$, e as outras submatrizes são dadas pelas expressões

$$\overline{\mathbf{B}}_i = \mathbf{L}_i^{-1}\mathbf{B}_i, \qquad i = 3, 5, \ldots, N-1, \tag{2.20}$$

$$\overline{\mathbf{B}}_i = \mathbf{B}_i\mathbf{U}_{i-1}^{-1}, \qquad i = 2, 4, \ldots, N, \tag{2.21}$$

$$\overline{\mathbf{C}}_i = \mathbf{L}_i^{-1}\mathbf{C}_i, \qquad i = 1, 3, \ldots, N-1, \tag{2.22}$$

$$\overline{\mathbf{C}}_i = \mathbf{C}_i\mathbf{U}_{i+1}^{-1}, \qquad i = 2, 4, \ldots, N-2, \tag{2.23}$$

$$\overline{\mathbf{D}}_i = -\overline{\mathbf{B}}_i\overline{\mathbf{B}}_{i-1}, \qquad i = 4, 6, \ldots, N, \tag{2.24}$$

$$\overline{\mathbf{E}}_i = -\overline{\mathbf{C}}_i\overline{\mathbf{C}}_{i+1}, \qquad i = 2, 4, \ldots, N-2, \tag{2.25}$$

$$\begin{aligned} \overline{\mathbf{A}}_i &= \mathbf{A}_i - \overline{\mathbf{B}}_i\overline{\mathbf{C}}_{i-1} - \overline{\mathbf{C}}_i\overline{\mathbf{B}}_{i+1}, \qquad i = 2, 4, \ldots, N-2, \\ \overline{\mathbf{A}}_N &= \mathbf{A}_N - \overline{\mathbf{B}}_N\overline{\mathbf{C}}_{N-1}. \end{aligned} \tag{2.26}$$

A estrutura em blocos de nível mais alto dada em (2.19) é diferente daquela na equação (2.16). Agora as matrizes $\mathbf{V}$ e $\mathbf{W}$ são computadas a partir dos fatores $\mathbf{L}$ e $\mathbf{U}$ das matrizes $\mathbf{A}$ de índice ímpar, respectivamente, de forma que a única analogia direta que pode ser estabelecida é entre as expressões (2.24)–(2.26) e aquelas em (2.15). Note que, se cada $\mathbf{A}_i$ for triangular superior, então $\mathbf{U}_i \equiv \mathbf{A}_i$ e $\mathbf{L}_i \equiv \mathbf{I}$, e as equações (2.20)–(2.23) tornam-se iguais àquelas em (2.14). Observe–se que as inversas de matrizes nas expressões (2.20)–(2.23) jamais são computadas, sendo os lados esquerdos obtidos como soluções de sistemas lineares triangulares. Usamos esta notação para enfatizar a nomenclatura.

O procedimento para recuperar a solução dada pelas equações (2.17)–(2.18) torna–se

$$\overline{\mathbf{y}}_o = \mathbf{L}^{-1}\mathbf{y}_o, \tag{2.27}$$

$$\overline{\mathbf{y}}_e = \mathbf{y}_e - \mathbf{W}\overline{\mathbf{y}}_o, \tag{2.28}$$

$$\mathbf{x}_e = \mathbf{A}_t^{-1}\overline{\mathbf{y}}_e, \tag{2.29}$$

$$\mathbf{x}_o = \mathbf{U}^{-1}(\overline{\mathbf{y}}_o - \mathbf{V}\mathbf{x}_e). \tag{2.30}$$

Conforme destacado no estudo de Arbenz ([**2**]), se o lado direito $\mathbf{y}$ é conhecido *a priori*, as expressões (2.27) e (2.28) podem ser computadas ao longo da fatoração. Isto pode ser usado para minimizar o uso de memória, descartando $\mathbf{L}$ e $\mathbf{W}$ da equação (2.19) ao fim de cada ciclo

de redução, o que é válido se o sistema não for ser resolvido para outro $\mathbf{y}$ posteriormente. Além disso, se o algoritmo for paralelizado, isto minimiza o número de comunicações, uma vez que todas as trocas de dados necessárias para a passo de retro–substituição são realizadas ao longo do passo de fatoração.

Será útil na discussão seguinte particionar e indexar os objetos $\overline{\mathbf{y}}$ definidos nas equações (2.27) e (2.28), de forma a relacioná-los com aqueles dados nas equações (2.20)–(2.26):

$$\overline{\mathbf{y}}_i = \mathbf{L}_i^{-1}\mathbf{y}_i, \qquad i = 1, 3, \ldots, N-1, \tag{2.31}$$

$$\overline{\mathbf{y}}_i = \mathbf{y}_i - \overline{\mathbf{B}}_i\overline{\mathbf{y}}_{i-1} - \overline{\mathbf{C}}_i\overline{\mathbf{y}}_{i+1}, \qquad i = 2, 4, \ldots, N-2, \tag{2.32}$$

$$\overline{\mathbf{y}}_N = \mathbf{y}_N - \overline{\mathbf{B}}_N\overline{\mathbf{y}}_{N-1}.$$

3.3. Paralelizando a solução. Denotamos por S_i o conjunto

$$S_i \equiv \{\mathbf{L}_i, \mathbf{U}_i, \overline{\mathbf{B}}_i\,\overline{\mathbf{B}}_{i+1}, \overline{\mathbf{C}}_i, \overline{\mathbf{C}}_{i-1}, \overline{\mathbf{D}}_{i+1}, \overline{\mathbf{E}}_{i-1}, \overline{\mathbf{y}}_i\}, \quad i = 1, 3, \ldots, N-1.$$

As expressões (2.20)–(2.26) revelam dependências de dados tais que a computação do conjunto S_i é desacoplada para cada i ímpar: o acoplamento inerente ao sistema linear está restrito à computação de $\overline{\mathbf{A}}_i$, dada pela equação (2.26), ou seja, a computação de cada S_i pode ser feita independentemente para cada valor de i, e a computação de $\overline{\mathbf{A}}_{i+1}$ e $\overline{\mathbf{y}}_{i+1}$ requer que ambos S_i e S_{i+2} já tenham sido computados, com exceção de $\overline{\mathbf{A}}_N$ e $\overline{\mathbf{y}}_N$ que só dependem de S_{N-1}.

A redução cíclica permite diferentes estratégias de paralelização, dependendo das características do ambiente computacional em vista. Ela pode, é claro, ser implementada numa única máquina seqüencial. No extremo oposto, pode–se considerar a paralelização de maior refinamento possível, onde $N/2$ elementos de processamento computam simultaneamente as expressões (2.20)–(2.25), e em seguida trocam dados com os elementos vizinhos para computar (2.26).

Considere primeiro a abordagem mais refinada: no primeiro ciclo de redução temos $N/2$ conjuntos S_i que podem ser alocados a $N/2$ elementos de processamento distintos. Se o algoritmo vier a realizar um novo ciclo de redução, entretanto, o número máximo de elementos de processamento cai pela metade junto com o tamanho do problema: apenas $N/4$

processadores podem ser usados. Num cenário realista, o número de processadores P é muito menor que N. Se insistirmos na abordagem de paralelismo refinado, o algoritmo pode implementar processadores virtuais e alocá-los a processadores reais [**18**]. Entretanto, isto é equivalente a alocar mais de um conjunto S_i a cada processador, resultando num paralelismo mais grosseiro e contradizendo a proposta original de paralelismo refinado.

A escolha de como dividir os S_i's pelos P processadores determina quanta comunicação será necessária. Se dois conjuntos vizinhos S_i e S_{i+2} estão alocados num mesmo processador, então o acoplamento entre eles é local a este processador e a computação de $\overline{\mathbf{A}}_{i+1}$ e $\overline{\mathbf{y}}_{i+1}$ não requer comunicação. Logo, os requerimentos de comunicação são minimizados se os $N/2$ conjuntos forem divididos em P partições contíguas, resultando em $P-1$ comunicações ao final de cada ciclo de redução. Uma escolha deve ser feita: se S_i e S_{i+2} estão alocados em diferentes processadores P e Q, então ou P deve enviar S_i para Q ou Q deve enviar S_{i+2} para P. Este segundo esquema mantém os dados igualmente distribuídos pelos elementos de processamento, o que é desejável, de forma que descartamos o outro. Além disso, não é necessário transmitir todo o conjunto S_i, mas apenas $\overline{\mathbf{C}}_i$, $\overline{\mathbf{B}}_{i+1}$ e $\overline{\mathbf{y}}_i$. O processador que detém S_1 não transmite para nenhum outro, e assim um total de $2(P-1)$ matrizes é transmitido ao fim de cada ciclo de redução. Note que o sistema reduzido mantém a contigüidade da partição de dados.

Para simplificar, admitimos que o tamanho N do problema tem a forma $2^n \times P$, onde $n \in \mathbb{N}$ e P é o número de elementos de processamento. Se um processador detém diversos conjuntos S, por exemplo, o segundo processador detém S_i, $i = N/P+1, \ldots, 2N/P-1$ no início do procedimento, ele pode computar as fórmulas (2.20)–(2.25) para cada um destes conjuntos em qualquer ordem, pois são tão independentes entre si quanto dos conjuntos S_i alocados a outros processadores. Assim sendo, $S_{N/P+1}$ pode ser processado primeiro e imediatamente transmitido para o primeiro processador: num ambiente paralelo de memória distribuída que permite comunicação assíncrona e não–bloqueante, isto significa que cada processador pode efetuar a maior parte de seu trabalho após enviar os dados de acoplamento.

Se N é grande o bastante para que a computação de todos os conjuntos S em cada processador demore mais que o tempo de transmissão, os custos de comunicação não têm nenhum reflexo no desempenho do algoritmo. Mas mesmo que esta hipótese seja válida no primeiro ciclo de redução, é possível que deixe de valer posteriormente, uma vez que N é dividido por dois a cada redução. Se N é da forma $2^n \times P$, após n reduções cada processador detém apenas um conjunto S, e o processo recursivo deve ser interrompido. Entretanto, é possível que seja vantajoso interromper este processo em algum nível $k < n$, devido a custos de comunicação. Dependendo da latência e da banda de transmissão, é possível determinar um valor crítico $k \leq n$ para o qual o tamanho do subproblema é tão pequeno que uma outra estratégia para sua solução torna–se vantajosa.

Qualquer que seja o valor $k \leq n$ usado, a fatoração e a solução deste ponto em diante requer comunicação global, para resolver um sistema linear menor, de tamanho $2^{n-k} \times P$. Isto pode ser feito seqüencialmente num elemento de processamento em particular. Da solução deste subproblema, a solução do sistema original pode ser obtida em paralelo de forma análoga ao processo de redução, sem nenhuma nova comunicação. Entretanto, a solução estará distribuída pelos P elementos de processamento. Isto não é um problema na aplicação que temos em vista, pois o algoritmo irá proceder iterativamente usando a mesma partição de dados: a agregação da solução completa para pós–processamento e visualização não tem impacto sobre o esquema numérico paralelo.

4. Divisão–e–conquista

Agora aplicamos esta técnica de forma direta à solução de sistemas bloco tridiagonais, e formalizamos um algoritmo simples e claro. Muitos algoritmos semelhantes foram apresentados na literatura, e uma comparação abrangente com o trabalho relacionado foi feita na Seção 1. Nossa formulação é semelhante àquela apresentada por Mehrmann ([**27**]), com a vantagem de seguir a notação de [**19**]; ter os três algoritmos escritos numa notação próxima permite ver claramente em quê eles diferem. Esta seção está estruturada como as duas anteriores. O algoritmo é apresentado num contexto escalar, depois para sistemas de blocos. Questões relativas à paralelização são discutidas em seguida.

4.1. Solução de sistemas tridiagonais escalares por divisão–e–conquista. Consideremos novamente o problema de resolver (2.1) com $N = KP+(P-1)$. Podemos agrupar as equações em P blocos de tamanho K da seguinte forma:

$$\mathbf{A} = \left[\begin{array}{ccccccc:ccc} a_1 & c_1 & & & & & & & \\ b_2 & a_2 & c_2 & & & & & & \\ & \ddots & \ddots & \ddots & & & & & \\ & & b_{K-1} & a_{K-1} & c_{K-1} & & & & \\ & & & b_K & a_K & c_K & & & \\ \hdashline & & & & b_{K+1} & a_{K+1} & c_{K+1} & & \\ & & & & & b_{K+2} & a_{K+2} & c_{K+2} & \\ & & & & & & \ddots & \ddots & \ddots \end{array}\right]; \tag{2.33}$$

na discussão a seguir, denotamos esta partição por

$$\left[\begin{array}{ccccccccccc} \mathbf{A}_1 & & & & & & & & & & \\ & & \theta_1 & & & & & & & & \\ & \gamma_1 & \lambda_1 & \varphi_1 & & & & & & & \\ & & \omega_1 & & & & & & & & \\ & & & & \mathbf{A}_2 & & & & & & \\ & & & & & \ddots & & & & & \\ & & & & & & \mathbf{A}_{P-1} & & & & \\ & & & & & & & & \theta_{P-1} & & \\ & & & & & & & \gamma_{P-1} & \lambda_{P-1} & \varphi_{P-1} & \\ & & & & & & & & \omega_{P-1} & & \\ & & & & & & & & & & \mathbf{A}_P \end{array}\right] \left[\begin{array}{c} \mathbf{x}_1 \\ \xi_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{P-1} \\ \xi_{P-1} \\ \mathbf{x}_P \end{array}\right] = \left[\begin{array}{c} \mathbf{y}_1 \\ \psi_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_{P-1} \\ \psi_{P-1} \\ \mathbf{y}_P \end{array}\right], \tag{2.34}$$

com $\mathbf{A}_p \in \mathbb{R}^{K\times K}$, $\mathbf{x}_p$, $\mathbf{y}_p \in \mathbb{R}^K$, $p = 1, \ldots, P$ e θ_p, λ_p, γ_p, φ_p, ω_p, $\xi_p, \psi_p \in \mathbb{R}$, $p = 1, \ldots, P-1$. As células de $\mathbf{A}$, $\mathbf{x}$ e $\mathbf{y}$ em (2.1) mapeiam–se nos elementos de (2.34) através de

$$\begin{array}{rl} \theta_p & \equiv c_{p\times(K+1)-1} \\ \lambda_p & \equiv a_{p\times(K+1)} \\ \gamma_p & \equiv b_{p\times(K+1)} \\ \varphi_p & \equiv c_{p\times(K+1)} \\ \omega_p & \equiv b_{p\times(K+1)+1} \\ \xi_p & \equiv x_{p\times(K+1)} \\ \psi_p & \equiv y_{p\times(K+1)} \end{array} \quad , \quad p = 1, \ldots, P-1; \tag{2.35}$$

o resto das células não–nulas de $\mathbf{A}$ estão contidos nas submatrizes $\mathbf{A}_p$, que também são tridiagonais, e o resto dos vetores $\mathbf{x}$ e $\mathbf{y}$ mapeiam–se em $\mathbf{x}_p$ e $\mathbf{y}_p$ analogamente, ou seja,

$$\mathbf{A}_p = [\mathbf{A}_{p,i,j}] = \begin{cases} \mathbf{A}_{p,i,i} = a_{(p-1)\times(K+1)+i}, & i = 1,\ldots,K \\ \mathbf{A}_{p,i,i+1} = c_{(p-1)\times(K+1)+i}, & i = 1,\ldots,K-1 \\ \mathbf{A}_{p,i,i-1} = b_{(p-1)\times(K+1)+i}, & i = 2,\ldots,K \\ \mathbf{A}_{p,i,j} = 0, \quad \text{nas demais posições}, \end{cases} \tag{2.36}$$

$$\mathbf{x}_p = [\mathbf{x}_{p,i}] = x_{(p-1)\times(K+1)+i}, \quad i = 1,\ldots,K, \tag{2.37}$$

$$\mathbf{y}_p = [\mathbf{y}_{p,i}] = y_{(p-1)\times(K+1)+i}, \quad i = 1,\ldots,K. \tag{2.38}$$

O sistema completo dado na equação (2.34) é resolvido em três estágios: primeiro, cada um dos P sistemas de K equações $\mathbf{A}_p\mathbf{x}_p = \mathbf{y}_p$ é resolvido independentemente, e as soluções obtidas são funções das incógnitas ξ_p; segundo, construímos a partir destas soluções um novo sistema menor, de tamanho $P-1$, e determinamos os valores das incógnitas ξ_p; finalmente, obtemos a solução do sistema completo compondo as soluções encontradas no segundo estágio com aquelas encontradas no primeiro.

Agora formalizamos o primeiro estágio: introduzindo os vetores em $\mathbb{R}^K$

$$\mathbf{u} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \text{e} \quad \mathbf{v} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix},$$

podemos escrever

$$\begin{array}{ll} \mathbf{A}_1\mathbf{x}_1 + \xi_1\theta_1\mathbf{v} = \mathbf{y}_1 & \rightarrow \mathbf{x}_1 = \mathbf{A}_1^{-1}(\mathbf{y}_1 - \xi_1\theta_1\mathbf{v}), \\ \xi_{p-1}\omega_{p-1}\mathbf{u} + \mathbf{A}_p\mathbf{x}_p + \xi_p\theta_p\mathbf{v} = \mathbf{y}_p & \rightarrow \mathbf{x}_p = \mathbf{A}_p^{-1}(\mathbf{y}_p - \xi_{p-1}\omega_{p-1}\mathbf{u} - \xi_p\theta_p\mathbf{v}), \quad p = 2,\ldots,P-1, \\ \xi_{P-1}\omega_{P-1}\mathbf{u} + \mathbf{A}_P\mathbf{x}_P = \mathbf{y}_P & \rightarrow \mathbf{x}_P = \mathbf{A}_P^{-1}(\mathbf{y}_P - \xi_{P-1}\omega_{P-1}\mathbf{u}). \end{array} \tag{2.39}$$

Definimos $\overline{\mathbf{y}}_p$, $\overline{\mathbf{u}}_p$, $\overline{\mathbf{v}}_p$ pela solução de sistemas lineares, ou seja,

$$\mathbf{A}_p\overline{\mathbf{y}}_p = \mathbf{y}_p, \qquad \mathbf{A}_p\overline{\mathbf{u}}_p = \omega_{p-1}\mathbf{u} \qquad \text{e} \qquad \mathbf{A}_p\overline{\mathbf{v}}_p = \theta_p\mathbf{v},$$

logo os vetores $\overline{\mathbf{y}}_p$, $\overline{\mathbf{u}}_p$ e $\overline{\mathbf{v}}_p$ podem ser computados usando qualquer método como fatoração LU de cada $\mathbf{A}_p$ e retro–substituição para cada lado direito $\mathbf{y}_p$, $\mathbf{u}$ e $\mathbf{v}$. De fato, uma vez que cada $\mathbf{A}_p$ é tridiagonal, o algoritmo apresentado na Seção 2 é a escolha óbvia. Agora as soluções para $\mathbf{x}_p$ em (2.39) podem ser escritas como funções de ξ_p, isto é,

$$\begin{array}{l} \mathbf{x}_1 = \overline{\mathbf{y}}_1 - \xi_1\overline{\mathbf{v}}_1, \\ \mathbf{x}_p = \overline{\mathbf{y}}_p - \xi_{p-1}\overline{\mathbf{u}}_p - \xi_p\overline{\mathbf{v}}_p, \quad p = 2,\ldots,P-1, \\ \mathbf{x}_P = \overline{\mathbf{y}}_P - \xi_{P-1}\overline{\mathbf{u}}_P. \end{array} \tag{2.40}$$

O segundo estágio é usar estas soluções para determinar as incógnitas ξ_p. Para cada linha de (2.34) que corresponde a um ψ_p, usamos o $\mathbf{x}_p$ definido em (2.40) para obter uma equação da forma

$$\gamma_p(\mathbf{v}^{\mathrm{T}}\mathbf{x}_p) + \lambda_p\xi_p + \varphi_p(\mathbf{u}^{\mathrm{T}}\mathbf{x}_{p+1}) = \psi_p, \quad p = 1, ..., P-1. \tag{2.41}$$

Cada solução $\mathbf{x}_p$ é uma função de ξ_p e ξ_{p-1}, exceto $\mathbf{x}_1$ e $\mathbf{x}_P$. Aplicando as expressões para $\mathbf{x}_p$ dadas em (2.40) a (2.41), obtemos

$$\begin{array}{l}
\gamma_1\mathbf{v}^{\mathrm{T}}\big(\overline{\mathbf{y}}_1 - \xi_1\overline{\mathbf{v}}_1\big) + \lambda_1\xi_1 + \varphi_1\mathbf{u}^{\mathrm{T}}\big(\overline{\mathbf{y}}_2 - \xi_1\overline{\mathbf{u}}_2 - \xi_2\overline{\mathbf{v}}_2\big) = \psi_1, \\
\gamma_p\mathbf{v}^{\mathrm{T}}\big(\overline{\mathbf{y}}_p - \xi_{p-1}\overline{\mathbf{u}}_p - \xi_p\overline{\mathbf{v}}_p\big) + \lambda_p\xi_p + \varphi_p\mathbf{u}^{\mathrm{T}}\big(\overline{\mathbf{y}}_{p+1} - \xi_p\overline{\mathbf{u}}_{p+1} - \xi_{p+1}\overline{\mathbf{v}}_{p+1}\big) = \psi_p, \ p = 2, \ldots, P-2, \\
\gamma_{P-1}\mathbf{v}^{\mathrm{T}}\big(\overline{\mathbf{y}}_{P-1} - \xi_{P-2}\overline{\mathbf{u}}_{P-1} - \xi_{P-1}\overline{\mathbf{v}}_{P-1}\big) + \lambda_{P-1}\xi_{P-1} + \varphi_{P-1}\mathbf{u}^{\mathrm{T}}\big(\overline{\mathbf{y}}_P - \xi_{P-1}\overline{\mathbf{u}}_P\big) = \psi_{P-1},
\end{array} \tag{2.42}$$

de onde derivamos os coeficientes de um novo sistema tridiagonal de $P-1$ equações

$$\begin{bmatrix}
\overline{\lambda}_1 & \overline{\varphi}_1 & & & \\
\overline{\gamma}_2 & \overline{\lambda}_2 & \overline{\varphi}_2 & & \\
 & \ddots & \ddots & \ddots & \\
 & & \overline{\gamma}_{P-2} & \overline{\lambda}_{P-2} & \overline{\varphi}_{P-2} \\
 & & & \overline{\gamma}_{P-1} & \overline{\lambda}_{P-1}
\end{bmatrix}
\begin{bmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_{P-2} \\ \xi_{P-1} \end{bmatrix}
=
\begin{bmatrix} \overline{\psi}_1 \\ \overline{\psi}_2 \\ \vdots \\ \overline{\psi}_{P-2} \\ \overline{\psi}_{P-1} \end{bmatrix}, \tag{2.43}$$

onde

$$\begin{array}{ll}
\overline{\lambda}_p = \lambda_p - \mathbf{v}^{\mathrm{T}}\gamma_p\overline{\mathbf{v}}_p - \varphi_p\mathbf{u}^{\mathrm{T}}\overline{\mathbf{u}}_{p+1}, & p = 1, \ldots, P-1, \\
\overline{\gamma}_p = -\gamma_p\mathbf{v}^{\mathrm{T}}\overline{\mathbf{u}}_p, & p = 2, \ldots, P-1, \\
\overline{\varphi}_p = -\varphi_p\mathbf{u}^{\mathrm{T}}\overline{\mathbf{v}}_{p+1}, & p = 1, \ldots, P-2, \\
\overline{\psi}_p = \psi_p - \gamma_p\mathbf{v}^{\mathrm{T}}\overline{\mathbf{y}}_p - \varphi_p\mathbf{u}^{\mathrm{T}}\overline{\mathbf{y}}_{p+1}, & p = 1, \ldots, P-1.
\end{array} \tag{2.44}$$

Note que os produtos internos envolvendo $\mathbf{u}$ e $\mathbf{v}$ são triviais, pois sua computação resume–se a isolar ou o primeiro ou o último elemento de um vetor, respectivamente.

O terceiro estágio é igualmente trivial: uma vez que o sistema (2.43) tiver sido resolvido, a solução do sistema completo é computada pelas expressões em (2.40).

4.2. Solução de sistemas bloco tridiagonais por divisão–e–conquista. Todas as fórmulas apresentadas na subseção anterior continuam valendo para o caso de blocos, mas nós as reescrevemos de forma a enfatizar as operações não–comutativas de matrizes. A

equação (2.34) passa a ser

$$\begin{bmatrix}
\mathbf{A}_1 & & & & & & & & & & \\
 & \boldsymbol{\Theta}_1 & & & & & & & & & \\
 \boldsymbol{\Gamma}_1 & \boldsymbol{\Lambda}_1 & \boldsymbol{\Phi}_1 & & & & & & & & \\
 & \boldsymbol{\Omega}_1 & & & & & & & & & \\
 & & & \mathbf{A}_2 & & & & & & & \\
 & & & & \ddots & & & & & & \\
 & & & & & \mathbf{A}_{P-1} & & & & & \\
 & & & & & & \boldsymbol{\Theta}_{P-1} & & & & \\
 & & & & & \boldsymbol{\Gamma}_{P-1} & \boldsymbol{\Lambda}_{P-1} & \boldsymbol{\Phi}_{P-1} & & & \\
 & & & & & & \boldsymbol{\Omega}_{P-1} & & & & \\
 & & & & & & & & \mathbf{A}_P &
\end{bmatrix}
\begin{bmatrix} \mathbf{x}_1 \\ \\ \boldsymbol{\xi}_1 \\ \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{P-1} \\ \\ \boldsymbol{\xi}_{P-1} \\ \\ \mathbf{x}_P \end{bmatrix}
=
\begin{bmatrix} \mathbf{y}_1 \\ \\ \boldsymbol{\psi}_1 \\ \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_{P-1} \\ \\ \boldsymbol{\psi}_{P-1} \\ \\ \mathbf{y}_P \end{bmatrix}, \tag{2.45}$$

com

$$\begin{array}{l}
\mathbf{A}_p \in \mathbb{R}^{KM\times KM}, \\
\mathbf{x}_p,\ \mathbf{y}_p \in \mathbb{R}^{KM}, \\
\boldsymbol{\Lambda}_p,\ \boldsymbol{\Theta}_p,\ \boldsymbol{\Gamma}_p,\ \boldsymbol{\Phi}_p,\ \boldsymbol{\Omega}_p \in \mathbb{R}^{M\times M},\ \text{e} \\
\boldsymbol{\xi}_p,\ \boldsymbol{\psi}_p \in \mathbb{R}^{M}.
\end{array}$$

Os vetores $\mathbf{u}$ e $\mathbf{v}$ correspondem agora a matrizes retangulares de tamanho $KM \times M$

$$\mathbf{U} = \begin{bmatrix} \mathbf{I} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix} \quad \text{e} \quad \mathbf{V} = \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{I} \end{bmatrix},$$

de forma que o análogo da equação (2.40) é

$$\begin{array}{l}
\mathbf{x}_1 = \overline{\mathbf{y}}_1 - \overline{\mathbf{V}}_1\boldsymbol{\xi}_1, \\
\mathbf{x}_p = \overline{\mathbf{y}}_p - \overline{\mathbf{U}}_p\boldsymbol{\xi}_{p-1} - \overline{\mathbf{V}}_p\boldsymbol{\xi}_p, \quad p = 2,\ldots,P-1, \\
\mathbf{x}_P = \overline{\mathbf{y}}_P - \overline{\mathbf{U}}_P\boldsymbol{\xi}_{P-1},
\end{array} \tag{2.46}$$

onde $\overline{\mathbf{y}}_p$, $\overline{\mathbf{U}}_p$, $\overline{\mathbf{V}}_p$ são computados como soluções de sistemas lineares de tamanho $KM \times KM$, ou seja,

$$\mathbf{A}_p\overline{\mathbf{y}}_p = \mathbf{y}_p, \qquad \mathbf{A}_p\overline{\mathbf{U}}_p = \mathbf{U}\boldsymbol{\Omega}_{p-1} \qquad \text{e} \qquad \mathbf{A}_p\overline{\mathbf{V}}_p = \mathbf{V}\boldsymbol{\Theta}_p.$$

A equação geral de acoplamento (2.41) torna–se

$$\boldsymbol{\Gamma}_p(\mathbf{V}^{\mathrm{T}}\mathbf{x}_p) + \boldsymbol{\Lambda}_p\boldsymbol{\xi}_p + \boldsymbol{\Phi}_p(\mathbf{U}^{\mathrm{T}}\mathbf{x}_{p+1}) = \boldsymbol{\psi}_p, \quad p = 1,...,P-1, \tag{2.47}$$

e as $P-1$ equações de blocos correspondentes a (2.42) passam a ser

$$\begin{aligned}
&\boldsymbol{\Gamma}_1\big[\mathbf{V}^{\mathrm{T}}(\overline{\mathbf{y}}_1-\overline{\mathbf{V}}_1\boldsymbol{\xi}_1)\big]+\boldsymbol{\Lambda}_1\boldsymbol{\xi}_1+\boldsymbol{\Phi}_1\big[\mathbf{U}^{\mathrm{T}}(\overline{\mathbf{y}}_2-\overline{\mathbf{U}}_2\boldsymbol{\xi}_1-\overline{\mathbf{V}}_2\boldsymbol{\xi}_2)\big]=\boldsymbol{\psi}_1,\\
&\boldsymbol{\Gamma}_p\big[\mathbf{V}^{\mathrm{T}}(\overline{\mathbf{y}}_p-\overline{\mathbf{U}}_p\boldsymbol{\xi}_{p-1}-\overline{\mathbf{V}}_p\boldsymbol{\xi}_p)\big]+\boldsymbol{\Lambda}_p\boldsymbol{\xi}_p\\
&\qquad+\boldsymbol{\Phi}_p\big[\mathbf{U}^{\mathrm{T}}(\overline{\mathbf{y}}_{p+1}-\overline{\mathbf{U}}_{p+1}\boldsymbol{\xi}_p-\overline{\mathbf{V}}_{p+1}\boldsymbol{\xi}_{p+1})\big]=\boldsymbol{\psi}_p,\quad p=2,\ldots,P-2,\\
&\boldsymbol{\Gamma}_{P-1}\big[\mathbf{V}^{\mathrm{T}}(\overline{\mathbf{y}}_{P-1}-\overline{\mathbf{U}}_{P-1}\boldsymbol{\xi}_{P-2}-\overline{\mathbf{V}}_{P-1}\boldsymbol{\xi}_{P-1})\big]+\boldsymbol{\Lambda}_{P-1}\boldsymbol{\xi}_{P-1}\\
&\qquad+\boldsymbol{\Phi}_{P-1}\big[\mathbf{U}^{\mathrm{T}}(\overline{\mathbf{y}}_P-\overline{\mathbf{U}}_P\boldsymbol{\xi}_{P-1})\big]=\boldsymbol{\psi}_{P-1}.
\end{aligned}\tag{2.48}$$

Agora o sistema bloco tridiagonal correspondente a (2.43) é escrito como

$$\begin{bmatrix}
\overline{\boldsymbol{\Lambda}}_1 & \overline{\boldsymbol{\Phi}}_1 & & & \\
\overline{\boldsymbol{\Gamma}}_2 & \overline{\boldsymbol{\Lambda}}_2 & \overline{\boldsymbol{\Phi}}_2 & & \\
& \ddots & \ddots & \ddots & \\
& & \overline{\boldsymbol{\Gamma}}_{P-2} & \overline{\boldsymbol{\Lambda}}_{P-2} & \overline{\boldsymbol{\Phi}}_{P-2}\\
& & & \overline{\boldsymbol{\Gamma}}_{P-1} & \overline{\boldsymbol{\Lambda}}_{P-1}
\end{bmatrix}
\begin{bmatrix}\boldsymbol{\xi}_1\\ \boldsymbol{\xi}_2\\ \vdots\\ \boldsymbol{\xi}_{P-2}\\ \boldsymbol{\xi}_{P-1}\end{bmatrix}=
\begin{bmatrix}\overline{\boldsymbol{\psi}}_1\\ \overline{\boldsymbol{\psi}}_2\\ \vdots\\ \overline{\boldsymbol{\psi}}_{P-2}\\ \overline{\boldsymbol{\psi}}_{P-1}\end{bmatrix},\tag{2.49}$$

com coeficientes matriciais dados por

$$\begin{aligned}
\overline{\boldsymbol{\Lambda}}_p&=\boldsymbol{\Lambda}_p-\boldsymbol{\Gamma}_p(\mathbf{V}^{\mathrm{T}}\overline{\mathbf{V}}_p)-\boldsymbol{\Phi}_p(\mathbf{U}^{\mathrm{T}}\overline{\mathbf{U}}_{p+1}),&&p=1,\ldots,P-1,\\
\overline{\boldsymbol{\Gamma}}_p&=-\boldsymbol{\Gamma}_p(\mathbf{V}^{\mathrm{T}}\overline{\mathbf{U}}_p),&&p=2,\ldots,P-1,\\
\overline{\boldsymbol{\Phi}}_p&=-\boldsymbol{\Phi}_p(\mathbf{U}^{\mathrm{T}}\overline{\mathbf{V}}_{p+1}),&&p=1,\ldots,P-2,\\
\overline{\boldsymbol{\psi}}_p&=\boldsymbol{\psi}_p-\boldsymbol{\Gamma}_p(\mathbf{V}^{\mathrm{T}}\overline{\mathbf{y}}_p)-\boldsymbol{\Phi}_p(\mathbf{U}^{\mathrm{T}}\overline{\mathbf{y}}_{p+1}),&&p=1,\ldots,P-1.
\end{aligned}\tag{2.50}$$

Novamente, as multiplicações envolvendo $\mathbf{U}$ e $\mathbf{V}$ na equação (2.50) são triviais: elas apenas indicam que o primeiro ou último bloco dos vetores $\overline{\mathbf{U}}_p$ ou $\overline{\mathbf{V}}_p$ deve ser isolado, respectivamente.

4.3. Paralelizando a solução. O algoritmo agora apresentado traz sua estratégia de paralelização em sua própria formulação. A constante P representa o número de elementos de processamento, e a constante K é o tamanho do sistema bloco tridiagonal que cada elemento de processamento deve resolver. A maior parte da fatoração do sistema (2.45) pelo algoritmo apresentado nesta seção pode ser feita em paralelo: cada subsistema $\mathbf{A}_p$ pode ser fatorado num processador diferente usando o algoritmo apresentado na Seção 2, e os resultados intermediários $\overline{\mathbf{U}}_p$, $\overline{\mathbf{V}}_p$ e $\overline{\mathbf{y}}_p$ podem ser computados localmente também. Entretanto, os valores de $\overline{\boldsymbol{\Lambda}}_p$, $\overline{\boldsymbol{\Gamma}}_p$ e $\overline{\boldsymbol{\Phi}}_p$ dados na equação (2.50) requerem que todos os processadores agreguem dados num único processador, que irá em seguida resolver o sistema de acoplamento (2.49).

Mas não é necessário, entretanto, agregar a totalidade dos objetos $\overline{\mathbf{U}}_p$ e $\overline{\mathbf{V}}_p$. A computação dos coeficientes dados em (2.50) requer apenas as matrizes $\mathbf{U}^{\mathrm{T}}\overline{\mathbf{U}}_p$, $\mathbf{U}^{\mathrm{T}}\overline{\mathbf{V}}_p$, $\mathbf{V}^{\mathrm{T}}\overline{\mathbf{U}}_p$ e $\mathbf{V}^{\mathrm{T}}\overline{\mathbf{V}}_p$, de tamanho $M \times M$, o que para valores grandes de K representa uma transferência de dados muito menor. Ainda assim, os objetos $\overline{\mathbf{U}}_p$ e $\overline{\mathbf{V}}_p$ são necessários para obter as soluções $\mathbf{y}_p$ depois que o sistema (2.49) tiver sido resolvido. Logo, é mais barato agregar e distribuir apenas os dados referentes ao acoplamento, e cada processador computa localmente sua porção da solução. Relembrando que a aplicação em vista resolve este tipo de sistema como parte de um processo iterativo, e que a solução de um tal sistema é usada para construir o sistema a ser resolvido na iteração seguinte, ter a solução distribuída pelos elementos de processamento ao final de uma iteração equivale a ter os dados iniciais distribuídos no início da iteração seguinte. A agregação da solução completa para fins de pós–processamento pode ser feita independentemente, isto é, enquanto cada elemento de processamento estiver ocupado em cálculos e os canais de comunicação estiverem livres.

5. Análise de complexidade

Agora determinamos valores aproximados para a contagem de operações de ponto flutuante dos algoritmos apresentados. Estas estimativas baseiam–se nas contagens de operações de algoritmos básicos de álgebra linear, que nós aproximamos pelas seguintes expressões:

$$\begin{array}{ll} C_1 = \dfrac{4M^3 + 3M^2 - M - 6}{6}, & \text{para a fatoração } LU \text{ de uma matriz } \mathbf{A}^{M\times M}, \\ C_2 = 2M^2 - M, & \text{para a retro–substituição de } \mathbf{Ax} = \mathbf{y}, \\ & \text{onde } \mathbf{A} \text{ sofreu fatoração } LU, \\ C_3 = 2M^3 + M^2, & \text{para o produto } \mathbf{AB} \text{ de matrizes } M \times M, \text{ e} \\ C_4 = 2M^2 + M, & \text{para o produto matriz vetor } \mathbf{Ax}, \\ & \text{onde } \mathbf{A} \text{ é } M \times M. \end{array} \tag{2.51}$$

Nós usaremos estas aproximações consistentemente nas comparações teóricas que seguem. Admitimos que todas as operações com números de ponto flutuante tomam o mesmo tempo de processamento. Uma discussão sobre a validade das expressões em (2.51) é dada no Apêndice A.

Por inspeção das equações (2.8) e (2.9) temos que o custo total da fatoração de um sistema bloco tridiagonal e sua solução para um único lado direito são dados pelas expressões abaixo:

$$\begin{aligned} F(N,M) &= NC_1 + (N-1)MC_2 + (N-1)C_3 \\ &= N\frac{28M^3 + 3M^2 - M - 6}{6} - 4M^3, \end{aligned} \tag{2.52}$$

$$\begin{aligned} S(N,M) &= NC_2 + 2(N-1)C_4 \\ &= N(6M^2 + M) - 4M^2 - 2M. \end{aligned} \tag{2.53}$$

Note que ambos os algoritmos são lineares em N, pois o sistema em blocos é tridiagonal. O custo total para resolver um sistema bloco tridiagonal com o algoritmo serial é dado por

$$\begin{aligned} C_{\text{ser}}(N,M) &= F(N,M) + S(N,M) \\ &= N\left(\frac{28M^3 + 39M^2 + 5M - 6}{6}\right) - 4M^3 - 4M^2 - 2M, \end{aligned}$$

que para $N \gg M$ pode ser aproximado por

$$C_{\text{ser}}(N,M) = N\frac{28M^3 + 39M^2 + 5M - 6}{6}. \tag{2.54}$$

As equações (2.52) e (2.53) valem também para $M = 1$ se considerarmos que as operações são executadas de forma análoga à descrita no Apêndice A. Entretanto, esta contagem de operações é maior que aquela obtida pela inspeção direta das equações (2.4) e (2.5).

5.1. Redução cíclica. Admitimos um cenário de melhor caso onde a eficiência da comunicação é tal que o paralelismo é máximo, ou seja, o problema de tamanho $N = 2^n \times P$ pode ser reduzido em paralelo n vezes e o passo serial é um problema de tamanho P.

Cada nível de redução $j = 1, \ldots, n$ lida com um problema de tamanho $2^\ell P$, $\ell = n, \ldots, 1$, ou seja, no nível j, cada um dos P processadores reduz um sistema de tamanho $K = 2^\ell P$ a um de tamanho $2^{\ell-1}P$, e $j + \ell = n + 1$. Inspecionando as expressões (2.20)–(2.26) e

(2.31)–(2.32), obtemos as seguintes contagens de operações para um único passo de redução:

$$\begin{aligned}
\frac{K}{2}C_1, &\qquad \text{a fatoração } LU \text{ de } \mathbf{A}_i \text{ para } i \text{ ímpar;}\\
M\frac{K-1}{2}C_2, &\qquad \text{das equações (2.20)–(2.21);}\\
M\frac{K-1}{2}C_2, &\qquad \text{das equações (2.22)–(2.23);}\\
2\left(\frac{K}{2}-1\right)C_3, &\qquad \text{das equações (2.24)–(2.25);}\\
KC_3, &\qquad \text{da equação (2.26);}\\
\frac{K}{4}C_2, &\qquad \text{das equações (2.31); e}\\
KC_4, &\qquad \text{da equação (2.32).}
\end{aligned} \tag{2.55}$$

Os C_i são aqueles dados por (2.51). Note que as equações (2.20)–(2.23) envolvem apenas a solução de um sistema triangular, ou seja, requerem apenas um passo de substituição e portanto custam $C_2/2$. Somando as expressões em (2.55) obtemos

$$K\left(\frac{C_1}{2}+(M+1/4)C_2+2C_3+C_4\right)-(MC_2+2C_3)=KR(M)+Q(M). \tag{2.56}$$

O valor absoluto de $Q(M)$ é sempre inferior ao de $R(M)$; além disso, estamos interessados no valor de (2.56) para valores grandes de K, e por isso descartamos $Q(M)$ nas deduções seguintes. Relembrando que $K=2^\ell P$, a contagem de operações para n reduções é dada por

$$\sum_{\ell=1}^{n} 2^\ell PR(M) = PR(M)\sum_{\ell=1}^{n}2^\ell = 2P(2^n-1)R(M),$$

e a contagem total de operações, incluindo a solução do sistema remanescente de tamanho P, é

$$C_{\text{cyc}} = 2P\big(2^n-1\big)R(M)+C_{\text{ser}}(P,M),$$

com C_{ser} dado por (2.54). Para $N \gg P$ grande, a expressão acima é claramente dominada pelo primeiro termo, ou seja, é bem aproximada por

$$C_{\text{cyc}}(N,M) = 2NR(M). \tag{2.57}$$

Agora nós derivamos o custo para o caso em que a redução é interrompida em algum nível $k < n$, e o sistema remanescente de tamanho $2^{n-k} \times P$ é resolvido serialmente:

$$\begin{aligned} C_{\text{cyc}}(N, M, k) &= \sum_{\ell=n-k+1}^{n} 2^\ell R(M) + C_{\text{ser}}(2^{n-k}P, M) \\ &= P\Big(\sum_{\ell=1}^{n} 2^\ell - \sum_{\ell=1}^{n-k} 2^\ell\Big) R(M) + C_{\text{ser}}(2^{n-k}P, M) \\ &= 2P\big(2^{n-k}(2^k - 1)R(M)\big) + C_{\text{ser}}(2^{n-k}P, M). \end{aligned}$$

Uma vez que $C_{\text{ser}}(N, M)$ é linear em N, o custo total derivado acima pode ser simplificado à forma

$$C_{\text{cyc}}(N, M, k) = \frac{2^k - 1}{2^{k-1}} NR(M) + \frac{C_{\text{ser}}(N, M)}{2^k}, \tag{2.58}$$

que é quase igual a (2.57) para $k = n$ e 2^n grande: a pequena discrepância deve–se à premissa de que $k < n$ na derivação de (2.58). Este resultado é válido para todo k no intervalo $[0, n]$: no caso $k = 0$, reduz–se a (2.54).

5.2. Divisão–e–conquista. O algoritmo de divisão–e–conquista resolve P sistemas bloco tridiagonais de tamanho K usando algum outro algoritmo menos custoso: nesta subseção, nós usamos a contagem de operações do algoritmo serial. Da mesma maneira, os valores admitidos em (2.51) também se aplicam; a estes nós acrescentamos a constante

$$C_5 = 2KM^2 + M, \quad \text{para o produto matriz–vetor } \mathbf{Ax} \text{ onde } \mathbf{A} \text{ é } KM \times M. \tag{2.59}$$

O passo de fatoração paralela consiste em realizar a fatoração LU das matrizes $\mathbf{A}_p$ e subseqüentemente determinar os objetos $\overline{\mathbf{U}}_p$, $\overline{\mathbf{V}}_p$ e $\overline{\mathbf{y}}_p$, conforme na equação (2.46). O custo destas operações é: $P \times F(K, M)$ para a fatoração LU de todas as matrizes $\mathbf{A}_p$, mais $(P-1)M \times S(K, M)$ para os objetos $\overline{\mathbf{U}}_p$ e $P \times S(K, M)$ para os objetos $\overline{\mathbf{y}}_p$. Note que a computação de $\overline{\mathbf{V}}_p$ usando o algoritmo serial é bastante simplificada: uma vez que apenas o último bloco do lado direito tem entradas não–nulas, a solução do primeiro sistema bloco triangular conforme na equação (2.9) reduz–se a resolver um único sistema linear, ou seja, apenas $\mathbf{z}_N$ na equação (2.9) precisa ser determinado resolvendo $\mathbf{Z}_{p,N} = \overline{\mathbf{A}}_{p,N}^{-1} \mathbf{V}_N$, onde $\mathbf{V}_N$ é a matriz identidade e $\mathbf{Z}_{p,N}$ é o último bloco do resultado intermediário que será usado para

resolver o sistema para $\overline{\mathbf{V}}_p$. A contagem de operações para este caso particular é menos da metade daquela dada na equação (2.53). Acrescentamos a expressão

$$S_V(K, M) = K(2M^2 + 2M) \tag{2.60}$$

para denotar este caso especial. A contagem total de operações para as expressões na equação (2.46) é

$$P \times F(K, M) + (P-1)M \times S(K, M) + (P-1)M \times S_V(K, M) + P \times S(K, M), \tag{2.61}$$

onde F e S são dados por (2.52) e (2.53), respectivamente. Até aqui o trabalho pode ser feito em paralelo: os dois passos seguintes requerem comunicação entre os processadores. Da equação (2.50), os coeficientes $\overline{\mathbf{\Lambda}}_p$, $\overline{\mathbf{\Gamma}}_p$ e $\overline{\mathbf{\Phi}}_p$ requerem

$$\big(2(P-1) + (P-2) + (P-2)\big)C_3 \tag{2.62}$$

operações, e a solução do sistema dado por (2.49) requer

$$F(P-1, M) + S(P-1, M) \tag{2.63}$$

operações. A solução do sistema completo é obtida usando a solução do sistema de acoplamento na equação (2.46), a um custo

$$2(P-1)C_5. \tag{2.64}$$

Lembramos que $N = PK + (P-1)$, e interessam-nos problemas grandes, isto é, onde $N > K \gg P$: sob tais premissas $N \approx PK$, e os valores das expressões (2.61) e (2.64) são muito superiores aos de (2.62) e (2.63). Substituindo (2.52) e (2.53) em (2.61), somando (2.64) e desprezando termos que não contêm K nos dá um custo total

$$C_{\text{div}}(N, M, P) = \frac{N}{6}\left(\left(76 - \frac{48}{P}\right)M^3 + \left(59 - \frac{20}{P}\right)M^2 + \left(15 - \frac{10}{P}\right)M - \left(17 - \frac{11}{P}\right)\right). \tag{2.65}$$

Para $P = 1$, $C_{\text{div}} \equiv C_{\text{ser}}$ da equação (2.54), o que é apropriado uma vez que nesta aproximação nós descartamos todo o custo das operações relativas ao acoplamento de múltiplas soluções obtidas pelo algoritmo da Seção 2 — neste caso uma única. Para M, N fixos e P crescente, C_{div} rapidamente tende a um limitante superior. A figura 2.1 mostra o fator de redundância de ambos algoritmos de redução cíclica e divisão–e–conquista, em relação ao algoritmo serial.

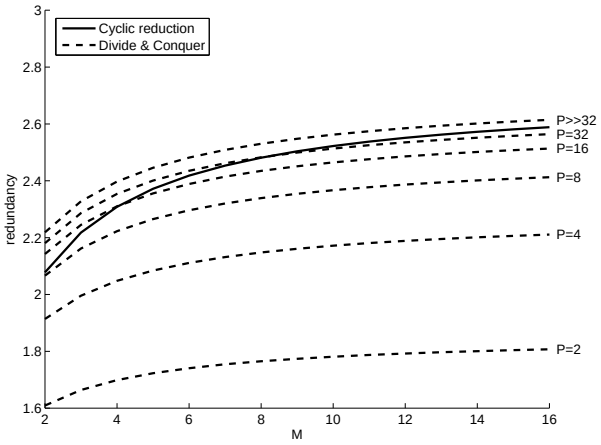


FIGURA 2.1. Redundância dos algoritmos de redução cíclica e divisão–e–conquista, em relação ao algoritmo serial, em termos do tamanho de bloco M.

5.3. Discussão. Uma comparação visual dos valores teóricos obtidos nas subseções precedentes para a contagem do número de operações de ponto flututante de cada um dos três algoritmos é dada na Figura 2.1. A redundância do algoritmo de redução cíclica é independente do número de processadores empregado, e cresce como função de M a partir de um mínimo de 2.08 para $M = 2$, conforme já apresentado na literatura. A redundância do algoritmo de divisão–e–conquista, por outro lado, depende do número de processadores pelos quais o problema é distribuído. Para P pequeno, ele realiza um número consideravelmente menor de cálculos que a redução cíclica: apenas para valores relativamente altos de P o algoritmo de divisão–e–conquista apresenta mais redundância.

No que diz respeito à redundância isoladamente, o algoritmo de redução cíclica não oferece nenhuma vantagem a menos que dezesseis processadores ou mais sejam usados, e mesmo no caso extremo e pouco realista onde $P \gg 32$ e $M = 2$, ele executa apenas 10% menos operações de ponto flutuante quando comparado com o algoritmo de divisão–e–conquista.

Certamente, estas observações se baseiam puramente na análise teórica dos algoritmos apresentados, e o desempenho destes há de refletir não apenas o número de operações de ponto flutuante realizadas, mas também a eficiência no uso de hierarquia de memória e na comunicação entre os processadores. Na próxima seção, realizamos experimentos numéricos

para aferir o comportamento real dos três algoritmos, e usamos esta análise de complexidade para interpretar os resultados.

6. Experimentos numéricos

Relembramos que a motivação deste trabalho é resolver eficientemente problemas "relativamente pequenos" usando máquinas paralelas "amplamente disponíveis atualmente". O tamanho dos problemas $\{N, M\}$ e o tipo de computador em que pretendemos resolvê-los são portanto parâmetros segundo os quais avaliaremos o desempenho dos algoritmos, a fim de determinar a aplicabilidade dos resultados obtidos.

Os algoritmos paralelos foram implementados sobre MPI (MPICH2), e uma versão serial baseada no código com MPI foi usada para avaliar o speedup relativo e a redundância. Quando aplicável, rotinas de troca de mensagens assíncronas (não–bloqueantes) foram usadas. Todo o código foi escrito em C, usando LAPACK para a álgebra linear elementar. Uma vez que o código fora testado e validado, as rotinas originais do LAPACK foram substituídas por código otimizado em C. Vale ressaltar que para blocos pequenos (por exemplo, $M = 3$), nós detectamos que mais de 15% do tempo de execução era gasto pelo LAPACK em suas rotinas de inicialização, ou seja, analisando parâmetros de entrada que permitem ao programa que o invoca especificar se as matrizes passadas como parâmetros devem ser multiplicadas pela esquerda ou pela direita, ou se um sistema é triangular superior ou inferior. Evidentemente, o LAPACK não foi escrito com problemas de dimensões tão pequenas em mente. Removemos esta computação desnecessária antes de comparar os tempos de execução com os valores teóricos obtidos na análise de complexidade, do contrário os valores experimentais estariam contaminados e a comparação não teria significado.

Problemas de cada tamanho $\{N, M\}$ foram resolvidos usando o algoritmo serial da Seção 2, usando implementações paralelas dos algoritmos dados nas Seções 3 e 4, e usando implementações seriais destes dois algoritmos, compilados condicionalmente a partir do mesmo código fonte, que alocam a mesma quantidade de memória e realizam exatamente as mesmas contas: no total, cinco programas diferentes foram avaliados. Os experimentos

foram realizados em dois clusters homogêneos aos quais tivemos acesso dedicado durante a execução dos testes. Finalmente, para obter resultados comparáveis e generalizáveis, o número de processadores utilizado foi sempre uma potência de dois.

Todos os cinco programas tiveram seu desempenho aferido para os mesmos intervalos de M e N. A aplicação em vista sugere os seguintes intervalos para o tamanho do problema: M no intervalo $[2, 12]$ e N no intervalo $[10^2, 10^9]$. Entretanto, para obter uma visão mais geral do desempenho dos algoritmos, nós escolhemos para M o intervalo $[2, 32]$. Para cada valor de M, usamos N no intervalo $[2^7, 2^n]$ para o maior n possível, isto é, para o qual o sistema operacional alocaria a memória física necessária. O valor de N também foi restrito a potências de dois, para simplificar a análise do algoritmo de redução cíclica: os valores efetivamente usados pertencem ao intervalo $[2^7, 2^{18}]$.

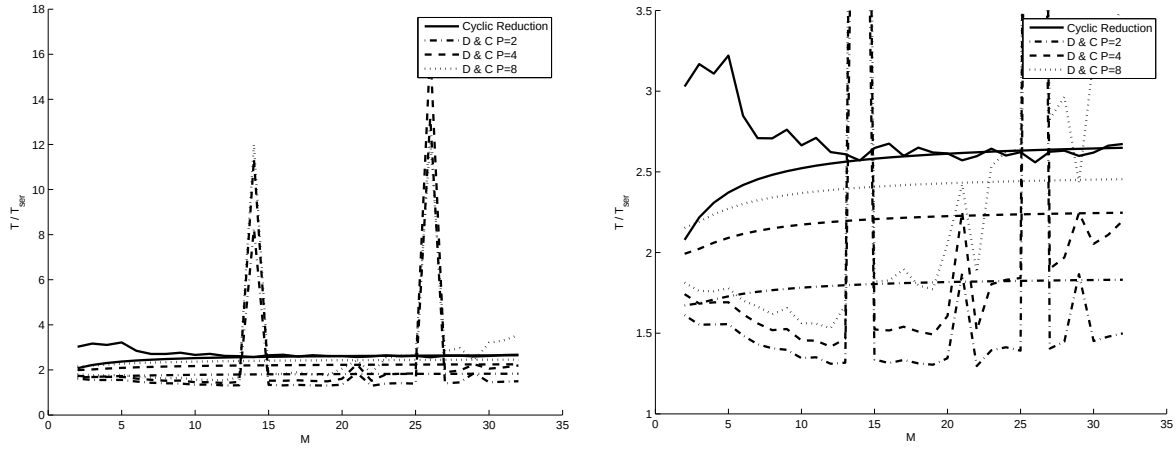
Os coeficientes da matriz $\mathbf{A}$ e do lado direito $\mathbf{y}$ foram determinados de forma semelhante à usada nos problemas–teste em [**1**], onde a a matriz $\mathbf{A}$ é simétrica e em banda, e a largura de sua banda é dada pela expressão (2.10). Em vez de matrizes em banda, que possuem uma estrutura peculiar de zeros nos blocos nas duas diagonais secundárias, usamos matrizes cheias em todos os blocos $\mathbf{A}_i$, $\mathbf{B}_i$ e $\mathbf{C}_i$. Cada célula destes blocos tem valor no intervalo $[1, p]$, $p \in \mathbb{N}$, exceto a diagonal principal que tem elementos num intervalo $[\alpha, \alpha p]$, $\alpha > 1$. O número de condicionamento do sistema é pequeno se $\alpha \gg p$: nestes experimentos usamos $p = 37$ e $\alpha = 1000$. O vetor $\mathbf{y}$ é computado desta matriz e uma solução $\mathbf{x}$ conhecida, de forma que possamos determinar o erro númerico do procedimento de solução.

Como a precisão das rotinas para aferir o tempo de execução é limitada, e como a margem de erro delas é de magnitude comparável aos tempo de computação que desejamos medir, cada experimento consistiu em resolver repetidamente um mesmo sistema linear de tamanho $\{N, M\}$, de forma que o tempo de computação total fosse de pelo menos um segundo: o tempo gasto para resolver o sistema linear uma única vez é obtido dividindo este tempo total pelo número de repetições. Com isso a perturbação inerente às rotinas de relógio torna-se desprezível. Mais detalhes sobre o procedimento experimental são dados no Apêndice B.

6.1. Experimentos no cluster I. O primeiro cluster no qual realizamos experimentos consistia em oito nós biprocessados de máquinas x86 com sistema operacional Linux 2.6. Cada nó possuía 4 Gb de memória física instalada, e cada processador de 2.4 GHz acusava uma média geométrica de 770 MFLOPS sob o benchmark "Livermore Loops" ([**26**]). Os nós estavam interconectados por um switch gigabit dedicado, e embora cada nó tivesse dois processadores nós alocamos apenas uma tarefa MPI a cada nó: optamos por não incluir os resultados com dezesseis processadores por tornarem a natureza do ambiente paralelo híbrida, ou seja, usamos o Cluster I como uma máquina exclusivamente de memória distribuída.

Os programas executados nestes testes foram compilados usando GCC 3.4.4 com todas otimizações habilitadas (chaves `-ffast-math` e `-O3`). Nós desativamos a aritmética IEEE uma vez que consta da literatura ([**1**]) que esta leva a tempos de execução erráticos e não–reprodutíveis. Enquanto executava, cada programa mediu o tempo total gasto nas rotinas de solução, sem incluir o tempo necessário para montar o sistema linear. A resolução do relógio, de acordo com o sistema operacional, era de 1 milisegundo, mas repetidas tomadas de tempo mostraram que a flutuação era de fato ± 5 milisegundos. Esta falta de precisão impediu–nos de medir os tempos de cada etapa interna do algoritmo, bem como os tempos de comunicação, sendo a única medida precisa a do tempo total.

A Figura 2.2 mostra as razões entre os tempos de execução dos algoritmos paralelos num único processador e o tempo de execução do algoritmo serial, ou seja, a "redundância experimental" de cada algoritmo paralelo. Comparando os valores teórico e experimental, vemos que o algoritmo de divisão–e–conquista apresenta melhor desempenho do que o esperado para a maioria dos tamanhos de bloco M. Atribuímos isto a uma utilização mais eficiente da cache em comparação ao algoritmo serial. Para $i = 1, \ldots, N$, o programa serial realiza poucas operações com os blocos da matriz $\mathbf{A}$ à medida que a percorre. Já o algoritmo de divisão–e–conquista acessa os dados correspondentes a cada valor de i mais vezes, e percorre cada um dos P segmentos de tamanho K mais de uma vez antes de passar para o próximo. Com isso sua localidade de dados é maior, e conseqüentemente o tempo médio de acesso à memória é menor. Isto se reflete na razão entre os tempos totais de execução como uma redundância

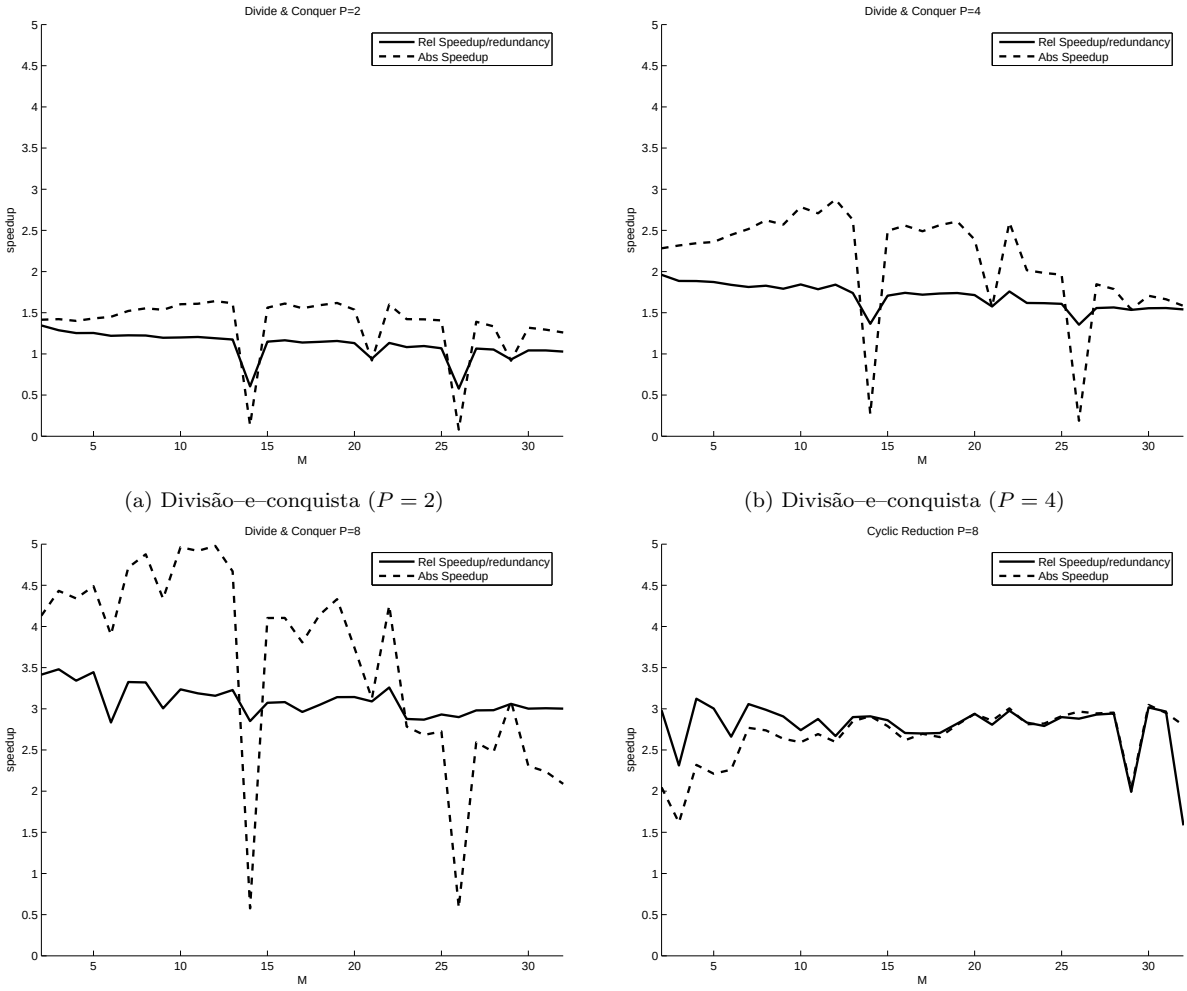


(a) (b)

FIGURA 2.2. Redundâncias teóricas e as razões de tempos medidos correspondentes no Cluster I. As linhas suaves correspondem aos valores teóricos da análise de complexidade, e as linhas angulosas são as razões entre os tempos experimentais correspondentes. As duas figuras apresentam os mesmos dados: a da direita foi cortada devido aos pontos que se destacam na figura da esquerda, correspondentes a comportamento degenerado do algoritmo de divisão–e–conquista.

aparentemente menor, uma vez que a grandeza que comparamos à redundância teórica é a razão entre os tempos de execução. A implementação do algoritmo de redução cíclica não tira a mesma vantagem pois percorre a matriz inteira várias vezes, uma a cada nível de redução. Em termos práticos, a introdução de cálculos redundantes não é tão prejudicial como os resultados exibidos na Figura 2.1 sugerem, posto que a computação adicional tem por efeito colateral um aumento do índice de acerto na cache.

Eficiência de cache também explica os picos observados na redundância experimental do algoritmo de divisão–e–conquista. Se as premissas de modelagem feitas na análise de complexidade — a saber, que todas as operações de ponto flutuante requerem a mesma quantidade de tempo e que todas as outras operações, como indexação e acesso à memória, podem ser desprezadas — fossem verdade, então a razão entre os speedups absoluto e relativo seria igual à redundância. Entretanto, estes dois speedups são obtidos dos tempos de execução de três programas diferentes, e cada um deles impõe diferentes demandas na hierarquia de memória. A Figura 2.3 ilustra melhor esta questão. As duas curvas em cada uma das quatro figuras deveriam ser muito próximas no cenário idealizado descrito acima, e de fato a



(a) Divisão–e–conquista ($P = 2$)

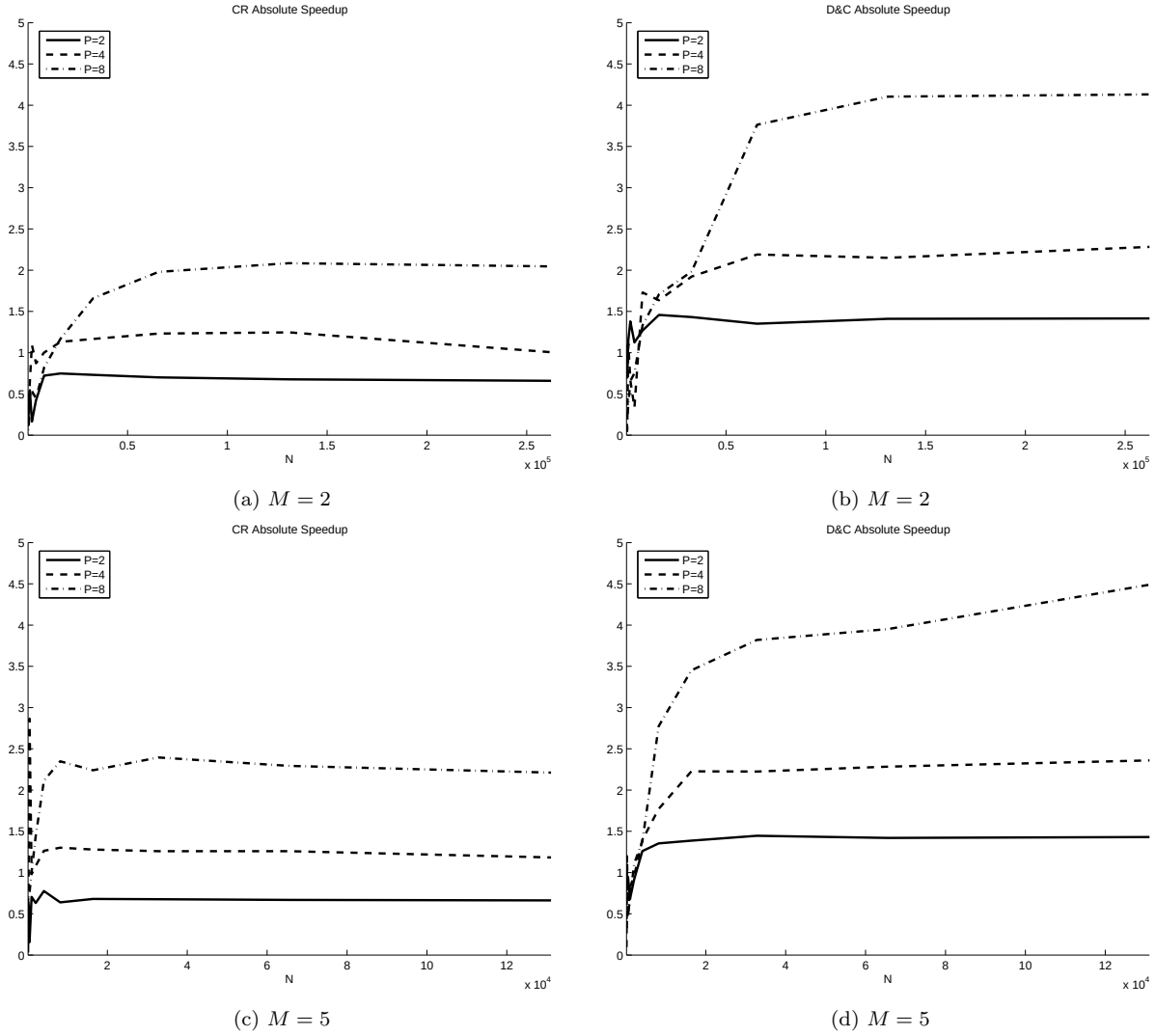
(b) Divisão–e–conquista ($P = 4$)

(c) Divisão–e–conquista ($P = 8$)

(d) Redução cíclica ($P = 8$)

FIGURA 2.3. Cada figura mostra o speedup relativo no Cluster I dividido pela redundância teórica e o speedup absoluto para o algoritmo de divisão–e–conquista em (a) dois, (b) quatro e (c) oito processadores, e para o algoritmo de redução cíclica em oito processadores (d). Discordâncias entre cada par de curvas devem–se principalmente a efeitos de cache.

concordância para o algoritmo de redução cíclica é excelente. Entretando, para o algoritmo de divisão–e–conquista observamos dois tipos de discordância. O primeiro relaciona–se à questão já levantada do melhor uso da cache: isto pode ser visto na Figura 2.3 como uma dominância consistente das linhas tracejadas sobre as linhas sólidas, em todos os casos. O segundo tipo de discordância é que há exceções localizadas nesta dominância, ou seja, o speedup absoluto possui dois pontos "patológicos", específicamente para $M = 14$ e $M = 26$. Estes são os mesmos valores para os quais observamos picos na Figura 2.2. Note que as



(a) $M = 2$ (b) $M = 2$

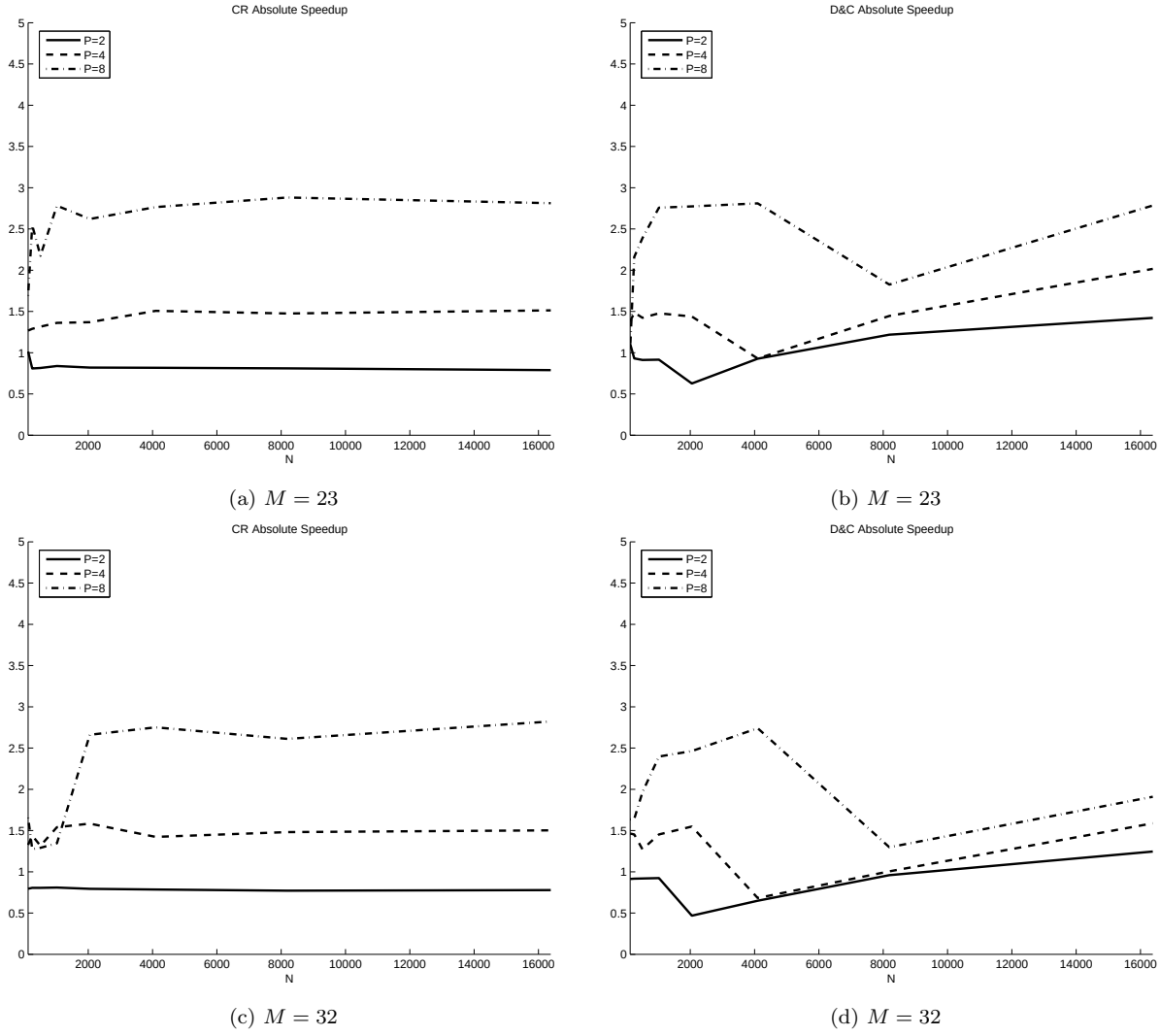
(c) $M = 5$ (d) $M = 5$

FIGURA 2.4. Speedup absoluto para tamanhos de bloco pequenos no Cluster I. As figuras da esquerda correspondem à redução cíclica, e as figuras da direita à divisão–e–conquista. As figuras de cima são para tamanho de bloco $M = 2$, e as de baixo para $M = 5$.

linhas sólidas, correspondentes ao speedup relativo, apresentam irregularidades semelhantes, mas menos pronunciadas: os dois programas sendo comparados usam a hierarquia de cache de forma parecida.

As razões apresentadas nas Figuras 2.2 e 2.3 correspondem, para cada valor de M, ao N máximo com o qual realizamos experimentos (relembrando que este N é limitado pela memória física disponível). Os dois valores $M = 14$ e $M = 26$ para os quais o algoritmo de

divisão–e–conquista apresenta um comportamento degenerado não são por si valores críticos: o tamanho do sistema N também influi, como pode ser observado na Figura 2.5.



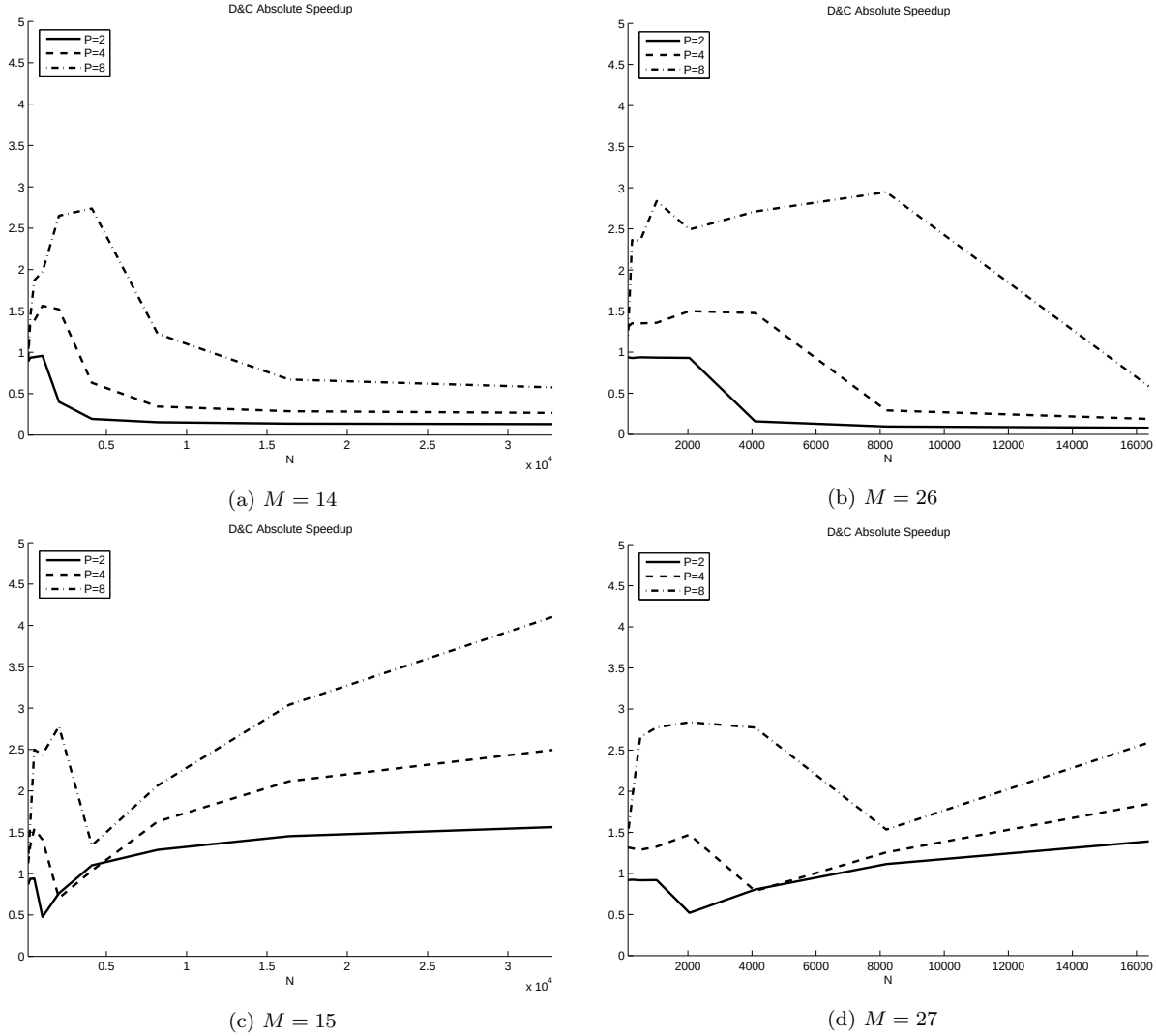
(a) $M = 23$ (b) $M = 23$

(c) $M = 32$ (d) $M = 32$

FIGURA 2.5. Speedup absoluto para tamanhos de bloco grandes no Cluster I. As figuras da esquerda correspondem à redução cíclica, e as figuras da direita à divisão–e–conquista. As figuras de cima são para tamanho de bloco $M = 23$, e as de baixo para $M = 32$.

As Figuras 2.4 a 2.6 mostram o speedup absoluto como função do tamanho N da matriz. A Figura 2.4 mostra os resultados para tamanhos de bloco pequenos $M = 2$ e $M = 5$. Observamos que ambos algoritmos apresentam um desempenho ruim para N relativamente pequeno, provavelmente devido ao impacto do overhead de comunicação: para M pequeno, o tamanho N da matriz deve ser grande o bastante para que o total de computação a ser

realizado em cada elemento de processamento compense o tempo gasto em comunicação. Fora isso, vemos que a redução cíclica apresenta um desempenho claramente inferior à divisão–e–conquista neste intervalo ($M \in [2, 12]$) para o tamanho de bloco.



(a) $M = 14$ (b) $M = 26$

(c) $M = 15$ (d) $M = 27$

FIGURA 2.6. Quatro casos patológicos do algoritmo de divisão–e–conquista no Cluster I, com tamanhos de bloco $M = 14$ (a), $M = 26$ (b), $M = 15$ (c) e $M = 27$ (d). Note que o ponto crítico nas figuras (c) e (d) é para um valor de N menor que nas figuras (a) e (b), nas quais M é menor de uma unidade.

Para valores maiores $M > 20$, os resultados obtidos são similares àqueles apresentados na Figura 2.5, onde os exemplificamos pelos casos $M = 23$ e $M = 32$. Se $P > 4$, a redução cíclica tem um desempenho superior ao algoritmo de divisão–e–conquista para todos os tamanhos de problema. Podemos observar ainda nas Figuras 2.5(b) e 2.5(d), onde $M = 32$,

uma degeneração semelhante àquela vista anteriormente para $M = 14$ e $M = 26$: aqui, entretanto, os valores de N críticos são diferentes dependendo do número de processadores empregados.

A Figura 2.6 mostra os resultados para os valores de M para os quais observamos discrepâncias na Figura 2.2, ou seja, onde o algoritmo de divisão–e–conquista usa a cache de forma relativamente ineficiente.

6.2. Experimentos no Cluster II. O segundo cluster no qual realizamos experimentos era um IBM iPSC690 com o sistema operacional AIX 5.3. Ele consistia em vários nós multiprocessados idênticos, mas cada tarefa paralela era restrita a um único nó, ou seja, ele só podia ser usado como máquina de memória compartilhada. Cada nó possuía 64 Gb de memória física instalada e trinta e dois processadores, cada um a 1.3 GHz e acusando 300 MFLOPS quando testado usando o benchmark "Livermore Loops" ([**26**]) — embora o valor dado pelo fabricante seja de 5.2 GFLOPS por nó, ou 165 MFLOPS por processador. Os programas executados nestes testes foram compilados com o IBM XL C usando todas as otimizações disponíveis. Assim como nos experimentos executados no Cluster I, medimos apenas o tempo total gasto nas rotinas de solução. A resolução do relógio dada pelo sistema operacional foi novamente de 1 milisegundo.

A Figura 2.7 é análoga à Figura 2.2 na página 41: ela mostra as razões entre os tempos de execução dos algoritmos paralelos num único processador e o tempo de execução do algoritmo serial. O algoritmo de divisão–e–conquista mais uma vez apresenta um desempenho melhor que o esperado. Os efeitos de cache são menos pronunciados, mas ocorrem para um número maior de valores de M.

A Figura 2.8 é análoga à Figura 2.3 da página 42. Vemos aqui o mesmo tipo de discordância observado nos experimentos no Cluster I, especificamente que a redundância experimental fica abaixo da teórica, e o desempenho do algoritmo de divisão–e–conquista se degrada para tamanhos específicos do problema. As Figuras 2.9 e 2.10 correspondem às Figuras 2.4 e 2.5, mostrando o speedup absoluto como função do tamanho N da matriz.

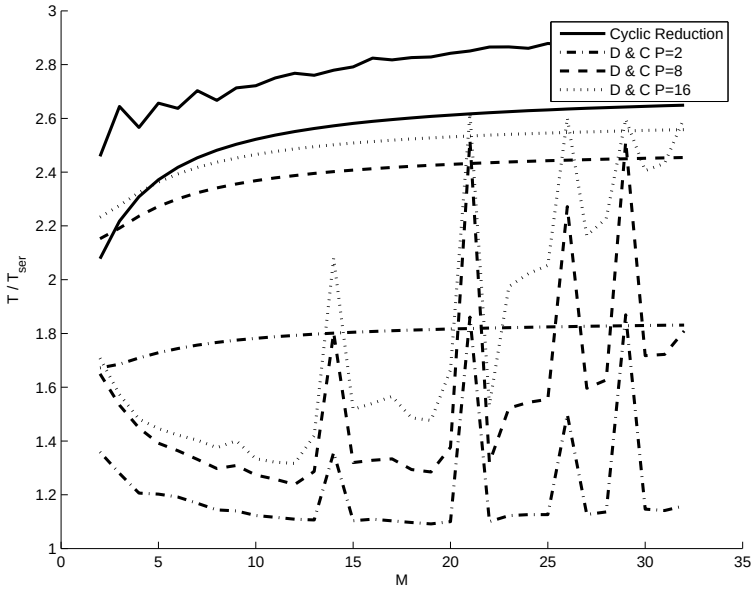
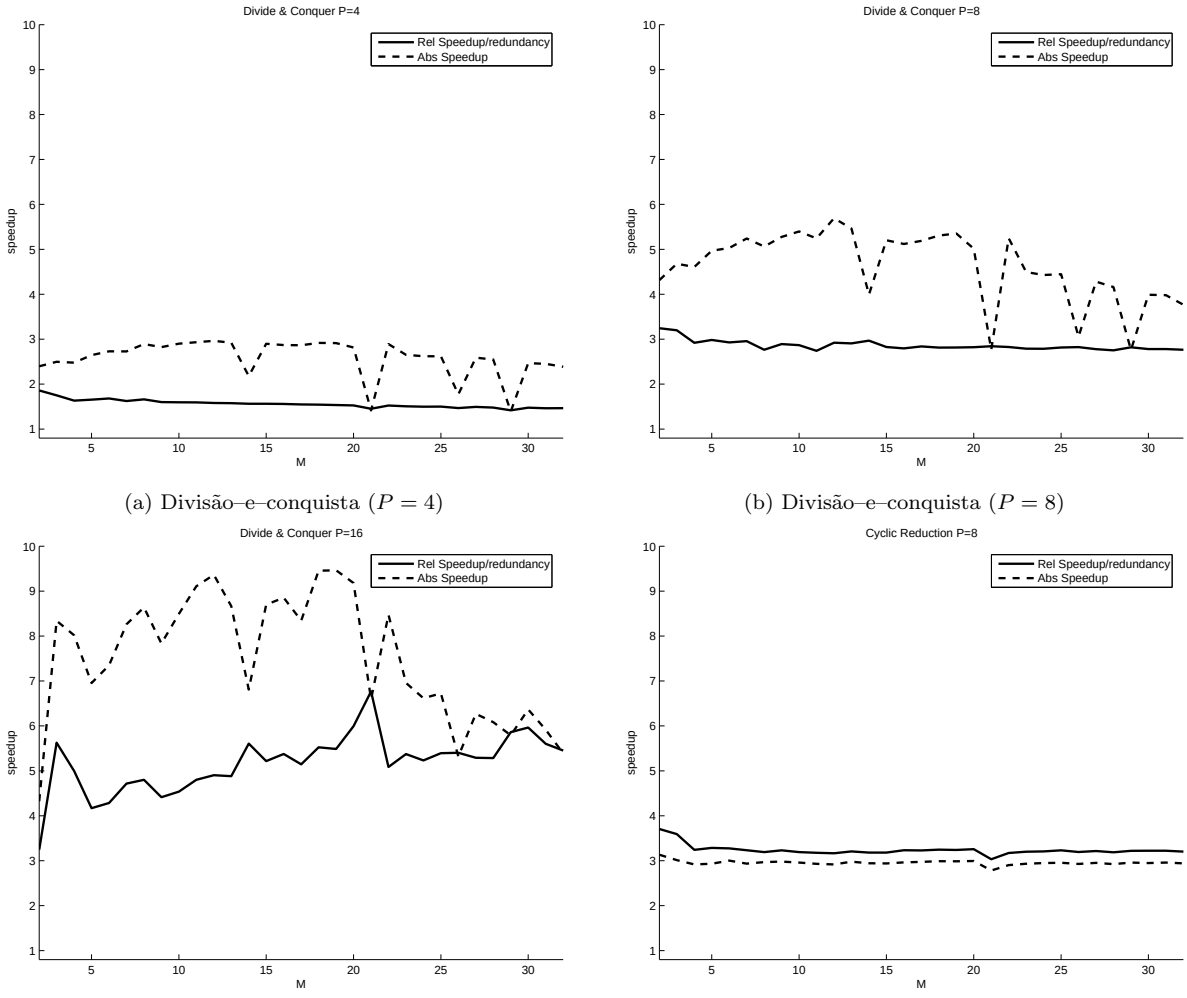


FIGURA 2.7. Redundâncias teóricas e as razões de tempos medidos correspondentes no Cluster II. As linhas suaves correspondem aos valores teóricos da análise de complexidade, e as linhas angulosas são as razões entre os tempos experimentais correspondentes.

Para M pequeno, os resultados são semelhantes aos obtidos no Cluster I, considerando até oito processadores, embora a vantagem do algoritmo de divisão–e–conquista seja menor do que a exibida no outro cluster. Há entretanto uma mudança qualitativa no desempenho de ambos os algoritmos quando dezesseis processadores são empregados, e o algoritmo de redução cíclica passa a ser mais rápido pra $M = 5$. Os resultados para valores maiores $M > 20$, apresentados na Figura 2.10, assemelham–se aos obtidos no Cluster I: a redução cíclica apresenta um desempenho comparável ao da divisão–e–conquista.

7. Discussão

A Figura 2.11 mostra os speedups absolutos de ambos os algoritmos paralelos no dois clusters para valores diversos do tamanho de bloco M. Fica claro que o algoritmo de divisão–e–conquista tem um desempenho muito superior que a redução cíclica para M pequeno. Entretanto, conforme M aumenta isto deixa de ser verdade, e dependendo também do número de processadores e das características do hardware do ambiente paralelo, a redução cíclica pode ter um desempenho melhor.



(a) Divisão–e–conquista ($P = 4$)

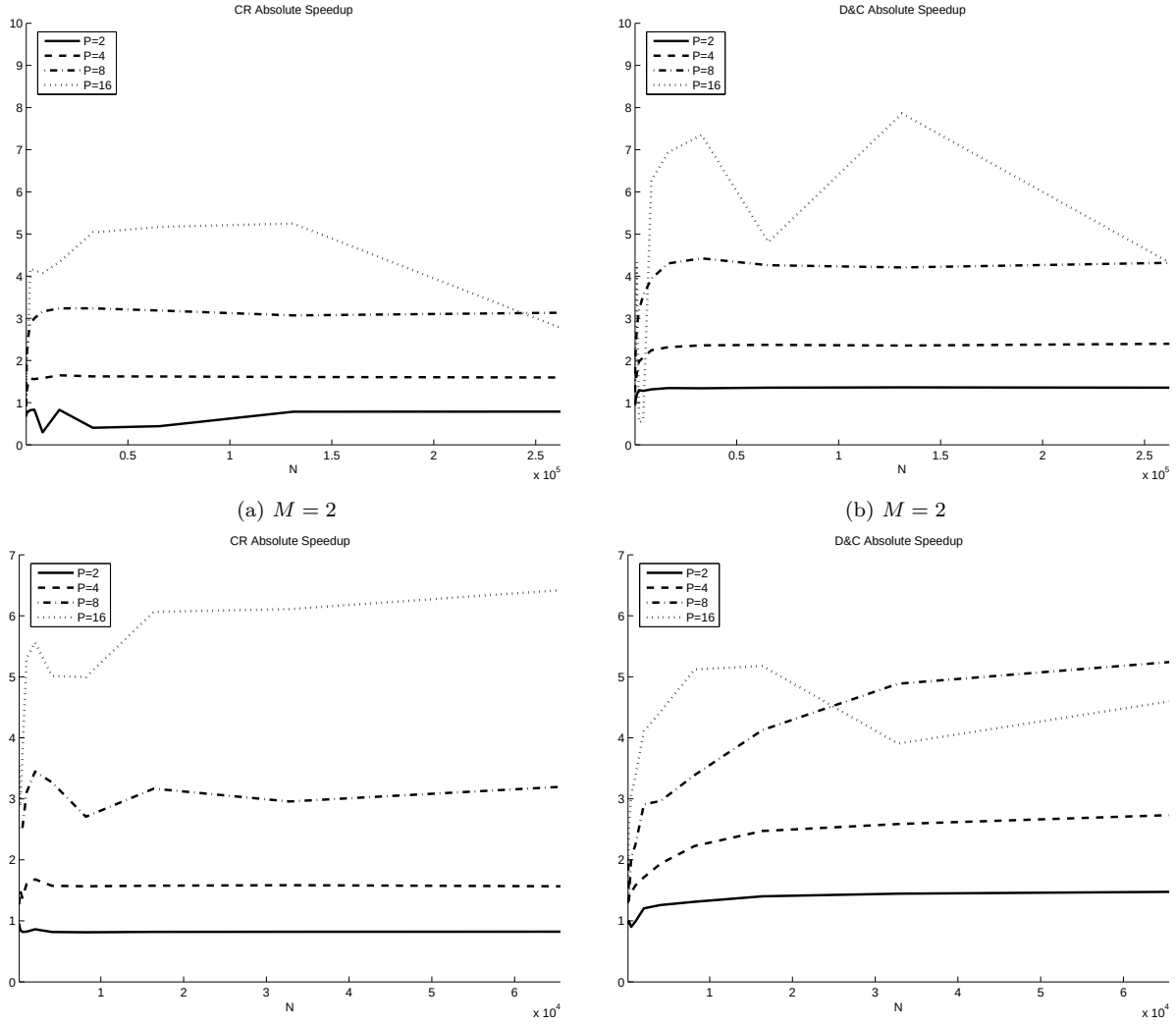
(b) Divisão–e–conquista ($P = 8$)

(c) Divisão–e–conquista ($P = 16$)

(d) Redução cíclica ($P = 8$)

FIGURA 2.8. Cada figura mostra o speedup relativo no Cluster II dividido pela redundância teórica e o speedup absoluto para o algoritmo de divisão–e–conquista em (a) quatro, (b) oito e (c) dezesseis processadores, e para o algoritmo de redução cíclica em oito processadores (d). Discordâncias entre cada par de curvas devem–se principalmente a efeitos de cache.

Conforme visto nas da seção anterior, nossa implementação do algoritmo de divisão–e–conquista é suscetível a severos efeitos de cache, para certos valores de M. Entretanto, se nos restringirmos ao intervalo de interesse $M \in [2, 12]$, para o qual os tempos de execução podem ser vistos na Figura 2.12, temos que a superfície na Figura 2.12(d) é dominada pela da Figura 2.12(c), ou seja, o algoritmo de divisão–e–conquista é mais rápido em todo o intervalo de interesse: os já mencionados pontos discrepantes surgem apenas para $M \geq 14$.

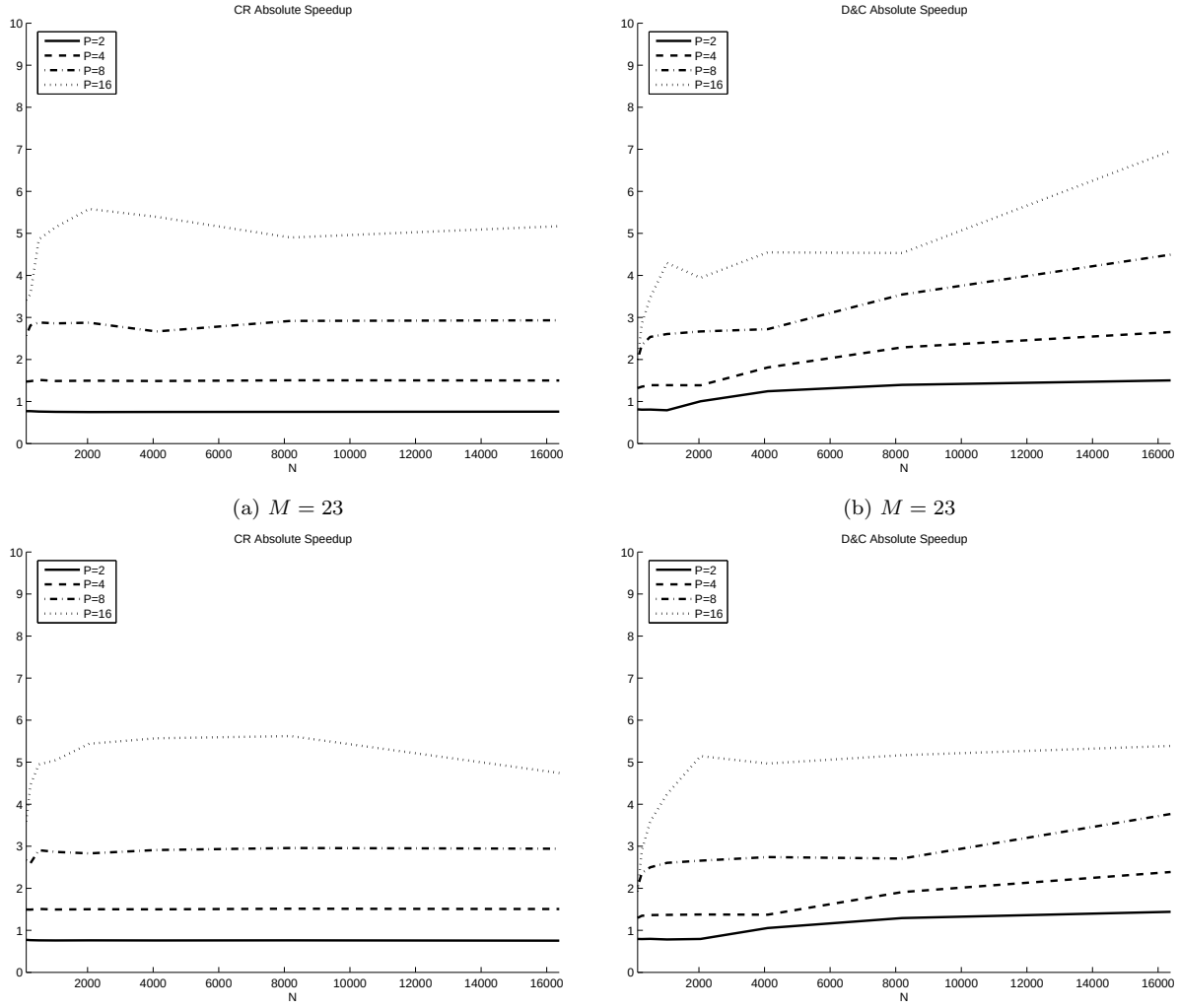


(a) $M = 2$ (b) $M = 2$

(c) $M = 5$ (d) $M = 5$

FIGURA 2.9. Speedup absoluto para tamanhos de bloco pequenos no Cluster II. As figuras da esquerda correspondem à redução cíclica, e as figuras da direita à divisão–e–conquista. As figuras de cima são para tamanho de bloco $M = 2$, e as de baixo para $M = 5$.

Em termos práticos, o algoritmo de divisão–e–conquista é geralmente vantajoso quando poucos processadores são empregados. Entretanto, para certos valores de $\{N, M\}$, os tamanhos dos blocos alocados por este algoritmo excedem ligeiramente o tamanho das páginas de cache, de forma que esta fica subutilizada. Este problema é dependente de implementação e de arquitetura, podendo vir a ser contornado.

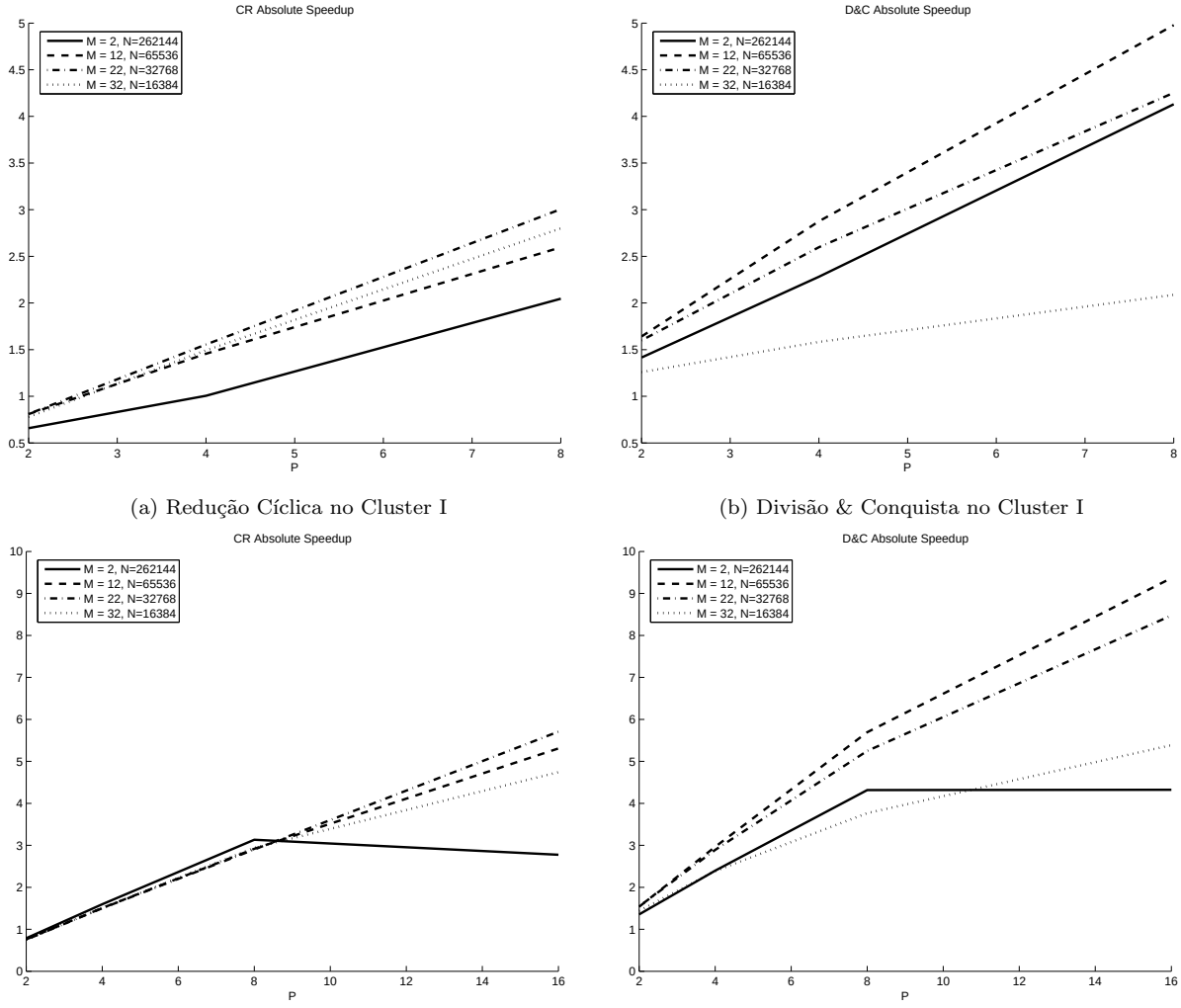


(a) $M = 23$ (b) $M = 23$

(c) $M = 32$ (d) $M = 32$

FIGURA 2.10. Speedup absoluto para tamanhos de bloco grandes no Cluster II. As figuras da esquerda correspondem à redução cíclica, e as figuras da direita à divisão–e–conquista. As figuras de cima são para tamanho de bloco $M = 23$, e as de baixo para $M = 32$.

Medidas de desempenho costumam ser realizadas em computadores estado–da–arte, para ter–se uma idéia de quão útil uma nova máquina pode ser na solução de um problema conhecido, e o tamanho do problema usado ao aferir este desempenho costuma ser influenciado pelas novas possibilidades oferecidas pelo hardware, ficando a aplicação final em segundo plano. Nosso problema impõe uma restrição muito específica no tamanho do problema, que é a banda muito estreita ou tamanho de bloco pequeno e um número reduzido de processadores. Por um lado, nossos experimentos confirmaram o que já consta da literatura: a



(a) Redução Cíclica no Cluster I

(b) Divisão & Conquista no Cluster I

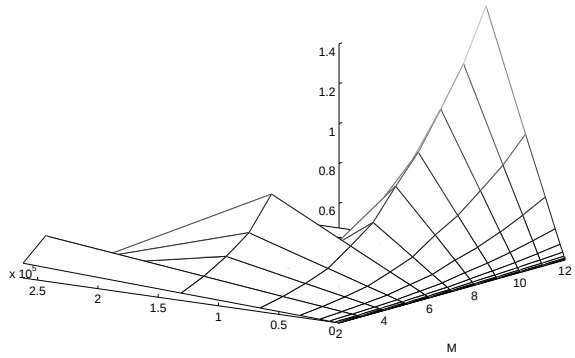
(c) Redução Cíclica no Cluster II

(d) Divisão & Conquista no Cluster II

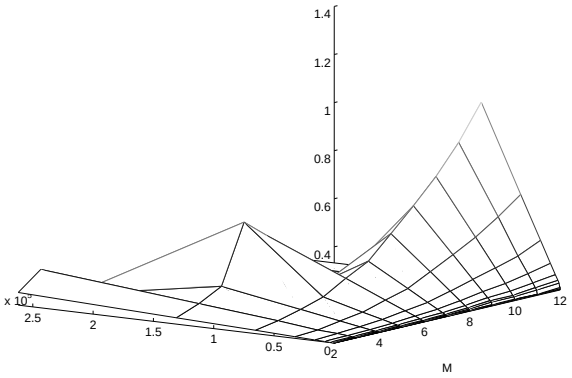
FIGURA 2.11. Speedup absoluto dos dois algoritmos para diferentes tamanhos de bloco. As figuras do topo correspondem ao Cluster I, às de baixo ao Cluster II.

redução cíclica apresenta um desempenho melhor que o algoritmo de divisão–e–conquista para tamanhos de bloco maiores (experimentos com $M \geq 6$ em arquiteturas mais sofisticadas podem ser encontrados em [**1, 18**]). Mas para M pequeno a situação se inverte, e esta diferença qualitativa que se observa quando M e P são pequenos aparentemente não foi relatada até o momento.

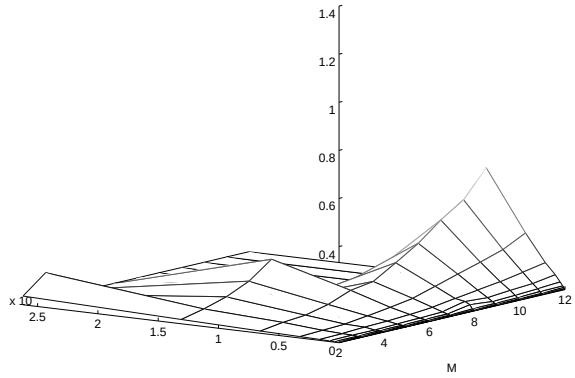
O tamanho $\{N, M\}$ do problema influencia o desempenho do algoritmo de mais de uma forma. As dependências óbvias foram mencionadas no Capítulo 2: valores maiores de N levam a cargas de trabalho maiores em cada elemento de processamento e conseqüentemente



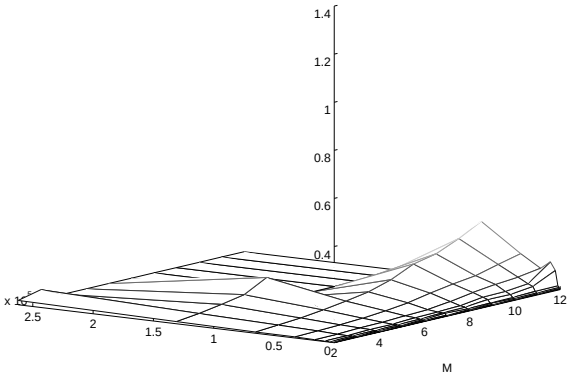
(a) Algoritmo Serial



(b) Divisão & Conquista ($P = 2$)



(c) Redução Cíclica ($P = 8$)



(d) Divisão & Conquista ($P = 8$)

FIGURA 2.12. Tempo gasto pelo Cluster I, em segundos, na solução de um sistema linear para diferentes valores de $M \in [2, 12]$ e N.

tornam os tempos de comunicação menos relevantes em comparação aos tempos de computação; se $N \gg P$ a etapa serial encontrada em ambos algoritmos é relativamente curta também; e o tempo de comunicação aumenta com o quadrado de M, de forma que o valor de M afeta o que pode ser considerado um valor grande de N nos dois pontos anteriores. Mas podemos ver nos resultados em dois clusters baseados em processadores diferentes que cada algoritmo sofre efeitos de gargalo na cache para valores diferentes de N e M em cada arquitetura. Esta influência mais sutil da arquitetura de hardware no desempenho de um programa não pode ser prevista, devendo ser detectada e tratada experimentalmente.

Embora não tenhamos incluído neste trabalho o estudo do erro numérico das soluções obtidas, vale mencionar que os valores de $p = 37$ e $\alpha = 1000$ usados nestes experimentos, conforme descrito na página 39, levaram a erros relativos quadráticos sempre inferiores a 10^{-15}. Um estudo mais detalhado da sensibilidade destes algoritmos aos dados pode vir a ser necessário caso os problemas aos quais eles venham a ser aplicados produzam matrizes mal–condicionadas; como este não é o caso das aplicações que temos em vista a curto e médio prazo, e há trabalhos mostrando as propriedades numéricas de algoritmos semelhantes ([**2, 27**]), postergamos este estudo.

CAPíTULO 3

Aplicações

Agora aplicamos os algoritmos paralelos estudados no capítulo anterior a um caso representativo da pesquisa em andamento. Na Seção 1, discutimos como aplicar o conhecimento sumarizado no Capítulo 2 ao problema de simulação do Capítulo 1. Na Seção 2 resolvemos uma instância particular do primeiro problema apresentado no Capítulo 1, e mostramos resultados numéricos. Na Seção 3, discutimos futuras linhas de pesquisa possibilitadas por estes novos resultados.

1. Diretivas de implementação e decisões estratégicas

No Capítulo 1, detalhamos o problema computacional a ser resolvido, ou seja, a repetida construção e solução de sistemas não–lineares, sendo a solução de um tal sistema usada para construir o próximo sistema a ser resolvido. Este aspecto iterativo tem duas origens: uma é intrínseca ao fenômeno sendo estudado, cujo domínio é o espaço e o tempo; o outro é a iteração de Newton para solução de sistemas não–lineares através de sucessivas aproximações lineares, ou seja, da solução de diversos sistemas lineares. Das equações discretizadas do modelo, obtivemos estes sistemas lineares na Seção 3 do Capítulo 1.

Estes sistemas lineares têm uma estrutura particular, ou seja, são bloco tridiagonais. No Capítulo 2, apresentamos e comparamos três algoritmos para resolver este tipo de sistema linear. Estes algoritmos foram testados isoladamente, isto é, com sistemas simples em vez daqueles obtidos das equações do modelo apresentadas no Capítulo 1. Nosso objetivo então era determinar o desempenho em paralelo apenas do algoritmo para solução de sistemas tridiagonais, sem misturar a paralelização das computações particulares ao modelo. As conclusões tiradas destes experimentos sugerem que o algoritmo de divisão–e–conquista da Seção 4 do Capítulo 2 é mais vantajoso se o tamanho dos blocos e o número de elementos

de processamento forem pequenos. Neste capítulo, nós combinamos este algoritmo com as equações do modelo do Capítulo 1 para produzir um simulador numérico paralelo.

Conforme foi discutido na página 18, nós usamos a abordagem de um estágio para resolver cada sistema linear em paralelo, o que tem como efeito colateral deixar a solução distribuída pelos elementos de processamento, ao mesmo tempo que requer que os dados de entrada sejam distribuídos da mesma forma. No problema que queremos resolver, o vetor de dados de entrada é o vetor de estado $\mathbf{u}^n$ a partir do qual obtemos o vetor solução $\mathbf{u}^{n+1}$. Se $\mathbf{u}^n$ encontra–se dividido em partições contíguas, o sistema (1.21) pode ser computado para cada uma destas partições localmente ao elemento de processamento que a contém. Comunicação é necessária apenas para trocar informações relativas aos extremos de cada partição.

Para ilustrar, agora descrevemos como os dados são distribuídos e trocados ao longo de uma iteração de Newton completa. No nível de tempo n, cada processador p guarda a porção do vetor $\mathbf{u}^n$ que corresponde a um subsistema $\mathbf{A}_p\mathbf{x}_p = \mathbf{y}_p$ na equação (2.45), que denotamos por $\mathbf{u}_p^n \in \mathbb{R}^{KM}$. Além disso, todos os processadores têm cópia dos valores de $\mathbf{u}_i^n$ correspondentes às $P-1$ linhas de acoplamento da equação (2.45): denotamos este vetor por $\mathbf{u}_\psi^n \in \mathbb{R}^{M(P-1)}$, ou seja, os valores do vetor de estado $\mathbf{u}^n$ correspondentes às linhas do sistema (2.45) que formarão o sistema de acoplamento (2.49). Cada processador pode computar sua matriz $\mathbf{A}_p$ e lado direito $\mathbf{y}_p$ pelas equações do Capítulo 1 usando $\mathbf{u}_p^n$, $\mathbf{u}_{\psi,p-1}^n$ e $\mathbf{u}_{\psi,p}^n$, exceto pelo primeiro processador, que usa a condição de contorno $\mathbf{u}_a$ e $\mathbf{u}_{\psi,1}^n$, e o último processador, que usa $\mathbf{u}_{\psi,P-1}^n$ e a condição de contorno $\mathbf{u}_b$. Os P sistemas são resolvidos em paralelo e os objetos na equação (2.46) são computados para serem transferidos para o primeiro processador ou processador mestre.

Entretanto, o processador mestre precisa de mais do que estes objetos para resolver o sistema de acoplamento. Os objetos $\mathbf{\Lambda}_p$, $\mathbf{\Gamma}_p$, $\mathbf{\Phi}_p$ and $\boldsymbol{\psi}_p$ da equação (2.50) ainda não foram computados, pois eles dependem de $\mathbf{u}_{p,K}^n$ e $\mathbf{u}_{p+1,1}^n$, que até agora só estavam disponíveis nos processadores p e $p+1$ respectivamente. Assim, cada processador deve transferir $\mathbf{u}_{p,1}^n$ e $\mathbf{u}_{p,K}^n$ juntamente com os objetos da equação (2.46) para o processador mestre. Só após ter recebido esta informação pode o processador mestre computar os coeficientes do sistema

(2.45) correspondentes às $P-1$ linhas que, juntamente com os objetos da equação (2.46), permitem montar o sistema de acoplamento (2.49). Da solução deste sistema obtemos os valores da solução global $\mathbf{u}_\psi^{n+1}$, que são então transferidos para todos os processadores. Cada processador usa este vetor para computar o trecho da solução $\mathbf{u}_p^{n+1}$ que lhe é local, de forma que ao final da iteração $\mathbf{u}^{n+1}$ encontra–se distribuído da mesma maneira que $\mathbf{u}^n$ encontráva–se no início, e uma nova iteração de Newton — ou um novo passo de tempo — pode ser iniciado. Não há necessidade de agregar todo o vetor de solução $\mathbf{u}$ em momento algum.

Deseja–se, é claro, coletar o vetor $\mathbf{u}$ para fins de saída do simulador. Para N grande, os trechos $\mathbf{u}_p$ da solução representam transferências de dados muito maiores do que toda a coleção de objetos requerida pelo procedimento de solução descrito acima. Esta comunicação comparativamente custosa provavelmente há de ocorrer a intervalos largamente espaçados da simulação, por exemplo a cada cinqüenta passos de tempo. Assim, o canal de comunicação terá uma carga extra a cada cinqüenta iterações, o que pode ter impacto no desempenho do programa uma vez que todo o vetor $\mathbf{u}$ pode requerer mais tempo para ser coletado no processador mestre (em comunicação assíncrona concomitante com o processamento) do que o tempo requerido para resolver cada sistema local $\mathbf{A}_p\mathbf{x}_p = \mathbf{y}_p$. Note que um impacto negativo análogo é gerado por saídas excessivas de simuladores numéricos para o disco local ou para a tela, isto é, a degradação do desempenho é intrínseca ao excesso de saída, e não ao fato dos dados a serem externados encontrarem–se distribuídos. Alternativamente, cada nó pode efetuar a saída de sua porção da solução para seu disco local, e a solução pode ser agregada por um outro canal de comunicação.

2. Modelo quadrático simplificado para escoamento trifásico

Implementamos uma versão bastante simplificada do modelo para escoamento trifásico apresentado na página 3, que é útil para analisar singularidades no espaço de fase da solução, mesmo que não capture tantos detalhes do fenômeno físico quanto o modelo mais detalhado dado pelas equações (1.1)–(1.3). Recordamos a equação diferencial parcial (1.1)

$$\frac{\partial s_w}{\partial t} + \frac{\partial}{\partial x} f_w(s_w, s_g) = D_w \quad \text{e} \quad \frac{\partial s_g}{\partial t} + \frac{\partial}{\partial x} f_g(s_w, s_g) = D_g,$$

onde o estado $\mathbf{u}(x,t) \in \mathbb{R}^2$ representa a saturação de água e óleo. Para descarregar a notação, usamos $\mathbf{u} = (u, v)$, então $u \equiv s_w$, $v \equiv s_o$ e $s_g = 1-u-v$. No modelo quadrático simplificado, o vetor *função de fluxo* $\mathbf{f}(\mathbf{u}) : \mathbb{R}^2 \to \mathbb{R}^2$ dado pela equação (1.2) é substituído por uma função quadrática homogênea, ou seja,

$$\begin{aligned} \mathbf{f}_u(u,v) &= \frac{1}{2}\, a_1 u^2 + b_1 uv + \frac{1}{2}\, c_1 v^2 + d_1 u + e_1 v \\ \mathbf{f}_v(u,v) &= \frac{1}{2}\, a_2 u^2 + b_2 uv + \frac{1}{2}\, c_2 v^2 + d_2 u + e_2 v, \end{aligned} \tag{3.1}$$

e a matriz *função de difusão* $\mathbf{g}(\mathbf{u}) : \mathbb{R}^2 \to \mathbb{R}^{2\times 2}$ na equação (1.3) é chamada *matriz de viscosidade* e freqüentemente é tomada como a identidade multiplicada por uma constante ϵ, isto é,

$$\mathbf{g}(\mathbf{u}) = \epsilon \mathbf{I}.$$

Para estas formas particulares de $\mathbf{f}$ e $\mathbf{g}$, as expressões para $\mathbf{f}'$ são

$$\frac{\partial \mathbf{f}}{\partial \mathbf{u}} = \begin{bmatrix} \dfrac{\partial \mathbf{f}_u}{\partial u} & \dfrac{\partial \mathbf{f}_u}{\partial v} \\ \dfrac{\partial \mathbf{f}_v}{\partial u} & \dfrac{\partial \mathbf{f}_v}{\partial v} \end{bmatrix} = \begin{bmatrix} a_1 u + b_1 v + d_1 & b_1 u + c_1 v + e_1 \\ a_2 u + b_2 v + d_2 & b_2 u + c_2 v + e_2 \end{bmatrix}, \tag{3.2}$$

e $\mathbf{g}' = 0$. Como $\mathbf{g}$ é constante, o esquema numérico dado pela expressão (1.16)–(1.17) reduz–se a

$$\frac{\mathbf{u}_i^{n+1}}{k} + \frac{\mathbf{f}_{i+1}^{n+1} - \mathbf{f}_{i-1}^{n+1}}{4h} - \frac{\mathbf{g}}{2h^2}(\mathbf{u}_{i+1}^{n+1} - 2\mathbf{u}_i^{n+1} + \mathbf{u}_{i-1}^{n+1}) = \tag{3.3}$$

$$\frac{\mathbf{u}_i^{n}}{k} - \frac{\mathbf{f}_{i+1}^{n} - \mathbf{f}_{i-1}^{n}}{4h} + \frac{\mathbf{g}}{2h^2}(\mathbf{u}_{i+1}^{n} - 2\mathbf{u}_i^{n} + \mathbf{u}_{i-1}^{n}), \tag{3.4}$$

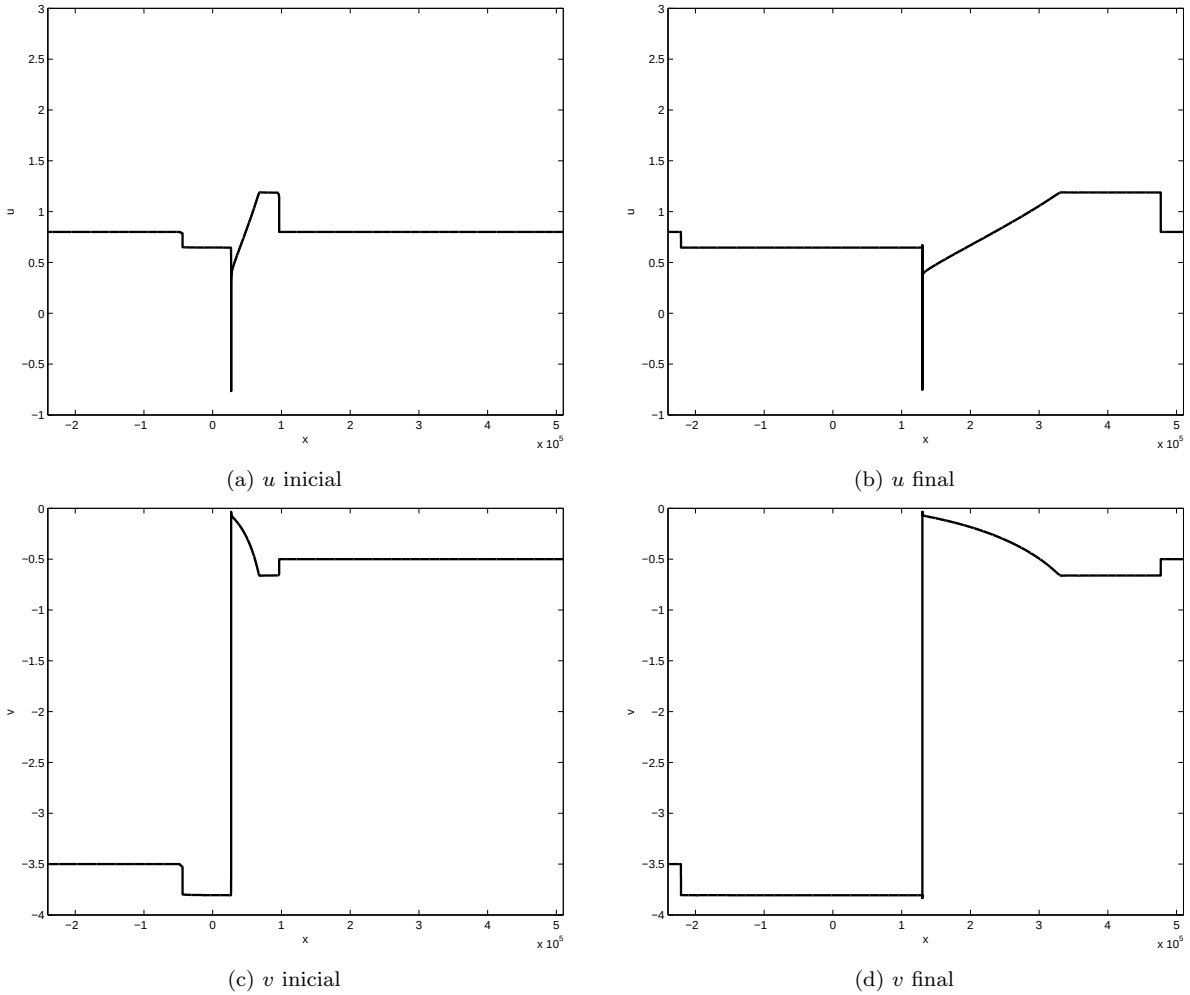
e as expressões para a solução iterativa (1.18) e (1.20) tornam–se

$$\begin{aligned} \mathbf{F}(\mathbf{u}_i^{n+1}) &= \frac{\mathbf{u}_i^{n+1}}{k} + \frac{\mathbf{f}_{i+1}^{n+1} - \mathbf{f}_{i-1}^{n+1}}{4h} - \frac{\mathbf{g}}{2h^2}(\mathbf{u}_{i+1}^{n+1} - 2\mathbf{u}_i^{n+1} + \mathbf{u}_{i-1}^{n+1}) \\ \mathbf{y}(\mathbf{u}_i^{n}) &= \frac{\mathbf{u}_i^{n}}{k} - \frac{\mathbf{f}_{i+1}^{n} - \mathbf{f}_{i-1}^{n}}{4h} + \frac{\mathbf{g}}{2h^2}(\mathbf{u}_{i+1}^{n} - 2\mathbf{u}_i^{n} + \mathbf{u}_{i-1}^{n}) \end{aligned} \tag{3.5}$$

e

$$\begin{aligned} \mathbf{C}_i &= \frac{1}{4h}\frac{\partial \mathbf{f}}{\partial \mathbf{u}} - \frac{\mathbf{g}}{2h^2} \\ \mathbf{A}_i &= \frac{1}{k}\mathbf{I} + \frac{\mathbf{g}}{h^2} \\ \mathbf{B}_i &= -\frac{1}{4h}\frac{\partial \mathbf{f}}{\partial \mathbf{u}} - \frac{\mathbf{g}}{2h^2} \end{aligned} \tag{3.6}$$

respectivamente, com $\mathbf{f}$ e $\partial\mathbf{f}/\partial\mathbf{u}$ dadas por (3.1) e (3.2).



(a) u inicial (b) u final

(c) v inicial (d) v final

FIGURA 3.1. À esquerda, dados iniciais de uma simulação demorada com 2×10^6 pontos de grade, cujos resultados finais são mostrados à direita. Note o deslocamento do choque para a direita.

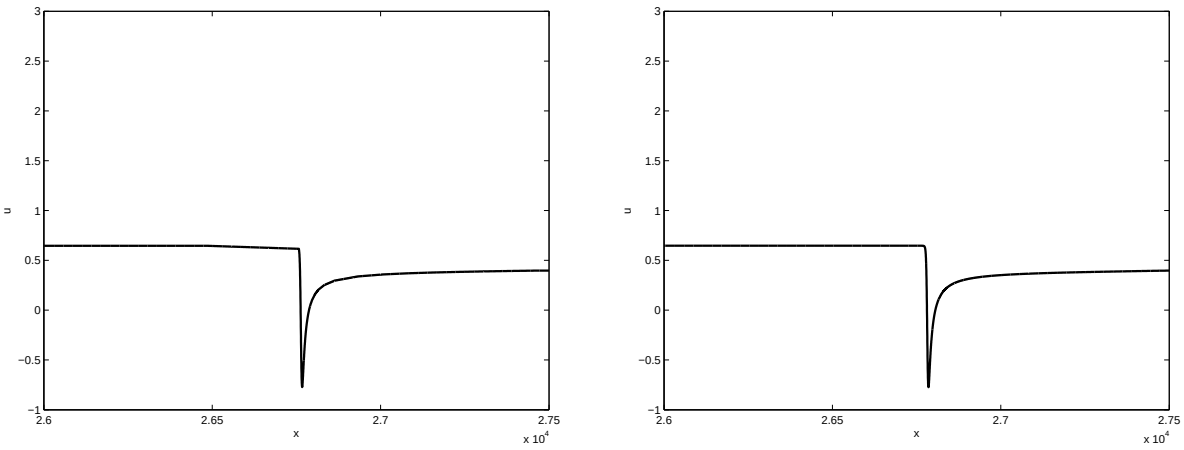
Um exemplo de simulação computacionalmente cara é mostrado na Figura 3.1. O perfil dos dados iniciais contém um choque pronunciado na vizinhança de $x = 26770$, onde o valor de u é mínimo, que se desloca para a direita durante a simulação a uma velocidade constante. O domínio sobre o qual precisamos calcular a solução numérica precisa ter um comprimento ordens de grandeza superior ao comprimento da região de interesse, devido ao deslocamento desta durante a simulação. Por outro lado, a grande variação do valor do estado na região de interesse relativamente pequena, ou seja, o fato de du/dx ter um valor absoluto muito alto, torna necessário o uso de uma malha com espaçamento h pequeno. Os resultados na

Figura 3.1 foram obtidos com uma malha de 2×10^6 pontos. O tempo consumido pelo simulador para evoluir o estado inicial exibido na Figura 3.1(a) para o estado final exibido na Figura 3.1(b) é mostrado na Tabela 3.1.

	Número de processadores					
	1		2	4	8	16
Máquina	Antigo	Novo	Speedup			
Cluster I	20:54:42	5:28:25	1.66	2.63	4.53	9.45

TABELA 3.1. Tempo gasto por cada programa na simulação apresentada como exemplo, e os speedups relativos obtidos pelo programa paralelo.

Estando de posse do resultado desta simulação pudemos aferir a velocidade de propagação do choque $dx/dt \equiv \overline{V} = 0.65$, o que nos permitiu restringir o simulador à região de interesse. Usando uma mudança de referencial na definição da função de fluxo dada na equação (3.1), pudemos executar uma nova simulação num intervalo de limites próximos ao choque, pois cancelamos sua velocidade de propagação. A Figura 3.2 mostra os tempos inicial e final da segunda simulação: o choque praticamente não se desloca.

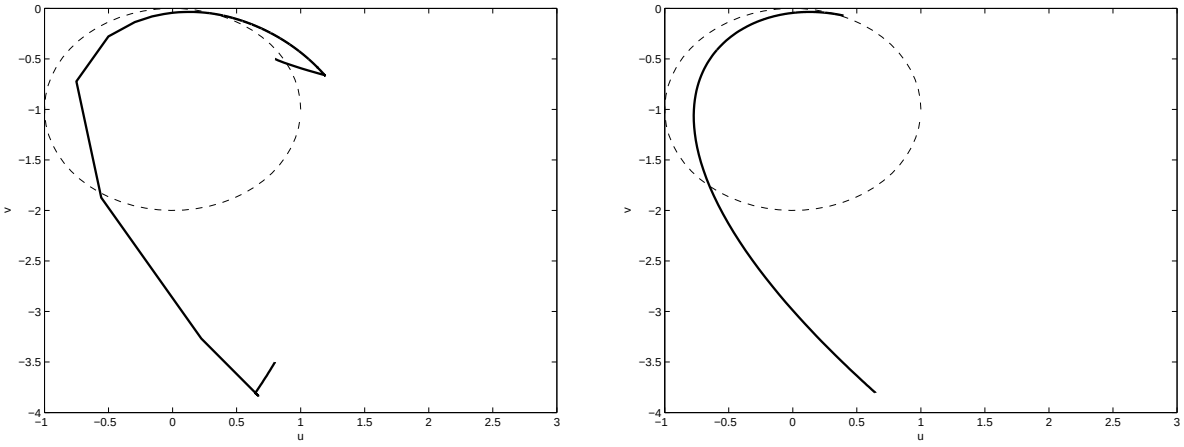


(a) u inicial (b) u final

FIGURA 3.2. Dados iniciais (a) e resultados finais (b) de uma simulação rápida definida pelos resultados obtidos da uma simulação anterior. Note que o choque não se desloca, e o perfil se mantém.

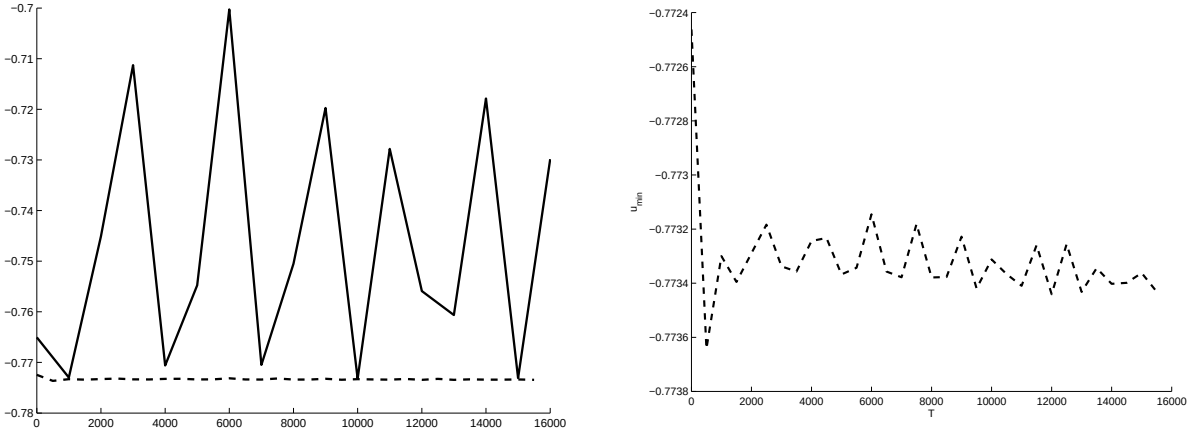
A segunda simulação foi efetuada em muito menos tempo, mas numa resolução relativamente muito maior: os 8000 pontos de malha empregados no intervalo $[26000, 27500]$

estavam espaçados de $h = 0.1875$, enquanto na simulação original tínhamos $h = 3.75$. Este refinamento reflete–se diretamente na qualidade obtida para o perfil da estrutura interna do choque no plano de fase, mostrado na Figura 3.3, e na diminuição da oscilação numérica mostrada na Figura 3.4. A amplitude das oscilações foi reduzida por um fator de 60.



(a) $h = 3.75$ (b) $h = 0.1875$

FIGURA 3.3. Perfil do choque ao fim de cada simulação no plano de fase $u \times v$. A figura da direita, correspondente à segunda simulação, captura toda a região de interesse com qualidade superior.



(a) $\min_x u(t)$ (b) $\min_x u(t)$

FIGURA 3.4. Oscilação do valor mínimo da componente u do estado, ao longo das duas simulações. Na figura da esquerda, a linha sólida corresponde à simulação com h maior. Nas duas figuras, a linha tracejada representa os valores correspondentes à simulação com h menor.

Nós analizamos os perfis de tempo de execução serial do simulador usando diferentes tamanhos de problema $\{N, 2\}$, e o intervalo para as razões entre o tempo gasto resolvendo

cada sistema linear e o tempo total requerido para calcular os coeficientes de cada sistema e resolvê–lo foi $[0.64, 0.70]$, isto é, quando utilizamos um único processador, entre 64% e 70% to tempo de execução total corresponde ao código que resolve os sistemas bloco tridiagonais. Uma vez que a computação da matriz de coeficientes e do lado direito de cada sistema é trivialmente paralelizável, podemos esperar que o simulador tenha um speedup absoluto maior que aqueles obtidos para os programas que simplesmente resolvem um sistema, como no capítulo anterior. Se r é a fração do tempo total de execução correspondente à rotina de solução, e portanto $(1 - r)$ é a fração correspondente à preparação de cada sistema linear, e se denotarmos por $T_{\text{sim}}(1)$ o tempo de execução num único processador, então

$$T_{\text{solve}}(1) = rT_{\text{sim}}(1) \quad \text{e} \quad T_{\text{setup}}(1) = (1 - r)T_{\text{sim}}(1) \tag{3.7}$$

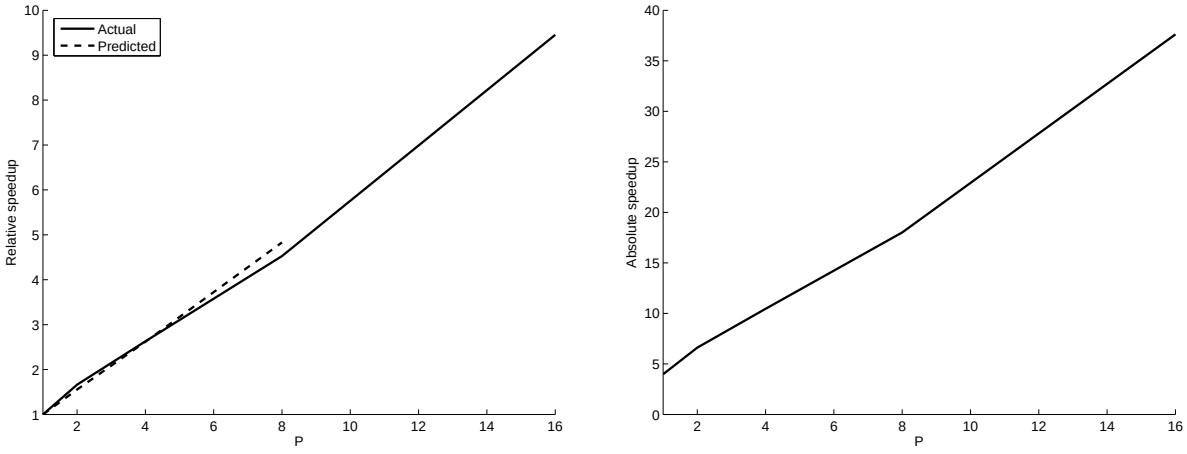
são os tempos correspondentes às duas etapas de cada iteração; denotando por $S(P)$ o speedup absoluto da rotina de solução em P processadores, medido no Capítulo 2, podemos prever os valores correspondentes quando o simulador executa em P processadores como

$$T_{\text{solve}}(P) = rT_{\text{sim}}(1)/S(P) \quad \text{e} \quad T_{\text{setup}}(P) = (1 - r)T_{\text{sim}}(1)/P. \tag{3.8}$$

Juntando as expressões (3.7) e (3.8) temos que o speedup teórico do simulador é

$$\frac{T_{\text{sim}(1)}}{T_{\text{sim}(P)}} = \frac{P}{rP/S(P) + (1 - r)}, \tag{3.9}$$

que depende apenas do speedup do algoritmo de solução e da fração do tempo de execução serial correspondente a ele. O valor r pertence ao intervalo $[0, 1]$. Ele jamais é 0 ou 1: conforme $r \to 0$, o tempo gasto na solução dos sistemas é desprezível em comparação ao tempo para calcular seus coeficientes, e o speedup tende a P; conforme $r \to 1$, as rotinas de solução tomam todo o tempo de execução e o speedup torna-se $S(P)$. Para os valores de $r \in [0.64, 0.70]$ obtidos e os speedups medidos para o algoritmo de divisão–e–conquista, a expressão (3.9) indica que o simulador deverá apresentar um speedup relativo no intervalo $[4.83, 5.00]$ usando oito processadores. O valor efetivamente obtido de 4.53 pode ser devido a diversos fatores, como a já mencionada falta de precisão do relógio interno, ou à falta de precisão do sistema de avaliação do perfil de tempo de execução, ou mesmo ao seu caráter invasivo.



(a) (b)

FIGURA 3.5. À esquerda, o speedup relativo do novo simulador no Cluster I. À direita, o speedup absoluto em relação ao simulador antigo.

3. Trabalhos futuros

Dois exemplos foram dados como motivação no Capítulo 1: um muito simples, com tamanho de bloco $M = 2$ mínimo, que usamos como caso de estudo e para o qual desenvolvemos um simulador piloto; e outro mais complicado, com tamanho de bloco $M = 5$ maior, e equações constitutivas mais custosas que não apresentamos em detalhe por estarem fora do escopo deste trabalho: basta mencionar que requerem a computação de mais do que os polinômios homogêneos dados na equação (3.1).

O simulador piloto apresentado na seção anterior já é uma ferramenta útil para análise qualitativa, e imediatamente aplicável à investigação de fenômenos conduzida atualmente, cujos resultados serão apresentados em [**25**]. Este novo simulador tem um desempenho muito superior ao do software que vinha sendo utilizado antes deste trabalho ser iniciado, mesmo que o paralelismo não seja explorado: mas este ganho pode ser ainda multiplicado por um speedup relativo de 9.45, resultando num speedup absoluto de 37.6 como mostra a Figura 3.5. O tempo de relógio efetivamente gasto numa simulação particularmente longa caiu de mais de vinte horas para menos de quarenta minutos, ou seja, houve um grande ganho na interação entre os pesquisadores e o simulador.

Os diversos problemas de natureza técnica apresentados na Seção 6 do Capítulo 2 serão estudados à medida que se mostrarem conflitantes com o bom prosseguimento da aplicação destes algoritmos. Para os problemas em estudo no momento, ou seja, para os valores de $\{N, M\}$ cuja demanda é imediata, as implementações existentes não apresentam comportamento degenerado por peculiaridades de hardware. No futuro, quando viermos a usar novas máquinas paralelas, os procedimentos experimentais desenvolvidos para avaliar os algoritmos servirão como ferramentas diagnósticas. Em particular, a tendência de mercado quando da conclusão deste trabalho é tornarem-se cada vez mais acessíveis as máquinas de memória compartilhada, cujo modelo de programação é mais simples, e nas quais os diversos gargalos relativos a comunicação que investigamos neste trabalho praticamente não se manifestam. Assim, todo o trabalho aqui desenvolvido se aplica prontamente a este novo cenário tecnológico.

O avanço da investigação numérica de problemas nesta classe para valores maiores de M teve sempre como obstáculo o aumento no custo computacional. Observe nas expressões de complexidade computacional dadas nas páginas 33–36 que o tempo necessário para a solução de um sistema bloco tridiagonal cresce com M^3, e que o custo da construção de um tal sistema cresce pelo menos com M^2, e no máximo com M^3. Assim, se estendermos o simulador piloto, no qual $M = 2$, para um problema onde $M = 5$ como o segundo problema apresentado no Capítulo 1, podemos esperar que o tempo de computação aumente por um fator entre $5^2/2^2 = 6.25$ e $5^3/2^3 \approx 15.6$, dependendo da complexidade das equações constitutivas (ver discussão na página 61). Este fator ainda é menor que o speedup absoluto obtido por nós, e a aplicação do conhecimento e das técnicas neste trabalho permite construir simuladores para estes modelos mais complexos que executarão, para uma mesma resolução de malha N, em menos tempo que o simulador anterior executava para o modelo mais simples.

Modelos de interesse no cenário de meios porosos incluem o modelo para injeção de nitrogênio e vapor para remoção de poluentes, apresentado brevemente no Capítulo 1, e o modelo para injeção de oxigênio em reservatórios de petróleo para causar combustão in–situ e subseqüente alteração do óleo pesado e recuperação mais eficiente [**10**]. Estas duas aplicações

vêm sendo estudadas há anos sob aspectos analíticos, mas a conclusão deste trabalho em muito aumentou a viabilidade de suas simulações. Resultados numéricos podem ser esperados num futuro próximo.

Também no futuro próximo, o desempenho destas rotinas especializadas deve ser avaliado e seu desempenho comparativo ao de pacotes bem difundidos (por exemplo, PETSc, ScaLAPACK e HYPRE, entre outros, todos disponíveis na Internet) e divulgado para usuários potenciais. De certa forma, a falta de tais rotinas especializadas acompanhadas deste estudo comparativo foi a motivação técnica para este trabalho.

A um prazo mais longo, de posse dos bons fundamentos para a solução computacional do caso mais simples obtidos neste trabalho, podemos agora considerar a introdução de novas facilidades que irão aumentar, por um lado, a complexidade do problema formal e do algoritmo para sua solução, e por outro, a flexibilidade e aplicabilidade da técnica. O problema computacional formalizado no Capítulo 1 era tão simples quanto um problema de diferenças finitas pode ser, ou seja, uma discretização unidimensional com pontos nodais igualmente espaçados. A física escolhida para este caso de estudo foi também a mais simples, pois o estado $\mathbf{u}$ tem o mesmo número de componentes e é governado pelas mesmas equações em todo o domínio físico. Permitir heterogeneidade na forma de distribuição dos pontos nodais permite simulações muito mais rápidas, com relativamente pouca complexidade adicionada ao simulador em si; entretanto, permitir heterogeneidade na estrutura em blocos, ou seja, que os blocos tenham tamanhos diferentes, tem impacto imediato nas discretizações aqui apresentadas, requerendo uma análise numérica mais profunda dos erros associados, e trazem dificuldades técnicas tais como distribuição de carga adequada, as quais não abordamos neste trabalho. Finalmente, a aplicação do conhecimento aqui agregado a simulações bidimensionais baseadas na abordagem de discretização por diferenças finitas usada neste trabalho é um campo ainda muito inexplorado em diversos grupos de pesquisa, em parte devido aos custos computacionais, em parte devido à complexidade de modelagem e formalização dos algoritmos.

CAPíTULO 4

Conclusões

Nos últimos anos, pouca atenção foi dada ao bom uso de clusters de computadores, cada vez mais comuns no meio acadêmico. Clusters com até dezesseis ou trinta e dois processadores são facilmente encontrados em instituições de pesquisa hoje em dia, mas sua exploração é limitada devido à dificuldade técnica de programar para uma máquina paralela. Estes clusters costumam ser compostos de simples computadores pessoais e switches de rede dedicados, e se eles talvez apresentam um desempenho inferior quando comparados aos clusters de arquitetura fechada oferecidos pelos fabricantes de máquinas de grande porte sob a alcunha "supercomputador de baixo custo", eles facilmente compensam na relação de custo e benefício.

Apresentamos uma classe de problemas que leva a um tipo específico de sistema linear, cuja estrutura é bloco tridiagonal, com dimensões $\{N, M\}$ num intervalo de interesse particular ($N \in [10^2, 10^9]$, $M \in [2, 12]$) para o qual poucos resultados de avaliação de desempenho encontram–se disponíveis: problemas relativamente pequenos, ou seja, que geram pouco volume de cálculo, e que portanto têm uma expectativa baixa de paralelização eficiente.

Estudamos e comparamos o desempenho de dois algoritmos paralelos para a solução deste tipo de sistema linear para este intervalo de dimensões. Nossos resultados foram compatíveis com aqueles apresentados em trabalhos anteriores, mas estes trabalhos anteriores não analisaram o desempenho destes algoritmos no nosso intervalo de interesse em particular, nem nas arquiteturas que temos em vista. Nossos experimentos mostram que o desempenho dos dois algoritmos muda qualitativamente para valores muito pequenos de M e valores relativamente pequenos de N. Além disso, poucos trabalhos mostram resultados comparativos entre máquinas ou algoritmos, o que nos motivou a fazer as análises apresentadas no Capítulo 2, onde mostramos vantagens e desvantagens de cada algoritmo em diferentes situações.

Com o desenvolvimento da tecnologia dos computadores nas últimas décadas, as abordagens para a solução paralela de sistemas lineares mudaram de uma granularidade fina para uma mais grossa: aumentar a carga computacional em cada elemento de processamento torna os tempos de comunicação, em comparação, desprezíveis; ao mesmo tempo, com os avanços nos sistemas de rede, o roteamento entre processadores deixou de ser uma preocupação maior na concepção dos algoritmos paralelos. Fatores como roteadores com buffers, memória hierarquizada, compiladores especializados e middleware específico para computação paralela diminuíram consideravelmente a dificuldade em produzir programas paralelos economicamente eficientes, ou seja, hoje é mais fácil escrever programas que usam eficientemente a máquina paralela em que executam sem perder um bom nível de abstração das suas peculiaridades técnicas ([**15**]). Entretanto, a solução paralela de problemas pequenos tira pouca vantagem destas novas tecnologias. Problemas pequenos não geram grandes cargas computacionais que tornem o tempo de computação comparativamente irrelevante, de forma que para beneficiar–se de um ambiente paralelo na sua solução é preciso ter mais controle sobre a comunicação. A solução paralela de problemas grandes foi implementada com sucesso numa ampla gama de arquiteturas, desde supercomputadores vetoriais a clusters de estações de trabalho. Neste trabalho, temos em vista problemas pequenos nestas máquinas de desempenho mais baixo.

O algoritmo de redução cíclica apresentado na Seção 3 do Capítulo 2 é muito complexo para implementar e manter, se comparado a formulações como a de divisão–e–conquista que apresentamos neste trabalho. Apesar de mostrar grande eficiência e paralelismo para problemas maiores e grande número de processadores, os resultados experimentais do Capítulo 2 não indicam nenhuma vantagem clara na aplicação deste algoritmo no lugar da formulação que desenvolvemos na Seção 4 do Capítulo 2. Nossa formulação é mais simples que qualquer outra encontrada na literatura técnica, e pode facilmente ser estendida para um espectro mais amplo de problemas, por exemplo, para acomodar blocos de diferentes tamanhos. Para valores de P pequenos, sua redundância é consideravelmente inferior à do algoritmo de redução cíclica, o que o torna bastante competitivo em aplicações de menor escala. Outra

vantagem desta abordagem apontada em [**27**] vale para esta versão mais simples também: a paralelização é obtida através da distribuição do sistema linear resultante da discretização do modelo, e não do modelo discreto em si. Isto torna o algoritmo prontamente aplicável a qualquer simulador serial a um custo mínimo de reestruturação, uma vez que as modificações estão restritas às rotinas de solução de sistemas lineares.

Os resultados contidos nesta tese, nominalmente a formalização dos três algoritmos em uma mesma notação e suas respectivas análises de complexidade, são uma compilação até então inédita e que há de ser útil para outros indivíduos que não os envolvidos diretamente com este trabalho. Em particular, há pouquíssimo material sobre sistemas bloco tridiagonais na Língua Portuguesa. Além disso, o código desenvolvido ao longo desta pesquisa é em grande parte reaproveitável, já havendo inclusive módulos para MATLAB à disposição do público.

Os programas desenvolvidos no decorrer desta pesquisa encontram–se disponíveis no endereço `<http://www.impa.br/~hime/blocktri>`. O código fonte e os scripts usados para executar os testes seguem a notação empregada nesta dissertação até onde permitido pela sintaxe das linguagens de programação. Resultados técnicos deste trabalho, apresentados no Capítulo 2, foram submetidos e aceitos no VII IEEE International Symposium on Cluster Computing and the Grid, a ser realizado antes da conclusão e defesa desta dissertação. Resultados de aplicações, como os apresentados no Capítulo 3, serão apresentados posteriormente no X Workshop on Partial Differential Applications.

APÊNDICE A

Análise de complexidade de rotinas básicas de álgebra linear

.

As expressões introduzidas em (2.51) e (2.59) aproximam a contagem de operações de ponto flutuante de quatro procedimentos de álgebra linear básica como funções do tamanho do problema M. O valor exato desta contagem de operações depende da implementação: nós usamos LAPACK/BLAS para executar a álgebra linear de baixo nível. Neste apêndice, analisamos as rotinas relevantes, ou seja, as implementações codificadas, para validar as conclusões que tiramos das aproximações dadas em (2.51) no intervalor de M que consideramos.

As rotinas das bibliotecas LAPACK/BLAS possuem longos preâmbulos de configuração e código repetido para acomodar diferentes entradas e flexibilizarem–se, aceitando uma mesma rotina uma matriz ou sua transposta, por exemplo. Analisamos apenas um tal caso de cada rotina: para fins práticos, a contagem de operações é a mesma em todos eles. Além disso, em nossas chamadas as matrizes que multiplicamos e para as quais resolvemos sistemas lineares são sempre quadradas. Nós reescrevemos as rotinas em FORTRAN em pseudo-código para fins de legibilidade. A versão final de nosso sofware usou rotinas C inline que implementam as variantes específicas de cada rotina do LAPACK utilizada, ou seja, usamos as formulações dos algoritmos do LAPACK, e não suas implementações.

1. Fatoração LU

Para $M \leq 32$, LAPACK usa uma rotina mais simples (`dgetf2.f`) para realizar fatoração LU, que implementa o algoritmo abaixo:

`for` $j = 1, \ldots, M$	
$j_{\texttt{piv}} = j - 1 +$ `argmax` $_{k \in [1, M-j+1]} \; (A_{j+k-1,j})$	$M - j$ comparações
`checar se o sistema é singular`	
`if` $j_{\texttt{piv}} \neq j$	
`swap` $A_{j,k} \leftrightarrow A_{j_{\texttt{piv}},k}, \; k = 1, \ldots, M$	
`if` $j < M$	
$\alpha \leftarrow \frac{1}{A_{j,j}}$	uma divisão
$A_{k,j} \leftarrow \alpha A_{k,j}, \; k = j+1, \ldots, M$	$M - j$ operações
`for` $i = j+1, \ldots, M$	$M - j$ passagens com:
$\alpha = -A_{i,j}$	um complemento
$A_{i,k} \leftarrow A_{i,k} + \alpha A_{j,k}, \; k = j+1, \ldots, M$	$2(M-j)$ operações

Ao terminar, o procedimento acima terá armazenado os fatores $\mathbf{L}$ e $\mathbf{U}$ no espaço que originalmente continha a entrada $\mathbf{A}$. A diagonal contém elementos de $\mathbf{U}$; a diagonal de $\mathbf{L}$ é unitária e não precisa ser armazenada. O segundo α é supérfluo e poderia ser descartado, mas o LAPACK implementa o algoritmo assim, usando subrotinas do BLAS (`dger.f` e `dscal.f`). A contagem de operações deste algoritmo é dada por

$$\sum_{j=1}^{M-1} 1 + (M-j)\big(3 + 2(M-j)\big) = \frac{4M^3 + 3M^2 - M - 6}{6}.$$

Note que a expressão acima anula–se para $M = 1$. Isto é consistente no sentido de que a inversa $\mathbf{A}^{-1}$ não é de fato computada, cujo análogo escalar seria computar o inverso multiplicativo de um escalar a. Isto também é consistente com os resultados dados na seção seguinte.

2. Solução usando fatoração LU

Usando a saída da rotina anterior, a rotina encontrada em `dgetrs.f` — que de fato é uma rotina de interface para outra (ver `dtrms.f`) que resolve sistemas triangulares — executa as duas passagens de retro–substituição da seguinte forma:

`for` $k = M, \ldots, 1$	
$y_k \leftarrow y_k / U_{k,k}$	uma divisão
$y_i \leftarrow y_i - y_k U_{i,k}, \; i = 1, \ldots, K-1$	$2(k-1)$ operações
`for` $k = 1, \ldots, M$	
$y_i \leftarrow y_i - y_k L_{i,k}, \; i = k+1, \ldots, M$	$2(M-k)$ operações

A entrada $\mathbf{y}$ é sobreescrita com a solução do sistema $\mathbf{Ax} = \mathbf{y}$. A contagem de operações deste algoritmo é dada por

$$\sum_{k=1}^{M} 1 + 2(k-1) + 2(M-k) = 2M^2 - M.$$

Para $M = 1$, a expressão acima tem valor 1, para a operação escalar de determinar x tal que $ax = y$. Isto resume–se a uma divisão, consistentemente com o resultado obtido na seção anterior.

3. Multiplicação e adição de matrizes e vetores

As implementações de produtos no BLAS realizam ambos soma e multiplicação em uma única chamada, substituindo o argumento de entrada $\mathbf{C}$ por $\beta\mathbf{C} + \alpha\mathbf{AB}$ no caso do produto matriz–matriz, ou $\mathbf{y}$ por $\beta\mathbf{y} + \alpha\mathbf{Ax}$ no caso do produto matriz–vetor. A saber, as rotinas às quais nos referimos encontram–se nos arquivos `dgemm.f` e `dgemv.f`. Nas chamadas que fazemos a estas rotinas, o escalar β é sempre zero ou unitário, o que reduz a contagem de operações em M^2 nos produtos matriz–matriz e em M nos produtos matriz–vetor. Ambos os algoritmos são triviais:

`for` $j = 1, \ldots, M$	
`for` $\ell = 1, \ldots, M$	
$t \leftarrow \alpha B_{\ell,j}$	uma multiplicação
$C_{i,j} \leftarrow C_{i,j} + tA_{i,\ell},\ i = 1, \ldots, M$	$2M$ operações

Isto corresponde a $2M^3 + M^2$ operações para $\mathbf{C} \leftarrow \mathbf{C} + \alpha\mathbf{AB}$; para $\mathbf{y} \leftarrow \mathbf{y} + \alpha\mathbf{Ax}$, o algoritmo

`for` $j = 1, \ldots, M$	
$t \leftarrow \alpha x_j$	uma multiplicação
$y_i \leftarrow y_i + tA_{i,j},\ i = 1, \ldots, M$	$2M$ operações

executa $2M^2 + M$ operações. Para $M = 1$, ambas as expressões valem 3 e representam a mesma operação escalar $c \leftarrow c + \alpha ab$. Por outro lado, se i varia de 1 a KM, ou seja, no caso de uma matriz $KM \times M$ ser multiplicada por um vetor em $\mathbb{R}^M$, a contagem de operações passa a ser $2KM^2 + M$, conforme na expressão (2.59).

APÊNDICE B

Procedimento Experimental para Medida de Tempos

.

Neste apêndice nós detalhamos a metodologia usada para aferir os tempos usados no cálculo dos fatores de speedup e redundância apresentados na Seção 6 do Capítulo 2. Os valores aferidos para os Clusters I e II encontram-se nas Tabelas B.1 e B.2, respectivamente.

Rotinas de relógio em geral, providas pelo sistema operacional ou por algum middleware como o MPI, devolvem um valor relativo a um referencial, como por exemplo o tempo

Algoritmo	Serial	Redução Cíclica				Divisão–e–conquista						
Implementação	Serial	Serial	Paralela			Serial				Paralela		
	1	1	2	4	8	1	2	4	8	2	4	8
$M = 2$, $N = 262144$	0.074	0.225	0.113	0.074	0.036	0.098	0.120	0.129	0.134	0.052	0.033	0.018
$M = 3$, $N = 262144$	0.180	0.570	0.284	0.160	0.111	0.230	0.279	0.303	0.317	0.127	0.078	0.041
$M = 4$, $N = 131072$	0.158	0.491	0.225	0.129	0.068	0.201	0.246	0.267	0.278	0.113	0.067	0.036
$M = 5$, $N = 131072$	0.245	0.789	0.370	0.207	0.111	0.311	0.381	0.415	0.436	0.358	0.104	0.055
$M = 6$, $N = 131072$	0.407	1.159	0.530	0.298	0.180	0.507	0.605	0.658	0.693	0.281	0.166	0.104
$M = 7$, $N = 65536$	0.290	0.786	0.366	0.210	0.105	0.350	0.415	0.453	0.483	0.191	0.115	0.062
$M = 8$, $N = 65536$	0.441	1.193	0.542	0.295	0.161	0.525	0.620	0.669	0.713	0.284	0.168	0.090
$M = 9$, $N = 65536$	0.579	1.599	0.759	0.415	0.220	0.691	0.809	0.884	0.959	0.377	0.225	0.134
$M = 10$, $N = 65536$	0.764	2.036	0.968	0.515	0.295	0.896	1.028	1.112	1.195	0.477	0.274	0.154
$M = 11$, $N = 65536$	0.959	2.599	1.229	0.654	0.356	1.112	1.295	1.394	1.495	0.596	0.354	0.195
$M = 12$, $N = 65536$	1.218	3.194	1.505	0.836	0.469	1.408	1.595	1.724	1.864	0.742	0.424	0.245
$M = 13$, $N = 32768$	0.737	1.922	0.915	0.497	0.259	0.841	0.970	1.080	1.232	0.456	0.280	0.158
$M = 14$, $N = 32768$	0.883	2.270	1.074	0.564	0.303	1.007	7.315	10.043	10.601	6.667	3.314	1.534
$M = 15$, $N = 32768$	1.036	2.743	1.298	0.702	0.371	1.180	1.385	1.577	1.885	0.663	0.415	0.253
$M = 16$, $N = 32768$	1.225	3.276	1.553	0.852	0.468	1.392	1.610	1.858	2.237	0.760	0.479	0.298
$M = 17$, $N = 32768$	1.410	3.665	1.716	0.933	0.523	1.588	1.881	2.172	2.674	0.908	0.566	0.371
$M = 18$, $N = 32768$	1.613	4.275	2.076	1.061	0.607	1.832	2.118	2.436	2.898	1.014	0.629	0.390
$M = 19$, $N = 32768$	1.866	4.888	2.343	1.224	0.665	2.083	2.434	2.784	3.306	1.152	0.715	0.431
$M = 20$, $N = 32768$	2.110	5.515	2.609	1.755	0.718	2.379	2.837	3.389	4.332	1.373	0.883	0.564
$M = 21$, $N = 32768$	2.417	6.214	2.940	1.539	0.846	2.708	4.518	5.440	5.851	2.627	1.540	0.774
$M = 22$, $N = 32768$	2.703	7.017	3.333	1.738	0.899	3.013	3.502	4.103	5.073	1.690	1.041	0.635
$M = 23$, $N = 16384$	1.502	3.971	1.905	0.993	0.534	1.677	2.094	2.709	3.807	1.057	0.745	0.540
$M = 24$, $N = 16384$	1.680	4.369	2.095	1.138	0.595	1.859	2.373	3.075	4.407	1.183	0.847	0.626
$M = 25$, $N = 16384$	1.867	4.894	2.368	1.238	0.641	2.074	2.597	3.438	4.931	1.327	0.952	0.685
$M = 26$, $N = 16384$	2.114	5.408	2.586	1.342	0.713	2.320	27.963	34.393	25.763	26.386	11.285	3.617
$M = 27$, $N = 16384$	2.299	6.034	2.901	1.491	0.781	2.497	3.229	4.365	6.500	1.653	1.246	0.887
$M = 28$, $N = 16384$	2.507	6.595	3.168	2.341	0.849	2.777	3.628	4.935	7.428	1.878	1.402	1.012
$M = 29$, $N = 16384$	2.781	7.226	3.489	1.827	1.373	3.036	5.188	6.243	6.769	3.034	1.806	0.899
$M = 30$, $N = 16384$	3.028	7.929	3.787	2.004	0.994	3.304	4.394	6.218	9.707	2.298	1.774	1.313
$M = 31$, $N = 16384$	3.271	8.706	4.193	2.151	1.109	3.572	4.835	6.899	10.829	2.528	1.966	1.461
$M = 32$, $N = 16384$	3.620	9.677	4.648	2.426	2.307	3.913	5.426	7.953	12.833	2.875	2.288	1.734

TABELA B.1. Tempos aferidos no Cluster I, em segundos.

Algoritmo	Serial	Redução Cíclica				Divisão–e–conquista						
Implementação	Serial	Serial	Paralela			Serial				Paralela		
	1	1	2	4	8	1	2	4	8	2	4	8
$M = 2$, $N = 262144$	0.867	2.132	1.100	0.543	0.277	0.858	1.178	1.363	1.430	0.640	0.362	0.201
$M = 3$, $N = 262144$	1.721	4.550	2.421	1.113	0.571	1.726	2.202	2.494	2.639	1.248	0.689	0.368
$M = 4$, $N = 131072$	1.507	3.868	2.062	1.012	0.517	1.502	1.818	2.090	2.183	1.081	0.608	0.327
$M = 5$, $N = 131072$	2.478	6.582	3.360	1.662	0.845	2.463	2.979	3.299	3.448	1.716	0.937	0.499
$M = 6$, $N = 131072$	3.838	10.123	7.375	2.545	1.278	3.829	4.573	5.080	5.235	2.638	1.406	0.763
$M = 7$, $N = 65536$	2.747	7.425	3.622	1.860	0.936	2.774	3.209	3.544	3.659	1.856	1.007	0.524
$M = 8$, $N = 65536$	3.883	10.356	5.445	2.551	1.307	3.856	4.441	4.864	5.038	2.591	1.343	0.767
$M = 9$, $N = 65536$	5.169	14.025	6.821	3.458	1.734	5.181	5.892	6.419	6.764	3.484	1.831	0.980
$M = 10$, $N = 65536$	6.808	18.528	12.081	4.520	2.301	6.771	7.644	8.233	8.669	4.442	2.348	1.261
$M = 11$, $N = 65536$	8.673	23.854	11.344	5.750	2.959	8.660	9.683	10.389	10.910	5.672	2.957	1.654
$M = 12$, $N = 65536$	10.907	30.190	14.312	7.242	3.739	10.911	12.099	12.863	13.505	7.097	3.681	1.916
$M = 13$, $N = 32768$	6.752	18.637	9.084	4.455	2.267	6.724	7.470	8.074	8.688	4.411	2.310	1.236
$M = 14$, $N = 32768$	8.256	22.945	10.821	5.440	2.805	8.271	11.203	13.149	14.883	7.384	3.789	2.068
$M = 15$, $N = 32768$	9.904	27.647	13.042	6.567	3.368	9.905	10.930	11.885	13.071	6.427	3.418	1.904
$M = 16$, $N = 32768$	11.734	33.136	15.486	7.809	3.959	11.730	13.017	14.197	15.586	7.655	4.089	2.292
$M = 17$, $N = 32768$	13.872	39.082	18.343	9.209	4.664	13.871	15.297	16.729	18.495	9.056	4.840	2.673
$M = 18$, $N = 32768$	16.220	45.837	21.358	10.759	5.427	16.236	17.789	19.170	20.983	10.453	5.560	3.056
$M = 19$, $N = 32768$	18.870	53.377	24.875	12.443	6.319	18.861	20.601	22.257	24.245	12.164	6.478	3.526
$M = 20$, $N = 32768$	21.688	61.643	28.399	14.379	7.242	21.704	23.859	26.323	29.856	14.078	7.702	4.325
$M = 21$, $N = 32768$	24.881	70.927	32.985	16.554	8.939	25.029	46.301	57.189	62.657	30.921	17.580	8.999
$M = 22$, $N = 32768$	28.158	80.678	37.328	18.746	9.705	28.386	30.982	33.295	37.167	18.202	9.740	5.366
$M = 23$, $N = 16384$	15.973	45.776	21.060	10.657	5.447	16.086	17.931	20.357	24.317	10.635	6.022	3.555
$M = 24$, $N = 16384$	18.035	51.586	23.858	11.959	6.119	18.040	20.310	23.133	27.840	12.026	6.879	4.071
$M = 25$, $N = 16384$	20.169	58.061	26.845	13.373	6.827	20.183	22.718	25.969	31.366	13.520	7.700	4.536
$M = 26$, $N = 16384$	22.606	64.950	29.929	15.068	7.719	22.819	33.880	42.100	51.352	23.095	12.756	7.400
$M = 27$, $N = 16384$	25.243	72.492	33.087	16.668	8.548	25.256	28.441	32.777	40.294	16.916	9.754	5.895
$M = 28$, $N = 16384$	27.929	80.335	36.827	18.580	9.549	27.894	31.727	36.536	45.435	18.978	10.968	6.712
$M = 29$, $N = 16384$	30.963	89.043	40.833	20.530	10.466	31.142	57.879	71.125	77.859	38.502	22.268	11.218
$M = 30$, $N = 16384$	33.816	97.782	44.769	22.494	11.473	33.829	38.768	45.640	58.070	23.525	13.710	8.477
$M = 31$, $N = 16384$	37.321	107.593	49.319	24.695	12.616	37.368	42.603	50.250	64.269	25.553	15.222	9.379
$M = 32$, $N = 16384$	40.832	117.813	54.156	27.086	13.882	40.924	47.315	56.556	73.881	28.354	17.111	10.839

TABELA B.2. Tempos aferidos no Cluster II, em segundos

.

transcorrido desde o início da execução do programa (tais rotinas excluem o tempo dedicado pelo processador a outros processos). Para aferir o tempo transcorrido durante a execução de um certo trecho de código, são necessárias duas chamadas à rotina de relógio, uma vez imediatamente antes e outra imediatamente depois da execução do trecho em questão. O tempo transcorrido é obtido pela diferença entre os dois valores retornados: o referencial é irrelevante, pois é uma constante que se cancela na subtração.

Mas há um erro inerente a cada chamada da rotina de relógio, cuja precisão nem sempre é a do *clock* da máquina — como de fato não foi nos ambientes em que executamos os experimentos. Após efetuar repetidos testes iniciais das rotinas de relógio disponíveis nas duas máquinas, ficou estabelecido um intervalo de confiança de 10ms para cada medida. As primeiras tentativas de medir o tempo gasto nas soluções dos sistemas bloco tridiagonais

pequenos mostrou que os valores que desejávamos medir podiam ser inferiores a 100ms, caso no qual este intervalo de confiança traduz-se num erro relativo de ±5%. Forçamos o tempo de cálculo a ser de no mínimo um segundo, resolvendo o mesmo sistema linear repetidas vezes, e dividindo o tempo total pelo número de repetições realizadas. O erro relativo das medidas, ou pelo menos sua componente advinda do erro inerente ao relógio, fica assim limitado a ±0.5%.

Entretanto, cada sistema $\mathbf{Ax} = \mathbf{y}$ resolvido precisa primeiro ser montado, ou seja, a matriz de coeficientes $\mathbf{A}$ e o lado direito $\mathbf{y}$ precisam ser preenchidos novamente após cada solução. Este tempo de inicialização não deve ser contabilizado: para aferir exclusivamente o tempo gasto na solução, cada programa executa algoritmo a seguir.

```
T_t ← 2                 Inicializa tempo total mínimo com 2 (segundos)
T_g ← 0                 Inicializa tempo total gasto com 0
R ← 0                   Inicializa o contador de repetições com 0
T_s ← clock             Chama o relógio do sistema, marca o tempo inicial
repeat
  Inicializa A e y
  Resolve Ax = y        Com algoritmo serial, redução cíclica ou divisão–e–conquista
  R ← R + 1
  T_e ← clock           Marca o tempo (possivelmente) final
  T_g ← T_e − T_s       Calcula o tempo gasto
  if T_g ≥ T_t break

T_s ← clock
for i = 1, ..., R
  Inicializa A e y
T_e ← clock
T_g ← T_g − (T_e − T_s) Abate do tempo total o tempo gasto inicializando A e y (R vezes)
```

A premissa de que a inicialização de $\mathbf{A}$ e $\mathbf{y}$ é menos custosa que a solução, válida nos problemas-teste usados, é suficiente para garantir que $T_t = 2$ levará a um tempo mínimo de um segundo efetivamente resolvendo os sistemas.

Referências Bibliográficas

[1] P. Arbenz, A. Cleary, J. Dongarra, and M. Hegland. A comparison of parallel solvers for diagonally dominant and general narrow-banded linear systems II. In *European Conference on Parallel Processing*, pages 1078–1087, 1999.

[2] P. Arbenz and W. Gander. A survey of direct parallel algorithms for banded linear systems. Technical Report 221, ETH Zürich, Computer Science Department, November 1994.

[3] A. Azevedo, D. Marchesin, B. J. Plohr, and K. Zumbrun. Capillary instability in models for three-phase flow. *Zeitschrift fur Angewandte Mathematik und Physik*, 53:713–746, 2002.

[4] R. M. Beam and R. F. Warming. An implicit finite–difference algorithm for hyperbolic systems in conservation law form. *J. Comp. Phys.*, 22(1):87–110, 1976.

[5] M. W. Berry and A. Sameh. Multiprocessor schemes for solving block tridiagonal linear systems. *Int. J. of Supercomp. App.*, 2:37–57, 1988.

[6] G. Brassard and P. Bratley. *Algorithmics: theory and practice*. Prentice-Hall, 1988.

[7] S. Buckley and M. Leverett. Mechanisms of fluid displacement in sands. *Trans. AIME*, 146:187–196, 1942.

[8] B. L. Buzbee, G. H. Golub, and C. W. Nielson. On direct methods for solving Poisson's equations. *SIAM J. Numer. Anal.*, 7(4):627–656, 1970.

[9] T. D. Cao, J. F. Hall, and R. A. van de Geijn. Parallel Cholesky factorization of a block tridiagonal matrix. In *Proceedings of the International Conference on Parallel Processing 2002 (ICPP-02)*, August 2002.

[10] G. Chapiro, A. A. Mailybaev, D. Marchesin, and A.J. Souza. Singular perturbation in combustion waves for gaseous flow in porous media. In *XXVI CILAMCE, Guarapari, ES, Brazil*, October 2005.

[11] A. Cleary and J. Dongarra. Implementation in ScaLAPACK of divide-and-conquer algorithms for banded and tridiagonal linear systems. Technical Report CS-97-358, University of Tennessee, Knoxville, TN 37996, USA, 1997.

[12] A. Corey, C. Rathjens, J. Henderson, and M. Wyllie. Three–phase relative permeability. *Trans. AIME*, 207:349–351, 1956.

[13] J. J. Dongarra and L. Johnsson. Solving banded systems on a parallel processor. *Parallel Computing*, 5:219–246, 1987.

[14] J. J. Dongarra and A. H. Sameh. On some parallel banded system solvers. *Parallel Computing*, 1:223–235, 1984.

[15] I. S. Duff and H. A. van der Vorst. Developments and trends in the parallel solution of linear systems. *Parallel Computing*, 25(13–14):1931–1970, 1999.

[16] C. Farhat and F. X. Roux. A method of finite element tearing and interconnecting and its parallel solution algorithm. *Int. J. Num. Meth. Engrg.*, 32(4):1205–1227, 1991.

[17] W. Gander and G.H. Golub. Cyclic reduction - history and applications. Online at http://citeseer.ist.psu.edu/95562.html.

[18] I. N. Hajj and S. Skelboe. A multilevel parallel solver for block tridiagonal and banded linear systems. *Parallel Computing*, 15:21–45, 1990.

[19] E. Isaacson and H.B. Keller. *Analysis of Numerical Methods*. Dover Publications, 1994.

[20] M. S. Khaira, G. L. Miller, and T. J. Sheffler. Nested dissection: A survey and comparison of various nested dissection algorithms. Technical Report CS-92-106, Carnegie Mellon University, 1992.

[21] G. Kron. A set of principles to interconnect the solutions of physical systems. *J. Appl. Phys.*, 24:965–980, 1953.

[22] J. López, O. Plata, F. Argüello, and E. L. Zapata. Unified framework for the parallelization of divide and conquer based tridiagonal systems. *Parallel Comput.*, 23(6):667–686, 1997.

[23] D. Marchesin and J. Bruining. Analysis of nitrogen and steam injection in a porous medium with water. *Transport in Porous Media*, 62(3):251–281, 2006.

[24] D. Marchesin and B. J. Plohr. Wave structure in WAG recovery. *SPE Journal*, 6:209–219, 2001.

[25] V. M. M. Matos, G. Hime, and D. Marchesin. Analysis of double shock generation and propagation in three–phase flow. 2007. Em preparação.

[26] F.H. McMahon. The Livermore Fortran Kernels test of the numerical performance range. In J. L Martin, editor, *Performance Evaluation of Supercomputers*, pages 143–186. Elsevier Science B.V., North-Holland, Amsterdam, 1988.

[27] V. Mehrmann. Divide and conquer methods for block tridiagonal systems. *Parallel Computing*, 19(3):257–279, 1993.

[28] D. Rixen, C. Farhat, R. Tezaur, and J. Mandel. Theoretical Comparison of the FETI and Algebraically Partitioned FETI Methods, and Performance Comparisons with a Direct Sparse Solver. *Int. J. Num. Meth. Engrg.*, 46:501–534, 1999.

[29] G.S. Shedler and M.M. Lehman. Evaluation of redundancy in a parallel algorithm. *IBM Systems Journal*, 6(3):142–149, 1967.

[30] J. C. Strikwerda. *Finite difference schemes and partial differential equations*. Wardsworth & Brooks/Cole, 1989.

[31] K. Sumiyoshi and T. Ebisuzaki. Performance of parallel solution of a block-tridiagonal linear system on Fujitsu VPP500. *Parallel Computing*, 24(2):287–304, 1998.

[32] F. Tisseur and J. Dongarra. A parallel divide and conquer algorithm for the symmetric eigenvalue problem on distributed memory architectures. *SIAM J. Sci. Comp.*, 20(6):2223–2236, 1999.

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