UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MATEMÁTICA APLICADA

Aplicação de técnicas de paralelização de programas usando OpenMP na solução numérica da equação de transporte de nêutrons

por

Júlia Domingues Lemos

Dissertação submetida como requisito parcial
para a obtenção do grau de

Mestre em Matemática Aplicada

Prof.ª Dr.ª Liliane Basso Barichello
Orientadora

Prof. Dr. Rudnei Dias da Cunha
Co-Orientador

Porto Alegre, Julho de 2018.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Domingues Lemos, Júlia

Aplicação de técnicas de paralelização de programas usando OpenMP na solução numérica da equação de transporte de nêutrons / Júlia Domingues Lemos.—Porto Alegre: PPGMAp da UFRGS, 2018.

88 p.: il.

Dissertação (mestrado) —Universidade Federal do Rio Grande do Sul, Programa de Pós-Graduação em Matemática Aplicada, Porto Alegre, 2018.
Orientadora: Barichello, Liliane Basso; Co-Orientador: da Cunha, Rudnei Dias

Dissertação: Matemática Aplicada,
Teoria de Transporte, Computação paralela

Aplicação de técnicas de paralelização de programas usando OpenMP na solução numérica da equação de transporte de nêutrons

por

Júlia Domingues Lemos

Dissertação submetida ao Programa de Pós-Graduação em Matemática Aplicada do Instituto de Matemática e Estatística da Universidade Federal do Rio Grande do Sul, como requisito parcial para a obtenção do grau de

Mestre em Matemática Aplicada

Linha de Pesquisa: Teoria de Transporte de Partículas

Orientadora: Prof.ª Dr.ª Liliane Basso Barichello

Co-Orientador: Prof. Dr. Rudnei Dias da Cunha

Banca examinadora:

Prof. Dr. Pedro Henrique de Almeida Konzen
UFRGS

Prof.ª Dr.ª Eliete Biasotto Hauser
PUCRS

Prof. Dr. Haroldo Fraga de Campos Velho
INPE

Dissertação apresentada e aprovada em
Julho de 2018.

Prof. Dr. Carlos Hoppen
Coordenador

Sumário

Lista de Figuras

Lista de Tabelas

LISTA DE SÍMBOLOS

$\mathbf{r}$	Vetor posição da partícula
σ_u	Seção de choque macroscópica da interação do tipo u
σ_T	Seção de choque macroscópica total
E	Energia cinética da partícula
v	Módulo da velocidade da partícula
S,Q,q	Termo fonte
$\mathbf{\Omega}$	Vetor unitário na direção da velocidade
$f_u(\mathbf{r},\mathbf{\Omega'} \rightarrow \mathbf{\Omega},E' \rightarrow E)$	Probabilidade de transferência de uma partícula com direção $\mathbf{\Omega'}$ e energia E para uma direção $\mathbf{\Omega}$ e energia E'
M_u	Número de nêutrons emitidos na reação u
$N(\mathbf{r},\mathbf{\Omega},E,t)$	Número provável de nêutrons
N	Ordem da quadratura, ou número de direções
$\psi(x,\mu)$	Fluxo angular de nêutrons
$\phi(x)$	Fluxo escalar de nêutrons
t	Variável temporal
x	Variável espacial
μ	Variável angular
P_ℓ	ℓ-ésimo polinômio de Legendre

f_ℓ	Coeficiente do ℓ-ésimo polinômio de Legendre na expansão do termo de espalhamento
L'	Grau de anisotropia do espalhamento
L	Comprimento do domínio
S_p	*Speedup*
T_1	Tempo gasto executando um programa com 1 processador
T_p	Tempo gasto executando um programa com p processadores
f_p	Fração do programa que é paralelizável
g	Ganho de tempo
$\psi_{e,d}$	Condição de contorno na fronteira esquerda ou direita
$\psi^{(0)}$	Estimativa inicial
$\psi^{(i)}$	ψ calculada na iteração i
$\psi^{(i+1/2)}$	ψ calculada pela varredura
e_r	Erro relativo
$\overline{\psi}(x_j)$	Fluxo angular médio
ϵ	Tolerância para o processo iterativo
F	Correção aditiva
$\phi_1^{(1+1/2)}$	Corrente (capítulo 6)
$\phi_0^{(i+1/2)}$	Fluxo escalar (capítulo 6)
Φ	Fluxo escalar na célula grossa
$\hat{D}_{k+1/2}^{(i+1/2)}$	Correção de transporte para a Lei de Fick

$\phi(\nu,\mu)$	Parte da solução buscada pelo ADO (capítulo 7)
ν_j	Constante de separação
c	Número médio de nêutrons secundários por colisão
A_j,B_j	Constantes da solução gerada pelo ADO
$\rho(x)$	Fluxo escalar (capítulo 7)
$j(x)$	Corrente (capítulo 7)

RESUMO

Neste trabalho foi realizado um estudo de estratégias de paralelização para métodos que resolvem numericamente a equação de transporte de nêutrons em domínio unidimensional, monoenergética, estacionária, com espalhamento isotrópico, em meio homogêneo e com uma fonte fixa. Foram estudados o *Source Iteration*, *Diffusion Synthetic Acceleration*, *Coarse Mesh Finite Differences* e o Método Analítico de Ordenadas Discretas. As versões paralelas, desenvolvidas utilizando OpenMP, foram obtidas a partir das versões sequenciais dos códigos, implementadas em Fortran 95.

O objetivo deste trabalho é escrever versões paralelas que sejam executadas em menor tempo de que as versões sequenciais, pelo menos a partir de algum tamanho de problema. Ganhos de tempo para os métodos *Source Iteration*, *Diffusion Synthetic Acceleration* e *Coarse Mesh Finite Differences* foram relatados em torno de 20% para problemas heterogêneos, chegando a registrar ganhos de mais de 50% em problemas homogêneos. A versão paralela do método analítico de ordenadas discretas chegou a apresentar 88% de ganho, registrando um *speedup* superlinear para um problema homogêneo.

ABSTRACT

This work is a study about parallelization strategies for methods that numerically solve the neutron transport equation, for an one dimensional domain, one energy group, steady state, isotropically scattering with a fixed source. The studied methods were the *Source Iteration*, the *Diffusion Synthetic Acceleration*, the *Coarse Mesh Finite Differences* and the *Analytical Discrete Ordinates*. The code's parallel versions were developed using OpenMP from the sequential versions, implemented in Fortran 95.

The main goal of this work was to write parallel versions that would run in less time than the sequential ones, at least from a certain size of problem on. As for the running times, gains were registered around 20% for the *Source Iteration*, the *Diffusion Synthetic Acceleration* and the *Coarse Mesh Finite Differences* solving a one-region problem, reaching up to 50% in multi-regions problems. The parallel version for the *Analytical Discrete Ordinates* attained an 88% gain in a one-region problem, which is a super-linear speedup.

1 INTRODUÇÃO

A equação de transporte, originalmente escrita por Ludwig Boltzmann [10] com a intenção de descrever o comportamento de gases, é utilizada como modelo matemático para diversas áreas. Entre elas, citamos transferência radiativa e transporte de partículas nêutras [30, 14, 19]. Este trabalho trata de casos específicos da equação de transporte de nêutrons, com domínio unidimensional, monoenergética, estacionária, com espalhamento isotrópico, em meio inicialmente homogêneo e com uma fonte fixa.

Como a equação de Boltzmann não possui solução analítica em muitos casos, métodos numéricos são frequentemente empregados em seu tratamento. Um conjunto de métodos de extrema importância são os iterativos de iteração de fonte (*Source Iteration*), bem como suas versões com aceleradores (ou pré-condicionadores), *Diffusion Synthetic Acceleration* e *Coarse-Mesh Finite Differences*. Os aceleradores, neste contexto, dizem respeito ao número de iterações necessárias até a convergência, ou seja, os dois últimos são versões que tendem a ser mais rapidamente convergentes de um método base, que é o *Source Iteration*.

Esquemas iterativos dependem fortemente de uma estimativa inicial e da possibilidade de discretização das variáveis, pois estes efetuam uma varredura na variável espacial para cada direção pré-determinada pela discretização da variável angular. Isto torna claro que, conforme as variáveis do problema são discretizadas em malhas cada vez mais refinadas, a quantidade de variáveis do programa a serem tratadas aumenta proporcionalmente, de modo que os problemas acabam se tornando computacionalmente muito custosos.

Outro ponto estudado neste trabalho foi o método analítico de ordenadas discretas, que trata a variável espacial do problema analiticamente. Este não é iterativo, portanto não pertence ao mesmo conjunto de métodos já citados, mas

possui a vantagem de apresentar tempo de execução melhores em diversos casos, além de não precisar iterar uma estimativa inicial até que esta se torne uma solução aceitável.

O objetivo deste trabalho é, dados os códigos sequenciais para os quatro métodos acima (implementados em Fortran 95, parcialmente em [35] e parcialmente nos estudos de doutorado da mesma autora), produzir versões paralelas destes códigos que sejam executadas em menor tempo.

Apesar de existirem diversas ferramentas que tornam possível a paralelização de programas, a escolha feita para este trabalho foi a OpenMP. Esta escolha não foi feita apenas em função da praticidade, e também do quão sucinto a OpenMP é, mas consideravelmente em função da arquitetura disponível, que era uma máquina de um único processador com múltiplos núcleos e múltiplas threads por núcleo. Assim, foi decidido que a paralelização seria conduzida sobre uma plataforma multithreading de memória compartilhada.

Apesar de surgir de um princípio relativamente simples (o de distribuir tarefas) a paralelização traz consigo diversos desafios. Algumas técnicas de paralelização são desenvolvidas para um tipo de hardware ao qual não se possui acesso trivialmente; eventualmente programas paralelos não serão determinísticos com relação a presença de erros em sua execução; é possível que o código simplesmente não seja paralelizável. Todos estes são aspectos da paralelização que precisam ser observados e devidamente tratados quando se paraleliza código.

De maneira geral, a estratégia adotada neste trabalho foi dividir o domínio (na variável espacial) em blocos, de modo que fosse possível varrê-los independentemente. Esta é uma estratégia já utilizada antes, ainda que com abordagens diferentes, como por exemplo em [31], no qual domínios bidimensionais foram particionados em blocos menores, fazendo uso da estratégia de particionamento da malha por sobrecarga. Estes blocos menores foram, então, processados independentemente,

onde o tempo em comunicações entre processadores foi maior do que a versão sequencial, porém garantindo ganho de tempo computacional. Este estudo levou em consideração malhas estruturadas e não estruturadas.

Existem também trabalhos como [16], que fazem estudos de diversas maneiras de conduzir a varredura sobre malhas não estruturadas e sobre como distribuí-los em diversos processadores. Isto envolveu conduzir um estudo sobre como ocorrerá a comunicação dos processadores, visto que versões paralelas podem tomar mais tempo do que versões sequenciais apenas em função de mau gerenciamento de comunicação dos processadores. A estratégia que se sobressaiu neste trabalho foi a de executar toda a lista de tarefas resolvíveis para um processador antes deste efetuar a comunicação com outros processadores, o que reduziu a quantidade de comunicações entre eles e chegou a atingir *speedups* (cf. capítulo 3) superlineares.

Outros trabalhos promovem estudos sobre como um elemento de processamento decide qual tarefa executará, caso mais de uma esteja disponível. Em [2] são tratadas varreduras em três dimensões, o que torna as dependências dos volumes vizinhos mais complexas do que no caso unidimensional, no qual uma célula depende apenas da anterior. Neste caso, um algoritmo ótimo de escolha de tarefas permite que os elementos de processamento se comuniquem de maneira eficiente, atingindo eficiência de até 80% em 131072 núcleos.

Estas referências nos mostram que paralelização é um caminho com muitas dificuldades quando se lida com varreduras, mas que pode trazer ganhos consideráveis em termos de tempo de execução.

O capítulo 2 faz uma breve dedução da equação de transporte em sua forma mais geral e em seguida a apresenta com uma série de hipóteses, já no formato que será tratado. O capítulo 3 faz uma breve contextualização sobre a programação paralela e sobre a OpenMP.

Os capítulos 4 e 6 contém as formulações dos métodos *Source Iteration* e *Coarse Mesh Finite Differences*, bem como as estratégias de paralelização que foram utilizadas em cada um deles. Além disso, possuem problemas teste iniciais que serviram para verificar se a estratégia em questão era de fato viável.

Os capítulos 5 e 7 possuem as formulações dos métodos *Diffusion Synthetic Acceleration* e Analítico de Ordenadas Discretas, munidos de estratégias de paralelização, porém não apresentam problemas teste iniciais, pois, como veremos, a estratégia de paralelização aderida não demandava testes para verificar se a mesma era viável ou não.

O capítulo 8 traz três problemas teste. O primeiro é um teste de exaustão, executado para mapear o desempenho do programa conforme o tamanho do problema aumentava; o segundo e o terceiro são um problema de transporte homogêneo e um problema de transporte heterogêneo, a fim de comparar os tempos de execução dos programas paralelos com suas versões sequenciais. A análise de desempenho das versões paralelas foi feita calculando-se, particularmente, o ganho no tempo de execução obtido pela versão paralela sobre a versão sequencial, conforme definido no capítulo 4. Por fim, comentários e considerações finais estão no capítulo 9, seguido de referências bibliográficas.

2 A EQUAÇÃO DE TRANSPORTE DE PARTÍCULAS

Esta seção é dedicada à dedução da equação de transporte, originalmente proposta por Ludwig Boltzmann em 1872 para descrever o movimento de gases, tratando-os como partículas. A equação de transporte aqui considerada, que é a equação linear de Boltzmann, descreve o número esperado de partículas migrando em determinada direção, com dada energia cinética em dado instante de tempo em um meio material [8, 10].

Neste caso, descreveremos uma situação em que as partículas são nêutrons e interagem com os núcleos dos átomos que compõem o meio, sem interagirem entre si, pois do contrário teríamos um modelo não linear. Tais interações podem ser de diversos tipos, sendo espalhamento e absorção as mais relevantes, e estas ainda podem ser subdivididas em tipos [12, 20].

O espalhamento pode ser elástico, se houver conservação de energia cinética total antes e depois do espalhamento, ou inelástico, caso não haja. Quando um nêutron é absorvido por um núcleo, pode ocorrer um processo de captura radioativa, que é quando o núcleo (acrescido do nêutron absorvido) emite radiação eletromagnética na forma de raios gama. Outro processo que pode ocorrer no caso de absorção é a fissão do núcleo em dois núcleos mais leves, com liberação de energia e de outros nêutrons, que possivelmente colidirão com outros núcleos do meio, iniciando assim uma reação de fissão em cadeia.

Podemos interpretar a seção de choque macroscópica como a probabilidade de interação de um nêutron com um núcleo por unidade de distância percorrida, e a denotamos σ_u, sendo que o subscrito u indica o tipo de interação. As seções de choque macroscópicas dependem, nesta dedução, da posição e da energia da partícula. Chamando de $\mathbf{r} = (x,y,z)$ o vetor posição da partícula e de E sua

energia cinética, podemos escrever as seções de choque macroscópicas de absorção e de espalhamento como

$$\sigma_a(\mathbf{r},E) = \sigma_\gamma(\mathbf{r},E) + \sigma_f(\mathbf{r},E) \tag{2.1}$$

e

$$\sigma_S(\mathbf{r},E) = \sigma_n(\mathbf{r},E) + \sigma_{n'}(\mathbf{r},E). \tag{2.2}$$

Na equação (2.1), σ_γ é a seção de choque de captura radioativa e σ_f é a seção de choque de fissão. Na equação (2.2), σ_n e $\sigma_{n'}$ são as seções de choque de espalhamento elástico e inelástico respectivamente. A seção de choque total, denotada σ_T, pode então ser escrita como

$$\sigma_T(\mathbf{r},E) = \sigma_a(\mathbf{r},E) + \sigma_S(\mathbf{r},E) + \sigma_{op}(\mathbf{r},E), \tag{2.3}$$

na qual σ_{op} é a seção de choque referente a outras interações que não sejam absorção ou espalhamento, como por exemplo, reações nas quais o nêutron causa a emissão de dois nêutrons, denotada $(n,2n)$, ou de um próton, denotada (n,p). Em alguns contextos, como, por exemplo, no estudo de reatores nucleares, tais interações não são tão importantes quanto absorção e espalhamento, e portanto frequentemente desconsideradas [13, 20].

Dado $\mathbf{\Omega}$ um vetor unitário, definimos, para uma reação genérica (n,u) a seção de choque diferencial σ_u, dada por

$$\sigma_u(\mathbf{r},\mathbf{\Omega'} \to \mathbf{\Omega},E' \to E) = \sigma_u(\mathbf{r},E')f_u(\mathbf{r},\mathbf{\Omega'} \to \mathbf{\Omega},E' \to E). \tag{2.4}$$

Na equação (2.4), $f_u(\mathbf{r},\mathbf{\Omega'} \to \mathbf{\Omega},E' \to E)d\mathbf{\Omega}dE$ é a probabilidade de que um nêutron, com direção Ω' e energia E', cause a emissão de pelo menos um

nêutron em alguma direção de $d\mathbf{\Omega}$ em torno de $\mathbf{\Omega}$, com alguma energia dE em torno de E, ao sofrer uma reação do tipo u com um núcleo. Além disso, $\sigma_u(\mathbf{r},E')$ é a seção de choque da interação u à energia E'. A probabilidade f_u é chamada de probabilidade de transferência, e é normalizada de modo que [20]

$$\int_E \int_{\mathbf{\Omega}} f_u(\mathbf{r},\mathbf{\Omega'} \to \mathbf{\Omega},E' \to E) d\mathbf{\Omega} dE = M_u, \tag{2.5}$$

na qual M_u é o número de nêutrons emitido na reação u. A seção de choque diferencial definida em (2.4) tem o propósito de descrever a distribuição em direção e energia dos nêutros emitidos em reações diversas. A probabilidade total de transferência é dada por

$$\sigma_T(\mathbf{r},E') f(\mathbf{r},\mathbf{\Omega'} \to \mathbf{\Omega},E' \to E) = \sum_u \sigma_u(\mathbf{r},E') f_u(\mathbf{r},\mathbf{\Omega'} \to \mathbf{\Omega},E' \to E), \tag{2.6}$$

na qual $\sum_u$ é a soma para todas as reações que produzem nêutrons. Definimos ainda dV um volume elementar centrado em $\mathbf{r}$ e $N(\mathbf{r},\mathbf{\Omega},E,t)$ o número provável de nêutrons no instante t em um volume dV, viajando em direções dentro de $d\mathbf{\Omega}$ em torno de $\mathbf{\Omega}$, com energia no intervalo dE em torno de E. Chamando de v' o módulo da velocidade do nêutron com energia E', sabemos que a taxa de produção de nêutrons em dV é dada por [8]

$$\int_{E'} \int_{\mathbf{\Omega'}} \Big[\sigma_T(\mathbf{r},E') f(\mathbf{r},\mathbf{\Omega'} \to \mathbf{\Omega},E' \to E) v' N(\mathbf{r},\mathbf{\Omega'},E',t) d\mathbf{\Omega'} dE' + \\ + S(\mathbf{r},\mathbf{\Omega},E,t) \Big] dV d\mathbf{\Omega} dE. \tag{2.7}$$

Na equação (2.7), o primeiro termo representa os nêutrons que, após interagirem com um núcleo em dV, são transferidos para os intervalos (ou originam

novos nêutrons) de direção e energia $d\mathbf{\Omega}$ e dE em torno de $\mathbf{\Omega}$ e E respectivamente, e o termo S representa nêutrons produzidos por uma fonte no interior do meio material. A taxa de perda de nêutrons, por sua vez, é dada por [8]

$$\left[\sigma_T(\mathbf{r},E)vN(\mathbf{r},\mathbf{\Omega},E,t) + v\mathbf{\Omega}\nabla \cdot N(\mathbf{r},\mathbf{\Omega},E,t)\right]dV d\mathbf{\Omega} dE, \tag{2.8}$$

na qual o primeiro termo representa nêutrons que deixam os intervalos de direção e energia $d\mathbf{\Omega}$ e dE após interações em dV, o segundo termo representa fuga líquida de nêutrons de dV e ∇ denota o operador gradiente. Ainda, a variação do número provável de nêutrons em um volume dV é dada por [20]

$$\left[\frac{\partial}{\partial t}N(\mathbf{r},\mathbf{\Omega},E,t)\right]dV d\mathbf{\Omega} dE. \tag{2.9}$$

Deste modo, desejamos efetuar um balanço de nêutrons em um volume dV. Sabemos, portanto, que a taxa de variação do número de nêutrons pode ser escrita como a taxa de produção de nêutrons, dada pela equação (2.7), diminuída da taxa de perda de nêutrons, dada pela equação (2.8), tudo em dV. Unindo esta informação à equação (2.9) e escrevendo $\psi(\mathbf{r},\mathbf{\Omega},E,t) = vN(\mathbf{r},\mathbf{\Omega},E,t)$, temos que o balanço de nêutrons é dado por

$$\begin{aligned}&\frac{1}{v}\frac{\partial}{\partial t}\psi(\mathbf{r},\mathbf{\Omega},E,t) + \mathbf{\Omega}\cdot\nabla\psi(\mathbf{r},\mathbf{\Omega},E,t) + \sigma_T(\mathbf{r},E)\psi(\mathbf{r},\mathbf{\Omega},E,t) = \\ &= \int_{E'}\int_{\mathbf{\Omega'}} \sigma_T(\mathbf{r},E')f(\mathbf{r},\mathbf{\Omega'}\to\mathbf{\Omega},E'\to E)\psi(\mathbf{r},\mathbf{\Omega'},E',t)d\mathbf{\Omega'}dE' + S(\mathbf{r},\mathbf{\Omega},E,t).\end{aligned} \tag{2.10}$$

A equação (2.10) é a equação íntegro-diferencial de transporte de nêutrons linear escrita em sua forma mais geral, e a quantidade a ser determinada é ψ, chamada de fluxo angular de nêutrons. Este trabalho, no entanto, considera problemas estacionários, em que os nêutrons já atingiram equilíbrio com relação ao

tempo. Desta forma, as dependências em t, bem como o primeiro termo da equação (2.10), podem ser omitidos. A equação (2.10) pode então ser escrita como

$$\begin{aligned}&\boldsymbol{\Omega}\cdot\nabla\psi(\mathbf{r},\boldsymbol{\Omega},E)+\sigma_T(\mathbf{r},E)\psi(\mathbf{r},\boldsymbol{\Omega},E)=\\&\quad=\int_{E'}\int_{\boldsymbol{\Omega'}}\sigma_T(\mathbf{r},E')f(\mathbf{r},\boldsymbol{\Omega'}\rightarrow\boldsymbol{\Omega},E'\rightarrow E)\psi(\mathbf{r},\boldsymbol{\Omega'},E')d\boldsymbol{\Omega'}dE'+S(\mathbf{r},\boldsymbol{\Omega},E).\end{aligned}\quad(2.11)$$

A dependência da energia é comumente tratada utilizando a aproximação de multigrupos de energia [8, 29], que divide o intervalo de energia em G grupos de energia e em seguida integra a equação (2.11) sobre cada uma das faixas de energia que definem estes grupos. Neste trabalho, ao invés de tratar a dependência da energia utilizando multigrupos, fizemos a suposição de que todos os nêutrons estavam na mesma faixa de energia, o que equivale à aproximação de multigrupos para $G = 1$ [8]. Consideramos também que o operador $\boldsymbol{\Omega}\cdot\nabla$, para uma geometria unidimensional, é expresso como

$$\boldsymbol{\Omega}\cdot\nabla=\mu\frac{\partial}{\partial x},\quad(2.12)$$

na qual x é a variável espacial e μ é o cosseno do ângulo formado pela direção na qual a partícula viaja com o eixo do domínio da variável espacial. Deste modo, a equação (2.11) pode ser escrita como

$$\mu\frac{\partial}{\partial x}\psi(x,\mu)+\sigma_T(x)\psi(x,\mu)=\int_{-1}^{1}f(\mu\rightarrow\mu')\psi(x,\mu')d\mu'+q(x,\mu),\quad(2.13)$$

na qual $q(x,\mu)$ é o termo fonte e $f(\mu\rightarrow\mu')$ a probabilidade de transferência, que é comumente expressa fazendo uso de uma expansão truncada em polinômios de Legendre (cf. capítulo 4). A equação (2.13), portanto, pode ser escrita como

$$\mu\frac{\partial}{\partial x}\psi(x,\mu)+\sigma_T(x)\psi(x,\mu)=\frac{1}{2}\sum_{\ell=0}^{L'}(2\ell+1)f_\ell P_\ell(\mu)\int_{-1}^{1}P_\ell(\mu')\psi(x,\mu')d\mu'+q(x,\mu). \tag{2.14}$$

Na equação (2.14) o limite L' da soma no lado direito é o grau de anisotropia do espalhamento. Como este trabalho lida com problemas de espalhamento isotrópico, ou seja, para $L'=0$. O peso f_0 é simplesmente a seção de choque de espalhamento $\sigma_S(x)$ e a equação (2.14) pode ser escrita como

$$\mu\frac{\partial}{\partial x}\psi(x,\mu)+\sigma_T\psi(x,\mu)=\frac{1}{2}\sigma_S\int_{-1}^{1}\psi(x,\mu')d\mu'+q(x,\mu) \tag{2.15}$$

e esta é a equação que será tratada nos capítulos a seguir. Na equação (2.15) (bem como em todas as a seguir) também consideramos que as seções de choque são constantes por região.

3 PARALELIZAÇÃO USANDO OPENMP

A computação paralela consiste, basicamente, em fazer com que diversas tarefas sejam executadas simultaneamente. Esta noção vem atrelada à ideia de que problemas grandes podem ser divididos em problemas menores e estes podem ser resolvidos concorrentemente. Paralelizações fazem uso de máquinas que possuam múltiplos elementos de processamento, que podem apresentar as mais variadas configurações, como por exemplo uma máquina com diversos processadores, uma máquina com apenas um processador com diversos núcleos, clusters de diversas máquinas, ou até mesmo hardware produzido especialmente para problemas específicos [17].

Este trabalho lidou com apenas um processador, com quatro núcleos, que contava com dois fluxos independentes de instruções (aos quais chamaremos de *threads*) por núcleo, totalizando 8 *threads*. O processador é o modelo Intel Core i7 2600k, 3,4GHz. A memória cache conta com 8Mb e a memória principal com 8Gb.

As diversas possibilidades de hardware e arquiteturas indicam que o paralelismo pode ocorrer em diversos níveis. Neste trabalho tratamos majoritariamente de paralelização das tarefas, ou seja, tarefas diferentes foram executadas simultaneamente sobre o mesmo (ou sobre outro) conjunto de dados. Também houve, no entanto, momentos em que uma mesma tarefa foi executada sobre conjuntos de dados diferentes. A paralelização a nível de instrução pode ser atingida fazendo uso de arquiteturas *pipeline* com múltiplos estágios [17], que é uma arquitetura recorrente nos dias atuais. Apesar desta paralelização ser parcialmente conduzida pelo compilador, ganhos mais substanciais podem ser atingidos quando os acessos à memória de acesso rápido (chamada de memória *cache*) são otimizados [21].

Cada vez que um dado é acessado, tal dado e um bloco de posições contíguas da memória principal são transferidos para a memória cache. Isto é feito

em cima da hipótese de que os próximos dados a serem acessados são precisamente as posições seguintes na memória, como tipicamente acontece em um programa no qual são feitos acessos a arranjos (matrizes e vetores). Eventualmente, cuidados podem ser tomados pelo programador a fim de otimizar os acessos à memória cache, diminuindo assim o número de acessos à memória principal e evitando que blocos sejam expulsos da cache apenas para serem transferidos de volta logo em seguida.[1]

Quando estes cuidados são tomados, os pipelines do processador podem ser mais efetivamente preenchidos e mais de uma instrução por ciclo de relógio pode ser, com sucesso, executada [17]. Infelizmente, otimizações profundas nos acessos à memória cache não foram executadas neste trabalho.

A paralelização de tarefas pode trazer dificuldades por aspectos que são inerentes ao código. Frequentemente, duas tarefas não podem ser executadas simultaneamente porque uma depende do resultado da outra, gerando assim condições de concorrência (*race conditions*) [24], o que pode causar erros na saída do programa quando tais condições não são devidamente consideradas.

Condições de concorrência ocorrem em máquinas de memória compartilhada, ou seja, máquinas nas quais os diversos elementos de processamento possuem o mesmo acesso à memória, quando duas ou mais *threads* acessam e escrevem sobre a mesma variável sem qualquer especificação sobre a ordem de acessos. Deste modo, é possível que uma *thread* acesse uma posição na memória que possuía conteúdo A, escreva B, e quando outra *thread* acessar a mesma posição na memória, esperando ler A, leia B, causando assim um erro devido ao não-determinismo inerente às condições de concorrência.

[1] Estes cuidados se traduzem, por exemplo, na escrita de laços cuja tarefa envolve acessar arranjos de forma a respeitar a maneira como tais arranjos são armazenados na memória. Numa matriz bidimensional em Fortran, por exemplo, isso significa que as colunas deve ser acessada por um índice que varia mais lentamente do que o índice que acessa as linhas.

Não apenas as condições de concorrência geram erros nas saídas dos programas, elas também não são consistentes com respeito a tais erros. Eventualmente a primeira *thread* acessará a posição na memória e escreverá B após a segunda *thread* ter lido A, como esperado, e nenhum erro será relatado. Isto ocorre porque a velocidade de cada *thread* pode ser levemente diferente, já que estão executando tarefas distintas, além de fatores externos, como a carga de trabalho do sistema operacional no momento da execução, também constituirem possíveis influências na velocidade das *threads* [15].

Isto faz com que programas paralelos com condições de concorrência, quando executados diversas vezes, possam apresentar diversas saídas diferentes. Além disso, tais condições são geralmente difíceis de serem detectadas e podem passar despercebidas em diversos testes. Fica clara, então, a importância da atenção aos detalhes quando se trabalha com computação paralela.

Dadas estas informações, é relevante notar que nem todos os trechos de um programa são paralelizáveis, visto que dependências de outros trechos ou laços cujas iterações referenciem elementos com outros índices são frequentemente comuns em métodos numéricos. Além disso, o trecho de código que é paralelizável tem implicações diretas no ganho de tempo pode ser atingido.

Uma forma de medir tal ganho é calculando *speedups*, que podem ser escritos como [15, 17]

$$S_p = \frac{T_1}{T_p} \tag{3.1}$$

na qual T_1 é o tempo necessário para que um programa seja executado sequencialmente e T_p é o tempo necessário para que o programa seja executado com p elementos de processamento. Existe um limite superior para S_p, dado pela Lei de Amdahl [15, 17], que pode ser escrita como

$$S_p = \frac{1}{f_p/p + (1 - f_p)} \tag{3.2}$$

na qual f_p é a fração do programa que pode ser paralelizada. Em casos ideais, onde todo o código pode ser paralelizado temos $f_p = 1$ e, portanto, $S_p = p$, ou seja, o *speedup* do melhor caso possível é o número de elementos de processamento.

Notamos, aqui, que a equação (3.1) pode ser empregada no sentido clássico, quando T_1 refere-se ao tempo do *melhor* programa sequencial possível que pode resolver a tarefa, enquanto T_p refere-se a algum programa em paralelo que completa a mesma tarefa. No entanto, também é possível utilizar a mesma equação para medir a *escalabilidade* de um programa paralelo quando este é obtido através da paralelização de um programa sequencial (que pode não ser o melhor programa para completar as tarefas. Como este trabalho preocupou-se em tomar um programa sequencial e paralelizá-lo, usaremos a equação (3.1) para medir a escalabilidade do programa paralelo.

Apesar de *speedups* consistirem uma forma bem estabelecida na literatura de medir o desempenho de programas paralelos, neste trabalho também tratamos de ganho de tempo, calculado por

$$g = \left(1 - \frac{T_p}{T_1}\right) 100\% \tag{3.3}$$

A equação (3.3) é uma maneira de exibir, em termos percentuais, qual foi a redução no tempo de execução da versão sequencial para a versão paralela em cada caso, enquanto os *speedups* geram dados compatíveis com padrões pré-estabelecidos pela literatura [17, 24].

Para este trabalho, o recurso escolhido para efetuar as paralelizações foi a OpenMP, que é uma interface de programação de aplicativos (API). Consiste de diretivas de compilador, rotinas de biblioteca e variáveis de ambiente, que permitem

ao programador determinar quais regiões do código serão executadas em paralelo, bem como de que maneira o trabalho será distribuído entre as *threads* envolvidas no processo.

A OpenMP permite que o programador insira diretivas diretamente no código sequencial, escrito em C/C++ ou em Fortran 95, como foi o caso neste trabalho. As diretivas da OpenMP iniciam todas com os caracteres `!$`, de modo que compiladores não compatíveis com a OpenMP identificam as linhas como comentários e o programa é executado sequencialmente.

O primeiro par de diretivas a ser comentado neste trabalho é `!$OMP PARALLEL` e `!$OMP END PARALLEL` que delimitam o início e o fim de regiões de código a serem executadas em paralelo. Regiões englobadas por estas diretivas devem ser blocos estruturados, ou seja, regiões com comandos executáveis mas que não contenham comandos que direcionem o fluxo de execução para fora da região paralela.

Enquanto o programa está sendo executado sequencialmente, este é comandado por uma única *thread* que recebe o nome de *master thread*. Quando esta encontra uma diretiva `!$OMP PARALLEL`, chama um time de *threads* para compor a região paralela, sendo que a *master thread* é também parte do time de *threads*, e a quantidade de *threads* pode ser definida pelo programador respeitando o número máximo de *threads* capazes de serem executadas pela arquitetura disponível [25].

Ao encerramento da região paralela, a diretiva `!$OMP END PARALLEL` faz com que a *master thread* termine todas as outras assim que estas estiverem com suas tarefas encerradas. Este é um processo de sincronização implícita e ocorre em diversas diretivas de encerramento.

Uma vez que uma região paralela foi aberta, é preciso determinar como o trabalho será distribuído entre o time de *threads*. Alguns pares de diretivas foram utilizados extensivamente neste trabalho, como por exemplo o par `!$OMP DO` e `!$OMP END DO`. Esta diretiva indica, quando posicionada antes de um laço com, di-

gamos, 1000 iterações em um processador com quatro *threads* disponíveis, que cada uma executará 250 iterações, dentre as 1000 necessárias.

Para o correto uso desta diretiva, é necessário que um mesmo elemento de um arranjo não seja referenciado por duas (ou mais) iterações distintas, e que os cálculos executados em uma iteração não dependam do resultado do cálculo executado em iterações anteriores. Se alguma destas ocorrer, há boas chances de ocorrerem as supracitadas condições de concorrência, as quais devem ser evitadas [25].

O par de diretivas `!$OMP WORKSHARE` e `!$OMP END WORKSHARE` também foi bastante utilizado. Esta é uma diretiva similar à `!$OMP DO`, mas feita para ser utilizada antes de comandos como `forall` e `where`. Além disso, também pode ser utilizada em intrínsecos da linguagem, como `matmul`, `transpose` ou `dot_product`. É importante notar que, para esta diretiva, é muito mais interessante utilizar as notações de arranjo e fazer uso dos intrínsecos da linguagem (cf. apêndice A.4)

Porém, todas as variáveis que serão atualizadas dentro de um `!$OMP WORKSHARE` precisam, obrigatoriamente, possuir o atributo `SHARED`, ou seja, precisa ser visível a todas as *threads* [15, 25]. Novamente, isto abre brechas para condições de concorrência, e é trabalho do programador evitá-las.

Estas duas diretivas foram frequentemente utilizadas neste trabalho, uma vez que métodos numéricos envolvem varrer e atualizar diversas matrizes e arranjos. Também é frequente, no entanto, que matrizes sejam montadas por blocos, ou que um laço possua diversas linhas contendo instruções independentes umas das outras. Neste caso, o par `!$OMP SECTIONS` e `!$OMP END SECTIONS` foi empregado, junto com a diretiva `!$OMP SECTION`, que delimita uma seção de código a ser executada por apenas uma *thread* (cf. A.2).

Deste modo, caso exista mais seções do que *threads*, estas serão executadas pelas *threads* conforme terminam suas seções atuais. Caso o número de seções não seja um múltiplo do número de *threads*, esta é uma estratégia que pode levar

a uso ineficiente de recursos. Por exemplo, cinco seções e quatro *threads* implica em quatro seções executadas em paralelo por quatro *threads*, seguido de uma *thread* trabalhando na quinta seção e três *threads* ociosas.

Todas as diretivas até o momento possuem sincronizações implícitas ao final, mas que podem ser suprimidas com o uso de uma cláusula `NOWAIT`. Sincronizações explícitas podem ser feitas em outros momentos do código se for necessário, com as diretivas `!$OMP FLUSH`, `!$OMP BARRIER` ou `!$OMP CRITICAL` [9, 25].

Além destas, quando uma variável recebe o atributo `SHARED` e se deseja atualizar uma variável através de um operador, a cláusula `REDUCTION` faz com que apenas uma *thread* atualize a variável a cada vez. Esta cláusula faz isso criando cópias privadas para cada *thread* da variável a ser atualizada, inicializando cada uma delas de acordo com o operador, e, ao final da diretiva, a variável `SHARED` é atualizada de acordo com suas cópias privadas (cf. A.3)

Outras diretivas produzem efeitos similares, mas esta cláusula pode ser utilizada junto a uma diretiva `!$OMP DO` ou `!$OMP SECTIONS`, contornando assim algumas das possíveis condições de concorrência [9, 15].

Outra abordagem sobre a OpenMP, que não envolve diretamente trabalhar com as diretivas e suas cláusulas, é atribuir trabalho diretamente às *threads* utilizando o número atribuído a cada uma delas. Se p é o número de *threads* disponíveis, então elas são enumeradas de 0 a $p-1$, sendo a *master thread* identificada por 0. Esta abordagem pode resolver problemas com os quais as diretivas não lidam bem, mas necessita que o programador escreva as sincronizações das *threads* manualmente, o que é permitido pela OpenMP, mas não é considerada uma boa prática e pode ser potencialmente problemático.

A escolha de trabalhar com a OpenMP foi feita em função da versatilidade do recurso, da grande gama de problemas que podem ser abordados e da variedade de arquiteturas distintas (de memória compartilhada, memória dis-

tribuída, com processadores de um ou múltiplos núcleos) com as quais é compatível. Além disso, a OpenMP é de simples familiarização por parte do programador, e de fácil inserção das diretivas no código, dado que já se possui um código sequencial em mãos.

Ainda, a possibilidade de escrever programas paralelos para computadores com apenas um processador, fazendo uso de *multithreading*, implica em menos exigências de hardware. Isto torna possível que usuários que não dispõem de clusters, ou mesmo de máquinas com multiprocessadores, ainda assim façam uso dos programas e usufruam dos ganhos de tempo.

4 *SOURCE ITERATION*

A equação linear de Boltzmann, também conhecida como equação de transporte de partículas neutras, descreve o transporte de nêutrons em meios materiais via princípios de conservação. Aqui trataremos de um problema unidimensional, monoenergético, estacionário, com espalhamento isotrópico, em meio homogêneo e com uma fonte q que independe da variável angular μ. A equação para este problema é escrita, para $0 \leq x \leq L$, como

$$\begin{gathered} \mu \tfrac{\partial}{\partial x}\psi(x,\mu) + \sigma_T \psi(x,\mu) = \tfrac{1}{2}\sigma_S \int_{-1}^{1} \psi(x,\mu')d\mu' + q(x) \\ \psi(0,\mu) = \psi_e(\mu), 0 < \mu \leq 1 \\ \psi(L,\mu) = \psi_d(\mu), \ -1 \leq \mu < 0, \end{gathered} \tag{4.1}$$

na qual σ_S é a seção de choque macroscópica de espalhamento, σ_T é a seção de choque macroscópica total $\psi(x,\mu)$ é o fluxo angular de nêutrons, $q(x)$ é uma fonte conhecida e μ é o cosseno do ângulo formado pela direção na qual a partícula viaja com o eixo do domínio da variável espacial. As quantidades $\psi_e(\mu)$ e $\psi_d(\mu)$ são condições de contorno para o problema [1].

O *Source Iteration* (SI) é o método iterativo mais simples [1, 34]. Pode ser descrito matematicamente, em sua i-ésima iteração, como

$$\mu \frac{\partial}{\partial x}\psi^{(i+1)}(x,\mu) + \sigma_T \psi^{(i+1)}(x,\mu) = \frac{1}{2}\sigma_S \int_{-1}^{1} \psi^{(i)}(x,\mu')d\mu' + q(x). \tag{4.2}$$

A quantidade $\psi^{(0)}(x,\mu)$ é fornecida como estimativa inicial para o processo iterativo. Quando calculada a nova estimativa para ψ, através da equação (4.2), é também calculada a nova estimativa para o fluxo escalar ϕ, dado por [20]

$$\phi(x) = \int_{-1}^{1} \psi(x,\mu')d\mu'. \tag{4.3}$$

O erro relativo para o fluxo escalar ϕ é dado por

$$e_r = \left\| \frac{\phi_{j\pm1/2}^{(i+1)} - \phi_{j\pm1/2}^{(i)}}{\phi_{j\pm1/2}^{(i+1)}} \right\|_{\infty}. \tag{4.4}$$

Na equação (4.4), os índices i indicam a iteração e os índices j indicam em qual posição $x_{j\pm1/2}$ a quantidade ϕ foi calculada. Quando o erro e_r for menor do que uma tolerância ϵ pré-determinada (ou quando o número máximo de iterações for atingido) o processo iterativo é interrompido.

Utilizamos o SI junto do método de ordenadas discretas, que aproxima a equação (4.1) com respeito à variável angular, calculando fluxo angular em direções pré-determinadas. Para isto, o termo integral é expresso através de um esquema de quadratura, que neste trabalho foi Gauss-Legendre. O método de ordenadas discretas foi utilizado no tratamento da variável angular nos capítulos 5, 6 e 7. Assim obtemos, para $0 \leq x \leq L$,

$$\mu_m \frac{d}{dx}\psi(x,\mu_m) + \sigma_T \psi(x,\mu_m) = \frac{1}{2}\sigma_S \sum_{n=1}^{N} \psi(x,\mu_n)\omega_n + q(x). \tag{4.5}$$

Em (4.5), a derivada que era parcial passa a ser ordinária, N (par) é a ordem da quadratura de Gauss-Legendre e as direções μ_m dos fluxos angulares são as raízes dos polinômios de Legendre de grau N. Os índices $m = 1,...,N$, assim como os índices n, indicam em qual direção discreta a quantidade ψ foi calculada. Os polinômios de Legendre de grau k são dados recursivamente por [11]

$$P_0(x) = 1$$

$$P_1(x) = x$$

$$kP_k(x) = (2k-1)xP_{k-1}(x) - (k-1)P_{k-2}(x). \tag{4.6}$$

Os pesos ω_i podem ser dados por

$$\omega_i = \frac{2}{(1-\mu_i^2)(P_n'(\mu_i))^2}. \tag{4.7}$$

Ainda, P_n' também é dado de forma recursiva por

$$P_0(x) = 0$$

$$P_1(x) = 1$$

$$P_k'(x) = \frac{k}{x^2-1}(xP_k(x) - P_{k-1}(x)). \tag{4.8}$$

Quando N é um número par, a quadratura de Gauss-Legendre traz raízes simetricamente opostas em relação à origem. Por simplicidade de notação, escreveremos $\psi(x,\mu_m) = \psi_m(x)$.

Para discretizar a variável espacial, consideraremos que o domínio unidimensional é composto de células de tamanho h_j e que as seções de choque σ_S e σ_T são constantes em todo o domínio.

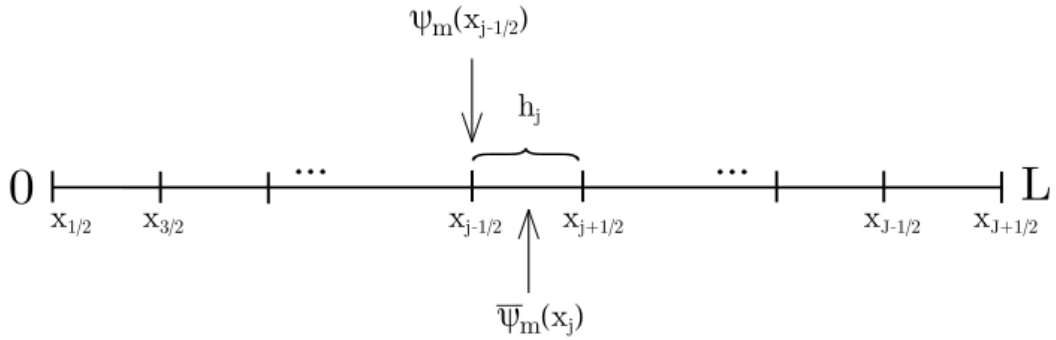


Figura 4.1: A discretização do domínio com respeito à variável espacial. A quantidade ψ é calculada nas paredes das células, enquanto $\overline{\psi}$ é calculada no centro

A equação (4.5) então pode ser escrita como

$$\frac{\mu_m}{h_j}(\psi_m(x_{j+1/2}) - \psi_m(x_{j-1/2})) + \sigma_T \overline{\psi}_m(x_j) = \frac{1}{2}\sigma_S \sum_{n=1}^{N} \overline{\psi}_n(x_j)\omega_n(x) + q(x_j), \quad (4.9)$$

na qual $\psi_m(x_{j+1/2})$ e $\psi_m(x_{j-1/2})$ são os fluxos angulares nas paredes da célula j e o fluxo angular médio no interior de cada célula é dado por

$$\overline{\psi}_m(x_j) = \frac{1}{h_j}\int_{x_{j-1/2}}^{x_{j+1/2}} \psi_m(x)dx. \quad (4.10)$$

Ainda, para a determinação dos fluxos médios, utilizamos o método de varredura chamado de *Diamond Difference*, dado por [1]

$$\overline{\psi}_m(x_j) = \frac{\psi_m(x_{j+1/2}) + \psi_m(x_{j-1/2})}{2}, \quad (4.11)$$

que aproxima o fluxo médio pela média dos fluxos nos contornos da célula.

A varredura é feita inicialmente da esquerda para a direita, calculando os fluxos quando $\mu_m > 0$, e em seguida da direita para a esquerda, calculando os fluxos quando $\mu_m < 0$. O fluxo escalar ϕ é aproximado na posição x como

$$\phi(x) = \sum_{n=1}^{N} \overline{\psi}_n(x)\omega_n. \tag{4.12}$$

4.1 O Algoritmo & Paralelização

É importante que esteja claro como o método acima foi implementado na sua versão sequencial. Esquematicamente, o código é executado nos seguintes passos:

1. Inicializar fluxos angulares médios em todas as células.
2. Varredura: para a direita, ou seja, i = 1,2,...,J e em seguida para a esquerda, com i = J,$J-1$,...,1 obtendo fluxos angulares ψ nas direções determinadas.
3. Calcular fluxo escalar ϕ.
4. Testar ϕ para convergência: se o erro relativo entre ϕ da iteração atual e ϕ calculado na iteração anterior for menor do que uma tolerância ϵ dada, ou quando o número máximo de iterações for atingido, o processo é interrompido. Senão, volta ao passo 2.
5. Calcular fluxo médio $\overline{\psi}$ e fluxo escalar nas paredes das células.

Ainda, regiões paralelas não permitem ramificações [25]. Isto induz uma troca de ordem entre os passos 4 e 5, para que não seja necessário abrir e fechar duas regiões paralelas por iteração. Claramente, esta troca introduz operações a

mais na última iteração, porém não em quantidade significativa e isto não ocorre em nenhuma outra iteração, apenas na última.

No passo 2 são percorridas todas as células do domínio e, em cada célula, todas as direções prescritas. Além disso, a cada célula, por direção são executados 10 acessos a elementos de arranjos e 27 operações aritméticas, das quais 9 são multiplicações e 3 são divisões. Estes aspectos fazem da varredura o trecho computacionalmente mais custoso.

As varreduras também são, no entanto, pouco amigáveis a paralelizações, pois o cálculo dos fluxos na célula j depende dos fluxos na célula $j-1$ (se $\mu_m > 0$) ou na célula $j+1$ (se $\mu_m < 0$). Uma distribuição direta de trabalho entre as *threads* não funcionaria neste caso [25, 18]

A estratégia adotada para que a varredura pudesse ser distribuída foi dividir o domínio em duas partes para que ambas sejam processadas simultaneamente. Para isto, fizemos uso da diretiva `!$OMP SECTIONS`. O primeiro bloco comporta as primeiras $\frac{J}{2}$ células, em sua totalidade numeradas da esquerda para a direita, e o segundo bloco comporta as células restantes. Cada bloco deve ser varrido para a direita e para a esquerda. Deste modo, o algoritmo em sua versão paralela pode ser escrito como

1. Inicializar fluxos angulares médios em todos os nodos.
2. Quatro varreduras conduzidas simultaneamente: para a direita, com $i = 1,2,.J/2$ e $i = J/2,...,J$ e para a esquerda, com $i = J,J-1,...,J/2$ e $i = J/2,...,1$ obtendo fluxos angulares ψ nas direções determinadas.
3. Calcular fluxo escalar ϕ.
4. Calcular fluxo médio $\overline{\psi}$ e fluxo escalar nas paredes das células.

5. Testar ϕ para convergência: se o erro relativo entre ϕ da iteração atual e ϕ calculado na iteração anterior for menor do que uma tolerância ϵ dada, ou quando o número máximo de iterações for atingido, o processo é interrompido. Senão, volta ao passo 2.

Digamos, por exemplo, que estamos conduzindo a varredura para a direita ($\mu_m > 0$) na primeira iteração. No primeiro bloco, a parede direita da primeira célula terá seus fluxos angulares calculados a partir da condição de contorno dada em $x = 0$ e todas as células até a célula central, que chamaremos de $x^{(c)}$ (e sua parede direita, que chamaremos de $x^{(c)}_{j+1/2}$, inclusive) terão seus fluxos angulares calculados a partir de células anteriores, conforme o esperado. Assim, apenas quando a varredura do primeiro bloco estiver terminada, o fluxo angular em $x^{(c)}_{j+1/2}$ será um valor calculado que reflete o valor esperado pelo método. Quando a varredura do segundo bloco inicia, no entanto, o fluxo angular em $x^{(c)}_{j+1/2}$ ainda não é um valor calculado, pois as varreduras dos dois blocos iniciam simultaneamente. Para que não seja necessário esperar que encerre a varredura do primeiro bloco, atribuimos a estimativa inicial $\psi^{(0)}$ como fluxo angular em $x^{(c)}_{j+1/2}$ para que a varredura do segundo bloco inicie.

Em função disso, todos os fluxos angulares do segundo bloco serão calculados incorretamente, pois serão calculados a partir de uma estimativa inicial que pode ser completamente arbitrária. Quando as varreduras do primeiro e do segundo bloco encerram, o último fluxo angular a ser calculado pelo primeiro bloco é precisamente em $x^{(c)}_{j+1/2}$. Isto quer dizer que, quando o segundo bloco iniciar a varredura para a direita na segunda iteração, o valor que antes era uma estimativa inicial terá sido sobrescrito por um valor calculado que reflete o valor esperado. Um aspecto positivo é que, a partir da segunda iteração, os fluxos do segundo bloco serão calculados a partir de um valor não mais completamente arbitrário, mas certamente melhor calibrado; um aspecto negativo é que este valor, ainda que mais próximo

do fluxo angular real, é o valor calculado pela iteração anterior (cf. apêndice A.1 e A.2).

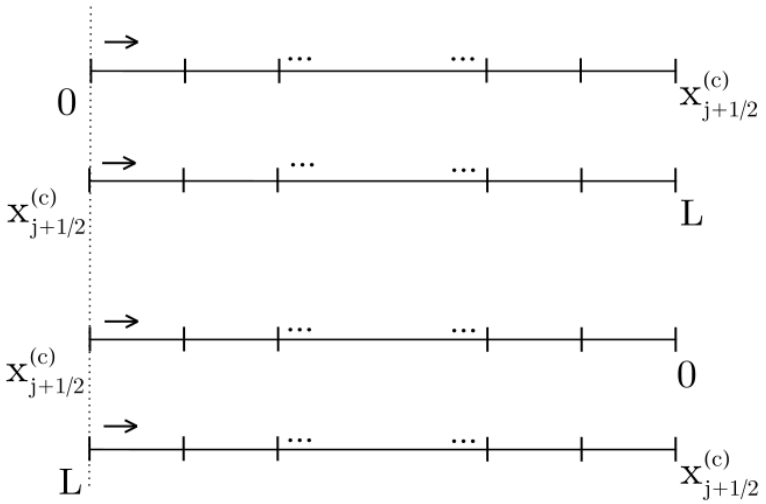


Figura 4.2: Os quatro blocos a serem varridos, iniciando todos ao mesmo tempo.

É válido ressaltar que esta discussão, feita acima para $\mu_m > 0$, acontece de maneira igualmente problemática para $\mu_m < 0$. Na primeira iteração, o primeiro bloco contará com valores calculados a partir da condição de contorno em $x = L$ e o primeiro bloco terá fluxos angulares incorretos calculados a partir da estimativa inicial em $x^{(c)}_{j+1/2}$. Estimativa inicial esta que, ao final da primeira iteração, será sobrescrita por um valor calculado e utilizado para iniciar a varredura do primeiro bloco na segunda iteração.

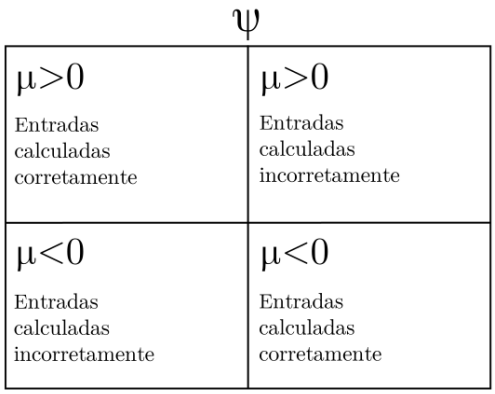


Figura 4.3: Estrutura da matriz que representa ψ após o processo de varredura.

De fato, ao final da primeira iteração, a matriz que representa ψ tem apenas metade das entradas que refletem o valor esperado pelo método, ao passo que a outra metade é calculada a partir de estimativas iniciais (figura 4.3). Intuitivamente, é esperado que, com todas as estimativas extras e fluxos calculados incorretamente, o número de iterações até a convergência suba. Da mesma forma é esperado que o tempo de execução de uma iteração diminua. É preciso avaliar, neste momento, se o tempo de execução de cada iteração diminui o suficiente para compensar iterações a mais, ou se o número de iterações aumenta a ponto de a versão paralela apresentar tempo de execução maior do que a versão sequencial.

4.2 Problemas teste iniciais

Visando obter informações sobre o número de iterações até a convergência, bem como sobre o tempo de execução da versão paralela do SI, alguns problemas teste foram executados. Outros métodos iterativos estudados neste trabalho, detalhados nos capítulos 5 e 7, não sofreram divisão na varredura do domínio e o número de iterações das versões paralela e sequencial permaneceu, portanto,

inalterado. Por este motivo, os problemas teste desta seção não foram executados para todos os métodos.

De certo modo, o intuito dos problemas teste iniciais é testar se esta é uma estratégia de paralelização válida, ou seja, se ela falhar em problemas computacionalmente simples, não é provável que traga bons resultados em problemas mais duros. Assim, foram escolhidos valores moderados de número de células e de direções nas discretizações das variáveis espacial e angular, bem como nas seções de choque. Deste modo, os problemas não são nem computacionalmente tão custosos, nem tão lentamente convergentes quanto poderiam.

O primeiro problema teste consiste de um domínio unidimensional homogêneo, de comprimento $L = 10\,\mathrm{cm}$, discretizado em 20000 células. A variável angular foi discretizada em 256 direções e o parâmetro σ_T foi mantido em $1\,\mathrm{cm}^{-1}$. É sabido que, conforme o quociente σ_S/σ_T se aproxima de 1, o número de iterações até a convergência cresce [29]. Por conta desta influência o parâmetro σ_S foi variado, assumindo os valores $0{,}2\,\mathrm{cm}^{-1}$, $0{,}4\,\mathrm{cm}^{-1}$, $0{,}8\,\mathrm{cm}^{-1}$ e $0{,}995\,\mathrm{cm}^{-1}$. As condições de contorno foram mantidas em 1 nêutron/$\mathrm{cm}^2\cdot$s em ambas as extremidades do domínio, a tolerância fornecida para o processo iterativo foi $\epsilon = 10^{-10}$, o domínio não possui fonte e o número máximo de iterações foi mantido em 5000. Os resultados estão expostos na tabela 4.1

	$\sigma_S = 0{,}2$		$\sigma_S = 0{,}4$	
	Iterações	Tempo (s)	Iterações	Tempo(s)
Sequencial	17	3,638000	28	5,885000
Paralelo	21	3,400000	33	4,673000

	$\sigma_S = 0{,}8$		$\sigma_S = 0{,}995$	
	Iterações	Tempo (s)	Iterações	Tempo (s)
Sequencial	94	19,308000	669	138,43600
Paralelo	104	13,388000	724	105,24404

Tabela 4.1: Número de iterações e tempo de execução do SI para cada caso do problema homogêneo nas versões sequencial e paralela

É possível perceber que, como esperado, o número de iterações até a convergência da versão paralela de fato foi maior do que o número de iterações da versão sequencial. Notemos que o tempo médio de execução de cada iteração na versão paralela é significativamente menor do que o tempo médio de execução de cada iteração na versão sequencial.[1] Isto ilustra o fato de que, apesar de apresentar maior número de iterações até a convergência, a versão paralela foi executada em menos tempo em todos os testes executados.

O segundo problema teste consiste de um domínio heterogêneo, de comprimento total $L = 25\text{cm}$, composto de três regiões com parâmetros distintos. A primeira região, de comprimento $L = 10\text{cm}$ e parâmetros $\sigma_S = 0{,}92\text{cm}^{-1}$ e $\sigma_T = 1\text{cm}^{-1}$, foi discretizada em 4000 células. A segunda região, de comprimento $L = 5\,\text{cm}$, foi discretizada em 2000 células, com parâmetros $\sigma_T = 1\,\text{cm}^{-1}$ e σ_S foi variado, assu-

[1] Tempo médio de execução de cada iteração foi calculado como T/K, onde T é o tempo total de execução e K é o número de iterações até a convergência. Para $\sigma_S = 0{,}995$, por exemplo, cada iteração da versão sequencial apresentou tempo de execução de, em média, 0,206929 segundos. Na versão paralela, no entanto, cada iteração apresentou tempo de execução de, em média, 0,145027 segundos.

mindo os valores assumindo os valores 0,2 cm^{-1}, 0,4 cm^{-1}, 0,8 cm^{-1} e 0,995 cm^{-1}. Esta região também possui uma fonte fixa $q(x) = 1$ nêutron/cm^3. A terceira região é idêntica à primeira. As condições de contorno foram mantidas em 1 nêutron/cm^2·s em ambas as extremidades do domínio A variável angular foi discretizada em 256 direções ao longo de todo o domínio.

Problemas heterogêneos, ao longo de todo este trabalho, são resolvidos dividindo o domínio em regiões com diferentes seções de choque e resolvendo o problema em cada região, fazendo uso de condições de continuidade no fluxo angular ao transitar de uma região para outra. Os resultados estão expostos na tabela 4.2.

	$\sigma_S = 0{,}2$		$\sigma_S = 0{,}4$	
	Iterações	Tempo (s)	Iterações	Tempo(s)
Sequencial	215	22,439001	216	21,966001
Paralelo	215	15,695000	216	14,031000

	$\sigma_S = 0{,}8$		$\sigma_S = 0{,}995$	
	Iterações	Tempo (s)	Iterações	Tempo (s)
Sequencial	227	23,775001	527	53,582002
Paralelo	228	17,222000	571	39,568001

Tabela 4.2: Número de iterações e tempo de execução para cada caso do problema heterogêneo nas versões sequencial e paralela

Podemos perceber que o número de iterações até a convergência da versão paralela de fato apresentou aumentos quando comparado com a versão sequencial. Podemos observar que $\sigma_S = 0{,}995\,\mathrm{cm}^{-1}$ apresentou substancial aumento de 44 iterações, enquanto $\sigma_S = 0{,}8\,\mathrm{cm}^{-1}$ apresentou aumento de uma iteração apenas, e $\sigma_S = 0{,}2\,\mathrm{cm}^{-1}$ e $\sigma_S = 0{,}4\,\mathrm{cm}^{-1}$ simplesmente não apresentaram aumento. Isto ilustra o fato de que, apesar de o aumento no número de iterações ser esperado, este

não necessariamente ocorre em todos os casos, ou, ainda, pode não ocorrer aumento significativo.

Ainda, podemos notar que os tempos de execução da versão paralela, mesmo com iterações a mais, é menor do que os tempos de execução da versão sequencial. Isto se deve, assim como no problema homogêneo, ao fato de que cada iteração do programa paralelo tem tempo médio de execução muito menor do que cada iteração do programa sequencial, o que diminui o tempo total de execução. Outros problemas teste serão executados e descritos no capítulo 8.

5 *DIFFUSION SYNTHETIC ACCELERATION*

Outros métodos iterativos também fazem uso de varreduras, porém com uso de correções que fazem com que o número de iterações até a convergência diminua. O *Diffusion Synthetic Acceleration* (DSA) faz uso da aproximação de difusão pra calcular uma correção aditiva. O DSA, assim com o SI, inicia com uma varredura do domínio que gera um valor para ψ, ao qual chamaremos $\psi^{(i+1/2)}$, e que pode ser escrita como [1, 4]:

$$\begin{gathered}\mu\frac{\partial}{\partial x}\psi^{(i+1/2)}(x,\mu)+\sigma_T\psi^{(i+1/2)}(x,\mu)=\tfrac{1}{2}\sigma_S\int_{-1}^{1}\psi^{(i)}(x,\mu')d\mu'+q(x) \qquad (5.1)\\ \psi^{(i+1/2)}(0,\mu)=\psi_e(\mu),\,0<\mu\leq 1\\ \psi^{(i+1/2)}(L,\mu)=\psi_d(\mu),\,-1\leq\mu<0.\end{gathered}$$

A discretização das variáveis espacial e angular foi dada do mesmo modo que no SI, pela equação (4.9), e a varredura foi feita utilizando o *Diamond Difference*, pela equação (4.11). A equação (5.1) discretizada pode ser escrita como

$$\begin{gathered}\frac{\mu_m}{h_j}[\psi_m^{(i+1/2)}(x_{j+1/2})-\psi_m^{(i+1/2)}(x_{j-1/2})]+\sigma_T\psi_m^{(i+1/2)}(x_j)=\\ =\frac{1}{2}\sigma_S\sum_{n=1}^{N}\psi_n^{(i)}(x_j)\omega_n(x)+q(x_j).\end{gathered} \qquad (5.2)$$

Subtraindo a equação (5.1) da equação (4.1), obtemos

$$\begin{gathered}\mu\frac{\partial}{\partial x}f(x,\mu)+\sigma_T f(x,\mu)-\tfrac{1}{2}\sigma_S\int_{-1}^{1}f(x,\mu')d\mu'=\tfrac{1}{2}\sigma_S[\phi^{(i+1/2)}(x)-\phi^{(i)}(x)] \qquad (5.3)\\ f(0,\mu)=0,\,0<\mu\leq 1\\ f(L,\mu)=0,\,-1\leq\mu<0.\end{gathered}$$

na qual f e $\phi^{(i+1/2)}$ foram definidas como

$$f(x,\mu) = \psi(x,\mu) - \psi^{(i+1/2)}(x,\mu), \tag{5.4}$$

$$\phi^{(i+1/2)} = \int_{-1}^{1} \psi^{(i+1/2)}(x,\mu)d\mu. \tag{5.5}$$

Neste caso, $f(x,\mu)$ é a correção aditiva que precisa ser calculada. A equação (5.3), no entanto, é um problema de transporte para f tão difícil de ser resolvida quanto a equação (4.1) para ψ. Uma aproximação utilizada para a equação (5.3) é [1]

$$-\frac{d}{dx}\frac{1}{3\sigma_t}\frac{dF^{(i+1)}}{dx}(x) + \sigma_a F^{(i+1)} = \sigma_s[\phi^{(i+1/2)}(x) - \phi^{(i)}(x)] \tag{5.6}$$

$$F^{(i+1)}(0) - \frac{2}{3\sigma_t(0)}\frac{dF^{(i+1)}}{dx}(0) = 0$$

$$\frac{dF^{(i+1)}}{dx}(L) = 0.$$

A equação (5.6) é chamada de *Aproximação de difusão*, ou *Pré - condicionamento de difusão*, ou *Aproximação* P_1 [1]. Nela, σ_a é a seção de choque de absorção e $F^{(i+1)}$ é uma aproximação para a componente de fluxo escalar de f, ou seja,

$$F^{(i+1)}(x) \approx \int_{-1}^{1} f(x,\mu)d\mu. \tag{5.7}$$

Integrando a equação (5.4) com relação a μ entre -1 e 1, e utilizando a aproximação (5.7) e a equação (5.5), obtemos

$$\phi^{(i+1)} = \phi^{(i+1/2)} + F^{(i+1)}, \tag{5.8}$$

o que nos mostra que $F^{(i+1)}$ é a correção aditiva a ser calculada antes que o método reitere. A discretização das variáveis espacial e angular foi dada do mesmo modo que

no SI, pela equação (4.9), e a varredura foi feita utilizando o *Diamond Difference*, pela equação (4.11). O problema para F é resolvido numericamente pela rotina DGTSVX, que computa a solução de um sistema de equações lineares tridiagonal utilizando a fatoração LU da matriz do sistema [3].

5.1 O Algoritmo & Paralelização

O DSA, esquematicamente, pode ser descrito pelos passos abaixo.

1. Inicializar fluxos angulares médios em todas as células.
2. Varredura: para a direita e em seguida para a esquerda, obtendo fluxos angulares $\psi^{(i+1/2)}$ nas direções determinadas.
3. Calcular fluxo escalar $\phi^{(i+1/2)}$ utilizando a equação (4.12).
4. Calcular $F^{(i+1)}$ utilizando a equação (5.6)
5. Calcular $\phi^{(i+1)}$ utilizando a equação (5.8)
6. Testar $\phi^{(i+1)}$ para convergência: se o erro relativo entre $\phi^{(i+1)}$ e $\phi^{(i)}$ for menor do que uma tolerância ϵ dada, ou quando o número máximo de iterações for atingido, o processo é interrompido. Senão, volta ao passo (2).

Podemos perceber que os métodos SI e DSA coincidem até o passo 3. Este estágio inicial consiste uma varredura do domínio, feita utilizando o método *Diamond Difference* descrito na equação (4.11), e é referido como *estágio de alta ordem*. Os passos 4 e 5 formam o segundo estágio de cada iteração, referido como *estágio de baixa ordem* [1].

Apesar de possuir uma estrutura similar ao SI até determinada etapa, o DSA não respondeu bem à estratégia de fornecer uma estimativa extra na célula central e dividir a varredura em quatro blocos (figura 4.2).

A quantidade $\psi^{(i+1/2)}$, no código, é uma matriz com N linhas e j colunas, sendo N o número de direções e j o número de células utilizadas na discretização. Novamente, $\psi^{(i+1/2)}$ possui apenas metade das suas entradas calculadas corretamente (figura 4.3). A ausência do estágio de baixa ordem no SI fez com que esta estimativa extra na célula central não causasse aumento significativo no número de iterações. Observando a equação (4.12), vemos que

$$\phi(x) = \sum_{n=1}^{N/2} \psi_n(x)\omega_n + \sum_{n=N/2+1}^{N} \psi_n(x)\omega_n \tag{5.9}$$

Como apenas metade das entradas de $\psi^{(i+1/2)}$ foram calculadas corretamente, para cada x, apenas um dos termos do lado direito da equação (5.9) é consistente com o que o método espera utilizar no cálculo da correção $F^{(i+1)}$, o outro termo foi calculado a partir de estimativas arbitrárias.[1] Isto faz com que a correção $F^{(i+1)}$ não seja calculada corretamente, o que pode causar aumento no número de iterações com relação à versão sequencial do programa.

Apenas um teste foi feito neste caso, para investigar se existia a possibilidade de o número de iterações aumentar, porém não a ponto de apresentar tempo de execução maior do que a versão sequencial. O problema testado foi similar ao problema homogêneo executado no capítulo 4. Para este problema teste, foram comparados o código sequencial e uma versão paralela na qual apenas a varredura foi distribuída. A ideia era que, antes de distribuir os outros trechos do código,

[1] Qual dos dois termos foi verdadeiramente calculado e qual foi calculado incorretamente, depende de x. Se $0 \leq x \leq L/2$, então $\sum_{n=1}^{N/2} \psi_n(x)\omega_n$ é o termo calculado corretamente; se $L/2 < x \leq L$, então $\sum_{n=N/2+1}^{N} \psi_n(x)\omega_n$ é o termo calculado corretamente.

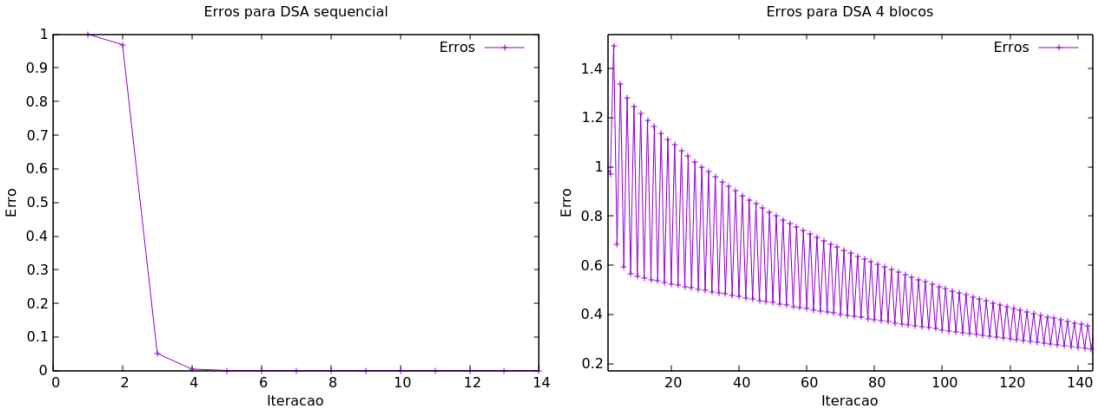
fosse possível obter informações sobre o número de iterações até a convergência sob a influência da estimativa extra na célula central.

O problema teste consistiu de um domínio unidimensional homogêneo, de comprimento $L = 10\,\text{cm}$, discretizado em 20000 células. A variável angular foi discretizada em 256 direções e as seções de choque σ_T e σ_S foram mantidas em $1\,\text{cm}^{-1}$ e $0.8\,\text{cm}^{-1}$ respectivamente. As condições de contorno foram mantidas em 1 nêutron/cm^2·s em ambas as extremidades do domínio, a tolerância fornecida para o processo iterativo foi $\epsilon = 10^{-10}$ e o número máximo de iterações foi mantido em 5000.

	Iterações	Tempo total (s)	Tempo médio de uma iteração (s)
Sequencial	14	3,181000	0,227214
Paralelo	3264	480,6530	0,147258

Tabela 5.1: Tempo de execução e número de iterações até a convergência das versões paralela e sequencial.

É possível perceber, pela tabela 5.1, que a estimativa extra na célula central e a divisão da varredura em quatro blocos não contribuíram para a diminuição do tempo de execução deste problema. Apesar de o tempo médio de execução de cada iteração ter diminuído, não foi o suficiente para compensar todas as 3250 iterações a mais executadas pela versão paralela.



(a) Sequencial (b) Paralelo

Figura 5.1: Erro relativo a cada iteração, para as versões paralela e sequencial.

Podemos observar o comportamento dos erros relativos do fluxo escalar ϕ em cada iteração (dados pela equação (4.4)) na figura 5.1. Os erros relativos na versão sequencial caem rapidamente e não aumentam, enquanto os erros relativos na versão paralela oscilam bruscamente e diminuem lentamente (foram necessárias 294 iterações para que o erro atingisse a ordem de 10^{-2}).

Como esta estretégia apresentou resutados pobres para $\sigma_S = 0{,}8$, não é esperado que o desempenho melhore para $\sigma_S > 0{,}8$. Portanto, ao invés de investigar como a versão paralela do código desempenha para σ_S/σ_T mais próximos de 1, foi preferível mudar a estratégia de paralelização.

Assim, lembrando que as varreduras (para $\mu > 0$ e $\mu < 0$) podem ser conduzidas independentemente, a segunda estratégia de paralelização testada para o DSA foi utilizar a diretiva `!$OMP SECTIONS` para processar apenas estes dois blocos, ao invés dos quatro pretendidos na estratégia anterior.

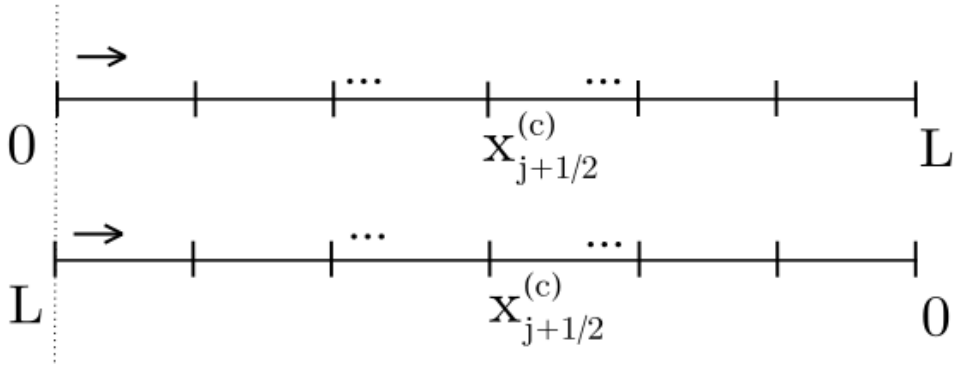


Figura 5.2: Os dois blocos a serem varridos, iniciando todos ao mesmo tempo.

Esta proposta apresenta duas grandes desvantagens. A primeira consiste no fato de que qualquer time de *threads* que conte com mais de duas componentes terá *threads* ociosas durante a varredura. A segunda diz respeito ao tempo médio de execução de cada iteração, que é esperado que seja menor do que na versão sequencial, porém não menor do que na estratégia de quatro blocos, adotada no SI.

Uma vantagem da abordagem com apenas dois blocos é a ausência de estimativas extras. As quantidades $\psi^{(i+1/2)}$, quando da saída da varredura, e $\phi^{(i+1/2)}$, quando calculada pela equação (4.12), serão integralmente o que o método espera para calcular a correção $F^{(i+1)}$ [4, 28]. Isto faz com que o número de iterações da versão paralela não mude em relação à versão sequencial. Neste sentido, qualquer ganho no tempo médio de execução de cada iteração pode ser convertido em ganho no tempo total de execução. Outros problemas teste serão executados no capítulo 8.

6 *COARSE MESH FINITE DIFFERENCE*

Assim como o DSA, o *Coarse Mesh Finite Difference* (CMFD) é um método iterativo que faz uso de varredura e de um esquema de aceleração para diminuir o número de iterações até a convergência. É um método composto de dois estágios, um de alta ordem, que contempla a varredura do domínio em uma malha fina, e um de baixa ordem, que faz uso de uma aproximação não linear de difusão para obter uma estimativa para o fluxo escalar em uma malha grossa. A malha fina possui mais células do que a malha grossa.

Para tal, este método lida com duas discretizações diferentes na variável espacial. A discretização do estágio de alta ordem foi a mesma utilizada nos dois métodos anteriores (como na equação (5.2)) e a varredura foi feita utilizando o *Diamond Difference*, pela equação (4.11).

Saídas da varredura, temos as quantidades

$$\phi_1^{(i+1/2)}(x_{j+1/2}) = \sum_{n=1}^{N} \mu_n \psi_n^{(i+1/2)}(x_{j+1/2})\omega_n, \tag{6.1}$$

e

$$\phi_0^{(i+1/2)}(x_j) = \sum_{n=1}^{N} \psi_n^{(i+1/2)}(x_j)\omega_n. \tag{6.2}$$

A equação (6.1) descreve a corrente na parede das células finas e a equação (6.2) descreve o fluxo escalar em cada célula fina, sobre o qual a correção será calculada [27].

O estágio de baixa ordem é conduzido sobre uma malha grossa, composta de K células, na qual X_k, a k-ésima célula grossa, é composta de p_k células finas. Chamaremos de P_k o número de células finas nas primeiras k células gros-

sas. Deste modo, podemos escrever a parede direita da k-ésima célula grossa como $X_{k+1/2}$ ou $x_{P_k+1/2}$.

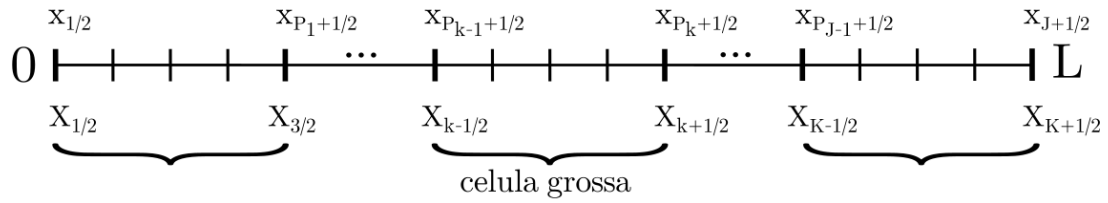


Figura 6.1: A discretização do domínio, na malha fina e na malha grossa.

É necessário introduzir a notação para as quantidades calculadas para as células grossas. Denotaremos como $\sum_{j\in k}$ a soma das j células finas na k-ésima célula grossa. Assim, definimos

$$\Delta_k = \sum_{j\in k} h_j, \tag{6.3}$$

$$\Phi_0(X_k) = \frac{1}{\Delta_k}\sum_{j\in k} \phi_0(x_j)h_j, \tag{6.4}$$

$$Q(X_k) = \frac{1}{\Delta_k}\sum_{j\in k} q(x_j)h_j, \tag{6.5}$$

$$\Phi_1(X_{k+1/2}) = \phi_1(x_{P_k+1/2}), \tag{6.6}$$

$$\Sigma_u = \frac{1}{\Delta_k}\sum_{j\in k} \sigma_u h_j \tag{6.7}$$

e

$$\Sigma_u^{(i+1/2)} = \frac{\sum_{j\in k} \sigma_u \phi_0^{(i+1/2)}(x_j)h_j}{\sum_{j\in k} \phi_0^{(i+1/2)}(x_j)h_j}. \tag{6.8}$$

Na k-ésima célula grossa, de comprimento Δ_k, $\Phi_0(X_k)$ representa o fluxo escalar e Q_k é um termo fonte. O termo $\Phi_1(X_{k+1/2})$ representa a corrente na parede direita da k-ésima célula grossa. As quantidades Σ_u e $\Sigma_u^{(i+1/2)}$ são, respectivamente, a média volumétrica das seções de choque nas células finas que compõem a k-ésima célula grossa e a média ponderada das seções de choque das células finas utilizando fluxo escalar em cada uma delas como peso. Nas equações (6.7) e (6.8), o subscrito u pode assumir os valores S, T ou γ, indicando seção de choque de espalhamento, seção de choque total ou seção de choque de captura radioativa.

No contexto deste trabalho estamos considerando que os domínios são homogêneos ou compostos de regiões homogêneas, ou seja, as seções de choque permanecem constantes ao longo de um conjunto de células. Cada célula grossa é composta de células finas que possuem a mesma seção de choque σ_u. Mais comumente, como em [27], se considera que as seções de choque são constantes em cada célula, e se utiliza a notação $\sigma_{u,j}$ para indicar a seção de choque na j-ésima célula fina ou $\Sigma_{u,k}$ para indicar a seção de choque na k-ésima célula grossa,

Multiplicando a equação (5.2) por ω_n e somando sobre todos os n, temos

$$\frac{1}{h_j}\Big(\phi_1^{(i+1/2)}(x_{j+1/2}) - \phi_1^{(i+1/2)}(x_{j-1/2})\Big) + \sigma_T\phi_0^{(i+1/2)}(x_j) = \sigma_S\phi_0^{(i)}(x_j) + q(x_j). \quad (6.9)$$

Em seguida, multiplicando a equação (6.9) por h_j e somando $\sum_{j\in k}$, podemos escrever, para a k-ésima célula grossa [26, 27]

$$\begin{aligned}\Phi_1^{(i+1/2)}(X_{k+1/2}) - \Phi_1^{(i+1/2)}(X_{k-1/2}) + \Sigma_T^{(i+1/2)}\Phi_0^{(i+1/2)}(X_k)\Delta_k = \\ = \Sigma_S^{(i)}\Phi_0^{(i)}(X_k)\Delta_k + Q(X_k)\Delta_k.\end{aligned} \quad (6.10)$$

No interior de cada célula grossa, definimos a quantidade $\hat{D}_{k+1/2}^{(i+1/2)}$ como

$$\Phi_1^{(i+1/2)}(X_{k+1/2}) = -\frac{2}{3}\left(\frac{\Phi_0^{(i+1/2)}(X_{k+1}) - \Phi_0^{(i+1/2)}(X_k)}{\Sigma_T^{(i+1/2)}\Delta_{k+1} + \Sigma_T^{(i+1/2)}\Delta_k}\right) + \\ +\hat{D}_{k+1/2}^{(i+1/2)}\Big(\Phi_0^{(i+1/2)}(X_{k+1}) + \Phi_0^{(i+1/2)}(X_k)\Big). \tag{6.11}$$

Neste caso, $\hat{D}_{k+1/2}^{(i+1/2)}$ é uma correção de transporte de acordo com a Lei de Fick[1]. Primeiramente, são calculadas as quantidades $\Phi_1^+(X_{1/2})$ e $\Phi_1^+(X_{K+1/2})$:

$$2\Phi_1^+(X_{1/2}) = \sum_{n=1}^{N}(\mu_n + |\mu_n|)\psi_n^{(i+1/2)}(x_{1/2})\omega_n \tag{6.12}$$

e

$$2\Phi_1^+(X_{K+1/2}) = \sum_{n=1}^{N}(-\mu_n + |\mu_n|)\psi_n^{(i+1/2)}(x_{J+1/2})\omega_n. \tag{6.13}$$

Utilizando as equações (6.12) e (6.13) (referidas no apêndice A.3) são definidas as quantidades $B_{1/2}^{(i+1/2)}$ e $B_{K+1/2}^{(i+1/2)}$ nas fronteiras esquerda e direta do domínio, respectivamente:

$$2\Phi_1^+(X_{1/2}) = \Phi_1^{(i+1/2)}(X_{1/2}) + B_{1/2}^{(i+1/2)}\Phi_0^{(i+1/2)}(X_1) \tag{6.14}$$

e

$$2\Phi_1^+(X_{K+1/2}) = -\Phi_1^{(i+1/2)}(X_{K+1/2}) + B_{K+1/2}^{(i+1/2)}\Phi_0^{(i+1/2)}(X_K) \tag{6.15}$$

Todas as quantidades até este ponto são calculadas sobre os fluxos angular e escalar produzidos pela varredura [27]. As equações (6.10), (6.11), (6.14) e (6.15) podem ser modificadas para produzir as quantidades $\Phi_0^{(i+1)}(X_k)$ e $\Phi_1^{(i+1)}(X_{k+1/2})$, gerando o sistema de equações lineares

[1] As Leis de Fick, combinadas com as leis de conservação, descrevem o transporte de massa por difusão.

$$\begin{aligned}\Phi_1^{(i+1)}(X_{k+1/2}) - \Phi_1^{(i+1)}(X_{k-1/2}) + \Sigma_T^{(i+1/2)}\Phi_0^{(i+1)}(X_k)\Delta_k = \\ = \Sigma_S^{(i+1/2)}\Phi_0^{(i+1)}(X_k)\Delta_k + Q(X_k)\Delta_k\end{aligned} \tag{6.16}$$

$$\begin{aligned}\Phi_1^{(i+1)}(X_{k+1/2}) = -\frac{2}{3}\left(\frac{\Phi_0^{(i+1)}(X_{k+1}) - \Phi_0^{(i+1)}(X_k)}{\Sigma_T^{(i+1/2)}\Delta_{k+1} + \Sigma_T^{(i+1/2)}\Delta_k}\right) + \\ + \hat{D}_{k+1/2}^{(i+1/2)}\Big(\Phi_0^{(i+1)}(X_{k+1}) + \Phi_0^{(i+1)}(X_k)\Big)\end{aligned} \tag{6.17}$$

$$2\Phi_1^{+}(X_{1/2}) = \Phi_1^{(i+1)}(X_{1/2}) + B_{1/2}^{(i+1/2)}\Phi_0^{(i+1)}(X_1) \tag{6.18}$$

$$2\Phi_1^{+}(X_{K+1/2}) = -\Phi_1^{(i+1)}(X_{K+1/2}) + B_{K+1/2}^{(i+1/2)}\Phi_0^{(i+1)}(X_K) \tag{6.19}$$

Estas equações geram um sistema tridiagonal com K equações para o fluxo escalar $\Phi_0^{(i+1)}(X_k)$ em cada uma das K células grossas. O termo $\Phi_1^{(i+1)}(X_{k+1/2})$ pode ser eliminado na manipulação e a correção para o fluxo escalar nas células finas pode ser escrita como

$$\phi_0^{(i+1)}(x_j) = \phi_0^{(i+1/2)}(x_j)\left(\frac{\Phi_0^{(i+1)}(X_k)}{\Phi_0^{(i+1/2)}(X_k)}\right). \tag{6.20}$$

A quantidade $\phi_0^{(i+1)}$ é então utilizada como nova estimativa para a iteração seguinte [27, 28].

6.1 O Algoritmo & Paralelização

Podemos descrever o CMFD esquematicamente pelos passos abaixo.

1. Inicializar fluxos angulares médios em todas as células.

2. Varredura: para a direita e em seguida para a esquerda, obtendo fluxos angulares $\psi^{(i+1/2)}$ nas direções determinadas.

3. Calcular fluxo escalar $\phi_0^{(i+1/2)}$ utilizando a equação (6.2).

4. Calcular $\hat{D}_{k+1/2}^{(i+1/2)}$ utilizando a equação (6.11), $B_{1/2}^{(i+1/2)}$ utilizando a equação (6.14) e $B_{K+1/2}^{(i+1/2)}$ utilizando a equação (6.15).

5. Calcular $\Phi_0^{(i+1)}(X_k)$ resolvendo o sistema gerado pelas eqs. (6.16)até(6.19)

6. Calcular $\phi_0^{(i+1)}$ utilizando a equação (6.20)

7. Testar $\phi_0^{(i+1)}$ para convergência: se o erro relativo entre $\phi^{(i+1)}$ e $\phi^{(i)}$ for menor do que uma tolerância ϵ dada, ou quando o número máximo de iterações for atingido, o processo é interrompido. Senão, volta ao passo (2).

Assim como o DSA, o CMFD também consiste de um estágio de alta ordem, contemplado nos passos 2 e 3, e um estágio de baixa ordem, nos passos 4 a 6, sobre uma malha mais grossa do que a utilizada no estágio de alta ordem.

Novamente, como no DSA e no SI, a varredura é o trecho computacionalmente mais custoso do método. Para a paralelização deste método, existem duas estratégias já utilizadas neste trabalho: processar a varredura para a direita e para a esquerda independentemente; ou fornecer uma estimativa extra na célula central e dividir a varredura em quatro blocos.

A estratégia de processar as duas varreduras independentemente tem a vantagem de manter inalterado o número de iterações até a convergência, porém soa menos promissora (em termos de ganho de tempo) do que a estratégia de quatro blocos. Esta, por sua vez, não apresentou bons resultados para o DSA, o que indica que pode também não apresentar bons resultados para o CMFD.

6.2 Problemas teste iniciais

Para decidir se a estratégia de quatro blocos era admissível enquanto estratégia de paralelização, testes similares aos conduzidos no capítulo 4 foram reproduzidos para o CMFD. Os mesmos dois problemas teste foram executados para as versões sequencial e paralela do código e estão descritos a seguir.

O primeiro problema teste consiste de um domínio unidimensional homogêneo, de comprimento $L = 10$ cm, discretizado em 20000 células finas e 100 células grossas. A variável angular foi discretizada em 256 direções e o parâmetro σ_T foi mantido em 1 cm^{-1}. Novamente, o parâmetro σ_S foi variado em cada teste, assumindo os valores 0,2cm^{-1}, 0,4cm^{-1}, 0,8cm^{-1} e 0,995cm^{-1}. As condições de contorno foram mantidas em 1 nêutron/cm^2·s em ambas as extremidades do domínio, a tolerância fornecida para o processo iterativo foi $\epsilon = 10^{-10}$, o domínio não possui fonte e o número máximo de iterações foi mantido em 5000.

	$\sigma_S = 0{,}2$		$\sigma_S = 0{,}4$	
	Iterações	Tempo total (s)	Iterações	Tempo total (s)
Sequencial	10	2,917000	15	3,140000
Paralelo 4 blocos	13	2,531000	12	2,933000

	$\sigma_S = 0{,}8$		$\sigma_S = 0{,}995$	
	Iterações	Tempo total (s)	Iterações	Tempo total (s)
Sequencial	15	4,298000	17	5,047000
Paralelo 4 blocos	20	3,893000	22	4,625000

Tabela 6.1: Número de iterações e tempo de execução do CMFD para cada caso do problema homogêneo nas versões sequencial e paralela

É esperado que a estratégia de quatro blocos leve a um aumento no número de iterações em função da estimativa extra na célula central. Por outro lado, em função do que foi observado no DSA, também é esperado que este aumento torne a estratégia impraticável caso o método calcule correções sobre o fluxo escalar. Podemos observar pela tabela 6.1 que esta expectativa não se concretizou com o CMFD.

Mesmo com o aumento no número de iterações, a versão paralelizada com a estratégia de quatro blocos apresentou tempo de execução menor do que a versão sequencial, inclusive para valores de σ_S que tornavam o quociente σ_S/σ_T próximo de 1.

O segundo problema teste consiste de um domínio heterogêneo, de comprimento total $L = 25$ cm, composto de três regiões com parâmetros distintos. A primeira região, de comprimento $L_1 = 10$ cm e parâmetros $\sigma_S = 0{,}92\ \text{cm}^{-1}$ e $\sigma_T = 1\ \text{cm}^{-1}$, foi discretizada em 4000 células. A segunda região, de comprimento $L_2 = 5$ cm, foi discretizada em 2000 células, com parâmetros $\sigma_T = 1\ \text{cm}^{-1}$ e σ_S foi variado, assumindo os valores assumindo os valores $0{,}2\ \text{cm}^{-1}$, $0{,}4\ \text{cm}^{-1}$, $0{,}8\ \text{cm}^{-1}$ e $0{,}995\ \text{cm}^{-1}$. Esta região também possui uma fonte fixa $q(x) = 1$ nêutron/cm^3. A terceira região é idêntica à primeira. As condições de contorno foram mantidas em 1 nêutron/cm^2·s em ambas as extremidades do domínio. Ao longo de todo o domínio, a variável angular foi discretizada em 256 direções e a malha grossa é composta de 100 células. Os resultados estão expostos na tabela 6.2.

	$\sigma_S = 0{,}2$		$\sigma_S = 0{,}4$	
	Iterações	Tempo total (s)	Iterações	Tempo total (s)
Sequencial	16	2,585800	16	2,619000
Paralelo 4 blocos	17	1,890000	17	1,940000

	$\sigma_S = 0{,}8$		$\sigma_S = 0{,}995$	
	Iterações	Tempo total (s)	Iterações	Tempo total (s)
Sequencial	16	2,556000	17	2,784000
Paralelo 4 blocos	19	2,310000	21	2,355000

Tabela 6.2: Número de iterações e tempo de execução do CMFD para cada caso do problema heterogêneo nas versões sequencial e paralela

Pelas tabelas 6.1 e 6.2, podemos identificar que o comportamento da versão de quatro blocos do CMFD foi similar à versão de quatro blocos do SI. O número de iterações subiu, em função da estimativa inicial, porém não o suficiente para que o tempo total de execução ultrapassasse o da versão sequencial.

Por um lado, era esperado que o CMFD apresentasse um comportamento mais parecido com o apresentado pelo DSA, pois ambos utilizam aproximações de difusão para calcular correções sobre ϕ (correções estas que já sabemos carregar erros oriundos da estimativa inicial). Por outro, a aproximação de difusão utilizada pelos dois métodos não é a mesma e o CMFD calcula a correção sobre uma malha grossa, enquanto o DSA calcula sobre a mesma malha da varredura.

Podemos observar tais diferenças de comportamento entre o DSA e o CMFD medindo o erro relativo entre a solução produzida pelo código sequencial e a solução produzida pelo código paralelo (em sua versão com quatro blocos) em cada um dos métodos.

Por exemplo, no primeiro problema teste, no caso em que $\sigma_S = 0{,}8$, após três iterações do DSA o erro relativo entre as soluções era de 1,293456. Já para o CMFD, nestas mesmas condições, o erro relativo entre as soluções era de $3{,}721172 \times 10^{-2}$. Após 10 iterações, o DSA apresenta erro relativo entre as soluções de 0,377847, enquanto o CMFD apresenta erro relativo entre as soluções de $1{,}027249 \times 10^{-5}$.

Isto coloca em evidência as diferenças de comportamento entre os dois métodos quando paralelizados sob a estratégia de quatro blocos. Além disso, corrobora os dados exibidos nas tabelas 6.1 e 6.2 no que diz respeito ao número de iterações do CMFD e indica que a estratégia de quatro blocos pode ser utilizada para a paralelização deste método. Outros problemas teste serão executados no capítulo 8.

7 MÉTODO ANALÍTICO DE ORDENADAS DISCRETAS

Diferentemente dos métodos estudados nos capítulos 4, 5 e 6, o Método Analítico de Ordenadas Discretas (ADO) não é um método iterativo. Neste método a variável angular μ foi discretizada mas a variável espacial x não, gerando uma solução analítica na variável espacial, o que torna desnecessário o uso de varreduras. A varredura era o trecho computacionalmente mais custoso em todos os três métodos anteriores, então é relevante, em termos de tempo de execução, o estudo de métodos cujas formulações não envolvam a discretização da variável espacial.

Tendo em vista que todos os problemas tratados neste trabalho são de espalhamento isotrópico, este método foi formulado neste capítulo apenas para este caso. Primeiramente, consideramos a equação (4.1), e a escrevemos no formato abaixo, considerando a fonte $q(x)$ nula.

$$\mu\frac{\partial}{\partial x}\psi(x,\mu) + \psi(x,\mu) = \frac{c}{2}\int_0^1 [\psi(x,\mu') + \psi(x,-\mu')]d\mu'. \tag{7.1}$$

Aqui consideraremos que $\mu \in [-1,1]$ e $x \in (0,x_0)$. Além disso, ψ segue sendo o fluxo angular e c é o número médio de nêutrons secundários por colisão. Para este problema, com estas hipóteses, c é calculado como σ_S/σ_T. Temos também as seguintes condições de contorno:

$$\psi(0,\mu) = \psi_e(\mu), \mu \in (0,1] \tag{7.2}$$

e

$$\psi(x_0,-\mu) = \psi_d(\mu), \mu \in (0,1]. \tag{7.3}$$

Assim, para a equação (7.1), buscamos soluções exponenciais da forma [7, 5, 30]

$$\psi(x,\mu) = \phi(\nu,\mu)e^{-x/\nu}. \tag{7.4}$$

Podemos substituir ψ dada pela equação (7.4) na equação (7.1), e, como $e^{-x/\nu} \neq 0$ para todos os x, obtemos

$$(\nu - \mu)\phi(\nu,\mu) = \frac{c\nu}{2}\int_0^1 [\phi(\nu,\mu') + \phi(\nu,-\mu')]d\mu'. \tag{7.5}$$

Agora, a intenção é resolver a equação (7.5) para ϕ. Para isto, utilizamos um esquema de quadratura para a integral do lado direito. Isto nos permite reescrever a equação (7.5) como

$$(\nu - \mu)\phi(\nu,\mu) = \frac{c\nu}{2}\sum_{k=1}^{N} \omega_k[\phi(\nu,\mu_k) + \phi(\nu,-\mu_k)]. \tag{7.6}$$

Na equação (7.6), ω_k e μ_k são N pesos e nós respectivamente, definidos para integração no intervalo [0,1]. Nos métodos anteriores, o esquema de quadratura utilizado foi Gauss-Legendre para integração no intervalo $[-1,1]$, gerando nós e pesos y_k e v_k [11]. Para o ADO, y_k e v_k foram mapeados para o intervalo [0,1], gerando assim os nós e pesos μ_k e ω_k, utilizando as relações

$$2\mu_k = y_k + 1 \tag{7.7}$$

$$\omega_k = \frac{1}{2}v_k. \tag{7.8}$$

Agora, se avaliarmos a equação (7.6) nos valores $\mu = \pm\mu_i$, podemos reescrevê-la como

$$(\nu \mp \mu_i)\phi(\nu, \pm \mu_i) = \frac{c\nu}{2} \sum_{k=1}^{N} \omega_k [\phi(\nu,\mu_k) + \phi(\nu, -\mu_k)]. \tag{7.9}$$

O interesse agora é escrever a equação (7.9) em formato matricial. Para isto, definiremos

$$\mathbf{M} = \mathrm{diag}(\mu_1,\mu_2,\ldots,\mu_\mathrm{N}), \tag{7.10}$$

$$(\mathbf{W})_{i,j} = \frac{c}{2}\omega_j, \tag{7.11}$$

e

$$\Phi_{\pm}(\nu) = \ [\phi(\nu, \pm \mu_1) \quad \phi(\nu, \pm \mu_2) \quad \ldots \quad \phi(\nu, \pm \mu_N)]^T. \tag{7.12}$$

Além destas, $\mathbf{I}$ representa a matriz identidade $N \times N$. Desta forma, podemos reescrever a equação (7.9) como

$$\frac{1}{\nu}\mathbf{M}\ \Phi_+(\nu) = (\mathbf{I} - \mathbf{W})\Phi_+(\nu) - \mathbf{W}\ \Phi_-(\nu) \tag{7.13}$$

e

$$-\frac{1}{\nu}\mathbf{M}\ \Phi_-(\nu) = (\mathbf{I} - \mathbf{W})\Phi_-(\nu) - \mathbf{W}\ \Phi_+(\nu). \tag{7.14}$$

Definimos então os vetores [7, 5, 30]

$$\mathbf{U}(\nu) = \Phi_+(\nu) + \Phi_-(\nu) \tag{7.15}$$

e

$$\mathbf{V}(\nu) = \Phi_+(\nu) - \Phi_-(\nu). \tag{7.16}$$

Subtraindo a equação (7.14) da equação (7.13), obtemos

$$\frac{1}{\nu}\mathbf{M}\mathbf{U}(\nu) = \mathbf{V}(\nu). \tag{7.17}$$

Ao somarmos a equação (7.14) com a equação (7.13), obtemos

$$\frac{1}{\nu}\mathbf{MV}(\nu) = (\mathbf{I} - 2\mathbf{W})\mathbf{U}(\nu). \tag{7.18}$$

A equação (7.17) pode ser utilizada para eliminar $\mathbf{V}$ da equação (7.18), gerando

$$(\mathbf{D} - 2\mathbf{M}^{-1}\mathbf{W}\mathbf{M}^{-1})\mathbf{MU}(\nu) = \frac{1}{\nu^2}\mathbf{MU}(\nu), \tag{7.19}$$

na qual

$$\mathbf{D} = \mathrm{diag}(\mu_1^{-2},\mu_2^{-2},\ldots,\mu_{\mathrm{N}}^{-2}). \tag{7.20}$$

Definimos $\mathbf{T}$ uma matriz diagonal, com entradas $t_i = \sqrt{\omega_i}$, com $i = 1,\ldots,N$. Ao multiplicarmos a equação (7.19) por $\mathbf{T}$, obtemos

$$(\mathbf{D} - c\mathbf{z}\mathbf{z}^T)\mathbf{X}(\nu) = \lambda\mathbf{X}(\nu). \tag{7.21}$$

Na equação (7.21), $\mathbf{X}$, λ e z são dados por

$$\mathbf{X}(\nu) = \mathbf{TMU}(\nu), \tag{7.22}$$

$$\lambda = \frac{1}{\nu^2} \tag{7.23}$$

e

$$\mathbf{z} = \left[\tfrac{1}{\mu_1}\sqrt{\omega_1} \quad \tfrac{1}{\mu_2}\sqrt{\omega_2} \quad \ldots \quad \tfrac{1}{\mu_N}\sqrt{\omega_N}\right]^T. \tag{7.24}$$

Os autovalores λ podem ser computados por rotinas de pacotes numéricos já existentes. Deste modo, uma vez calculados os autovalores, a solução de ordenadas discretas pode ser escrita como

$$\psi(x, \pm\mu_i) = \sum_{j=1}^{N} \left[A_j \phi(\nu_j, \pm\mu_i) e^{-x/\nu_j} + B_j \phi(\nu_j, \mp\mu_i) e^{-(x_0-x)/\nu_j} \right], \tag{7.25}$$

na qual

$$\phi(\nu_j,\mu_i) = \frac{c\nu_j}{2(\nu_j - \mu_i)}. \tag{7.26}$$

Para facilitar o cálculo das exponenciais envolvidas e evitar potenciais erros numéricos devido a "overflow", foi imposta a condição de normalização

$$\sum_{k=1}^{N} \omega_k [\phi(\nu,\mu_k) + \phi(\nu, -\mu_k)] = 1. \tag{7.27}$$

Na equação (7.25) as constantes de separação ν_j são calculadas a partir dos autovalores correspondentes, utilizando a equação (7.21). Para computar as constantes A_j e B_j a partir das condições de contorno, dadas pelas equações (7.2) e (7.3), primeiramente é preciso escrever as condições em versões discretas, para $i = 1, \ldots, N$:

$$\psi(0,\mu_i) = \psi_e(\mu_i) \tag{7.28}$$

e

$$\psi(x_0, -\mu_i) = \psi_d(\mu_i). \tag{7.29}$$

Deste modo, é gerado um sistema de equações lineares que pode ser resolvido para encontrar as constantes A_j e B_j e a solução ψ está, portanto, determinada. De posse do fluxo angular, ainda podemos calcular o fluxo escalar e a corrente (aqui chamados de $\rho(x)$ e $j(x)$) usando as equações

$$\rho(x) = \sum_{j=1}^{N} \left[A_j e^{-x/\nu_j} + B_j e^{-(x_0-x)/\nu_j} \right] \tag{7.30}$$

e

$$j(x) = (1-c) \sum_{j=1}^{N} \left[A_j e^{-x/\nu_j} + B_j e^{-(x_0-x)/\nu_j} \right]. \tag{7.31}$$

Isto conclui a formulação para o ADO. Esta formulação, no entanto, não inclui o caso de regiões com fontes constantes, que são problemas que gostaríamos de conseguir tratar. Para isto, utilizamos que a equação (7.1) é linear e portanto a solução para o problema com uma fonte q constante consiste na solução apresentada na equação (7.25) acrescida de uma solução particular dada por

$$\psi^P(x, \pm\,\mu) = \left[\psi^P(x, \pm\,\mu_1) \quad \psi^P(x, \pm\,\mu_2) \quad \ldots \quad \psi^P(x, \pm\,\mu_N) \right]^T, \tag{7.32}$$

resultando em uma solução da forma

$$\psi(x, \pm\mu_i) = \sum_{j=1}^{N} \left[A_j \phi(\nu_j, \pm\mu_i) e^{-x/\nu_j} + B_j \phi(\nu_j, \mp\mu_i) e^{-(x_0-x)/\nu_j} \right] + \psi^P(x, \pm\mu). \tag{7.33}$$

Para o caso em que q é constante, uma solução particular conhecida é [6, 32]

$$\psi^P = \frac{q}{\sigma_T - c}. \tag{7.34}$$

o que conclui a formulação para o ADO com fonte constante em cada região, com a restrição de $\sigma_T \neq c$.

7.1 O Algoritmo & Paralelização

Diferentemente dos outros três métodos estudados neste trabalho, a implementação do ADO não é feita de maneira iterativa, mas sim de maneira direta. Todos os passos são executados apenas uma vez e, ao término do código estão computados os fluxos.

1. Determinar um esquema de quadratura. Neste caso, Gauss-Legendre mapeado para o intervalo [0,1].
2. Calcular os autovalores λ referentes à equação (7.21).
3. Calcular as constantes A_j e B_j resolvendo os sistemas lineares resultantes das equações (7.28) e (7.29).
4. Calcular ρ e j utilizando as equações (7.30) e (7.31).

Comparativamente, este é um algoritmo muito mais enxuto do que os apresentados nos capítulos anteriores. Em termos de tempo de execução, sua versão sequencial chega a ser mais rapidamente executada do que algumas das versões paralelas de outros métodos produzidas neste trabalho.

O código sequencial sofreu alterações antes de ser paralelizado. Por questões de legibilidade e organização do código sequencial, todas as etapas descritas acima, bem como etapas intermediárias, foram escritas como subrotinas ou funções. Deste modo, o programa principal era constituído basicamente de chamadas, o que de fato facilita para o leitor, que precisa compreender como o código está estruturado, e para o programador, que fica em posse de um código mais limpo e claro.

Esta não é, no entanto, a melhor forma de escrever um código quando se deseja paralelizá-lo, por dois motivos. O primeiro consiste no fato de que OpenMP não permite ramificações dentro de uma região paralela. Assim, se desejássemos

manter a estrutura de chamadas de subrotinas, seria necessário paralelizar cada uma das subrotinas. Isto implicaria a abertura de uma região paralela por subrotina que não contenha chamadas, e mais de uma região paralela para subrotinas que contenham chamadas para outras subrotinas.

A abertura e o encerramento de regiões paralelas tomam tempo, pois é preciso chamar um time de *threads* na abertura e sincronizá-las no encerramento [9]. No caso do ADO, entre subrotinas chamadas pelo programa principal e subrotinas chamadas por outras subrotinas, seria necessário um total de 12 regiões paralelas. Esta é uma quantidade desnecessariamente elevada de regiões paralelas.

O segundo motivo consiste no fato de que códigos com chamadas a subrotinas externas ao programa principal possui tempo de execução maior do que códigos sem tais chamadas. O tempo de chamar uma subrotina, o tempo de retornar os argumentos de saída da subrotina, o tempo (e a memória) que o compilador gasta armazenando variáveis desnecessariamente e todas as otimizações que o compilador poderia detectar e executar são perdidos com este processo.

O objetivo deste trabalho é, em última instância, diminuir os tempos de execução. Não faz sentido, neste contexto, manter esta estrutura. Todas as subrotinas que estavam sendo chamadas, inclusive as que eram chamadas por outras subrotinas, foram migradas diretamente no código, de forma manual, um processo conhecido em otimização de programas como *inlining*. Todas as medições de tempo de execução do código sequencial relatadas neste trabalho são referentes à versão com *inlining* das subrotinas, e não à versão com chamadas.

Ainda que tenham sido feitos todos os *inlining* possíveis, algumas rotinas permaneceram sendo chamadas. O cálculo dos autovalores, na equação (7.21), foi executado pela rotina DGEEVX, e a resolução do sistema linear, nas equações

(7.28) e (7.29), foi executada pelas rotinas DGETRF e DGETRS[1]. Outras rotinas poderiam ter sido utilizadas nestes pontos, uma vez que existem diversos algoritmos para cálculo de autovalores e resolução de sistemas lineares [22].

As três rotinas mencionadas acima fazem parte do LAPACK [3], pacote de rotinas de álgebra linear desenvolvido para Fortran. Não paralelizar ou interferir de qualquer modo nas rotinas do LAPACK foi uma escolha feita no início deste trabalho em função dos prazos envolvidos.

Devido à existência destas três chamadas, a paralelização do ADO foi feita em três regiões paralelas. A primeira engloba o mapeamento dos nós e pesos para o intervalo [0,1] e o cálculo das entradas das matrizes necessárias para o problema de autovalor. Nesta região foram utilizadas as diretivas `!$OMP DO` e `!$OMP WORKSHARE`.

A segunda calcula as quantidades $\phi(\nu_j,\mu_i)$, bem como as entradas da matriz e do vetor referentes ao sistema linear. As entradas da matriz referente ao sistema linear podem ser calculadas independentemente, então foi utilizada majoritariamente a diretiva `!$OMP SECTIONS`.

A terceira calcula as exponenciais presentes na solução de ordenadas discretas e as quantidades ρ e j. Nesta região, tanto ρ quanto j foram expressas como reduções, de modo que a diretiva `!$OMP DO` pôde ser utilizada junto da cláusula `REDUCTION`.

Os resultados obtidos com essa versão paralela do ADO são mostrados no capítulo 8.

[1]DGEEVX computa autovalores e autovetores para matrizes não simétricas, enquanto DGETRF computa a fatoração LU de uma matriz utilizando pivotamento parcial e troca de linhas e DGETRS resolve sistemas lineares utilizando tal fatoração.

8 PROBLEMAS TESTE

Neste capítulo, foram executados problemas teste a fim de avaliar como foi o comportamento dos métodos numéricos descritos nos capítulos anteriores em suas versões paralelizadas quando comparados com suas versões sequenciais.

Inicialmente, é preciso fazer um panorama de quais versões estão sendo avaliadas. Para o SI, foram avaliadas quatro versões: sequencial, paralela na estratégia de dois blocos, paralela na estratégia de quatro blocos e paralela na estratégia de oito blocos. A versão sequencial e a versão com dois blocos iteram o mesmo número de vezes até a convergência, pois não possuem divisões no domínio.

A versão com quatro blocos possui o domínio dividido igualmente em duas partes, e como cada varredura ocorre em dois sentidos, são gerados quatro blocos. A versão de oito blocos é gerada do mesmo modo que a versão de quatro blocos, apenas dividindo o domínio em quatro partes iguais ao invés de duas, gerando assim oito blocos de varredura.

Já o DSA, como não respondeu bem à estratégia de divisão do domínio, possui apenas duas versões: sequencial e paralela na estratégia de dois blocos. O CMFD, apesar de ter respondido bem à estratégia de divisão do domínio, possui apenas três versões: sequencial, paralela na estratégia de dois blocos e paralela na estratégia de quatro blocos.

Por fim, para o ADO, que não possui varredura e não itera, foi escrita apenas uma versão paralela.

Para os métodos iterativos, os problemas teste com até 1024 direções foram executados no mínimo 5 vezes e o tempo de execução, medido com a rotina `date_and_time`, é uma média dos tempos obtidos em cada caso. Problemas com

2048 direções ou mais foram executados no mínimo 3 vezes. Para o ADO, todos os casos foram executados no mínimo 5 vezes.

8.1 Problema teste 1

O problema teste 1 é similar ao problema homogêneo descrito no capítulo 4. Ele consiste de um domínio unidimensional homogêneo, de comprimento $L = 10\text{cm}$, com as seções σ_T mantida em 1cm^{-1} e σ_S mantida em $0{,}8\text{cm}^{-1}$. As condições de contorno foram mantidas em 1 nêutron/cm^2·s em ambas as extremidades do domínio; a tolerância fornecida para o processo iterativo foi $\epsilon = 10^{-10}$; o domínio não possui fonte e o número máximo de iterações foi mantido em 5000.

A intenção deste problema é mapear a partir de qual tamanho de problema a versão paralela passa a valer a pena, ou seja, em quantas células e em quantas direções as variáveis espacial e angular precisam ser discretizadas para que a versão paralela do código seja executada em menos tempo do que a versão sequencial.

Para isto, o número de células foi variado, assumindo valores entre 5000 e 150000 com incrementos de 5000. Para os métodos iterativos, o número de direções foi variado entre 64 e 4096, sempre assumindo potências de 2. Para o ADO, o número de direções foi variado entre 64 e 2672, com incrementos de 16. Para todos os gráficos apresentados sobre os métodos iterativos foi fixado um número de direções, variado o número de células e medido o tempo de execução.

8.1.1 DSA

Para o DSA, com a variável angular discretizada em 128 direções, foi gerado o gráfico abaixo.

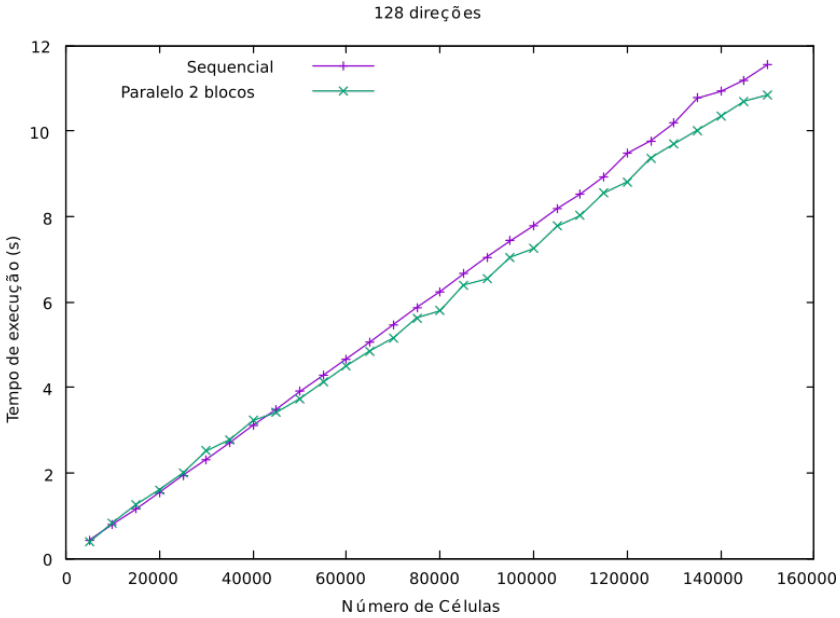


Figura 8.1: Tempo de execução do DSA, em segundos, para diversas quantidades de células e número de direções fixo em 128.

Podemos perceber, pela imagem 8.1, que a versão paralela do DSA passa a ser mais rapidamente executada do que a versão sequencial em 128 direções, entre 40000 e 45000 células, como pode ser observado na tabela.

	Tempo de execução (s)	
Número de células	Sequencial	Paralelo
40000	3,118	3,228
45000	3,495	3,420
145000	11,194	10,699
150000	11,553	10,843

Tabela 8.1: Tempo de execução das versões sequencial e paralela do DSA com 128 direções para o problema 1

Ainda, podemos perceber que, para 45000 células, a versão paralela é melhor (no sentido de executada em menor tempo) por apenas 0,07 segundos. Esta

diferença aumenta para 0,71 segundos em 150000 células, o que representa uma redução de 6,1% no tempo do código sequencial. Para problemas com mais direções, no entanto, esta redução pode ser mais drástica, como mostrado a seguir.

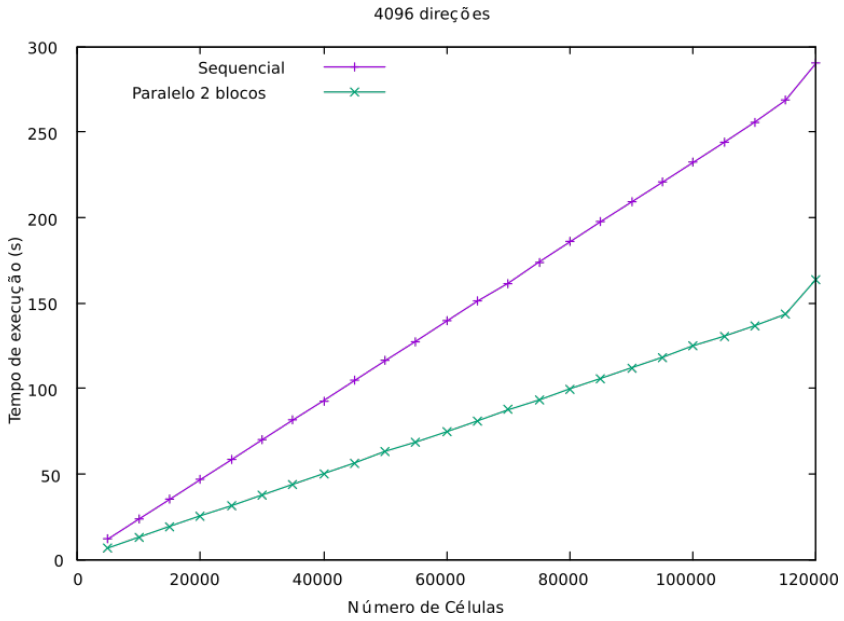


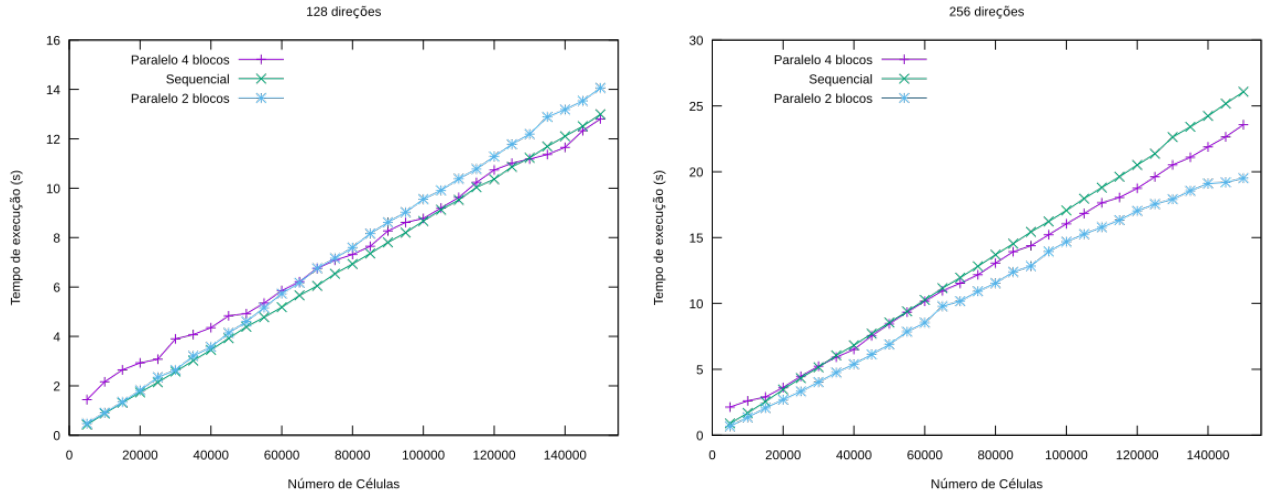
Figura 8.2: Tempo de execução do DSA, em segundos, para diversas quantidades de células e número de direções fixo em 4096.

Pela figura 8.2 fica claro que, conforme a dimensão do problema aumenta, a versão paralela começa a apresentar ganhos mais significativos. Para 120000 células, a versão sequencial levou 290,397 segundos, enquanto a versão paralela levou 164,022 segundos. Isto representa uma redução de 43,5%.

Para estes valores de σ_S e σ_T, as versões sequencial e paralela do DSA convergiram em 14 iterações.

8.1.2 CMFD

Para o CMFD, 128 e 256 direções são momentos de transição sobre qual versão do código possui menor tempo de execução, como mostram os gráficos a seguir.



(a) 128 direções

(b) 256 direções

Figura 8.3: Tempo de execução do CMFD, em segundos, para diversas quantidades de células e número de direções fixo em 256.

Para 128 direções, podemos perceber que a versão paralela na estratégia de dois blocos não é melhor do que a versão sequencial, mas a versão paralela na estratégia de quatro blocos apresenta ganhos a partir de 130000 células.

Já usando 256 direções, a versão de quatro blocos é melhor do que a versão sequencial apenas a partir de 65000 células, enquanto a versão paralela de dois blocos é melhor que ambas para todas as quantidades de células. Em 150000 células, a versão de quatro blocos leva 23,573 segundos e a versão de dois blocos leva 19,506 segundos. Isto representa uma redução de 9,5% para a versão de quatro

blocos e 25,2% para a versão de dois blocos, ambas em relação à versão sequencial, que leva 26,071 segundos.

Também percebemos que, para um número alto de direções, o ganho é mais significativo, como mostra o gráfico a seguir.

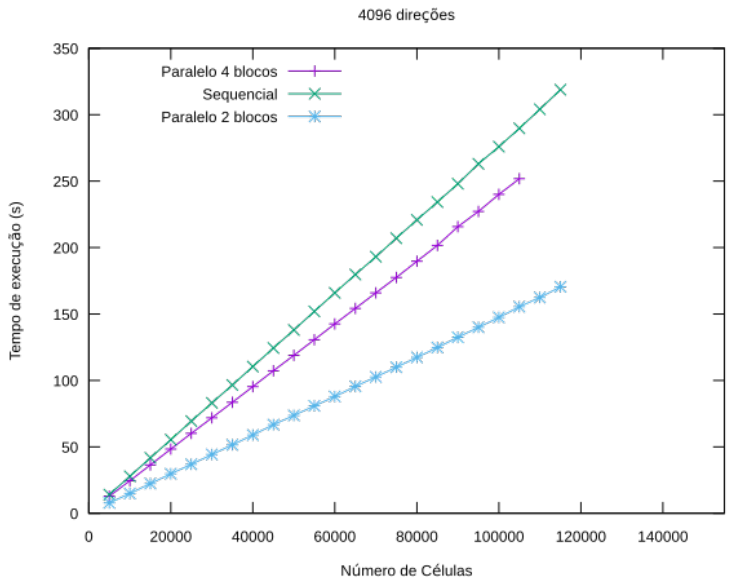


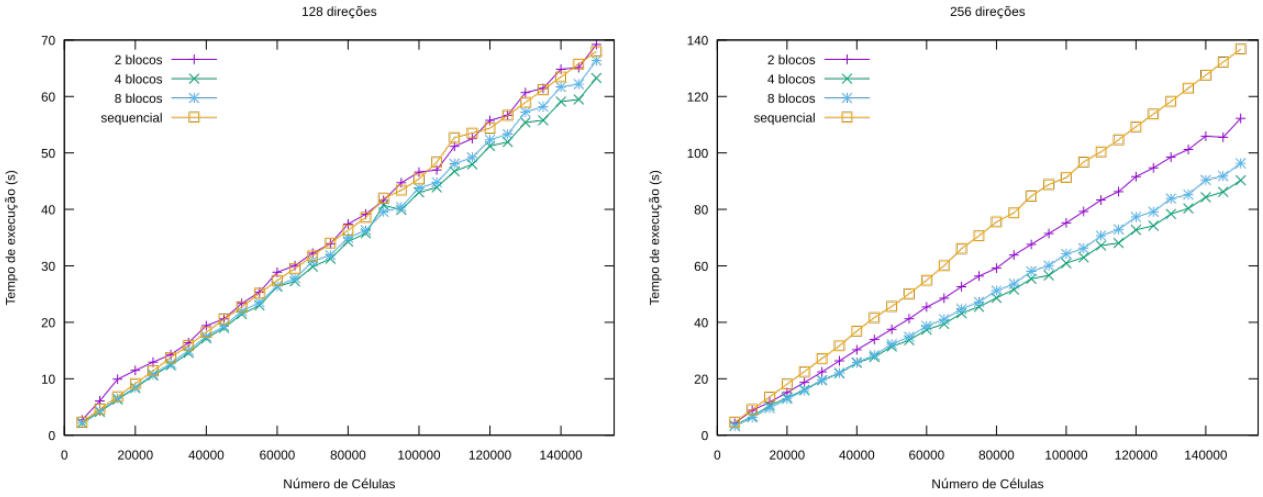
Figura 8.4: Tempo de execução do CMFD, em segundos, para diversas quantidades de células e número de direções fixo em 4096.

Em 105000 células, a versão sequencial leva 289,763 segundos. A versão de quatro blocos leva 251,865 segundos, o que representa um ganho de 25,5%, e a versão de dois blocos leva 155,455 segundos, representando um ganho de 46,3%.

Para estes valores de σ_S e σ_T, as versões sequencial e de dois blocos do CMFD convergiram em 15 iterações, enquanto a versão de quatro blocos convergiu em 20 iterações. É relevante notar aqui que, apesar das 5 iterações a mais que a versão de quatro blocos necessitou para convergência, o ganho de tempo para este problema foi o maior até o momento.

8.1.3 SI

O SI, assim como o CMFD, também possui um momento de transição entre 128 e 256 direções, como ilustrado a seguir.



(a) 128 direções

(b) 256 direções

Figura 8.5: Tempo de execução do SI para diversas quantidades de células e número de direções fixo em 256.

Para 128 direções, as versões com quatro e oito blocos apresentam tempo de execução menor do que a versão sequencial, mas o desempenho da versão de dois blocos varia com a quantidade de células. Para 256 direções, no entanto, é mais claro que as versões paralelas são todas executadas em menos tempo do que a versão sequencial. Ainda, dentre as versões paralelas, é possível identificar que a versão com quatro blocos apresenta o menor tempo de execução.

	Tempo de execução (s)			
Número de células	Sequencial	2 blocos	4 blocos	8blocos
150000	136,759	112,267	90,319	96,309

Tabela 8.2: Tempo de execução das versões sequencial e paralelas do SI com 256 direções para o problema 1

Os valores apresentados na tabela 8.2 nos permitem dizer que as versões de dois, quatro e oito blocos apresentaram ganhos de 17,9%, 33,9% e 29,5% respectivamente. Podemos avaliar também o desempenho das versões sequencial e paralelas do SI para quantidades maiores de direções:

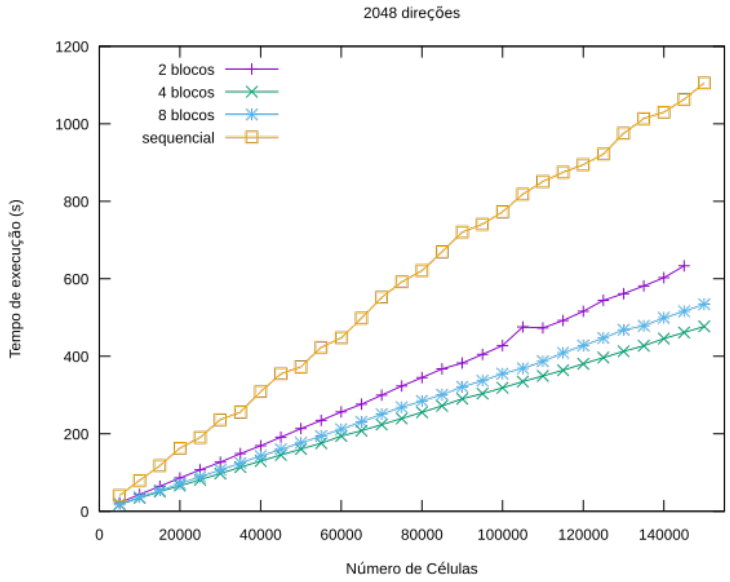


Figura 8.6: Tempo de execução do SI para diversas quantidades de células e número de direções fixo em 2048.

O gráfico gerado para 2048 direções possui estrutura similar ao gráfico gerado para 256 direções. De fato, neste caso, para este problema, todas as versões paralelas apresentam tempo de execução menor do que a versão sequencial, sendo a versão com quatro blocos a melhor delas.

	Tempo de execução (s)			
Número de células	Sequencial	2 blocos	4 blocos	8blocos
145000	1063,085	633,476	461,216	516,056

Tabela 8.3: Tempo de execução das versões sequencial e paralelas do SI com 4096 direções para o problema 1

As versões de dois, quatro e oito blocos apresentaram ganhos de 40,4%, 56,6% e 51,5% respectivamente. Para estes valores de σ_S e σ_T, as versões sequencial e de dois blocos do SI convergiram em 94 iterações, enquanto a versão de quatro blocos convergiu em 104 iterações e a versão com oito blocos convergiu em 110 iterações.

8.1.4 ADO

Para o ADO, os testes foram executados de maneira levemente diferente. Como a variável espacial não foi discretizada, só foi necessário variar o número de direções, entre 64 e 2672 com incrementos de 16. Os gráficos abaixo mostram os tempos de execução das versões paralela e sequencial.

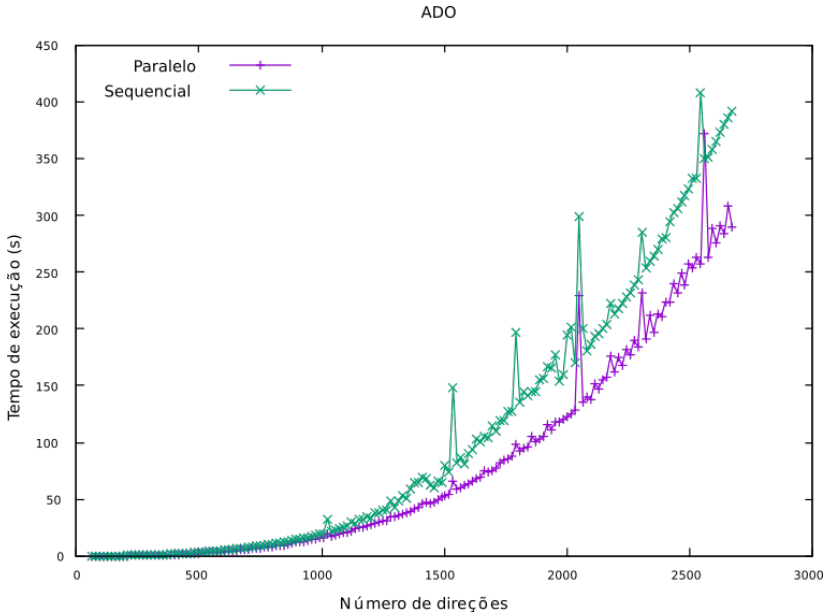
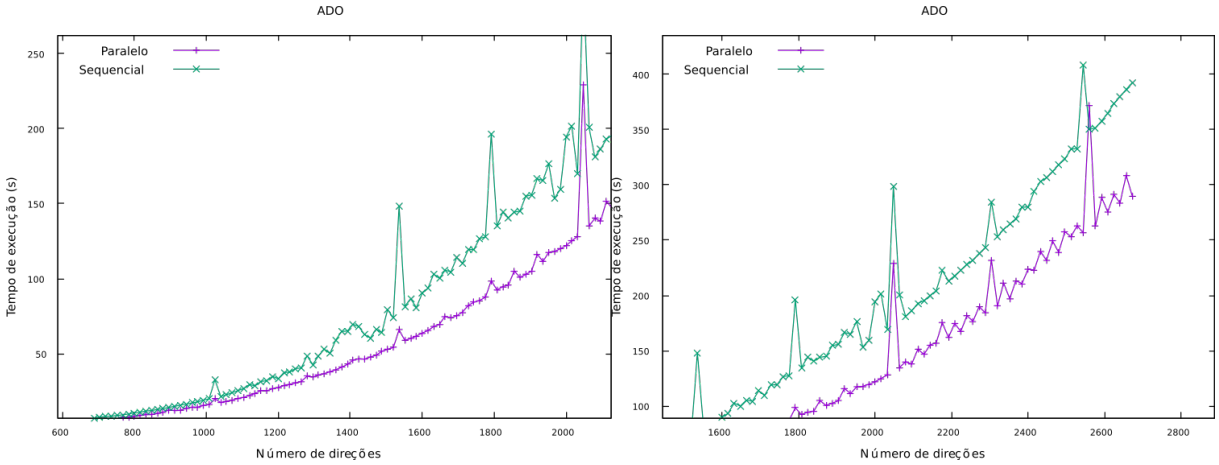


Figura 8.7: Tempo de execução do ADO para diversas quantidades de direções.

Podemos perceber, pela figura 8.7, que o ADO não apresentou curvas suaves de crescimento do tempo de execução conforme aumentava o número de direções no qual a variável angular foi discretizada. De fato, diversos picos podem ser verificados, tanto na versão sequencial quanto na versão paralela. Ainda, tais picos ocorrem quase nos mesmos tamanhos de problema em ambas as versões.

	Pico 1	Pico 2	Pico 3	Pico 4	Pico 5	Pico 6	Pico 7
Sequencial	1024	1280	1536	1792	2048	2304	2544
Paralelo	1024	1280	1536	1792	2048	2304	2560

Tabela 8.4: Tamanhos de problema para os quais ocorrem picos nas versões parela e sequencial do ADO



(a) 128 direções

(b) 256 direções

Figura 8.8: Seções da figura 8.7

Este comportamento pode ser interpretado como um fenômeno de *cache-miss*, que é quando o processador tenta buscar um dado na memória *cache* e não o encontra, fazendo necessária a busca pelo dado na memória principal. Cache-misses depreciam o desempenho, visto que a busca pelo dado na memória principal é muito mais lenta do que a busca na memória cache.

Existem diversos tipos de cache-misses, visto que uma busca por um dado na memória cache pode não ser bem sucedida por diversos motivos. O primeiro deles seria porque não há dados na memória cache, pois algum bloco de dados precisa ser o primeiro a ser transferido da memória principal para a memória cache. O segundo seria porque a capacidade máxima da memória cache já foi atingida. O terceiro, que é uma possível interpretação da situação relatada, ocorre quando dois endereços da memória principal são mapeados sobre o mesmo endereço na memória cache, criando um conflito.

Os compiladores possuem maneiras de estimar quantos cache-misses por esgotamento da capacidade da memória cache ocorrerão durante a execução de um programa, considerando variáveis declaradas que não foram alocadas dinami-

camente, mas cache-misses por conflitos de posições na memória são muito mais difíceis de serem previstos [36]. Consideramos, por exemplo, dois endereços z e y na memória principal mapeados sobre a mesma localização na memória cache, sendo que cada um deles será acessado três vezes. A ordem com que estes acessos acontecem determina a ocorrência ou não de cache-misses por conflito, pois acessar z três vezes e em seguida y três vezes implica em nenhum cache-miss por conflito, mas acessos do tipo $yzyzyz$ implica em cinco cache misses por conflito.

Para o problema teste 1, vemos que os valores relatados na tabela 8.4 são todos múltiplos de 64, o que pode ser indicativo de um possível cache-miss por conflito. Este efeito ainda pode ser amplificado quando se trabalha com várias *threads* porque não se sabe qual delas faz seus acessos antes das outras. Além disso, não são todos os níveis hierárquicos da memória cache que são de acesso comum a todas as *threads*, o que também pode ter influenciado na ocorrência dos picos relatados acima.

Apesar dos picos aparentes nos gráficos, é possível perceber que, de maneira geral, o tempo de execução da versão paralela é menor do que o tempo de execução da versão sequencial. De fato, para problemas com poucas direções (80 direções e acima) a versão paralela é executada em menos tempo.

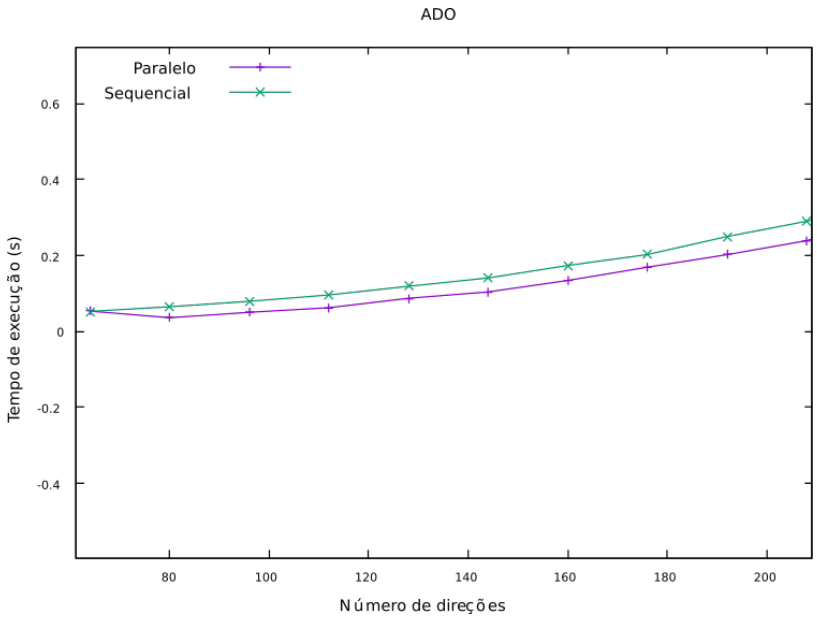


Figura 8.9: Tempo de execução do ADO para quantidades pequenas de direções.

Para 2672 direções, a versão sequencial leva 392,035 segundos, enquanto a versão paralela leva 289,824 segundos, representando um ganho de 26% no tempo de execução.

8.1.5 *Speedups*

Conforme descrito no capítulo 3, os ganhos de tempo das versões paralelas descritos neste trabalho são relativos às suas versões sequenciais. É possível calcular, no entanto, qual foi o *speedup* para alguns dos casos do problema teste 1, calculados pela equação (3.1)

	4096 direções		2048 direções	2672 direções
	120000 células	105000 células	145000 células	
DSA	1,770			
CMFD 2 blocos		1,864		
CMFD 4 blocos		1,150		
SI 2 blocos			1,678	
SI 4 blocos			2,305	
SI 8 blocos			2,060	
ADO				1,352

Tabela 8.5: *Speedups* das versões paralelas para o problema 1

Aqui vemos que, para este problema, a maioria dos*speedups* não atingiu 2. É relevante lembrar que, pela equação (3.2), que relata a Lei de Amdahl, o *speedup* esperado não pode ser maior do que o número de processadores. Apesar de ser uma limitação teórica, é possível que, na prática, ocorram *speedups* acima do que a Lei de Amdahl prevê. Este efeito é chamado de *speedup superlinear*, ocorre raramente, e quando ocorre pode ser em função do uso correto da memória cache, ou em função de outros fatores, como por exemplo as mudanças que o paralelismo gera nos algoritmos [33, 23]. Para encontrar um *speedup* superlinear, seria necessário que algum dos valores na tabela fosse maior do que ou igual a 8, o que não foi o caso.

8.2 Problema teste 2

Uma vez estabelecido como as estratégias de paralelização desempenharam em um problema como o problema 1, foi decidido testá-las para um problema similar, mas para o qual seja necessário mais iterações até a convergência, e desta vez para quantidades definidas de células e de direções.

O segundo problema teste consiste de um domínio unidimensional homogêneo, de comprimento $L = 10\,\text{cm}$, discretizado em 20000 células. A variável angular foi discretizada em 256 direções e os parâmetros σ_T e σ_S foram mantidos em $1\,\text{cm}^{-1}$ e em $0{,}995\,\text{cm}^{-1}$. As condições de contorno foram mantidas em 1 nêutron/cm^2·s em ambas as extremidades do domínio; a tolerância fornecida para o processo iterativo foi $\epsilon = 10^{-10}$; o domínio não possui fonte e o número máximo de iterações foi mantido em 5000.

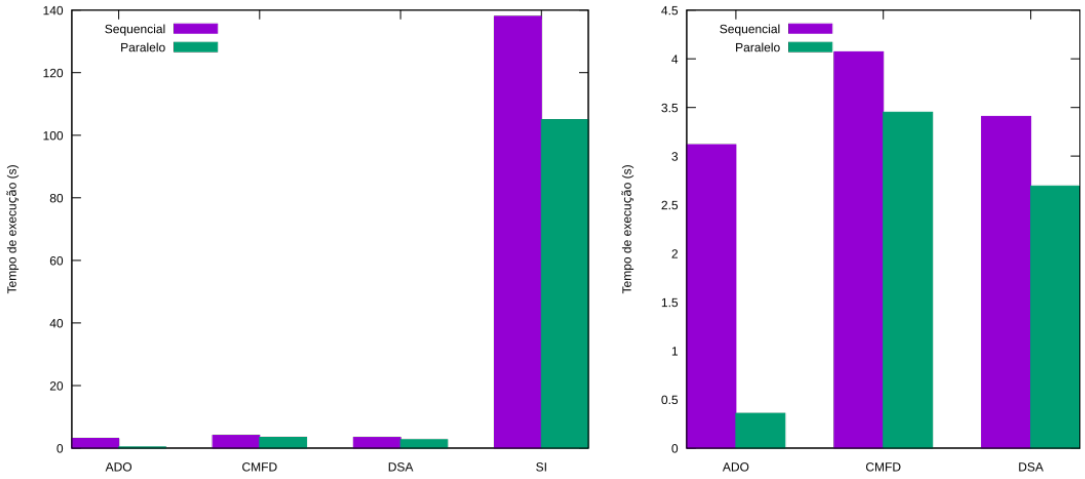
Para este problema teste, foram executadas apenas as versões que melhor desempenharam no problema 1. Para o ADO e para o DSA foram executadas suas versões sequenciais e paralelas, já que ambos possuem apenas uma de cada, para o CMFD foi executada a versão sequencial e a versão com dois blocos, e para o SI foram executadas as versões sequencial e com quatro blocos. Os tempos de execução estão tabelados a seguir.

	ADO	CMFD	
	Tempo (s)	Tempo (s)	Iterações
Sequencial	3,117000	4,071000	17
Paralelo	0,357333	3,451000	17

	DSA		SI	
	Tempo (s)	Iterações	Tempo (s)	Iterações
Sequencial	3,408000	14	138,436	669
Paralelo	2,693000	14	105,244	724

Tabela 8.6: Tempos e número de iterações para o problema 3

Podemos verificar visualmente os resultados nos gráficos mostrados a seguir



(a) Todos os quatro métodos

(b) Apenas ADO, CMFD e DSA

Figura 8.10: Tempo de execução para o problema 2.

Para este problema, o SI apresentou uma redução de 23,9%, enquanto o DSA e o CMFD apresentaram reduções de 20,9% e 15,2%. Já o ADO apresentou 88,5% de redução no seu tempo de execução. É relevante ressaltar que todos os ganhos de tempos mencionados aqui têm sido sempre relativos às versões sequenciais do mesmo método.

Se tomarmos a versão sequencial que executou este problema mais rapidamente, ou seja, ADO sequencial, perceberemos que o único método iterativo cuja versão paralela apresenta tempo de execução menor é o DSA, com ganho de 13,6% sobre a versão sequencial do ADO.

Esta comparação, no entanto, não está no cerne deste trabalho. O objetivo era fazer com que as versões paralelas dos códigos fossem executadas em menor tempo do que suas versões sequenciais, e não que métodos iterativos pudessem ser comparados a métodos semi-analíticos. De todo modo, este ganho de 13,6% do DSA sobre o ADO está devidamente mencionado e documentado.

8.3 Problema teste 3

Outro teste relevante é verificar como as versões paralelas desempenham em meios heterogêneos. Os testes executados para este problema são similares aos executados na seção 8.2. As quantidades de direções e células estavam fixas e apenas algumas versões foram executadas.

Para o ADO e para o DSA foram executadas suas versões sequenciais e paralelas, já que ambos possuem apenas uma de cada, para o CMFD foi executada a versão sequencial e a versão com dois blocos, e para o SI foram executadas as versões sequencial e com quatro blocos.

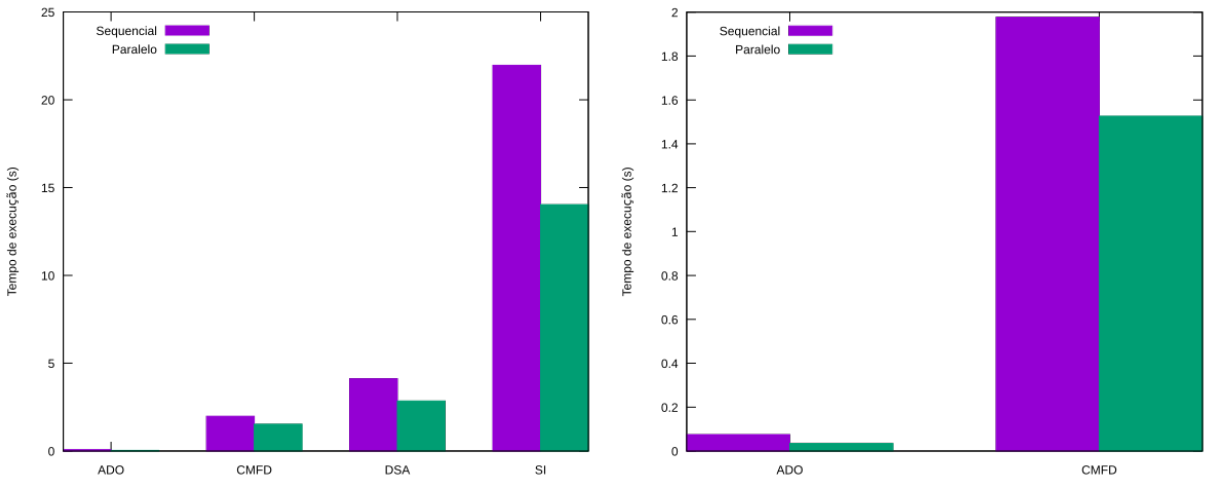
O terceiro problema teste consiste de um domínio heterogêneo, de comprimento total $L = 25\text{cm}$, composto de três regiões com parâmetros distintos. A primeira região, de comprimento $L = 10\text{cm}$ e parâmetros $\sigma_S = 0{,}92\text{cm}^{-1}$ e $\sigma_T = 1\text{cm}^{-1}$, foi discretizada em 4000 células. A segunda região, de comprimento $L = 5\,\text{cm}$, foi discretizada em 2000 células, com parâmetros $\sigma_T = 1\,\text{cm}^{-1}$ e $\sigma_S = 0{,}995\,\text{cm}^{-1}$. Esta região também possui uma fonte fixa $q(x) = 1$ nêutron/cm^3. A terceira região é idêntica à primeira. As condições de contorno foram mantidas em 1 nêutron/cm^2·s em ambas as extremidades do domínio A variável angular foi discretizada em 256 direções ao longo de todo o domínio. A tolerância fornecida para o processo iterativo foi $\epsilon = 10^{-10}$ e o número máximo de iterações foi mantido em 5000.

	ADO	CMFD	
	Tempo (s)	Tempo (s)	Iterações
Sequencial	$7{,}5\times10^{-2}$	1,977	16
Paralelo	$3{,}46\times10^{-2}$	1,526	16

	DSA		SI	
	Tempo (s)	Iterações	Tempo (s)	Iterações
Sequencial	4,117	14	21,966	216
Paralelo	2,851	14	14,031	216

Tabela 8.7: Tempos e número de iterações para o problema 3

Os resultados podem ser visualmente verificados.



(a) Todos os quatro métodos

(b) Apenas ADO e CMFD

Figura 8.11: Tempo de execução para o problema 3.

Para este problema o ADO apresentou um ganho de 53,8%, enquanto o CMFD e o DSA apresentaram ganho de 22,8% e de 30,7%. O SI, por sua vez, apresentou ganho de 36,1%.

De maneira geral, os problemas teste 2 e 3 serviram para verificar o que foi inicialmente sugerido pela seção 8.1: que as versões paralelas de todos os métodos apresentam ganho de tempo a partir de algum tamanho de problema (no sentido de quantidade de células e de direções). Claramente os parâmetros do problema não são negligenciáveis, visto que os métodos apresentaram ganhos e comportamentos muito diferentes, entre os problemas 1, 2 e 3.

O que é relevante ressaltar, neste ponto, é o fato de que os únicos casos em que as versões paralelas não apresentaram tempo de execução menor do que as versões sequenciais, foi quando as variáveis angular e espacial eram discretizadas em poucas direções (64 ou 128) e/ou em poucas células respectivamente. Além disso, a solução produzida pelas versões paralelas concorda em no mínimo 12 dígitos com a solução produzida pelas versões sequenciais, com gráficos no apêndice B

9 CONCLUSÕES

No que diz respeito aos objetivos deste trabalho, pode-se dizer que este cumpriu tudo aquilo a que se propôs. Não apenas todas as versões paralelas dos métodos apresentaram ganho de tempo em relação às suas versões sequenciais a partir de algum tamanho de problema, como várias apresentaram ganhos superiores a 50%, o que indica um *speedup* maior que 2.

Além disso, o tamanho mínimo de problema para o qual a versão paralela é executada em menor tempo não é impraticável. O DSA, por exemplo, no problema 1 em 128 direções não ultrapassa 12 segundos de execução. Isto indica que esta paralelização não apenas apresentou bons resultados para problemas grandes (mais de 4096 direções, mais de 100000 células), como também apresentou ganhos para casos limítrofes das duas versões.

Este trabalho ainda possui, no entanto, diversos outros aspectos a serem explorados. As rotinas do LAPACK, por exemplo, que aqui não foram alteradas, podem vir a ser tratadas no futuro, assim como os acessos à memória cache podem ser otimizados, e outras modificações no código sequencial podem ser feitas.

Outros tipos de paralelização também podem ser explorados, como por exemplo *Message-Passing Interface* (MPI), utilizado para paralelizar programas em máquinas com vários processadores, que demandam comunicação interprocessador. Esta seria uma abordagem muito diferente da que foi adotada neste trabalho, uma vez que OpenMP e MPI possuem propostas e sintaxes muito diferentes.

Ainda, é possível estudar quais seriam os efeitos da mesma abordagem que foi utilizada neste trabalho, com as mesmas estratégias de paralelização, em problemas formulados para duas ou três dimensões espaciais, ou em geometrias especiais, como cilíndrica ou esférica. Estes são todos possíveis trabalhos futuros

e que, dados os resultados favoráveis apresentados neste trabalho, possuem bons prognósticos.

Referências Bibliográficas

[1] Adams, M. L., and W., L. E. Fast iterative methods for discrete-ordinates particle transport calculations. *Progress in Nuclear Energy 40*, 1 (2002), 3–159.

[2] Adams, M. P., Adams, M. L., Hawkins, W. D., Smith, T., Rauchwerger, L., Amato, N. M., Bailey, T. S., and Falgout, R. D. Provably optimal parallel transport sweeps on regular grids. *Proceedings of the international conference on mathematics and computational methods applied to nuclear science and engineering 1* (May 5–9, 2013), 180–189.

[3] Anderson, E., Bai, Z., Bischof, C., Blackford, L. S., Demmel, J., Dongarra, J., du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. *LAPACK User's Guide*. Society for Industrial and Applied Mathematics, 1999.

[4] Ashby, S. F., Brown, P. N., Dorr, M. R., and C., H. A. A linear algebraic analysis of diffusion synthetic acceleration for the boltzmann transport equation. *Society for Industrial and Applied Mathematics Journal on Numerical Analysis 32*, 1 (1995), 128–178.

[5] Barichello, L. B. *Thermal Measurements and Inverse Techniques*. In [30], 2011, ch. Explicit Formulations Radiative Transfer Problems.

[6] Barichello, L. B., Garcia, R. D. M., and Siewert, C. E. Particular solutions for the discrete-ordinates method. *Journal of Quantitative Spectroscopy & Radiative Transfer 64*, 3 (2000), 219–226.

[7] Barichello, L. B., and Siewert, C. E. A new version of the discrete-ordinates method. *Proceedings of the 2nd International Conference on Computational Heat and Mass Transfer* (2001).

[8] Bell, G., and Glasstone, S. *Nuclear reactor theory.* Robert E. Krieger Publishing Company, New York, 1979.

[9] Board, A. R. *OpenMP Application Programming Interface.* OpenMP ARB, 2015.

[10] Boltzmann, L. Weitere studien über das wärmegleichgewicht unter gas molekülen. *Sitzungsberichte Akademie der Wissenschaften, 60:275376* (1872).

[11] Borche, A. *Métodos Numéricos.* UFRGS Editora, 2008.

[12] Cacuci, D. *Handbook of Nuclear Engineering: Vol. 1: Nuclear Engineering Fundamentals.* Handbook of Nuclear Engineering. Springer, 2010.

[13] Case, K., and Zweifel, P. *Linear Transport Theory.* Addison-Wesley Publishing Company, 1967.

[14] Chandrasekhar, S. *Radiative Transfer.* Oxford University Press, 1950.

[15] Chapman, B., Jost, G., and van der Pas, R. *Using OpenMP.* The Scientific and Engineering Computation. The MIT Press, 2008.

[16] Colomer, G., Borrell, R., Trias, F. X., and Rodríguez, I. Provably optimal parallel transport sweeps on regular grids. *Journal of Computational Physics 232* (2013), 118–135.

[17] De Rose, C. A. F., and Navaux, P. O. A. *Arquiteturas Paralelas.* Livros Didáticos. Bookman, Porto Alegre, 2008.

[18] Dorr, M. R., and Still, C. H. Concurrent source iteration in the solution of three-dimensional, multigroup discrete ordinates neutron transport equations. *Nuclear Science and Engineering 122* (1996), 287–308.

[19] Duderstadt, J., and Hamilton, L. *Nuclear Reactor Analysis.* Wiley, 1976.

[20] Garcia, R. *Métodos para solução da equação de transporte de partículas íntegro-diferencial.* Escola de Verão em Teoria de Transporte de Partículas Neutras, PUC - Porto Alegre - RS, 2002.

[21] gfortran team, T. *Using GNU Fortran.* Free Software Foundation, Boston, US, 2014.

[22] Golub, G., and Van Loan, C. *Matrix Computations*, 4th ed. John Hopkins University Press, Baltimore, USA, 2013.

[23] Gustafson, J. L. Reevaluating Amdahl's law. *Communications of the ACM 31*, 5 (1988), 532–533.

[24] Hennessy, J. L., and Patterson, D. A. *Computer Architecture: A Quantitative Approach.* The Scientific and Engineering Computation. The MIT Press, 2008.

[25] Hermanns, M. *Parallel Programming in Fortran 95 using OpenMP.* Universidad Politécnica de Madrid, Espanha, 2002.

[26] Larsen, E. W., and W., K. B. CMFD acceleration of spatial domain-decomposed neutron transport problems. *PHYSOR* (Abril 15–20, 2012), 9–178.

[27] Larsen, E. W., and W., K. B. CMFD and coarse-mesh DSA. *PHYSOR* (Abril 15–20, 2012).

[28] Larsen, E. W., and W., K. B. The relationship between the CMFD and the coarse-mesh DSA methods. *PHYSOR* (Abril 15–20, 2012), 9–178.

[29] Lewis, E., and Miller, W. *Computational Methods of Neutron Transport.* John Wiley & Sons, New York, 1984.

[30] Orlande, H. R. B., Fudym, O., Maillet, D., and Cotta, R. M. *Thermal Measurements and Inverse Techniques*. CRC Press, Taylor and Francis Group, 2011.

[31] Pautz, A. D., and Bailey, T. S. Parallel deterministic transport sweeps of structures and unstructures meshes with overload mesh decompositions. *Nuclear Science and Engineering 185* (2017), 70–77.

[32] Pazinatto, C. B. Formulação ado para o problema adjunto de transporte unidimensional e aplicação em um problema inverso de reconstrução de fonte. *Dissertaçao de Mestrado, Universidade Federal do Rio Grande do Sul, Programa de Pós-graduação em Matemática Aplicada* (2015).

[33] Ristov, S., Prodan, R., Gusev, M., and Skala, K. Superlinear speedup in HPC systems: Why and when? *Federated Conference on Computer Science and Information Systems* (2016).

[34] Scheben, F. Iterative methods for criticality computations in neutron transport theory. *University of Bath* (2011).

[35] Schulz, D. Métodos analíticos e computacionais em geofísica nuclear. *Dissertaçao de Mestrado, Universidade Federal do Rio Grande do Sul, Programa de Pós-graduação em Matemática Aplicada* (2014).

[36] Temam, O., Fricker, C., and Jalby, W. Evaluating the impact of cache interferences on numerical codes. *Proceedings of he 1993 International Conference on Parallel Processing 1* (1993), 180–189.

Apêndice A CÓDIGOS

A.1 Varredura

```
do i=2,j+1
    do m=1,n/2
        psi(m,i) = ((mu(m)/h(i-1)-.5*sigma_t(i-1)*psi(m,i-1)+
                    0.5*sigma_s0(i-1)*s1(i-1)+&
                    1.5*sigma_s1(i-1)*mu(m)*s2(i-1)+.5*q(i-1))/(mu(m)/h(i-1)+
                    .5*sigma_t(i-1))
    end do
end do
do i=j,1,-1
    do m=n/2+1,n
        psi(m,i) = ((abs(mu(m))/h(i)-.5*sigma_t(i))*psi(m,i+1)+
                    .5*sigma_s0(i)*s1(i)+&
                    1.5*sigma_s1(i)*mu(m)*s2(i)+0.5*q(i))/(abs(mu(m))/h(i)+
                    .5*sigma_t(i))
    end do
end do
```

A.2 !$OMP SECTIONS

```
!$OMP SECTIONS
!$OMP SECTION
do i=2,ishft(j+1,-1)+1
    do m=1,n/2
        !varredura para a direita
    end do
end do
!$OMP SECTION
do i=ishft(j+1,-1)+1,j+1
    do m=1,n/2
        !varredura para a direita
    end do
end do
!$OMP SECTION
do i=j,ishft(j+1,-1)+1,-1
    do m=n/2+1,n
        !varredura para a esquerda
    end do
end do
!$OMP SECTION
do i=ishft(j+1,-1)+1,1,-1
    do m=n/2+1,n
        !varredura para a esquerda
    end do
end do
!$OMP END SECTIONS
```

A.3 REDUCTION

```
!$OMP DO REDUCTION (+:phi_plus1)
do m=1,n/2
    phi_plus1=phi_plus1+mu(m)*psi(m,1)*w(m)
end do
!$OMP END DO
!$OMP DO REDUCTION (+:phi_plus2)
do m=1,n/2
    phi_plus2=phi_plus2+abs(mu(n/2+m))*psi(n/2+m,j+1)*w(n/2+m)
end do
!$OMP END DO
```

A.4 !$OMP WORKSHARE

```
!$OMP WORKSHARE
forall (i=1:nn)
    F(:, i)=(1.d0/mi(i))*F(:,i)
end forall
!$OMP END WORKSHARE
!$OMP WORKSHARE
FE=matmul(F, E)
!$OMP END WORKSHARE
```

Apêndice B QUALIDADE DA SOLUÇÃO CALCULADA EM PARALELO

Os gráficos a seguir permitem avaliar a qualidade da solução obtida através dos códigos paralelos, medida através da diferença, em módulo.

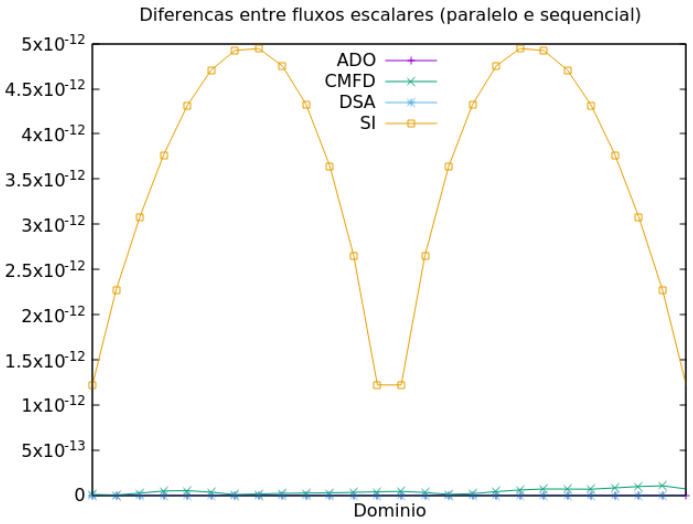


Figura B.1: Diferença entre a solução calculada pela versão paralela e pela versão sequencial para o problema 2

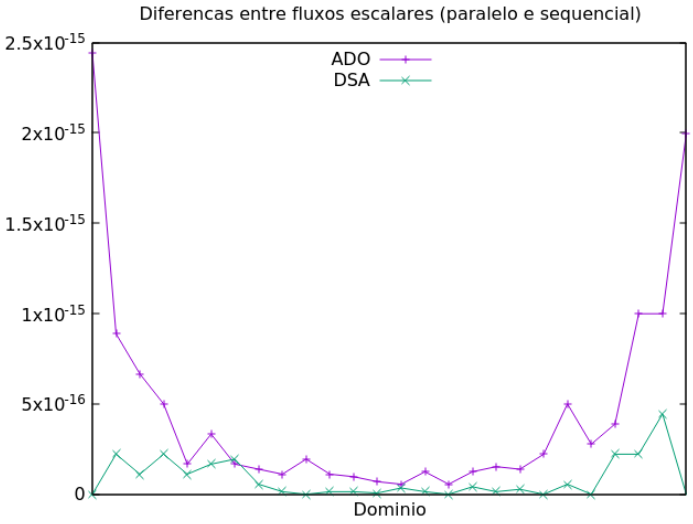


Figura B.2: Diferença entre a solução calculada pela versão paralela e pela versão sequencial para o problema 2

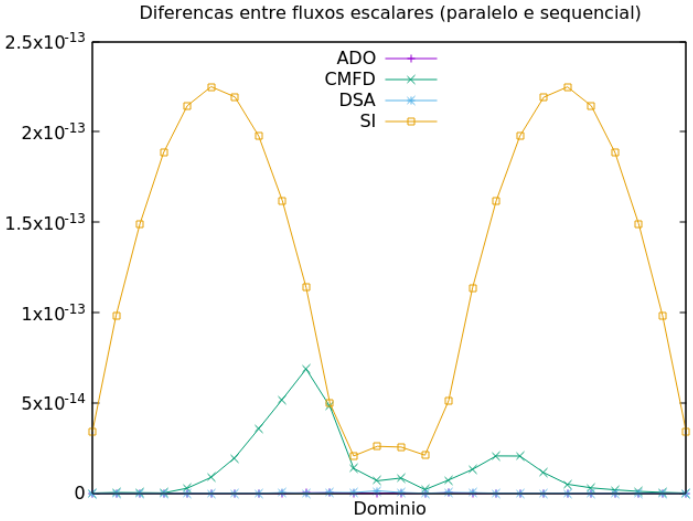


Figura B.3: Diferença entre a solução calculada pela versão paralela e pela versão sequencial para o problema 3

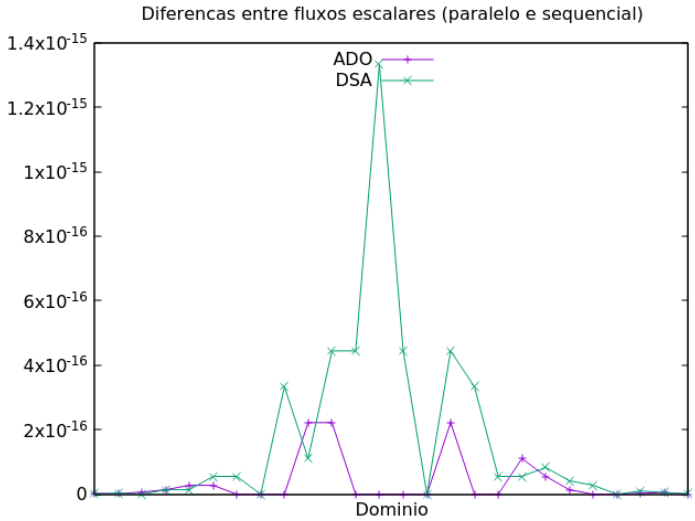


Figura B.4: Diferença entre a solução calculada pela versão paralela e pela versão sequencial para o problema 3