UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

AYMAR ANTONIO VILAS BOAS PESCADOR JR

DETERMINAÇÃO DE PARÂMETROS ÓTIMOS PARA O MÉTODO MULTIGRID DE CORREÇÕES ADITIVAS

Florianópolis
2010

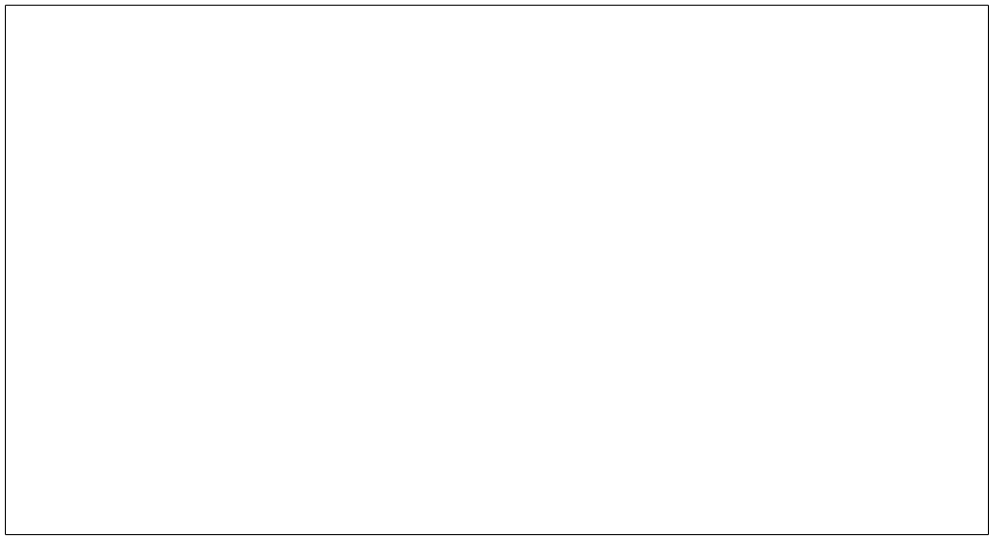
AYMAR ANTONIO VILAS BOAS PESCADOR JR

DETERMINAÇÃO DE PARÂMETROS ÓTIMOS PARA O MÉTODO MULTIGRID DE CORREÇÕES ADITIVAS

Trabalho de Curso submetido ao Curso de Graduação em Engenharia Mecânica da Universidade Federal de Santa Catarina
Orientador: Prof. Clovis R Maliska, Ph.D.

Florianópolis
2010

Catalogação na fonte elaborada pela biblioteca da
Universidade Federal de Santa Catarina



AYMAR ANTONIO VILAS BOAS PESCADOR JR

DETERMINAÇÃO DE PARÂMETROS ÓTIMOS PARA O MÉTODO MULTIGRID DE CORREÇÕES ADITIVAS

Este Trabalho de Curso foi julgado adequado para obtenção do Título de Engenheiro Mecânico, e aprovado em sua forma final pelo Curso de Graduação em Engenharia Mecânica da Universidade Federal de Santa Catarina

Florianópolis, 01 de janeiro de 2010

Prof. Jonny Carlos da Silva, Dr.Eng.
Coordenador do Curso

Banca Examinadora:

Prof. Clovis R Maliska, Ph.D.
Orientador

Fernando Sandro Velasco Hurtado, M.Eng.

Prof. Dylton do Vale Pereira Filho, M.Eng.

DEDICATÓRIA

Aos meus pais, que me ensinaram a ser um homem justo, determinado e honesto.
À minha irmã, que me enriqueceu como pessoa.
E à minha namorada, que me deu apoio, confiança e felicidade.

AGRADECIMENTOS

"Agradeço todas as dificuldades que enfrentei; não fosse por elas, eu não teria saído do lugar. As facilidades nos impedem de caminhar. Mesmo as críticas, nos auxiliam muito." Chico Xavier.

Agradeço à Universidade Federal de Santa Catarina, em especial ao departamento de Engenharia Mecânica, pela formação profissional de qualidade que recebi.

Ao laboratório de Simulação Numérica em Mecânica dos Fluidos e Transferência de Calor - SINMEC, onde este trabalho foi desenvolvido, por sua infraestrutura que me permitiu estudar, aprender e desenvolver muito na área de simulação numérica.

À Petrobras e à rede SIGER (Simulação e Gerenciamento de Reservatórios de Petróleo), pelos projetos em parceria com o SINMEC, os quais me envolvi e recebi muita motivação e incentivo.

Ao prof. Clovis Raimundo Maliska, por ter me orientado neste trabalho, por ter depositado sua confiança em mim e me dado condições de entrar para a equipe do SINMEC e me aprofundar na área de simulação numérica.

Ao prof. António Fábio C. da Silva, que me apresentou a área de simulação numérica pela primeira vez, através da disciplina de Transferência de Calor e Mecânica dos Fluidos Computacional, logo no começo da graduação.

Ao Fernando, ao Carlos, à Karime e ao Jaime, que tiveram paciência para me ensinar e me aconselhar muito no meu primeiro ano no laboratório.

A todos os colegas e amigos do SINMEC, Gustavo, Tada, Leonardo, Tati, Axel, Arthur, Bruno, Cristiano, Michele, Vitinho e Ederson, por me ensinarem tanto, e sempre estarem dispostos a tirar dúvidas e ajudarem no que for preciso.

A toda a minha família pela força e pelo apoio. Aos meus pais, Josiane e Aymar, que me ensinaram o significado de humildade, honestidade, responsabilidade, justiça e honra. À minha irmã Suzana, que sempre esteve do meu lado, torcendo por mim.

À minha namorada Deise, por compartilhar comigo todos os momentos e trazer felicidade aos meus dias mais tristes.

E aos meus grandes amigos Bernardo, Lucas, Mauro, Alex, Fran, Natasha e Fabíola pelos momentos de descontração, que são tão importantes para que os de concentração valham a pena.

"Há pessoas que transformam o sol numa simples mancha amarela, mas há também aquelas que fazem de uma simples mancha amarela o próprio sol."

Pablo Picasso

RESUMO

Em simulações numéricas de engenharia frequentemente são necessárias soluções de grandes sistemas de equações lineares. Logo, existe a necessidade de se dispor de métodos robustos, consistentes e eficientes de resolução de tais sistemas de equações. Uma classe específica destes resolve os sistemas empregando simultaneamente várias malhas para acelerar a convergência de métodos convencionais. Estes são denominados métodos *multigrid*. Tais métodos são compostos basicamente de um procedimento de aglomeração, operadores de restrição e prolongação e ciclos. A literatura está repleta de algoritmos para tais procedimentos. Porém, para sua aplicação prática é necessário determinar parâmetros ótimos de funcionamento desses algoritmos. Portanto, neste trabalho é apresentado um estudo na tentativa de identificar algum método passível de ser automatizado para obter tais parâmetros.

Palavras-chave: Multigrid, multigrid algébrico, Multigrid de Correções Aditivas, ACM, iterações nos níveis, iterações nas malhas, parâmetros multigrid.

ABSTRACT

In numerical engineering simulations many large linear equations system's solutions are frequently required. Therefore, robust, consistent and efficient methods to solve such equation systems are needed. An specific class of those uses many grids at once to accelerate the convergence of conventional solvers. They are the multigrid methods. Such methods are basically composed of an aglomeration procedure, a restriction and a prolongation operators and cycles. The literature has a lot of algorithms for such procedures. However, it is required to determine optimum parameters for its proper usage. Thus, this work presents an attempt to identify some method that can be automated to obtain these quantities.
Keywords: Multigrid, algebraic multigrid, Additive Correction Multigrid, ACM, iterations in each level, iterations in each grid, multigrid parameters.

LISTA DE FIGURAS

LISTA DE TABELAS

CONTEÚDO

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Em simulações numéricas de problemas de engenharia, frequentemente são necessárias soluções de sistemas lineares diversos. Principalmente, quando modelos diferenciais são resolvidos numericamente. Tais metodologias se utilizam de malhas com entidades finitas (volumes, elementos, pontos), de modo que seja possível realizar discretizações e linearizações das equações governantes. Obtendo-se, portanto, sistemas de equações discretizadas e lineares.

Os sistemas são construídos de forma que suas soluções representem os campos de determinadas propriedades de interesse em todo o domínio simulado.

Portanto, existe a necessidade de se dispor de métodos robustos, consistentes e eficientes de resolução de tais sistemas de equações.

Uma série de métodos de solução de sistemas lineares estão presentes na literatura. E, podem ser classificados segundo os critérios abaixo:

- Métodos diretos: Trabalham com todos os elementos da matriz e obtêm o resultado exato do sistema linear com um número finito de operações [4]. Ex.: eliminação de Gauss-Jordan; obtenção da solução através das matrizes L e U; regra de Kramer; inversão da matriz de coeficientes e posterior multiplicação pelo vetor independente.

- Métodos iterativos: Requerem uma estimativa inicial sobre a qual serão aplicadas várias operações de modo a reduzir seus erros. Só obtêm a solução exata se executarem um número infinito de operações. Tais métodos utilizam-se de critérios de convergência e tolerâncias para parar o processo [4]. Existe também uma subdivisão destes métodos:

  - Estáticos: Os coeficientes utilizados em cada iteração não mudam ao longo do processo [5]. Ex.: método de Jacobi; método de Gauss-Seidel; decomposição LU incompleta; método de sobrerrelaxações sucessivas.
  - Não estáticos: São calculados novos coeficientes auxiliares em cada iteração[5].

Ex.: métodos baseados no subespaço de Krilov [6] (método de gradientes conjugados, método de gradientes biconjugados, método de gradientes biconjugados estabilizado, etc); método do resíduo mínimo generalizado (GMRES);

Destes, os métodos diretos costumam exigir muito esforço computacional para sistemas lineares grandes, já que muitas vezes o esforço exigido é equivalente a inverter uma matriz, processo que possui complexidade cúbica [4] (esta complexidade é atribuída a algoritmos cujo tempo de cálculo despendido para rodá-lo aumenta com o cubo da quantidade de incógnitas envolvidas). Logo, muitas vezes tornam-se inviáveis e são geralmente descartados. Portanto, utiliza-se quase de maneira exclusiva métodos iterativos (que possuem complexidade quadrática ou inferior).

Se uma solução para um sistema linear for estimada, é comum que tal solução possua erros, ou seja, não satisfaça o sistema em questão. Os métodos iterativos visam reduzir estes erros até que a solução corrigida possa ser considerada aceitável, dentro de determinada tolerância pré-estabelecida. Tais erros, presentes em uma solução estimada, podem ser decompostos por série de Fourier, de modo que o erro total seja composto de infinitos erros harmônicos (tais harmônicos variando de frequência nula a infinita), como ilustrado na figura 1.

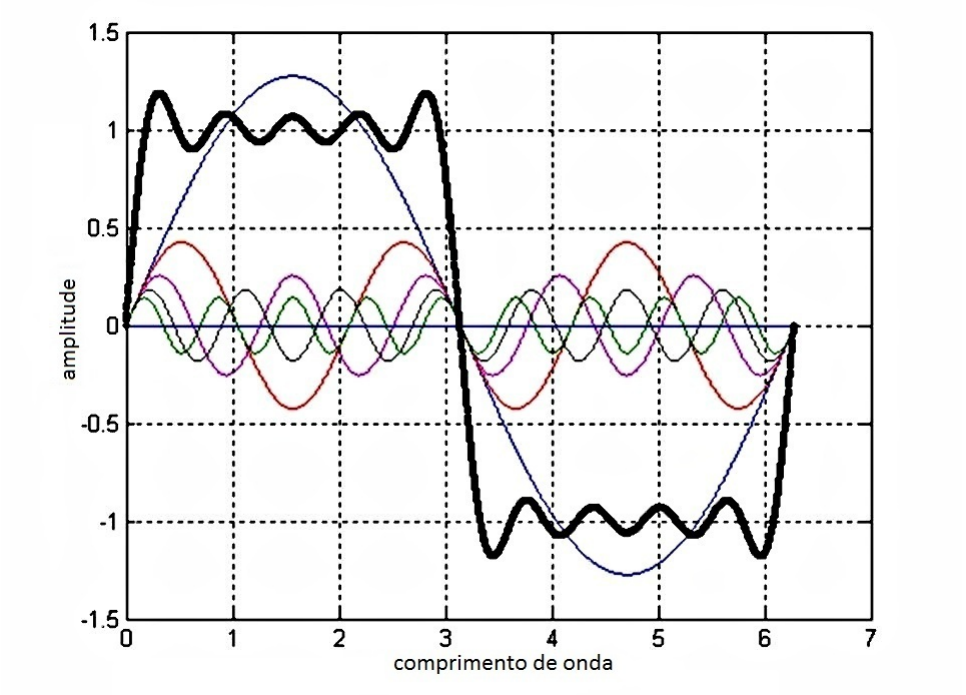


Figura 1: Decomposição de uma função arbitrária por série de Fourier

A grande limitação dos métodos iterativos é que são eficientes apenas nas componentes do erro que possuam comprimento de onda compatíveis com o tamanho da malha empre-

gada [4]. Tais componentes são minimizadas rapidamente e, logo em seguida, o método perde eficiência, pois normalmente sobram erros de comprimento de onda não compatíveis com o tamanho da malha. A figura 2 ilustra tal comportamento.

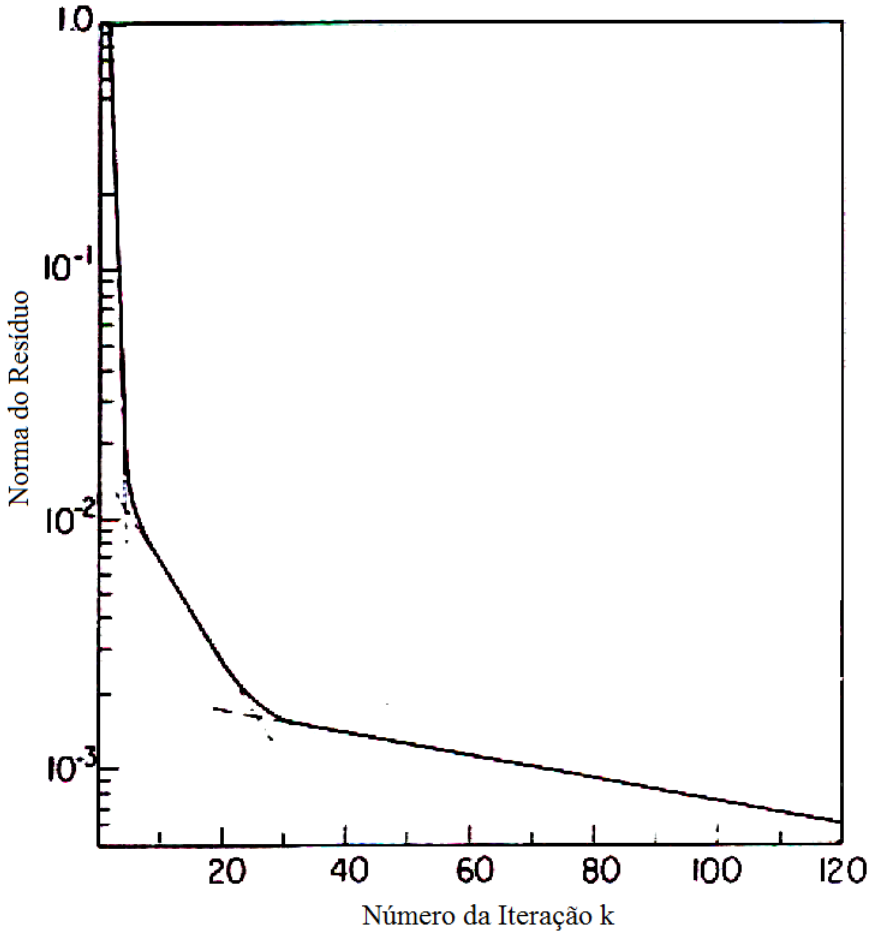


Figura 2: Norma do resíduo pela iteração k usando um método iterativo, adaptado de [1]

Surge então a idéia de resolver um problema empregando simultaneamente várias malhas, eliminando com eficiência os erros em todo o espectro de frequências, ou seja, empregando malhas compatíveis com erros de diversos comprimentos de onda. Desta forma, é possível acelerar a convergência através do melhor aproveitamento dos métodos iterativos convencionais (obtendo muitas vezes complexidade linear, ou seja, o tempo de cálculo cresce linearmente com o tamanho da matriz). Estes são denominados métodos *multigrid*.

Os métodos *multigrid* compõem uma nova gama de métodos de solução de sistemas lineares. Podem ser considerados métodos iterativos (com algumas sofisticações adicionais), já que a cada nova iteração o erro da solução é reduzido. Entretanto, também podem ser vistos como aceleradores da taxa de convergência dos métodos convencionais, já que em cada iteração *multigrid* são efetuadas várias iterações destes para várias malhas diferentes.

Normalmente, a malha mais fina do sistema de malhas utilizados por um *multigrid* é a malha do problema discreto sendo resolvido. Em seguida, são executados recursivamente algoritmos de aglomeração de malha, gerando assim um sistema de malhas. Após a geração destas inicia-se a solução do sistema de equações. A figura 3 ilustra um sistema de malhas hipotético.

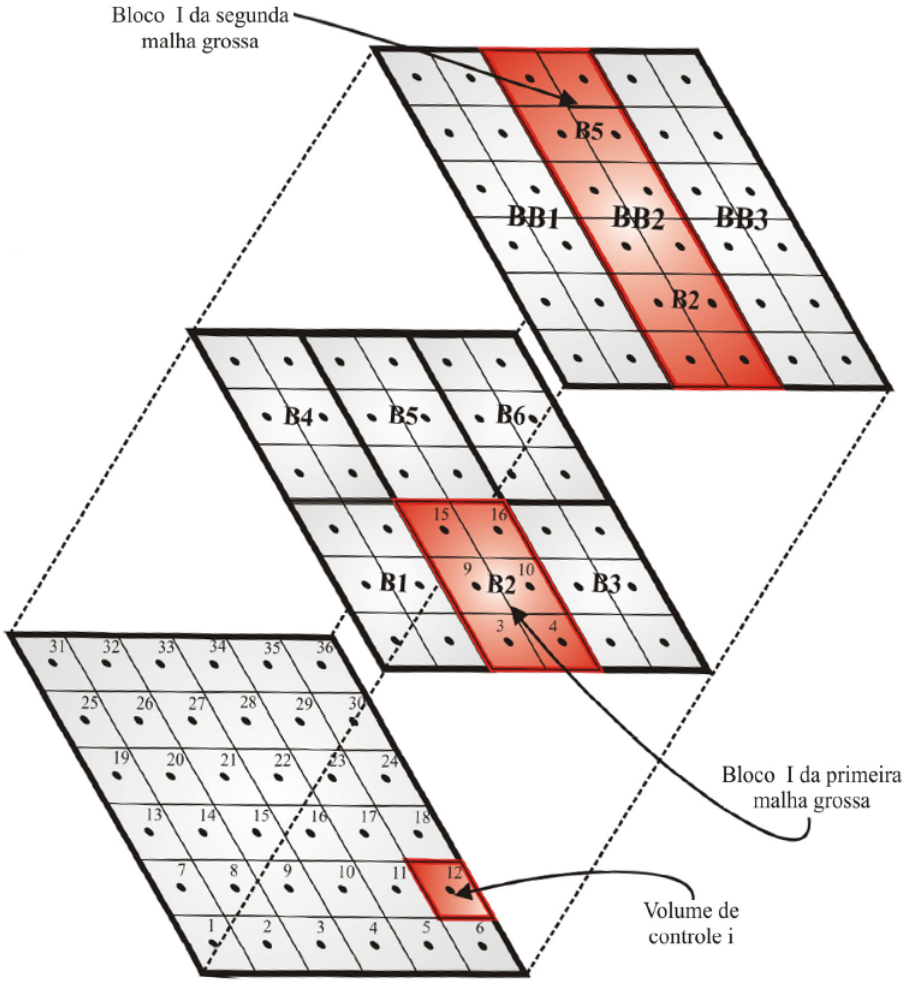


Figura 3: Níveis de malha gerados após aglomeração, adaptado de [2].

No processo de solução usual visita-se determinado nível de malha; algumas iterações de um método iterativo convencional são realizadas - se o nível for o mais grosseiro, pode-se resolvê-lo com um método direto, já que a matriz gerada é muito pequena, e, portanto, a relação de custo (de tempo) e benefício (qualidade do resultado) vale a pena - ; suas informações são transmitidas para a próxima malha mais fina ou mais grosseira; repete-se tal procedimento segundo algum ciclo até que se obtenha uma solução convergida na malha original.

A transmissão de informações de uma malha para a próxima mais grosseira é denominada restrição. Enquanto que a transferência de dados de uma malha para a próxima mais

fina é chamada prolongação.

A ordem com que as malhas são visitadas em uma iteração *multigrid* é chamada de ciclo [2]. A figura 4 ilustra os ciclos mais utilizados.

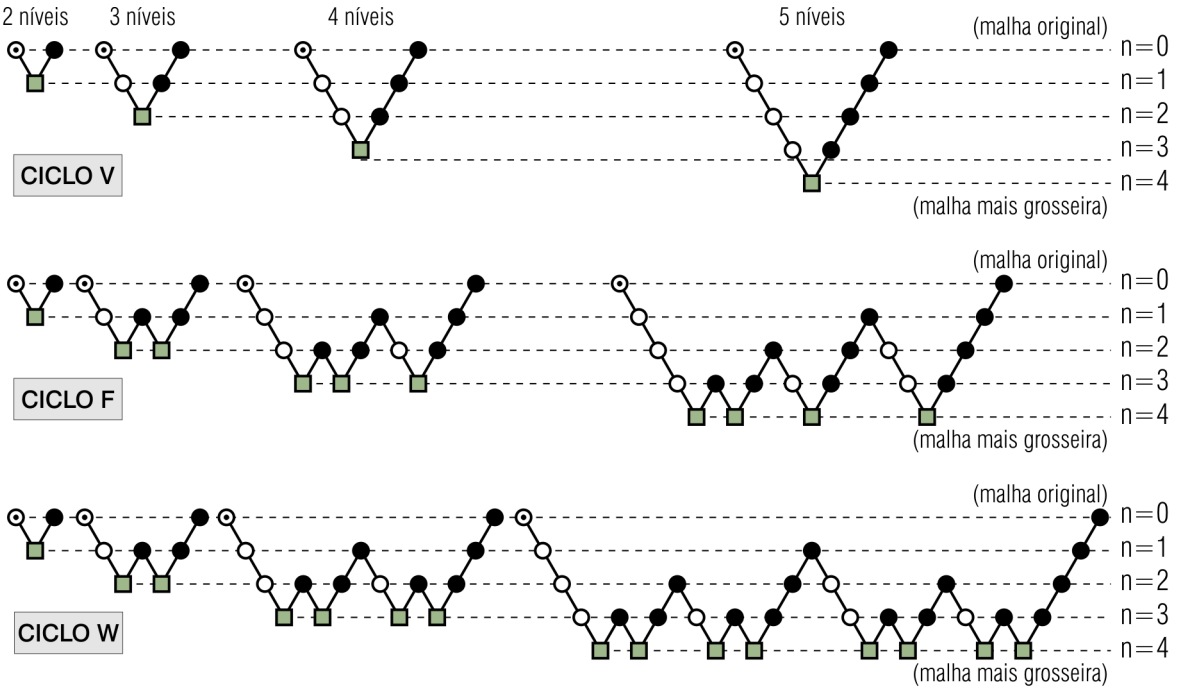


Figura 4: Ciclos *multigrid* mais utilizados, adaptado de [3]

A literatura está repleta de algoritmos de aglomeração, operadores de restrição e prolongação, ciclos. Porém, faltam métodos para avaliar quantas iterações dos métodos iterativos convencionais devem ser realizadas em cada malha do sistema *multigrid*. Portanto, decidiu-se realizar um estudo na tentativa de identificar algum método passível de ser automatizado para obter estas quantidades.

1.2 Objetivos

O laboratório SINMEC (Laboratório de Simulação Numérica em Mecânica dos Fluidos e Transferência de Calor) em parceria com a rede SIGER (Simulação e Gerenciamento de Reservatórios de Petróleo) realizou um projeto cujo objetivo foi a confecção de uma biblioteca computacional para auxiliar o desenvolvimento de simuladores que utilizam o Método dos Volumes Finitos baseado em Elementos (EbFVM - *Element based Finite Volume Method*).

Uma parte desta biblioteca é um conjunto de métodos de solução de sistemas lineares. Entre estes foi desenvolvido o *Multigrid* de Correções Aditivas (ACM - *Additive Correction Multigrid*).

Logo, foi percebido que o *multigrid* depende de alguns parâmetros e que seu desempenho é bastante sensível a estes. Principalmente quando se trata da quantidade de iteração nos níveis.

Portanto, objetiva-se avaliar vários métodos de obtenção de tais quantidades através de comparação de tempos de computação gastos e da capacidade de evitar a divergência. Permitindo, desta forma, a identificação e classificação dos métodos avaliados.

1.3 Organização do Trabalho

No capítulo 2 é explanada a estrutura geral dos métodos *multigrid* tradicionais e a estrutura específica do método utilizado no trabalho (*Multigrid* de Correções Aditivas).

Em seguida, no capítulo 3, alguns métodos para decidir se deve-se continuar resolvendo o método iterativo em determinado nível são apresentados.

Finalmente, no capítulo 4 encontram-se alguns resultados obtidos e, no capítulo 5, a conclusão do trabalho.

2 MULTIGRID DE CORREÇÕES ADITIVAS

Um método *multigrid* é constituído de um esquema de aglomeração e da definição dos operadores de restrição e prolongação. Já que, se estiverem disponíveis tais informações, é possível estipular alguns parâmetros (tolerâncias, parâmetros de aglomeração, ciclos) e resolver sistemas lineares.

O *Multigrid* de Correções Aditivas (método utilizado no trabalho) tem como objetivo estruturar os níveis de malha de forma que o resultado do sistema linear de um nível grosseiro seja a correção a ser aplicada na estimativa do próximo nível mais fino [2] [1]. Para isto, parte-se do sistema linear

$$[\mathbf{A}]\vec{\phi} = \vec{b} \tag{2.1}$$

sendo $\vec{\phi}$ o vetor solução do sistema linear, ou seja, a saída esperada de um método de solução. Se $\vec{\phi}'$ é uma estimativa para $\vec{\phi}$, então o vetor erro pode ser definido por

$$\vec{e} = \vec{\phi}' - \vec{\phi} \tag{2.2}$$

e, consequentemente, pode-se definir o vetor de correções como

$$\vec{c} = -\vec{e} \tag{2.3}$$

Substituindo (2.3) em (2.2) nota-se que $\vec{\phi}$ pode ser obtido adicionando a correção $\vec{c}$ na estimativa $\vec{\phi}'$, ou seja,

$$\vec{\phi} = \vec{\phi}' + \vec{c} \tag{2.4}$$

Substituindo (2.2) em (2.1) obtém-se

$$[\mathbf{A}](\vec{\phi}' - \vec{e}) = \vec{b} \tag{2.5}$$

logo,

$$[\mathbf{A}]\vec{e} = [\mathbf{A}]\vec{\phi}' - \vec{b} \tag{2.6}$$

Ainda, definindo o vetor resíduo correspondente à estimativa da solução

$$\vec{r} = [\mathbf{A}]\vec{\phi}' - \vec{b} \tag{2.7}$$

portanto,

$$[\mathbf{A}]\vec{e} = \vec{r} \tag{2.8}$$

Logo, substituindo (2.3) em (2.8)

$$[\mathbf{A}]\vec{c} = -\vec{r} \tag{2.9}$$

Portanto, se forem tomados dois níveis do sistema (um nível qualquer e o próximo mais grosseiro), o sistema linear do mais grosseiro é construído com base na equação (2.9), de forma que sua matriz seja a própria matriz do nível fino aglomerada (neste procedimento o esquema de aglomeração é utilizado); e seu vetor independente seja a aglomeração do oposto do resíduo do nível fino.

Logo, a solução deste nível grosseiro representa a correção da estimativa do nível fino.

Desta forma, após a aglomeração da matriz (que permanece inalterada durante todo o processo de solução), são utilizados apenas os operadores de restrição e prolongação para, respectivamente, transmitir o oposto do resíduo do nível fino para o grosseiro e transmitir a correção de volta ao nível fino.

2.1 AGLOMERAÇÃO

A aglomeração é um processo que visa gerar uma malha mais grosseira à partir de uma refinada. O procedimento é realizado recursivamente um número fixo de vezes (gerando esse mesmo número de novas malhas) ou até que a última malha gerada satisfaça algum critério. O critério utilizado no trabalho é que a última malha gerada possua menos células do que uma tolerância especificada (exemplo: a malha mais recente é considerada a última do sistema se possuir menos de 50 células).

A aglomeração é composta por duas etapas:

- Selecionar as células que irão fazer parte de um mesmo bloco na nova malha, ou seja, decidir em qual bloco da nova malha cada célula vai estar;
- Através das informações obtidas na etapa anterior aglomerar o sistema linear, ou seja, definir de que forma os coeficientes da nova malha serão computados.

A primeira etapa pode ser feita através do processamento de informações geométricas (método geométrico) ou da análise dos coeficientes do sistema linear (método algébrico).

Os métodos geométricos costumam ser de fácil implementação em malhas simples ou estruturadas, porém não tratam anisotropia das propriedades físicas do meio e são bastante restritos (quanto aos tipos de malhas).

Já os métodos algébricos independem da malha, tratam todos os tipos de anisotropia, geram resultados melhores, entretanto são mais complexos e elaborados que os geométricos.

No trabalho foi utilizado um procedimento algébrico que analisa a matriz de coeficientes e, por consequência, leva em consideração a força (ordem de grandeza) do acoplamento entre duas células. Mais detalhes do algoritmo utilizado em [2].

A segunda etapa pode ser feita através da aplicação do método númerico na nova malha (discretização das equações, etc) ou através da aglomeração dos coeficientes da matriz pertencente ao nível fino. O ACM(*Additive Correction Multigrid*, ou seja, o *Multigrid* de Correções Aditivas) realiza a segunda opção, partindo do princípio que os coeficientes provém de equações de balanço do tipo

$$a_p \phi_P = \sum a_{nb} \phi_{NB} + B^{\phi} \tag{2.10}$$

de modo que o sistema linear seja

$$\begin{pmatrix} a_p^1 & -a_{nb,2}^1 & \cdots & -a_{nb,n}^1 \\ -a_{nb,1}^2 & a_p^2 & \cdots & -a_{nb,n}^2 \\ \vdots & \vdots & \ddots & \vdots \\ -a_{nb,1}^n & -a_{nb,2}^n & \cdots & a_p^n \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{pmatrix} = \begin{pmatrix} B_1^{\phi} \\ B_2^{\phi} \\ \vdots \\ B_n^{\phi} \end{pmatrix} \tag{2.11}$$

Como já foi citado, a matriz de coeficientes de uma malha grosseira deve ser a própria matriz do nível fino aglomerada. Tal condição é satisfeita através do seguinte procedimento: o coeficiente da diagonal de um bloco gerado é computado somando os termos de diagonal e todas as conexões internas das células que o compõe; os coeficientes de conexão deste bloco com outros são computados somando os termos de conexão entre as células deste bloco e as células dos blocos vizinhos.

Para exemplificar será tomada a aglomeração mostrada na figura 5.

Os coeficientes relativos ao bloco 5 seriam calculados da seguinte forma:

$$\begin{aligned} a_p^{B5} = & a_p^{15} + a_p^{16} + a_p^{21} + a_p^{22} - a_{nb,16}^{15} - a_{nb,21}^{15} - a_{nb,15}^{16} \\ & - a_{nb,22}^{16} - a_{nb,15}^{21} - a_{nb,22}^{21} - a_{nb,16}^{22} - a_{nb,21}^{22} \end{aligned} \tag{2.12}$$

$$a_{nb,B2}^{B5} = a_{nb,9}^{15} + a_{nb,10}^{16} \tag{2.13}$$

$$a_{nb,B4}^{B5} = a_{nb,14}^{15} + a_{nb,20}^{21} \tag{2.14}$$

$$a^{B5}_{nb,B6} = a^{16}_{nb,17} + a^{22}_{nb,23} \tag{2.15}$$

$$a^{B5}_{nb,B8} = a^{21}_{nb,27} + a^{22}_{nb,28} \tag{2.16}$$

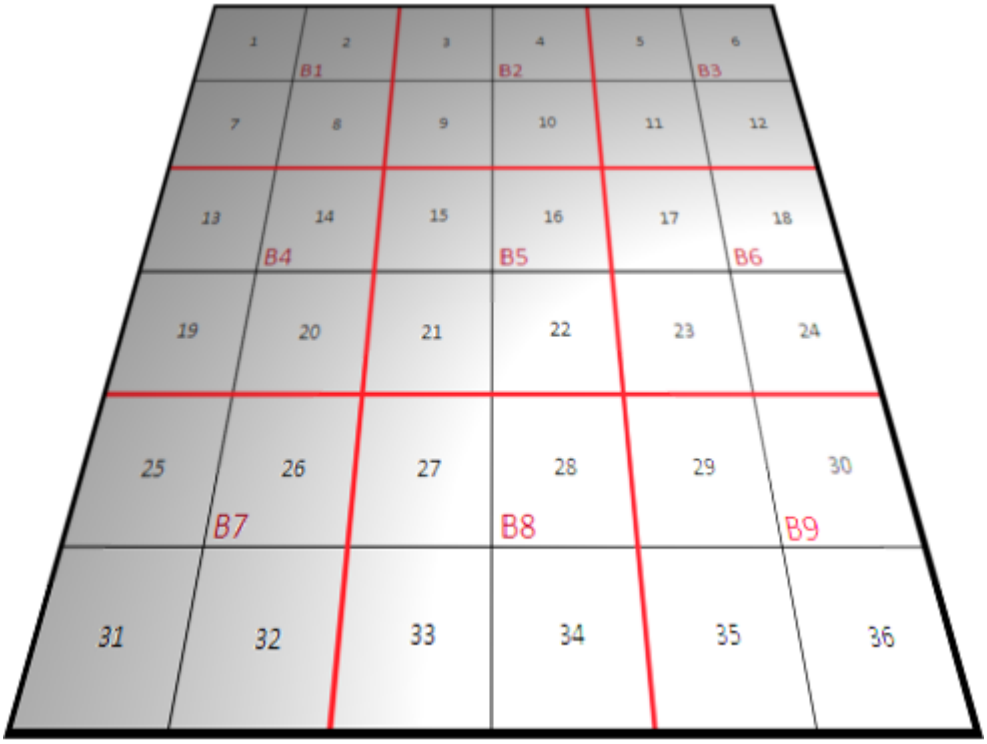


Figura 5: Aglomeração hipotética

2.2 Restrição e Prolongação

Os operadores de restrição e de prolongação definem a maneira com que os níveis de malha compartilham informações.

No *Multigrid* de Correções Aditivas o sistema linear do nível que recebe dados através da restrição se baseia na equação (2.9). De modo que a restrição seja feita computando o vetor independente de tal nível através do oposto da soma dos resíduos relativos a todas as células que compõem cada bloco. Se for tomado como exemplo a aglomeração ilustrada na figura 5 o termo independente do bloco B5 será

$$B^{\phi}_{B5} = -(r_{15} + r_{16} + r_{21} + r_{22}) \tag{2.17}$$

desde que

$$r_i = a^i_p \phi'_i - \sum a^i_{nb,j} \phi'_j - B^{\phi}_i \tag{2.18}$$

Já a prolongação efetuada no ACM é a operação que dá nome ao método. Sendo que, em tal operação, a *correção aditiva* calculada numa malha grosseira é adicionada à estimativa da malha fina. Portanto, no exemplo utilizado, a solução referente ao bloco B5 será adicionada às estimativas para as células 15, 16, 21 e 22 (contidas em B5).

$$\phi_{15} = \phi'_{15} + \phi_{B5} \tag{2.19}$$

$$\phi_{16} = \phi'_{16} + \phi_{B5} \tag{2.20}$$

$$\phi_{21} = \phi'_{21} + \phi_{B5} \tag{2.21}$$

$$\phi_{22} = \phi'_{22} + \phi_{B5} \tag{2.22}$$

2.3 Métodos Iterativo e Direto Empregados

Como o intuito de um *multigrid* é acelerar a taxa de convergência de métodos iterativos, dificilmente se justifica sua aplicação em métodos que possuam taxa de convergência alta, como os não estáticos (gradientes conjugados, GMRES).

Logo, optou-se por empregar um método simples, rápido e que utilize de forma otimizada a estrutura de armazenamento de matrizes esparsas utilizada.

Portanto, o método de Gauss-Seidel foi empregado em todos os casos avaliados neste trabalho [4] [6]. E foi desenvolvido de forma que uma iteração realize duas varreduras do método tradicional, uma da primeira linha do sistema linear até a última e outra da última de volta à primeira ("limpando" a solução de eventuais erros relacionados à direção de varredura). Assim, sempre que se referenciar uma iteração do método iterativo, deve-se entender por estas duas varreduras citadas.

Entretanto, o sistema linear do nível mais grosseiro normalmente possui poucas equações (dependendo do critério de parada da aglomeração). Logo, muitas vezes vale o esforço de resolver tal nível através de um método direto, obtendo a solução exata (resíduos nulos em tal solução) e, por consequência, melhores correções aditivas.

O método direto empregado é a decomposição da matriz de coeficientes em L e U, e subsequente obtenção da solução através de tais matrizes [2] [4]. Tal estratégia gera grande desempenho, já que as matrizes L e U dependem apenas da matriz de coeficientes e podem ser geradas uma única vez, no início do processo, já que todas matrizes do sistema *multigrid* permanecem inalteradas durante todo o procedimento de solução.

2.4 IMPLEMENTAÇÃO COMPUTACIONAL

Procurou-se estruturar o código computacional utilizando fortemente conceitos de programação orientada a objetos. Logo, foi possível conceber um esquema de classes computacionais inter-relacionadas de forma a deixar o código claro, fácil de ser reusado e robusto. Foi utilizada a linguagem C++ por ser uma linguagem de médio nível. Permitindo, desta forma, obter um código com alto desempenho, e ao mesmo tempo estruturado através de classes, hierarquias e orientação a objetos.

O diagrama de classes UML(*Unified Modeling Language*) que representa a estrutura implementada está ilustrado na figura 6.

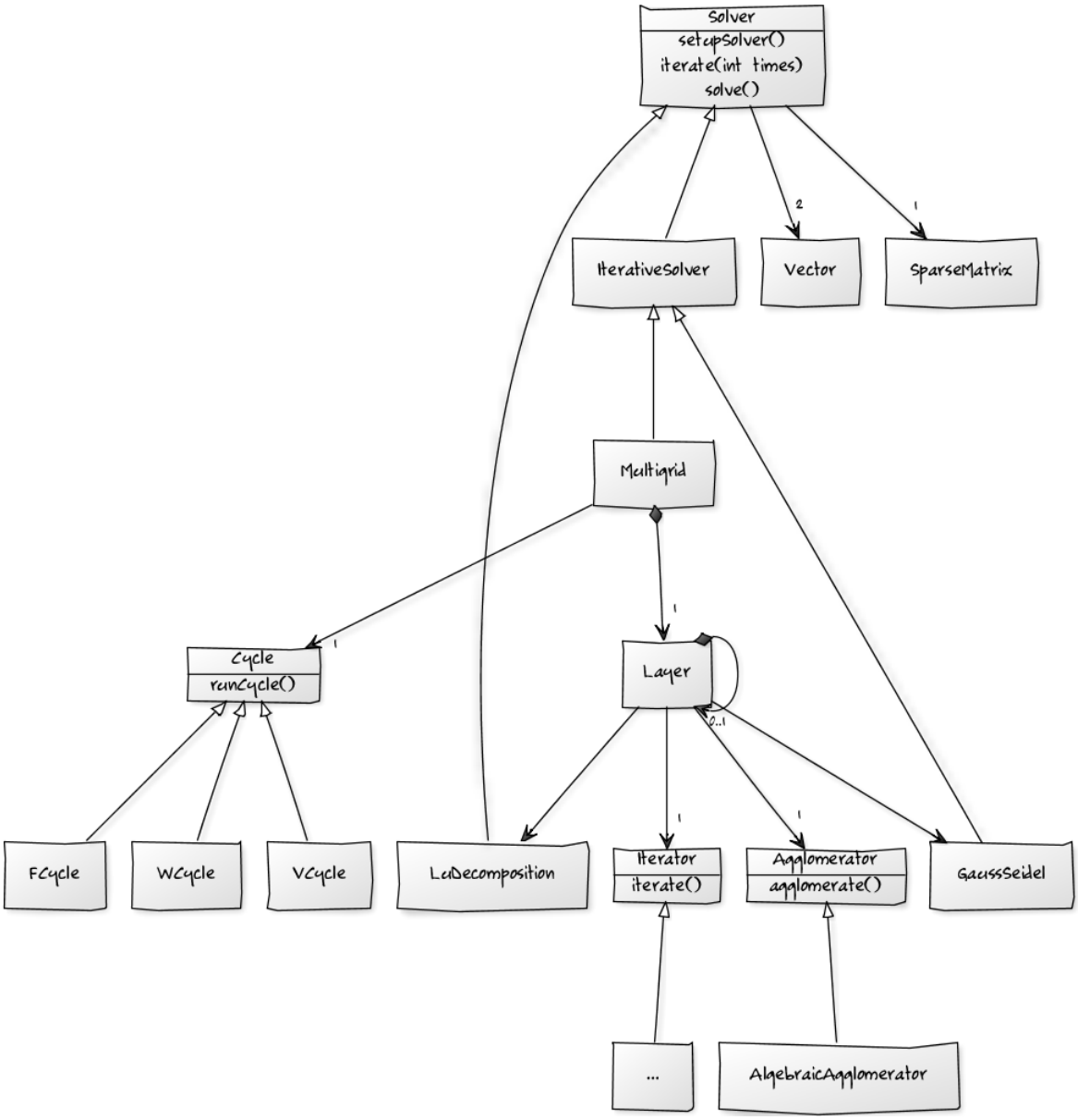


Figura 6: Diagrama de classes

Sendo que a classe abstrata *Solver* recebe um sistema linear representado por dois objetos *Vector* e um *SparseMatrix* (os vetores de solução, de termos independentes e a matriz de coeficientes). Esta classe deriva então a *LuDecomposition*, que é capaz de resolver um sistema linear através da decomposição LU; e a *IterativeSolver* que representa métodos iterativos de solução.

As classes *GaussSeidel* e *Multigrid* são derivadas da *IterativeSolver*, de modo que cada uma possua sua implementação específica para resolver sistemas lineares de maneira iterativa.

O algoritmo para iterar a solução de um sistema linear através do método de Gauss-Seidel encontra-se implementado na classe *GaussSeidel*. Tanto esta, quanto a *LuDecomposition* podem ser utilizadas como métodos de solução independentes. Todavia, neste trabalho elas serão tratadas apenas como classes auxiliares, já que o objetivo do estudo é o método *multigrid*.

A classe *Multigrid* utiliza uma derivação da *Cycle* (*VCycle*, *WCycle* ou *FCycle*), que é responsável por determinar a ordem de varredura dos níveis de malha em cada ciclo. Novos ciclos podem ser desenvolvidos, já que para realizar tal tarefa basta derivar a classe abstrata *Cycle* e implementar o ciclo desejado. Adicionalmente, um objeto *Multigrid* possui uma instância da *Layer*, para representar a malha original.

Cada objeto da *Layer* possui outro da mesma classe, representando o próximo nível de malha mais grosseiro. Para que os níveis grosseiros sejam gerados, as instâncias da *Layer* utilizam alguma implementação da classe *Agglomerator*, que recebe como entrada um nível de malha e retorna a aglomeração deste (no caso, a única implementação disponível é a *AlgebraicAgglomerator*, que realiza a aglomeração algébrica já comentada). A *Layer* ainda se utiliza dos métodos *LuDecomposition* e *GaussSeidel* para resolver ou iterar os sistemas lineares. Quando deseja-se iterar a solução de uma malha, esta utiliza uma derivação da classe *Iterator*, que se encarrega de calcular o número de vezes que deve-se iterar em tal nível e de realizar tais iterações.

Desta forma, facilitou-se muito realizar o estudo proposto no trabalho. Uma vez que, após a concepção inicial, só foi necessário criar derivações com algoritmos variados para a classe *Iterator*, substituindo o bloco "..." presente na figura 6.

2.4.1 Armazenamento da matriz de coeficientes

As matrizes oriundas de métodos numéricos diferenciais costumam ter tamanho, no mínimo, igual ao número de pontos da malha. Portanto, se uma malha com um milhão de pontos for utilizada (discretização comum em tais simulações) a matriz terá pelo menos um milhão de linhas e colunas.

Contudo, tais matrizes são altamente esparsas. Se tomarmos uma malha tridimensional cartesiana, na qual, cada ponto está conectado a 6 vizinhos, a matriz gerada possuirá em torno de 7 milhões de elementos não-nulos (cada linha possui um elemento na diagonal e seis de conexão), enquanto que a matriz completa comportará um trilhão de elementos. Isto significa que 99,9993% dos elementos desta matriz são zeros e apenas 0,0007% são coeficientes relevantes.

Logo, se for utilizada dupla precisão no armazenamento de cada coeficiente (usualmente empregada), cujo consumo de memória seja de 8 *bytes* por elemento, a matriz completa consumiria em média 8.000GB (*Gigabytes*). Sendo que as máquinas atuais possuem de 2 a 16GB de memória total (a memória efetivamente disponível é pelo menos 0,5Gb menor do que o valor total, já que o sistema operacional e alguns aplicativos necessitam constantemente de memória para seu funcionamento correto). De modo que seria impraticável tal simulação.

Entretanto, se forem armazenados apenas os elementos não-nulos (considerando vetores auxiliares, comuns nos métodos de armazenamento de matrizes esparsas) o consumo seria em torno de 21MB(*Megabytes*). Desta forma, tal simulação pode ser realizada sem problemas de falta de memória, já que a quantia consumida é muito inferior à quantia disponível nos sistemas atuais.

Assim, percebe-se claramente a necessidade de utilizar algum método de armazenamento de matrizes esparsas.

Muitos métodos estão presentes na literatura, desde esquemas simples que armazenam um vetor com os elementos, outro com as linhas onde se encontram cada elemento e outro com as colunas, até esquemas mais complexos que armazenam linhas comprimidas, tratam as diagonais e utilizam as informações de posicionamento de cada elemento nos vetores.

O método utilizado no trabalho e implementado na classe *SparseMatrix* do diagrama é o MCSR (*Modified Compressed Sparse Row*, ou seja, Linha Esparsa Comprimida Modificado) [3]. O MCSR é um dos métodos que menos consomem memória. O método consiste em armazenar a matriz com o uso de apenas dois vetores, um com elementos e outro com índices auxiliares que permitem a localização dos coeficientes na matriz.

Desta forma, ocupa algo em torno de 66% do consumo obtido no método mais simples (3 vetores, um para os coeficientes, outro para as linhas e outro para as colunas), sendo que os métodos mais eficientes utilizados normalmente (Linha Esparsa Comprimida, Coluna Esparsa Comprimida) raramente consomem menos de 70% do método comum.

Outra vantagem do método escolhido é que os coeficientes das diagonais (que são requisitados frequentemente pelos métodos iterativos convencionais) são acessados diretamente, sem necessidade de algoritmos de busca. Logo, a escolha do MCSR para armazenar

as matrizes culminou em menores tempos de processamento dos métodos.

Para representar uma matriz quadrada (nxn), esparsa, com m elementos não-nulos, através do MCSR, segue-se o seguinte procedimento:

1. Os dois vetores (coeficientes e índices) são divididos em 3 setores: o primeiro contém n elementos; o segundo possui tamanho unitário; e o último, tamanho igual a $m-n$.
2. Constrói-se o vetor de elementos da seguinte maneira:
   - No primeiro setor são armazenados os valores das diagonais, de forma ordenada. Logo, a diagonal $a_{(0,0)}$ situa-se na posição 0, enquanto que a diagonal $a_{(n-1,n-1)}$ situa-se no endereço $n-1$.
   - O segundo setor do vetor de coeficientes armazena o valor 0 (zero).
   - O terceiro setor armazena os coeficientes não-nulos restantes da matriz (fora das diagonais), sendo que estes são armazenados na ordem que eles aparecem na matriz (seguindo o método empregado no CSR tradicional).
3. Constrói-se o vetor de índices auxiliares:
   - No primeiro setor, é armazenado em cada posição $i = 0..n-1$, a posição no terceiro setor do vetor de coeficientes em que o primeiro elemento da linha i aparece.
   - No segundo setor armazena-se o valor $m+1$.
   - Em cada posição do terceiro setor armazena-se a coluna onde está localizado o elemento de mesma posição no vetor de coeficientes.

A figura 7 ilustra tais procedimentos e o armazenamento segundo o método MCSR.

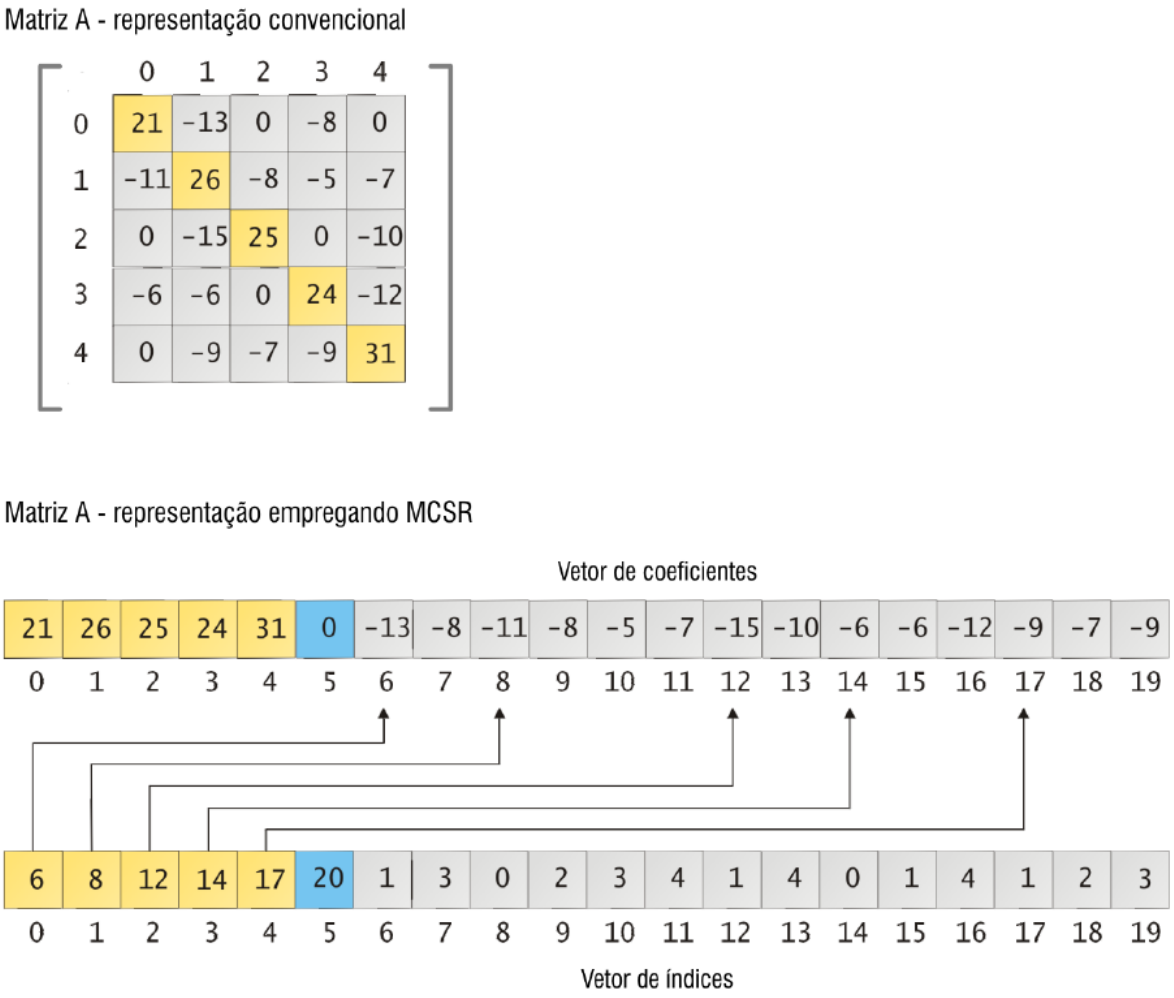


Figura 7: Esquema de armazenamento de matrizes esparsas MCSR

2.4.2 Sistema de *script* para configuração

Cada vez mais os sistemas e linguagens de *script* se tornam ferramentas fundamentais no desenvolvimento de *softwares*.

A idéia é desenvolver o núcleo numérico (que frequentemente requer grande esforço computacional) em linguagens de médio ou baixo nível, de forma a obter grande desempenho. E utilizar algum sistema de *script* para configurar, estender e controlar tal núcleo numérico.

Logo, as linguagens de *script* devem ser simples, legíveis (tanto para humanos quanto para computadores) e mais próximas da linguagem natural. Portanto, é comum utilizar linguagens de alto nível (Lua, Python, Ruby), linguagens de marcação (XML - *eXtensible Markup Language*) ou até novas linguagens (desenvolvidas a partir de alguma necessidade específica).

Assim, além da implementação das classes que compõem a estrutura capaz de resolver sistemas lineares através do *multigrid*, foi desenvolvido um sistema de *script* simples e fácil de usar. Podendo ser utilizado em inúmeras aplicações, não ficando restrito apenas ao *multigrid*.

O sistema desenvolvido se baseia em XML, que possibilita a criação de campos sem limitação. Então, é possível criar qualquer campo, nomeá-lo, e atribuir informações ao mesmo. As regras do sistema são:

- Cada arquivo de *script* deve ter um e apenas um campo raiz, que deve estar presente no início no arquivo e deve conter, direta ou indiretamente, todos os campos necessários;
- Cada campo deve ter um nome único;
- Cada campo deve possuir um e apenas um parâmetro (seja numérico ou textual);
- Cada campo pode conter um novo conjunto de campos;
- Um conjunto de campos inicia com o caractere "{" e termina com "}";
- Comentários podem ser adicionados ao arquivo de *script*, basta iniciar o comentário com "//" e a linha inteira é ignorada pelo sistema;
- Os arquivos de *script* podem ter qualquer nome e qualquer extensão, o que importa é o conteúdo.

Desta forma, um exemplo de arquivo *script* pode ser o seguinte:

```
Arquivo setup.multigrid
------------------------------------------------------------
//campo raiz, nome ‘‘multigrid’’, parâmetro textual ‘‘setup’’
multigrid setup
{
    tolerance 1.0e-8
    max_number_of_iterations 1000
    average_cells_per_group 8
    max_cells_on_coarsest 50
    cycle F

    //iterador com número de iterações fixo e igual a 5
    iterator fixed
    {
```

```
        iterations 5
    }
}
-------------------------------------------------------
```

de modo que a estrutura construída no momento que o sistema ler tal arquivo será

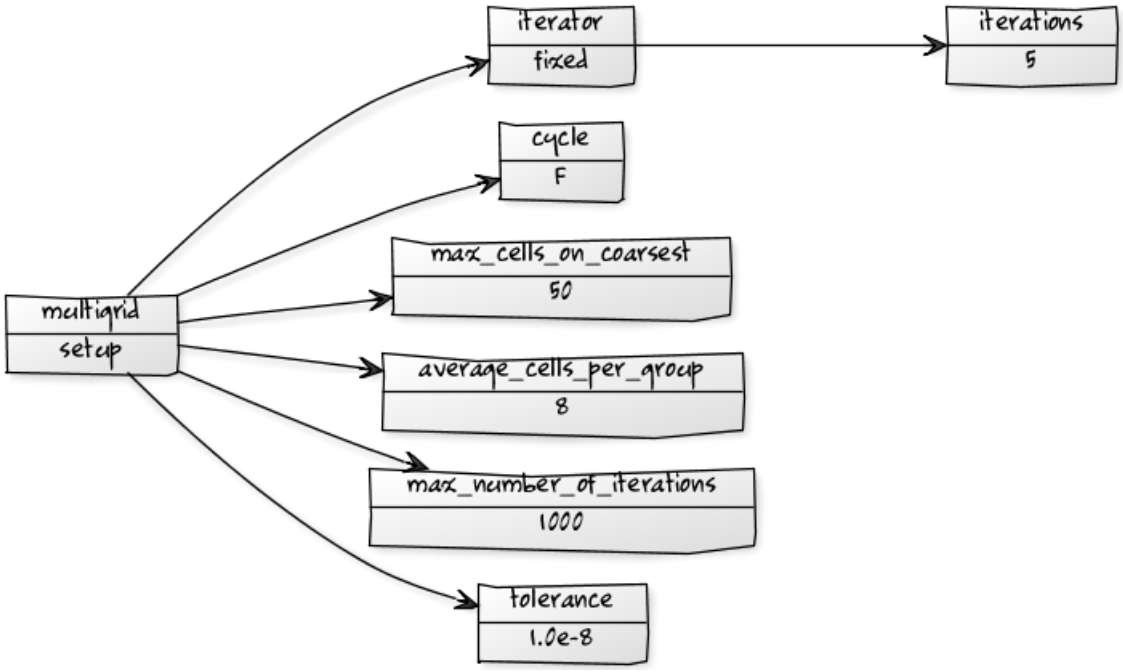


Figura 8: Estrutura construída a partir do arquivo setup.multigrid

As classes responsáveis por tal leitura são *ScriptFile* e *ScriptEntry*. A *ScriptEntry* representa um campo, logo, ela pode conter uma variável numérica ou uma textual, e ainda pode conter um conjunto de novas instâncias da própria classe. Deste modo, o campo raiz do arquivo é representado pela *ScriptEntry*.

Já a classe *ScripFile* funciona como um *parser*, ou seja, ela lê um arquivo com as informações e constrói a estrutura análoga a apresentada na figura 8. Desta forma, o sistema de *script* implementado é simples, fácil de usar e legível.

Padrão do arquivo de *script* para o *multigrid*

Visando configurar todos os parâmetros relativos a uma estrutura *multigrid*, foi definido um padrão para arquivos de *script* que seja reconhecido pela classe *Multigrid*.

Um arquivo de *script* utilizado para configurar o *multigrid* deve ter a seguinte estrutura:

- O campo raiz deve ter o nome *multigrid*;

- O campo raiz deve conter os seguintes campos:

    1. *tolerance*, com um parâmetro numérico que seja a tolerância para o critério de convergência global do *multigrid*;
    2. *max_number_of_iterations*, com um parâmetro numérico que seja a quantia máxima de vezes que deve ser realizado o ciclo *multigrid* na tentativa de obter uma solução convergida;
    3. *average_cells_per_group*, seguido de um parâmetro numérico que seja a quantia média de células que cada bloco deve conter em cada nova aglomeração;
    4. *max_cells_on_coarsest*, com um valor numérico que represente o número máximo de células que um nível deve ter para ser considerado o último da hierarquia;
    5. *cycle*, com um parâmetro textual indicando qual ciclo *multigrid* deve ser empregado;
    6. *iterator*, seguido de um parâmetro textual indicando qual iterador deve ser empregado na determinação da quantia de iterações do método iterativo convencional em cada malha.

Desta forma, o arquivo exemplo ilustrado na figura 8 serve como configuração do *multigrid*.

3 MÉTODOS PARA A DETERMINAÇÃO DA QUANTIA DE ITERAÇÕES NOS NÍVEIS

No momento que um nível de malha é visitado no decorrer de um ciclo *multigrid* é necessário que haja algum esquema para determinar quantas iterações do método iterativo devem ser efetuadas.

Todos os *iteradores* (forma como estes métodos serão referenciados no trabalho) são configurados através do arquivo de *script*. Tal configuração é realizada através da criação de um campo *iterator* com um parâmetro textual específico e com um conjunto de novos campos, relativos aos dados internos de cada iterador.

Neste capítulo serão apresentados alguns esquemas para realizar tal tarefa.

3.1 NÚMERO FIXO DE ITERAÇÕES

Este esquema é o mais simples de todos. Consiste em iterar, uma quantia fixa de vezes (configurada através do arquivo de *script*), em todas as malhas visitadas. Isto significa que o mesmo número de iterações será executado em cada nível.

Para escolher e configurar este esquema deve-se definir o parâmetro do campo *iterator* com a palavra-chave *fixed* e deve-se adicionar ao *iterator* um campo *iterations* com o número de iterações. Exemplo de configuração de um iterador fixo com número de iterações igual a 2:

```
------------------------------------------------------------
multigrid setup_fixed
{
    ...
    iterator fixed
    {
        iterations 2
    }
}
```

```
-----------------------------------------------------------
```

3.2 Número de Iterações Específico para cada Nível

Este esquema é semelhante ao anterior. A diferença se dá no fato de que utilizando este método é possível fixar uma quantia específica de iterações para cada malha através de um vetor de valores.

Desta forma, é possível estipular qualquer combinação de valores. Supondo que se deseje iterar 10 vezes na malha mais fina e reduzir em 2 esta quantia para cada nível avançado, a configuração através do arquivo de *script* ficaria com a seguinte estrutura:

```
-----------------------------------------------------------
multigrid setup_vector
{
    ...
    iterator vector
    {
        vector_size 4
        iterations_on_level_0 10
        iterations_on_level_1 8
        iterations_on_level_2 6
        iterations_on_level_3 4
    }
}
-----------------------------------------------------------
```

3.3 Critério: Redução Relativa do Resíduo

Este esquema avalia quanto o resíduo já foi reduzido em relação ao obtido no início das iterações da malha em questão. Tal valor é computado de forma relativa. De forma que, quando o resíduo atual representar menos do que certa porcentagem do resíduo inicial, dever-se-á interromper as iterações e realizar a troca de nível.

Tal procedimento é bem simples, e pode ser configurado através do arquivo de *script* com a seguinte estrutura para o campo *iterator* (supondo que deva-se trocar de nível quando o último resíduo for inferior a 10% do primeiro resíduo):

```
-----------------------------------------------------------
multigrid setup_relative
```

```
{
    ...
    iterator relative
    {
        alpha 0.1
        max_iterations 10
    }
}
-------------------------------------------------------
```

3.4 Critério: Taxa de Redução do Resíduo

Neste procedimento avalia-se a taxa de redução do resíduo ao longo das iterações, já que no momento que o método iterativo perde eficiência tal taxa torna-se muito pequena (inferior a determinada tolerância). Este fenômeno é ilustrado na figura 2. Logo, o critério para continuar iterando em certo nível de malha é avaliar se a taxa de redução do resíduo é superior a uma tolerância pré fixada.

A idéia de tal procedimento é que, como os sistemas de equações para cada nível de malha exceto o original são redefinidos a cada ciclo *multigrid*, a taxa de redução dos erros deve voltar a adquirir valores elevados a cada novo ciclo. Porém, o que foi verificado é que tais taxas elevam-se pouco e logo assumem valores inferiores aos anteriores. De modo que este procedimento degenera rapidamente ao caso em que se aplica uma única iteração para cada nível de malha. Porém, consumindo mais tempo, devido às checagens envolvidas.

Portanto, este esquema não foi formalmente implementado. Apenas foram calculadas as taxas de redução dos resíduos ao longo do procedimento de solução e a partir dos valores observados concluiu-se que o esquema não trazia nenhuma melhoria no desempenho, pelos motivos citados acima.

4 RESULTADOS

4.1 PROBLEMA BASE

A fim de avaliar os esquemas de determinação da quantidade de iteração nos níveis, foi utilizado um problema base. Este consiste em resolver a difusão de uma propriedade arbitrária, em regime permanente, em um domínio tridimensional e com geração.

O problema é determinado pela equação (4.1)

$$-\nabla^2\phi = F(x,y,z) \tag{4.1}$$

em que $F(x,y,z)$ é o termo fonte, dado por

$$F(x,y,z) = 12\pi^2 sin(2\pi x)sin(2\pi y)sin(2\pi z) \tag{4.2}$$

logo, a solução analítica do problema é

$$\phi(x,y,z) = sin(2\pi x)sin(2\pi y)sin(2\pi z) \tag{4.3}$$

O domínio do problema é um cubo unitário em cujas fronteiras são prescritas condições de Dirichlet com valores iguais aos obtidos pela solução analítica ϕ. Este problema, tal como descrito aqui, foi retirado de [7].

Para se obter o sistema linear oriundo da discretização do problema foi escolhido o método EbFVM (*Element based Finite Volume Method*) [4] [8] [9]. Este método utiliza elementos de diversas geometrias permitindo malhas com domínio complexo, e ao mesmo tempo, integra em volumes finitos as equações governantes, garantindo assim caráter conservativo.

Sobre o domínio deve-se gerar uma malha para realizar a discretização das equações e construir o sistema linear referente ao problema. O tipo de elemento mais adequado para gerar uma malha sobre o domínio cúbico em questão é o hexaedro. A figura 9 ilustra um elemento hexaédrico, enquanto que a figura 10 apresenta uma malha construída com tais elementos.

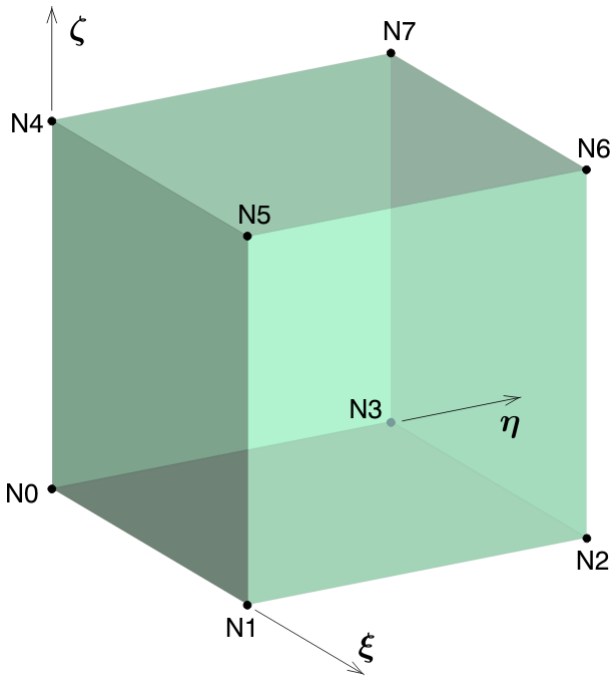


Figura 9: Elemento hexaédrico, retirado de [3]

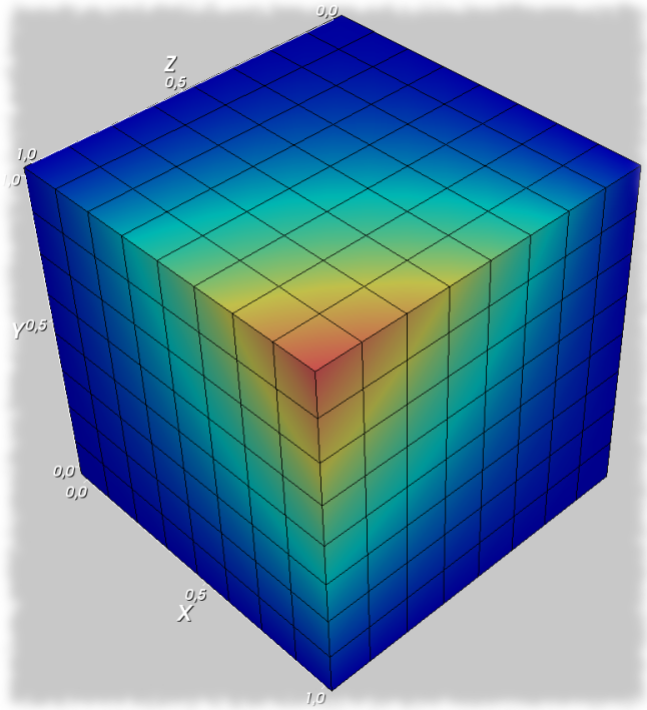


Figura 10: Malha de elementos hexaédricos

Resumindo, o problema base para se obter sistemas lineares é de difusão pura, em regime permanente, com geração, e utilizando o método EbFVM para discretizar as equações.

Contudo, o nível de refino da malha permaneceu variável. Desta forma, é possível obter

o desempenho de cada método em relação ao tamanho do sistema linear.

4.2 Testes com Número Fixo de Iterações

Com o intuito de avaliar o método que utiliza um número fixo de iterações definiu-se três configurações diferentes para o *multigrid*. Cada configuração foi testada utilizando 1, 2, 4, 8, 16, 32 e 64 iterações.

Não houve necessidade de gerar malhas tão grandes quanto possível (usual quando se deseja medir e comparar tempos de métodos). Já que sempre se comparou o *multigrid* com ele próprio, não havendo outro método com comportamento diferente para ser avaliado em situações mais críticas. Também foi visto, com a obtenção dos primeiros resultados, que as malhas utilizadas já são suficientes para se estimar o comportamento dos casos para malhas maiores, ou seja, os resultados apresentam um comportamento estável a partir de determinado tamanho de malha.

Logo, para cada configuração e número de iterações foram utilizadas as malhas 8x8x8, 14x14x14, 20x20x20, 26x26x26, 32x32x32 que geram, respectivamente, sistemas lineares com 729, 3375, 9261, 19683 e 35937 equações. Logo, para cada configuração proposta há 35 casos diferentes. A tabela abaixo ilustra tais casos.

Tabela 1: Combinações entre malha e quantidade de iterações por nível

	Iterações por nível						
Malha	1	2	4	8	16	32	64
8x8x8 (729)	1	2	3	4	5	6	7
14x14x14 (3375)	8	9	10	11	12	13	14
20x20x20 (9261)	15	16	17	18	19	20	21
26x26x26 (19683)	22	23	24	25	26	27	28
32x32x32 (35937)	29	30	31	32	33	34	35

A fim de se obter valores confiáveis da taxa de convergência (tempo de cálculo para se obter a solução convergida), cada combinação de número de iterações e malha é executado 5 vezes, em seguida a média e o desvio padrão da taxa de convergência desta amostra são obtidos. Caso o desvio padrão dividido pela média resulte em um valor maior do que determinada tolerância (foi utilizada a tolerância 0,01) a amostra em questão é desprezada e o procedimento é repetido. Desta forma, garante-se que o desvio padrão relativo seja baixo (menor do que a tolerância), de tal maneira que a média da taxa de convergência seja realmente representativa da amostra em questão.

A máquina utilizada para efetuar as simulações e obter os resultados é um Intel Core 2 Quad Q6600 2,4GHz com 4GB de mémoria RAM, utilizando o sistema operacional Windows 7 Ultimate 64 bits e o compilador Visual C++ 2008 Express.

As configurações testadas foram:

- Configuração 1:
  - Número médio de células em cada bloco: 8;
  - Número máximo de células na malha mais grosseira: 10;
  - Ciclo: F.
- Configuração 2:
  - Número médio de células em cada bloco: 5;
  - Número máximo de células na malha mais grosseira: 30;
  - Ciclo: V.
- Configuração 3:
  - Número médio de células em cada bloco: 11;
  - Número máximo de células na malha mais grosseira: 50;
  - Ciclo: W.

Após terem sido resolvidos todos os casos possíveis relativos a cada configuração, os resultados obtidos foram compilados nos gráficos das figuras 11 a 16. Para cada configuração foram gerados dois gráficos distintos: no primeiro cada linha representa uma quantidade de iterações, desta forma pode-se avaliar o comportamento do *multigrid* com o tamanho da malha; e no segundo cada linha representa uma malha, podendo ser avaliado o comportamento do *multigrid* com a quantidade de iterações.

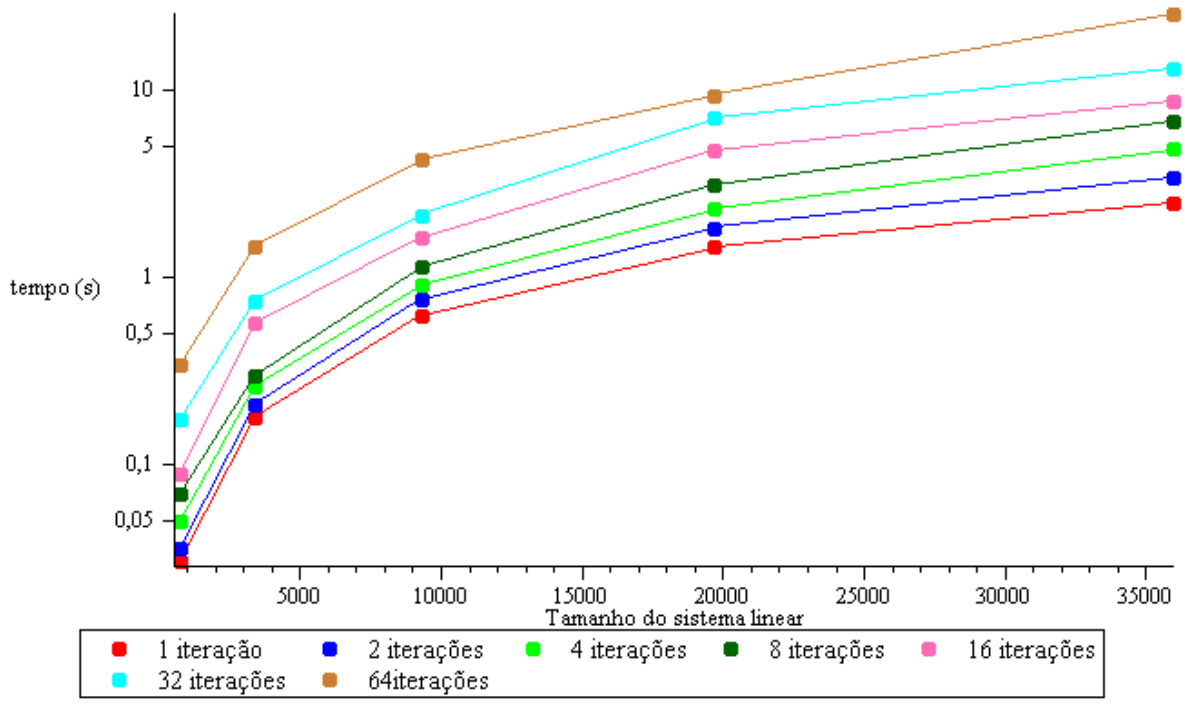


Figura 11: Configuração 1: Variação do tempo de computação em relação ao número de iterações

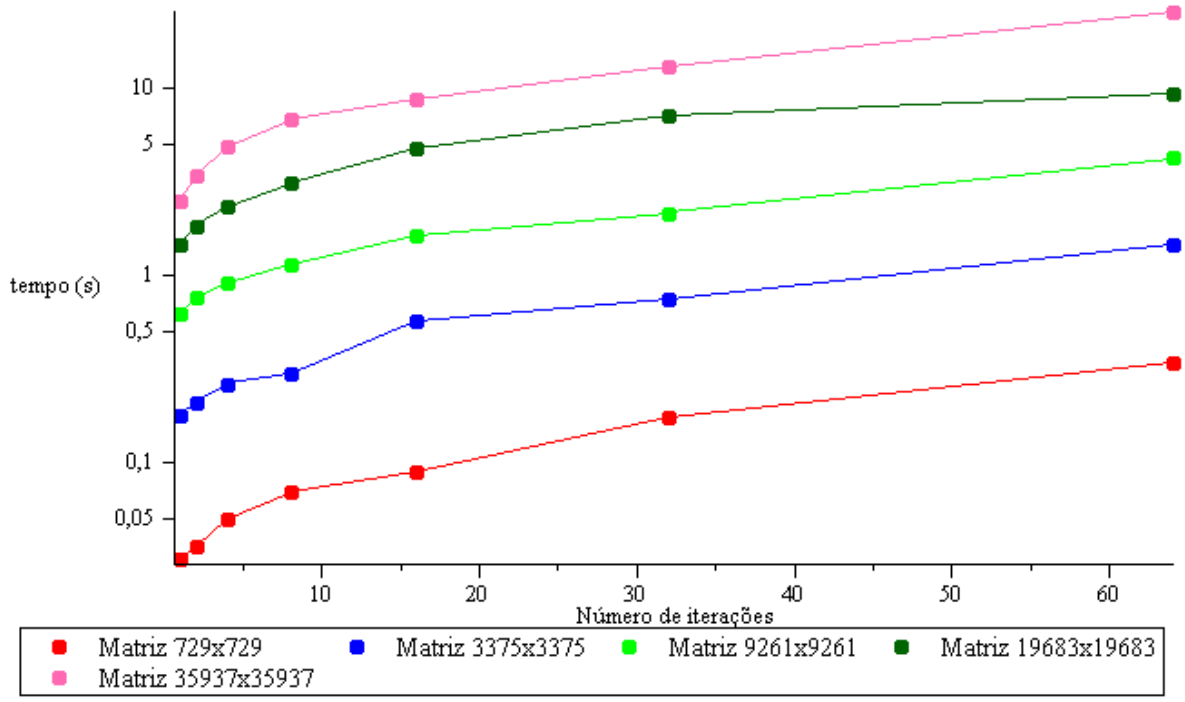


Figura 12: Configuração 1: Variação do tempo de computação para cada matriz

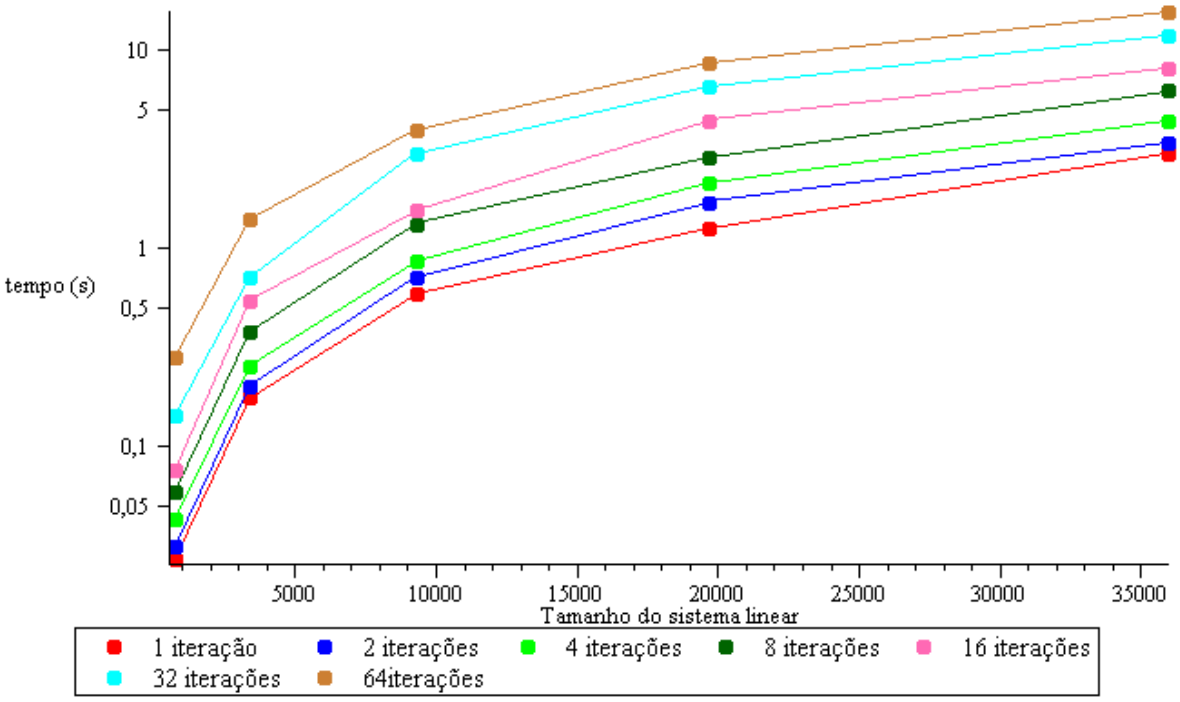


Figura 13: Configuração 2: Variação do tempo de computação em relação ao número de iterações

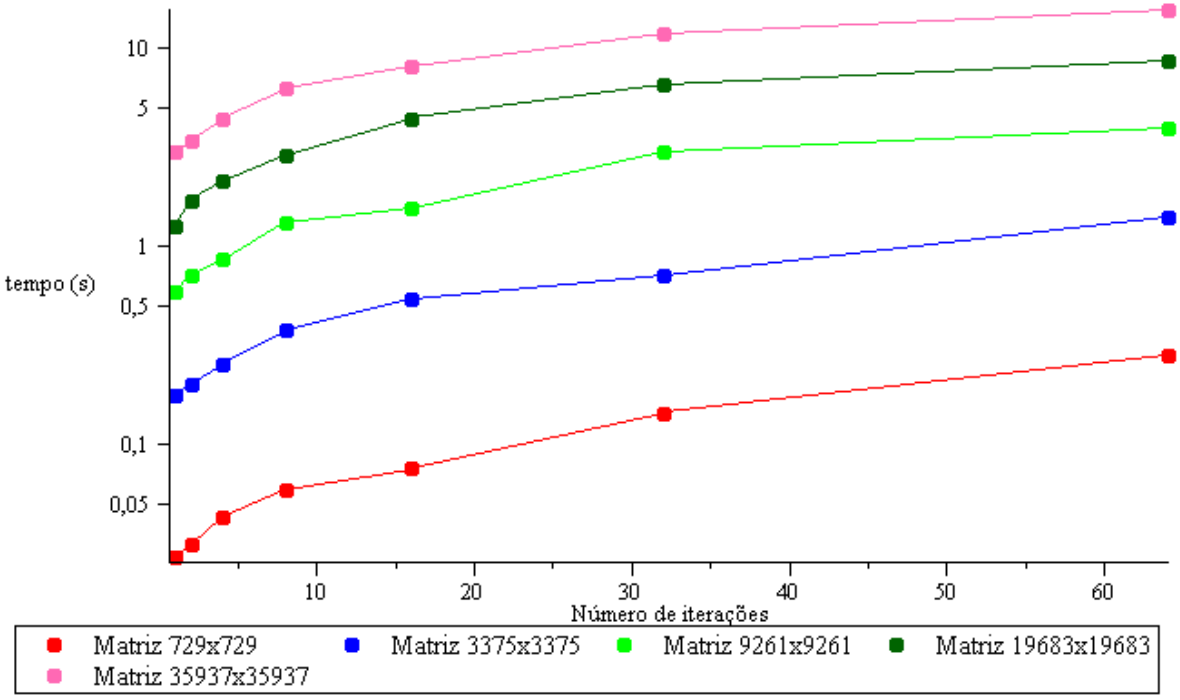


Figura 14: Configuração 2: Variação do tempo de computação para cada matriz

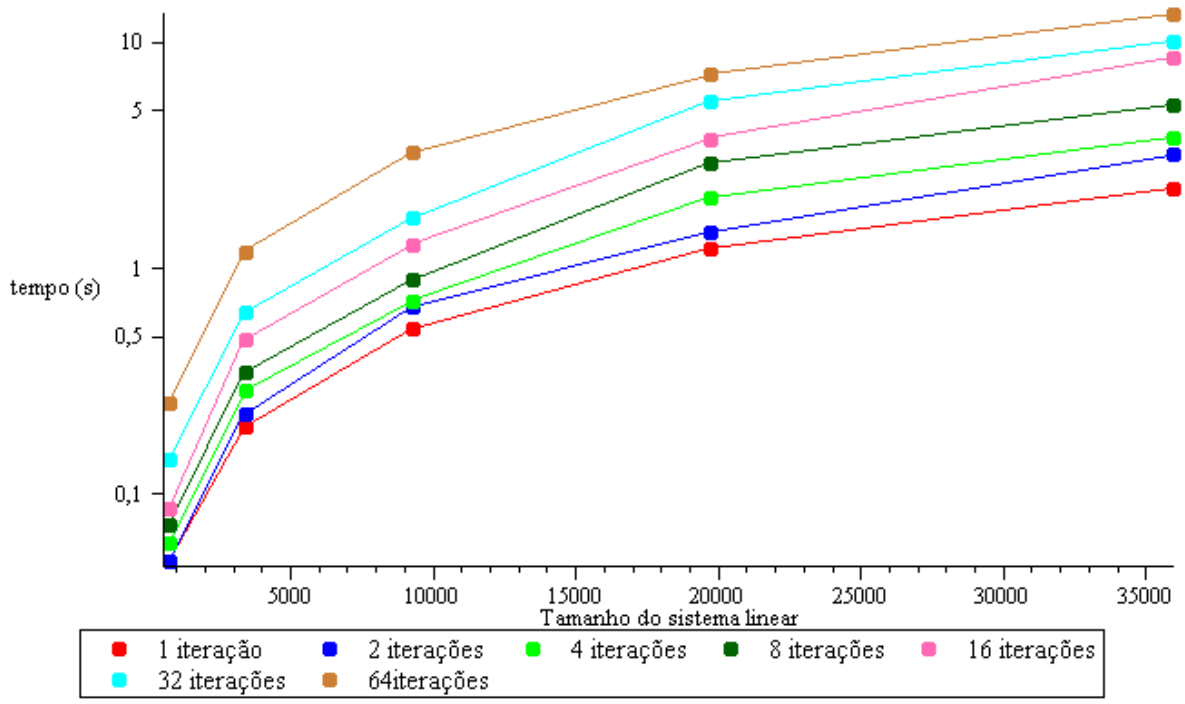


Figura 15: Configuração 3: Variação do tempo de computação em relação ao número de iterações

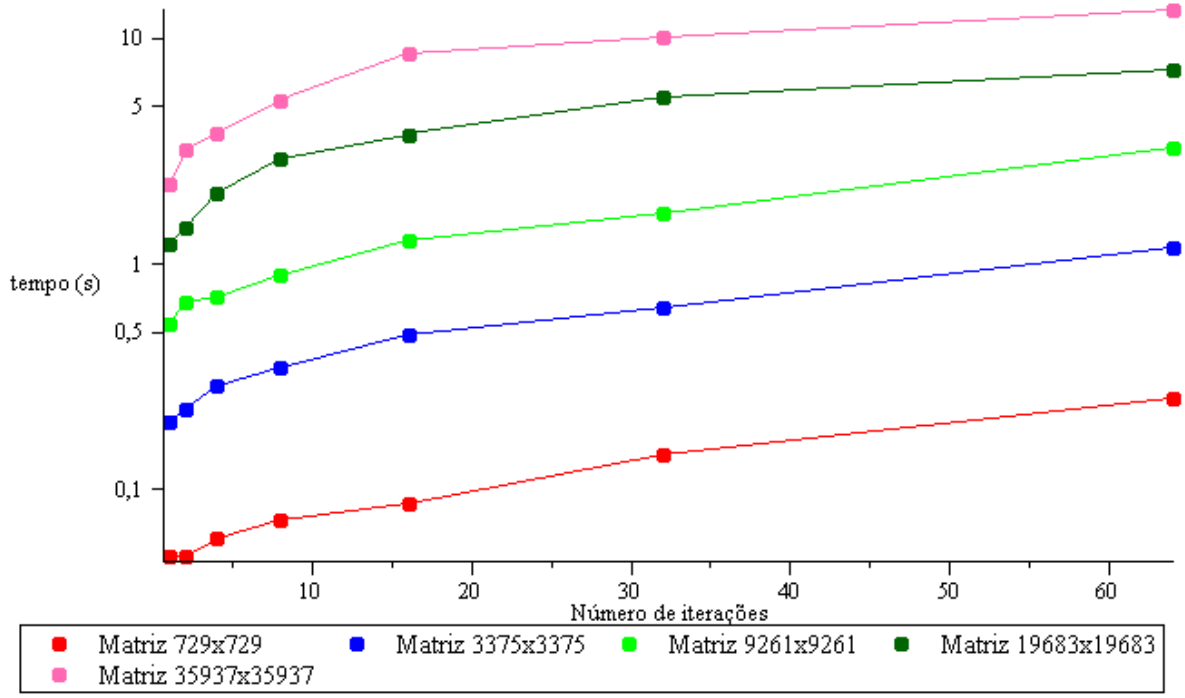


Figura 16: Configuração 3: Variação do tempo de computação para cada matriz

Portanto, como pode ser visto nestes resultados, o comportamento do tamanho da matriz e da quantidade de iterações por malha com o tempo de cálculo independem dos demais

parâmetros *multigrid* (quantidade média de células aglomeradas em um mesmo bloco; número máximo de células na malha mais grosseira; ciclo utilizado).

Nos gráficos das iterações nota-se claramente que, se for fixada uma quantidade de iterações e o problema for resolvido com várias malhas diferentes, quanto maior a malha, mais tempo será solicitado para que o *multigrid* atinja a solução convergida, como esperado de qualquer método de solução. Ao passo que, nos gráficos das matrizes, nota-se que, para qualquer matriz, se o número de iterações por nível aumentar, o tempo despendido pelo método também aumentará.

Logo, a quantidade de iterações ideal quando é permitido apenas configurar um número fixo de iterações é sempre um.

Para avaliar os outros métodos será utilizado apenas a configuração 1, já que nos gráficos já apresentados foi verificado que o comportamento do *multigrid* não é afetado pelas diferentes configurações. Adicionalmente, os testes dos outros métodos serão comparados com os resultados obtidos quando a quantidade de iteração nos níveis foi fixada em um, já que tal caso foi sempre o mais eficiente.

4.3 Testes com Número de Iterações Específico para cada Nível

O método que fornece um número específico de iterações para cada nível permite que qualquer perfil de valores sejam utilizados. Observa-se que no caso de iterações fixas a melhor situação é a de menos iterações possíveis, logo, os perfis configurados para este método estarão sempre próximos de tal condição (iterações constantes e iguais a um).

O perfil mais simples que se possa utilizar (excluindo o caso constante, que já foi testado) é o linear. Por isso, o primeiro teste para tentar superar o desempenho do método anterior é configurar o *multigrid* para utilizar um perfil linear. As quantidades de iterações configuradas foram 1, 2, 3, 4, 5, ... da malha mais fina à mais grosseira, respectivamente. O arquivo de *script* está ilustrado abaixo.

```
------------------------------------------------------------
multigrid setup_linear
{
    tolerance 1.0e-5
    max_number_of_iterations 50
    average_cells_per_group 8
    max_cells_on_coarsest 10
    cycle F
    iterator vector
```

```
    {
        vector_size 6
        iterations_on_grid_0 1
        iterations_on_grid_1 2
        iterations_on_grid_2 3
        iterations_on_grid_3 4
        iterations_on_grid_4 5
        iterations_on_grid_5 6
    }
}
-------------------------------------------------------------
```

Após o teste com o perfil linear, resolveu-se testar um caso com o perfil constante e igual a um (melhor caso no método anterior), exceto na malha mais fina, cuja quantidade de iterações foi configurada em 2, e depois em 4. O arquivo de *script* está ilustrado abaixo (a incógnita "X" presente no arquivo simboliza a quantidade de iterações na malha mais fina).

```
-------------------------------------------------------------
multigrid setup_X_on_finest
{
    ...
    iterator vector
    {
        vector_size 6
        iterations_on_grid_0 X
        iterations_on_grid_1 1
        iterations_on_grid_2 1
        iterations_on_grid_3 1
        iterations_on_grid_4 1
        iterations_on_grid_5 1
    }
}
-------------------------------------------------------------
```

Os resultados dos perfis testados estão ilustrados na figura abaixo.

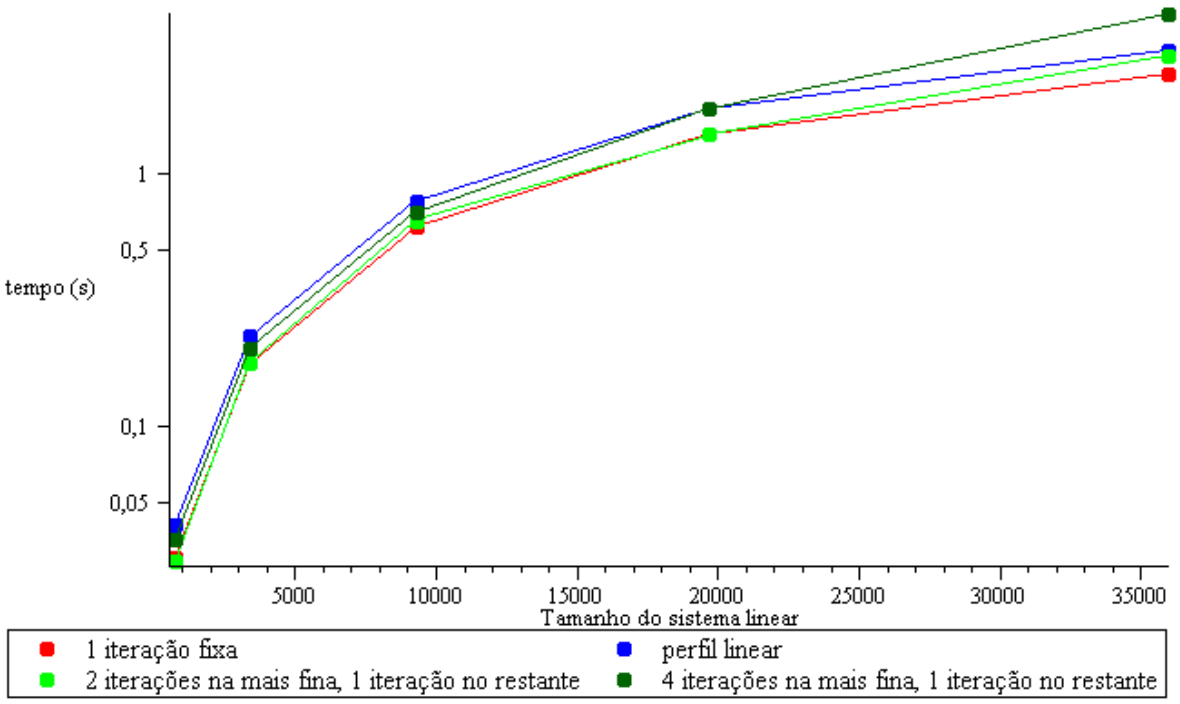


Figura 17: Comparação de diferentes perfis de iterações

Como pode ser visto nos gráficos, tanto a tentativa de aumentar a quantia de iterações nas malhas mais grosseiras, quanto a de aumentar na malha mais fina sofreram uma piora no desempenho quando comparadas com o perfil constante (fixo) e igual a uma iteração por malha. Logo, esta última configuração parece ser a que fornece melhor eficiência.

4.4 Testes com o Método da Redução Relativa do Resíduo

Vários testes com várias configurações diferentes de tal método foram conduzidos, todos geraram tempos sensivelmente mais elevados do que a situação de iterações fixas iguais a um. Este comportamento era esperado!

Para realizar o método em questão é necessário que em cada iteração do método convencional seja calculado o resíduo do sistema linear e sejam feitas algumas verificações. Tais procedimentos consomem tempo e atrasam o processo.

A vantagem de tal método é que teoricamente ele é mais robusto do que os outros, ou seja, dificilmente diverge. Porém, como não foi possível elaborar um caso ou uma situação que causasse a divergência do *multigrid* não foi possível verificar tal propriedade.

5 CONCLUSÕES

5.1 CONSIDERAÇÕES FINAIS

Os testes para o método de iterações fixas e para o que usa uma iteração específica para cada nível mostraram que a melhor configuração para o problema simulado é um perfil de iterações constante e igual a um.

O método de redução relativa do resíduo foi implementado devido a sua aparente robustez. Porém, como não se conseguiu elaborar um problema ou caso que divergisse utilizando os métodos anteriores, não foi possível verificar tal propriedade. O método de redução relativa do resíduo foi rodado várias vezes, com várias combinações diferentes de parâmetros, e todas geraram resultados piores do que o caso com perfil de iterações constante e igual a um.

Portanto, recomenda-se o uso do perfil unitário para as iterações. Porém, se em algum caso o *multigrid* divergir, então pode-se tentar aplicar o método de redução relativa do resíduo.

O trabalho realizado é bastante complexo, já que existem inúmeros perfis de iterações distintos que podem ser configurados no *multigrid* e cada um pode gerar um tipo de comportamento diferente. Ainda, cada problema físico ou conjunto de parâmetros pode levar a um comportamento diferente do *multigrid*.

Logo, os resultados obtidos são específicos para os casos testados.

Porém, algumas contribuições foram obtidas.

Neste trabalho nota-se um desempenho superior no método de iterações fixas e iguais a um em relação aos outros métodos, logo, este é um bom ponto de partida para outros casos.

Também, através do uso de diversas malhas, eliminou-se o fator *malha* na análise do comportamento do *multigrid*, já que para todas o comportamento do *multigrid* permaneceu semelhante.

Outra contribuição foi compilar num mesmo documento diversos métodos distintos de obtenção do número de iterações nos níveis, já que tais métodos são pouco explorados na

literatura.

5.2 Sugestões para Trabalhos Futuros

- Avaliar o comportamento do *multigrid* com diferentes tipos de problemas físicos;
- Verificar que conjuntos de parâmetros de aglomeração (quantidade de células a serem aglomeradas em um mesmo bloco; número máximo de células na malha mais grosseira) geram resultados melhores;
- Implementar o ciclo flexível apresentado em [10] e comparar com os ciclos clássicos (V, W e F);
- Paralelizar o *multigrid* para obter melhor desempenho;
- Utilizar a GPU (*Graphics Processing Unit*) para realizar os cálculos do *multigrid* a fim de obter melhor desempenho;
- Avaliar o *multigrid* utilizando diferentes métodos iterativos para realizar as iterações nos níveis.

REFERÊNCIAS

1 HUTCHINSON, B. R.; RAITHBY, G. D. A multigrid method based on the additive correction strategy. *Numerical Heat Transfer*, v. 9, p. 511–537, 1986.

2 KELLER, S. C. *O Método Multigrid de Correções Aditivas para a Solução Numérica Acoplada das Equações de Navier-Stokes com Malhas Não Estruturadas*. Tese (Doutorado) — Universidade Federal de Santa Catarina, 2007.

3 MALISKA, C. R. et al. *Relatório Técnico nº 3 - Projeto Simulação de Reservatórios de Petróleo pelo Método EbFVM com Solver Multigrid (SimReP)*. [S.l.], 2009.

4 MALISKA, C. R. *Transferência de Calor e Mecânica dos Fluidos Computacional*. 2ª. ed. [S.l.]: LTC - Livros Técnicos e Científicos Editora, 2004.

5 CFD ONLINE. *Iterative Methods*. 2008. Disponível em: <http://www.cfd-online.com/Wiki/Iterative_methods>.

6 SAAD, Y. *Iterative Methods for Sparse Linear Systems*. [S.l.]: PWS Publishing Company, Boston, EUA, 2000.

7 MALISKA, C. R. et al. *Relatório Técnico nº 4 - Projeto Simulação de Reservatórios de Petróleo pelo Método EbFVM com Solver Multigrid (SimReP)*. [S.l.], 2010.

8 HURTADO, F. S. V. *Uma Formulação de Volumes Finitos Baseada em Elementos para a Simulação do Deslocamento Bifásico Imiscível em Meios Porosos*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, 2005.

9 CORDAZZO, J. *Simulação de Reservatórios de Petróleo Utilizando o Método EbFVM e Multigrid Algébrico*. Tese (Doutorado) — Universidade Federal de Santa Catarina, 2006.

10 FLUENT INCORPORATED. *Fluent 6.1 Documentation - Seção 24.5.3 Algebraic Multigrid (AMG)*. novembro 2010. Disponível em: <http://jullio.pe.kr/fluent6.1/help/html/ug/node837.htm>.