JEAN ANTONIO RIBEIRO

UMA AVALIAÇÃO DE IMPLEMENTAÇÕES VIA OPENMP E PTHREADS DE DUAS HEURÍSTICAS PARA REDUÇÕES DE LARGURA DE BANDA DE MATRIZES

LAVRAS – MG

2018

JEAN ANTONIO RIBEIRO

UMA AVALIAÇÃO DE IMPLEMENTAÇÕES VIA OPENMP E PTHREADS DE DUAS HEURÍSTICAS PARA REDUÇÕES DE LARGURA DE BANDA DE MATRIZES

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Inteligência Computacional e Processamento Digital de Imagens, para a obtenção do título de Mestre.

Prof. Dr. Sanderson L. Gonzaga de Oliveira

Orientador

LAVRAS – MG

2018

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo(a) próprio(a) autor(a).

Ribeiro, Jean Antonio.
Uma avaliação de implementações via OpenMP e Pthreads de duas heurísticas para reduções de largura de banda de matrizes / Jean Antonio Ribeiro. - 2018.
81 p. : il.

Orientador(a): Sanderson Lincohn Gonzaga de Oliveira.
.
Dissertação (mestrado acadêmico) - Universidade Federal de Lavras, 2018.
Bibliografia.

1. Redução de largura de banda. 2. Heurísticas. 3. Método dos Gradientes Conjugados precondicionado. I. Gonzaga de Oliveira, Sanderson Lincohn. . II. Título.

JEAN ANTONIO RIBEIRO

UMA AVALIAÇÃO DE IMPLEMENTAÇÕES VIA OPENMP E PTHREADS DE DUAS HEURÍSTICAS PARA REDUÇÕES DE LARGURA DE BANDA DE MATRIZES

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Inteligência Computacional e Processamento Digital de Imagens, para a obtenção do título de Mestre.

APROVADA em 4 de Maio de 2018.

Prof. Carla Osthoff Ferreira de Barros	LNCC
Prof. Mauricio Kischinhevsky	CEFET/RJ

Prof. Dr. Sanderson L. Gonzaga de Oliveira
Orientador

LAVRAS – MG
2018

Aos meus amados pais, grandes e eternos colaboradores na minha vida, José e Ilda, dedico.

AGRADECIMENTOS

Acima de tudo, a Deus, por mais esta conquista. Também agradeço às pessoas mais importantes da minha vida, os meus pais. Obrigado, José e Ilda, por terem me apoiado nessa longa caminhada.

À Universidade Federal de Lavras (UFLA) e ao Departamento de Ciência da Computação (DCC), pela oportunidade concedida para realização do mestrado.

Ao meu orientador, Professor Sanderson, por todos os ensinamentos e conselhos dados. Ainda, agradeço a todos os professores e funcionários do DCC-UFLA.

A todos os meus amigos que participaram desta caminhada. Em especial, aos amigos Carlos Henrique, Gabriel Rodrigues e Libério Martins, pelos conselhos e produtivas conversas diárias.

Finalmente, ao apoio financeiro da Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG).

Dentro da noite que me rodeia
Negra como um poço de lado a lado
Eu agradeço aos deuses que existem
Por minha alma indomável
Nas garras cruéis das circunstâncias
Eu não tremo ou me desespero
Sob os duros golpes da sorte
Minha cabeça sangra, mas não se curva
Além deste lugar de raiva e choro
Paira somente o horror da sombra
E ainda assim, a ameaça do tempo
Vai me encontrar e deve me achar destemido
Não importa se o portão é estreito
Não importa o tamanho do castigo
Eu sou dono do meu destino
Eu sou o capitão de minha alma
(Nelson Mandela)

RESUMO

Neste trabalho, são descritos dois algoritmos paralelos, em arquiteturas *multicore*, para solucionar os problemas de reduções de largura de banda e de *profile* de matrizes simétricas e esparsas. Para isso, as linhas e colunas da matriz de coeficientes são permutadas, deixando-as com uma estrutura compacta, de modo que os coeficientes não nulos estejam próximos à diagonal principal. Os novos algoritmos paralelos foram comparados a algoritmos disponíveis na biblioteca HSL e nos *softwares* Octave e Matlab. Nas simulações, esses algoritmos são aplicados como um pré-processamento na resolução de sistemas de equações lineares. Mais especificamente, os sistemas de equações lineares serão resolvidos pelo método dos gradientes conjugados precondicionado pela fatoração incompleta de Cholesky, em que a matriz de coeficientes é simétrica e positiva definida. Uma localidade de memória adequada contribui para a eficiência do método dos gradientes conjugados precondicionado, e essa característica pode ser obtida por reordenações de linhas e colunas da matriz de coeficientes. As bibliotecas utilizadas foram OpenMP e Pthreads. A qualidade da solução entre os algoritmos sequenciais e paralelos se manteve para a maioria das instâncias testadas; contudo o tempo computacional foi melhor com a biblioteca OpenMP do que com a biblioteca Pthreads, especialmente quando foram utilizadas duas *threads* nas execuções dos algoritmos. Todavia os experimentos computacionais, realizados neste trabalho, mostram que heurísticas paralelas, para reduções de largura de banda, têm potencial de acelerar o método paralelo dos gradientes conjugados precondicionado.

Palavras-chave: Redução de largura de banda. Redução de *profile*. Reordenação. Busca em largura. *Reverse Cuthill-McKee*. KP-band. Heurísticas. Algoritmos paralelos. *Pthreads*. OpenMP. Método dos gradientes conjugados precondicionado. Fatoração incompleta de Cholesky.

ABSTRACT

In this work, two parallel algorithms are described in multicore architectures to solve the bandwidth and profile reduction problems of symmetric and sparse matrices. For this, the rows and columns of the coefficient matrix are permuted, leaving it with a compact structure so that the nonzero coefficients are close to the main diagonal. The new parallel algorithms were compared to algorithms available in the HSL library and in the Octave and Matlab softwares. In the simulations, these algorithms are applied as a preprocessing in the resolution of systems of linear equations. More specifically, systems of linear equations will be solved by the conjugate gradients method with incomplete Cholesky factorization preconditioning, where the coefficient matrix is symmetric and positive definite. A suitable memory location contributes to the efficiency of the preconditioned conjugate gradient method, and this characteristic can be obtained by rearrangements of rows and columns of the coefficient matrix. The libraries used were OpenMP and Pthreads. The quality of the solution for sequential and parallel algorithms was maintained for most of the tested instances. However, computational time was better with the OpenMP library than with the Pthreads library, especially when two threads were used in the execution of the algorithms. Nevertheless, in this work, it is shown that parallel heuristics for bandwidth reductions have the potential to accelerate the parallel method of the preconditioned conjugate gradients. However, the computational experiments carried out in this work showed that parallel heuristics for bandwidth reductions have the potential to accelerate the parallel method of preconditioned conjugate gradients.

Keywords: Bandwidth Reduction. Profile Reduction. Parallel Computing. Reordering. Breadth-First Search Reverse Cuthill-McKee. KP-band. Heuristics. *Pthreads*. OpenMP. Parallel Algorithms. Preconditioned Conjugate Gradient Method. Incomplete Cholesky Factorization.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

CM	*Cuthill-McKee*
RCM-GL	*Cuthill-McKee* reverso iniciando por um vértice pseudo-periférico pelo algoritmo de George e Liu (1979)
CPU	Unidade Central de Processamento
CSC	*Column Sparse Symmetric*
CSR	*Compressed Sparse Row*
EDP	Equação Diferencial Parcial
FAPEMIG	Fundação de Amparo à Pesquisa do Estado de Minas Gerais
GL	Algoritmo de George-Liu
GMRES	Método Iterativo do Resíduo Mínimo Generalizado
GPHH-band	*Genetic Programming Hyper-Heuristic for Bandwidth Reduction*
HSL	*Harwell Subroutine Library*
IC	Precondicionamento por fatoração incompleta de Cholesky
IC(0)	Precondicionamento por fatoração incompleta de Cholesky com zero *fill-in*
ICCG	Método dos gradientes conjugados precondicionado pela fatoração incompleta de Cholesky
ILU	Fatoração Incompleta LU
ILUT	Fatoração Incompleta LU com regra de eliminação de elementos não-nulos
KP-band	Heurística gerada pela *Genetic Programming Hyper-Heuristic*
L-RCM	*Leveled Reverse Cuthill-McKee*
MEF	Método dos elementos finitos
MGC	Método dos gradientes conjugados
MGCP	Método dos gradientes conjugados precondicionado
MPI	Message Passing Interface
NS	Não especulativo
NS-RCM	Implementação não especulativa do RCM *Unordered*
OL-RCM	Implementação paralela e otimizada do método RCM nivelado
OpenMP	*Open Multi-Processing*
RBMC	*Reverse Block-Multicolor*
RCM	*Cuthill-McKee* reverso
SEL	Sistema de equações lineares
SpMV	Multiplicação de matrizes esparsas por vetores
SSS	*Skyline*
UFLA	Universidade Federal de Lavras

SUMÁRIO

1 INTRODUÇÃO

Uma variedade de problemas científicos, de engenharia e industriais, são descritos por equações diferenciais parciais (EDPs). Para que essas equações sejam tratadas computacionalmente, elas precisam ser discretizadas. Na discretização de EDPs, são gerados sistemas de equações lineares (SELs). As matrizes contidas nesses SELs são de grande porte e esparsas. Tais sistemas são provenientes da utilização de métodos como os dos elementos finitos, das diferenças finitas e dos volumes finitos (OLIVEIRA; CHAGAS, 2014, p. 3-12).

Para a resolução de um sistema de equações lineares, podem ser aplicados tanto métodos diretos como iterativos. Métodos diretos encontram a solução exata do sistema de equações lineares, a menos de erros de arredondamento, em um número finito de passos. Por outro lado, a partir de um valor inicial, métodos iterativos realizam sucessivas aproximações que convergem para a solução exata em seu limite. A eliminação de Gauss e a decomposição LU são exemplos de métodos diretos. Para métodos iterativos, o método mais popular é o método dos gradientes conjugados (MGC) (HESTENES; STIEFEL, 1952; LANCZOS, 1952), indicado para resolução de sistemas de equações lineares de grande porte, nas ocorrências em que a matriz dos coeficientes é simétrica e positiva definida.

Com o objetivo de reduzir o tempo de execução de métodos iterativos ou diretos, na resolução de sistemas de equações lineares, o reordenamento das linhas e colunas da matriz esparsa pode ser aplicado, antes de serem realizados os cálculos matriciais. Esse reordenamento é aplicado como um passo de pré-processamento na matriz de coeficientes. Há métodos que realizam esse reordenamento para reduzir a largura de banda ou o *profile* de matrizes. A redução de largura de banda de matrizes consiste em realizar permutações de linhas e colunas, deixando-as com uma estrutura compacta, de modo que os coeficientes não nulos estejam próximos à diagonal principal. Desse modo, a banda de uma matriz é definida como a banda diagonal que compreende as diagonais da matriz, contendo coeficientes não nulos mais distantes da diagonal principal em ambas as direções. Por sua vez, largura de banda de uma matriz simétrica é a quantidade de subdiagonais da matriz triangular inferior ou da superior (OLIVEIRA; CHAGAS, 2014, p. 9). O *profile* é definido como o somatório das larguras de banda de cada linha da matriz (OLIVEIRA; CHAGAS, 2014). A seguir, largura de banda e *profile* de matrizes simétricas são formalizados (OLIVEIRA; CHAGAS, 2014).

Definição 1. *Largura de banda: seja A uma matriz simétrica* $n \times n$*, com coeficientes* a_{ij}*, em que* $1 \leq i,\ j \leq n$*. A largura de banda da linha i é* $\beta_i\ (A) = i - min[(1 \leq j < i)\ a_{ij} \neq 0]$*. A largura*

de banda $\beta(A)$ é a maior distância de coeficiente não nulo da matriz triangular inferior até a diagonal principal, considerando-se todas as n linhas da matriz, ou seja, $\beta(A) = max[(1 \leq i \leq n)\ \ \beta_i(A)]$.

Definição 2. *Profile: seja A uma matriz simétrica, o profile de A é definido como $\Sigma_{i=1}^{n}\ \ \beta_i(A)$.*

Como exemplo, na Figura 1.1, tem-se a representação de uma matriz simétrica com largura de banda 5, correspondente à diferença entre linha 5 e coluna 0. O *profile* dessa matriz é 13, pois as diferenças entre os índices de cada linha e coluna são (1-0)+(2-0)+(3-1)+(4-1)+(5-0)=13. A diagonal principal é representada pela coloração cinza de tom claro e os índices das linhas e colunas da matriz são representados pela coloração cinza de tom escuro.

Figura 1.1 – Exemplo de matriz simétrica com largura de banda 5 e *profile* 13. Na primeira coluna e na primeira linha, de cor cinza com tom mais escuro, são representados os índices da matriz.

-	0	1	2	3	4	5
0	**5**	1	4	0	0	1
1	1	**6**	3	1	1	4
2	4	3	**0**	4	0	1
3	0	1	4	**1**	1	1
4	0	1	0	1	**3**	1
5	1	4	1	1	1	**5**

Fonte: Do autor (2018)

Os problemas de minimização de largura de banda e de *profile* são pertencentes à classe NP-Difícil (LIN; YUAN, 1994; PAPADIMITRIOU, 1976). Por causa da importância desses problemas, existe uma grande variedade de métodos sequenciais utilizados para reordenação de linhas e colunas de matrizes. Atualmente, com a utilização de processamento paralelo, pode-se obter custo de processamento muito menor das resoluções, especialmente quando os tamanhos dos problemas tratados são muito grandes. Nesse contexto, algumas implementações em paralelo já foram desenvolvidas com a utilização de bibliotecas para criação de algoritmos em arquiteturas *multicore*. Neste projeto, pretende-se projetar um ou mais algoritmos em arquiteturas *multicore*, que tenham custos de processamento mais baixos que os custos dos algoritmos existentes na bibliografia.

Será utilizado o MGC (HESTENES; STIEFEL, 1952; LANCZOS, 1952) em paralelo, para resolução de sistemas de equações lineares esparsos, formados por matrizes simétricas e positivas definidas. Embora o MGC seja eficiente na resolução de SELs, a convergência desse

método pode ser lenta nas ocorrências em que a matriz é mal condicionada. Neste caso, os cálculos matriciais podem ser prejudicados, ampliando o efeito de erros de arredondamento. Se a matriz de coeficientes é mal condicionada, então, o custo de processamento do MGC é dependente da utilização de precondicionadores adequados. O precondicionamento é utilizado, para melhorar as propriedades espectrais da matriz de coeficientes, consequentemente, diminuindo a quantidade de iterações do MGC. Em geral, ele é aplicado como um pré-processamento na matriz de coeficientes dos SELs. Exemplos de precondicionadores podem ser encontrados em Saad (2003). Neste trabalho, é utilizado o método dos gradientes conjugados precondicionado pela fatoração incompleta de Cholesky, descrito nas seções 2.2 e 2.3.

A seguir, nas seções 1.1 e 1.2, são apresentados, respectivamente, a justificativa e os objetivos deste trabalho. No Capítulo 2, é abordado o referencial teórico necessário para a compreensão do projeto. No Capítulo 3, são descritas as heurísticas que foram implementadas em paralelo. No Capítulo 4, é descrita a metodologia utilizada neste projeto de pesquisa. As análises e os resultados de simulações com resoluções de SELs são apresentados no Capítulo 5. Por fim, as conclusões sobre este trabalho e as proposições de trabalhos futuros são apresentadas no Capítulo 6.

1.1 Justificativa

Conforme mencionado, a resolução de sistemas de equações lineares é necessária em diversas áreas da ciência e da engenharia. O custo computacional exigido nessa resolução, seja por métodos diretos ou iterativos, é alto, quando aplicado a sistemas de equações lineares de grande porte, tornando-se o verdadeiro gargalo em tais simulações computacionais.

Um método bastante utilizado e computacionalmente eficiente na resolução de SELs é o método dos gradientes conjugados precondicionado. Contudo, se não for utilizado um precondicionador adequado, ainda é um método que demanda muito tempo na resolução de SELs. Como exemplos de aplicações reais do método dos gradientes conjugados precondicionado, tem-se o sensoriamento de tubos gasosos ou de fluidos passando por tubos, projeto de aviões e automóveis, previsões meteorológicas, simulações aeroespaciais e modelagem de ondas eletromagnéticas.

Uma otimização para o método dos gradientes conjugados precondicionado é realizar uma permutação adequada nas linhas e colunas da matriz de coeficientes (renumeração dos vértices do grafo). Segundo Oliveira e Chagas (2014, p. 89), com a redução de largura de banda,

obtida por meio de reordenações de linhas e colunas da matriz de coeficientes do sistema de equações lineares, é possível melhorar coerência de *cache*, diminuindo o custo de processamento na resolução de SELs por métodos iterativos. Desse modo, também é importante que se paralelizem os métodos utilizados para redução de largura de banda e de *profile* de matrizes. Com essa medida, toda a aplicação é implementada em paralelo.

Uma das métricas mais utilizadas, para se avaliar o desempenho computacional de um algoritmo paralelo, é compará-lo com o melhor algoritmo sequencial disponível para o problema. Desse modo, uma evidência da qualidade do algoritmo paralelo projetado é obtida pelo *speedup* em relação ao algoritmo sequencial. Já foram publicados diversos algoritmos paralelos que produzem altos *speedups*, para o método dos gradientes conjugados precondicionado, mas não o suficiente para uma ampla variedade de aplicações em tempo real. Em algumas dessas aplicações, são necessárias resoluções de SELs de forma frequente em tempo computacional extremamente pequeno. Em vista disso, o desenvolvimento de algoritmos paralelos, para resolução de SELs, é um campo de pesquisa considerável.

Dada a abrangência de estudos que utilizam métodos, para se conseguir a reordenação de linhas e colunas de matrizes e o custo exigido por eles, a busca por métodos que consigam reduzir largura de banda ou *profile* de matrizes com baixo custo computacional é relevante. Além disso, uma implementação paralela do método dos gradientes conjugados precondicionado é esperada, de modo que o desempenho computacional seja melhor do que o desempenho computacional, apresentado por outras abordagens, tais como do método dos gradientes conjugados precondicionado executado sequencialmente.

Foram encontradas na bibliografia dez abordagens de paralelização (nove artigos e uma dissertação de mestrado) dos métodos Cuthill e McKee (1969) e Reverse Cuthill Mckee (RCM) (GEORGE, 1971). Neste projeto, pretende-se projetar novas implementações paralelas do método RCM com vértice pseudoperiférico dado pelo algoritmo de George e Liu (1979) (GL) (nomeada aqui como RCM-GL (GEORGE; LIU, 1981)), de modo que o desempenho computacional seja melhor em relação às abordagens encontradas na bibliografia. Além disso, será projetado um algoritmo paralelo, também em arquiteturas *multicore*, da heurística KP-band (KOOHESTANI; POLI, 2011). Para essa heurística, não foi encontrada nenhuma abordagem de paralelização na bibliografia. A heurística KP-band é descrita como uma das heurísticas mais promissoras para áreas de aplicação específicas (OLIVEIRA; BERNARDES; CHAGAS, 2018). Desse modo, é apresentada uma justificativa para a relevância deste trabalho.

1.2 Objetivos

O objetivo geral deste trabalho é propor algoritmos paralelos de algoritmos sequenciais existentes na bibliografia em arquiteturas *multicore*, que sejam eficientes nas reduções de largura de banda ou de *profile* de matrizes esparsas. Espera-se que esses novos algoritmos paralelos sejam melhores em custo computacional quando comparados com algoritmos sequenciais existentes na bibliografia. Mais especificamente, essa abordagem será desenvolvida com enfoque em ambientes *multicore* com a utilização das bibliotecas OpenMP e Pthreads. As novas heurísticas paralelas serão implementadas, de modo a obter desempenho computacional satisfatório, na resolução de SELs ou aumentar o *speedup* da aplicação, quando comparado a outras abordagens encontradas na bibliografia. Dessa forma, alguns objetivos específicos, descritos a seguir, serão necessários para se alcançar o objetivo principal deste projeto.

1. Serão realizadas implementações paralelas de algoritmos sequenciais encontrados na bibliografia. Esses algoritmos paralelos serão implementados em arquiteturas *multicore* com duas bibliotecas de paralelização. Heurísticas sequenciais promissoras, para serem implementadas, foram identificadas em Bernardes e Oliveira (2015), Chagas e Oliveira (2015), Oliveira e Chagas (2015), Oliveira, Bernardes e Chagas (2018). Algumas delas são RCM-GL (GEORGE; LIU, 1981) e KP-band (KOOHESTANI; POLI, 2011). O objetivo deste projeto, então, é desenvolver algoritmos paralelos que tenham custo de processamento mais baixos que os custos de algoritmos sequenciais já publicados. As heurísticas sequenciais mencionadas foram utilizadas para reduzir largura de banda e/ou *profile* de matrizes esparsas simétricas e assimétricas. Tais métodos serão implementados com a utilização das bibliotecas OpenMP e Pthreads.

2. Realização de experimentos numéricos e avaliação dos resultados obtidos pelos algoritmos propostos. Serão realizadas comparações dos custos computacionais obtidos com as execuções dos algoritmos, tanto em paralelo como sequencial.

3. Avaliação do custo computacional, envolvido na resolução de sistemas de equações lineares pelo método dos gradientes conjugados precondicionado, proporcionado pelos métodos implementados no item 1, utilizados para reduzir largura de banda e/ou *profile* de matrizes esparsas.

4. Implementações e aplicações do método dos gradientes conjugados precondicionado pela fatoração incompleta de Cholesky (veja as seções 2.2 e 2.3), de forma que a estrutura de dados utilizada proporcione uma complexidade de $\mathcal{O}(N)$, em que N é a quantidade de vértices do grafo associado à matriz de coeficientes. Tanto o precondicionador quanto o método dos gradientes conjugados serão implementados em paralelo.

2 REFERENCIAL TEÓRICO

Neste Capítulo, é apresentado o conteúdo necessário para o entendimento deste trabalho. Recomenda-se a leitura dos capítulos 1 e 2 do livro de Oliveira e Chagas (2014, p. 3-43). Nesses capítulos, são encontradas as definições de elementos básicos utilizados em teoria de grafos e uma introdução a métodos para reduções de largura de banda e de *profile* de matrizes esparsas.

Este Capítulo está organizado conforme descrito a seguir. Na seção 2.1, é mostrada como foi realizada a revisão sistemática, para a elaboração deste projeto e, também, são descritas as abordagens paralelas de heurísticas, para reduzir largura de banda ou *profile* de matrizes, encontradas durante a realização da revisão sistemática da bibliografia. O método dos gradientes conjugados precondicionado é apresentado e discutido na subseção 2.2. Na subseção 2.3, é apresentado o precondicionamento por fatoração incompleta de Cholesky. Na subseção 2.4, são apresentadas as bibliotecas que serão utilizadas para a paralelização das heurísticas.

2.1 Algoritmos paralelos para reduções de largura de banda e/ou *profile* de matrizes

O presente estudo sobre implementações paralelas de heurísticas, para reduções de largura de banda e de *profile* de matrizes, teve início em abril de 2016. Trata-se de uma revisão sistemática realizada na base de dados *Scopus*® (<https://www.scopus.com/>). A busca na base foi realizada utilizando a chave de busca

((TITLE-ABS-KEY(bandwidth) OR TITLE-ABS-KEY (profile)) AND (TITLE-ABS-KEY (reduction) OR TITLE-ABS-KEY(reordering)) AND TITLE-ABS-KEY(matrix) AND TITLE-ABS-KEY(parallel))

OR

(TITLE-ABS-KEY(ordering) AND TITLE-ABS-KEY(preconditioned conjugate gradient) AND TITLE-ABS-KEY(parallel)).

Essa busca, realizada em abril de 2018, resultou em 174 artigos. Em cada um dos artigos recuperados na busca foram verificados título, resumo e palavras-chave. Somente abordagens paralelas de métodos, para reduzir largura de banda ou *profile* de matrizes, foram selecionadas entre os artigos recuperados. Dessa maneira, os dados foram extraídos de forma a adquirir uma comparação clara entre cada uma das publicações. Com isso, foram selecionados 15 artigos e uma dissertação de mestrado para serem analisados detalhadamente.

Na Tabela 2.1, são listadas as publicações com algoritmos para reduções de largura de banda ou de *profile* encontrados na bibliografia. Na Tabela 2.2, são listadas informações sobre as instâncias de testes utilizadas pelos autores em seus respectivos artigos.

Tabela 2.1 – Quinze artigos e uma dissertação de mestrado com descrições de algoritmos paralelos para redução de largura de banda e de *profile* de matrizes publicados de 1992 a 2018. Na primeira coluna, estão listados os algoritmos paralelos para reduzir largura de banda ou *profile* de matrizes esparsas. Na segunda coluna, são descritas as bibliotecas utilizadas para a paralelização dos algoritmos. Nas duas últimas colunas, são apresentadas as heurísticas sequenciais que foram paralelizadas para reduzir largura de banda e/ou *profile* de matrizes esparsas, respectivamente.

Algoritmo paralelo	Biblioteca utilizada	Algoritmos para redução de largura de banda	Algoritmos para redução de *profile*
Prasad, Patnaik e Murthy (1992)	–	Puttonen (1983)	–
Esposito e Tarricone (1996)	CRAY Shared Memory Access e Parallel Virtual Machine	Busca tabu (REEVES, 1993)	
Tarricone (2000)	Parallel Virtual Machine	GA (TARRICONE, 2000)	
Manguoglu et al. (2010)	METIS e ParMETIS	Reordenação espectral ponderada Barnard *et al.* (1995) / RCM (GEORGE, 1971)	
Gibou e Min (2012)	OpenMP	Cuthill e McKee (1969)	
Deb e Srirama (2014)	pacote GA em *R*		
Leveled-RCM e Unordered-RCM (KARANTASIS et al., 2014)	Galois	RCM (GEORGE, 1971)	Sloan (1989)
Tsuburaya, Okamoto e Sato (2015)	OpenMP		–
MSC-BW (MAFTEIU-SCAI; CORNIGEANU, 2016a)	MPI	Mafteiu-Scai e Cornigeanu (2016a)	
MSC-ABR1 e MSC-ABR2 (MAFTEIU-SCAI; CORNIGEANU, 2016b)		Mafteiu-Scai e Cornigeanu (2016b)	
Azad et al. (2017)	MPI e OpenMP	RCM (GEORGE, 1971)	
RCM Unordered, OL-RCM e NS-RCM Unordered (RODRIGUES, 2017), (RODRIGUES; BOERES; CATABRIGA, 2017c), (RODRIGUES; BOERES; CATABRIGA, 2016) e (RODRIGUES; BOERES; CATABRIGA, 2017a)	OpenMP	RCM (KARANTASIS et al., 2014)	Sloan NS-Bag (RODRIGUES; BOERES; CATABRIGA, 2017c) e Sloan NS-Bag Modificada (RODRIGUES, 2017)

Fonte: Do autor (2018)

Na maioria dessas publicações, os autores comparam o algoritmo paralelo com a versão sequencial e não compararam o algoritmo paralelo proposto com outros algoritmos paralelos. Rodrigues, Boeres e Catabriga (2016), por exemplo, mostraram que a execução do algoritmo paralelo implementado apresentou desempenho computacional melhor que a execução do algoritmo paralelo proposto por Karantasis et al. (2014). Mafteiu-Scai e Cornigeanu (2016b) apresentaram duas heurísticas paralelas, comparando os resultados experimentais entre elas. Como não há na bibliografia avaliações entre esses algoritmos, não é possível reconhecer o estado da arte nos problemas.

A seguir, são comentadas, de forma breve, heurísticas paralelas para reduzir largura de banda ou *profile* de matrizes identificadas na bibliografia. Além disso, são abordadas as compa-

Tabela 2.2 – Informações sobre instâncias de testes utilizadas pelos autores de algoritmos paralelos para reduções de largura de banda e de *profile* de matrizes.

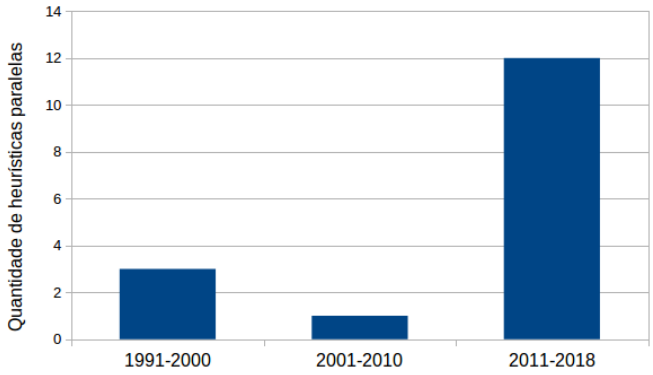
Autores	Dimensão mínima	Dimensão máxima	Quantidade mínima de coeficientes não nulos	Quantidade máxima de coeficientes não nulos
Prasad, Patnaik e Murthy (1992)	–	–	–	–
Esposito e Tarricone (1996)	500	1.000	–	–
Tarricone (2000)	284	1.231	–	–
Manguoglu et al. (2010)	7.548	2.063.494	41.746	14.612.663
Gibou e Min (2012)	–	–	–	–
Deb e Srirama (2014)	–	–	–	–
Karantasis et al. (2014)	503.712	23.947.347	7.660.826	100.663.202
Tsuburaya, Okamoto e Sato (2015)	93.879	381.197	34.396.220	82.050.304
Mafteiu-Scai e Cornigeanu (2016a)	936	12.452	–	–
Mafteiu-Scai e Cornigeanu (2016b)	24	23.133	–	–
Azad et al. (2017)	72.000	27.993.600	28.715.634	760.648.352
Rodrigues (2017)	97.578	1.585.478	1.641.672	77.651.847

Fonte: Do autor (2018)

rações, realizadas pelos pesquisadores, dos resultados apresentados pelas heurísticas paralelas identificadas nesta revisão sistemática.

Esse é um tópico de pesquisa recente. Na Figura 2.1, enfatiza-se que, entre 15 artigos e uma dissertação de mestrado com abordagens paralelas identificadas, 12 foram publicados nos últimos sete anos.

Figura 2.1 – Quantidade de heurísticas paralelas para reduções de largura de banda e/ou de *profile* de matrizes, de 1991 a 2018.



Fonte: Do autor (2018)

2.1.1 Proposta de Prasad e colaboradores

Prasad, Patnaik e Murthy (1992) introduziram um método paralelo para a redução da largura de banda de matrizes simétricas. Trata-se de uma permutação realizada com linhas e colunas de matrizes projetada para o mapeamento de matrizes lineares em arquiteturas *Single Instruction, Multiple Data*. Esse método é uma versão modificada do método sequencial de

Puttonen (1983), também utilizado para redução de largura de banda de matrizes simétricas. Nas resoluções de sistemas de equações lineares, foi utilizada a decomposição LU. A paralelização do método foi apenas descrita e mostrada possível de ser utilizada, por meio de análise de complexidade, ou seja, o método em paralelo não foi implementado.

2.1.2 Proposta de Esposito e Tarricone

Esposito e Tarricone (1996) descreveram um método paralelo para redução de largura de banda de matrizes esparsas, baseado na meta-heurística busca tabu (REEVES, 1993). Por meio de uma função de custo, uma pontuação é calculada para cada vizinho e a melhor delas é selecionada como a solução na iteração seguinte. A função de custo, nesse caso, é representada pela soma ponderada da largura de banda ou do *profile* da matriz de coeficientes.

A implementação do método em paralelo foi realizada e testada nas plataformas IBM SP2 e Cineca CRAY T3D. Na plataforma SP2, foi testado um paradigma de programação mestre-escravo e uma abordagem *Single Program, Multiple Data* (SPMD). Para a implementação dos códigos, foi utilizada a interface de programação *Parallel Virtual Machine*. Na plataforma T3D, o paradigma de programação utilizado foi SPMD com memória compartilhada. A comunicação entre os processos foi implementada com a biblioteca *CRAY Shared Memory Access*, utilizando um recurso de mapeamento lógico com memória compartilhada. Para ambas as plataformas, em que essas duas abordagens foram projetadas, há várias abordagens possíveis de paralelização da meta-heurística. Uma forma possível consiste na execução de várias buscas em paralelo e na comparação periódica dos resultados. A busca continua, em todos os processos, a partir da melhor solução encontrada até o momento. Cada uma dessas buscas se diferencia uma da outra por alguns parâmetros de ajustes. Outra técnica utilizada nessa proposta consiste em distribuir partições do problema entre os processos, realizar as devidas computações e intercalar os resultados em uma única solução.

Foram aplicados dois métodos numéricos na resolução de sistemas de equações lineares. Um deles, chamado de *Mode-Matching* (TARRICONE; DIONIGI; SORRENTINO, 1995), alcançou altos *speedups* ao utilizar estratégias adequadas de renumeração dos vértices. O outro, baseado no método dos elementos finitos (MEF), oferece pacotes apropriados com redutores de largura de banda. Ao apresentar soluções ineficientes, em que a largura de banda não foi reduzida, dois níveis de otimizações heurísticas foram propostos para a estratégia utilizando o

MEF: *otimizações intermediária* e *fina*. Essas abordagens diferem basicamente na quantidade de iterações executadas e no critério de parada.

Nos resultados experimentais, foi verificado que o método paralelo apresentou um desempenho computacional eficiente em aplicações reais (baseadas na modelagem de circuitos eletromagnéticos). Além disso, a estratégia, utilizando o método numérico *Mode-Matching*, foi mais eficaz e escalável do que a estratégia, utilizando o método dos elementos finitos, garantindo um balanceamento de carga na execução do algoritmo.

2.1.3 Proposta de Tarricone

Tarricone (2000) apresentou uma solução por algoritmos genéticos para reduzir largura de banda de matrizes esparsas. Nesse método, a geração da população inicial foi realizada em paralelo e os resultados da pesquisa evolutiva, de cada população, foram coletados periodicamente, de modo que o cruzamento e as mutações foram realizados sobre os cromossomos de populações diferentes com um aumento do nível da hibridação. Em seguida, a função de custo é computada em paralelo, ao realizar uma decomposição em blocos adequada dos dados que compõem a matriz de coeficientes e o vetor de permutação. Contudo a tarefa de computar a função de custo é difícil de ser realizada, especialmente, quando o tamanho do problema é grande.

Aplicado em problemas de modelagem de circuitos eletromagnéticos, os resultados apresentados pelo método paralelo, baseado em algoritmos genéticos, foram comparados com resultados apresentados pelo pacote MEF do *software* EMAP1 (HUBING; ALI; BHAT, 1993), ambas as abordagens em conjunto com um método direto de resolução de sistemas de equações lineares para matrizes em banda. O método paralelo, baseado em algoritmos genéticos, foi implementado, utilizando a interface de programação *Parallel Virtual Machine*, em uma máquina IBM SP2 com oito núcleos. Segundo os autores, os resultados, em relação ao desempenho computacional e eficácia, apresentados pelo método paralelo, foram superiores em relação aos resultados apresentados por algoritmos contidos no pacote MEF do *software* EMAP1.

2.1.4 Abordagem de Manguoglu e colaboradores

Manguoglu et al. (2010) apresentaram uma estratégia de reordenamento das linhas e colunas de uma matriz de tal maneira que é possível obter um precondicionador para resolutores iterativos de sistemas de equações lineares. Para isso, é utilizada uma reordenação não

simétrica para mover os coeficientes para perto da diagonal principal. Uma reordenação espectral ponderada é utilizada para mover os maiores coeficientes para perto da diagonal principal. Desse modo, uma matriz com largura de banda pequena é obtida, para ser utilizada como um precondicionador, para métodos baseados no subespaço de *Krylov*, deixando de fora da banda diagonal os coeficientes de menor magnitude.

Na versão paralela do método, a matriz foi particionada em blocos, para que fossem distribuídos entre os processadores, combinando os resultados de cada processador no final da computação. Nesse caso, o tamanho reduzido do sistema de equações lineares é determinado pela largura de banda da matriz e pela quantidade de partições da matriz.

De acordo com os autores, essa abordagem superou os resultados apresentados por resolutores de sistemas de equações lineares dos pacotes MUMPS (<http://mumps.enseeiht.fr/>) e SuperLU (<http://crd-legacy.lbl.gov/~xiaoye/SuperLU/>). Nesse caso, o reordenamento das linhas e colunas da matriz foi realizado por funções contidas nas bibliotecas METIS e ParMETIS, respectivamente. Por ser um esquema de precondicionamento de matrizes em banda, a estratégia em paralelo proporcionou um desempenho computacional melhor do que uma ampla classe de precondicionadores, baseados em fatoração incompleta, tornando-se escalável em arquiteturas paralelas.

2.1.5 Abordagem de Gibou e Min

Gibou e Min (2012) aplicaram a reordenação de linhas e colunas da matriz pelo método Cuthill-McKee (CM), proposto por Cuthill e McKee (1969), em conjunto com o método dos gradientes conjugados precondicionado paralelo, para resolver a equação de Poisson, em domínios irregulares. O precondicionador paralelo utilizado foi o ILU.

Dada a estrutura de nível gerada, a partir de um vértice inicial, a ordenação dos vértices do grafo associado à matriz A é aplicada de modo incremental, ou seja, a ordenação é iniciada, a partir dos níveis com menor quantidade de vértices, para os níveis com maior quantidade de vértices. Além disso, vértices pertencentes a um mesmo nível são ordenados lexicograficamente. Dessa forma, foi possível obter um equilíbrio entre paralelismo e precondicionamento.

O método Cuthill-McKee permite uma implementação paralela natural dos blocos diagonais da matriz correspondente ao grafo e o precondicionamento é tão eficiente quanto a ordenação lexicográfica. O método Cuthill-McKee foi implementado totalmente em paralelo. A ordenação lexicográfica também foi implementada em paralelo; contudo o precondiciona-

dor ILU foi implementado sequencialmente. Nessa abordagem, os autores mostraram que a aplicação é escalável em arquiteturas *multicore*.

2.1.6 Proposta de Deb e Srirama

Deb e Srirama (2014) projetaram um algoritmo genético paralelo baseado em uma técnica de agrupamento *two-mode* que, por sua vez, é uma adaptação do método Cuthill e McKee (1969). O problema da escalabilidade da organização de um conjunto de dados dispostos em matrizes de dados *two-mode* foi abordado utilizando *clusters* com processadores *multicores*. Como exemplo de matrizes de dados *two-mode*, tem-se os dados gerados a partir de experiências com DNA, RNA e micromatrizes de proteínas. Nessas matrizes, as linhas podem estar relacionadas aos genes e as colunas podem ser representações das particularidades de cada gene.

Na paralelização do método, foi utilizado um pacote de algoritmos genéticos em *R* (SCRUCCA, 2013), baseado no padrão de paralelismo *Simple Network of Workstations*. Nessa abordagem, a codificação dos cromossomos e os dados são enviados para cada um dos núcleos. Em seguida, o algoritmo genético paralelo é executado, em cada um dos núcleos, com diferentes valores atribuídos aos parâmetros de entrada.

O método foi projetado por um modelo de programação mestre-escravo, em que o núcleo mestre envia tarefas para os núcleos trabalhadores, que executam as tarefas e retornam os resultados para o mestre. Por sua vez, o núcleo mestre agrega os resultados e seleciona a solução com menor largura de banda.

Nessa abordagem, os autores consideraram que o algoritmo paralelo é escalável em processadores *multicore* quando comparado com sua implementação sequencial. Nesse caso, a aceleração proporcionada pelo algoritmo genético paralelo foi aumentada de 1 para 2,42, 3,33, 4,0, 5,74 e 6,49 com processamento de dois a seis núcleos, respectivamente. A consistência dos resultados foi verificada, com base no valor médio do tempo de execução do método paralelo, baseado em algoritmos genéticos, executado cinco vezes.

2.1.7 Proposta de Karantasis e colaboradores

Karantasis et al. (2014) apresentaram implementações de três algoritmos de reordenação aplicados em ambiente *multicore*: busca em largura, *Reverse Cuthill-McKee* (RCM) (GEORGE, 1971) e de Sloan (1989). Para encontrar vértices pseudoperiféricos, os autores utilizaram o algoritmo descrito em Kumfert (2000), que é o algoritmo de Sloan com algumas melhorias.

As paralelizações dos métodos foram baseadas na paralelização da busca em largura. Além disso, a biblioteca Galois foi utilizada na implementação dos algoritmos. Uma otimização adotada foi dividir um conjunto de iteradores ordenados em um conjuntos de iteradores não ordenados. Em um conjunto de iteradores não ordenados, cada iteração pode ser paralelizada, processando múltiplas iterações de uma só vez, verificando se iterações simultâneas acessam os mesmos dados. Por outro lado, um conjunto de iterações ordenadas é como um conjunto de iterações não ordenadas com a restrição de que a execução sequencial das iterações deverá ser consistente com uma relação de ordenação definida pelo usuário.

Karantasis et al. (2014) compararam a qualidade das soluções apresentadas pela paralelização das heurísticas *Reverse Cuthill-McKee* (GEORGE, 1971) e de Sloan (1989) com os resultados decorrentes de uma implementação baseada na biblioteca de rotinas matemáticas HSL (Oxford, 2017). Para avaliar o desempenho computacional dos algoritmos implementados, os autores utilizaram um algoritmo que realiza multiplicação de matrizes esparsas por vetores (SpMV), disponível na biblioteca PETSc (<http://www.mcs.anl.gov/petsc>). Foi verificado que a execução dos algoritmos paralelos alcançou desempenho computacional melhor em relação ao desempenho computacional apresentado pelas implementações da biblioteca de rotinas HSL. Além disso, a qualidade das soluções permaneceu a mesma entre os algoritmos implementados.

A seguir, na subseção 2.1.7.1, é descrito o método busca em largura *Unordered*. Na subseção 2.1.7.2, é apresentado o método RCM nivelado. Na subseção 2.1.7.3, é mostrado o método RCM *Unordered*. A implementação da heurística de Sloan é comentada na subseção 2.1.7.4.

2.1.7.1 Método busca em largura *Unordered*

A implementação do algoritmo busca em largura *Unordered* é baseada em um conjunto de iteradores não ordenados. Nesse algoritmo, os vértices são percorridos em paralelo e não precisam ser renumerados à medida que são descobertos. Além disso, na execução do algoritmo, o vértice pode ser computado mais de uma vez. Desse modo, esse algoritmo pode ser utilizado para gerar os níveis finais para cada vértice do grafo.

Inicialmente, os níveis de cada vértice são computados. Em seguida, a quantidade de vértices em cada nível é encontrada. Isso é realizado localmente por cada *thread*. No final dessa etapa, esses valores se tornam conhecidos por todas as *threads*. A próxima etapa é computar a soma do prefixo em paralelo. Esse etapa fornece um intervalo de índices em que os vértices de

determinado nível serão alocados no vetor de permutação. Finalmente, os vértices são armazenados no vetor de permutação. Isso é realizado dividindo os vértices entre as *threads* que, por sua vez, cada *thread* computa a posição de seus vértices no vetor de permutação. Uma vez que os vértices estão armazenados no vetor de permutação, eles são renumerados.

2.1.7.2 Método RCM nivelado

No método RCM nivelado (*Leveled* RCM), o algoritmo utiliza uma abordagem incremental; em cada passo do algoritmo, a permutação dos vértices é computada e não muda nas etapas seguintes. Além disso, cada iteração do algoritmo consiste de uma sequência de etapas, em que cada etapa é executada em paralelo com sincronização entre elas. Quando uma iteração é concluída, os vértices do nível corrente são renumerados. Os principais passos do algoritmo são:

1. expansão: os vértices são descobertos em paralelo. A partir de cada vértice v do grafo, seus adjacentes u são descobertos. Quando um vértice u é descoberto, seu antecessor v e a sua distância ao vértice origem são guardados. De todos os vértices v possíveis, o vértice armazenado é aquele que está mais próximo do vértice origem.

2. redução: para cada vértice do grafo, a quantidade de vértices adjacentes é computada.

3. soma de prefixo: para cada vértice u, uma posição no vetor de permutação é computada. Essa posição é calculada de acordo com o vértice antecessor.

4. reordenação: todos os vértices são armazenados no vetor de permutação, respeitando o intervalo de posições computado na etapa anterior. A sequência desse arranjo respeita a ordem imposta pelo método RCM, ou seja, a ordem de busca determinada para cada vértice antecessor. Uma vez que os vértices estão armazenados no vetor de permutação, eles são renumerados.

2.1.7.3 Método RCM *Unordered*

Essa abordagem é semelhante ao algoritmo de reordenamento da busca em largura paralelo com uma fase de reordenação mais rigorosa. Suas três primeiras etapas são as mesmas do RCM nivelado. Nesse algoritmo, os níveis são gerados, a quantidade de vértices em cada nível é encontrada e a soma de prefixo dos níveis é computada. Uma permutação direta é gerada em relação ao método RCM, mas que só é válida após a geração da estrutura de nível.

Na fase de reordenação, os vértices são computados em uma ordem específica para produzir um reordenação correta para o nível seguinte. Com isso, vértices de cada nível são atribuídos a uma única *thread* e as *threads* se comunicam com o nível seguinte por meio de filas do tipo produtor e consumidor. Assim, uma *thread* recebe os vértices, na ordem correta, de acordo com o nível e os produz na ordem correta para o nível seguinte.

O algoritmo RCM ordena os vértices de duas formas. Primeiro, os vértices em um nível são ordenados de acordo com seus antecessores. Essa ordem é trivialmente respeitada, uma vez que cada vértice é computado na ordem em que foi produzido no nível anterior. Além disso, os vértices em um mesmo nível são ordenados em ascendente de grau. Isso é assegurado, porque cada *thread*, ao processar um vértice v, ordena os seus vértices adjacentes u por grau antes de encaminhá-los para o nível seguinte. À medida que os vértices são computados, eles podem ser renumerados no vetor de permutação. O intervalo alocado, no vetor de permutação, para cada nível da estrutura de nível enraizada, no vértice inicial, é calculado utilizando a etapa soma do prefixo e a quantidade de vértices em cada nível.

A paralelização da fase de renumeração é similar a um *pipelining*. Uma *thread*, no nível atual da estrutura de nível enraizada, armazena os vértices u, de acordo com a ordem imposta pelo método RCM, para que uma *thread* no nível seguinte possa computar. Essa paralelização é possível porque os vértices de cada nível da estrutura de nível enraizada, no vértice inicial, são computados assim que os vértices do nível anterior são computados.

2.1.7.4 Heurística de Sloan

A heurística de Sloan (1989) foi proposta para reduzir o *profile* de matrizes esparsas e simétricas. Nessa heurística, é atribuído a cada um dos vértices o estado inativo. Se o vértice já foi numerado, ele recebe o estado pós-ativo. Para todo vértice que é adjacente a um vértice pós-ativo e ainda não possui o estado pós-ativo, ele é definido com o estado ativo. Vértices que são adjacentes a um vértice ativo e não possuem um estado ativo ou pós-ativo são definidos com o estado pré-ativo.

Inicialmente, o algoritmo escolhe dois vértices pseudoperiféricos, sendo um deles o vértice inicial e o outro o vértice final. A partir do vértice inicial, escolhe-se o próximo vértice a ser renumerado entre os vizinhos dos vértices atualmente numerados e seus vizinhos. Para isso, é criada uma fila de prioridades. À medida que as informações dos vértices são atualizadas, por meio de informações referentes ao grafo, o vértice com a maior prioridade é escolhido

para receber um novo rótulo. A prioridade de cada vértice está relacionada com o estado desse vértice, com seu grau e com sua distância até o vértice final e. O algoritmo termina quando todos os vértices forem renumerados.

Na paralelização da heurística de Sloan, os autores propuseram uma abordagem em que a ordem de prioridade dos vértices visitados é relaxada. Para isso, utilizaram uma fila de prioridade aproximada, fornecida pela biblioteca Galois. Nessa biblioteca, os vértices são retornados em ordem de prioridade, mas podem admitir um certo número de inversões de prioridade quando um vértice é recuperado fora da ordem estrita. Para evitar conflitos na atualização dos vértices do grafo, cada atualização foi realizada de forma atômica, por meio de instruções de comparação e troca.

2.1.8 Proposta de Tsuburaya e colaboradores

Tsuburaya, Okamoto e Sato (2015) apresentaram um método, chamado de *Reverse Block-Multicolor* (RBMC), para reordenação de linhas e colunas da matriz, baseado na coloração multicolor dos vértices e no método RCM. O objetivo foi melhorar o desempenho computacional da reordenação multicolor em blocos na paralelização do precondicionador incompleto de Cholesky em conjunto com o método dos gradientes conjugados. Como a quantidade de vértices em cada nível da estrutura de nível enraizada se torna não uniforme, o balanceamento de carga no precondicionamento se torna também não uniforme. Desse modo, o método RBMC foi combinado com a técnica de decomposição da matriz de coeficientes em blocos, baseada na árvore rubro-negro (IWASHITA; NAKANISHI; SHIMASAKI, 2005; IWASHITA; SHIMASAKI, 2003a).

O objetivo dessa abordagem foi reduzir a quantidade de sincronizações progressiva e retroativa despendidas na resolução de sistemas de equações lineares. Adicionalmente, um balanceamento de carga uniforme no precondicionamento foi conseguido por meio do método RBMC paralelo.

O método RBMC paralelo foi avaliado ao ser comparado com os métodos de reordenação multicolor (ADAMS; JORDAN, 1986) e reordenação multicolor em blocos (IWASHITA; NAKASHIMA; TAKAHASHI, 2012; SEMBA et al., 2013). Como resultado, o método RBMC foi considerado computacionalmente mais eficaz que os demais métodos de reordenação testados, em virtude da redução na quantidade de sincronização nas substituições progressiva e

retroativa e na melhoria do balanceamento de carga realizado pelo precondicionamento na matriz de coeficientes.

2.1.9 Propostas de Mafteiu-Scai e Cornigeanu

Mafteiu-Scai e Cornigeanu (2016a) mostraram uma heurística paralela e híbrida para reduzir largura de banda de matrizes esparsas e simétricas. Nessa heurística, é utilizada uma seleção gulosa de linhas/colunas, para serem permutadas, dependendo da posição dos coeficientes não nulos, localizados nas extremidades de cada linha/coluna e de alguns parâmetros relacionados com a matriz de coeficientes.

Mafteiu-Scai e Cornigeanu (2016b) apresentaram duas heurísticas paralelas e híbridas, utilizadas para redução da largura de banda média de matrizes esparsas. Segundo os autores, a largura de banda da matriz não fornece informações suficientes sobre o padrão de agrupamento de coeficientes não nulos ao redor da diagonal principal. Com isso, a largura de banda média da matriz foi definida como a razão entre a largura de banda e a quantidade de coeficientes não nulos da matriz.

Nessa abordagem, o objetivo é obter uma distribuição uniforme dos coeficientes não nulos ao redor da diagonal principal. Com base em um processamento direto na matriz e inspirada em leis da física, a primeira heurística realiza uma seleção gulosa de linhas e colunas da matriz para realizar a permutação. Na segunda heurística, o objetivo foi melhorar a primeira heurística, por meio da utilização de uma fórmula exata, determinando as permutações mais adequadas na matriz.

Na paralelização dos algoritmos, foi utilizado o modelo mestre-escravo. Para isso, as matrizes foram divididas em blocos compactos de tamanhos iguais, dependendo da quantidade de processos utilizados. Além disso, foi utilizado um modelo síncrono de paralelização, produzindo um balanceamento de carga entre os processos escravos.

Todos os algoritmos foram implementados em um supercomputador *IBM Blue Gene/P*, utilizando o compilador IBM C++. A biblioteca MPI foi empregada para a comunicação entre processos. Para o armazenamento das matrizes, foi utilizado o formato de coordenadas.

Segundo os autores, os resultados experimentais apresentados pelas heurísticas paralelas foram considerados melhores quando comparados com os resultados experimentais apresentados pelas heurísticas sequenciais. Além disso, o *speedup* da segunda heurística foi inferior ao

speedup da primeira heurística por causa das tarefas paralelas adicionais no processo mestre e da comunicação realizada entre os processos.

2.1.10 Proposta de Azad e colaboradores

Azad et al. (2017) apresentaram uma implementação paralela do método RCM com memória distribuída para reduzir o *profile* de matrizes esparsas. Para encontrar um vértice pseudo-periférico, foi utilizado o algoritmo de Gibbs, Poole e Stockmeyer (1976). Na paralelização do método, foi utilizada uma decomposição da matriz de coeficientes em submatrizes, de modo que o desempenho computacional melhorasse em relação ao algoritmo sequencial. Além disso, operações em grafos não estruturados foram substituídas por operações básicas envolvendo vetores e matrizes. Com isso, foi apresentado um algoritmo baseado em primitivas paralelas síncronas e otimizadas para serem executadas em sistemas com memória compartilhada e distribuídas.

Para a distribuição das matrizes e dos vetores esparsos entre os processadores, foi utilizado o *framework Combinatorial BLAS* (CombBLAS) (BULUÇ; GILBERT, 2011). Além disso, foram utilizadas as bibliotecas OpenMP e MPI. A implementação com essas ferramentas atingiu um *speedup* satisfatório em matrizes originadas de várias aplicações executadas em um supercomputador Cray XC30 com 1024 núcleos. Somente uma *thread* em cada processo realizou chamadas MPI para comunicação entre os nós da rede.

Os autores mostraram que a implementação é escalável em computadores com até 1024 núcleos para matrizes de pequeno porte e até 4096 núcleos para matrizes de grande porte. A memória distribuída, utilizada pela primitiva SPMSPV, é o passo mais caro na implementação desse algoritmo paralelo. O tempo de comunicação despendido pela primitiva SPMSPV começa a dominar o tempo de computação quando a concorrência é maior. Além disso, grafos com diâmetros maiores possuem uma sobrecarga de comunicação maior do que grafos com diâmetros menores.

2.1.11 Propostas de Rodrigues e colaboradores

Na tese de Rodrigues (2017), foram mostradas três abordagens de paralelização do método RCM e duas abordagens de paralelização da heurística de Sloan (1989). Com a implementação do método RCM, foram produzidos os artigos de Rodrigues, Boeres e Catabriga (2017c), Rodrigues, Boeres e Catabriga (2017a) e Rodrigues, Boeres e Catabriga (2016). Com a implementação da heurística de Sloan (1989), foi produzido o artigo de Rodrigues, Boeres e

Catabriga (2017b). A biblioteca OpenMP e a linguagem de programação C++ foram utilizadas nas implementações dos algoritmos.

Na subseção 2.1.11.1, é mostrado o método RCM *Unordered*. Na subseção 2.1.11.2, é apresentado o método OL-RCM. Na subseção 2.1.11.3, é descrito o método NS-RCM *Unordered*. A heurística de Sloan paralela, baseada em *bags* e numa abordagem não especulativa, é apresentada na subseção 2.1.11.4. Por fim, na subseção 2.1.11.5, é descrita uma segunda abordagem paralela da heurística de Sloan baseada na versão anterior.

2.1.11.1 Método RCM *Unordered*

Rodrigues, Boeres e Catabriga (2017c) descreveram uma implementação paralela do método RCM, chamada pelos autores de RCM *Unordered*, que é comparada com a abordagem sequencial do método RCM nivelado de Karantasis et al. (2014). Para o conjunto de instâncias testadas, a redução de tempo computacional variou entre 78% e 94%. Os autores também relataram que o método RCM *Unordered* paralelo obteve *speedups* de 2 a 17 vezes, em relação à versão sequencial do algoritmo RCM nivelado de Karantasis et al. (2014). Sobre a qualidade das soluções, a redução de largura de banda alcançada foi a mesma para as implementações sequenciais e paralelas.

Baseado no algoritmo busca em largura não ordenado de Karantasis et al. (2014), com a criação da estrutura de nível enraizada no vértice inicial, o algoritmo mantém uma lista de trabalho *ws* com vértices visitados, a partir da qual qualquer vértice pode ser selecionado. Com isso, vários vértices são computados em paralelo. Para cada iteração do algoritmo na lista *ws*, uma *thread* remove um vértice arbitrário v de *ws* e calcula o nível apropriado para os vértices adjacentes de v. Os vértices visitados são adicionados à lista *ws* e o algoritmo itera até que a lista *ws* esteja vazia.

Para reduzir a sobrecarga de acesso das *threads* na lista de trabalho *ws*, cada *thread* computa um conjunto de vértices de *ws* em vez de somente um vértice. O resultado dessa computação é armazenado em uma lista de trabalho local e, logo depois que todo o conjunto de vértices é computado, esses vértices são armazenados na lista de trabalho global. Outra otimização adotada é o balancear a quantidade de trabalho em cada *thread*. Como a lista de trabalho *ws* é acessada arbitrariamente pelas *threads*, essa lista pode estar vazia. Nesse caso, algumas *threads* podem se tornar ociosas. Assim, a estratégia utilizada foi manter uma ordem

em que as *threads* removem vértices de uma extremidade da lista de trabalho e adicionam na outra extremidade da lista de trabalho.

2.1.11.2 Método OL-RCM

Rodrigues, Boeres e Catabriga (2016) apresentaram uma implementação paralela e otimizada do método *Leveled Reverse Cuthill-McKee* (L-RCM) (CHAN; GEORGE, 1980), chamada pelos autores de OL-RCM. Para a determinação dos vértices pseudoperiféricos, os autores utilizaram um algoritmo descrito em Kumfert (2000); porém o tempo computacional necessário, para computar esses vértices, não foi contabilizado.

Nessa abordagem, duas otimizações no método RCM *Unordered* foi realizada. Uma delas consiste na representação das matrizes de coeficientes que foi realizada por meio de um vetor de *buckets*. Um vetor de *buckets* consiste em um vetor B de tamanho n, em que cada célula de B é considerada um *bucket*, ou seja, uma coleção de pares do tipo valor-chave. Desse modo, os vértices são arranjados nesse vetor por meio de uma função *hash*. Para evitar conflitos no algoritmo, a computação do nível de cada vértice na estrutura de nível enraizada no vértice inicial é realizada de forma atômica. Além disso, a maioria das operações realizadas em cada vértice ocorre no *bucket*. Como exemplo, cada *thread* utiliza uma certa quantidade de vértices no *bucket* para determinar os rótulos dos vértices. Com isso, o custo computacional envolvido no acesso a um *bucket* está relacionado com operações executadas pela função *hash*, adicionado ao custo computacional de acessar uma posição específica no *bucket*.

Outra otimização proposta está relacionada ao número de etapas executadas pelo algoritmo. No método RCM nivelado de Karantasis et al. (2014), o algoritmo é dividido em quatro etapas. Na abordagem baseada em *buckets*, apenas as etapas de expansão e reordenação foram implementadas. Com a utilização de *buckets*, as fases de expansão e redução foram combinadas em apenas uma etapa. De maneira semelhante, como os vértices adjacentes de u já estão agrupados pelo vértice u (*bucket*), após a primeira etapa, os vértices de cada *bucket* podem ser armazenados, no vetor de permutação, sem a necessidade da etapa soma de prefixo.

No método OL-RCM, os autores compararam os resultados de sua implementação paralela com os resultados apresentados pela versão sequencial do RCM nivelado de Karantasis et al. (2014) e pelo método RCM paralelo de Karantasis et al. (2014). Nesse caso, os resultados experimentais apresentados pela abordagem OL-RCM foram considerados melhores que os resultados obtidos com as outras abordagens.

2.1.11.3 Método NS-RCM *Unordered*

Rodrigues, Boeres e Catabriga (2017a) mostraram uma implementação paralela e otimizada do método RCM nivelado de Karantasis et al. (2014), que é baseado em quatro passos. Este método, denominado NS-RCM *Unordered*, é uma versão não especulativa. Em uma versão não especulativa, operações como sincronização e particionamento de dados são responsabilidades do programador. A biblioteca Galois, por exemplo, segue um padrão especulativo em que essas operações já estão implementadas na biblioteca. Além disso, no método NS-RCM *Unordered*, a reordenação final dos vértices só é válida depois que toda a estrutura de nível enraizada no vértice inicial é gerada e seus vértices computados. O algoritmo de Kumfert (2000), utilizado para encontrar vértices pseudoperiféricos, também foi paralelizado.

Nessa implementação, os vértices são computados em paralelo. Com isso, o nível de cada vértice também é computado em paralelo. Como não há uma ordem estrita na computação dos vértices, pode ser atribuído ao vértice um valor de nível maior do que o valor de nível correto. Desse modo, o valor do nível desse vértice diminuirá até atingir o valor correto.

A principal diferença entre essa abordagem e as anteriores é a utilização de um operador de ordenação que é utilizado na ordenação dos vértices, denotado por *ordered foreach*. Esse operador foi empregado nas fases de expansão e renumeração. A utilização do operador de ordenação elimina a necessidade de se utilizar uma estrutura de dados auxiliar como *buckets* em associação com um ponto de sincronização entre fases do algoritmo, de modo que a ordem dos vértices computados, imposta pelo método RCM, seja mantida. Isso é possível porque a ordem de computação adotada pelas *threads* é criada pelo operador de ordenação na fase de expansão. Como essa ordem é transferida para o segundo operador de ordenação na fase de renumeração, não há necessidade de uma sincronização extra para manter a ordem imposta pelo método RCM. Além disso, quando uma *thread* conclui a etapa de expansão, ela adiciona os vértices adjacentes do vértice corrente no vetor de permutação. Desse modo, não é mais necessária uma sincronização de *threads* em cada iteração do algoritmo.

Quando os vértices são descobertos, eles são armazenados na lista trabalho para serem computados posteriormente. Para reduzir a sobrecarga gerada, durante o acesso aos vértices, dessa lista de trabalho, cada *thread* computa um conjunto desses vértices. Com isso, uma lista de trabalho local é criada em *cache* para cada *thread*. Com essa otimização, a sincronização necessária entre as *threads* só é realizada, após a computação de um conjunto de vértices, em vez de ser realizada durante a computação de cada vértice.

Para evitar que *threads* fiquem ociosas, vértices são removidos de uma extremidade da lista de trabalho e adicionados à outra extremidade da lista de trabalho. O acesso simultâneo, na extremidade final dessa lista, é realizada por meio de bloqueios. Para reduzir o tempo de acesso à lista de trabalho, durante esses bloqueios, um conjunto de vértices recém-adicionados são deslocados, para a extremidade da lista de trabalho, logo que uma *thread* retira um conjunto de vértices da lista de trabalho. Após essa mudança, o bloqueio de acesso ao início da lista de trabalho é liberado e outra *thread* inicia a operação de remoção do conjunto de vértices. Simultaneamente, outra *thread* realiza o bloqueio de acesso e também o deslocamento de um novo conjunto de vértices para a posição inicial da lista de trabalho.

Por fim, na segunda fase do algoritmo, a quantidade de vértices por nível na estrutura de nível enraizada no vértice inicial também é computada em paralelo. Na fase de soma de prefixo, cada *thread* calcula as somas de prefixo dos vértices que possui localmente. A quantidade total de vértices corresponde ao nível máximo que foi computado na etapa anterior. Na fase de renumeração, similar a fase de Karantasis et al. (2014), cada *thread* é responsável por renumerar um conjunto de vértices.

A qualidade da solução e o desempenho computacional foram avaliados em comparação com uma implementação sequencial do método RCM da biblioteca de rotinas HSL (HU; SCOTT, 2003). Para algumas das instâncias testadas, o desempenho computacional do método NS-RCM *Unordered* melhorou em até 57,82%. As reduções de largura de banda alcançadas pela execução da abordagem paralela do método NS-RCM *Unordered* foram em mais de 90% nas matrizes testadas.

2.1.11.4 Heurística NS-Bag Sloan

Na primeira implementação da heurística de Sloan paralela, foi utilizada uma estrutura de dados baseada em *bags*. Essa estrutura permite obter uma paralelização computacionalmente eficiente, em que os vértices são armazenados de forma contígua na memória. Essa estrutura de dados é uma coleção não ordenada de elementos em que a ordem de inserção é irrelevante. O desempenho computacional apresentado por essa implementação é comparado com uma implementação sequencial disponibilizada pela biblioteca Boost (SIEK; LEE; LUMSDAINE, 2002). O algoritmo paralelo alcançou taxas de redução de *profile* e melhorias significativas no tempo de execução.

2.1.11.5 Heurística NS-Bag Sloan modificada

Na segunda implementação da heurística de Sloan paralela, baseada em na primeira versão não especulativa do algoritmo, o menor valor de prioridade é utilizado para atualizar a prioridade dos vértices. Inicialmente, o algoritmo constrói uma lista de trabalho global, ordenada em ordem decrescente por prioridade. Para isso, as referências para os vértices de maior e menor prioridade são armazenadas localmente. Uma vez que cada *thread* é responsável por computar um conjunto de vértices locais, os valores de prioridades e de *status* dos vértices são computados em paralelo. Porém, a atualização desses valores na lista de trabalho global é realizada sequencialmente. Da mesma forma que na versão de Karantasis et al. (2014), essa versão do algoritmo de Sloan paralelo gera diferentes permutações de linhas e colunas na matriz de coeficientes para cada execução do algoritmo. Isso acontece porque várias *threads* competem pelo acesso aos vértices da lista de prioridades.

2.2 Método dos gradientes conjugados precondicionado

O método dos gradientes conjugados precondicionado está entre os mais eficientes métodos, para solução de sistemas de equações lineares esparsos, formado por matrizes simétricas e positivas definidas. Dado um sistema $Ax = b$ e um ponto para a aproximação inicial $x^{(0)}$ da solução do sistema, o objetivo do método dos gradientes conjugados (MGC) é reduzir o resíduo dado por $r^{(0)} = b - Ax^{(0)}$ até que o resíduo seja nulo. Neste caso, a matriz de coeficientes $A = [a_{ij}]$ é uma matriz $n \times n$ esparsa, simétrica e positiva definida, o vetor x é o vetor de incógnitas e o vetor b é o vetor de termos independentes. Para que o resíduo diminua, toma-se a direção v, de modo que $x^{(0)}$ tome essa direção, ou seja, $x^{(1)} = x^{(0)} + \lambda\vec{v}$, de forma que $r^{(1)} = b - Ax^{(1)} < r^{(0)}$. Dessa forma, constrói-se uma sequência $\{x^{(k)}\}$ que converge para a solução do sistema $Ax = b$, considerando que resíduo é diminuído em cada passo do processo iterativo, ou seja, $r^{(k+1)} < r^{(k)}$.

Teoricamente, o MGC converge para a solução do sistema linear em n iterações, em que n é a dimensão da matriz A. Entretanto, nem sempre isso acontece em virtude dos erros de arredondamento e cancelamento que fazem com que o vetor resíduo perca precisão ou os vetores direção deixem de ser conjugados. Isso ocorre quando a matriz A é mal condicionada. Desse modo, o custo de processamento do MGC depende da utilização de precondicionadores adequados. Um precondicionador realiza uma transformação linear adequada no sistema de

equações lineares, da forma $Ax = b$, com o objetivo de obter um sistema equivalente, que possua propriedades espectrais favoráveis à convergência do método, ou seja, o número de condição espectral e a distribuição dos autovalores são melhorados no sistema resultante. Em vista disso, o MGCP é frequentemente utilizado no lugar do MGC.

O pseudocódigo do método dos gradientes conjugados precondicionado sequencial é mostrado no Algoritmo 1 (SAAD, 2003). Inicialmente, define-se a precisão numérica ε desejada, por exemplo, com 10^{-8}. A variável *erro* é inicializada na linha 2, em que *erro* é a variável para o critério de parada. Na linha 3, inicializam-se os vetores x, r, z e p. A variável j é o contador para a estrutura de repetição. A cada iteração atualiza-se o valor de $erro = \frac{||x_j - x_{j-1}||_\infty}{||x_j||_\infty}$, em que $||x||_\infty = max\{|x_1|, |x_2|, \cdots, |x_n|\}$. Se a condição $erro \geq \varepsilon$ for satisfeita, o algoritmo para.

Algoritmo 1 - Método dos gradientes conjugados precondicionado sequencial.

1: $\varepsilon = 10^{-8}$;
2: $erro = \varepsilon + 1$;
3: Compute $r_0 = b - A \cdot x_0$, $z_0 = M^{-1} \cdot r_0$, $p_0 = z_0$;
4: **para** (j = 0, 1, ..., até que ocorra a convergência e $erro > \varepsilon$) **faça**
5: $\alpha = (r_j, z_j)/(Ap_j, p_j)$;
6: $x_{j+1} = x_j + \alpha_j \cdot p_j$;
7: $r_{j+1} = r_j - \alpha_j \cdot A \cdot p_j$;
8: $z_{j+1} = M^{-1} \cdot r_{j+1}$;
9: $erro = \frac{||x_k - x_{k-1}||_\infty}{||x_k||_\infty}$;
10: $\beta_j = (r_{j+1}, z_{j+1})/(r_j, z_j)$;
11: $p_{j+1} = z_{j+1} + \beta_j \cdot p_j$;
12: **fim para**

A eficiência do precondicionador depende de se escolher uma matriz precondicionadora M que se aproxime da matriz de coeficientes A, tenha a mesma solução, porém, seja mais fácil de resolver (SAAD, 2003). A seguir, são mostradas as três formas, seja por aproximação da matriz esparsa inversa ou pela matriz descomposta, para o precondicionamento de matrizes.

- Precondicionadores de aproximação da matriz esparsa inversa: $M \approx A^{-1}$:
  - precondicionamento pelo lado esquerdo: $M \cdot A \cdot x = M \cdot b$;
  - precondicionamento pelo lado direito: $A \cdot M \cdot y = b$, $x = M \cdot y$;
  - precondicionamento por ambos os lados: $M_2 \cdot A \cdot M_1 \cdot y = M_2 \cdot b$, $x = M_1 \cdot y$.
- Precondicionadores de matriz descomposta: $M \approx A$:
  - precondicionamento pelo lado esquerdo: $M^{-1} \cdot A \cdot x = M^{-1} \cdot b$;

- precondicionamento pelo lado direito: $A \cdot M^{-1} \cdot y = b$, $x = M^{-1} \cdot y$;
- precondicionamento por ambos os lados: $M_2^{-1} \cdot A \cdot M_1^{-1} \cdot y = M_2^{-1} \cdot b$, $x = M_1^{-1} \cdot y$.

Os vetores x e y são vetores de incógnitas e o vetor b corresponde ao vetor de termos independentes. No caso do precondicionamento por ambos os lados ou *split*, a matriz precondicionada M é separada nas matrizes M_1 e M_2. O precondicionamento é realizado pela esquerda na matriz M_1 e pela direita na matriz M_2. No precondicionamento por aproximação esparsa da matriz inversa, a matriz A é multiplicada pela inversa da matriz pré-condicionadora. Por fim, no precondicionamento por matriz descomposta, a matriz precondicionadora é obtida por meio da decomposição da matriz A em fatores triangulares.

De acordo com Saad (2003), essas três formas produzem sistemas de equações lineares semelhantes e com as mesmas propriedades espectrais. A seguir, é listado um precondicionador baseado em fatoração incompleta de Cholesky, que será utilizado em conjunto com o MGC.

2.3 Precondicionamento por fatoração incompleta de Cholesky

O método dos gradientes conjugados precondicionado pela fatoração incompleta de Cholesky, abreviado por ICCG (Incomplete Cholesky Conjugate-Gradient), é utilizado para resolver uma ampla variedade de problemas científicos (BENZI; TUMA, 2003; MEIJERINK; VAN DER VORST, 1977; VAN DER VORST; DEKKER, 1988). Porém sua utilização, em paralelo, não é considerada adequada em razão das substituições sucessivas e retroativas (MOGHNIEH; LOWTHER, 2010; NOGUEIRA, 2011). Desse modo, variantes desse precondicionador são desenvolvidas para que possam melhorar o desempenho computacional do MCG (ČIEGIS, 2005; CONCUS; GOLUB; MEURANT, 1985; KARDANI; LYAMIN; KRABBENHOFT, 2013).

O precondicionamento IC é baseado na fatoração da matriz A, na forma $M = L \cdot L^T$, em que L é uma matriz triangular inferior de dimensão $(n \times n)$ tal que $L = (l_{ij})$ e $(l_{ij}) = 0$ se $i < j$, e L^T é a transposta. Segundo Golub e Loan (2012), esse precondicionamento é ideal para matrizes positivas definidas e que possuam dominância da diagonal principal. Se a matriz não for estritamente diagonal dominante, podem existir pivôs nulos ou negativos, durante a fatoração, originando erro de execução pelo cálculo de raiz quadrada de número negativo ou divisão por zero. Para matrizes que não possuem dominância da diagonal principal, foi utilizada a estratégia de Kershaw (1978), em que é escolhido um valor positivo, utilizado para substituir

o valor do pivô negativo. Nesse caso, foi escolhido o oposto do valor negativo do pivô. O pseudocódigo do precondicionador incompleto de Cholesky sequencial com zero *fill-in* IC(0) é mostrado no Algoritmo 2 (GOLUB; LOAN, 2012).

Algoritmo 2 - Precondicionador incompleto de Cholesky com zero *fill-in*

Entrada: matriz A;
Saída: matriz M.

1: $M = A$;
2: **para** ($k = 1$ até n) **faça**
3: $\quad M_{k,k} = \sqrt{A_{k,k}}$;
4: $\quad$ **para** ($i = k+1$ até n) **faça**
5: $\quad\quad$ **se** ($M_{i,k} \neq 0$) **então**
6: $\quad\quad\quad M_{i,k} = M_{i,k}/M_{k,k}$;
7: $\quad\quad$ **fim se**
8: $\quad$ **fim para**
9: $\quad$ **para** ($j = j+1$ até n) **faça**
10: $\quad\quad$ **para** ($i = j$ até n) **faça**
11: $\quad\quad\quad$ **se** ($M_{i,j} \neq 0$) **então**
12: $\quad\quad\quad\quad M_{i,j} = M_{i,j} - M_{i,k} \cdot M_{j,k}$;
13: $\quad\quad\quad$ **fim se**
14: $\quad\quad$ **fim para**
15: $\quad$ **fim para**
16: **fim para**
17: **retorna** M.

De acordo com Almeida (1999), a implementação do método em um ambiente paralelo não apresenta características tão vantajosas. Isso ocorre tanto na montagem da matriz triangular inferior ou superior, quanto na obtenção da matriz precondicionada. Nesse caso, o precondicionador foi aplicado, em dois exemplos de estruturas, que foram avaliados em processamento paralelo, pavimento e treliça tridimensional, apresentando resultados computacionais melhores, quando comparados com resultados computacionais, apresentados pelo precondicionador de Jacobi.

Semba et al. (2013) aplicaram o precondicionador de Cholesky incompleto em conjunto com o método dos gradientes conjugados em ambientes *multithreaded*, na resolução de problemas envolvendo a análise de campos eletromagnéticos. A ordenação multicolor de blocos algébricos é introduzida na paralelização do resolutor para melhorar a taxa de acerto na *cache*. O método paralelizado melhora a convergência sem aumentar a quantidade de sincronizações entre as *threads*, ao contrário de técnicas que utilizam ordenação multicolor.

Seshima, Tanaka e Tsuboi (2006) descreveram uma abordagem paralela do método ICCG utilizando o ambiente OpenMP em sistemas *Symmetric Multi-Processing*. Nessa aborda-

gem, os resultados computacionais apresentados foram considerados melhores, quando comparados com resultados computacionais, apresentados pela sua respectiva versão sequencial.

Iwashita e Shimasaki (2003a), por exemplo, implementaram o método ICCG em ambiente *multithreaded*. Nessa implementação, foi apresentada uma abordagem algébrica em que os vértices (dados) são agrupados em blocos e distribuídos em uma malha; em seguida, é aplicada uma ordenação multicolor (*red-black*) em cada um desses blocos. Com essa estratégia, os autores conseguiram obter uma abordagem eficiente com altas taxa de acerto na *cache* e rápida convergência do método ICCG.

Iwashita e Shimasaki (2003b) propuseram uma abordagem paralela utilizando estratégias de bloqueios e ordenação vermelho-preto. Neste método, os vértices do grafo são divididos em vários blocos e a ordenação vermelho-preto é aplicada em cada um deles. Esse método tem a vantagem de exigir poucos pontos de sincronização durante as iterações do método ICCG. Segundo os autores, a taxa de convergência é melhorada pelo aumento na quantidade de vértices por bloco.

Embora essas abordagens de paralelização do método ICCG demonstrem um desempenho computacional eficiente, na resolução de sistemas de equações lineares, ainda é necessário um alto custo computacional para realizar as substituições sucessivas e retroativas. Com isso, outras abordagens de paralelização do método ICCG devem ser investigadas. Neste trabalho, o objetivo é melhorar o desempenho computacional do método ICCG paralelo quando comparado com o desempenho computacional do método ICCG sequencial. Para isso, foi empregada uma variação do precondicionamento por fatoração incompleta de Cholesky com zero *fill-in*, a fatoração incompleta de Cholesky *shifted* (KERSHAW, 1978) utilizando a estrutura de dados CRS/CCS *Skyline*.

2.4 Bibliotecas para projeto de algoritmos em arquiteturas *multicore*

Nesta subseção, são comentadas, de forma breve, duas bibliotecas utilizadas para desenvolvimento de algoritmos em arquiteturas *multicore*: OpenMP e Pthreads. De modo geral, ambas as bibliotecas oferecem ao programador as mesmas funcionalidades, tais como:

- determinar regiões críticas e especificar a quantidade de *threads* utilizadas na execução do algoritmo;

- criação e sincronização de *threads*: esse procedimento é mais rápido do que realizar o mesmo procedimento com processos;
- sobreposição de trabalho de CPU com E/S: por exemplo, enquanto uma *thread* está aguardando por uma chamada de sistema de E/S para concluir sua tarefa, a CPU pode ser utilizada por outras *threads*;
- manipulação de eventos assíncronos: um servidor web, por exemplo, pode transferir dados de solicitações anteriores e gerenciar a chegada de novas solicitações;
- comunicação eficiente entre *threads*: isso ocorre por causa do espaço de endereçamento compartilhado.

A principal diferença entre essas bibliotecas é que OpenMP é de alto nível e Pthreads é de baixo nível. A biblioteca OpenMP é apresentada na subseção 2.4.1. Na subseção 2.4.2, é abordada a biblioteca Pthreads.

2.4.1 OpenMP

OpenMP (<http://www.openmp.org/>) é uma interface de programação de aplicativos (API). É uma implementação *multithreading*, portátil, baseada no modelo de programação paralela de memória compartilhada para arquiteturas com múltiplos processadores. Com OpenMP, é permitido, de maneira eficiente, fazer programação concorrente em linguagens de programação C, C++ e Fortran, podendo ser executada em ambientes Unix/Linux e Windows.

A biblioteca OpenMP é gerenciada por uma associação sem fins lucrativos, a OpenMP ARB (OpenMP Architecture Review Board). Essa organização é composta um grupo de empresas de *hardware* e *software*, tais como AMD, IBM, Intel, Cray, HP, Fujitsu, Nvidia, NEC, Microsoft, Texas Instruments e Oracle Corporation.

Essa biblioteca é constituída por um conjunto de diretivas de compilador, rotinas de biblioteca para serem executadas em tempo de execução e variáveis de ambiente que influenciam o comportamento do tempo de execução. A inserção adequada das diretivas OpenMP em um programa sequencial deverá permitir que muitas das aplicações se beneficiem da arquitetura paralela com memória compartilhada e com o mínimo de modificação no código. Além disso, o programador não precisa se preocupar com aspectos de sincronização e comunicação. Com esses procedimentos, os compiladores geram automaticamente o código paralelo.

A paralelização é realizada explicitamente com múltiplas *threads* dentro de um mesmo processo. A criação de *threads* é uma forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas simultaneamente. Cada *thread* possui sua própria pilha de execução; porém, compartilhando o mesmo endereço de memória com as outras *threads* do mesmo processo. Com esse modelo, uma só *thread* executa as instruções do código até encontrar uma região paralela, identificada por uma diretiva especifica. Nesse ponto, a *thread* cria um grupo de outras *threads*, que juntas continuam a execução do código paralelo. Quando as *threads* completam a execução, elas são sincronizadas e somente a *thread* inicial segue na execução do código até que uma nova região paralela seja encontrada ou que essa *thread* seja encerrada.

Entre suas principais vantagens, destacam-se o alto desempenho em diferentes plataformas, é portátil e de fácil manutenção. Além disso, permite o ajuste dinâmico da quantidade de *threads* e oferece suporte ao paralelismo aninhado. Como é uma biblioteca projetada, para lidar com E/S paralela, o programador é responsável pela sincronização de entrada e saída de dados (MARZULO, 2011).

Como desvantagem da biblioteca OpenMP, existem aplicações, como a paralelização de algoritmos irregulares ou com padrões de paralelismo similares a um *pipeline*, que são complexos de serem implementados (MARZULO, 2011). Além disso, laços de repetição paralelos geralmente contêm barreiras ao seu término. A sincronização imposta por uma barreira deste tipo impede que blocos de *threads* sem dependência de dados prossigam com segurança. Desse modo, ao utilizar tais construções, grandes potenciais de paralelismo podem ser descartados.

2.4.2 Pthreads

Pthreads (<https://computing.llnl.gov/tutorials/pthreads/>) são implementações do padrão POSIX *threads*, padronizada pelo IEEE POSIX 1003.1c. É uma biblioteca portável de *threads* e eficaz, em arquiteturas *multicore*, cujo fluxo de processo é agendado para ser executado em outros núcleos. O foco das funções padrão dessa biblioteca está na criação, destruição e sincronização de *threads*.

Ganhos de desempenho computacional também são encontrados em sistemas com um único núcleo. Para isso, a biblioteca consegue explorar a latência de E/S e outras funções do sistema que podem interromper a execução do processo. A sincronização de *threads* é realizada por *mutexes*, *joins* ou variáveis de condição. *Mutexes* garantem exclusão mútua durante a atu-

alização de uma variável global e *joins* são utilizadas para bloquear uma região crítica, que é a região compartilhada pelas *threads*.

Essa biblioteca é composta por um conjunto de tipos de dados e rotinas implementadas em linguagens de programação C. O acesso às rotinas é realizado, por meio do cabeçalho *pthread.h*, embora a biblioteca possa fazer parte de outras implementações, como a *libc*.

A interface Pthreads é composta por mais de 60 subrotinas e todos os identificadores na biblioteca começam com o prefixo *pthread_*. A fase de ligação com qualquer biblioteca dinâmica ou estática, quando necessária, é realizada conforme o sistema operacional.

Na biblioteca Pthreads, as *threads* compartilham o mesmo espaço de endereço dentro de um único processo. Nesse caso, pode ser tão eficiente computacionalmente quanto simplesmente passar um ponteiro por referência. No pior caso, as comunicações, utilizando Pthreads, são mais numerosas, quando envolvem comunicação da *cache* para CPU ou da memória para CPU. Sincronização de *threads*, localização de erros e imprevisibilidade na execução do código são alguns exemplos de problemas encontrados em aplicações que utilizam essa biblioteca.

Aplicações utilizando Pthreads oferecem ganhos potenciais de desempenho e vantagens práticas em relação às aplicações que não utilizam *threads*. A seguir, são sumarizadas as principais vantagens dessa biblioteca.

- Programação flexível com um nível de abstração relativamente elevado.
- Programação prioritária em tempo real: tarefas importantes podem ser programadas para substituir ou interromper tarefas de menor prioridade.
- Biblioteca portável de *threads* criada para fornecer uma interface de programação comum para diferentes sistemas operacionais.
- Funções como determinar prioridades de *threads* não fazem parte da biblioteca Pthreads padrão, mas podem ser disponibilizadas por implementações específicas da biblioteca.

3 IMPLEMENTAÇÕES

Neste Capítulo, são mostradas as implementações dos métodos propostos neste trabalho. Na seção 3.1, é mostrado o método proposto por George e Liu (1979), utilizado para encontrar vértices pseudoperiféricos. O método RCM-GL é descrito na seção 3.2. Na seção 3.3, é apresentada a heurística KP-band. Na seção 3.4, é mostrado um exemplo de reordenação dos vértices em uma matriz simétrica de ordem cinco. Por fim, as paralelizações dos algoritmos RCM-GL, KP-band e ICCG são descritas na seção 3.5.

3.1 Algoritmo de George e Liu (1979)

Neste trabalho, foi utilizado o algoritmo descrito por George e Liu (1979) que computa um par de vértices pseudoperiféricos, ou seja, no pseudodiâmetro do grafo. De acordo com resultados obtidos pelas heurísticas sequencias, descritas na bibliografia, as melhores ordenações são geradas por vértices pseudoperiféricos. Isso ocorre, porque a largura de banda está relacionada à largura da estrutura de nível utilizada no esquema de ordenação, e esses vértices tendem a produzir estruturas de nível longas e estreitas (OLIVEIRA; CHAGAS, 2014).

Para encontrar os vértices pseudoperiféricos, pode-se aplicar a busca em largura em todos os vértices do grafo. Gibbs, Poole e Stockmeyer (1976) foram os primeiros autores a utilizarem um vértice pseudoperiférico como vértice inicial para a renumeração. A distância que cada vértice possui, em relação ao vértice inicial, representa o nível em que o respectivo vértice está na estrutura de nível. Por exemplo, se o vértice inicial está no nível 1, os vértices adjacentes ao vértice inicial estão no nível 2, os vértices adjacentes a cada um dos vértices no nível 2 estão no nível 3 e, assim, sucessivamente.

Outro algoritmo, para encontrar vértices pseudoperiféricos, foi apresentado por George e Liu (1979). Inicialmente, um vértice qualquer $v \in V$ é escolhido como vértice inicial. Em seguida, a estrutura de nível do vértice v é gerada. Depois, seleciona-se um vértice u de grau mínimo pertencente ao último nível da estrutura de nível do vértice v. Gera-se a estrutura de nível do vértice u e verifica-se se a excentricidade de v é menor do que a excentricidade de u. Se essa condição é satisfeita, o vértice u passa a ser o vértice v e o processo é repetido. O pseudocódigo do algoritmo sequencial de George e Liu (1979), para encontrar vértices pseudoperiféricos, é descrito no Algoritmo 3 (OLIVEIRA; CHAGAS, 2014).

Algoritmo 3 - Algoritmo de George e Liu (1979)

Entrada: grafo $G = (V, E)$;
Saída: vértices pseudoperiféricos.

1: selecione um vértice v de grau mínimo;
2: constrói-se a estrutura de nível do vértice v obtendo-se $L(v)$;
3: **repita**
4: escolha um vértice de grau mínimo u pertencente ao último nível da estrutura de nível enraizada em v;
5: constrói-se a estrutura de nível do vértice u obtendo-se $L(u)$;
6: **se** (excentricidade(u) > excentricidade(v)) **então**
7: $v \leftarrow u$;
8: $L(v) \leftarrow L(u)$;
9: **fim se**
10: **até** ($u \neq v$)
11: **retorna** v.

3.2 Método RCM-GL

Proposto por Cuthill e McKee (1969), o algoritmo Cuthill-Mckee (CM) é um método de reordenamento de linhas e colunas aplicado a matrizes esparsas para reduções de largura de banda e de *profile*. Posteriormente, George (1971) demonstrou que a solução do método CM pode ser melhorada simplesmente invertendo a ordem de renumeração final dos vértices. Desse modo, o método CM resulta em um *profile* tão bom quanto o *profile* sem inversão da renumeração. Além disso, a largura de banda não é alterada (LIU; SHERMAN, 1976). Para mais detalhes sobre as heurísticas CM e RCM, veja Oliveira e Chagas (2014).

O método RCM-GL (GEORGE; LIU, 1981), baseado no método RCM (GEORGE, 1971), é iniciado por um vértice pseudoperiférico de George e Liu (1979). Portanto existe uma restrição, na reordenação das linhas e colunas da matriz de coeficientes, representada por um grafo: os vértices são percorridos em ordem nível em nível na estrutura de nível enraizada no vértice inicial. Além disso, vértices do mesmo nível são descobertos em ordem crescente de grau. No final do algoritmo, todos os vértices do grafo são descobertos. Essa ordem corresponde à renumeração dos vértices. Por fim, a ordem de renumeração dos vértices é invertida.

O pseudocódigo do método RCM-GL sequencial é mostrado no Algoritmo 4 (OLIVEIRA, 2016). Nas linhas 1 - 3, são inicializadas as variáveis do algoritmo. A variável s, na linha 3, representa o vértice inicial escolhido para iniciar a execução do algoritmo. Nas linhas 4 - 10, os vértices do grafo são descobertos. Nessa etapa, os vértices são renumerados e o nível de cada um desses vértices é identificado na estrutura de nível enraizada no vértice inicial

s. Na linha 11, é realizada a inversão da numeração dos vértices do grafo. Por fim, na linha 12, a renumeração dos vértices no grafo G é retornada.

Algoritmo 4 - *Reverse* Cuthill-Mckee

Entrada: grafo $G \leftarrow (V,E)$ conexo, vértice inicial $s \in V$.
Saída: renumeração S do grafo G.

1: $i \leftarrow 1$;
2: $j \leftarrow 1$;
3: $S(1) \leftarrow s$;
4: **enquanto** $(i < |V|)$ **faça**
5: **para** $(w \in Adj(G,S(j)) - S(1),S(2),\cdots,S(i)$ em ordem crescente de grau) **faça**
6: $i \leftarrow i+1$;
7: $S(i) \leftarrow w$;
8: **fim para**
9: $j \leftarrow j+1$;
10: **fim enquanto**
11: inverta a numeração final de S.
12: **retorna** S.

3.3 Método KP-band

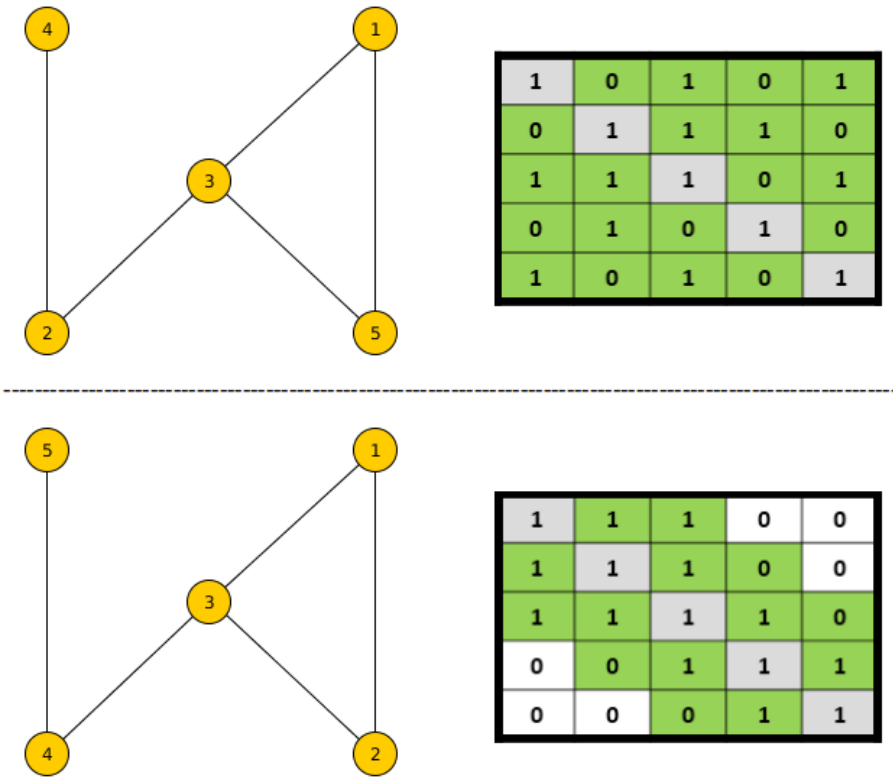
Koohestani e Poli (2011) empregaram a hiper-heurística *Genetic Programming Hyper-Heuristic for Bandwidth Reduction* (GPHH-band), na geração de heurísticas, para a redução de largura de banda. Segundo Cowling, Kendall e Soubeiga (2001), hiper-heurística é uma heurística de alto nível especializada na geração de heurísticas para um problema específico. Todas as heurísticas geradas pela hiper-heurística GPHH-band realizam a renumeração dos vértices de forma similar ao método RCM, ou seja, nível por nível na estrutura de nível enraizada gerada, a partir de um vértice pseudoperiférico.

A única diferença entre as heurísticas geradas pela GPHH-band e a heurística RCM-GL está na forma em que os vértices de cada nível da estrutura de nível enraizada são renumerados. No método RCM-GL, os vértices pertencentes a um mesmo nível na estrutura de nível enraizada são renumerados por ordem crescente de grau. No método KP-band, os vértices pertencentes a um mesmo nível na estrutura de nível enrizada são renumerados por ordem crescente do valor determinado pela fórmula $0,179492928171 \times SDV^3 + 0,292849834929 \times SDV^2 - 0,208926175433 \times |V| - 0,736485142138 \times |V| \times SDV - 1,77524579882 \times SDV - 1,75681383404$, em que V são os vértices do grafo e SVD é a soma dos graus dos vértices adjacentes ao vértice em questão. Essa equação foi encontrada pela hiper-heurística GPHH-Band, por meio de testes experimentais realizados por Koohestani e Poli (2011).

3.4 Exemplo de reordenação de vértices

Na figura 3.1, é exemplificada a reordenação de vértices de um grafo de ordem cinco. Ao lado de cada grafo está representada a sua respectiva matriz de coeficientes. Na reordenação dos vértices, foi utilizado o método CM, em que os vértices são descobertos (renumerados) em ordem crescente de grau. Nesse caso, o primeiro vértice encontrado pelo método CM foi o um, então, esse vértice continua com o rótulo um. O segundo vértice encontrado foi o cinco, então, esse vértice foi renumerado com o rótulo dois. O vértice três foi o terceiro a ser encontrado, então, ele foi renumerado com o rótulo três. Em seguida, os vértices dois e quatro foram encontrados nessa ordem, então, foram renumerados com os rótulos quatro e cinco, respectivamente. A largura de banda da matriz de coeficientes original é quatro. Com o reordenamento dos vértices, a largura de banda da matriz de coeficientes passou a ser dois.

Figura 3.1 – Exemplo de reordenação das linhas e colunas de uma matriz simétrica de ordem cinco.



Fonte: Do autor (2018)

3.5 Implementações paralelas

Nesta seção, são apresentadas as abordagens de paralelização empregadas nos algoritmos propostos. Na subseção 3.5.1, são descritas as paralelizações das heurísticas RCM-GL e KP-band. Na subseção 3.5.2, é mostrada a paralelização do método ICCG.

3.5.1 Implementações paralelas das heurísticas RCM-GL e KP-band

Baseada nas implementações paralelas de Karantasis et al. (2014) e Rodrigues, Boeres e Catabriga (2017c), uma implementação paralela do método RCM nivelado foi realizada. Para isso, foi utilizada a biblioteca OpenMP. O algoritmo é dividido em quatro etapas; contudo, com a execução das duas primeiras etapas, o tempo computacional ficou maior do que o tempo computacional obtido com a execução do algoritmo RCM-GL sequencial, desenvolvido por Oliveira, Bernardes e Chagas (2017). Nessas duas etapas, o nível de cada vértice é encontrado e a quantidade de vértices por nível é computada.

Na implementação do método RCM nivelado, não foram utilizadas as mesmas otimizações que Rodrigues, Boeres e Catabriga (2017a) utilizaram em suas implementações, tais como a estruturada de dado CSR. Possivelmente, essas diferenças fizeram que os tempos computacionais obtidos com a execução do método RCM nivelado não fossem melhores que os tempos computacionais obtidos com a execução do RCM-GL sequencial. Logo, a implementação do método RCM nivelado deve ser melhorada. Para isso, regiões críticas e gargalos do algoritmo devem ser investigados. Além disso, outras estruturas de dados podem ser incorporadas ao algoritmo. Na tabela 3.1, são mostrados os tempos computacionais obtidos com a execução dos algoritmos RCM nivelado e RCM-GL sequencial. As execuções foram realizadas nas instâncias *apache2*, *tmt_sym*, *G2_circuit*, *2cubes_sphere* e *Andrews*.

Tabela 3.1 – Resultados de experimentos com os algoritmos RCM nivelado e RCM-GL sequencial em cinco instâncias.

Instância	Dimensão	Quantidade de coeficientes não nulos	RCM nivelado com duas *threads*	RCM-GL sequencial
apache2	715.176	4.817.870	1.57	0,92
tmt_sym	726.713	5.080.961	1.09	0,73
G2_circuit	150.102	726.674	0,16	0,15
2cubes_sphere	101.492	1.647.264	0.35	0,19
Andrews	60.000	760.154	0,20	0,17

Fonte: Do autor (2018)

Como mencionado, o objetivo principal deste trabalho é propor implementações paralelas de algoritmos sequencias, existentes na bibliografia, que sejam eficientes nas reduções de

largura de banda ou de *profile* de matrizes. Com isso, é esperado que se mantenha a qualidade das soluções quando comparadas com soluções obtidas de execuções com suas respectivas versões sequenciais. Além disso, com as implementações paralelas dessas heurísticas, espera-se melhorar o desempenho computacional do método ICCG paralelo. Desse modo, outra abordagem de paralelização foi adotada nas heurísticas RCM-GL e KP-band. Como essas heurísticas são semelhantes, diferindo apenas na estratégia utilizada, para percorrer os vértices adjacentes, a paralelização delas também é semelhante. Na implementação dos algoritmos, não foi utilizada nenhuma abordagem ou dica de paralelização descrita nos artigos encontrados na revisão sistemática da bibliografia.

De modo geral, na implementação dos algoritmos, foram utilizadas dois conjuntos de vértices A e B, compartilhados por todas as *threads*. Inicialmente, o vértice pseudoperiférico, é inserido no conjunto A. A partir desse vértice pseudoperiférico, seus adjacentes são percorridos de acordo com uma ordem específica (determinada pela heurística utilizada) e adicionados no conjunto B. À medida que os vértices do conjunto A são computados, eles são renumerados e removidos de A. Quando o conjunto A estiver vazio, os vértices do conjunto B passam para o conjunto A. O algoritmo termina, quando ambos os conjuntos A e B estão vazias. Nesse momento, todos os vértices já foram renumerados. Nessa abordagem de paralelização, os vértices do conjunto A são divididos entre as *threads*. Com isso, cada *thread* computa seus vértices e adiciona os vértices recém-renumerados no conjunto B. Para a ordenação dos vértices, foi utilizado o método *sort* da estrutura de dados *list*, que está disponível na biblioteca padrão da linguagem de programação C++ (Standard Template Library - STL).

A renumeração dos vértices em níveis na estrutura de nível enraizada nesses dois algoritmos restringe o paralelismo. Especificamente, todos os vértices no nível corrente podem ser computados simultaneamente, mas nenhum vértice no nível seguinte é computado até que todos os vértices no nível corrente sejam computados. Em consequência disso, as *threads* são sincronizadas ao final do processamento do nível corrente da estrutura de nível enraizada no vértice inicial.

Na implementação dos algoritmos, os vértices de cada nível na estrutura de nível enraizada no vértice inicial são particionados entre as *threads*. Para evitar que *threads* fiquem ociosas, o algoritmo foi implementado de modo que a quantidade de *threads* seja menor ou igual à quantidade de vértices do nível corrente na estrutura de nível enraizada no vértice inicial. Ao contrário da abordagem de Karantasis et al. (2014), a paralelização do método RCM-GL é base-

ada em um único passo. Operações como renumeração de vértices e escolha do próximo vértice a ser computado foram implementadas para serem executadas como operações atômicas. Por sua vez, a forma de se percorrer os vértices e a renumeração dos vértices são realizadas em paralelo obedecendo aos níveis da estrutura de nível enraizada no vértice inicial. Com isso, à medida que cada vértice é descoberto, ele é renumerado e seu nível é identificado.

As implementações paralelas dos métodos RCM-GL e KP-band não são versões fiéis às suas respectivas implementações sequenciais. Por exemplo, suponha que os vértices A e B sejam percorridos nessa ordem, então, na versão sequencial, os vértices adjacentes de A são percorridos antes que os vértices adjacentes de B sejam percorridos. Na versão paralela, essa restrição não é satisfeita: uma *thread* pode percorrer os vértices adjacentes de B antes que os vértices adjacentes de A sejam percorridos. Logo, podem ser geradas diferentes soluções para cada execução do algoritmo.

Neste projeto, as implementações paralelas das heurísticas RCM-GL e KP-band são ingênuas. O objetivo foi obter implementações que busquem reduzir largura de banda de matrizes esparsas com um tempo de computação menor que o tempo computacional de suas respectivas implementações sequencias. Em seguida, o intuito foi mostrar que, com essas reordenações de vértices, quando aplicadas como um pré-processamento para o método ICCG, é possível acelerar esse método para resolução de sistemas de equações lineares.

Uma característica utilizada, nos métodos heurísticos, para redução de largura de banda, é estabelecer que o vértice inicial seja um vértice pseudoperiférico (OLIVEIRA; CHAGAS, 2014). O algoritmo, utilizado neste presente trabalho, para encontrar o vértice pseudoperiférico tanto para o método RCM (GEORGE, 1971) quanto para a heurística KP-band (KOOHESTANI; POLI, 2011), foi proposto por George e Liu (1979), que foi executado, sequencialmente, tal como foi realizado no trabalho de Karantasis et al. (2014). Uma boa escolha dos vértices pseudoperiféricos resulta em uma estrutura de nível rotulada longa e estreita (OLIVEIRA; CHAGAS, 2014).

O pseudocódigo das heurísticas RCM-GL e KP-band é mostrado no Algoritmo 5. Nas linhas 1-3, a variável i e os conjuntos de vértices A e S são inicializados. O conjunto A é utilizado, para armazenar os vértices, que estão sendo processados na iteração atual e que já foram renumerados. O conjunto S recebe os vértices renumerados. A variável s é um vértice pseudoperiférico de George e Liu (1979) que é renumerado e inserido no conjunto A nas linhas 2 e 3, respectivamente. No laço da linha 4, é garantido que todos os vértices do conjunto A

sejam renumerados. À medida que os vértices adjacentes são renumerados, na linha 9, eles são inseridos no conjunto B. Quando todos os vértices adjacentes são renumerados, o conjunto A, que estará vazio, recebe os vértices do conjunto B na linha 14. Na linha 15, o conjunto B é esvaziado. Desse modo, os vértices que estão no conjunto A, na linha 7, estão no nível l. Por sua vez, os vértices que estão sendo renumerados, estão no nível $l+1$.

A principal diferença entre os algoritmos RCM-GL e KP-band é mostrada no laço da linha 8. Nessa etapa, os vértices adjacentes são descobertos em ordem crescente de acordo com a heurística utilizada. A quantidade de *threads*, criada para a execução do algoritmo, é determinada na linha 5. Na linha 6, é representada a diretiva do padrão OpenMP utilizada na paralelização do laço *for* na linha 7. O parâmetro *shared* indica que os conjuntos A e B são compartilhados. O parâmetro *private* denota que cada *thread* possui um valor local para a variável *currentVertex*. Por fim, o parâmetro *num_threads* determina a quantidade de *threads* necessária para a execução do código paralelo.

Algoritmo 5 - Algoritmo paralelo para as heurísticas RCM-GL e KP-band

Entrada: grafo $G \leftarrow (V,E)$ conexo, vértice inicial $s \in V$.
Saída: renumeração S do grafo G.

```
 1:  i ← 1;
 2:  S(i) ← s;
 3:  A ← s;
 4:  enquanto (NaoEstaVaria(A)) faça
 5:      nthreads = (A.size() < max_total_threads) ? visitationOrder.size() : max_total_threads;
 6:      #pragma omp parallel shared(B, A) private (currentVertex) num_threads(nthreads)
 7:      para (v ∈ A) faça
 8:          para (w ∈ Adj(G,S(v)) − S(1),S(2),···,S(i)) em ordem crescente faça
 9:              i ← i + 1;
10:              S(i) ← w;
11:              B ← w;
12:          fim para
13:      fim para
14:      A ← B;
15:      Esvazie(B);
16:  fim enquanto
17:  retorna S.
```

3.5.2 Paralelização do método ICCG

O precondicionador escolhido como precondicionamento, para o método dos gradientes conjugados, foi a fatoração incompleta de Cholesky *shifted* com zero *fill-in* (IC(0)). Esse precondicionador é baseado na fatoração da matriz de coeficientes A, de dimensão $(n \times n)$, na

forma $M = L \cdot L^T$, em que L é uma matriz triangular inferior, tal que $L = (l_{ij})$ e $(l_{ij}) = 0$ se $i < j$, e L^T é a transposta. Esse precondicionador é utilizado nos casos em que a matriz de coeficientes é do tipo positiva definida e possui dominância da diagonal principal (GOLUB; LOAN, 2012). Por motivos de eficiência computacional, a estrutura CSR/CSC *skyline* foi utilizada para armazenar os sistemas de equações lineares (GKOUNTOUVAS et al., 2013). Posteriormente, essa estrutura de dados foi passada para o método ICCG que resolve o sistema de equações lineares.

Na implementação do precondicionador IC(0), baseada na implementação sequencial de Oliveira, Bernardes e Chagas (2017), o código foi dividido em três etapas e cada etapa está relacionada a um laço de repetição. Na primeira delas, a matriz triangular inferior é percorrida e, para cada um dos coeficientes não nulos, seu valor é dividido pela raiz quadrada de um determinado coeficiente na diagonal principal da matriz de coeficientes. Na segunda etapa, uma computação envolvendo soma e multiplicação é realizada para cada um dos coeficientes não nulos da matriz triangular inferior. Nessa etapa, os coeficientes da diagonal principal também são envolvidos na computação. Para facilitar a realização dessas computações, foi necessária a utilização de um vetor adicional que, na última etapa, as posições desse vetor (que foram modificadas nas duas etapas anteriores) são estabelecidas como zero. Com isso, cada uma das etapas foi paralelizada adicionando as diretivas adequadas do padrão OpenMP.

No Algoritmo 6, é mostrado o pseudocódigo do algoritmo IC(0) paralelo. A entrada desse algoritmo é a matriz de coeficientes A e a saída é a matriz de coeficientes precondicionada M. O vetor auxiliar, utilizado nas computações do algoritmo, é representado por *column(n)*, em que n é a dimensão da matriz de coeficientes. Os parâmetros $M.D$, $M.C$ e $M.P$ são vetor dos coeficientes da diagonal principal, vetor de coeficientes da matriz estritamente triangular inferior e vetor de onde se inicia cada linha no vetor $M.L$, respectivamente. Além do vetor auxiliar, o vetor *M.D(j)->l* é utilizado nas operações envolvendo o precondicionador IC(0), em que j é o índice do coeficiente não nulo no respectivo vetor. As variáveis *l_kk* e *invl_kk* são utilizadas, para calcular o valor inverso dos coeficientes, localizados na diagonal principal da matriz M.

Inicialmente, na linha 1, a matriz de coeficientes M é inicializada com a matriz de coeficientes A. No laço da linha 4, existem outros três laços de repetição que são referidos como três etapas. Na primeira etapa, linhas 18 - 22, todos os coeficientes não nulos da matriz triangular inferior e da coluna k são computados. Nessa computação, os valores dos coeficientes não nulos são invertidos e o valor de M_{ik} é adicionado no vetor auxiliar, em que i e k corres-

pondem às linhas e colunas da matriz de coeficientes M. Na segunda etapa, linhas 26 - 36, o valor final na posição M_{ij} é calculado, em que i e j correspondem às linhas e colunas da matriz de coeficientes M. Para encontrar os valores de $M.D(j)->l$ e $(M_{jk} * M_{jk})$, necessário por M_{ij}, o vetor auxiliar é utilizado. Na terceira etapa, linhas 40 - 43, as posições do vetor auxiliar que foram modificadas nas etapas anteriores são estabelecidas como zero. Cada uma dessas etapas foi paralelizada adicionando as diretivas adequadas do padrão OpenMP. Por fim, na linha 45, a matriz precondicionada M é retornada.

O método dos gradientes conjugados sequencial utilizado (OLIVEIRA; BERNARDES; CHAGAS, 2017) é baseado na descrição de Saad (2003). Nessa implementação, também existem várias etapas baseadas em laços de repetição. Cada laço de repetição é dependente do laço anterior, ou seja, existe uma dependência de dados durante a execução desse algoritmo. Além disso, a estrutura de dados CSR/CSC *skyline* utilizada gera uma otimização, no desempenho computacional do método ICCG, dificultando uma paralelização ingênua desse método. Assim, somente duas etapas do algoritmo sequencial foram paralelizadas. A primeira delas corresponde à etapa em que a solução inicial do algoritmo é gerada. Na outra etapa, uma aproximação da solução é gerada, em que existem várias operações de soma e multiplicação (SAAD, 2003).

Algoritmo 6 - Algoritmo paralelo da fatoração incompleta de Cholesky (IC(0)

Entrada: matriz de coeficientes A;
Saída: matriz de coeficientes precondicionada M.

```
 1: M = A;
 2: Vetor column(n);
 3:
 4: para (k=0 até n) faça
 5:     se (k == 0) então
 6:         l_kk = √(M.C(k));
 7:     senão se (M.D(j)->l < 0) então
 8:         l_kk = √(−M.D(k) → l);
 9:     senão
10:         l_kk = √(M.D(k) → l);
11:     fim se
12:
13:     M.D(k)->l = l_kk;
14:     invl_kk = 1/l_kk;
15:
16:     // Primeira etapa de paralelização
17:     #pragma omp parallel for shared(M, column)
18:     para (i=0 até n) faça
19:         r = L(i);
20:         M.C(i) *= invl_kk;
21:         column(r) = M.C(i);
22:     fim para
23:
24:     // Segunda etapa de paralelização
25:     #pragma omp parallel for shared(M, column)
26:     para (j=k+1 até n) faça
27:         se (k == 0) então
28:             M.D->l = M.D(j) - (column(j)*column(j));
29:         senão
30:             M.D(j)->l -= (column(j)*column(j));
31:         fim se
32:         para (i=0 até n) faça
33:             c = M.L(i);
34:             M.C(i) -= column(c)*column(j);
35:         fim para
36:     fim para
37:
38:     // Terceira etapa de paralelização
39:     #pragma omp parallel for shared(M, column)
40:     para (i=0 até n) faça
41:         r = M.L(i);
42:         column(r) = 0;
43:     fim para
44: fim para
45: retorna M.
```

4 METODOLOGIA

Neste capítulo, são apresentados os materiais necessários e a metodologia; os detalhes de implementação do precondicionador, utilizado em conjunto com o método dos gradientes conjugados e os detalhes de implementações das heurísticas, utilizadas, no desenvolvimento deste trabalho, além das especificações e ferramentas utilizadas nas simulações computacionais.

A busca por trabalhos que abordaram métodos paralelos para reduções de largura de banda ou de *profile* de matrizes foi realizada utilizando o portal de trabalho científico *Scopus*®. Cada documento encontrado foi analisado com o objetivo de identificar quais foram os melhores métodos utilizados, para se obter a sua paralelização. Entre as diversas heurísticas que existem na bibliografia, as heurísticas RCM-GL (GEORGE; LIU, 1981) e KP-band (KOOHESTANI; POLI, 2011) foram escolhidas para serem implementadas com a utilização de duas bibliotecas paralelas: OpenMP e Pthreads. O método RCM é bem difundido na bibliografia e consegue gerar bons resultados. A heurística KP-band, baseada no método RCM, ainda não foi paralelizada.

Para as simulações dos algoritmos, foram utilizados computadores com processador Intel® Core(TM) i7-4770 de 3,40 Ghz com quatro núcleos e duas *threads* por núcleo. Além disso, são computadores com 8Mb de memória *cache* e 8Gb de memória RAM DDR3 1333MHz. O sistema operacional utilizado foi uma distribuição Linux Ubuntu 16,04 LTS 64 bits com *kernel* versão 4.10.0-42-*generic*. Especificamente, o compilador empregado foi o g++ versão 5.4.0 em cuja compilação foi utilizada a *flag* -O3.

A linguagem de programação utilizada na implementação dos algoritmos foi C++, por se tratar de uma linguagem de programação de alto desempenho. Os experimentos com os algoritmos desenvolvidos foram realizados com instâncias da base *SuiteSparse Matrix Collection* (<https://www.cise.ufl.edu/research/sparse/matrices/>) (DAVIS; HU, 2011).

As heurísticas paralelas projetadas em arquiteturas *multicore* foram aplicadas como um pré-processamento de matrizes esparsas, para o método dos gradientes conjugados precondicionado paralelo que, por sua vez, foi aplicado na resolução de sistemas de equações lineares de grande porte. O precondicionador utilizado em conjunto com o método dos gradientes conjugados paralelo foi a fatoração incompleta de Cholesky, também, em paralelo.

Como são matrizes esparsas e de grande porte, geralmente, a representação dos dados é realizada de forma compacta por estruturas de dados como *Compressed Row Storage* (CRS) e *Compressed Column Storage* (CCS) (OLIVEIRA, 2015). Neste projeto, foi utilizada

a estrutura de dados CRS-SSS, que é uma combinação das estruturas de dados CRS e *Skyline Storage Scheme* (SSS) (FELIPPA, 1975). Essa estrutura de dados proporciona uma capacidade de acesso rápido aos dados que, por sua vez, são armazenados de forma compacta. Para isso, os coeficientes da diagonal principal são armazenados separadamente e os coeficientes não nulos da matriz triangular inferior são armazenados como no formato CSR/CSC (OLIVEIRA, 2015).

Com a paralelização e possíveis estratégias de paralelização dos algoritmos, independentemente da biblioteca utilizada, o desempenho computacional dos métodos implementados foi avaliado e comparado com o desempenho computacional de outros métodos, encontrados na revisão sistemática. Desse modo, espera-se, com a paralelização dos métodos, obter desempenho computacional melhor em relação ao desempenho computacional dos principais métodos existentes na bibliografia.

Para o desenvolvimento deste trabalho, utilizou-se como base o projeto denominado LinearSystemSolver, desenvolvido por Oliveira, Bernardes e Chagas (2017). Nesse projeto computacional, foram implementadas as heurísticas para redução de largura de banda e *profile*, selecionadas nas revisões sistemáticas realizadas por Chagas e Oliveira (2015) e Oliveira e Chagas (2015). Nesse código, a precisão numérica utilizada para as execuções do método ICCG foi de 10^{-8}. Com essa precisão, obteve-se um desempenho computacional satisfatório nas execuções com o método ICCG.

É importante ressaltar que o objetivo deste trabalho não foi superar os resultados das implementações originais das heurísticas, mas obter implementações paralelas razoavelmente eficientes, de modo que se possa avaliar o desempenho computacional dos algoritmos implementados. A comparação dos experimentos será realizada por meio da qualidade das soluções e do *speedup*.

5 SIMULAÇÕES

Neste Capítulo, apresentam-se os resultados obtidos com a execução das heurísticas paralelas para a redução de largura de banda e *profile*. Além disso, são mostrados os resultados das execuções com a resolução de SELs pelo método dos gradientes conjugados precondicionado pela fatoração incompleta de Cholesky com zero fill-in (ICCG) nas instâncias da base *SuiteSparse Matrix Collection* (DAVIS; HU, 2011). As duas heurísticas para a redução de largura de banda e *profile* foram testadas nessas matrizes.

O método RCM, bastante difundido na bibliografia, foi paralelizado por diversos autores como Karantasis et al. (2014) e Rodrigues, Boeres e Catabriga (2016). Nesse trabalho, esse método foi implementado em paralelo a partir da versão sequencial de Oliveira, Bernardes e Chagas (2017). Por sua vez, a heurística KP-band (KOOHESTANI; POLI, 2011) é descrita como uma das heurísticas mais promissoras para áreas de aplicação específicas (OLIVEIRA; BERNARDES; CHAGAS, 2018).

O objetivo foi encontrar as heurísticas paralelas que obtivessem os melhores resultados em comparação com as respectivas versões sequenciais. Dessa forma, uma nova abordagem de paralelização das heurísticas propostas foi adotada, ao contrário das abordagens existentes na bibliografia. Os tempos computacionais foram definidos de forma empírica. Além disso, procuraram-se identificar as melhores heurísticas paralelas que conseguiram reduzir a largura de banda das matrizes.

Na Tabela 5.1, são mostradas as instâncias utilizadas nos experimentos que são compostas por matrizes esparsas, simétricas e positivas definidas. Essas instâncias foram obtidas da base *SuiteSparse Matrix Collection* (DAVIS; HU, 2011). Na execução do método ICCG com essas instâncias, obtiveram-se altos *speedups*, de acordo com a publicação de Oliveira, Bernardes e Chagas (2017), em que foram realizadas comparações entre as execuções do método ICCG em sequencial com e sem reordenação dos vértices. Na reordenação dos vértices, também foram utilizadas as heurísticas RCM-GL e KP-band. Além disso, essa tabela contém informações como nome da instância, dimensão da matriz de coeficientes, esparsidade, largura de banda e *profile* original e, por fim, o tipo de problema ao qual a instância pertence.

Primeiramente, na seção 5.1, são mostrados os resultados obtidos com execuções das heurísticas RCM-GL e KP-band sequenciais e paralelas utilizando as bibliotecas OpenMP e Pthreads. Em seguida, na seção 5.2, para cada instância, resolveu-se o sistema de equações lineares pelo método ICCG paralelo com e sem as reduções de largura de banda e de *profile*

Tabela 5.1 – Instâncias utilizadas nos experimentos.

Instância	Dimensão	Coef. não nulos	β	*Profile*
G3_circuit	1.585.478	7.660.826	947.128	119.101.864.114
bone010	986.703	47.851.783	13016	8.846.266.758
audikw_1	943.695	77.651.847	925.946	3.671.706.312
tmt_sym	726.713	5.080.961	1.921	593.531.565
apache2	715.176	4.817.870	65.837	1.717.106.409
parabolic_fem	525.825	3.674.625	525.820	1.196.569.325
offshore	259.789	4.242.673	237.738	3.588.201.815
G2_circuit	150.102	726.674	93.719	1.098.802.048
bmw7st_1	141.347	7.318.399	121.856	1.371.669.955
shipsec1	140.874	3.568.176	5.237	431.771.001
boneS01	127.224	5.516.602	3.722	331.330.356
cfd2	123.440	3.085.406	4.501	156.107.322
x104	108.384	8.713.602	4.463	179.250.468
thermomech_TC	102.158	711.558	102.138	2.667.823.445
thermomech_TK	102.158	711.558	102.138	2.667.823.445
2cubes_sphere	101.492	1.647.264	100.407	483.241.271
shipsec1	140.874	3.568.176	5.237	431.771.001
thermal1	82.654	574.458	80.916	175.625.317
Andrews	60.000	760.154	59.925	1.501.971.521

Fonte: Do autor (2018)

pelas heurísticas RCM-GL e KP-band. Também foi executado o método ICCG com e sem reordenação em sequencial. Nas execuções com o método ICCG paralelo, a biblioteca utilizada na implementação dos algoritmos foi o OpenMP.

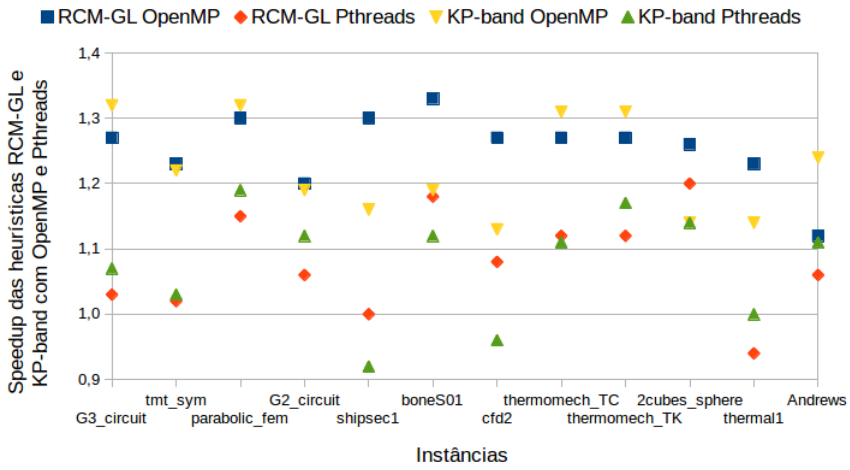
5.1 Experimentos com as heurísticas RCM e KP-band paralelas utilizando as bibliotecas OpenMP e Pthreads

Cada um dos algoritmos implementados foi executado 3 vezes, em sequencial e em paralelo. Nas execuções das implementações paralelas, o algoritmo foi executado com 2 a 4 *threads*. Em investigações exploratórias com até 12 *threads*, verificou-se que não houve *speedup* melhor do que com duas *threads*.

Nas Tabelas 5.2 e 5.3, são mostrados os tempos computacionais resultantes das execuções dos algoritmos RCM-GL e KP-band, respectivamente, implementados com as bibliotecas OpenMP e Pthreads. Nas primeiras colunas, são mostrados os nomes das instâncias. Em seguida, é mostrada a quantidade de *threads* (#) utilizada na execução do algoritmo. Nesse caso, a execução sequencial do algoritmo é representado por 1. O tempo computacional do algoritmo utilizado, para encontrar o vértice pseudoperiférico de George e Liu (GEORGE; LIU, 1981), é mostrado na coluna indicada por GL. O símbolo % representa a porcentagem de redução de largura de banda da heurística comparada com a largura de banda original da matriz. Para cada

instância, os melhores resultados, em relação às versões com as bibliotecas OpenMP e Ptrheads, são destacados em negrito. Na Figura 5.1, são mostrados os *speedups* obtidos com a execução dos algoritmos RCM-GL e KP-band, implementados com as bibliotecas OpenMP e Pthreads.

Figura 5.1 – *Speedup* das heurísticas implementadas com OpenMP e Pthreads.



Fonte: Do autor (2018)

Em algumas instâncias como *G2_circuit* e *thermal1*, o tempo de execução das implementações paralelas com a biblioteca OpenMP, de ambas as heurísticas, foi reduzido em quase 50% em comparação com os tempos de execução das implementações sequencias. Entretanto, os resultados foram piores nessas duas instâncias com a implementação utilizando a biblioteca Pthreads. Na instância *G3_circuit*, que é a maior das instâncias testadas, o *speedup* com o método RCM-GL (GEORGE; LIU, 1981) (KP-band (KOOHESTANI; POLI, 2011)) foi de 1,27 e 1,03 (1,32 e 1,07) nas implementações com as bibliotecas OpenMP e Pthreads, respectivamente. Em geral, baseado nos experimentos, verifica-se que a biblioteca OpenMP é mais adequada para ser utilizada em implementações das heurísticas RCM-GL (GEORGE; LIU, 1981) e KP-band (KOOHESTANI; POLI, 2011) do que a biblioteca Pthreads. Com duas *threads*, as execuções com OpenMP foram, em média, 1,2 vezes mais rápidas do que as execuções com Pthreads. Contudo é necessária uma análise aprofundada, nos códigos implementados, para descobrir as possíveis causas desses resultados.

Na Tabela 5.4, são mostradas comparações entre reduções de largura de banda das 12 matrizes testadas. Nessa tabela, são mostrados os tempos de execução sequenciais do algoritmo RCM com a biblioteca de rotinas HSL (<http://www.hsl.rl.ac.uk>) e com os *softwares* Octave (<https://www.gnu.org/software/octave>) (versão 4.2.1) e Matlab (<https://www.mathworks.com/products/matlab.html>) (versão 8.4.0.150421 - R2014b). Além disso, são mostradas os

Tabela 5.2 – Resultados de experimentos com 12 instâncias utilizando o método RCM-GL (GEORGE; LIU, 1981).

Instância	#	GL	OpenMP					Pthreads				
			β	*Profile*	RCM t(s)	%	Speedup	β	*Profile*	RCM t(s)	%	Speedup
G3_ circuit	–	–	947128	119101864114	–	–	–	947128	119101864114	–	–	–
	1	0,9	**5088**	1337252610	0,91	99	–	**5088**	1337252610	0,91	99	–
	2		7584	1510776039	0,53	99	**1,27**	5095	1341825072	0,85	99	1,03
	3		8460	1526034712	0,65	99	1,17	5095	1347250640	0,87	99	1,02
	4		8958	1503310693	0,87	99	1,02	5103	1362334685	0,98	99	0,96
tmt_ sym	–	–	1921	593531565	–	–	–	1921	593531565	–	–	–
	1	0,4	**1142**	623279883	0,45	41	–	**1142**	623279883	0,45	41	–
	2		1144	623396148	0,29	40	**1,23**	1143	622977422	0,43	40	1,02
	3		1144	623209105	0,42	40	1,04	1147	623213826	0,51	40	0,93
	4		1144	623209105	0,55	40	0,89	1147	623213826	0,62	40	0,83
parabolic _fem	–	–	525820	1196569325	–	–	–	525820	1196569325	–	–	–
	1	0,3	**514**	225052075	0,39	100	–	**514**	225052075	0,39	100	–
	2		517	225025530	0,23	100	**1,30**	517	225150252	0,30	100	1,15
	3		521	225138414	0,30	100	1,15	528	225445034	0,40	100	0,99
	4		524	225349599	0,38	100	1,01	531	225447811	0,45	100	0,92
G2_ circuit	–	–	93719	1098802048	–	–	–	93719	1098802048	–	–	–
	1	0,1	1954	179299511	0,08	98	–	1954	179299511	0,08	98	–
	2		**1953**	178665667	0,05	98	**1,20**	1955	179054798	0,07	98	1,06
	3		1958	179426060	0,06	98	1,13	**1954**	179680048	0,09	98	0,95
	4		1960	179597586	0,08	98	1,00	1964	180029643	0,10	98	0,90
shipsec1	–	–	5237	431771001	–	–	–	5237	431771001	–	–	–
	1	0,2	5939	450339567	0,41	-13	–	5939	450339567	0,42	-13	–
	2		5939	450025485	0,27	-13	**1,30**	5939	450025485	0,41	-13	1,00
	3		5939	450339567	0,50	-13	0,87	5939	450339567	0,50	-13	0,87
	4		5939	450025485	0,68	-13	0,69	5939	450025485	0,70	-13	0,68
boneS01	–	–	3722	331330356	–	–	–	3722	331330356	–	–	–
	1	0,2	**7621**	398438037	0,40	-105	–	7621	398438037	0,40	-105	–
	2		8072	403977156	0,25	-117	**1,33**	8451	434182443	0,31	-127	1,18
	3		8163	414690087	0,39	-119	1,02	**7476**	393009999	0,43	-104	0,95
	4		8251	397041713	0,57	-119	0,78	8157	414729676	0,54	-119	0,81
cfd2	–	–	100407	483241271	–	–	–	100407	483241271	–	–	–
	1	0,1	**2270**	162467287	0,18	98	–	2270	162467287	0,18	98	–
	2		2354	162826327	0,12	98	**1,27**	**2259**	162040137	0,16	98	1,08
	3		2342	162509363	0,21	98	0,90	2543	163993685	0,21	97	0,90
	4		2438	163480073	0,27	98	0,76	2343	162577436	0,30	98	0,70
thermo-	–	–	102138	2667823445	–	–	–	102138	2667823445	–	–	–
	1	0,1	268	17849234	0,09	100	–	268	17849234	0,09	100	–
	2		280	17914230	0,05	100	**1,29**	**253**	17725979	0,07	100	1,12
	3		**261**	17789298	0,06	100	1,20	272	17851343	0,08	100	1,06
	4		273	17861946	0,07	100	1,13	299	17808271	0,09	100	1,00
thermo- mech_TK	–	–	102138	2667823445	–	–	–	102138	2667823445	–	–	–
	1	0,1	253	17730868	0,09	100	–	**253**	17730868	0,09	100	–
	2		**252**	17736727	0,05	100	**1,27**	278	17913796	0,07	100	1,12
	3		265	17811771	0,06	100	1,19	271	17839780	0,08	100	1,06
	4		255	17754953	0,07	100	1,12	292	17843140	0,09	100	1,00
2cubes _sphere	–	–	100407	483241271	–	–	–	100407	483241271	–	–	–
	1	0,1	4738	272854596	0,14	95	–	4738	272854596	0,14	95	–
	2		4887	280752830	0,09	95	**1,26**	4776	267290332	0,10	95	1,20
	3		**4737**	262695916	0,11	95	1,14	4856	280443875	0,13	95	1,04
	4		4804	271131411	0,14	95	1,00	**4676**	279419040	0,15	95	0,96
thermal1	–	–	80916	175625317	–	–	–	80916	175625317	–	–	–
	1	0,1	231	12088938	0,06	100	–	**231**	12088938	0,06	100	–
	2		240	12118517	0,03	100	**1,23**	234	12181242	0,07	100	0,94
	3		**226**	12038528	0,05	100	1,07	253	12066862	0,09	100	0,84
	4		237	12124223	0,06	100	1,00	239	12079591	0,09	100	0,84
Andrews	–	–	59925	1501971521	–	–	–	59925	1501971521	–	–	–
	1	0,1	**16032**	459651430	0,09	73	–	**16032**	459651430	0,09	73	–
	2		16469	470700516	0,07	73	**1,12**	17017	503201048	0,08	72	1,06
	3		16471	490567257	0,08	73	1,06	17263	507071081	0,10	71	0,95
	4		16955	487501151	0,10	72	0,95	17100	504586725	0,11	71	0,90

Fonte: Do autor (2018)

Tabela 5.3 – Resultados de experimentos com 12 instâncias utilizando a heurística KP-band (KOOHESTANI; POLI, 2011).

Instância	#	GL	OpenMP					Pthreads				
			β	*Profile*	KP-band t(s)	%	Speedup	β	*Profile*	KP-band t(s)	%	Speedup
G3_ circuit	–	–	947128	119101864114	–	–	–	947128	119101864114	–	–	–
	1	0,9	**5111**	2014663194	1,06	99	–	**5111**	2014663194	1,06	99	–
	2		7581	2142946244	0,59	99	**1,32**	5119	1993814010	0,93	99	1,07
	3		8305	2151631197	0,70	99	1,23	5112	1987094266	0,94	99	1,07
	4		8878	2136010986	0,84	99	1,13	5166	1990558115	0,98	99	1,04
tmt_ sym	–	–	1921	593531565	–	–	–	1921	593531565	–	–	–
	1	0,3	1143	642864287	0,49	40	–	1143	642864287	0,49	40	–
	2		**1143**	642673918	0,35	40	**1,22**	**1142**	642259532	0,47	41	1,03
	3		1147	643044925	0,45	40	1,05	1147	643030823	0,55	40	0,93
	4		1147	643044925	0,62	40	0,86	1147	643030823	0,63	40	0,85
parabolic _fem	–	–	525820	1196569325	–	–	–	525820	1196569325	–	–	–
	1	0,3	**514**	225052418	0,45	100	–	**514**	225052418	0,45	100	–
	2		517	225077954	0,27	100	**1,32**	516	225007943	0,33	100	1,19
	3		523	225141270	0,33	100	1,19	525	225373267	0,42	100	1,04
	4		524	225387323	0,41	100	1,06	536	225486619	0,47	100	0,97
G2_ circuit	–	–	93719	1098802048	–	–	–	93719	1098802048	–	–	–
	1	0,1	**1959**	202044068	0,09	98	–	**1980**	202044068	0,09	98	–
	2		1963	201379857	0,06	98	**1,19**	2009	203217749	0,07	98	1,12
	3		1981	200905319	0,07	98	1,12	1987	200840023	0,09	98	1,00
	4		1975	200816276	0,08	98	1,06	1997	201830737	0,10	98	0,95
shipsec1	–	–	5237	431771001	–	–	–	5237	431771001	–	–	–
	1	0,2	5939	500249589	0,38	-13	–	5939	500249589	0,40	-13	–
	2		5939	500587539	0,30	-13	**1,16**	5939	500249589	0,43	-13	0,92
	3		5939	500587539	0,52	-13	0,81	5939	500249589	0,50	-13	0,83
	4		5939	500249589	0,72	-13	0,63	5939	500249589	0,70	-13	0,64
boneS01	–	–	3722	331330356	–	–	–	3722	331330356	–	–	–
	1	0,2	7531	563492289	0,37	-102	–	7531	563492289	0,37	-102	–
	2		7193	580766268	0,28	-93	**1,19**	7911	594355002	0,31	-113	1,12
	3		8140	602601291	0,42	-119	0,92	7818	615188493	0,40	-110	0,95
	4		8155	600358919	0,61	-119	0,70	8242	582210955	0,62	-121	0,70
cfd2	–	–	100407	483241271	–	–	–	100407	483241271	–	–	–
	1	0,1	**2245**	168545079	0,17	98	–	**2245**	168545079	0,17	98	–
	2		2448	169656324	0,14	98	**1,13**	2339	168863910	0,18	98	0,96
	3		2348	169017066	0,22	98	0,84	2351	169288381	0,23	98	0,82
	4		2548	170884722	0,28	97	0,71	2366	169389041	0,30	98	0,68
thermo- mech_TC	–	–	102138	2667823445	–	–	–	102138	2667823445	–	–	–
	1	0,1	271	18816340	0,11	100	–	271	18816340	0,11	100	–
	2		272	18803813	0,06	100	**1,31**	271	18816686	0,09	100	1,11
	3		**270**	18819007	0,07	100	1,24	**264**	18786399	0,10	100	1,05
	4		284	18895211	0,08	100	1,17	276	18843514	0,11	100	1,00
thermo- mech_TK	–	–	102138	2667823445	–	–	–	102138	2667823445	–	–	–
	1	0,1	267	18791841	0,11	100	–	267	18791841	0,11	100	–
	2		**252**	18685742	0,06	100	**1,31**	267	18787853	0,08	100	1,17
	3		281	18874869	0,07	100	1,24	**255**	18710187	0,09	100	1,11
	4		270	18818366	0,08	100	1,17	295	18808710	0,10	100	1,05
2cubes_ _sphere	–	–	100407	483241271	–	–	–	100407	483241271	–	–	–
	1	0,1	4980	363512510	0,15	95	–	4980	363512510	0,15	95	–
	2		4742	351878926	0,12	95	**1,14**	4823	349482381	0,12	95	1,14
	3		4851	349395239	0,15	95	1,00	**4753**	346146890	0,14	95	1,04
	4		**4688**	333150715	0,19	95	0,86	4811	349559415	0,16	95	0,96
thermal1	–	–	80916	175625317	–	–	–	80916	175625317	–	–	–
	1	0,1	240	12997244	0,06	100	–	240	12997244	0,06	100	–
	2		239	12986673	0,04	100	**1,14**	**225**	12886895	0,06	100	1,00
	3		**232**	13113453	0,05	100	1,07	242	13102675	0,07	100	0,94
	4		235	13029619	0,06	100	1,00	242	13058035	0,08	100	0,89
Andrews	–	–	59925	1501971521	–	–	–	59925	1501971521	–	–	–
	1	0,1	16257	625314750	0,11	73	–	**16257**	625314750	0,11	73	–
	2		**15509**	611209359	0,07	74	**1,24**	17277	680155161	0,09	71	1,11
	3		16371	609266053	0,09	73	1,11	16727	655239135	0,10	72	1,05
	4		17288	685333382	0,10	71	1,05	16563	653517154	0,11	72	1,00

Fonte: Do autor (2018)

tempos de execução do algoritmo RCM-GL utilizando a biblioteca OpenMP. Todas essas execuções foram obtidas na máquina mencionada. Os tempos de execução do *software* Matlab foram obtidos em uma máquina com sistema operacional Windows 7. Verificou-se que os tempos das execuções sequenciais do método RCM-GL (GEORGE; LIU, 1981) com o *software* Matlab foram mais rápidos do que os tempos de execução com o *software* Octave e com a biblioteca de rotinas HSL. Em particular, como a implementação na biblioteca de rotinas HSL retorna facilmente o *profile*, esses valores também foram incluídos na Tabela 5.4.

Em média, as execuções sequenciais do método RCM com o *software* Matlab foram 6,8 vezes mais rápidas do que as execuções com a implementação em paralelo do método RCM-GL (GEORGE; LIU, 1981). Em relação ao *software* Octave e à biblioteca de rotinas HSL, as execuções com o *software* Matlab foram, em média, mais rápidas nas execuções do algoritmo RCM. Dessa forma, a presente implementação do método RCM-GL (GEORGE; LIU, 1981), por meio da biblioteca OpenMP, não é melhor que implementações sequenciais no estado da arte.

Os tempos computacionais com os softwares Matlab e Octave foram obtidos, por meio da execução da função *symrcm*, em conjunto com a execução do procedimento utilizado, para calcular a largura de banda das matrizes. Na biblioteca de rotinas HSL, o método RCM está implementado em Fortran. Com isso, as chamadas das rotinas relacionadas ao método RCM, que se encontram nessa biblioteca, foram realizadas por meio de uma implementação na linguagem de programação C++. Nesse caso, o tempo computacional foi obtido por meio da execução de quatro rotinas: MC60AD, MC60BD, MC60CD, MC60DD e MC60FD. Essas funções são encontradas no pacote MC60 (REID; SCOTT, 2012). Para mais informações sobre a biblioteca de rotinas HSL, veja HSL (Oxford, 2017).

5.2 Experimentos com o método ICCG paralelo

Na Tabela 5.5, são mostrados os resultados dos experimentos realizados com as instâncias *tmt_sym* e *apache2*, utilizando o método ICCG sem reordenação dos vértices. Nessa tabela, # representa a quantidade de *threads* utilizadas para execução dos algoritmos. Também nessa tabela, constam os tempos computacionais obtidos com as execuções do precondicionador IC(0) e do método dos gradientes conjugados, executados três vezes para cada combinação de *threads*, inclusive com a versão sequencial. Com a execução do precondicionador IC(0), o melhor resultado foi obtido com a utilização de oito *threads*. Em investigação exploratória

Tabela 5.4 – Resultados de experimentos com 12 instâncias utilizando a heurística (sequencial) RCM-GL, executadas nos *softwares* Octave e Matlab e na biblioteca de rotinas HSL. Além disso, o método RCM-GL foi executado na biblioteca OpenMP.

Instância	β	Profile	OpenMP			Matlab		Octave		HSL		
			β	Profile	t(s)	β	t(s)	β	t(s)	β	Profile	t(s)
G3_circuit	947128	119101864114	7584	**1510776039**	1,43	5097	0,24	**5069**	0,45	**5069**	5632378743	0,86
tmt_sym	1921	593531565	1142	**623279883**	0,69	1146	0,11	1145	0,20	**1139**	623979576	0,52
parabolic_fem	525820	1196569325	**514**	**225052075**	0,53	**514**	0,11	515	0,16	515	225662727	0,25
G2_circuit	93719	1098802048	1954	179299511	0,15	1963	0,01	**1946**	0,03	1974	**178939972**	0,06
shipsec1	5237	431771001	5939	450339567	0,47	5963	0,05	**5836**	0,16	5964	**450171183**	0,14
boneS01	3722	331330356	7621	398438037	0,45	8168	0,04	8247	0,25	**6168**	**365073840**	0,14
cfd2	100407	483241271	2270	**162467287**	0,22	**2179**	0,04	2499	0,09	2372	163322340	0,15
thermomech_TC	102138	2667823445	268	17849234	0,15	263	0,03	277	0,16	**252**	**17838067**	0,09
thermomech_TK	102138	2667823445	253	**17730868**	0,15	263	0,03	277	0,16	**252**	17838067	0,08
2cubes_sphere	100407	483241271	4738	**272854596**	0,19	4990	0,02	4694	0,08	**4662**	281457521	0,13
thermal1	80916	175625317	231	**12088938**	0,13	**213**	0,01	225	0,05	216	12152950	0,05
Andrews	59925	1501971521	15509	**450171183**	0,17	16315	0,02	16115	0,06	**14979**	458034004	0,08

Fonte: Do autor (2018)

com outras instâncias, os melhores *speedups* foram obtidos também com essas quantidades de *threads*. Na execução do método dos gradientes conjugados, o melhor resultado foi obtido com duas *threads*. Portanto, os melhores resultados dessas implementações foram obtidos com as execuções do precondicionador IC(0) com oito *threads* e com o método dos gradientes conjugados com duas *threads*. São essas as quantidades de *threads* que serão utilizadas como referência para se obter *speedup* nas implementações paralelas com reordenação dos vértices do grafo. O *speedup* foi calculado ao se dividir os tempos computacionais da execução paralela do ICGG pela execução sequencial do método ICCG.

Nas Tabelas de 5.6 a 5.9 e na Figura 5.2, são mostrados os resultados dos experimentos com as heurísticas RCM-GL e KP-band em conjunto com o método ICCG, realizados com as instâncias mostradas na Tabela 5.1. Nessas tabelas, *s* e *p* referem-se às execuções sequencial e paralela do método ICCG, respectivamente, sem reordenação dos vértices.

Nos experimentos realizados com as instâncias *audikw_1*, *bmw7st_1*, *shipsec1*, *cfd2* e *thermomech_TK*, as quantidades de iterações nas execuções do método ICCG com reordenação foi muito menor do que a quantidade de iterações no método sem reordenação. Com isso, obtiveram-se *speedups* com as execuções dos algoritmos paralelos nessas instâncias, especialmente, nas duas primeiras instâncias mencionadas, em que o *speedup* obtido foi alto. Todavia, na instância *shipsec1*, o *speedup* foi marginal.

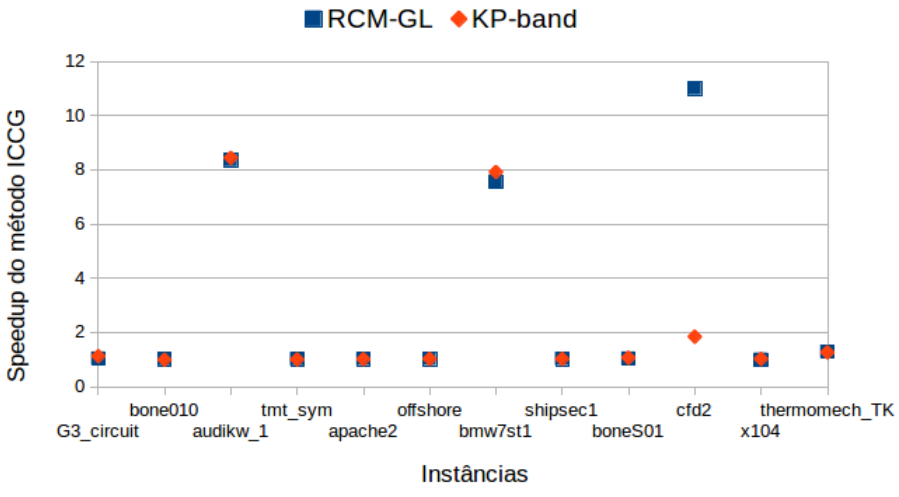
Com as execuções do método ICCG nas instâncias *x104* e *offshore*, o número de iterações apresentado pela versão paralela se manteve em relação à versão sequencial. Nessas instâncias, os *speedups* obtidos foram de 1,02 para ambas as heurísticas, exceto na instância *x104*, em que o *speed-down* foi de 0,99 com a execução do algoritmo RCM-GL. Além disso,

Tabela 5.5 – Resultados da aplicação do método ICCG paralelo em duas instâncias.

Instância	# IC(0)	# MGC	Tempo (s)			Itera-ções	*Speedup*
			IC(0)	MGC	ICCG		
tmt_sym	1	1	111,1	8535	8645	1323	–
	2	2	8012	109,8	8122		1,06
	3	3	7134	110,1	7244		1,19
	4	4	6440	110,2	6550		1,32
	8	8	6254	111,5	6365		1,36
	3	2	7184	109,6	7293		1,19
	4	2	6492	109,6	6601		1,31
	5	2	6810	109,7	6920		1,25
	6	2	6560	109,7	6669		1,30
	7	2	6557	109,7	6666		130
	8	**2**	**6254**	**109,6**	**6364**		**1,36**
	9	2	6551	109,7	6661		1,30
apache2	1	1	8013	41,9	8055	502	–
	2	2	7590	42,0	7632		1,06
	3	3	6840	42,2	6882		1,17
	4	4	6118	42,1	6160		1,31
	8	8	5954	42,3	6040		1,33
	3	2	6829	42,1	6871		1,17
	4	2	6112	42,1	6154		1,31
	5	2	6495	42,1	6537		1,23
	6	2	6216	42,0	6258		1,29
	7	2	5992	42,0	6034		1,33
	8	**2**	**5947**	**42,0**	**5989**		**1,34**
	9	2	6202	42,2	6244		1,29

Fonte: Do autor (2018)

Figura 5.2 – *Speedup* do método ICCG implementado com a biblioteca OpenMP.



Fonte: Do autor (2018)

Tabela 5.6 – Resultados de experimentos com seis instâncias em que são utilizadas oito *threads* na fatoração incompleta de Cholesky com zero *fill-in* e duas *threads* no método dos gradientes conjugados com reordenamento dos vértices realizado pela heurística RCM-GL (GEORGE; LIU, 1981) (continua na Tabela 5.7).

	#	β	*Profile*	Tempo(s)				Itera-ções	Speedup
				RCM	GL	RCM-GL	ICCG		
	s	–	947128	3137747122	–	–	–	37559	958	1,37
	p							27375		
G3_circuit	1	**5088**	1337252610	0,9	0,9	1,9	37073	1205	0,74
	2	7584	1510776039	0,5		1,5	26838	1219	1,02
	3	8460	1526034712	0,7		1,6	26653	1231	**1,03**
	4	8958	1503310693	0,9		1,9	26547	1220	1,03
	8	9531	1481937265	1,2		2,1	26494	1244	1,03
	12	9800	1495377625	1,1		2,1	26544	1239	1,03
	s	–	13016	8846266758	–	–	–	50772	4551	1,09
	p							46777		
bone010	1	**13055**	2984010428	3,9	21,2	25,1	50029	1842	0,93
	2	19157	3682460098	2,7		23,8	46733	1927	1,00
	3	21764	3990708551	4,2		25,4	46157	1952	**1,01**
	4	23603	4178628626	6,1		27,3	48068	4640	0,97
	8	28318	494101300	10,1		31,3	47599	4307	0,98
	12	28254	489436842	10,1		31,3	47650	4201	0,98
	s	–	925946	3671706312	–	–	–	395626	943695	0,99
	p							392745		
audikw_1	1	36108	3289552617	4,9	2,54	7,5	49901	1	7,87
	2	39170	3561659420	2,8		5,4	47271	1	8,31
	3	**35087**	3198850349	4,6		7,1	47112	1	8,34
	4	35087	3199137035	6,5		9,1	46995	1	**8,36**
	8	35066	3199137035	6,5		9,0	48256	1	8,14
	12	35095	3199137035	7,0		9,5	48402	1	8,11
	s	–	1921	593531565	–	–	–	8645	1322	1,36
	p							6364		
tmt_sym	1	1144	623343043	0,5	0,4	0,8	8542	1366	0,74
	2	**1143**	623392602	0,3		0,7	6280	1371	**1,01**
	3	1145	623522105	0,4		0,8	6364	1393	1,00
	4	1153	623421998	0,4		0,8	6319	1429	1,01
	8	1153	623878865	0,5		0,9	6290	1417	1,01
	12	1600	623637667	0,5		0,9	6373	1458	1,00
	s	–	65837	1717106409	–	–	–	8055	502	1,34
	p							5989		
apache2	1	4454	1885426466	0,4	0,7	1,2	8038	508	0,74
	2	**33**68	1664604543	0,3		0,9	5887	366	**1,02**
	3	3956	1734514637	0,4		1,1	5934	457	1,01
	4	4201	1855330711	0,5		1,3	5928	485	1,01
	8	4493	1885879285	0,7		1,8	6000	506	1,00
	12	5361	1858811366	0,7		1,7	5923	462	1,01
	s	–	237738	3588201815	–	–	–	1167	4	2,11
	p							552		
offshore	1	21326	2648041422	0,4	0,3	0,7	1133	4	0,48
	2	20815	2640799695	0,2		0,5	548	4	1,01
	3	23213	2837052742	0,3		0,6	545	4	1,01
	4	**20351**	2650427258	0,4		0,7	542	4	**1,02**
	8	21128	2637985620	0,6		1,0	539	4	1,02
	12	25128	2652687165	0,6		1,1	550	4	1,00

Fonte: Do autor (2018)

Tabela 5.7 – Resultados de experimentos com seis instâncias em que são utilizadas oito *threads* na fatoração incompleta de Cholesky com zero *fill-in* e duas *threads* no método dos gradientes conjugados com reordenamento dos vértices realizado pela heurística RCM-GL (GEORGE; LIU, 1981) (continuação da Tabela 5.6).

	#	β	*Profile*	Tempo(s)				Itera-	Speedup
				RCM	GL	RCM-GL	ICCG	ções	
s / **p**	–	4463	179250468	–	–	–	5145 / 5133	141347	1,00
bmw7st_1	1	3742	287092827	0,4	0,2	0,6	770	1	6,67
	2	3891	278206974	0,2		0,4	653	1	7,52
	3	**3734**	274743165	0,4		0,7	648	1	7,48
	4	3794	79481469	0,6		0,8	657	1	**7,57**
	8	3759	271752501	1,0		1,2	674	1	7,56
	12	3840	270519625	0,9		1,2	668	1	7,53
s / **p**	–	5237	431771001	–	–	–	811 / 703	5237	1,15
shipsec1	1	5939	450339567	0,4	0,2	0,6	802	1	0,88
	2	5939	450025485	0,3		0,5	687	1	**1,02**
	3	5939	450339567	0,5		0,7	690	1	1,02
	4	5939	450025485	0,7		1,0	687	1	1,02
	8	**5930**	450159171	1,1		1,4	688	1	1,02
	12	5975	451351767	1,1		1,3	689	1	1,02
s / **p**	–	3722	331330356	–	–	–	641 / 553	686	1,16
boneS01	1	**7621**	398438037	0,4	0,2	0,7	633	371	0,87
	2	8072	403977156	0,3		0,5	534	379	**1,03**
	3	8163	414690087	0,4		0,6	547	376	1,01
	4	8251	397041713	0,3		0,5	537	376	1,03
	8	7650	410311484	1,0		1,2	540	381	1,02
	12	7502	397833012	0,9		1,2	541	383	1,02
s / **p**	–	4501	156107322	–	–	–	2502 / 2436	686	1,03
cfd2	1	2553	163992266	0,2	0,1	0,3	338	1	7,19
	2	2395	163127506	0,1		0,2	226	1	10,76
	3	2394	162610026	0,2		0,3	1323	1	1,83
	4	**2272**	162212841	0,3		0,4	221	1	**11,00**
	8	2386	163242973	0,4		0,7	228	1	10,62
	12	2272	162921223	0,4		0,6	1355	1	1,79
s / **p**	–	4463	179250468	–	–	–	698 / 646	1	1,08
x104	1	**4827**	158537880	0,6	0,2	0,8	702	1	0,92
	2	5219	172415544	0,4		0,5	657	1	0,98
	3	5023	165421260	0,6		0,8	656	1	0,98
	4	5023	165476712	0,9		1,1	649	1	0,99
	8	5462	162686700	1,0		1,0	650	1	0,98
	12	5653	165473684	1,0		1,0	651	1	0,97
s / **p**	–	102138	2667823445	–	–	–	131 / 78	102138	1,52
thermomech_TK	1	276	17908417	0,1	0,1	0,4	121	2469	0,71
	2	**266**	17814324	0,1		0,2	73	2116	1,18
	3	281	17910052	0,1		0,2	67	1382	**1,28**
	4	269	17831705	0,1		0,2	73	2089	1,18
	8	290	17957051	0,2		0,4	69	1816	1,25
	12	279	17773442	0,2		0,3	68	1748	1,26

Fonte: Do autor (2018)

Tabela 5.8 – Resultados de experimentos com seis instâncias em que são utilizadas oito *threads* na fatoração incompleta de Cholesky com zero *fill-in* e duas *threads* no método dos gradientes conjugados com reordenamento dos vértices realizado pela heurística KP-band (KOOHESTANI; POLI, 2011) (continua na Tabela 5.9).

	#	β	*Profile*	Tempo (s)				Iterações	Speedup
				KP-band	GL	KP-band-GL	ICCG		
	s	–	947128	3137747122	–	–	–	37559	958	1,37
	p							27375		
G3_circuit	1	**5111**	2014663194	1,1	0,9		2,0	27590	1281	0,99
	2	7581	2142946244	0,6			1,5	24384	1298	**1,12**
	3	8305	2151631197	0,7			1,6	24405	1279	1,12
	4	8878	2136010986	0,8			1,7	36812	1294	0,74
	8	9477	2114385143	1,2			2,0	36173	1277	0,76
	12	9717	2132403249	1,1			1,9	36257	1285	0,76
	s	–	13016	8846266758	–	–	–	50772	4551	1,09
	p							46777		
bone010	1	**13016**	3678949907	4,2	2,3		6,4	51095	4060	0,92
	2	21886	2185746948	3,1			5,3	47870	4116	0,98
	3	24244	269173819	5,2			7,5	47255	4161	0,99
	4	26618	862562974	6,7			8,9	48088	4200	0,97
	8	25576	680236494	12,1			14,3	48438	4248	0,97
	12	25451	673236591	12,1			14,5	48541	4354	0,96
	s	–	925946	3671706312	–	–	–	395626	943695	0,99
	p							392745		
audikw_1	1	39155	3784981803	4,7	2,0		6,7	49791	1	7,89
	2	**39155**	3784292439	3,3			5,2	47203	1	8,32
	3	39155	3784981803	5,0			7,0	46512	1	**8,44**
	4	39158	3770258001	6,9			8,8	48141	1	8,16
	8	39158	3770258001	6,6			8,5	48326	1	8,13
	12	39158	3770258001	6,6			8,6	49122	1	7,99
	s	–	1921	593531565	–	–	–	8645	1322	1,36
	p							6364		
tmt_sym	1	**1142**	642636319	0,5	0,8		1,2	8665	1654	0,74
	2	1146	642575940	0,4			1,1	6368	1638	1,00
	3	1149	642859203	0,4			1,2	6357	1646	1,00
	4	1148	643048542	0,6			1,3	6350	1676	1,00
	8	1164	643589903	0,7			1,5	6409	1699	0,99
	12	1757	643344474	0,7			1,5	6349	1666	1,00
	s	–	65837	1717106409	–	–	–	8055	502	1,34
	p							5989		
apache2	1	**3172**	1589269247	0,5	0,7		1,2	8045	529	0,74
	2	4473	1885504516	0,3			0,9	5967	552	1,00
	3	3895	1725258987	0,4			1,2	5981	533	1,00
	4	3640	1695847981	0,5			1,3	5941	520	**1,01**
	8	3379	1665906444	0,8			1,9	5946	512	1,01
	12	3634	1695574157	0,7			1,8	5950	522	1,01
	s	–	237738	3588201815	–	–	–	1167	4	2,11
	p							552		
offshore	1	21064	3505609954	0,4	0,3		0,8	1141	4	0,48
	2	22171	3625720057	0,4			0,7	540	4	**1,02**
	3	**19992**	3516352865	0,4			0,7	550	4	1,00
	4	20000	3527313972	0,5			0,8	545	4	1,01
	8	21234	3565410703	0,7			1,0	541	4	1,02
	12	29380	3575348963	0,6			0,9	542	4	1,02

Fonte: Do autor (2018)

Tabela 5.9 – Resultados de experimentos com seis instâncias em que são utilizadas oito *threads* na fatoração incompleta de Cholesky com zero *fill-in* e duas *threads* no método dos gradientes conjugados com reordenamento dos vértices realizado pela heurística KP-band (KOOHESTANI; POLI, 2011) (continuação da Tabela 5.8).

	#	β	*Profile*	Tempo (s)				Iterações	Speedup
				KP-band	GL	KP-band-GL	ICCG		
s / **p**	–	4463	179250468	–	–	–	5145 / 5133	141347	1,00
bmw7st_1	1	3793	317337317	0,5	0,2	0,5	765	1	6,71
	2	3826	310629117	0,4		0,4	650	1	7,90
	3	3590	339072544	0,6		0,7	655	1	7,50
	4	**3571**	316031758	0,7		0,8	648	1	**7,92**
	8	4260	360197584	1,1		1,3	659	1	7,80
	12	3597	338745617	1,0		1,2	656	1	7,82
s / **p**	–	3722	331330356	–	–	–	641 / 553	686	1,16
boneS01	1	7531	563492289	0,4	0,2	0,6	618	425	0,89
	2	**7193**	580766268	0,3		0,5	523	427	**1,06**
	3	8140	602601291	0,4		0,6	521	426	1,06
	4	8155	600358919	0,6		0,8	524	428	1,05
	8	7912	597249582	1,0		1,2	522	423	1,06
	12	8336	596704010	0,9		1,1	531	425	1,03
s / **p**	–	5237	431771001	–	–	–	811 / 703	5237	1,15
shipsec1	1	5939	500249589	0,4	0,2	0,6	810	572	0,87
	2	5939	500587539	0,3		0,5	696	572	1,01
	3	5939	500587539	0,5		0,7	688	572	**1,02**
	4	5939	500249589	0,7		0,8	703	572	1,02
	8	5885	502123749	1,2		1,4	686	297	1,02
	12	5939	500107221	1,1		1,3	773	3178	0,91
s / **p**	–	4501	156107322	–	–	–	2502 / 2436	686	1,03
cfd2	1	2393	169839814	0,2	0,1	0,3	2469	123440	0,99
	2	**2250**	168321469	0,1		0,2	1320	123440	**1,84**
	3	2397	169287999	0,2		0,4	1322	61721	1,84
	4	2404	169384228	0,3		0,5	2440	123440	1,00
	8	2537	170766543	0,5		0,6	1375	61721	1,78
	12	2553	171181891	0,4		0,6	2492	123440	0,98
s / **p**	–	4463	179250468	–	–	–	698 / 646	1	1,08
x104	1	5219	268562904	0,5	0,2	0,7	687	1	0,94
	2	5219	268562904	0,4		0,7	636	1	**1,02**
	3	**4827**	246155520	0,7		0,8	644	1	1,00
	4	5219	268562904	0,9		1,1	643	1	1,00
	8	5219	268436580	1,0		1,1	647	1	0,99
	12	5219	266524907	1,0		1,3	646	1	0,99
s / **p**	–	102138	2667823445	–	–	–	131 / 78	102138	1,52
thermomech_TK	1	280	18865337	0,1	0,2	0,3	116	1852	0,74
	2	**266**	18775759	0,1		0,2	73	2301	1,17
	3	269	18801695	0,1		0,3	76	2715	1,13
	4	283	18887045	0,1		0,3	68	1952	**1,26**
	8	291	18980249	0,2		0,4	73	1985	1,15
	12	287	18973509	0,2		0,4	69	1970	1,25

Fonte: Do autor (2018)

na maioria das instâncias testadas, a heurística KP-band apresentou resultados ligeiramente melhores do que com o método RCM-GL, exceto na instância *cfd2*, em que o *speedup* obtido com o método RCM-GL foi muito melhor do que o *speedup* obtido com a heurística KP-band.

Em geral, os melhores resultados de experimentos realizados com as implementações paralelas das heurísticas RCM e KP-band foram obtidos com duas *threads*. Isso ocorre porque as paralelizações dos vértices do grafo são realizadas nível a nível da estrutura de nível enraizada no vértice inicial e isso pode limitar o paralelismo. Em testes com instâncias como *G3_circuit*, as reduções de largura de banda, em mais de 98% com as duas heurísticas, foram suficientes para melhorar o tempo de execução na resolução dos sistemas de equações lineares pelo método ICCG.

6 CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho, as heurísticas RCM-GL e KP-band foram implementadas e avaliadas, em relação às reduções de largura de banda e de *profile*. Foram encontradas na bibliografia dez abordagens de paralelização dos métodos Cuthill e McKee (1969) e RCM (GEORGE, 1971) e três abordagens de paralelização da heurística de Sloan (1989). Para a heurística KP-band (KOOHESTANI; POLI, 2011), descrita como uma das heurísticas mais promissoras, para áreas de aplicação específicas (OLIVEIRA; BERNARDES; CHAGAS, 2018), não foi encontrada nenhuma abordagem de paralelização na bibliografia.

Nas seções seguintes, são descritas algumas observações sobre este trabalho. Na seção 6.1, são apresentadas as considerações sobre simulações com as instâncias da base *SuiteSparse Matrix Collection*. Finalmente, na seção 6.2, são descritas as propostas de trabalhos futuros.

6.1 Conclusões

Neste projeto, foram avaliadas implementações, por meio das bibliotecas OpenMP e Pthreads, das heurísticas RCM-GL (GEORGE; LIU, 1981) e KP-band (KOOHESTANI; POLI, 2011), utilizadas para o reordenamento de linhas e colunas de matrizes esparsas. Essas heurísticas foram propostas com o objetivo de reduzir o custo computacional quando aplicadas em conjunto com o método dos gradientes conjugados precondicionado pela fatoração incompleta de Cholesky *shifted* (MANTEUFFEL, 1980) com zero *fill-in*.

Com os resultados experimentais, percebe-se que a qualidade da soluções obtidas com as execuções dos algoritmos sequenciais e paralelos se manteve próxima na maioria da instâncias testadas; contudo o tempo computacional foi melhor com a biblioteca OpenMP do que com a biblioteca Pthreads. Assim, foi verificado que as implementações com a biblioteca OpenMP podem ser mais promissoras do que as implementações com a biblioteca Pthreads para essas heurísticas, mesmo que o desempenho computacional dessas heurísticas não seja melhor do que o desempenho computacional em execuções sequenciais apresentado pelo método RCM, implementado na biblioteca de rotinas matemáticas HSL e nos *softwares* Matlab e Octave. Desse modo, uma otimização nos códigos implementados deverá ser realizada de modo que o desempenho computacional dos algoritmos seja melhorado.

Os melhores resultados das implementações paralelas das heurísticas RCM e KP-band foram obtidos com a utilização de apenas duas *threads*. Desse modo, as paralelizações dessas heurísticas neste trabalho devem ser melhoradas de modo que o *speedup* também melhore

quando executadas com mais de duas *threads*. Isso ocorre porque as paralelizações dos vértices do grafo são realizadas nível a nível na estrutura de nível enraizada no vértice inicial. Apesar disso, neste trabalho, mostra-se que heurísticas paralelas para reduções de largura de banda têm potencial de acelerar o método paralelo dos gradientes conjugados precondicionado.

6.2 Trabalhos Futuros

Em relação à paralelização do método ICCG, embora se tenha obtido um ganho de desempenho computacional, foi observado que o *speedup* e a eficiência computacional ficaram distantes do que seria ideal. Como trabalhos futuros, pretende-se melhorar a implementação paralela do método ICCG. Com isso, outros precondicionadores em paralelo também deverão ser implementados, tais como SSOR, *Multigrid* Algébrico, polinomial, Poisson e variações do ILU como ILUT, para que possam ser utilizados no precondicionamento de matrizes simétricas e positivas definidas. O precondicionador de Poisson possui a vantagem ser paralelizado facilmente em arquiteturas *multicore* (AMENT et al., 2010; HELFENSTEIN; KOKO, 2012). Essas técnicas devem ser utilizadas como precondicionadores dos métodos dos gradientes conjugados e *Generalized Minimal Residual* (GMRES) (SAAD; SCHULTZ, 1986) para avaliar o seu desempenho computacional em conjunto com heurísticas de reordenação. O método GMRES é aplicado em matrizes assimétricas, com isso, também é esperado que as heurísticas de reordenação sejam adaptadas para reduções de largura de banda e de *profile* de matrizes não simétricas.

Mesmo que as implementações das heurísticas RCM-GL e KP-band sejam ingênuas e, por isso, pode ser que os melhores resultados tenham sido obtidos com a biblioteca OpenMP do que com a biblioteca Pthreads, otimizações deverão ser realizadas nessas heurísticas. Desse modo, pretendem-se implementar essas heurísticas por meio da estrutura de dados *bags* proposta por Leiserson e Schardl (2010). Cada *bag* consiste de uma coleção de vértices munida de duas operações: *add*, que insere um vértice na coleção, e *iterate*, que percorre os vértices da coleção. Com a utilização dessa estrutura, esperam-se obter melhorias no desempenho computacional dos algoritmos utilizados para reduzir largura de banda de matrizes. Além disso, com uma implementação eficiente do procedimento busca em largura, pretende-se também implementar o algoritmo de George e Liu (1981) em paralelo.

Em relação aos métodos heurísticos, para reduções de largura de banda e de *profile*, pretendem-se implementar, em paralelo, as heurísticas busca em largura, Sloan (1989), NSloan

(KUMFERT; POTHEN, 1997), Sloan-MGPS (REID; SCOTT, 1999) e MPG (MEDEIROS; PIMENTA; GOLDENBERG, 1993). Também é possível que existam na bibliografia outras heurísticas com características mais propicias à paralelização. Desse modo, uma busca por tais heurísticas será necessária. Por fim, esperam-se implementar as heurísticas paralelas que foram encontradas na revisão sistemática da bibliografia. Com essas implementações, em arquitetura *multicore*, será possível identificar as melhores heurísticas para reduzir largura de banda e *profile* de matrizes.

REFERÊNCIAS

ADAMS, L. M.; JORDAN, H. F. Is SOR color-blind? **SIAM Journal on Scientific and Statistical Computing**, Philadelphia, v. 7, n. 2, p. 490–506, 1986.

ALMEIDA, V. **Uma adaptação do MEF para análise em multicomputadores: aplicações em alguns modelos estruturais.** 1999. 152 p. Dissertação (Mestrado em Engenharia de Estruturas) — Universidade de São Carlos, São Paulo, 1999.

AMENT, M. et al. A parallel preconditioned conjugate gradient solver for the Poisson problem on a multi-GPU platform. In: EUROMICRO CONFERENCE ON PARALLEL, DISTRIBUTED AND NETWORK-BASED PROCESSING, 18., 2010, Pisa. **Proceedings...** Pisa: Parallel, Distributed and Network-Based Processing (PDP), 2010. p. 583–592.

AZAD, A. et al. The reverse Cuthill-McKee algorithm in distributed-memory. In: IEEE INTERNATIONAL PARALLEL & DISTRIBUTED PROCESSING SYMPOSIUM, 2017, Orlando. **Proceedings...** Orlando: IEEE, 2017.

BENZI, M.; TUMA, M. A robust incomplete factorization preconditioner for positive definite matrices. **Numerical Linear Algebra with Applications**, Chichester, v. 10, n. 5/6, p. 385–400, July/Sept 2003.

BERNARDES, J. A. B.; OLIVEIRA, S. L. G. de. A systematic review of heuristics for profile reduction of symmetric matrices. **Procedia Computer Science**, Amsterdam, v. 51, p. 221–230, 2015.

BULUÇ, A.; GILBERT, J. R. The combinatorial BLAS: design, implementation, and applications. **The International Journal of High Performance Computing Applications**, Thousand Oaks, v. 25, n. 4, p. 496–509, May 2011.

CHAGAS, G. O.; OLIVEIRA, S. L. G. de. Metaheuristic-based heuristics for symmetric-matrix bandwidth reduction: A systematic review. **Procedia Computer Science**, Amsterdam, v. 51, p. 211–220, 2015.

CHAN, W. M.; GEORGE, A. A linear time implementation of the reverse Cuthill-Mckee algorithm. **BIT Numerical Mathematics**, Oxford, v. 20, n. 1, p. 8–14, Mar 1980.

ČIEGIS, R. Analysis of parallel preconditioned conjugate gradient algorithms. **Informatica**, New York, v. 16, n. 3, p. 317–332, 2005.

CONCUS, P.; GOLUB, G. H.; MEURANT, G. Block preconditioning for the conjugate gradient method. **SIAM Journal on Scientific and Statistical Computing**, Philadelphia, v. 6, n. 1, p. 220–252, 1985.

COWLING, P. I.; KENDALL, G.; SOUBEIGA, E. A hyperheuristic approach to scheduling a sales summit. In: INTERNATIONAL CONFERENCE ON PRACTICE AND THEORY OF AUTOMATED TIMETABLING, 3., 2001, Berlin. **Proceedings...** Berlin: Springer Berlin Heidelberg, 2001. p. 176–190.

CUTHILL, E.; MCKEE, J. Reducing the bandwidth of sparse symmetric matrices. In: NATIONAL CONFERENCE, 24., 1969, New York. **Proceedings...** New York: ACM, 1969. p. 157–172.

DAVIS, T. A.; HU, Y. SuiteSparse Matrix Collection. **ACM Transactions on Mathematical Software**, New York, v. 38, n. 1, p. 1–25, 2011.

DEB, B.; SRIRAMA, S. N. Scalability of parallel genetic algorithm for two-mode clustering. **International Journal of Computer and Applications**, New York, v. 94, n. 14, p. 23–26, May 2014.

ESPOSITO, A.; TARRICONE, L. Parallel heuristics for bandwidth reduction of sparse matrices with IBM SP2 and Cray T3D. In: WAŚNIEWSKI, J. et al. (Ed.). **Applied Parallel Computing Industrial Computation and Optimization**. Berlin: Springer Berlin Heidelberg, 1996. p. 239–246.

FELIPPA, C. A. Solution of linear equations with skyline-stored symmetric matrix. **Computers & Structures**, Elmsford, v. 5, n. 1, p. 13–29, Apr 1975.

GEORGE, A.; LIU, J. W. **Computer Solution of Large Sparse Positive Definite Systems**. New York: Prentice Hall Professional Technical Reference, 1981. 314 p.

GEORGE, A.; LIU, J. W. H. An implementation of a pseudoperipheral node finder. **ACM Transactions on Mathematical Software**, New York, v. 5, n. 3, p. 284–295, 1979.

GEORGE, J. A. **Computer implementation of the finite element method.** 1971. 228 p. Tese (PhD) — Computer Science Department, Stanford University, USA, 1971.

GIBBS, N. E.; POOLE, J. W. G.; STOCKMEYER, P. K. An algorithm for reducing the bandwidth and profile of a sparse matrix. **SIAM Journal on Numerical Analysis**, Philadelphia, v. 13, n. 2, p. 236–250, Apr 1976.

GIBOU, F.; MIN, C. On the performance of a simple parallel implementation of the ILU-PCG for the Poisson equation on irregular domains. **Journal of Computational Physics**, Orlando, v. 231, n. 14, p. 4531–4536, May 2012.

GKOUNTOUVAS, T. et al. Improving the performance of the symmetric sparse matrix-vector multiplication in multicore. In: INTERNATIONAL SYMPOSIUM ON PARALLEL AND DISTRIBUTED PROCESSING, 27., 2013, Boston. **Proceedings...** Boston: IEEE, 2013. p. 273–283.

GOLUB, G. H.; LOAN, C. F. V. **Matrix computations**. Baltimore: JHU Press, 2012. 728 p.

HELFENSTEIN, R.; KOKO, J. Parallel preconditioned conjugate gradient algorithm on GPU. **Journal of Computational and Applied Mathematics**, Antwerpen, v. 236, n. 15, p. 3584–3590, Sept 2012.

HESTENES, M. R.; STIEFEL, E. Methods of conjugate gradients for solving linear systems. **Journal of Research of the National Bureau of Standards**, Washington, v. 49, n. 36, p. 409–436, Dec 1952.

HSL. **The HSL Mathematical Software Library**. Oxford, 2017. Disponível em: <http://www.hsl.rl.ac.uk>. Acessado: 16 mar. 2018.

HU, Y.; SCOTT, J. **HSL MC73:** A fast multilevel Fiedler and profile reduction code. Oxfordshire: Rutherford Appleton Laboratory, 2003. 19 p.

HUBING, T. H.; ALI, M. W.; BHAT, G. K. EMAP: A 3-D finite element modeling code for analyzing time-varying electromagnetic fields. **Applied Computational Electromagnetics Society Journal**, Monterey, v. 8, n. 1, p. 116–124, Jan 1993.

IWASHITA, T.; NAKANISHI, Y.; SHIMASAKI, M. Comparison criteria for parallel orderings in ILU preconditioning. **SIAM Journal on Scientific Computing**, Philadelphia, v. 26, n. 4, p. 1234–1260, 2005.

IWASHITA, T.; NAKASHIMA, H.; TAKAHASHI, Y. Algebraic block multi-color ordering method for parallel multi-threaded sparse triangular solver in ICCG method. In: INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 26., 2012, Washington. **Proceedings...** Washington: IEEE, 2012. p. 474–483.

IWASHITA, T.; SHIMASAKI, M. Algebraic block red-black ordering method for parallelized ICCG solver with fast convergence and low communication costs. **IEEE Transactions on Magnetics**, New York, v. 39, n. 3, p. 1713–1716, May 2003a.

IWASHITA, T.; SHIMASAKI, M. Block red-black ordering: A new ordering strategy for parallelization of ICCG method. **International Journal of Parallel Programming**, New York, v. 31, n. 1, p. 55–75, Feb 2003b.

KARANTASIS, K. I. et al. Parallelization of reordering algorithms for bandwidth and wavefront reduction. In: SC14: INTERNATIONAL CONFERENCE FOR HIGH PERFORMANCE COMPUTING, NETWORKING, STORAGE AND ANALYSIS, 2014, Piscataway. **Proceedings...** Piscataway: IEEE, 2014. p. 921–932.

KARDANI, O.; LYAMIN, A.; KRABBENHOFT, K. A comparative study of preconditioning techniques for large sparse systems arising in finite element limit analysis. **International Journal of Applied Mathematics**, Sofia, v. 43, n. 4, p. 195–203, Nov 2013.

KERSHAW, D. S. The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations. **Journal of Computational Physics**, Orlando, v. 26, n. 1, p. 43–65, Jan 1978.

KOOHESTANI, B.; POLI, R. A hyper-heuristic approach to evolving algorithms for bandwidth reduction based on genetic programming. In: SGAI INTERNATIONAL CONFERENCE ON INNOVATIVE TECHNIQUES AND APPLICATIONS OF ARTIFICIAL INTELLIGENCE, 31., 2011, London. **Proceedings...** London: Springer, 2011. p. 93–106.

KUMFERT, G.; POTHEN, A. Two improved algorithms for envelope and wavefront reduction. **BIT Numerical Mathematics**, Oxford, v. 37, n. 3, p. 559–590, Sept 1997.

KUMFERT, G. K. **An Object-Oriented Algorithmic Laboratory For Ordering Sparse Matrices**. 189 p. Tese (PhD) — Unversity of California, Livermore, 2000.

LANCZOS, C. Solution of systems of linear equations by minimized iterations. **Journal Of Research Of The National Bureau Of Standards Section. B. Mathematics And Mathematical**, Washington, v. 49, n. 1, p. 33–53, July 1952.

LEISERSON, C. E.; SCHARDL, T. B. A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers). In: ANNUAL ACM SYMPOSIUM ON PARALLELISM IN ALGORITHMS AND ARCHITECTURES, 22., 2010, New York. **Proceedings...** New York: ACM, 2010. p. 303–314.

LIN, Y.; YUAN, J. Profile minimization problem for matrices and graphs. **Acta Mathematicae Applicatae Sinica**, Beijing, v. 10, n. 1, p. 107–112, Jan 1994.

LIU, W.-H.; SHERMAN, A. H. Comparative analysis of the Cuthill-McKee and the Reverse Cuthill-McKee ordering algorithms for sparse matrices. **SIAM Journal on Numerical Analysis**, Philadelphia, v. 13, n. 2, p. 198–213, Apr 1976.

MAFTEIU-SCAI, L. O.; CORNIGEANU, C. A. A parallel heuristic for bandwidth reduction based on matrix geometry. In: INTERNATIONAL SYMPOSIUM ON SYMBOLIC AND NUMERIC ALGORITHMS FOR SCIENTIFIC COMPUTING, 18., 2016, Timisoara. **Proceedings...** Timisoara, Romania: IEEE: Romania, 2016a. p. 424–427.

MAFTEIU-SCAI, L. O.; CORNIGEANU, C. A. Parallel heuristics for systems of equations preconditioning. In: INTERNATIONAL SYMPOSIUM ON SYMBOLIC AND NUMERIC ALGORITHMS FOR SCIENTIFIC COMPUTING, 18., 2016, Timisoara. **Proceedings...** Timisoara: IEEE: Romania, 2016b. p. 319–322.

MANGUOGLU, M. et al. Weighted matrix ordering and parallel banded preconditioners for iterative linear system solvers. **SIAM Journal on Scientific Computing**, Philadelphia, v. 32, n. 3, p. 1201–1216, Jan 2010.

MANTEUFFEL, T. A. An incomplete factorization technique for positive definite linear systems. **European Journal of Operational Research**, Amsterdam, v. 185, n. 3, p. 1319–1335, 1980.

MARZULO, L. A. J. **Explorando linhas de execução paralelas com programação orientada por fluxo de dados.** 2011. 140 p. Tese (Doutorado em Engenharia de Sistemas e Computação) — Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2011.

MEDEIROS, S.; PIMENTA, P.; GOLDENBERG, P. An algorithm for profile and wavefront reduction of sparse matrices with a symmetric structure. **Engineering Computations**, Swansea, v. 10, n. 3, p. 257–266, 1993.

MEIJERINK, J. A.; VAN DER VORST, H. A. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. **Mathematics of Computation**, Providence, v. 31, n. 137, p. 148–162, Jan 1977.

MOGHNIEH, H.; LOWTHER, D. A. Understanding the efficiency of parallel incomplete Cholesky preconditioners on the performance of ICCG solvers for multi-core and GPU systems. In: IEEE CONFERENCE ON ELECTROMAGNETIC FIELD COMPUTATION, 14., 2010, Chicago. **Proceedings...** Chicago: Electromagnetic Field Computation, 2010. p. 1.

NOGUEIRA, E. C. **Estudo da paralelização para solução do sistema linear do método dos elementos finitos generalizados.** 2011. 88 p. Dissertação (Mestrado em Modelagem Matemática e Computacional) — Centro Federal de Educação Tecnológica de Minas Gerais, Belo Horizonte, 2011.

OLIVEIRA, S. L. G. de. **Introdução à geração de malhas triangulares**. São Carlos: SBMAC, 2015. 94 p.

OLIVEIRA, S. L. G. de. Heurísticas para reduções de largura de banda (CM, RCM) e de *profile* (Sloan, NSloan, Sloan-MGPS, MPG, Snay). 2016. Notas de Aula da Disciplina Projeto e Análise de Algoritmos. 2016.

OLIVEIRA, S. L. G. de; BERNARDES, J. A. B.; CHAGAS, G. O. An evaluation of reordering algorithms to reduce the computational cost of the incomplete Cholesky-conjugate gradient method. **Computational & Applied Mathematics**, Petrópolis, p. 1–40, Aug 2017.

OLIVEIRA, S. L. G. de; BERNARDES, J. A. B.; CHAGAS, G. O. An evaluation of low-cost heuristics for matrix bandwidth and profile reductions. **Computational and Applied Mathematics**, Petrópolis, v. 37, n. 2, p. 1412–1471, May 2018.

OLIVEIRA, S. L. G. de; CHAGAS, G. O. **Introdução a heurísticas para redução de largura de banda de matrizes**. São Carlos: SBMAC, 2014. 108 p.

OLIVEIRA, S. L. G. de; CHAGAS, G. O. A systematic review of heuristics for symmetric-matrix bandwidth reduction: methods not based on metaheuristics. In: BRAZILIAN SYMPOSIUM ON OPERATIONS RESEARCH, 2015, Pernambuco. **Proceedings...** Pernambuco: Sobrapo, 2015.

PAPADIMITRIOU, C. H. The NP-Completeness of the bandwidth minimization problem. **Computing**, New York, v. 16, n. 3, p. 263–270, Sept 1976.

PRASAD, I. D.; PATNAIK, L.; MURTHY, I. An optimal parallel algorithm for sparse matrix bandwidth reduction on a linear array. **International Journal of Computer Mathematics**, London, v. 42, n. 1/2, p. 7–20, 1992.

PUTTONEN, J. Simple and effective bandwidth reduction algorithm. **International Journal for Numerical Methods in Engineering**, Chichester, v. 19, n. 8, p. 1139–1152, Aug 1983.

REEVES, C. R. (Ed.). **Modern Heuristic Techniques for Combinatorial Problems**. New York: John Wiley & Sons, Inc., 1993. 320 p.

REID, J. K.; SCOTT, J. A. Ordering symmetric sparse matrices for small profile and wavefront. **International Journal for Numerical Methods in Engineering**, Chichester, v. 45, n. 12, p. 1737–1755, Aug 1999.

REID, J. K.; SCOTT, J. A. MC60**. Science & Technologia,** New York, v. 1, p. 1-12, Oct. 2012. Disponível em: <http://www.hsl.rl.ac.uk/specs/mc60.pdf>. Acessado: 16 mar. 2018.

RODRIGUES, T. N. **Non-Speculative Data-Driven Parallelizations of Irregular Algorithms for Sparse Matrices Reordering.** 2017. 212 p. Dissertação (Mestrado em Informática) — Universidade Federal do Espírito Santo, Vitória, 2017.

RODRIGUES, T. N.; BOERES, M. C. S.; CATABRIGA, L. An optimized leveled parallel RCM for bandwidth reduction of sparse symmetric matrices. In: BRAZILIAN SYMPOSIUM ON OPERATIONS RESEARCH, 2016, Vitória. **Proceedings...** Vitória: SOBRAPO, 2016. p. 12.

RODRIGUES, T. N.; BOERES, M. C. S.; CATABRIGA, L. A non-speculative parallelization of reverse Cuthill-McKee algorithm for sparse matrices reordering. In: FEDERATED CONFERENCE ON COMPUTER SCIENCE AND INFORMATION SYSTEMS, 2017, Prague. **Proceedings...** Prague: ACSIS, 2017a. v. 11, p. 527–536.

RODRIGUES, T. N.; BOERES, M. C. S.; CATABRIGA, L. A parallel Sloan algorithm for the profile minimization problem of sparse matrices. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 49., 2017, Blumenau. **Anais...** Blumenau: SOBRAPO, 2017b. p. 1–12.

RODRIGUES, T. N.; BOERES, M. C. S.; CATABRIGA, L. An implementation of the unordered parallel RCM for bandwidth reduction of large sparse matrices. In: Proceeding SERIES OF THE BRAZILIAN SOCIETY OF APPLIED AND COMPUTATIONAL MATHEMATICS, 2017, Gramado. **Proceedings...** Gramado: CNMAC, 2017c. v. 5, p. 7.

SAAD, Y. **Iterative methods for sparse linear systems**. 2th. ed. Philadelphia: SIAM, 2003. 547 p.

SAAD, Y.; SCHULTZ, M. H. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. **SIAM Journal on Scientific and Statistical Computing**, Philadelphia, v. 7, n. 3, p. 856–869, 1986.

SCRUCCA, L. GA: a package for genetic algorithms in R. **Journal of Statistical Software**, Los Angeles, v. 53, n. 4, p. 1–37, 2013.

SEMBA, K. et al. Parallel performance of multithreaded ICCG solver based on algebraic block multicolor ordering in finite element electromagnetic field analyses. **IEEE Transactions on Magnetics**, New York, v. 49, n. 5, p. 1581–1584, May 2013.

SESHIMA, N.; TANAKA, M.; TSUBOI, H. Electromagnetic field analysis by using parallel processing based on OpenMP. In: BIENNIAL IEEE CONFERENCE ON ELECTROMAGNETIC FIELD COMPUTATION, 12., 2006, Miami. **Proceedings...** Miami: IEEE, 2006. p. 114–114.

SIEK, J.; LEE, L.; LUMSDAINE, A. **The Boost Graph Library:** User Guide and Reference Manual. Boston: Pearson Education, 2002. 352 p.

SLOAN, S. W. A FORTRAN program for profile and wavefront reduction. **International Journal for Numerical Methods in Engineering**, Chichester, v. 28, n. 11, p. 2651–2679, Nov 1989.

TARRICONE, L. A genetic approach for the efficient numerical analysis of microwave circuits. **Applied Computational Electromagnetics Society Journal**, Monterey, v. 15, n. 2, p. 87–93, 2000.

TARRICONE, L.; DIONIGI, M.; SORRENTINO, R. A strategy for the efficient mode matching analysis of complex waveguide networks. In: MICROWAVE CONFERENCE, 25., 1995, Perugia. **Proceedings...** Perugia: IEEE, 1995. v. 1, p. 425–429.

TSUBURAYA, T.; OKAMOTO, Y.; SATO, S. Parallelized ICCG method using block-multicolor orderings in real symmetric linear system derived from voltage-driven FEM in time domain. **COMPEL:** The international journal for computation and mathematics in electrical and electronic engineering, Dublin, v. 34, n. 5, p. 1433–1446, 2015.

VAN DER VORST, H. A.; DEKKER, K. Conjugate gradient type methods and preconditioning. **Journal of Computational and Applied Mathematics**, Antwerpen, v. 24, n. 1/2, p. 73–87, Nov 1988.