Impacto de estratégias combinatórias no precondicionador paralelo híbrido SPIKE

Brenno A. Lugon[1], Lucia Catabriga[1], Maria C. Rangel[1], Leonardo M. de Lima[2]

[1]Departamento de Informática – Universidade Federal do Espírito Santo (UFES)
Goiabeiras — Vitória/ES - Brasil — CEP 29075-910

[2]Instituto Federal do Espírito Santo (IFES)

{blugon,luciac,crangel}@inf.ufes.br, lmuniz@ifes.edu.br

Abstract. *In this paper, we use the parallel hybrid SPIKE algorithm as a preconditioner for a nonstationary iterative method combining distributed and shared memory architectures, MPI and OpenMP. In order to obtain a good preconditioner we solve a set of combinatorial problems such as reorderings and graph partitioning. We present the results evaluating the influence of each strategy on the convergence and CPU time of the iterative solver.*

Resumo. *Neste trabalho, utilizamos o algoritmo paralelo híbrido SPIKE como um precondicionador para um método iterativo não-estacionário combinando as arquiteturas de memória distribuída e compartilhada, MPI e OpenMP. A fim de obter um bom precondicionador, resolvemos um conjunto de problemas combinatórios como reordenamentos e particionamento de grafos. Apresentamos os resultados avaliando a influência de cada estratégia na convergência e tempo de CPU do método iterativo.*

1. Introdução

Para a resolução de sistemas lineares de grande porte, é interessante a utilização de métodos iterativos baseados em projeções de subespaços de Krylov por suas boas propriedades numéricas e computacionais. Além disso, o uso de precondicionadores acelera a convergência e, consequentemente, melhora a eficiência desses métodos. na obtenção da solução. Em aplicações paralelas, o algoritmo híbrido SPIKE [Polizzi and Sameh 2006, Polizzi and Sameh 2007], criado inicialmente para resolver um sistema linear, pode também ser usado como precondicionador para o método iterativo. Ele é dito um método híbrido por ser capaz de tirar vantagem da robustez dos métodos diretos e do baixo custo computacional dos métodos iterativos.

Recentemente o uso de *matchings* para *scalings* e reordenamento de matrizes demonstraram ser poderosas técnicas no processo de solução de sistemas lineares esparsos. Em [Hogg and Scott 2015], o algoritmo de escalonamento `MC64` aplicado a problemas com matrizes simétricas indefinidas, amplamente utilizado, é particularmente eficaz em comparação com outros *scalings* e técnicas testadas.

Neste trabalho iremos examinar os impactos de técnicas de otimização na convergência e tempo de execução do método iterativo GMRES precondicionado com o algoritmo híbrido baseado no SPIKE. As etapas que envolvem obter soluções por métodos diretos foram computadas utilizando o software PARDISO[1], em uma abordagem conhecida como PSPIKE (PARDISO + SPIKE) [Manguoglu et al. 2009].

[1]http://www.pardiso-project.org/

2. SPIKE clássico

O algoritmo paralelo híbrido SPIKE tem como objetivo encontrar a solução de um sistema linear da forma $Ax = f$ usando computação paralela, sendo $A = \{a_{ij}\}_{n \times n}$ uma matriz esparsa diagonal dominante. Podemos dividir o algoritmo SPIKE em três etapas principais: pré-processamento, fatoração e pós-processamento [Polizzi and Sameh 2006].

2.1. Pré-processamento

A primeira etapa do pré-processamento consiste em transformar uma matriz esparsa geral em uma matriz com estrutura de banda, i.e., com os elementos não nulos dispostos próximos da diagonal principal e largura de banda relativamente pequena. Esta transformação é uma condição necessária para que o método SPIKE possa ser aplicado e este objetivo pode ser alcançado através de um reordenamento, detalhado na Seção 4.2. Em seguida, é necessário particionar a matriz com estrutura de banda em uma matriz bloco tridiagonal, a fim de se obter os blocos C_i, A_i e B_i, como mostra a Figura 1. Este problema pode ser interpretado como um problema combinatório de particionamento de grafos que será explicado na Seção 4.3.

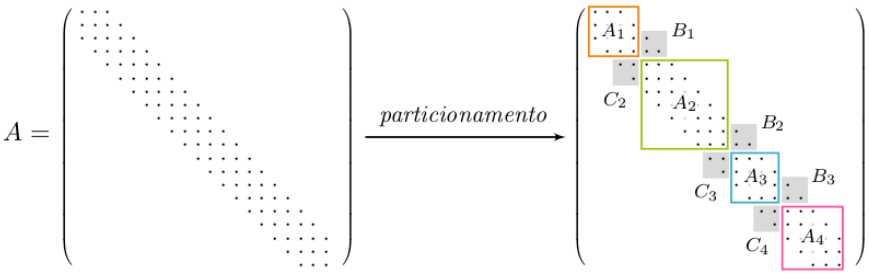


Figura 1. Particionamento bloco tridiagonal em 4 processadores

2.2. Fatoração

Diferente das fatorações usuais muito usadas pelos métodos diretos como $A = \mathbf{LU}$ ou $A = \mathbf{LDL^T}$, o algoritmo SPIKE se baseia na fatoração $A = DS$, onde D é uma matriz bloco diagonal e S é chamada de matriz de *spikes*. A Figura 2 mostra um exemplo dessa fatoração para um particionamento em 4 processadores.

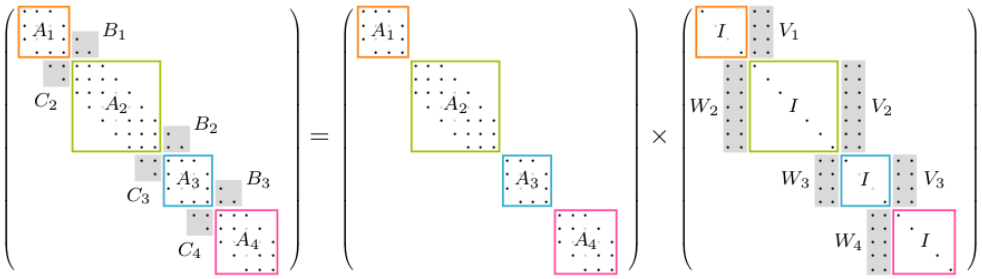


Figura 2. Decomposição $A = DS$

A matriz A é formada pelas matrizes bloco diagonais A_i $(i = 1, ..., p)$ e pelas matrizes bloco de acoplamento B_i $(i = 1, ..., p-1)$ e C_i $(i = 2, ..., p)$. Para garantir que todos os elementos não nulos dispostos fora das matrizes bloco diagonais sejam cobertos pelas matrizes bloco de acoplamento, B_i e C_i possuem dimensão $\mathbf{k} \times \mathbf{k}$, onde $\mathbf{k}$ é a

largura de banda de A. A matriz D é formada pelas matrizes bloco diagonais quadradas A_i e, assumindo a não singularidade de cada bloco A_i, a matriz $S = D^{-1}A$ é formada pelas matrizes *spikes* V_i $(i = 1, ..., p-1)$ e W_i $(i = 2, ..., p)$, densas e de dimensão $n_i \times \mathbf{k}$, sendo n_i o número de linhas dos blocos A_i.

As matrizes *spikes* V_i e W_i são dadas por

$$V_i = A_i^{-1} \begin{pmatrix} 0 \\ B_i \end{pmatrix} \text{ e } W_i = A_i^{-1} \begin{pmatrix} C_i \\ 0 \end{pmatrix}, \tag{1}$$

e, em cada processador i, podem ser calculadas resolvendo os sistemas

$$A_i V_i = \begin{pmatrix} 0 \\ B_i \end{pmatrix} \text{ e } A_i W_i = \begin{pmatrix} C_i \\ 0 \end{pmatrix}, \tag{2}$$

onde o primeiro processador calcula apenas V_1 e o último processador p calcula apenas W_p. Os sistemas da Equação (2) possuem múltiplos vetores de termos independentes e, consequentemente, terão múltiplos vetores como solução. Estes sistemas serão resolvidos via método direto de fatoração $\mathbf{LU}$, com $A_i = \mathbf{L}_i\mathbf{U}_i$, e podem ser representados por

$$\mathbf{L}_i\mathbf{U}_i[V_i, W_i] = \left[\begin{pmatrix} 0 \\ B_i \end{pmatrix}, \begin{pmatrix} C_i \\ 0 \end{pmatrix} \right]. \tag{3}$$

2.3. Pós-processamento

O sistema $Ax = f$ com a fatoração $A = DS$ deve ser resolvido em duas etapas:

1. resolver $Dg = f$
2. resolver $Sx = g$

O sistema $Dg = f$ é totalmente desacoplado, ou seja, pode ser resolvido em paralelo sem gastos com comunicação e sincronização entre os processadores. Isso porque a matriz D é formada apenas pelas matrizes bloco diagonais A_i, independentes entre si. Portanto, podemos resolver esta etapa usando a fatoração $\mathbf{LU}$, i.e., em cada processador i, resolver o sistema $\mathbf{L}_i\mathbf{U}_i g = f$.

O segundo sistema, $Sx = g$, é fracamente acoplado, ou seja, pode ser resolvido com pouca comunicação entre processadores. Podemos observar que resolver este sistema é equivalente a resolver um sistema $\hat{S}\hat{x} = \hat{g}$ de menor complexidade, chamado de sistema reduzido. Para resolvê-lo, considere o particionamento de V_i, W_i e x_i, g_i como

$$V_i = \begin{pmatrix} V_i^{(t)} \\ V_i^{(m)} \\ V_i^{(b)} \end{pmatrix}, W_i = \begin{pmatrix} W_i^{(t)} \\ W_i^{(m)} \\ W_i^{(b)} \end{pmatrix} \quad \text{e} \quad x_i = \begin{pmatrix} x_i^{(t)} \\ x_i^{(m)} \\ x_i^{(b)} \end{pmatrix}, g_i = \begin{pmatrix} g_i^{(t)} \\ g_i^{(m)} \\ g_i^{(b)} \end{pmatrix}, \tag{4}$$

onde $V_i^{(t)}$, $V_i^{(b)}$, $W_i^{(t)}$ e $W_i^{(b)}$ são as extremidades das matrizes *spikes* V_i e W_i, de tamanho $n_i \times \mathbf{k}$. Da mesma forma, $x_i^{(t)}$, $x_i^{(b)}$ e $g_i^{(t)}$, $g_i^{(b)}$ representam as extremidades dos vetores x_i e g_i de tamanho $\mathbf{k}$. Os sobrescritos (t), (m), e (b) são abreviações de *top* (cima), *middle* (meio) e *bottom* (baixo), respectivamente. A partir destas definições, o sistema reduzido $\hat{S}\hat{x} = \hat{g}$ pode ser representado por $p-1$ blocos diagonais, onde o i-ésimo bloco, seus blocos correspondentes fora da diagonal, o vetor de solução e o vetor de termos independentes são dados, respectivamente, por

$$\begin{pmatrix} I & V_i^{(b)} \\ W_{i+1}^{(t)} & I \end{pmatrix}, \begin{pmatrix} W_i^{(b)} & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & V_{i+1}^{(t)} \end{pmatrix}, \begin{pmatrix} \hat{x}_i^{(b)} \\ \hat{x}_{i+1}^{(t)} \end{pmatrix} \text{ e } \begin{pmatrix} \hat{g}_i^{(b)} \\ \hat{g}_{i+1}^{(t)} \end{pmatrix} \tag{5}$$

Uma vez obtida a solução $\hat{x}$ do sistema reduzido, representado em (5), a solução x do sistema global pode ser obtida com perfeito paralelismo fazendo

$$\begin{cases} x_1 = g_1 - V_1\hat{x}_2^{(t)} \\ x_i = g_i - W_i\hat{x}_{i-1}^{(b)} - V_i\hat{x}_{i+1}^{(t)} \qquad i = 2, ..., p-1 \\ x_p = g_p - W_p\hat{x}_{p-1}^{(b)} \end{cases} . \tag{6}$$

Entretanto, utilizando a abordagem descrita, a etapa de montagem do sistema reduzido $\hat{S}\hat{x} = \hat{g}$ pode se tornar muito custosa computacionalmente. Isso porque as matrizes V_i e W_i são densas com quantidade de elementos igual a $n_i \times \mathbf{k}$ para cada i, o que demanda uma grande quantidade de memória para armazenamento. Com o objetivo de melhorar a eficiência do algoritmo, reduzindo os cálculos computacionais envolvidos na decomposição **LU** e o uso de memória – sem comprometer a exatidão, a menos de erros de arredondamento –, podemos calcular apenas as extremidades superiores e inferiores de tamanho $\mathbf{k} \times \mathbf{k}$ das matrizes V_i, W_i e do vetor g_i. Assim, a solução final do sistema $Ax = f$ pode ser recuperada fazendo

$$\begin{cases} \mathbf{L}_1\mathbf{U}_1x_1 = f_1 - B_1\hat{x}_2^{(t)} \\ \mathbf{L}_i\mathbf{U}_ix_i = f_i - C_i\hat{x}_{i-1}^{(b)} - B_i\hat{x}_{i+1}^{(t)} \qquad i = 2, ..., p-1 \\ \mathbf{L}_p\mathbf{U}_px_p = f_p - C_p\hat{x}_{p-1}^{(b)} \end{cases} . \tag{7}$$

3. SPIKE como precondicionador

O algoritmo SPIKE atuando como precondicionador, também chamado de SPIKE-TU ou SPIKE Truncado, apresenta mudanças, em relação ao SPIKE clássico, essenciais para aprimorar o paralelismo e diminuir o número de operações de ponto flutuante.

A primeira mudança do algoritmo SPIKE, atuando como precondicionador, está na obtenção das matrizes *spikes* V_i e W_i. Considerando que as matrizes bloco diagonais A_i sejam diagonais dominantes, podemos usar somente os blocos inferiores, de dimensão $\mathbf{k} \times \mathbf{k}$, das matrizes **L** e **U** da fatoração **LU** para obter V_i e, analogamente, os blocos superiores, de dimensão $\mathbf{k} \times \mathbf{k}$, de **U** e **L** da fatoração **UL** para obter W_i. Apesar de essa estratégia necessitar de mais uma fatoração, a **UL**, [Polizzi and Sameh 2006] afirmam, baseados em seus experimentos, que essa estratégia é mais eficiente do que usar apenas a fatoração **LU**, pois assim seria necessário calcular a matriz de *spikes* W_i total em cada bloco diagonal.

O próxima mudança no SPIKE-TU é com relação ao sistema reduzido. Como o algoritmo SPIKE-TU será usado como um precondicionador, não há necessidade de se obter solução exata desse sistema, portanto, uma solução aproximada e mais simples de construir é suficiente. Neste caso, resolver o sistema reduzido truncado diminui a quantidade de comunicação e tem pouca influência na convergência do método iterativo. Sendo assim, o sistema reduzido truncado $\tilde{S}_i$ do processador i é formado somente pelas matrizes *spikes* $V_i^{(b)}$ e $W_{i+1}^{(t)}$ e pode ser representado matricialmente como

$$\tilde{S}_i = \begin{pmatrix} I & V_i^{(b)} \\ W_{i+1}^{(t)} & I \end{pmatrix} . \tag{8}$$

Como podemos observar, os blocos $W_i^{(b)}$ e $V_{i+1}^{(t)}$ mostrados em (5) são desconsiderados. Portanto, sendo p a quantidade total de processadores, o sistema

reduzido truncado $\tilde{S} = diag\{\tilde{S}_1, \tilde{S}_2, ..., \tilde{S}_{p-1}\}$ será resolvido por partes em cada processador. Como o este sistema no processador i é formado pelas matrizes *spikes* $V_i^{(b)}$ e $W_{i+1}^{(t)}$, é necessário que todo processador $i+1$ envie a matriz $W_{i+1}^{(t)}$ para o processador i que deve montar e resolver sua respectiva parte do sistema.

Para criar a matriz precondicionadora M, é conveniente aplicar um conjunto de reordenamentos na matriz A de forma que os blocos diagonais A_i e as matrizes bloco de acoplamento B_i e C_i contenham os elementos mais significativos da matriz. Sendo assim, a matriz M é criada considerando os seguintes reordenamentos

$$M = KQPD_rAD_cP^TK^T, \tag{9}$$

sendo D_r e D_c fatores de *scaling*, P uma permutação simétrica com o objetivo de reduzir a largura de banda da matriz, Q uma permutação não simétrica para mover os maiores elementos para a diagonal e K uma permutação simétrica que move os elementos mais significativos para os blocos de acoplamento.

Esses reordenamentos (ou permutações) são modelados por problemas combinatórios, resumidos na próxima seção.

4. Estratégias combinatórias

Cada transformação aplicada na matriz original pode ser abordada como problemas combinatórios em grafos, a saber: *matching* perfeito em grafos, particionamento de grafos e rerrotulação de vértices de um grafo. Dentre os objetivos que desejamos alcançar estão:

1. garantir a não-singularidade de cada bloco diagonal.
2. obter uma matriz com estrutura de banda.
3. obter as matrizes bloco diagonais e de blocos de acoplamento.
4. mover elementos mais significativos para os blocos de acoplamento.

4.1. *Matching* e *Scaling*

Na etapa de fatoração do algoritmo SPIKE, mostrado na Equação (2), as matrizes bloco diagonais A_i precisam ser resolvidas via método direto usando a fatoração **LU**. Para isso, é necessário que esses blocos sejam não-singulares, ou seja, admitam uma inversa. Esse objetivo pode ser interpretado como um *matching* perfeito em grafos capaz de mover os elementos mais significativos para a diagonal principal. Portanto, seja Q uma permutação de linha não simétrica do *matching*, o novo sistema linear resultante é da forma

$$QAx = Qf\ . \tag{10}$$

Os algoritmos mais eficientes para encontrar *matchings* maximais em um grafo bipartido, descritos em [Duff and Koster 2000], estão presentes na biblioteca `HSL`, especificamente no algoritmo `MC64`[2]. O algoritmo usado neste trabalho é chamado de *matching* bipartido valorado (*weighted bipartite matching*) e seu objetivo é encontrar um conjunto de entradas, que não estejam na mesma linha e coluna, de forma que o produto ou a soma dessas entradas sejam maximizadas.

A sub-rotina de cálculo do *matching* presente na biblioteca `HSL MC64`, adicionalmente, é capaz de calcular fatores que tornam a matriz diagonal dominante. A técnica, chamada de *scaling*, consiste em multiplicar a matriz por fatores de linha

[2]http://www.hsl.rl.ac.uk/catalogue/mc64.html

e coluna de tal forma que as entradas não nulas da diagonal principal assumam valor absoluto igual a 1.0 e as demais entradas, valor absoluto menor ou igual a 1.0. Segundo [Sathe et al. 2012], o *scaling* tem grande influência na escalabilidade e robustez do SPIKE quando usado como precondicionador. Para aplicar o *scaling*, considere os logaritmos naturais dos fatores do *scaling* $D_r = exp\{u_1, u_2, u_3, ..., u_n\}$ para as linhas, e $D_c = exp\{v_1, v_2, v_3, ..., v_n\}$ para as colunas. Assim, o novo sistema após o *scaling* será

$$D_r A D_c^T D_c x = D_r f \ . \tag{11}$$

onde cada elemento não nulo da matriz A sofre a influência $a_{ij} = a_{ij} * exp(u_i + v_j)$.

4.2. Reordenamento

Para que o algoritmo SPIKE possa ser aplicado para uma matriz geral, devemos primeiramente transformá-la em uma matriz com estrutura de banda através de um reordenamento que minimize sua largura de banda. Isso porque o SPIKE necessita que todos os elementos não nulos estejam cobertos pelas matrizes bloco diagonais A_i e pelas matrizes bloco de acoplamento, B_i e C_i.

Neste trabalho, os algoritmos utilizados foram o Espectral, proposto por [Barnard et al. 1993] e o Espectral Valorado, proposto por [Manguoglu et al. 2010]. Os algoritmos da biblioteca `HSL MC73`[3] usam o conceito de conectividade algébrica da matriz, bem como o vetor de *Fiedler* [Fiedler 1973]. O objetivo do Espectral Valorado, além de minimizar a largura de banda da matriz, é tentar deslocar as entradas mais significativas para próximo da diagonal principal garantindo maior eficácia do precondicionador SPIKE. Ao fim da execução dessa estratégia, temos uma matriz de permutação simétrica P que minimiza a largura de banda da matriz original e deve ser aplicada em todo o sistema linear de acordo com

$$PAP^T Px = Pf \ . \tag{12}$$

4.3. Particionamento

Os principais objetivos do particionamento são reduzir a comunicação entre os processadores e garantir que cada partição tenha uma quantidade balanceada de trabalho a ser realizado, i.e., garantir que cada partição tenha aproximadamente a mesma quantidade de elementos não nulos. Como o tempo de execução de um programa paralelo é medido em função do processo mais lento, equilibrar a quantidade de elementos não nulos é assegurar que cada nó distribuído realize, aproximadamente, a mesma quantidade de operações de ponto flutuante.

Este particionamento é modelado como um problema específico, chamado de *Chains-on-chains partitioning* (CCP) [Pinar and Aykanat 2004], que pode ser resolvido em tempo polinomial. Neste trabalho, o algoritmo *MinMax* [Manne and Sørevik 1995] é empregado para obter a solução exata.

4.4. Problema Quadrático da Mochila

Nesta estratégia, o objetivo é aplicar uma permutação simétrica K para concentrar elementos mais significativos, em módulo, dentro dos blocos de acoplamento B_i e C_i de tamanho $\mathbf{k} \times \mathbf{k}$. Para este fim, utilizamos a heurística *DeMin*, proposta por [Sathe et al. 2012], que encontra uma permutação tal que o novo sistema deve ser

$$KAK^T Kx = Kf \ . \tag{13}$$

[3]http://www.hsl.rl.ac.uk/catalogue/hsl_mc73.html

5. Testes computacionais

Os testes computacionais apresentados a seguir, utilizam uma abordagem híbrida entre MPI e OpenMP. Neste paradigma de programação, multicomputadores exploram o paralelismo usando memória distribuída enquanto que multiprocessadores garantem as vantagens do uso de memória compartilhada. A Figura 3 exibe um exemplo do esquema arquitetural dessa abordagem híbrida. Diferentes nós (Memória + 4 *cores*) se comunicam usando MPI via rede, e os *cores* em cada nó se comunicam usando OpenMP.

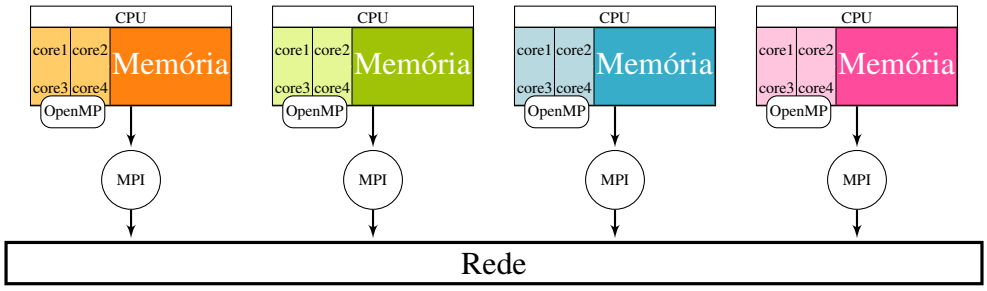


Figura 3. Abordagem híbrida MPI e OpenMP

Os experimentos apresentados nesta seção foram realizados em dois *clusters*: o *cluster* Enterprise 3[4] da Universidade Federal do Espírito Santo e o *cluster* Altix-xe[5] do Laboratório Nacional de Computação Científica (LNCC) localizado no Rio de Janeiro. O primeiro é um *cluster* mais antigo e, portanto, possui configurações modestas comparadas com *hardwares* atuais. Com isso, desejamos avaliar o comportamento dos nossos experimentos neste *cluster*, assim como em um *cluster* mais potente, o Altix-xe.

O *cluster* Enterprise 3 conta com 24 nós Quad Core Intel 2 Q6600 (96 *cores*), com frequência de *clock* de 2.4GHz, 4MB de memória cache L2 e 4GB de RAM, interconectados com um *switch* Gigabit Ethernet 48-Port 4200G 3COM. O *cluster* Altix-xe opera com 30 unidades de processamento, cada qual com 24 GB de RAM e 2 processadores Intel Xeon E5520 2.27GHz Quad Core, totalizando 8 *cores* por unidade. Neste *cluster*, o máximo de processadores permitido por usuário é 96.

Todas as etapas do precondicionador SPIKE-TU que envolvem cálculos usando métodos diretos foram resolvidos utilizando o software PARDISO. Para os experimentos realizados neste trabalho, aplicamos o método iterativo não-estacionário GMRES. Os códigos em C foram compilados com GNU no *cluster* Enterprise 3 e com compiladores Intel no *cluster* Altix-xe. A Tabela 1 apresenta as principais características de um conjunto de matrizes obtidas do repositório de matrizes esparsas da Universidade da Flórida [Davis and Hu 2011]. As informações na tabela são o nome da matriz, dimensão n e quantidade de elementos não nulos nnz, seu padrão de simetria e área de aplicação.

5.1. Influência do precondicionador SPIKE

Neste primeiro teste, realizado no *cluster* Altix-xe, iremos avaliar o influência do precondicionador SPIKE-TU no conjunto de matrizes da Tabela 1, com relação ao tempo computacional e número de iterações do GMRES de uma execução sem precondicionador e com o precondicionador. Foram usados quatro nós com memória distribuída e quatro

[4]enterprise3.lcad.inf.ufes.br/ganglia
[5]http://www.lncc.br/altix-xe

Tabela 1. Características das matrizes testadas

	matriz	dimensão(n)	não nulos (nnz)	simétrica?	área de aplicação
1	finan512	74.752	596.992	sim	Otimização Financeira
2	fem_3D_thermal1	17.880	430.740	não	Elementos Finitos 3D
3	fp	7.548	834.222	não	Eletromagnetismo
4	rail_79841	79.841	553.921	sim	Transferência de Calor
5	mario001	38.434	204.912	não	Redução de Modelo
6	dw8192	8.192	41.746	não	Ondas Dielétricas
7	gemat12	4.929	33.111	não	Rede de Energia

threads. O símbolo † significa que o GMRES não convergiu após 100.000 iterações. A tolerância usada para o GMRES foi 1×10^{-8} para todas as matrizes, exceto para a matriz fp, que foi 1×10^{-14}. O tamanho k dos blocos de acoplamento foi fixado em 50. A influência das estratégias combinatórias no precondicionador SPIKE são apresentadas na Tabela 2. As estratégias combinatórias são representadas por símbolos na ordem (*scaling*, *matching*, reordenamento, problema quadrático da mochila), onde os símbolos • e ○ representam estratégias combinatórias utilizadas e não utilizadas, respectivamente. No caso do reordenamento, o símbolo • indica Espectral e •, Espectral Valorado.

Tabela 2. Influência do precondicionador SPIKE

			sem precondicionador		com precondicionador	
	matriz	estratégias	iterações	tempo	iterações	tempo
1	finan512	○ ○ • ○	45	0,11	7	0,99
2	fem_3D_thermal1	• ○ • ○	59	0,04	20	0,97
3	fp	○ ○ • ○	†	†	48	0,80
4	rail_79841	• • • ○	42.413	171,90	163	2,71
5	mario001	○ ○ • ○	25.619	44,43	97	1,34
6	dw8192	• • • ○	†	†	27	0,11
7	gemat12	○ • • •	†	†	14	0,04

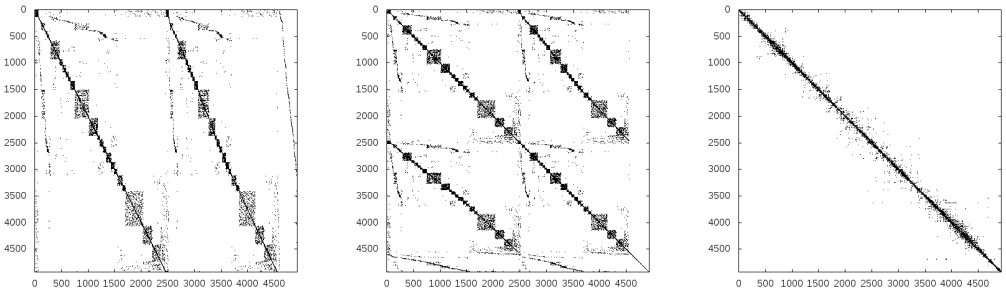
Analisando os resultados da Tabela 2, vemos que o uso do precondicionador é, na maioria das vezes, muito vantajoso. Matrizes com convergência lenta, como a rail_79841 e a mario001, apresentam grande redução no número de iterações e, consequentemente, do tempo de execução. Além disso, parte das matrizes que não convergem para a solução sem o uso do precondicionador, passaram a convergir após seu uso, o que mostra a importância desta técnica. Entretanto, matrizes que convergem rapidamente não apresentaram grandes vantagens computacionais, pois o precondicionador SPIKE, apesar de reduzir o número de iterações, torna cada iteração mais custosa. Por esse motivo, o tempo de processamento para matrizes como finan512 e fem_3D_thermal1 não reduziu com relação ao método sem precondicionador. É importante dizer que nem sempre a aplicação de todas as estratégias combinatórias garantirão um melhor precondicionador. Na prática, é preciso conhecer as características do problema e testar a melhor combinação de estratégias.

5.2. Influência das estratégias combinatórias

Nessa seção, iremos analisar o comportamento do GMRES com o precondicionador SPIKE quando aplicado às estratégias combinatórias separadamente. Todos os testes apresentados a seguir foram executados no *cluster* Altix-xe.

5.2.1. *Matching* e *Scaling*

O *matching* é usado para mover os elementos mais significativos, em módulo, para diagonal principal. Esta técnica é fundamental para matrizes que não possuem elementos na diagonal principal, pois além da dificuldade de realizar a fatoração **LU**, outras estratégias também não apresentam um bom comportamento. A matriz `gemat12` não possui a diagonal principal bem definida e, por isso, ela não converge (ver Tabela 2). Além disso, o reordenamento Espectral encontra dificuldades em reduzir sua largura de banda. A Figura 4 mostra a matriz `gemat12` em três estágios: (a) configuração de esparsidade original, (b) após a aplicação do *matching* e (c) após o reordenamento Espectral. Podemos notar que a utilização do *matching* antes do reordenamento possibilitou a convergência do método GMRES com precondicionador SPIKE (ver Tabela 2).



(a) (b) (c)

Figura 4. Influência do *matching* - Configuração da esparsidade

A Tabela 3 mostra a influência na construção do precondicionador SPIKE quando aplicamos o *scaling* na matriz `rail_79841`, com $\mathbf{p}$ representando o número de processadores distribuídos. Podemos perceber a expressiva redução do número de iterações e do tempo de processamento quando usamos o *scaling*. Também vale destacar que o precondicionador SPIKE, em algumas situações, pode não apresentar um bom *speedup*. Isso ocorre porque a medida que a quantidade de processadores aumenta, o número de iterações do método GMRES tende a crescer.

Tabela 3. Influência do *scaling* na matriz `rail_79841`

sem *scaling*	$\mathbf{p = 2}$	$\mathbf{p = 4}$	$\mathbf{p = 8}$	$\mathbf{p = 16}$
iterações	150	549	1.644	4.640
tempo (seg)	5,60	9,34	15,37	16,16
com *scaling*	$\mathbf{p = 2}$	$\mathbf{p = 4}$	$\mathbf{p = 8}$	$\mathbf{p = 16}$
iterações	98	329	791	2.296
tempo (seg)	3,92	5,77	7,51	8,08

5.2.2. Espectral Valorado

O reordenamento Espectral Valorado tem por objetivo mover os elementos mais significativos para próximo da diagonal principal, podendo exercer bastante influência na convergência do método iterativo. A Tabela 4 detalha o impacto desse reordenamento aplicado à matriz `dw8192`. Note que, o Espectral Valorado reduziu de forma significativa

o número de iterações e o tempo de processamento com relação ao Espectral. Além disso, o Espectral Valorado diminuiu o tempo de processamento, em quase todas as situações, quando aumentamos o número de processadores.

Tabela 4. Influência do Espectral Valorado na matriz `dw8192`

Espectral	$\mathbf{p = 2}$	$\mathbf{p = 4}$	$\mathbf{p = 8}$	$\mathbf{p = 16}$
iterações	196	614	1.165	1.992
tempo (seg)	0,83	2,28	6,04	18,81
Espectral Valorado	$\mathbf{p = 2}$	$\mathbf{p = 4}$	$\mathbf{p = 8}$	$\mathbf{p = 16}$
iterações	14	27	42	88
tempo (seg)	0,15	0,11	0,09	0,10

5.2.3. Problema quadrático da mochila

A ação de mover os elementos mais significativos para dentro dos blocos de acoplamento B_i e C_i pode ser muito benéfica. A Tabela 5 mostra a matriz `gemat12` onde podemos ver a redução do número de iterações e tempo de CPU após a aplicação dessa estratégia. Medimos a qualidade dessa estratégia pela quantidade de elementos movidos para os blocos. Considerando $\mathbf{p = 2}$ foram movidos 138 elementos e para $\mathbf{p = 4}$, 298 elementos.

Tabela 5. Influência do problema quadrático da mochila da matriz `gemat12`

sem problema quadrático da mochila	$\mathbf{p = 2}$	$\mathbf{p = 4}$
iterações	46	83
tempo (seg)	0,10	0,09
com problema quadrático da mochila	$\mathbf{p = 2}$	$\mathbf{p = 4}$
iterações	8	13
tempo (seg)	0,05	0,04

5.3. Desempenho das estratégias combinatórias

Foi considerada uma matriz de grande porte (n = 1.043.474 e nnz = 7.294.192), denotada por `fem_2d_pud`, oriunda de um problema bidimensional de convecção-difusão discretizado pelo método dos elementos finitos. Este problema apresenta uma convergência lenta e, portanto, o precondicionador SPIKE utilizado é bastante eficaz.

Para garantir uma matriz com estrutura de banda, no primeiro conjunto de testes foi aplicado o reordenamento Espectral padrão. Em seguida, escolhemos a melhor configuração de estratégias combinatórias para este problema, isto é, Espectral Valorado com o *scaling*. Vale ressaltar que os tempos da aplicação das estratégias Espectral padrão e Espectral Valorado com *scaling* estão na mesma ordem de grandeza.

Considerando que todo o processo de solução fosse sequencial, o pré-processamento representaria menos de 10% do tempo para resolver o sistema linear. Contudo, nas execuções paralelas, este pré-processamento pode representar mais que o dobro do tempo para resolver este mesmo sistema. As Tabelas 6 e 7 mostram o número de iterações e o tempo de execução dessa aplicação para 2, 4, 8 e 16 processadores MPI e 1, 2 e 4 *threads* OpenMP, nos *clusters* Enterprise 3 e Altix-xe, respectivamente. Como podemos perceber nas tabelas, a aplicação das estratégias combinatórias reduziu o tempo de processamento e quantidade de iterações em ambos os *clusters*.

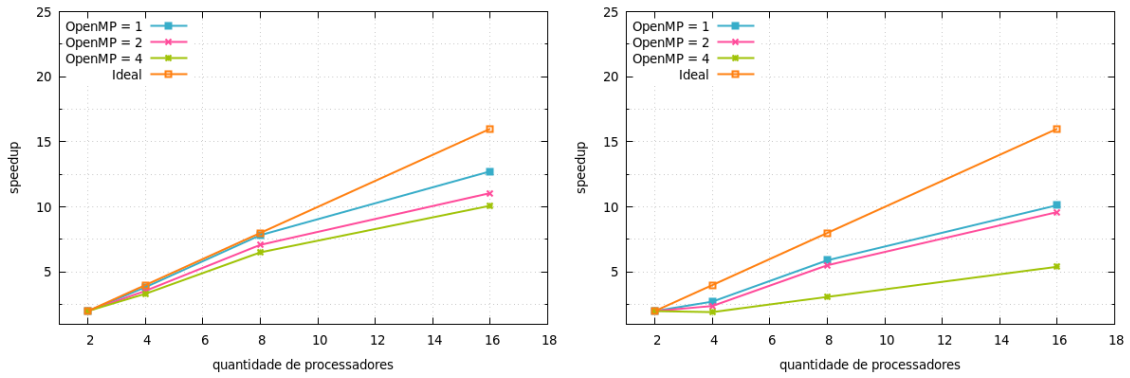
Tabela 6. Tempos (seg.) para diferentes estratégias combinatórias - Enterprise 3

		Espectral				Espectral Valorado + *scaling*			
			OpenMP				OpenMP		
		iterações	1	2	4	iterações	1	2	4
MPI	2	26	79,49	54,53	52,47	20	71,90	52,32	44,24
	4	42	55,12	39,95	36,03	33	37,50	29,44	26,55
	8	72	29,34	25,60	25,60	39	18,37	14,76	13,60
	16	82	16,96	13,37	13,70	55	11,30	9,46	8,77

Tabela 7. Tempos (seg.) para diferentes estratégias combinatórias - Altix-xe

		Espectral				Espectral Valorado + *scaling*			
			OpenMP				OpenMP		
		iterações	1	2	4	iterações	1	2	4
MPI	2	26	45,59	34,00	26,09	20	38,88	25,73	19,85
	4	42	31,22	22,80	26,61	33	28,46	21,57	20,67
	8	68	17,94	13,37	13,20	39	13,19	9,32	12,84
	16	80	9,11	6,80	6,61	56	7,68	5,36	7,35

A Figura 5 apresenta os *speedups* correspondentes à aplicação das estratégias Espectral Valorado e *scaling* nos *clusters* Enterprise 3 e Altix-xe. Apesar dos tempos obtidos pelo cluster Altix-xe serem menores, o *cluster* Enterprise 3 apresentou melhor escalabilidade, o que evidencia proporcionalmente o tempo de comunicação entre os nós MPI. Este fato ocorre, uma vez que a configuração de cada nó do *cluster* Altix-xe é superior quando comparado a cada nó do Enterprise 3.



(a) Enterprise 3 (b) Altix-xe

Figura 5. *Speedup: matriz* `fem_2d_pud`

6. Conclusão e trabalhos futuros

Nos experimentos realizados, observamos que o precondicionador paralelo híbrido SPIKE depende fortemente das transformações da matriz original através das estratégias combinatórias. Destacamos que nem sempre a aplicação de todas as estratégias simultaneamente conduz a um melhor precondicionador. Entretanto observamos que o precondicionador SPIKE foi eficiente na redução do número de iterações e tempo de processamento do método iterativo GMRES. Como trabalhos futuros, sugerimos implementações paralelas das estratégias combinatórias, visando reduzir o tempo total de processamento e o consumo de memória.

7. Agradecimentos

Os autores agradecem às agências de fomento FAPES, CAPES e CNPq pelo apoio recebido através de bolsas de estudo e suporte financeiro. Agradecemos também ao Laboratório Nacional de Computação Científica (LNCC) pelo uso do *cluster* Altix-xe.

Referências

Barnard, S. T., Pothen, A., and Simon, H. D. (1993). A spectral algorithm for envelope reduction of sparse matrices. In *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, Supercomputing '93, pages 493–502, New York, NY, USA. ACM.

Davis, T. A. and Hu, Y. (2011). The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software*, (1):1:1 – 1:25.

Duff, I. S. and Koster, J. (2000). On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Anal. Appl.*, 22(4):973–996.

Fiedler, M. (1973). Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23:298–305.

Hogg, J. and Scott, J. (2015). On the use of suboptimal matchings for scaling and ordering sparse symmetric matrices. *Numerical Linear Algebra with Applications*.

Manguoglu, M., Koyutürk, M., Sameh, A. H., and Grama, A. (2010). Weighted matrix ordering and parallel banded preconditioners for iterative linear system solvers. *SIAM J. Sci. Comput.*, 32(3):1201–1216.

Manguoglu, M., Sameh, A. H., and Schenk, O. (2009). PSPIKE: A parallel hybrid sparse linear system solver. In Sips, H., Epema, D., and Lin, H.-X., editors, *Euro-Par 2009 Parallel Processing*, volume 5704 of *LNCS*, pages 797–808. Springer Berlin Heidelberg.

Manne, F. and Sørevik, T. (1995). Optimal partitioning of sequences. *J. Algorithms*, 19(2):235–249.

Pinar, A. and Aykanat, C. (2004). Fast optimal load balancing algorithms for 1d partitioning. *J. Parallel Distrib. Comput.*, 64(8):974–996.

Polizzi, E. and Sameh, A. H. (2006). A parallel hybrid banded system solver: The SPIKE algorithm. *Parallel Comput.*, 32(2):177–194.

Polizzi, E. and Sameh, A. H. (2007). SPIKE: A parallel environment for solving banded linear systems. *Computers & Fluids*, 36(1):113–120.

Sathe, M., Schenk, O., Uçar, B., and Sameh, A. (2012). A Scalable Hybrid Linear Solver Based on Combinatorial Algorithms. In Naumann, U. and Schenk, O., editors, *Combinatorial Scientific Computing*, Chapman-Hall/CRC Computational Science, pages 95–128. Taylor & Francis.