Universidade do Estado do Rio de Janeiro

Centro de Tecnologia e Ciências

Faculdade de Engenharia

João Paulo Kely Zanardi

CTPR: Uma variante para o CPR

Rio de Janeiro

2019

João Paulo Kely Zanardi

CTPR: Uma variante para o CPR

Dissertação apresentada, como requisito parcial para obtenção do título de Doutor em Engenharia Mecânica, ao Programa de Pós-Graduação em Engenharia Mecânica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Fenômenos de Transporte.

Orientador: Prof. Dr. Luiz Mariano Paes de Carvalho Filho

Rio de Janeiro

2019

CATALOGAÇÃO NA FONTE

UERJ / REDE SIRIUS / BIBLIOTECA CTC/B

Z27 Zanardi, João Paulo Kely.
CTPR: uma variante para o CPR / João Paulo Kely Zanardi. – 2019.
50f.

Orientador: Luiz Mariano Paes de Carvalho Filho.
Tese (Doutorado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.

1. Engenharia mecânica - Teses. 2. Sistemas lineares de controle - Teses. 3. Controle automático - Teses. 4. Reservatórios - Teses. 5. Petróleo - Teses. I. Carvalho Filho, Luiz Mariano Paes de. II. Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia. III. Título.

CDU 681.51

Bibliotecária: Júlia Vieira – CRB7/6022

Assinatura

Data

João Paulo Kely Zanardi

CTPR: uma variante para o CPR

Dissertação apresentada, como requisito parcial para obtenção do título de Doutor em Engenharia Mecânica, ao Programa de Pós-Graduação em Engenharia Mecânica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Fenômenos de Transporte.

Aprovado em: 09 de dezembro de 2019.

Banca Examinadora:

Prof. Dr. Luiz Mariano Paes de Carvalho Filho (Orientador)
Instituto de Matemática e Estatística da UERJ

Prof. Dr. Norberto Mangiavacchi
Faculdade de Engenharia da UERJ

Prof. Dr. Paulo Goldfeld
Universidade Federal do Rio de Janeiro - UFRJ

Prof. Dr. Nelson Maculan
Universidade Federal do Rio de Janeiro - UFRJ

Prof. Dr. Michael Souza
Universidade Federal do Ceará - UFC

Rio de Janeiro

2019

DEDICATÓRIA

Dedico este trabalho a minha mãe Vera Lúcia de Oliveira Kely.

AGRADECIMENTO

Agradeço primeiramente a minha família que sempre me apoiou.

Agradeço ao meu orientador Luiz Mariano de Carvalho pela excelente orientação e paciência.

Agradeço aos meus amigos que me aturam mesmo eu não sendo fácil.

Agradeço a minha cadela Donatela.

Agradeço a todas as mulheres as quais me relacionei amorosamente pois me ajudaram com o estresse emocional.

Por fim, agradeço a todos os astros do rock que forneceram a trilha sonora para minha pesquisa.

It's not tragic to die
doing what you love.

Bhodi

RESUMO

ZANARDI, João Paulo. *CTPR: uma variante para o CPR*. 50 f. Dissertação (Doutorado em Engenharia Mecânica) - Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro (UERJ), Rio de Janeiro, 2019.

O estudo de sistemas lineares de grande porte ainda é um importante campo em álgebra linear computacional. Apesar dos diversos métodos de resolução e precondicionadores existentes, ainda há uma grande demanda para novos métodos, que sejam eficientes e preferencialmente paralelizáveis. Este trabalho apresenta um estudo em sistemas lineares de grande porte oriundos da simulação de reservatórios de petróleo, com enfoque principal em precondicionadores.

Palavras-chave: Sistemas Lineares; Precondicionadores; Petróleo; Álgebra Linear.

ABSTRACT

ZANARDI, João Paulo. CTPR: a variant to CPR. 50 f. Dissertação (Doutorado em Engenharia Mecânica) - Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro (UERJ), Rio de Janeiro, 2019.

The study of large linear systems is a very important area in computational linear algebra. Despite the diversity of existing solvers and preconditioners, a huge demand for new efficient and parallel methods still exists. This work presents a research in large linear systems focusing mostly on preconditioners.

Keywords: Linear Systems; Preconditioners; Oil; Linear Algebra.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

INTRODUÇÃO

Combustíveis fósseis ainda são largamente utilizados atualmente, sendo importantíssima a descoberta de novos reservatórios e a consequente avaliação da viabilidade da perfuração dos mesmos. Em nosso trabalho, realizamos uma pesquisa em cooperação com a Petrobras voltada para a simulação da explotação de reservatórios de petróleo. Mais especificamente, com a parte da simulação que diz respeito a resolução de sistemas lineares de grande porte.

A resolução eficiente dos sistemas lineares que aparecem nas simulações é de extrema importância pois a resolução do mesmo é um grande gargalo computacional na simulação. Neste texto, contaremos a trajetória da nossa pesquisa, desde a apresentação de alguns núcleos de álgebra linear computacional, em especial precondicionadores, relacionados com sistemas lineares até culminar na proposta de um novo precondicionador.

A linha do tempo será descrita nos capítulos da seguinte forma: no capítulo 1 entraremos em mais detalhes sobre sistemas lineares, tais como métodos iterativos e precondicionadores, produto matriz-vetor e método de ortogonalização de Gram-Schmidt. Ainda no primeiro capítulo, mostraremos resultados obtidos em um estágio realizado no Intel Labs. No Capítulo 2, faremos um apanhado sobre uma classe específica de precondicionadores: os precondicionadores baseados na inversa aproximada. Estes precondicionadores já foram estudados anteriormente e dentro desde trabalho foram revisitados com fins de avaliar sua robustez e possíveis novas ideias. Por fim, no Capítulo 3 apresentaremos o CPR, um precondicionador padrão para problemas da indústria petrolífera e alguns operadores de pré-processamento. Ao longo do capítulo mostraremos resultados e uma nova proposta de precondicionador, baseado no CPR, o qual batizamos de CTPR.

1 MÉTODOS DE KRYLOV E PRECONDICIONADORES

1.1 Métodos de Krylov e Precondicionadores

Em problemas reais é comum nos depararmos com sistemas lineares da forma

$$Ax = b, \tag{1.1}$$

onde $A \in \mathbb{R}^{n\times n}$ e $x \in \mathbb{R}^n$. Em nosso caso, estaremos também considerando que A é uma matriz esparsa, não singular e de grande porte. Resolver (1.1) é um grande desafio no meio científico, principalmente com problemas cada vez maiores.

Obviamente, poderíamos resolver (1.1) com algum método direto, por exemplo, com a bem conhecida eliminação gaussiana. O problema em utilizar métodos diretos é que a ordem de grandeza destes métodos é de n^3, portanto ficando computacionalmente inviáveis conforme a ordem do problema aumenta. Então, para resolvermos (1.1) utilizamos métodos iterativos [1], isto é, métodos que resolvem aproximadamente o sistema linear. Em particular, estamos interessados em métodos iterativos baseados em subespaços de Krylov [1]. Em álgebra linear, um subespaço de Krylov gerado por uma matriz $n \times n$ A e um vetor b de dimensão n é o subespaço expandido pelas imagens de b sob as primeiras k potências de A (começando por $A^0 = I$), isto é,

$$\mathcal{K}_k(A, b) = \text{span}\{b, Ab, Ab^2, \ldots, A^{k-1}b\}. \tag{1.2}$$

Os métodos de Krylov funcionam criando uma base de Krylov para o sistema e então aproximações para a solução são construídas minimizando o resíduo, o erro ou algum outro vetor em algum dos subespaços formados.

Para que um método iterativo seja eficiente é necessário o uso de precondicionadores [1]. Um precondicionador pode ser uma matriz P tal que o produto $P * A$ tenha um número de condicionamento menor. O precondicionador pode ser aplicado de três maneiras, pela direita:

$$APP^{-1}x = b, \tag{1.3}$$

então resolvemos

$$APy = b, \tag{1.4}$$

onde $y = P^{-1}x$. Pela esquerda:

$$PAx = Pb. \tag{1.5}$$

Ou ainda por ambos os lados:

$$P_1AP_2P_2^{-1}x = P_1b, \tag{1.6}$$

onde P_1 e P_2 são precondicionadores. Então resolvemos

$$P_1AP_2y = b, \tag{1.7}$$

com $y = P_2^{-1}x$.

Em todos os casos, a solução do sistema é a mesma que a do sistema original. Existem diversos tipos de precondicionadores como os baseados na fatoração LU incompleta, neste trabalho vamos dar ênfase no CPR e em uma classe de precondicionadores baseados na aproximação da inversa de A.

1.2 Produto matriz-vetor

O produto matriz-vetor é uma operação $y = A * x$. Em particular, trabalharemos apenas em casos onde A é uma matriz esparsa. O produto matriz-vetor é um núcleo muito importante em álgebra linear computacional, aparecendo constantemente em métodos iterativos, tais como métodos de Krylov. Portanto, é crucial para a eficiência de um método de Krylov uma implementação otimizada do produto matriz vetor.

O produto matriz-vetor também é altamente paralelizável, como podemos ver na Figura 1, podemos calcular o produto $A * x$ como n produtos internos $A_i * x$, onde n é a ordem da matriz e A_i é i-ésima linha de A. Obviamente cada produto $A_i * x$ é independente, deixando a implementação paralela bem direta.

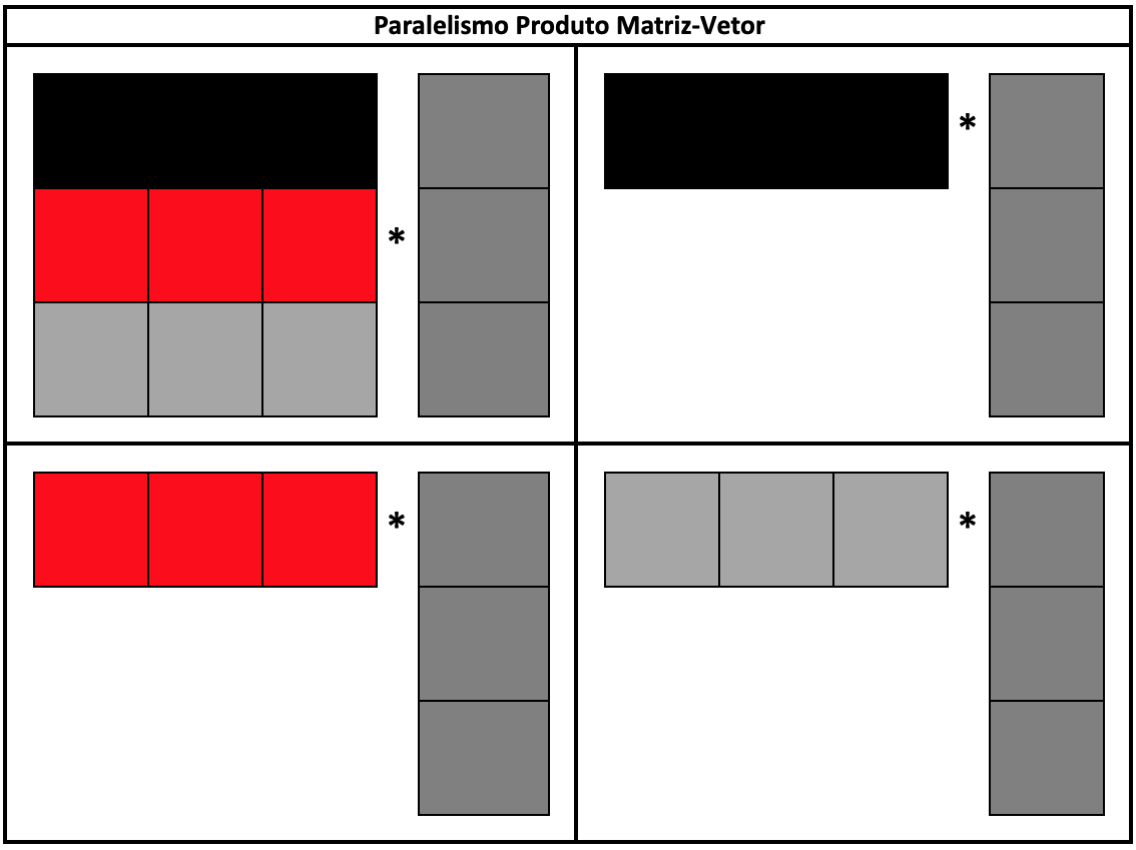

Figura 1 Paralelismo do produto matriz-vetor

1.2.1 Descrição do Código

O código foi implementado em C++, e OpenMP [2] foi a escolha para a paralelização. Os códigos foram implementados e otimizados durante um estágio no Intel Labs. As arquiteturas alvo foram Intel Xeon e Intel Xeon Phi.

O código base para o produto matriz-vetor no formato BCRS [3], pode ser escrito como:

```
1 for(int i = 0; i < size ; i++)
2     for(int j = row[i]; j < row[i+1]; j++)
3         for(int k = 0; k < bs; k++)
4             for(int l = 0; l < bs; i++)
5                 y[i*bs+j] += val[j*bs*bs+k*bs+l]*x[col[j]+k];
```

No código acima, *size* representa a ordem da matriz, *row* as linhas, *col* as colunas, *val* os valores e *bs* o tamanho do bloco.

A primeira otimização feita no código foi implementá-lo como uma função *template* do C++ em relação ao tamanho do bloco, isto é, o compilador cria, em tempo de

compilação, um código otimizado para um determinado tamanho de bloco, deixando o programa mais rápido. Para otimização do código foram utilizadas instruções do tipo SIMD usando a *flag* -xHost para habilitar instruções AVX2. As instruções utilizadas foram:

- ***#pragma simd:*** Instrução para o compilador vetorizar o loop;
- ***#pragma unroll:*** Instrução para o compilador desenrolar o loop;
- ***#pragma vector aligned:*** Instrução para o compilador utilizar dados de memória alinhados.

Para a paralelização do código com OpenMP, optamos por paralelizar o *loop* mais externo, utilizando a seguinte instrução:

#pragma omp parallel for schedule(static, chunk),

onde *chunk* é a parte inteira da divisão do número de blocos linhas pelo número de threads. A instrução ***schedule(static, chunk)*** significa que cada thread vai computar *chunk* linhas. O valor do *chunk* é calculado dividindo-se o número de linhas da matriz pelo número de threads. Caso a divisão não seja exata, se pega o próximo valor inteiro.

Além disso, foram implementados códigos em intrinsics para tamanhos de bloco 4 e 8, esses códigos são apenas para arquitetura Intel Xeon Phi. Um trecho do código em intrinsics pode ser visto abaixo.

```
1 for( k = 0;k < bs; k++){
2     __m512d a = _mm512_load_pd( &A[ j*bs*bs + k*bs]);
3     __m512d xt = _mm512_set1_pd( x[col[j]*bs + k]);
4     const __m512d axxR = _mm512_mul_pd( a, xt);
5     sR   =    _mm512_add_pd(  sR, axxR);
6 }
```

1.2.2 Resultados

Os testes foram realizados numa máquina Intel Xeon Phi com arquitetura *Sandy Bridge* de 2 sockets e 8 cores em cada um. Também foram realizados testes num acelerador

Xeon Phi com arquitetura Knights Corner. As especificações completas das máquinas podem ser vistas na (Tabela 1). Em nossos experimentos só utilizamos 1 socket e variamos o tamanho do bloco. A performance foi mensurada comparando com a largura de banda da máquina medida através do *stream bandwidth* [4].

Tabela 1 Especificações da Máquina

Sistema	Intel Xeon	Intel Xeon Phi
Microarquitetura	Sandy Bridge E5-2670	Knights Corner
Frequência	2.6 GHz	1.09 GHz
Sockets	2	1
Cores	8	60
Threads por core	1	4
Bandwidth	37 GB/s	143 GB/s

As matrizes utilizadas são oriundas de simulações reais de exploração de reservatórios de petróleo e do tradicional modelo SPE10 [5]. Essas matrizes, exceto a SPE10, foram conseguidas através de um projeto em parceria com a Petrobras. Todas as matrizes apresentam bloco de tamanho 4, como a finalidade era testar a performance foram criadas matrizes de outros tamanhos de bloco expandindo a matriz de adjacência dessas matrizes. Como se tratam de matrizes reais não podemos referenciá-las pelo seu nome, por isso designaremos as matrizes por letras. Os dados podem ser vistos na (Tabela 6).

Tabela 2 Matrizes Testadas

Matriz	A	B	C	D	E	SPE10
Bloco Linhas	66077	182558	408865	617459	1378899	1094421
Número de blocos	1797516	4894792	2731921	17516572	9249819	7478137

A Tabela 3 mostra o percentual da largura de banda atingido para blocos de tamanho 2, 4, 6 e 8, para Intel Xeon, enquanto Tabela 4 mostra o mesmo para o Intel Xeon Phi. Os testes foram feitos com 8 threads em um socket no Intel Xeon e 240 threads em 60 cores no Intel Xeon Phi.

No Intel Xeon foi possível alcançar mais de 80% da largura de banda em quase todos os casos, resultado que consideramos bons. No Intel Xeon Phi, graças ao uso do intrinsics, foi possível ultrapassar o pico teórico medido pelo *stream bandwidth* para blocos

Tabela 3 Resultados matriz-vetor no Intel Xeon

Bloco	Matriz					
	A	B	C	D	E	SPE10
2	94%	86%	79%	77%	79%	79%
4	94%	92%	87%	87%	90%	91%
6	95%	96%	93%	93%	96%	96%
8	94%	99%	96%	97%	98%	99%

Tabela 4 Resultados matriz-vetor no Intel Xeon Phi

Bloco	Matriz					
	A	B	C	D	E	SPE10
2	37%	42%	44%	41%	44%	45%
4	76%	87%	90%	84%	82%	97%
6	75%	79%	79%	78%	80%	80%
8	106%	109%	108%	109%	110%	110%

de tamanho 8, os blocos de tamanho 4 e 6 tiveram resultados bons, enquanto os resultados com bloco 2 foram ruins. A explicação para os resultados ruins com blocos de tamanho 2 é o pouco montante de trabalho por thread. A Figura 2 mostra uma comparação entre o *bandwidth* alcançado pelo Xeon e Xeon Phi para o bloco original de tamanho 4.

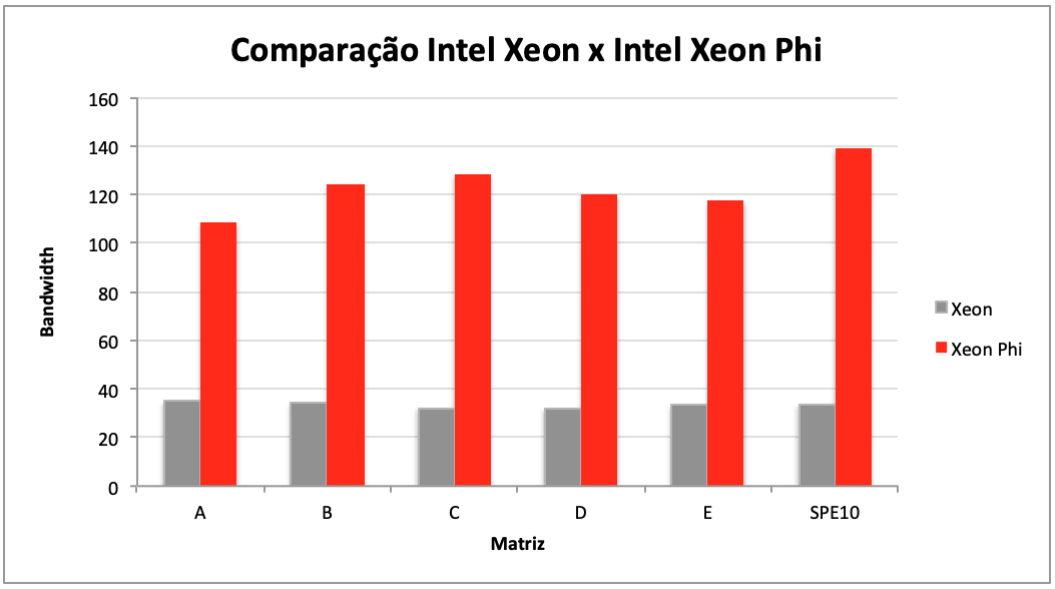

Figura 2 Comparação Intel Xeon x Intel Xeon Phi

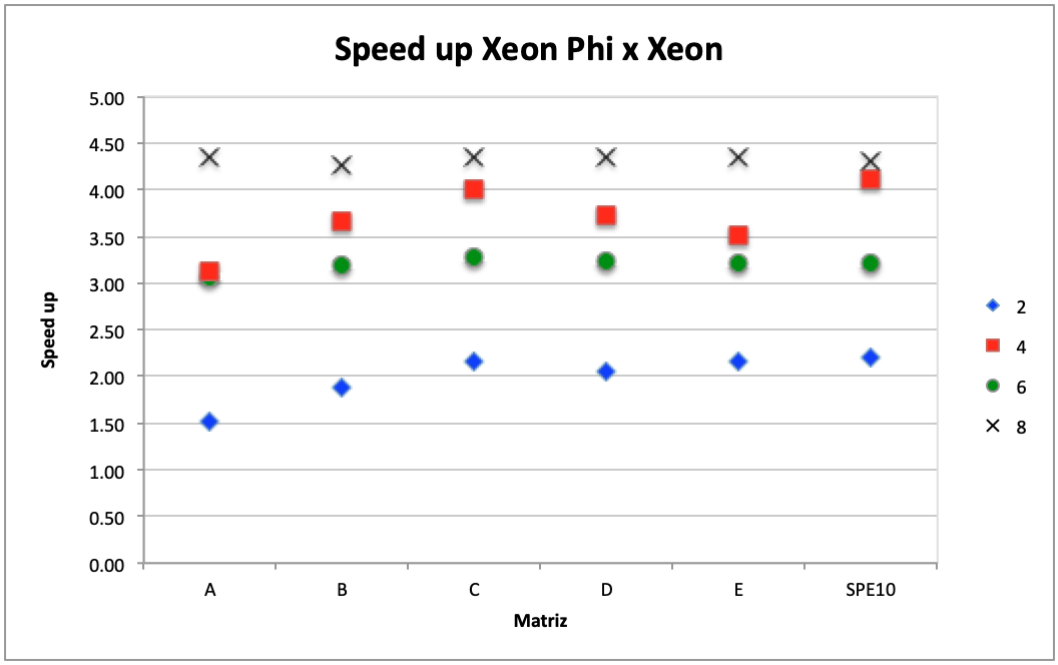

Figura 3 Speed Up do Xeon Phi sobre o Xeon

A Figura 3 mostra o *speed up* do Xeon Phi sobre o Xeon. Nela podemos notar que com exceção para os blocos de tamanho 2, é possível obter um *speed up* de no mínimo 3 vezes mudando da arquitetura para a arquitetura Xeon Phi. Resultado este considerado muito bom, uma vez que o escalamento teórico do Zeon para o Xeon Phi é de aproximadamente 3,8.

1.3 Orthomin

O Orthomin [6] é um método de Krylov para a resolução de sistemas lineares muito utilizado em simuladores de reservatórios de petróleo. O Orthomin tem como seus principais núcleos a multiplicação matriz vetor, a construção e aplicação do precondicionador e o método de ortogonalização de Gram-Schmidt [7]. Esses núcleos são os que apresentam maior custo computacional, por isso, torna-se crucial uma implementação eficaz dos mesmos. Na seção anterior, apresentamos uma versão otimizada do produto matriz vetor, agora apresentaremos uma versão otimizada para o método de Gram-Schmidt e resultados para o Orthomin como solver.

1.3.1 Gram-Schmidt

O método de ortogonalização de Gram-Schmidt é um método clássico para se obter uma base ortonormal para um conjunto de vetores, isto é, um conjunto de vetores que são ortogonais entre si e que possuem norma unitária. O método de Gram-Schmidt possui duas versões, a clássica e a modificada, ambas são equivalentes matematicamente porém distintas computacionalmente. Enquanto a versão modificada é mais estável a alternativa clássica tende a ser mais rápida em paralelo.

Vale ressaltar que apesar de nos referirmos como Gram-Schmidt estamos na verdade utilizando o método de Arnoldi [8]. O método de Arnoldi é uma versão especial do Gram-Schmidt onde os vetores a serem ortogonalizados são calculados a cada passo da iteração, enquanto no Gram-Schmidt partimos de um conjunto de vetores dados.

As versões distintas podem ser codificadas como:

```
Arnoldi

Input: A, n

for k = 1:n
    w = a_k
    for j = 1:k-1
        r_jk = <q_j,w>
    end
    for j = 1:k-1
        w = w - r_jk*q_j
    end
    r_kk = ||w||
    q_k = w/r_kk
```

```
Arnoldi Modificado

Input: A, n

for k = 1:n
    w = a_k
```

```
7     for j = 1:k-1
8         r_jk = <q_j,w>
9         w = w - r_jk*q_j
10    end
11    r_kk = ||w||
12    q_k = w/r_kk
```

As duas versões foram implementadas. A versão modificada foi implementada de maneira tradicional, já a versão clássica foi implementada substituindo as operações do tipo BLAS1 por operações do tipo BLAS2. Além disso, os *loops* foram divididos em blocos para que o compilador pudesse efetuar as contas na cache. A versão clássica também foi implementada em paralelo com OpenMP, paralelizando-se o *loop* mais externo.

1.3.2 Resultados

Os testes foram realizados numa máquina com arquitetura Ivy Bridge para Intel Xeon e Knights Corner para Intel Xeon Phi, as informações completas sobre as arquiteturas podem ser encontrados na Tabela 5.

Tabela 5 Especificações da Máquina

Sistema	Intel Xeon	Intel Xeon Phi
Microarquitetura	Ivy Bridge	Knights Corner
Frequência	3 GHz	1.09 GHz
Sockets	2	1
Cores	10	60
Threads por core	1	4
Bandwidth	40 GB/s	143 GB/s

A Tabela 6 mostra os dados das matrizes testadas, todas são oriundas de problemas reais e possuem bloco de tamanho 4.

Tabela 6 Matrizes Testadas

Matriz	A	B	C	D
Bloco Linhas	66077	182558	408865	617459
Número de blocos	1797516	4894792	2731921	17516572

Alguns de nossos experimentos foram realizados com apenas 1 *core* de Xeon, isto é, testes sequenciais. Para motivos de comparação também calculamos o *stream bandwidth* para 1 core, obtendo o resultado de 9 GB/s.

Todos os testes foram realizados 10 vezes. Dos resultados obtidos foram descartados o menor e maior valores. O resultado apresentado é a média aritmética dos demais valores.

A Tabela 7 mostra o percentual da largura de banda atingida pelo código Gram-Schmidt em sequencial, com 10 threads no Xeon e 240 threads no Xeon Phi.

Tabela 7 Resultados Gram-Schmidt

Cores	Matriz			
	A	B	C	D
1	96%	90%	68%	66%
10	91%	89%	88%	88%
240	60%	78%	85%	88%

O *speed up* pode ser visto na Tabela 8, a escalabilidade do código Gram-Schmidt para a matriz A pode ser vista na Figura 4.

Tabela 8 Speed up Gram-Schmidt

Cores	Matriz			
	A	B	C	D
$1 \rightarrow 10$	4.21	4.39	5.71	5.90
$10 \rightarrow 240$	2.38	3.14	3.49	3.59

A Tabela 9 mostra os parâmetros utilizados no Orthomin. Já Tabela 10 mostra o número de iterações gasto pelo Orthomin. Todos os casos testados convergiram.

A Figura 5 mostra o tempo gasto para a convergência do Orthomin no Intel Xeon. Além disso, a tabela também mostra a porção de tempo gasta para cada um dos núcleos. A Figura 6 mostra o mesmo para o Intel Xeon Phi. É possível ver que no Xeon o Gram-Schmidt é o núcleo mais custoso enquanto o Solver Triangular é o mais custoso no Xeon

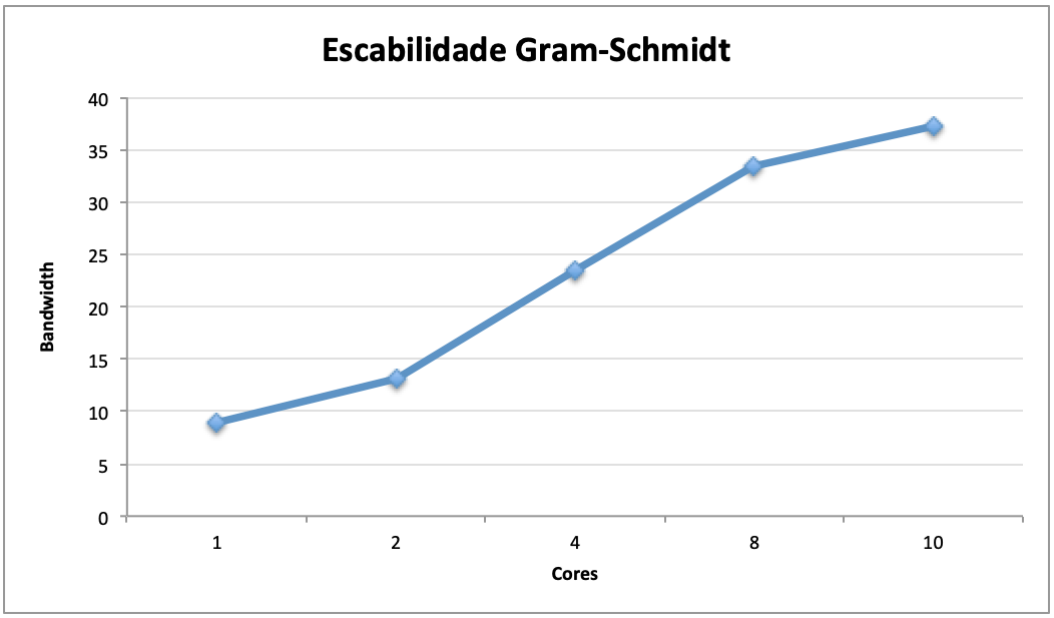

Figura 4 Escalabilidade Gram-Schmidt

Tabela 9 Parâmetros Orthomin

Restart	30
MaxIter	1000
tol	1E-6
x_0	**0**
rhs	**A*1**
Precond	ILU(1)

Tabela 10 Número de Iterações Orthomin

Matriz	A	B	C	D
its	50	63	77	38

Phi. A explicação para essa diferença é a falta de paralelismo do Solver Triangular.

A Figura 7 mostra o *speed up* do Xeon Phi sobre o Xeon. O baixo escalamento do código é justificável pois o precondicionador utilizado BILUK (versão em blocos do ILUK) e o Solver Triangular são núcleos bastante sequenciais, podendo ficar mais lentos em alguns casos. Como visto anteriormente, tanto o produto matriz-vetor quanto o método de Gram-Schmidt são altamente paralelos e escalam muito bem com mais *threads*.

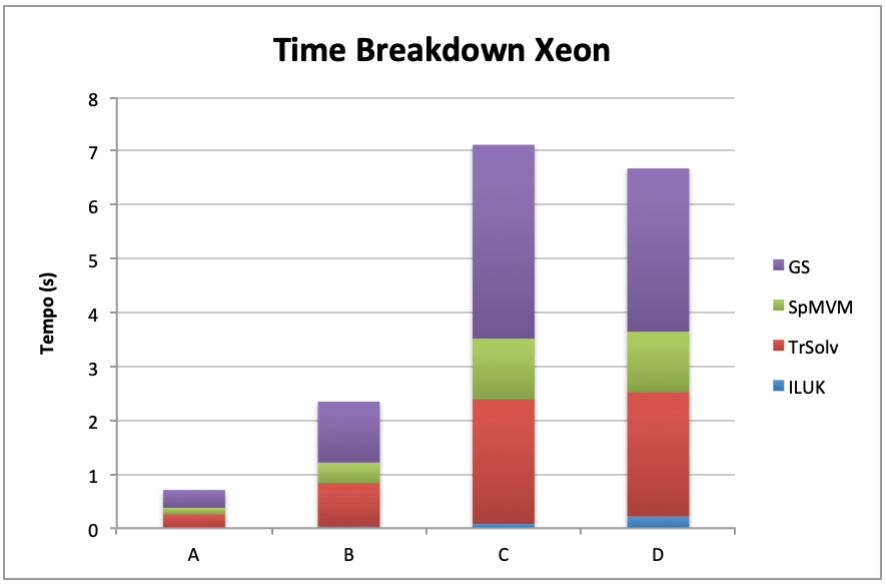

Figura 5 Time Breakdown Orthomin Intel Xeon

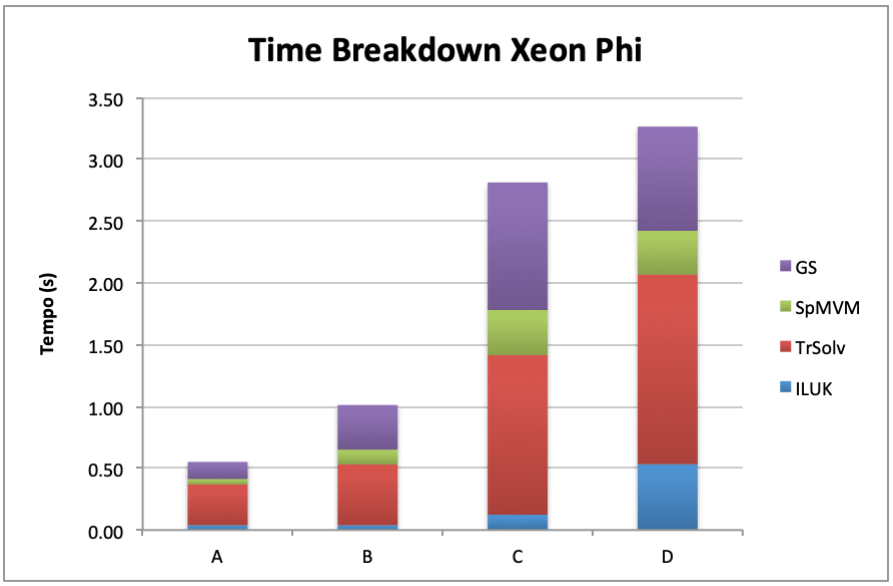

Figura 6 Time Breakdown Orthomin Intel Xeon Phi

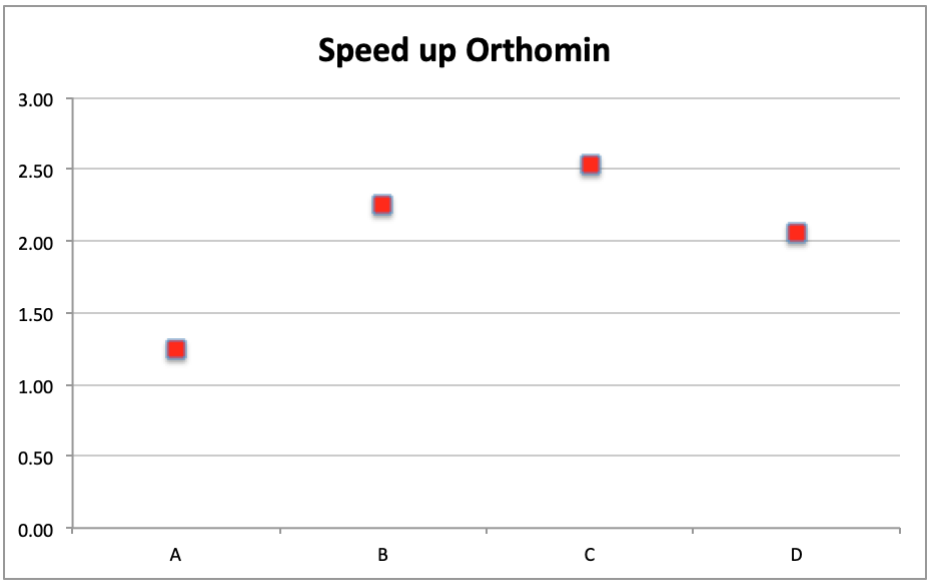

Figura 7 Speed up do Intel Xeon Phi sobre o Intel Xeon

2 PRECONDICIONADORES DE INVERSA APROXIMADA

Novamente iremos considerar a resolução de um sistema linear

$$Ax = b \tag{2.1}$$

onde $A \in \mathbb{R}^{n \times n}$ e $x \in \mathbb{R}^n$, com A sendo grande e esparsa.

Precondicionadores de Inversa Aproximada [9] são uma classe de precondicionadores M tal que $M \approx A^{-1}$ em algum sentido. Existem dois métodos para a construção dos Precondicionadores de Inversa Aproximada, métodos baseados na minimização da norma de Frobenius ou métodos de aproximação de fatores inversos. Neste capítulo, vamos fazer uma breve descrição de cinco métodos de Inversa Aproximada e apresentar resultados numéricos.

2.1 Métodos de Minimização da Norma de Frobenius

A ideia básica os métodos de minimização consiste em calcular $M \approx A^{-1}$ como a solução do seguinte problema de minimização:

$$\min_{M \in \mathcal{S}} ||I - MA||_F, \tag{2.2}$$

onde $\mathcal{S}$ é um conjunto de matrizes esparsas e $||.||_F$ denota a norma de Frobenius. Como

$$||I - AM||_F = \sum_{j=1}^{n} ||e_j - Am_j||_2^2, \tag{2.3}$$

onde e_j e m_j representam a j-ésima coluna da matriz identidade e de A, respectivamente. Então, o cálculo de M se reduz a n problemas de mínimos quadrados independentes restritos ao padrão de esparsidade do conjunto $\mathcal{S}$. Note que em 2.2 calculamos um precondicionador à esquerda, para calcular um precondicionador à direita resolvemos $\min_{M \in \mathcal{S}} ||I - AM||_F$.

A maior dificuldade dos métodos baseados na minimização da norma de Frobenius está na definição do conjunto $\mathcal{S}$. Este conjunto pode ser definido de maneira estática ou adaptativa. O **SPAI** [10] é o método mais conhecido de minimização da norma de

Frobenius, sendo subdividido em dois **SPAIS** e **SPAIA**.

SPAIS

O **SPAIS** é a versão estática do **SPAI**, isto é, nele um padrão de esparsidade $\mathcal{S}$ é definido a priori e cada coluna m_j de M é calculada da seguinte forma: para um determinado j considere o conjunto $J = \{i|(i,j) \in \mathcal{S}\}$, que especifica o padrão de elementos não nulos de m_j. Se a_i são as colunas de A e m_{ij} são os componentes de m_j, então

$$Am_j = \sum_{i=1}^{n} a_i m_{ij}. \tag{2.4}$$

Assim, as únicas colunas de A relacionadas a m_j são aquelas cujos índices pertencem a J. Seja $A(:,J)$ a submatriz de A com tais colunas e seja I o conjunto de linhas não-nulas de $A(:,J)$, podemos reduzir 2.3 para a submatriz $\tilde{A} = A(I,J)$, e para os vetores $\tilde{m_j} = m_j(J)$ e $\tilde{e_j} = e_j(I)$. As entradas de $\tilde{m_j}$ podem ser calculadas como

$$\min_{\tilde{m_j}} ||\tilde{e_j} - \tilde{A}\tilde{m_j}||_2. \tag{2.5}$$

SPAIA

Em geral, definir um bom padrão de esparsidade para a inversa de uma matriz é uma tarefa difícil. Vale a pena lembrar que mesmo A sendo esparsa, sua inversa é por diversas vezes densa. Então um modelo que computa o padrão $\mathcal{S}$ adaptativamente (**SPAIA**) foi proposto em [10].

No **SPAIA**, começamos como um padrão $\mathcal{S}$ inicial (geralmente o padrão diagonal) e, a cada iteração, a redução do resíduo de uma nova entrada (i,j) em $\mathcal{S}$ pode ser estimada por

$$\min_{\alpha_i} ||r_j + \alpha_i A e_i||_2, \tag{2.6}$$

onde e_i é a i-ésima coluna da identidade e $r = \min_{m_j \in \mathcal{S}} ||I - Am_j||_2$ é o resíduo atual. A solução de (2.6) é dada por

$$\alpha_i = -\frac{r^t A e_i}{||Ae_i||_2^2}, \tag{2.7}$$

então a norma do novo resíduo $r + \alpha_i Ae_i$ pode ser estimada como

$$\rho^2 = ||r||_2^2 - \frac{r^t Ae_i}{||Ae_i||_2^2}. \tag{2.8}$$

Apenas as entradas associadas com os melhores valores de α_i são adicionadas a $\mathcal{S}$. Este processo é repetido até que $||e_j - Am_j||$ alcance uma determinada tolerância ou que um máximo de entradas não nulas em m_j seja atingido.

2.1.1 Métodos de Aproximação dos Fatores Inversos

Os métodos de aproximação dos fatores inversos calculam uma aproximação de A^{-1} através de uma dada fatoração de A. Os métodos mais conhecidos de aproximação dos fatores inversos são o **FSAI** [11], **AINV** [12] e **NBIF** [13].

FSAI

O **FSAI** calcula aproximações $Z \approx L^{-1}$ e $W \approx U^{-1}$ a partir de dois padrões de esparsidade triangulares $\mathcal{S}_Z$ e $\mathcal{S}_w$ para Z e W, respectivamente. Estes padrões são definidos de tal forma que

$$\{(i,j)|i > j\} \subseteq \mathcal{S}_Z\{(i,j)|i = j\}\ \{(i,j)|i < j\} \subseteq \mathcal{S}_W\{(i,j)|i = j\}.$$

As entradas de Z e W podem então ser calculadas através das equações

$$\begin{aligned} Z_{ij} &= 0, \forall (i,j) \notin \mathcal{S}_Z, \\ (ZA)_{ij} &= \delta_{ij}, \forall (i,j) \in \mathcal{S}_Z, \\ W_{ij} &= 0, \forall (i,j) \notin \mathcal{S}_W, \\ (AW)_{ij} &= \delta_{ij}, \forall (i,j) \in \mathcal{S}_W. \end{aligned}$$

Por fim, os fatores Z e W são ajustados por

$$\begin{aligned} Z &= \text{diag}(|Z_{11}|^{-1/2}, \ldots, |Z_{nn}|^{-1/2})Z, \\ W &= W\,\text{diag}(\text{sign}(W_{11})|W_{11}|^{-1/2}, \ldots, \text{sign}(W_{nn})|W_{nn}|^{-1/2}), \end{aligned}$$

onde sign(a) é o sinal de um número $a \in \mathbb{R}$.

AINV

No **AINV** calcula-se $Z \approx U^{-1}$ e $W \approx L^{-T}$ onde $A = LDU$. De fato, se A é não singular e admite uma fatoração $A = LDU$, onde L é triangular inferior unitária, D é diagonal e U é triangular superior unitária, então A^{-1} pode ser fatorada como $A^{-1} = U^{-1}D^{-1}L^{-1} \approx ZD^{-1}W^T$.

O **AINV** utiliza um método de biconjugação para calcular dois conjuntos de vetores $\{z_i\}_{i=1}^n$ e $\{w_i\}_{i=1}^n$, biconjugados a A, isto é, $w_i^T A z_j = 0$, se e somente se, $i \neq j$. Assim, se definirmos as matrizes $Z = [z_1, \ldots, z_n]$ e $W = [w_1, \ldots, w_n]$, cujo a i-ésima coluna são os vetores z_i e w_i, então temos que $W^T AZ = D$, onde D é uma matriz diagonal como elementos $w_i^T A z_j \neq 0$ para todo $i = 1, \ldots, n$. Resultando em

$$A^{-1} = ZD^{-1}W^T. \tag{2.9}$$

Caso o método seja feito sem algum controle os fatores Z e W tendem a serem densos, por isso uma estratégia de *drop* é adotada para garantir a esparsidade, ou seja, as entradas de z_i e w_i que são menores que uma determinada tolerância são descartadas.

NBIF

O precondicionador *NBIF* é baseado na fórmula da inversa de Sherman-Morrison (ISM) [14]. A fórmula de ISM é dada por

$$(A + xy^T)^{-1} = A^{-1} + \frac{A^{-1}xy^TA^{-1}}{1 + y^TA^{-1}x}. \tag{2.10}$$

Se A pode ser escrita como

$$A = S - \sum_{k=1}^{n} x_k y_k^T, \tag{2.11}$$

onde S é uma matriz não singular e $\{x_k\}_{k=1}^n$ e $\{y_k\}_{k=1}^n$ são dois conjuntos de vetores.

Então por 2.10

$$A^{-1} = S^{-1} - S^{-1} Z_S \Gamma_S V_S^T S^{-1}, \tag{2.12}$$

onde $Z_S = Z(S)$ e $V_S = V(S)$ tem como vetores coluna z_k e v_k dados por

$$z_k = x_k - \sum_{i=1}^{k-1} \frac{v_i^T S^{-1} x_k}{d_i} z_i \text{ e } v_k = y_k - \sum_{i=1}^{k-1} \frac{y_k^T S^{-1} z_i}{d_i} v_i,$$

respectivamente, e $\Gamma_S = \Gamma(S) = diag(d_1, \ldots, d_n)$,

$$d_k = 1 + y_k^T S^{-1} z_k = 1 + v_k^T S^{-1} x_k$$

para $k = 1, \ldots, n$. Por simplicidade escolhemos

$$S = sI, s > 0, x_k = e_k \text{ e } y_k = (a^k - s^k)^T$$

onde a^k e s^k são as k-ésimas colunas de A e S, respectivamente. Então

$$A^{-1} = s^{-1} I - s^{-2} Z_S \Gamma_S^{-1} V_S^T. \tag{2.13}$$

Em [15] os autores mostram que se (i) Z_S, Γ_S e V_S são exatamente a ISM para alguma $s > 0$ e Z, Γ e V são exatamente a ISM para $s = 1$, então

$$Z_S = Z, V_s = V - (s-1)W \text{e } \Gamma_S = s^{-1}\Gamma,$$

onde a k-ésima coluna w_k de W é escrita como

$$w_k = x_k - \sum_{i=1}^{k-1} \frac{y_k^t z_i}{d_i} w_i,$$

com d_i representando o i-ésimo elemento da diagonal D.

Em [13] os autores melhoraram a proposição descrita em [15], eles provaram que é possível ao mesmo tempo computar a fatoração LDU e seus fatores inversos. De fato, pode-se mostrar que se (i) $A = LDU$ é a fatoração LDU exata de A e (ii) existe uma

ISM para algum $s > 0$, então

$$L = W^{-T}, D = \Gamma, U = Z^{-1} \text{ e } V_S = U^T\Gamma - sL^{-T}.$$

Então um cálculo incompleto da ISM dá uma aproximação da fatoração LDU e também aproximações de L^{-1} e U^{-1}. A estratégia de *drop* utilizada pelo **NBIF** é feita da seguinte maneira: seja w_{ik} o elemento da linha i e coluna j de $W \approx L^{-1}$ e seja ϵ uma dada tolerância, então retiramos uma entrada w_{ik} se

$$|w_{ik}|||e_k^T\bar{L}|| < \epsilon, \tag{2.14}$$

onde $\bar{L}$ é um fator incompleto da fatoração LDU. Note que em (2.14) utilizamos as duas fatorações incompletas, dos fatores e a de seus inversos.

2.1.2 Experimentos Numéricos

Os métodos **SPAIS**, **SPAIA**, **FSAI**, **AINV** e **NBIF** foram implementados em Matlab e C. Para fins comparativos também consideraremos os resultados do ILU(0) disponível para Matlab. O método iterativo escolhido para a resolução dos problemas foi o GMRES, utilizamos uma tolerância de 10^{-4} e um máximo de 1000 iterações, isto é, assumimo que o GMRES falhou quando ele não atingir a convergência em até 1000 iterações.

Os testes foram realizados em um computador Intel$^{(R)}$ Core^{TM} 2 Quad com 2.83GHz de frequência e 4Gb de memória. O compilador utilizado foi o GNU C/C++ 4.63.

As matrizes testadas são oriundas de quatro conjuntos de matrizes `SAYLOR`, `MVMMCD`, `OILGEN` e `SHERMAN`, todos conjuntos do repositório de matrizes *Matrix-Market* [16]. Estes conjuntos de matrizes foi escolhido devido à sua aplicação em modelagem de reservatório de petróleo, simulação de extração de reservatório de petróleo ou dinâmica de fluídos. A Tabela 12 especifica a área de cada um dos conjuntos. Na Tabela 11 é possível ver as matrizes de cada conjunto, nela N representa a ordem da matriz e NNZ representa o número de elementos não nulos.

Os resultados para os precondicionadores de inversa aproximada, ILU(0) e GMRES sem precondicionador podem ser visto na Tabela 13.

A Tabela 14 mostra a razão entre o número de entrada não nulas dos precondicio-

Tabela 11 Conjuntos de Matrizes

Matriz	Conjunto	N	NNZ
CDDE1	MVMMCD	961	4681
CDDE2	MVMMCD	961	4681
CDDE3	MVMMCD	961	4681
CDDE4	MVMMCD	961	4681
CDDE5	MVMMCD	961	4681
CDDE6	MVMMCD	961	4681
SAYLR1	SAYLOR	238	1128
SAYLR3	SAYLOR	1000	3750
SAYLR4	SAYLOR	3564	22316
SHERMAN1	SHERMAN	1000	3750
SHERMAN2	SHERMAN	1080	23094
SHERMAN3	SHERMAN	5005	20033
SHERMAN4	SHERMAN	1104	3786
SHERMAN5	SHERMAN	3312	20793
ORSIRR1	OILGEN	1030	6858
ORSIRR2	OILGEN	886	5970
ORSIRR3	OILGEN	2205	14133

Tabela 12 Matrizes

Conjunto	Especificação
MVMMCD	Mecânica dos fluídos
SAYLOR	Modelagem de reservatório de petróleo
SHERMAN	Modelagem de reservatório de petróleo
OILGEN	Simulação de reservatório de petróleo

nadores de inversa aproximada e o número de entradas não nulas de cada matriz, isto é, a densidade relativa de cada precondicionador. Como no ILU(0) não há preenchimento sua densidade relativa é sempre igual a 1.

A Figura 8 mostra o percentual de casos resolvidos pelos métodos. O ILU(0) mostrou robustez em resolver os problemas, os métodos FSAI, SPAIS, AINV e NBIF mostraram robustez semelhante ao ILU(0). Em relação ao número de iterações o NBIF obteve a menor média entre todos os métodos porém a alta densidade do precondicionador é um ponto negativo. É importante comentar que é possível fazer um controle de esparsidade dos métodos, porém em nossos experimentos utilizamos os métodos como foram descritos nos artigos.

Tabela 13 Resultados GMRES

Matriz	GMRES	ILU(0)	FSAI	SPAIS	SPAIA	AINV	NBIF
CDDE1	328	25	32	26	25	24	20
CDDE2	171	8	25	32	14	15	11
CDDE3	*	52	58	53	51	51	31
CDDE4	167	8	25	38	15	15	12
CDDE5	*	*	*	*	*	*	*
CDDE6	*	10	26	87	17	16	14
SAYLR1	*	13	*	383	*	*	21
SAYLR3	206	*	23	17	*	*	*
SAYLR4	*	76	23	*	*	*	*
SHERMAN1	208	18	23	17	*	18	20
SHERMAN2	*	7	7	4	2	49	9
SHERMAN3	*	172	576	300	266	398	90
SHERMAN4	217	23	24	21	24	26	17
SHERMAN5	*	17	21	18	19	24	17
ORSIRR1	*	30	76	43	*	56	19
ORSIRR2	*	28	77	17	42	59	19
ORSIRR3	119	27	74	39	*	30	16

Tabela 14 Densidade Relativa dos Precondicionadores

Matriz	FSAI	SPAIS	SPAIA	AINV	NBIF
CDDE1	2.7	4.8	5.3	4.9	9.5
CDDE2	2.7	4.8	5.3	4.9	9.5
CDDE3	2.7	4.8	5.3	4.9	9.5
CDDE4	2.7	4.88	5.3	4.9	9.5
CDDE5	2.7	4.8	5.3	4.9	9.5
CDDE6	2.7	4.8	5.3	4.9	9.5
SAYLR1	2.7	4.5	5.3	4.5	9.1
SAYLR3	2.8	5.5	3.8	3.4	6.2
SAYLR4	3.4	7.8	5.7	4.8	9.7
SHERMAN1	2.8	5.5	3.8	3.4	6.2
SHERMAN2	4.7	7.9	4.4	4.0	7.9
SHERMAN3	3.2	6.7	3.1	3.1	5.6
SHERMAN4	3.0	5.6	3.2	2.6	4.8
SHERMAN5	4.5	8.4	2.0	2.5	4.3
ORSIRR1	3.5	8.4	5.4	4.7	9.2
ORSIRR2	3.5	8.6	5.3	4.6	9.0
ORSIRR3	3.4	7.5	5.6	5.0	9.8

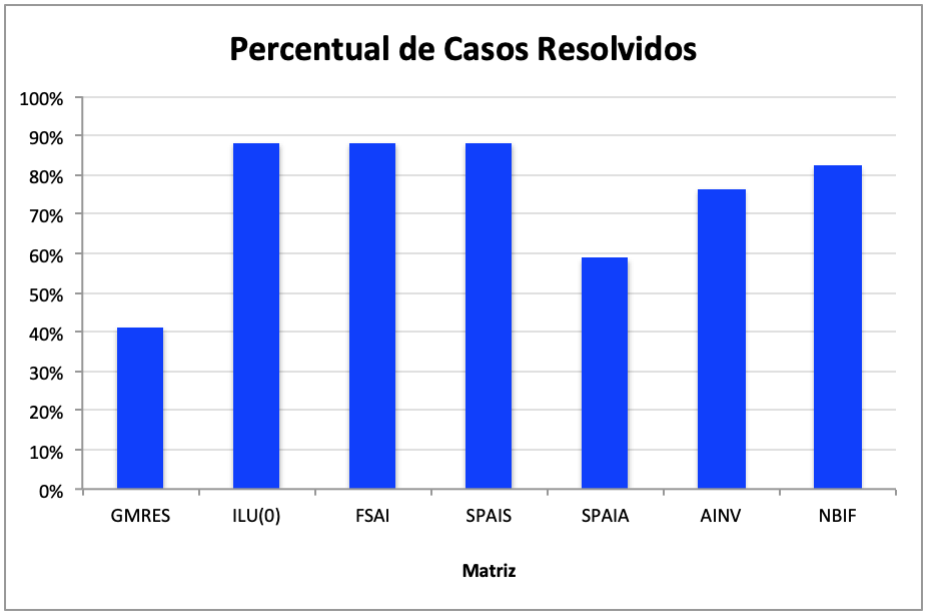

Figura 8 Percentual de Casos Resolvidos

Como esperado os resultados do ILU(0) foram melhores que os dos precondicionadores de inversa aproximada. Vale lembrar que o potencial da inversa aproximada não está na sua capacidade de resolver o problema com poucas iterações mas sim no paralelismo na construção e/ou aplicação do precondicionador. Tal fato encoraja mais estudos e testes da inversa aproximada principalmente em ambientes altamente paralelos mas impossibilita o uso a curto prazo dos mesmos em aplicações industriais.

3 O PRECONDICIONADOR CPR

3.1 Descrição do CPR

O precondicionador CPR [17], do inglês, *Constraint Pressure Residual*, é um precondicionador de dois estágios amplamente utilizado para resolver sistemas lineares oriundos de modelos totalmente implícitos (FIM). Não vamos descrever o modelo matemático aqui mas uma descrição detalhada das equações e da discretização utilizados podem ser encontrados em [18].

Suponha que temos que resolver o seguinte sistema linear:

$$Ax = b. \tag{3.1}$$

A matriz A possui uma estrutura em blocos, onde em cada bloco as linhas estão associadas as equações e as colunas associadas as variáveis. Podemos então, ordenar a matriz A variável a variável. Distinguindo p por pressão e s por saturação, uma matriz A pode ser escrita variável a variável como:

$$\begin{bmatrix} A_{pp} & A_{ps} \\ A_{sp} & A_{ss} \end{bmatrix} \begin{bmatrix} x_p \\ x_s \end{bmatrix} = \begin{bmatrix} b_p \\ b_s \end{bmatrix}.$$

Na matriz acima temos duas entidades físicas distintas, pressões e saturações. A submatriz A_{pp} representa a parte do problema relacionada às pressões, já a submatriz A_{ss} representa a parte do problema relacionada às saturações. As submatrizes $A_{ps} e A_{sp}$ representam as partes em que existe um acoplamento entre pressões e saturações.

3.1.1 Precondicionadores de dois estágios

Sejam M_1 e M_2 dois precondicionadores para a 3.1, cujo aplicaremos suas inversas (genéricas), M_1^{-1} e M_2^{-1}. Aplicar um precondicionador multiplicativo de dois estágios pode ser descrito como:

1. Precondicionar usando M_1: $x_1 = M_1^{-1}b$;
2. Computar o novo resíduo: $b_1 = b - Ax_1$;
3. Precondicionar usando M_2 e depois corrigir a solução: $x = M_2^{-1}b_1 + x_1$.

Em outras palavras, um precondicionador de dois estágios pode ser escrito como:

$$M_{1,2}^{-1} = M_2^{-1}[I - AM_1^{-1}] + M_1^{-1} \tag{3.2}$$

3.1.2 O precondicionador CPR

O precondicionador CPR, é um precondicionador multiplicativo de dois estágios com escolhas específicas para M_1 e M_2. No CPR temos

$$M_1^{-1} \approx \begin{bmatrix} A_{pp}^{-1} & 0 \\ 0 & 0 \end{bmatrix}.$$

Ou seja, no primeiro estágio do CPR resolvemos o problema das pressões. No segundo estágio escolhemos um precondicionador $M_2^{-1} \approx A^{-1}$.

O precondicionador CPR, para uma matriz totalmente implícita A, é um caso especial de (3.2) dado por

$$M_{CPR}^{-1} = M^{-1}[I - AC(W^TAC)^{-1}W^T] + C(W^TAC)^{-1}W^T \approx A^{-1}. \tag{3.3}$$

Em (3.3) $C(W^TAC)^{-1}W^T$ e M^{-1} estão identificados com M_1^{-1} e M_2^{-1} em (3.2).

Em (3.3), C é uma matriz bloco diagonal de ordem $(n_{eq} \cdot n_c) \times n_c$, onde n_{eq} e n_c denotam o número de equações e o número de células de A, respectivamente. Considerando a pressão como a primeira variável em cada célula, podemos expressar C como

$$\mathrm{C} = \begin{bmatrix} e_p & & & \\ & e_p & & \\ & & \ddots & \\ & & & e_p \end{bmatrix}, \text{ onde } \; e_p = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Já W^T é uma matriz bloco diagonal $n_c \times (n_{eq}.n_c)$ que pode ser escrita na forma

$$W^T = C^T \cdot \text{DIAG}^{-1}(A), \tag{3.4}$$

neste contexto $\text{DIAG}^{-1}(A)$ representa a matriz formada pela inversa dos blocos diagonais de A. Tendo essas matrizes, podemos restringir a matriz inicial A à matriz das pressões

$$A_{pp} = W^T AC. \tag{3.5}$$

Utilizando a mesma lógica da aplicação de um precondicionador de dois estágios descritos na seção anterior, a aplicação do CPR pode ser escrita, passa-a-passo, como:

1. Restrinja o resíduo do sistema global ao resíduo das pressões $r_p = W^T r$.
2. Resolva o sistema $x_p = (W^T AC)^{-1} r_p$ com algum solver linear e expanda a solução do sistema das pressões para o sistema completo $C \cdot x_p$ (primeiro estágio).
3. Corrija o resíduo do sistema completo usando a solução do primeiro estágio $r_{\text{corrigido}} = r - A(C \cdot x_p)$.
4. Resolva o sistema completo com o resíduo corrigido usando o precondicionador do segundo estágio $M^{-1} r_{\text{corrigido}} = M^{-1}(r - A(C \cdot x_p))$.
5. Combine as soluções de ambos os estágios

$$x = M^{-1}(r - A(C \cdot x_p)) + C \cdot x_p. \tag{3.6}$$

Tradicionalmente, as escolhas para a solução do primeiro e segundo estágio são Multigrid Algébrico [19] e fatoração LU incompleta (ILU) [20], respectivamente.

3.2 Operadores de Desacoplamento

Ao resolvermos o primeiro estágio do CPR, estamos na verdade, encontrando uma solução para o problema $A_{pp}x_p + A_{ps}x_s = b_p$. Note então, que estamos ignorando o fator

$A_{ps}x_s$ nessa solução aproximada. Para termos uma solução melhor, é comum multiplicarmos a matriz A por uma matriz triangular superior L tal que:

$$LA = \tilde{A} = \begin{bmatrix} \tilde{A}_{pp} & 0 \\ \tilde{A}_{sp} & \tilde{A}_{ss} \end{bmatrix} = \begin{bmatrix} A_{pp} - DA_{sp} & A_{ps} - DA_{pp} \\ A_{sp} & A_{ss} \end{bmatrix},$$

onde o operador L tem a seguinte forma

$$L = \begin{bmatrix} 1 & -D \\ 0 & 1 \end{bmatrix}.$$

Resolvemos então o CPR para a matriz $\tilde{A} = LA$. Note que resolver o problema para $\tilde{A}$ é o mesmo que encontrar uma solução para o complemento de Schur

$$\tilde{A}/\tilde{A}_{pp} = A_{ss} - A_{sp}(A_{pp} - DA_{sp})^{-1}(A_{ps} - DA_{pp}). \tag{3.7}$$

A submatriz D de L é escolhida de tal forma que que a submatriz $\tilde{A_{ps}}$ seja 0. Em outras palavras, $A_{ps} - DA_{ss} = 0$. Existem algumas escolhas possíveis, uma que surge de forma bem intuitiva é utilizar Eliminação-Gaussiana, assim

$$L = \begin{bmatrix} 1 & -\frac{A_{ps}}{A_{ss}} \\ 0 & 1 \end{bmatrix}.$$

Analogamente, podemos definir construir um operador R, triangular inferior, que ao ser multiplicado à direita de A (AR), desacopla pressões e saturações.

Construir um operador de desacoplamento tal que $\tilde{A_{ps}} = 0$ pode ser custoso, então construímos L (e R) que reduza o impacto de $\tilde{A_{ps}}$ e cujo custo seja inexpressivo.

Também é importante comentar que um operador de desacoplamento não é um precondicionador, isto é, a aplicação destes operadores não reduz o número de condicionamento da da matriz original A.

3.2.1 Notação

Para definirmos matematicamente os operadores de desacoplamento, usaremos a notação ponto a ponto. Seja n_c o número de células e n_{eq} o número de variáveis por célula, a matriz jacobiana pode ser ordenada em diferentes blocos $[A]^{i,j}$. A notação variável a variável apresentada anteriormente pode ser aplicada em cada um desses blocos.

$$A = \begin{pmatrix} [A]^{1,1} & \cdots & [A]^{1,n_p} \\ \vdots & \ddots & \vdots \\ [A]^{n_p,1} & \cdots & [A]^{n_p,n_p} \end{pmatrix} = \begin{pmatrix} [A_{pp}]^{1,1} & [A_{ps}]^{1,1} & & [A_{pp}]^{1,n_p} & [A_{ps}]^{1,n_p} \\ [A_{sp}]^{1,1} & [A_{ss}]^{1,1} & \cdots & [A_{sp}]^{1,n_p} & [A_{ss}]^{1,n_p} \\ & \vdots & \ddots & \vdots & \\ [A_{pp}]^{n_p,1} & [A_{ps}]^{n_p,1} & \cdots & [A_{pp}]^{n_p,n_p} & [A_{ps}]^{n_p,n_p} \\ [A_{sp}]^{n_p,1} & [A_{ss}]^{n_p,1} & & [A_{sp}]^{n_p,n_p} & [A_{ss}]^{n_p,n_p} \end{pmatrix}$$

Por último, cada entrada da matriz pode ser endereçada como $[a_{x,y}]^{i,j}$ com x e y definindo a posição dentro do bloco i, j:

$$[A_{pp}]^{i,j} = [a_{1,1}]^{i,j} \qquad [A_{ps}]^{i,j} = ([a_{1,2}]^{i,j} \cdots [a_{1,n_u}]^{i,j})$$

$$[A_{sp}]^{i,j} = \begin{pmatrix} [a_{2,1}]^{i,j} \\ \vdots \\ [a_{n_u,1}]^{i,j} \end{pmatrix} \qquad [A_{ss}]^{i,j} = \begin{pmatrix} [a_{2,2}]^{i,j} & \cdots & [a_{2,n_u}]^{i,j} \\ \vdots & \ddots & \vdots \\ [a_{n_u,2}]^{i,j} & \cdots & [a_{n_u,n_u}]^{i,j} \end{pmatrix}.$$

Tendo estabelecido a notação, vamos descrever os operadores de desacoplamento propostos em [21]. Esses operadores de pré-processamento que além de desacoplar pressões e saturações não afetam a convergência do Multigrid, o que é fundamental para a eficiência do CPR.

3.3 Operadores pela Esquerda

Vamos começar descrevendo o operador pela esquerda L. O operador L tem uma estrutura bloco diagonal, onde cada bloco diagonal é triangular superior. Podemos então,

escrever L da seguinte forma

$$L = \begin{pmatrix} [L]^1 & & & \\ & [L]^2 & & \\ & & \ddots & \\ & & & [L]^n \end{pmatrix}.$$

O primeiro operador L é definido como

$$[L]^i = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix}.$$

Note que após a aplicação de L, a matriz $\tilde{A}_{pp}$ será a soma todas as equações em A. Consequentemente, a equação da pressão é próxima da equação da pressão total usada no modelo IMPES, e assim este operador é conhecido como quasi-IMPES. É esperado que este modelo não apresente nenhuma nova dificuldade ao Multigrid, porém se o método apresentar dificuldades a matriz precondicionada continuará com os mesmos problemas.

O segundo operador é o sugerido pelos autores em [21] chamado de DRS. O DRS, do inglês *Dynamic Row Sum* é um operador de desacoplamento que não só tem como finalidade desacoplar pressões de saturações mas também auxiliar na convergência do Multigrid.

No DRS cada bloco $[L]^i$ é definido como

$$[L]^i = \begin{pmatrix} \delta_1^i & \delta_2^i & \cdots & \delta_{n_u}^i \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix},$$

onde

$$\delta_x^i = \begin{cases} 0, & \text{se} \quad \frac{[a_{x,1}]^{i,i}}{\sum_{j=1,j\neq i}^{n_p} |[a_{x,1}]|^{i,j}} < \epsilon_{dd} \\ 1, & \text{c.c} \end{cases}$$

com o parâmetro ϵ_{dd} atendendo a seguinte desigualdade, $0 < \epsilon_{dd} < 1$.

A vantagem da aplicação do DRS é que apenas as partes de A que não vão apresentar dificuldades significantes ao Multigrid são consideradas na construção de $\tilde{A}_{pp}$, o que pode impedir dificuldades na convergência do Multigrid. E, em particular, o DRS previne o aparecimento de elementos diagonais negativos.

Observações:

- De acordo com o texto base, o valor para ϵ_{dd} que teve os melhores resultados foi $\epsilon_{dd} = 0,2$.
- Caso $\delta_1^i = 0$ a matriz $[L]^i$, e portanto L, será singular. Isto pode ser corrigido pelo seguinte procedimento
  - Se $\delta_x^i \neq 0$ para algum $x > 1$, então na x-ésima linha a diagonal deve ser movida para a primeira linha.
  - Se $\delta_x^i = 0$ para todo x, defina $\delta_x^i = 1$, onde x é a linha de A que menos viola a diagonal dominância. Caso $x \neq 1$, o problema cai no caso anterior.

3.4 Operadores pela Direita

Ao construir a matriz precondicionada $\tilde{A}$, o desacoplamento algébrico de pressão e saturação (i.e. reduzir $||\tilde{A}_{ps}||$) não foi considerado. Para isso, apresentaremos dois operadores a serem aplicados pela direita. Tais operadores garantem que a matriz de pressão construída pela aplicação de operadores pela esquerda não seja afetada pelo desacoplamento. Analogamente aos operadores pela esquerda, estes operadores também possuem estrutura bloco diagonal:

$$R = \begin{pmatrix} [R]^1 & & & \\ & [R]^2 & & \\ & & \ddots & \\ & & & [R]^n \end{pmatrix},$$

sendo o CPR agora aplicado a matriz $\hat{A} = \tilde{A}R$.

O primeiro operador é conhecido como qABF (quasi-ABF) constrói $[R]^i$ invertendo matrizes pequenas V^i formada pela parte triangular superior de $[\tilde{A}]^{i,i}$:

$$V^i = \begin{pmatrix} [\tilde{A}_{pp}]^{i,i} & [\tilde{A}_{ps}]^{i,i} \\ & [\tilde{A}_{ss}]^{i,i} \end{pmatrix} \quad \text{e} \quad (V^i)^{-1} = \begin{pmatrix} S^i & T^i \\ & U^i \end{pmatrix},$$

com $S^i \in \mathbb{R}, T^i \in \mathbb{R}^{1\times n_u}$ e $U^i \in \mathbb{R}^{n_u \times n_u}$.

Então, $[R]^i$ pode ser finalmente definido como

$$[R]^i = \begin{pmatrix} 1 & T^i \\ & U^i \end{pmatrix}.$$

É garantido que após a aplicação, nenhum novo bloco não nulo será introduzido, porém novas entradas podem ser introduzidas nos blocos já existentes.

O segundo método apresenta um risco menor de aparecimento de novas entradas, isso se deve ao fato de uma saturação ser utilizada para eliminar as saturações em outras colunas. Este método, chamado de SCE (*Saturation-Column-Eliminating*), não desacopla perfeitamente pressão e saturação (nem nos blocos diagonais), contudo é assegurado que as colunas relacionadas a pressão e a saturação não são misturadas.

Para construir o SCE, primeiro o residual inicial $r = b - Ax_0$ é computado. Então a saturação z com o menor resíduo é escolhida. A construção de $[R]^i$ é novamente baseada no bloco diagonal $[\tilde{A}]^{i,i}$:

$$[R]^i = \begin{pmatrix} 1 & 0 & & \cdots & & \cdots & & 0 \\ 0 & 1 & & & & & & \\ \vdots & & \ddots & & & & & \\ & & & 1 & & & & \\ 0 & -\frac{\tilde{a}^{i,i}_{1,2}}{\tilde{a}^{i,i}_{1,z}} & \cdots & -\frac{\tilde{a}^{i,i}_{1,z-1}}{\tilde{a}^{i,i}_{1,z}} & 1 & -\frac{\tilde{a}^{i,i}_{1,z+1}}{\tilde{a}^{i,i}_{1,z}} & \cdots & -\frac{\tilde{a}^{i,i}_{1,n_u}}{\tilde{a}^{i,i}_{1,z}} \\ \vdots & & & & & 1 & & \\ & & & & & & \ddots & \\ 0 & & & & & & & 1 \end{pmatrix}.$$

Após a aplicação do SCE, a pressão na diagonal é desacoplada de todas as saturações exceto na z-ésima posição, porém esta é a com menor resíduo.

3.5 CTPR: uma variante para o CPR

Vamos lembrar que no primeiro estágio do CPR estamos tentando encontrar uma solução aproximada para o problemas das pressões $A_{pp}x_p + A_{ps}x_s = b_p$. E que aplicamos um operador de desacoplamento pela esquerda (ou direita) de tal forma que

$$LA = \tilde{A} = \begin{bmatrix} \tilde{A}_{pp} & 0 \\ \tilde{A}_{sp} & \tilde{A}_{ss} \end{bmatrix}.$$

De acordo com [18] podemos fazer a seguinte suposição:

$$-\frac{A_{ps}}{A_{ss}} \approx 1. \tag{3.8}$$

Esta suposição dá origem ao modelo quasi-IMPES e serve de base para nossa variante proposta no CPR. Matematicamente falando, ao aplicarmos o operador quasi-IMPES na matriz A estamos somando todas as equações na primeira equação e com isso reduzindo o acoplamento entre pressões e saturações. Como o desacoplamento é importante para a resolução do primeiro estágio do CPR, sugerimos uma alteração na restrição ao primeiro estágio, no qual faríamos o desacoplamento ao mesmo tempo que estamos restringindo o problema.

Para fazer esta alteração, trocamos o operador W^t por um operador P, tal que :

$$[P] = \begin{pmatrix} v1 & & & \\ & v1 & & \\ & & \ddots & \\ & & & v1 \end{pmatrix},$$

com $v1 = [1\ 1\ \ldots\ 1]$.

Neste novo operador não fazemos o *scaling* com a inversa da diagonal de A, pois ao fazermos o produto da inversa da diagonal com P estaríamos alterando o valor da inversa do componente pressão, o que não acontece na versão tradicional. O operador C

continuará sendo o mesmo, e ainda será aplicado pela direita. Como dito anteriormente, é esperada que a utilização da pressão total não adicione novas dificuldades ao Multigrid, assim sendo não interferindo na convergência do primeiro estágio. Batizamos essa variante do CPR de CTPR (*Constrained Total Pressure Residual*).

O CTPR pode ser descrito, de forma análoga ao CPR, da seguinte maneira:

1. Restrinja o resíduo do sistema global ao resíduo das pressões $r_p = P^T r$.
2. Resolva o sistema $x_p = (P^T AC)^{-1} r_p$ com algum solver linear e expanda a solução do sistema das pressões para o sistema completo $C \cdot x_p$ (primeiro estágio).
3. Corrija o resíduo do sistema completo usando a solução do primeiro estágio $r_{\text{corrigido}} = r - A(C \cdot x_p)$.
4. Resolva o sistema completo com o resíduo corrigido usando o precondicionador do segundo estágio $M^{-1} r_{\text{corrigido}} = M^{-1}(r - A(C \cdot x_p))$.
5. Combine as soluções de ambos os estágios

$$x = M^{-1}(r - A(C \cdot x_p)) + C \cdot x_p. \tag{3.9}$$

Uma dúvida que pode surgir é: O CTPR não seria igual ao CPR combinado com o modelo quasi-IMPES?
A resposta é não. No CTPR, o problema do primeiro estágio, ou seja, o problema das pressões é o mesmo que no CPR combinado com o modelo quasi-IMPES, mas o segundo estágio (problema global) continua sendo o mesmo, enquanto no CPR com quasi-IMPES mudamos o sistema original também.

Vale ressaltar, que a restrição ao primeiro estágio do CPR é similar ao procedimento de espaços grosseiros, e portanto diversos operadores de restrição poderiam ser utilizados. Inclusive, um modelo dinâmico que calcula quais entradas da pressão serão somadas poderia ser feito. Este modelo já seria similar ao DRS, mas dependente de alguma heurística para decidir qual entrada é relevante na resolução do problema.

Na próxima seção, apresentaremos resultados de testes feitos com o CPR e o CTPR, e com ambos combinados com os operadores de desacoplamento descritos.

3.6 Resultados

O código foi implementado em Matlab de acordo com [17]. Os operadores de desacoplamento descritos também foram implementados.

Na Tabela 15 estão os dados das matrizes testadas. O conjunto de matrizes testadas consiste no modelo SPE01 e quatro matrizes de problemas reais fornecidas pela Petrobras. Todas as matrizes possuem estrutura de bloco com tamanho de bloco igual a 4.

Tabela 15 Matrizes Testadas

Matriz	SPE01	A	B	C	D
Bloco Linhas	300	4591	7507	9558	10408
Número de blocos	7120	116412	193132	234128	240904

Os testes foram realizados em um Notebook i5 com Ubuntu 18.04. Um pacote de Multigrid [22] foi usado para o primeiro estágio, no segundo estágio utilizamos GMRES como solver e ILU(0) como precondicionador. Foram realizados testes com o CPR, o CTPR e com ambos após a aplicação dos operadores de desacoplamento. Para fins comparativos realizamos testes utilizando GMRES com ILU(0). Os parâmetros utilizados para o GMREs foram *restart = 30*, *tol = 1E-6* e *maxiter = 500*.

A Tabela 16 mostra um comparativo entre o número de iterações do GMREs com ILU(0) com o CPR e CTPR para todos os modelos testados. Na tabela também são mostrados o resíduo e erro relativos. O GMRES com ILU(0) convergiu para todos os casos, mas apresentou um alto número de iterações. Já o CPR obteve convergência para dois dos cinco casos testados. O CTPR se mostrou o melhor dos três métodos tendo convergido para todos os casos e em menos iterações que o GMRES.

Em nenhum dos casos testados o Multigrid convergiu. Nos casos 3 e 4 o CPR apresenta o número de iterações igual a 1 pois a matriz precondicionada é mal condicionada, assim a redução do resíduo na primeira iteração é drástica resultando em uma falsa convergência.

Tabela 16 Comparativo ILU x CPR x CTPR

	GMRES			CPR			CTPR	
Matriz	its	relRes	err	its	relRes	err	its	relRes	err
SPE01	17	7,29E-5	9,99E-4	2	2,26E-5	2,29E-5	5	9,49E-6	4,24E-6
A	27	9,55E-5	0,0667	4	1,08E-5	1E-5	6	8,22E-5	7,81E-5
B	63	9,69E-5	0,18	7	2,63E-6	46,49	5	3,51E-5	1,45E-4
C	85	9,83E-5	0,0018	1	1,33E-15	0,95	13	7,69E-5	2,66E-4
D	55	9,20E-5	0,0012	1	1,64E-13	0,99	17	4,48E-5	1,61E-4

Tabela 17 Resultados CPR com Operadores

	Matrizes				
Método	SPE01	A	B	C	D
qIMP	2	4	7	1*	1*
DRS	2	4	7	1*	1*
qABF	10*	25*	51*	97	60
SCE	2	*	*	*	1*
qIMP + qABF	4	6	2*	1*	1*
DRS + qABF	*	*	*	*	*
qIMP + SCE	2	4	7	1*	1*
DRS + SCE	2	*	*	*	*

A Tabela 17 mostra o resultado do CPR com os operadores de pré-processamento. Fizemos testes aplicando somente os operadores à esquerda, somente os operadores à direita e aplicando à direita e esquerda simultaneamente. Por simplicidade, colocaremos só o número de iterações e quando o número de iterações significar uma falsa convergência adicionares um * ao lado deste número.

Os operadores quasi-IMPES e DRS, quando aplicados sozinhos, não introduziram novas dificuldades a resolução dos problemas, porém não ofereceram, no que se refere a solução do problema, nenhuma vantagem. A combinação quasi-IMPES com SCE teve o mesmo resultado descrito. O operador qABF conseguiu obter convergência nos casos C e D, mas não convergiu para os casos SPE01, A e B, os quais convergem apenas com o CPR. Em geral, os casos de não convergência foram causados por singularidades no ILU. Essas singularidades podem ser explicadas pelo fato de estarmos utilizando o ILU nativo do Matlab e não uma versão em blocos do ILU que seria mais apropriada.

A Tabela 18 mostra os resultados do CTPR combinado com os operadores de

Tabela 18 Resultados CTPR com Operadores

	Matrizes				
Método	SPE01	A	B	C	D
qIMP	5	6	5	13	17
DRS	5	6	5	13	17
qABF	4	6	4	14	17
SCE	5	*	*	*	17
qIMP + qABF	4	6	4*	13	17
DRS + qABF	*	*	*	*	*
qIMP + SCE	4	5	5*	13	17
DRS + SCE	5	*	*	*	*

desacoplamento. O modelo tabela é o mesmo apresentado Tabela 17.

Quando combinado com os operadores de pré-processamento, o CTPR apresentou resultados melhores que o CPR. Apenas as combinações DRS + qABF e DRS + SCE apresentaram resultados similares. Porém os resultados são similares a aplicação do CTPR. Além disso, uma vez que o CTPR desacopla pressões e saturações no primeiro estágio entendemos que o CTPR não precisaria ser combinado com operadores de desacoplamento.

CONCLUSÃO

Neste trabalho apresentamos a linha do tempo da nossa pesquisa de doutorado, começando pela implementação de diversos núcleos de álgebra linear e posteriormente aprofundando os estudos em precondicionadores. Revisitamos os precondicionadores de inversa aproximada e uma análise dos mesmos optamos por investir no CPR.

No primeiro capítulo, apresentamos o trabalho feito no estágio na Intel. Este estágio foi focado na implementação de códigos otimizados de alguns núcleos de álgebra linear computacional, a saber, o produto matriz vetor e o método de Gram-Schmidt. Por mais que nada novo tenha sido apresentado neste estágio, a pesquisa e desenvolvimento ainda está relacionada com o tema da nossa tese e consideramos que os resultados obtidos foram satisfatórios.

Já no segundo capítulo, revisitamos os precondicionadores baseados na inversa aproximada, estudados anteriormente no mestrado. A ideia, era fazer uma avaliação profunda de desempenho destes precondicionadores quando comparados com o tradicional ILU. Em geral, o ILU mostrou se mais eficiente que os precondicionadores de inversa aproximada. Dentre os precondicionadores de inversa aproximada, o AINV teve os melhores resultados mas ainda é um precondicionador com construção sequencial.

No terceiro capítulo, estudamos o CPR, um precondicionador de dois níveis muito utilizado em problemas da indústria petrolífera. O estudo do CPR foi de crucial importância para nosso trabalho pois o mesmo foi feito em parceria com a Petrobras. Também, estudamos e apresentamos alguns operadores de desacoplamento que tem como intuito reduzir o acoplamento entre as diferentes grandezas físicas presentes na matriz, a saber, pressão e saturações. Esses operadores tem como desvantagem introduzir dificuldades aos estágios do CPR, principalmente o primeiro estágio resolvido com o Multigrid. Originalmente, pretendíamos propor um operador que além de desacoplar pressões e saturações não fosse prejudicial ao Multigrid, mas as ideias que surgiram não se mostraram eficientes. No entanto, o entendimento da necessidade dos operadores de desacoplamento somados ao estudo de um operador específico, conhecido como modelo quasi-IMPES, nos

inspirou para propor uma variação do CPR, o CTPR, que resolve o primeiro estágio para o espaço das pressões totais, fazendo nesse estágio um desacoplamento de pressões e saturações.

Resultados foram apresentados para um conjunto de matrizes e o CTPR se mostrou superior ao CPR e ao ILU. Vale ressaltar, que o ambiente de testes não foi o melhor, porém todos os métodos testados estavam em igualdade de condições. Na continuidade da pesquisa, pretendemos migrar os códigos para um ambiente computacional adequado, provavelmente PETSc, e efetuar testes com matrizes mais significativas. Além disso, no PETSc teremos um código nativo para Multigrid e uma versão em bloco do ILU, que consideramos serem cruciais para a eficiência do método.

REFERÊNCIAS

[1] BARRETT, R. et al. *Templates for the solution of linear systems: building blocks for iterative methods*. [S.l.]: Siam, 1994.

[2] DAGUM, L.; MENON, R. Openmp: An industry-standard api for shared-memory programming. *Computing in Science & Engineering*, IEEE, n. 1, p. 46–55, 1998.

[3] INTEL. *Sparse BLAS BSR Matrix Storage Format*. Disponível em: <https://software.intel.com/en-us/mkl-developer-reference-c-sparse-blas-bsr-matrix-storage-format>. Acesso em: 13 de Março de 2019.

[4] MCCALPIN, J. D. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, p. 19–25, dez. 1995.

[5] CHRISTIE, M. A.; BLUNT, M. et al. Tenth spe comparative solution project: A comparison of upscaling techniques. 2001.

[6] VINSOME, P. et al. Orthomin, an iterative method for solving sparse sets of simultaneous linear equations. 1976.

[7] BJÖRCK, Å. Numerics of gram-schmidt orthogonalization. *Linear Algebra and Its Applications*, Elsevier, v. 197, p. 297–316, 1994.

[8] ARNOLDI, W. E. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of applied mathematics*, v. 9, n. 1, p. 17–29, 1951.

[9] BENZI, M.; TUMA, M. A comparative study of sparse approximate inverse preconditioners. *Applied Numerical Mathematics*, Elsevier, v. 30, n. 2-3, p. 305–340, 1999.

[10] GROTE, M. J.; HUCKLE, T. Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing*, SIAM, v. 18, n. 3, p. 838–853, 1997.

[11] KOLOTILINA, L. Y.; YEREMIN, A. Y. Factorized sparse approximate inverse preconditionings i. theory. *SIAM Journal on Matrix Analysis and Applications*, SIAM, v. 14, n. 1, p. 45–58, 1993.

[12] BENZI, M.; TUMA, M. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM Journal on Scientific Computing*, SIAM, v. 19, n. 3, p. 968–994, 1998.

[13] BRU, R. J. J.; TUMA, M. Balanced incomplete factorization. *SIAM Journal on Scientific Computing*, SIAM, v. 30, n. 5, p. 2302–2318, 2008.

[14] HAGER, W. W. Updating the inverse of a matrix. *SIAM review*, SIAM, v. 31, n. 2, p. 221–239, 1989.

[15] BRU, R. et al. Preconditioning sparse nonsymmetric linear systems with the sherman–morrison formula. *SIAM Journal on Scientific Computing*, SIAM, v. 25, n. 2, p. 701–715, 2003.

[16] NIST. *Matrix Market*. Disponível em: <https://math.nist.gov/MatrixMarket/>. Acesso em: 13 de Março de 2019.

[17] CAO, H. et al. Parallel scalable unstructured cpr-type linear solver for reservoir simulation. 2005.

[18] SCHEICHL, R.; MASSON, R.; WENDEBOURG, J. Decoupling and block preconditioning for sedimentary basin simulations. *Computational Geosciences*, Springer, v. 7, n. 4, p. 295–318, 2003.

[19] RUGE, J. W.; STÜBEN, K. *Algebraic multigrid*. [S.l.]: SIAM, 1987. 73–130 p.

[20] MEIJERINK, J. A.; VORST, H. A. V. D. An iterative solution method for linear systems of which the coefficient matrix is a symmetric m-matrix. *Mathematics of computation*, v. 31, n. 137, p. 148–162, 1977.

[21] GRIES, S. et al. Preconditioning for efficiently applying algebraic multigrid in fully implicit reservoir simulations. *SPE Journal*, Society of Petroleum Engineers, v. 19, n. 04, p. 726–736, 2014.

[22] CHIN-TIEN WU. *algebraic multigrid linear solver*. Disponível em: <https://www.mathworks.com/matlabcentral/fileexchange/35866/-algbraic-multigrid-linear-solver>. Acesso em: 13 de Março de 2019.