Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Faculdade de Engenharia

Daniel Estrela Lima Fonseca

Comparação do Desempenho spMv entre Formatos de Armazenamento de Matrizes Esparsas Provenientes do Método AIM de Simulação de Reservatórios

Rio de Janeiro
2016

Daniel Estrela Lima Fonseca

Comparação do Desempenho spMv entre Formatos de Armazenamento de Matrizes Esparsas Provenientes do Método AIM de Simulação de Reservatórios

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, 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 Carvalho

Rio de Janeiro

2016

CATALOGAÇÃO NA FONTE

UERJ / REDE SIRIUS / BIBLIOTECA CTC/B

F676 Fonseca, Daniel Estrela Lima.
Comparação do desempenho spMv entre formatos de armazenamento de matrizes esparsas provenientes do método AIM de simulação de reservatórios / Daniel Estrela Lima Fonseca. – 2016.
41f.

Orientadores: Luiz Mariano Paes de Carvalho Filho.
Dissertação (Mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.

1. Engenharia Mecânica. 2. Matriz esparsa - Dissertações. 3. Simulação numérica - Dissertações. 4. Análise de desempenho - Dissertações. I. Carvalho Filho, Luiz Mariano Paes de. II. Universidade do Estado do Rio de Janeiro. III. Título.

CDU 512.643

Assinatura

Data

Daniel Estrela Lima Fonseca

Comparação do Desempenho spMv entre Formatos de Armazenamento de Matrizes Esparsas Provenientes do Método AIM de Simulação de Reservatórios

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

Aprovada em 28 de Novembro de 2016.
Banca Examinadora:

Prof. Dr. Luiz Mariano Carvalho (Orientador)
Departamento de Matemática Aplicada – UERJ

Dr. José Roberto Pereira Rodrigues
Centro de Pesquisas e Desenvolvimento Leopoldo Américo Miguez de Mello - CENPES

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

Rio de Janeiro
2016

RESUMO

FONSECA, D. E. L. *Comparação do Desempenho spMv entre Formatos de Armazenamento de Matrizes Esparsas Provenientes do Método AIM de Simulação de Reservatórios.* 2016. 41 f. Dissertação (Mestrado em Engenharia Mecânica) – Faculdade de Engenharia
, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2016.

O presente trabalho faz uma avaliação de desempenho da multiplicação matriz esparsa por vetor denso (spMv), comparando dois formatos de armazenamento para as matrizes esparsas que surgem na simulação numérica de reservatórios de petróleo, mais especificamente a abordagem AIM (Adaptative Implicit Method). Num dos formatos, a malha do problema é previamente reordenada de modo que a matriz possa ser subdividida em quatro matrizes, todas em bloco, com o tamanho dos seus respectivos blocos homogêneos. No outro, utiliza-se a ordenação natural da malha e como resultado temos uma matriz em blocos com tamanho dos blocos variado. É mostrado que o segundo formato, de um modo geral, possui pouca perda de desempenho em relação ao primeiro, que apresenta problemas quanto a aplicação de pré-condicionadores para melhorar a utilização de solvers lineares iterativos. O software de perfilamento Vtune foi utilizado para identificar o que ocorre com cada um dos formatos quando o spMv é executado.

Palavras-chave: Matriz esparsa; Multiplicação matriz vetor; Simulação de reservatórios; Diferenças finitas.

ABSTRACT

FONSECA, D. E. L.*spMv Performance Comparison Between Sparse Matrix Storage Formats that rise in Reservoir Simulation AIM Method* . 2016. 41 f. Dissertação (Mestrado em Engenharia Mecânica) – Faculdade de Engenharia
, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2016.

In this work we aim to compare two sparse matrix storage formats, ievaluating the performance of isparse matrix dense vector multiplication (spMv). The kind of matrices presented in this paper comes from numerical simulations of oil reservoirs, specifically on AIM (Adaptive Implicit Method) aproach. In one format, as a previous step, the mesh of the problem is reordered, so the matrix is divided in four block submatrices, with all blocks of the same size. The other format uses the natural order for the mesh, which results in a block matrix withs blocks of different sizes. It is shown that the second format has little loss of performance when compared to the first one, but the first exhibit problems to improve the utilization of iterative linear solvers through application of preconditioners. The profiling software VTune had been utilized to identify the behavior and hotspots of each format.

Keywords: Sparse matrix; Matrix vector multiplication; Reservoir Simulation; Finite diferences.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

INTRODUÇÃO

No processo de extração de petróleo, uma etapa importante é o desenvolvimento da produção de um reservatório. Nesta etapa são definidos alguns parâmetros de produção, tais como quantidade de poços produtores, quantidade de poços injetores, pressão no fundo poço, etc. Estes parâmetros visam otimizar a produtividade de determinado campo, levando em consideração uma série de restrições, como por exemplo o preço do barril de petróleo, a cotação do dólar, o custo para a produção, taxa de recuperação de óleo, comportamento da reserva de óleo ao longo do tempo de produção, etc. O objetivo final da otimização é conseguir uma rentabilidade máxima para um determinado campo.

Nesta etapa, o engenheiro de petróleo lança mão de uma série de técnicas para determinar como o campo deve produzir. Uma das mais importantes é a simulação do reservatório. Nela, através de modelos físicos e matemáticos, é possível fazer simulações numéricas de como será a produção de óleo de um determinado reservatório ao longo do tempo.

A simulação de reservatório de petróleo pode ser considerada um problema de escoamento de fluidos multifásicos em meios porosos. Isto porque um reservatório de petróleo é basicamente um conjunto de rochas porosas impregnadas com petróleo, apresentando, na maioria dos casos, camadas de gás, hidrocarbonetos e água em altas pressões. Uma das classificações usadas para os simuladores é modelo matemático usado, podendo ser do tipo composicional, *black-oil* ou térmico.

O modelo composicional é considerado o caso mais geral, onde cada uma das três fases pode ter N espécies químicas ou componentes e é necessário avaliar a transferência de massa dessas componentes entre as fases. Este modelo tem como vantagem uma representação melhor do que ocorre no reservatório, porém o custo computacional é alto, sendo utilizado normalmente em reservatórios com óleo de alta volatilidade.

Num sistema de óleo de baixa volatilidade, basicamente composto por metano e componentes pesados, é possível utilizar um modelo mais simples, de apenas duas componentes, para descrever os hidrocarbonetos. Neste modelo simplificado, chamado de *black-oil*, assume-se que não existe transferência de massa entre a fase água e as outras duas fases. No sistema de hidrocarbonetos são consideradas apenas duas componentes: o óleo (conhecido como *stock-tank oil*) sendo a parte líquida residual após um teste de vaporização diferencial na pressão atmosférica, e o gás sendo o fluido restante (PEACEMAN, 2000).

O modelo numérico para representar o escoamento dos fluidos de um reservatório de petróleo leva a um sistema de equações diferenciais parciais não-lineares. Estas equações representam a conservação de massa de cada componente. Essas equações podem ser discretizadas através de diferenças finitas ou volumes finitos(PEACEMAN, 2000), dando

origem a um sistema não linear de equações algébricas, que por sua vez pode ser resolvido por um método de Newton. Métodos de Newton precisam resolver um sistema linear a cada passo de tempo. Os sistemas lineares esparsos que surgem podem ser tratados através de métodos diretos (DAVIS, 2006) ou iterativos (SAAD, 2003). Como o maior gasto de tempo na aplicação do método de Newton, em geral, pode estar na resolução dos sistemas lineares, um formato eficiente para multiplicação da matriz por vetores e a sua armazenagem são fundamentais para uma implementação viável da simulação numérica de reservatórios.

Em relação à discretização do tempo, o método de diferenças finitas, por exemplo, pode ser classificado como explícito ou implícito, dependendo de como é calculado o estado do sistema no tempo t_{n+1}. No método explícito o estado do sistema no tempo t_{n+1} é calculado em função do seu estado no tempo t_n, ou seja, não há incógnitas, todos os valores já são conhecidos de um tempo anterior. No método implícito o tempo t_{n+1} é resultado de uma equação envolvendo tanto t_n, quanto t_{n+1}. No caso da simulação de reservatório, estes métodos dão origem a três maneiras de discretização no tempo: o método IMPES (*Implicit Pressure Explicit Saturation*), o método FIM (*Fully Implicit Method*) e o método AIM(*Adaptive Implicit Method*).

No método IMPES somente a pressão do fluido é calculada implicitamente, levando a sistemas com somente uma incógnita por célula de malha. Utilizando-se a malha de exemplo da Figura 1, neste método as matrizes jacobianas apresentam um padrão de esparsidade como o mostrado na Figura 2:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Figura 1 - Malha bidimensional com todas células IMPES.

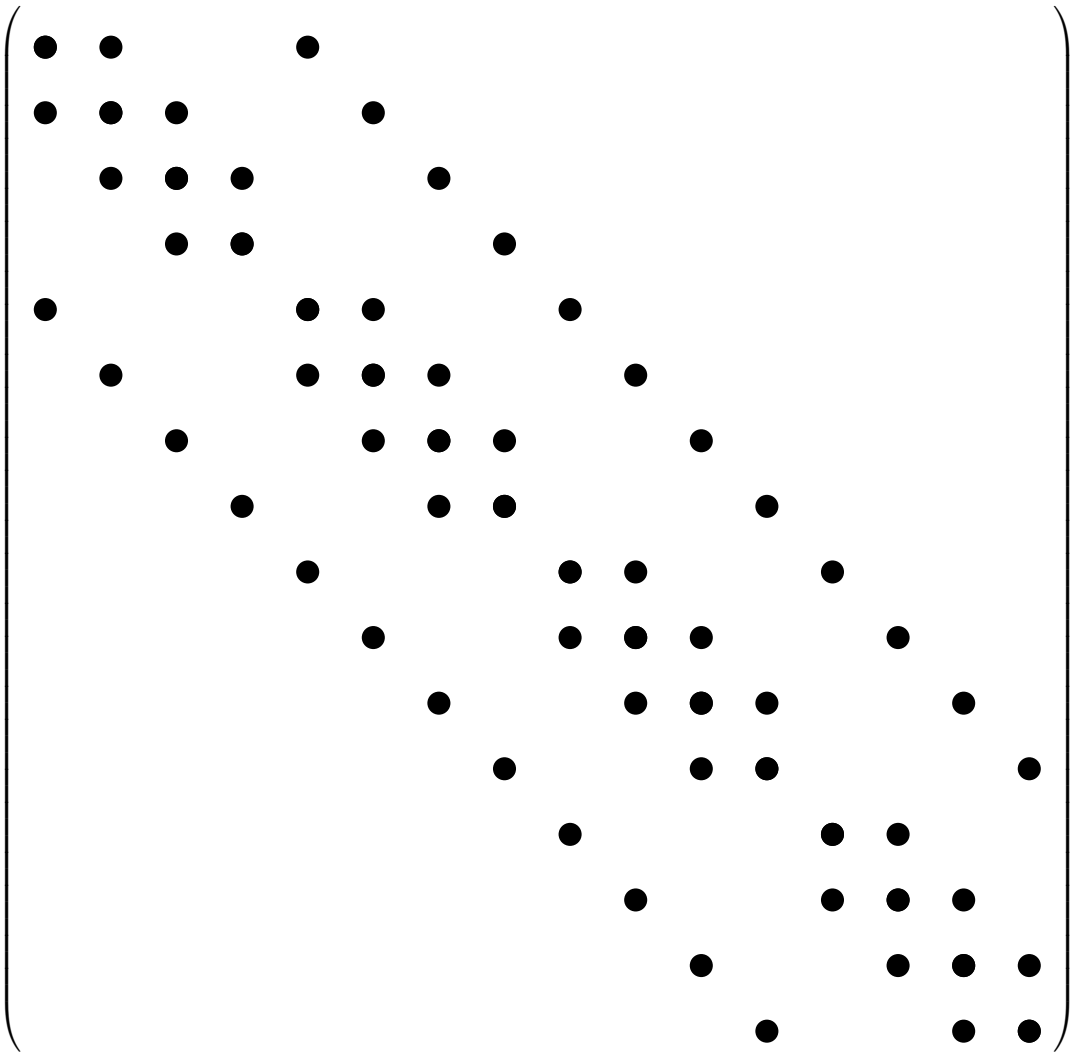


Figura 2 - Matriz associada à malha bidimensional com todas células IMPES.

No método FIM, todas as variáveis são calculadas implicitamente, ou seja, existem tantas incógnitas por célula da malha quanto o número de componentes do modelo. A Figura 4 mostra o padrão de esparsidade da malha da Figura 3, onde todas as células são implícitas e o modelo utilizado é o BlackOil, ou seja, existem três incógnitas.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Figura 3 - Malha bidimensional com todas células FIM.

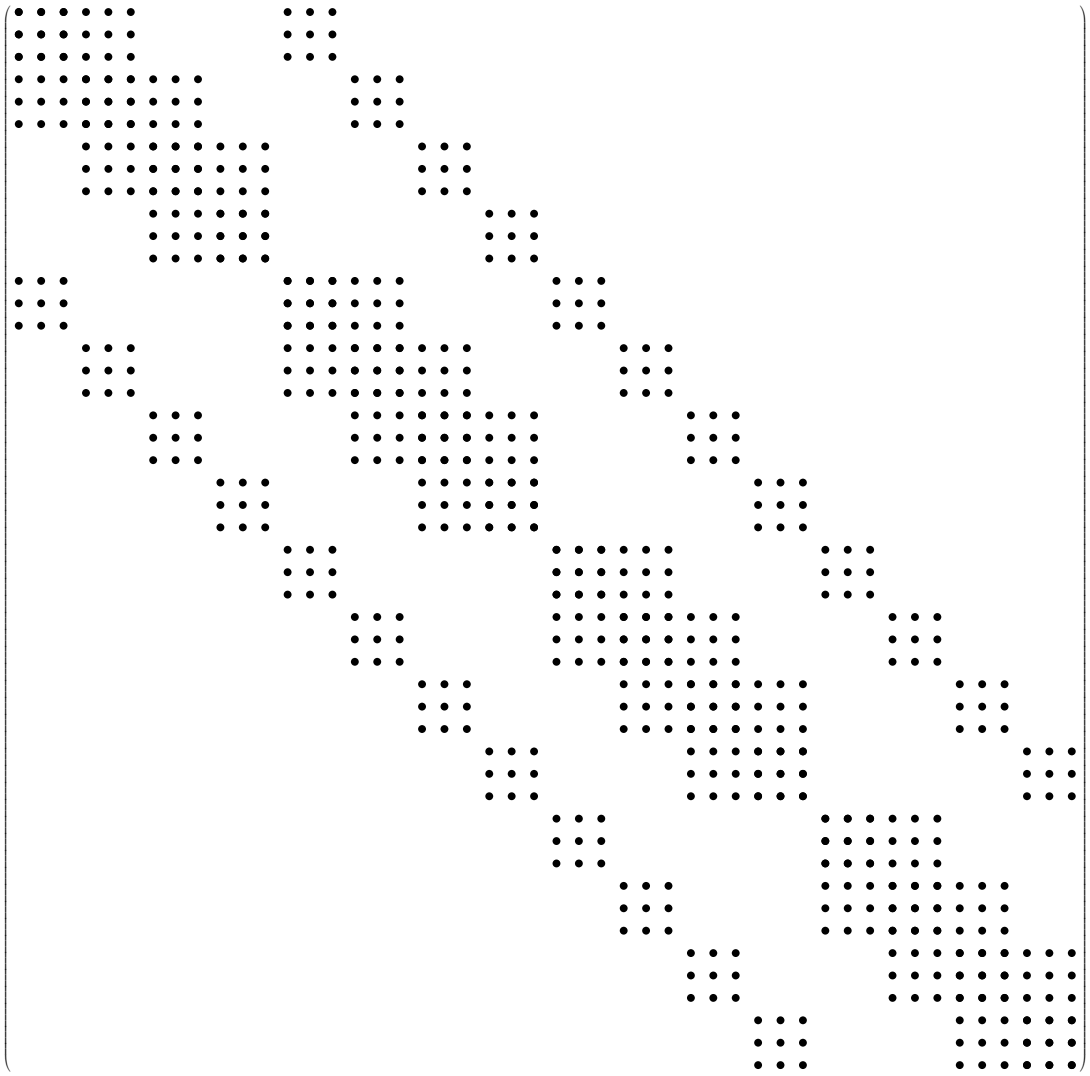


Figura 4 - Matriz associada à malha bidimensional com todas células FIM.

Finalmente, no método AIM, a cada iteração newtoniana, somente algumas células são calculadas de forma implícita. O padrão de esparsidade das matrizes geradas pelo método AIM será visto no capítulo Matrizes AIM.

O presente trabalho mostra uma comparação entre dois formatos de armazenamento de matrizes esparsas, quanto ao desempenho na multiplicação matriz vetor (spMv). As matrizes de interesse surgem a partir da aplicação do método AIM, que gera matrizes esparsas em bloco, com blocos de tamanhos variáveis.

Os formatos mostrados têm duas premissas distintas. Na primeira, é feita a reordenação da matriz, deixando os blocos de mesmo tamanhos agrupados, possibilitando assim o armazenamento em quatro estruturas com tamanho de blocos homogêneos. Na segunda, não há reordenação, mantendo a ordem natural da matriz, que apesar do aumento da complexidade para a multiplicação, possibilita outros artifícios para acelerar a resolução do problema.

O trabalho está dividido em 4 capítulos. No primeiro apresentamos dois formatos tradicionais para armazenamento de matrizes esparsas. No segundo capítulo discutimos o

formato HBCRS, proposto como um formato mais adequado que o BCRS para utilização em simulação de reservatórios, mais especificamente quando do uso do método AIM. No terceiro capítulo são mostrados os resultados de experimentos realizados com diversas matrizes, tanto de campos reais, quanto de modelos sintéticos, e é feita uma análise do seu comportamento no software de perfilagem de programas VTune, da Intel. Por fim, o último capítulo faz algumas considerações finais do trabalho e sugestões para trabalhos futuros.

1 MATRIZES ESPARSAS

A discretização de problemas envolvendo equações diferenciais parciais leva, usualmente, a matrizes esparsas. Uma matriz esparsa tem poucos elementos não nulos e essa estrutura é levada em consideração nas operações com essa matriz (SAAD, 2003).

Sendo assim, pode-se aproveitar o fato de se conhecer as posições dos zeros da matriz, armazenando e computando somente os resultados com os elementos não-nulos. Este tipo de armazenamento pode economizar não somente o espaço de armazenagem, como também a quantidade de operações na multiplicação da matriz por um vetor. Esta economia possibilita a resolução de problemas com *solvers esparsos* e que não seriam possíveis utilizando-se *solvers densos*.

A seguir serão apresentados dois formatos de armazenamento de matrizes esparsas. Estes formatos estão entre os mais utilizados para as matrizes provenientes da discretização de equações diferenciais parciais (SAAD, 2003).

1.1 Formato CSR

O formato CSR (Compressed Sparse Row) utiliza 3 vetores para representar os elementos da matriz:

values contém os valores dos elementos não nulos;

columns contém os índices das colunas onde os elementos não-nulos estão em cada linha;

ptx indica em qual índice do vetor *values* começa cada linha e em qual índice termina a última linha.

Como exemplo, a representação da matriz

$$\begin{pmatrix} 1. & 2. & & \\ & 3. & 4. & \\ 5. & & 6. & 7. \\ & & 8. & 9. \end{pmatrix}$$

fica como se segue, no formato CSR:

$$\begin{aligned} \text{values} &= [1., 2., 3., 4., 5., 6., 7., 8., 9.] \\ \text{columns} &= [1, 2, 2, 3, 1, 3, 4, 3, 4] \\ \text{ptx} &= [1, 3, 5, 8, 10] \end{aligned}$$

1.1.1 Multiplicação Matriz Vetor - CSR

Para se executar a multiplicação de uma matriz armazenada no formato CSR por um vetor denso representado por x e armazenar o resultado no vetor y, pode ser usado o algoritmo (**os arrays abaixo, têm seus índices começando por 1**):

Algoritmo 1 - spMv utilizando CSR

```
1 for (i = 1; i < size(ptx); i++){
      for (j = ptx[i]; j < ptx[i+1]; j++){
3         y[i] += values[j] * x[columns[j]];
      }
5 }
```

1.2 **Formato BCSR**

Se a matriz esparsa tiver estrutura de blocos é possível utilizar-se o formato BCSR (Blocked Compressed Sparse Row). Neste tipo de armazenamento, primeiramente a matriz é subdividida em blocos densos de mesmo tamanho e estes blocos são tratados como elementos individuais da matriz. Os blocos não são necessariamente quadrados, podendo ser blocos retangulares também. Como exemplo, considere as seguintes matrizes, uma contendo blocos quadrados e outra com blocos retangulares:

$$\begin{pmatrix} 1. & 2. & 5. & 6. & & \\ 3. & 4. & 7. & 8. & & \\ & & 9. & 10. & & \\ & & 11. & 12. & & \\ & & 13. & 14. & 17. & 18. \\ & & 15. & 16. & 19. & 20. \end{pmatrix}$$

A matriz acima pode ser divida em blocos 2x2, resultando na representação abaixo.

$$\begin{pmatrix} A & B & & \\ & C & & \\ & D & E & \end{pmatrix}$$

onde,

$$A = \begin{bmatrix} 1. & 2. \\ 3. & 4. \end{bmatrix} \quad B = \begin{bmatrix} 5. & 6. \\ 7. & 8. \end{bmatrix} \quad C = \begin{bmatrix} 9. & 10. \\ 11. & 12. \end{bmatrix} \quad D = \begin{bmatrix} 13. & 14. \\ 15. & 16. \end{bmatrix} \quad E = \begin{bmatrix} 17. & 18. \\ 19. & 20. \end{bmatrix}$$

$$\begin{pmatrix} 1. & 4. & & 7. & & \\ 2. & 5. & & 8. & & \\ 3. & 6. & & 9. & & \\ & & 10. & 13. & 16. & \\ & & 11. & 14. & 17. & \\ & & 12. & 15. & 18. & \end{pmatrix}$$

A matriz acima pode ser divida em blocos 3x1, resultando na representação abaixo.

$$\begin{pmatrix} A & B & & C & & \\ & & D & E & F & \end{pmatrix}$$

Onde,

$$A = \begin{bmatrix} 1. \\ 2. \\ 3. \end{bmatrix} \quad B = \begin{bmatrix} 4. \\ 5. \\ 6. \end{bmatrix} \quad C = \begin{bmatrix} 7. \\ 8. \\ 9. \end{bmatrix} \quad D = \begin{bmatrix} 10. \\ 11. \\ 12. \end{bmatrix} \quad E = \begin{bmatrix} 13. \\ 14. \\ 15. \end{bmatrix} \quad F = \begin{bmatrix} 16. \\ 17. \\ 18. \end{bmatrix}$$

A representação do formato BCSR é feita de forma similar ao CSR, com a adição de mais dois números inteiros que indicam o número de linhas e colunas do bloco.

Assim, a representação da primeira matriz no formato BCSR, fica:

$$\begin{aligned} values &= [1., 2., 3., 4., 5., 6., 7., 8., 9., 10., 11., 12., 13., 14., 15., 16., 17., 18., 19., 20.] \\ columns &= [1, 2, 2, 2, 3] \\ ptx &= [1, 3, 4, 6] \\ blki &= 2 \\ blkj &= 2 \end{aligned}$$

E a representação da segunda matriz no formato BCSR, fica:

$$\begin{aligned} values &= [1., 2., 3., 4., 5., 6., 7., 8., 9., 10., 11., 12., 13., 14., 15., 16., 17., 18.] \\ columns &= [1, 2, 4, 3, 4, 6] \\ ptx &= [1, 4, 7] \\ blki &= 3 \\ blkj &= 1 \end{aligned}$$

1.2.1 Multiplicação Matriz Vetor - BCSR

A multiplicação de uma matriz no formaro BCSR por um vetor denso x e tendo seu resultado armazenado em um vetor y. pode ser representada pelo algoritmo abaixo (**os arrays abaixo, têm seus índices começando por 1**):

Algoritmo 2 - spMv BCRS

```
for (i = 1; i < size(ptx); i++){
  lineshift = (i - 1) * blki;
  for (j = ptx[i]; j < ptx[i+1]; j++){
    columnshift = (column[j] - 1) * blkj;
    for (lineblk = 1 ; lineblk <= blki; lineblk++){
      for (columnblk = 1; columnblk <= blkj; columnblk++){
        y[lineblk + lineshift] =+ \
        values[blki * blkj * j + ((lineblk - 1) * blkj) + columnblk] * \
        x[columnblk + columnshift];
      }
    }
  }
}
```

2 MATRIZES AIM

As matrizes geradas pelo método AIM possuem blocos de dimensões distintas. Os elementos das células IMPES são escalares e os elementos das células FIM são representados por um bloco NxN, onde N é a quantidade de variáveis implícitas no problema. As conexões entre células IMPES e células FIM, são representadas por blocos Nx1 ou blocos 1xN.

Tais matrizes são quadradas, possuem blocos quadrados, NxN ou 1x1, no bloco diagonal e podem ter até quatro formatos de bloco no restante de sua estrutura: 1x1, NxN, 1xN e Nx1.

2.1 Formato com Blocos Homogêneos

Uma forma de trabalhar com este tipo de matriz é reordenar a malha do problema. Pode-se considerar as células totalmente implícitas primeiro, para só então numerar as células IMPES.

Desta forma a matriz gerada é composta por quatro submatrizes, sendo cada uma delas com blocos de dimensões homogêneas.

A Figura 5 mostra como fica a malha após a reordenação e a Figura 6 ilustra o padrão de esparsidade da matriz AIM.

1	2	5	6
3	7	8	9
10	11	12	13
14	15	16	4

Figura 5 - Malha reordenada para geração de matriz com blocos homogêneos.

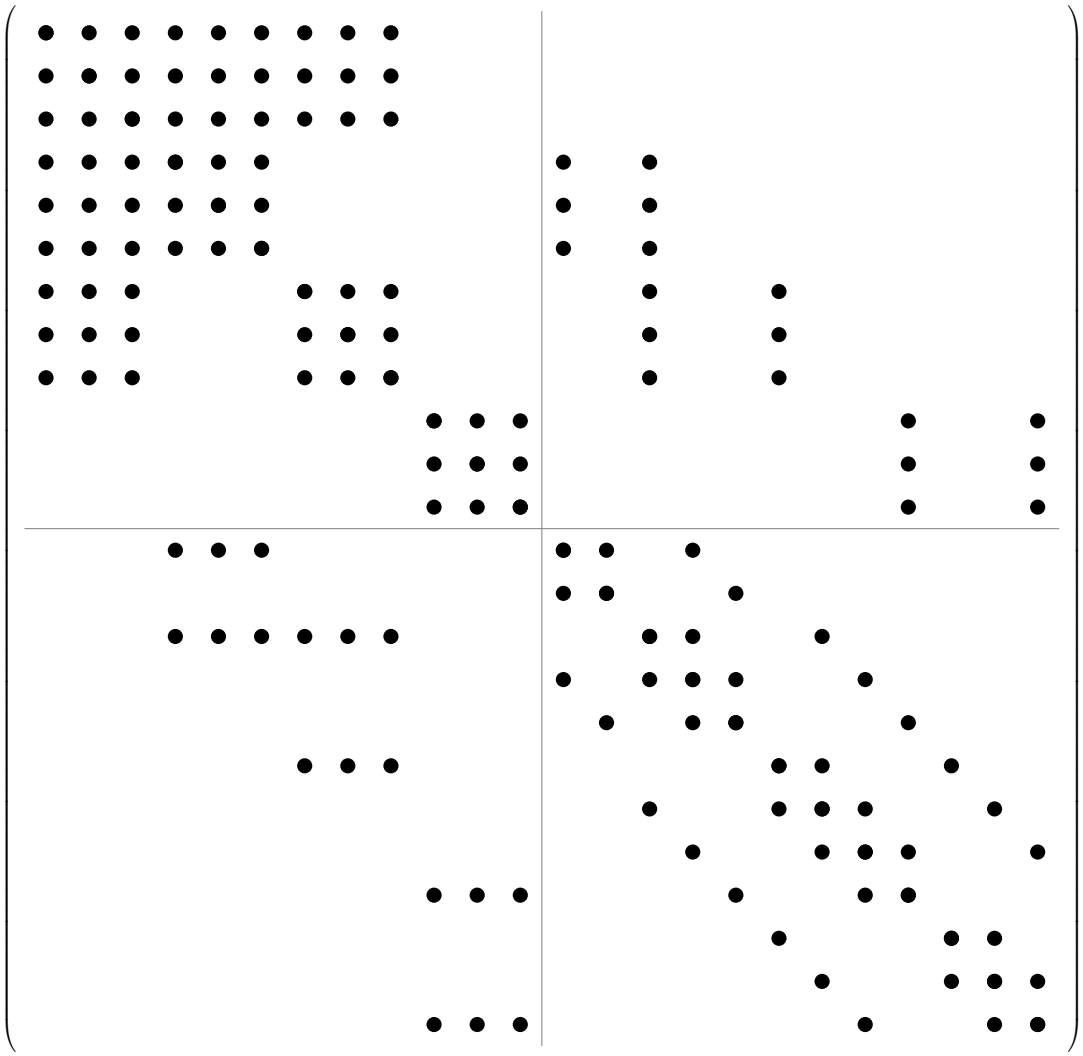


Figura 6 - Padrão de esparsidade da matriz gerada pela malha reordenada.

A implementação da matriz da Figura 6 é feita utilizando-se o paradigma da orientação a objetos (TIMOTHY, 2008). É possível definir-se uma classe, por exemplo MatrizAim, que tem como atributos quatro instâncias de outra classe (StorageBCSR, por exemplo), que conterão as submatrizes que formam a matriz principal.

Assim, a multiplicação spMv poderá ser feita submatriz por submatriz utilizando-se a multiplicação descrita na Seção 1.2.1 para o formato BCSR.

2.2 **Formato HBCSR**

Se nenhuma reordenação for feita na malha do problema, o padrão de esparsidade da matriz laplaciana resultante será como o mostrado na Figura 7.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

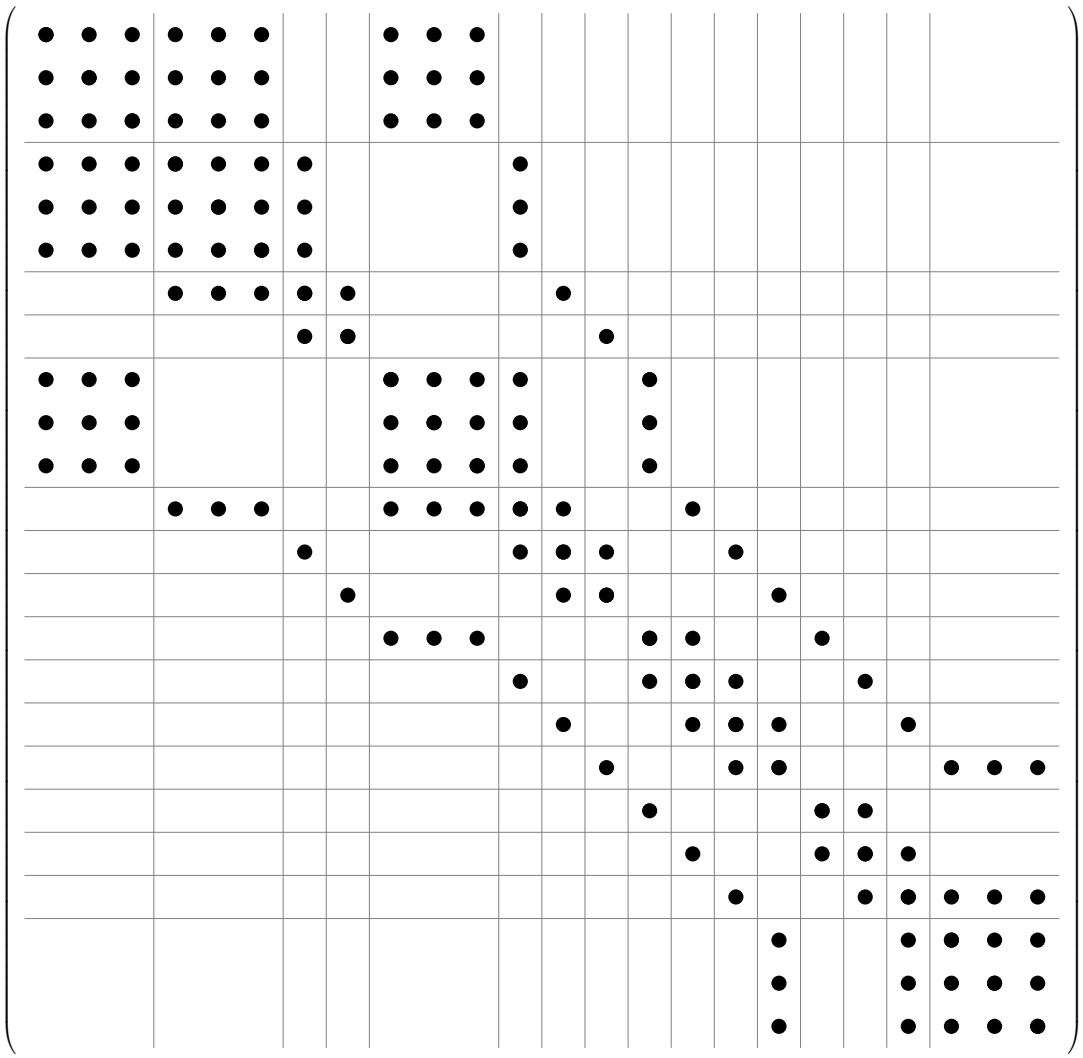


Figura 7 - Padrão de esparsidade da matriz da malha não-reordenada

Para o armazenamento da matriz com blocos híbridos, além dos vetores já utilizados no formato BCSR, é necessário a utilização de mais um array de inteiros.

columnsUnblk armazena os índices iniciais de cada coluna blocada da matriz;

A matriz da Figura 7 ficará representada da seguinte forma:

$$
\begin{aligned}
values &= [\bullet, \bullet, \bullet, \bullet, \bullet, \ldots] \\
columns &= [1, 2, 5, 1, 2, 3, 6, 2, 3, 4, 7, 3, 4, 8, 1, 5, 6, 9, 2, 5, 6, 7, 10, 3, 6, 7, 8, 11, \\
&\quad 4, 7, 8, 12, 5, 9, 10, 13, 6, 9, 10, 11, 14, 7, 10, 11, 12, 15, 8, 11, 12, 16, \\
&\quad 9, 13, 14, 10, 13, 14, 15, 11, 14, 15, 16, 12, 15, 16] \\
ptx &= [1, 4, 8, 12, 15, 19, 24, 29, 33, 37, 42, 47, 51, 54, 58, 62, 65] \\
columnsUnblk &= [1, 4, 7, 8, 9, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 25]
\end{aligned}
$$

O algoritmo para a multiplicação de uma matriz no formato HBCSR por um vetor denso, representado pelo vetor x, com o resultado sendo armazenado no vetor y, pode ser definido pelo trecho de código apresentado em algoritmo 3:

Algoritmo 3 - spMv utilizando HBCSR

```
int valuesShift = 0;
for (int i = 1; i < size[ptx]; i++){
    int neqi = columnsUnblk[i+1] - columnsUnblk[i];
    for (int j = ptx[i]; j < ptx[i+1]; j++){
        int neqj = columnsUnblk[columns[j+1]] - columnsUnblk[columns[j]]
        for (int lnShift = 0; lnShift < neqi; lnShift++){
            for (int colShift = 0; colShift < neqj; colShift++){
                y[columnsUnblk[columns[i]] + lnShift] =+ \
                values[valuesShift + lnShift * neqj + colShift] * \
                x[columnsUnblk[columns[j]] + colShift];
            }
        }
        valuesShift += neqi * neqj;
    }
}
```

Com o intuito de diminuir o número de indireções e também separar a multiplicação de acordo com o formato dos blocos presentes em cada linha da matriz, é possível acrescentar mais quatro arrays e mudar um pouco o algoritmo de multiplicação. Os arrays extras são:

implMap amazena quais blocos da diagonal principal tem dimensãosão 1x1 e quais tem dimensão NxN, ou seja, a largura de cada linha/coluna blocada da matriz. Este vetor pode ser considerado como um mapa de implicitude das células da grade relacionada à matriz e por isso é chamado de *implMap*. A posição com índice n recebe o valor 1 se o bloco da posição n do bloco diagonal for NxN e recebe o valor 0 se for 1x1;

groupIdx amazena os índices das linhas que contem blocos com as mesmas dimensões, de forma agrupada;

groupType armazena qual o tipo e dimensão dos blocos contido em cada grupo de linhas do vetor *groupIdx*. Os valores contidos neste vetor podem ser *ROW_GROUP_SINGLE*

(representado pelo inteiro 1) quando todos os blocos da linha são quadrados 1x1, *ROW_GROUP_BLOCK* (representado pelo inteiro 2) quando todos os blocos são quadrados NxN e *ROW_GROUP_VARIABLE* (representado pelo inteiro 3) quando os blocos possuem dimensões diferentes.

ptxAcc contém as posições de cada bloco no vetor *values*. Este pode ser visto como um vetor que contém os índices dos blocos da matriz de forma acumulada, e por isso neste trabalho é chamado de *ptxAcc*.

A matriz da Figura 7 ficará representada da seguinte forma:

$$
\begin{aligned}
values &= [\bullet, \bullet, \bullet, \bullet, \bullet, \ldots] \\
columns &= [1, 2, 5, 1, 2, 3, 6, 2, 3, 4, 7, 3, 4, 8, 1, 5, 6, 9, 2, 5, 6, 7, 10, 3, 6, 7, 8, 11, \\
&\quad 4, 7, 8, 12, 5, 9, 10, 13, 6, 9, 10, 11, 14, 7, 10, 11, 12, 15, 8, 11, 12, 16, \\
&\quad 9, 13, 14, 10, 13, 14, 15, 11, 14, 15, 16, 12, 15, 16] \\
ptx &= [1, 4, 8, 12, 15, 19, 24, 29, 33, 37, 42, 47, 51, 54, 58, 62, 65] \\
blocksize &= 3 \\
columnsUnblk &= [1, 4, 7, 8, 9, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 25] \\
ptxAcc &= [1, 10, 19, 28, 37, 46, 49, 52, 55, 56, 57, 58, 59, 60, 61, 70, 79, 82, 85, 88, \\
&\quad 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 106, 107, 108, 109, \\
&\quad 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 125, 126, \\
&\quad 127, 128, 129, 130, 131, 132, 133, 134, 135, 138, 141, 144, 153] \\
implMap &= [1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1] \\
groupIdx &= [1, 2, 4, 5, 7, 9, 10, 12, 13, 15, 17] \\
groupType &= [1, 3, 2, 3, 2, 3, 2, 3, 2, 3]
\end{aligned}
$$

Quando uma nova matriz é instanciada os vetores *groupIdx* e *groupType* são pre-enchidos automaticamente. Para isso, são utilizados os vetores columns, ptx e *implMap*. O trecho de código que faz essas operações é apresentado no algoritmo 4.

O algoritmo para a multiplicação de uma matriz no formato HBCSR por um vetor denso, representado pelo vetor x, com o resultado sendo armazenado no vetor y, pode ser definido pelo trecho de código apresentado em algoritmo 5:

2.3 Vantagens e Desvantagens

A grande vantagem da utilização de matrizes AIM no formato com blocos híbridos é a maior eficiência na aplicação de pré-condicionadores, como por exemplo, o Imcomplete LU (ILU). Isto acontece devido ao reordenamento prévio da matriz, necessário para o armazenamento BCSR. Em (RODRIGUES, 2015), é mostrado que há uma diminuição

Algoritmo 4 - Preenchimento automatico de group_idx e group_type

```
int previousGroupType = ROW_GROUP_UNDEFINED;
int nLines = size(implMap);
int nRowGroups = 1;
for (int i = 1; i < nLines; i++){
  int impl_i = implMap[i];
  bool constBlockSize = true;

  for (int idx = ptx[i]; idx < ptx[i+1]; idx++){
    int j = columns[idx];
    int impl_j = implMap[j];

    if (impl_i != impl_j){
      constBlockSize = false;
      break;
    }
  }

  if (constBlockSize == true){
    if (impl_i == 1)
      currentGroupType = ROW_GROUP_BLOCK;
    else
      currentGroupType = ROW_GROUP_SINGLE;
  }
  else
    currentGroupType = ROW_GROUP_VARIABLE;

  if (currentGroupType != previousGroupType){
    groupType[nRowGroups] = currentGroupType;
    nRowGroups++;
    groupIdx[nRowgroups] = groupIdx[nRowgroups-1] + 1;
  }
  else{
    groupIdx[nRowGroups]++;
  }
  previousGroupType = currentGroupType;
}

```

significativa do número de iterações no método de gradientes conjugados quando a matriz com blocos híbridos é utilizada, assim como há também um aumento do número de elementos não nulos quando é utilizada a matriz com blocos homogêneos.

Como desvantagem da utilização do formato com blocos híbridos, temos um uso maior de memória, para que possam ser armazenados os tamanhos dos blocos de cada elemento da diagonal principal.

É esperado uma multiplicação spMv mais rápida no formato com blocos homogêneos, uma vez que cada submatriz possui blocos de mesma dimensão. Também a paralelização é mais fácil neste tipo de matriz, já que é possível multiplicar cada submatriz de forma independente.

As vantagens e desvantagens de ambos os formatos de armazenamento são resu-

Algoritmo 5 - spMv utilizando HBCSR

```
int nRowGroup = size(groupType);

for (int ig = 1; ig < nRowGroup; ig++){
  if (groupType[ig] == ROW_GROUP_SINGLE){
    for (int i = groupIdx[ig]; i < groupIdx[ig+1]; i++){
      for (int block = ptx[i]; block < ptx[i+1]; block++){
        jidx = columns[block];
        jidxUnblk = columnsUnblk[jidx];
        y[i] =+ values[pxtAcc[block]] * x[jidxUnblk];
      }
    }
  }

  if (groupType[ig] == ROW_GROUP_BLOCK){
    for (int i = groupIdx[ig]; i < groupIdx[ig+1]; i++){
      for (int block = ptx[i]; block < ptx[i+1]; block++){
        j = columns[block];
        jUnblk = columnsUnblk[j];
        for (int lnShift = 0; lnShift < blocksize; lnShift++){
          iIdx = jUnblk + lnShift;
          for (int colShift = 0; colShift < blocksize; colShift++){
            jIdx = colShift + jUnblk;
            valShift = lnShift * blocksize + colShift;
            y[iIdx] =+ values[ptxAcc[block] + valShift] * x[jIdx];
          }
        }
      }
    }
  }

  if (groupType[ig] == ROW_GROUP_VARIABLE){
    for (int i = groupIdx[ig]; i < groupIdx[i+1]; i++){
      neqi = columnsUnblk[i+1] - columnsUnblk[i];
      for (int block = ptx[i]; block < ptx[i+1]; block++){
        j = columns[block];
        jUnblk = columnsUnblk[j];
        neqj = columnsUnblk[j+1] - jUnblk;
        for (int lnShift = 0; lnShift < neqi; lnShift++){
          iIdx = jUnblk + lnShift;
          for (int colShift = 0; colShift < neqj; colShift++){
            jIdx = colShift + jUnblk;
            valShift = lnShift * blocksize + colShift;
            y[iIdx] =+ values[ptxAcc[block] + valShift] * x[jIdx];
          }
        }
      }
    }
  }
}
```

midamente descritas na Tabela 1.

Formato	Vantagens	Desvantagens
Homogêneo	Multiplicação spMv mais rápida	Dificultade para aplicação de pré-condicionadores
	Mais fácil de paralelizar a multiplicação spMv, do que o formato HBCSR	
Híbrido	Maior eficiência na aplicação de pré-condicionadores	Utilização de mais memória para armazenar os indices e tamanhos dos blocos de cada liha da matriz

Tabela 1 - Resumo das vantagens e desvantagens de cada formato.

3 EXPERIMENTOS

As implementações do dois formatos de matrizes AIM discutidos no capítulo anterior foram testados para se verificar qual dos dois é mais rápido para fazer a multiplicação spMv.

As matrizes de testes são construídas da seguinte forma: primeiro um padrão de esparsidade é gerado e em seguida a matriz é preenchida com números aleatórios.

O padrão de esparsidade pode ser gerado por uma função que simula o padrão de uma malha regular em formato de um cubo, ou pode ser extraído de uma matriz utilizada em um caso real. Quando a função é utilizada, é necessário informar os seguintes parâmetros de entrada: a quantidade de células em cada dimensão do cubo, o tamanho dos blocos implícitos e o percentual de células FIM existentes na malha. Ao segundo parâmetro é dado o nome de **implicitude** da malha.

3.1 O Programa de Benchmark

O objetivo do programa de benchmark é gerar um relatório contendo algumas estatísticas do tempo de execução das multiplicações spMv. O relatório é um arquivo separado por vírgula (arquivo csv), que contém as seguintes estatísticas:

- Mediana;
- Média;
- Tempo mínimo;
- Tempo máximo;
- Variância;
- Desvio padrão.

Além das estatíscas do tempo de execução, as seguintes informações também estão incluídas no relatório:

- Nome do arquivo com o *dump* da matriz de teste (somente quando foi utilizada a função laplaciano);
- Número de elementos não nulos da matriz;
- Dimensão da matriz (tamanho da diagonal principal);

- Tamanho dos blocos da matriz;
- Número de amostras para geração das estatísticas de tempo;
- Número de vezes que a multiplicação foi executada para gerar uma amostra (utilizando-se a média das execuções).

O programa executa *Ntrials* vezes o núcleo de multiplicação. Após esta tomada de valores, é calculada a média e o resultado é guardado como uma amostra. Quando é gerado o número de amostras determinado pelo parâmetro *NSamples*, o programa gera as estatísticas utilizadas no relatório.

Um resumo dos parâmetros aceitos, extraído da ajuda do próprio programa, pode ser encontrado no apêndice Manual de Uso do Programa SolverBenchmark.

3.2 Função Laplaciano

A função Laplaciano é responsável por criar uma matriz com o mesmo padrão de esparsidade de uma matriz proveniente de uma discretização em um domínio no formato de um cubo, usando um esquema de diferenças finitas com sete pontos.

Esta malha poderá possuir uma porcentagem de células totalmente implícitas. Estas celulas serão distribuídas uniformemente ao longo da malha.

A função então cria o padrão de esparsidade da matriz e em seguida a preenche com números aleatórios.

A Figura 8 mostra o padrão de esparsidade de uma matriz criada a partir de um cubo 3x3x3 e com 10% de implicitude.

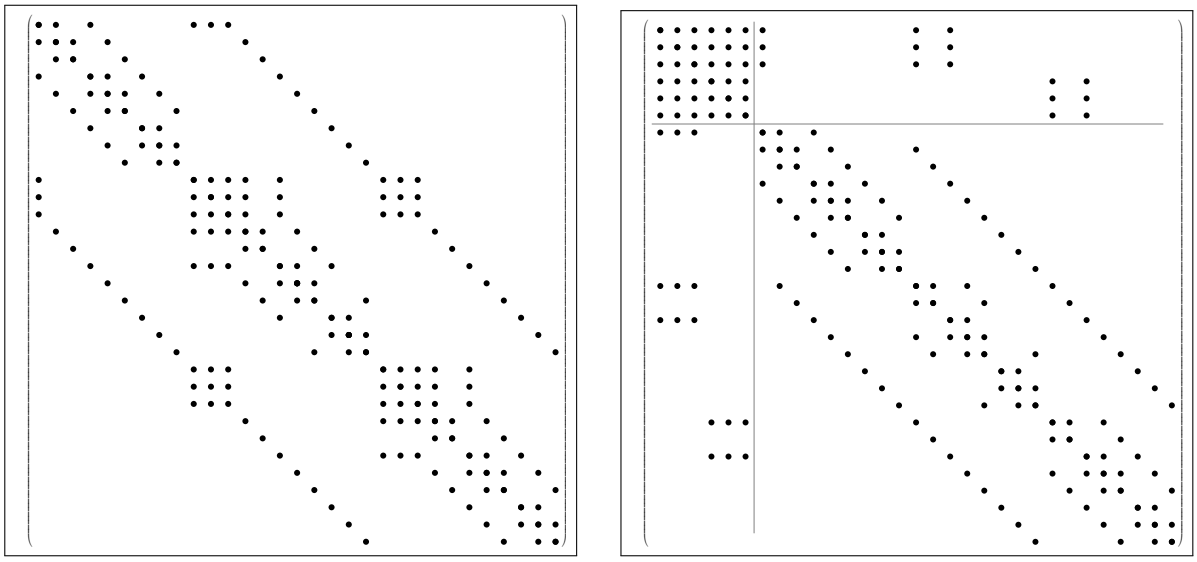
3.3 Dump de Matrizes de Projetos Reais

O programa de benchmark pode tambem ler arquivos no formato de dados da CMG (Computing Modeling Group) (Computer. . . , 2015). Ao ler o dump das matrizes é possível extrair qual é o padrão de esparsidade das mesmas e gerar matrizes com números aleatórios com o mesmo padrão.

Os dumps são feitos de matrizes do jacobiano de alguma iteração newtoniana do solver não-linear da simulação do modelo a ser estudado. O critério para escolha da iteração newtoniana é a quantidade de iterações do solver linear daquela iteração. Foram escolhidas as iterações newtonianas com o maior número de iterações no solver linear.

A tabela 2 mostra as matrizes escolhidas para os testes.

Figura 8 - Padrões de esparsidades das matrizes do Laplaciano de um cubo 3x3x3, com 10% de células implícitas e bloco com tamanho 3.



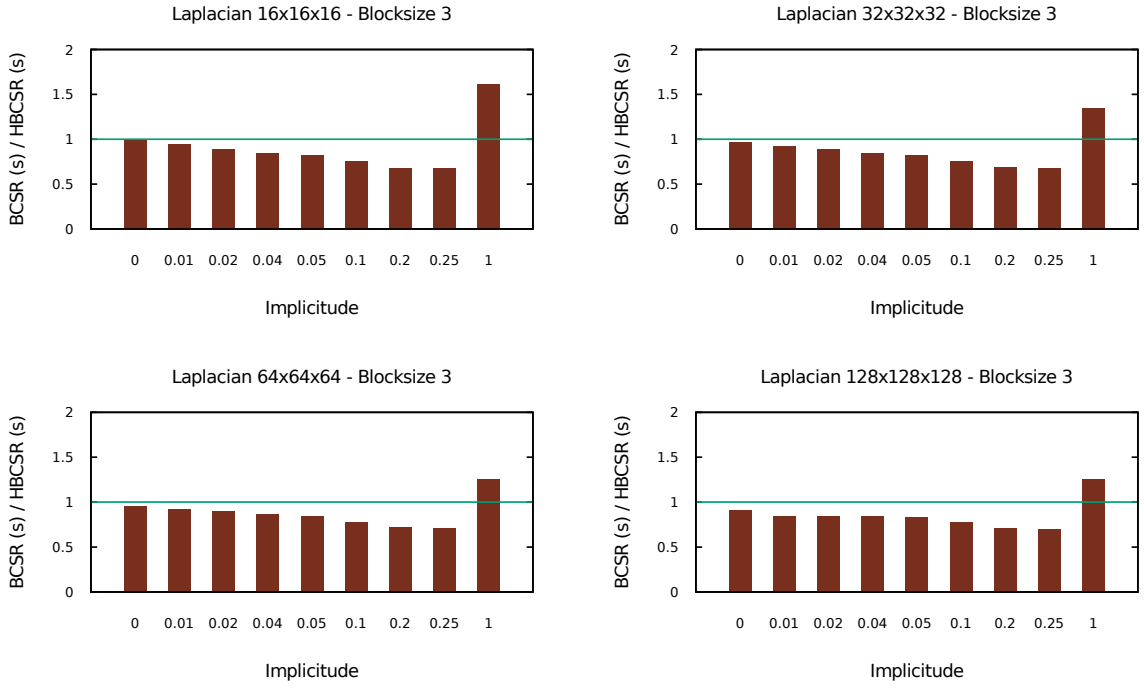
(a) (b)

Legenda: (a) Formato híbrido. (b) Formato homogêneo.

Tabela 2 - Lista das matrizes dos casos reais, com seus respectivos número de elementos não-nulos e tamanho da matriz.

Nome	**NNZ**	**Tamanho**	**Implicitude**
Pituba Natural	184.699	28,107	0.017
Pituba RedBlack	184,699	28,107	0.017
Bijupira	223,698	182,558	0.050
Jubarte sister 2,013	730,128	128,246	0.004
SPE10Model2	17,796,613	1,517,437	0.197

Figura 9 - Resultados para as matrizes geradas pela função laplaciano, com *blocksize* igual a 3



3.4 **Hardware**

O hardware utilizado para os testes foi uma estação HP Z800, com a seguinte configuração:

CPU Intel Xeon X5680 3.33GHz com 12MB de cache

Memória Ram 40GB 1333MHz DIMM

Sistema Operacional GNU/Linux CentOS 6.5

Compilador Intel 2013

3.5 **Resultados**

Abaixo seguem os resultados obtidos com os dois formatos de armazenagem das matrizes. Os primeiros gráficos mostram os resultados obtidos com as matrizes geradas pela função laplaciano. A nomenclatura utilizada para o tamanho do cubo do laplaciano foi a seguinte: para um cubo de $nxnxn$ células, foi dado o nome Ln. O último gráfico mostra a comparação entre os diversos *dumps* de casos reais.

Tabela 3 - Desempenho (GFlops/s) - híbrido - *blocksize* 3

Implicitude	L16	L32	L64	L128
0.01	1.09	1.09	0.94	0.86
0.02	1.00	1.02	0.88	0.83
0.04	0.89	0.91	0.81	0.78
0.05	0.86	0.88	0.78	0.76
0.10	0.77	0.78	0.72	0.72
0.20	0.78	0.80	0.74	0.73
0.25	0.81	0.83	0.77	0.77
1.00	3.23	2.30	2.13	2.11

Tabela 4 - Desempenho (GFlops/s) - homogêneo - *blocksize* 3

Implicitude	L16	L32	L64	L128
0.01	1.16	1.19	1.03	1.03
0.02	1.12	1.15	0.99	0.99
0.04	1.06	1.09	0.94	0.94
0.05	1.05	1.07	0.93	0.92
0.10	1.03	1.05	0.93	0.93
0.20	1.16	1.17	1.04	1.05
0.25	1.22	1.24	1.09	1.11
1.00	2.01	1.72	1.69	1.69

Figura 10 - Resultados para as matrizes geradas pela função laplaciano, com *blocksize* igual a 4

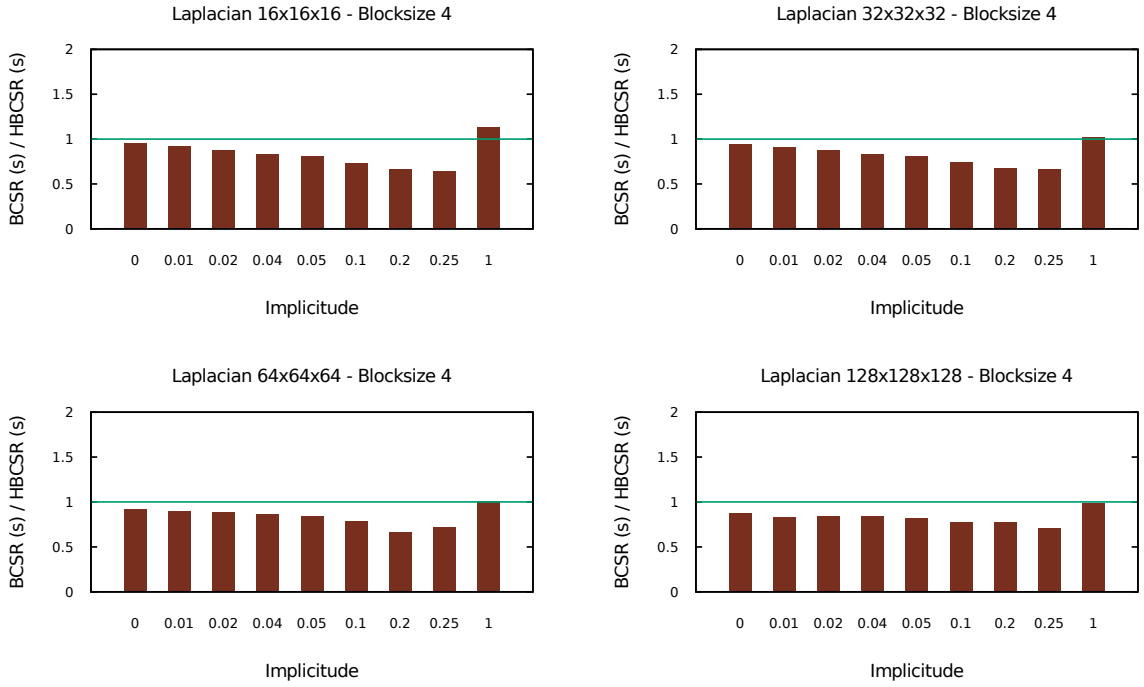


Tabela 5 - Desempenho (GFlops/s) - híbrido - *blocksize* 4

Implicitude	L16	L32	L64	L128
0.01	1.11	1.12	0.96	0.90
0.02	1.05	1.07	0.92	0.88
0.04	0.97	0.99	0.88	0.85
0.05	0.95	0.97	0.86	0.84
0.10	0.91	0.93	0.84	0.82
0.20	1.00	1.02	0.83	0.92
0.25	1.06	1.08	0.98	0.98
1.00	4.25	2.46	2.42	2.33

Tabela 6 - Desempenho (GFlops/s) - homogêneo - *blocksize* 4

Implicitude	L16	L32	L64	L128
0.01	1.21	1.24	1.07	1.08
0.02	1.19	1.22	1.04	1.05
0.04	1.17	1.19	1.02	1.02
0.05	1.17	1.19	1.03	1.03
0.10	1.25	1.25	1.08	1.07
0.20	1.52	1.51	1.26	1.19
0.25	1.65	1.64	1.37	1.40
1.00	3.77	2.41	2.42	2.35

Figura 11 - Resultados para as matrizes geradas pela função laplaciano, com *blocksize* igual a 10

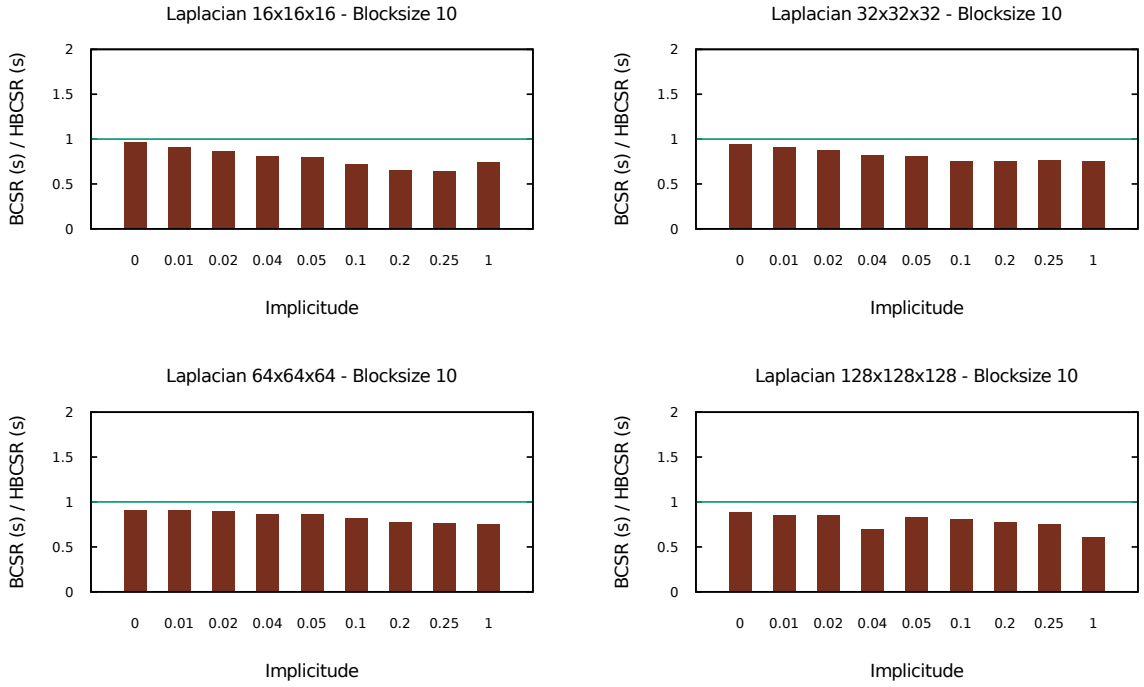


Tabela 7 - Desempenho (GFlops/s) - híbrido - *blocksize* 10

Implicitude	L16	L32	L64	L128
0.01	1.27	1.28	1.07	1.01
0.02	1.31	1.33	1.12	1.07
0.04	1.39	1.41	1.19	0.96
0.05	1.45	1.45	1.23	1.21
0.10	1.64	1.60	1.38	1.37
0.20	1.94	1.63	1.59	1.57
0.25	2.05	1.68	1.66	1.64
1.00	2.10	2.06	2.04	1.62

Tabela 8 - Desempenho (GFlops/s) - homogêneo - *blocksize* 10

Implicitude	L16	L32	L64	L128
0.01	1.39	1.42	1.18	1.18
0.02	1.52	1.54	1.25	1.26
0.04	1.73	1.73	1.38	1.39
0.05	1.82	1.81	1.43	1.45
0.10	2.29	2.15	1.68	1.70
0.20	2.96	2.16	2.04	2.04
0.25	3.18	2.22	2.17	2.17
1.00	2.83	2.76	2.73	2.67

Figura 12 - Resultados para as matrizes geradas pela função laplaciano, com *blocksize* igual a 11

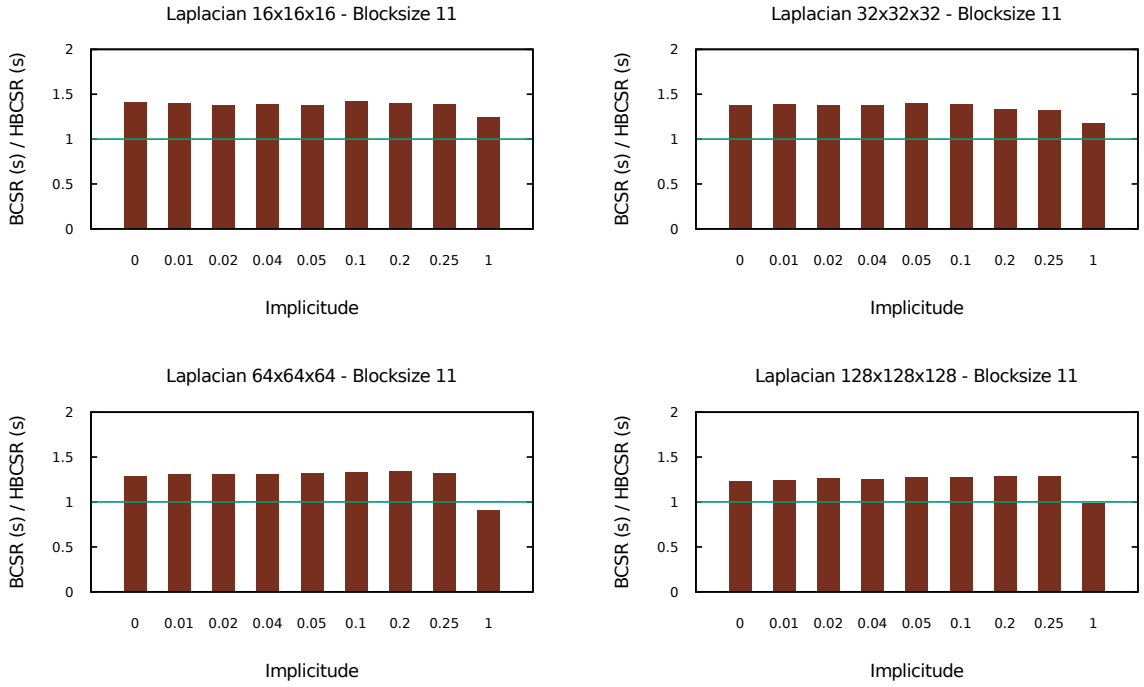


Tabela 9 - Desempenho (GFlops/s) - híbrido - *blocksize* 11

Implicitude	L16	L32	L64	L128
0.01	1.23	1.23	1.04	0.98
0.02	1.25	1.26	1.07	1.03
0.04	1.30	1.31	1.12	1.08
0.05	1.32	1.34	1.16	1.13
0.10	1.49	1.43	1.29	1.25
0.20	1.73	1.52	1.50	1.45
0.25	1.83	1.58	1.58	1.53
1.00	2.17	2.06	1.59	1.71

Tabela 10 - Desempenho (GFlops/s) - homogêneo - *blocksize* 11

Implicitude	L16	L32	L64	L128
0.01	0.88	0.89	0.79	0.79
0.02	0.91	0.92	0.82	0.82
0.04	0.94	0.95	0.86	0.87
0.05	0.96	0.96	0.88	0.89
0.10	1.06	1.03	0.97	0.98
0.20	1.24	1.14	1.12	1.13
0.25	1.32	1.20	1.19	1.19
1.00	1.75	1.75	1.75	1.74

Figura 13 - Resultados para as matrizes dos dumps dos casos reais

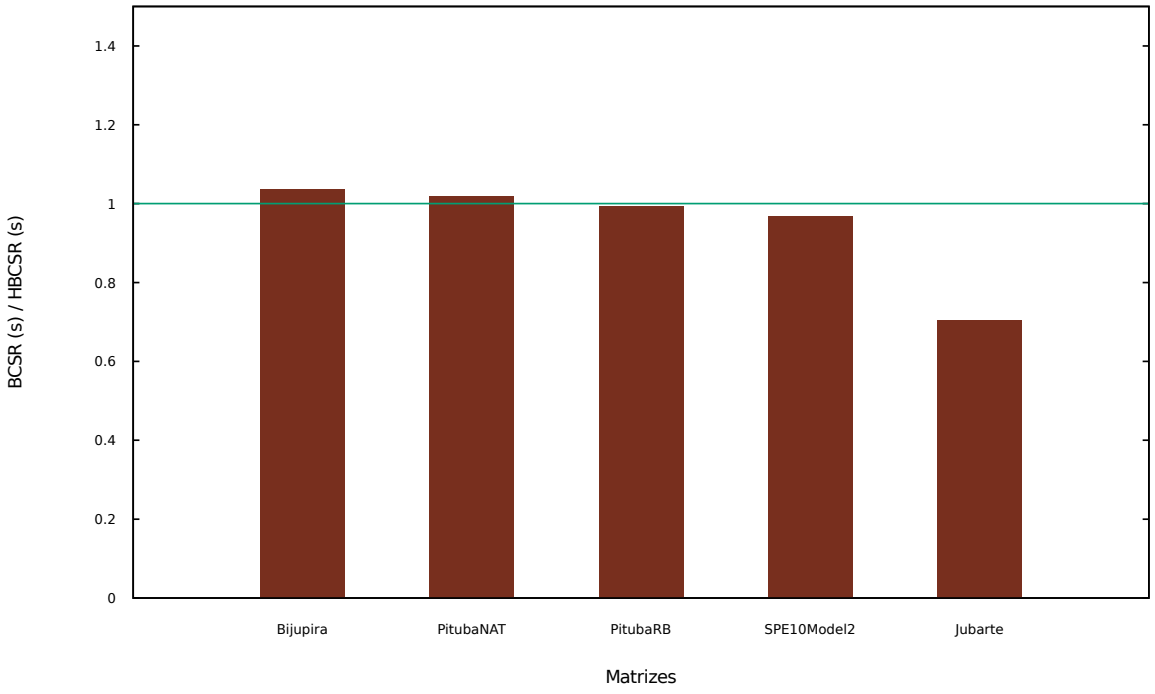


Tabela 11 - Desempenho (GFlops/s) - híbrido - *Campos reais*

Matrizes	GFlops/s
PitubaNAT	1,08
PitubaRB	1,04
SPE10Model2	1,21
Jubarte	0,63

Tabela 12 - Desempenho (GFlops/s) - homogêneo - *Campos reais*

Matrizes	GFlops/s
PitubaNAT	1,07
PitubaRB	1,05
SPE10Model2	1,25
Jubarte	0,90

Tabela 13 - Tabela de resultados no VTune - Bijupira

Métrica	HBCSR	BCSR
Elapsed Time	8.004s	8.037s
CPU_CLK_UNHALTED.THREAD	11,260,000,000	10,952,000,000
INST_RETIRED.ANY	18,186,000,000	20,686,000,000
CPI Rate	0.619	0.529
Retire Stalls	0.371	0.283
LLC Miss	0.090	0.101
LLC Load Misses Serviced By Remote DRAM	0.035	0.000
Instruction Starvation	0.000	0.009
Branch Mispredict	0.053	0.075
Execution Stalls	0.177	0.118
Data Sharing	0.000	0.000
Paused Time	4.832s	4.953s

3.6 **Perfilagem do Loop da Multiplicação**

Foi utilizado o programa Vtune (VTune..., 2014) da Intel para tentar compreender o que ocorre na multiplicação spMv com ambos os formatos de armazenamento das matrizes.

Para gerar as análises da forma mais correta possível, o programa teve que ser modificado. Primeiramente, foi incluido o *header* ittnotify.h. Depois foram inseridos duas funções desse *header* no código fonte, de tal maneira que somente o loop principal da multiplicação fosse levado em consideração na coleta de dados. As funções utilizadas foram: a *itt_resume()* antes de começar o loop e a *itt_pause()* após o laço. Ao iniciar o VTune, foi utilizado um parâmetro para que a análise começasse pausada, ou seja, a coleta de dados só seria iniciada no laço principal e seria suspensa novamente tão logo o loop fosse executado.

A análise utilizada foi a *General Exploration - GE*, que gera as métricas descritas no apêndice Descriçaão das Métricas do VTune.

Da tabela 13 até a tabela 16, são mostrados os resultados das diversas rodadas executadas, cada uma com uma matriz diferente. As métricas que ultrapassaram o valor do gatilho padrão do VTune, estão destacadas. Este gatilho é o valor limite que o VTune considera como um bom desempenho.

É possível observar pelas tabelas que o formato HBCSR é ligeiramente mais lento maioria dos casos. Um dos motivos é uma taxa de *branch mispredict* maior que a taxa do formato BCSR. Este valor mais alto acontece principalmente por causa dos *if* contidos

Tabela 14 - Tabela de resultados no VTune - Jubarte

Métrica	HBCSR	BCSR
Elapsed Time	14.412s	10.533s
CPU_CLK_UNHALTED.THREAD	29,238,000,000	15,720,000,000
INST_RETIRED.ANY	25,390,000,000	24,752,000,000
CPI Rate	1.159	0.635
Retire Stalls	0.685	0.446
LLC Miss	0.210	0.169
LLC Load Misses Serviced By Remote DRAM	0.168	0.000
Instruction Starvation	0.017	0.073
Branch Mispredict	0.060	0.092
Execution Stalls	0.366	0.205
Data Sharing	0.000	0.000
Paused Time	6.146s	6.110s

Tabela 15 - Tabela de resultados no VTune - Pituba

Métrica	HBCSR	BCSR
Elapsed Time	3.201s	3.186s
CPU_CLK_UNHALTED.THREAD	1,544,000,000	1,520,000,000
INST_RETIRED.ANY	2,804,000,000	2,884,000,000
CPI Rate	0.551	0.527
Retire Stalls	0.295	0.253
LLC Miss	0.000	0.000
LLC Load Misses Serviced By Remote DRAM	0.000	0.000
Instruction Starvation	0.109	0.111
Branch Mispredict	0.308	0.156
Execution Stalls	0.062	0.047
Data Sharing	0.000	0.000
Paused Time	2.761s	2.752s

Tabela 16 - Tabela de resultados no VTune - SPE10Model2

Métrica	HBCSR	BCSR
Elapsed Time	53.051s	52.066s
CPU_CLK_UNHALTED.THREAD	118,278,000,000	115,246,000,000
INST_RETIRED.ANY	167,572,000,000	197,506,000,000
CPI Rate	0.706	0.584
Retire Stalls	0.482	0.417
LLC Miss	0.108	0.140
LLC Load Misses Serviced By Remote DRAM	0.000	0.000
Instruction Starvation	0.112	0.092
Branch Mispredict	0.144	0.123
Execution Stalls	0.218	0.161
Data Sharing	0.000	0.000
Paused Time	19.828s	19.682s

no algorítmo de multiplicação do formato HBCSR. Quando a matriz tem poucos blocos do mesmo tipo em sequência na diagonal principal, as opções são trocadas muitas vezes, causando uma falha maior na previsão do próximo conjunto de instruções.

Um fator que influencia mais fortemente no tempo de execução da multiplicação é o *LLC Miss*. Esta taxa aumenta proporcionalmente em relação ao tamanho da matriz, ou seja, quanto maior a matriz, maior é a perda de cache. Este fato acontece porque o programa executa a multiplicação repedidas vezes, no caso destes testes 110 vezes. Caso a matriz não caiba inteira no cache, na próxima iteração do loop, ao tentar acessar as primeiras linhas da matriz novamente, estas não estarão mais lá e o *cache miss* irá acontecer. Outro característica do formato híbrido que contribui para um aumento do *cache miss* é que como existem mais vetores para o controle do tamanho dos blocos sendo multiplicados, existe um número maior de indireções no código, causando um acesso mais intenso à memória. Os casos onde estas características são mais claramente observadas são Jubarte e SPE10Model2.

É notável o fato de que se, por outro lado, a matriz couber no cache e for suficientemente homogênea, os formatos se equivalem, pois haverá pouca ou nenhuma troca de opções no *if* da multiplicação do HBCSR e os dados necessários para a multiplicação dos blocos estarão sempre no cache. Isto pode ser observado com os dados da matriz Pituba.

Apesar de ser mais afetada por *cache misses*, a multiplicação do formato HBCSR leva aproximadamente o mesmo tempo que a do formato BCSR. Isto acontece porque a quantidade de instruções efetivas (*retired instructions*) é menor no formato HBCSR, ou seja, apesar das instruções demorarem mais tempo para serem executadas (*CPI rate* maior) devido a *branch mispredict* e *LLC misses*, como elas são em menores quantidades, os tempos se equivalem.

CONSIDERAÇÕES FINAIS

Os testes com os formatos de armazenamento BCRS e HBCRS mostraram que nos casos reais, com uma distribuição de células implícitas mais aleatória que a distribuição gerada pela função laplaciano, as multiplicações nos dois formatos se equivalem em tempo de execução. O acesso a memória é um pouco mais intenso no caso do formato HBCRS, ocasionando um pouco mais de *LCC miss*. Caso a quantidade de células implícitas aumente e sua distribuição seja pouco concentrada, o formato HBCRS também apresentará uma maior *branch mispredict*, devido a maneira como ele faz a transição entre um tipo de bloco e outro dentro da mesma matriz.

A análise feita com o software VTune, mostrou também que o tamanho da matriz implica no aproveitamento ou não dos dados contidos no cache da CPU, e que a quantidade de *LCC miss* impacta mais no desempenho da aplicação que a quantidade de *branch misredict*.

Um fato importante para a perda de dados no cache é a quantidade de vezes que a multiplicação foi executada. Dado que o tempo de multiplicação matriz vetor se equivale nos dois formatos e dado que em (RODRIGUES, 2015) é estabelecido que a utilização da ordenação não natural da malha AIM , pode aumentar em até 17% o número de iterações dos solvers lineares, a utilização do formato HBCRS tem o potencial para reduzir muito tempo de simulação, dependendo da quantidade de iterações newtonianas que o problema precisar para ser resolvido.

Como próximo passo, é necessário fazer uma investigação de quanto é ganho de tempo real nas simulações utilizando-se o formato HBCRS. Para tanto é necessário implementar um simulador que aceite ambos os formatos e que gere um arquivo de saída com todas as informações dos tempos de processamento em cada fase da simulação. O VTune poderá ser usado novamente para estabelecer gargalos tanto de um quanto de outro formatos.

REFERÊNCIAS

Computer Modeling Group. 2015. Disponível em: <http://www.cmgl.ca>.

CPU Metrics Reference | Intel Software. 2016. Disponível em: <https://software.intel.com/en-us/node/544393>.

DAVIS, T. A. *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2006. (Fundamentals of Algorithms). ISBN 9780898718881. Disponível em: <https://books.google.com.br/books?id=aTLYrafw3vUC>.

PEACEMAN, D. W. *Fundamentals of Numerical Reservoir Simulation*. Elsevier Science, 2000. (Developments in Petroleum Science). ISBN 9780080868608. Disponível em: <https://books.google.com.br/books?id=-DujQRDF4kwC>.

RODRIGUES, J. 2015. Confidential Technical Report.

SAAD, Y. *Iterative Methods for Sparse Linear Systems: Second Edition*. [s.n.], 2003. ISBN 0898715342. Disponível em: <https://books.google.com/books?hl=pt-BR&lr=&id=qtzmkzzqFmcC&pgis=1>.

TIMOTHY, B. *Introduction to Object-Oriented Programming*. [S.l.]: Pearson Education India, 2008.

VTune Amplifier XE 2015. Intel, 2014. Disponível em: <https://software.intel.com/pt-br/intel-vtune-amplifier-xe>.

APÊNDICE A – Manual de Uso do Programa SolverBenchmark

A.1 Manual de Uso do Programa SolverBenchmark

```
Usage: SolverBenchmark [kernels] [params]
kernels: At least one of the above kernels must be used:
    --kernel-cmg-aim-matvecmul
    --kernel-cmg-laplacian-ilusolve
    --kernel-cmg-laplacian-matvecmul
    --kernel-drms-aim-matvecmul
Parameters:
    --pause: Pause waiting a keyboard action before execute the kernels.
    --file-patterns <list-of-file-patterns>: List of file patterns
      (i.e., filenames without extension) separated by comma (without white
      spaces between them). Not used with the Laplacian option.
    --nx <number>: Number of cells in the X direction for the Laplacian option
      (default is 50). Not used with the file option.
    --ny <number>: Number of cells in the Y direction for the Laplacian option
      (default is 50). Not used with the file option.
    --nz <number>: Number of cells in the Z direction for the Laplacian option
      (default is 50). Not used with the file option.
    --block-size <number>: Block size for the Laplacian option (default is 3).
      Not used with the file option.
    --impl-cell-ratio <number>: Ratio of implicit cells for the Laplacian option
      (default is 0.1). Not used with the file option.
    --output-prefix <prefix>: The sufix of the output file name. The name of
      output file will be <prefix><kernel name><sufix>.csv (default prefix is the
      empty string).
    --output-suffix <suffix>: The sufix of the output file name. The name of
      output file will be <prefix><kernel name><sufix>.csv (default suffix is the
      empty string)
    --n-samples <number>: Number of samples to obtain statistics metrics
      (default is 1).
    --n-trials <number>: Number of times that each kernel will be executed,
      in a for loop, inside each sample. The time of one sample will be the
      average of the nTrials executions. (default is 10)
```

APÊNDICE B – Descriçaão das Métricas do VTune

B.1 Descriçaão das Métricas do VTune

Elapsed Time Tempo total que durou a análise.

CPU_CLK_UNHALTED.THREAD Total de *clockticks* que todas as instruções chamadas por uma *thread* utilizaram.

INST_RETIRED.ANY Os processadores modernos executam mais instruções do que o fluxo do programa realmente precisa. Isto é chamado de "execução especulativa". Quando é verificado que uma instrução foi de fato utilizada, esta instrução é chamdada *retired instruction*. A métrica INST_RETIRED.ANY indica a quantidade de *retired instructions* executadas.

CPI (Clockticks per Instructions Retired) Rate É a razão entre CPU_CLK_UNHALTED.T e INST_RETIRED.ANY. Indica a média de *clockticks* utilizados pelas *retired instructions*. Quanto maior esta razão, pior o desempenho da aplicação. Uma CPI Rate ruim pode ser causada por uma alta taxa de *branch misprediction*, *cache miss*, etc.

Retire Stalls É a razão entre a quantidade de ciclos de CPU sem *retired instructions* e a quantidade total de ciclos de CPU. Quanto maior pior. Um número alto indica uma latência muito grande na execução das instruções. Tal latência pode ser causada por *branch mispredict*, *instruction starvation*, *instruções de alta latência*, etc.

LLC (Last Level Cache) Miss LLC é o último nível, e de mais longa latência, da hierarquia de memória antes da memória principal. LLC Miss é a razão entre o número de ciclos com não acerto de dados no LLC e todos os ciclos de CPU.

LLC Load Misses Serviced By Remote DRAM Em uma máquina NUMA (Non-Uniform Memory Access), quando uma requisição de memória não é atendida pelo LLC, ela pode ser servida pelo banco local de memória DRAM ou pela memória DRAM remota. Esta métrica é definida como a razão entre o número de ciclos em que a memória remota serviu dados e todos os ciclos de CPU.

Instruction Starvation É a razão entre o número de ciclos que é perdido esperando alguma instrução ser entregue e o número total de ciclos de CPU. Um número alto aqui pode indicar um excesso de *branch mispredict* ou um conjunto de trabalho do código muito grande. As duas razões podem causar uma perda de dados no cache L1, ocasionando a espera de entrega do código.

Branch Mispredict Quando um ramo de instrução não é corretamente previsto, o seu fluxo de dados continua no pipeline. Isto representa um desperdício de tempo, uma vez que ao final do processo ele não será de fato executado. Esta medida representa a fração de slots de CPU perdidas por causa predições de ramos do programa mal feitas. Para melhorar esta taxa é necessário considerar forma de fazer o programa mais previsível, como por exemplo fazendo if's o mais cedo possível e com a maior quantidade de código possível em seus ramos, caso o comando *case* seja usado, este não deve conter muitas opções e as opçõs mais usadas devem vir primeiro.

Execution Stalls Na maioria das vezes significa que a máquina está rodando com a capacidade máxima, ou seja, não existe mais recursos para rodar aquela instrução. Porém, às vezes operações de alta latência podem serializar enquanto esperam por recursos críticos da máquina. Esta métrica é a razão entre o número de ciclos que a cpu passa sem executar micro-operações e o número total de ciclos de CPU.

Data Sharing Os dados compartilhados entre *threads* podem aumentar a latência de acesso ao LLC devido à coerência de cache. Esta métrica permite avaliar o impacto de manter esta coerência. É definida como a razão entre o número de ciclos lidando com o compartilahmento de dados e o total de ciclos de CPU.

Paused Time Tempo em que a análise ficou pausada.

(CPU..., 2016)