Núcleos Esparsos de Cálculos Intensivos de Álgebra Linear em Problemas de Reservatório

Eduardo D. Corrêa **Luiz Mariano Carvalho**
Departamento de Matemática Aplicada, UERJ,
20550-013, Rio de Janeiro, RJ
E-mail: edudc2000@gmail.com, luizmc@gmail.com.

Resumo: *Descrevemos algumas características de um simulador numérico tridimensional em desenvolvimento para estudar as propriedades físico-químicas da água durante as inundações de represas hidrelétricas. Utilizamos métodos iterativos para resolver os sistemas lineares esparsos que surgem a partir da discretização tridimensional de equações diferenciais parciais. Buscamos otimizar o núcleo de cálculos dos sistemas.*

1 Introdução

O nosso trabalho foi desenvolvido no grupo de pesquisa Gesar (Grupo de estudos e simulações ambientais em reservatórios), sediado na UERJ, que possui como principal objetivo o desenvolvimento de um simulador para análise da qualidade da água durante o enchimento de reservatórios de hidrelétricas. O simulador utiliza como base um sistema não linear de equações diferenciais parciais: equações de Navier-Stokes e equações de transporte de massa, representadas abaixo.

$$\begin{aligned} \rho\left(\frac{\partial \mathbf{v}}{\partial t} + (\nabla \mathbf{v})\mathbf{v}\right) &= -\nabla p + \mu \Delta \mathbf{v} + \rho \mathbf{g} \\ \nabla \cdot \mathbf{v} &= 0 \\ \frac{\partial c}{\partial t} + \mathbf{v} \cdot \nabla c &= \frac{1}{Re \cdot\ Sc} \nabla \cdot D_t \nabla c + S. \end{aligned} \tag{1}$$

Estas equações são responsáveis pela descrição do movimento de um fluido (e de uma massa respectivamente) onde Re representa o número de Reynolds (razão entre as forças inerciais e as forças viscosas do escoamento), $Sc = \nu/D$ é o número de Schmidt (valor que mede a relação entre a espessura da camada limite hidrodinâmica e a camada limite de difusão de massa), μ é a viscosidade dinâmica, $\mathbf{v}$ é o vetor velocidade, g a gravidade, p a pressão, c é a concentração em massa de uma espécie, S é um termo fonte devido a alguma reação química e ρ é uma constante independente de t.

No tratamento das equações (1) utilizamos o método de Galerkin [Quarteroni and Valli, 1994] de maneira a transformar as equações em um sistema de equações diferenciais ordinárias. Método semi-Lagrangiano [Pironneau, 1982] é utilizado para discretização temporal. Daí, nosso sistema toma a seguinte forma matricial discreta:

$$\begin{aligned} M\big(\frac{\mathbf{v}_i^{n+1} - \mathbf{v}_d^n}{\Delta t}\big) + \frac{1}{Re} K \mathbf{v}^{n+1} - G p^{n+1} &= b_1 \\ D\mathbf{v}^{n+1} &= b_2 \\ M_c\big(\frac{c_I^{n+1} - c_d^n}{\Delta t}\big) + \frac{1}{ReSc} K_c c^{n+1} &= b_3. \end{aligned} \tag{2}$$

Nas equações (2), $\mathbf{v}$ e c representam variáveis contínuas e discretas. M, K, G e D são matrizes oriundas da aproximação pelo método de Galerkin.

Nossa simulação modela em um espaço de três dimensões, originado assim um sistema linear de grande porte e esparso. Dessa forma, métodos diretos se tornam proibitivos em virtude do custo computacional. Utilizamos para esta situação métodos iterativos.

As duas primeiras equações de (2) geram um sistema linear do tipo:

$$\begin{bmatrix} B & G \\ D & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{v} \\ p \end{bmatrix} = \begin{bmatrix} b_1^* \\ b_2 \end{bmatrix}, \tag{3}$$

sendo $B = (\frac{1}{\Delta t}M + \frac{1}{Re}K) \in \mathbb{R}^{n \times n}$ uma matriz simétrica e definida positiva, as demais matrizes $G, D^T \in \mathbb{R}^{n \times m}$. No lado direito temos os vetores $b_1^* = b_1 + \frac{1}{\Delta t}M\mathbf{v}_d^n$ e b_2. A terceira equação em (2) produz um sistema linear de resolução fácil, que não abordamos no nosso trabalho.

1.1 Núcleo de cálculos intensivos

Consideramos núcleos de cálculos intensivos de um sistema linear esparso as operações computacionais em matrizes cujas entradas, em sua grande maioria, é composta por elementos nulos. O principal núcleo de cálculo de um sistema esparso que focamos no nosso trabalho é a operação de multiplicação matriz esparsa-vetor, também conhecida como SpMV (sparse matrix-vector).

Na utilização dos métodos iterativos para solução dos sistemas lineares provenientes da equação 3 encontramos as operações SpMV nos algoritmos dos métodos que utilizamos: Gradiente Conjugado [Hestenes and Stiefel, 1952] e GMRes [Saad and Schultz, 1986].

2 Estrutura de dados

Procuramos otimizar o desempenho da operação de multiplicação matriz esparsa-vetor e, ao trabalharmos com matrizes esparsas, precisamos de uma estrutura de dados adequada com a finalidade de evitar o armazenamento explícito de elementos nulos desnecessariamente. Uma estrutura de dados em bloco também se torna interessante pois podemos manter blocos em nível rápido de memória durante a realização da operação SpMV.

2.1 Hierarquia de memória

Alterações nas estruturas de dados e nos códigos podem prover uma melhoria na performance. A memória do computador está diretamente ligada a essa melhoria. Essa memória possui um hierarquização com diferentes níveis, que vão desde o nível *rápido, caro e pequeno* até o nível *lento, barato e grande*. Os registradores compõem a memória mais rápida. É nos registradores que ocorrem as operações aritméticas e lógicas. Os dados que estão armazenados em um nível de hierarquia podem se mover para níveis adjacentes. A velocidade desses movimentos é mais rápida perto do topo da hierarquia e mais lenta perto da base da hierarquia.

Um dos principais obstáculos para se ter bom desempenho é a velocidade de memória: ler ou escrever dados da memória requer mais tempo do que realizar uma operação de ponto flutuante. Fato conhecido como *engarrafamento de tamanho de banda de memória* (memory bandwidth bottleneck). Ou seja, uma aplicação pode ter um gasto maior de tempo devido à espera dos dados [van der Pas, 2002].

Consideraremos a operação SpMV como sendo $y \leftarrow y + Ax$, onde A é uma matriz esparsa de ordem $m \times n$ com k elementos não-nulos e x e y vetores densos. Em notação de Einstein, a operação SpMV fica

$$y_i \leftarrow y_i + a_{ij}x_j, \quad \forall a_{ij} \neq 0, \tag{4}$$

com $A = (a_{ij})_{m \times n}$. Neste tipo de operação cada elemento a_{ij} é lido uma única vez. Se os vetores x e y podem ser armazenados totalmente em um nível de memória rápida, então o tempo de execução da operação SpMV fica diretamente dependente do tempo gasto ao se movimentar os

elementos de A entre níveis de memória. O ônus ao se realizar a operação (2 flops por cada elemento da matriz) é insignificante em comparação ao custo de leitura, em nível de memória, da matriz A. Daí, o cuidado que devemos ter ao se armazenar uma matriz.

2.2 Formatos esparsos

Existem várias estruturas de dados para armazenar matrizes esparsas. Em nossos testes, utilizamos praticamente dois formatos de armazenamento de matrizes esparsas, que detalhamos a seguir.

O formato CSR (compressed sparse row) consiste em armazenar contiguamente cada linha da matriz, possibilitando assim o acesso direto a cada linha. CSR utiliza três vetores: *val* - armazena os valores dos elementos não nulos, *ind* - armazena os índices de coluna de cada elemento e *ptr* - indica o início de cada linha. O vetor *ptr* possui $m+1$ elementos, sendo que o último elemento aponta para a próxima posição vazia em *val* e *ind*.

Na implementação de SpMV, utilizando matriz no formato CSR, ocorre um problema de indireção, grifado em vermelho na notação abaixo.

$$y[i] \leftarrow y[i] + val[l] \cdot x[ind[l]] \tag{5}$$

A reutilização dos elementos de y pode ocorrer durante a realização do loop interno na operação de multiplicação, tendo a princípio o y armazenado no registrador durante a execução do loop.

O formato BCSR (block compressed sparse row) pode ser visto como o formato CSR em bloco. Este formato é bastante utilizado para explorar a estrutura natural de blocos densos de uma matriz esparsa, típico de matrizes originárias do método de elementos finitos. BCSR divide a matriz em várias submatrizes densas de tamanho $r \times c$. Os valores de A são armazenados em um vetor *val* de $K_{rc} \cdot r \cdot c$ elementos, onde K_{rc} representa o número de blocos não nulos. Os blocos são armazenados consecutivamente por linha e cada bloco é armazenado em algum formato denso. Nesse formato, cada bloco é tratado como um bloco denso, necessitando, em alguns casos, preenchimento explícito de zeros. Em BCSR os blocos não são únicos. As bibliotecas de álgebra linear possuem convenções distintas para selecionar os blocos, como em [Im et al., 2004] e [Saad, 1990].

3 Blocagem de matrizes

Procuramos otimizar o desempenho da operação SpMV trabalhando com matrizes armazenadas em estruturas de blocos. Dessa forma podemos aproveitar a reutilização de dados, isto é, manter blocos de matrizes em memória alta, durante a realização do loop interno no processamento da multiplicação. Utilizando linguagem de programação C, os conteúdos dos registradores podem ser controlados diretamente pelo programa e, dessa forma pode ser escolhido um conjunto de valores para ser armazenado nos registradores.

Para realizarmos a operação SpMV, $y \leftarrow y + Ax$, com *blocagem no registrador*, armazenamos a matriz A no formato BCSR. O formato BCSR explora a presença natural de blocos densos na matriz reorganizando a estrutura de dados em uma sequência de pequenos blocos densos (blocos pequenos, o necessário, para caberem em registradores). O desenvolvimento de SpMV ocorre bloco a bloco. Para cada bloco, podemos reutilizar os correspondentes c elementos do vetor x e os r elementos do vetor y mantendo-os em registradores.

O formato BCSR armazena uma quantidade menor de índices de coluna do que o formato CSR, um índice por bloco, ao invés de um índice por elemento não nulo. Dessa forma, temos o tráfego de memória reduzido em virtude da redução de custo de armazenamento. Entretanto, ao se fixar o tamanho do bloco em uma determinada matriz, pode-se necessitar de preenchimento

explícito de elementos nulos em cada bloco armazenado. Enfim, um armazenamento extra de zeros além de cálculos extras com esses zeros.

A principal vantagem de *blocar* as matrizes através da forma *blocagem no registrador* é a diminuição de tráfego de dados entre registradores e memória, fato que pode levar a um aumento substancial da velocidade de processamento, como veremos em nossos testes. Podemos citar como desvantagem, o custo obtido ao se converter a matriz para o formato em bloco.

4 Otimização da multiplicação matriz esparsa-vetor

Nesta seção mostramos um modelo de seleção de bloco, com a finalidade de melhorar a performance da operação de multiplicação matriz esparsa-vetor, baseado no trabalho de [Im et al., 2004] e [Vuduc et al., 2005].

O modelo para a escolha do tamanho de bloco de uma determinada matriz em uma determinada máquina é baseado, em grande parte, na informação de perfil para a determinada máquina. Os componentes básicos do modelo são: 1- uma aproximação da taxa de operações de ponto flutuante (em Mflops - 10^6 flops) de uma matriz com um determinado tamanho de bloco e 2- uma aproximação para a quantidade de cálculos, desnecessários, que serão realizados devido ao preenchimento explícito de zeros ao transformar a matriz em uma estrutura de blocos.

O primeiro item pode ser determinado ao se registrar a performance da matriz em bloco em cada máquina de interesse. Utiliza-se, como medida para essa taxa, a performance de uma matriz densa armazenada em formato esparso em bloco.

O segundo item requer informação a respeito dos elementos não nulos da matriz. Entretanto não é realizada a construção e medição da performance da matriz armazenando-a em cada tamanho de bloco possível. Isto seria muito caro, isto é, elevaríamos o tempo de execução. Para se calcular o número exatos de zeros preenchidos, deve ser feita uma varredura sobre toda a estrutura de dados. Ao invés disso, utiliza-se uma estimativa ao se realizar uma pequena e rápida varredura sobre somente uma parte da matriz.

Para estimar a performance de uma matriz em blocos, utiliza-se uma matriz densa em formato esparso com o tamanho de bloco determinado. Como esta estimativa requer uma quantidade significante de computação ao ser realizada para todos os tamanhos de bloco (dentro de um limite), logo ela é realizada apenas uma vez por cada máquina e não por cada matriz. Dessa forma definimos como *Perfil de performance* o conjunto de valores de performance (em Mflops) ao executar a operação SpMV com uma matriz densa armazenada em formato esparso em blocos, sendo os blocos de tamanho $r \times c$, com $r = 1, \ldots, r_{max}$ e $c = 1, \ldots, c_{max}$. Denotamos como *perfil de performance*: P_{rc}(denso).

Mostramos na figura (1) os resultados coletados no *perfil de performance* da máquina (2.2GHz AMD Athlon 64x2 e 3.0GHz Intel Pentium 4, respectivamente), utilizando formato BCSR. Cada quadrado, na figura, representa um bloco de tamanho $r \times c$. Em cada bloco temos o percentual de performance da operação SpMV em relação ao resultado do bloco 1×1, pois este bloco representa o formato CSR, ou seja, uma matriz armazenada sem estrutura de bloco.

Para termos uma aproximação do cálculo desnecessário, estimamos a quantidade de zeros que foi adicionada à estrutura de dados. Definimos como *Taxa de preenchimento* (f_{rc}) a razão entre o número de valores armazenados (incluindo os zeros explícitos) e o número de elementos não nulos da matriz original.

A *taxa de preenchimento* ocorre em função do tamanho $r \times c$ do bloco. Como, na operação SpMV, o número de operações de ponto flutuante é proporcional ao número de elementos armazenados, essa taxa pode ser vista como uma estimativa do custo computacional.

Para simplificar notação, denotamos K_{rc} como sendo a quantidade de blocos de tamanho $r \times c$ requeridos para converter a matriz A, de ordem $m \times n$ com k elementos não nulos, inicialmente armazenada no formato CSR, para o formato BCSR.

Cada bloco de tamanho $r \times c$ possui $r \cdot c$ elementos. Logo, a quantidade de valores armazenados é igual a quantidade de blocos (K_{rc}) multiplicada por $r \cdot c$. Daí, a *taxa de preenchimento* pode

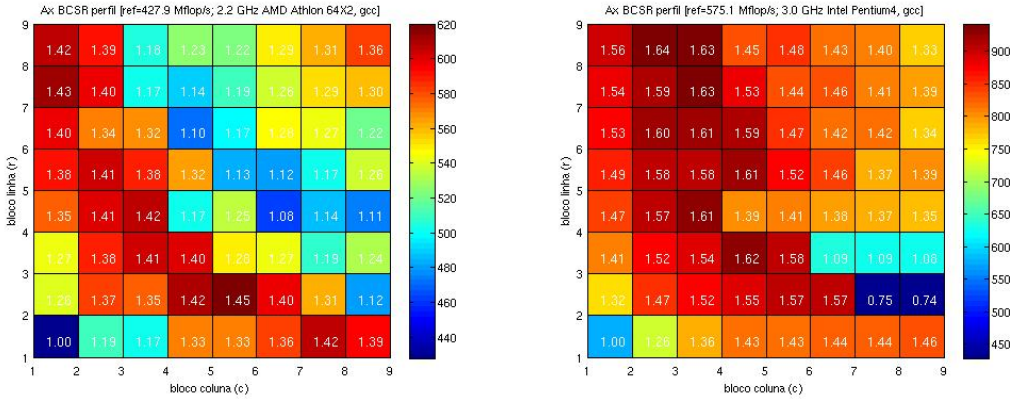

Figura 1: Perfil de performance da operação $A \cdot x$ nas máquinas 2.2GHz AMD Athlon 64x2 e 3.0GHz Intel Pentium 4.

ser equacionada como $f_{rc} = \frac{K_{rc} \cdot r \cdot c}{k}$. Consequentemente, $f_{rc} \geq 1$.

4.1 Modelo para seleção de bloco

Detalhamos neste tópico o modelo, adotado em nossos testes, para seleção do tamanho de bloco de uma matriz esparsa em tempo real. O modelo é uma heurística proposta por [Im et al., 2004] e [Vuduc et al., 2005]. As etapas da heurística são:1- computar, uma única vez por máquina utilizada, o *perfil de performance*; 2- quando a matriz A é conhecida em tempo real, calcular *estimativa da taxa de preenchimento* para todos r e c, tais que $1 \leq r \leq r_{max}$ e $1 \leq c \leq c_{max}$ e 3- escolher r e c que maximizam a *estimativa de perfil de performance*.

Definimos como *estimativa da taxa de preenchimento* uma porcentagem σ da taxa de preenchimento, sendo σ um parâmetro selecionado pelo usuário (variando de 0 a 1) que indica a acurácia da taxa estimada. Denotamos esta estimativa como $\hat{f}_{rc}(A, \sigma)$. O parâmetro σ indica uma porcentagem com a qual o programa efetua uma varredura na matriz. Por exemplo: se $\sigma = 0,1$ então será feita uma varredura em 10% da matriz.

Definimos como *estimativa de perfil de performance* a razão entre o perfil de performance e a estimativa da taxa de preenchimento. Denotamos esta estimativa como $\hat{P}_{rc}(A, \sigma)$. Logo, temos $\hat{P}_{rc}(A, \sigma) = \frac{P_{rc}}{\hat{f}_{rc}(A,\sigma)}$. Esta equação, além de encontrar o melhor tamanho de bloco, também pode listar otimais tamanhos de blocos.

5 Resultados numéricos

As matrizes utilizadas em nossos testes foram recolhidas do repositório [Davis, 1999]. Na tabela 1, listamos algumas das matrizes utilizadas, informando dimensão e número de elementos não nulos de cada matriz.

Efetuamos operações de multiplicação matriz esparsa-vetor (SpMV) utilizando 2 coleções distintas: *OSKI* (Optimized Sparse Kernel Interface) [Vuduc et al., 2005] - uma coleção de códigos na linguagem *C* que promove otimização de núcleos esparsos de álgebra linear, através de heurística detalhada anteriormente; e *MKL* (Intel Math Kernel Library) [Intel, 2007] - coleção de rotinas e funções em *Fortran* e *C* que realizam uma variedade de operações com vetores e matrizes. Utilizamos em MKL rotinas do BLAS [BLAST Forum, 2001] para matrizes esparsas (Sparse BLAS), sem estruturas em blocos.

Utilizando as matrizes selecionadas efetuamos (com a máquina Intel Core 2 Duo T7500 2200MHz) em cada uma das coleções citadas acima, a operação de multiplicação matriz-vetor (SpMV) executadas 500 vezes em cada caso (utilizamos essa quantidade de operações para uma melhor comparação de tempo com os dados obtidos). Relatamos na tabela 1 (parte à direita da

Número	Matriz	Dimensão	Não-nulos	OSKI	MKL	Razão (MKL/OSKI)
1	e30r0000	9661	306356	4,2e-01	7,9e-01	1,88
2	e40r0000	17281	553956	8,17e-01	1,47	1,80
3	venkat01	62424	1717792	2,13	5,28	2,48
4	bcsstk16	4884	147631	1,2e-01	3,7e-01	3,08
5	bcsstk35	30237	740200	9,0e-01	2,19	2,43
6	bcsstk36	23052	583096	7,5e-01	1,66	2,21
7	gyro	17361	519260	4,7e-01	1,37	2,91
8	nemeth26	9506	760633	9,0e-01	2,02	2,24
9	af23560	23560	484256	6,1e-01	1,20	1,97
10	ct20stif	52329	1375396	2,14	4,21	1,97

Tabela 1: Na tabela à esquerda temos a descrição das matrizes utilizadas. Na tabela à direita temos os tempos (em segundos) da operação SpMV utilizando as coleções OSKI e MKL

tabela) os tempos (em segundos) de execução da operação de multiplicação para cada matriz. Nesta tabela, a última coluna representa a relação do tempo em MKL em comparação ao OSKI.

Realizamos uma comparação do tempo processado desde o início da heurística de otimização em blocos até o final de uma certa quantidade de operações SpMV (no loop interno) e do tempo processado na mesma quantidade de operações SpMV sem utilizar a heurística de otimização. Essa comparação foi realizada no mesmo código. Mostramos, no gráfico 2 esta comparação com a matriz 9. Registramos os dados comparativos no gráfico . Empregamos uma função, baseada em série de Fourier, para ajuste de curva que se aproxima dos pontos provenientes dos testes.

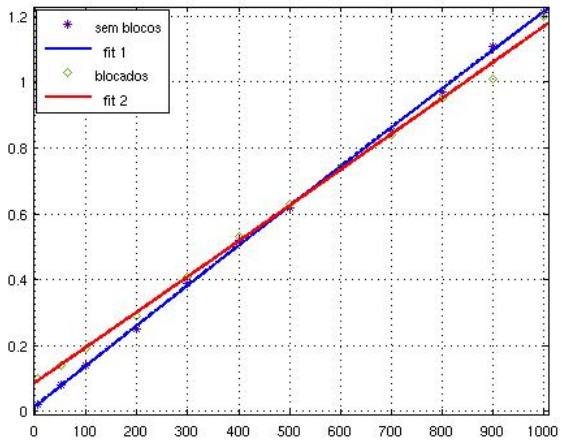

Figura 2: Comparação realizada com a matriz 9. A curva azul mostra o desempenho da operação SpMV sem otimização e a curva vermelha mostra a operação com otimização. O eixo horizontal representa a quantidade de multiplicações realizadas e o eixo vertical representa o tempo verificado (em segundos).

5.1 Conclusões

Constatamos que o tempo de processamento da operação SpMV se torna menor ao se trabalhar com matrizes em bloco. O tempo verificado (tabela 1), otimizado com matrizes em bloco, variou, em nossos testes, em um intervalo que vai de 44% (matriz 2) até 75% menor (matriz 14) do que o tempo verificado ao utilizar a coleção [Intel, 2007], que não utiliza estrutura em blocos.

Não podemos deixar de discutir o custo da otimização, ou seja, o custo de execução da heurística e da conversão da matriz para estrutura de dados em bloco. Com as matrizes utilizadas

verificamos que o custo da otimização não é superior a 10 vezes o tempo de execução, sem otimização, da operação SpMV.

O custo maior da otimização é dominado pela conversão da estrutura de dados da matriz, mudança da estrutura para uma formato em bloco. Fizemos a mudança da taxa *sigma* (σ da *taxa de preenchimento estimada*) que determina a fração da matriz a ser verificada para se ter a estimativa da quantidade de zeros (elementos nulos) a ser armazenada explicitamente ao se converter a matriz (blocar a matriz). A variação de tempo registrada em todos os casos se mostrou muito pequena.

Vimos nos nossos testes que a vantagem de otimizar a operação de multiplicação pode somente ocorrer quando efetuamos a operação SpMV com a mesma matriz por uma quantidade relativamente grande. Como constatado na figura 2, essa quantidade pode ultrapassar a faixa de 500 produtos. Esses casos nos mostram que a dimensão da matriz e sua utilização determinará o uso da heurística. Em alguns métodos numéricos, como o Gradiente Conjugado, que mantém a estrutura da matriz a cada iteração, a heurística pode ter uma utilidade eficaz.

Referências

[BLAST Forum, 2001] BLAST Forum (2001). *Basic linear algebra subprograms techical (BLAST) forum*. Univesity of Tennessee.

[Davis, 1999] Davis, T. (1999). *Sparse Matrix Collection*. University of Florida, http://www.cise.ufl.edu/ davis/sparse/.

[Hestenes and Stiefel, 1952] Hestenes, M. and Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Stand*, 49:409–436.

[Im et al., 2004] Im, E.-J., Yelick, K. A., and Vuduc, R. (2004). SPARSITY: Framework for optimizing sparse matrix-vectormultiply. *International Journal of High Performance ComputingApplications*, 18(1):135–158.

[Intel, 2007] Intel (2007). Math kernel library. http://developer.intel.com/software/products/mkl/.

[Pironneau, 1982] Pironneau, O. (1982). On the transpor-diffusion algorithm and its applications to the Navier-Stokes equation. *Numer. Math*, 38:309–332.

[Quarteroni and Valli, 1994] Quarteroni, A. and Valli, A. (1994). *Numerical Approximation of Partial Differential Equations*. Springer–Verlag, Berlin.

[Saad, 1990] Saad, Y. (1990). SPARSKIT: A basic toolkit for sparse matrix computations. Technical Report 90-20, NASA Ames Research Center, Moffett Field, CA.

[Saad and Schultz, 1986] Saad, Y. and Schultz, M. H. (1986). GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7(3):856–869.

[van der Pas, 2002] van der Pas, R. (2002). Memory hierarchy in cache-based systems. *Sun Blueprints*.

[Vuduc et al., 2005] Vuduc, R., Demmel, J. W., and Yelick, K. A. (2005). OSKI: A library of automatically tuned sparse matrix kernels. In *Proceedings of SciDAC 2005*, Journal of Physics: Conference Series, San Francisco, CA, USA. Institute of Physics Publishing. (*to appear*).