Proceeding Series of the Brazilian Society of Applied and Computational Mathematics, Vol. 5, N. 1, 2017.

Trabalho apresentado no CNMAC, Gramado - RS, 2016.

Proceeding Series of the Brazilian Society of Computational and Applied Mathematics

Nova implementação do precondicionador AINV em acelerador NVIDIA utilizando a biblioteca CUSP

Michael Souza[1]
Departamento de Estatística e Matemática Aplicada, UFC, Fortaleza, CE
Luiz Mariano Paes de Carvalho[2]
Instituto de Matemática e Estatística, UERJ, Rio de Janeiro, RJ
João P. Zanardi[3]
Faculdade de Engenharia, UERJ, Rio de Janeiro, RJ
Douglas A. Augusto[4]
Fundação Oswaldo Cruz, Rio de Janeiro, RJ
Paulo Goldfeld[5]
Instituto de Matemática, UFRJ, Rio de Janeiro, RJ

Resumo. Comparamos o desempenho em placa gráfica (GPU) do precondicionador `AINV` baseado na aproximação da inversa. Os resultados de nossos experimentos numéricos e computacionais indicam que nossa implementação é competitiva e possui resultados melhores do que a versão disponível na biblioteca `CUSP` largamente utilizada em aplicações com aceleradores NVIDIA. Além disso, apresentamos as ideias principais na definição do precondicionador e detalhes sobre a implementação.

Palavras-chave. Inversa Aproximada, Sistemas lineares, Precondicionadores, Paralelismo, Aceleradores

1 Introdução

Computadores paralelos são uma realidade onipresente. Neste contexto, o desenvolvimento de *solvers* paralelos eficientes para a resolução de sistemas lineares $Ax = b$ esparsos de grande porte passa a ser um dos problemas mais importantes da computação científica. Esta importância decorre da frequente utilização direta de modelos lineares ou ainda de métodos baseados em linearizações de modelos mais complexos.

Os métodos iterativos modernos para resolução de sistemas lineares esparsos de grande porte como, por exemplo, os métodos de subespaço de Krylov, substituem operações do tipo matriz-matriz por operações vetor-vetor ou ainda matriz-vetor, mas sua eficiência depende fortemente de técnicas de precondicionamento. Sendo assim, para alcançar escalabilidade, tanto os métodos de Krylov quando os precondicionadores precisam ser paralelos.

[1]michael@ufc.br
[2]luizmc@ime.uerj.br
[3]jpzanardi@gmail.com
[4]daa@fiocruz.br
[5]goldfeld@ufrj.br

Contudo, muitos dos precondicionadores de uso geral, como os baseados em fatorações incompletas do operador A, são altamente sequenciais e implementações eficientes em computadores paralelos têm sido um grande desafio [6]. Por isso, nas últimas décadas, os métodos que aproximam diretamente a inversa de A, sendo portanto bons candidados a precondicionadores, têm atraído bastante atenção [8, 10]. A mais importante vantagem destes métodos é que sua aplicação consiste simplesmente no produto matriz-vetor. Isto os torna extremamente convenientes para aplicação em ambientes computacionais paralelos.

O presente trabalho é a continuação do artigo [9], onde apresentamos as comparações com respeito ao número de iterações dos *solvers* lineares utilizando diferentes precondicionadores baseados em aproximações da inversa. As comparações anteriormente realizadas foram feitas sem explorar o paralelismo não levando em consideração o desempenho computacional (tempo). Neste artigo, focamos no desempenho paralelo da aplicação do precondicionador `AINV` proposto por Benzi [6], o mais promissor dentre os analisados no trabalho anterior.

A seção 2 deste artigo será dedicada a uma rápida apresentação do precondicionador `AINV`. Na seção 3, faremos a descrição da implementação que ora propomos. As comparações e resultados experimentais serão apresentados na seção 4. Finalmente, na seção 5, concluímos com um resumo do presente trabalho.

2 O precondicionador *AINV*

O precondicionador ideal para o sistema $Ax = b$ seria a matriz $M = A^{-1}$, pois permitiria resolver o sistema linear em uma única iteração. Infelizmente, esta alternativa é inviável pelo tempo necessário para calcular a inversa e, mais ainda, pelo consumo de memória. De fato, a elevada esparsidade dos operadores de muitos problemas reais faz que com que sistemas cada vez maiores sejam considerados. Atualmente, não é difícil encontrar sistemas com dezenas de milhões de colunas. Neste contexto, o armazenamento de inversas potencialmente densas é proibitivo. Para contornar estas dificuldades, os precondicionadores baseados em aproximações da inversa operam determinando fatores Z e W tais que $Z \approx U^{-1}$ e $W^t \approx L^{-1}$, onde $A = LDU$ é decomposição triangular de A.

O precondicionador `AINV` proposto por Benzi e Tuma aproxima os inversos dos fatores da decomposição $A = LDU$ utilizando um esquema de biconjugação [3, 4, 6]. Mais especificamente, dada uma matriz não-singular $A \in \mathbb{R}^{n\times n}$, o precondicionador `AINV` gera dois conjuntos de vetores $\{z_i\}_{i=1}^n$ e $\{w_i\}_{i=1}^n$ A-biconjugados, ou seja, para $W = [w_1, \ldots, w_n]$ e $Z = [z_1, \ldots, z_n]$ temos $W^t AZ = D$, ou ainda, $W^t = L^{-1}$ e $Z = U^{-1}$. Os fatores Z e W podem ser densos, então para evitar que isto ocorra os elementos z_{ij} e w_{ij} em magnitude menores que um valor especificado são eliminados.

3 Comentários sobre a implementação

O precondicionador `AINV` pode ser implementado como um processo de ortogonalização de Gram-Schmidt generalizado com respeito a forma bilinear associada ao operador A [5, 6]. A forma geral da nossa implementação do precondicionador `AINV` pode ser vista

DOI: 10.5540/03.2017.005.01.0093

no Algoritmo 1 (apresentado em notação do Matlab apenas por simplicidade). Como pode ser visto, as construções dos fatores W e Z são independentes podendo ser feitas paralelamente.

Algoritmo 1: AINV

```
1  function [Z,D,W] = ainv(A,droptol,nmax)
2  [Z,p] = ainv_core(A,droptol,nmax);
3  W = ainv_core(A',droptol,nmax);
4  W = W';
5  D = sparse(1:n,1:n,1./p);
6  end
7
8  function [Z,p] = ainv_core(A,droptol,nmax)
9  n = length(A);
10 Z = speye(n);
11 s = sset(Z);
12 for i = 1:n
13     p(i) = A(:,i)' * Z(:,i);
14
15     % eliminates the possibility of division by zero
16     if(abs(p(i))<1E-8) p(i)=1; end
17
18     if(i == n) break; end
19
20     cols = s.get_cols(i, A(:,i));
21
22     % update Z and W
23     for k = 1:length(cols)
24        j = cols(k);
25        p(j) = A(:,i)' * Z(:,j);
26        Z(:,j) = Z(:,j) - (p(j)/p(i))*Z(:,i);
27        Z(:,j) = drop(Z(:,j),nmax,droptol,j);
28        s.update_col(j, Z(:,j));
29     end
30 end
31 end
```

O custo de construção do precondicionador `AINV` é fortemente dependente do número de iterações do laço mais interno (linha 23), onde a ortogonalização é efetivamente realizada. Para otimizar a performance é necessário fazer o menor número possível de iterações. Pela estratégia de eliminação (*drop*) dos valores com magnitude menor que um dado valor, garantimos a esparsidade das colunas dos fatores e, portanto, a ortogonalidade simbólica entre muitas colunas. Exploramos esta característica das colunas, reduzindo o laço mais interno a apenas as colunas que não sejam simbolicamente ortogonais, ou seja, consideramos apenas as colunas que precisam ser atualizadas. Mais precisamente, na i-ésima iteração do laço externo (linha 12), consideramos apenas as colunas de Z cujo produto interno $A(:,i)' * Z(:,j)$ não é nulo. Estas colunas são as mesmas que possuem ao menos um elemento não nulo na mesma posição (linha) que $A(:,i)$. Nossa implementação utiliza a classe especializada `sset` para gerenciar as informações sobre a estrutura de esparsidade do fator Z (ver Algoritmo 2). Com estas estratégias, obtemos uma implementação com

DOI: 10.5540/03.2017.005.01.0093

complexidade $O(nm^2)$, onde n é o número de colunas de A e m é o número médio de elementos não-nulos em cada uma das colunas de A.

Algoritmo 2: SSET

```
classdef sset < handle
    properties
        s = {}; % empty cell
    end

    methods
        function this = sset(n)
            this.s = cell(n,1);
            for i = 1:n
                this.s{i} = i;
            end
        end

        function this = update_col(this, j, Zj)
            irow = find(Zj);
            this.s{irow(k)} = [this.s{irow(k)},j];
        end

        function cols = get_cols(this,i,Ai)
            cols = [];
            rows = find(Ai); % index of nnz elements of A(:,i)
            for j = 1:length(rows)
                irow = rows(j);
                % remove duplicated indices and keep the
                % indices greater than i
                this.s = unique(this.s{j}(this.ss{i} > i));
                cols = [cols,this.ss{i}];% concatenate
            end
            cols = unique(cols(cols>i));
        end
    end
end
```

4 Experimentos computacionais

Nossos experimentos foram feitos utilizando os métodos de Krylov `GMRES` e `BICGSTAB` com implementações em GPU (*Graphics Processing Unit*). Utilizamos a versão 0.4.0 da biblioteca `CUSP`, que fornece uma interface flexível e de alto nível para manipulação de matrizes esparsas e resolução de sistemas lineares [2]. Além da interface de manipulação de matrizes esparsas, a biblioteca `CUSP` também disponibiliza os precondicionadores (a) `DIAGONAL` formado pela matriz diagonal cujos elementos são os inversos dos elementos diagonais de A; (b) a clássica decomposição incompleta `ILU(0)`; (c) uma variante de inversa aproximada chamada `BRIDSON` baseada em uma formulação de produto tensorial (*outer product*) [7]; (d) e, por completude, também aplicamos os *solvers* diretamente sem precondicionamento (`NONE`).

DOI: 10.5540/03.2017.005.01.0093

A utilização de qualquer precondicionador pode ser separada em duas etapas: (i) a *fase de construção* do precondicionador e (ii) a *fase de aplicação* durante as iterações dos *solvers* lineares. A fase de construção de nossa implementação é feita sequencialmente em CPU, ou seja, não paralelizamos a fase de construção. Nossa opção justifica-se pelo caráter fortemente sequencial da construção do precondicionador `AINV` que basicamente realiza uma biconjugação. A implementação foi feita em linguagem `C++`.

Fixamos a tolerância relativa dos resíduos dos métodos de Krylov em 10^{-4} e realizamos no máximo 500 iterações. Além disso, reinicializamos o método `GMRES` a cada 30 iterações. Com estas configurações, consideramos um método como bem sucedido em uma dada instância se uma solução com resíduo relativo $\epsilon = ||Ax - b||_2/||b||_2 < 10^{-4}$ pôde ser encontrada em, no máximo, 500 iterações.

Utilizamos em nossos experimentos uma CPU Intel Core i7-3820 de 3.66GHz com 16GB de RAM, e uma GPU Tesla K20c com 2096 cores e 5GB de memória GDDR5. O sistema operacional utilizado foi o Linux em distribuição Ubuntu 15.04 e compiladores GCC 4.8.2 e NVCC 7.0.

A Tabela 1 apresenta as 18 matrizes utilizadas em nossos experimentos. O número n de colunas das matrizes consideradas vai de aproximadamente 500 a 17750, já o número nnz de elementos não-nulos varia de aproximadamente 3500 a 126k. Esse é o mesmo conjunto de matrizes não-simétricas apresentadas em [5]. As matrizes foram armazenadas no formato `CSR` (*Compressed Storage Row*) que é geral, não assume qualquer hipótese estrutural e é bastante eficiente em termos de memória já que armazena apenas os valores não-nulos. Por outro lado, não é o mais eficiente em termos de acesso já que faz uso de endereçamento indireto [1].

Tabela 1: Número de colunas e elementos não-nulos das matrizes consideradas.

#	matriz	n	nnz	#	matriz	n	nnz
01	*orsreg_1*	2205	14133	10	*sherman1*	1000	3750
02	*orsirr_1*	1030	6858	11	*sherman2*	1080	23094
03	*orsirr_2*	886	5970	12	*sherman3*	5005	20033
04	*saylr3*	1000	3750	13	*sherman4*	1104	3786
05	*saylr4*	3564	22316	14	*sherman5*	3312	20793
06	*add32*	4960	23884	15	*pores_2*	1224	9613
07	*add20*	2395	17319	16	*pores_3*	532	3474
08	*memplus*	17758	126150	17	*watt_1*	1856	11360
09	*swang1*	3169	20841	18	*watt_2*	1856	11550

Na Tabela 2, podemos ver o percentual de instâncias resolvidas por cada par de precondicionador e método de Krylov. O precondicionador `AINV` apresentou os melhores resultados, deixando de resolver apenas uma das dezoito instâncias. A Tabela 2 também ilustra a importância do precondionamento, pois quase a metade das instâncias não puderam ser resolvidas sem ele.

Na Figura 1 podemos ver a performance normalizada, ou seja, o tempo de construção mais o tempo de aplicação de cada um dos métodos e precondicionadores divididos pelo maior deles, considerando-se apenas os testes bem sucedidos. Como pode ser visto, nossa implementação do precondicionador `AINV` obteve os melhores resultados com o método `GMRES`. Já com o método `BICGSTAB`, parece haver uma leve vantagem do com o precondicionador `DIAGONAL`, mas mesmo neste caso o precondicionador `AINV` apresentou bom desempenho e estabilidade, sendo o melhor ou um dos dois melhores em praticamente todas as instâncias.

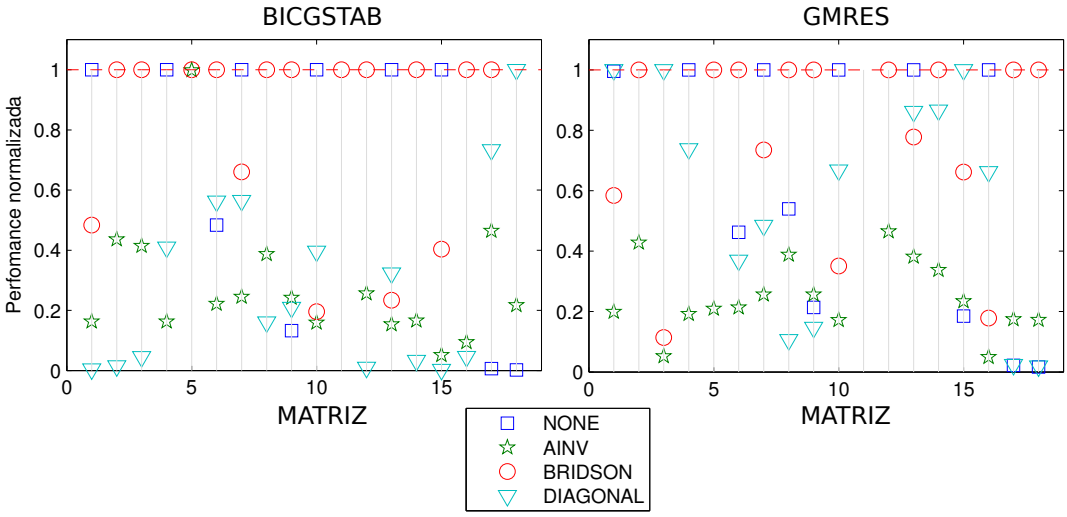

Figura 1: Performance normalizada de cada precondicionador e método de Krylov.

A partir dos resultados apresentados, podemos concluir que nossa implementação do precondicionador `AINV` é competitiva e seu desempenho pode ser comparado ao dos precondicionadores disponibilizados pela biblioteca `CUSP`.

Tabela 2: Percentual de instâncias resolvidas por cada precondicionador.

PRECOND	GMRES	BICGSTAB
NONE	67%	56%
DIAGONAL	78%	89%
BRIDSON	89%	89%
AINV	94%	94%

5 Conclusões

Apresentamos no presente trabalho um estudo comparativo da performance em GPU de uma nova implementação do precondicionador `AINV` baseado na aproximação da inversa da matriz do sistema linear. Comparamos nossa implementação com os precondicionadores disponibilizados pela biblioteca `CUSP`, desenvolvida especificamente para GPU's da NVIDIA. Finalmente, nossos resultados indicam que nossa implementação do precondicionador `AINV` é competitiva e seu desempenho pode ser comparado ao dos precondicionadores da biblioteca `CUSP`.

Referências

[1] N. Bell and M. Garland. Efficient sparse matrix-vector multiplication on cuda. Technical report, Nvidia Technical Report NVR-2008-004, Nvidia Corporation, 2008.

[2] N. Bell and M. Garland. Cusp: Generic parallel algorithms for sparse matrix and graph computations, 2012. Version 0.3.0.

[3] M. Benzi. *A direct row-projection method for sparse linear systems.* PhD thesis, North Carolina State University, 1993.

[4] M. Benzi, C. D. Meyer, and M. Tůma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM Journal on Scientific Computing*, 17(5):1135–1149, 1996.

[5] M. Benzi and M. Tůma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM Journal on Scientific Computing*, 19(3):968–994, 1998.

[6] M. Benzi and M. Tůma. A comparative study of sparse approximate inverse preconditioners. *Applied Numerical Mathematics*, 30(2):305–340, 1999.

[7] R. Bridson and W.-P. Tang. Ordering, anisotropy, and factored sparse approximate inverses. *SIAM Journal on Scientific Computing*, 21(3):867–882, 1999.

[8] R. Bru, J. Marín, J. Mas, and M. Tůma. Improved balanced incomplete factorization. *SIAM Journal on Matrix Analysis and Applications*, 31(5):2431–2452, 2010.

[9] L. M. P. Carvalho, J. P. Zanardi, Ítalo Nievinski, and M. Souza. An overview of approximate inverse methods. In *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics*, volume 3, pages 010097–1 – 010097–7, 2015.

[10] L. Y. Kolotilina and Y. A. Yu. Factorized sparse approximate inverse preconditioning. i:theory. *SIAM Journal on Matrix Analysis and Applications*, 14:45–58, 1993.

DOI: 10.5540/03.2017.005.01.0093