Trabalho apresentado no XXXVIII CNMAC, Campinas - SP, 2018.

Proceeding Series of the Brazilian Society of Computational and Applied Mathematics

Precondicionador de inversa aproximada esparsa com construção paralela para matrizes SP

Ítalo C. L. Nievinski [1]
Faculdade de Engenharia, PPGEM, UERJ, Rio de Janeiro - RJ
Luiz Mariano Carvalho[2]
Instituto de Matemática e Estatística , UERJ, Rio de Janeiro - RJ
Michael Ferreira de Souza[3]
Instituto de Estatística e Matemática Aplicada, UFC, Fortaleza - CE

Resumo. Apresentamos um precondicionador para a solução de um sistema linear $Ax = b$, com A simétrica e positiva, através de um método iterativo de Krylov. Esse precondicionador é baseado na aproximação de uma fatoração da inversa de A e tem sua construção feita de forma massivamente paralela.

Palavras-chave. precondicionadores, aproximação da inversa, paralelismo massivo, KNL.

1 Introdução

Precondicionadores que aproximem a inversa de uma matriz são particularmente úteis em arquiteturas paralelas, pois sua aplicação resume-se a uma multiplicação matriz-vetor. Um destes algoritmos é o AINV [1] que constrói uma aproximação para a fatoração $ZDZ^{-1} = A^{-1}$, com A simétrica e positiva[4]. Ainda que a aplicação do precondicionador AINV seja paralela (multiplicação matriz-vetor), sua construção é intrinsecamente sequencial. No presente trabalho, propomos o algoritmo alternativo PARAINV para a construção de uma aproximação da fatoração de A^{-1}. O PARAINV apresenta resultados similares ao do AINV, mas possui construção massivamente paralela.

Na Seção 2, apresentamos o algoritmo AINV como motivação do algoritmo PARAINV. Discutimos na Seção 3 a dificuldade de se obter paralelismo na construção da inversa fatorada através do AINV e propomos uma maneira de se obter uma aproximação da inversa fatorada onde cada coluna é calculada independentemente, obtendo um paralelismo massivo. Na Seção 4 discutimos alguns detalhes relevantes da implementação da construção do precondicionador. Apresentamos, na seção 5, os resultados dos experimentos realizados com até 64 núcleos de um processador Xeon Phi da Intel (KNL), que demonstram

[1]italonievinski@gmail.com
[2]luizmc@ime.uerj.br
[3]michael@ufc.br
[4]Matriz SP é simétrica, $A = A^T$, e positiva, $x^T Ax > 0$, para qualquer vetor x não nulo.

DOI: 10.5540/03.2018.006.02.0312

a escalabilidade da construção do precondicionador PARAINV e comparamos os resultados com os do algoritmo AINV. A Seção 6 apresenta as conclusões obtidas através dos experimentos, os pontos fortes e fracos do precondicionador proposto e aponta possíveis desdobramentos deste trabalho.

2 Construção da Inversa aproximada fatorada

Se A é uma matriz simétrica e positiva, podemos obter A^{-1} através de um conjunto de direções conjugadas $z_1, z_2, \ldots, z_n$. Seja Z a matriz cuja coluna i é igual à z_i, temos que: $Z^T AZ = D$ onde D é uma matriz diagonal e $d_{ii} = z_i^T A z_i$. Segue que $A^{-1} = ZD^{-1}Z^T$. $ZD^{-1}Z^T$ é uma inversa fatorada de A.

Como pode ser visto em [1], um conjunto de direções conjugadas é obtido através da A-ortogonalização de um conjunto de vetores linearmente independentes $v_1, v_2, \ldots, v_n$. A escolha $v_i = e_i$ é computacionalmente conveniente. A matriz resultante Z é triangular superior com diagonal unitária. De fato,

$$Z = L^{-T}, \text{ onde } A = LDL^T. \tag{1}$$

O processo para construir a inversa fatorada é apresentado através do Algoritmo 1

```
1  [Z,D] = inverse_factorization(A)
2      n = length(A)
3      Z = eye(n)
4      for i = 1:n
5              pii = A(i,:) * Z(:,i)    % produto interno
6              for j = (i+1):n
7                  pij = A(i,:) * Z(:,j) % produto interno
8                  Z(:,j) = Z(:,j) - pij / pii * Z(:,i) % spmv
9              D(i,i) = pii
```

Algoritmo 1: Algoritmo de fatoração inversa

Em [1], Benzi e colaboradores propõem precondicionador AINV que aproxima Z descartando componentes com magnitude menor que uma dada tolerância, obtendo

$$M^{-1} := \bar{Z}\bar{D}^{-1}\bar{Z}^T \approx A^{-1}.$$

O uso da aproximação da inversa é particularmente interessante devido à crescente necessidade do uso do paralelismo para acelerar aplicações computacionais, dado que sua aplicação é feita a partir do produto de matriz esparsa por vetor denso, que pode ser paralelizado de maneira eficiente. Essa característica coloca esse precondicionamento em posição de destaque em comparação com outros precondicionadores difundidos na indústria. Um destes casos é fatoração Cholesky incompleta [3], aplicada através um solver triangular, que possui paralelismo limitado. No entanto, a construção do AINV, como proposta em [1], apresenta pouca oportunidade de paralelismo, devido a dependência de dados no seu algoritmo original, como veremos abaixo. Considerando o uso desse precondicionador em

DOI: 10.5540/03.2018.006.02.0312

arquiteturas paralelas, é conveniente que sua construção possa também tirar proveito do paralelismo, reduzindo assim o tempo total da solução do sistema linear.

3 Fatoração inversa aproximada com paralelismo

No Algoritmo 1, vemos que não é possível paralelizar o laço mais externo (linha 4), pois cada z_i depende dos $z_j,\ j < i$, (linhas 7 e 8). O laço interno (linha 6) pode ser paralelizado, no entanto levando em conta a esparsidade da matriz e do precondicionador calculado, a intensidade computacional deste laço é pequena. Pois além de serem operações com vetores esparsos, a maioria dos produtos internos, representados na linha 7, será nulo. Com esta condição, apenas um pequeno grau de paralelismo pode ser explorado, pois ao aumentar o número de núcleos de computação, não haverá dados suficientes para alimentar cada um deles, tornando a implementação paralela ineficiente, principalmente se tratando de aceleradores como o KNL ou uma GPU.

Note-se que o problema de obter uma inversa aproximada fatorada, para matrizes SP, resume-se a encontrar uma matriz Z triangular superior tal que AZ é triangular inferior, ou seja,

$$a_i^t z_j = 0, i < j, \tag{2}$$

pois seja Z tal que AZ é uma matriz triangular inferior, como Z^T é triangular inferior, teremos que $Z^T AZ$ é uma matriz triangular inferior. Ao mesmo tempo, teremos que $Z^T A$ é triangular superior, e portanto como Z é triangular superior $Z^T AZ$ além de triangular inferior, será também triangular superior, assim temos que $Z^T AZ = D$, onde D é uma matriz diagonal.

A propriedade (2) é obtida pelo Algoritmo 1 através do processo de A-ortogonalização que cria, aparentemente, uma relação de dependência entre as colunas de Z durante sua construção. No entanto, com um olhar mais atento, podemos perceber que essa propriedade não impõe nenhuma dependência entre as colunas de Z. Neste trabalho, exploramos esta independência para efetuar a construção paralela de um precondicionador baseado na aproximação da inversa.

Passamos a descrever um algoritmo iterativo para obter uma matriz Z triangular superior esparsa com diagonal unitária. Neste caso, uma escolha natural de ponto inicial para a coluna z_j é a j-ésima coluna da identidade, e_j, pois é triangular superior e, dada a esparsidade de A, a propriedade (2) será violada um numero reduzido de vezes. No caso, geral, apesar da esparsidade de e_j e de A, existirão algumas colunas de A que não serão ortogonais a $z_j = e_j$. Seja C a matriz composta por estas colunas. Para satisfazer a propriedade (2) com respeito as colunas de C, basta projetar z_j no núcleo de C.

No entanto, com este procedimento, podemos perder a ortogonalidade com respeito a outras colunas de A. Uma alternativa é incrementar iterativamente a matriz C acrescentando a cada iteração as colunas de A que não satisfazem a propriedade (2) e realizar uma nova projeção de z_j. No pior cenário, em uma dada iteração, a matriz C seria composta por todas as colunas de A. Este algoritmo já seria massivamente paralelo pois cada coluna de Z seria calculada independentemente.

DOI: 10.5540/03.2018.006.02.0312

Infelizmente, esta estratégia seria extremamente ineficiente, pois o cálculo da projeção no núcleo do espaço requer uma representação do espaço gerado pelas colunas de C. Neste algoritmo, para atenuar este custo computacional, propomos três estrategias (1) manter a esparsidade de z_j ao longo das iterações o que garante a ortogonalidade simbólica de z_j com respeito a maioria das colunas de A; (2) diminuir o custo da projeção, reduzindo o número de colunas em C; (3) mitigar a perda de ortogonalidade devido à projeção no núcleo de C reduzida, minimizando o ângulo entre a nova aproximação z_j e a anterior.

A primeira estratégia deriva do algoritmo do precondicionador AINV, onde é feito o descarte de valores abaixo de uma dada tolerância após a projeção de Z no núcleo de C. Com isso preservamos a esparsidade de Z e reduzimos o crescimento da matriz C, controlando o custo computacional das iterações do algoritmo paralelo. A segunda estratégia se trata de, a cada nova iteração para o cálculo de z_j, substituir C pelo conjunto das colunas de A que não são ortogonais à iteração atual de z_j, ao invés de incrementar C com novas colunas de A. Com isso, deixamos de levar em consideração as colunas que foram utilizadas para calcular z_j na iteração anterior, podendo perder assim a ortogonalidade com as colunas de A utilizadas anteriormente ao calcular a nova iteração. Para mitigar este efeito, ao projetarmos z_j no núcleo de C, realizamos uma projeção ortogonal. Deste modo, o ângulo entre z_j e a sua projeção será o menor possível. A ideia é tentar preservar parcialmente os efeitos da ortogonalidade obtida na iteração anterior através da minimização do ângulo entre as aproximações. Calculamos a projeção ortogonal $Proj(z_j, kernel(C))$ que pode ser escrita como $(I - C(CC^t)^{-1}C)z_j$.

O Algoritmo 2 descreve a construção paralela do precondicionador de inversa aproximada fatorada. Todas as colunas de Z são calculadas paralelamente no laço externo (linha 4). Na linha 6, obtém-se S, o conjunto dos índices das colunas de A que não satisfazem a propriedade (2). A matriz C na linha 7 é formada pelas colunas de A com índices em S. Na linha 8 é feita a projeção ortogonal de z_j no núcleo de C. O laço iniciado na linha 5 define a região iterativa do algoritmo, e pode possuir diferentes critérios de parada. Na implementação utilizada neste trabalho, limitamos o número de iterações e interrompemos o laço caso z_j possua elementos não nulos exatamente nas mesmas posições da iteração anterior. A linha 11 do algoritmo 2 calcula a j-ésima entrada da matriz D.

```
1  [Z,D] = parainv(A)
2    n = length(A)
3    Z = eye(n)
4    for j = 2:n   % Paralelo
5      while !done
6        S = {i: i < j & |A(i,:)*Z(:,j)| > 0}
7        C = A(1:j,S)
8        Z(:,j) = Proj(Z(:,j),nucleo(C))  % Projeção ortogonal de Z(:,j) no núcleo de C
9        Z(:,j) = drop(Z(:,j))  % Descarte
10       done = criterio_de_parada()
11     D(j,j) = A(j,:) * Z(:,j)
```

Algoritmo 2: Algoritmo de fatoração inversa aproximada paralelo

DOI: 10.5540/03.2018.006.02.0312

4 Detalhes da implementação

Para os experimentos, realizamos a implementação da construção do precondicionador na linguagem C, utilizando formato *Compressed Sparse Column* (CSC) para matrizes esparsas, com paralelismo em memória compartilhada, obtido através de OpenMP (linha 4 do Algoritmo 2). A matriz C, obtida na linha 7, é reduzida removendo as linhas nulas, consequência da esparsidade das colunas de A, e é armazenada em sua forma densa na memória, reduzindo assim as indireções causadas pela estrutura esparsa. Para realizar a projeção ortogonal (linha 8), utilizamos uma fatoração QR densa, de modo que se $C = QR$, rescrevemos $(I - C(CC^t)^{-1}C)z_j$ como $(I - CR^{-1}R^{-T}C^T)z_j$, onde resolvemos os sistemas triangulares com métodos diretos. As colunas de Z são armazenadas separadamente e ao final é construída a matriz esparsa no formato CSC. Este último passo, apesar de poder contar com paralelismo, não foi implementado de forma paralela para a versão utilizada nos experimentos.

5 Experimentos Numéricos

A seguir apresentamos alguns experimentos que ilustram o comportamento do precondicionador de inversa aproximada proposto para uma matriz obtida através da discretização, por diferenças finitas, do operador de convecção-difusão. O operador sendo discretizado é $L(u) = \nabla \cdot (u\vec{v}) - \epsilon \Delta u$, com condições de fronteira de Dirichlet homogêneas. O domínio se trata de um cubo unitário com $N \times N \times N$ elementos iguais, resultando em um estêncil de 7 pontos. Os testes foram realizados em um processador Intel® Xeon Phi™ 7250F (KNL) com 68 núcleos de 1.4GHz. Vamos analisar o precondicionador quando aplicado junto ao solver gradiente conjugado precondicionado (PCG) [4] e analisar a escalabilidade e tempo de construção em paralelo.

O critério de parada utilizado no solver foi a redução do resíduo relativo à 10^{-9}. O problema apresentado foi resolvido pelo gradiente conjugado (não precondicionado) com 768 iterações. O primeiro conjunto de testes (Tabela 1) compara o tempo da construção paralela (em segundos) do PARAINV variando o numero de iterações máximo (Maxit) no cálculo das colunas de Z para avaliar o impacto no tempo de construção e na qualidade do precondicionador. Para este teste foram utilizados 64 núcleos de processamento.

Tabela 1: Variação do número máximo de iterações da construção do precondicionador.

Maxit	Construção (s)	Solver (s)	Total (s)	Iterações
1	1,44	1,03	2,47	234
2	2,18	1,00	3,18	222
3	2,23	1,00	3,23	222
4	2,25	1,00	3,25	222

Como pode ser visto na Tabela 1, para o problema utilizado, o aumento do número de iterações gera uma redução pouco relevante no número de iterações do PCG, com um impacto considerável no tempo de construção. Deste modo o tempo total aumenta,

DOI: 10.5540/03.2018.006.02.0312

mesmo com a redução no número de iterações. Apesar do aumento de mais de 50% no tempo de construção ao passar do limite de uma iteração, para duas iterações, o tempo de construção passando para três ou quatro iterações da construção não aumenta, assim como a quantidade de iterações do PCG não se altera. A razão principal é o segundo critério de parada da construção das colunas, que interrompe o laço quando z_j possui elementos não nulos exatamente nas mesmas posições da iteração anterior. Como, para este problema o resultado mais satisfatório foi obtido com apenas uma iteração, utilizamos este parâmetro para os próximos testes.

Na Tabela 2 apresentamos os resultados para construção e solução do problema utilizando os precondicionadores PARAINV e AINV com diferentes números de núcleos de processamento, para avaliar a escalabilidade e comparar a qualidade do precondicionador gerado. Utilizamos uma implementação própria do AINV, utilizado em [2] que implementa estruturas de dados eficientes para lidar com as operações esparsas, deixando o custo do algoritmo linear em relação ao crescimento do problema.

Tabela 2: Teste de escalabilidade e comparação com o precondicionador AINV

Núcleos	Precond	Construção (s)	Solver (s)	Total (s)	Iterações
1	PARAINV	37,31	31,60	68,91	234
	AINV	7,54	30,51	38,05	227
2	PARAINV	18,42	15,91	34,33	234
	AINV	7,54	15,44	22,98	227
4	PARAINV	9,32	8,01	17,33	234
	AINV	7,54	7,78	15,32	227
8	PARAINV	4,76	4,09	8,85	234
	AINV	7,54	3,98	11,52	227
16	PARAINV	2,49	2,16	4,65	234
	AINV	7,54	2,10	9,64	227
32	PARAINV	1,83	1,20	3,03	234
	AINV	7,54	1,16	8,70	227
64	PARAINV	1,44	1,03	2,47	234
	AINV	7,54	1,00	8,54	227

Podemos observar que para este problema, o número de iterações com os dois precondicionadores é semelhante, cerca de 3% de diferença, indicando que apesar da diferença no método de construção, o precondicionador proposto preserva os mesmos aspectos da inversa aproximada que o AINV para esse conjunto de testes. Como o número de iterações é semelhante, o que determina qual resolve o problema mais rapidamente, dado que o preenchimento de ambos foi o mesmo, é o tempo de construção. O número de operações necessárias para calcular o PARAINV é maior, porém ao aumentar o número de núcleos podemos ver que rapidamente a vantagem do cálculo paralelo das colunas do precondicio-

DOI: 10.5540/03.2018.006.02.0312

nador fica clara. A partir de oito núcleos há um ganho de 30% no tempo total de solução do problema em relação ao AINV, alcançando um ganho de até 345% com 64 núcleos.

Quanto à escalabilidade, notamos um crescimento próximo ao linear de 1 até 16 núcleos, onde atinge uma aceleração de aproximadamente 15 vezes. De 16 para 32 e para 64 núcleos, observamos que a escalabilidade se deteriora atingindo uma aceleração máxima de 25 vezes com 64 núcleos. Uma das razões é devido ao fato da montagem da matriz Z após o cálculo das colunas, apesar de possível, não foi paralelizado até o momento destes experimentos, e o impacto deste passo não paralelo começa a se tornar relevante com o aumento do número de núcleos.

6 Conclusões

O precondicionador proposto apresentou uma redução no número de iterações na solução do sistema linear para o problema escolhido. Esta redução se mostrou em nível competitivo com o precondicionador AINV, outro precondicionador baseado na aproximação da inversa fatorada, presente na literatura. A escalabilidade do algoritmo foi atestada nos experimentos realizados e, com isso, o método se mostrou superior ao AINV, considerando o tempo total de solução do problema em questão. Ainda assim, a escalabilidade não demonstrou todo o potencial esperado devido principalmente à falta de maturidade da implementação. O aumento de iterações na construção do precondicionador para refinar sua qualidade não se mostrou favorável, pois o aumento do custo de construção não resultou em redução suficiente de iterações do PCG, mas outras alternativas para o refinamento serão avaliadas em trabalhos futuros. Apesar de superar o precondicionador AINV em um ambiente paralelo, acreditamos que o tempo de construção ainda se mostra muito elevado e iremos buscar alternativas que reduzam o seu custo computacional. Além disso, iremos aumentar o espectro de problemas testados e avaliar o aumento do custo da construção da matriz Z para matrizes com diferentes esparsidades. Pretendemos também estender este estudo para matrizes não-simétricas.

Referências

[1] Michele Benzi, Carl D Meyer, and Miroslav Tuma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM Journal on Scientific Computing*, 17(5):1135–1149, 1996.

[2] Luiz M P Carvalho, João P Zanardi, Ítalo Nievinski, and Michael Souza. An overview of approximate inverse methods. *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics*, 3(1), 2015.

[3] David S Kershaw. The incomplete Cholesky—conjugate gradient method for the iterative solution of systems of linear equations. *Journal of Computational Physics*, 26(1):43–65, 1978.

[4] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

