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

Implementação paralela de um precondicionador algébrico de dois níveis de decomposição de domínios baseado em ILU(k)

Douglas A. Augusto[1]
Fundação Oswaldo Cruz, Rio de Janeiro - RJ
Luiz M. Carvalho[2]
Instituto de Matemática e Estatística , UERJ, Rio de Janeiro - RJ
Paulo Goldfeld[3]
Instituto de Matemática, UFRJ, Rio de Janeiro - RJ
Ítalo C. L. Nievinski[4]
Faculdade de Engenharia Mecânica, PPGEM, UERJ, Rio de Janeiro - RJ
José R. P. Rodrigues[5]
CENPES, Petrobras, Rio de Janeiro - RJ
Michael Souza[6]
Departamento de Estatística e Matemática Aplicada, UFC, Fortaleza - CE

Resumo. Discutimos a implementação paralela em Message-Passing Interface (MPI) de um precondicionador algébrico de dois níveis de decomposição de domínios baseado em fatoração incompleta LU (ILU(k)) utilizando a biblioteca PETSc e estratégias para melhorar a performance e reduzir a comunicação entre os processadores durante a construção e aplicação.

Palavras-chave. precondicionador de dois níveis, decomposição de domínios, métodos de Krylov, sistemas lineares, paralelismo, PETSc.

1 Introdução

Este trabalho descreve a implementação paralela de um precondicionador algébrico de dois níveis de decomposição de domínios baseado em ILU(k) descrito em [1].

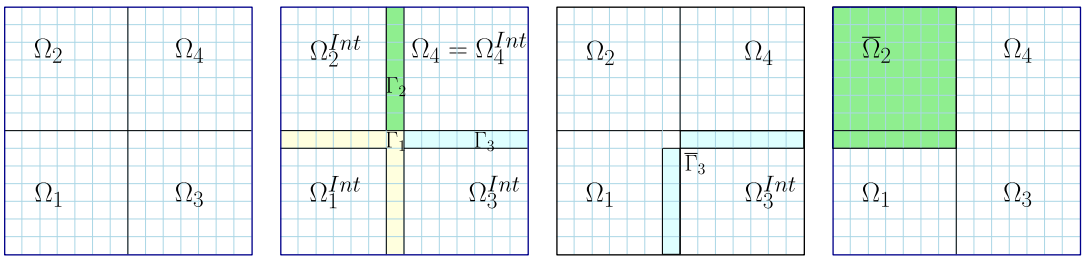
Neste trabalho apresentamos e discutimos detalhes da implementação utilizando MPI através da biblioteca PETSc [2], um conjunto de estruturas de dados e rotinas para soluções paralelas de aplicações científicas modeladas por equações diferenciais parciais. Apresentamos também resultados de experimentos computacionais envolvendo matrizes advindas

[1]daa@labma.ufrj.br
[2]luizmc@ime.uerj.br
[3]goldfeld@ufrj.br
[4]italonievinski@gmail.com
[5]jrprodrigues@petrobras.com.br
[6]michael@ufc.br

da simulação numérica de reservatórios de petróleo. Estes testes são realizados com diversos números de processos e os resultados obtidos são comparados com precondicionadores implementados no PETSc.

2 Implementação Paralela

Nesta seção discutiremos detalhes da implementação paralela do precondicionador, levando em consideração a localidade do dado, comunicação entre os processos e estratégias para evitar comunicação, que pode se tornar um gargalo importante no desempenho computacional de aplicações paralelas.



(a) Subdomínios disjuntos Ω_J.

(b) Interiores Ω_J^{Int} e interfaces locais Γ_J

(c) Interface estendida $\overline{\Gamma}_3$.

(d) Subdomínio estendido $\overline{\Omega}_2$.

Figura 1: Domínios 2D particionados em 4 subdomínios.

Para facilitar a compreensão da notação, a Figura 1 apresenta um exemplo de um domínio Ω decomposto em subdomínios Ω_J.

Inicialmente vale observar que a matriz A do problema é carregada no PETSc de forma distribuída, de modo que as linhas referentes a cada subdomínio estarão associadas a um processo. Os processos por sua vez são distribuídos entre os núcleos físicos do cluster ou máquina multi-core.

O algoritmo 1 apresenta a construção dos dois componentes do precondicionador, o componente fino e o grosso.

O passo 1 do Algoritmo 1 é a fatoração LU incompleta dos subdomínios. Este passo será realizado em cada processo paralelamente, nenhuma comunicação é necessária. O PETSc disponibiliza uma rotina própria para realizar a fatoração LU incompleta.

Os passos 2 e 3 constroem os fatores $\widetilde{B}_J$ e $\widetilde{C}_J$ através de solvers triangulares incompletos realizados também localmente. Sabemos que $A_{J\Gamma}$ só terá elementos não nulos nas colunas $\overline{\Gamma}_J$, podemos então construir $\widetilde{C}_J^{ext} = R_{\overline{\Gamma}_J}\widetilde{C}_J$, onde $R_{\overline{\Gamma}_J}$ é um operador de restrição tal que apenas as colunas associadas a $\overline{\Gamma}_J$ em $\widetilde{C}_J$ permanecem em $\widetilde{C}_J^{ext}$. Construímos $\widetilde{B}_J^{ext}$ de modo análogo. Assim ao tomar a interface estendida dos subdomínios, será necessária apenas uma comunicação de cada subdomínio com seus vizinhos, ao invés de uma comunicação com todas as interfaces, e portanto, com todos os processos.

O passo 4 constrói o complemento de Schur através de um produto incompleto como é descrito em [1]. A submatriz $A_{\Gamma\Gamma}$ é formada pelas interfaces dos subdomínios e portanto

DOI: 10.5540/03.2017.005.01.0101

Algoritmo 1 – Construção do precondicionador

Fine Component

1: $[\widetilde{L}_J, \widetilde{U}_J] = \mathrm{ILU}(A_{JJ}, k_{\mathrm{Int}}), \quad J = 1, \ldots, P;$

2: $\widetilde{C}_J = \mathrm{IFS}(\widetilde{L}_J, A_{J\Gamma}, k_{\mathrm{Border}}), \quad J = 1, \ldots, P;$

3: $\widetilde{B}_J = \left(\mathrm{IFS}(\widetilde{U}_J^T, A_{\Gamma J}^T, k_{\mathrm{Border}}) \right)^T, \quad J = 1, \ldots, P;$

4: $\widetilde{S} = A_{\Gamma\Gamma} - \sum_{J=1}^{P} \mathrm{IP}(\widetilde{B}_J, \widetilde{C}_J, k_{\mathrm{Prod}})$

5: $\widetilde{S}_J = R_J \widetilde{S} R_J^T, \quad J = 1, \ldots, P;$

6: $[L_{\widetilde{S}_J}, U_{\widetilde{S}_J}] = \mathrm{ILU}(\widetilde{S}_J, k_\Gamma), \quad J = 1, \ldots, P;$

Coarse Component

7. $[L_C, U_C] = \mathrm{LU}(R_0 A R_0^T).$

está distribuída entre os processos. Cada processo calcula localmente sua contribuição $\mathrm{IP}(\widetilde{B}_J, \widetilde{C}_J, k_{Prod})$ e subtrai globalmente de $A_{\Gamma\Gamma}$ para construir S, que estará distribuída entre os processos da mesma maneira que $A_{\Gamma\Gamma}$. Novamente cada subdomínio precisa realizar comunicação com seu vizinho, e há uma barreira de sincronização para formar a matriz S global.

Nos passos 5 e 6, cada processo toma uma cópia local da parte de $\widetilde{S}$ relativa à interface estendida $\overline{\Gamma}_J$ (comunicação com processos vizinhos) e realiza a fatoração LU incompleta. O uso da cópia local irá evitar comunicação durante a aplição paralela de $\widetilde{S}$ nas iterações do solver.

No passo 7, a matriz $R_0 A R_0^T$ é inicialmente montada como uma matriz paralela, distribuída entre processos (sua montagem envolve comunicação entre processos vizinhos). São feitas então cópias locais desta matriz em todos os processos e a fatoração LU é entao feita de maneira redundante. Convém lembrar que trata-se de uma matriz de dimensão $P \times P$.

A aplicação do precondicionador é apresentada no Algoritmo 2. O passo 1 resolve em cada processo um sistema linear triangular envolvendo o fator $\widetilde{L}_J$ e a porção local interior do resíduo, obtendo z_J. Este processo é realizado localmente e nenhuma comunicação é necessária pois o resíduo r é distribuído entre os processos seguindo o layout paralelo da matriz do reservatório.

O passo 2 aplica $\widetilde{B}_J$ localmente multiplicando z_J subtraindo de r_Γ obtendo z_Γ. Esse passo é realizado localmente para cada processo, e é necessária a comunicação entre os subdomínios em r_Γ e z_Γ.

A aplicação do complemento de Schur é realizada no passo 3. Como cada processo possui uma cópia local da sua parte de S fatorada em L_S e U_S, esta aplicação é feita

DOI: 10.5540/03.2017.005.01.0101

Algoritmo 2 – Aplicação do precondicionador

Aplicação do precondicionador M^{-1} ao vetor r, i.e., $z = M^{-1}r$. Definimos dois operadores de restrição adicionais, $R_{\Omega_J} : \Omega \to \Omega_J^{\text{Int}}$ e $R_\Gamma : \Omega \to \Gamma$ e usamos a notação (inspirada em Matlab) $A \setminus b$ para representar a solução exata de $Ax = b$.

1: $z_J = \widetilde{L}_J \setminus (R_{\Omega_J} r), \quad J = 1, \ldots, P;$

2: $z_\Gamma = R_\Gamma r - \sum_{J=1}^{P} \widetilde{B}_J z_J$

3: $z_\Gamma = \sum_{J=1}^{P} T_J \left(U_{\widetilde{S}_J} \setminus \left(L_{\widetilde{S}_J} \setminus (R_J z_\Gamma) \right) \right);$

4: $z_J = \widetilde{U}_J \setminus \left(z_J - \widetilde{C}_J z_\Gamma \right), \quad J = 1, \ldots, P;$

5: $z_F = R_\Gamma^T z_\Gamma + \sum_{J=1}^{P} R_{\Omega_J}^T z_J$

Coarse correction:

6. $z = z_F + R_0^T \left(U_C \setminus (L_C \setminus R_0 r) \right)$.

sem comunicação na matriz, mas possui comunicação em z_Γ entre cada subdomínio e seus vizinhos. Conforme descrito em [1], T_J é um operador diagonal, assim podemos construí-lo como um vetor, qué é aplicado através do produto entrada a entrada. O resultado é então somado globalmente em z_Γ. Nesse ponto há uma barreira de sincronização entre os processos.

No passo 4 aplicamos o fator U dos interiores em $z_J - \widetilde{C}_J z_\Gamma$, que é um poduto realizado localmente em cada processo, com comunicação em z_Γ. O resultado é armazenado em z_J.

Os vetores z_J e o vetor z_Γ são reunidos em z_F no passo 5, onde z_F é o resíduo precondicionado pelo componente fino.

O passo 6 irá aplicar o componente grosso finalizando o precondicionamento do resíduo. Os solvers triangulares envolvidos são locais (redundantes). Apenas comunicação entre vizinhos é necessária, nas aplicações de R_0 e R_0^T.

3 Testes

Iremos nos referir ao componente fino do precondicionador aqui descrito como *iSchur*, e ao componente grosso como *Coarse*. Para avaliar a performance do precondicionador foram realizados testes onde consideramos o número de iterações e o tempo de solução. Nesta seção iremos descrever a plataforma onde foram realizados os testes, apresentar os resultados do iSchur e também comparar com o precondicionador Jacobi em blocos, nativo do PETSc.

DOI: 10.5540/03.2017.005.01.0101

Utilizamos duas matrizes provenientes de um simulador de reservatórios. Tratam-se de sistemas lineares a serem resolvidos no método de Newton para a solução das equações não lineares advindas da discretização do sistema de EDPs que modelam o escoamento multifásico em meios porosos. O Caso 1 foi gerado a partir de um reservatório da Petrobras e possui 1378899 células ativas. O Caso 2 se trata do modelo SPE10 [4] com 1094421 células ativas. Ambos são modelos *black oil*.

Para ambos os precondicionadores o problema foi resolvido utilizando o método GMRES [5] nativo do PETSc, com tolerância 1e-4 e restart 30.

As partições foram construídas utilizando o particionador PT-Scotch [3].

O iSchur terá nível de preenchimento 1 nos interiores dos subdomínios e zero nas interfaces, ou seja, $k_{Int} = 0$, $k_{Boarder} = 0$ e $k_{Prod} = 0$. Utilizamos para comparação o precondicionador Jacobi por blocos, nativo da biblioteca PETSc, com ILU(1) nos blocos.

Os testes foram realizados em uma máquina com processador Intel Xeon(R) CPU E5-2650 v2 @ 2.60GHz, 16 cores (hyper-threading desativada), 20MB LLC, 64GB RAM (ECC ligada). Sistema operacional Linux gnu 3.2.0-4-amd64 #1 SMP Debian 3.2.65-1 x86 64 GNU/Linux. Intel Parallel Studio XE 2015.3.187 Cluster Edition.

Tabela 1: Número de iterações para os Casos 1 e 2.

	Caso 1				Caso 2			
	Bloco Jacobi		iSchur		Bloco Jacobi		iSchur	
Procs	Coarse	—	Coarse	—	Coarse	—	Coarse	—
2	88	108	88	117	73	74	73	74
4	86	109	82	111	74	73	72	73
8	93	131	87	145	77	75	71	74
16	129	130	102	140	84	80	72	77
32	146	263	87	157	84	79	74	77
64	113	296	129	319	87	85	89	84
128	128	315	134	320	85	83	79	86

A tabela 1 apresenta o número de iterações para os Casos 1 e 2 com os precondicionadores iSchur e Jacobi em blocos, ambos com e sem o componente grosso (Coarse). É importante porém compreender o comportamento dos precondicionadores quando número de partições aumenta, por isso foram realizados testes até 128 processos. Os tempos no entanto só foram medidos para os testes até 16 processos, de forma a não termos mais de um processo competindo pelo mesmo núcleo. Os resultados se encontram na tabela 2.

A figura 2 apresenta um gráfico que mostra a escalabilidade dos precondicionadores.

4 Conclusões e trabalho futuro

Apesar do componente fino *iSchur* não apresentar vantagem em relação ao Jacobi em blocos no número de iterações, ao aplicar o componente grosso podemos perceber ganhos significativos, principalmente quando o número de subdomínios cresce. Podemos ver que

Tabela 2: Tempo total do solver linear para os Casos 1 e 2.

	Caso 1				Caso 2			
	Block Jacobi		iSchur		Block Jacobi		iSchur	
Procs	Coarse	—	Coarse	—	Coarse	—	Coarse	—
2	50.75	50.88	66.07	71.56	28.54	24.16	42.09	36.25
4	27.95	39.17	40.11	46.69	16.25	14.44	27.77	21.37
8	15.51	19.31	18.51	19.23	9.11	6.51	11.21	10.04
16	10.57	11.34	14.25	15.05	7.69	5.08	9.56	7.82

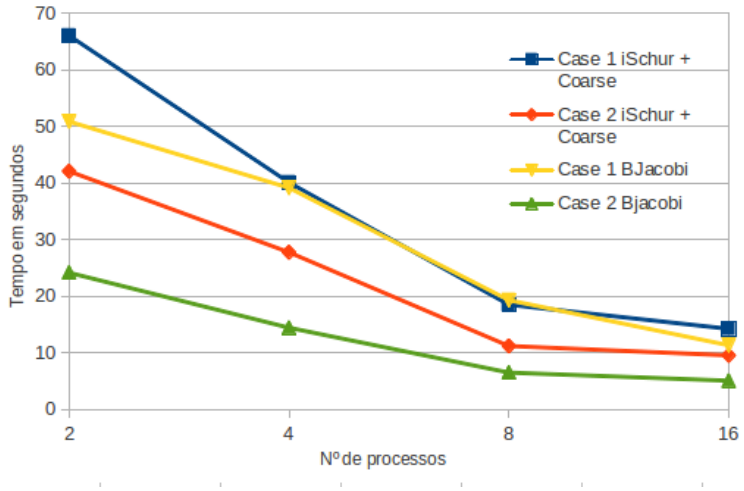


Figura 2: Comparação de escalabilidade entre iSchur+Coarse e Jacobi em blocos

o Jacobi em blocos também se beneficia do componente grosso assim como o iSchur, os ganhos porém se mostram menores.

Apesar do ganho em iterações, esses ganhos não são integralmente transportados para o tempo do solver linear, pois o custo por iteração é maior no iSchur e Coarse. É importante ressaltar que o Jacobi por blocos é uma implementação nativa do PETSc, um código maduro e otimizado. Apesar do cuidado na implementação acreditamos que o código ainda pode possuir diferentes oportunidades de otimização, o que será buscado no futuro.

Referências

[1] D. A. Augusto, L. M. Carvalho, P. Goldfeld, I. C. L. Nievinski, J. R. P. Rodgrigues, and M Souza. An algebraic ILU(k) based two-level domain decomposition preconditioner. In *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics*, volume 1, 2015. DOI:10.5540/03.2015.003.01.0093.

[2] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, Ruppm K., B. S. Smith, S. Zampini, and H. Zhang. PETSc Web page. http://www.mcs.anl.gov/petsc, 2015.

[3] C. Chevalier and F. Pellegrini. PT-Scotch: A tool for efficient parallel graph ordering. *Parallel computing*, 34(6):318–331, 2008.

[4] M. Christie, M. Blunt, et al. Tenth SPE comparative solution project: A comparison of upscaling techniques. In *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers, 2001.

[5] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

DOI: 10.5540/03.2017.005.01.0101