Implementação de Bibliotecas de Álgebra Linear Numérica via Interface Nativa no INSANE

Marcella Passos Andrade

Ramon Pereira da Silva

marcella.pandrade@hotmail.com

ramon@dees.ufmg.br

Universidade Federal de Minas Gerais

Av. Presidente Antônio Carlos, 6627, Pampulha, 31270-901, Belo Horizonte - MG, Brasil.

Abstract. *A simulação numérica de meios parcialmente frágeis torna necessária uma análise fisicamente não-linear. À medida que a complexidade dos problemas abordados evolui, aumenta-se a demanda computacional para a solução do sistema linear de equações do modelo discreto resultante, seja em termos de consumo de memória quanto de uso de CPU. Nessa análise, a matriz de rigidez perde a positividade à medida que ocorre as iterações, em particular quando da mudança da trajetória de equilíbrio do ramo ascendente para o descendente. Ainda há certos modelos constitutivos que podem levar também à perda da simetria numérica da matriz de rigidez. Há diversas bibliotecas de álgebra linear numérica, com uso acadêmico ou comercial difundido, comprovadamente rápidas, tais como* `BLAS`*,* `Lapack`*,* `SuperLU`*,* `SuiteSparse` *e* `libFlame`*. O* ***INSANE*** *(Interactive Structural Analysis Environment) é um software livre, em desenvolvimento no DEES-UFMG, escrito em* `Java`*, no qual estão implementados o Método dos Elementos Finitos (MEF), o Método de Elementos de Contorno (MEC), bem como um método sem malha, o Element-Free Galerkin (EFG). Atualmente o armazenamento da matriz de rigidez dá-se em formato cheio ou simétrico esparso com altura variável de colunas. Este trabalho objetiva estender o núcleo numérico do sistema considerando-se o armazenamento consistente com as bibliotecas* `SuiteSparse` *e* `libFlame`*, bem como disponibilizar um algoritmo de solução direta de sistema de equações lineares fornecidos pelas respectivas bibliotecas, utilizando-se dos mecanismos da interface nativa de* `Java`*. A síntese da implementação e uma comparação quanto ao uso de memória e tempo de processamento para um conjunto de matrizes de tamanhos variados será apresentada.*

Keywords: *Interface Nativa Java, JNI, Análise não linear, Álgebra Linear Numérica, Método dos Elementos Finitos*

1 INTRODUÇÃO

O **INSANE** (Interactive Structural Analysis Environment) é uma plataforma computacional para análise estrutural, Pitangueira et al. (2008), originalmente concebida para utilizar o Método os Elementos Finitos (MEF). Trata-se de um *software* livre, escrito em `Java`, e dividido em módulos de pré-processamento, processamento e pós-processamento.

Para minimizar o consumo de memória e ao mesmo tempo agilizar a solução de sistemas de equações lineares, foi implementado o armazenamento com altura de coluna variável da parte simétrica da matriz de rigidez, *skyline* (Bathe (1996)), após renumeração nodal utilizando o método Reverse Cuthill-McKee (George e Liu (1981)), assumindo-se uma matriz definida positiva.

Todavia, problemas de ordem prática ocorrem em uma análise fisicamente não linear. Nesta circunstância, a matriz de rigidez perde a positividade à medida que as iterações avançam, sobretudo no limiar da passagem do ramo ascendente para o descendente da trajetória de equilíbrio. Há, ainda, que considerar que alguns modelos constitutivos disponíveis no **INSANE** a matriz de rigidez perde a simetria numérica.

Para contornar tais demandas computacionais, optou-se por fazer uso de bibliotecas de álgebra linear numérica existentes, sabidamente testadas e extremamente eficientes em termos uso de memória e velocidade, a saber, `SuiteSparse` (Davis (2004)) e `libFlame` (Van Zee e van de Geijn (2015)).

Estas bibliotecas são escritas em linguagem `C`. O acesso às mesmas a partir de uma aplicação `Java` é realizado através da interface nativa de `Java` (*Java Native Interface* – JNI), a qual oferece mecanismos para realizar a transferência de dados de/para `Java`, bem como provê regras para adaptar nome e assinatura de métodos nativos chamados a partir de uma aplicação `Java`.

Nesse trabalho abordaremos na seção (2) as bibliotecas já implementadas no ambiente computacional **INSANE**. Na seção (3) mostram-se os resultados preliminares obtidos nesta fase da implementação, considerando-se a capacidade de computador de mesa, para realizar análise elástico-linear em alguns problemas estruturais de variados tamanhos. Os ganhos de tempo e economia de memória relativos à solução em matriz cheia pelo método de Crout implementado em `Java`. Na seção (4) encontra-se a conclusão.

2 Bibliotecas de Álgebra Linear

Há diversas bibliotecas de álgebra linear numérica, com uso acadêmico ou comercial difundido, comprovadamente rápidas e disponíveis, escritas em `C` ou `Fortran`, tais como `BLAS`, `Lapack` (Anderson et al. (1999)), `SuperLU` (Li et al. (1999)), `SuiteSparse` e `libFlame`.

As duas últimas bibliotecas foram utilizadas nesta implementação, e são brevemente descritas abaixo.

`libFlame`: Desenvolvida na Universidade do Texas, Van Zee e van de Geijn (2015), é uma biblioteca de alta performance para álgebra linear densa. Sua metodologia se diferencia na abordagem pois, além de realizar operações matriciais em blocos, também organiza a sequência de processamento das operações administrando o fluxo de dados entre a memória da unidade de

processamento de ponto flutuante, de baixa latência, até a memória RAM, de maior quantidade e grande latência, conforme descrito, por exemplo, em Goto e van de Geijn (2008).

`SuiteSparse`: Conjunto de rotinas extremanente velozes para a solução direta de sistemas de equações lineares esparsos, Davis (2004). Esta biblioteca é utilizada no sistema `MATLAB`, entre outros.

Por sua vez, as referidas bibliotecas dependem de outras bibliotecas clássicas, em suas versões comerciais ou de *software* livre, quais sejam `BLAS` e `Lapack`.

Cada uma das bibliotecas usadas armazena uma matriz retangular real de m linhas e n colunas em formato distinto. No **INSANE** esta matriz é representada pela classe `IMatrix`. Em `Java` esta matriz é armazenada linha por linha A biblioteca `libFlame` armazena esta mesma matriz no formato usado em `Fortran`, i.e., coluna por coluna. Por sua vez, na biblioteca `SuiteSparse` uma matriz esparsa armazena apenas os valores não nulos coluna por coluna, utilizando uma estrutura de dados denominada CCS (*compressed column storage*), Davis (2006). Para cada formato de armazenamento, uma classe `Java` derivada de `IMatrix` foi desenvolvida.

3 RESULTADOS

Uma descrição sucinta de alguns exemplos utilizados para validar a implementação são apresentados a seguir.

1. Chapa plana de 8m de largura por 6m de altura, com furo em seu centro, em estado plano de tensões (EPT) e submetida a um carregamento de tração. Discretização: 9307 elementos finitos T3 e 4755 nós;
2. Chapa em L engastada na base e carga vertical na extremidade livre em EPT. Discretização: 8613 elementos finitos T3 e 4403 nós;
3. A mesma geometria anterior, todavia considerando-se uma placa de Reissner-Mindlin. Discretização: 25518 elementos T3 e 12933 nós;
4. Malha teste do gerador de malhas do **INSANE**. EPT em tração em uma das extremidades. Discretização: 9132 elementos finitos T3 e 5058 nós;
5. Mesma malha, entretanto simulando uma viga bi-apoiada;
6. Viga em balanço. Discretização: 20000 elementos finitos Q4 e 20301 nós.

Na Tab. 1 encontram-se as comparações relativas ao armazenamento cheio da matriz de rigidez. Nesta tabela, neq é o número de equações, nnz o número de elementos diferentes de zero armazenados, sem considerar simetria, e nwk o número de entradas no armazenamento de altura de coluna variável da parte simétrica da matriz de rigidez. Os valores em percentuais consideram o número de bytes utilizados pela estrutura de dados de cada formato, relativo ao número de bytes utilizados para armazenar a matriz cheia. A implementação utilizou números reais do tipo `double` ocupando 8 bytes e números inteiros do tipo `int`, de 4 bytes, usados pelos apontadores.

Na Tab. 2 listam-se o tempo de solução relativo de cada tipo de armazenamento e respectiva biblioteca comparado ao tempo de solução do sistema de equações utilizando o método de Crout escrito em `Java` e armazenamento cheio. Os resultados apresentados foram obtidos em

Table 1: Comparação relativa do armazenamento da estrutura de dados

Exemplo	neq	nnz	nwk	Flame (%)	SuiteSparse (%)	Skyline (%)
1	9.478	130.984	1.537.799	100	0,224	1,717
2	8.774	121.245	1.084.517	100	0,242	1,415
3	38.646	804.420	15.476.286	100	0,082	1,038
4	10.113	133.757	788.067	100	0,200	0,776
5	10.071	133.051	783.226	100	0,203	0,777
6	40.400	719.992	9.534.396	100	0,067	0,585

um MacBook Pro de 4GB de memória RAM e processador Intel Core i7. Os exemplos 3 e 6 não puderam ser executados para matriz cheia no hardware utilizado devido à pouca memória RAM disponível. Portanto, o tempo relativo não será mostrado. Ressalta-se que foi possível, entretanto, a execução usando-se a biblioteca `SuiteSparse`.

Table 2: Comparação relativa dos tempos de processamento

Exemplo	Flame(%)	SuiteSparse (%)	Skyline (%)
1	2.747	0.017	0.083
2	5.187	0.022	0.057
4	4.025	0.009	0.042
5	5.033	0.018	0.066

4 DISCUSSÃO E CONCLUSÃO

Os exemplos ilustrativos demonstram claramente a vantagem da utilização de armazenamento esparso ou por altura variável de colunas da parte simétrica da matriz de rigidez. Todavia, esta última opção impede a análise a problemas fisicamente não lineares, devido à perda da positividade da matriz de rigidez. Nota-se que, mesmo armazenando a matriz esparsa sem considerar a simetria da matriz de rigidez, o espaço utilizado é muito inferior ao utilizado no armazenamento skyline.

Com relação à velocidade de processamento, a solução de sistemas usando o método Crout implementado em `Java` tem performance muito inferior àquela obtida tanto pelo método LDL^T associado ao armazenamento skyline, quanto aqueles obtidos usando-se as bibliotecas nativas, utilizando o método LU. Nota-se, também que, considerando-se armazenamento cheio da matriz de rigidez, a biblioteca `libFlame` gastou apenas 5% do tempo gasto por Crout.

Chama a atenção a extrema velocidade da biblioteca `SuiteSparse`. Considerando-se a quantidade de memória utilizada e a velocidade de processamento, é fácil considerar a mesma a melhor escolha.

Apresentou-se a implementação de métodos nativos para a solução de sistemas de equações lineares oriundas de discretização pelo MEF no **INSANE** utilizando-se a interface nativa de `Java` (JNI). Este mecanismo mostrou ser de extrema importância.

Um dos próximos passos deste projeto será a implementação das operações matriciais utilizando-se métodos nativos, com o objetivo de melhorar a performance do sistema **INSANE** em operações que demandam operações matricias tipo vetor-vetor, matriz-vetor e matriz-matriz.

AGRADECIMENTOS

Os autores agradecem pela bolsa de iniciação científica ao CNPq no projeto 486.959/2013-9, e apoio através dos projetos 309.005/2013-2 e 308.785/2014-2, bem como à FAPEMIG através do projeto PPM-00669-15.

REFERÊNCIAS

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., e Sorensen, D. (1999). *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 3a edição.

Bathe, K. J. (1996). *Finite Element Procedures*. Prence-Hall.

Davis, T. A. (2004). A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30(2):165–195.

Davis, T. A. (2006). *Direct Methods for Sparse Linear Systems*. SIAM.

George, A. e Liu, J. W. (1981). *Computer Solution of Large Sparse Positive Definite*. Prentice Hall Professional Technical Reference.

Goto, K. e van de Geijn, R. (2008). High-performance implementation of the level-3 BLAS. *ACM Trans. Math. Softw.*, 35(1):4:1–4:14.

Li, X., Demmel, J., Gilbert, J., iL. Grigori, Shao, M., e Yamazaki, I. (1999). SuperLU Users' Guide. Relatório Técnico LBNL-44289, Lawrence Berkeley National Laboratory. Last update: August 2011.

Pitangueira, R. L. S., Fonseca, F. T., Fuina, J. S., Camara, L., Ferreira, R. L., Moreira, R. N., Penna, S. S., Saliba, S. S., e Fonseca, M. T. (2008). Insane – versão 2.0. In *Anais do XXVII Latin American Congress on Computational Methods*.

Van Zee, F. G. e van de Geijn, R. A. (2015). BLIS: A framework for rapidly instantiating BLAS functionality. *ACM Trans. Math. Softw.*, 41(3):14:1–14:33.