Blucher Mechanical Engineering Proceedings
May 2014, vol. 1 , num. 1
www.proceedings.blucher.com.br/evento/10wccm

10th World Congress on Computational Mechanics
8-13 July 2012 • São Paulo • Brazil

PERFORMANCE ANALYSIS OF PARALLEL SOFTWARE FOR SOLVING SYSTEMS OF LINEAR EQUATIONS

A. B. Ferreira[1], P. M. Pimenta[2], O. A. Marques[3]

[1]Department of Structural and Geotechnical Engineering, Polytechnic School at the University of São Paulo (higuita@usp.br)

[2] Department of Structural and Geotechnical Engineering, Polytechnic School at the University of São Paulo

[3]Lawrence Berkeley National Laboratory

Abstract. *This paper aims to present and analyze the performance of various software solving systems of linear equations. Such systems can be seen as dense or sparse matrices. For software like SuperLU there is a need to address the problem at hand in a special way, developing an algorithm to parallel systems that make use of distributed memory and need to work with a specific interface, such as MPI. For software like Pardiso there is no need for data parallel approach of the algorithm, because the software takes care of itself, making the parallelism of areas where this situation is advantageous, but it is not possible to verify what is actually happening with the code in terms of the division of tasks and resources.*

Keywords: *Parallel Solvers, SuperLU, Sparse Matrix.*

1. INTRODUÇÃO

O presente trabalho tem como objetivo dar início a apresentação e análise de performance de diversos softwares de resolução de sistemas de equações lineares. Tais sistemas podem ser vistos como matrizes densas ou esparsas.

O grupo dos softwares utilizados para resolução de sistemas densos é composto por: *Lapack*, *ScaLapack*, *Magma* e *Plasma*.

O grupo dos softwares utilizados para resolução de sistemas esparsos é composto por: *Pardiso*, *Mumps*, *SuperLU* e *Spike*.

Para softwares como, por exemplo, *SuperLU* existe a necessidade de abordar o problema em questão de forma especial, desenvolvendo um algoritmo para sistemas paralelos que fazem uso de memória distribuída e precisam de uma interface específica para funcionarem, como por exemplo o *MPI*.

No caso de softwares como, por exemplo, *Pardiso* não existe a necessidade da abordagem paralela dos dados do algoritmo, pois o software encarrega-se de fazer por si só o parale-

lismo de áreas em que tal situação seja vantajosa, porém não é possível verificar o que realmente está ocorrendo com o código em termos da divisão das tarefas e dos recursos.

Serão utilizadas matrizes de diversos problemas da engenharia, sendo essas matrizes de ordem 500.000 até 1.500.000, rodando em um supercomputador chamador de *Hopper* (*Cray XE6*), com 153.408 núcleos, com capacidade de 1.054.000 *Gigaflops* (bilhões de operações por segundo), sendo considerado atualmente o oitavo computador mais rápido do mundo.

Até a conclusão deste artigo, apenas os testes com o software *SuperLU* foram realizados e os resultados serão apresentado no decorrer do texto.

2. RESOLUÇÃO DE SISTEMAS DE EQUAÇÕES LINEARES

A resolução de sistemas de equações lineares pode ser dividida em dois grupos básicos: métodos diretos e métodos iterativos. Nos métodos diretos pode-se determinar previamente a quantidade de operações matemáticas necessárias para obter-se a resolução do sistema antes de resolvê-lo.

Entre os métodos diretos, pode-se citar o método da eliminação de *Gauss*, além dos métodos onde existe a decomposição da matriz dos coeficientes por um produto de matrizes. Este último caso tem como exemplo os métodos de *Crout*, *Doolite* e *Cholesky*.

No caso dos métodos iterativos são baseados em processos de iterações, calculando uma sequência de aproximações da solução do sistema linear que se pretende resolver, condicionando-o a uma aproximação inicial e sujeito a uma tolerância estipulada inicialmente.

Entre os métodos iterativos pode-se citar *Gauss-Seidel*, *Gauss-Jacobi*, gradientes conjugados e ainda gradientes biconjugados [1] [5].

2.1. Métodos Diretos – Decomposição LU

O sistema de equações lineares, descrito como uma matriz, é tido como:

$$\mathbf{Ax} = \mathbf{b}\,, \tag{1}$$

onde, $\mathbf{A}$ é a matriz dos coeficientes das equações algébricas, $\mathbf{x}$ é o vetor das incógnitas e $\mathbf{b}$ é o vetor dos termos independentes.

Feita a decomposição multiplicatia da $\mathbf{A}$ tem-se:

$$\mathbf{A} = \mathbf{LU}\,, \tag{2}$$

sendo a matriz $\mathbf{L}$ triangular inferior de diagonal unitária e a matriz $\mathbf{U}$ triangular superior. Dessa forma, o novo sistema a ser resolvido terá a seguinte forma:

$$\mathbf{LUx} = \mathbf{b}\,. \tag{3}$$

Esse sistema pode ser resolvido, utilizando as duas etapas descritas abaixo:

$$\mathbf{Ly} = \mathbf{b} \quad \text{(sustituição descendete ou progressiva)} \tag{4}$$

$$\mathbf{Ux} = \mathbf{y} \quad \text{(sustituição ascendete ou regressiva).} \tag{5}$$

O vetor $\mathbf{y}$ é apenas auxiliar no processo de decomposição.

Observe também que no método de decomposição LU, também chamado de *Crout*, a matriz não precisa ser simétrica.

3. ARQUITETURA MIND – *MULTIPLE INSTRUCTION MULTIPLE DATA*

Considerado o modelo de execução paralelo, onde cada processador está trabalhando em uma estrutura independente, existindo fluxos múltiplos de instruções e dados, como se fosse um conjunto de arquiteturas *SISD* (*Single Instruction Single Data*), como pode ser visto na Figura 1.1. As máquinas paralelas de maneira geral se encaixam nessa categoria, como os clusters e os sistemas MPP (*Massively Parallel Processing*).

Pode-se utilizar aqui um hardware heterogêneo ou homogêneo, memória compartilhada ou distribuída, estrutura de rede dedicada ou compartilhada, criando diversas divisões que Flynn não contemplou em sua modelagem inicial e derivam duas possibilidades de *MIMD* [3].

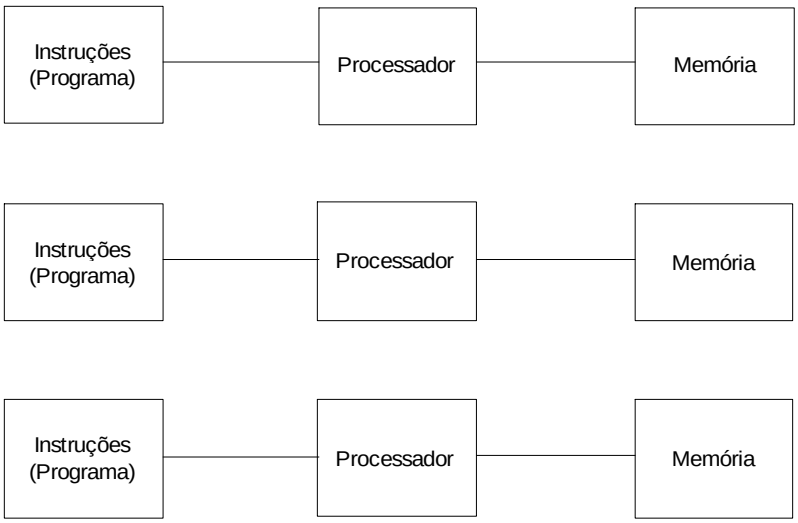

Figura 1.1 – Arquitetura MIMD.

3.1. *MIMD* – Memória Compartilhada (*Shared Memory*)

Essa estrutura de *MIMD* abrange todas as máquinas de possuem múltiplos processadores ou ainda múltiplos núcleos em um único processador, mas que acessam uma memória em comum.

Por exemplo, na estrutura *SMP* (*Symetric Multiprocessing*) o processador sempre está executando instruções, porém como todos estão utilizando uma memória em comum, existem questões de conflitos de acesso, que podem ser reduzidas via cache ou ainda via escalonamento do sistema operacional que será instalado nessa estrutura. Os dados precisam ser constantemente copiados das caches para um acesso a uma memória principal pelo software que esteja fazendo uso desas estrutura (Figura 1.2).

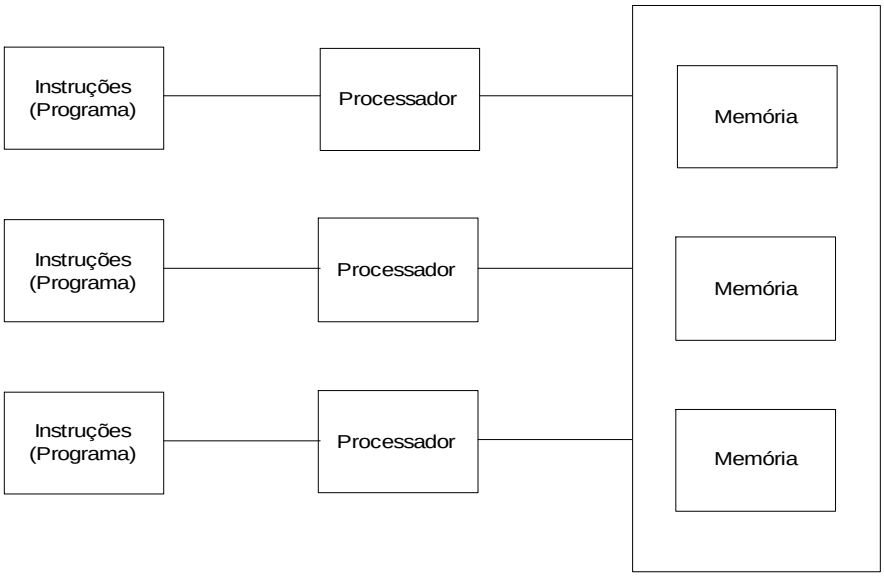

Figura 1.2 – *MIMD* de Memória Compartilhada.

3.2. *MIMD* – Memória Distribuída (*Distributed Memory*)

Aqui estão inclusas todas as estruturas paralelas que que são compostas por unidades processadoras [4], cada um com sua memória e seu próprio barramento de comunicação. Em um cluster essa unidade processadora recebe o nome de nó ou *node*.

Não existe memória comum nessas estruturas, apenas a comunicação dos dados é feita via dispositivos físicos que conectam essas estruturas independentes, como *hubs* ou *switchs* (concentradores de rede). A Figura 1.3 ilustra a arquitetura em questão.

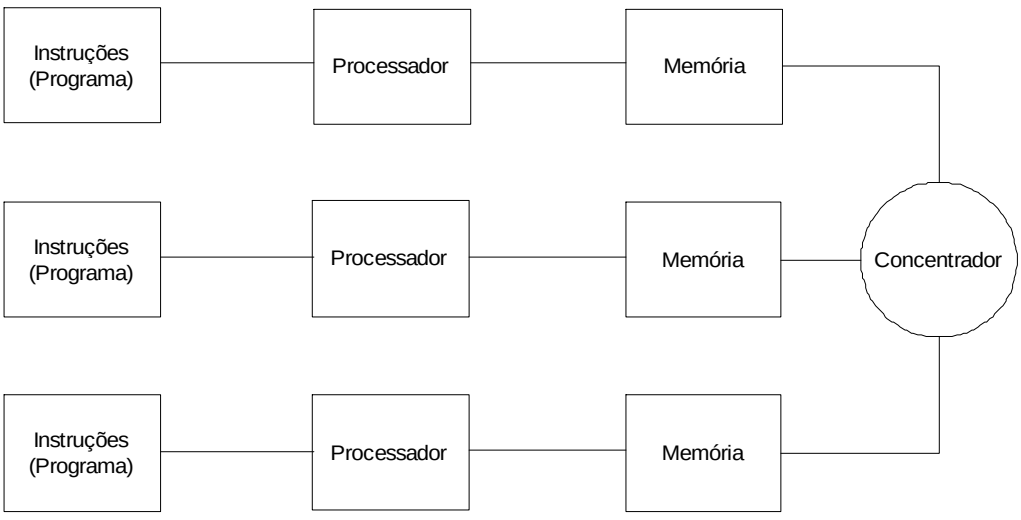

Figura 1.3 – MIMD de Memória Distribuída.

4. SUPERLU

A biblioteca *SuperLU*, desenvolvida principalmente por Xiaoye Shery Li [6], basicamente foi escrita para resolver sistemas de equações lineares esparsos do tipo $AX = B$, ou seja, onde a quantidade de elementos não nulos no sistema é consideravelmente grande. A matriz A é quadrada, não singular e esparsa, enquanto X e B são densas. A matriz a não precisa ser simétrica, por *SuperLU* é apropriado para matrizes com uma estrutura não simétrica.

Foram escritas três versões deste software: a versão seqüencial (*Sequential*), a versão de memória compartilhada (*Multithreaded*) e a versão de memória distribuída (*Distributed*).

A versão seqüencial não faz uso de recursos como threads, apenas utiliza os recursos de um único processador ou núcleo. A segunda versão, chamada de *Multithreaded* foi escrita para múltiplos processadores de memória compartilhada, podendo trabalhar eficientemente com máquinas que possuem mais de um processador fisicamente ou ainda que possuem processadores com mais de um núcleo. A última versão e objetivo de estudo deste trabalho é a versão chamada de *Distributed*, escrita para máquinas paralelas, que combinam o uso de vários processadores de máquinas distintas para realizar os computos do procedimento. Trabalha utilizando MPI para a comunicação entre os processos envolvidos.

Todas as rotinas das três versões do *SuperLU* foram escritas em linguagem C, sendo que na versão de memória compartilhada foi feito uso de *Pthtreads* e ainda *OpenMP*, assim como na versão de memória distribuída que, como dito anteriormente, fez uso de *MPI*. As três versões possuem a interface de uso com programas escritos em linguagem *Fortran* [6].

4.1. Dados de Entrada e Saída

A matriz A é armazenada uma estrutura de dados esparsa, podendo ser representada no formato de compressão por colunas (*Harwell-Boeing*) ou ainda no formato de compressão por linha. O vetor *B*, como dito anteriormente, é armazenado de forma densa, e posteriormente o vetor solução *X*, irá substituí-lo respeitando também essa estrutura densa.

Em se tratando do uso da versão de memória distribuída, a matriz *A* e o vetor *B* podem ser distribuídos por todos os processadores envolvidos no computo.

Aqui estão inclusas todas as estruturas paralelas que que são compostas por unidades processadoras [4], cada um com sua memória e seu próprio barramento de comunicação. Em um cluster essa unidade processadora recebe o nome de nó ou *node*.

4.1. Operações Básicas de Álgebra Linear

Todas as versões do *SuperLU* fazem uso do *BLAS*, para aumentar a performance de execução do código. É necessário utilizar uma versão do *BLAS* já otimizada para a arquitetura utilizada no desenvolvimento do código, como por exemplo o *MKL* da *Intel*, o *BLAS* do *Cray*, entre outros.

As operações básicas de álgebra linear utilizadas do *BLAS* são geralmente multiplicação de uma matriz por um vetor, ou de duas matrizes, todos tratados de forma densa pelo *BLAS*.

O ideal é criar pequenas matrizes densas que tenham o tamanho ideal para que o *BLAS* possa tirar o melhor proveito da arquitetura em questão.

4.3. Diferenças entre as três versões

Todas as rotinas da versão seqüencial e de memória compartilhada estão disponíveis em precisão simples e dupla (tanto para dados complexos quanto reais). A versão de memória distribuída possui apenas precisão dupla (dados complexos e reais).

As matrizes *L* e *U* são armazenadas de maneira distinta nas três bibliotecas, conforme pode ser observado no manual do *SuperLU* [6].

Na versão seqüencial, as matrizes *L* e *U* são armazenadas de forma esparsa, utilizando o conceito de super nó por coluna, onde as colunas consecutivas de estrutura idêntica são armazenadas como blocos densos, para a matriz *L*. A matriz *U* é armazenada no formato de compressão por coluna.

A versão de memória compartilhada, por conta do paralelismo existente, as colunas de *L* e *U* não podem ser computadas de maneira consecutiva, então elas são alocadas e armazenadas fora de ordem, utilizando o conceito de super só por coluna permutado, para a matriz *L*. A matriz *U* é, mais uma vez, armazenada no formato de compressão por coluna.

Por fim, na versão de memória distribuída as matrizes *L* e *U* estão distribuídas pelos processadores, usando o formato de bloco cíclico *2D*, utilizado em bibliotecas que trabalham com matrizes densas, como por exemplo *ScaLAPACK*. Existe porém a diferença que os blocos não são idênticos em tamanho no caso de matrizes esparsas, variando baseado na estrutura de elementos não nulos de *L* e *U*.

4.4. Paralelismo

A versão seqüencial de SuperLU não possui paralelismo explícito, exceto no caso do uso de uma versão do BLAS otimizada para a arquitetura que estiver sendo utilizada, que explore possibilidades de paralelismo.

Na versão de memória compartilhada, é possível escolher o número de threads que serão utilizados no processamento, e o uso de threads varia baseado na arquitetura em questão e no sistema operacional utilizado.

A versão de memória distribuída permite que se especifique o número de processadores envolvidos, além de dizer como os processadores estão organizados em uma malha de duas dimensões (*2D grid*).

5. MATRIZES

Para os testes envolvendo o SuperLU foram escolhidas quatro matrizes do conhecido conjunto de matrizes esparsas de Tim Davis, da Universidade da Flórida [2].

Este conjunto de matrizes é descrito como uma grande e ativa base de dados de matrizes esparsas oriundas de problemas reais. Tal coleção tem seu uso muito difundido na comunida-

de que trabalha utilizando álgebra linear numérica para o desenvolvimento de aplicativos e teste de performance de códigos escritos para trabalharem principalmente com matrizes esparsas.

O conjunto em questão permite testes robustos, por conta dos dados das matrizes serem de problemas reais e ainda permite a repetição e comparação de experimentos, tendo em vista que as matrizes estão disponíveis em diversos formatos de armazenamento.

Cabe ainda dizer que as matrizes cobrem um largo espectro de domínios, incluindo aqueles decorrentes de problemas em geometria 2D ou 3D, como por exemplo, casos de engenharia estrutural.

Para aqueles que desejam utilizar este conjunto, exitem diversos formatos disponíveis no site, além de softwares que facilitam o uso das matrizes em Matlab, Mathematica, Fortran e C, bem como aplicativos online dentro do próprio site.

A Tabela 1.1 ilustra as quatro matrizes utilizadas e suas principais características:

Tabela 1.1 – Quatro matrizes selecionadas do conjunto de Tim Davis.

Nome da Matriz	**Ordem**	**Número de Não Nulos**	**Densidade (%)**
parabolic_fem	525.825	3.674.625	0,0013
af_shell8	504.855	17.579.155	0,0068
inline_1	503.712	36.816.170	0,0145
af_1_k101	503.625	17.550.675	0,0069

Como pode ser observado, a ordem das matrizes é muito parecida, bem como a esparsidade das mesmas, ou seja, a quantidade e elementos nulos presentes em cada uma destas matrizes em uso neste trabalho.

Deve-se dizer também que a estrutura das matrizes aqui utilizadas é idêntica, todas as matrizes utilizadas são simétricas e positivo definidas.

A Figura 1.4 ilustra os valores não nulos presentes em cada uma das matrizes utilizadas nos testes.

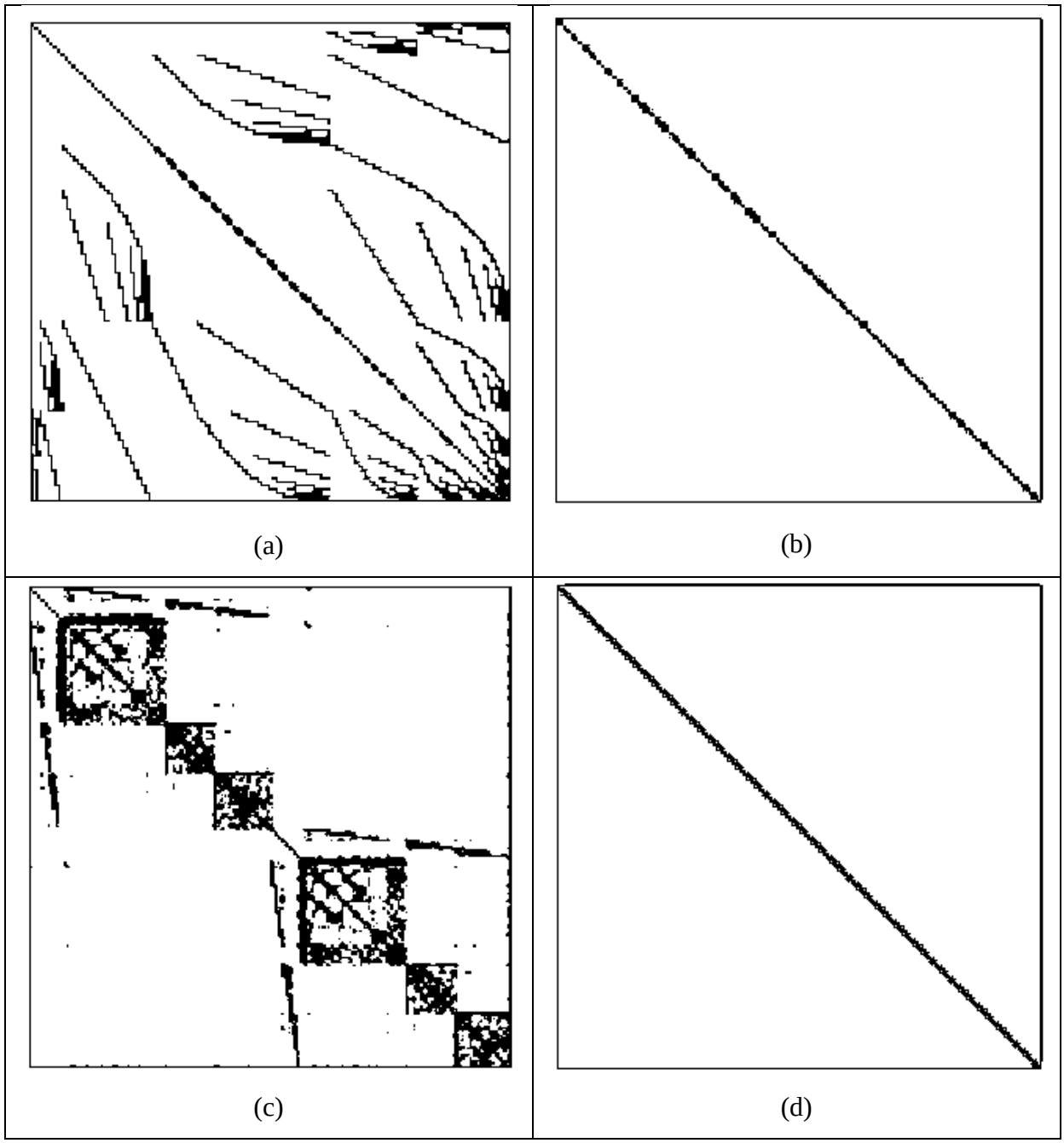

Figura 1.4 – Elementos não nulos de cada matriz. (a) parabolic_fem, (b) af_shell8, (c) inline1 e (d) af_1_k101. Fonte: [2].

5.1. Infra Estrutura de Hardware

Foram realizados testes com as quatro matrizes em questão, utilizando o supercomputador Hopper, produzido pela Cray, sendo seu modelo um Cray XE6.

O Hopper possui 6.384 nós, sendo que cada nó possui 2 processadores AMD Magny-Cours, cada um com um clock de 2.1 Ghz. Cada processador possui 12 núcleos, totalizando 24 núcleos por nó. A máquina toda possui 153.216 núcleos.

A memória RAM é de 32 GB DDR3 1.333 Mhz para 6.000 nós, sendo que os outros 384 nós possuem 64 GB DDR3 1.333 Mhz. Estas memórias, em ambos os casos, estão distribuídas por 4 canais de memória.

Os picos de processamento são de 8.4 Gflops por núcleo, 201.6 Gflops por nó e de 1.28 Peta-flops considerando a máquina toda.

Vale dizer ainda que cara núcleo tem seu próprio cache L1 e L2, com 64 KB e 512 KB respectivamente, além de 6 MB de cache L3 compartilhado por cada grupo de 6 núcleos do processador Magny-Cours.

5.2. Resultados

A versão de memória distribuída de *SuperLU* foi selecionada para fazer os testes iniciais relativos ao desempenho de softwares paralelos utilizados para a resolução de sistemas de equações lineares.

Como pode observado na Figura 1.5, o tempo de execução do software, inicialmente executado em 4 processadores, foi diminuindo ao longo do aumento do número de processadores utilizados. O número máximo de processadores utilizados foi de 16.384.

Vale lembrar que quando o nó possui mais de um processador ou núcleo, essa versão do *SuperLU* escrita para memória distribuída, trata as núcleos ou processadores como nós da rede, ou seja, a comunicação entre eles é feita utilizando *MPI*, passando mensagens para a comunicação.

Em tese, a comunicação poderia ser melhorada, em uma versão híbrida, que estabelecesse a comunicação entre os processos locais (de um mesmo nó), via threading utilizando, por exmeplo, *Pthreads* ou ainda *OpenMP*.

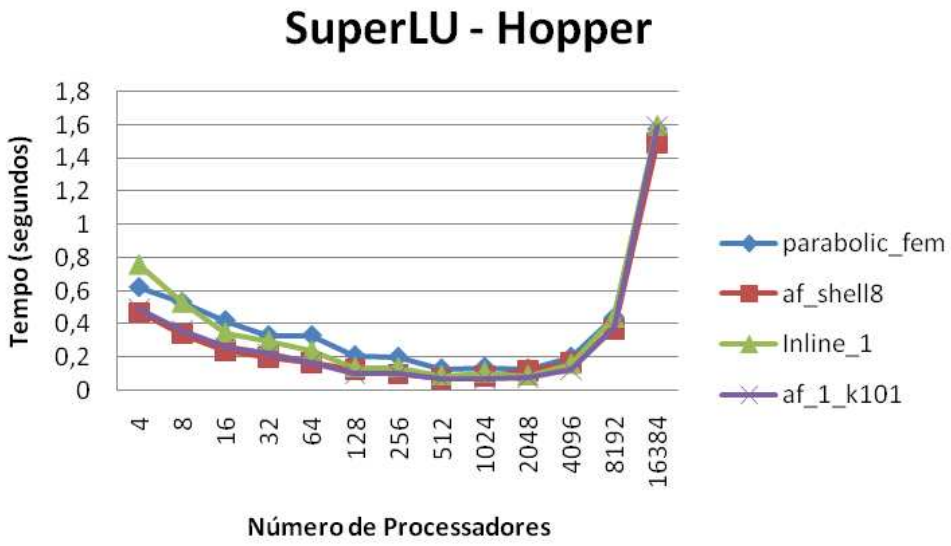

Figura 1.5 – Testes do *SuperLU* no supercomputador Hopper.

O menor tempo da resolução de cada sistema foi obtido com uso de 512 processadores para a versão distribúida de *SuperLU*, como visto na Figura 1.5. Isso não quer dizer que para

outros software o melhor tempo seria também com 512 processadores, pois o uso das rotinas de passagem de mensagens via *MPI* é relativo, dependendo da meneira como a biblioteca foi implementada.

Existem ferramentas específicas para mensurar o tempo gasto em comunicação *MPI*, podendo análisar com maior precisão qual o momento ideal para cessar o aumento do número de processadores utilizados em uma determinada biblioteca.

Agradecimentos

Este trabalho foi possível graças ao empenho dos pesquisadores Alexandre Beletti Ferreira, Paulo de Mattos Pimenta e Osni Marques.
Para a realização do trabalho, foram utilizadas as estruturas do LMC (Laboratório de Mecânica Computacional) do Departamento de Estruturas, pertencente a Escola Poletécnica da USP no Brasil, além dos supercomputadores do NERSC (*National Research Scientific Computing Center*), através do LBNL (*Lawrence Berkeley National Laboratory*), ambos pertencentes ao governo dos Estados Unidos da América.
Agradecimentos também a Capes (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) por apoiar esta pesquisa.

6. REFERÊNCIAS

[1] Arenales S., Darezzo A., "Cálculo Numérico: aprendizagem com apoio de software". *Thompson*, 2008.

[2] Davis T. A., "University of Florida Sparse Matrix Collection". *http://www.cise.ufl.edu/research/spase/matrices*, 2011.

[3] Pitanga, M., "Construindo Supercomputadores com Linux". *Brasport*, 2004.

[4] Pitanga, M., "Computação em Cluster: O Estado da Arte da Computação". *Brasport*, 2003.

[5] Ruggiero M. A. G., Lopes V. L. R.,"Cálculo Numérico: aspectos teóricos e computacionais". *Makron Books*, 1996.

[6] Li X. S., Demmel, J. W., Gilbert, J. R., "SuperLU Users Guide.". Technical Report 44289-Lawrence Berkeley National Laboratory, 2011.