第3回 クラスタゼミ

ゼミ担当者　：　斉藤宏樹，釘井睦和，勝崎俊樹
指導院生　　：　谷村勇輔，児玉憲造，片浦哲平，下坂久司
開催日　　：　2002 年 5 月 21 日

ゼミ内容: クラスタの性能評価を行うベンチマークについて，代表的なリンパックについて理解する，そして ATLAS，HPL・LAM のインストールの方法・実行方法について理解する．

1　ベンチマーク

1.1　性能をあらわす指標

コンピュータのコスト性能比を調べる際，何らかの指標に基づいて実際に計測し，その結果を比較しなければ，計算能力の客観的な判断ができない．コンピュータの性能を示す指標としては，以下のものがある．

- MIPS（Million Instrucctions Per Second）
  マイクロプロセッサの性能を示す値．1 秒間に何百万回の命令を処理できるかを示す．
- Flops（Floating Operations Per Second）
  1 秒間に何回の浮動小数点演算ができるかを示す．浮動小数点演算とは，浮動小数点数の四則演算である．浮動小数点演算は，科学技術計算において多用される．
- MOPS（Million Operation Per Second）
  ディジタル信号処理を行なうプロセッサなどの性能指標に用いられるベンチマーク．1 秒間に何百万回の命令を実行できるかを示す．

これらの指標は，アーキテクチャや計測に用いるプログラムによって，結果が異なる．一方で，コンピュータの用途は多様であり，同じコンピュータでも問題ごとに性能が異なる．つまり，これらはコンピュータの性能をあらわす絶対的な指標ではない．

1.2　ベンチマークの目的

ベンチマークはコンピュータの計算処理能力を測るために，前節のような指標を表すものであるが，上記のような理由からその指標が絶対であるとはいえない．つまり，ベンチマークを用いた計算機の性能の測定は困難である．しかし，ベンチマークの特性を知ることにより，ある特定の問題に対する性能比較を行ったり，ある計算機の得意の問題を調べることはできる．

1.3　ベンチマークの種類

ベンチマークは，評価の対象からいくつかの種類に分類できるが，大きくは以下の 2 つに分類される．

1. マクロベンチマーク（総合ベンチマーク）
2. ミクロベンチマーク（機能別ベンチマーク）

マクロベンチマークを用いてシステムの総合的な評価を下し，ミクロベンチマークテストを用いて機能別の評価を下す．このようにベンチマークを使い分けることで，システムの改善すべき点，それによる総合評価への影響などを知ることができ，さらなるシステムの総合的な処理能力を効率的に向上させる手がかりとなる．つまり，マクロベンチマークでは全体の，ミクロベンチマークでは個々のベンチマークを行うことになる．以下に，マクロベンチマーク，ミクロベンチマークについて詳しく示す．

1.3.1　マクロベンチマーク

マクロベンチマークは，システム全体の性能を評価するためのものである．すなわち，マクロベンチマークの実行結果には，CPU 性能，ネットワーク性能，メモリ性能等のミクロベンチマークの集合が総合的に反映される．

1.3.2　ミクロベンチマーク

ミクロベンチマークは，システムのいくつかの機能の性能を測定する．その例としては，以下に挙げるようなものがある．

1. 演算性能ベンチマーク
   MIPS などを用いて，コンピュータの演算性能を測る．
2. グラフィック性能ベンチマーク
   あるグラフィック動作を起こさせ，1 秒間に何回行えるかを計測する．
3. ネットワーク性能ベンチマーク
   単位時間内における処理量，待ち時間，往復時間（RTT）などで測定する．

2　Linpack Benchmark

Linpack Benchmark（以下 Linpack）は，米国テネシー大学の J.Dongarra 博士によって開発された LU 分

解にもとづく連立一次方程式の解法プログラムである．もとは，線形代数問題の LU 分解を行うためのサブルーチンの集合であり，ベンチマークにも利用されていると表現するほうが正確である．ベンチマークとしての Linpack は，計算機で頻繁に利用する命令を集めたカーネルベンチマークであったが，実行結果に分散メモリ型並列計算機のためのベンチマークとして HPL が開発された．

Linpack の概要に関して，以下の記述がある．

> Linpack ベンチマークは連立一次方程式を解くことができる．この性能が現実問題において反映されるシステムは 1 台もない．しかしながら，連立一次方程式を解くことによって，特定用途で使用されるシステムの性能を反映する．その問題がとても規則的なので，実行結果はかなり高い．そして，性能の多くは良いピーク性能の良い補正となる．

世界のスーパーコンピュータの性能を競って，その結果をランキングする TOP500 で Linpack が使用されている都合上，本ゼミでは Linpack についておおまかな説明をする．

2.1 パラメータ

Linpack には 17 のパラメータが存在する．それらの中で，計測結果に大きく影響を及ぼすものについて説明する．

2.1.1 問題サイズ（N）

問題サイズ（N）は Linpack で解く問題の大きさである．つまり，Linpack では N 次元連立方程式を解くことになる．N は Linpack の実行結果に最も影響を及ぼす．大体において，N が大きくなるほど良い結果が得られるが，N が大きくなるほどメモリ使用量が増える．以下に最適な結果をもたらす N の値について述べる．

1. N は，メモリの 80 %を使用するように設定すると良い結果が得られる．
2. N の 2 乗が係数行列の要素数となる．そして行列の各要素は double 型なので，上の条件を満たす N の値は，以下の式によって求められる．

$$N = \sqrt{(\text{計算対象となる計算機の全メモリ容量 } [byte]) \times 0.8/8}$$

2.1.2 ブロックサイズ（NB）

ブロックサイズ（NB）は，粒度のことである．NB が大きくなると，通信量が減るがロードバランスが悪くなり，NB が小さくなると，通信量が増えるがロードバランスが良くなる．NB の値を 32〜256 にすると，良い結果が得られる．

2.1.3 プロセスグリッド（P，Q）

プロセスグリッド（P と Q）は，問題の行列をそれぞれのプロセスにどのように分割するかを示す．必然的に P と Q の積が実行ノード数となる．P と Q は等しいか P より Q が大きい方が良い．

2.2 バリエーション

Linpack を測定するに当たってのルールとして，TPP（Toward Peak Performance）と N=100 と呼ばれる二つのルールがある．TOP500 で用いられているルールは，TPP である．以下に各ルールの特徴に付いて述べる．

2.2.1 TPP（Toward Peak Performance）

TPP では，ソース修正による最適化が許されており，ベストな性能に達するため，ユーザが問題サイズを決めるなど，多くのパラメータを変更することができる．また，コンパイラ等のソフトウェアを最大限に利用することができる．また，問題が規則的なので，結果として得られる性能はかなり高い．このように，Linpack では，各システムの特徴にあわせたチューニングが可能である．

このルールでは，絶対的な性能値とともに，与えられた問題規模において，ハードウェアのピーク性能にどこまで近づけるかが，評価のポイントになる．

2.2.2 N=100

N=100 の基本的なルールとしては，問題サイズを N=100 に固定で，ソースの修正による最適化が許されていないということがある．これは，ベクトル長の短い場合の性能を示し，ハードウェア性能のみならずコンパイラの機能も併せて，総合性能を測定することができる．

3 Linpack インストール

Linpack は米テネシー大学の Jack Dongarra 博士が開発した連立 1 次方程式の解法プログラムである．行列計算のライブラリである ATLAS と，Linpack のインストールの手順を以下に示す．

3.1 ATLAS

ブラウザで http://math-atlas.sourceforge.net/から，「Software」を選んで atlas3.3.15.tar.bz2 をダウンロードする．次に，解凍ソフト bzip2 をインストールし，atlas3.3.15.tar.bz2 を解凍し，ディレクトリ ATLAS ができていることを確認する．

```
# apt-get install bzip2
$ tar xvfj atlas3.3.15.tar.bz2
$ ls
ATLAS atlas3.3.15.bz2 .....
```

次にディレクトリ ATLAS にて，make config を実行する．

```
~/ATLAS$ make config CC=gcc
```

gcc の部分は使用するコンパイラに応じて書き換える．また，コンパイラのデフォルトは gcc になっているので，gcc を使用する場合は以下のようにしても良い．

```
~/ATLAS$ make
```

コンパイルを実行すると, いくつかの質問をされるが, 全てデフォルトで答えてよい．次に，ATLAS のインストールを行う．以下の ATLAS インストールコマンドを入力することになるが，Linux_P4SSE2 の部分は CPU によって異なり，ATLAS のディレクトリにできているファイルの名前（Make.Linux_P4SSE2）を用いる．

```
~/ATLAS$ make install arch=Linux_P4SSE2
```

これで，ATLAS のインストール終了である．

4 HPL

ブラウザで http://netlib.org/benchmark/hpl から「Software」を選んで hpl.tgz をダウンロードする．そして hpl.tgz を解凍する．

```
$ tar zxvf hpl.tgz
```

そしてディレクトリ hpl/setup に移動する．

```
$ cd hpl/setup
```

ここで，hpl/setup/Make.Linux_PII_CBLAS を hpl/ に名前を変えてコピーする．

```
$ cp Make.Linux_PII_CBLAS
../Make.Linux_P4SSE2_CBLAS
```

HPL を実行するには mpich か lam をインストールする必要がある．それぞれの場合について述べる．

4.1 MPICH

MPICH をインストールする．

```
# apt-get install mpich
```

このあと，cd .. でコピーした Make.Linux_P4SSE2_CBLAS のあるディレクトリへ移動し，Make.Linux_P4SSE2_CBLAS を編集する．

```
$ cd ..
emacs Make.Linux_P4SSE2_CBLAS
```

編集で行うことは以下のとおりである．

1. MPdir = /usr/lib/mpich
2. MPlib = $(MPdir)/lib/libmpich.a
3. LAdir = $(HOME)/ATLAS/lib/Linux_P4SSE2
4. ファイル中の全ての P　を P4SSE2 に置き換える

最後に，make を実行してインストールを終える．

```
$ make arch=Linux_P4SSE2_CBLAS
```

hpl/bin/Linux_P4SSE2_CBLAS に実行ファイル xhpl とパラメータファイルを設定する HPL.dat ができていればよい．

4.1.1 実行方法

実行方法を示していく．mpich では rsh を用いるため, rsh をインストールする．

```
# apt-get install rsh-client rsh-server
```

次に, emacs で/etc/hosts.equiv を編集する．hosts.equiv に以下に示す localhost を追加する．

```
localhost
```

mpich のコマンドを使用できるようにパスを通す．

```
$ export PATH=/usr/lib/mpich/bin:$PATH
```

そして実行ファイルのあるディレクトリへ移動する．

```
$ cd hpl/bin/Linux_P4SSE2_CBLAS
```

xhpl を実行する．

```
$ mpirun -np 4 xhpl
```

4.2 LAM

ブラウザで http://www.lam-mpi.org/download/から lam-6.5.6.tar.gz をダウンロードし，解凍する．そして，コンパイル，コンフィグ，メイクを行う．

```
$ tar zxvf lam-6.5.6.tar.gz
$ cd lam-6.5.6
$ ./configure --prefix=/usr/local/lam --with-
romio
$ make
# make install
```

このあと，Linux_P4SSE2_CBLAS を編集する．編集で行うことは以下のとおりである．

1. MPdir = /usr/local/lam

2. MPlib = $(MPdir)/lib/libmpi.a $(MPdir)/lib/liblam.a

3. LAdir = $(HOME)/ATLAS/lib/Linux_P4SSE2

4. ファイル中の全ての PII を P4SSE2 に置き換える

最後に，make を実行してインストールを終える．

```
$ make arch=Linux_P4SSE2_CBLAS
```

hpl/bin/Linux_P4SSE2_CBLAS に実行ファイル xhpl とパラメータファイルを設定する HPL.dat ができていればよい．

4.2.1 実行方法

実行方法を示していく．まず，/usr/local/lam/bin に host というファイルを作り，localhost と書く．

```
# emacs /usr/local/lam/bin/host
localhost
```

lamboot や，mpi などのコマンドが使えるように PATH を通す．

```
$ export PATH=/usr/local/lam/bin:$PATH
```

/home/ユーザ名/.bashrc ファイルにも PATH を書き加える．

```
export PATH=/usr/local/lam/bin:$PATH
```

次にデーモンを立ち上げる．

```
$ /usr/local/lam/bin/lamboot -v host
```

実行ファイルのあるディレクトリへ移動する．

```
$ cd hpl/bin/Linux_P4SSE2_CBLAS
```

xhpl を実行する．

```
$ mpirun -np 4 xhpl
```

実行した後はデーモンを削除する．

```
$ wipe -v host
```

5 Linpack Benchmark の結果

前節までで説明した Linpack Benchmark を行った結果を以下に示す．第第 2 節で説明したように，問題サイズ（N），ブロックサイズ（NB）の値を調整することで，Linpack の実行結果は変化する．まず，ブロックサイズを 4 に固定し，問題サイズを変化させた場合の CPU 性能（GFlops）について考察する．

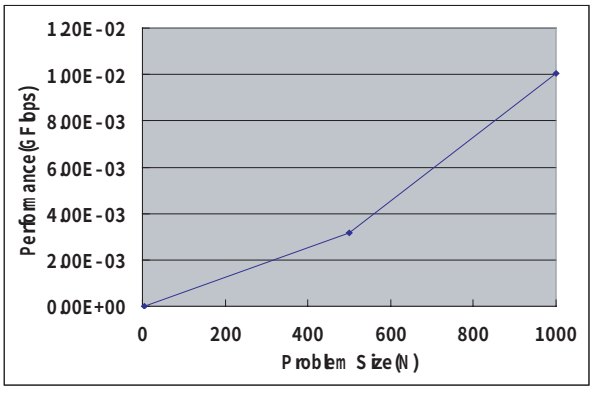

Fig. 1　問題サイズを変化させた場合

Fig. 1 のように，問題サイズを増やしていくほどその CPU 性能の結果は向上していることが分かる．これは，問題サイズが第 2 節で示した理想の値に近づいているためだと考えられる．

次に，問題サイズ 1000 に固定し，ブロックサイズを変化させたときの CPU 性能（GFlops）と，要した時間について考察する．

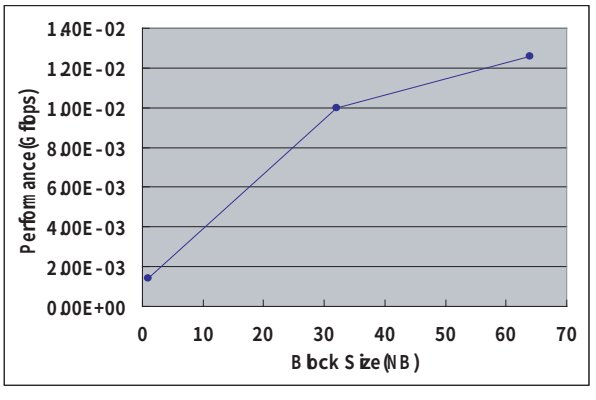

Fig. 2　ブロックサイズを変化させた場合

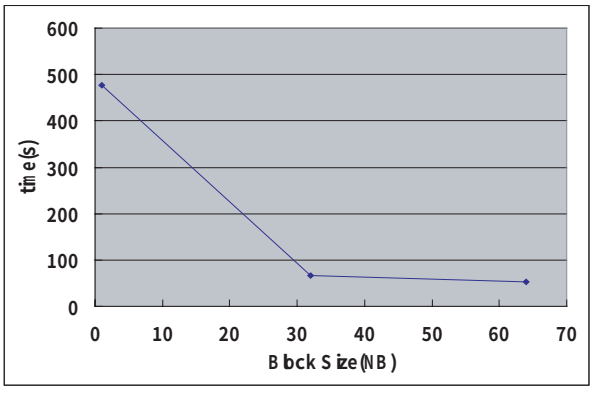

Fig. 3　ブロックサイズを変化させた場合要した時間

まず，Fig. 2 から考察する．Fig. 2 では，NB=1 のと

きと比較して，NB=32，NB=64 のときに高い CPU 性能を得られていることが分かる．これは，第 2 節で説明されたブロックサイズの理想的な値であることが原因と考えられる．

次に，Fig. 3 について考察する．Fig. 2 から分かるように，理想的なブロックサイズを実現すると，高い CPU 性能の結果が得られることが分かった．これに加えて Fig. 3 の結果から，理想的なブロックサイズを用いることで演算時間も節約できることが分かった．

最後に，理想的だといわれるメモリの 80 %の利用かつブロックサイズ 32 で実行した場合の結果を示す．

Table 1　理想的な Linpack の結果

問題サイズ	7000
ブロックサイズ	32
Time（秒）	1924.24
CPU 性能（GFlops）	1.189e-1

Table 1 では，今までの結果と比較して格段に高い CPU 性能を計測できている．このことから，問題サイズ，ブロックサイズともに理想的な値を適用することで，より高い CPU 性能の結果を得られるといえる．

6　課題

個人で Linpack Benchmark を行う．まず ATLAS をインストール・コンパイルし，HPL をインストールする．さらに MPICH か LAM をインストールして，実行し計測すること．(Fig. 4 参照)

7　参考文献

- 大規模クラスタにおける処理能力の向上と管理ツールの検討および Access Grid の導入と評価
  下神納木淳　小池政輝　田村隆一　松山靖彦
  同志社大学工学部知識工学科卒業論文
  2002 年 3 月

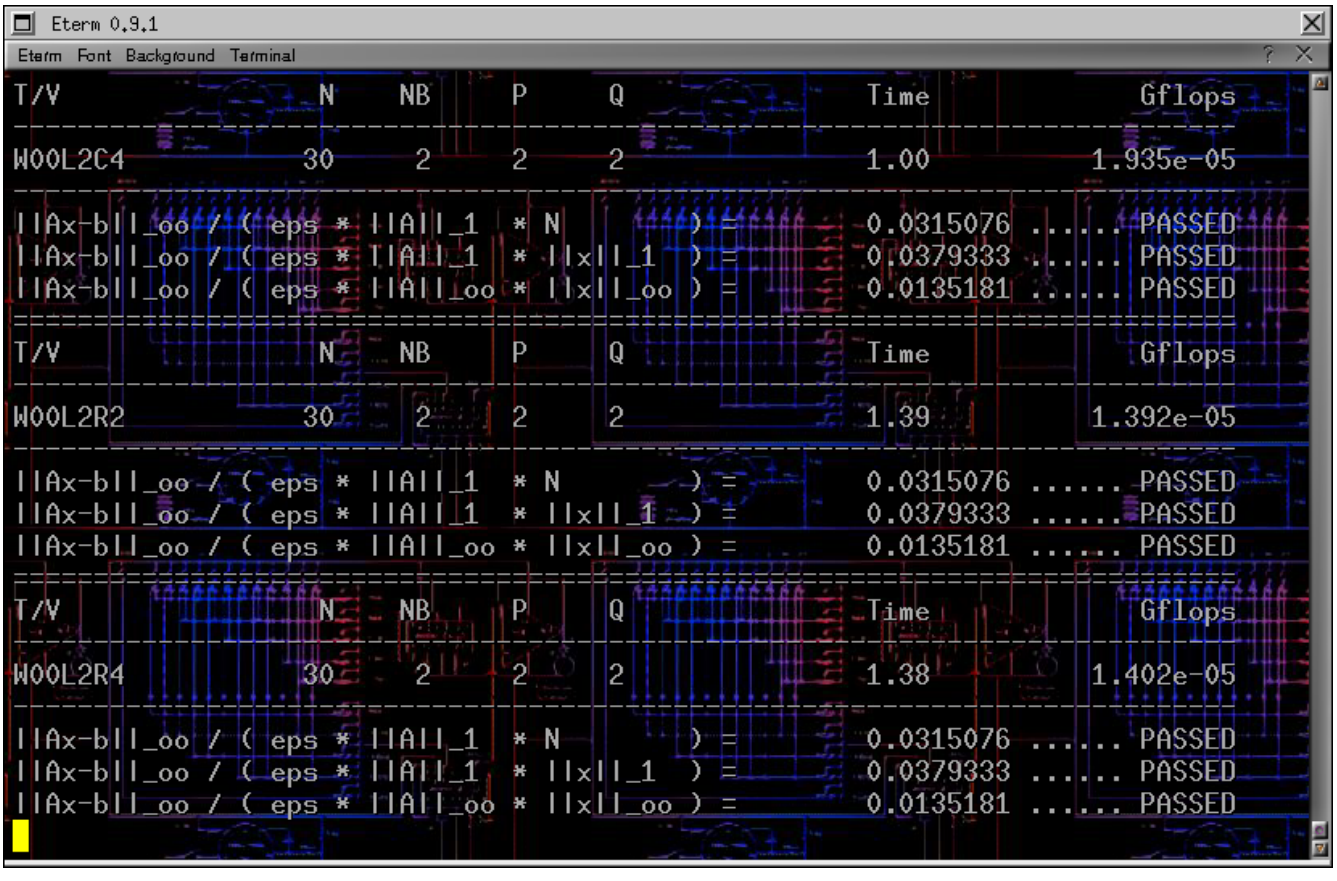
Eterm 0.9.1
Eterm Font Background Terminal
T/V N NB P Q Time Gflops
W00L2C4 30 2 2 2 1.00 1.935e-05
||Ax-b||_oo / (eps * ||A||_1 * N) = 0.0315076 ..... PASSED
||Ax-b||_oo / (eps * ||A||_1 * ||x||_1) = 0.0379333 ..... PASSED
||Ax-b||_oo / (eps * ||A||_oo * ||x||_oo) = 0.0135181 ..... PASSED
T/V N NB P Q Time Gflops
W00L2R2 30 2 2 2 1.39 1.392e-05
||Ax-b||_oo / (eps * ||A||_1 * N) = 0.0315076 ..... PASSED
||Ax-b||_oo / (eps * ||A||_1 * ||x||_1) = 0.0379333 ..... PASSED
||Ax-b||_oo / (eps * ||A||_oo * ||x||_oo) = 0.0135181 ..... PASSED
T/V N NB P Q Time Gflops
W00L2R4 30 2 2 2 1.38 1.402e-05
||Ax-b||_oo / (eps * ||A||_1 * N) = 0.0315076 ..... PASSED
||Ax-b||_oo / (eps * ||A||_1 * ||x||_1) = 0.0379333 ..... PASSED
||Ax-b||_oo / (eps * ||A||_oo * ||x||_oo) = 0.0135181 ..... PASSED

Fig. 4　実行結果画面