応用数理工学特論
第 1 回授業ノート

澤田賢吾, 澤藤俊平

2008 年 4 月 28 日

目次

1 序論

1.1 ハイパフォーマンスコンピューティング技術とは

ハイパフォーマンスコンピューティング技術とは HPC 技術とも呼ばれ、大規模な計算を高速かつ高精度に行うための技術をさす。HPC 技術は以下のような技術を含む

- 単体プロセッサ向けの性能最適化技術
- 高速・高精度な計算アルゴリズム
- 並列化技術
- ネットワーク・GRID 技術

1.2 PC 向けプロセッサのクロック周波数の向上

プロセッサの周波数はここ 15 年で飛躍的に伸び、1993 年からの 10 年で約 50 倍にもなった。また、同時にウェハに回路を転写する際の光学的分解能であるプロセスルールも大幅に縮小されている（図 1)。2008 年 4 月現在ではコンシューマ向けプロセッサの動作周波数は定格で最大 3.20GHz で、プロセスルールは 45nm までシュリンクが進んでいる。しかし、近年では消費電力あたりの性能が重要視されてきたため AMD などマイクロプロセッサメーカー各社は動作周波数の向上をあきらめ、単一プロセッサ内に複数のコアをおくことによって処理能力の向上を図っている。

AMD 社 Opteron プロセッサの 1.6GHz のピーク性能は 3200**MFLOPS**(Mega Floating point Operations Per Second) である。これは 1 秒間に最高で 32 億回の浮動小数点演算を実効可能だということを表している。

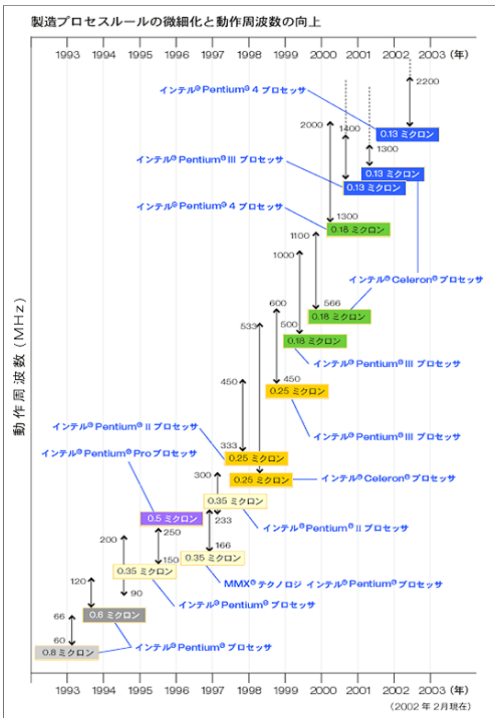

図 1　製造プロセスの微細化と動作周波数の向上

1.3 プロセッサの実効性能

先にあげた Opteron を用いて一般的な科学技術計算であるガウスの消去法を以下のアルゴリズムで行うと行列のサイズが 1000 のとき 225MFLOPS でありピークのわずか 7% の性能しか得られない（図 2)。また、計算する行列のサイズが大きくなるほど実効性能が下がることがわかる。

ガウスの消去法のアルゴリズム

```
do k=1, n
    do i=k+1, n
        a(i,k)=a(i,k)/a(k,k)
    end do
    do j=k+1, n
        do i=k+1, n
            a(i,j)=a(i,j)-a(i,k)*a(k,j)
        end do
    end do
end do
```

このように実効性能がピーク性能より大きく下回る最大の要因はデータ転送速度の遅さにある。プロセッサが計算を行う際には計算に必要なデータをメインメモリから演算器に転送しなければならない。現在のプロセッサは演算器の処理速度に比べメインメモリからの転送速度が遅いため、データ転送が追いつかずこれが律速となっている。これの解消法を考えるためにはプロセッサのアーキテクチャを知る必要があるため、次節ではそれを紹介する。

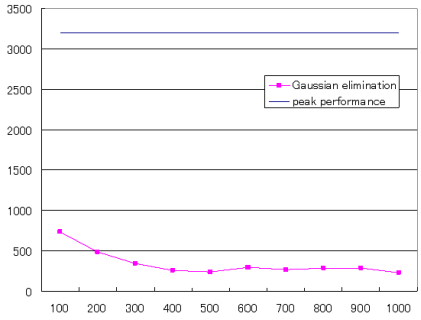

図 2　ガウスの消去法を行う際の Opteron の性能と行列のサイズ

2　プロセッサのメモリ階層

コンピュータプログラムによる計算とデータ処理はマイクロプロセッサとメインメモリによって行われる。データと命令はメインメモリからプロセッサ内のキャッシュへ送られさらにレジスタを通って演算器へ転送され、命令に従った計算が行われる。その際、それぞれの記憶領域に通ったデータと命令が保存される（図 3）。各記憶領域は演算器に近いほど転送速度が高速である反面、容量が小さい。また、演算器の処理速度と主メモリのデータ転送速度は年々差が大きくなってきている。そのため計算の最適化は今後さらに重要性が増してくる。

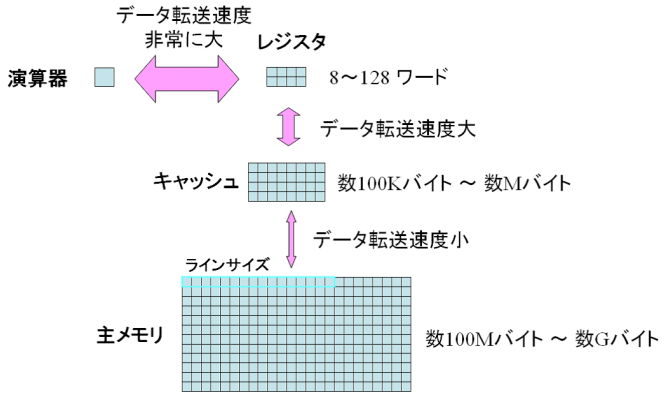

図 3　メモリ階層

3　単体プロセッサ向け高性能化手法

3.1　データの局所性の向上

演算器と密接に結びついているレジスタの中にあるデータはプロセッサの持つ最高速度で計算することが可能である。したがって、いったんレジスタに入れたデータに対して必要な計算を集中して行うように計算の順序を変えることによって性能を最適化することができる。これを**データ参照の局所性**を高めるという。キャッシュやメインメモリについても同様の方針で最適化を行う。

3.1.1　レジスタブロッキング

レジスタの再利用性を高める最適化のことを**レジスタブロッキング**という。行列の乗算を例にして具体的なレジスタブロッキングの手法を説明する。最適化を行っていない行列乗算のアルゴリズムは以下のとおりである。

最適化前の行列乗算アルゴリズム

```
do i=1, n
    do j=1, n
        sum=0.0d0
        do k=1, n
            sum=sum+a(i,k)*b(k,j)
        end do
        c(i,j)=sum
    end do
end do
```

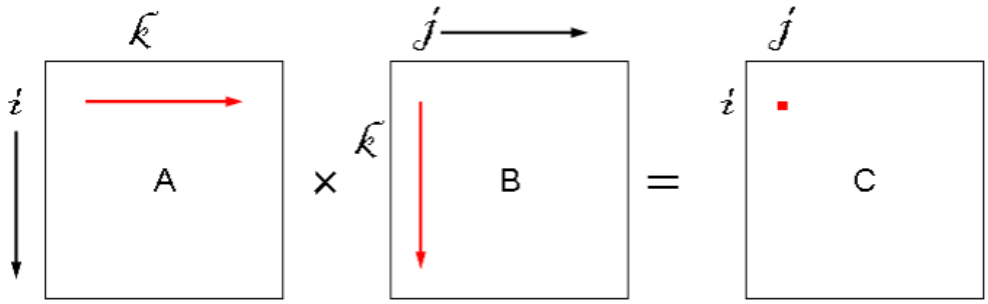

図 4　最適化前の行列の乗算

1 ループにつき sum=sum+a(i,k)*b(k,j) と加算と乗算の 2 演算を行っているので、最適化を行わず乗算を行った場合

$$\left.\begin{array}{l} a(i,k)\ldots 1\text{ 回ロード} \\ b(k,j)\ldots 1\text{ 回ロード} \end{array}\right\} 2\text{ 演算} \Rightarrow 2\text{ 演算/2 ロード}$$

となる。ここで、ロードとはデータをレジスタに転送することである。実際に Opteron 1.6GHz を用いてこのアルゴリズムで 500×500 の行列乗算を行った場合、ピーク性能のわずか 2.4% である 77.7MFLOPS しか発揮することができない。

行列の乗算ではレジスタブロッキングを行うために**ループ展開**を用いる。ループ展開は性能最適化のための重要で基本的な手法である。まず、i のループを 2 倍展開する場合を考える。a(i,k), a(i+1,k), b(k,j) の 3 要素をレジスタにロードし、a(i,k) と b(k,j), a(i+1) と b(k,j) のそれぞれの組み合わせについて加算と乗算の 2 演算、合計で 4 演算行う（図 5)。

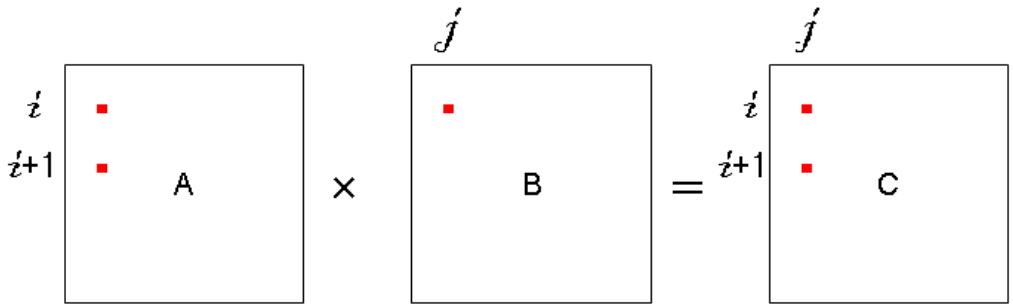

図 5　i のループを 2 倍展開した場合

この場合の演算とロードの比は

$$\left.\begin{array}{ll} a(i,k) & \ldots \text{ 1 回ロード} \\ a(i+1,k) & \ldots \text{ 1 回ロード} \\ b(k,j) & \ldots \text{ 1 回ロード} \end{array}\right\} \text{4 演算} \Rightarrow \text{4 演算/3 ロード}$$

となる。これによって性能はピーク性能の 5.1%(162.8MFLOPS) となった。

さらに同様にして j に対しても 2 倍展開した場合（図 6)、

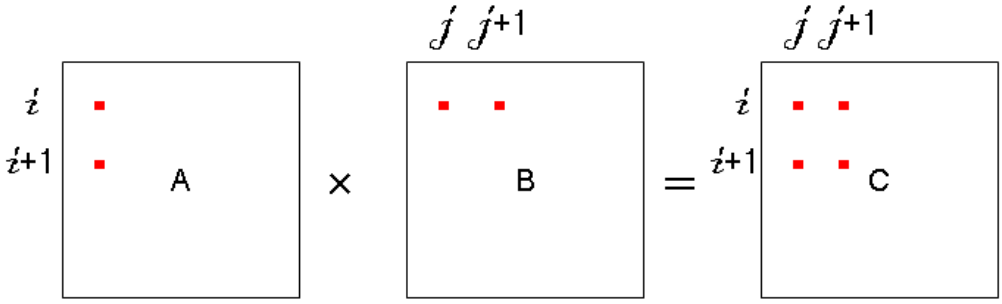

図 6　i, j のループを 2 倍展開した場合

$$\left.\begin{array}{ll} a(i,k) & \ldots \text{ 1 回ロード} \\ a(i+1,k) & \ldots \text{ 1 回ロード} \\ b(k,j) & \ldots \text{ 1 回ロード} \\ b(k,j+1) & \ldots \text{ 1 回ロード} \end{array}\right\} \text{8 演算} \Rightarrow \text{8 演算/4 ロード}$$

となり、性能はピーク性能の 7.5%(240.8MFLOPS) となった。また、i を 4 倍、j を 2 倍展開した場合は 16 演算/6 ロードで 10.1%(324.7MFLOPS)、i,j の各ループを 4 倍展開した場合は 32 演算/8 ロードで 15.5%(495.5MFLOPS) となった（図 7)。

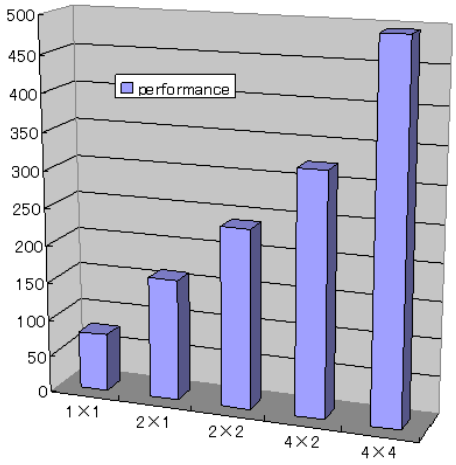

図 7　レジスタブロッキングの効果

このように 1 ロードあたりの演算数を多くしていくことによってレジスタを有効に活用できる。ただし、どこまでも多く展開を行えばよいというものではなく、ロードする要素数と計算の結果がレジスタに収まるサイズにしなければ高速化は望めない。例えば、i を l 倍、j を m 倍展開した場合ロードする要素数は l+m で計算結果は l*m であり、これがレジスタに収まるように調節をする必要がある。

3.1.2　**キャッシュブロッキング**

キャッシュの再利用性を高める最適化のことを**キャッシュブロッキング**という。

- 行列を部分行列 (1 個が L × L) に分割
- 部分行列単位で乗算を行う
- L は部分行列 3 個がキャッシュに格納できるサイズにすると最適

```
do I = 1, N/L
   do  J = 1, N/L
     C(I,J) = 0
     do K = 1, N/L
       C(I,J) = C(I,J) + A(I,K)B(K,J)
     end do
   end do
end do
```

転送データ量　：$3L^2$
演算量　　：$2L^3$
演算量/転送量 = 2L/3

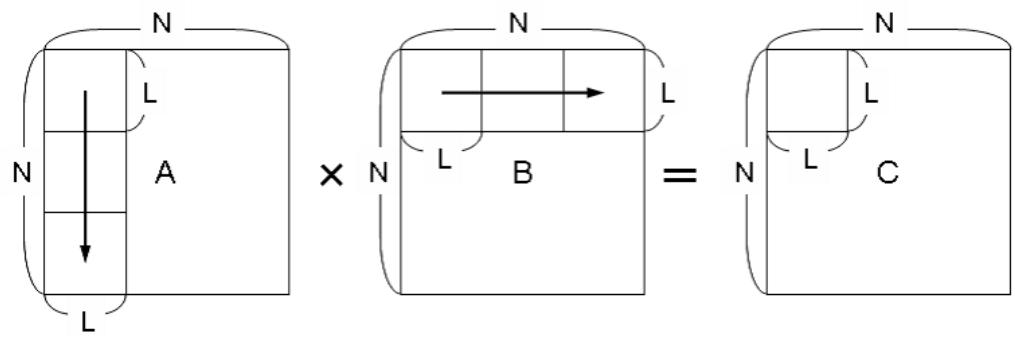

図 8　キャッシュブロッキングのアルゴリズム

L=100 なら 1 回の転送で約 60 個の演算が可能になる。逆に言えば同じ演算量で主メモリへのアクセスが約 1/L になる。レジスタブロッキング、キャッシュブロッキングを行うことでピーク性能の約 4 分の 1 の性能を引き出すことができる。

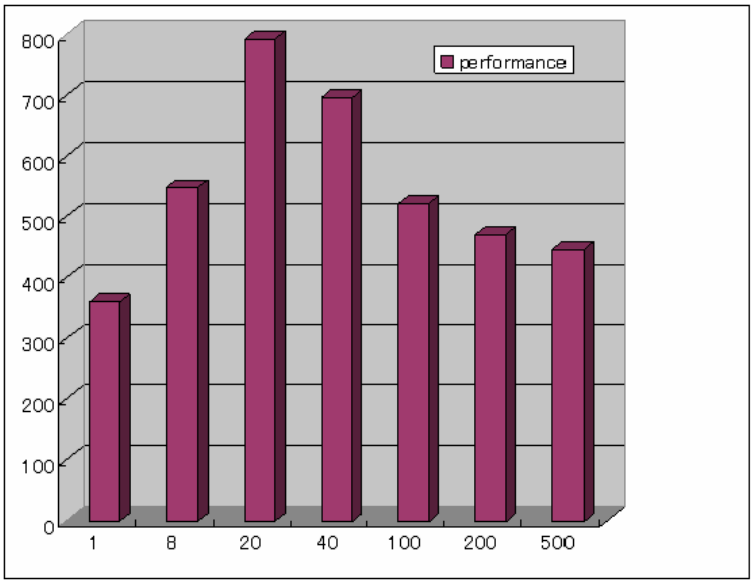

図 9　キャッシュブロッキングの効果

しかし、ピークの性能には程遠くまだまだ遅い。これはキャッシュが 2，3 階層になっていることやアクセスの遅延時間を考慮していないことが原因である。性能を最大限に引き出すためにはこれらの点も考慮した最適化が必要である。

3.2 各マシンの環境に合わせた最適化

3.2.1 BLAS(Basic Liner Algebra Subprograms)

BLAS というのは基本的な行列演算に対する最適化済みのライブラリ群のことで、各プロセッサに対して提供されている。基本的な行列演算とは

- 行列の積や和
- 行列とベクトルの積
- ベクトルの和、内積

などである。

3.2.2 ATLAS(Automatically Tuned Linear Algebra Software)

各マシンの環境に合わせ自動で最適なキャッシュブロッキングのサイズ、ループ展開のサイズを探してくれるソフトウェア。
n=1000 のときピーク性能の 95 %以上を達成している。

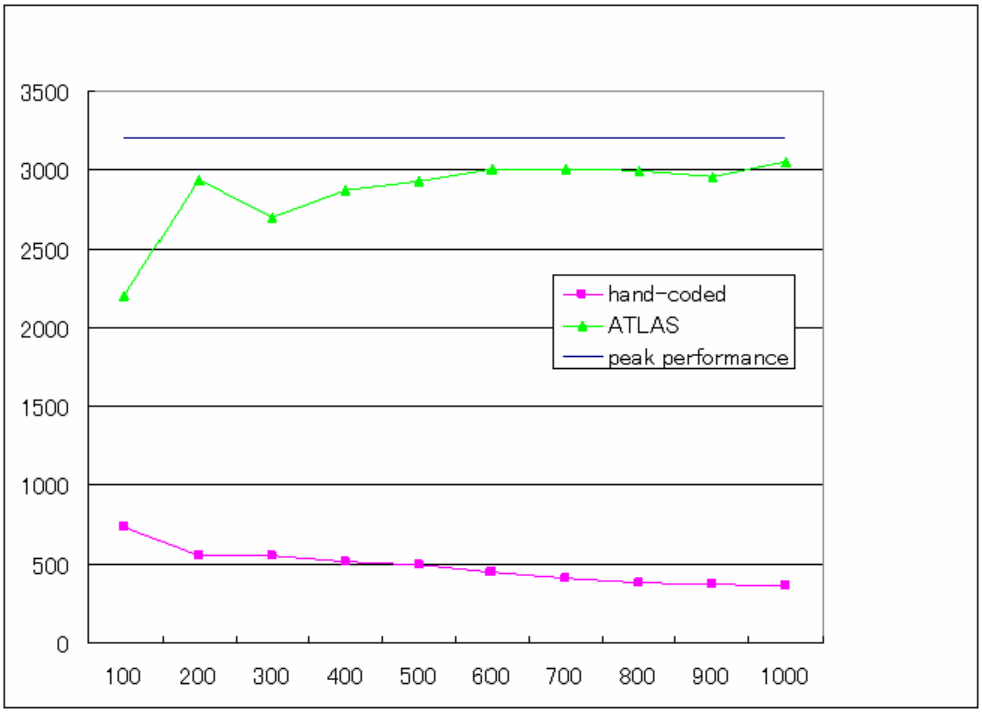

図 10　ATLAS の性能

4　アルゴリズムの工夫による演算量の削減

ここまでに紹介した最適化の方法は演算量は変えずに計算時間を短くする手法であったが、ここでは演算量そのものを減らすアルゴリズムを紹介する。

4.1　行列乗算のための Strassen アルゴリズム

- A,B,C をそれぞれ 2×2 に分割して乗算を行う
- 乗算の回数を 7/8 に削減可能

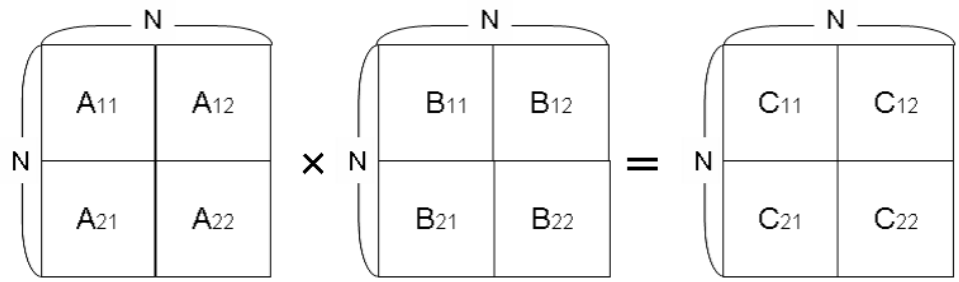

図 11　Strassen のアルゴリズム

$$P_1 = (A_{11} + A_{22})(B_{11} + B_{22}) \tag{1}$$
$$P_2 = (A_{21} + A_{22})B_{11} \tag{2}$$
$$P_3 = A_{11}(B_{12} - B_{22}) \tag{3}$$
$$P_4 = A_{22}(B_{21} - B_{11}) \tag{4}$$
$$P_5 = (A_{11} + A_{12})B_{22} \tag{5}$$
$$P_6 = (A_{21} - A_{11})(B_{11} + B_{12}) \tag{6}$$
$$P_7 = (A_{12} - A_{22})(B_{21} + B_{22}) \tag{7}$$

$$C_{11} = P_1 + P_4 - P_5 + P_7 \tag{8}$$
$$C_{12} = P_3 + P_5 \tag{9}$$
$$C_{21} = P_2 + P_4 \tag{10}$$
$$C_{22} = P_1 + P_3 - P_2 + P_6 \tag{11}$$

この様に計算する。

4.1.1　Strassen のアルゴリズムの計算量

$(\frac{N}{2})$ 行列× $(\frac{N}{2})$ 行列の乗算…7 回　　$2(\frac{N}{2})^3 \times 7 = \frac{7}{4}N^3$

$(\frac{N}{2})$ 行列同士の加算減算　…18 回　　$(\frac{N}{2})^2 \times 18 = \frac{9}{2}N^2$

総計算量　　$\frac{7}{4}N^3 + \frac{9}{2}N^2$

第 2 項が無視できるほど N が大きいとき、

$\frac{7}{4}N^3 + \frac{9}{2}N^2 \simeq \frac{7}{4}N^3 < 2N^3$

この様に $\frac{7}{8}$ の演算量で済む。$\frac{N}{2} \times \frac{N}{2}$ の行列の乗算で再帰的に Strassen のアルゴリズムを使用することにより、更なる計算量の削減が可能になる。しかし、加減算を多く繰り返しすぎると丸め誤差が増大するので、いくらでも使えばいいというものではない。誤差に対してあまり敏感でない問題に関して使用すべきアルゴリズムである。