ハイパフォーマンス
コンピューティング 49－1
（１９９３．１０．１４）

最近のワークステーションアーキテクチャと行列乗算性能について

前野年紀　太田昌孝

東京工業大学 総合情報処理センター

数種の計算機上で行列乗算プログラムを作成し、実行時間を測定し、評価した。行列乗算はN^2のデータにたいして$2N^3$の演算がおこなわれるので、スーパーコンピュータに比べてメモリバンド幅の小さいワークステーションでも、レジスタとキャッシュメモリを活用することでその演算性能を発揮できる。

素朴なプログラムではピーク性能の２%程度しか利用されていなかったのがキャッシュとレジスタのブロッキング、キャッシュコピーイング、ソフトウェアパイプライニングなどのプログラミング技法を駆使することで、各計算機の約７０%の演算性能を引き出すことができた。

Timing matrix multiplications on superscalar workstations

Toshinori Maeno, Masataka Ohta

Computer Center, Tokyo Institute of Technology
Oh-okayama 2-12-1, Meguro-ku, Tokyo 152, Japan

It is very important to use registers and cache memory for computation intensive algorithms, e.g. matrix multiplication, since the microprocessor performance growth unmatched with that of the memory in the last 8 years. We have compared matrix multiply programs on a few superscalar workstations, and found that the naive program could run at 2% or less of peak speed with many cache and TLB misses. Applying register/cache blocking, cache copying and software pipeling, we had made the program run faster, at 70% or more of peak speed.

1 はじめに

最近のワークステーションは演算速度は初期のスーパーコンピュータ CRAY-1 を追い越すほどに向上したが、メモリ速度はゆっくりとしか向上していないため[5]、キャッシュメモリを活用することの重要性が増している。[1, 7] また、高速演算も演算器の並列/パイプライン利用を前提としているため、活用には工夫が必要である。

行列乗算を題材にして、素朴なプログラムとハードウェアを活用したプログラムとを比較してみる。

2 素朴な内積型行列乗算

64 ビット精度の素朴な内積型プログラム(図 1)によって、行列 a,b の積を行列 c に作る。測定した計算機は Sparcstation-10/30(SS10), HP 715/33(HP), TITAN2-400(TITAN)の 3 機種である。大きさを 50 行 50 列あたりから 500 行 500 列まで変えて、時間を測定した(図 2)。行列サイズ[1]に対して、演算速度[2]をプロットしている。

2.1 測定結果

SS10 では行列の大きさが 40 のあたりで速度が低下したあと、180 近辺までほぼ一定、そこで急に低下した後、ゆるやかに低下していく。HP も SS10 に似ており、90-100 近辺で急に低下、250 近くまで一定、そこで急に低下後、ゆるやかに低下していく。

どちらも最初の速度の急低下はキャッシュミスの発生によるもので、2 番目の急低下は TLB ミスの発生によるものである。SS10 はキャッシュ容量が 16KB と小さく、TLB も 64 と少ない。キャッシュミスのコストは 12 サイクルと普通である。HP715 はキャッシュ容量が 64KB とやや大きく、TLB も 120 と多いが、キャッシュミスコストが 22 から 24 サイクルとやや大きい。

[1] NxN の行列乗算では浮動小数点演算は $2*N^3$ 回実行される。

[2] かかった時間を 1 マイクロ秒の間に実行された浮動小数点演算回数(MFLOPS)に換算

2.2 メモリ参照について

プログラムは i,j,k をインデクスとする 3 重のループからなっており、そのメモリ参照とキャッシュミスはつぎのようになっている。

1. 最も内側の k ループでは行列 a の i 行が順にアクセスされるが、行列 b の k 列は(行列の大きさの N おきに)とびとびにアクセスされる。

2. ひとつ外側の j ループでは行列 a については再び i 行をアクセスするので、(測定した範囲では)キャッシュが再利用される。行列 b の参照もキャッシュのラインサイズ分は再利用される可能性はあるが、j のループで参照する分全体をキャッシュに保持することはできないので、再利用回数は最大でも 4 程度になる。行列 c の要素の参照は j ループで 1 回ずつ順におきる。キャッシュミスも発生するが、参照回数が少ないので影響しない。

3. j のループの終了後、i ループでは行列 a,c の行の参照はつぎの行に移る。

3 改善

今回測定した計算機はスーパースカラ方式の複数命令発行とパイプライン化された浮動小数点演算器を装備しているのが特徴である。ただし、複数命令発行といっても浮動小数点命令はひとつしか発行できない。また、HP には加算と乗算を同時にできる命令がある。クロックが 33MHz の場合、毎サイクル加算と乗算を実行すると最大 66MFLOPS までの速度が可能である。測定では 500x500 の行列乗算速度は 1.07MFLOPS なので、1.6%しか発揮できていないことになる。すべてのデータがキャッシュに収容されている 52x52 以下の場合でも 17MFLOPS で最高性能はでていない。つまり、演算器はほとんど遊んでいることになる。

(パイプライン)演算器を活用する技法としてはアンローリングとソフトウェアパイプラインが知ら

```
#define N       500
#define R       1

double a[N][N],b[N][N],c[N][N];

main() {
  int i,j,k,r;
  double s;
        for(i=0;i<N;i++)
            for(j=0;j<N;j++) { a[i][j]=1.0; b[i][j]=2.0; }
        for (r=0; r<R; r++)
            for (i=0; i<N; i++)
            for (j=0; j<N; j++){
                s = 0.0;
                for (k=0; k<N; k++)
                        s += a[i][k] * b[k][j];
                c[i][j] = s;
                }
        printf("test end %f\n",c[N-1][N-1]);
}
```

図1　素朴な行列乗算プログラム

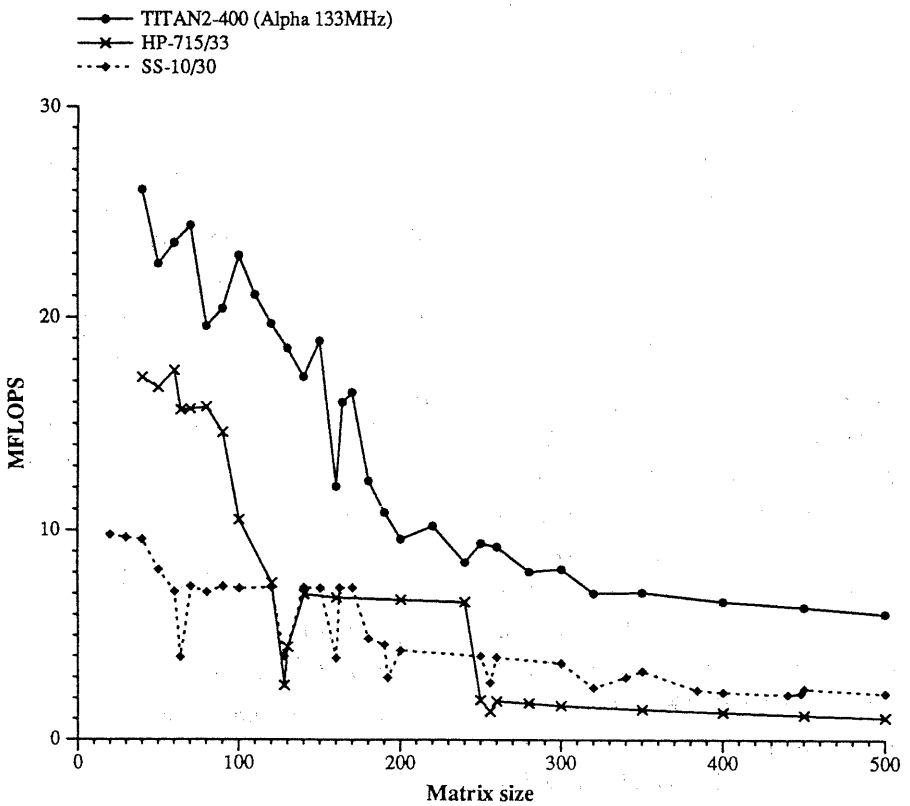


図2　行列乗算の速度

れている。また、キャッシュ活用法としてはキャッシュブロッキングとキャッシュコピーイングがある。

3.1 アンローリングとレジスタブロッキング

演算器を並行して働かせるために、ループ内の演算数を増やすには多数(16から32程度)装備されているレジスタを活用する。図1のプログラムをみると、ループ内で2個のロードにたいして2個の演算(HPでは1命令で実行可能)が行なわれていることがわかる。これではロードがボトルネックとなって、演算性能はあがらない。さいわい、行列乗算ではアンロール(展開)技法によりロードを減らすことが可能である。

展開するときにijkのどのループを対象とするか、何回展開するかなどはコンパイラの出力するアセンブラリストを見て、各計算機の持っているレジスタを最大限に活用するように決定した。[3] HP,DECでは一番内側のkループでi,jそれぞれについて4重、つまり16重の展開を行なった。これにより、8回のロードに対して16個の乗算加算が実行できることになった。SS10ではiについては2重、jについては4重に展開した。

アンロールはレジスタレベルのブロッキングでもあって、レジスタを使ってメモリ参照を削減しているともいえる。

3.2 キャッシュブロッキング

内積型プログラムでは行列aの参照はひとつの行が繰り返しアクセスされて、キャッシュが再利用される可能性が高いが、行列bは列方向に参照されるので再利用されるまえにキャッシュの自己干渉をおこして、キャッシュから失われている可能性が高い。再利用度を大きくするにはブロッキングを行なうとよいことが知られている。[2, 6, 4] ただし、ブロック(区画)の大きさはキャッシュの容量や属性に合わせて決定する必要がある。キャッシュの自己干渉が少ない範囲で、できるだけ大きい区画を選ぶのがよい。また、行列間の相互干渉もあるので、キャッシュ容量をcsとすると、$\sqrt{cs/2}$程度の大きさの区画を選ぶとキャッシュミス回数が最小となることが知られている[6]。(64KBつまり8K語のキャッシュを持つHPの場合では、64程度が適当となる。)

3.3 キャッシュコピーイング

行列の大きさ(例えばHPでは512x512のケースなど)とキャッシュ容量の関係によっては区画長が小さくても自己干渉を起こす場合がある。このような場合にそなえて、ブロックを連続領域にコピーすることで自己干渉を避けられる。ついでに、それに続けて行列aの行もコピーしておくと、a,c間の相互干渉も避けられる。これらによるキャッシュミス削減効果は大きいが、コピーの手間に見合うかどうかは行列のサイズ、キャッシュ属性などにもよる。ダイレクトマップ方式のHP,TITANと4重セットアソシアティブ方式のSS10でも有効であった。(IBM RS/6000-550でも有効であった。)

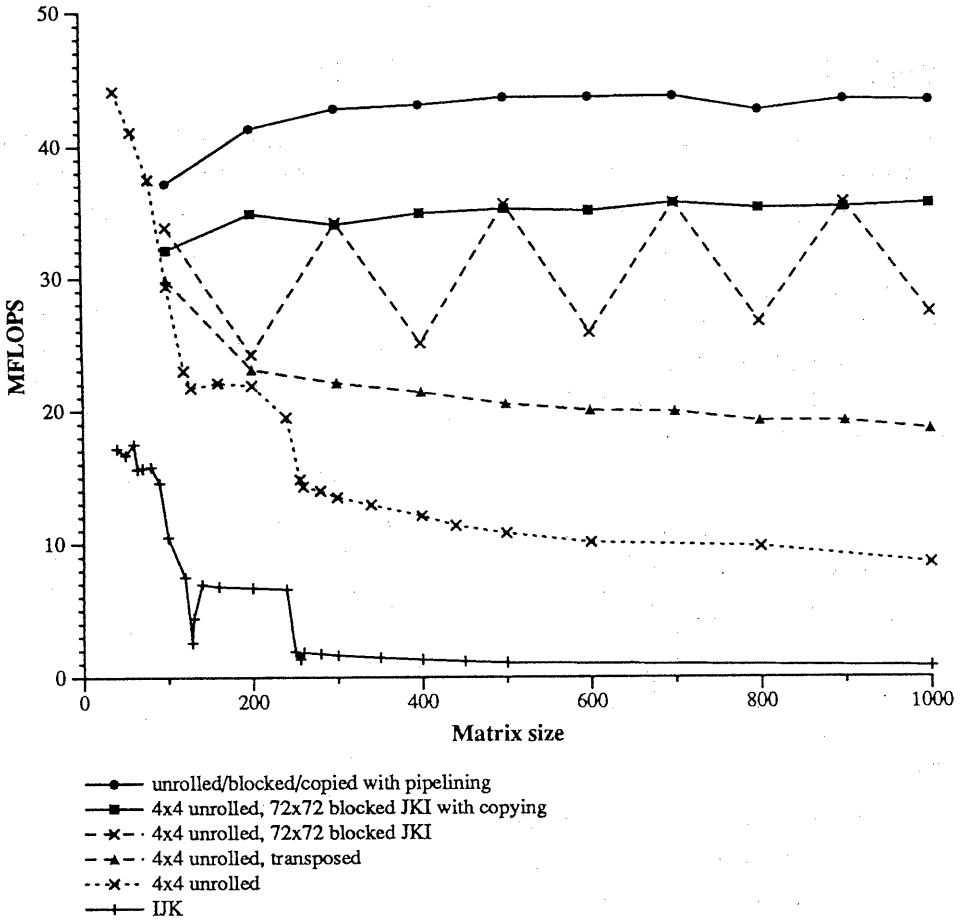
3.4 ソフトウェアパイプライン

レジスタブロッキングにより4x4重の展開を行なうことで最内側ループには各16の乗算と加算が含まれるようになった。しかし、HPでは先頭にロード命令が2つあって待ちが発生していた。ロードを一段前の繰り返しで前もって実行しておく[3]と、ループの先頭から毎サイクル演算(乗算加算)が実行できる。また、ループの始めと終りで別れて実行されていた2つずつの加算と乗算を乗加算命令で同時実行できる。この結果、HPでは20サイクルあった最内側ループを16サイクルのループに短縮できた。

3.5 改良されたプログラム

これまでのべた技法を適用してできあがったのが付録のプログラムである。

[3] ロードは演算と並列実行できる。



HP 715/33 Matrix Multiply speed

図3　行列乗算の速度(改良版)

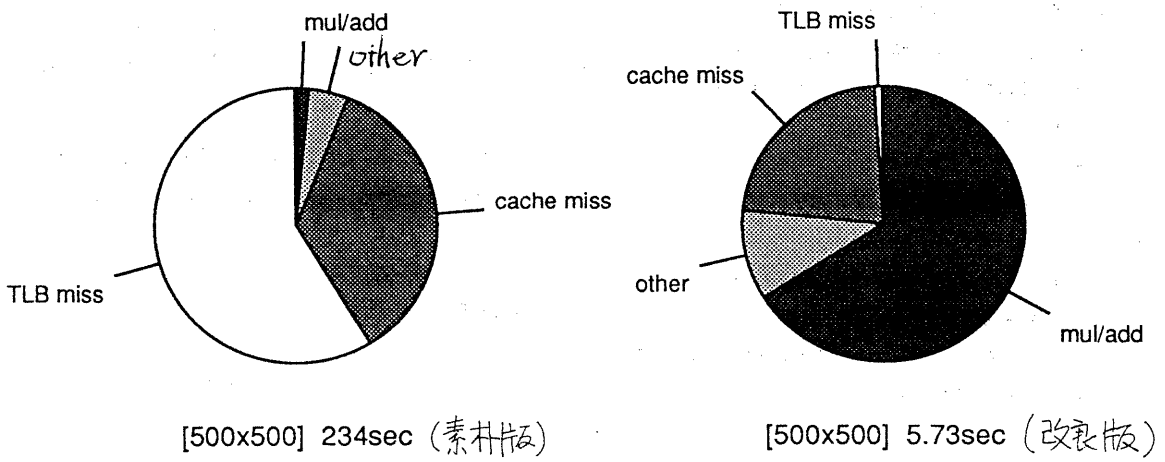


図4　行列乗算時間のうちわけ

4 改善結果

できあがったプログラムを使って測定を行ない、機種毎に効果のあった技法だけを残して測定しなおした(表1)。また、技法の効果をみるために、一部の技法だけを適用したプログラムも測定した。HPではすべての技法が有効であった(図3)。また、最初のプログラムと改良されたプログラムについて、500x500 の場合の所要時間の内訳を円グラフにした(図4)。当初 1.6%にすぎなかった浮動小数点演算時間が67%まであがっている。約40倍の速度向上が達成できた。

表1 500x500行列乗算速度(MFLOPS)

計算機	素朴	改良
SS10/30	2.2	24.7
HP715/33	1.1	43.6
TITAN2-400	6.0	71.5

5 結論

過去8年程度の傾向として、計算機の演算速度の向上に比べてメモリの速度はあまり向上していない。メモリバンド幅の小さいワークステーションは大規模計算に向いていないように思われているが、キャッシュを活用できれば、その演算速度を引き出すことができる。

キャッシュと演算器を活用する技法を適用した行列乗算プログラムを作成し、数種のスーパースカラ計算機上で、実行時間を測定して評価した結果、各計算機の限界性能の約70%の性能を引き出すことができた。

キャッシュとレジスタのブロッキング、キャッシュコピーイング、ソフトウエアパイプライニングなどのプログラミング技法が有効であった。

所要時間を分析すると、現在のメモリバンド幅でも演算速度の改善には意味があることがわかる。いっぽう、キャッシュやTLBを活用するためにアルゴリズムの改良が望まれる。

参考文献

[1] 寒川 光: 数値計算プログラミングにおけるデータ移動制御のためのブロック化アルゴリズム, 情報処理学会論文誌, **33**, No.10, (Oct. 1992), pp.1183-1192.

[2] E. Anderson ほか: LAPACK : A Portable Linear Algebra Library for High-Performance Computers, *Proceedings of Supercomputing '90, IEEE*, (1990), pp.2-11.

[3] D. Callahan, S. Carr, and K. Kennedy: Improving register allocation for subscripted variables, *Proceedings of the ACM SIGPLAN '90 Conference on Programming Language Design and Implementation*, June 1990.

[4] J.-Fr. Hake, W. Homberg: The Impact of Memory Organization on the Performance of Matrix Multiplication, *Supercomputing '90, IEEE*, (1990), pp.34-40.

[5] J. L. Hennessy, D. A. Patterson: *Computer Architecture A Quantitative Approach*, Morgan Kaufmann Publishers, Inc., ISBN 1-55880-069-8 (1990).

[6] M.S. Lam, E.E. Rothberg, M.E. Wolf: The Cache Performance and Optimizations of Blocked Algorithms, *ASPLOS IV, ACM*, April, 1991, pp.63-74.

[7] A. J. Smith: Cache Memories, *ACM Computing Surveys*, **14**, (1982), pp.473-530.

付録

```
/* JKIijk  with copying, alpha,omega-motion pipeline */
/* 1993-09-01  T. Maeno, M. Ohta */
/* size of matrix, spacing */
#define N       500
#define M       0
#define R       10
/* subarea for cache tiling */
#define Li      64
#define Lj      64
#define Lk      64
#define min(x,y)        (x<y ? x : y)

double a[N][N+M],b[N][N+M],c[N][N+M];
double d[Lk][Lj];
double e[4][Lk];  /* block copy area */

main()
{int i,j,k,k2,i2,j2,r;
int it,jt,kt;
int j1,k1;
double t00,t01,t02,t03;
double t10,t11,t12,t13;
double t20,t21,t22,t23;
double t30,t31,t32,t33;
double s0,s1,s2,s3;
double u0,u1,u2,u3;
double v0,v1;

        for(i=0;i<N;i++)
        for(j=0;j<N;j++) {
                a[i][j]=1.0; b[i][j]=2.0;
                }
    for (r=0; r<R; r++)
        for (j2=0; j2<N; j2+=Lj) {
                jt = min(j2+Lj, N);
          for (k2=0; k2<N; k2+=Lk) {
                kt = min(k2+Lk, N)-1; k1=kt-k2;
                for (k=k2; k<=kt; k++)
                for (j=j2; j<jt; j++) d[k-k2][j-j2]=b[k][j];
            for (i=0; i<N; i+=4) {
                for (k=k2; k<=kt; k++){
                        e[0][k-k2]=a[i][k]; e[1][k-k2]=a[i+1][k];}
                for (k=k2; k<=kt; k++){
                        e[2][k-k2]=a[i+2][k]; e[3][k-k2]=a[i+3][k];}
                for(j=j2; j<jt; j+=4) {
                   if (k2 == 0) {
                                t00=t01=t02=t03=0.0;
                                t10=t11=t12=t13=0.0;
                                t20=t21=t22=t23=0.0;
                                t30=t31=t32=t33=0.0;
                   } else {
                                t00=c[i][j];   t01=c[i][j+1];
```

```
                        t02=c[i][j+2]; t03=c[i][j+3];
                        t10=c[i+1][j];   t11=c[i+1][j+1];
                        t12=c[i+1][j+2]; t13=c[i+1][j+3];
                        t20=c[i+2][j];   t21=c[i+2][j+1];
                        t22=c[i+2][j+2]; t23=c[i+2][j+3];
                        t30=c[i+3][j];   t31=c[i+3][j+1];
                        t32=c[i+3][j+2]; t33=c[i+3][j+3];
                }
                j1=j-j2;
                s0=e[0][0]; s1=e[1][0];
                u0=d[0][j1]; u1=d[0][j1+1];
                u2=d[0][j1+2]; u3=d[0][j1+3];
                v0=s0*u0; v1=s0*u1;
                for(k=0; k<k1; k++) {
                     t00 +=    v0; t01 +=    v1; t02 += s0*u2; t03 += s0*u3;
                     t10 += s1*u0; t11 += s1*u1; t12 += s1*u2; t13 += s1*u3;
                     s0=e[0][k+1]; s1=e[1][k+1];
                     s2=e[2][k]; s3=e[3][k];
                     t20 += s2*u0; t21 += s2*u1; t22 += s2*u2; t23 += s2*u3;
                     t30 += s3*u0; t31 += s3*u1; t32 += s3*u2; t33 += s3*u3;
                     u0=d[k+1][j1]; u1=d[k+1][j1+1];
                     v0=s0*u0; v1=s0*u1;
                     u2=d[k+1][j1+2]; u3=d[k+1][j1+3];
                     }
                s2=e[2][k1]; s3=e[3][k1];
                t00 +=    v0; t01 +=    v1; t02 += s0*u2; t03 += s0*u3;
                t10 += s1*u0; t11 += s1*u1; t12 += s1*u2; t13 += s1*u3;
                t20 += s2*u0; t21 += s2*u1; t22 += s2*u2; t23 += s2*u3;
                t30 += s3*u0; t31 += s3*u1; t32 += s3*u2; t33 += s3*u3;
                c[i][j]  =t00; c[i][j+1]=t01;
                c[i][j+2]=t02; c[i][j+3]=t03;
                c[i+1][j]  =t10; c[i+1][j+1]=t11;
                c[i+1][j+2]=t12; c[i+1][j+3]=t13;
                c[i+2][j]  =t20; c[i+2][j+1]=t21;
                c[i+2][j+2]=t22; c[i+2][j+3]=t23;
                c[i+3][j]  =t30; c[i+3][j+1]=t31;
                c[i+3][j+2]=t32; c[i+3][j+3]=t33;
            }
        }
        }
        }
        printf("test end %f\n",c[N-1][N-1]);
}
```