ハイパフォーマンス
コンピューティング 58－8
（1995．10．18）

パソコンの行列乗算性能について

前野年紀　村上 弘†

東京工業大学 総合情報処理センター
†電気通信大学 電気通信学部

パソコンの演算性能を行列乗算とベクトル内積により測定し、ワークステーションと比べてみた。Pentiumパソコンの演算性能とメモリ転送性能は一段上の価格のワークステーションに匹敵することがわかった。

ただし、この性能を得るにはハードウェアの特徴を理解し、コンパイラの癖を知って、オプションを指定する必要があることや、プログラムの書き方を工夫する必要があるなど、コンパイラ(gcc)には改善の余地があることもわかった。

パソコンでもすでに演算性能とメモリ転送性能のアンバランスが起きており、キャッシュブロッキングなどのメモリ階層むけプログラミング技術が重要となっている。

Timing matrix multiplications on personal computers

Toshinori Maeno, Hiroshi Murakami

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

We have measured the time of matrix multiply programs on personal computers and workstations. The speed of personal computers are comparable with that of workstations.

It is very important to use registers and cache memory effectively for computation intensive algorithms, e.g. matrix multiplication, because of the memory bandwidth unmatching. Gcc has many things to do for generating improved code for pentium processors.

1 はじめに

FreeBSD などの 4.4BSD 系 UNIX 環境を使う目的で購入したパソコンで行列乗算速度を測ってみたところ、我々の予想を越えて速かった。数値計算にも使えるのではないかと思い、コードのチューニングを行なって測定し直した。

測定環境、中間積型行列乗算の速度、ベクトル内積計算の速度、キャッシュブロック化した内積型行列乗算の速度についてワークステーションでの測定結果とともに報告する。

2 測定環境

使用したパソコンは Intel の Pentium[1]を使ったもので、構成は

内部クロック 100MHz
外部クロック 66MHz
一次キャッシュ 8KB + 8KB (I+D)
(2 way set associative)
二次キャッシュ 256KB
ラインサイズ 32B
主記憶 32MB

である。コンパイラは gcc-2.6.3 を使用した。

Pentium の浮動小数点演算機構(FPU)はスタック構造を採用しており、浮動小数点レジスタ 8 個がスタックになっている。加算と乗算器はパイプライン化されていて、毎サイクル起動できる。結果を利用できるのは 3 サイクル後である。スタック操作命令では、ほぼすべての演算にトップが関係する。このため、トップが演算の隘路になる。これを緩和するために、スタック内のデータ位置を交換する FXCH 命令があり、Pentium では実行時間がゼロと高速化された。

比較参考のために

HP9000/755 (99MHz)
SONY NWS5000X (200MHz)
SPARCstation-10/51 (50MHz)
SPARCstation-5 (70MHz)

などのワークステーション 4 機種でも測定を行なった。

以下、行列乗算とは 64 ビット精度のデータからなる行列 a,b の積を行列 c に作るものとする。Pentium ではデータが 8 バイト境界が合っていないと遅くなるが、コンパイラ gcc はまだ Pentium に対応していないため、8 バイトデータの境界合わせをしてくれない。このため、ダミー変数を宣言することで境界合わせを行なった。

3 中間積型行列乗算

中間積型[1]のプログラム(図 1)では最内側ループで乗算と加算各 1 回、load が 2 回、store が 1 回実行される。演算の密度[2]が低いため、メモリ性能がみえてくる。大きさは 10 行 10 列あたりから 500 行 500 列まで変えて測定した(図 2)。

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。演算回数をかかった時間で割って、1 マイクロ秒の間に実行された浮動小数点演算回数(MFLOPS)に換算して、演算速度としてプロットしている。

3.1 測定結果

測定したシステムのキャッシュサイズは最小は 8KB 、最大(NWS5000)でも 1MB なので、行列サイズが 500x500 程度になると行列ひとつでもキャッシュには収まらない。SS5 を除いて、他は 10 〜 13MFLOPS とほぼ同程度になっている。キャッシュとメモリ間の転送発生により、全体の性能が落ちていくことが読みとれる。

4 ベクトルの内積

中間積型の演算は演算命令に比べてメモリ参照 (load/store) [2]命令が多い。ブロック化された行列乗算では演算命令の比率が高い内積型の方が速い

[1] ループが外側から添字 ikj の順になっているので、IKJ 型ともいう。

[2] Pentium は store に 2-3 サイクル必要とする。

```
matmul(n,a,b,c)
        int n;
        double *a,*b,*c;
{
        double *bk,s;
        int i,j,k;
        for (i=0; i<n; i++) {
            for (j=0; j<n; j++) c[j] = 0.0;
            bk = b;
            for (k=0; k<n; k++) {
                s = a[k];
                for (j=0; j<n; j++) c[j] += s * bk[j];
                bk += n;
            }
            a += n; c += n;
        }
}
```

図1 中間積型行列乗算プログラム

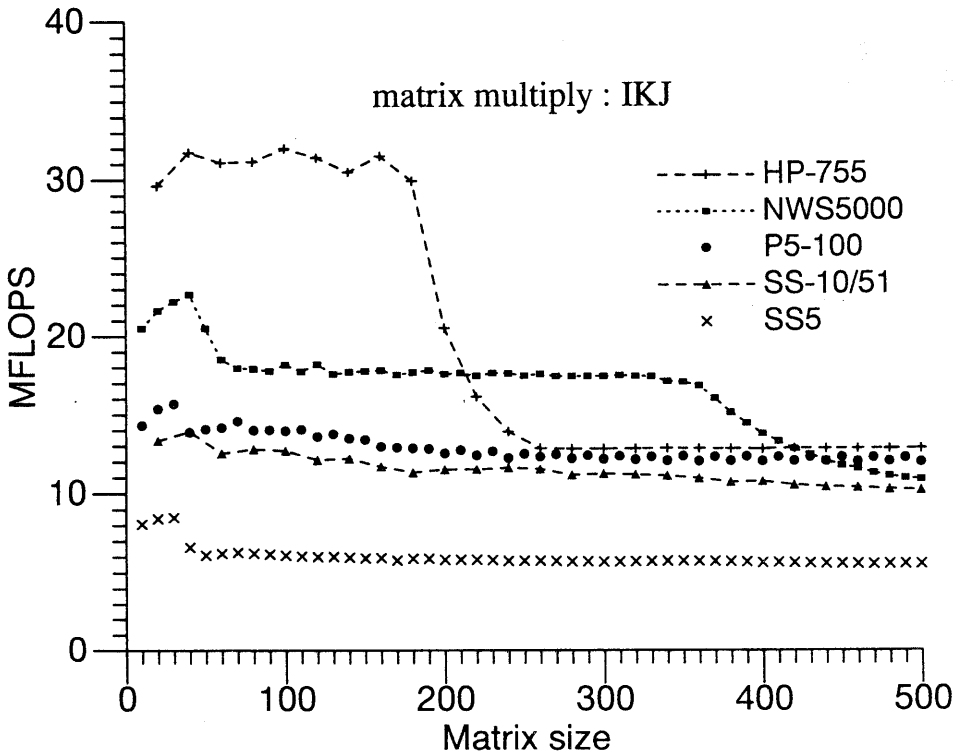

図2 行列乗算速度(中間積型)

```
L30:
  fldl  (%eax)
  fmull (%edx)
  -- 2 cycle --
  faddp %st,%st(1)
  addl  $8,%eax
  addl  $8,%edx
  cmpl  %ebx,%eax
  jle   L30
```

3-a dot

```
  fldl    (%edx)
  fmull   (%eax)
  -- 2 cycle  --
  faddp   %st,%st(1)
  fldl    8(%edx)
  fmull   8(%eax)
  -- 2 cycle --
  faddp   %st,%st(1)
  fldl    16(%edx)
  fmull   16(%eax)
  -- 2 cycle --
  faddp   %st,%st(1)
  ...
```

3-b dot-unroll

```
  fldl    (%edx)
  fmull   (%eax)
  fxch    %st(2)
  faddp   %st,%st(1)
  fldl    8(%edx)
  fmull   8(%eax)
  fxch    %st(1)
  faddp   %st,%st(2)
  fldl    16(%edx)
  fmull   16(%eax)
  fxch    %st(2)
  faddp   %st,%st(1)
  ...
```

3-c dot-opt

図3 内積計算のオブジェクト

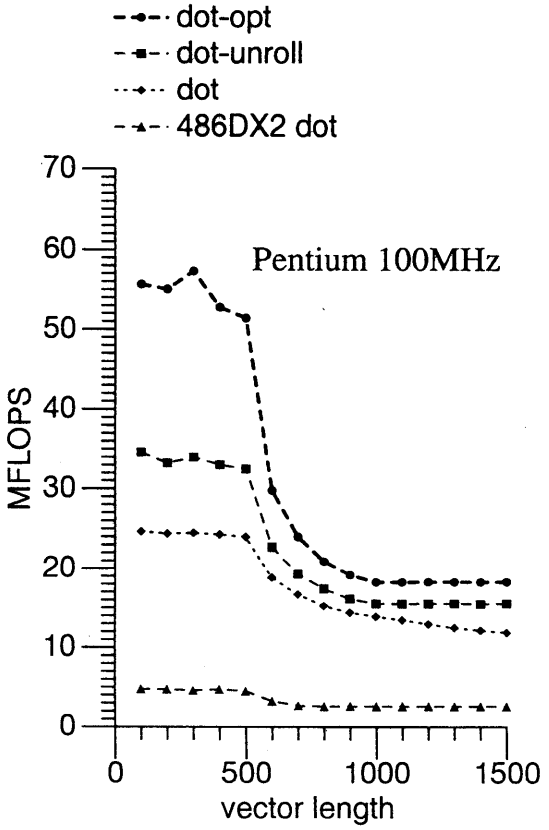

図4 内積計算速度(Pentium)

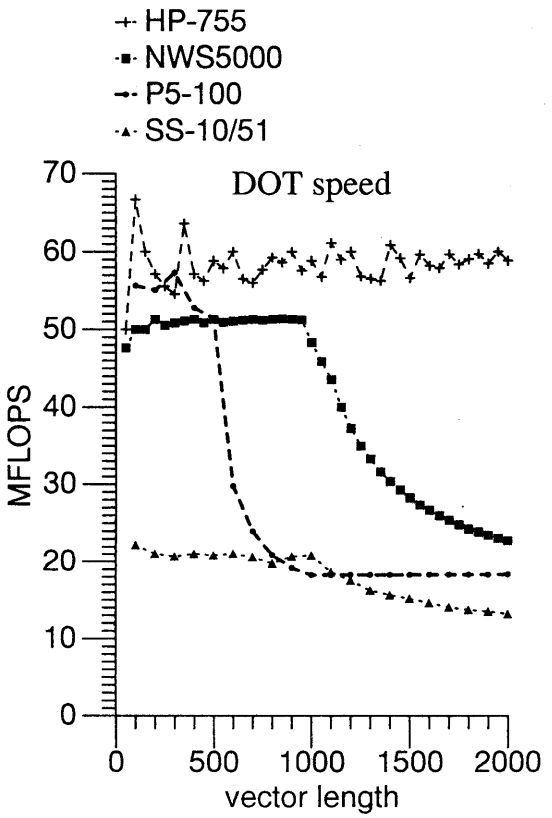

図5 内積計算速度

ことが多い。そこで、ピーク演算性能の目安をつけるためにベクトルの内積を以下のように計算する。

```
for (i=0; i<N; i++) s += a[i]*b[i];
```

ここでのチューニングにはアセンブラリストを出力させて、機械語プログラムレベルでの最適化程度をみる。

gcc で通常の最適化オプション(-O)を指定しただけの場合、オブジェクトプログラム(図3-a)を見るとおおいに改善の余地があることがわかる。ループのアンロールオプション(-funroll-loops)を指定することでループ本体が展開され(図3-b)、ループ管理のオーバヘッドが減る。また、直後の命令で演算の結果を使っているため、演算待ちがおきているが、FXCH 命令を使って演算の順序を入れ換えてやることで演算待ち時間を減らせる。これらにより、ほぼ最適のプログラム(図3-c)が出来上がった。

下位機種の 486DX2(66MHz)も合わせて測定した結果が図4である。キャッシュメモリに収まっていると、50MFLOPS 以上の演算性能がある。キャッシュミスするようになると、20MFLOPS まで低下する。

同様の測定をいくつかのワークステーションでも行なった。(図5) RISC ワークステーションでは演算パイプラインを活用するために、レジスタブロッキング技法を適用すれば、さらに高速化できることがわかっているが、ここでは行なっていない。

5 内積型行列乗算

ベクトルの内積演算速度からキャッシュ内と外での演算性能がわかる。これより、行列乗算にキャッシュブロッキング技法[5, 7, 9]が有効であることが推測できたので、ブロッキング技法を適用した内積型の行列乗算プログラムを作成し、測定した。(図6) 素朴な内積型ではサイズが大きくなるにつれ3MFLOPS 弱まで低下するのに対し、ブロック化されたものは 40MFLOPS 程度の性能を保っている。

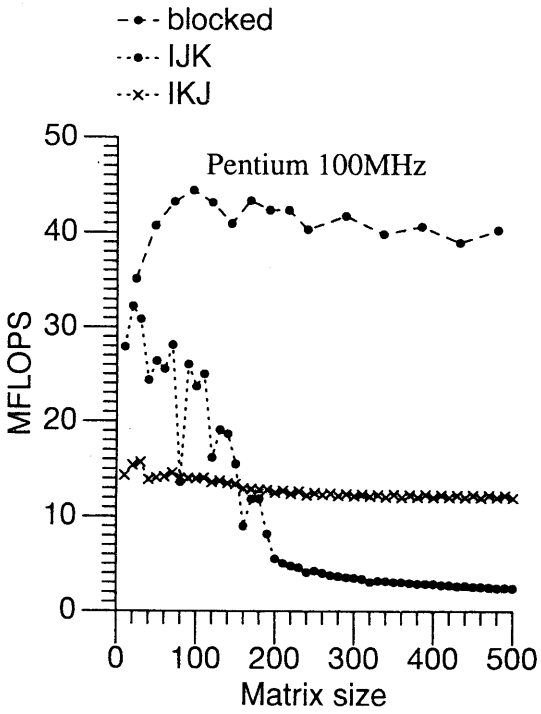

図6 行列乗算速度

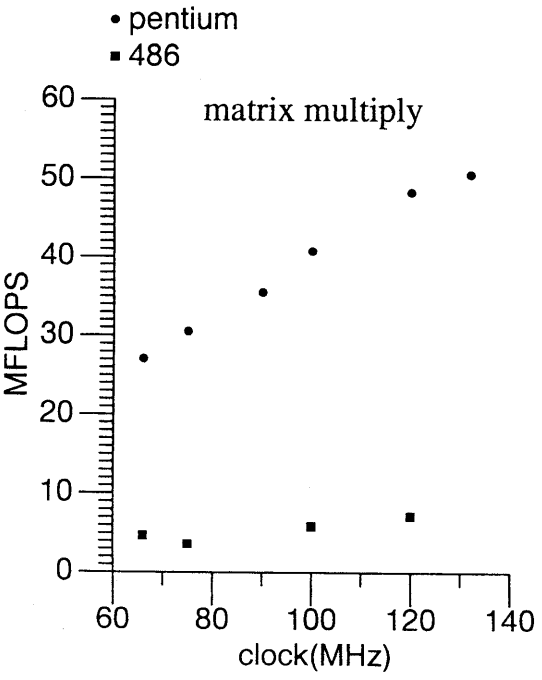

図7 ブロック化行列乗算速度

6 クロックとの関係

上記の行列乗算プログラムをネットワークニュースに投稿して測定者を募ったところ、Pentium と 486 を使ったシステムについて、クロックなどの異なる各種システムの結果を入手できた。(図 7) Pentium ではクロックに比例した性能向上が見られる。

7 おわりに

パソコンの性能を行列乗算によって測定した。それにより Pentium パソコンの性能はワークステーションの性能に見劣りしないばかりか、勝っていることもあることがわかった。

演算速度の向上に比べ、メモリ速度の向上は遅く、1995 年現在パソコンでもワークステーションと同様にメモリネックの傾向が見られる。キャッシュブロッキング、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 Pentium でも有効であった。メモリ階層を意識したアルゴリズムの開発が重要である。さいわいメモリ速度向上の重要性が認識されてきたため、バンド幅については強化されたメモリが提供されるようになってきている。

Pentium むけの gcc コンパイラにはデータ境界合わせや命令スケジューリングなどについて改善の余地がある。

参考文献

[1] Pentium ファミリー ユーザーズマニュアル下巻, アーキテクチャーとプログラミング, インテルジャパン株式会社.

[2] 寒川 光: *RISC* 超高速化プログラミング技法, 共立出版株式会社、ISBN 4-320-02750-7 (1995).

[3] 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.

[4] 前野年紀,太田昌孝: パイプラインとキャッシュを活用するためのプログラミング技法, 第 *35* 回プログラミングシンポジウム報告集,情報処理学会, (1994), pp.31-42.

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

[6] 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.

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

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

[9] 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.