ハイパフォーマンス
コンピューティング 64－2
（１９９６．１２．１２）

パソコンの行列演算性能の評価

前野年紀　村上 弘†

東京工業大学 総合情報処理センター
†東京都立短期大学 経営情報学科

最近のパソコンの演算性能を行列乗算を測定し、以前のものと比べてみた。演算性能の向上とメモリ転送性能の向上が読み取れる。演算性能はクロックの高速化とプロセッサ内部の演算の並列実行により向上した。一方、メモリ転送性能はメモリバンド幅の向上による。現在、これらの性能を得るにはハードウェアの特徴を理解し、プログラムの書き方を工夫する必要がある。このため、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の研究がより重要となっている。コンパイラにも改善の余地がある。

Matrix multiplication performance on current PCs

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 and BLAS1 routines on three personal computers, and compared the performance.

The memory bandwidth is improved much in the last year, however, it is still important to use registers and cache memory effectively to reduce memory access latency. So, the research to improve the compilers for generating fast code for the hierarchical memory systems should be encouraged.

1 はじめに

パソコンの演算性能が向上してワークステーション並になり [1]、メモリ性能とメモリ参照パタンによって全体の性能が左右されるようになった。[2, 3]

そこで、最新のパソコンで行列乗算性能を測定し、演算性能とメモリ性能を調査した。

測定環境を説明したあと、いくつかのメモリ参照パタンの異なる行列乗算プログラムの測定を行い、考察する。

2 測定環境

測定には2台の Intel Pentium[4] と Pentium Pro(以下 PPro) を使用した。

いずれも外部クロックは66MHz、一次キャッシュ容量は 8KB + 8KB (I+D) である。

2台の Pentium は

内部クロック 100MHz
二次キャッシュ 256KB
(Asynchronous SRAM)
主記憶 32MB Fast Page DRAM

のものと

内部クロック 133MHz
二次キャッシュ 512KB
(Pipeline Burst SRAM)
主記憶 32MB EDO DRAM

である。

PPro は

内部クロック 200MHz
二次キャッシュ 256KB
(CPU と同パッケージ)
主記憶 32MB EDO DRAM

の構成である。

これらはほぼ同一の命令セットを持っている。浮動小数点演算ユニットは浮動小数点レジスタ 8 個がスタックになった構造をしている。加算器と乗算器はパイプライン構造をしており、毎サイクル演算を開始できる。結果が利用できるまでに 3/4 サイクルかかる。スタック操作命令では隘路になるスタックトップとスタック中のデータとを交換する FXCH 命令があり、実行時間ゼロで実行できる。

Pentium は命令を順に実行していくだけであるが、PPro には以下の特徴がある。実行可能な命令から実行していく命令の動的並べ替え (リオーダリング) 機能と命令間のレジスタ干渉を減らすためのリネーミングレジスタである。[1]また、キャッシュミスの場合でも実行を続けるノンブロッキングキャッシュ機構がある。

3 プログラム

時間を測定する行列乗算プログラムは行列 a と b の積を計算して行列 c に加えるものを考え、データは 64 ビットからなるものとする。

行列の大きさは 20 行 20 列から 500 行 500 列まで 20 ずつ変えて測定した。

測定には以下の 5 種類のプログラムを使用した。(プログラムリスト)

IJK (内積型)
IKJ (中間積型)
JKijk (ブロック化内積型)
JKijk-copy (同上、コピー付き)
JKijk-o (同上、パイプライン化)

これらを上記の 3 機種で実行し時間を測定した。コンパイラはいずれも gcc-2.7.2 を使用した。gcc のオプションは以下の指定をした。

-O3 -m486 -funroll-loops

アンロールオプションによりループ本体が展開され、ループ管理のオーバヘッドが減る。PPro では命令の動的並べ替えを活用できる。

[1]これでプログラムで指定できるレジスタ数の少ないことを補っている。

```
matmul(n,a,b,c)   /*  IJK */
  int n;
  double *a,*b,*c;
{
  int i, j, k;
  double s, *ai, *bj, *ci;

  for (i=0; i<n; i++){
    for (j=0; j<n; j++){
      bj = &b[j];
      s = 0.0;
      for (k=0; k<n; k++) {
        s += a[k] * bj[0];
        bj +=n;
      }
    c[j] += s ;
    }
  a += n;
  c += n;
  }
}
```

内積型

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

中間積型

```
#define REAL double
#define B 20
matmul(n,a,b,c)      /* JKijk */
  int n;
  REAL *a,*b,*c;
{
  REAL s0,s1;
  REAL *aik, *bkj, *bkj2, *cij;
  int i,j,k;
  int i2,j2,k2;
    for     (j2=0; j2<n; j2+=B){
      for   (k2=0; k2<n; k2+=B){
        aik = &a[k2];
        cij = &c[j2];
        for (i=0; i<n; i++){
          for (j=0; j<B; j++){
            bkj = &b[k2*n+j2+j];
            s0 = 0.0;
            for (k=0; k<B; k++){
              s0 += aik[k] * bkj[0];
              bkj+= n;
            }
            cij[j] += s0;
          }
          aik += n;
          cij += n;
        }
      }
    }
}
```

ブロック化内積型

今回使ったPCでは64ビットデータが8バイト境界に合っていないとメモリ/キャッシュアクセスが遅くなる。しかし、コンパイラはこの境界合わせをしてくれなかったので、実行時に境界合わせを行なうようにプログラムしてある。

3.1 測定結果

大きさNxNの行列乗算では$2*N^3$回の浮動小数点演算が実行される。行列の大きさを横軸に、縦軸には1マイクロ秒の間に実行された浮動小数点演算回数(MFLOPS)をとって、図示してある。(図1,2,3)

どの機種も速度の違いはあれ、プログラムの性質を反映して似た形の結果となっている。

単純なプログラムIJKではサイズが大きくなるにつれキャッシュミスとTLBミスのため、急激に性能が低下する。(特にPentiumで目立つ) これをさけるにはキャッシュブロッキング技法[1, 5, 6, 8]が有効である。

ベクトル型プログラムIKJ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。IJKに比べて演算の密度[2]が低いため、メモリ性能が支配的になる。

キャッシュブロック化行列乗算JKijkでは演算命令の比率が高い内積型を使った。サイズが大きくなってもほとんど性能低下がおきないが、ところどころに谷が見える。キャッシュの自己干渉である。

小ブロックを連続領域に集めて(コピー操作)キャッシュの自己干渉を減らしたのがJKijk-copyである。谷が消えている。

命令を順に実行するPentiumではすなおなプログラムだと演算結果待ちが発生する。行列乗算に内在する並列性を利用することで浮動小数点演算器を並列動作させ、待ちが少なくなるようにしたのがJKijk-oである。PProでは命令リオーダリング等によって効果が少ない。

3.2 機種/世代の違い

クロックの違いを割り引いて測定結果を眺めてみると、メモリ性能の違いが読みとれる。非同期SRAMとFast Page DRAMを使った機種に比べて、同期SRAMやEDO DRAMを使った機種はメモリ転送性能が向上している。

しかし、メモリバンド幅が向上していてもキャッシュブロッキング、キャッシュコピーイング、などのプログラミング技法は有効であった。

ノンブロッキングキャッシュの効果は行列乗算に関しては不明である。

PProの命令の動的並べ替え機構はソフトウエアパイプライニング技法を使わないでもある程度の性能を保証してくれる。コンパイラの仕事が少なくなるように見えるが、実際にはループ内で使える総資源が重要なので、コンパイラの重要性には変わりはない。

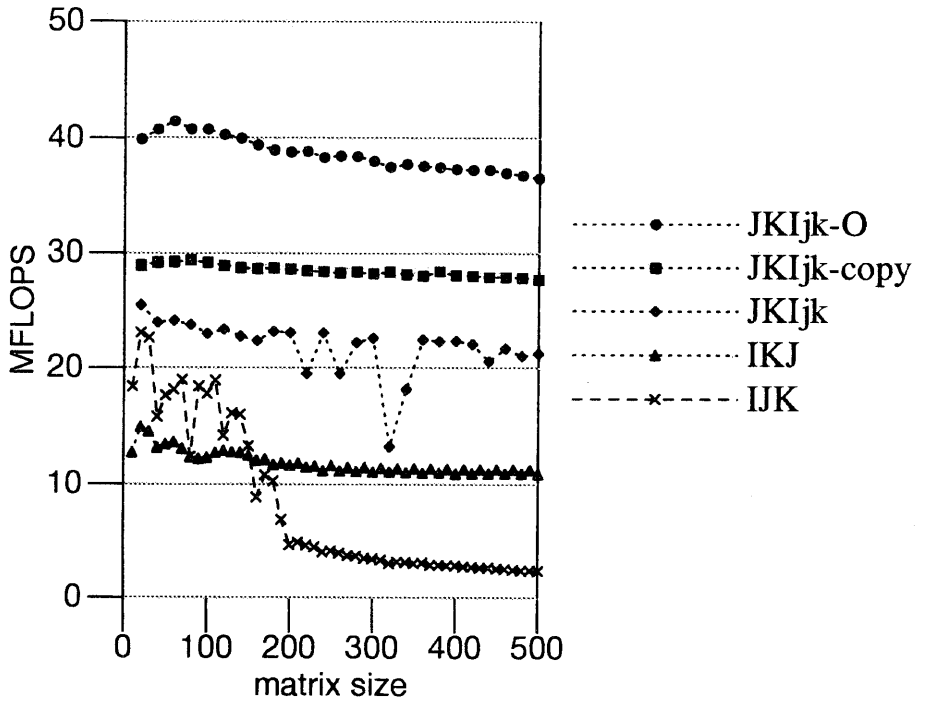
4 おわりに

パソコンの性能を行列乗算によって測定し、演算性能とメモリ性能の向上が続いていることを確認した。

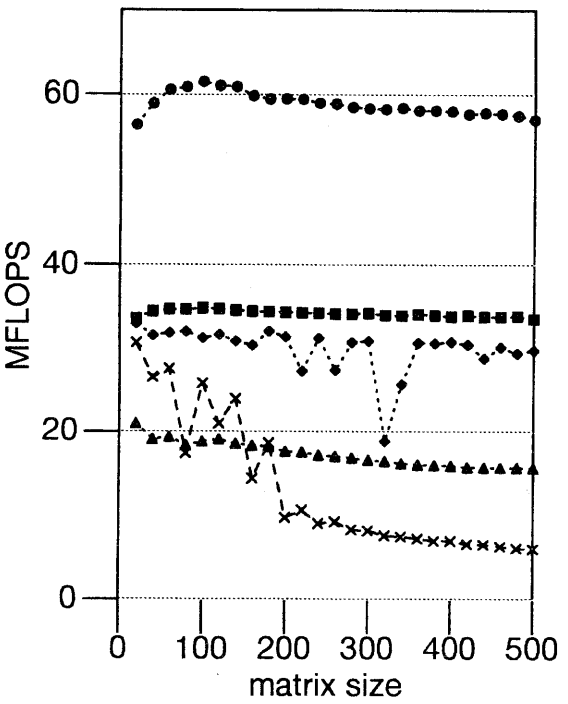
昨今、DRAMのアクセス速度にはわずかな向上しか見られず、メモリアクセスのボトルネックがはっきりしてきた。それを補うために、パソコンでもメモリバンド幅を向上させる努力が払われていることが分かった。同期SRAMやEDO DRAMに続いて、SDRAMが使われるようになる日も近い。

メモリバンド幅を有効利用するために、メモリ階層に注目したプログラミングの重要度が増すであろう。

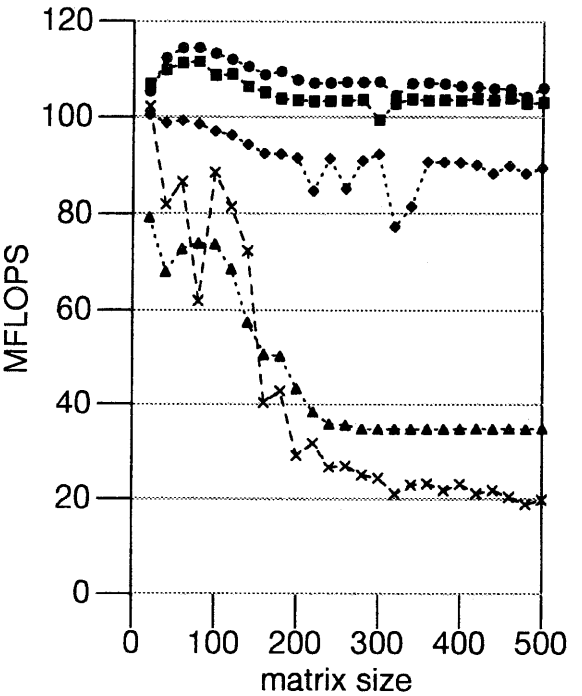
命令の動的スケジューリングを前提にしても、コンパイラの命令スケジューリングも重要である。



Pentium 100MHz



Pentium 133MHz



Pentium Pro 200MHz

参考文献

[1] 前野 年紀, 村上 弘: パソコンの行列乗算性能について, **第 *58* 回 *HPC* 研究会資料, 情報処理学会**, (1995), pp.45-50.

[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] 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.

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

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

[8] 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.


```
#define REAL double
#define B 20
REAL bb[B][B];
matmul(n,a,b,c)  /* JKIjk */
  int n;
  REAL *a,*b,*c;
{
  REAL *aik, *bkj, *bkj2, *cij;
  REAL s0, s1, r0, r1;
  int i,j,k;
  int i2,j2,k2;
  for     (j2=0; j2<n; j2+=B){
    for   (k2=0; k2<n; k2+=B){
      for (k=0; k<B; k++) {
        bkj = &b[(k2+k)*n+j2];
        for (j=0; j<B; j++) {
        bb[j][k] = bkj[j];
        }
      }
      aik = &a[k2];
      cij = &c[j2];
      for (i=0;  i<n;  i++){
        bkj = &bb[0][0];
        for (j=0; j<B; j++){
          r0 = aik[0]*bkj[0];
          s0 = 0.0;
          r1 = aik[1]*bkj[1];
          for (k=2; k<B; k+=2) {
            s0 += r0;
            r0 = aik[k]*bkj[k];
            s0 += r1;
            r1 = aik[k+1]*bkj[k+1];
            }
          bkj += B;
          s1 = cij[j];
          s0 +=  r0 + r1;
          cij[j] = s0 + s1;
          }
        aik += n;
        cij += n;
        }
      }
    }
}
```

ブロック化内積型
パイプライン化