愛知工業大学研究報告
第３５号Ｂ，平成12年

ノート

pthreadによる行列計算の高速化の試み

On a matrix calculation using pthread

小池 慎一† 山住 富也††
Shin-ichi KOIKE Tomiya YAMAZUMI

Abstract

We tried to calculate matirx multiply and inverse using pthread on linux. CPU has 2 proccecors. Then, we expected to speed up more 1 proceccor system. Where, a size of the matrix is 1000 by 1000. We get, if size of thread is not samll, then calculation time is about half. But size is so small, then so slow.

1 はじめに

PCの価格が年率1/2と言う勢いで下落し、研究室レベルでも、学生一人当たり1台を越えるようにり、PVMなどのように複数のPCを平行に動かして計算処理の高速化が計ることが可能になった。また、2CPUのPCについても、linuxを搭載することによりpthreadがユーザがプログラムから利用できて、その性能を引き出すことが可能になった。

そこで、数値計算の並列処理の例として行列積と逆行列の演算を例にとって、その性能評価を試みた。具体的には、2CPU、SMP構成のパソコンにlinuxのカーネル2.2を搭載して、gccコンパイラを用いて、threadの効果を検証した。また、市販のPGI並列処理コンパイラとも比較した。

その結果、threadのサイズ、すなわちthreadが処理する乗算の回数が小さくなければ、ほぼ2倍の計算速度を得た。反対に、threadのサイズが小さい場合には、計算速度は減少した。

2 threadとは

一般にunix系のOSでは、動いているプログラムをprocessと言う。マルチユーザ・マルチタスクのシステムでは、常時数十個のプロセスが走っている。

プロセスは、プログラムカウンタ、状態フラグ、各種レジスタ、メモリなどを保有しており、その意味では、1個のプロセスは、1個のシングルタスクのコンピュータとみなせる。ということは、プロセスは生成・消滅にかなりの手間が掛かり、重いと言われる。

それに対してthreadは、プロセスの内部で、メモリ、状態フラグなどを共有し、実行のみが平行になされるモジュールであり軽い。

並列処理は、複数のプロセスを生成することによっても可能であるが、数値計算処理などは変数(メモリ)を共有して処理するのに適しているので、threadを利用する効果が期待できる。また、threadによる処理の分割は、processによるものよりも、コード化が簡明でわかりやすい。

†愛知工業大学計算センター（豊田市）
††名古屋文理大学情報文化学部（稲沢市）

linuxにはpthread(POSIXで規程されたthread)が用意されている。基本的な使用法は、

1. pthread_create スレッドの生成

2. pthread_join 同期をとるためのスレッドの終了を待ち

の関数を必要に応じて呼び出すことである。ここに、同期とは、threadに分けられた複数の処理が共に終了しないと次の処理に進めないプログラムの位置において、終了待ちをすることである。

3 行列積と逆行列の演算による検証

threadの効果を調べるために、行列積と逆行列の演算を実際に行わせてテストしてみる。以下の節でも述べるがおのおのの演算には次の特徴がある。

1. 行列積 演算結果は他の要素の処理には使用されない

2. 逆行列ピボット毎に、すべての行の処理が必要
   あるピボットに対する処理の結果は、次のピボットの計算に影響する。

3.1 行列積

大きさ$n \times n$の行列$\mathbf{A}$と$\mathbf{B}$の積を行列$\mathbf{C}$に得る演算において$\mathbf{C}$の要素c_{ij}は

$$c_{ij} = \sum_{k=0}^{n} a_{ik} \times b_{kj}$$

にて計算される。ここにおいて、右辺には左辺の要素c_{ij}を含まないので、c_{ij}の計算は、すでに計算された結果に影響されない。

これを通常の方法で計算するには、例えば、添え字(i,j)に対して

```
c[i][j]=0;
for(k=0;k<n;k++)
  c[i][j] += a[i][k] * b[k][j];
```

のようにコード化する。これは、添字に関して数学的な定義をそのままコード化したものである。

$c_{i0}, c_{i1}, \ldots, c_{in}$ を同一の thread で処理する場合には、1行の処理をする関数を get_a_row(int i) として

```
for(i=0; i<n; i++)
  pthread_create(&thread[i],NULL,
            (void*)get_a_row, &row[i]);
for(i=0; i<n; i++)
   pthread_joein(thread[i],NULL);
```

のようにコード化される。すなわち、n 個の thread を順次生成し、thread_join にてそれらの終了についての同期をとる。もっとも、行列積の場合には演算は互いに独立なので、同期をとならくとも正しい結果を得る。

thread の構成は、このように、行毎に1個の thread にしてもよいが、PC が 2CPU のシステムなので、前半の 1/2n 行と後半の 1/2n 行をまとめて2個の thread にして、並列処理の効果があると期待できる。そこで、この2通りの処理についてテストする。

実際に大きさ $n = 1000$ の正方行列の行列積を計算させた結果を以下に示す。以下の表で、thread の数が1とは、プロセスそのものが1個の thread と見なせることを意味するので、本文の意味では thread を利用しないことに相当する。

表1．大きさ 1000x1000 の行列積

thread の数	乗算回数/thread	実行時間 (sec)	比率
1	n^3	399.2	1.00
2	n^3/2	225.4	0.56
n	n^2	212.4	0.53

(注)n^3 は n の3乗を意味する

この結果より、thread を利用することにより処理時間は53%～56%に減少することがわかる。2CPU のシステムで、約 1/2 の処理時間が得られた。thread の利用の効果は大きいといえる。

3.2 逆行列

逆行列の演算について thread の効果を調べた。

行列は大きさ $n = 1000$、すなわち 1000×1000 の正方行列として、疎行列ではない通常の行列を掃き出し法で演算した。性質の悪い行列に対する行と列の入れ替えなどはせず、対角要素を pivot とする、教科書的な算法を用いた。

a_{ii} を pivot とすると、i 行を除いたすべての行 $j(j \neq i)$ に対して

$$c_{jk} = a_{jk} - a_{ik}a_{ji}, \quad (k = 0, 1, \ldots, n)$$

を計算する。行列 $\mathbf{C} = (\mathbf{c_{ij}})$ が逆行列である。

この場合、pivot a_{ii} に対して、残りの n-1 行の処理がすべて済むまでは次の pivot の計算に移れない。したがって、この j 行を処理する関数を inv_sub とすると、

```
/* 以下のコードでは j<>i の処理等は略されている*/
for(i=0; i<n; i++){
  /* a[i][i] を pivot とする*/
  for(j=0; j<n; j++)
    pthread_create(&thread[j],NULL,
              (void *)inv_sub, &p[j]);
  for(j=0; j<n;  j++)
    pthread_join(thread[j], NULL);
}
```

のようになる。すなわち、pivot a_{ii} に関して、すべての行が終了するまで、pthread_join 関数で待ち合わせる。

これに対して、1個の pivot に対して、前半の n/2 行と後半の n/2 行をまとめて2個の thread にした場合と比較した。実行結果を以下に示す。

表2．大きさ 1000x1000 の逆行列

thread の数	乗算回数/thread	実行時間 (sec)	比率
1	n^3	499.0	1.00
2n	n^2/2	257.7	0.52
n^2	n	475.7	0.95

上の結果より、1個の pivot につき、2個の thread で処理した場合がほぼ 1/2 の所用時間となり、予期した結果を得た。しかし、thread を1行毎に小さく分けた場合には 2CPU の効果が現れなかった。このことについては、検討結果を考察にて述べる。

3.3 並列コンパイラとの比較

上の結果を 2CPU の PC をもサポートしている市販の PGI コンパイラと比較してみた。上の表の thread の数が1のコードをコンパイルして実行させた。以下に結果を示す。

表3.PGI コンパイラとの比較 (sec)

	thread=2 の場合	PGI コンパイラ
行列積	212.4	82.4
逆行列	257.8	170.4

いずれも、PGI コンパイラが優れている。PGI コンパイラの内部処理は不明であるが、linux の gcc コンパイラに関しては、数値計算の立場からみた最適化については、良い評価がなされてないことの裏付けとなった。

4 高速化のためのコードの改良

行列計算では、添え字計算の高速化が重要な問題となる。行サイズが n の配列 a[i][j] の実効アドレス EA は、

配列の先頭アドレスをaとすると

$$EA = a + i * n + j$$

でなされる。コンパイラがコードから得られる情報を用いて最適化しない場合には、配列の要素1個毎にこの計算がなされる。もし、配列のアドレス計算の高速化が計られれば、全体の処理時間は短縮される。

C言語では、ポインタ型の変数が利用できるので、インクリメント演算と組み合わせて配列の実効アドレスの計算が以下のように簡単になる。2.1節で示したコードをポインタを用いて書き直すと以下のようになる。

```
/* Ap,Bp,Cp を配列 A,B,C の型のポインタとする*/
   Cp=&C[i][j]; Bp=&B[0][j];  Ap=&A[i][0];
   *Cp=0;
  for(k=0;k<n;k++){
     *Cp=*Cp+*Ap * *Bp;
     Ap++; Bp+=n;
  }
```

逆行列についても同様な処理を行うと以下の結果を得た。

表4. 大きさ1000x1000の行列積(ポインタ使用)

threadの数	乗算回数/thread	実行時間(sec)	比率
1	n^3	103.6	1.00
2	n^3/2	54.1	0.52
n	n^2	53.2	0.51

表5. 大きさ1000x1000の逆行列(ポインタ使用)

threadの数	乗算回数/thread	実行時間(sec)	比率
1	n^3	169.0	1.00
2n	n^2/2	112.5	0.66
n^2	n	318.7	1.89

行列積の場合には、threadを使用しない場合にはなおPGIコンパイラの方が高速であるが、threadを使用するとそれを上回る。また、逆行列では、threadを使用しない場合でもPGIコンパイラに匹敵し、threadを使用すると、それを上回る。しかし、threadの数を多くするとthreadを使用しない場合よりも遅くなる。

5 考察および結論

以上のデータより、2CPUシステムの場合、threadを用いるとほぼ2倍の実行速度が得られることが確かめられた。しかし、逆行列の場合にはthreadを増やすとかえって遅くなる。この点に関しては、thread生成の時間がかかるのではないか、と考えられる。

逆行列の場合、ポインタを使用しない場合の乗算1回にかかるアドレス計算も含めた平均時間をtm_1、thread生成に要する時間をtsとして連立方程式を立てると以下のようになる。

$$\begin{aligned}(n^2/2 \times tm_1 + ts) \times 2n &= 257.7\\ (n \times tm_1 + ts) \times n^2 &= 475.7\end{aligned}$$

を解いて、$tm_1 = 0.257(\mu sec)$, $ts = 22(msec)$を得る。同様に、ポインタを使用した場合には、乗算に掛かる平均時間をtm_2とすると

$$\begin{aligned}(n^2/2 \times tm_2 + ts) \times 2n &= 112.5\\ (n \times tm_1 + ts) \times n^2 &= 318.7\end{aligned}$$

を解いて、$tm_2 = 0.357(\mu sec)$, $ts = 21(msec)$を得る。

どちらについても、tsの値がほぼ同じなので、この評価は妥当であると判定する。

この場合のthreadの生成時間の総計は、$n = 1000$より、$ts \times 10^6 = 200(sec)$となり、threadの発行回数が多い場合に実行時間が大きくなることを説明する。

また、threadの寿命は、ポインタを使用しない場合、それぞれ$n^2/2 \times tm_1 + ts = 128.5(msec)$と$n \times tm_1 + ts = 0.257 + 0.2(msec)$となる。

以上より、threadの寿命が短い、すなわち、threadで処理する計算量が少ない場合には、threadを発行するコストが多く掛かりthreadを利用する利益はない。したがって、コーディングに際しては、1個のthreadの分担する計算量について十分な検討を要する。

謝辞

linuxのインストールおよびthreadを利用可能にするためのカーネルの再構築等に人力頂いた、卒研生の佐原君に謝意を表する。

参考文献

[1] B.Nichols, D.Buttlar, J.P.Farrell (榊正憲訳) "Pthreads プログラミング" オライリー・ジャパン (1998)

(受理 平成12年3月18日)