ハイパフォーマンス
コンピューティング 73－9
（1998．10．9）

SMPクラスタにおけるコレスキー分解の並列化

佐藤茂久 草野和寛 松田元彦 田中良夫 佐藤三久

新情報処理開発機構 つくば研究センタ

概要

SPLASH2の中のCholeskyを、Solarisスレッドとリモートメモリ転送を用いて並列化し、SMPクラスタCOMPaS上で性能を評価した。COMPaSは4個のPentiumProを持つノードをMyrinetを用いて8台接続している。ノード内の並列処理はSolarisスレッドを用いて行ない、ノード間の並列処理はMyrinetを通したリモートメモリ転送を行なうSPMD方式で実現した。Choleskyは疎行列のコレスキー分解を行なうプログラムで、不規則に通信が発生し、グローバルな同期を持たないという特徴を持つ。同数のプロセッサを持つ分散メモリシステムとSMPクラスタでは、後者の方がデータ共有のオーバーヘッドは小さくなるが、Choleskyの場合、負荷の不均衡による性能の低下が大きかった。

Parallel Cholesky Factorization on a SMP Cluster

Shigehisa Satoh, Kazuhiro Kusano, Yoshio Tanaka,

Motohiko Matsuda, and Mitsuhisa Sato

Tsukuba Research Center, Real World Computing Partnership

Abstract

We have parallelized Cholesky in the SPLASH-2 suite using Solaris threads and remote memory operations, and evaluated the performance on a SMP Cluster COMPaS. COMPaS consists of eight SMP nodes connected by Myrinet, where each node has four Pentium Pro processors. Cholesky is a sparse cholesky factorization kernal which is characterized by high communication to computation ratio and no global synchronization between steps.

1 はじめに

低価格で高性能なマイクロプロセッサと、高速なネットワークの普及により、SMPクラスタを用いた高性能計算が注目されている。

SMPクラスタでの並列処理は、ノード内（共有メモリ）とノード間（分散メモリ）で行なわれる。ノード内での並列プログラミングにはスレッド(POSIX,Solaris)が広く利用されている。一方ノード間の並列プログラミングには、MPIのようなメッセージパッシングによる方法やリモートメモリ参照を用いた方法がある。

プログラミングの容易さでは、共有メモリプログラムの方がメッセージパッシングやリモートメモリ参照よりも優れている。そのため、グローバルなアドレス空間上で書かれた並列プログラムを、分散メモリシステム上で実行するための、分散共有メモリシステムの研究も行なわれている。

SPLASH-2[1]はグローバルなアドレス空間を持つ並列システムの研究のための並列プログラム集である。ここではSPLASH-2の中のCholeskyを取り上げ、Solarisスレッドとリモートメモリ参照を用いて書き換え、SMPクラスタ上で性能の評価結果について報告する。

Choleskyのアルゴリズムは、不規則な通信が頻繁に発生し、計算の過程でグローバルな同期を持たないという特徴を持つ。ここでの目的は、疎行列のコレスキー分解の効率の良い並列アルゴリズムを求めることではないので、アルゴリズムの変更は極力行な

わずに、並列化を行なう。

以下の各節で、まずSMPクラスタCOMPaSの概要とCholeskyの特徴を述べた後、SMPクラスタ向けの並列化方法を述べ、性能評価と考察、そして最後にまとめを記す。

2 SMPクラスタCOMPaS

SMPクラスタCOMPaS[3]の各ノードは、共有バスを介して接続された4個のPentiumProプロセッサ(200Mhz)からなる。このようなノードが8個、Myrinetを介して接続される。各プロセッサは8+8KBの1次キャッシュと512KBの2次キャッシュを持ち、128MBの主記憶をノード内で共有する。

OSはSolaris 2.5.1が搭載され、ノード内の並列処理にはSolarisスレッドを用いることが出来る。ノード間の並列処理のためには、Myrinet上の通信レイヤとしてNICAM[4]を用いたリモートメモリ転送を行なう。

これまでのCOMPaSを用いた研究から、規則的な計算を行なうプログラムで高性能を得るためには、次のような方法が有効であることがわかっている[3][4][5]。まず、ノード内の並列処理では共有バスが性能のボトルネックとなるため、キャッシュ局所性の高め、バスのトラフィックを少なくする必要がある。一方、ノード間の並列処理では、通信と計算をオーバーラップさせ、通信レイテンシを隠蔽することが重要である。

3 コレスキー分解

本節ではSPLASH-2のカーネルプログラムの一つであるCholeskyの特徴を述べる。

コレスキー分解は、正定値対称行列$A \in R^{n \times n}$を下三角行列Lを用いて$A = LL^t$に分解する。大規模な疎行列のコレスキー分解は、有限要素法や、線形計画法における内点法などで必要とされる。これらの問題に現れる連立一次方程式系の係数行列は、構造に規則性を持たない疎行列である。

SPLASH-2のCholeskyで実装されたアルゴリズムは、E. Rothbergのブロック化ファンアウト法に基づいている[2]。疎行列のコレスキー分解は、一般に次のような手順で行なわれる。

消去木の作成: 疎行列の構造から、各行の間の依存関係を表す木を作る。この木のノードは行列の各行に対応する。

順序付け: 非零要素を少なくするため、あるいは並列性の向上などのために行の順序を変更する。

スーパーノード化: 構造の同じあるいは似た列同士を一つのスーパーノードに併合する。以下ではこのスーパーノードを消去木のノードとする。

記号分解: 元の行列の構造から、分解によって得られる下三角行列(L)の構造を求める。

数値分解: 下三角行列の各非零要素の値を求める

このうち、実行時間の大半を占めるのは数値分解であるため、数値分解のみを並列に実行し、その他は逐次処理される。また、Choleskyの中では順序づけは行なわない。数値分解は、次の二段階で行なわれる。

まず、消去木のあるノードに属する要素の値の計算には、消去木でそのノードの子孫にあるノードの要素の値のみを必要とする。そのため、消去木を根に近いいくつかのノードで分割し、そのノードの子孫に当たる各部分木をプロセッサに割り当てる。そうすると、その部分木内の非零要素の計算はそのプロセッサ内でローカルに行なえる。この部分木をドメインと呼ぶ。従って、消去木のノードは、ドメインのいずれか一つに属するものと、ドメインに属さないものに分かれる。

上記の分割で分割点となったノードとその先祖の計算にはその子孫となるノード(ドメイン)の計算結果を必要になる。それらの計算は複数のプロセッサにまたがって行なわれるので、計算結果を共有する必要がある。ドメインに含まれないノードに属する非零要素の計算は、疎行列をブロック化して、各ブロック内の要素を集めた密行列の分解として行なう。この部分がブロック化ファンアウト法と呼ばれる。ドメイン分割を行なわずに全行列をブロック化して計算することも出来るが、ローカルな計算はブロック化せずに計算した方が効率が良いため、このようなドメイン内の計算とブロック毎の計算の二段階で行なっている。プロセッサ数が多いほどドメイン内の計算が少なくなり、

ブロック内の計算の比率が高まる (1 プロセッサで計算する場合には全体で 1 つのドメインとなる)。

消去木からドメインとブロックを作成する際に、それらの計算を行なうプロセッサを静的に割り当てる。ドメインの計算は独立に行なえることから、各プロセッサの演算回数の合計が均等になるように、ドメインをプロセッサに割り付ける (言い替えると、均等に割り付けられるようにドメイン分割を行なう)。一方ブロックは、ブロックに行番号と列番号をつけ、プロセッサが 2 次元格子に配置されているものとして、行方向と列方向にサイクリックにプロセッサを割り当てていく (Scatter Decomposition と呼んでいる)。これによってブロックの計算結果は、同じ行または列が割り当てられたプロセッサしか利用されないため、全プロセッサへのデータ転送は不要となる。

各ブロックに属する要素の計算には、消去木で子孫に当たるドメインやブロックの計算結果が必要である。そのため、各ドメインやブロックの計算が終ると、その結果を転送すると共に、それらの値が利用可能になったことを各プロセッサに伝える。すると、それらのドメインやブロックの値を参照するブロックの計算を、ブロックを所有するプロセッサが順次行なっていく。新たに得られたドメインやブロックの値を用いて行なう計算の集まりをタスクと呼び、各プロセッサ毎にタスクのキューを設ける。各プロセッサはドメインやブロックの計算が終了すると、その計算結果を利用するプロセッサのタスクキューに、新たなタスクを登録する。各ブロックの計算は、データ依存によって定まる半順序が守られていれば、任意の順序で処理することが出来る (実際には到着順に処理される)。全てのプロセッサで処理するタスクがなくなった時に、求める下三角行列が得られたことになる。

Cholesky の並列処理は、以下のような特徴を持つ [1]。

- スケーラビリティは SPLASH2 の中で最も悪く、32 プロセッサで 15 倍程度の速度向上しか得られない (データは tk15.O)。
- タスクキューの更新の際の同期のオーバーヘッドがプロセッサにより大きく異なる。
- プロセッサ数に比例してデータ転送量が増加する。
- ブロック内の計算では、密行列の計算となるため、局所性は高い。

4 SMP クラスタ向け並列化

4.1 共有メモリプログラムからの書き換え

前節で述べたことを踏まえて、SMP クラスタ向けに Cholesky のコードを書き換える。元のプログラムでの並列処理の記述には PARMACS と呼ばれるマクロが使用されている。このマクロは次の機能を持つ。

- 排他制御 (ロックとアンロック)
- バリア同期
- 並行プロセスの生成
- ホームを指定した共有変数の割り付け

ノード内でマルチスレッド並列化を行なう際には、これらのマクロを適切に定義することで、プログラムの書き換えなしに並列化が行なえる。

さらに、Cholesky では分散共有メモリシステムを意識して、同じプロセッサをホームとする記憶領域の割り付けの際には、まず大きなブロックを割り付けて、その中を分割した領域をデータに割り付けている。これにより、ページなどの単位での局所性の増加と、メモリ割り付けのオーバーヘッドの削減などが期待できる。

このように書かれた Cholesky の SMP クラスタ向けの並列化を行なう上での問題点とその対処法を以下に示す。以下では、元のプログラムでの実行のスレッドを、プロセスと呼ぶことにする。

- プロセス間の排他制御
  ノード内での排他制御は Solaris スレッドでサポートされているが、NICAM では排他制御をサポートしていないため、リモートメモリ参照を用いた排他制御を実装し、これらを組み合わせてプロセス間の排他制御を実現する。
- プロセス間のバリア同期
  NICAM ではノード間のバリアをサポートしているが、スレッド間のバリアはない。スレッド間のバリアを新たに実装し、ノード間のバリアと組み合わせて実現する。

- プロセスの生成
  ノード内ではスレッドを用い、ノード間ではSPMD方式で並列処理を実現する。

- 静的に割り付けられる共有データ
  ノード間の共有データを使用する際には、リモートメモリ参照を行なうために、そのデータの各ノードでの仮想アドレスを知る必要がある。SPMD方式では静的データは同一の仮想アドレスに割り付けられるため、アドレス変換のオーバーヘッドなしに共有データの参照が行なえる。共有データを更新した場合の処理については次節で述べる。

- 動的に割り付けられる共有データ
  動的に割り付けられる共有データでは、プロセス毎に個別に割り当てる必要があり、割り付けの時期や仮想アドレスがプロセスにより異なる場合がある。またホームでないプロセスで参照するときに、メモリをいつ割り当てるか、ホームの共有データのアドレスとの対応をどうとるか、コヒーレンシの管理をどう行なうかなどの問題がある。この解決法は次節で述べる。

4.2 共有データの管理方法

共有メモリプログラムでは、共有されるデータの実体はただ一つしかないため、排他制御の他は特に共有データの管理上の問題は生じない。それに対して分散メモリ上に配置された共有データの場合は、各ノードで個別にコピーを持つことが出来るため、キャッシュや分散共有メモリシステムと同様にコヒーレンシの管理が必要となる。さらに、動的に割り付けられる共有データの場合には、各ノードでのメモリの確保や、局所アドレスの管理などの問題が生じる。

ここでは分散共有メモリのような一般的な方法は取らずに、Choleskyで可能な方法として以下のような方法により共有データの管理を行なう。

まず、共有データの動的な割り付けは、各ノードの逐次実行部分でのみ行ない、全ノードが同じデータを同じ順序で割り付けるものとする。これにより各ノードでの仮想アドレスが一致し、仮想アドレスをグローバルアドレスとして用いることが出来る。また、プロセッサ間で同じ共有データのアドレスの対応を管理する必要もない。しかし、そのデータを参照しないノードもメモリを確保しなければならないことと、単一のノード内で確保できる大きさのデータしか扱えないという欠点がある。

元のプログラムでは、並列実行中にメモリを確保するのは、タスクキューのエントリと、プライベートなデータのみである。そのため、後者はそのまま各プロセスで独立にメモリを確保することにし、タスクキューは予め固定長のキューを作成するように変更した。

次に共有データのコヒーレンシ管理については、共有データの性質に応じて異なる方法を取った。そのために、先にCholeskyで使用される共有データについて説明する。共有データには、タスクキューと、ドメインおよびブロックの非零要素がある。

タスクキューはプロセス毎にあり、他のプロセスが書き換えようとする時にはロックによる排他制御を行なっている。ドメインやブロックの計算が終ると、その計算を行なったプロセスが、それらの結果を利用するプロセスのタスクキューに新たなタスクを追加する。同一ノード内のプロセスのタスクキューを書き換える際には、Solarisスレッドのロック機構が使える。他のノードのタスクキューを書き換える際には、リモートメモリ転送を用いて実装したノード間のロック機構を使用する。ノード間のロック時には、そのキューの所有者からロックしたプロセスに、最新のキューの内容がコピーされる。アンロックする際には、逆に所有者へ新しいキューの内容が書き戻される。キューはポインタを用いて実現されているが、同じ共有データは各ノードの仮想アドレスが一致するため、ポインタの値もそのまま転送することが出来る。

一方、ドメインとブロックの非零要素は、アルゴリズムからその所有者以外は書き換えないことが保証される。また、他のプロセスは所有者から、そのドメインやブロックの計算が終ったことを知らせるタスクが届くまで参照しない。そのため、タスクを登録する前に、計算結果を一度だけ転送すればよい。

以上をまとめると、タスクの処理を疑似コードで書くと、表1のようになる。

```
while (タスクがある間) {
  タスクを取り出す;
  for (依存関係のある各ブロックについて) {
    ブロックの非零要素の更新;
    if (ブロックの全ての更新が終った) {
      for (ブロックの値を参照する各プロセス) {
        ブロックの値の転送;
        タスクキューのロック;
        タスクキューの更新;
        更新したタスクキューの転送;
        タスクキューのアンロック;
      }
    }
  }
}
```

表 1: タスクの処理フロー

5 性能評価

この節では、前節に述べた方法で並列化した Cholesky の SMP クラスタでの実行性能を示す。

ここでは特に、同数のプロセッサを持つ分散メモリシステムと SMP クラスタを比較する。即ち、8 つのノードでそれぞれ一つのプロセッサを使用した場合と、4 つのノードで 2 つずつ、2 つのノードで 4 つずつのプロセッサを使用した場合の三通りについて測定結果を比較する。

コレスキー分解する疎行列には、SPLASH2 に附属するデータのうち、最も規模の大きな tk29 を用いた。このデータは 316740 個の非零要素を持つ 13992 次元の行列である。この分解には浮動小数点演算が約 405M 回必要で、COMPaS で 1 プロセッサのみ用いた逐次計算では実行時間が 11.9 秒かかる。ここでの実行時間は、数値分解の時間のみを表しており、消去木の作成や記号分解は含まない。これらは並列実行時にも逐次実行されるため、以後の並列実行時間にも含んでいない。

表 2 に三通りの方法の実行時間と、その内訳を示す。

PE 数: ノード数×ノード内のスレッド数を表す。

並列実行時間: 並列実行部の全体の実行時間。

ドメイン計算: プロセッサに割り当てられたドメイン内の計算時間。

ブロック計算: 消去木で下位にあるドメインとブロックの演算結果を使った計算の時間。

PE 数		8x1	4x2	2x4
並列実行時間		2.42	2.39	2.49
ドメイン計算	max	0.93	0.92	0.95
	avg	0.84	0.84	0.86
	min	0.72	0.71	0.70
ブロック計算	max	1.71	1.66	1.78
	avg	1.58	1.53	1.63
	min	1.49	1.45	1.54
タスクキュー (ノード内)	max	0.018	0.037	0.074
	avg	0.010	0.020	0.041
	min	0.007	0.011	0.012
タスクキュー (ノード間)	max	0.67	0.66	0.77
	avg	0.59	0.57	0.41
	min	0.47	0.41	0.25
タスク待ち	max	0.49	0.48	0.65
	avg	0.30	0.27	0.38
	min	0.07	0.10	0.09

表 2: 並列実行時間と内訳

タスクキュー更新: 他のプロセスのタスクキューの更新にかかる時間。同一ノード内のプロセスと他のノードのプロセスを区別する。これらはドメイン計算とブロック計算の中に含まれる。

タスク待ち: タスクキューが空の時に新たなタスクが登録されるのを待つ時間。これはブロック計算の中に含まれる。

並列実行時間以外の時間は、各プロセス毎に測定した値の最大値、平均値、最小値を上から順に示す。

並列実行時間はドメイン計算とブロック計算の和であるが、ドメイン計算は各プロセスで独立に行ない、最後に他のプロセスのタスクキューを更新する。そのため、三つの方式で性能の差はほとんどないはずであるが、2x4 のとき若干遅い。これは、ドメイン計算は局所性が少ないため、バスの競合によるものと思われる。

ブロック計算は、局所性の高い行列演算と、タスクキューの更新と非零要素の送信からなる。タスクキューの更新では、同じ行または列のブロックを持つプロセスだけが対象になるので、この場合各ブロック

の計算毎に4つのプロセスのタスクキューの更新が必要である。8x1では4つ全てが他のノードであるのに対し、4x2と2x4ではそれぞれ3つと1つがノード外のプロセスである。ノード外の更新時間を見ると、更新数の少ない2x4では平均値は低いものの、最大値が他より大きく、プロセッサ間で更新時間のばらつきが大きいことが分かる。一方、ノード内の更新時間を見ると、更新数に比例して増加しているが、ノード間の更新よりも高速に行なえている。タスクキューでの待ち時間は、タスクキューの更新時間の大きい2x4がもっとも悪い。

以上から、2x4の場合がもっともノード間通信が少ないにも関わらず、ノード間の同期の待ち時間が大きく実行時間を増加させていることが分かる。8x1と4x2を比べても、ノード間通信の減少ほどには性能に差が出ていない。CholeskyはSPLASH-2のプログラム中でもっともロードバランスが悪く特に同期にかかる時間はプロセスによって10%から70%と大変ばらつきが大きい[1]。このために、SMPクラスタにおいても、通信時間が減少するにも関わらず、それが実行時間に反映されないものと思われる。

6 まとめ

SMPクラスタCOMPaS上でSolarisスレッドとリモートメモリ転送を用いた並列処理について、SPLASH-2のCholeskyを例に、共有メモリ向けプログラムを元にした並列化と、性能評価を行なった。

共有データの参照のために不規則な通信が頻繁に発生するCholeskyでは、効率良く排他制御が出来、データの転送も不要なノード内の並列処理によって良い性能を得られるはずであった。しかし、負荷の不均衡のためにその効果を活かしきれていない。コレスキー分解のアルゴリズム自体の変更によらずに、SMPクラスタを効果的に利用できるか否かは今後の課題である。

参考文献

[1] S.C.Woo, et.al., "The SPLASH-2 Programs: Characterization and Methodological Cosiderations", 22nd ISCA, 1995.

[2] E.Rothberg, et.al., "An Efficient Block-Oriented Approach To Parallel Sparse Cholesky Factorization", Supercomputing '93.

[3] 田中他, "COMPaS: Pentium Proを用いたSMPクラスタとその評価", JSPP '98.

[4] 松田他, "SMPクラスタ向けネットワーク インターフェース上AM通信", SWoPP '97.

[5] 田中他, "SMPクラスタ上でのリモートメモリ転送を用いた通信と計算のオーバーラップによる性能改善", SWoPP '98.