バックエンドPCクラスタを用いた並列ソルバーに関する研究

武村興治，坂上仁志，新居学，高橋豊
姫路工業大学大学院　工学研究科　電気系工学専攻

我々は既に，汎用システムMATLABからバックエンドPCクラスタを呼び出し，大規模な疎行列の計算を並列実行することが可能なMATSOLというシステムを開発した．本稿では，MATSOLの並列性能向上のためにICCG法の並列実装をプロセス間通信の観点から改良し，MATSOLの計算性能が15%向上した．

Development of Parallel Solver using Backend PC Cluster

Koji TAKEMURA, Hitoshi SAKAGAMI, Manabu NII and Yutaka TAKAHASHI
Computer Engineering, Himeji Institute of Technology

A parallel solver MATSOL, which is executed from MATLAB and computes a linear algebra for a large sparse matrixes using a parallel backend PC cluster, has been developed in this paper. We aim at improving the interprocess communication in parallel implementation of ICCG method to improved parallel performance of MATSOL. We show that the parallel performance of MATSOL is improved by our suggested modification.

1．はじめに

MATLABはGUI機能を持つユーザインタフェースに優れた行列計算ソフトウェアであり，種々の分野で利用されている．しかし，MATLABでは計算を並列化して実行することができないので，大規模な計算を行う場合には計算時間がかかるという問題点がある．この問題点を解決するために，我々はMATSOLというシステムを開発した．MATSOLは，フロントエンドにMATLABをそのまま用いるためGUIの利便性を損なうことなくバックエンドPCクラスタで計算を並列実行することにより，計算処理にかかる時間の短縮が可能である[1].

本稿では，MATSOLの並列性能向上のためにICCG法の並列実装をプロセス間通信の観点から改良する．ICCG法に用いられるプロセス間通信は通常のsend／receive型と集計演算のためのreduction型の2種類がある．このため，send／receive型，reduction型についてそれぞれ通信の最適化を検討し，MATSOLの改良を行う．また，数値実験により改良したMATSOLの性能評価を行う．

2．MATSOL

MATSOLは各種プログラムやシェルスクリプトを経てバックエンドPCクラスタを呼び出し，大規模な行列計算を並列に行うシステムである．図1にMATSOLのシステム構成図を示す．MATSOLはMATSOLメインモジュールの他にms_begin, ms_end, ms_solverの4つのモジュールにより構成されている．ms_beginは計算の実行可否の判断を行い，ms_endは計算終了処理を行う．また，ms_solverは実際の行列計算を行うモジュールである．これら3つのモジュールはMATSOLメインモジュールから呼び出される．

ユーザはジョブとして実行する際に，実際の計算に使用するプロセッサ数を指定し，MATLABのコマンドウィンドウ上からMATSOLを呼び出す．

MATSOLは始めに，ユーザの指定したプロセッサ数でジョブが実行可能であるかを判断するために，ms_beginモジュールを用いてシステムファイルを調べ，クラスタ内のプロセッサ使用状況を確認する．その結果，ユーザの指定するプロセッサ数が利用可能であれば，プロセッサの使

用状況を保持したシステムファイルを更新し，処理をMATSOLメインモジュールへ戻す．逆に，ユーザの指定するプロセッサ数が利用不可能な場合には，未使用のプロセッサ数を表示してMATSOLシステムを終了する．

ユーザの指定するプロセッサ数を用いてジョブの実行が可能な場合，ms_solverモジュールを呼び出し，フロントエンドおよびバックエンド上で起動する．このとき，計算に用いる行列データはMATSOLメインモジュールからshmemを用いてフロントエンドで起動されたms_solverモジュールへ転送する．ms_solverモジュールは行列データを受信するとバックエンドPCクラスタを用いて行列計算を並列実行する．フロントエンドはMATLABの実行に使用しているので並列計算には用いない．PCクラスタ上の行列計算プログラムはMPIを用いて並列化されている．アルゴリズムの詳細は次節で述べる．

ms_solverモジュールは計算が終了すると，shmemを用いてその計算結果をMATSOLメインモジュールへ返す．その後，ms_endモジュールによりシステムファイルが更新されすべての処理が終了する．また，MATSOLはジョブが同時に実行される場合も考慮し，システムファイルの更新にはロックファイルを用いた排他制御を行っている．

3．ICCG法の並列化

ICCG法は，行列演算アルゴリズムの1つであり，元の行列に不完全コレスキー分解を行った後に，共役勾配法を用いる手法である[2]．

各プロセッサに行列データを分散させる方法を図2に示す．行列**A**，右辺ベクトル**b**はそれぞれ行を基準として各プロセッサへ均等に分散される．

図3に本システムで使用するICCG法のフローチャートを示す．このアルゴリズムの従来の並列化手法では，1ループ中にプロセッサ間で必要

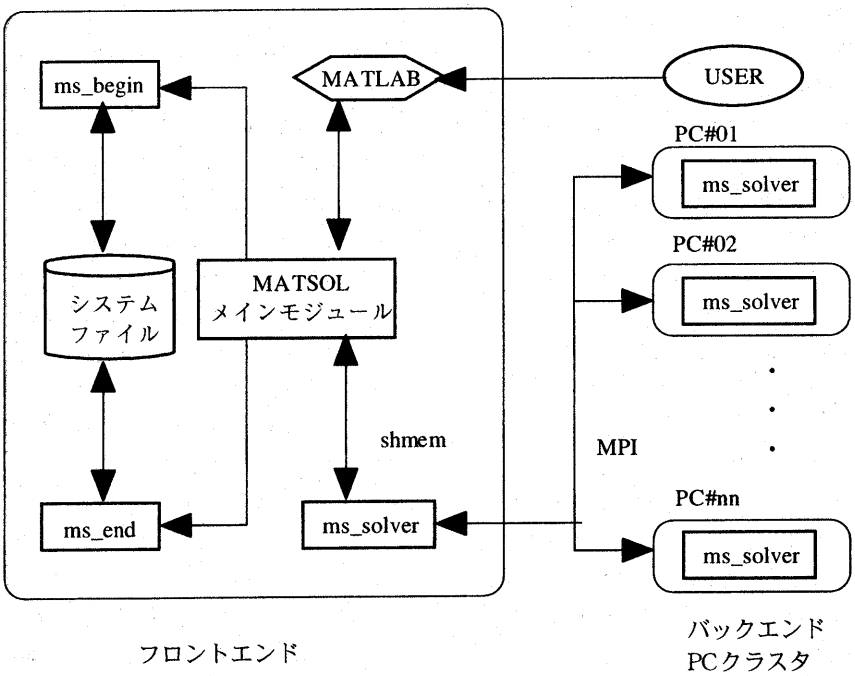

図1　システム構成図

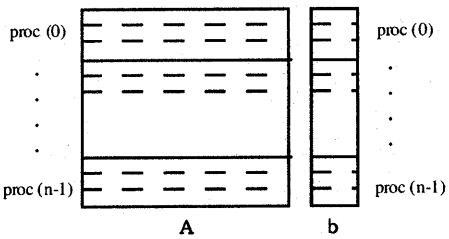

図2　行列データの分散

なデータを通信する send／receive 型通信が1回，reduction 型通信が3回含まれている．send／receive 型通信は行列とベクトルの積で行われ，reduction 型通信はベクトルの内積で行われる．

ICCG 法はループ中に不完全コレスキー分解を含む．不完全コレスキー分解は，計算ループの収束回数を減少させる効果がある．しかし，不完全コレスキー分解はループ中に回帰参照を含んでいるので，厳密な意味では並列化を行うことはできない．不完全コレスキー分解の回帰参照を便宜的に回避し強制的に並列化を行う場合は，並列化を行わない場合と比較しループ計算の収束回数が増加し，計算が収束しない場合もある．また，並列化行わない場合は少ない回数で収束するが，並列性能は悪くなる．MATSOL ではそのような状況も考慮し，不完全コレスキー分解の並列化を行わずに実行することも可能である．

4．通信の最適化

ICCG 法の並列化により多数のプロセス間通信が発生する．これらのプロセス間通信では，プロセッサ台数が増加するに従い，それらの通信にかかる時間も増加する．よって，通信時間を減少させる工夫をすることは，計算を高速化するのに重要である．そこで，前節で作成した ICCG 法の並列化について，プロセス間通信に重点を置き改良する．

ICCG 法の計算の reduction 型通信は従来法では1ループ中に3回呼び出されているが，それぞれの通信においてデータの転送量は非常に少なく，複数回に分けての reduction 型通信は通信のオーバーヘッドが増加するだけ，効率が悪くなる．このため，これらの reduction 型通信をまとめて実行することができれば，reduction 型通信の回数が減り，通信時間を短縮することができる．

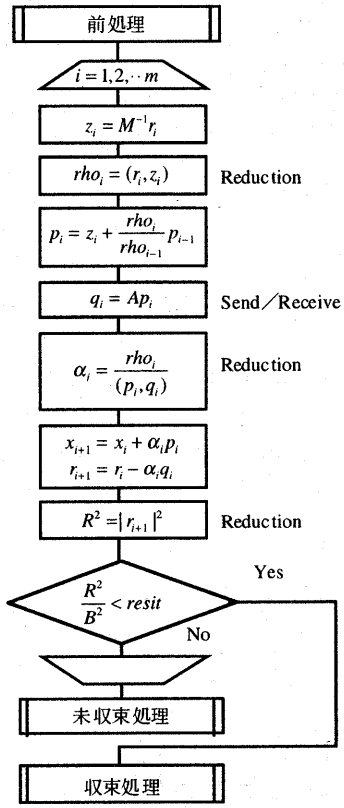

図3　ICCG 法のフローチャート

reduction型通信を必要とする収束判定は，ループ中の任意の場所で実行可能である．そこで，図4で示すように収束判定を rho を計算する場所に移動し，これら2つの処理で必要とする reduction 型通信を1度に行うこととする．これにより，reduction 型通信の1ループ中の呼び出し回数を3回から2回に減少することができる．

また，reduction 型通信を必要とする場所は，rho や収束判定の他に α を計算する場合にも使用される．そこで，図5で示すように α を図4の

reduction 型通信を行っている場所に移動させ，1 ループ中で必要とされる全ての reduction 型通信を 1 度に行うこととする．これにより，reduction 型通信の 1 ループ中の呼び出し回数を 3 回から 1 回に減少させることができる．ただし，この手法の欠点として，p_{i-1}，q_{i-1} は前回の値を使用しているので収束が遅くなる．

一方，send／receive 型通信はプロセッサ間で大量のデータ交換を行うため，通信時間がかかる．従来法では，図 6(a) にも示すようにプロセッサ番号の昇順に転送を行っていたため，特定のプロセッサでデータのコンフリクトが起こることが考えられる．そこで，データのコンフリクトを避けるため

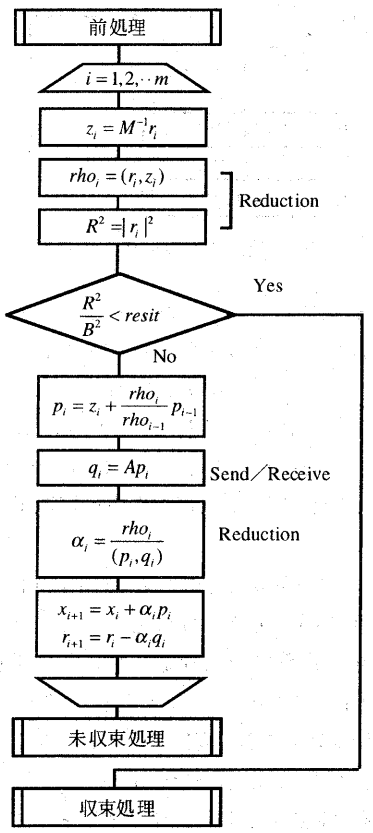

図 4　収束判定の修正

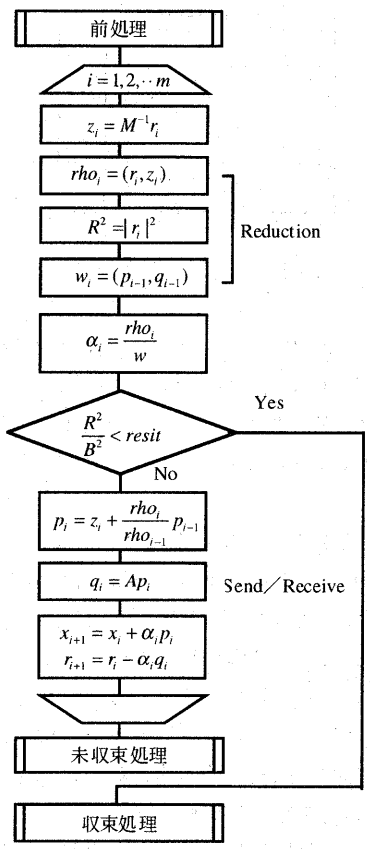

図 5　全 reduction 演算の修正

に，各プロセッサごとにデータ転送の順序を指定し，send と receive の処理手順を変更することにより，データのコンフリクトを可能な限り避けるようにする．改良手法として図 6(b),(c) に示す 2 種類の手法を検討する．ただし，myid を自プロセッサ番号，numprocs をプロセッサ数とする．ただし，プロセッサ番号 0 はフロントエンドを意味しここで計算は行われない．

図 6(b) に示す手法は，図 6(a) の従来法と比較しデータの転送順序のみを改良したものである．この手法では送信バッファに転送データが集中するという欠点がある．そこで，図 6(c) に示す手法は，同じくデータのコンフリクトの発生を避けるものであるが，1 ループ毎に receive を用いることによって，送信バッファに転送データが集中するのを防いでいる．

```
do i = 1, numprocs
  if(myid .ne. i) then
    send ( i )
  end if
end do
do i = 1, numprocs
  if(myid .ne. i) then
    receive ( i )
  end if
end  do
```

(a) 従来法

```
do i=1, numprocs-1
  id=mod(myid+i,numprocs)
  send ( id )
end do
do i=1, numprocs-1
  is=mod(myid-i,numprocs)
  recv ( is )
end do
```

(b) 改良後の転送手法1

```
do i=1, numprocs-1
  id=mod (myid+i,numprocs)
  send ( id )
  is=mod (myid-i,numprocs)
  recv ( is )
end do
```

(c) 改良後の転送手法2

図6　転送手法

5．性能評価

MATSOLを用いて大型疎行列を並列に解き，その性能評価を行った．実行時間の計測には10000×10000の5重優対角行列を用いた．計測の結果として得られたスピードアップと収束回数を表1に示す．また，1ループ当たりの平均実行時間の詳細を表2に示す．実行環境には350MHzのK6-2プロセッサ，メモリ64MB，OSとしてLinux2.2.17(Debian2.2)を用いたバックエンドPCクラスタを用いる．また，各ノードをつなぐネットワークは100Base-TXである．

表1より並列化による総処理時間が短縮できていることが確認できる．表2よりプロセッサ数が増加する毎に，計算時間は短縮されるが総処理時間に占める通信時間の割合が増加していることが分かる．

次に，計算手順を変更した場合の実行結果の性能について表3に示す．また，1ループあたりの平均実行時間の詳細を表4に示す．

計算手順を変更して計測した表3の実行結果は，もとの計測結果と比較して約15%性能が向上している．なお，表4のreductionの部分に着目し，

表1　スピードアップと収束回数

台数(台)	スピードアップ	収束回数(回)
1	1.00	8
2	1.26	12
4	2.31	11
8	2.64	12

表2　1ループ当たりの実行時間の詳細

台数(台)	通信時間(ms)		計算時間(ms)	total(ms)
	send/recv	reduction		
1	0.20	1.09	45.17	371.7
2	0.55	1.22	22.63	292.8
4	1.11	2.10	11.34	160.1
8	2.19	3.37	6.12	140.2

表3　スピードアップと収束回数

台数(台)	スピードアップ	収束回数(回)
1	1.00	9
2	1.28	13
4	2.37	12
8	3.04	13

表4　1ループ当たりの実行時間の詳細

台数(台)	通信時間(ms)		計算時間(ms)	total(ms)
	send/recv	reduction		
1	0.19	0.75	43.71	357.6
2	0.54	0.88	21.81	279.2
4	1.06	1.37	11.24	151.1
8	2.11	1.54	6.08	117.5

表5　実行結果の詳細

台数(台)	通信時間(ms)		計算時間(ms)	total(ms)
	send/recv	reduction		
1	0.18	0.75	43.59	356.5
2	0.48	0.89	21.97	280.5
4	0.96	1.16	11.43	149.6
8	1.87	1.56	6.12	115.4

表2のreductionの部分と比較すると，reduction型通信にかかる時間は2／3になっていることが分かる．

これは，収束判定とrhoの値を1つのreduction型通信で同時に転送を行うことによって通信の確立に費やされる時間が1回分削られるためである．

また，3つ全てのreduction型通信を1度に行う方法で計測を行った結果，1ループあたりのreduction型通信に費やされる時間は1／3となった．しかし，最終的に収束しなかったため，この手法は用いられないことがわかった．

次に，send／receive型通信に着目しデータの転送手法を改良した実行結果の詳細をそれぞれ表5に示す．8台で計測を行った結果，最大3.1倍の性能が出た．表4の実行結果と比較しsend／receiveの転送時間が約10%短縮できたことがわかる．これは，改良後の手法でデータ転送を行った結果，データのコンフリクトが避けられ，転送時間が短縮できたものと考えられる．send／receiveの転送は，行列サイズが拡大するあるいは非ゼロ要素数が増加するにつれて，データ転送量も増加する．その結果，データの転送手法を誤ると性能の低下を招く可能性がある．

また，転送を行う際に，送信バッファにデータが蓄積しないように，1ループ毎にreceiveを置いた場合の計算も行ったが，その性能は著しく低下した．本計測では，スイッチングハブを用いているため，あるプロセッサがデータ送信中に，他方のプロセッサはそのプロセッサにデータを送信するのを待たなければならないために，性能が低下したものと考えられる．

6．おわりに

以上で，プロセッサ数に応じた並列化の性能について示し，計算，通信に費やされる時間の詳細を示した．また，プロセス間通信に着目しreduction型通信の改良を行い，send／receive型通信の通信の最適化を行った後にその性能を評価した．

今後の課題は，ICCG法以外のアルゴリズムについて検討することが挙げられる．また，行列のコンディションを改良し，収束を改善するオーダリングについても検討し，収束しにくい行列についても計算できるように改良していく予定である．

参考文献

[1] 武村　興治，他：“バックエンドPCクラスタを用いた並列計算システムの開発”，並列計算シンポジウムJSPP2002(2002).

[2] 小国　力，“行列演算ソフトウェア”，丸善株式会社(1991).