ショートノート

並列型スーパコンピュータを用いたFDTD並列計算

打矢 匡[†]（学生員） 柏 達也[†a)]（正員）

Parallel FDTD Computation Using Parallel Supercomputer

Masashi UCHIYA[†], *Student Member* and
Tatsuya KASHIWA[†a)], *Regular Member*

† 北見工業大学，北見市
Kitami Institute of Technology, Kitami-shi, 090-8507 Japan
a) E-mail: KASHIWA-Tatsuya/elec@king.cc.kitami-it.ac.jp

あらまし スーパコンピュータは数T FLOPS/数百GByteの性能をもつに至っている．本研究では，汎用的電磁界解析法であるFDTD法についてMPIを用いて並列型スーパコンピュータに対応した並列化を行い，その実行性能を調べた．

キーワード FDTD法，並列計算，MPI，スーパコンピュータ，電磁界解析

1. まえがき

近年の計算機の大幅な進歩は電磁界の大規模解析を可能としている．中でもスーパコンピュータは数T FLOPS/数百GByteの性能をもつに至っている．しかしながら，シングルプロセッサの性能向上の限界や開発コストの上昇から，これらの性能は並列計算によって達成されている．FDTD法は領域分割型解法であるため，データ転送は境界面でしか必要ではなく，極めて分散メモリアーキテクチャに適合したアルゴリズムとなっている．従来，Transputerやi860を用いた並列計算機，またはベクトル型スーパコンピュータを用いた高速大容量FDTD計算が行われている[1]．しかしながら，近年性能向上の著しい並列型スーパコンピュータを用いたFDTD並列計算の研究に関する報告は筆者らの知る限りなされていない．

本研究では，最新の並列化ライブラリであるMPIを用いて汎用的電磁界解析法であるFDTD法の並列計算を行い，並列型スーパコンピュータを用いたFDTD並列計算の特性を調べた．なお，並列型スーパコンピュータとしてはSR8000を用いた．

2. 並列計算機[2]

並列計算機はそのメモリモデルより，二つに分けることが可能である．複数のプロセッサがバスやクロスバスイッチを介して単一のメモリを共有する共有メモリ型と，各プロセッサごとに個別にメモリをもつ分散メモリ型である．共有メモリ型はメモリの帯域幅などの制約によりある程度以上プロセッサを増やしても速度上昇が飽和してしまい一般には数十プロセッサが限界である．一方，分散メモリ型では各プロセッサごとに個別にメモリをもつため何百何千とプロセッサを追加させることが可能である．そのため，並列型スーパコンピュータでは共有メモリ型と分散メモリ型を組み合わせたアーキテクチャとなっているのが一般的である．しかしながら，共有メモリ型のシステムにおいては，自動的に並列化コードを出力する自動並列化コンパイラがほぼ完全な域に達しているものの，分散メモリ型のシステムにおいてはいまだ実用の域には達していない．したがって，並列型スーパコンピュータの性能を最大限に発揮させるためには分散メモリアーキテクチャを意識した並列プログラムを自作する必要がある．

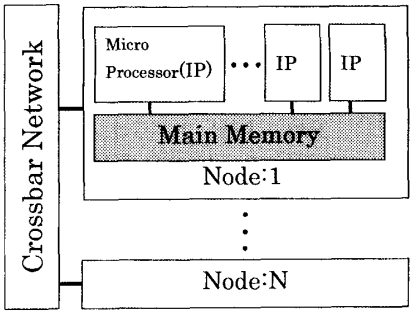

図1 SR8000構成図
Fig. 1 Architecture of SR8000.

3. SR8000の構造と特徴[3]

並列型スーパコンピュータSR8000は図1に示すようにNodeと呼ばれる専用のRISCプロセッサ(IP)8個からなる共有メモリ型マルチプロセッサシステムを基本ブロックとしている．なお，1Node当り6.5GByteのメモリが割り当てられている．このNodeを次々と追加することにより性能を向上させることが可能である．一方，SR8000システムはNodeを1プロセッサとした分散メモリアーキテクチャとなっている．そのため，複数Nodeを使用して計算を行うためにはプログラムを分散メモリアーキテクチャに対応させる必要がある．また，SR8000は協調型マイクロプロセッサ機構，擬似ベクトル化機能によるNode内での並列処理能力の向上，リモートDMA転送機能，高性能なノード間通信機能による高い並列化効率の実現を果たしており，通常のPCクラスタや並列型計算機以上に高いパフォーマンスを得ることが可能となっている．

4. 並列プログラミング[2]

並列プログラミングを行うためには並列プログラミング言語を用いてプログラミングを行うか，メッセー

ジパッシングライブラリを使いデータ転送命令をプログラム中で明示的に指示するという二つの方法が存在する．しかしながら，二つの方法はそれぞれ一長一短があるため使用に際してはその点を留意する必要がある．

並列プログラミング言語としてはHPF(High Performance Fortran)などがあり，今までのプログラムに指示文を付加してコンパイルするとコンパイラが自動的に並列化を行う．プログラムの変更は不要なため使用は容易ではあるが，自動的に並列化を行うため期待通りの性能を発揮させることは難しい．

一方，メッセージパッシングライブラリを使用する方法はプログラムの変更が必要だが手動での最適化が可能であり，並列プログラミング言語を利用するよりは手間がかかるものの期待どおりの性能を実現することができる．メッセージパッシングライブラリとしてはMPI(Message Passing Interface)，PVM(Parallel Virtual Machine)などが広く用いられている．

本研究では最新のメッセージパッシングライブラリであるMPI [4] を使用している．

5. FDTDアルゴリズムの並列化

FDTD法は領域分割型解法であるため，データ転送は境界面での値のみが必要であり，極めて分散メモリアーキテクチャに適合したアルゴリズムとなっている．しかしながら，高い転送効率を得るためにはFDTDアルゴリズムの特性を踏まえた上で必要な成分のみを転送する必要がある．FDTD法はリープフロッグアルゴリズムであるため，電界節点と磁界節点が交互に配置されている．そのため異なる二つの面，E面とH面が定義される．また，一般的にFDTD法はE面を境界に配置する場合がほとんどである．したがって，転送パターンは境界のE面とH面をそれぞれ転送するか，あるいは境界のE面と次のH面を飛ばして境界から2番目のE面を転送するかの2通りを考えることができる．一般的にFDTD法はCell単位で考えられるので，境界のE面と次のH面を飛ばして境界から2番目のE面を転送する方法がプログラミングが簡便である．ただし，境界のE面とH面をそれぞれ転送する方法よりも1節点余分に配列を確保しなくてはならず注意が必要である．具体的にZ方向で分割した場合，境界面の節点配置は図2のようになり，E面はE_x，E_y，H_z成分，H面はH_x，H_y，E_z成分で構成される．H_z成分はE_x，E_y成分より求めることができるので，転送が必要な成分はE_x，E_y成分となる．なお，この部分のプログラムを付録に示した．

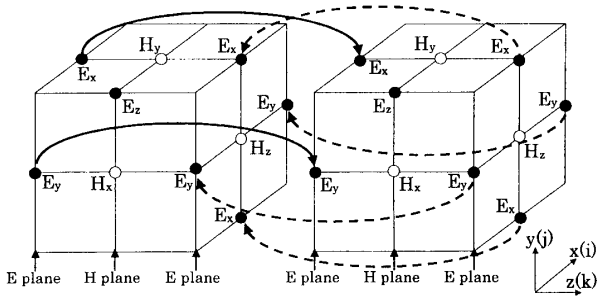

図2 FDTD共通境界面におけるデータ転送
Fig. 2 Data transfer in the FDTD shared boundary.

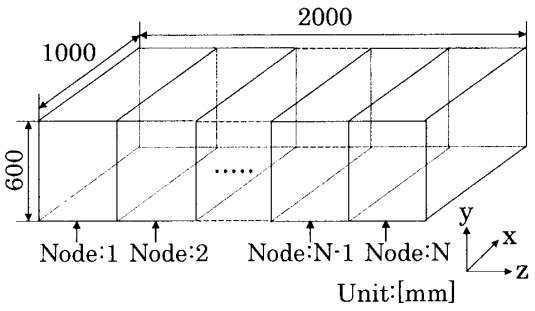

図3 大規模解析モデル
Fig. 3 Analyzed model of large structure.

6. 大規模FDTD並列計算

SR8000を使用した大規模FDTD並列計算の性能を調べるために図3に示される大規模系の解析を行った．解析領域を1000×600×2000[mm]と設定し，周波数は1.7GHzとした．また，両端にMurの1次境界条件を適用した．空間離散間隔は5×5×5[mm]，2×2×2.5[mm]とした．並列化はZ方向N(1,2,4,8,10,16)分割することによって行い，各Nodeに割り付けた．

実行結果は空間離散間隔を5 × 5 × 5[mm]としたときを表1，2 × 2 × 2.5[mm]としたときを表2に示す．実行時の使用メモリを表3に示した．また，Node数の増加による性能向上比を図4に示した．空間離散間隔2 × 2 × 2.5[mm]の場合は1Node当りのメモリ限界に達したため2Nodeの実行結果から示している．図4より，空間離散間隔を5 × 5 × 5[mm]時はNode数を増加させていくと次第に性能が劣化していく．これは，後述のように，1Node当りに割り付けられる配列数が減りすぎてしまったため性能が発揮できなかったものと考えられる．それに対して空間離散間隔2 × 2 × 2.5[mm]時は，ほぼ直線の関係を維持しており16Node使用時においても速度上昇比は14.10倍とかなり良好な結果となった．更に，転送による影響を

表1 実行時間 (5 × 5 × 5[mm])
Table 1 Elapsed time(5 × 5 × 5[mm]).

Node N	Elapsed time[S]			Speedup recorded
	Transfer T_N	No Transfer T_1	Ratio	
1	-	3034	-	-
2	1470	1484	0.99	2.06
4	853	863	0.99	3.56
8	471	471	1.00	6.44
10	406	405	1.00	7.47
16	307	307	1.00	9.88

表2 実行時間 (2 × 2 × 2.5[mm])
Table 2 Elapsed time(2 × 2 × 2.5[mm]).

Node N	Elapsed time[S]			Speedup recorded
	Transfer T_N	No Transfer T_1	Ratio	
1	-	-	-	-
2	52857	53117	1.00	2.00
4	25590	25735	0.99	4.13
8	13468	13442	1.00	7.85
10	11115	11088	1.00	9.51
16	7499	7572	0.99	14.10

調べるため，N Nodeを使った場合の計算時間をT_N，$1/N$のメモリを1Nodeで計算した場合の計算時間をT_1として，$(T_N - T_1)/T_N$をデータ転送比として定義し，使用Node数による変化を図5に示した．図5より，本解析の条件においては転送オーバヘッドが無視できることわかる．以上の結果より1)データ転送オーバヘッドは無視できるほど小さいことがわかる．2)並列計算を行うことにより，Node数の増加に応じた線型的な速度上昇を見込むことができる．3)しかしながら，1Nodeに割り付ける配列数が減少してくると本来の性能が発揮できなくなり，速度上昇が飽和してしまうことがわかった．従来のPCクラスタやスーパコンピュータと比べて並列型スーパコンピュータの転送速度が高速であり，FDTD並列計算が工学的に実用的な段階に達してきていることが示された．

7. むすび

MPIを用いて並列型スーパコンピュータに対応したFDTDアルゴリズムの並列化を行った．なお，並列型スーパコンピュータとしてSR8000を用いた．また，大規模FDTD並列計算を行い，本並列アルゴリズムの有効性について検証した．従来の並列計算機においてボトルネックであったプロセッサ間のデータ転送時間がほとんど無視できることがわかった．並列計算を行うことにより，Node数の増加に応じた線型的な速度上昇を見込むことができる．本技法は高速大容量の計算を行うためには有効な方法であり将来的にも重要な技術である．本技法は汎用的な並列化手法を用いており，他の並列型計算機にも適用可能である．今後は，他の並列型計算機での実行結果についても調べていきたい．また，本技法を実際的な問題に適用していく予定である．

謝辞 本研究において北海道大学大型計算機センターのSR8000を使用した．ここに謝意を表する．

表3 1Node当りの使用メモリ
Table 3 Memory used per node.

Node N	Memory used per node[MB]	
	5 × 5 × 5[mm]	2 × 2 × 2.5[mm]
1	801	-
2	428	4877
4	237	2477
8	146	1262
10	111	1007
16	80	656

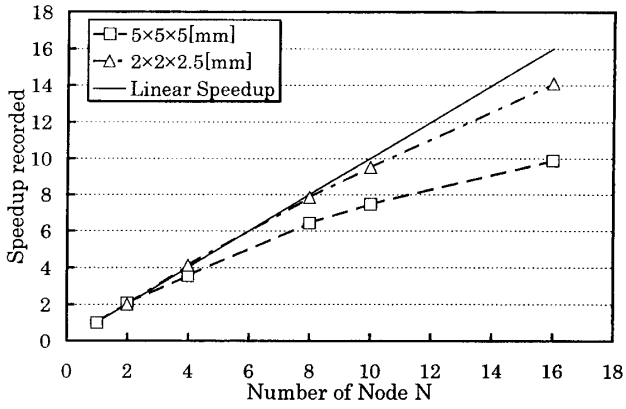

図4 速度向上比
Fig. 4 Speed-up recorded.

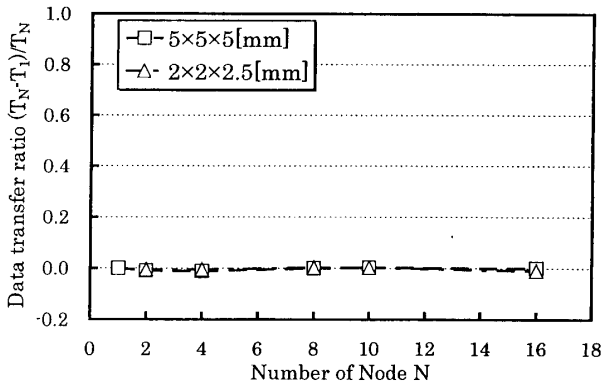

図5 データ転送比
Fig. 5 Data transfer ratio.

文　献

[1] A. Taflove, Computational Electrodynamics: The Finite-Difference Time-Domain Method, Chap.16, Artech House, 1995.
[2] 湯淺太一, 安村通晃, 中田登志之, 初めての並列プログラミング, 共立出版, July 1999.
[3] 日立製作所, スーパーテクニカルサーバ HITACHI SR8000 シリーズ, http://www.hitachi.co.jp/Prod/comp/hpc/jpn/sr81.html.
[4] MPI forum, MPI : A Message-Passing Interface Standard, 1995.

付　録

本解析に使用したMPI転送部分についてソースコードを示す.

```
C**********************************
C    ノードの右側転送部分 -->
C**********************************
      IF (MYRANK.NE.(NSIZE-1)) THEN
       S_TAG=100+RIGHT
       R_TAG=200+MYRANK
C------送るデータのパック------
       DO 8100 I=1,IEXMAX
        DO 8100 J=1,JEXMAX
         S_BUFF(I,J,1)=EX1(I,J,DKEXMAX)
 8100  CONTINUE
       DO 8300 I=1,IEYMAX
        DO 8300 J=1,JEYMAX
         S_BUFF(I,J,2)=EY1(I,J,DKEYMAX)
 8300  CONTINUE
C------送信と受信を同時に実行------
       CALL MPI_SENDRECV(S_BUFF,
     & N_BUFF,MPI_REAL,RIGHT,S_TAG,
     & R_BUFF,N_BUFF,MPI_REAL,RIGHT,
     & R_TAG,MPI_COMM_WORLD,STAT,IERR)
C------受け取ったデータのアンパック------
       DO 8600 I=1,IEXMAX
        DO 8600 J=1,JEXMAX
         EX1(I,J,(DKEXMAX+1))=R_BUFF(I,J,1)
 8600  CONTINUE
       DO 8800 I=1,IEYMAX
        DO 8800 J=1,JEYMAX
         EY1(I,J,(DKEYMAX+1))=R_BUFF(I,J,2)
 8800  CONTINUE
      END IF
C**********************************
C    ノードの左側転送部分 <--
C**********************************
      IF (MYRANK.NE.0) THEN
       S_TAG=200+LEFT
       R_TAG=100+MYRANK
C------送るデータのパック------
       DO 8500 I=1,IEXMAX
        DO 8500 J=1,JEXMAX
         S_BUFF(I,J,1)=EX1(I,J,2)
 8500  CONTINUE
       DO 8700 I=1,IEYMAX
        DO 8700 J=1,JEYMAX
         S_BUFF(I,J,2)=EY1(I,J,2)
 8700  CONTINUE
C------送信と受信を同時に実行------
       CALL MPI_SENDRECV(S_BUFF,
     & N_BUFF,MPI_REAL,RIGHT,S_TAG,
     & R_BUFF,N_BUFF,MPI_REAL,RIGHT,
     & R_TAG,MPI_COMM_WORLD,STAT,IERR)
C------受け取ったデータのアンパック------
       DO 8200 I=1,IEXMAX
        DO 8200 J=1,JEXMAX
         EX1(I,J,1)=R_BUFF(I,J,1)
 8200  CONTINUE
       DO 8400 I=1,IEYMAX
        DO 8400 J=1,JEYMAX
         EY1(I,J,1)=R_BUFF(I,J,2)
 8400  CONTINUE
      END IF
```

（平成13年3月21日受付，6月7日再受付）