[全国共同利用情報基盤センター研究開発論文集 No. 34 より]

東北大学サイバーサイエンスセンターにおける ユーザコードの高速化支援活動

佐々木大輔†，山下毅†，小野敏†，大泉健治†
江川隆輔‡，小林広明‡

†東北大学　情報部　情報基盤課
‡東北大学サイバーサイエンスセンター　スーパーコンピューティング研究部

1　はじめに

東北大学サイバーサイエンスセンター(以下，本センター)は，その前身の一つである大型計算機センターの設立(1969 年)以来，研究室のレベルを遙かに超える最高性能の計算機システムを設置し，最先端の学術研究を強力に支援してきた．さらに，利用者にとって使い勝手の良いシステムの構築，他では実行できない大規模プログラムの実行環境の整備を行うと共に，スーパーコンピュータの性能を最大限に発揮することを目的に，利用者，本センター教員・技術職員，ベンダーが一体となってプログラムの高速化技術および新しいシミュレーション技術の研究・開発に取り組み，計算科学・計算機科学の発展に貢献してきた．本稿では，はじめに本センターのシステム概要および自動バッファリング機能を持つコンパイラへのアップデート，次に共同研究を通した高速化支援活動の取り組みについて述べる．

2　大規模科学計算システム

2.1 システム概要

本センターでは，全国のベクトル型計算機ユーザの高いニーズに応えられるシステム構成を目的に，ベクトル型スーパーコンピュータSX-9とスカラ型並列コンピュータExpress5800を運用している．

ベクトル型スーパーコンピュータSX-9は1ノードあたり16個のCPUで構成され，シングルコアのCPUは1CPUあたり102.4Gflop/sの演算性能を有する．また，1ノードあたり1Tバイトの共有メモリと1CPUあたり256Gバイト/秒と高いデータ転送能力を有し，大規模かつ高性能な処理を可能にしている．ベクトル型のスーパーコンピュータは，高いメモリバンド幅とループ中で繰り返し処理されるような大規模な配列データを一括して演算実行することで，流体解析や気象解析，電磁界解析をはじめとするメモリバンド幅に対する要求の高い大規模科学シミュレーションを高効率で実行可能である．

Express5800はSX-9システムのフロントエンドサーバとして機能するばかりでなく，汎用のアプリケーションや，ベクトルスーパーコンピュータに適さないアプリケーションの実行に用いられている．このように，本センターではベクトル・スカラと2種類のスーパーコンピュータを配備・運用することで多様なユーザの計算要求に応えられる計算サービスを提供している．

2.2　コンパイラによるADB向け最適化の評価

本センターは，FortranコンパイラおよびC/C++コンパイラに対し年に数回のアップデートを実施し，最新の環境を利用者に提供している．本年は既に2回のアップデートを行い，機能強化,機能追加およびバグ修正に対応している．ベクトル化促進等の最適化強化や性能解析ツールの強化が主な内容だが，本年はADBを用いた自動ベクトルデータ・バッファリング機能を新たに追加している．

2.2.1　ADB(Assignable Data Buffer)

SX-9ではCPUと主記憶装置間にADBと呼ばれるベクトルデータをバッファする機構が新たに追加されている．図１にADBの概念図を示す．ADBはCPU―主記憶装置間よりメモリバンド幅が広く，レイテンシ（遅延）が短いという特徴を持っている．また，通常のキャッシュメモリとは異なり利用者が選択的にデータをバッファリングすることが可能である．ベクトル計算機は，大きいメモリバンド幅を持ち，高いデータ供給能力を有する．データ供給能力は，ベクトル演算性能あたりのメモリバンド幅であるB/F(Bytes/FLOP)で表され，これまでのベクトル計算機は4B/Fと高いデータ供給能力を有していた．ところが，近年マイクロプロセッサの様々な設計制約により，SX-9のデータ供給能力は2.5B/Fに低下している．そこで，ADBを用いる事で，SX-9のデータ供給能力を2.5B/Fから4.0B/Fに向上させることが可能になる．

2.2.2 自動ベクトルデータ・バッファリング機能

ベクトルデータ・バッファリングとは，再利用性のあるベクトルデータをADBに保持することである．プログラム中で頻繁にアクセスされる配列をADBに保持することでデータ供給能力が向上し，アプリケーションの高速化が期待できる．これまで，バッファリングをするには指示行を用いて，保存する配列を指定する必要があり，ADBを効果的に利用するには利用者の判断によるところが大きかった．しかし，本年3月のコンパイラアップデートにより自動ベクトルデータ・バッファリング機能が追加され，FortranおよびC/C++ともにバッファリングの自動化が可能になった．センターで利用されている各種プログラムを，自動バッファリング機能の有無で比較した結果を図２に示す．前コンパイラは自動バッファリング”なし”，現コンパイラは自動バッファリング”あり”の場合の性能を示している．各プログラムとも実行時間にして概ね数パーセントの性能向上が得られていることがわかる．性能が低下しているプログラムも一部見られるがこれはスカラデータ・命令がベクトルデータによりADBから追い出されたためと考えられる．この結果から，本センターで実行されている多くのプログラムが自動化の効果を得られると判断し，規定値で自動化を有効とし利用者への提供を開始している．

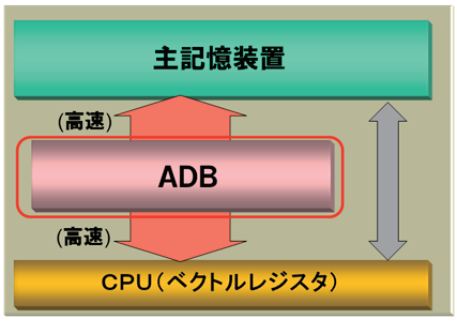

図１　ADBの概念図

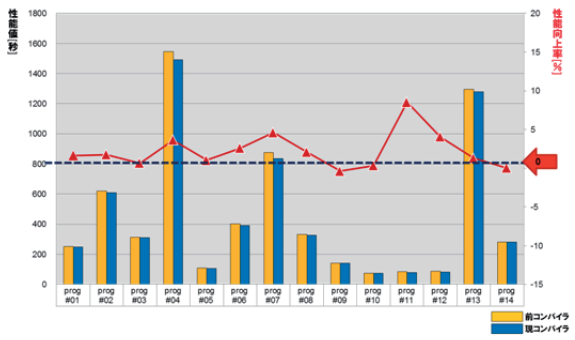

図２　自動バッファリング機能の有無による性能比較

3　高速化支援活動

スーパコンピュータシステムの潜在能力を最大限に享受し，大規模かつ高精度な計算機シミュレーションを実現するためには，コンピュータシステム，およびプログラミング技術に関する高度な知識が求められるのが実情である．このような状況下で，本センターでは「最先端，かつ世界最高性能クラスのコンピュータシステムの導入，利用環境の整備・運用・研究・開発」という使命を果たすために，1997年

より利用者・センター運用スタッフ（技術系職員・スーパーコンピューティング研究部教員），ベンダーの技術者から構成される高速化支援体制を整備している．

本センターでは，この支援体制のもと，自動ベクトル化，自動並列化機能のみに頼ることなく，実シミュレーションコード解析に基づく臨床学的研究開発を行い，ユーザプログラムの高速化支援活動に取り組んできた．表１に1999年から2012年にかけて本活動を通して，取り組んできた高速化支援の結果を示す．この間，合計163件のプログラムの高速化に取り組み，平均で単体性能向上比は39.3倍，並列性能向上比12.2倍を実現しており，高速化支援が高い成果を上げていることがわかる．

さらに，1997年から全国に広がる本センターのユーザ(計算科学者)とセンター(計算機科学者)が連携し，サイバーサイエンスセンター共同研究として131件の共同研究を推進している．また，2008年からは文部科学省先端研究施設共用促進事業，2010年からは大規模情報基盤共同利用・共同研究拠点の一拠点として，全国の計算科学者らとの共同研究も推進している．次に，本年度の代表的な高速化支援活動例をとりあげ説明する．

表１高速化実績

年	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
件数	8	9	10	7	18	20	8	29	10	15	8	8	13
単体性能向上比	4.5	2.5	1.6	2.2	6.7	2.9	1.5	3.1	33.0	9,3	381.0	47.0	16.2
並列性能向上比	31.7	8.6	4.9	2.8	18.6	4.5	4.1	8.0	1.9	5.1	3.6	48.0	17.2

3.1. ベクトル化率促進による高速化

ベクトル処理はDOループによる配列データの繰り返し演算を，ベクトルユニットを用いて一括に実行するものである．ベクトル型スーパーコンピュータの高い実効性能を得るためには，プログラム全体に対するベクトル処理可能な割合であるベクトル化率を，出来る限り 100%に近づける必要がある．また，ベクトル処理を高速に行うためには，主記憶装置からベクトルユニットに対して，データを滞りなく供給する必要がある．

DOループのベクトル化を阻害する要因としては，1) 配列要素の定義・引用関係に依存関係がある　2) 配列要素が確定されない IF・CASE 文を含む　3) 配列データのアクセス時に複雑な間接参照を伴う　4) DOループ中の関数・サブルーチン呼び出しおよび入出力文がある，などが挙げられる．

データ供給の遅延の要因としては，1) CPU 内における同一 CPU ポートへのロード･ストアの集中　2) 同一のメモリバンクへのアクセス競合および CPU と主記憶装置間の経路上で発生するアクセス競合　3) 主記憶装置への不連続なメモリアクセス，などが挙げられる．メモリアクセスの効率化では，前述の ADB を利用することで， メモリバンド幅を補うことが可能となり，性能向上を図れる場合がある．

最適化例として以下で取り上げるユーザコードは，Red Black 法を用いて SOR 法の反復計算を行うものである[1]．オリジナルコードはユーザにより OpenMP による並列化が行われていた．SX-9 の簡易性能解析ツール（Ftrace）によりプログラムの実行プロファイルを取得したところ，プラズマ流の各支配方程式を解くためのサブルーチン A でベクトル化率が低いため，実効性能が低いことが分かった．このサブルーチンのベクトル化率の促進と，OpenMP による並列化の促進について検討を行った．

3.1.1 IF 文の除去によるベクトル化と作業配列の導入による並列化の促進

図３に示すサブルーチン A のオリジナルコードでは 5625 行目の if 文が真の場合 5626-5633 行が実行され，さらにこの中の if 文(5628 行目)が真になると RSMAX や IRMU,JRMU,KRMU が更新される構造である．

この構造のままではコンパイラは 5628 行目の if 文に依存関係が存在すると判断し，ベクトル化が行われない．5625 行目の if 文の結果はループ中で不変なので，ループの外側に移動することが可能である．また最外側ループ K を並列化するために作業配列 WKRSMAX を導入した．この作業配列 WKRSMAX により，最外側ループ K における各最大値を保存することができ，OpenMP 指示行による並列化が可能になった．

図４に SX-9 の 16CPU でサブルーチン A のオリジナルコードと修正コードを実行した際の Ftrace 結果を示す．オリジナルコードは並列化されていないため 1CPU の値，修正コードは 16CPU（スレッド）の合計値である．修正コードはベクトル化率 98.95%となり，16CPU 合計の FLOPS 値が約 6.5 倍向上した．その結果サブルーチン A の実行時間は約 1/24 に短縮された．

(オリジナルコード)

```
  5620:                  RSMAX=ZERO
  5621: +------>         DO K=1,KEND
  5622: |+----->          DO J=2,JEND
  5623: ||+---->           DO I=2,IEND
  5625: |||                 IF (IPHI.EQ.IU) THEN
  5626: |||                   RSMU=RESP/RSU0
  5627: |||                   RSU(I,J,K)=RESI(I,J,K)/RSU0
  5628: |||                   IF (DABS(RSU(I,J,K)).GT.RSMAX) THEN
  5629: |||                     RSMAX=DABS(RSU(I,J,K))
  5630: |||                     IRMU=I
  5631: |||                     JRMU=J
  5632: |||                     KRMU=K
  5633: |||                   END IF
   (略)
```

(修正コード)

```
  6179:                   IF (IPHI.EQ.IU) THEN
  6180:                   RSMAX=ZERO
  6181:                   RSMU=RESP/RSU0
  6196:             !$OMP PARALLEL DO
  6197: P------>           DO K=1,KEND                                  !  P は並列化された DO ループ,
  6198: |                   WKRSMAX(K)=ZERO                            !  V はベクトル化された DO ループを示す.
  6199: |+----->            DO J=2,JEND
  6200: ||V---->             DO I=2,IEND
  6201: |||                   RSU(I,J,K)=RESI(I,J,K)/RSU0     ! Non-dimensionalization
  6202: |||                   IF (DABS(RSU(I,J,K)).GT.WKRSMAX(K)) THEN
  6203: |||                     WKRSMAX(K)=DABS(RSU(I,J,K))
  6204: |||                     IWIR(K)=I
  6205: |||                     IWJR(K)=J
  6206: |||                     IWKR(K)=K
  6207: |||                   END IF
  6208: ||V----               END DO
  6209: |+-----              END DO
  6210: P------             END DO
  6211:             !$OMP END PARALLEL DO
```

図３　サブルーチン A のオリジナルコードと修正コード

PROC.NAME	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK
original	5100	432.549(7.9)	84.813	3057.3	1472.4	88.65	250.0	79.495	0.252	56.389	0.561	47.425
modified	5100	17.984(0.7)	7.340	35438.2	9558.2	98.95	235.1	12.754	0.326	1.019	0.499	5.434

図４　サブルーチン A のオリジナルコードと修正コードの Ftrace 結果比較

3.2　MPI 並列による高速化

近年の大規模コンピューティングでは分散メモリ並列による実行が不可欠であり，各種の並列化ライ

ブラリを用いて大規模並列計算が行われている．SX-9 では並列化ライブラリとして MPI-2（Message Passing Interface）をサポートしている．MPI は標準化された並列化ライブラリであるため可搬性が高く，システムのリプレイスの際も従来の MPI 化されたプログラムを大規模な修正なしに実行可能であるなど，そのメリットは大きい．しかしながら，逐次プログラムあるいは OpenMP による並列化プログラムを MPI 化する際には，プログラムを分析し，データ処理の分割方法，通信の内容と方法をユーザが選択し記述する必要がある．これらは実行時の通信性能に大きく影響を与えるため，MPI 化の前には適切な設計が必要である．

MPI 化例として以下で取り上げるユーザコードは，乱流モデルには k-ω Shear Stress Transport (SST) モデルを用い，3 次元差分法を用いて Navier Stokes 方程式を解くものである[2]．現状の計算規模（グリッド点）は i,j,k =141×39×45 である．MPI 化に必要なデータ分割は，データ配列の第 3 次元目の k で行った．本コードはオリジナルコードの自動並列化版，MPI によるフラット MPI 並列化版および，MPI 化に加えてコンパイラによる自動並列化機能を併用したハイブリッド並列版の作成を行った．

3.2.1 フラット MPI 化コード

ここでは４つのプロセスに分割した場合を例に説明する．MPI による分散並列化は k の上端と下端を除いた，k=2,..,44 の範囲を４等分して k=2,..,12，k=13,..,23，k=24,..,34，k=35,..,44 をそれぞれのプロセスに割り当てる．上端と下端を除くのは，多くの配列の更新は k=2,..,nz-1 の範囲で行われているためである．ただし，インデックス k での配列の更新において，k-1 または k+1 のインデックスを参照する場合があるため，各プロセスにさらに k=1,..,13，k=12,..,24，k=23,..,35，k=34,..,45 を割り当てる必要がある．

k のループをプロセスごとに分割するとともに，各配列は分割され，プロセスごとに各領域の大きさが小さくなる．この領域分割した部分は並列に実行することができ，並列化前と同じ結果が得られることが確認されている．図５に，MPI 化に必要なデータ分割を行ったソースコードの一部を示す．

```
!宣言 (lz はプロセス 0,1,2 で 13, プロセス 3 で 12)
111: allocate( mxf(nx,ny,lz,nsp))
117: allocate(amuis(nx,ny,lz,nsp))
134: allocate( xfai(nx,ny,lz,nsp))
184: allocate( amu(nx,ny,lz) )
!実行コード (local_kend はプロセス 0,1,2 で 12, プロセス 3 で 11)
326: +------> do k=2,local_kend
327: |+-----> do j=2,ny-1
328: ||V----> do i=2,nx-1                                   !  V はベクトル化された DO ループ,
329: |||                                                    !  *は展開された DO ループを示す.
330: ||| amumix=0.d0
331: ||| akpmix=0.d0
332: ||| !cdir expand=nsp
333: |||*---> do l=1,nsp
334: |||| amumix=amumix+amuis(i,j,k,l)/(1.d0+(1.d0/mxf(i,j,k,l))*xfai(i,j,k,l))
335: |||| akpmix=akpmix+akpis(i,j,k,l)/(1.d0+(1.065d0/mxf(i,j,k,l))*xfai(i,j,k,l))
336: |||*--- end do
337: |||
338: ||| amu(i,j,k)=amumix
339: ||| anu(i,j,k)=amu(i,j,k)/rhos(i,j,k)
340: ||| akpl(i,j,k)=akpmix
341: |||
342: ||V---- end do
343: |+----- end do
344: +------ end do
```

図５　各プロセスにデータを分割した MPI 化コード

3.2.2　ハイブリッド MPI 化コード

ハイブリッド MPI 化コードは MPI による並列化に加えて，コンパイラによる自動並列化機能を併用したコードである． MPI 並列で 16 プロセス生成された場合ループ長は，以下の式より 2 または 3 である．

(オリジナルの k のループ長 43)÷(プロセス数 16) = 2 または 3

MPI プロセスで分割されたこのループを自動並列化の対象とすると，2 または 3 スレッドしか並列実行できないため，高い並列度を実現するには MPI とスレッド並列のループを別のものにする必要がある．自動並列化では一般的に最外側ループを並列化の対象とするため，MPI による並列化が行われていないインデックス j のループを最外側に来るようにソースを修正し，最外側の j のループで自動並列化が行われるようにした．j のループ長は 37 なので，37 スレッドまで並列実行可能である．

3.2.3　性能測定結果

表２にオリジナルコードの自動並列化による SMP 並列化実行，フラット MPI 並列化によるマルチプロセス実行の結果を，表３にハイブリッド MPI 並列化コードによるマルチノード実行の結果を示す．

表２　性能測定結果（シングルノード）

プログラム		ノード数	ノードあたりプロセス数	ノードあたりスレッド数	実行時間/sec
オリジナル（自動並列化のみ）	(a)	1	1	16	42.80
フラットMPI版（MPI化のみ）	(b)	1	16	1	26.72

実行時間の測定結果より，1 ノード内実行におけるフラット MPI 版(b)はコンパイラの自動並列化による(a)よりも 1.6 倍の性能が得られた．このことから本プログラムでは MPI による並列化が高速化に有用であることが示された．

表３　性能測定結果（マルチノード）

プログラム		ノード数	ノードあたりプロセス数	ノードあたりスレッド数	実行時間/sec
ハイブリッドMPI版（MPI化＋自動並列化）	(c)	2	1	16	32.40
	(d)	2	2	8	30.46
	(e)	2	4	4	29.44

次に，ハイブリッド MPI 版での(c) ～(e)では，2 ノードを用い，ノード内のプロセス数とスレッド数を変化させて実行したものである．プロセス数を増加させると実行時間が短くなっている．これは，本プログラムは格子点数が並列数で割り切れず，ロードインバランスが発生するが，プロセス数を増加させることで影響を小さくできるためと考えられる．プロセス数およびスレッド数は計算規模（グリッド点）の拡張や，利用可能なノード数（CPU 数）をパラメータとして，計算を実行するシステムに最適な設定値を決定する必要がある．今回のユーザコードの並列化には，汎用性のある MPI ライブラリによる並列化とコンパイラの自動並列化を用いているので，将来的にシステムの大規模化やアーキテクチャの変更があったとしても，コードの長期間の利用が可能である．

4　まとめ

本稿では，本センターの有する大規模科学計算システムと 1999 年から取り組んでいる高速化支援活動の取り組みと成果について述べた．この活動により利用者のプログラムは大幅に性能改善しているが，大規模化・長時間化する様々なプログラムに対応すべく，引き続き高速化支援活動を推進していきたい．

本稿に記載されている成果は，本センターを利用してくださる利用者の方々の多大なるご協力により得られたものである．高速化・MPI 化の事例には東北大学工学研究科機械システムデザイン工学専攻茂田先生，東北大学工学研究科航空宇宙工学専攻渡辺先生のご協力をいただいた．ここにあらためて感謝の意を表する．

参考文献

[1]日本電気株式会社　プログラム最適化作業報告
[2]日本電気株式会社　渡辺先生コード MPI 化作業報告書