サイエンティフィック・システム研究会　　2015 年 8 月 28 日
SS 研 HPC フォーラム 2015 資料

ステンシル系プログラムによる FX100 の性能評価と高速化チューニング

高木　亮治
宇宙航空研究開発機構 宇宙科学研究所

宇宙航空研究開発機構（JAXA）は航空宇宙分野における基礎研究から応用・利用研究までを一貫して行っているが，高性能計算機を用いた数値シミュレーション技術の重要性を認識し，高性能・高機能な大規模計算機システムの整備・運用を積極的に推進してきた．2014 年 10 月には JAXA の二代目のスーパーコンピュータシステム JSS2 が稼働を開始し，2015 年 4 月からは JSS2 の中核システムである Supercomputer for earth Observation, Rockets and Aeronautics - MAin (SORA-MA)と呼ばれる新システムが稼動を開始した．SORA-MA は富士通社製 FX100 で，マルチコアスカラーCPU を用いた大規模超並列計算機であり 1.31Pflops の理論演算性能と 40TByte の主記憶容量を持っている．本講演では JSS2 の概要を紹介すると同時に，航空宇宙分野における CFD プログラムを用いた SORA-MA の性能評価と高速化チューニングについて報告する．

高速流解析，数値流体力学，高速化チューニング，FX100，SIMD

１．はじめに

平成 27 年度から JAXA の新スーパーコンピュータシステム（JAXA Supercomputer System 2:JSS2）[1]の中核となる SORA-MA（Supercomputer for earth Observation, Rockets and Aeronautics - MAin）が稼働を開始した．SORA-MA は FUJITSU Supercomputer PRIMEHPC FX100[2]（以後 FX100 と呼称）で構成されており，これまでの主計算機システム（富士通製ハイエンドテクニカルコンピューティングサーバ FX1）との比較で考えると SIMD（Single Instruction Multiple Data）演算器の採用，メニーコアといった特徴を有している．ここでは，JSS2 のシステム概要を紹介すると同時に，基礎的なベンチマークプログラムである STREAM[3]および汎用的な圧縮性流体解析プログラム UPACS[4]のベンチマーク版プログラム UPACS-Lite を用いた FX100 の性能評価，高速化チューニングについて紹介する．

２．JSS2

JSS2 は計算リソースを提供する SORA とデータアーカイバである J-SPACE から構成されている．SORA は計算システム（SORA-MA），プレポストシステム（SORA-PP (PrePost)），大規模メモリ計算システム（SORA-LM Large Memory)），ログインシステム（SORA-LI (LogIn)），ファイルシステム部（SORA-FS）などから構成された複合システムである．SORA-MA を構成する FX100 は，富士通製ハイエンドテクニカルコンピューティングサーバ FX1，スーパーコンピュータ「京」および PRIMEHPC FX10 のアーキテクチャーを継承しつつ 100PFLOPS 以上のシステム構成が可能なスーパーコンピュータである．FX100 では各ノードに 20nm プロセスで製造される SPARC64™XIfx プロセッサを 1 つ搭載し，それらのノードを Tofu（Torus Fusion）インターコネクト 2 で接続した超並列計算機であ

る．メモリには 3 次元積層メモリである HMC（Hybrid Memory Cube）を採用し，メモリバンド幅は 480GB/s となっている．Tofu2 は 12.5GB/s でノード間を 6 次元メッシュ/トーラスで接続している．SPARC64™XIfx プロセッサは 32 個の演算コアの他に 2 つのアシスタントコアを有し 1TFLOPS 以上の理論性能を有している．アシスタントコアはシステムデーモン，ファイル I/O 処理，通信処理（特に非同期通信）を担当し大規模システムの性能向上に寄与している．

SPARC64™XIfx プロセッサは 2 つの CMG（Core Memory Group），Tofu2 コントローラ，PCI Express コントローラなどで構成されている．1 つの CMG は 16 個の演算コア，1 個のアシスタントコア，17 コアで共有される 12MiB の L2 キャッシュ，メモリコントローラで構成される．各コアは 8 つの FMA（Floating-point Multiply and Add）を有し 4-wide SIMD により 1 サイクルあたり 16 個（=2 倍精度浮動小数点演算/FMA×4FMA/SIMD×2SIMD/サイクル），ノード（32 演算コア）あたり 512 個の倍精度浮動小数点演算が実行可能となっている．ちなみに，単精度浮動小数点演算であれば，1 サイクルあたり 2 倍の演算が可能である．　SPARC64™XIfx プロセッサで性能を出すためには 32 個の演算コアおよび SIMD を如何にうまく活用するかが重要となる．

３．性能評価

SORA-MA（FX100）の性能評価を，基礎的なベンチマークプログラムである STREAM[3] および汎用的な圧縮性流体解析プログラム UPACS[4]のベンチマーク版プログラム UPACS-Lite を用いて実施した．ここでは主にノード性能に着目した．なお，ここでの性能値は測定時点での値・特性であり，今後変更・改善されることがある．

測定に用いた SORA-MA，比較のために用いたインテル CPU のマシン（SORA-PP）の諸条件を表 1 に示す．

表1 測定環境（一部に推定値あり）

	SORA-MA（FX100）	SORA-PP
マシン	PRIMEHPC FX100	PRIMERGY RX350S8
CPU	Fujitsu SPARC64™XIfx	Intel Xeon E5-2643V2
周波数	1.975GHz	3.5GHz
CPU/ノード	1	2
コア/CPU	32	6
コア/ノード	32	12
理論性能	1.011TFLOPS	0.336TFLOPS
メモリ性能	431GB/s	119.4GB/s

３．１　STREAM

STREAM は主にメモリ性能を測定するベンチマークプログラムである. STREAM では 1 次元配列に対して COPY（配列コピー），SCALE（スカラー値の掛け算），ADD（2 つの配列の足し算），TRIAD の性能を測定できるが，ここでは TRIAD を用いた．TRIAD は

```
do i=1,N
  a(i) = b(i) + S * c(i)
enddo
```

となる．a,b,c は 1 次元配列，S はスカラーである．図 1 に TRIAD の測定結果を示す．

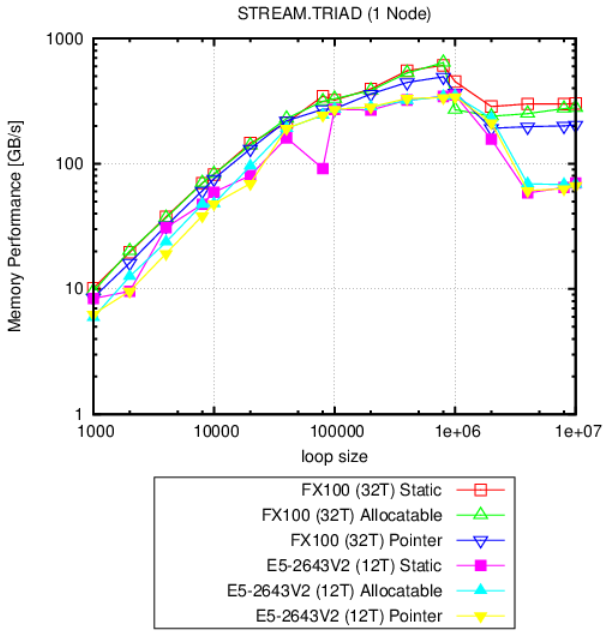


図 1 TRIADによるメモリ性能（SORA-MA, PP）

SORA-MA（FX100），SORA-PP（図中では E5-2643V2 と表記）ともにノード内の全コアを使用している．SORA-MA，SORA-PP ともにループ長が 10^6 付近（メモリ容量としては 24Mbyte 程度で L2 キャッシュの容量程度）まではキャッシュによるアクセスであるが，それを超えるとキャッシュが溢れ，10^7 では完全なメモリアクセスになっていると考えられる．SORA-MA の場合，利用する配列（静的配列，ポインター，アロケータブル）によって性能が異なることがわかる．今後改善されることが期待されるが，現状では静的配列が 302GB/s で一番性能が高く，次にアロケータブル配列が 278GB/s，ポインターがかなり遅く 206GB/s となった．なお，この結果を得るためにはラージページオプション（具体的には lpgparm –l demand）を指定する必要がある．指定しないと動的配列（アロケータブル，ポインターともに）は 100GB/s 程度の性能となる．

図 2 に SORA-MA のスレッドスケーラビリティを示す．静的配列のケースで，ループ長は 10^7 で固定（キャッシュは溢れてメモリ性能を計測）してスレッドを 1 から 32 まで増やしたときのメモリ性能の変化を示している．図よりスレッドの増加に伴って特異な傾向を示していることがわかる．ここで，32 スレッド時のメモリ性能をコア数 32 で割った値が 9.375 GB/s である．1 スレッドから 8 スレッドまではスレッドの増加にともない線形に性能が向上．8 スレッドから 16 スレッドまでは性能が頭打ちになっていることから，8 スレッド以下では 1 コア当たり最大で 2 コア分のメモリ性能を使っていると推測できる．JSS の主計算機システムであった FX1 で同種の計測を行った場合，ノード内で 1 スレッド実行を行うと，そのスレッドはノードの全メモリ性能を占有できることがわかっており，それ

と似た状況である．16 スレッドを超えると再び線形（増加率は 1 コア当たりの性能である 9.375GB/s）に増加している．16 コアが 1CMG に所属しているため，16 スレッドを超えると別の CMG へのアクセスが発生する（いわゆる NUMA 構成）ためと考えられる．実際，numactl -interleave=all を使うことでほぼ線形な挙動を得ることができる．

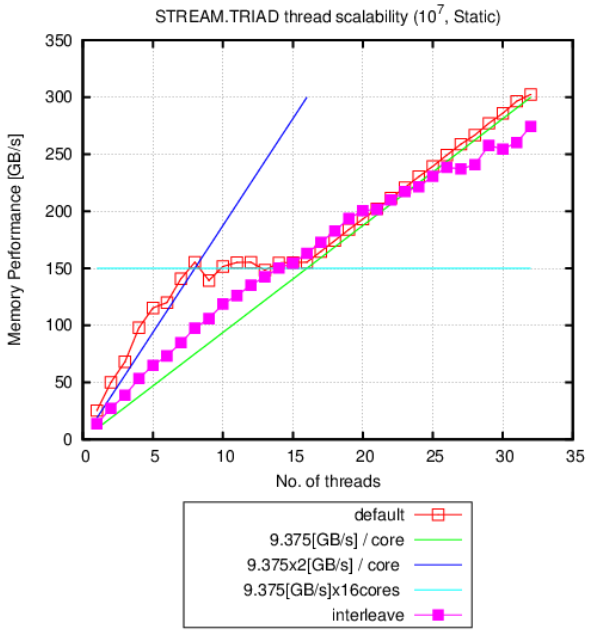
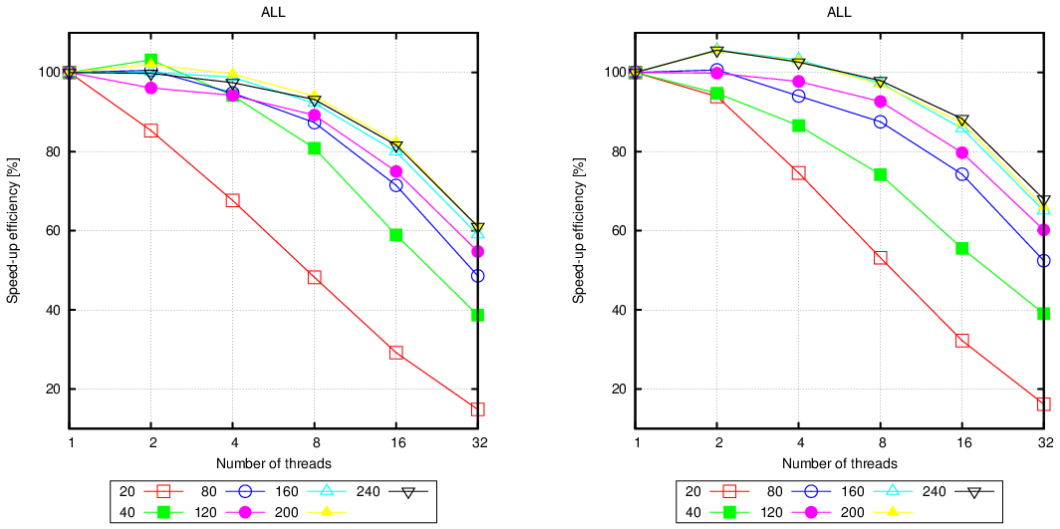


図 2　TRIAD（ループ長は10^7）のスレッドスケーラビリティ（SORA-MA）

３．２　UPACS-Lite

汎用的圧縮性流体解析プログラム UPACS をベースに，基本的な機能だけに絞って性能評価や高速化チューニングのためのプラットフォームとして整備したものが UPACS-Lite である．UPACS-Lite では対流項は MUSCL＋SHUS，時間積分は 2 時精度 Euler 陰解法とし，MFGS で内部反復を 2 回，MFGS のスレッド並列化には Block Red-Black 法を利用，乱流モデルはなしの粘性計算，MPI と OpenMP を用いたハイブリッド並列としている．UPACS-Lite では、支配方程式を有限体積法で離散化した離散方程式を解くことになるが，その際の主要な構成要素としては，時間積分（左辺），対流項，粘性項が存在する．それぞれの計算は全てステンシル計算となるが，ステンシルの形（メモリアクセスパターン，量）や演算パターン，量が異なる．また，UPACS では複数ブロックの構造格子を用いているため，通常の計算ではブロックの大きさ，形が異なり，その結果プログラム中では様々なループ長，特にインデックス毎にループ長が異なることになり，高速化の際に注意する必要がある．

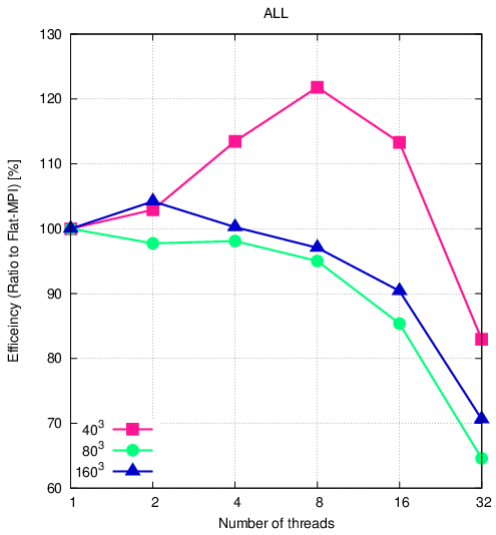
まず初めに，図 3 に SORA-MA のスレッドスケーラビリティの結果を示す．



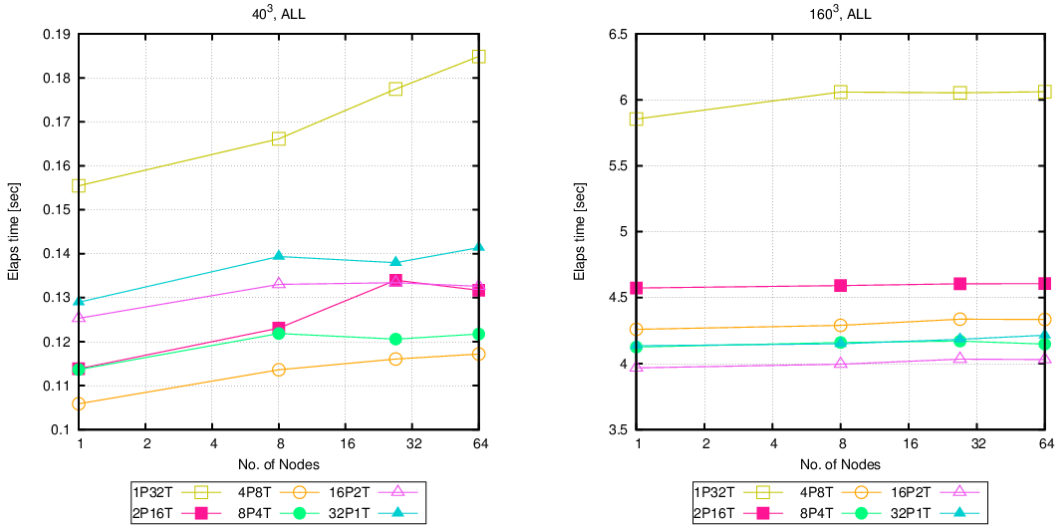
a) default　　b) interleave

図 3　UPACS-Liteのスレッドスケーラビリティ（SORA-MA）

ループ長が短い場合はスレッドスケーラビリティが悪いが、ループ長の増加にともない改善されている．interleave を使うことでスケーラビリティは若干向上しているように見えるが，これは 1 スレッド時の性能が低下したためである．図 4 にハイブリッド並列とフラット MPI 並列との比較を示す．ノードに対して同一規模の計算をプロセス数とスレッド数の組み合わせを変えて測定を行った．ノードに対して 32 プロセス×1 スレッドがフラット MPI 実行となる．図 4 a)はノード内、図 4 b), c)は 64 ノードまでの多ノードの結果である．1 ノードの結果ではブロックサイズが小さい（40^3），もしくはある組み合わせの時にハイブリッド並列が高速な場合があった．80^3 のケースではフラット MPI が常に高速という結果になった．1 ノードでの傾向は 64 ノードまであまり変化はなかった．もっと大規模ノードでの評価が必要と考えている．



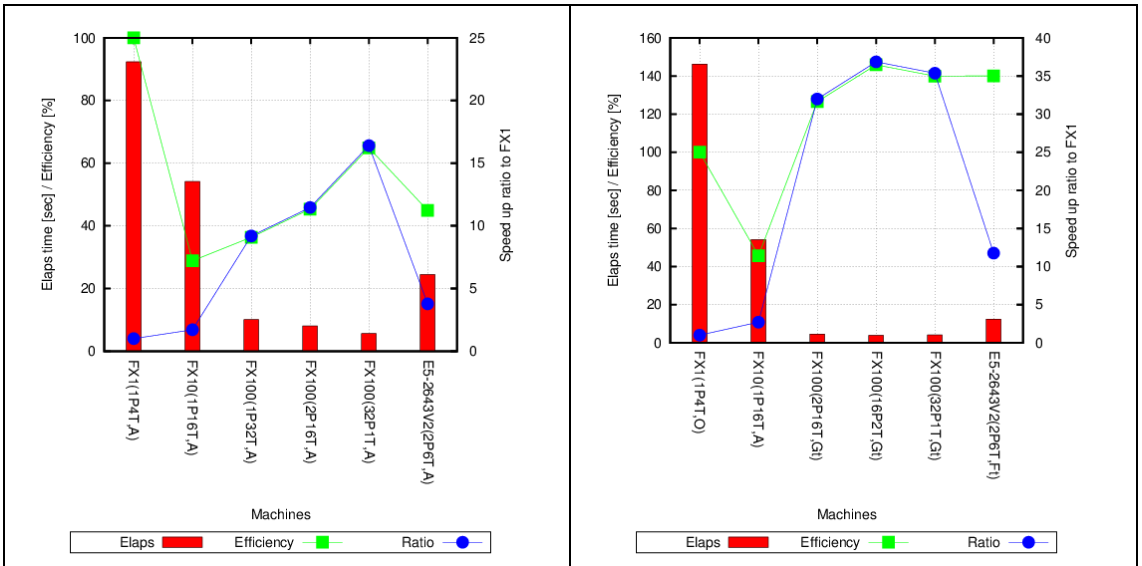
a) ノード内



b) 多ノード（40^3）　　c) 多ノード（160^3）

図 4　ハイブリッド並列とフラットMPIとの比較

JSS（FX1），FX10，JSS2 SORA-MA（FX100）間での比較を図 5 に示す．図 5 a)では基準となるプログラム A を使ってシステムの差だけを比較した場合，図 5 b)ではシステムの変更にともない，それに対応した高速化チューニングの効果も込みで比較している．二つの図において，赤い棒グラフは計算時間を，●シンボル付折れ線グラフは，JSS（FX1）を 1 とした時の計算速度向上比を，■シンブル付折れ線グラフは理論性能の向上に対する実際の計算速度向上の比率（理論性能が 10 倍になれば，本来は 10 倍速くなるはずだが，5 倍にしか早くならなかった場合は 50%）を示す．



a) 同一プログラム A での比較　　b) 最速版での比較

図 5　JSS（FX1），FX10，　JSS2 SORA-MA（FX100）の比較

何れのケースもハイブリッド並列を使っており JSS（FX1）では 4 コア/Node なので 1 プロセス 4 スレッド(1P4T), FX10 は 16 コア/Node なので 1 プロセス 16 スレッド(1P16T), SORA-MA（FX100）は 16 コア/CMG×2CMG/Node なので 1 プロセス 32 スレッド（1P32T），SORA-MA の標準モデルである 2 プロセス 16 スレッド（2P16T），フラット MPI モデルの 32 プロセス 1 スレッド(32P1T), SORA-PP では 6 コア/CPU×2CPU/Node なので 2 プロセス 6 スレッド（2P6T）とした．SORA-MA 向けの現時点での最速版（Gt）は FX1(O, 1P4T)の約 32 倍高速となっている．ちなみにフラット MPI（32P1T）で比較すると約 37 倍となった. 同じプログラム(基準となる A)を用いた場合 SORA-MA(JSS2)は FX1 に対して約 11 倍，フラット MPI では約 16 倍高速となった．

４．UPACS-Lite の SORA-MA 向け高速化チューニング

UPACS-Lite を用いて SORA-MA 向けに高速化チューニングを実施した．表 2　高速化チューニングの実施項目概要に高速化チューニングの実施内容を示す.

表 2　高速化チューニングの実施項目概要

Version	Tuning
O	オリジナル（FX1 で利用）
A	flux 配列の構造体の変更＋SIMD 化促進
B	jk ループの融合（スレッド並列数の確保）
C	MFGS の書き下し＋OCL の挿入
D	flux 配列のインデックス変更 （i,j,k,:）→（:,i,j,k）
E	flux 配列のインデックスの変更（C と D の比較で高速版を選択） 対流項：(i,j,k,:) 粘性項：(:,i,j,k)
F	手動アンローリング 初期化のベクトル記述：dq=0 → OpenMP で並列化 保存量ループの展開（do n=1,nPhys を削除）
G	2 重ループの一重化をやめて OpenMP の collapse(2)を利用 データ通信の前後処理部の OpenMP 化 nPhys ループの位置の変更 MFGS（時間積分）で使われている Block Red-Black のブロック分割の最適化
?t	?の cell 配列インデックスの変更 （i,j,k,:）→（:,i,j,k）

これらの高速化チューニングは大きく分けると

a) 配列のインデックスの変更

b) データ構造（構造体）の変更

c) SIMD 化の促進
に分けられる。

a）の配列に関しては、プログラムの主要な配列としては有限体積法のセルで定義される cell 配列と、セル面の流束で定義される flux 配列がある．cell 配列はプログラムの中で大域的に使われる配列であるのに対し，flux 配列は対流項，粘性項の flux を計算する時だけに使われる配列である，これらの配列のインデックスが所謂ベクトル型（i,j,k,n）なのかスカラー型（n,i,j,k）が最適なのかを比較した．ここで、i,j,k が空間のインデックスを，n が物理量（乱流モデルなしの完全気体では 5 となる）を示す.

b）の構造体に関しては Structure of Array:SOA（配列の構造体）か Array of Structure:AOS（構造体の配列）のどちらが適切かを確認した．図 6 に具体例を示す.

O（AOS）

type cellFaceType
  real(8) :: area
  real(8), dimension(5) :: flux
end type
type(cellFaceType), dimension(:,:,:), pointer :: cface

do n; do k; do j; do i   cface(i,j,k)%flux = … enddo
A（SOA）

type cellFaceType
  real(8) :: area
  real(8), dimension(:,:,:,:), pointer :: flux
end type
type(cellFaceType) :: cface

do n; do k; do j; do i   cface%flux(i,j,k,n) = … enddo

図 6　データ構造（AOS, SOA）の例

c) に関しては現状のコンパイラの最適化性能に依存する部分が大きいが，ソースを書き換えることで SIMD 化を促進する．この部分に関しては，今後コンパイラの成熟とともに書き換えが不要になることが期待される.

1) O から A
流束（対流項、粘性項）の計算部分に関して

- flux配列をAOSからSOAに変更
- SIMD化を促進するためにループ内の一時配列をスカラー化：a(3,3)をa_11, a_12,

a_13, a_21, a_22, a_23, a_31, a_32, a_33に修正

- 組み込み関数の手動展開

を実施した．特に SIMD 化に関しては現状では最内ループ SIMD 化が適用されるため，ループボディに配列のベクトル表記があるとその部分が SIMD 化されてしまい，他のループボディが SIMD 化されない．例えば

```
allocate(a(imax,5),b(imax,5),c(imax,5))
do i=1,imax
  u = a(i,2)/a(i,1)
  v= a(i,3)/a(i,1)
  a(i,:) = b(i,:) + c(i,:)     ←ここだけ SIMD 化
enddo
```

の場合、a(i,:) = ….の部分だけが SIMD 化されてしまい、残りの u = …, v = …, などが SIMD 化されない．この場合

```
allocate(a(imax,5),b(imax,5),c(imax,5))
do i=1,imax
  u = a(i,2)/a(i,1)
  v= a(i,3)/a(i,1)
  a(i,1) = b(i,1) + c(i,1)
  a(i,2) = b(i,2) + c(i,2)
  a(i,3) = b(i,3) + c(i,3)
  a(i,4) = b(i,4) + c(i,4)
  a(i,5) = b(i,5) + c(i,5)
enddo
```

のように書き下してベクトル表記をなくすと全てが SIMD 化される．

2) A から B

OpenMP によるスレッド並列は最外の k ループで実施するが、ループ長がスレッド数より短い場合はスレッド数が確保できないので j,k ループを融合してループ長を稼いだ．

3) B から C

MFGS に関して

- 一時配列のスカラー化
- OCLの挿入により依存関係を無視（計算は早くなるが収束性が悪化するため、トータルでみた場合にどちらが良いかは未確認）

4) C から D

flux 配列のインデックスを（i,j,k,:）から（:,i,j,k）へ変更した．

5) D から E

C と D を比較したところ対流項、粘性項のそれぞれで適したインデックスが異なること

がわかった．そのため、それぞれに最適なインデックスとして対流項は（i,j,k,:）、粘性項は（:,i,j,k）を採用した．この理由であるが，粘性項は速度の差分を多く計算するため，物理量（N=1～5、特に 2,3,4）の再利用性が高く，そのためスカラー型のインデックスが適している．一方対流項は物理量の再利用性が多くはないのでベクトル型のインデックスが適していると推測している．

6) E から F

flux 配列はローカルな配列で毎回初期化を行っており，その部分は自動並列に任せていたが，コンパイラが十分な最適化を実施できないことが判明したので OpenMP 化を実施した．更に，3 次元空間の 3 重ループに物理量のループ（n=1,nPhys）の 4 重ループがある部分に関して，物理量ループをアンロールした．変更前は

```
do n=1,nPhys
!$omp parallel do private(jk,i,j,k)
  do jk=1,jkmax
    k = (jk-1)/jmax+1
    j = jk-jmax*(k-1)
    do i=1,imax
      q(i,j,k,n) = …
      …
    enddo
  enddo
!$omp end parallel do
enddo
```

となっており, この場合 OpenMP のオーバーヘッドが nPhys 分発生するなどのデメリットが考えられる．そのため

```
!$omp parallel do private(jk,i,j,k)
  do jk=1,jkmax
    k = (jk-1)/jmax+1
    j = jk-jmax*(k-1)
    do i=1,imax
      q(i,j,k,1)  = …
      q(i,j,k,2) = …
      …
    enddo
  enddo
!$omp end parallel do
```

とすることで n のループに対しても最適化が適用され性能の向上が期待できる．

7) F から G

B で行ったスレッド数確保のために j,k ループの手動融合を OpenMP の collapse を使う

方式に変更した．ちなみに両方式で性能面での変化は見られなかった．MPI を使ったデータ通信部の前後処理部に関して OpenMP を用いたスレッド並列化を行った．F で実施した nPhys ループの書き下しは，汎用的なプログラムでは適用が難しいので nPhys ループの位置を変更することでベストではないがベターな性能を得ることができた．時間積分の MFGS（ガウスザイデル法の変形）をスレッド並列化する際に用いている Block Red-Black 法のブロック分割に関してパラメトリックスタディを行い，ブロック数の最適化を行った．図 7 に 160^3 の例を示す．

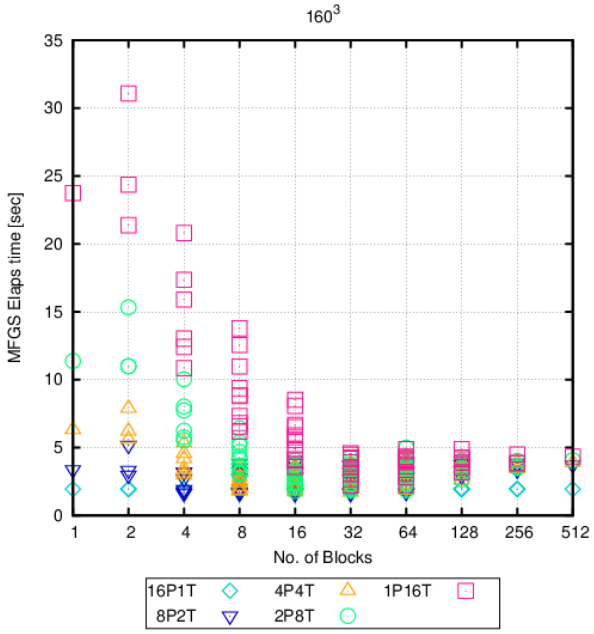


図 7　Block Red-Blackのブロック分割による性能への影響

8) ?t

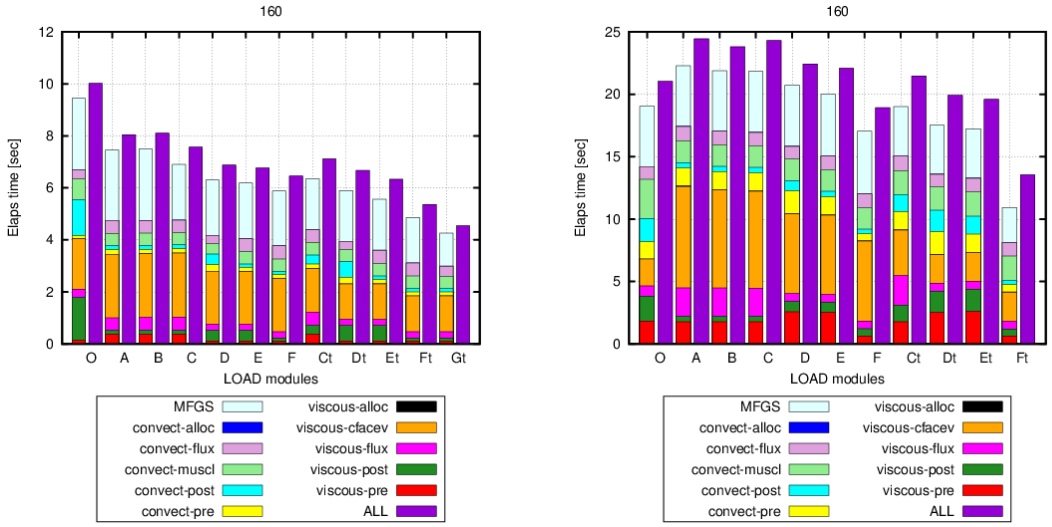
高速化チューニングは主に flux 配列のインデックスに着目していたが、cell 配列のインデックスを（i,j,k,:）から（:,i,j,k）に変更したものを?t とした．これによりケースによっては cell 配列と flux 配列のインデックスが異なる場合もある．

図 8 に高速化チューニングの結果を示す．興味深いことに，A の修正は SORA-MA にとっては高速化になっているが、SORA-PP に対しては逆に遅くなっている．SORA-MA も SORA-PP も同じスカラーCPU であるが，うれしい話ではないがチューニングによっては正反対の結果になる場合があることがわかる．

今回の計測ではそれぞれの部分毎に計算時間を取得しているため高速化チューニングによって、ある部分は高速になるが，他の部分は逆に遅くなるという事が発生することもわかった．図 9 に示すように対流項と粘性項に関して比較を行うと，C から D で flux 配列のインデックスを変化させると，対流項は遅くなり，粘性項は速くなっていることがわかる．この結果を受けて E では対流項と粘性項の flux 配列のインデックスをそれぞれ最適なものとした．この違いが発生した原因であるが，粘性項は対流項に比べて，速度などの物理量の微分を多く計算する必要が有り，そのため flux 配列の使い回しが多いことが原因と考え

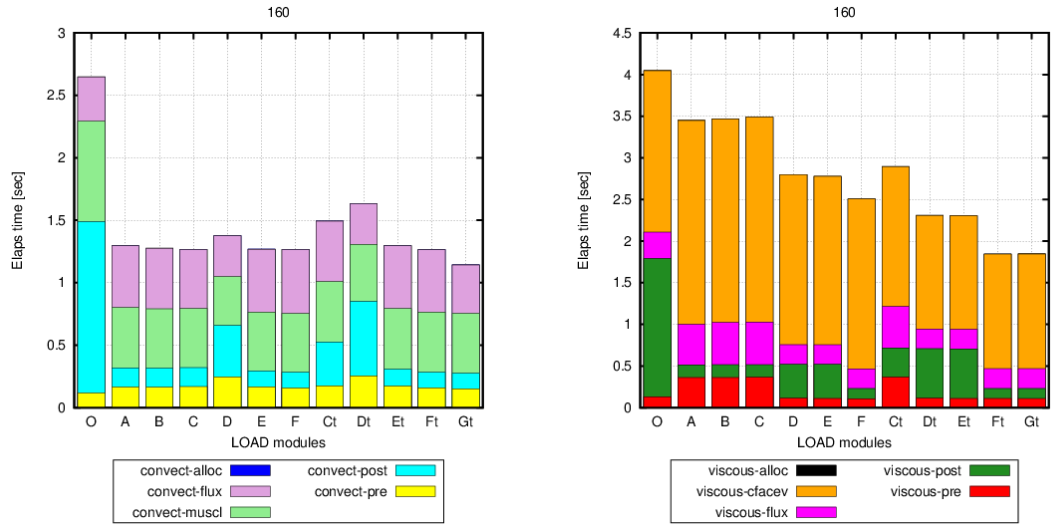
ているが，未検証である．

プログラム全体で見た場合，全体の計算時間を陰解法、粘性項、対流項が大体 3 分割していることがわかる．その中でも viscous_cfacev（各種物理量の微分を計算する部分），MFGS（時間積分）が比較的大きな割合を占めている．



a) SORA-MA (FX100)　　SORA-PP (Intel)

図 8　高速化チューニング結果（ブロックサイズは160^3）



a）対流項　　b）粘性項

図 9　対流項と粘性項

６．おわりに

平成 27 年度から稼働した JAXA の新スーパーコンピュータシステム JSS2 の中核である SORA –MA について紹介すると同時に，アプリケーションによる性能評価、高速化チュー

ニングについて紹介した．本報告ではノード性能に注目して性能評価・高速化チューニングを実施したが，今後は更なる高速化とノード間での性能評価・高速化チューニングを実施する予定である．

参考文献

(1) 藤田直行, JSS2 システム概要と活用・運用, 第 47 回流体力学講演会/第 33 回航空宇宙数値シミュレーション技術シンポジウム 1C08, 2015
(2) http://www.fujitsu.com/jp/products/computing/servers/supercomputer/primehpc-fx100/
(3) STREAM: Sustainable Memory Bandwidth in High Performance Computers, http://www.cs.virginia.edu/stream/.
(4) R.Takaki, K. Yamamoto, T. Yamane, S. Enomoto, and J. Mukai. The Development of the UPACS CFD Environment. Vol. 2858 of Lecture Notes in Computer Science, pp.307-319. Springer, 2003.