2.1 圧縮性流体解析プログラムによるFX100の性能評価と高速化チューニング

宇宙航空研究開発機構 宇宙科学研究所
高木　亮治

2.1.1　はじめに

JAXAで開発を行っている三次元圧縮性流体解析プログラムUPACS[1)]はステンシル系プログラムである。ここではUPACSから機能を最低限に絞り込んだカーネルプログラムUPACS-Liteを用いてFX100の性能評価ならびにチューニングを実施したのでその結果について紹介する。

2.1.2　UPACS-Lite

UPACS-LiteはJAXAが開発している実用的なUPACSの機能を絞り込んで性能評価や高速化チューニングを実施するためのカーネルプログラムである。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、並列化は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、図　1で示すようにステンシルの形（メモリアクセスパターン、量）や演算パターン・量が異なる。

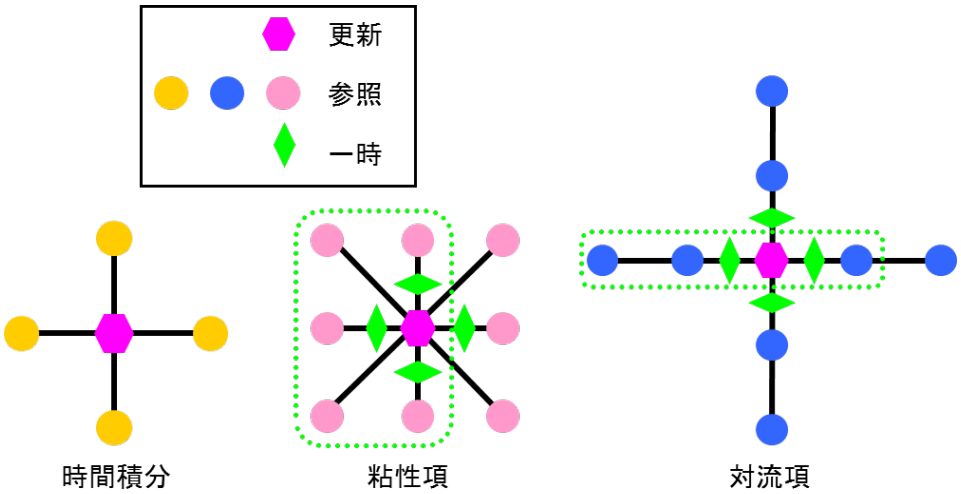


図　1　UPACS-Liteで使われるステンシル

また、UPACSでは図　2で示すように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や形が異なり、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。

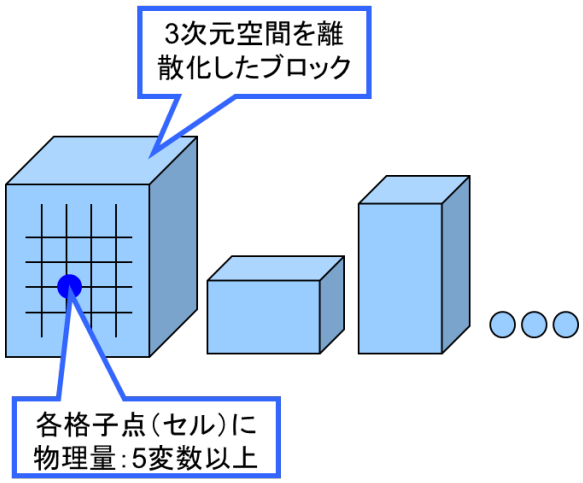


図 2　複数ブロックの構造格子をデータ構造にマッピング

2.1.3　測定環境

FX100（JAXA の JSS2）を用いて性能評価を行った。表 1 に FX100 の諸元を示す。

表 1　測定環境（JAXA の FX100、一部に推定値あり）

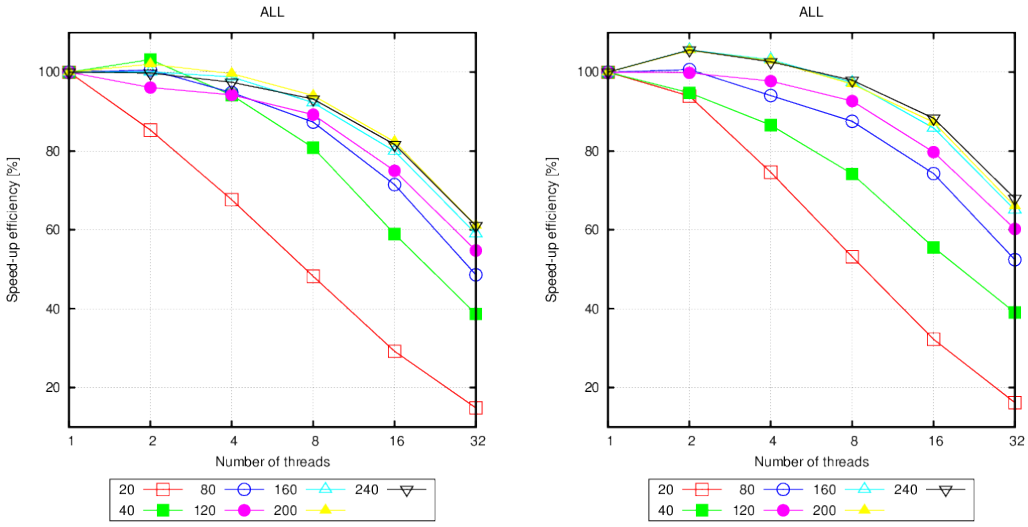
	SORA-MA (FX100)	SORA-PP
マシン	PRIMEHPC FX100	PRIMERGY RX350S8
CPU	Fujitsu SPARC64™XIfx	Intel Xeon E5-2643V2
周波数	1.975GHz	3.5GHz
CPU/ノード	1	2
コア/CPU	32	6
コア/ノード	32	12
理論性能	1.011TFLOPS	0.336TFLOPS
メモリ性能	431GB/s	119.4GB/s
コンパイルオプション	-Kfast, parallel, openmp, noalias=s、array_private, preex, ocl, XFILL -Qt -x-	-O3 -parallel -openmp -fast

2.1.4　測定結果

2.1.4.1　スレッドスケーラビリティ

ノード内スレッド並列性能のスケーラビリティを測定した結果を図 3 に示す。ここでは問題規模を一定にしたストロングスケーリングの性能を評価している。ブロックサイズを増やす事で並列性能が向上していることがわかる。また 80%を目安と考えると 8 スレッドまでは十分な効率が出ていると判断した。図 3a)はデフォルトの状態、 図 3b)は実行時の

オプションでインターリーブを用いた場合を示す。インターリーブを用いることで１スレッドの性能が悪化する。そのため多スレッドの性能は見かけ上向上する。



a) default　　b) interleave

図 ３ ノード内スレッドスケーラビリティ

2.1.4.2　ハイブリッド並列

ハイブリッド並列の評価結果を図４に示す。

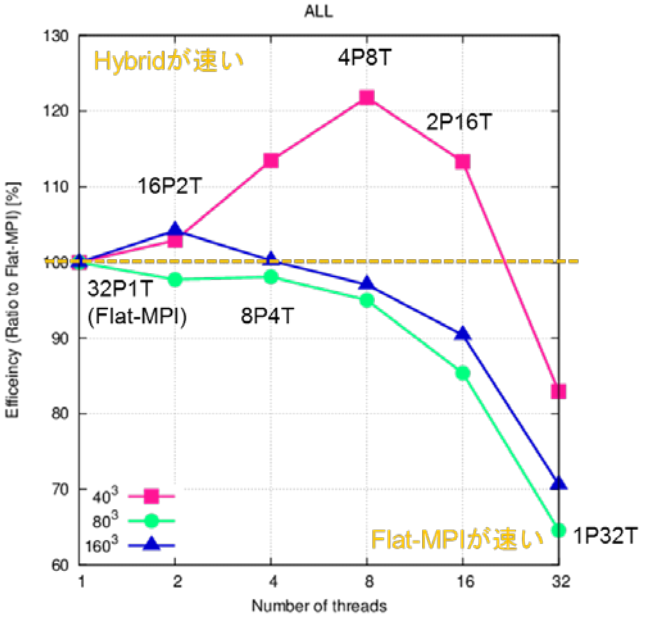
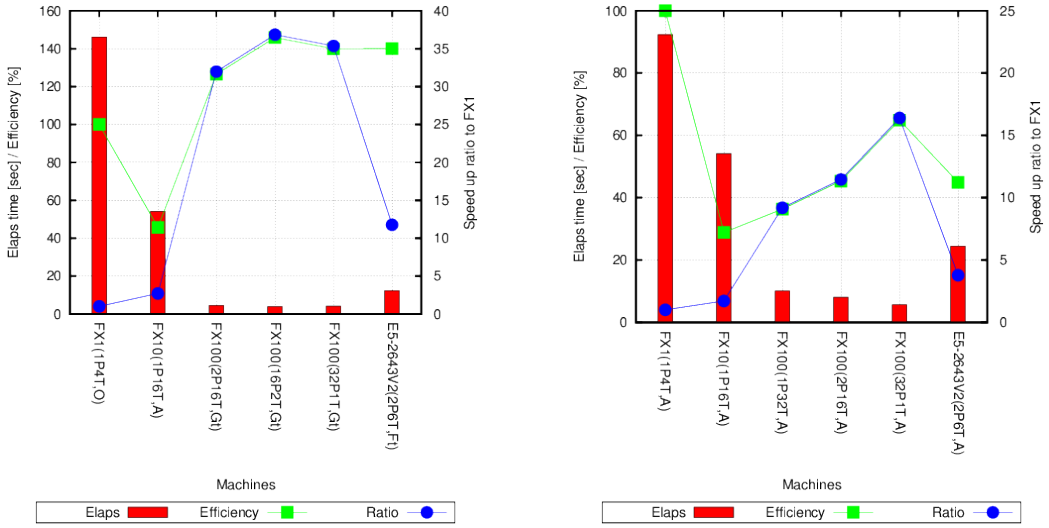


図 ４ ハイブリッド並列 vs フラット MPI

1 ノードの結果でノードに割り当てるブロックサイズを 40^3、80^3、160^3とした。ブロックの1辺の格子点数をNとすると、計算負荷と通信負荷はN^3対N^2となるのでNが大きいほど、相対的に通信負荷が減りフラット MPI 優位になると考えられる。40^3と 160^3の比較ではその傾向が見られるが、80^3は異なる傾向となった。原因は不明である。64 ノードまでの計測を行ったが結果は 1 ノードと変わらなかった。た多ノードになるほどハイブリッド並列が優位になると想定されるが、この問題規模、H/W のバランス（CPU 演算速度と通信速度のバランス）ではハイブリッド並列の優位は見られなかった。今後大規模ノードでの計測を試みる。

2.1.4.3　FX1、FX10、FX100、インテル CPU との比較

FX1(JSS)、FX10、FX100(SORA-MA)、インテル CPU(SORA-PP)の比較を行った結果を図 5に示す。図 5a）は H/W の性能差だけでなく後述する S/W チューニングを実施した効果を含んだもの、図 5b)は同じプログラムで H/W の差異（各 H/W で再コンパイルは実施した） を比較したものになる。S/W チューニングを含めると FX100 では FX1 の約 32 倍の性能が達成できた。この時 FX100 で標準的な 1CPU を 2 プロセス 16 スレッドで利用している。なお、フラット MPI の方が速く、その場合は約 37 倍の性能が出せた。一方、同一プログラムの再コンパイルだけの比較では FX100 は FX1 に比べて約 11 倍となり、フラット MPI では約 16 倍となった。理論ピーク性能では FX100 は FX1 の約 25 倍となるべきであるが、H/W の性能を活用するためにはチューニングが不可欠であることがわかる。



a）S/W チューニングを含む比較　　b）H/W だけの比較

図 5 FX1，FX10，FX100，インテル CPU との比較

2.1.5　高速化チューニング

FX100 向けにいくつかの高速化チューニングを実施した。詳細を表 2 に示す。

表 2　FX100 向け高速化チューニングの実施項目

版	実施項目
0	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	?={C,D,E,F,G}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 flux 配列で使われるが、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。表 3 に具体例を示す。

表 3　データ構造（AOS、SOA）の例

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

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

主にFX10向けの高速化チューニングとして流束（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に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化されない状況となる。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,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ループ内全てが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手動でj,kループを融合してループ長を稼いだ(二重ループの一重化)。

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）

を実施した。

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適した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重ループに追加して物理量のループ（n=1,nPhys、計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自動並列が未対応な部分があり、その部分を 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 6 に 160^3の例を示す。理由は不明だがブロック数がスレッド数の 2 倍の時に最速となった。

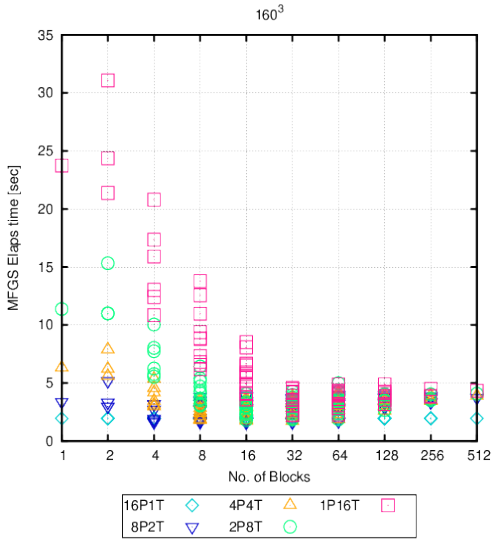


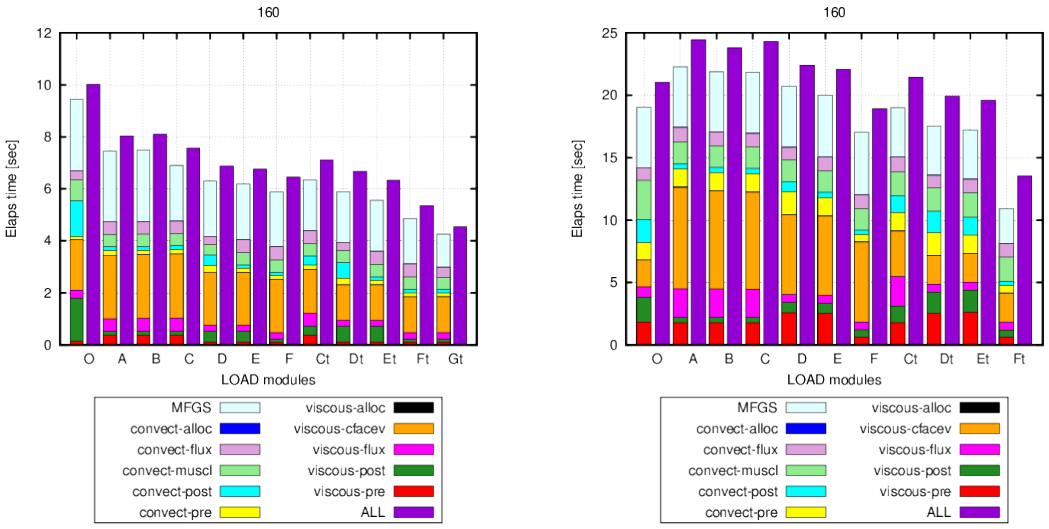
図 6 Block Red-Black のブロック分割による性能への影響

8) Ct, Dt, Et, Ft, Gt

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。

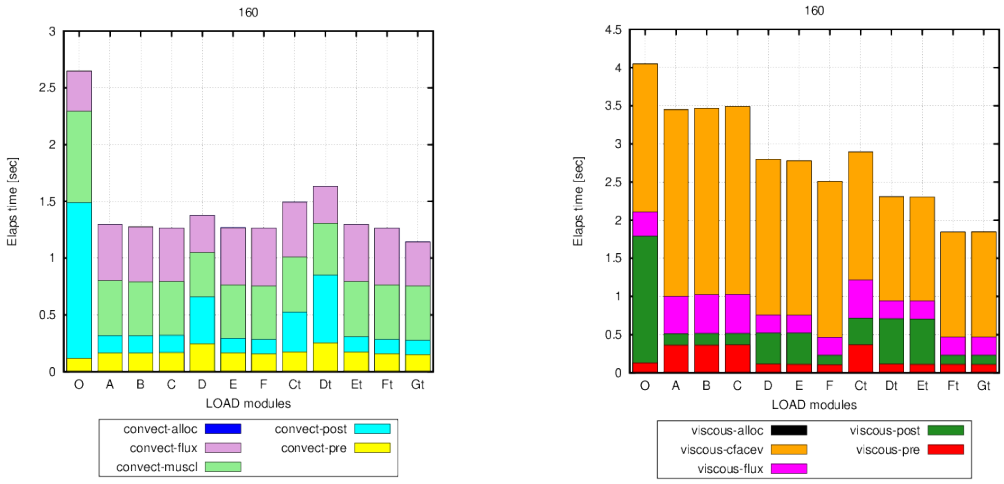
図 7 にこれらの高速化チューニングの結果を示す。A の修正は SORA-MA にとっては高速化になっているが SORA-PP に対しては逆に遅くなっていることがわかる。SORA-MA も SORA-PP も同じスカラーCPU であるがチューニングによっては正反対の結果になる場合があることがわかった。また、ある高速化チューニングによってある部分は高速になるが、他

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 8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は前述したように、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）の再利用性が高くスカラー型のインデックスが適しているが、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。



a）SORA-MA(FX100)　　b）SORA-PP(Intel CPU)

図 7 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 8 対流項と粘性項の差

プログラム全体では時間積分（陰解法）、粘性項、対流項が大体 3 分割しており、その中でも viscous_cfacev（粘性流束の計算のため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。所謂ホットスポットが存在するようなプログラムとは異なるため、高速化チューニングではより多くの部分を対象とする必要がある。

2.1.6　おわりに

ステンシル系アプリケーションによる FX100 の性能評価、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。

A.　STREAM の性能評価

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 では幾つかの演算種類があるが、ここでは一般的な 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 A-1 に TRIAD の測定結果を示す。

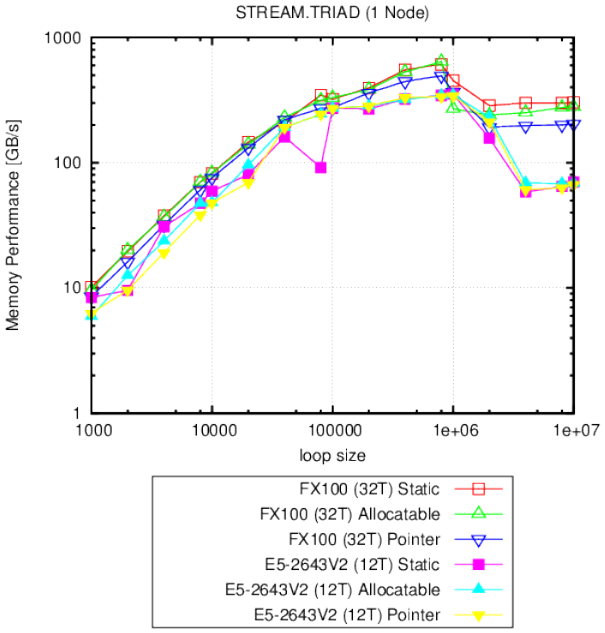


図 A-1 TRIAD によるメモリ性能(SORA-MA, PP)

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（現状では、contiguos 属性を設定することで性能が改善される）。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 A-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。

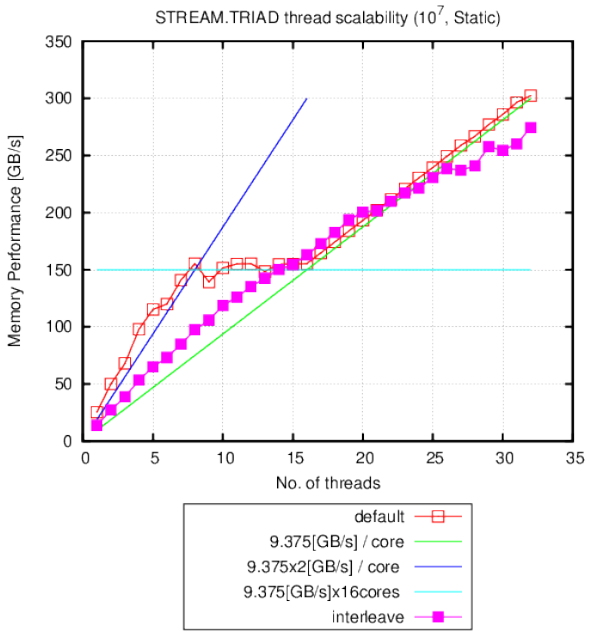


図 A-2 TRIAD(ループ長は 10^7)のスレッドスケーラビリティ (SORA-MA)

参考文献

1) 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.

UPACSのFX10での性能評価

高木　亮治
宇宙航空研究開発機構

性能評価概要

- コード：圧縮性流体解析プログラムUPACS-L
  - 陰解法（Block Red-Black、2nd Euler、内部反復は2回）
  - 160x160x160の1ブロック（1プロセス計算）
- 環境：
  - FX10@東大
    - オプション：-Kfast, parallel, openmp, swp, XFILL, optmsg=2, noalias=s, array_private, preex, dynamic_iteration -Qt -x-

流束データの階層構造

Original	Tune1（flux等の配列を変更）
type cellFaceType real(8) :: area,nt real(8), dimension(3) :: nv real(8), dimension(5) :: q_l,q_r real(8), dimension(5) :: flux end type type visCellFaceType real(8) :: area, mu real(8), dimension(3) :: nv, dTdx, u real(8), dimension(5) :: flux real(8), dimension(3,3) :: dudx end type	type cellFaceType real(8), pointer, dimension(:,:,:) :: area,nt real(8),pointer, dimension(:,:,:,:) :: nv real(8), pointer, dimension(:,:,:,:) :: q_l,q_r real(8), pointer, dimension(:,:,:,:) :: flux end type type visCellFaceType real(8), pointer, dimension(:,:,:) :: area, mu real(8), pointer, dimension(:,:,:,:) :: nv, dTdx, u real(8), pointer, dimension(:,:,:,:) :: flux real(8), pointer, dimension(:,:,:,:,:) :: dudx end type
type(*), pointer, dimension(:,:,:) :: cface do n ; do k ; do j ; do i cface(i,j,k)%flux(n) … enddo	type(*) :: cface do n ; do k ; do j ; do i cface%flux(i,j,k,n) … enddo

チューニング概要

- original
- tune1：構造体の変更
  - tune1a：tune1 + 粘性項の修正
    - 一時配列のスカラー化：a(3,3) -> a_11,a_12,…
      - SIMD化の促進（viscous_cfacev：0% → 99.19%）
    - 組み込み関数（matmul, dot_product）の手動展開
      - SIMD化の促進（viscous_flux：10.64% → 99.11%）
  - tune1b:対流項はtune1、粘性項はoriginal

計測結果（経過時間 [sec]）

		Original	Tune1	Tune1a	Tune1b
ALL		58.00	623.0	50.20	46.83
Convect（対流項）	alloc ①	6.760	5.791	5.763	5.253
	pre	0.4674	0.4852	0.4854	0.4853
	muscl	3.913	1.978	1.984	2.034
	flux	1.334	②a 1.041	1.052	1.037
	post	4.418	0.6837	0.6832	0.6823
Viscous（粘性項）	alloc ①	8.286	②b 5.335	5.331	8.414
	pre	0.5870	1.720	1.720	0.5929
	cfacev	8.741	③a 27.80	18.01	9.007
	flux	1.482	564.5	③b 1.314	1.482
	post	4.421	②a 0.6803	0.6798	4.422
MFGS（時間積分）		11.98	8.188	8.436	8.243

① alloc（配列のallocation）に時間がかかっている。
② original→tune1で高速化
  a. 対流項のmuscl, flux, post、粘性項のpost
  b. 粘性項のalloc（？）
③ original→tune1で低速化
  a. 粘性項のpre, cfacev, flux（cfacevとfluxではSIMD化が減少）
  b. tune1aでSIMD化を促進

original→tune1で高速化

convect-{muscl, flux, post}

viscous-post

（viscous-alloc）

convect_muscl：

	Original 3.913[s]	Tune1 1.978[s]	Tune1a 1.984[s]
浮動小数点演算ピーク比	6.4	12.7	12.6
メモリ/L2/L1ビジー率 [%]	81/31/14	67/39/27	64/30/27
SIMD演算命令率 [%]	98.8	98.8	98.8
SIMDロード/ストア命令率 [%]	5.1/3.7	0/0	0/0
L1Dミス率 [%]	11.3	4.6	4.6
L1Dミスdm/hwpf/swpf率 [%]	43.8/56.2/0	40.3/60.0/0	40.7/59.3/0
L2ミス率 [%]	8.96	3.37	3.37
L2ミスdm率/pf率 [%]	24.4/75.6	15.3/84.7	15.3/84.7
ロード・ストア命令率 [%]	24.0	24.7	24.7
SIMD演算/演算命令率 [%]	34.0/0.42	25.5/0.32	25.5/0.32
SIMD積和演算/積和演算命令率 [%]	17.6/0.23	13.3/0.17	13.3/0.17
pf/分岐/その他命令率 [%]	0/0.19/23.6	0/3.26/32.8	0/3.26/32.8

convect_flux：

	Original 1.334[s]	Tune1 1.041[s]	Tune1a 1.052[s]
浮動小数点演算ピーク比	16.9	21.7	21.5
メモリ/L2/L1ビジー率 [%]	90/26/14	65/40/20	63/35/19
SIMD演算命令率 [%]	99.2	99.2	99.2
SIMDロード/ストア命令率 [%]	0/0	0/0	0/0
L1Dミス率 [%]	6.34	6.53	6.72
L1Dミスdm/hwpf/swpf率 [%]	46.5/53.5/0	26.1/73.9/0	26.4/73.6/0
L2ミス率 [%]	6.36	6.02	6.02
L2ミスdm率/pf率 [%]	4.52/95.5	19.6/80.4	19.5/80.5
ロード・ストア命令率 [%]	19.0	18.3	18.3
SIMD演算/演算命令率 [%]	48.1/0.37	43.4/0.34	43.42/0.34
SIMD積和演算/積和演算命令率 [%]	27.8/0.27	25.1/0.25	25.1/0.25
pf/分岐/その他命令率 [%]	0/0.5/3.91	0/0.13/12.5	0/0.13/12.5

convect_post：

	Original 4.418[s]	Tune1 0.6837[s]	Tune1a 0.6832[s]
浮動小数点演算ピーク比	0.35	2.23	2.24
メモリ/L2/L1ビジー率 [%]	73/52/11	86/50/30	86/50/30
SIMD演算命令率 [%]	98.8	98.8	98.8
SIMDロード/ストア命令率 [%]	0/0	0/0	0/0
L1Dミス率 [%]	33.6	4.23	4.23
L1Dミスdm/hwpf/swpf率 [%]	90.6/9.38/0	1.13/98.9/0	1.13/98.9/0
L2ミス率 [%]	31.8	4.11	4.11
L2ミスdm率/pf率 [%]	57.6/42.4	7.86/92.1	7.84/92.2
ロード・ストア命令率 [%]	47.7	47.7	47.7
SIMD演算/演算命令率 [%]	4.67/0.06	4.68/0.06	4.68/0.06
SIMD積和演算/積和演算命令率 [%]	4.67/0.06	4.68/0.06	4.68/0.06
pf/分岐/その他命令率 [%]	0/1.2/41.6	0/1.2/41.6	0/1.2/41.6

{convect, viscous}_post

convect-post

original（4.418 [s]）

```
 do n=1,bdtv_nFlowVar
!$omp parallel do private(i,j,k,im,jm,km)
      do k=1,blk%kn
        do j=1,blk%jn
          do i=1,blk%in
            im = i-idelta(1) ; jm = j-idelta(2) ; km = k-idelta(3)
            blk%dq(i,j,k,n) = blk%dq(i,j,k,n) - blk%inv_vol(i,j,k) &
                  * ( cface(i,j,k)%flux(n) - cface(im,jm,km)%flux(n) )
          end do ; enddo ; enddo
!$omp end parallel do
     end do
```

tune1（0.6837 [s]）

```
  do n=1,bdtv_nFlowVar
!$omp parallel do private(i,j,im,jm,km)
      do k=1,blk%kn
        do j=1,blk%jn
          do i=1,blk%in
            im = i-idelta(1) ; jm = j-idelta(2) ; km = k-idelta(3)
            blk%dq(i,j,k,n) = blk%dq(i,j,k,n) - blk%inv_vol(i,j,k) &
                  * ( cface%flux(i,j,k,n) - cface%flux(im,jm,km,n) )
          end do ; enddo ; enddo
!$omp end parallel do
     end do
```

viscous-post

original（4.421 [s]）

```
   do n=1,bdtv_nFlowVar
!$omp parallel do private(i,j,k,im,jm,km)
      do k=1,blk%kn
        do j=1,blk%jn
          do i=1,blk%in
            im = i -idelta(1) ; jm = j -idelta(2) ; km = k -idelta(3)
            blk%dq(i,j,k,n) = blk%dq(i,j,k,n)  - blk%inv_vol(i,j,k) &
                        * ( cface(i,j,k)%flux(n) - cface(im,jm,km)%flux(n) )
          end do ; enddo ; enddo
!$omp end parallel do
     end do
```

tune1（0.6803 [s]）

```
  do n=1,bdtv_nFlowVar
!$omp parallel do private(i,j,k,im,jm,km)
      do k=1,blk%kn
        do j=1,blk%jn
          do i=1,blk%in
            im = i -idelta(1) ; jm = j -idelta(2) ; km = k -idelta(3)
            blk%dq(i,j,k,n) = blk%dq(i,j,k,n) - blk%inv_vol(i,j,k) &
                        * ( cface%flux(i,j,k,n) - cface%flux(im,jm,km,n) )
          end do ; enddo ; enddo
!$omp end parallel do
     end do
```

original→tune1で低速化

viscous-{pre,cfacev, flux}

tune1のチューニング（tune1a）

viscous_pre

original（0.5870 [s]）	tune1（1.720 [s]）
!$omp parallel do private(i,j,k) do k=-1,blk%kn do j=-1,blk%jn do i=-1,blk%in cface(i,j,k)%dudx(:,:) = 0.0 cface(i,j,k)%dTdx(:) = 0.0 ; cface(i,j,k)%u(:) = 0.0 cface(i,j,k)%area = 0.0 ; cface(i,j,k)%nv(:) = 0.0 cface(i,j,k)%mu = 0.0 ; cface(i,j,k)%flux(:) = 0.0 end do ; end do ; end do !$omp end parallel do	cface%dudx = 0.0 cface%dTdx = 0.0 ; cface%u = 0.0 cface%area = 0.0 ; cface%nv = 0.0 cface%mu = 0.0 ; cface%flux = 0.0

	original	tune1	tune1a
メモリ/L2/L1ビジー率 [%]	93/40/21	27/12/31	27/12/31
SIMDロード/ストア命令率 [%]	0/31.6	0/24.8	0/24.8
L1Dミス率 [%]	6.25	6.04	6.04
L1Dミスdm/hwpf/swpf率 [%]	21.6/4.11/74.3	65.9/23.0/11.1	65.9/23.0/11.1
L2ミス率 [%]	6.25	3.99	3.99
L2ミスdm率/pf率 [%]	4.39/95.6	19.6/80.4	19.7/80.4
ロード・ストア命令率 [%]	9.82	23.5	23.5
pf/分岐/その他命令率 [%]	1.55/14.2/74.4	0.7/18.9/56.9	0.7/18.9/56.9

viscous_pre

Original（0.5870 [s]）

```
31                !$omp parallel do private(i,j,k)
32    1  p              do k=-1,blk%kn
33    2  p               do j=-1,blk%jn
34    3  p                 do i=-1,blk%in
                <<< Loop-information Start >>>
                <<<  [OPTIMIZATION]
                <<<    SIMD
                <<<    PREFETCH       : 2
                <<<      cface: 2
                <<< Loop-information  End >>>
35    3  p   v              cface(i,j,k)%dudx(:,:) = 0.0
                <<< Loop-information Start >>>
                <<<  [OPTIMIZATION]
                <<<    FULL UNROLLING
                <<< Loop-information  End >>>
36    3  p   f              cface(i,j,k)%dTdx(:)   = 0.0 ; …
                <<< Loop-information Start >>>
                <<<  [OPTIMIZATION]
                <<<    SIMD
                <<< Loop-information  End >>>
37    3  p   m              cface(i,j,k)%area      = 0.0 ; …
                <<< Loop-information Start >>>
                <<<  [OPTIMIZATION]
                <<<    SIMD
                <<< Loop-information  End >>>
38    3  p   m              cface(i,j,k)%mu        = 0.0 ; …
39    3  p                 end do
40    2  p               end do
41    1  p             end do
42                !$omp end parallel do
```

Tune1（1.720 [s]）

```
                <<< Loop-information Start >>>
                <<<  [PARALLELIZATION]
                <<<    Standard iteration count: 2
                <<<  [OPTIMIZATION]
                <<<    SIMD
                <<<    OTHER PREFETCH : 1
                <<<    OTHER XFILL    : 1
                <<< Loop-information  End >>>
37      pp   v        cface%dudx = 0.0
                <<< Loop-information Start >>>
                <<<  [PARALLELIZATION]
                <<<    Standard iteration count: 2
                <<<  [OPTIMIZATION]
                <<<    SIMD
                <<<    OTHER PREFETCH : 2
                <<<    OTHER XFILL    : 2
                <<< Loop-information  End >>>
38      pp   v        cface%dTdx = 0.0 ; cface%u    = 0.0
                <<< Loop-information Start >>>
                <<<  [PARALLELIZATION]
                <<<    Standard iteration count: 2
                <<<  [OPTIMIZATION]
                <<<    SIMD
                <<<    OTHER PREFETCH : 4
                <<<    OTHER XFILL    : 4
                <<< Loop-information  End >>>
39      pp   v        cface%area = 0.0 ; cface%nv   = 0.0
                <<< Loop-information Start >>>
                <<<  [PARALLELIZATION]
                <<<    Standard iteration count: 2
                <<<  [OPTIMIZATION]
                <<<    SIMD
                <<<    OTHER PREFETCH : 4
                <<<    OTHER XFILL    : 4
                <<< Loop-information  End >>>
40      pp   v        cface%mu   = 0.0 ; cface%flux = 0.0
```


convect_pre

convect-pre

original（0.4674 [s]）

```
!$omp parallel do private(i,j,k)
   do k=1,blk%kn
   do j=1,blk%jn
   do i=1,blk%in
     cface(i,j,k)%area = 0.0
     cface(i,j,k)%nt   = 0.0
     cface(i,j,k)%nv(:) = 0.0
     cface(i,j,k)%q_l(:)  = 0.0
     cface(i,j,k)%q_r(:)  = 0.0
     cface(i,j,k)%flux(:) = 0.0
   enddo ; enddo ; enddo
!$omp end parallel do
```

tune1（0.4852 [s]）

```
cface%area = 0.0
cface%nt   = 0.0
cface%nv   = 0.0
cface%q_l  = 0.0
cface%q_r  = 0.0
cface%flux = 0.0
```

viscous-pre

original（0.5870 [s]）

```
!$omp parallel do private(i,j,k)
    do k=-1,blk%kn
     do j=-1,blk%jn
       do i=-1,blk%in
         cface(i,j,k)%dudx(:,:) = 0.0
         cface(i,j,k)%dTdx(:)   = 0.0 ; cface(i,j,k)%u(:)    = 0.0
         cface(i,j,k)%area      = 0.0 ; cface(i,j,k)%nv(:)   = 0.0
         cface(i,j,k)%mu        = 0.0 ; cface(i,j,k)%flux(:) = 0.0
       end do ; enddo ; enddo
!$omp end parallel do
```

tune1（1.720 [s]）

```
cface%dudx = 0.0
cface%dTdx = 0.0 ; cface%u    = 0.0
cface%area = 0.0 ; cface%nv   = 0.0
cface%mu   = 0.0 ; cface%flux = 0.0
```

※doループを使って明示的に書き下しても性能は変わらず

viscous_cfacev：変数のスカラー化

Tune1（27.8 [s]）

```
!$omp parallel do private(i,j,k,….)
  do k = isrt(3),iend(3)
    do j = isrt(2),iend(2)
      do i = isrt(1),iend(1)

        i1 = i  + idelta(1) ; j1 = j  + idelta(2) ; k1 = k  + idelta(3)

        !    xi derivative

        u_0(1:3) = blk%q(i ,j ,k ,2:4) / blk%q(i ,j ,k ,1)
        u_1(1:3) = blk%q(i1,j1,k1,2:4) / blk%q(i1,j1,k1,1)

        T_0     = GAMMA * blk%p(i ,j ,k ) / blk%q(i ,j ,k ,1)
        T_1     = GAMMA * blk%p(i1,j1,k1) / blk%q(i1,j1,k1,1)

        dudx1(1,:) = blk%xix(i ,j ,k ,ixi,:) * ( u_1(1) - u_0(1) )
        dudx1(2,:) = blk%xix(i ,j ,k ,ixi,:) * ( u_1(2) - u_0(2) )
        dudx1(3,:) = blk%xix(i ,j ,k ,ixi,:) * ( u_1(3) - u_0(3) )
        dTdx1(  :) = blk%xix(i ,j ,k ,ixi,:) * ( T_1    - T_0    )

…
```

Tune1a（18.01 [s]）

```
!$omp parallel do private(i,j,k,…)
  do k = isrt(3),iend(3)
    do j = isrt(2),iend(2)
      do i = isrt(1),iend(1)

        i1 = i  + idelta(1) ; j1 = j  + idelta(2) ; k1 = k  + idelta(3)

        !    xi derivative

        u0_1 = blk%q(i ,j ,k ,2) / blk%q(i ,j ,k ,1)
        u0_2 = blk%q(i ,j ,k ,3) / blk%q(i ,j ,k ,1)
        u0_3 = blk%q(i ,j ,k ,4) / blk%q(i ,j ,k ,1)
        u1_1 = blk%q(i1,j1,k1,2) / blk%q(i1,j1,k1,1)
        u1_2 = blk%q(i1,j1,k1,3) / blk%q(i1,j1,k1,1)
        u1_3 = blk%q(i1,j1,k1,4) / blk%q(i1,j1,k1,1)

        T_0     = GAMMA * blk%p(i ,j ,k ) / blk%q(i ,j ,k ,1)
        T_1     = GAMMA * blk%p(i1,j1,k1) / blk%q(i1,j1,k1,1)

        dudx1_11 = blk%xix(i ,j ,k ,ixi,1) * ( u1_1 - u0_1 )
        dudx1_12 = blk%xix(i ,j ,k ,ixi,2) * ( u1_1 - u0_1 )
        dudx1_13 = blk%xix(i ,j ,k ,ixi,3) * ( u1_1 - u0_1 )
        dudx1_21 = blk%xix(i ,j ,k ,ixi,1) * ( u1_2 - u0_2 )
        dudx1_22 = blk%xix(i ,j ,k ,ixi,2) * ( u1_2 - u0_2 )
        dudx1_23 = blk%xix(i ,j ,k ,ixi,3) * ( u1_2 - u0_2 )
        dudx1_31 = blk%xix(i ,j ,k ,ixi,1) * ( u1_3 - u0_3 )
        dudx1_32 = blk%xix(i ,j ,k ,ixi,2) * ( u1_3 - u0_3 )
        dudx1_33 = blk%xix(i ,j ,k ,ixi,3) * ( u1_3 - u0_3 )
        dTdx1_1 = blk%xix(i ,j ,k ,ixi,1) * ( T_1    - T_0    )
        dTdx1_2 = blk%xix(i ,j ,k ,ixi,2) * ( T_1    - T_0    )
        dTdx1_3 = blk%xix(i ,j ,k ,ixi,3) * ( T_1    - T_0    )
```


viscous_cfacev：変数のスカラー化

Tune1（27.8 [s]）

```
121 !$omp parallel do private(i,j,k,…)
122
123   1  p            do k = isrt(3),iend(3)
124   2  p              do j = isrt(2),iend(2)
125   3  p                do i = isrt(1),iend(1)
126   3
127   3  p                  i1 = i  + idelta(1) ; …
128   3
129   3                    !    xi derivative
130   3
                 <<< Loop-information Start >>>
                 <<<  [OPTIMIZATION]
                 <<<    FULL UNROLLING
                 <<< Loop-information  End >>>
131   3  p  f              u_0(1:3) = …
132   3  p  f              u_1(1:3) = …
133   3
134   3  p                 T_0     = …
135   3  p                 T_1     = …
136   3
                 <<< Loop-information Start >>>
                 <<<  [OPTIMIZATION]
                 <<<    FULL UNROLLING
                 <<< Loop-information  End >>>
137   3  p  f              dudx1(1,:) = …
138   3  p  f              dudx1(2,:) = …
139   3  p  f              dudx1(3,:) = …
140   3  p  f              dTdx1(  :) = …
```

Tune1a（18.01 [s]）

```
139 !$omp parallel do private(i,j,k,…)
148   1  p            do k = isrt(3),iend(3)
149   2  p              do j = isrt(2),iend(2)
                 <<< Loop-information Start >>>
                 <<<  [OPTIMIZATION]
                 <<<    SIMD
                 <<< Loop-information  End >>>
150   3  p  v            do i = isrt(1),iend(1)
151   3
152   3  p  v              i1 = i  + idelta(1) ; …
153   3
154   3                    !    xi derivative
155   3
158   3  p  v              u0_1 = …
159   3  p  v              u0_2 = …
160   3  p  v              u0_3 = …
161   3  p  v              u1_1 = …
162   3  p  v              u1_2 = …
163   3  p  v              u1_3 = …
164   3
165   3  p  v              T_0     = …
166   3  p  v              T_1     = …
167   3
172   3  p  v              dudx1_11 = …
173   3  p  v              dudx1_12 = …
174   3  p  v              dudx1_13 = …
175   3  p  v              dudx1_21 = …
176   3  p  v              dudx1_22 = …
177   3  p  v              dudx1_23 = …
178   3  p  v              dudx1_31 = …
179   3  p  v              dudx1_32 = …
180   3  p  v              dudx1_33 = …
181   3  p  v              dTdx1_1 = …
182   3  p  v              dTdx1_2 = …
183   3  p  v              dTdx1_3 = …
```

viscous_cfacev：

	Original 8.741[s]	Tune1 27.80[s]	Tune1a 18.01[s]
浮動小数点演算ピーク比	6.81	2.19	2.93
メモリ/L2/L1ビジー率 [%]	27/19/33	7/11/16	11/13/12
SIMD演算命令率 [%]	99.1	0	99.2
SIMDロード/ストア命令率 [%]	0/0	0/0	0/0
L1Dミス率 [%]	1.97	3.94	6.97
L1Dミスdm/hwpf/swpf率 [%]	78.4/21.6/0	99.9/0.13/0	99.9/0.14/0
L2ミス率 [%]	1.01	0.68	1.66
L2ミスdm率/pf率 [%]	53.9/47.1	91.3/8.75	91.2/8.78
ロード・ストア命令率 [%]	48.7	39.1	36.8
SIMD演算/演算命令率 [%]	8.6/0.11	0/12.3	10.1/0.11
SIMD積和演算/積和演算命令率 [%]	14.9/0.11	0/16.9	15.9/0.11
pf/分岐/その他命令率 [%]	0/0.13/27.5	0/3.62/28.0	0/4.26/32.8

viscous_flux：組み込み関数の手動展開

Tune1（564.5[s]）

```
!$omp parallel do private(i,j,k,div_u,tau)
   do k = isrt(3),iend(3)
     do j = isrt(2),iend(2)
       do i = isrt(1),iend(1)

[途中略]

        f%flux(i,j,k,1) = 0.
        f%flux(i,j,k,2:4) = f%mu(i,j,k) * matmul(tau(:,:), f%nv(i,j,k,:))
        f%flux(i,j,k,5) = dot_product(f%flux(i,j,k,2:4),f%u(i,j,k,1:3))  &
                    + dot_product(f%dTdx(i,j,k,:) ,f%nv(i,j,k,:)  )  &
                    * f%mu(i,j,k)/PRANDTL/GAMMA_1

        f%flux(i,j,k,:) = -f%area(i,j,k) * f%flux(i,j,k,:)

       end do
     end do
   end do
```

Tune1a（1.314[s]）

```
!$omp parallel do private(i,j,k,div_u,tau)
   do k = isrt(3),iend(3)
     do j = isrt(2),iend(2)
       do i = isrt(1),iend(1)

[途中略]

        f%flux(i,j,k,1) = 0.
        f%flux(i,j,k,2) = f%mu(i,j,k)  &
                * (tau(1,1)*f%nv(i,j,k,1)+tau(1,2)*f%nv(i,j,k,2)+tau(1,3)*f%nv(i,j,k,3))
        f%flux(i,j,k,3) = f%mu(i,j,k)  &
                * (tau(2,1)*f%nv(i,j,k,1)+tau(2,2)*f%nv(i,j,k,2)+tau(2,3)*f%nv(i,j,k,3))
        f%flux(i,j,k,4) = f%mu(i,j,k)  &
                * (tau(3,1)*f%nv(i,j,k,1)+tau(3,2)*f%nv(i,j,k,2)+tau(3,3)*f%nv(i,j,k,3))
        f%flux(i,j,k,5) = f%flux(i,j,k,2)*f%u(i,j,k,1) &
                + f%flux(i,j,k,3)*f%u(i,j,k,2) + f%flux(i,j,k,4)*f%u(i,j,k,3) &
                + f%mu(i,j,k)/PRANDTL/GAMMA_1 &
                * ( f%dTdx(i,j,k,1)*f%nv(i,j,k,1)+f%dTdx(i,j,k,2)*f%nv(i,j,k,2) &
                + f%dTdx(i,j,k,3)*f%nv(i,j,k,3) )

        f%flux(i,j,k,1) = -f%area(i,j,k) * f%flux(i,j,k,1)
        f%flux(i,j,k,2) = -f%area(i,j,k) * f%flux(i,j,k,2)
        f%flux(i,j,k,3) = -f%area(i,j,k) * f%flux(i,j,k,3)
        f%flux(i,j,k,4) = -f%area(i,j,k) * f%flux(i,j,k,4)
        f%flux(i,j,k,5) = -f%area(i,j,k) * f%flux(i,j,k,5)

       end do
     end do
   end do
```

viscous_flux：組み込み関数の手動展開

Tune1（564.5[s]）

```
259                 !$omp parallel do
private(i,j,k,div_u,tau)
260    1  p            do k = isrt(3),iend(3)
261    2  p              do j = isrt(2),iend(2)
262    3  p                do i = isrt(1),iend(1)

[途中略]

277    3  p                  f%flux(i,j,k,1) = …
                  <<< Loop-information Start >>>
                  <<<  [OPTIMIZATION]
                  <<<    FULL UNROLLING
                  <<< Loop-information  End >>>
278    3  p  f               f%flux(i,j,k,2:4) = …
                  <<< Loop-information Start >>>
                  <<<  [OPTIMIZATION]
                  <<<    SIMD
                  <<<    SOFTWARE PIPELINING
                  <<< Loop-information  End >>>
279    3  p  6v              f%flux(i,j,k,5) =  …

                  <<< Loop-information Start >>>
                  <<<  [OPTIMIZATION]
                  <<<    SIMD
                  <<< Loop-information  End >>>
283    3  p  6v              f%flux(i,j,k,:) = …
284    3
285    3  p                end do
286    2  p              end do
287    1  p            end do
```

Tune1a（1.314[s]）

```
442                 !$omp parallel do private(i,j,k,div_u,tau)
443    1  p            do k = isrt(3),iend(3)
444    2  p              do j = isrt(2),iend(2)
                  <<< Loop-information Start >>>
                  <<<  [OPTIMIZATION]
                  <<<    SIMD
                  <<<    SOFTWARE PIPELINING
                  <<< Loop-information  End >>>
445    3  p  v            do i = isrt(1),iend(1)

[途中略]

459    3
460    3  p  v              f%flux(i,j,k,1) = 0.
461    3  p  v              f%flux(i,j,k,2) = …
463    3  p  v              f%flux(i,j,k,3) = …
465    3  p  v              f%flux(i,j,k,4) = …
467    3  p  v              f%flux(i,j,k,5) = …

481    3  p  v              f%flux(i,j,k,1) = -f%area(i,j,k) * f%flux(i,j,k,1)
482    3  p  v              f%flux(i,j,k,2) = -f%area(i,j,k) * f%flux(i,j,k,2)
483    3  p  v              f%flux(i,j,k,3) = -f%area(i,j,k) * f%flux(i,j,k,3)
484    3  p  v              f%flux(i,j,k,4) = -f%area(i,j,k) * f%flux(i,j,k,4)
485    3  p  v              f%flux(i,j,k,5) = -f%area(i,j,k) * f%flux(i,j,k,5)
486    3
487    3  p  v            end do
488    2  p              end do
489    1  p            end do
```

19

viscous_flux：

	Original 1.482[s]	Tune1 564.5[s]	Tune1a 1.314[s]
浮動小数点演算ピーク比	3.81	0.01	4.21
メモリ/L2/L1ビジー率 [%]	93/43/21	0/3/7	60/38/18
SIMD演算命令率 [%]	99.1	10.6	99.1
SIMDロード/ストア命令率 [%]	0/0	0/0	0/0
L1Dミス率 [%]	4.45	2.41	6.49
L1Dミスdm/hwpf/swpf率 [%]	49.3/50.7/0	97.2/1.71/1.07	44.0/56.0/0
L2ミス率 [%]	4.46	0.03	5.96
L2ミスdm率/pf率 [%]	16.0/84.1	92.2/7.83	35.2/64.8
ロード・ストア命令率 [%]	61.5	26.3	49.4
SIMD演算/演算命令率 [%]	19.9/0.19	0.02/0.10	20.2/0.21
SIMD積和演算/積和演算命令率 [%]	13.8/0.10	0/0.09	14.7/0.11
pf/分岐/その他命令率 [%]	0/0.92/3.65	0.04/22.7/50.8	0/0.53/14.9

その他

PAツールの挙動不審

スレッドスケーリング

{convect, viscous}_postの補足

PAを取ると性能が変化する

		tune1	tune1(PA)	tune1b	tune1b(PA)
ALL		634.4	627.8	51.24	46.27
Convect	alloc	5.999	6.128	6.121	5.241
	pre	0.4854	0.4852	0.4879	0.4882
	muscl_flux	3.041	3.545	3.584	3.040
	post	0.6834	0.6778	0.6781	0.6842
Viscous	alloc	5.539	6.894	8.240	8.397
	pre	1.720	1.719	0.5846	0.5863
	cfacev_flux	596.5	596.7	9.650	10.14
	post	0.6793	0.6741	4.425	4.421
MFGS		11.82	8.452	11.79	8.086

古いversionのデータ。最新versionでも一部発生

スレッド並列性能（強スケーリング）

80x80x80					
スレッド数	1	2	4	8	16
計算時間 [sec]	38.05	20.21	10.90	6.268	4.113
速度比	1	1.88	3.49	6.07	9.25
効率 [%]	-	94.1	87.3	75.9	57.8
160x160x160					
スレッド数	1	2	4	8	16
計算時間 [sec]	364.4	190.1	107.7	69.40	57.78
速度比	1	1.917	3.383	5.251	6.307
効率 [%]	-	95.8	84.6	65.6	39.4
320x320x320					
スレッド数	1	2	4	8	16
計算時間 [sec]	3413	1878	1113	676.1	533.0
速度比	1	1.817	3.066	5.048	6.403
効率 [%]	-	90.9	76.7	63.1	40.0

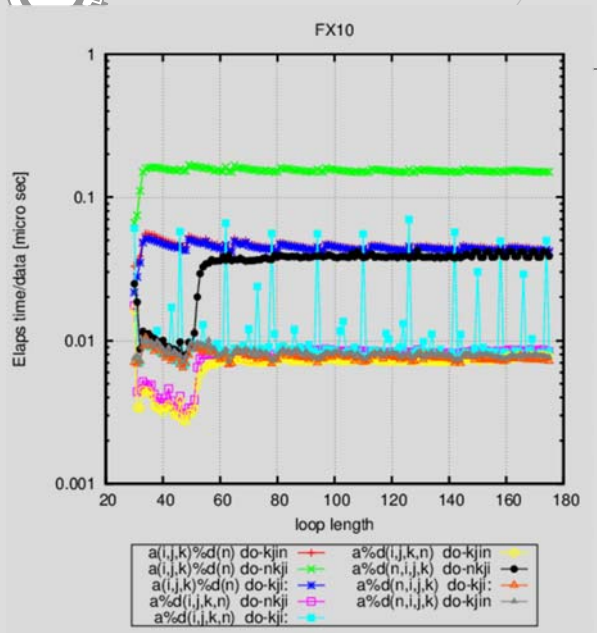
{convect, viscosu}_postのカーネル評価

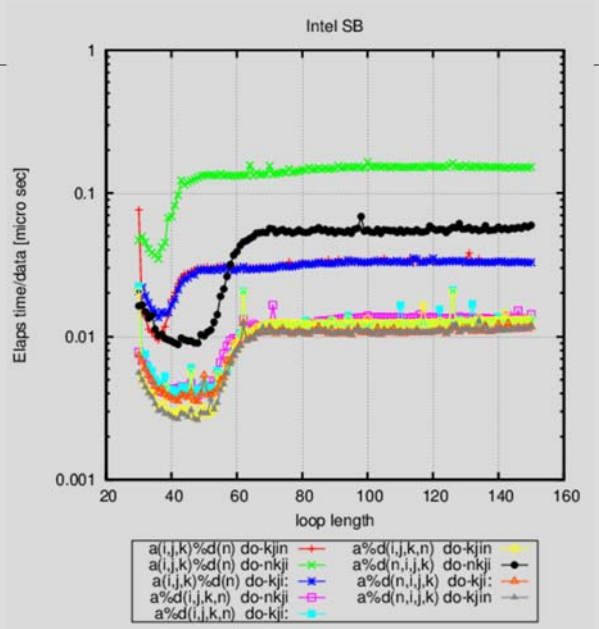
- カーネル化
  - データ：3D構造格子上で定義される物理量
    - 保存量：dq、流束：flux
  - 計算式：保存量と流束の関係（ステンシル計算）
    - dq(i,j,k,n) = dq(i,j,k,n)+S(i,j,k)*(flux(i,j,k,n)-flux(i-1,j,k,n))
    - dq(i,j,k,n) = dq(i,j,k,n)+S(i,j,k)*(flux(i,j,k,n)-flux(i,j-1,k,n))
    - dq(i,j,k,n) = dq(i,j,k,n)+S(i,j,k)*(flux(i,j,k,n)-flux(i,j,k-1,n))
- 性能の確認
  - データ構造（dq,flux）
    - a(i,j,k)%d(n) vs a%d(i,j,k,n), (i,j,k,n) vs (n,i,j,k)
    - i,j,k：ブロックサイズ、n:物理変数（～5）
- ループ構造
  - ループの順番、ベクトル表記

a(i,j,k)%d(n) do-kjin	△
!$omp parallel do do k=1,kmax ; do j=1,jmax ; do i=1,imax do n=1,nmax im = i-idelta(1,dir) dq(i,j,k)%d(n) = dq(i,j,k)%d(n) + S(i,j,k)*(flux(i,j,k)%d(n)-flux(im,jm,km)%d(n)) enddo enddo ; enddo ; enddo !$omp end parallel do	
a(i,j,k)%d(n) do-nkji	×
do n=1,nmax !$omp parallel do do k=1,kmax ; do j=1,jmax ; do i=1,imax im = i-idelta(1,dir) dq(i,j,k)%d(n) = dq(i,j,k)%d(n) + S(i,j,k)*(flux(i,j,k)%d(n)-flux(im,jm,km)%d(n)) enddo ; enddo ; enddo !$omp end parallel do enddo	
a(i,j,k)%d(n) do-kji:	△
!$omp parallel do do k=1,kmax ; do j=1,jmax ; do i=1,imax im = i-idelta(1,dir) dq(i,j,k)%d(:) = dq(i,j,k)%d(:) + S(i,j,k)*(flux(i,j,k)%d(:)-flux(im,jm,km)%d(:)) enddo ; enddo ; enddo !$omp end parallel do	

a%d(i,j,k,n) do-nkji	○
do n=1,nmax !$omp parallel do do k=1,kmax ; do j=1,jmax ; do i=1,imax im = i-idelta(1,dir) dq%d(i,j,k,n) = dq%d(i,j,k,n) + S(i,j,k)*(flux%d(i,j,k,n)-flux%d(im,jm,km,n)) enddo ; enddo ; enddo !$omp end parallel do edndo	
a%d(i,j,k,n) do-kji;	○
!$omp parallel do do k=1,kmax ; do j=1,jmax ; do i=1,imax im = i-idelta(1,dir) dq%d(i,j,k,:) = dq%d(i,j,k,:) + S(i,j,k)*(flux%d(i,j,k,:)-flux%d(im,jm,km,:)) enddo ; enddo ; enddo !$omp end parallel do	
a%d(i,j,k,n) do-kjin	○
!$omp parallel do do k=1,kmax ; do j=1,jmax ; do i=1,imax do n=1,nmax im = i-idelta(1,dir) dq%d(i,j,k,n) = dq%d(i,j,k,n) + S(i,j,k)*(flux%d(i,j,k,n)-flux%d(im,jm,km,n)) edond enddo ; enddo ; enddo !$omp end parallel do	

a%d(n,i,j,k) do-nkji	
do n=1,nmax !$omp parallel do do k=1,kmax ; do j=1,jmax ; do i=1,imax im = i-idelta(1,dir) dq%d(n,i,j,k) = dq%d(n,i,j,k) + S(i,j,k)*(flux%d(n,i,j,k)-flux%d(n,im,jm,km)) enddo ; enddo ; enddo !$omp end parallel do edndo	△
a%d(n,i,j,k) do-kji;	
!$omp parallel do do k=1,kmax ; do j=1,jmax ; do i=1,imax im = i-idelta(1,dir) dq%d(:,i,j,k) = dq%d(:,i,j,k) + S(i,j,k)*(flux%d(:,i,j,k)-flux%d(:,im,jm,km)) enddo ; enddo ; enddo !$omp end parallel do	○
a%d(n,i,j,k) do-kjin	
!$omp parallel do do k=1,kmax ; do j=1,jmax ; do i=1,imax do n=1,nmax im = i-idelta(1,dir) dq%d(n,i,j,k) = dq%d(n,i,j,k) + S(i,j,k)*(flux%d(n,i,j,k)-flux%d(n,im,jm,km)) edond enddo ; enddo ; enddo !$omp end parallel do	○





- a(i,j,k)%d(n) よりも a%d(i,j,k,n) が高速。
- a%d(i,j,k,n) と a%d(n,i,j,k) はほとんど差がない。
- a%d(i,j,k,n) と a%d(i,j,k,:) もほとんど差がない。

まとめ

- UPACS-LをFX10上で（ちょこっと）チューニングを行った。
  - 流束のデータ構造を変更。
  - ソースの書き換えでSIMD化を促進。
- ステンシル計算（更新ベクトルの計算）の基礎的なデータを取得した。
  - データ構造：インデックスの順番
  - ループ構造：ループの順番、ベクトル表記など