ステンシル系プログラムの低メモリバンド幅 CPU 向け 高速化手法の検討

高木亮治†、杉崎由典‡、鈴木清文‡
†宇宙航空研究開発機構、‡富士通株式会社

Study on speed-up algorithms of stencil programs for low memory bandwidth CPUs

by
Ryoji Takaki, Yoshinori Sugisaki and Kiyofumi Suzuki

ABSTRACT

Stencil programs, which are mainly used for numerical simulations of continuum dynamics like fluid mechanics, require relatively high memory bandwidth of CPUs. On the other hand, current supercomputers have relatively low memory bandwidth compared to high computational performance of CPUs. It is called a memory wall problem, namely low B/F (Bytes/s per FLOP/s, FLoating OPeration/s) ratio. This paper makes a study on how to increase computational performance of stencil programs on current CPUs whose memory bandwidth is relatively lower. A practical methodology, which can enhance the computational performance, is proposed according to a study of a basic performance of SORA-MA (JAXA's new supercomputer) by a basic benchmark program. This methodology is applied to an actual stencil program, showing an improvement of computational performance by using characteristics of SORA-MA.

１．はじめに

ペタフロップス級のスーパーコンピュータである次世代スーパーコンピュータ「京」は 2012 年 9 月に共用が開始された。さらにより高速な演算性能、例えばエクサフロップス（エクサはペタの 1,000 倍）級の演算能力や大規模メモリを有するスーパーコンピュータの開発が欧米や我が国で進められている。それらスーパーコンピュータではいくつかの技術的課題が存在するが、特に利用者からみた課題としてはメモリ、時には演算装置の深い階層構造、超並列性、相対的に低いメモリアクセス性能などが挙げられる。特に流体解析など連続体解析で多く使われるステンシル系プログラムでは、高計算効率という観点では所謂高 B/F（Bytes/s per FLOP/s, FLoating OPeration/s）が望ましいが、近年のスーパーコンピュータの B/F は低下する一方である（表 1 を参照）。

表 1　H/W、S/W の B/F と実行効率の最大値

H/W：B/F	4	1	0.5	0.1
S/W：B/F=1	100%	100%	50%	10%
S/W：B/F=4	100%	25%	12.5%	2.5%
システム例	VPP SX	FX1	京 FX100	

更にシステムの複雑化によりハードウェアの性能を十分に活用することが困難になってきている。今後ますます低 B/F 化や複雑化する CPU を用いて最先端の計算を行うためには、ユーザープログラムにこれまで以上に色々な工夫が求められるようになる。そもそもの計算手法として低 B/F なアルゴリズムを採用することも必要であるが、ここではハードウェアの性能を十分に活用することを考える。そのため、これまで十分な実績のあるステンシル系プログラムのアルゴリズムを工夫することで、与えられた B/F において十分な性能[1]（そのハードウェアが出し得る限界性能）を実現することを試みる。その手始めとして、本報告ではステンシル系プログラムで使われるストリーム型演算に注目してメモリアクセス性能の向上を図る。

２．計算機システムの概要

計算機システムとしては平成 28 年度からフル稼働を開始した JSS2 の SORA-MA（FUJITSU Supercomputer PRIMEHPC FX100）を対象とする。フル稼働にともない、これまでよりも高い周波数（2.2GHz）で稼働する CPU を使ったノードが導入されている。SORA-MA の概要を表 2 に示す。

表 2　SORA-MA の概要

	SORA-MA
マシン	PRIMEHPC FX100
CPU	Fujitsu SPARC64™XIfx
周波数	2.2GHz or 1.975GHz
CPU/ノード	1
コア/CPU	32 + (2:アシスタントコア)
コア/CMG	16 + (1:アシスタントコア)
CMG/CPU	2
理論性能	1.126TFLOPS (2.2GHz)
メモリアクセス性能	240GB/s×2

SORA-MA の CPU である SPARC64TMXIfx 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。SPARC64TMXIfx プロセッサで演算性能

[1] メモリバンド幅ネックとなるケースでは、プログラムの原理的な B/F と利用する計算機の B/F よりそのプログラムがその計算機で実現できる理論的な最大性能が求められる。例えばあるプログラムの原理的な B/F が 2 の時に、そのプログラムを B/F が 1 の計算機で実行すると実行効率の理論的な最大値は 50%となる。「十分な計算性能」とは両者の B/F の関係に見合った最大性能のこととする。

This document is provided by JAXA.

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が、相対的に高い B/F を要求するステンシル系プログラムでは如何にメモリアクセス性能を引き出すかが重要となる。

3．STREAM によるメモリアクセス性能

SORA-MA のメモリアクセス性能を基礎的なベンチマークプログラムである STREAM の TRIAD およびこれを実用アプリに近い形に変換したプログラムを用いて評価し、メモリアクセス性能向上のための方策を検討した。

3.1 TRIAD

STREAM は主にメモリアクセス性能を測定するベンチマークプログラムであり、計算機のメモリアクセス性能の実行性能を評価するのに広く使われている。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（2 つの配列とスカラー値を用いた足し算と掛け算）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 はスカラー定数である。

TRIAD は非常に簡単なプログラムであり、計測結果はその計算機のメモリアクセス性能の最大値として利用される。図 1 に 1CMG（16 コア）での測定結果を示す。ループ長が短い場合は、データが L2 キャッシュ（12MB）に載ってしまうので、結果的に高いメモリアクセス性能を示すが、データ量が L2 キャッシュの容量を超えると本来のメモリ性能で律速されるようになる。

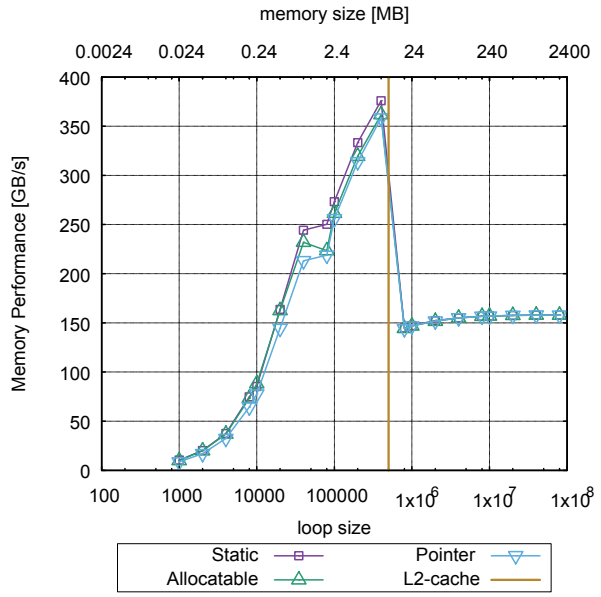

図 1　STREAM(TRIAD)を用いた
SORA-MA のメモリアクセス性能評価

以前の性能評価 [1]では Fortran の配列（静的配列、アロケータブル配列、ポインター配列）の違いでメモリアクセス性能に差が見られたが、その後の調査によりラージページオプション、スレッドのコアへの貼り付け方法の指定、contiguous 属性の指定などを行うことでどの配列でも同じ性能が出るようになった。この結果より 1CMG では 158GB/s がメモリアクセス性能の最大性能と考えられるので、今後はこの値を一つの目標として評価および高速化手法の検討を実施することとする。ちなみに、SORA-MA のメモリアクセス性能は理論ピーク性能が 240GB/s×2 である。これはデータの READ で 240GB/s、WRITE で 240GB/s を意味している。つまり READ と WRITE が対称の場合には理論最大性能が 480GB/s となる。TRIAD では READ と WRITE が非対称であり理論最大性能は 360GB/s となる（READ×2、WRITE×1）。但しこの値は SORA-MA が有する XFILL [2]を使った場合で、XFILL を使わない一般的な場合には理論最大性能は 240GB/s となる。XFILL はデータの書き込み時に発生するキャッシュラインの読み込みを削減する機能である。例えば TRAID の計算をする場合、XFILL がない場合（通常のキャッシュを有する CPU に共通する事象）、プログラム上は READ が 2（b と c）に対して WRITE は 1（a）のはずが、実際の動きとしては READ が 3(a,b,c)、WRITE が 1（a）となる。これは a のキャッシュの整合性を保つためと言われている。一方、SORA-MA のメモリ特性として READ と WRITE が別動作となっている。そのため表 3 で示す様に、READ と WRITE の比率（読み書き比）によってメモリバンド幅の理論性能が大きく変化するが、XFILL によって性能劣化が救済されることがわかる。SORA-MA ではその特異なメモリ特性のため、メモリアクセス性能を向上させるためには XFILL の活用が必要であることがわかる。特に構造格子ソルバーでは READ と WRITE の比率は比較的小さいため、XFILL の活用が効果的と考えられる。

表 3　XFILL と読み書き比による
メモリアクセス性能の理論最大値

XFILL	READ/WRITE (1CPU) [GB/s]						
	1	2	3	4	5	6	7
あり	480	360	320	300	288	280	274
なし	240						

STREAM の計測では XFILL が効いており、理論最大性能が 180GB/s（1CMG あたり）となる。これに対して実際の性能が 158GB/s となるので実行効率は 87.5%となる。この時、演算性能は 13GFLOPS で実行効率は B/F を勘案した理論性能に対して 88.3%となる。

3.2 TRIAD の拡張（マルチブロック、多次元配列）

TRIAD は 1 次元配列を使った簡単な演算ループであり、実際のアプリケーションプログラムのデータ、ループ構造とはかなりの違いがある。ここでは実際のアプリケーションプログラムで最大のメモリアクセス性能を引き出すための方策を検討する。そのため最大メモリアクセス性能を引き出している TRIAD を出発点として、実アプリケーションプログラムのデータ、ループ構造に拡張していくことを考えた。

まず手始めにデータ構造の検討を行った。TRIAD は 1 次元配列であるが、実アプリケーションプログラムでは多次元配列を用いることが多い。そのため構造格子を想定して多次元配列版（ここでは 3 次元配列）として以下のものを考えた。

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

This document is provided by JAXA.

ここで i,j,k のループ長（格子サイズ）は全て同じで N とした。これを TRIAD3D と呼ぶ。更に構造格子法では複雑形状への対応を考えた場合マルチブロック法など複数ブロックに分割した手法が一般的であるため、複数ブロックを想定したデータ構造として以下のようなものを考えた。

```
type blkDataType
   real(8), dimension(:,:,:), allocatable :: a,b,c
end type blkDataType
type(blkDataType), dimension(:), allocatabe :: blk
```

ここで、配列としてはアロケータブル配列を使う事とした。また、配列 a,b,c は

```
blk(:)%a(1-ovlp:N+ovlp,1-ovlp:N+ovlp,1-ovlp:N+ovlp), …
```

のように袖（ovlp）を持つようにした。このように配列を 3 次元化し構造体を用いて複数ブロックにしたものを MB-TRIAD3D と呼ぶことにする。MB-TRIAD3D のデータ構造は UPACS などのマルチブロック構造格子法で一般的なデータ構造である。MB-TRIAD3D では、各ブロックの格子点数（N^3）×ブロック数=総格子点数となるが、ループ長の影響を評価するために総格子点数をほぼ一定にして N（ブロックの辺の格子点数）を変化させて性能計測を行った。実際の計測では総格子点数は 2,700 万点とした。そのため最小ブロック（N=10）の場合、10x10x10 x 27,000 ブロック、最大ブロック（N=300）の場合、300x300x300 x 1 ブロックとなる。ブロック数=Int（総格子点数/N^3）としたのでブロックのサイズを変えた場合に、総格子点数はブロックの大きさによっては若干変動していることになる。

図 2 に測定結果を示す。

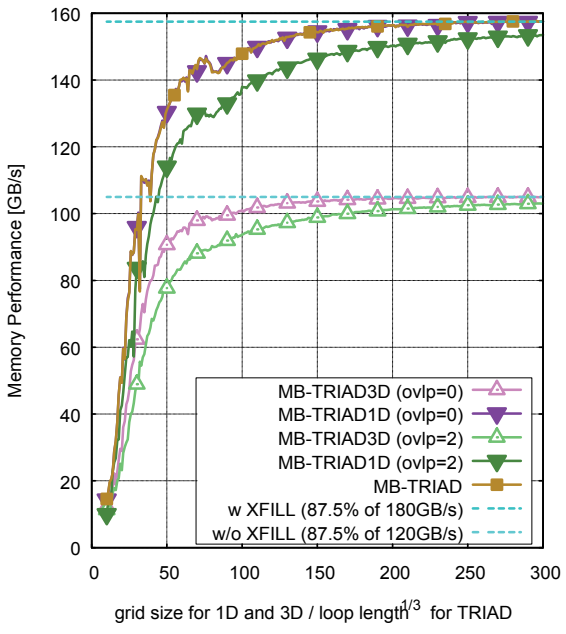

図 2　各種 TRIAD の測定結果

図の凡例で「MB-TRIAD」は通常の TRIAD を複数ブロック化したもので、プログラムの概略は以下となる。

メインプログラム：

```
type blkDataType
real(8), dimension(:), allocatable :: a,b,c
end type blkDataType

type(blkDataType), dimension(:), allocatabe :: blk

do nb=1,NB
   call kernel(blk(nb)%a, blk(nb)%b, blk(nb)%c, …)
enddo
```

サブルーチンプログラム（kernel）：

```
subroutine kernel(a,b,c,…)
real(8), dimension(:) :: a,b,c

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

end subroutine kernel
```

また MB-TRIAD3D のプログラムの概略は以下となる。MB-TRIAD との違いは配列が 1 次元配列か 3 次元配列かの違いである。

メインプログラム：

```
type blkDataType
   real(8), dimension(:,:,:), allocatable :: a,b,c
end type blkDataType

type(blkDataType), dimension(:), allocatabe :: blk

do nb=1,NB
   call kernel(blk(nb)%a, blk(nb)%b, blk(nb)%c, …)
enddo
```

サブルーチンプログラム（kernel）：

```
subroutine kernel(a,b,c,…)
real(8), dimension(:,:,:) :: a,b,c

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

end subroutine kernel
```

ovlp は袖の長さ（ovlp=0：袖なし、ovlp=2：袖の長さが 2、a(-1:n+2,-1:n+2,-1:n+2), …）を示す。ここで引数を構造体ではなく配列としているのは現状の富士通コンパイラの問題であり、配列渡しにしないと最適化が促進されないためである。

通常の TRIAD では 1 次元配列を使っているため、図 1 で示すようにループ長が短い場合はデータ全てが L2 キャッシュに乗ってしまい、L2 キャッシュの性能を測定していることになるが、MB-TRIAD の場合 L2 キャッシュには乗り切らないので、メモリアクセス性能が計測できている。その結果、L2 キャッシュの影響を排除したループ長の効果が観測でき、短いループではメモリアクセス性能が低下することがわかる。ループ長が短い場合にメモリアクセス性能が低下するのは、パイプラインなどの最適化のオーバーヘッドやメモリアクセスのレイテンシが隠ぺいできない等の理由によると思われる。

MB-TRIAD のメモリアクセス性能はループ長が短い領域では低いが、ループ長の増加とともに単調に増加し、最終的には 160GB/s 弱となる。この値は通常の TRIAD のメモリアクセス性能の最大値と同じである。この結果により、構

This document is provided by JAXA.

造体を用いて複数ブロック化した場合でも最大メモリアクセス性能を発揮できることが確認できた。

次に MB-TRIAD と MB-TRIAD3D、特に ovlp=0 での結果を比較する。両者の違いは、配列がそれぞれ 1 次元配列（a(:),b(:),c(:)）の 1 重ループ（do i=1,N*N*N）か 3 次元配列（a(:,:,:),b(:,:,:),c(:,:,:)）の 3 重ループ（do k=1,N; do j=1,N; do i=1,N）かである。ちなみに N に対して MB-TRIAD では N×N×N をループ長とし、どちらも演算量は同じにしてある。この比較によると 3 次元配列を用いた MB-TRIAD3D は 1 次元配列を用いた MB-TRIAD よりもかなり性能が低いことがわかる。MB-TRIAD3D は最大でも 100GB/s 程度しか出せていない。この値は XFILL を使わない場合の実行性能にほぼ等しい。実際にコンパイルリストを見ると、XFILL が効いていないことが確認できる。また ovlp=0 と 2 を比較すると ovlp=2 の性能が低い。実際の演算には袖部分は含まれていないので演算量による差ではなく、メモリ、例えばキャッシュラインによる不要データのアクセス等が原因と考えられる。

以上の性能評価結果をまとめると

① 3 次元配列を用いた場合メモリアクセス性能は最大で 100GB/s 強となり、1 次元配列の場合の 160GB/s 弱よりもかなり低下する、コンパイルリストおよび計測結果から XFILL が効いていないと考えられる、

② キャッシュが溢れた状態ではループ長が短いと性能が低下する。メモリアクセス性能を出すためにはある程度以上のループ長が必要、

ということがわかった。MB-TRIAD3D の性能劣化の大きな原因は XFILL と考えられる。XFILL に関して調査を行った結果 MB-TRIAD3D で XFILL が効かないのは、ループ長が不足していることが主な理由であることがわかった。ループ長が 256（デフォルトの設定。コンパイルオプションなどで変更可能）以上ないと XFILL が効かないのである。XFILL を効かすために contiguous 属性を指定し、かつブロックサイズを大きくした条件で計測した結果を図 3 に示す。

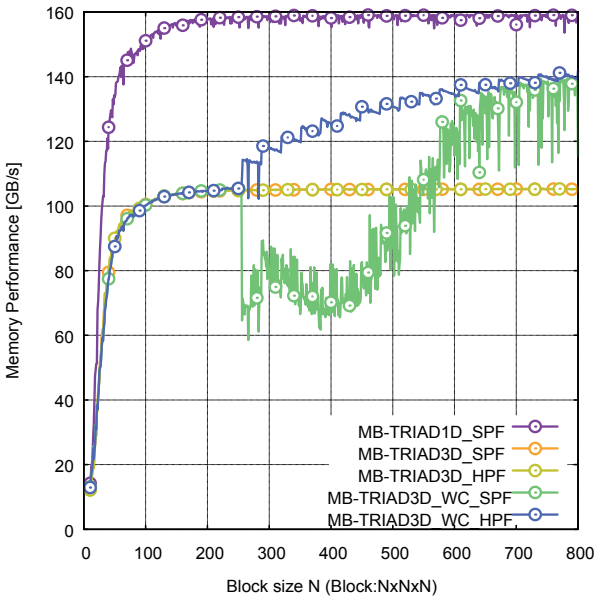

図 3 XFILL の活用による性能向上

最内ループ長が 256 を超えた辺りから性能が向上する様子が観察できる。この図の凡例で_WC は contiguous 属性を指定したことを意味している。また、_SPF と_HPF はそれぞれソフトウェアプリフェッチ（S/W PF）、ハードウェアプリフェッチ（H/W PF）を適用した結果を示している。S/W PF と H/W PF で挙動が異なり、特に S/W PF では XFILL 適用後に性能が一時悪化しているが、これはコンパイラの問題で、今後改善される予定である。

図 3 で示す様に、MB-TRIAD3D の最内ループを 256 以上にすることで XFILL が効き性能が向上することがわかった。そこで、3 次元配列（対応して 3 重ループ）のインデックス（i,j,k）のうち j と k を比較的小さな値に固定して i を大きく（最内ループを長く）することを試行した。結果を図 4 に示す。ループ長の増大とともに性能が向上しスレッド数が不足している（N,1,1）と(N,2,2)の 2 ケース以外は最終的に 160GB/s 弱の性能を示すことがわかった。XFILL の閾値である 256 の辺りでは XFILL 無しの最大値に起因する変曲点が見られる。この結果より多次元配列の場合でも、最内ループ長が長ければ最大のメモリ性能が達成できることを示している。しかしながら、最大メモリ性能を達成するのに必要となるループ長は 10,000 から 100,000 と非常に長いループが必要であることも同時に示している。多次元配列の 1 次元目を 10,000～100,000 のように非常に大きなサイズにすることは非現実的であり、何某かの現実的な対応が必要である。

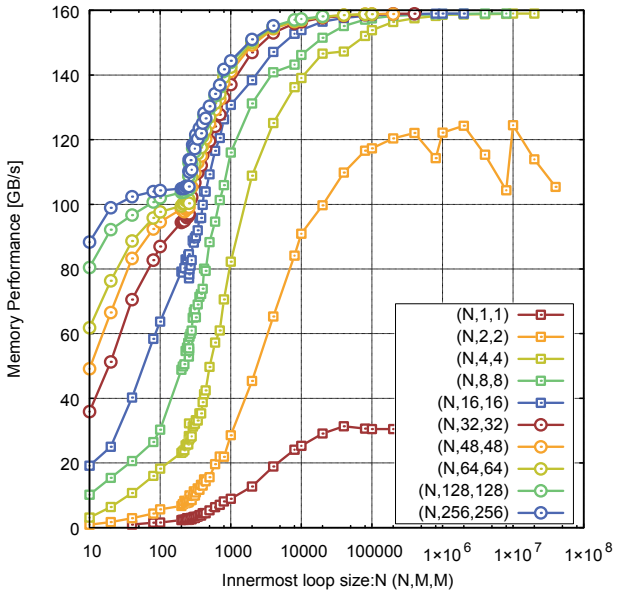

図 4 ループ長の増加による性能向上

以上の結果から、メモリアクセスの最大性能を引き出すためには長いループが必要であるということが想像できる。そのため、MB-TRIAD と同じような形にするため、以下のようにサブルーチンの変数を渡すときに 3 次元配列を 1 次元配列として渡し、処理ループを 1 重ループにすることを考えた。ここでは 1 次元化と呼ぶ。

メインプログラム：

```
type blkDataType
   real(8), dimension(:,:,:), allocatable :: a,b,c
end type blkDataType

type(blkDataType), dimension(:), allocatabe :: blk

do nb=1,NB
  call kernel(blk(nb)%a, blk(nb)%b, blk(nb)%c, …)
enddo
```

This document is provided by JAXA.

サブルーチンプログラム（kernel）：

```
subroutine kernel(a,b,c,…)
real(8), dimension(*) :: a,b,c

do l=lstart,lend
   a(l) = b(l) + S * c(l)
enddo

end subroutine kernel
```

このやり方の結果が図 2 で MB-TRIAD1D として示されている。サブルーチンを通じて渡される 3 次元配列を 1 次元配列として受け取ることで、MB-TRIAD と同じメモリアクセス性能を達成できていることがわかる。ただ、袖がある場合（ovlp=2）は、袖部分のデータアクセスと演算処理を行うため、その分メモリアクセス性能が劣化するが、N が大きくなるにつれてその影響が小さくなることもわかる。袖がない場合は 1 次元版と同じメモリアクセス性能（現状で出し得る最大性能：160GB/s 弱）を達成することが可能であり、袖による劣化を差し引いても 3 次元配列をそのまま使った場合に比べてメモリアクセス性能は向上している。

4．ステンシル系プログラム（UPACS-Lite）

基礎的なベンチマークプログラム STREAM TRIAD を使って、現状で出し得る最大性能の把握と、実アプリケーションプログラムを模擬した改良版で最大性能を出し得る手法（1 次元化）を提案した。次のステップとして、ほぼ実アプリケーションプログラムである UPACS-Lite [1]から切り出した各要素（右辺の流束計算）に本提案手法を適用し評価を行った。UPACS-Lite における各要素としては① muscl（セル面での物理量の補間）、② cflux（セル面の非粘性流束の計算）、③ cfacev（セル面での物理量の微分の計算）、④ vflux（セル面の粘性流束の計算）を対象とした。ここで① muscl と③ cfacev はステンシル型計算（隣接するデータを使った計算）であり、② cflux と④ vflux はストリーム型計算（該当するインデックスのみ使った計算）である。これらの要素の計算に対して 1 次元化を適用した結果を図 5 から図 8 に示す。これらの図では計算時間で比較している。

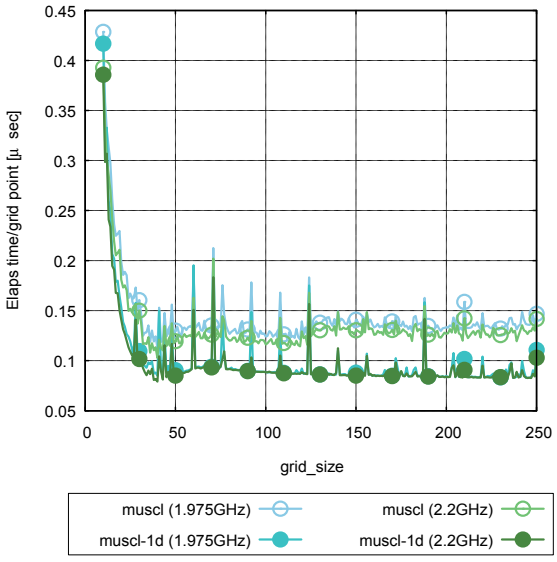

図 5　① muscl

それぞれの図の凡例で muscl, cflux, cfacev, vflux がオリジナルの 3 次元配列の結果で muscl-1d, cflux-1d, cfacev-1d, vflux-1d が提案手法による 1 次元化した結果である。全てのケースで袖の長さは 2 としている。そのため、1 次元化したものは袖の分だけ、演算数およびメモリアクセス数がオリジナルに対して増大している。これらの図には CPU の周波数（1.975GHz と 2.2GHz）による違いも示しているが、ほとんどメモリアクセス性能ネックになっており、特に 1 次元化のケースでは周波数の違いによる性能差は小さい。

② cflux を除くいずれの要素においても 1 次元化により大幅な性能向上（計算時間の短縮）が見られる。② cflux もブロックサイズが小さい領域では 1 次元化の改善が見られるが、ブロックサイズが大きい領域では 1 次元化による性能向上はほとんど見られない。もっとも、袖の部分のオーバーヘッドを考えれば 1 次元化によって性能向上が見られるが、実質的な性能向上という観点では効果がないと判断される。メモリアクセス性能値が XFILL なしの場合の予測値に近いことから 1 次元化による性能向上が見られないのは XFILL が効いていないのが原因と推察している。

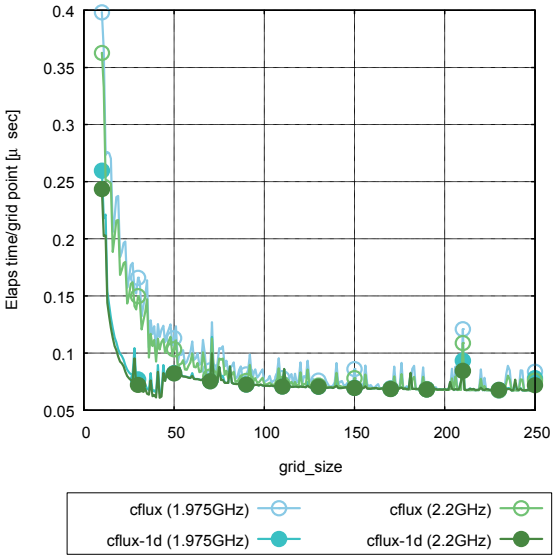

図 6　② cflux

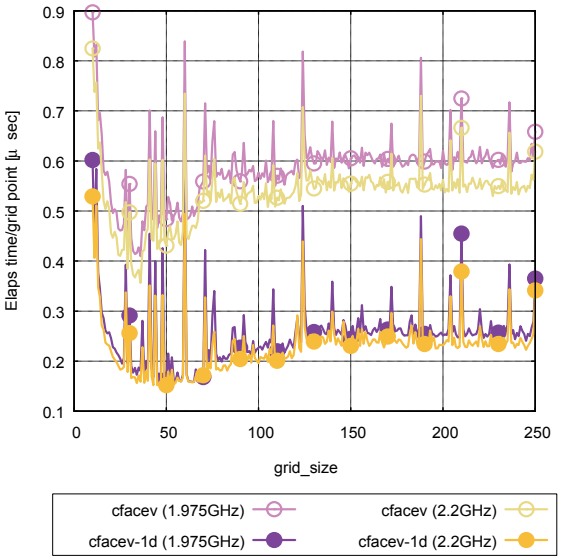

図 7　③ cfacev

This document is provided by JAXA.

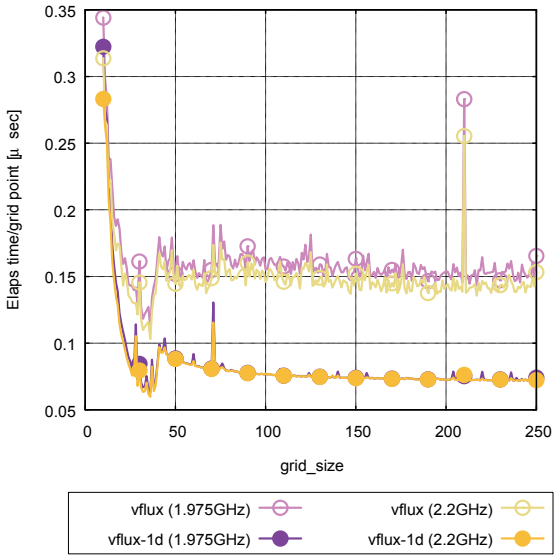

図 8 ④ vflux

プログラムの実装を工夫することでプログラムにおける B/F を下げることを試みた。通常これらの要素の計算は順番にループを分けて以下の様に実施する。

```
do k; do j; do i
  [muscl op.]
enddo; enddo; enddo

do k; do j; do i
  [cflux op.]
enddo; enddo; enddo

do k; do j; do i
  [cfacev op.]
enddo; enddo; enddo

do k; do j; do i
  [vflux op.]
enddo; enddo; enddo
```

このやり方では muscl-cflux 間、cfacev-vflux 間においてセル面での物理量を保持する配列をやり取りしている。これらのループを融合することで、この配列のアクセスを削除し、その分 B/F を小さくすることができる（表 4 のループ融合 1）。

表 4 ループ融合

ループ融合 1	ループ融合 2
do k do j do i [muscl op.] [cflux op.] [cfacev op.] [vflux op.] enddo enddo enddo	do k; do j do i [muscl op.] [cflux op.] enddo do i [cfacev op.] [vflux op.] enddo enddo; enddo

しかしながら、この方法ではループボディーが巨大になり、レジスタスピルが発生し、ソフトウェアパイプライ、SIMD 他の最適化が適用できなくなり、結果的に性能が悪化する。そのためループボディーを縮小し、かつキャッシュの再利用により B/F を下げることを考えた（表 4 のループ融合 2）。このループは j と k を融合して、i ループで分割している。最内の i ループは長くないので、各要素の計算ではキャッシュが有効に利用できる。

各要素、および全体に対して 1 次元化やループ融合 2 を適用して測定を行った結果を図 9 に示す。ここで all は通常の実装で① muscl＋② cflux＋③ cfacev＋④ vflux を順番に計算、all-1d はそれぞれの要素に 1 次元化を適用したもの、comb はループ融合 2 のやり方でループを融合したもの、comb-1d は comb を 1 次元化したものである。図 9 より、それぞれの手法の効果を比較すると、従来実装（all）に対して、ループ融合、1 次元化を適用することでそれぞれ速くなっている。ループ融合と 1 次元化はほぼ同程度の高速化となっている。ループ融合と 1 次元化の組み合わせが最も速く、従来実装に比べて 3 倍程度の高速化が実現できた。

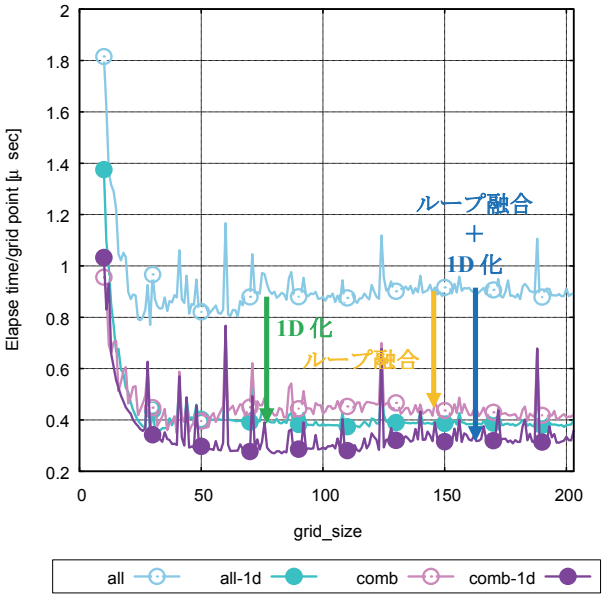

図 9 1 次元化、ループ融合による性能向上

５．おわりに

ステンシル系プログラムの高速化手法の検討を行った。JSS2 SORA-MA を対象に基礎的なベンチマークプログラム STRAM を使ってメモリアクセス性能の実行最大性能を把握し、その実行最大性能を出すための手法を提案した。実アプリケーションプログラムの主な要素で検証しその有効性を示した。本論文で用いた手法は、アプリケーションのチューニングを行う従来の方法とは異なるアプローチであるが、その有効性が確認された。高速化結果は SORA-MA に特有のものになっているが、本手法は他マシンでの高速化チューニングにも適用可能であると考える。今後は詳細な分析を行うと同時に他アーキテクチャへの適用を行う。

参照文献

[1] 高木亮治, “JAXA 新スーパーコンピュータ（SORA-MA）の性能評価,” *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論文集（JAXA-SP-15-013）*, pp. 53-58, 2016.

[2] 富士通株式会社, *Fortran 使用手引書（PRIMEHPC FX100 用） FUJITSU Software Technical Computing Suite V2.0,* 2015.

This document is provided by JAXA.