Vol.2015-HPC-152 No.10
2015/12/17

テラフロップス級メニーコアアーキテクチャにおけるステンシル計算の最適化手法の開発

伊奈拓也[†1]　朝比祐一[†1]　井戸村泰宏[†1]

概要：核融合プラズマ乱流シミュレーションには多大な計算資源が必要となる．特に，国際熱核融合実験炉 ITER のシミュレーションを実行するためには将来のエクサスケールマシンが必要不可欠である．エクサスケールマシンのアーキテクチャは未定であるが，現存するメニーコアアーキテクチャがベースになると考えられる．このため，現存するメニーコアアーキテクチャに対する最適化手法は，エクサスケールマシンにおいても有効であると考えられる．本研究ではテラフロップス級の演算性能を持つメニーコアアーキテクチャである Xeon Phi 5110P, Tesla K20X, SPARC64 XIfx に対するステンシル計算の最適化手法の開発を行い，その効果を検証し，更なる最適化に向けた課題を議論する．

キーワード：GPGPU，Xeon Phi，並列計算

Development of optimization of stencil calculation on Tera-flops many-core architecture.

TAKUYA INA[†1]　ASAHI YUICHI[†1]　YASUHIRO IDOMURA[†1]

Abstract: Fusion plasma turbulence simulation requires significant computational resources. In particular, in order to simulate the International Thermonuclear Experimental Reactor ITER, future Exa-scale machines are essential. Although Exa-scale architectures are still uncertain, they are expected to be based on the existing many core architectures. Therefore, optimization techniques for the existing many core architectures will be effective also on future Exa-scale machines. In this work, we develop optimization techniques of stencil calculations for Tera Flops many core architechtures such as Xeon Phi 5110P, Tesla K20X and SPARC64 XIfx, validate them, and discuss issues towards further performance improvements.

Keywords: GPGPU，Xeon Phi，Parallel computating

1. はじめに

核融合炉内のプラズマ乱流シミュレーションを行うには高い演算性能を持つスーパーコンピュータが必要となる．現在，建設が進められている国際熱核融合実験炉 ITER[1]のシミュレーションを行うためには，現在のペタフロップスの演算性能を持つペタスケールスーパーコンピュータでは計算資源が不足しており，将来実現されるエクサスケールスーパーコンピュータの利用が必要不可欠である．エクサスケールスーパーコンピュータのアーキテクチャがどのようになるか現時点では不明であるが，現在のスーパーコンピュータで使用されているテラフロップス級メニーコアアーキテクチャをベースに開発されると予想される．このため，現在のメニーコアアーキテクチャに対するステンシル計算の最適化手法を開発することにより，将来のエクサスケールアーキテクチャにおける最適化技術の問題点を抽出することが可能であると考えられる．本研究では，現在のテラフロップス級メニーコアアーキテクチャに対するステンシル計算の最適化手法を開発し，その有効性と課題を議論する．

2. GT5D

日本原子力研究開発機構では核融合炉のプラズマ乱流シミュレーションコード GT5D[2][3]が開発されている．GT5D はプラズマ乱流とプラズマ粒子分布関数の時間発展を移流拡散方程式とポアソン方程式から構成される第一原理モデルに基づいて計算するプログラムである．GT5D において最も計算時間がかかる部分は移流拡散方程式の移流項の計算であることから，本研究ではこの移流項に対して最適化手法の開発を行う．移流項$\mathcal{G}$の式を示す．

$$\mathcal{G}[f] = -\mathbf{U}_1 \cdot \frac{\partial f}{\partial \mathbf{R}} - U_2 \frac{\partial f}{\partial v_{||}} \quad (1)$$

移流項$\mathcal{G}$は 3 次元の位置微分と 1 次元の速度微分の項で表される．粒子分布関数 f の時間積分に半陰的ルンゲ・クッタ方を適用するため疎行列の連立一次方程式を解く必要がある．連立一次方程式の解法にはクロリフ部分空間法に基づく一般化共役残差法を使用するため，粒子分布関数 f を 4 次元格子上で近似し，4 次精度中心差分に基づく 17 点のステンシル計算を図 1 のように実装している．

```
!$OMP DO
!$OMP& SCHEDULE(static,1)
 do i = 1,nx
 do j = 1,ny
    compute (vxyv ~ vxl2)
 do k = 1,nz
 do l = 1,nv
    df(l,k,j,i)= &
           vxyv(l)*f(l,k,j,i) &
        + vvr(l) *f(l+1,k,j,i) - vvl(l) *f(l-1,k,j,i) &
        - vvr2(l)*f(l+2,k,j,i) + vvl2(l)*f(l-2,k,j,i) &
        + vzl(l) *(f(l,k+1,j,i) - f(l,k-1,j,i) ) &
        - vzr2(l)*(f(l,k+2,j,i) - f(l,k-2,j,i) ) &
        + vyr(l) *f(l,k,j+1,i) - vyl(l) *f(l,k,j-1,i) &
        - vyr2(l)*f(l,k,j+2,i) + vyl2(l)*f(l,k,j-2,i) &
        + vxr(l) *f(l,k,j,i+1) - vxl(l) *f(l,k,j,i-1) &
        - vxr2(l)*f(l,k,j,i+2) + vxl2(l)*f(l,k,j,i-2)
    dfc=dfc+df(l,k,j,i)*fc(l,k,j,i)
 enddo
 enddo
 enddo
 enddo
```

図 1　4 次元格子の CPU 版ステンシル計算コード

3. 計算環境

現在のスーパーコンピュータで一般に利用されているテラフロップス級メニーコアアーキテクチャはアクセラレータである Xeon Phi 5110P (MIC)[4]，Tesla K20X (K20X)[5]，汎用 CPU である SPARC64XIfx (FX100)[6]などがある．最適化手法の開発を行う計算機環境を表 1 に示す．本研究では MIC, K20X，FX100 を対象に最適化手法の開発を行う。

表 1　計算機環境

プロセッサ	Xeon Phi 5110P (MIC)	Tesla K20X (K20X)	SPARC64 XIfx (FX100)
コア数	60	896	32+2
メモリ[GB]	8	6	32
キャッシュ[MB]	30	1.5	24
理論演算性能(倍精度) [Gflops]	1056	1310	1011
理論バンド幅[GB/s]	320	250	480
SIMD 幅[bit]	512	-	256
コンパイラ	intel compiler 13.1.3	pgfortran 15.1 nvcc 6.5	Fujitsu compiler 2.0.0
コンパイラオプション	-O3 -mmic -align array 64byte -no-prec-div -openmp -r8 -shared-intel -mcmodel= large	-mp -fast -Mcuda r8	-Kfast, openmp, mfunc=2 -CcdRR8

3.1 Xeon Phi 5110P

Xeon Phi 5110P は 60 個の演算コアを持つインオーダ実行のプロセッサである．SIMD 幅は 512bit である．一度の演算命令で 8 個の倍精度浮動小数点を処理することが可能である．各コアはローカルな L2 キャッシュを持つ．L2 キャッシュは完全なコヒーレンシを持ち，他のコアへデータ供給が可能である．複数のコアで使用される共有データは共有データを使用する各コアのローカルキャッシュにコピーされる．全てのコアが異なるデータを使用して処理を行う場合の L2 キャッシュ容量は 30MB であるが，全てのコアが同一データを使用して処理を行う場合は各コアが同一のデータをローカルキャッシュにコピーするため L2 キャッシュ容量は 512KB になる．

MIC 上では汎用 CPU で動作するプログラムをコード修正無しで使用可能である. 最適化手法の開発は MIC 上のみで動作させるネイティブモードで行う．最新版のコンパイラ intel compiler 15.0.2 で作成したバイナリファイルは旧バージョンのコンパイラ intel compiler 13.1.3 で作成したバイナリファイルよりも性能が落ちることを確認しているため旧バージョンのコンパイラを使用する．

3.2 Tesla K20X

Tesla K20X は Kepler アーキテクチャの GPU である. SMX と呼ばれるプロセッサを 14 個持ち, SMX は 64 個の倍精度演算ユニットを持つ．各 SMX は 64KB の SRAM を持ち，L1 キャッシュとシェアードメモリに分割して使用する．SMX 間でデータ統合の役割を持つ L2 キャッシュは 1.5MB 搭載されている．

GPU 上で計算を行うためには GPU 向けのプログラムを作成する必要がある．プログラミング環境として C 言語向けの CUDA と Fortran 用の CUDA Fortran が提供されている．CUDA の実行モデルはグリッド，ブロック，スレッドの階層構造を持つ．グリッドは複数のブロックをまとめたものである. ブロックは複数のスレッドをまとめたものである．1 つのブロックは 1 つの SMX で計算される．スレッドは warp と呼ばれる 32 スレッド単位で処理される．

3.3 SPARC64 XIfx

SPARC64 XIfx は 32 個の演算コアと 2 個のアシスタントコアを持つ．演算コアの SIMD 幅は 256bit である．一度の SIMD 演算で 4 個の倍精度浮動小数点演算を処理することが可能である．アシスタントコアはシステムの割り込み処理と通信処理を担当する．

32+2 個のコアは 16+1 コア単位のコアメモリグループ(以降，CMG と呼ぶ)に分けられる．1 つの CMG は CMG 内で共有される 12MB の L2 キャッシュ，メモリコントローラを持つ．ここで，コアがローカル CMG に接続されているメモリにアクセスする時間とリモート CMG に接続されているメモリにアクセス時間は異なる．

3.4 各プロセッサの実行性能評価

プロセッサの性能とプログラムの実装からアプリケーシ

ョンの実行演算性能を評価する方法としてルーフラインモデルが提案されている[7]．プログラムの演算数 f（Flops），メモリ参照量 b（Byte）のアプリケーションを演算性能 F（Flops），メモリバンド幅 B（Byte/s）のプロセッサで処理する場合の実行演算性能 S（Flops）がルーフラインモデルでは

$$S = \frac{f}{\frac{f}{F} + \frac{b}{B}} \quad (2)$$

と与えられる．このモデルを用いて GT5D のステンシル計算の処理性能を評価する．最適化手法の開発を行う問題サイズは nx=32,ny=32,nz=16,nv=128 とし，各方向とも上端と下端にそれぞれ袖領域を 2 要素持つ．図 1 のコードから，MIC と FX100 向けコードの演算数は 33 となる．一方，図 2 に示す GPU 版コードの演算数は 64 である．メモリ参照数は問題サイズとキャッシュサイズによって異なる．l 方向差分では f（l-2,k,j,i）～f（l+2,k,j,i）を参照するので 40B のアドレス範囲を参照する．k 方向差分では f（l,k-2,j,i）～f（l,k+2,j,i）を参照する．l 方向の問題サイズから（128+2+2）*5*8=5.28KB のアドレス範囲を参照する．j 方向差分では f（l,k,j-2,i）～f（l,k,j+2,i）を参照する．lk 方向の問題サイズから（128+2+2）*（16+2+2）*5*8=105.68KB のアドレス範囲を参照する．i 方向差分では f（l,k,j,i-2）～f（l,k,j,i+2）を参照する．l,k,j 方向の問題サイズから（128+2+2）*（16+2+2）*（32+2+2）*5*8=3.80MB のアドレス範囲を参照する．処理単位当たりのキャッシュサイズが各差分方向で参照するアドレス範囲よりも大きい場合は読み込んだ f（l,k,j,i）をキャッシュから再利用すると考えメモリ参照数に含まない．MIC は 60 個の演算コアと 30MB のキャッシュを持ち，1 コアに 4 スレッド生成されるため処理単位当たり 125KB．K20X は 14 個の SMX と 1.5MB のキャッシュを持つため処理単位当たり 107.14KB．FX100 は 32 個の演算コアと 24MB キャッシュを持つため処理単位当たり 750KB．各プロセッサの処理単位当たりのキャッシュサイズが 105.68KB よりも大きく，3.80KB より小さいため，i 方向差分で参照する配列をメモリアクセス，lkj 方向の差分で参照する配列をキャッシュ読み込みとして扱う．それに加えて，GT5D は Open MP によってループ変数 i のループがサイクリック分割，もしくは，ダイナミック分割で並列化されており隣接スレッドが読み込んだ f（l,k,j,i）は i 方向差分で参照する配列と一致し，共有キャッシュ上のデータを再利用できるため，MIC と FX100 については i 方向差分で参照する配列に対してもキャッシュ読み込みとして扱う．したがって，メモリ参照数は MIC と FX100 が 32B，K20X が 64B である．ルーフラインモデルで評価した実行演算性能を表 2 に示す．表 2 の実行演算性能が最適化手法の開発で目標とする演算性能である．

```
tidl     = threadidx%x
tidk     = threadidx%y
k          = (blockidx%y-1)*blockDim%y + tidk
l          = (blockidx%x-1)*blockDim%x + tidl
tid_lk = tidl + (tidk-1) * blockDim%x
do i = 1, nx
do j = 1, ny
        if(tidk == 1) then
           vxl(tidl) = (rg(i)*vx(l,j,i)+rg(i-1)*vx(l,j,i-1))
           vxr(tidl) = (rg(i)*vx(l,j,i)+rg(i+1)*vx(l,j,i+1))
           vyl(tidl) = (vy(l,j,i)+vy(l,j-1,i))
           vyr(tidl) = (vy(l,j,i)+vy(l,j+1,i))
           vvl(tidl) = (vv(l,j,i)+vv(l-1,j,i))
           vvr(tidl) = (vv(l,j,i)+vv(l+1,j,i))
           vxl2(tidl)= (rg(i)*vx(l,j,i)+rg(i-2)*vx(l,j,i-2))
           vxr2(tidl)= (rg(i)*vx(l,j,i)+rg(i+2)*vx(l,j,i+2))
           vyl2(tidl)= (vy(l,j,i)+vy(l,j-2,i))
           vyr2(tidl)= (vy(l,j,i)+vy(l,j+2,i))
           vvl2(tidl)= (vv(l,j,i)+vv(l-2,j,i))
           vvr2(tidl)= (vv(l,j,i)+vv(l+2,j,i))
           vzl(tidl) = 2*vz(l,j,i)*c_if5d
           vzr(tidl) = 2*vz(l,j,i)*c_if5d
           vzl2(tidl)= 2*vz(l,j,i)*c_if5d
           vzr2(tidl)= 2*vz(l,j,i)*c_if5d
           b0s(tidl) = c_b0/bbs(i,j,l)
        end if
        call syncthreads()

        fxl   = vxl(tidl)     * (f(l,k,j,i) + f(l,k,j,i-1) )
        fxl2 = vxl2(tidl)    * (f(l,k,j,i) + f(l,k,j,i-2) )
        fyl   = vyl(tidl)     * (f(l,k,j,i) + f(l,k,j-1,i) )
        fyl2 = vyl2(tidl)    * (f(l,k,j,i) + f(l,k,j-1,i) )
        fzl   = vzl(tidl)     * (f(l,k,j,i) + f(l,k-1,j,i) )
        fzl2 = vzl2(tidl)    * (f(l,k,j,i) + f(l,k-2,j,i) )
        fvl   = vvl(tidl)     * (f(l,k,j,i) + f(l-1,k,j,i) )
        fvl2 = vvl2(tidl)    * (f(l,k,j,i) + f(l-2,k,j.i) )

        flx   = (vxr(tidl)     * (f(l,k,j,i) + f(l,k,j,i+1)) - fxl    )
        flx2 = (vxr2(tidl)    * (f(l,k,j,i) + f(l,k,j,i+2)) - fxl2 )
        fly   = (vyr(tidl)     * (f(l,k,j,i) + f(l,k,j+1,i)) - fyl    )
        fly2 = (vyr2(tidl)    * (f(l,k,j,i) + f(l,k,j+2,i)) - fyl2 )
        flz   = (vzr(tidl)     * (f(l,k,j,i) + f(l,k+1,j,i)) - fzl    )
        flz2 = (vzr2(tidl)    * (f(l,k,j,i) + f(l,k+2,j,i)) - fzl2 )
        flv   = (vvr(tidl)     * (f(l,k,j,i) + f(l+1,k,j,i)) - fvl    )
        flv2 = (vvr2(tidl)    * (f(l,k,j,i) + f(l+2,k,j,i)) - fvl2 )

        flow =   (     flx   * dxi     &
                     + fly   * dyi     &
                     + flz   * dzi     &
                     + flv   * dvi     &
                     ) * (-cc1)        &
                +   ( flx2 * dx2i   &
                     + fly2 * dy2i   &
                     + flz2 * dz2i   &
                     + flv2 * dv2i   &
                     ) * cc2
        df(l,k,j,i) = f(l,k,j,i) - adt*ca*flow*b0s(tidl)
        dfc(tid_lk) = df(l,k,j,i)*fc(l,k,j,i) + dfc(tid_lk)
    end do
    end do
```

図 2　GPU 版のステンシル計算コード

表 2　ルーフラインモデルで評価した実行性能

	MIC	K20X	FX100
演算数	33	64	33
メモリ参照量[B]	32	64	32
実行性能[Gflops]	251.43	209.94	332.30

4. 最適化手法

GT5G は MPI によるプロセス並列化と OpenMP によるスレッド並列化がされているが，本研究における最適化手法

の開発は単体性能（1 プロセス+複数スレッド）を対象に行う. MIC と FX100 では OpenMP を使用し, GPU では CUDA Fortran を使用する.

4.1 Xeon Phi 5110P 向けの最適化手法

(1) 多重ループの一重化

MIC は 60 個の演算コアを持つため，コアを有効活用するためには多数のスレッドを作成する必要がある．OpenMP 構文の collapse 指示行によって多重ループの一重化を行うことで多数のスレッドを作成させてコアを使い切ることで性能を向上させる.

```
!$OMP DO
!$OMP&
SCHEDULE(static,1)
  do i = 1,nx
  do j = 1,ny
     compute (vxyv ~ vxr2)
  do k = 1,nz
  do l = 1,nv
!Finite difference computation
  enddo
  enddo
  enddo
  enddo
```

```
!$OMP DO
!$OMP&
SCHEDULE(static,1)
!$OMP& COLLAPSE(2)
  do i = 1,nx
  do j = 1,ny
     compute (vxyv ~ vxr2)
  do k = 1,nz
  do l = 1,nv
!Finite difference computation
  enddo
  enddo
  enddo
  enddo
```

図 3　オリジナルのコード(左)およびループを一重化したコード(右)

(2) 動的スケジューリング

MIC は発行された命令を順番通りに処理するインオーダ実行であるため，メモリアクセスが発生した場合にデータが揃うまでのレイテンシが発生する．レイテンシを隠蔽させるためには一つのコアに複数のスレッドを割り当てる必要がある．各スレッドに処理を割り振る方法を静的スケジューリングから動的スケジューリングに変更することで，遊休スレッドを減らしパイプラインを埋めてレイテンシを減らし性能を向上させる.

```
!$OMP DO
!$OMP&
SCHEDULE(static)
!$OMP& COLLAPSE(2)
  do i = 1,nx
  do j = 1,ny
     compute (vxyv ~ vxr2)
  do k = 1,nz
  do l = 1,nv
!Finite
difference computation
  enddo
  enddo
  enddo
  enddo
```

```
!$OMP DO
!$OMP&SCHEDULE(dynamic)
!$OMP& COLLAPSE(2)
  do i = 1,nx
  do j = 1,ny
     compute (vxyv ~ vxr2)
  do k = 1,nz
  do l = 1,nv
!Finite difference computation
  enddo
  enddo
  enddo
  enddo
```

図 4　静的スケジューリングのコード(左)および動的スケジューリングのコード(右)

以上の最適化手法を適用した MIC の性能評価結果を表 3 に示す．MIC ではアクティブなスレッドを多数生成する方法が有効である.

表 3　最適化手法を適用した MIC の性能評価結果

	オリジナル	(1)	(1)+(2)
演算性能[Gflops]	14.24	38.65	44.37
ピーク性能比[%]	1.4	3.7	4.2
処理時間[ms]	1080	398	295

4.2 Tesla K20X 向けの最適化手法

(1) Warp divergence の回避

GPU は 1warp 内のスレッドに同じ命令が発行される．条件分岐のように warp 内のスレッドで異なる命令を発行する場合は異なる命令を実行するスレッドのみが動作し，他スレッドは遊休スレッドとなる warp divergence が発生する.

図 1 に示すコードでステンシル計算の係数は k に依存しないという特徴があるが，CPU 版のループ構造に従って最内の l, k ループにスレッドを割り当てるオリジナルの GPU 向けコード[8]では図 4 に示すような warp 分岐が発生していた．この問題を回避するために，ループ順序の変更により warp divergence を削除し，性能を向上させる.

```
l = threadidx%x
    +(blockidx%x-1)
    *blockDim%x
k = threadidx%y
    +(blockidx%y-1)
    * blockDim%y
do i = 1, nx
do j = 1, ny
if( threadidx%y == 1) then
Ci1(threadidx%x) = A(l,j,i)
+ A(l,j,i+1)
Ci2(threadidx%x) = A(l,j,i)
+ A(l,j,i-1)
end if
call syncthreads()
!Finite difference computation
end do
end do
```

```
l = threadidx%x
    +(blockidx%x-1)
    *blockDim%x
j = threadidx%y
    +(blockidx%y-1)
    *blockDim%y
do i = 1, nx
Ci1 = A(l,j,i)
    + A(l,j,i+1)
Ci2 = A(l,j,i)
    + A(l,j,i-1)
do k = 1, nz
!Finite difference computation
end do
end do
```

図 5　Warp divergence 回避前のコード(左)および回避後のコード(右)

(2) メモリアクセス最適化

複数スレッドが同時にメモリアクセスする場合，参照するアドレスの先頭が 128 バイト境界かつシーケンシャルアクセスの条件を満たす場合は高速なメモリアクセスであるコアレッシングロードを行う．GT5D のステンシル計算で参照する配列は要素数 2 の袖領域を持つため参照するアドレスの先頭が 128 バイト境界からずれる．参照する配列のアドレスが 128 バイト境界になるように袖領域に計算に使用しない領域を確保することでコアレッシングロードの条件を満たし高速なメモリアクセスによる性能向上を行う．さらに，ループ内で複数回参照するデータを読み込み速度の速いレジスタに配置して再利用することで性能向上を行う.

```
do k = 1, nz
! X difference computation
! Y difference computation
! V difference computation
  fzl =f(l,k,j,i)+f(l,k-1,j,i)
  fzr =f(l,k,j,i)+f(l,k+1,j,i)
  fzl2=f(l,k,j,i)+f(l,k-2,j,i)
  fzr2=f(l,k,j,i)+f(l,k+2,j,i)
end do
```

```
  fk1=f(l,k-2,j,i)
  fk2=f(l,k-1,j,i)
  fk3=f(l,k,j,i)
  fk4=f(l,k+1,j,i)
  do k = 1,nz
     fk5=f(l,k+2,j,i)
! X difference computation
! Y difference computation
! V difference computation
     fzl =fk3+fk2
     fzr =fk3+fk4
     fzl2=fk3+fk1
     fzr2=fk3+fk5
     fk1 = fk2; fk2 = fk3;
     fk3 = fk4; fk4 = fk5;
  enddo
```

図 6　レジスタを再利用しないコード(左)およびレジスタを再利用したコード(右)

以上の最適化手法を適用した K20X の性能評価結果を表 4 に示す．GPU は遊休スレッドの発生を防ぐために条件分岐の削除．コアッレシングロードによる高速なメモリアクセス．再利用データをメモリアクセスが高速なレジスタに配置する方法が有効である．

表 4　最適化手法を適用した K20X の性能評価結果

	オリジナル	(1)	(1)+(2)
演算性能[Gflops]	19.89	77.82	124.32
ピーク性能比[%]	1.5	5.9	9.5
処理時間[ms]	1026	262	164

4.3 SPARC64 XIfx 向けの最適化手法

(1) numactl によるメモリアクセス最適化

FX100 はローカル CMG のメモリにアクセスする場合とリモート CMG のメモリにアクセスする場合でアクセス時間が異なるため，numactl コマンドで各 CMG のメモリを平均して使用する interleave モードを利用することで各 CMG からのメモリアクセスを平均化させて性能向上を行う．

(2) 総和計算方法の変更

FX100 の SIMD 幅は 256 ビットであるため一つのレジスタに 4 つの倍精度浮動小数を持つ．総和計算を行う場合にレジスタ内で縮約計算が発生する．縮約計算は非 SIMD 演算になる．縮約計算をループ外に追い出すことで縮約の回数を減らし SIMD 命令率を上げて性能向上を行う．

```
!$OMP DO
!$OMP&
SCHEDULE(static,1)
  do i = 1,nx
  do j = 1,ny
     compute (vxyv ~ vxr2)
  do k = 1,nz
  do l = 1,nv
!Finite
  difference computation

dfc=dfc+df(l,k,j,i)*fc(l,k,j,i)
  enddo
  enddo
  enddo
  enddo
```

```
!$OMP DO
!$OMP& SCHEDULE(static,1)
  do i = 1,nx
  do j = 1,ny
     compute (vxyv ~ vxr2)
  do k = 1,nz
  do l = 1,nv
!Finite difference computation
dfc(l)=dfc(l)+df(l,k,j,i)*fc(l,k,j,i)
  enddo
  enddo
  enddo
  enddo
!$OMP END DO NOWAIT

  do l = 1,nv
    dfc=dfc+dfc(l)
  enddo
```

図 7　総和計算をループ内で行うコード(左)および総和計算をループ外で行うコード(後)

上記の最適化手法を適用した FX100 の性能評価結果を表 5 に示す．FX100 では numactl によってメモリアクセスを最適化するだけで，コード修正無しに高い実効性能が得られた．さらに，非 SIMD 演算の低減が有効であることがわかった．

表 5　最適化手法を適用した FX100 の性能評価結果

	オリジナル	(1)	(1)+(2)
演算性能[Gflops]	99.25	139.25	141.31
ピーク性能比[%]	9.8	13.8	14.0
処理時間[ms]	130	93	91

5. 性能評価結果の比較

各アーキテクチャの最適化手法適用後の性能評価結果とルーフラインモデルで評価した実行性能を表 6 に示す．全てのアーキテクチャで最適化手法を適用することで演算性能が向上する結果を得られたが，ルーフラインモデルに基づく理論的予測に比べてかなり処理性能が低いので改善の余地がある．

MIC の結果は共有キャッシュを持つ FX100 よりも理論演算性能との開きが大きい．ここで，FX100 のキャッシュサイズは 24MB．MIC は各コアのローカルキャッシュがリング接続された構成の 30MB である．しかし，MIC ではローカルキャッシュへのアクセスは数十クロックで完了するのに対して，リモートキャッシュへのアクセスには数百クロックかかる[9]．メインメモリへのアクセスが数百クロックかかるのでリモートキャッシュへのアクセスはメインメモリへのアクセスに近い性能である．そのため，MIC が高速にアクセスできるキャッシュはローカルキャッシュの 512KB だけである．高速にアクセスできるキャッシュが FX100 と比べて MIC では小さく，再利用するデータを高速に供給することができていないことが性能劣化の原因であると考えられる．現在のメニーコアアーキテクチャは演算性能と比べてバンド幅が不足している．メモリからデータ

を供給する時間が、供給されたデータの演算処理の時間を上回る，いわゆるメモリバンド幅律速のアプリケーションではメモリ待ちが発生して演算コアが動作しない時間が発生してしまう．GT5D のステンシル計算は演算数とメモリ参照数がほぼ同じであるため現在のメニーコアアーキテクチャではメモリバンド幅によって性能が決まってしまう．これは，理論演算性能では MIC，FX100 よりも高い演算性能を持つ K20X がルーフラインモデルでは最も低い実行性能を与えることからもわかる．汎用コアである FX100 はアクセラレータである MIC，K20X よりも理論演算性能と理論バンド幅の差が小さいため演算性能が最も良い結果になった．

表 6　最適化手法適用後の性能評価結果

	MIC	K20X	FX100
演算性能[Gflops] (実測値)	45.92	124.32	141.31
ピーク性能比[%]	4.4	9.5	14.0
処理時間[ms]	335	164	91
実行性能[Gflops] (ルーフライン)	251.20	209.94	332.03
演算性能/実行性能	0.18	0.59	0.46

6. おわりに

本研究では核融合プラズマ乱流コード GT5D におけるステンシル計算を対象に各アーキテクチャで演算性能を改善させる最適化手法を開発した．開発手法を適用することで性能が MIC で 3.66 倍，K20X で 6.26 倍，FX100 で 1.42 倍の性能向上がそれぞれ得られた．しかしながら，これらの演算性能はルーフラインモデルに基づく実行性能予測よりも低いため，更なる性能向上の余地がある．GT5D のステンシル計算はメモリアクセスによって性能が律速されるアプリケーションであるため，メモリアクセスを高速化させる最適化手法の開発が今後の課題である．

謝辞　SPARC64 XIfx における最適化手法に関して助言を頂きました富士通株式会社の三吉郁夫様にお礼を申し上げます．

参考文献

1) ITER. https://www.iter.org
2) Y. Idomura, et al., Conservative global gyrokinetic toroidal full-f five-dimensional Vlasov simulation., Comput. Phys. Commun. 179 391, 2008.
3) Y. Idomura, et al., Communication-overlap techniques for improved strong scaling of gyrokinetic Eulerian code beyond 100k cores on the K-computer, International Journal of High Performance Computing Applications 28, 73-86, 2014.
4) INTEL. Intel xeon phi coprocessor inst. set archi. ref. manual. https:/software.intel.com/en-us/mic-developer.
5) Nvidia. TESLA K20X GPU ACCELERATOR http://www.nvidia.co.jp/content/PDF/kepler/Tesla-K20X-BD-06397-001-v07.pdf
6) Fujitsu. FUJITSU Supercomputer PRIMEHPC SPARC64 CPU http://www.fujitsu.com/global/Images/primehpc-SPARC64 CPU-datasheet-en.pdf
7) S. Williams et al., Commun. ACM 52, 65 (2009).
8) N. Fujita, Hideo Nuga, Taisuke Boku, Yasuhiro Idomura, Nuclear Fusion Simulation Code Optimization on GPU Clusters, The 15th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing, 2014.
9) Jianbin Fang, Ana Lucia Varbanescu, Henk J. Sips, Lilun Zhang, Yonggang Che, and Chuanfu Xu, An empirical study of Intel Xeon Phi, CoRR, 2013.