2.3 惑星磁気圏 MHD コードの性能評価

京都大学 学術情報メディアセンター
深沢 圭一郎

2.3.1 はじめに

MHD シミュレーションはプラズマの振る舞いを正確に記述できる Vlasov 方程式の近似から求められる MHD 方程式を利用したシミュレーションである。MHD 方程式は Vlasov 方程式を近似する際に、速度空間を落とすことで、計算する次元数を減らし、プラズマの大規模構造を計算する際によく利用されている。惑星磁気圏 MHD コードは惑星の磁場の勢力範囲である磁気圏をグローバルに解くシミュレーションコードである。太陽からのプラズマの流れ（太陽風）と惑星の磁場が相互作用することで形作られる惑星磁気圏は、名に見える現象ではオーロラに代表されるように様々な電磁気現象が起きている。これらは宇宙空間で運用されている GPS などの衛星や宇宙ステーションだけで無く、誘導電流などの形で地上にも影響を与えている。これらの現象は宇宙天気と呼ばれており、世界中で広く研究されている。惑星磁気圏 MHD コードはこの宇宙天気予報において基本となるシミュレーションコードとして利用されている。この惑星磁気圏 MHD コードはベクトル機全盛時代から利用されているが、近年では超並列スカラ計算機にも対応し、様々な計算機上で利用されている。今回ポストペタスケール計算に向けて、京コンピュータの後継機である FX10 とさらにその後継機である FX100 において、惑星磁気圏 MHD コードの性能評価を行った。

2.3.2 プログラム概要

MHD 方程式は以下のようになる。

$$
\begin{aligned}
\frac{\partial \rho}{\partial t} &= -\nabla\cdot(\mathbf{v}\rho) \\
\frac{\partial \mathbf{v}}{\partial t} &= -(\mathbf{v}\cdot\nabla)\mathbf{v} - \frac{1}{\rho}\nabla p + \frac{1}{\rho}\mathbf{J}\times\mathbf{B} \\
\frac{\partial p}{\partial t} &= -(\mathbf{v}\cdot\nabla)p - \gamma p\nabla\cdot\mathbf{v} \\
\frac{\partial \mathbf{B}}{\partial t} &= \nabla\times(\mathbf{v}\times\mathbf{B})
\end{aligned}
\qquad (1)
$$

上から、連続の式、運動方程式、圧力変化の式（エネルギーの式）、最後が磁場の誘導方程式となる。簡単に言えば、電磁場を考慮した流体力学方程式と呼べる。MHD 方程式を解く数値計算法としては、*Ogino* ら[1992]によって開発された Modified Leap Frog（MLF）法という差分法を使用する。これは最初の 1 回を two step Lax-Wendroff 法で解き、続く （l － 1）回を Leap Frog 法で解き、その一連の手続きを繰り返す手法である。l の値は数値的に安定の範囲で大きい方が望ましいので、数値精度の線形計算と予備的シミュレーションから l= 8 に選んでいる。惑星磁気圏 MHD コードは直交格子を利用しており、いわゆる stencil 計算である。並列は MPI と OpenMP を利用し、領域分割を行っている。1、２、３次元分割を性能評価に利用している。通信は基本的には袖領域の隣接通信のみである。

2.3.3 測定環境

性能測定には、東京大学 FX10、名古屋大学 FX100 を利用した。各計算機の構成は表 1 の通りである。測定には、最大で FX10 を 4800 ノード（76,800 コア）と FX100 を 512 ノード（16,384 コア）を利用した。

表 1. FX10、FX100 のシステム構成

		PRIMEHPC FX10	PRIMEHPC FX100
CPU	Architecture	16 cores SPARC64 IXfx	32 cores SPARC64 XIfx
	Frequency	1.848 GHz（236.544 GFlops)	2.2 GHz （1126.4 GFlops)
	Cache	L1: 32 KB/core L2: 12 MB/CPU	L1: 64 KB/core L2: 24 MB/CPU
Memory	Band width	85 GB/s /CPU (=node)	240 x2 (in/out) GB/s /CPU
B/F		0.36	0.21 or 0.42
Node	Number of CPUs	1	1
	Memory size	32 GB	32 GB
System	Number of nodes	4,800 (76,800 cores)	2,880 (92,1600 cores)
	Rmax	1.135 PFlops	3.244 PFlops
	Node comm。	Tofu Interconnect (5 GB/s)	Tofu 2 (12.5 GB/s)

2.3.4 測定結果

FX10 において、MHD コードの性能評価結果を図 1 に載せる。ここでは 1、2、3 次元領域分割と 3 次元領域分割において計算する配列並びが f(i,j,k,m)：typeA と f(m,i,j,k)：typeB の場合を評価した。最も性能が高かったのは 3 次元領域分割 typeB で、約 210TFlops（実行効率 19%）となった。一方で配列の順序が違うだけの typeA では 110TFlops と半分近くの性能になっている。この二つの L2 キャッシュミス率などを調べた結果、表 2 のようになっており、L2 キャッシュミス率とメモリバンド幅の違いが性能の違いに効いていることがわかる。また、L1 キャッシュスラッシングにより typeA の性能が劣化していることも分かっている。最も性能の高い 3 次元領域分割 typeB に式の共通化、配列の次元削減とループ融合といった最適化を施した結果、32 ノード利用時に実行効率が 30%となった。

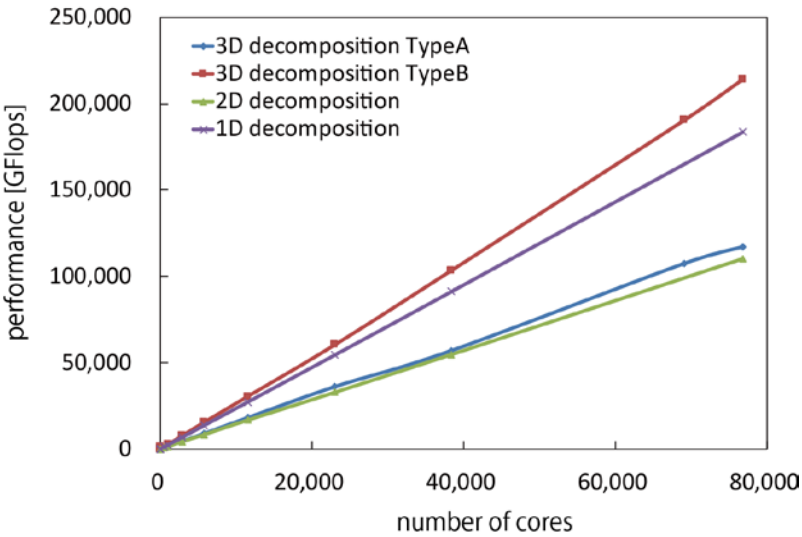


図 1　FX10 における性能測定結果

表 2　3 次元領域分割 typeA と B における SIMD 化率、L2 demand キャッシュミス率、メモリバンド幅

Array type	SIMD efficiency [%]	L2 demand cache miss [%]	Memory bandwidth [GB/s]
Type A	65.58	19.81	7.49
Type B	54.26	3.64	24.94

FX100 での性能評価結果を図 2 に載せる。ここでは FX10 での最適化は加えていない。最も性能が高くなったのは 1 次元領域分割（実行効率 13%）で、次が 3 次元領域分割 typeA(同 10.6%)であった。FX10 で最も性能の高かった typeB は FX100 では性能が最も出なかった。このように FX10 と FX100 はコードに対する振る舞いが異なり、FX100 ではベクトル的なコードが速いという結果になった。今度は 3 次元領域分割 typeA に対してループや文の分割、プロセスと計算サイズの調整、配列の分割、空間タイリングなどを施したところ、16.7%の実行効率を達成することができた。

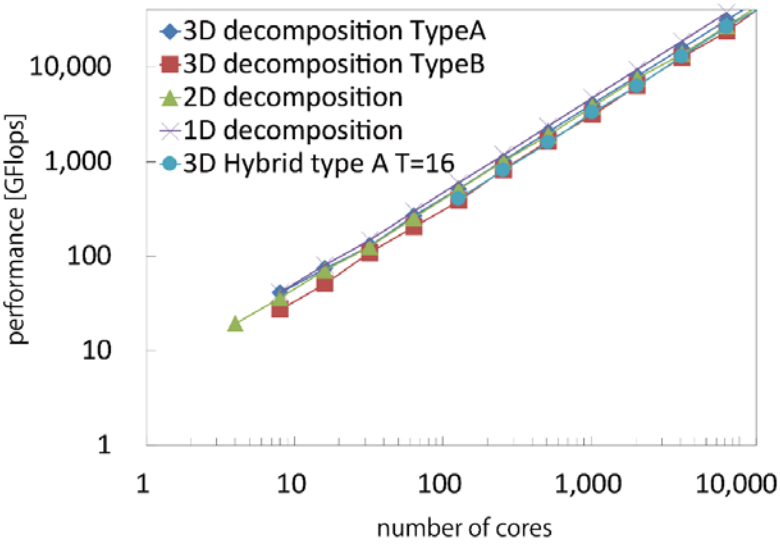


図 2　FX100 における性能測定結果

2.3.5 まとめ

惑星磁気圏を解く MHD シミュレーションコードを用いて、京の後継機種である FX10 と FX100 の性能評価を行った。基本性能測定として、1 次元、2 次元、3 次元領域分割 typeA と typeB を用いて、それぞれの性能を測定した。最も性能が高くなったのは 3 次元領域分割 typeB（実行効率 19%）であり、typeA と比べて 2 倍程度の性能差があった。これは HPC2500 から続く SPARC64v 系の CPU と同様の傾向であり、キャッシュヒットを考慮したコードの性能が高いことを示している。最も性能が高かった 3 次元領域分割 typeB にキャッシュをより効率よく利用できるようないくつか最適化を加えたところ、最大で実行効率が 30%まで向上した。

一方で FX100 に対して、FX10 と同様の性能測定を行ったところ、1 次元領域分割がもっとも性能が高く（実行効率 13%）、次に 3 次元領域分割 typeA（10.6%）となった。FX10 で最も性能の高かった 3 次元領域分割 typeB は FX100 では逆に最も性能が低かった。これはキャッシュヒットを考慮するよりも、ベクトル化を考慮した方が良いことを示している。FX100 では 3 次元領域分割 typeA に最適化を加えた。ループや文の分割によりコンパイラの最適化が促進され、プロセス分割の調整、配列分割やサイクリック並列化により L2 キャッシュミスが減少し、最終的には 16.7%の実行効率となった。

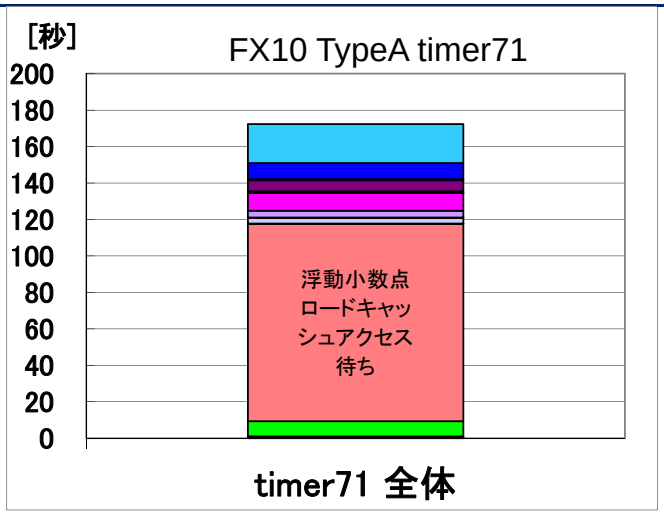
また、今回の FX100 での結果を他の計算機システムと比較すると、1CPU あたりで SX-ACE の 1.6 倍程度の性能になっており、KNC の Xeon Phi の 2 倍程度の性能となっていた。

惑星磁気圏 MHDコードの性能評価

深沢圭一郎
京都大学学術情報メディアセンター

SS研SS研ポストペタアプリ性能WG

【FX10】TypeA timer71でL1Dキャッシュスラッシング



TypeA	L1D ミス率（/ロード・ストア数）	L1D ミス dm率（/L1D ミス数）	L1Dミス数
timer71	2.62%	33.15%	4.96E+10

配列が連続アクセスであるにも関わらずL1Dキャッシュミス率が高く(目安:3.125%以上)、L1Dキャッシュミスのデマンドの内訳が高い(目安:20%以上) ストリーム数が58個と多い
⇒ L1Dキャッシュスラッシングの疑い有り

ソース(一部)

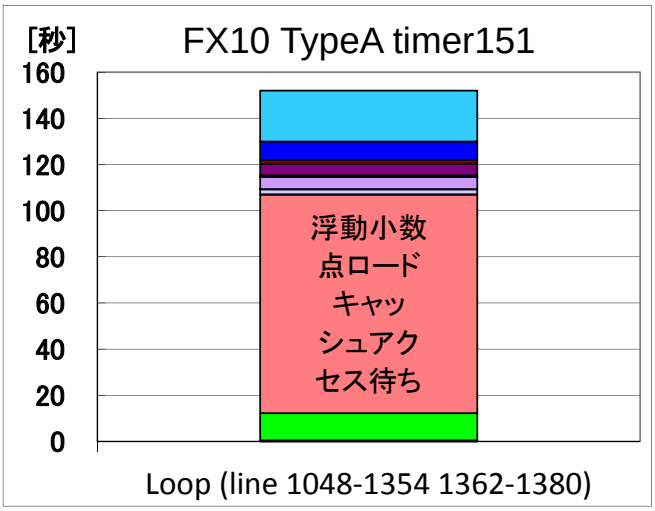
```
340  3 p v   do i = is, ie1
341  3 p v     x =0.5*hx*float(2*(iss+i)-nx2+2*nxp)
342  3 p v     ra2 = x * x + y * y + z * z
343  3 p v     ra2 = max(ra2, ra_i2)
344  3 p v     ra3 = ra2 * sqrt(ra2)
346  3 p v     u(i,j,k,8) = u(i,j,k,8)                    &
347  3         + dx1 * (                                  &
348  3           f(i+1,j+1,k+1,4) * f(i+1,j+1,k+1,6)      &
349  3         - f(i+1,j+1,k+1,2) * f(i+1,j+1,k+1,8)      &
350  3         + f(i+1,j,k+1,4) * f(i+1,j,k+1,6)          &
351  3         - f(i+1,j,k+1,2) * f(i+1,j,k+1,8)          &
352  3         + f(i+1,j+1,k,4) * f(i+1,j+1,k,6)          &
353  3         - f(i+1,j+1,k,2) * f(i+1,j+1,k,8)          &
354 3         + f(i+1,j,k,4) * f(i+1,j,k,6)              &
(省略)
```

2インデックスずれているが2ライン（インデックス）先をプリフェッチする影響で**L1Dキャッシュスラッシングが発生している。**

L1D競合ツール結果(一部)

配列名	アドレス	Index
f(i,j+1,k+1,5)	0x9BB812F0	37→39
f(i,j,k,8)	0xF861D3F8	39
ff(i,j,k,7)	0x7FEC2469388	39

【FX10】TypeA timer151でL1Dキャッシュスラッシング



ソース(一部)

```
1362  3  p do i = is1, ie1
 (省略)
1371  3  p   f(i,j,k,8)=f(i,j,k,8)*x2+ff(i,j,k,8)*(1.0-x2)
1372  3  p   f(i,j,k,7)=f(i,j,k,7)*x2+ff(i,j,k,7)*(1.0-x2)
1373  3  p   f(i,j,k,6)=f(i,j,k,6)*x2+ff(i,j,k,6)*(1.0-x2)
1374  3  p   f(i,j,k,5)=f(i,j,k,5)*x2+ff(i,j,k,5)*(1.0-x2)
1375  3  p   f(i,j,k,4)=f(i,j,k,4)*x2+ff(i,j,k,4)*(1.0-x2)
1376  3  p   f(i,j,k,3)=f(i,j,k,3)*x2+ff(i,j,k,3)*(1.0-x2)
1377  3  p   f(i,j,k,2)=f(i,j,k,2)*x2+ff(i,j,k,2)*(1.0-x2)
1378  3  p   f(i,j,k,1)=f(i,j,k,1)*x2+ff(i,j,k,1)*(1.0-x2)
1379  3   !
1380  3  p end do
```

2インデックスずれているが2ライン(インデックス)先をプリフェッチする影響で**L1Dキャッシュスラッシングが発生している。**

TypeA	L1D ミス率 (/ロード・ストア数)	L1D ミス dm率 (/L1D ミス数)	L1Dミス数
timer151	9.62%	55.81%	7.51E+10

配列が連続アクセスであるにも関わらずL1Dキャッシュミス率が高く(目安:3.125%以上)、L1Dキャッシュミスのデマンドの内訳が高い(目安:20%以上) ストリーム数が91個と多い
⇒ L1Dキャッシュスラッシングの疑い有り

L1D競合ツール結果(一部)

配列名	アドレス	Index
ff(i,j-1,k,2)	0x7FE27C5D0B0	34→36
u(i,j,k,5)	0x192E29098	34→36
u(i,j-1,k-1,8)	0x7FE08DF5120	36
p(i,j,k,2)	0x7FCD3DDD198	36

式の共通化@FX10

計算式が長いとコンパイラが適切にスケジューリングできない場合がある

```
u(8,i,j,k) = u(8,i,j,k)                                  &
           + dx1 * (                                      &
             f(4,i+1,j+1,k+1) * f(6,i+1,j+1,k+1)          &
           - f(2,i+1,j+1,k+1) * f(8,i+1,j+1,k+1)          &
           + f(4,i+1,j  ,k+1) * f(6,i+1,j  ,k+1)          &
           - f(2,i+1,j  ,k+1) * f(8,i+1,j  ,k+1)          &
           + f(4,i+1,j+1,k  ) * f(6,i+1,j+1,k  )          &
           - f(2,i+1,j+1,k  ) * f(8,i+1,j+1,k  )          &
           + f(4,i+1,j  ,k  ) * f(6,i+1,j  ,k  )          &
           - f(2,i+1,j  ,k  ) * f(8,i+1,j  ,k  )          &
           - f(4,i  ,j+1,k+1) * f(6,i  ,j+1,k+1)          &
           + f(2,i  ,j+1,k+1) * f(8,i  ,j+1,k+1)          &
           - f(4,i  ,j  ,k+1) * f(6,i  ,j  ,k+1)          &
           + f(2,i  ,j  ,k+1) * f(8,i  ,j  ,k+1)          &
           - f(4,i  ,j+1,k  ) * f(6,i  ,j+1,k  )          &
           + f(2,i  ,j+1,k  ) * f(8,i  ,j+1,k  )          &
           - f(4,i  ,j  ,k  ) * f(6,i  ,j  ,k  )          &
           + f(2,i  ,j  ,k  ) * f(8,i  ,j  ,k  ) )        &
```

```
 ffff11 = f(4,i+1,j+1,k+1) * f(6,i+1,j+1,k+1)   &
        - f(2,i+1,j+1,k+1) * f(8,i+1,j+1,k+1)   &
        + f(4,i+1,j,k+1) * f(6,i+1,j,k+1)       &
        - f(2,i+1,j,k+1) * f(8,i+1,j,k+1)
 ffff21 = f(4,i+1,j+1,k) * f(6,i+1,j+1,k)       &
        - f(2,i+1,j+1,k) * f(8,i+1,j+1,k)       &
        + f(4,i+1,j,k) * f(6,i+1,j,k)           &
        - f(2,i+1,j,k) * f(8,i+1,j,k)
 ffff31 = f(4,i,j+1,k+1) * f(6,i,j+1,k+1)       &
        - f(2,i,j+1,k+1) * f(8,i,j+1,k+1)       &
        + f(4,i,j,k+1) * f(6,i,j,k+1)           &
        - f(2,i,j,k+1) * f(8,i,j,k+1)
 ffff41 = f(4,i,j+1,k) * f(6,i,j+1,k)           &
        - f(2,i,j+1,k) * f(8,i,j+1,k)           &
        + f(4,i,j,k) * f(6,i,j,k)               &
        - f(2,i,j,k) * f(8,i,j,k)
!
u(8,i,j,k) = u(8,i,j,k)                         &
           + dx1 * (                            &
             ffff11 + ffff21                    &
           - ffff31 - ffff41                    &
                    )                           &
           - dy1 * (                            &
             ffff12 + ffff22                    &
           - ffff32 - ffff42                    &
                    )
```

Asis: 192.231sec
最適化後:156.230sec
1.23倍の性能向上

配列の次元削減とループ融合@FX10

必要最低限の配列次元にすることで、効率化

```
 do k = ks, ke1
     do j = js, je1
       do i = is, ie1
         do m = 1, nb
           ff(m,i,j,k) = u(m,i,j,k)
         end do
       end do
!
       do i = is, ie1
.
.
.
.
```

```
 do k = ks, ke1
      do j = js, je1
!OCL LOOP_NOFUSION
        do i = is, ie1
          ff_wk(1,i) = u(1,i,j,k)
          ff_wk(2,i) = u(2,i,j,k)
          ff_wk(3,i) = u(3,i,j,k)
          ff_wk(4,i) = u(4,i,j,k)
          ff_wk(5,i) = u(5,i,j,k)
          ff_wk(6,i) = u(6,i,j,k)
          ff_wk(7,i) = u(7,i,j,k)
          ff_wk(8,i) = u(8,i,j,k)
        end do
!
        do i = is, ie1
.
.
.
.
```

Asis: 192.231sec
最適化後:172.109sec
1.12倍の性能向上

京都大学
KYOTO UNIVERSITY

最適化結果@FX10

今までの最適化すべてを適用した

4ノード利用の結果（計算サイズは800×800×800）

- Flat MPIは64プロセス（4×4×4）
  asis:192.321sec（22.66%）→135.242sec（28.59%）
- Hybrid MPI：4プロセス（2×2×1）+16スレッド
  asis:192.570sec（22.78%）→133.895sec（27.67%）

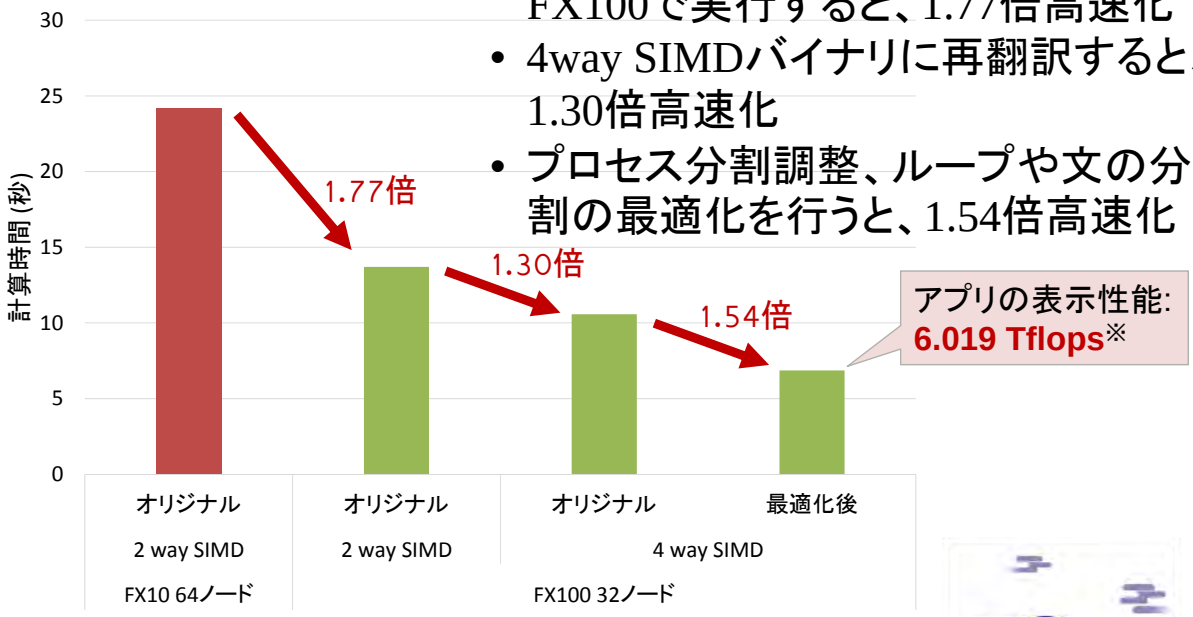
32ノード利用の結果（計算サイズは1600×400×400）

- Flat MPIは512プロセス（8×8×8）
  asis:12.783sec（21.05%）→8.765sec（27.34%）
- Hybrid MPI：64プロセス（4×4×4）+8スレッド
  asis:13.247sec（20.51%）→7.672sec（30.34%）

京都大学
KYOTO UNIVERSITY

結果概要

- FX10用2way SIMDバイナリをFX100で実行すると、1.77倍高速化
- 4way SIMDバイナリに再翻訳すると、1.30倍高速化
- プロセス分割調整、ループや文の分割の最適化を行うと、1.54倍高速化



※:コンパイラ最適化による演算数変化は考慮せず

京都大学
KYOTO UNIVERSITY

7

FX10とFX100の比較

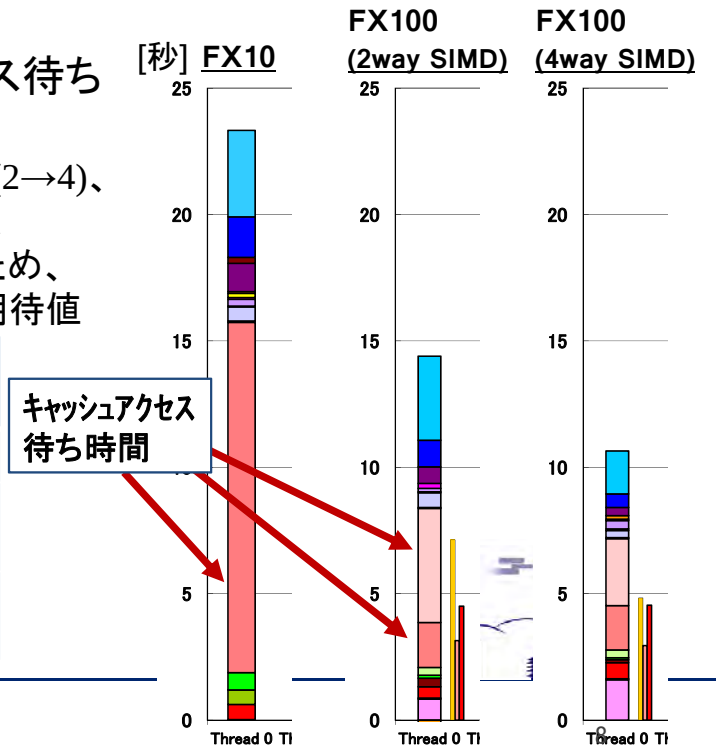
サイクルアカウンティング結果を比較

asisコードで測定

FX100では、キャッシュアクセス待ち時間が減少し、性能が向上

- L1D$容量(32→64KB)、ウェイ数(2→4)、およびスループットの倍増が貢献
- ラインサイズ2倍(128→256B)のため、連続アクセス時はミス率半減が期待値

	キャッシュアクセス待ち時間	L1D$ ミス率
FX10	13.9秒	3.95%
FX100 (2way SIMD)	6.3秒	1.54%
FX100 (4way SIMD)	4.4秒	1.53%



京都大学
KYOTO UNIVERSITY

8

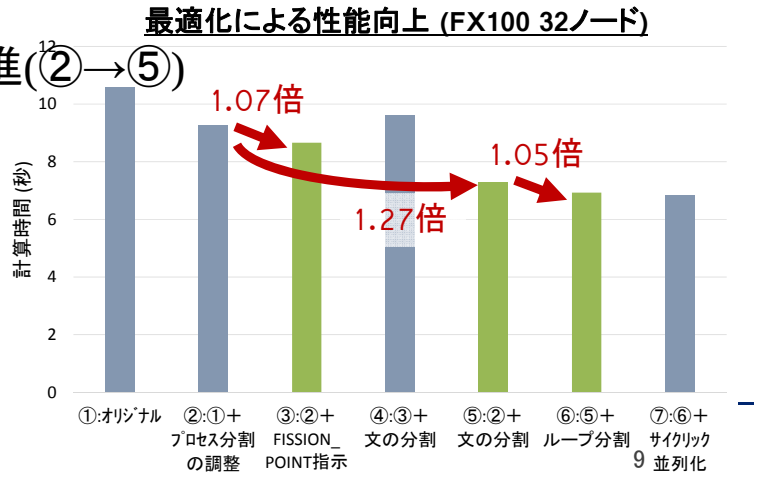
ループや文の分割による性能向上

ステンシル計算だが、L1D$ミス率が連続アクセスより高い

- データ再利用により、連続アクセス時以下が期待値
  →L1D$スラッシング?
- FISSION_POINT指示行の挿入により、ループ分割
- ループ分割: ループ分割を人手で実施
- 文の分割: 右辺が長い代入文を、複数の部分式からなる文に人手で分割
  ⇒コンパイラ最適化が促進(②→⑤)

	L1D$ミス率
②	1.71%
②＋FISSION_POINT指示	1.57%
⑤	1.86%
⑤＋ループ分割	1.81%

まだ高く、改善余地あり



プロセス分割の調整、サイクリック並列化、他

プロセス分割の調整

- スレッド並列化対象ループの繰返し数をスレッド数の倍数に
- 最内ループの繰返し数を多めに

	プロセス分割	格子点数/プロセス
調整前	4×4×4	200×200×**200**
調整後	1×32×2	**800**×25×400

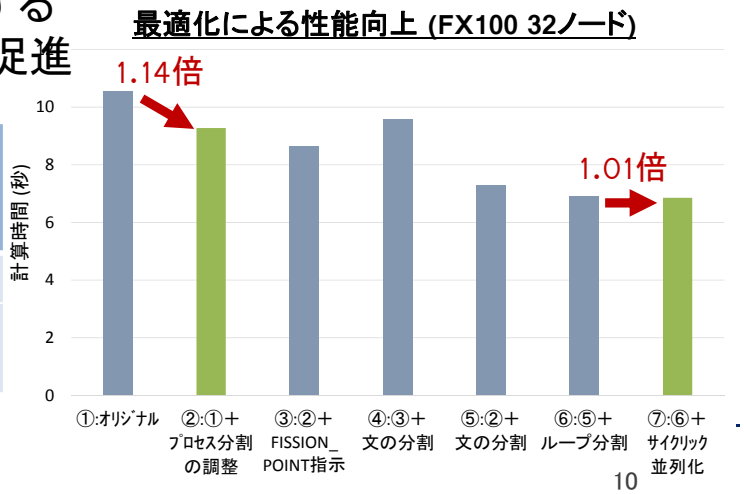
サイクリック並列化

- Z方向の差分計算で参照するデータのスレッド間共有を促進

```
!$OMP PARALLEL DO SCHEDULE(STATIC,1) ...
   do k = ks, ke1
```

	L2$ ミス率	メモリスループット/ノード
⑥	1.09%	167.7GB/秒
⑥＋ｻｲｸﾘｯｸ並列化	0.95%	140.4GB/秒

余力があるため、性能向上幅が小



配列の分割

配列f(nx,ny,nz,8)をf1(nx,ny,nz,4)とf2(nx,ny,nz,4)に分割

```
 real ::  f(nx2, ny2, 0:nzz+1, 8)
 real ::  u(nx2, ny2, 0:nzz+1, 8)
!
 u(i,j,k,8) = u(i,j,k,8)                                  &
              + dx1 * (                                   &
                f(i+1,j+1,k+1,4) * f(i+1,j+1,k+1,6)       &
              - f(i+1,j+1,k+1,2) * f(i+1,j+1,k+1,8)       &
              + f(i+1,j,k+1,4) * f(i+1,j,k+1,6)           &
              - f(i+1,j,k+1,2) * f(i+1,j,k+1,8)           &
              + f(i+1,j+1,k,4) * f(i+1,j+1,k,6)           &
              - f(i+1,j+1,k,2) * f(i+1,j+1,k,8)           &
              + f(i+1,j,k,4) * f(i+1,j,k,6)               &
              - f(i+1,j,k,2) * f(i+1,j,k,8)               &
              - f(i,j+1,k+1,4) * f(i,j+1,k+1,6)           &
              + f(i,j+1,k+1,2) * f(i,j+1,k+1,8)           &
              - f(i,j,k+1,4) * f(i,j,k+1,6)               &
              + f(i,j,k+1,2) * f(i,j,k+1,8)               &
              - f(i,j+1,k,4) * f(i,j+1,k,6)               &
              + f(i,j+1,k,2) * f(i,j+1,k,8)               &
              - f(i,j,k,4) * f(i,j,k,6)                   &
              + f(i,j,k,2) * f(i,j,k,8)  )                &
```

```
 real :: f1(0:nxx+1,0:nyy+1,0:nzz+1,4)
 real :: f2(0:nxx+1,0:nyy+1,0:nzz+1,4)
 real :: u1(0:nxx+1,0:nyy+1,0:nzz+1,4)
 real :: u2(0:nxx+1,0:nyy+1,0:nzz+1,4)
!
 u2(i,j,k,4) = u2(i,j,k,4)                                  &
               + dx1 * (                                    &
                 f1(i+1,j+1,k+1,4) * f2(i+1,j+1,k+1,2)      &
               - f1(i+1,j+1,k+1,2) * f2(i+1,j+1,k+1,4)      &
               + f1(i+1,j,k+1,4) * f2(i+1,j,k+1,2)          &
               - f1(i+1,j,k+1,2) * f2(i+1,j,k+1,4)          &
               + f1(i+1,j+1,k,4) * f2(i+1,j+1,k,2)          &
               - f1(i+1,j+1,k,2) * f2(i+1,j+1,k,4)          &
               + f1(i+1,j,k,4) * f2(i+1,j,k,2)              &
               - f1(i+1,j,k,2) * f2(i+1,j,k,4)              &
               - f1(i,j+1,k+1,4) * f2(i,j+1,k+1,2)          &
               + f1(i,j+1,k+1,2) * f2(i,j+1,k+1,4)          &
               - f1(i,j,k+1,4) * f2(i,j,k+1,2)              &
               + f1(i,j,k+1,2) * f2(i,j,k+1,4)              &
               - f1(i,j+1,k,4) * f2(i,j+1,k,2)              &
               + f1(i,j+1,k,2) * f2(i,j+1,k,4)              &
               - f1(i,j,k,4) * f2(i,j,k,2)                  &
               + f1(i,j,k,2) * f2(i,j,k,4)  )               &
```

Asis:　　19.82sec
分割後: 18.71sec
1.06倍の性能向上
*SB Xeonでは1.37倍

タイリング

ループをブロック化

```
 real ::  f(nx2, ny2, 0:nzz+1, 8)
 real ::  u(nx2, ny2, 0:nzz+1, 8)
!
 u(i,j,k,8) = u(i,j,k,8)                                  &
              + dx1 * (                                   &
                f(i+1,j+1,k+1,4) * f(i+1,j+1,k+1,6)       &
              - f(i+1,j+1,k+1,2) * f(i+1,j+1,k+1,8)       &
              + f(i+1,j,k+1,4) * f(i+1,j,k+1,6)           &
              - f(i+1,j,k+1,2) * f(i+1,j,k+1,8)           &
              + f(i+1,j+1,k,4) * f(i+1,j+1,k,6)           &
              - f(i+1,j+1,k,2) * f(i+1,j+1,k,8)           &
              + f(i+1,j,k,4) * f(i+1,j,k,6)               &
              - f(i+1,j,k,2) * f(i+1,j,k,8)               &
              - f(i,j+1,k+1,4) * f(i,j+1,k+1,6)           &
              + f(i,j+1,k+1,2) * f(i,j+1,k+1,8)           &
              - f(i,j,k+1,4) * f(i,j,k+1,6)               &
              + f(i,j,k+1,2) * f(i,j,k+1,8)               &
              - f(i,j+1,k,4) * f(i,j+1,k,6)               &
              + f(i,j+1,k,2) * f(i,j+1,k,8)               &
              - f(i,j,k,4) * f(i,j,k,6)                   &
              + f(i,j,k,2) * f(i,j,k,8)  )                &
```

```
 do kt = ks, ke1, kb
     do jt = js, je1, jb
       do it = is, ie1, ib
         do k = kt, min(ke1, kt+kb-1)
           do j = jt, min(je1, jt+jb-1)
             do i = it, min(ie1, it+jb-1)
!
               u(i,j,k,8) = u(i,j,k,8)                      &
               + dx1 * (                                    &
                 f(i+1,j+1,k+1,4) * f(i+1,j+1,k+1,6)        &
               - f(i+1,j+1,k+1,2) * f(i+1,j+1,k+1,8)        &
               + f(i+1,j,k+1,4) * f(i+1,j,k+1,6)            &
               - f(i+1,j,k+1,2) * f(i+1,j,k+1,8)            &
               + f(i+1,j+1,k,4) * f(i+1,j+1,k,6)            &
               - f(i+1,j+1,k,2) * f(i+1,j+1,k,8)            &
               + f(i+1,j,k,4) * f(i+1,j,k,6)                &
               - f(i+1,j,k,2) * f(i+1,j,k,8)                &
               - f(i,j+1,k+1,4) * f(i,j+1,k+1,6)            &
               + f(i,j+1,k+1,2) * f(i,j+1,k+1,8)            &
               - f(i,j,k+1,4) * f(i,j,k+1,6)                &
               + f(i,j,k+1,2) * f(i,j,k+1,8)                &
               - f(i,j+1,k,4) * f(i,j+1,k,6)                &
               + f(i,j+1,k,2) * f(i,j+1,k,8)                &
               - f(i,j,k,4) * f(i,j,k,6)                    &
```

Asis:　　19.82sec
最適化後:42.69sec
2.5倍の性能劣化
PAによると、SIMD化がうまくいっていない。

Performance measurement of MHD code

Comparison of FX100 with other computer systems

	Core/CPU	Rmax [TFlops]	Rpeak [TFlops]	Rpeak /CPU [GFlops]	Efficiency [%]	Suitable domain decomposition	CPU architecture
SX-ACE	1024/256	65.50	29.20	114.0	45	3D_A	Vector
K	262144/32768	4194.30	914.12	27.9	22	3D_B	SPARC64 VIIIfx
FX10	76800/4800	1135.41	234.59	48.9	21	3D_B	SPARC64 IXfx
FX100	16384/512	576.72	91.49	178.7	17	3D_A	SPARC64 XIfx
RX200S6	864/144	10.13	3.51	24.4	35	3D_A	Xeon (Westmere)
CX400	23616/2952	510.11	104.23	35.3	20	3D_A	Xeon (SB)
HA8000	23160/1930	500.26	83.42	43.2	17	2D	Xeon (IB)
XC30-HSW	448/32	16.49	1.37	42.8	8	2D	Xeon (HSW)
SR16000/L2	1344/672	25.27	5.38	8.0	21	3D_B	POWER6
Xeon Phi 5120	60/1	1.00	0.08	84.0	8.4	3D_A	Knights Corner
Tesla K20X	896/1	1.31	0.15	153.3	11.7	3D_A	Kepler