Vol.2020-HPC-175 No.15
2020/7/31

惑星磁気圏 MHD シミュレーションコードによる A64FX プロセッサ（FX700）の性能評価

深沢 圭一郎[†]　三吉 郁夫[†2]

概要：A64FX は「京」の後継機である「富岳」にも搭載される ARM プロセッサであり，SVE と呼ばれる HPC 拡張機能が搭載されており，また高バンド幅を実現した HBM2 メモリを実装した新しい CPU である．このように A64FX はこれまでのスーパーコンピュータ（スパコン）に採用されてきた CPU とは異なった特徴を持っており，その実際の性能に注目が集まっている．そこで，惑星磁気圏シミュレーションに利用されている MHD シミュレーションコードを利用し，A64FX の性能評価を行った．その結果，理論性能や CPU の構成に共通点のある Xeon Phi KNL と比べて，高い実行効率を示し，また最新の Xeon プロセッサよりも高い実行性能を示した．本研究では，これら性能評価と初期最適化結果を議論する．

キーワード：ARM プロセッサ，性能評価，MHD シミュレーション，富岳

Performance evaluation of A64FX ARM processor with MHD simulation of a planetary magnetosphere

KEIICHIRO FUKAZAWA[†1]　IKUO MIYOSHI[†2]

Abstract: A64FX is the Armv8 based new processor developed for Fugaku which is the successor supercomputer of K-computer. The HPC extension SVE is introduce to A64FX and which equipped with the high bandwidth memory HBM2. Considering these features, A64FX is completely different from the CPUs used in the previous supercomputer systems. To know the effective performance of A64FX, the evaluation using the MHD simulation code of planetary magnetosphere (not benchmark code) has been conducted. As the results, the execution efficiency of A64FX become higher than the Xeon Phi KNL which configuration resembles A64FX and achieved the good effective performance compared to the latest Xeon. In this study the performance evaluation and the primary optimization are discussed.

Keywords: ARM processor, Performance evaluation, MHD simulation, Fugaku

1. はじめに

2 度の Top500 の 1 位を獲得し，Graph500 など様々なランキングでも 1 位を獲得した京コンピュータは，2019 年 8 月に 7 年間の運用を終えた．京コンピュータはその共用を HPCI（High Performance Computing Infrastructure）が行ったことにより，これまでにスーパーコンピュータ（スパコン）を利用していなかった科学計算アプリケーションだけでなく，企業においても利用されるなどスパコンを利用するアプリケーションの拡大に寄与してきた．また，京コンピュータの供用開始前には，想像されていなかった機械学習が現在ではスパコン利用の大きな 1 分野になるなど，スパコンを利用するアプリケーションは，京コンピュータ完成前と現在では，大きく変わってきており，スパコンに求められる性能も多様化している．

このようなアプリケーションの計算需要を満たすために，スパコンの CPU も京コンピュータ完成前とは大きく変わってきている．2012 年頃には京コンピュータと同じく 8 コア CPU であった Xeon（Sandi Bridge 世代）はプロセス微細化技術やリーク電流の問題もあり，周波数の向上が難しくなった結果，コア数の増加（メニーコア化）や同時演算数の増加により，CPU 性能の向上を達成している．例えば，現在の Xeon（Skylake 世代）であれば，2.5GHz 程度の周波数に 20 コアを超えるコア数（ノード当たり 40 コア超）となり，AVX-512 による 512bit の SIMD 幅，FMA×2 により，クロックあたりの同時演算数が 32 となっている．これにより 1.5TFlops 以上の性能を達成している（京コンピュータの CPU である SPARC 64 VIIIfx の 10 倍以上）．また，Xeon Phi KNL では，1.5GHz 程度の周波数に 60 を超える CPU コア，同時演算数が 32 であり，その結果，理論性能は 3 TFlops に達する．

一方，新しい HPC 向け CPU アーキテクチャとして ARM プロセッサがある．モバイル系で主に利用されている ARM プロセッサは，その主な用途のため，計算性能を制限する

†1 京都大学・学術情報メディアセンター
Academic Center for Computing and Media Studies, Kyoto University
†2 富士通株式会社
Fujitsu Limited

ことで消費電力を下げており，HPC のような高性能計算には利用されてこなかった．このような中 Cavium（現 Marvell）は，計算性能を高めたサーバ向け ARM プロセッサ ThunderX2 をリリースしている．ThunderX2 は 32 コアとコア数が多い一方で同時演算数が 8 のため，CPU の理論性能が約 1TFlops と Skylake 世代の Xeon と比べても半分近く低い．しかしながら，ベンチマークや実アプリケーションの性能測定結果では，ThunderX2 は，Skylake 世代の Xeon と比べて，80%程度の性能と報告がなされており[1, 2]，HPC 分野でも十分に利用可能な CPU となっている．

このような中，京コンピュータの後継機である富岳では CPU アーキテクチャとしてこれまでの SPARC64V ではなく，ARM アーキテクチャを採用した A64FX が搭載されている．A64FX では，HPC 向け拡張である SVE（Scalable Vector Extension）により，同じ ARM プロセッサである ThunderX2 と比べて高い SIMD 演算が可能となっている（Skylake Xeon や Xeon Phi KNL と同じ SIMD 幅）．また高バンド幅メモリである HBM2 を搭載するなど，高性能計算に最適化された CPU となっている．現在富岳は 2021 年の運用開始を目指して構築中であるため，まだ一般には利用できない．しかしながら，富岳と同じ CPU を採用した商用計算機である FX1000 と FX700 が富士通からリリースされており，A64FX の評価が可能となっている．

そこで，本研究ではこれまでに様々なスパコンで性能評価を行ってきた電磁流体力学（MHD）シミュレーションコード[3]を用いて，FX700 に搭載されている A64FX の性能評価を行う．MHD シミュレーションは通常の流体シミュレーションに電磁場の効果を考慮したシミュレーションになっており，本性能評価の結果は流体系のアプリケーションに広く応用でき，また，これまでに評価してきた計算機システムの性能と比較することで，現実的な A64FX の計算性能を見積もることも可能と考えられる．

本研究報告の構成は以下の通りである．第 2 章では，A64FX 搭載計算機システム（FX700）について説明し，第 3 章では MHD シミュレーションコードについて説明をする．第 4 章で性能評価の結果を述べ，その結果と他システムとの比較を第 5 章で行い，最後に研究のまとめをする．

2. A64FX 計算機システム FX700

本性能評価で利用する A64FX 搭載計算機システムは，次世代のスパコンシステムの評価を目的として，2020 年 3 月に試験的に京都大学学術情報メディアセンターに導入されたシステムであり，4 ノードの Fujitsu PRIMEHPC FX700 により構成される．FX700 は，富岳の商用機である FX1000 と異なり，次の変更点がある．ノード間通信が Tofu D（FX1000）から InfiniBand（FX700）へ変更，水冷（FX1000）から空冷（FX700）へ変更，A64FX の周波数 2.2GHz（FX1000）から 1.8 または 2.0GHz（FX700）へ変更，アシスタントコアの有無（FX700 は無し）などである．本評価では，4 ノード利用であり，並列性能より単ノード性能に注目しているため，これら FX1000 と FX700 との違いは周波数を除きほとんど影響がない．

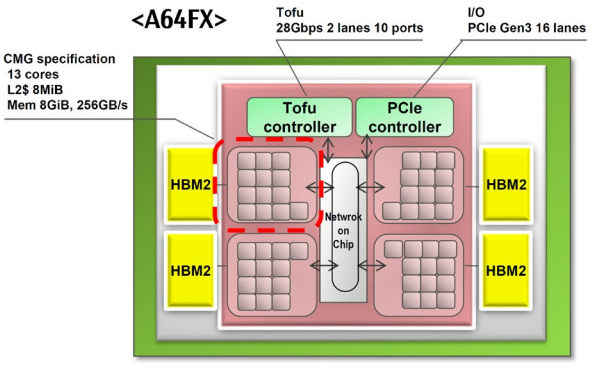

図 1　A64FX の構造[4]
Figure 1　Architecture of A64FX [4]

表 1　A64FX 搭載計算機システムの諸元
Table 1　A64FX Computer System

System	Fujitsu PRIMEHPC FX700	
Node	CPU	A64FX × 1 /node
	Num. of core	48 cores /CPU
	Frequency	1.8 GHz
	Cache	L1：64 KB/core L2：8 MB/CMG
	SIMD	SVE 512bit
	Rpeak	2,765 GFlops /node (DP)
	Memory	HBM2 32 GB /node
	Bandwidth	1,024 GB/s /node
	B/F	0.37
Num. of node	4 nodes	
Rpeak	11 TFlops	
Node comm.	InfiniBand EDR 100 Gbps	

A64FX は図 1 に示されるような 4 つの Core Memory Group（CMG）から構成され，この CMG 毎に HBM2 やアシスタントコア（FX700 では利用不可）が付随する．FX700 では 1 ノードあたり 1CPU，48 コア（理論演算性能 2,765GFlops），32GB メモリ（バンド幅 1024GB/s）からなる．システムの諸元を表 1 に示す．

A64FX は，同時演算数が 32 であり，Skylake Xeon（Gold 以上）や Xeon Phi KNL と同等の同時演算数となっている．一方で，同じ ARM アーキテクチャである ThunderX2 は同時演算数が 8 となっており，理論演算性能が相対的に低くなっている．メモリバンド幅は，高バンド幅メモリである

HBM2 を採用しているため，Xeon などに比べて高く，理論演算性能とメモリバンド幅の比である B/F 値が 0.37 となっている．近年の Xeon 搭載計算機では B/F 値が 0.1 より小さいなど，CPU の性能向上に DDR 系のメモリバンド幅の向上が付いて行けていない．また，ThunderX2 では，理論性能がそれほど高くない一方で，メモリチャンネルが 8 となっているため，B/F 値が 0.30 と Xeon 搭載計算機に比べ大きくなっている．メモリ参照が多いアプリケーションの場合，B/F 値が実効効率を制限するため，注意が必要である．

A64FX の試作プロセッサにおける HPC 系のベンチマーク結果が Skylake Xeon，ThunderX2 との性能比較の形で報告されている[5]. A64FX の特徴である高いメモリバンド幅を必要とするアプリケーションでは，A64FX が高い性能を示し，浮動小数点演算器を多く使うアプリケーションでは，A64FX のハードウェアリソース不足により Skylake Xeon が高い性能を示している．A64FX を効率的に利用するには，このハードウェアリソース不足対策が重要と考えられる．

3. MHD シミュレーションコード

ここでは，本性能評価で利用するアプリケーションである惑星磁気圏 MHD（MagnetoHydroDynamic：電磁流体力学）シミュレーションコードについて説明する．このコードは，地球だけで無く，木星・土星磁気圏シミュレーションにも利用されている実アプリケーションである．

宇宙空間は真空と思われているが，その 99 %はプラズマで満たされている．プラズマとは電離した気体のことであり，帯電している電子とイオンが分かれて存在する状態である．宇宙空間，特に我々の暮らす太陽系においては太陽から太陽風と呼ばれるプラズマの風が常時吹き出しており，太陽系全体にそのプラズマが充満している．この太陽風が惑星の固有磁場と相互作用することで，惑星磁場が変形し，磁気圏と呼ばれる領域が形成される．磁気圏では，磁気嵐など様々な現象が起きており，その領域だけでなく，地上にも様々な形で影響を及ぼしている．このような現象は宇宙天気と呼ばれ，古くから研究が行われている．また，HPCI 萌芽課題の一分野として富岳を利用する数値シミュレーション研究も進められている．

このような宇宙プラズマ現象である宇宙天気を理解するためには，宇宙プラズマの振る舞いを記述する方程式，Vlasov-Maxwell 方程式を解く必要がある．これは，無衝突 Boltzmann 方程式と Maxwell 方程式から成る．Vlasov（無衝突 Boltzmann）方程式は以下の形をとる．

$$\frac{\partial f_s}{\partial t}+\vec{v}\frac{\partial f_s}{\partial \vec{r}}+\frac{q_s}{m_s}(\vec{E}+\vec{v}\times\vec{B})\frac{\partial f_s}{\partial \vec{v}}=0 \qquad (1)$$

ここで $\vec{E}$，$\vec{B}$，$\vec{r}$ と $\vec{v}$ はそれぞれ電場，磁場，距離，速度を表す．また，$f_s(\vec{r},\vec{v}_s,t)$ は位置－速度位相空間における分布関数であり，S はイオンや電子など種類を示す．q_s は電荷を m_s は質量を表す．

Vlasov 方程式はプラズマの振る舞いを正確に記述しているが，多くの成分からなる非線形方程式であり，近年の計算機システムを用いても，磁気圏全体をグローバルに解くことが非常に難しい．例えば，3 次元位置と速度空間をそれぞれ 1000 メッシュ取るとすると，1000^6 メッシュが利用され，8 EB のメモリが必要となる．そこで，Vlasov 方程式のモーメントをとること（速度空間を近似する）で求められる電磁流体力学（MHD）方程式が，グローバルなプラズマ構造を調べるときには使用されている．MHD 方程式は以下のようになる．

$$\begin{aligned}
\frac{\partial \rho}{\partial t}&=-\nabla\cdot(\mathbf{v}\rho)\\
\frac{\partial \mathbf{v}}{\partial t}&=-(\mathbf{v}\cdot\nabla)\mathbf{v}-\frac{1}{\rho}\nabla p+\frac{1}{\rho}\mathbf{J}\times\mathbf{B}\\
\frac{\partial p}{\partial t}&=-(\mathbf{v}\cdot\nabla)p-\gamma p\nabla\cdot\mathbf{v}\\
\frac{\partial \mathbf{B}}{\partial t}&=\nabla\times(\mathbf{v}\times\mathbf{B})
\end{aligned} \qquad (2)$$

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．このため，3 次元空間にそれぞれ 1000 メッシュ取ると，$1000^3\times 8$ 成分となり，64 GB のメモリを必要とする．詳しい導出方法は参考文献を参照されたい[6].

MHD 方程式を解く数値計算法としては，Modified Leap Frog（MLF）法[3，7]という数値計算法を使用する．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．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，本手法で採用する 2 次精度の中心空間差分では，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．本評価で利用する MHD コードは Fortran で実装している．

並列化にはプロセス並列に MPI を使用する．プロセス並列化手法としては 3 次元空間を分割する領域分割法を用いる．領域分割には，1 次元，2 次元，3 次元分割が考えられ，本性能評価ではこれらすべての評価を行う．領域分割の次元数により，MPI 通信に伴うデータのパック/アンパックや通信量自体が変わるため，それぞれの性能評価を行う．

一般的に B/F 値の低いスカラ CPU で性能を出すためにはキャッシュの有効活用が重要である．基本的な動作としてはメモリアクセス時に，その周辺数 KB のデータをキャッシュに格納する．キャッシュの量や，一度にキャッシュに格納するデータ量は CPU アーキテクチャ毎に変わるため，最高のパフォーマンスを出すにはそれぞれの調整が必要である．MHD シミュレーションにおいては，物理変数が

プラズマ密度，速度 3 成分，圧力，磁場 3 成分の計 8 変数となる．そのため，配列を $f(x, y, z, m)$ と定義し，$m = 8$ としている．数値計算時に同じ場所の物理変数を何度も使うことになるため，一般に Fortran では，$f(m, x, y, z)$ と定義した方がキャッシュヒット率は上がることがわかっている [8]．しかしながら，近年の Xeon 系 CPU のように SIMD 幅向上に伴いベクトル化が性能向上にとって重要な機構であるため，更に配列を $f(x, m, y, z)$ と $f(x, y, m, z)$ と定義した場合の性能評価も行う．

4. 計算性能評価

FX700 では，Fortran コンパイラと MPI ライブラリとして Fujitsu Software Compiler Package V1.0L10 を利用し，性能評価を行った．Fujitsu Fortran では，コンパイルオプションに -Kfast （ターゲットマシン上で高速に実行させることを指示する）を指定した．計算サイズは基本的にはプロセス当たり，64MB（100^3 グリッド）となり，Weak scaling の評価である．計測は 5 回行い，その平均値を取った．

図 3 に FX700 を利用して，3 種類（1 次元，2 次元，3 次元）の領域分割を行った MHD コードの評価結果を示す．前述のように，プロセス当たりの計算量がどの領域分割においても $(x, y, z) = (100, 100, 100)$ となるように設定し，ここでは主に領域分割による性能の違いを見ている．並列化としては，Flat MPI を利用している．1 次元領域分割が最も性能が高く，3 次元領域分割の性能が最も低い結果となっている．ThunderX2 の性能評価では，1 次元領域分割と 2 次元領域分割の差はそれほど無かったが（実効効率で 1%未満の差），A64FX では差が大きくなっており，利用コア数によっては，3 次元領域分割と同程度の性能となっている．この評価では，プロセス毎の通信するデータ量は全く同じであり，違いは Halo 通信に伴うパック・アンパック処理とプロセスにおける通信相手の数であり，通信の負荷が見えてくる評価である．現代のスパコンは富岳も含め大並列実行を行うため，1 次元領域分割を選択することは現実的では無いが，今回のように通信による負荷が高次元の領域分割で大きい場合は，3 次元領域分割よりも 2 次元領域分割を選ぶ方が高い実行性能を実現できる場合がある．

実際の性能としては，192 プロセスを利用した場合，1 次元領域分割で 1,004 GFlops（実行効率 9.1 %），2 次元領域分割で 923 GFlops（同 8.4 %），3 次元領域分割で 846 GFlops（同 7.7 %）となっている．

次に，配列の並びによる MHD コードの性能変化を評価した．図 4 は 3 次元領域分割において，前述のように配列の並びが異なる 4 つの場合に MHD コードがどのような性能を示したかを示している．図中の 3D xyzm は，図 3 の 3 次元領域分割の結果と同じであり，xyzm は配列の並びを表

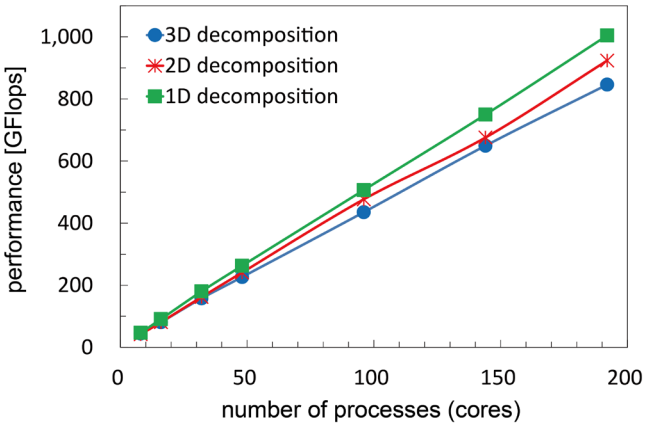

図 3　3 種類の領域分割による MHD コードの性能
Figure 3　Performance of MHD code with three domain decompositions

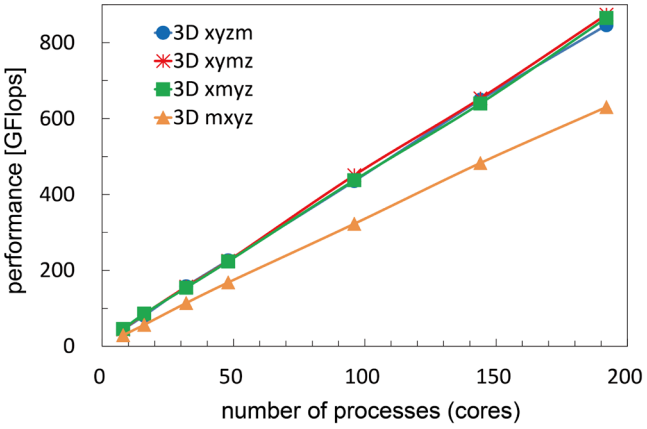

図 4　配列並びが異なる場合の MHD コードの性能
Figure 4　Performance of MHD code with different array order

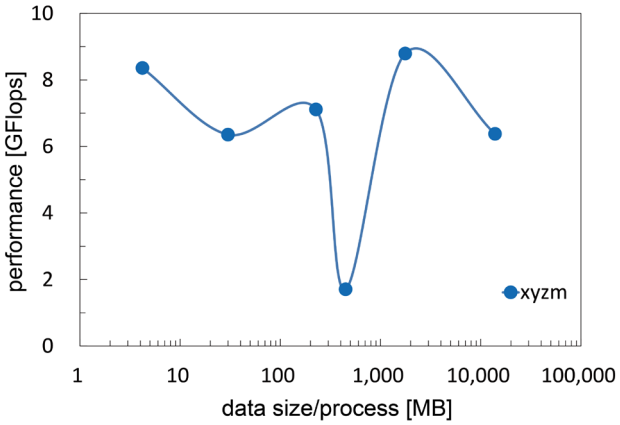

図 5　異なる配列サイズにおける MHD コードの性能
Figure 5　Performance of MHD code with different array data size

している（これは $f(x, y, z, m)$を意味する）．図から明らかにmxyz の配列並びの性能が，他の配列並びと比べて低くなっている．この mxyz は京コンピュータでは最も性能が高い結果を示していた[8]．一番性能が高くなった配列並びはxymz であり，873 GFlops（実行効率 7.9 %）となっている．ほぼ同じ性能で 2 番目となったのは xmyz であり，865 GFlops（同 7.8 %），3 番目が xyzm となった．物理変数を示す m が最内にある，いわゆる AoS（Array of Structure）形式は，これまでの研究結果によりキャッシュヒット率が高くなる配列構造であり[7]，京コンピュータには最適であったが，高 SIMD 計算が必要な A64FX では，ベクトル計算に効率的な SoA（Structure of Array）形式が重要であり，二つを合わせた xyzm や xmyz の性能が高くなっていると考えられる．

最後に，計算に利用する配列のサイズによる FX700 の性能を評価した．これは非並列で xyzm の配列並びを利用し，基本的な性能を見ている．図 5 では，1 プロセスでプロセス当たりの総配列利用サイズ（ワーク配列を含む）を 4.2～13,878 MB と変化させた場合の MHD コードの性能を示している．A64FX の L2 キャッシュサイズは 8 MB/CMG であり，最小の配列サイズでは L2 のサイズを下回っている．

最も性能が高いサイズは 1,760 MB（計算サイズは 200×200×200）であり，8.79 GFlops（実効効率 15.3 %）となり，オンキャッシュの 4.2 MB（計算サイズは 25×25×25）の 8.36 GFlops（同 14.5 %）を上回った．他のサイズは，一番サイズが小さいオンキャッシュサイズより性能が低く，キャッシュの効果が見えている．A64FX は高 SIMD 幅計算を効率的に利用することで性能が高くなるため，ベクトル長がある程度長く，MHD 変数含めて配列の 1 次元目がすべてキャッシュに載るサイズがベクトル計算とキャッシュヒットのバランスが良く性能が高くなったと推測される．図 5 で明らかに性能が悪い計算サイズが 1 つあるが，これは計算サイズが 128×128×128 の時であり，バンクコンフリクトが起きているため，性能が極端に劣化している．

この計算サイズが異なる評価から，MHD コードでは，計算配列の 1 次元がある程度大きくベクトル計算が効率的に行え，1 次元目と MHD 変数がオンキャッシュの場合に性能が高くなることがわかる．これは図 4 の配列並びにより性能が変化する理由となっており，xymz や xmyz という並びがこの条件を満たしていると考えられる．

最後に OpenMP を利用したハイブリッド MPI 並列の性能を評価する．1 ノード 48 コアを利用し，8 プロセス×6 スレッドと 4 プロセス×12 スレッド実行を，配列並びが異なる 4 つの場合に行った．計算サイズはどの並列計算においても 300×400×400 となっている．図 6 に評価結果を示す．図 4 の Flat MPI 実行時の 48 コア利用結果も比較のために加えている．

A64FX は 1CMG に 12 コアの構成のため，1 ノードあたり 4 プロセス×12 スレッド実行が推奨されている．京や FX10 などでも同様の推奨であったが，実測では Flat MPI の性能が高くなることがこれまでの評価で分かっている[7]．A64FX では，すべての配列並びにおいて，ハイブリッド MPI の性能が Flat MPI より高くなっている．図 3 の並列計算負荷結果から，プロセス数が 48 と大きい Flat MPI 利用時の性能が低くなることが理解できる．一方で，12 スレッドと 6 スレッドのどちらの性能が高くなるかは，配列並びによって異なる結果となった．xyzm と mxyz では，6 スレッド（プロセス数が多い場合）で性能が高くなり（273 GFlops， 189 GFlops），xymz と xmyz では，12 スレッド（プロセス数が少ない場合）の性能が高くなった（276 GFlops，258 GFlops）．xyzm の結果は明らかな違いがあり，プロセス数の影響よりスレッド数が多くなる影響が大きいと考えられる．スレッド並列は，z 次元に対して行っているが，xyzm の場合には，m が z の外側にあるため，12 スレッド並列時にキャッシュの利用などに影響がある可能性が考えられる．

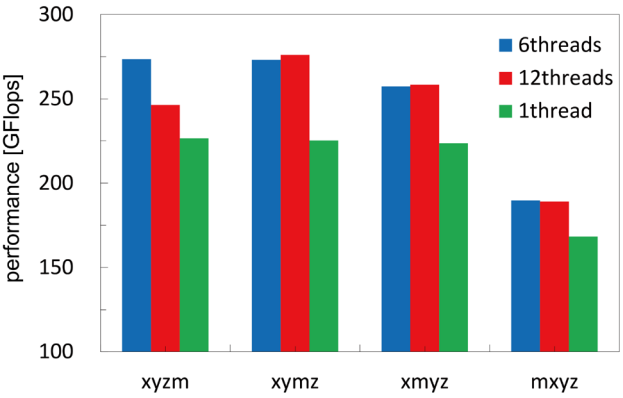

図 6　異なる配列サイズにおける MHD コードのハイブリッド MPI 並列性能

Figure 6　Hybrid MPI parallel computing performance of MHD code with different array data size

5. A64FX に対する最適化

これまでの A64FX に対する MHD コードの性能評価から，xymz 配列並びを利用し，1 ノードあたり 4 プロセス×12 スレッド並列実行が最も性能が高い結果となった（276GFlops，実行効率 10 %）．ここでは，いくつかの最適化をこの結果に対して行い，性能向上への効果を調べる．FX700 には，性能プロファイラとして FX100 などでも利用可能な CPU 性能解析レポートが利用できる．詳細レポートの場合は，17 回実行する必要があるが，取得可能なプロファイラ情報を 1 つのレポートとして出力可能であり，最適化を行う際に役立つ．このレポートから得られた図 7 のような Busy rate を見ると，L2 とメモリが効率よく利用され

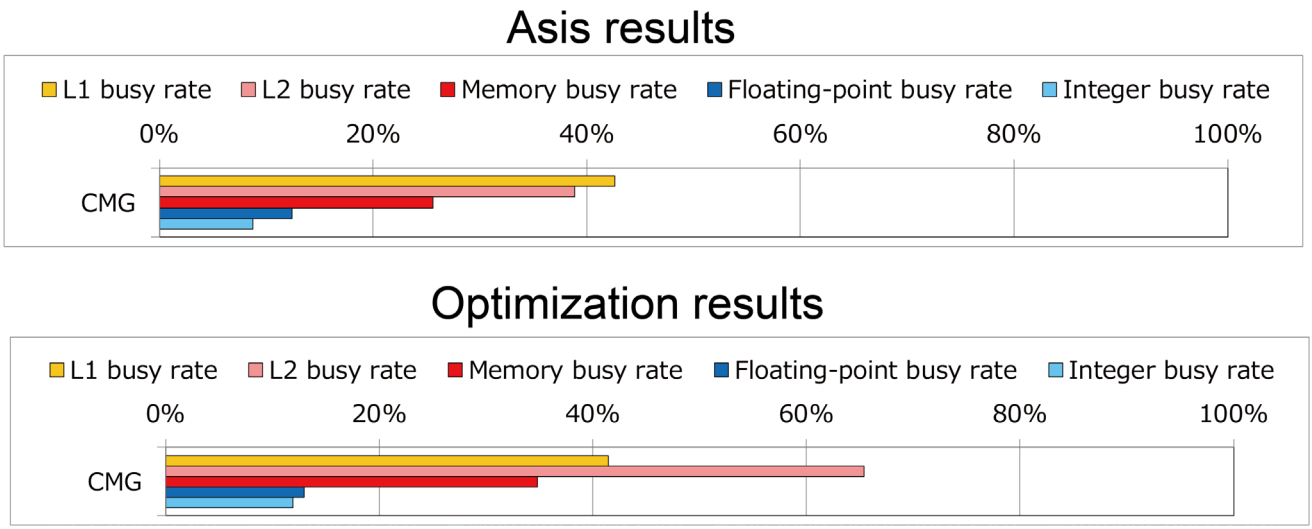

図 7　最適化前と後における Busy rate

Figure 7　Busy rate of before and after optimization

ていないことが分かった（Asis results）．また，前述の通り A64FX では，レジスタ数が多くないためにループを分割することが性能に効いてくる．そこで，キャッシュ・メモリ利用率向上とループ分割を行う．

キャッシュの効果を見た図 5 では，配列サイズを各次元等しく設定していた（例えば 100×100×100）．ここでは配列サイズを一定にし，各次元のサイズを変更することで，ベクトル計算性能とキャッシュ利用効率の向上を見込み，性能評価を行った．表 2 にあるように 5 種類の配列サイズパターン（総配列サイズは一定）を利用した．評価結果は表 2 にあるように約 239 GFlops～321 GFlops と大きく性能の幅が出た．ベクトル計算のために，x 次元のサイズはある程度大きい必要があり，(200, 400, 300)と(100, 400, 600)では性能が出ていないと考えられる．また，図 5 の結果とお同じように x と y 次元までキャッシュに載るようなサイズが性能に効いてくるため，(600, 400, 100)は性能がそれほど高くなっていないと考えられる．うまくバランスが取れているのが，(300, 200, 400)と(600, 200, 200)と考えられ，性能が高くなっている．この結果，16%の性能向上となった．

Fujitsu コンパイラにはループ分割を行うコンパイルオプションがいくつかある．基本的には-Kloop_fission という自動ループ分割オプションをコンパイル時に付けることによりにより，コンパイラによるループ分割が行われる．この自動ループ分割により，295 GFlops と約 7 %の性能向上となった．また，この自動ループ分割をどのように行うかを指定できる-Kloop_fission_threshold=n というオプションがあり，コードに指示文を挿入する必要はあるが，ハンドコーディングより容易に様々なループ分割を行うことができ，最適化が行える．しかしながら，本研究で利用したコンパイラでは非対応だったのか（コンパイルエラーは出ないので，オプションは有効），このオプションはコンパイル後の出力リストを見ても効果が無かった．ループ分割をハンドコーディングで行うには手がかかるため，今後の課題とする．

表 2　各次元の計算サイズを変更した MHD コードの性能

Table 2　Performance of MHD code with changing the calculation size in the each dimension.

Size (x, y, z)	Performance [GFlops]
200, 400, 300	239.2
100, 400, 600	243.9
300, 200, 400	311. 1
600, 200, 200	321.1
600, 400, 100	302.4

次に，ループ分割同様にレジスタ利用量を減らす効果を期待し，ループ内の計算に利用されるいくつかの変数をループ外で計算させ，配列に収納し，ループ内の計算時に参照できるような最適化を行った. 図 8 にコードの例を示す. 図中で赤字になっている箇所が変更部分になる．MHD コードでは，MHD 変数の時間発展を差分で計算する際に，式(2)にあるように発散（*div*）・回転（*rot*）・勾配（*grad*）を計算する必要があり，これらは差分式内で計算されていた．このため，ループ内の計算が多くなり，レジスタ不足に陥る可能性が高くなる．また，実際の速度ベクトルの時間発展時には，重力を考慮した項が計算に含まれ，その項の計算時に惑星からの距離が必要となる．この距離はループ内で計算していたが，単純計算のためレジスタ不足回避を狙い，ループ外で計算し配列参照に変更した．これらをコードに施した結果, 336.7 GFlops の性能を得た. これは約 22 %の性能向上となった.

最後にキャッシュ利用効率を更に上げるために，プリフ

Asis code

```
f(i,j,2,k) = ff(i,j,2,k) + t * (
      ( p(i,j,10,k) * p(i,j,8,k)
      - p(i,j,11,k) * p(i,j,7,k)) / p(i,j,1,k)
      - gra * x / ra3)
      - dx2 * p(i,j,2,k) * (
        u(i,j,2,k) + u(i,j-1,2,k)
      + u(i,j,2,k-1) + u(i,j-1,2,k-1)
      - u(i-1,j,2,k) - u(i-1,j-1,2,k)
      - u(i-1,j,2,k-1) - u(i-1,j-1,2,k-1) )
      - dy2 * p(i,j,3,k) * (
        u(i,j,2,k) - u(i,j-1,2,k)
      + u(i,j,2,k-1) - u(i,j-1,2,k-1)
      + u(i-1,j,2,k) - u(i-1,j-1,2,k)
      + u(i-1,j,2,k-1) - u(i-1,j-1,2,k-1) )
      - dz2 * p(i,j,4,k) * (
        u(i,j,2,k) + u(i,j-1,2,k)
      - u(i,j,2,k-1) - u(i,j-1,2,k-1)
      + u(i-1,j,2,k) + u(i-1,j-1,2,k)
      - u(i-1,j,2,k-1) - u(i-1,j-1,2,k-1) )
      - dx2*(u(i,j,5,k) + u(i,j-1,5,k)
      + u(i,j,5,k-1) + u(i,j-1,5,k-1)
      - u(i-1,j,5,k) - u(i-1,j-1,5,k)
      - u(i-1,j,5,k-1) - u(i-1,j-1,5,k-1) )
      / p(i,j,1,k)
      + vmu * (
        dx4 * (ff(i+1,j,2,k)-2.0*ff(i,j,2,k)+ff(i-1,j,2,k))
      + dy4 * (ff(i,j+1,2,k)-2.0*ff(i,j,2,k)+ff(i,j-1,2,k))
      + dz4 * (ff(i,j,2,k+1)-2.0*ff(i,j,2,k)+ff(i,j,2,k-1))
       ) / ff(i,j,1,k)
```

Optimization code

```
f(i,j,2,k) = div2(i,j,2,k) + t * (
         ( p(i,j,10,k) * p(i,j,8,k)
         - p(i,j,11,k) * p(i,j,7,k)) / p(i,j,1,k)
         - gra * x(i) / ra3(i,j,k))
         - dx2 * p(i,j,2,k) * (
           u(i,j,2,k) + u(i,j-1,2,k)
         + u(i,j,2,k-1) + u(i,j-1,2,k-1)
         - u(i-1,j,2,k) - u(i-1,j-1,2,k)
         - u(i-1,j,2,k-1) - u(i-1,j-1,2,k-1) )
         - dy2 * p(i,j,3,k) * (
           u(i,j,2,k) - u(i,j-1,2,k)
         + u(i,j,2,k-1) - u(i,j-1,2,k-1)
         + u(i-1,j,2,k) - u(i-1,j-1,2,k)
         + u(i-1,j,2,k-1) - u(i-1,j-1,2,k-1) )
         - dz2 * p(i,j,4,k) * (
           u(i,j,2,k) + u(i,j-1,2,k)
         - u(i,j,2,k-1) - u(i,j-1,2,k-1)
         + u(i-1,j,2,k) + u(i-1,j-1,2,k)
         - u(i-1,j,2,k-1) - u(i-1,j-1,2,k-1) )
         - dx2*(u(i,j,5,k) + u(i,j-1,5,k)
         + u(i,j,5,k-1) + u(i,j-1,5,k-1)
         - u(i-1,j,5,k) - u(i-1,j-1,5,k)
         - u(i-1,j,5,k-1) - u(i-1,j-1,5,k-1) )
         / p(i,j,1,k)
```

図 8　ループ内変数計算部における最適化前後
Figure 8　Code of before and after optimization in the loop

エッチを促進するコンパイルオプションを追加した．プリフェッチ関するコンパイルオプションは効果の違いから多くのオプションが用意されている．その中で，prefetch_infer, prefetch_line_L2=8，prefetch_stride の 3 つを利用した．prefetch_infer の有無では性能に変化が現れず，追加の効果は得られなかった．prefetch_line_L2=8，prefetch_stride は性能向上が見られ，特に prefetch_line_L2=8 では，8%程度の性能向上があり，prefetch_line_L2=8，prefetch_stride 両方ともオプションとして付与すると，303.7 GFlops の性能（約 10 %の性能向上）を得ることができた．

基本的に MHD シミュレーションコードは流体コードであり，格子系流体差分計算のため，本研究で得られた最適化の知見は，同様の流体計算にも広く効果があると考えられる．

これまでにそれぞれ効果があった最適化を，すべて行い実行すると，431.7 GFlops（実行効率 15.7 %）と，1CPU で元のコードと比べ 155 GFlops（実行効率で 5 %）の性能向上となった．図 7 にあるように最適化後は，L2 の利用率が 6 割を超え，メモリバンド幅も 35 %の利用率に向上している．しかしながら，基本的に MHD シミュレーションがメモリバンドの計算（高 B/F 計算）であることを考えると，メモリバンドの利用率が足りていない．FX700 の B/F 値が 0.37 であるため，まだ最適化の余地がある．CPU 性能解析レポートによると，Floating-point L2 cache miss や Floating-point operation wait，Floating-point mem wait の時間が多く見える．これらの時間を削減するために，ループ分割を含めた最適化が必要と考えられる．

6. 他計算機システムとの比較

今回評価をした A64FX 搭載計算機 FX700 での MHD コードの性能を他の計算機システムでの MHD コードの性能と比較することで，A64FX の相対的な性能や各計算機の実性能を理解することに繋がると考えられる．表 3 にこれまで MHD シミュレーションコードの性能を計測したいくつかの計算機システムの結果と FX700 の測定結果を示す[2, 7, 10, 11]．今回の性能評価では 3 次元領域分割において，4 種類の配列構造を利用したが，ITO-A と ThunderX2 の評価以外はいわゆる SoA（xyzm）と AoS（mxyz）を利用した測定しかしていない．また，CPU（GPU，コプロセッサ）自体の性能を比較しやすいように，CPU 当たりの性能（Rmax/CPU）を表に加えている．

FX700 の結果は，最適化をすべて加え，4 ノードで実行した結果を記載している．そのため，CPU 当たりの性能では，1 ノード実行に比べ性能がわずかに下がっている．今回はシステム全体の性能では無く，CPU 単体の性能比較に興味があるため，Rmax/CPU に注目する．A64FX は表中のどの CPU/GPU よりも高い性能を示しており，Skylake Xeon

表 3　様々な計算機システムにおける性能の傾向[2，7，10，11]
Table 3　Performance trend of various computer systems [2，7，10，11]

	Core/CPU	Rmax [TFlops]	Rpeak [TFlops]	Rmax /CPU [GFlops]	Efficiency [%]	Suitable domain decomposition	CPU architecture
SX-ACE	1024/256	29.20	65.50	114.0	45	3D xyzm	Vector
K	262144/32768	914.12	4194.30	27.9	22	3D mxyz	SPARC64 VIIIfx
FX100	16384/512	91.49	576.72	178.7	17	3D xyzm	SPARC64 XIfx
CX400	23616/2952	104.23	510.11	35.3	20	3D xyzm	Xeon (SandyBridge)
HA8000	23160/1930	83.42	500.26	43.2	17	2D xyzm	Xeon (IvyBridge)
XC30	448/32	1.37	16.49	42.8	8	2D xyzm	Xeon (Haswell)
ITO-A	72000/4000	470.10	6912.00	117.5	7	1D xyzm	Xeon (Skylake)
XC40	1088/16	4.32	48.86	273.3	9	3D xyzm	Xeon Phi KNL
Tesla K20X	896/1	0.15	1.31	153.3	12	3D xyzm	Kepler
ITO-B	3584/1	0.38	5.30	382.2	7	3D_xyzm	Pascal
ThunderX2	256/8	0.70	4.50	86.9	16	3D mxyz	Arm v8
FX700	192/4	1.70	11,06	425.5	15	3D xymz	A64FX

の 3.6 倍, Xeon Phi KNL の 1.6 倍程度の性能となっている．Xeon は 2 ソケット/1 ノードを考えてもノードあたり 1.8 倍の性能となった．GPU と比べると，Pascal Tesla と比べ，1.1 倍，Kepler Tesla の 2.8 倍の性能となった．また，京コンピュータの SPARC64 VIIIfx と比べて，15 倍の性能向上となった．

実行効率では，ベクトル機である SX-ACE が飛び抜けて高いが，A64FX は ThunderX2 や FX100，Ivy Bridge Xeon と同程度となっている．これらは SIMD 幅が A64FX より低いことを考えると，A64FX は高い実行効率を示していると考えられる．近年は同時演算回数の増加により理論性能を高くし，低い実行効率を示す CPU が多いが A64FX はそれらと比較して実利用向けの CPU と考えられる．

7. まとめ

京都大学に試験的に導入された A64FX 搭載計算機 FX700 に対して，宇宙プラズマを解く MHD シミュレーションコードの性能測定を行った．3 種類の領域分割を行った結果は，高次元領域分割の方がパック/アンパックや通信に係わる時間により，性能が低くなる傾向が見えた．異なる配列の並びによる MHD コードの A64FX での性能を評価では，キャッシュヒット率の高い AoS 形式の配列形状の性能が他の配列形状に比べ明らかに性能が低くなった．その他はそれほど大きな差は無いが，ベクトル計算を効率的に行うために x が最内に，キャッシュヒットのために m が z よりも内側になる形状が好ましい結果となった．次に，キャッシュの効果を調べるために，配列サイズを小さくして，性能を評価したところ，配列形状と同じ理由で，大きすぎるサイズでは性能が劣化するが，すべてがオンキャッシュで無くともある程度の性能が出ることが分かった．また，ハイブリッド MPI 並列実行におけるスレッド数の変化による性能変化を調べたところ，Flat MPI 実行よりもハイブリッド MPI 並列実行の方が高い性能を示した．スレッド数による性能変化は配列形状に依存するが，基本的には 12 スレッド×4 プロセス/CPU の実行が A64FX には適していた．

更に高い性能を目指し，いくつかの A64FX に対する最適化を MHD コードに施した．配列の各次元におけるサイズの変更，自動ループ分割，ループ内変数のループ外での計算，プリフェッチの促進を行うと，1CPU で 155 GFlops の性能向上がえられた．これにより，キャッシュの利用率やメモリバンド利用率の向上が見られたが，MHD コードの B/F 値と FX700 の B/F 値を考えると，まだ最適化の余地があることが分かった．

今回の評価結果を，これまでに性能評価した計算機システムと比較したところ，A64FX（1CPU）の性能は，SKL Xeon の 3.6 倍，Xeon Phi KNL の 1.6 倍の性能となっていた．GPU と比べても，Pascal Tesla の 1.1 倍，Kepler Tesla の 2.8 倍の性能となった．1.1 倍ではあるが，40 GFlops 以上の性能差となっている．実行効率で見ると，ThunderX2 や Ivy Bridge Xeon，FX100 と同程度であり，高 SIMD 幅 CPU としては高い実行効率を示した．

今回は，ループ分割に対する追加の最適化を行えなかった．ループ分割は A64FX を利用する上では，重要な最適化であるため，今後の課題である．

謝辞　本研究は，JSPS 科学研究費 18H03249，18K11336

の助成による．

参考文献

[1] 辻 美和子, Jean-christophe Weill, Jean-philippe Nomine，佐藤 三久，ThunderX2 Arm プロセッサにおける Fiber ミニアプリスイートの性能評価，情報処理学会研究報告，2019-HPC-171(4)，1-8，2019.
[2] 深沢圭一郎，惑星磁気圏 MHD シミュレーションコードによる ThunderX2 ARM プロセッサの性能評価，情報処理学会研究報告, 2019-HPC-172(1), 1-6, 2019.
[3] Ogino. T, R. J. Walker, M. Ashour-Abdalla, "A global magnetohydrodynamic simulation of the magnetopause when the interplanetary magnetic field is northward", IEEE Trans. Plasma Sci. vol. 20, 1992, 817-828.
[4] Yoshida, T., Fujitsu High Performance CPU for the Post-K Computer, Hot Chips 30, 2018.
[5] 小田嶋 哲哉，児玉 祐悦，辻 美和子，松田 元彦，丸山 豊，佐藤 三久，HPC ベンチマークプログラムによる A64FX プロセッサ試作機の性能評価，情報処理学会研究報告，2019-HPC-173(17)，1-8，2020.
[6] F. F. Chen, 1974. Introduction to Plasma Physics. Plenum Press, NY.
[7] Fukazawa, K., T. Nanri and T. Umeda, "Performance Measurements of MHD Simulation for Planetary Magnetosphere on Peta-Scale Computer FX10", Parallel Computing: Accelerating Computational Science and Engineering (CSE), Advances in Parallel Computing 25, pp.387-394, IOS Press, 2014. (DOI: 10.3233/978-1-61499-381-0-387)
[8] Fukazawa, K., T. Ogino, and R. J. Walker (2012), "A Magnetohydrodynamic Simula-tion Study of Kronian Field-Aligned Currents and Aurora", J. Geophys. Res., 117, A02214, doi:10.1029/2011JA016945.
[9] Arm Allinea Studio Web site (https://developer.arm.com/tools-and-software/server-and-hpc/arm-architecture-tools/arm-allinea-studio).
[10] Fukazawa, K., T. Soga, T. Umeda, T. Nanri, Performance Evaluation and Optimization of MagnetoHydroDynamic Simulation for Planetary Magnetosphere with Xeon Phi KNL, Parallel Computing is Everywhere: Accelerating Computational Science and Engineering (CSE), Advances in Parallel Computing, 178 - 187, DOI:10.3233/978-1-61499-843-3-178, 2018.
[11] 深沢圭一郎，南里豪志，本田宏明，スーパーコンピュータシステム ITO における MHD シミュレーションコードの計算性能・消費電力評価，情報処理学会研究報告, 2018-HPC-166(4), 1-6, 2018.