2.8 核融合プラズマ流体解析コード GT5D および GKV の性能測定

日本原子力研究開発機構　井戸村　泰宏、伊奈　拓也、朝比　祐一
名古屋大学　前山　伸也
富士通株式会社　三吉　郁夫、千葉　修一

2.8.1 はじめに

核融合プラズマ流体解析コード GT5D および GKV は磁場閉じ込め型の核融合炉における炉心プラズマの閉じ込め性能を決定する乱流輸送現象の評価を目的として開発された。核融合プラズマの第一原理モデルは 5 次元位相空間（3 次元空間×2 次元速度）におけるプラズマ粒子分布の発展を記述する移流・拡散方程式（ボルツマン方程式）で与えられる（スライド p3、以降、ページ番号のみ表記する）。核融合プラズマ流体解析ではこの方程式を 5 次元格子上で離散化して CFD スキームを適用するため、3 次元流体モデルに比べて～100^2倍程度大きい自由度の流体計算となる。「京」を用いてようやく ITER 規模の炉心プラズマの数値実験を水素、あるいは、重水素のみの単一イオン種のイオン系乱流に対して行うことが可能になったが、電子系乱流を含む数値実験、あるいは、重水素と三重水素を燃料とする核融合反応によってヘリウムが生成される多種イオン系の核燃焼プラズマの数値実験を行うには将来のエクサスケールマシンが必要となる。このような大規模計算を実現する上で最新のメニーコアプロセッサによって構成される超並列環境の利用技術の高度化が必要不可欠となっている。本性能測定では、特に、FX100 において強化されたキャッシュ機構や新たに機能拡張されたアシスタントコアの利用技術に焦点を当てて最適化技術の開発およびその有効性検証を行った。

2.8.2 プログラム概要

GT5D（p4）は炉心プラズマ全体を計算領域としてプラズマ乱流やプラズマ分布の大域的な非線形発展を追跡する。この特徴を活かして、これまで非局所的な乱流輸送現象の研究や乱流輸送の装置サイズスケーリングの研究が行われてきた。計算手法としては 5 次元移流・拡散方程式に差分法を適用し、時間積分に半陰解法を用いている。移流項を陰的に計算する差分・陰解法カーネルは共役残差法によって実装されており、コードの主要コストを占める。並列化手法としては MPI と OpenMP を用いるハイブリッド並列モデルを採用している。さらに、OpenMP によって実装した通信スレッドによる通信隠蔽処理、階層的な多次元領域分割モデル、および、Tofu ネットワーク向けのランクマップ最適化を実装することによって、「京」全系に至る強スケーリングを達成し、約 1000 億格子を用いる ITER 規模の数値実験が可能になった。本報告では GT5D の差分・陰解法カーネル（GT5D カーネル）を対象として FX100 における性能測定を実施する。

GKV（p5）は一本の磁力線に沿った局所領域を計算領域として高解像度のプラズマ乱流計算を低コストで行う。この特徴を活かして、これまで多くのパラメータスキャンを必要とする実験データ解析やイオンスケールから電子スケールにおよぶマルチスケール乱流の研究が行われてきた。計算手法としては 5 次元移流・拡散方程式の磁力線に垂直な 2 次元方向に擬スペクトル法、それ以外の 3 次元方向に差分法を適用し、時間積分に陽解法を用いている。非線形項を擬スペクトル法で計算するカーネルでは 5 次元データに対して並列化 2 次元 FFT を適用するため、高コストの転置通信処理が必要となる。並列化手法としては MPI と OpenMP を用いるハイブリッド並列モデルを採用している。さらに、GT5D と同様に OpenMP によって実装した通信スレッドによる通信隠蔽処理、階層的な多次元領域分割モデル、および、Tofu ネットワーク向けのランクマッ

プ最適化を実装することによって、「京」全系に至る強スケーリングを達成し、約 6000 億格子を用いるマルチスケール電子乱流計算が可能になった。本報告では GKV のスペクトル法カーネル（GKV カーネル）を対象として FX100 における性能測定を実施する。

2.8.3 通信と演算のオーバーラップ

「京」全系の強スケーリングを実現する上で通信と演算をオーバーラップする通信マスク手法が鍵となる技術となっている。p4 に示す例では差分演算、および、2 次元 FFT の並列処理における通信オーバーラップの事例が示されている。差分演算の例では、通常は境界データを通信してから差分演算を行うが、通信オーバーラップの場合には、差分演算を境界データを参照する周辺領域とそれ以外の中心領域に分離し、後者の背後で境界データの通信処理を同時実行する。この事例では、差分演算を中心領域と周辺領域で分離したが、これ以外にも異なる次元方向に分離する、あるいは、異なる項毎に分離するといったバリエーションが考えられる。一方、2 次元 FFT の例では、通常は全データの転置処理を実行してから 1 次元 FFT を実行する処理を繰り返すが、通信オーバーラップの場合には、まず、xy 方向の 2 次元 FFT 処理を z 方向（GKV カーネルの場合には残りの 3 次元方向）に分割する。次に、異なる z 方向位置の FFT と転置処理を同時実行する通信オーバーラップをパイプライン的に繰り返すことで、高コストの転置通信を隠蔽する。

上記の通信オーバーラップを実装する手法として「京」では p8 に示す通信マスク手法を開発している。この手法では OpenMP の MASTER 構文によってマスタースレッドを通信スレッドとして使用し、残りのスレッドに演算処理を動的に割り当てる。この実装によって、通信処理の完了後にマスタースレッドも演算に復帰させ、スレッド間の負荷分散を最適化することが可能となる。この手法は n スレッドを用いて通信処理のオーバーヘッドを 1/n に削減できるため、今後のメニーコア環境において非常に有望な技術である。また、通信マスク手法は標準的な MPI と OpenMP の機能のみで実装可能であり、非同期 1 対 1 通信のみならず、同期通信や集団通信も処理できるため、極めて汎用性が高い。

一方、p9 に示すように、FX100 ではアシスタントコアを用いる新たな通信オーバーラップが利用可能である。この機能を用いると、ソースコードの書き換えなしに、オーバーラップ区間をディレクティブで指定することによって、対象とする演算処理の背後で p10 に示す MPI-3.0 の非同期通信処理をアシスタントコアで同時実行することができる。以下では GT5D カーネルの境界データ通信と GKV カーネルの転置通信を対象にこの機能の有効性を検証する。

2.8.3.1 GT5D カーネルにおける境界データ通信のオーバーラップ

まず、GT5D カーネルを用いて通信オーバーラップの効果を調べた。このカーネルは境界データ通信を含む xyzv 方向の 4 次元差分演算を行うが、領域分割は xy 方向の 2 次元方向のみとなっている。このため、差分演算のループを zv 差分演算と xy 差分演算に分離し、zv 差分演算の背後で境界データ通信を実行し、その後で xy 差分演算を実行する。通信処理は p12 に示す 5 パターンの実装を用意した。

- original 版：MPI_Isend/Rrecv/Waitall を連続して実行し、通信の同期をとってから差分演算を実行。
- overlap 版：マスタースレッドのみが MPI_Isend/Rrecv/Waitall を実行し、他は演算処理を実行。

- unroll 版：32 スレッドの FX100 で十分なループ長を確保するために外側 2 重ループを 1 重化。
- unroll+overlap 版：unroll 版でオーバーラップを実装し、演算負荷バランスを改善。
- 通信スレッド解除版：unroll+overlap 版において MASTER 構文を削除し、MPI_Isend/Recv、および、MPI_Waitall をそれぞれ演算処理の前後に発行。さらに、スレッドスケジューリングを DYNAMIC から STATIC に変更。

FX100 における性能測定結果を p13 に示す。ここで、処理時間はマスタースレッドで計測した境界データコピー（copy 区間）、xy 差分演算（kernel 区間）、通信処理（waitall 区間）、および、zv 差分演算（kernel_p 区間）の内訳を示す。マスタースレッドを通信スレッドとして使用する overlap 版、および、unroll+overlap 版では zv 差分演算と通信処理の同時実行により kernel_p 区間がマスタースレッド上で縮小する。一方、アシスタントコアを用いる通信スレッド解除版（通信促進あり）では通信処理の大部分がマスタースレッドからアシスタントコアに移行するため、waitall 区間が縮小する。overlap 版では 16 スレッド（1CMG）に対し、OpenMP を適用している最外の x 方向のループ長が 32 となるのに対し、unroll+overlap 版では xy 方向のループ長が 1024 となり、スレッド間の動的負荷分散のバランスが改善する。通信スレッド解除版では、アシスタントコアを用いない場合には unroll 版とほぼ同じコスト分布となるが、アシスタントコアを用いる通信促進を適用することにより、従来の通信マスク手法を上回る性能を達成した。p14 に示す kernel_p 区間の詳細プロファイラでは、unroll+overlap 版ではマスタースレッドの演算数はほぼゼロとなり、通信処理に専有されているが、通信スレッド解除版（通信促進あり）では、マスタースレッドも含めてほぼ均等な演算量となっていることがわかる。しかしながら、MPI 起動処理とアシスタントコアへの移行処理のオーバーヘッドが残っており、静的スレッドスケジューリングを行った場合には約 5%程度の遅延がマスタースレッドで発生している。このため、アシスタントコアを利用する場合でも動的スケジューリングを用いてスレッド間の負荷分散を最適化する必要があることに注意が必要である。

2.8.3.2 GKV カーネルにおける転置通信のオーバーラップ

次に、GKV カーネルを用いて通信オーバーラップの効果を調べた。このカーネルは k_xk_yzvw 空間の 5 次元データに対して k_xk_y 波数空間の 2 次元逆フーリエ変換を実行し、擬スペクトル法によって xy 実空間で非線形項を計算した後で非線形項を再び k_xk_y 波数空間に 2 次元順フーリエ変換する。この処理を w 方向にパイプライン化し、w 方向のインデックスの異なる転置通信と FFT 処理、および、非線形項計算をオーバーラップする。通信処理は p16 に示す 5 パターンの実装を用意した。

- オーバーラップなし MPI_Alltoall 版：MPI_Alltoall を用いる転置通信と FFT 処理を逐次的に実行するオリジナル版。
- オーバーラップあり MPI_Alltoall 版：FFT 処理の背後でマスタースレッドが MPI_Alltoall を用いる転置通信を同時実行。
- オーバーラップなし MPI_Ialltoall 版：MPI-3.0 の非同期集団通信 MPI_Ialltoall を用いる転置通信と FFT 処理を逐次的に実行。
- オーバーラップあり MPI_Ialltoall 版：FFT 処理の背後でアシスタントコアが MPI_Ialltoall を用いる転置通信を同時実行。
- オーバーラップあり MPI_Ialltoall、静的スレッドスケージュール版：オーバーラップあり MPI_Ialltoall 版においてスレッドスケジューリングを DYNAMIC から STATIC に変更。

FX100 における性能測定結果を p17 に示す。ここで、処理時間はマスタースレッドで計測した逆フーリエ変換用の転置通信（Comm.(bwd)）、FFT（Comp.(bwd)）、順フーリエ変換用の転置通信（Comm.(fwd)）、FFT（Comp.(fwd)）、および、非線形項処理等（Others）の内訳を示す。マスタースレッドを通信スレッドとして使用するオーバーラップあり MPI_Alltoall 版ではマスタースレッドが転置通信を実行するため FFT の処理コストが縮小する。一方、アシスタントコアを用いるオーバーラップあり MPI_Ialltoall 版では通信コストの一部がマスタースレッドからアシスタントコアに移行しマスタースレッドから見えなくなる。測定結果では通信マスク手法が最も良い性能を示しているが、非同期集団通信とアシスタントコアの利用でも同程度の性能向上が確認できた。アシスタントコア利用によるスレッド間の負荷分散の均等化を期待して、オーバーヘッドの少ない静的スレッドスケジューリングを適用した結果、性能が低下した。また、オーバーラップあり版では、スレッド同期の回数が減少するため、スレッド間の負荷バランスが改善し、Others 区間のコストが減少している。

より詳細なコスト分析を p18 に示す。通信コストを比較した結果、オーバーラップあり/なし、MPI_Alltoall/MPI_Ialltoall、全ての組み合わせでほぼ同等の通信コストとなることがわかった。インターコネクトのバンド幅とメモリバンド幅は一桁以上異なることから、通信オーバーラップ適用時の演算処理による通信性能への影響は限定的である。また、MPI_Ialltoall では Tofu 向け高速アルゴリズムが利用できないが、今回テストを行った小規模ノード数では性能劣化は少ないことがわかった。MPI-3.0 の MPI_Ialltoall は非同期集団通信であるが、MPI_Isend/Irecv と同様にアシスタントコアを用いないと非同期に動作しないことに注意が必要である。アシスタントコアを用いる場合には通信コストが約 3/1 に減少するが、MPI_Ialltoall の起動とアシスタントコアへの処理移行のオーバーヘッドがかなり残っている。

スレッド間の負荷バランスを p11 に示す。オーバーラップなしの場合にはマスタースレッドの通信処理の間、残りのスレッドにかなり大きいバリア同期待ちのコストが見られる。オーバーラップありの場合にはスレッド間の負荷分散が改善し、バリア同期待ちのコストが大幅に減少する。しかしながら、アシスタントコアを利用して静的スレッドスケジューリングを適用すると、MPI_Ialltoall 起動とアシスタントコアへの移行処理のオーバーヘッドによってスレッド間の負荷バランスが崩れて再びバリア同期待ちのコストが顕在化する。このため、動的スレッドスケジューリングによって負荷分散を均等化する実装が最も良い性能となった。アシスタントコアの利用によって比較的簡易に通信オーバーラップを実現できることを集団通信についても示したが、大規模問題で Tofu 専用アルゴリズムの利用が必須となる場合には従来の通信マスク手法の利用が推奨される。

2.8.4 GT5D カーネルにおける演算処理の性能測定

FX100 で強化されたキャッシュ機構等、演算機能を評価するために GT5D カーネルの逐次処理性能を「京」と FX100 で比較した。p21 に示す GT5D カーネルは xyzv 方向の 4 次元移流項を 4 次精度中心差分によって評価する 17 ステンシルの計算となっている（ここでは、逐次処理性能を比較するため、通信オーバーラップの際に適用した zv 差分演算と xy 差分演算の分割は適用していない）。ソースコードにおいて浮動小数点演算数は 35Flop となるのに対し、メモリアクセス数は 32B となる。ここで、l-2~l+2、k-2~k+2、j-2~j+2 の範囲はコアあたりのキャッシュサイズ以下となるため、オンキャッシュとして取り扱う。i-2~i+2 についてはメモリアクセスとなるが、動的スケジューリングを適用する場合には隣接スレッドがロードしたデータを共有キャッシュ上で再利用できるため、やはりオンキャッシュとして取り扱う。この結果、メモリアクセスは f(l,k,j,i)、fc(l,k,j,i)のロー

ド、および、df(l,k,j,i)のロード・ストアで合計 4 となり、演算密度は Flop/Byte~1.1 となる。この演算密度からルーフラインモデルによって見積もった演算性能の対ピーク性能比は「京」、および、FX100 でそれぞれ 28.4%、および、25.1%となるが、実測した演算性能は約半分となり、理論的な予測を大幅に下回っている。プロファイラの数値を比較すると、「京」では L1D ミス数が大きく L1D ミス dm 率（プリフェッチが効かない実質的な L1D ミスの比率）も高い。一方、FX100 では L2 ミス dm 率（プリフェッチが効かない実質的な L2 ミスの比率）が高く、FMA 命令率（ここでは浮動小数点命令数に対して定義）が「京」よりかなり低いことがわかった。ここで、非 FMA 命令では FMA 命令の半分の演算しか処理できないため、FMA 命令率が低いと処理性能が劣化する。

この状況をより詳しく調べるために、p22 ではさらにカーネルを簡約化し、定数係数の 17 ステンシル計算を行った。この結果、「京」ではルーフラインモデルによる対ピーク性能比の 8 割以上の演算性能を達成した。プロファイラの数値を見ると、「京」では L1D ミス数が p24 の理論見積とほぼ一致し、L1D ミス dm 率、L2 ミス dm 率も低く、ほぼ理想的なメモリアクセスが実現していることがわかった。ここで、L1D ミス数の理論評価は同じキャッシュライン上の l-2~l+2、および、プリフェッチされる 3 キャッシュライン上の k-1~k+1 までを含む 7 要素をオン L1 キャッシュとして扱い、残りの 10 要素にループ回転数をかけてキャッシュラインサイズで割った値によって定義している。また、ソースコードは典型的な積和処理のパターンになっており、FMA 命令率も 9 割以上という理想的な性能となっていた。一方、FX100 ではキャッシュラインサイズが 128B から 256B に拡張されたことに伴い L1D ミス数が半減し、L1 キャッシュについてはほぼ理想的な性能が得られているが、L2 ミス dm 率が高くプリフェッチの効率が低い、また、FMA 命令率が低いという問題が明らかになった。

次に、p23 では係数を定数から配列に変更し、17 ストリーム版に近い形に修正した。この結果、「京」では L1D ミス数、および、L1D ミス dm 率が増大し、演算性能がルーフラインモデルによる理論的な予測から大幅に劣化することが確認された。L1 キャッシュが 2way の「京」ではループ内の配列数の増大に伴って、L1 キャッシュ競合が頻発する。この問題の回避策としては、通信オーバーラップで適用したようなループ分割によってループ内の配列数を調整することが有効である。一方、FX100 では L1 キャッシュが 4way に増え、キャッシュラインサイズも 256B に拡張されているため、「京」で顕在化したような L1 キャッシュ競合は見られず L1D ミス数は変化しなかった。ただし、L1D ミス dm 率が増大しており、プリフェッチの効率が劣化していることがわかった。

以上の性能測定結果から、FX100 では「京」の弱点であった L1 キャッシュの問題が解消しているが、FMA 命令率とプリフェッチ効率の低下という新たな問題が発生していることがわかった。この 2 つの問題の原因調査を行ったが、本報告書の執筆時点でプリフェッチ効率低下の問題は未解決であったため、以下では FMA 命令率の問題について調査結果をまとめる。

2.8.4.1 FX100 における FMA 命令率低下

FMA 命令率低下の原因調査にあたり、コンパイラの動作を解析した結果、ループボディの演算特性に応じてコンパイラが以下の 2 つの最適化アルゴリズムから最適と判断されるものを選択していることがわかった。p26 の FMA 促進アルゴリズムでは積和型の多項式演算を FMA 命令の繰り返しによって深い木構造で表現するのに対し、p27 の並列化促進アルゴリズムでは同じ演算を非 FMA 命令が多い浅い木構造で表現する。前者は FMA 命令率が高い最適化であるが、命令の並列度が低くソフトウェアパイプライニングが有効化できるループに対して高い性能を発揮する。一方、後者は FMA 命令率が低いが、命令の並列度が高く、ソフトウェアパイプライニング

がそれほど効かない場合でも、命令スケジューリングを最適化できる。通常はコンパイラが自動的にどちらの最適化を採用するか選択するが、今回は強制的に 2 つの最適化アルゴリズムを適用し、p28 に示すループボディのサイズが異なる 5 つのプログラムの処理性能を比較した。

p.29 に 9 ステンシルのコード C の処理性能を示す。このプログラムに対してコンパイラは並列化促進アルゴリズムを選択するため、オリジナルの場合には FMA 命令率（ここではロード・ストア、整数演算等を含む全命令数に対して定義）が低くなる。これに FMA 促進アルゴリズムを適用すると FMA 命令率は FX10 とほぼ同じ値に向上するが、処理速度は低下する。ここで、FMA 促進アルゴリズムを適用した場合には SIMD 幅が 128bit の FX10 に比べて SIMD 幅が 256bit の FX100 では総命令数がほぼ半分になる。p30 に 5 つのプログラムに対する最適化アルゴリズムの比較を示すが、ループボディが小さい場合には並列化促進アルゴリズムが有効であるが、ループボディが大きくなるにつれて FMA 促進アルゴリズムの性能が逆転する傾向にあることがわかった。p22 の簡約化カーネルに相当するコード D の場合には、コンパイラは並列化促進アルゴリズムを選択していたが、この調査結果から FMA 促進アルゴリズムの方が高い処理性能を示すことがわかった。

2.8.5 まとめ

核融合プラズマ流体解析コード GT5D、および、GKV の主要計算カーネルを対象として FX100 の新機能の有効性を検証した。アシスタントコアを利用した通信オーバーラップでは 1 対 1 通信、および、集団通信の両方に対して従来の通信マスク手法と同程度の処理性能が得られたが、MPI を起動してアシスタントコアに処理を移行するまでのオーバーヘッドが無視できないため、通信マスク手法と同様の動的スケジューリングによるスレッド間の負荷分散の最適化が必要である。演算性能に関しては、L1 キャッシュの機能強化によって「京」の弱点であった L1 キャッシュ競合が改善しているが、「京」に比べて FMA 命令率やプリフェッチ効率が低下するという新たな問題点が明らかになった。これらの問題は今後のコンパイラ改善によって解消されるものと期待する。

SS研ポストペタアプリ性能WG

核融合プラズマ流体コード GT5DおよびGKVの性能測定

日本原子力研究開発機構
井戸村泰宏、伊奈拓也、朝比祐一

名古屋大学
前山伸也

富士通株式会社
三吉郁夫、千葉修一

1

コードの概要

2

核融合プラズマ流体解析の第一原理計算モデル

[Brizard-Hahm, Rev. Mod. Phys. 2010]

- プラズマ分布の移流・拡散方程式

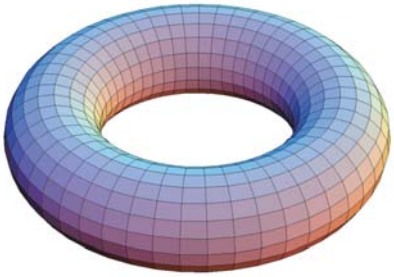
$$\frac{\partial f}{\partial t} = \mathcal{G}[f] + \mathcal{C}[f]$$

$$\boxed{\mathcal{G}[f] = -\mathbf{U}_1[\phi]\cdot\frac{\partial f}{\partial \mathbf{R}} - U_2[\phi]\frac{\partial f}{\partial v_{\|}}}$$ 熱運動や乱流場による移流

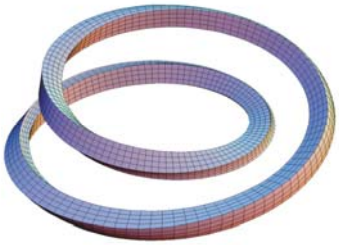
$$\mathcal{C}[f] = -\frac{\partial}{\partial \mathbf{v}}\cdot(\mathbf{D}_1 f) + \frac{\partial^2}{\partial \mathbf{v}\partial \mathbf{v}} : (\mathbf{D}_2 f)$$ クーロン衝突効果

- 乱流場を決めるポアソン方程式

$$-\nabla_\perp\cdot(P_1\nabla_\perp\phi) + P_2(\phi - \langle\phi\rangle) = \rho[f]$$



大域的モデル（GT5D、5D差分）



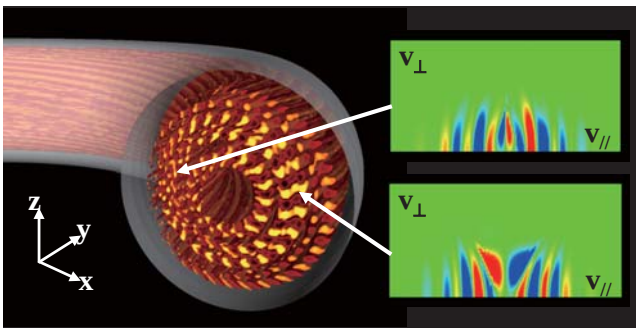
局所的モデル（GKV、3D差分+2DFFT）

核融合プラズマ乱流コードGT5Dの概要

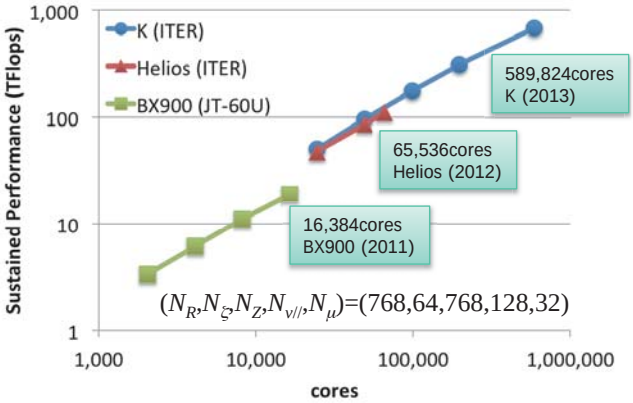
- Gyrokinetic Toroidal 5D Eulerian code
  完全トーラス配位において5次元ジャイロ運動論モデルによる核融合炉心プラズマの第一原理乱流計算シミュレーションを行い、乱流輸送特性を評価
- 計算手法
  MPI+OpenMPハイブリッド並列モデル
  5次元位相空間の移流・拡散→差分法
  時間積分→半陰的ルンゲ・クッタ法
  （カーネル→共役残差法+通信隠蔽）
- 計算規模（ITER規模）
  610x128x610x96x20 ~ 1000億格子
  ~10^5ステップ
  →約4百万ノード時間/ショット
- 参考文献

Idomura et al., Comput.Phys.Commun.2008
Idomura et al., Nucl. Fusion 2009
Idomura et al., Int. J. HPC Appl. 2014
Idomura-Nakata, Phys. Plasmas 2014
Idomura, Phys. Plasmas 2014

5次元位相空間（速度2次元）の摂動粒子分布



GT5Dコードのストロングスケーリング

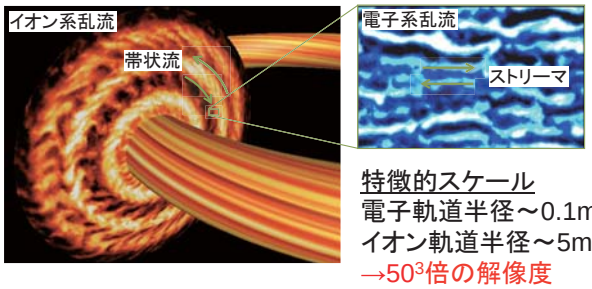


核融合プラズマ乱流コードGKVの概要

- GyroKinetic Vlasov code
  磁力線に沿った局所領域において5次元ジャイロ運動論モデルによる核融合炉心プラズマの第一原理乱流計算を行い、乱流輸送特性を評価する
- 計算手法
  MPI+OpenMPハイブリッド並列モデル
  5次元位相空間の移流・拡散
  →スペクトル法(x,y)、差分法(z,v,w)
  時間積分→ルンゲ・クッタ法
  （カーネル→2次元FFT+通信隠蔽）
- 計算規模
  1024x1024x96x96x32x2 ~ 6000億格子
  ~10^5ステップ
  →約4百万ノード時間/ショット
- 参考文献

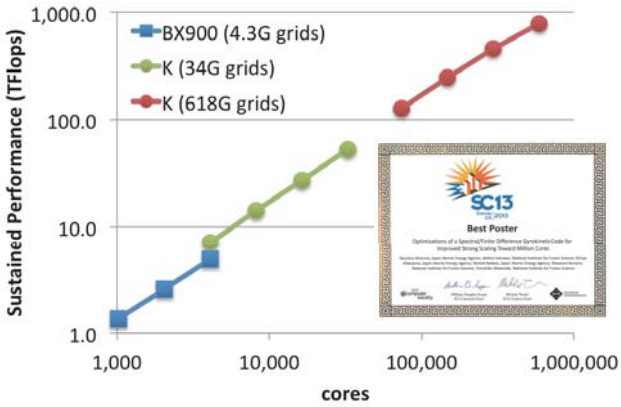
Watanabe-Sugama, Nucl. Fusion 2006
Watanabe et al., Phys. Rev. Lett. 2008
Maeyama et al., Parallel Computing 2015
Maeyama et al., Phys. Rev. Lett. 2015

GKVによるマルチスケール電子乱流計算



特徴的スケール
電子軌道半径～0.1mm
イオン軌道半径～5mm
→50^3倍の解像度

GKVコードのストロングスケーリング

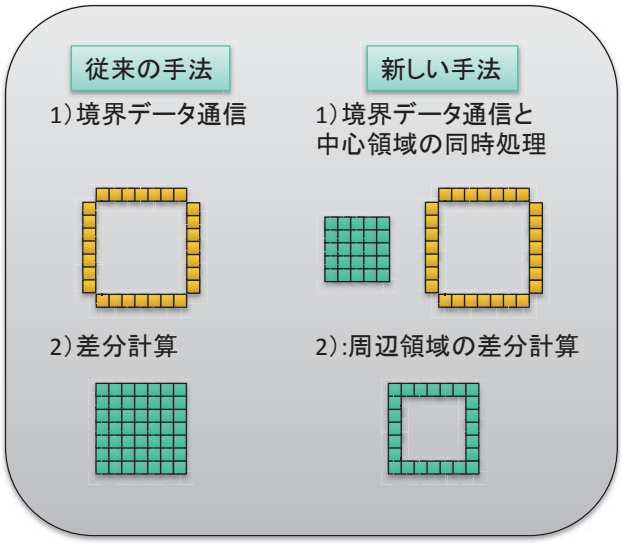


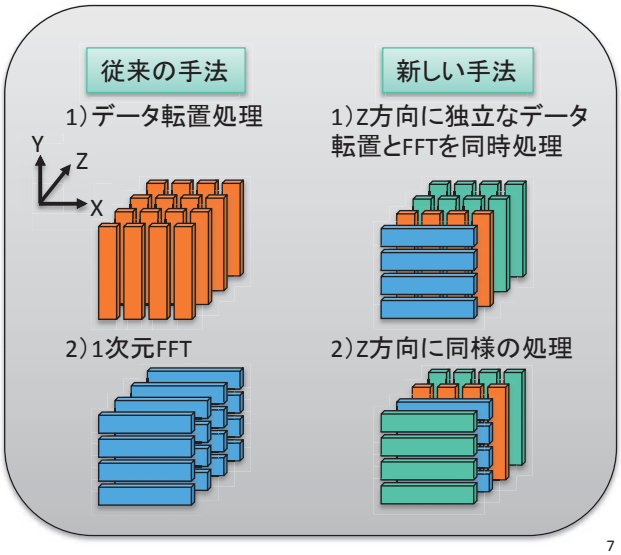
通信と演算のオーバーラップ

通信と演算の同時処理によるオーバーヘッド削減

通信データと独立な演算処理は原理的に同時処理が可能

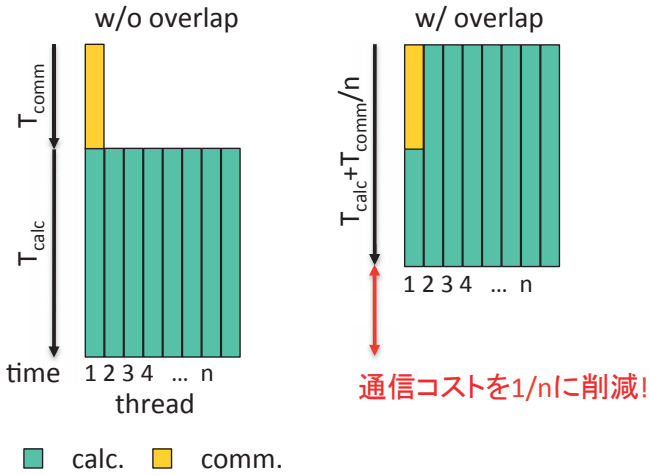
- 領域分割した差分演算の例
  - 境界データを①参照する演算と②参照しない演算に分離
  - 演算処理②の背後で境界データの1対1通信を実行
- 並列化2次元FFTの例
  - データ転置、FFTを全てZ方向に分割
  - Z方向に異るインデックスのデータ転置とFFTを同時処理





通信と演算の同時処理を実現する通信マスク手法

- OpenMPで通信スレッドを実装し、依存関係の無い通信と演算を同時処理
  - 1対1通信（同期通信、非同期通信）
    - 差分計算の袖領域通信（GT5D、GKV）
  - 集団通信
    - 多次元並列化FFT（GKV）
    - データ転置処理（GT5D）



コーディングのイメージ

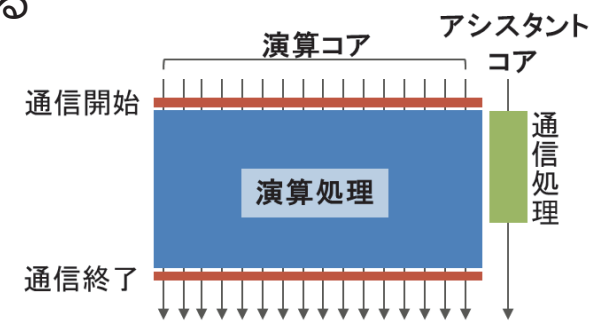
```
!$OMP PARALLEL

!$OMP MASTER
  call MPI communication
!$OMP END MASTER

!$OMP DO SCHEDULE(DYNAMIC)
  do
    call calculation
  enddo

!$OMP END PARALLEL
```

アシスタントコアによる通信オーバラップ促進

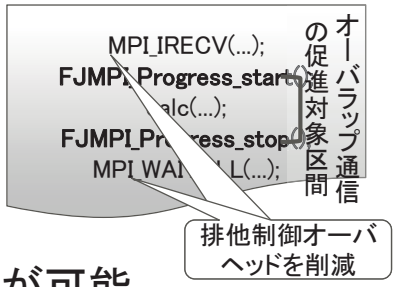
FUJITSU

- アシスタントコアに通信処理を任せる
  - MPIライブラリに動作モードを追加
    a. progress threadなし (デフォルト)
    b. progress threadあり
      i. 自動区間モード
      ii. 指定区間MPI呼び出しなしモード
      iii. 指定区間MPI呼び出しありモード
  - 動作概要:
    アシスタントコアがノンブロッキング通信処理の大半を実行
  - 利用者の対処
    1. MCAパラメータでprogress thread有無を選択
    2. 必要に応じて、オーバラップ通信の対象区間を指定
       - FJMPI_Progress_start: 非同期通信の促進処理を開始
       - FJMPI_Progress_stop: 非同期通信の促進処理を停止

progress thread使用時の動作イメージ



対象区間を明示する場合



- メリット: コード書き換えなしにオーバラップ通信が可能、
  簡易な区間指定で排他制御オーバヘッドも抑制できる

Copyright 2014 FUJITSU LIMITED

MPI-3.0のノンブロッキング集団通信API (抜粋)

FUJITSU

- 1対多通信
  - MPI_IBCAST(buffer, count, datatype, root, comm, request)
  - MPI_ISCATTER(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, request)
- 多対1通信
  - MPI_IGATHER(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, request)
- 多対多通信
  - MPI_IALLGATHER(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm, request)
  - MPI_IALLTOALL(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm, request)
- リダクション
  - MPI_IREDUCE(sendbuf, recvbuf, count, datatype, op, root, comm, request)
  - MPI_IALLREDUCE(sendbuf, recvbuf, count, datatype, op, comm, request)
- バリア同期
  - MPI_IBARRIER(comm, request)

Copyright 2015 FUJITSU LIMITED

GT5Dカーネルにおける
境界データ通信のオーバーラップ

通信オーバラップ効果の評価(1/3)

FUJITSU

- 評価条件
  - 測定環境
    - FX100 32ノード (1.975GHz、32.4Tflops)
    - FX10 64ノード (1.848GHz、15.1Tflops)
  - 測定条件
    - 問題サイズ:
      nnx×nny×nnz×nnv＝256×256×64×128
    - 並列数:
      npx×npy＝8×8 (16スレッド×64プロセス)
  - 使用バイナリ
    - original版: MPI_Isend/Irecv/Waitallを連続して実行
    - overlap版: マスタスレッドがMPI_Isend/Irecv/Waitを実行し、他は演算処理を担当
    - unroll版: 通信直後の演算処理(kernel_p区間)で外側2重ループを一重化
    - unroll+overlap版: overlap版とunroll版を融合させ、演算負荷不均衡を改善
    - 通信スレッド解除版: unroll+overlap版をもとに、MPI_Waitallを演算処理後に移動
  - その他
    - FX100のオーバラップ促進は自動区間モードを使用
      (--mca opal_progress_thread_mode 3)

unroll+overlap版のコードイメージ

```
!$OMP PARALLEL
!$OMP MASTER
   MPI_Isend
   MPI_Irecv
   MPI_Waitall
!$OMP END MASTER
!$OMP DO SCHEDULE(DYNAMIC)
   do ij=…
      演算処理
   enddo
!$OMP END PARALLEL
```

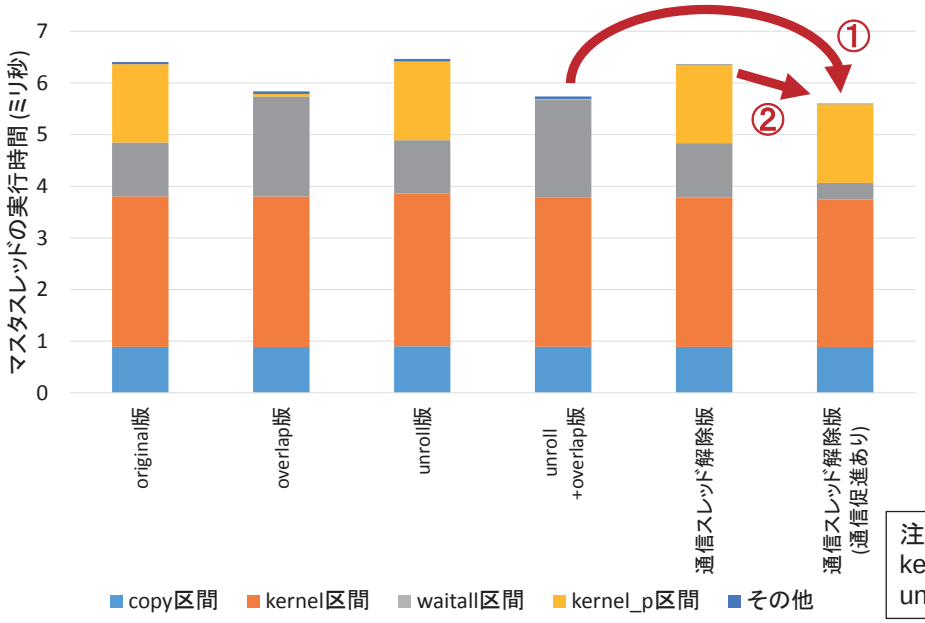
通信スレッド解除版では
分離し、演算処理後に移動

Copyright 2015 FUJITSU LIMITED

通信オーバラップ効果の評価(2/3)

FUJITSU

■ 通信スレッド対アシスタントコアの効果比較



注) 通信スレッド解除版の kernel_p区間時間は unroll版からの推定による

■ アシスタントコアのオーバラップ促進効果は、通信スレッド方式と同等以上①

■ 通信スレッド解除版は、アシスタントコアによる通信促進で14%性能向上②

Copyright 2015 FUJITSU LIMITED

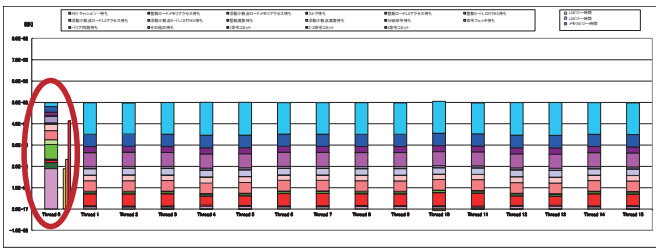
通信オーバラップ効果の評価(3/3)

FUJITSU

■ サイクルアカウンティング結果の比較(kernel_p区間)

■ unroll+overlap版

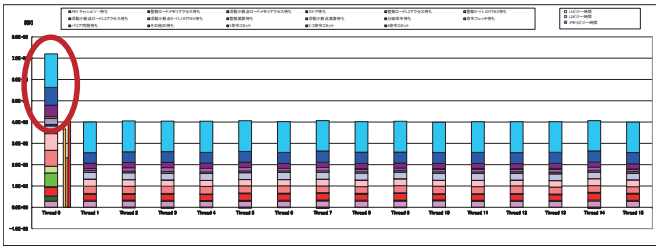
- マスタスレッドの演算数はほぼゼロ
  ⇒マスタスレッドは通信処理に専念



■ 通信スレッド解除版※

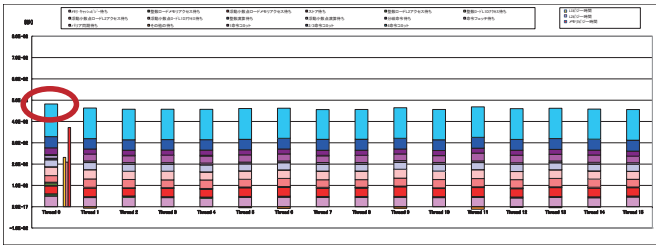
- 全スレッドで演算数は同等
- マスタスレッドは通信処理も実行

※ただしkernel_p区間のスレッド並列ループは静的スケジューリング



■ 通信スレッド解除版※(通信促進あり)

- 全スレッドで演算数は同等
- マスタスレッドの遅れは5%程度



➡ **スレッド数が2冪、動的スケジューリング不要、ループ一重化も不要**

Copyright 2015 FUJITSU LIMITED

GKVカーネルにおける集団通信の性能測定

GKVカーネルにおけるオーバラップ効果 (1/4)

FUJITSU

- 評価条件
  - 測定環境:
    - FX100 16ノード (2.2GHz、18.0Tflops)
  - 測定条件
    - 問題サイズ: nxw×nyw＝128×128、global_nz×global_nv×global_nm＝16×16×15
    - 並列数: 16スレッド×32プロセス、nprocw×nprocz×nprocv×nprocm×nprocs＝4×2×2×2×1
  - 使用バイナリ
    a. オーバラップなし (MPI_Alltoall)
    b. オーバラップあり (MPI_Alltoall＋OpenMP通信処理スレッド)
    c. オーバラップなし (MPI_Ialltoall)※
    d. オーバラップあり (MPI_Ialltoall)※
    e. オーバラップあり (MPI_Ialltoall＋静的スレッドスケジュール)※
  - その他
    - オーバラップ促進は自動区間モードを使用 (--mca opal_progress_thread_mode 3)

※:富士通側で作成

d. オーバラップあり (MPI_Ialltoall) のコードイメージ

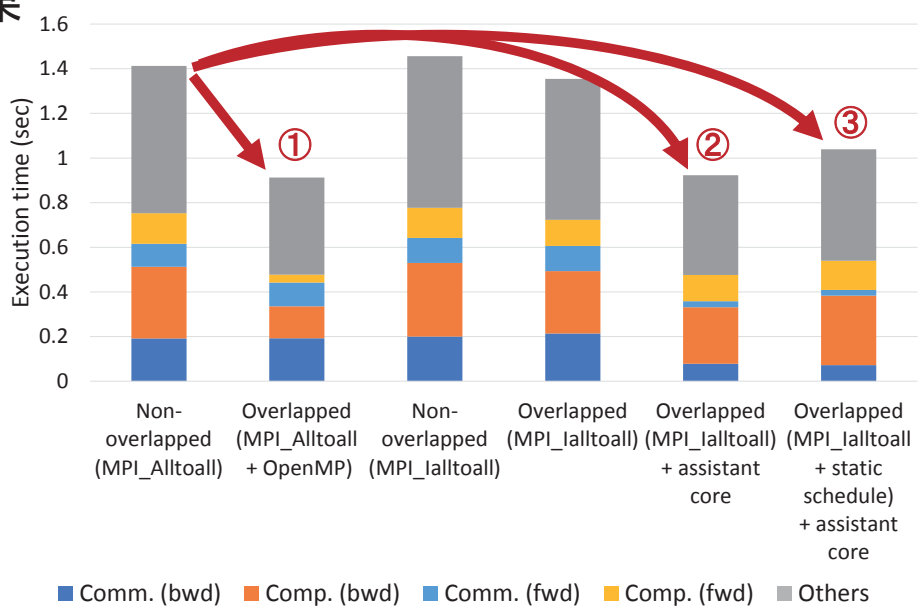
```
!$OMP PARALLEL
   do im = 2, nm-2
!$OMP MASTER
      CALL MPI_Ialltoall
!$OMP END MASTER
!$OMP DO SCHEDULE(DYNAMIC)
      do i = 1, num_trans
         FFT演算
      enddo
!$OMP MASTER
      CALL MPI_Waitall
!$OMP END MASTER
!$OMP BARRIER
   enddo
!$OMP END PARALLEL
```


Copyright 2015 FUJITSU LIMITED

GKVカーネルにおけるオーバラップ効果 (2/4)

FUJITSU

■ 測定結果



- MPI_Alltoall＋OpenMP通信処理スレッドでは、35.4%の実行時間短縮①
- MPI_Ialltoall＋アシスタントコア有効化では、34.7%の実行時間短縮②
- ②に対し静的スレッドスケジュールを行うと、短縮幅は26.5%に減少③

Copyright 2015 FUJITSU LIMITED

GKVカーネルにおけるオーバラップ効果 (3/4)

FUJITSU

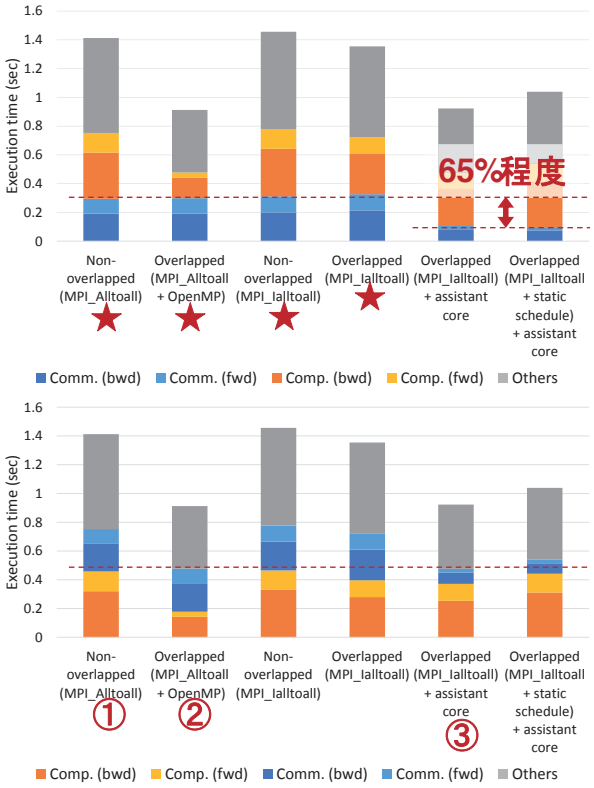
■ 通信処理時間短縮および
オーバラップ度合いの確認

- アシスタントコア無効時★は
MPI_AlltoallとMPI_Ialltoallが
ほぼ同じ処理時間
  - 後者はTofu向けアルゴリズムでないが、ノード数が少ないため影響小
- アシスタントコアを有効にすると、
MPI_Ialltoallは65%程度時間短縮

- ②MPI_Alltoall＋OMP通信スレッドや
③MPI_Ialltoall＋アシスタントコアでは
bwd＋fwd区間の経過時間が
①オーバラップなし時の演算時間に
概ね一致

➡ **通信時間の隠ぺいに成功**

各測定の時間内訳を並べ替えた結果



Copyright 2015 FUJITSU LIMITED

GKVカーネルにおけるオーバラップ効果 (4/4)

FUJITSU

■ サイクルアカウンティング結果の比較(bwd:comm.+comp.区間)

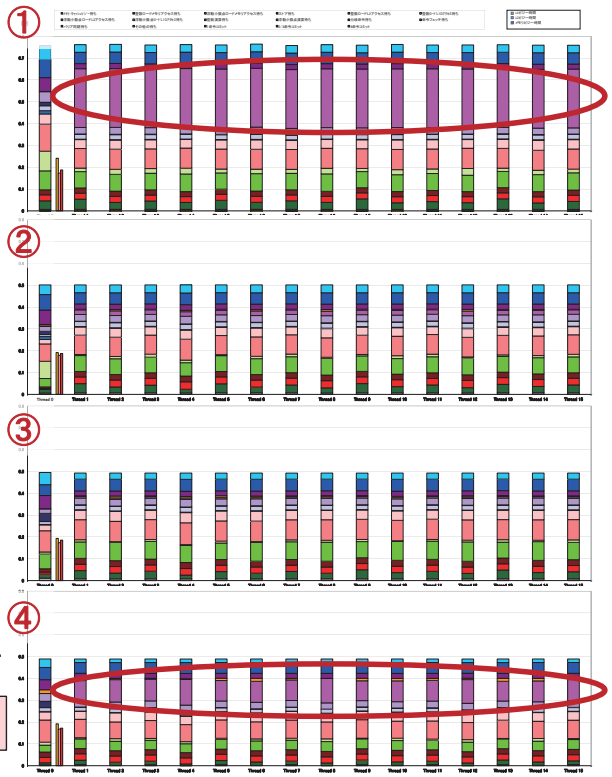
① オーバラップなし (MPI_Alltoall)
- 非マスタスレッドのバリア同期待ち大

② オーバラップあり
(MPI_Alltoall＋OMP通信スレッド)
- バリア同期待ちをほぼ解消

③ オーバラップあり (MPI_Ialltoall)
＋アシスタントコア
- バリア同期待ちをほぼ解消

④ オーバラップあり
(MPI_Ialltoall＋静的ｽﾚｯﾄﾞｽｹｼﾞｭｰﾙ)
＋アシスタントコア
- 各スレッドの演算量が同じ
⇒非マスタスレッドに通信処理待ちが発生

今回ケースは動的スレッドスケジュールが有利



Copyright 2015 FUJITSU LIMITED

GT5Dカーネルにおける演算処理の性能測定

GT5Dカーネル

演算数	35	
メモリアクセス数	32	
環境	K	FX100-1CMG
ルーフライン[%]	28.4	25.1
ルーフライン[msec]	2.01	0.58
演算性能[%]	14.0	13.4
実行時間[msec]	4.39	1.17
メモリバンド幅 [GB/s]	23.4	89.3
L1Dミス数	2.73e6	1.04e6
L1Dミスdm率[%]	43.4	24.8
L2ミス数	6.44e5	3.14e5
L2ミスdm率[%]	8.9	37.4
SIMD命令率[%]	99.7	99.8
積和SIMD演算/SIMD演算[%]	72.3	34.3

※STREAM Benchmark: K~46.6GB/s, FX100~310GB/s

※Compiler option

K: -Kfast -Kparallel,openmp -X9 -Kmfunc=2 -Kocl -CcdRR8 -fw -Qt -Ntl_notrt

FX100: -xpre_diff -Kfast -Kopenmp -Kocl,optmsg=2 -X9 -Kmfunc=2 -Kloop_nofusion -CcdRR8 -Kprefetch_sequential=soft,prefetch_cache_level=1

- ルーフラインから大きく性能劣化
  →性能劣化の原因を調査したい

```
real*8 f(-1:nv+2,-1:nz+2,-1:ny+2,-1:nx+2)
!$OMP DO SCHEDULE(dynamic)
    do i = 1,nx    (nx=32)
    do j = 1,ny    (ny=32)
    compute vxyv~vxl2 from vx(l,j,i),vy(l,j,i),vz(l,j,i),vv(l,j,i)
    do k = 1,nz    (nz=16)
    do l = 1,nv    (nv=128)
      df(l,k,j,i)=                                  LOAD
  &       vxyv(l)*f(l,k,j,i)                        LOAD/STORE
  &     + 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)
  &     +vzl2(l)*(f(l,k-2,j,i)-f(l,k+2,j,i))
  &     +vzl(l) *(f(l,k+1,j,i)-f(l,k-1,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
```


簡約化カーネル①:定数係数

演算数	33	
メモリアクセス数	24	
環境	K	FX100-1CMG
ルーフライン[%]	33.3	29.7
ルーフライン[msec]	1.62	0.46
演算性能[%]	28.3	16.5
実行時間[msec]	1.91	0.83
メモリバンド幅 [GB/s]	38.9	87.6
L1Dミス数	1.41e6	7.35e5
L1Dミスdm率[%]	18.8	6.6
L2ミス数	4.17e5	1.87e5
L2ミスdm率[%]	4.0	34.2
SIMD命令率[%]	100.0	100.0
積和SIMD演算/SIMD演算[%]	93.9	37.5

- 京ではルーフラインの8割以上となるのに対し、FX100では大幅に性能劣化
  - L2ミスdm率が高くプリフェッチがうまく機能していない？
  - FMA演算率が低く。単純な積和のカーネルをうまく解釈できていない？

```
    real*8 f(-1:nv+2,-1:nz+2,-1:ny+2,-1:nx+2)
!$OMP DO SCHEDULE(dynamic)
    do i = 1,nx    (nx=32)
    do j = 1,ny    (ny=32)
    do k = 1,nz    (nz=16)
    do l = 1,nv    (nv=128)                         LOAD
        df(l,k,j,i)= &                              LOAD/STORE
  &         f(l,k,j,i)    *1.11d0
  &         +f(l-2,k,j,i) *1.12d0
  &         +f(l-1,k,j,i) *1.13d0
  &         +f(l+1,k,j,i) *1.14d0
  &         +f(l+2,k,j,i) *1.15d0
  &         +f(l,k-2,j,i) *1.16d0
  &         +f(l,k-1,j,i) *1.17d0
  &         +f(l,k+1,j,i) *1.18d0
  &         +f(l,k+2,j,i) *1.19d0
  &         +f(l,k,j-2,i) *1.21d0
  &         +f(l,k,j-1,i) *1.22d0
  &         +f(l,k,j+1,i) *1.23d0
  &         +f(l,k,j+2,i) *1.24d0
  &         +f(l,k,j,i-2) *1.25d0
  &         +f(l,k,j,i-1) *1.26d0
  &         +f(l,k,j,i+1) *1.27d0
  &         +f(l,k,j,i+2) *1.28d0
    enddo
    enddo
    enddo
    enddo
```

簡約化カーネル②:配列係数

演算数	33	
メモリアクセス数	24	
環境	K	FX100-1CMG
ルーフライン[%]	33.3	29.7
ルーフライン[msec]	1.62	0.46
演算性能[%]	15.9	12.6
実行時間[msec]	3.40	1.09
メモリバンド幅 [GB/s]	21.6	72.3
L1Dミス数	2.54e6	7.75e5
L1Dミスdm率[%]	30.0	25.2
L2ミス数	4.11e5	2.10e5
L2ミスdm率[%]	5.0	17.0
SIMD命令率[%]	100.0	100.0
積和SIMD演算/SIMD演算[%]	94.0	37.4

```
      real*8 f(-1:nv+2,-1:nz+2,-1:ny+2,-1:nx+2)
!$OMP DO SCHEDULE(dynamic)
      do i = 1,nx    (nx=32)
      do j = 1,ny    (ny=32)
      do k = 1,nz    (nz=16)
      do l = 1,nv    (nv=128)
          df(l,k,j,i)= &
     &          vv_(l) *f(l,k,j,i)
     &         +vvl2(l)*f(l-2,k,j,i)
     &         -vvl(l) *f(l-1,k,j,i)
     &         +vvr(l) *f(l+1,k,j,i)
     &         -vvr2(l)*f(l+2,k,j,i)
     &         +vzl2(l)*f(l,k-2,j,i)
     &         -vzl(l) *f(l,k-1,j,i)
     &         +vzr(l) *f(l,k+1,j,i)
     &         -vzr2(l)*f(l,k+2,j,i)
     &         +vyl2(l)*f(l,k,j-2,i)
     &         -vyl(l) *f(l,k,j-1,i)
     &         +vyr(l) *f(l,k,j+1,i)
     &         -vyr2(l)*f(l,k,j+2,i)
     &         +vxl2(l)*f(l,k,j,i-2)
     &         -vxl(l) *f(l,k,j,i-1)
     &         +vxr(l) *f(l,k,j,i+1)
     &         -vxr2(l)*f(l,k,j,i+2)
      enddo
      enddo
      enddo
      enddo
```

LOAD
LOAD/STORE

- 京でも大幅に性能劣化
  - L1Dミス数、dm率ともに大幅に増加。
  - FX100ではL1Dミス数の増大は無い。

    Cf. L1Dミス理論見積

    K:　8B×10要素×(128×16×32×32)回転÷128B/$ライン＝1.31e6

    FX100:8B×10要素×(128×16×32×32)回転÷256B/$ライン＝6.55e5

    ※l-2~l+2,k-1~k+1をオンL1$と推定

FX100における FMA命令率低下の原因

技術情報 (1)

FUJITSU

■Tree Height ReductionのFMA化変形

■特徴

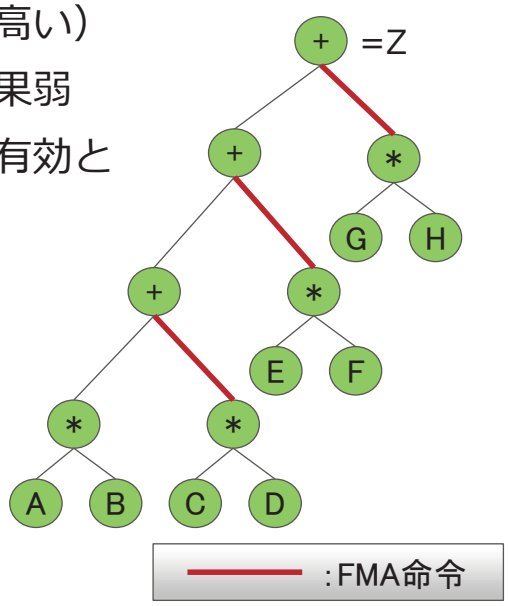
- FMA化率が高い
- 命令の並列性が低い（演算ツリーが高い）
- 局所的な命令スケジューリングの効果弱
- ソフトウェアパイプラインニングが有効と判断した場合に選択

■変形例

Z=A*B+C*D+E*F+G*H

↓

Z=(((A*B+C*D)+E*F)+G*H)



Copyright 2016 FUJITSU LIMITED

技術情報 (2)

FUJITSU

■Tree Height Reductionの並列化変形

■特徴

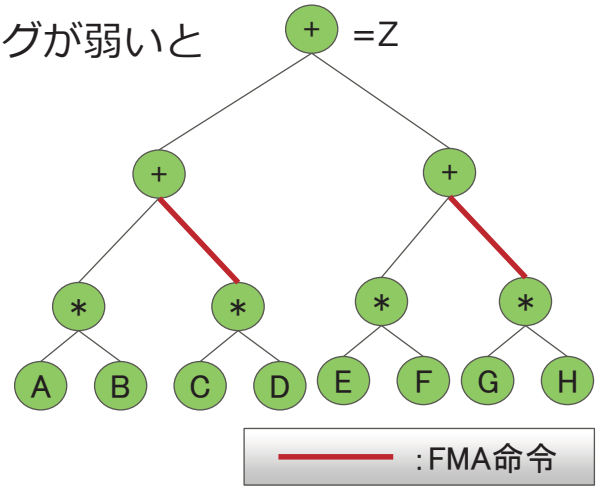
- FMA化率が低い
- 命令の並列性が高い（演算ツリーが低い）
- 局所的な命令スケジューリングの効果高
- ソフトウェアパイプラインニングが弱いと判断した場合に選択

■変形例

Z=A*B+C*D+E*F+G*H

↓

Z=((A*B+C*D)+(E*F+G*H))



Copyright 2016 FUJITSU LIMITED

対象コード (2)

追加

FUJITSU

■検証コード

オリジナルコードのループボディのサイズを変化

コード A

```
~
 df(…) = f(l,k,j,i)    * 1.11d0 + 1.12d0
~
```

コード B

```
~
 df(…) = f(l,k,j,i)    * 1.11d0 + f(l+1,k,j,i) * 1.12d0 &
       + f(l,k+1,j,i) * 1.13d0 + f(l,k,j+1,i) * 1.14d0 &
       + f(l,k,j,i+1) * 1.15d0
~
```

コード C

```
~
 df(…) = f(l,k,j,i)    * 1.11d0 + f(l-1,k,j,i) * 1.12d0 &
       + f(l+1,k,j,i) * 1.13d0 + f(l,k-1,j,i) * 1.14d0 &
       + f(l,k+1,j,i) * 1.15d0 + f(l,k,j-1,i) * 1.16d0 &
       + f(l,k,j+1,i) * 1.17d0 + f(l,k,j,i-1) * 1.18d0 &
       + f(l,k,j,i+1) * 1.19d0
~
```

コード D

```
~
 df(…) = f(l,k,j,i)    * 1.11d0 + f(l-2,k,j,i) * 1.12d0 &
       + f(l-1,k,j,i) * 1.13d0 + f(l+1,k,j,i) * 1.14d0 &
       + f(l+2,k,j,i) * 1.15d0 + f(l,k-2,j,i) * 1.16d0 &
       + f(l,k-1,j,i) * 1.17d0 + f(l,k+1,j,i) * 1.18d0 &
       + f(l,k+2,j,i) * 1.19d0 + f(l,k,j-2,i) * 1.21d0 &
       + f(l,k,j-1,i) * 1.22d0 + f(l,k,j+1,i) * 1.23d0 &
       + f(l,k,j+2,i) * 1.24d0 + f(l,k,j,i-2) * 1.25d0 &
       + f(l,k,j,i-1) * 1.26d0 + f(l,k,j,i+1) * 1.27d0 &
       + f(l,k,j,i+2) * 1.28d0
~
```

コード E

```
~
 df(…) = f(l,k,j,i)    * 1.11d0 + f(l-3,k,j,i) * 1.12d0 &
       + f(l-2,k,j,i) * 1.13d0 + f(l-1,k,j,i) * 1.14d0 &
       + f(l+1,k,j,i) * 1.15d0 + f(l+2,k,j,i) * 1.16d0 &
       + f(l+3,k,j,i) * 1.17d0 + f(l,k-3,j,i) * 1.18d0 &
       + f(l,k-2,j,i) * 1.19d0 + f(l,k-1,j,i) * 1.20d0 &
       + f(l,k+1,j,i) * 1.21d0 + f(l,k+2,j,i) * 1.22d0 &
       + f(l,k+3,j,i) * 1.23d0 + f(l,k,j-3,i) * 1.24d0 &
       + f(l,k,j-2,i) * 1.25d0 + f(l,k,j-1,i) * 1.26d0 &
       + f(l,k,j+1,i) * 1.27d0 + f(l,k,j+2,i) * 1.28d0 &
       + f(l,k,j+3,i) * 1.29d0 + f(l,k,j,i-3) * 1.30d0 &
       + f(l,k,j,i-2) * 1.31d0 + f(l,k,j,i-1) * 1.32d0 &
       + f(l,k,j,i+1) * 1.33d0 + f(l,k,j,i+2) * 1.34d0 &
       + f(l,k,j,i+3) * 1.35d0
~
```

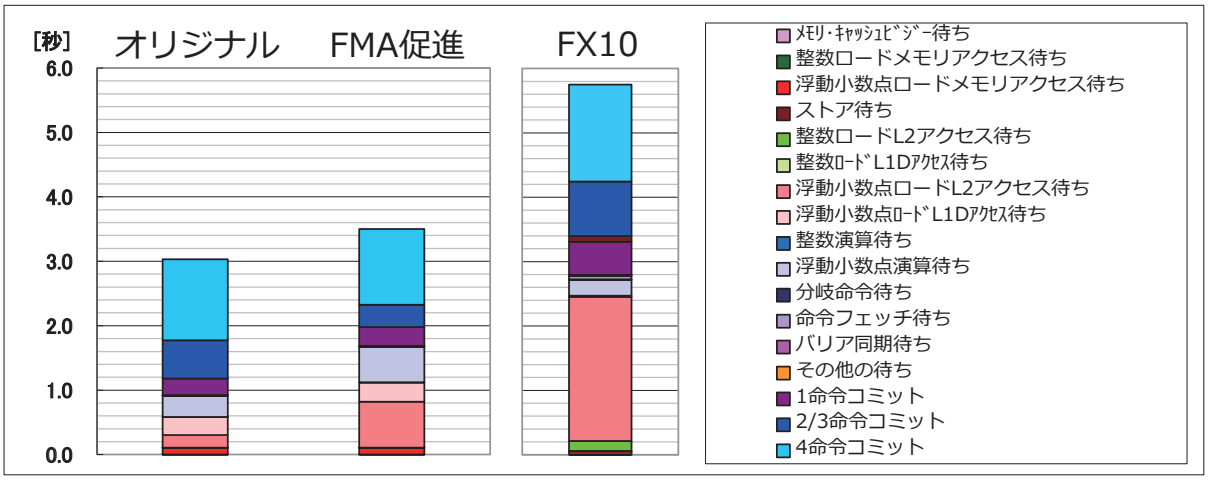

Copyright 2016 FUJITSU LIMITED

調査結果 (2)

変更

FUJITSU

■コードC 詳細プロファイラ



FX100	演算ピーク比(%)	FMA命令率(%)	LoadStore命令率(%)	ﾒﾓﾘｽﾙｰﾌﾟｯﾄ (GB/sec)	総命令数
オリジナル	37.38 %	19.93 %	31.98 %	21.61	1.00E+10
FMA促進	32.36 %	36.39 %	36.49 %	18.71	8.79E+09
FX10	32.01 %	38.86 %	44.95 %	11.36	1.65E+10

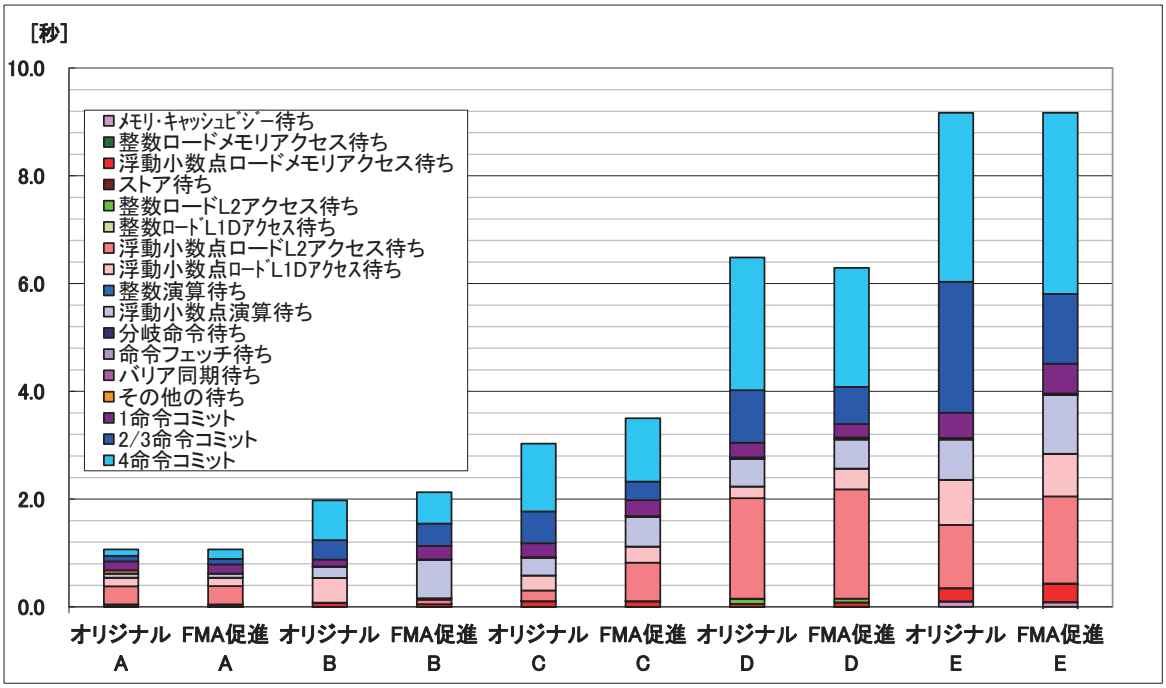
Copyright 2016 FUJITSU LIMITED

調査結果 (3)

追加

FUJITSU

■全体データ



Copyright 2016 FUJITSU LIMITED

まとめ

核融合プラズマ流体解析コードの主要計算カーネルを対象としてFX100の新機能の有効性を検証した。

- アシスタントコアを用いた通信と演算のオーバーラップ
  - 最小限のコード変更で従来の通信マスク手法と同程度の効果

    課題
    - MPI起動からアシスタントコアへの移行までのオーバーヘッド
    - 非同期集団通信へのTofu専用アルゴリズム適用
- 演算処理
  - L1キャッシュ強化によって「京」の弱点であったL1キャッシュ競合を解決

    課題
    - FMA命令率の向上
    - プリフェッチ効率の向上