Vol.2013-HPC-139 No.2
2013/5/29

京速コンピュータ「京」における核融合シミュレーションコードGTC-Pの評価

下坂健則[1]　佐藤三久[1,2]　朴泰祐[2]　William Tang[3]

概要：理化学研究所 計算科学研究機構では，国内外における既存の様々なシミュレーションコードを京速コンピュータ「京」上で評価することで，計算機環境を高度に整備し，ユーザのプログラム開発における生産性向上に寄与することをミッションの一つとしている．本稿では，「京」のシステム標準のプロファイラ，および「京」のユーザ向けに公開済みの性能解析ツール Scalasca を使用して，3 次元核融合シミュレーションコード GTC-P のコード解析を実施した．その結果，GTC-P では，通信負荷バランスと SIMD 化の問題により，ノード内性能、および並列化効率に課題があることが分かった．

1. はじめに

理化学研究所 計算科学研究機構では，国内外における既存の様々なシミュレーションコードを京速コンピュータ「京」上で評価することで，システムソフトウェアやプログラミング環境などの計算機環境を高度に整備し，ユーザのプログラム開発における生産性向上に寄与することを目的の一つとしている．本稿では，プリンストン・プラズマ物理研究所が開発している 3 次元核融合シミュレーションコード GTC(Gyrokinetic Toroidal Code) [1] の並列アルゴリズム改善版 GTC-P [2] を扱い，特に京上で利用可能な性能評価環境を使って解析した結果を報告する．

核融合シミュレーションは，日本学術振興会が推進する G8 多国間国際共同研究事業の一分野を成している．GTC-P は当該共同事業の研究対象プログラムである．現在の GTC-P コードは，国際熱核融合実験炉 ITER [3] 規模の計算量を想定したプログラムとなっている．

「京」では，システム標準の性能解析ツールとしてプロファイラ [4] を提供している．本稿では，今後の性能改善に向けて必要な基礎情報および課題を得ることを目的に，主に GTC-P の時間発展ループについて，ノード内，およびノード間のプロファイル情報を収集し，考察した．

また，計算科学研究機構では，テネシー大学と Julich Supercomputing Centre が開発した並列プログラム向け性能分析ツール Scalasca [5] を京ユーザ向けに公開している．GTC-P に対して本ツールを適用した結果についても合わせて報告する．

第 2 章では，「京」を含む性能評価環境を説明する．第 3 章では，「京」のプロファイラおよび Scalasca を使用したコード解析を説明し，第 4 章で解析結果の考察と今後の課題を示す．

2. 性能評価環境

2.1 京速コンピュータ「京」

京 [4] は，82,944 個の計算ノードを，「Tofu(Torus fusion)」と呼ばれる 6 次元メッシュ／トーラス構造のネットワークで相互に接続した並列計算機システムである．システム全体のピーク性能は 10.6PFLOPS に達する．各計算ノードは，1 個の CPU(SPARC64^{TM}VIIIfx)，1 個のネットワーク用 LSI(ICC: InterConnect Controller) および 16GB のメモリを持つ．ICC は，4 つの DMA エンジンを有し，4 方向の同時通信および RDMA(Remote Direct Memory Access) 通信ができる．通常，Tofu インターコネクトは，論理的な 3 次元トーラス構造として利用される．その場合の帯域は 3 次元の正負各方向にそれぞれ 5GB/s(双方向) である．SPARC64^{TM}VIIIfx の諸元を表 1 に示す．

SPARC64^{TM}VIIIfx は，SPARC-V9 アーキテクチャに対する科学技術計算向けの拡張命令セット (HPC-ACE) を有する．HPC-ACE は，レジスタ数の拡張，SIMD 演算，セクタキャッシュ機構，条件付き実行，三角関数の高速化命令，除算・平方近似の機能を有している．特に SIMD 演算では，浮動小数点積和演算，およびロード・ストア命令に対する SIMD(Single Instruction Multiple Data) 実行が可能となっているため，SIMD 命令の効果的な利用が，絶対性能に大きな影響を及ぼす．

[1] 理化学研究所 計算科学研究機構
[2] 筑波大学 計算科学研究センター
[3] Princeton University

表 1　SPARC64^{TM}VIIIfx 緒元

演算性能	128GFLOPS(16GFLOPS x 8 cores)
コア数	8
クロック周波数	2.0 GHz
浮動小数点演算器	乗加算ユニット x 4 (2 SIMD) 除算器 x 2
レジスタ数	浮動小数点レジスタ (64bit)：256 汎用レジスタ (64bit)：188
キャッシュ	L1I：32 KB (2way) L1D：32 KB (2way) L2：Shared 6 MB (12way)
メモリ帯域	64GB

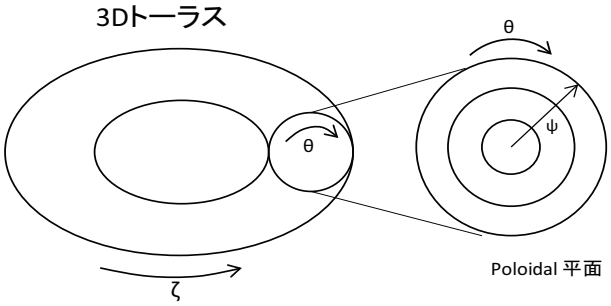

図 1　3D トーラス上の座標系

2.2　「京」のプロファイラ

「京」では，アプリケーションのチューニングを支援するため，システム標準のプロファイラを用意している．プロファイラには，アプリケーション全体の性能の概要を把握するための基本プロファイラと，アプリケーションの性能向上に重要なホットスポットを詳細に解析するための詳細プロファイラがある．本稿では，主に詳細プロファイラを使用して解析した．

2.3　GTCP

GTC は，トカマク型核融合プラズマの乱流現象を研究するための 3 次元 PIC(Particle in cell) シミュレーションコードである [1].

図 1 では，GTC を表現する空間である 3D トーラス上の座標系を示す．ζ は，3D トーラスの toroidal 方向の位置，θ は，poloidal 平面での poloidal 方向の位置，ψ は，poloidal 平面の radial 方向の位置を示している．

GTC は，PIC 法に基づき，時間発展ループを，以下の 6 つのパートで構成している．

該当粒子に対して最も近くに位置する，円上で隣接する 4 格子点に，粒子の電荷を均等に分配する (charge)．電荷密度と位置ベクトルを平準化する (smooth)．グリッド上でポアソン方程式を解く (poisson)．グリッド上の電場を計算する (field)．個々の荷電粒子の位置での電場の値を使って，荷電粒子の位置と速度を 1 時間発展分進める (push)．プロセス間もしくは toroidal 方向に分割した領域間で粒子を移動する (shift).

表 2　GTCP-C の問題規模

条件	mpsi	mthetamax	mzetamax	(ノード数)
A	90	640	64	(512)
B	180	1280	64	(2048)
C	360	2560	64	(8192)
D	720	5120	64	(32768)

GTC の並列化では，次の 3 階層の方式を採用している．第一に toroidal 方向への一次元的な領域分割に対して，MPI で並列する．第二に，分割された toroidal 領域内の粒子群を MPI で並列化する．第三に，ループレベルで OpenMP 並列化する．

GTC-P では, GTC の並列化に加えて，radial 方向を領域分割して並列化している．これにより、GTC に対して，著しく並列化効率が改善している [2]．本稿では，特に GTC-P の C 言語版 (GTCP-C) について，京上で評価した．

2.4　性能分析ツール SCALASCA

Scalasca は，テネシー大学と Julich Supercomputing Centre が開発した並列プログラム向け性能分析ツールである．主に，実行時関数の振る舞いを解析することによって，並列プログラムの性能最適化を支援する．Scalasca は，HPC アプリケーションの主要プログラミング言語である Fortran, C/C++をサポートし，MPI, OpenMP を使用したプログラムに対応している．

Scalasca では，アプリケーションの実行関数別の実行回数，同期待ち時間，通信データ量といったプロファイル結果を，実行時関数のコールパスに沿って表示することができる．本稿では，GTCP-C への適用結果を報告する．

3.　コード解析

3.1　全体性能

最初に，GTCP-C に対して，Weak scaling による全体性能の傾向を確認する．測定条件を，表 2 に示す．

`mpsi` は, radial 方向のグリッドの数を示し, `mthetamax` は, radial 方向に同心円状に描かれたリングの中で最外側の円上におけるグリッドの数を示す．`mzetamax` は，toroidal 方向の最大グリッド数を示す．問題規模は，`mpsi`，`mthetamax`，`mzetamax` によって決まる．表 2 では，512, 2048, 8192, 32768 のそれぞれのノード数に対して，`mzetamax` を固定し，`mpsi`，`mthetamax` をそれぞれ 2 倍づつ増加させていくことで，Weak scaling となる問題規模に対応させている．問題サイズ D は，ITER を想定した規模となっている．時間発展の回数と単位セルあたりの粒子数は，いずれの問題規模でも，それぞれ 100 と 100 としている．

Weak scaling による測定結果を，図 2 に示す．

図 2 では，問題規模が大きくなるにつれて，実行時間が増加し，32768 ノードのときには，512 ノードのときより

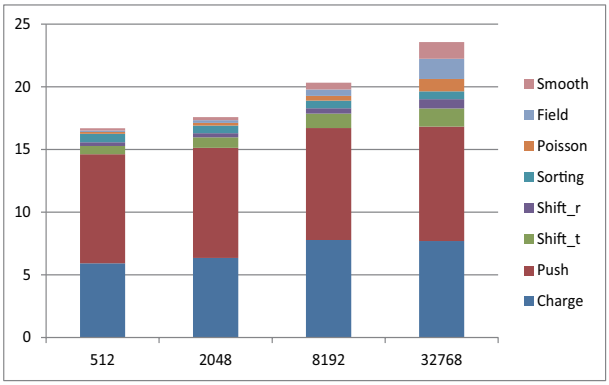

図 2　Weak scaling 性能

表 3　ノード内プロファイル情報

問題規模	A	B	C	D
ノード数	512	2048	8192	32768
浮動小数点演算実行効率 (%)	3.43	3.25	2.79	2.46
命令実行効率 (%)	18.43	17.60	15.53	10.20
浮動小数点演算占有率 (%)	32.92	32.70	31.86	31.10
SIMD 化率 (%)	4.42	4.36	4.30	4.23
Load/StoreSIMD 化率 (%)	0.35	0.36	0.36	0.40
Float+fma SIMD 化率 (%)	4.05	4.01	3.93	3.83
メモリスループット (%)	6.80	6.58	5.94	6.16
総命令数最大差異 (%)	1.3	1.1	2.6	4.9

も 41% 増加している．個別の関数では，charge と push による処理時間が大部分を占めている．実行時間の増加率は，その他の関数の方が，2〜20 倍程度までと著しく増加している．

3.2　ノード内プロファイル情報

次に GTCP-C に詳細プロファイラを用いて，ノード内のプロファイル情報を確認する．表 3 に，表 2 の条件 A から D に対応したプロファイル情報を示す．

表 3 からは，ノード数が大きくなるにつれて，絶対性能を示す浮動小数点演算実行効率，命令実行効率，および浮動小数点演算率が徐々に低下していることが分かる．SIMD 化率はほぼ一定であるが，絶対値としては 4% 台と低く，特にロード・ストアに関しては，1% を大きく下回っており，ほとんど SIMD 化できていないことが分かる．

総命令数最大差異は，ノード間の総命令数の最大差異を示したものである．絶対値を示す数値は小さいものの，8192, 32768 ノードでは，2 倍，4 倍と急激に差異が大きくなっている．ただし，差異の割合が小さいため，これだけでは全体の実行時間に対して 4 割増加する原因になっているか判別できない．

次に SIMD 化率に着目して，時間発展を成している主要関数別では，SIMD 化率がどのような数値を示しているか確認する．表 4 に SIMD 化率に焦点をあてたプロファイル情報を示す．なお，関数名は，shifti_toroidal と shifti_radial をそれぞれ shifti_t と shifti_r に略記している．

表 4　主要関数別の演算性能と SIMD 化率 (単位:%, 条件 A)

関数名	演算効率	浮動小数点演算占有率	SIMD 化率
chargei	2.78	26.9	0.05
pushi	4.55	38.3	7.93
shifti_t	0.73	13.9	0.06
shifti_r	4.18	40.6	0.007
Sorting	1.15	20.5	0.004
poisson	0.92	9.93	0.67
field	0.11	4.63	1.38
smooth	0.164	6.45	0.80

表 5　主要関数別の演算性能と SIMD 化率 (単位:%, 512 ノード+条件 D)

関数名	演算効率	浮動小数点演算占有率	SIMD 化率
chargei	2.37	26.8	0.02
pushi	4.10	38.2	7.90
shifti_t	0.88	14.5	0.190
shifti_r	6.24	42.3	0.002
Sorting	0.97	20.3	0.001
poisson	1.47	10.4	0.58
field	0.36	8.41	1.54
smooth	0.82	12.22	0.66

表 4 では，浮動小数点演算占有率が 20% を超えている関数の中でも pushi,field 以外は，軒並み 1% を大きく下回っており，ほぼ何も SIMD 化がなされていないといってよい．

次にノード数を 512 に固定して，問題規模を大きくしたときの演算特性に変化がないか確認する．表 5 では，表 2 の条件 D に対して，512 ノードで実行したときの結果を示す．

問題規模を大きくした場合でも，個別の関数では，field, smooth で，演算効率と浮動小数点演算効率の数値に大きな変化を確認できるが，SIMD 化率については，表 4 の傾向とほぼ同じである．

3.3　通信プロファイル情報

3.3.1　全体情報

GTC-P の時間発展ループ全体に対する通信関数のプロファイル情報を図 3 に示す．図 3 は，各通信関数に対して，各ノードでの通信時間の平均値を積み上げている．各通信関数の実行時間は，ノード数が増えるにつれてどの関数も増加している．512 ノードでは，図 2 の 1 割以下だった通信の合計実行時間が，32768 ノードでは，3 割程度まで上昇している．また，個別の通信関数の実行時間では，512, 2048 ノードのとき，MPI_Sendrecv と MPI_Allreduce の割合が過半を占めるものの，8192, 32768 ノードのときには，MPI_Barrier と MPI_Waitall といった同期待ち時間の増加が顕著である．GTCP では，通信のオーバーラップがな

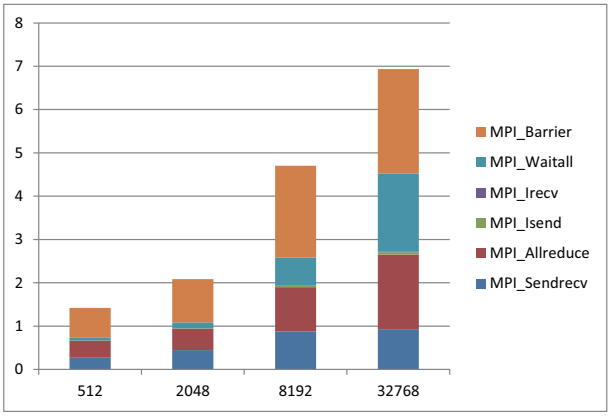

図 3　通信関数プロファイル情報

表 6　各通信関数のデータ通信量と最大差異

		Sendrecv	Allreduce	Isend
A	平均 (kB)	451	0.62	48.7
	最大差異 (倍)	1.08	0	2.00
B	平均 (kB)	493	1.22	53
	最大差異 (倍)	1.21	0	2.00
C	平均 (kB)	577	2.42	59.4
	最大差異 (倍)	1.49	0	2.22
D	平均 (kB)	747	4.82	67.5
	最大差異 (倍)	2.01	0	3.02

されていないため，この通信時間の増加はそのまま並列化効率の劣化要因となる．

次に主な通信関数のノード間の平均データ通信量と最大差異を表 6 に示す．ノード間最大差異は，最大データ通信量が最小データ通信量の何倍かを示す．以降では，データ通信量という場合，同一ノード内かつ指定範囲での，該当通信関数の呼び出し 1 回あたりの通信量の平均値とする．

表 6 では，各関数ともデータ通信量がノード数の増大とともに増加している．ノード間最大差異については，MPI_Sendrecv の差異が線形的に増大していることが分かる．また，MPI_Isend では，各ノード数で 2〜3 倍の著しいアンバランスが発生している．

3.3.2　GTCP-C の主要関数別の通信プロファイル情報

次に個別の通信関数の状況を，GTCP-C の主要関数別に確認していく．GTCP-C を 512 ノードで実行した場合の主要関数別の通信関数の使用状況，各経過時間，平均データ通信量を, それぞれ表 7，表 8，表 9 に示す．表 7，表 8，表 9 は，いずれも全ノードに対する平均値を示している．

表 7, 表 8, 表 9 からは，charge と shift_t が他と比べて相対的に粒度の大きい 1 対 1 通信が行われていると分かる．pushi の MPI_Allreduce の経過時間が突出して大きいのは，使用しているコミュニケータが MPI_COMM_WORLD であるためである．他に，MPI_Allreduce で MPI_COMM_WORLD を使っている関数には，shift_t があるが，ノードあたりのデータ通信量が 8 バイトと小さいため，実行時間も小さい．pushi と shift_t 以外で，MPI_Allreduce を使っている関数は，MPI_COMM_WORLD を分割したコミュニケータを使用している．

表 7　GTCP-C の関数構成と主な通信関数の単位時間発展あたりの呼び出し回数 (条件 A)

関数名	MPI_Sendrecv	MPI_Allreduce	MPI_Waitall
chargei	2	4	8
pushi	0	3	0
shifti_t	8	2	0
shifti_r	8	2	0
poisson	0	0	16
field	6	0	4
smooth	12	1	8

表 8　GTCP-C の主要関数における各通信関数の経過時間 (単位:秒, 条件 A)

関数名	MPI_Sendrecv	MPI_Allreduce	MPI_Waitall
chargei	0.025	0.075	0.016
pushi	0	0.239	0
shifti_t	0.163	0.043	0
shifti_r	0.018	0.005	0
poisson	0	0	0.021
field	0.025	0	0.012
smooth	0.039	0.002	0.013

表 9　GTCP-C の主要関数における主な通信関数のデータ送信量 (単位:kB, 条件 A)

関数名	Sendrecv	Allreduce	Isend
chargei	162	1.46	48.7
pushi	0	0.54	0
shifti_t	1670	0.01	0
shifti_r	22.6	0.01	0
poisson	0	0	24.4
field	205	0	146
smooth	193	0.02	97.4

表 10　GTCP-C の主要関数における各通信関数の経過時間 (単位:秒, 条件 C)

関数名	MPI_Sendrecv	MPI_Allreduce	MPI_Waitall
chargei	0.064	0.197	0.064
pushi	0	0.368	0
shifti_t	0.508	0.211	0
shifti_r	0.025	0.023	0
poisson	0	0	0.021
field	0.118	0	0.081
smooth	0.131	0.009	0.064

GTCP-C を 8192 ノードで実行した場合の，各経過時間を表 10 に示す．なお，主要関数別の通信関数の呼び出し回数は，表 7 と等しい．

8192 ノードの場合には，shift_t の合計通信時間が pushi を上回っている．shift_t において，MPI_Sendrecv と MPI_Allreduce の増加率が 3〜5 倍に達し，pushi の増加率を大きく上回っているためである．

次に GTCP-C の主要関数における各通信関数のノード

表 11　GTCP-C の主要関数における各通信関数のデータ通信量と最大差異 (条件 C)

		Sendrecv	Allreduce	Isend
chargei	平均 (kB)	499	5.78	59.4
	最大差異 (倍)	6.16	0	1.64
pushi	平均 (kB)	0	1.98	0
	最大差異 (倍)	0	0	0
shifti_t	平均 (kB)	1650	0.01	0
	最大差異 (倍)	1.06	0	0
shifti_r	平均 (kB)	99.8	0.01	0
	最大差異 (倍)	13.56	0	0
poisson	平均 (kB)	0	0	59.4
	最大差異 (倍)	0	0	2.52
field	平均 (kB)	542	0	178
	最大差異 (倍)	4.76	0	2.52
smooth	平均 (kB)	418	16	119
	最大差異 (倍)	3.47	0	2.52

表 12　GTCP-C の主要関数における各通信関数の経過時間 (単位:秒, 512 ノード+条件 D)

関数名	MPI_Sendrecv	MPI_Allreduce	MPI_Waitall
chargei	0.595	1.879	0.052
pushi	0	24.641	0
shifti_t	9.147	1.465	0
shifti_r	0.078	0.012	0
poisson	0	0	0.062
field	0.508	0	0.071
smooth	0.617	0.019	0.052

間平均データ通信量とノード間最大差異について，表 11 に示す．

表 11 では，表 6 の最大差異が大きいときの状況を確認するため，8192 ノードの場合を示す．

MPI_Sendrecv では，ノード間平均データ通信量の最も大きい shift_t の最大差異が 1.06 倍とほとんどない．その他の関数では，最大差異が 4.8 倍から 13.6 倍と非常に高い．MPI_Isend では，使用しているすべての主要関数で，最大差異が 1.6 倍から 2.5 倍と大きい．図 3 の結果は，ノード数が小さいときには，全体として shift_t に隠れていたものが，ノード数が大きくなるにつれて，データ通信量のアンバランスが原因で起こる待ち時間が顕在化していったと考えられる．

次にノード数を 512 に固定して，問題規模を大きくしたときの通信時間の傾向を見ていく．表 12 は，表 2 の条件 D を 512 ノードで実行したときのプロファイル情報である．

表 12 からは，MPI_Sendrecv と MPI_Allreduce に対する MPI_Waitall の経過時間の差が，表 8 のときよりも著しく大きくなっていることがわかる．したがって，GTCP-C の通信処理を対策する場合には，MPI_Sendrecv と MPI_Allreduce の合計時間が相対的に大きな pushi および shift_t が関わる処理を優先的に対策していく必要がある．

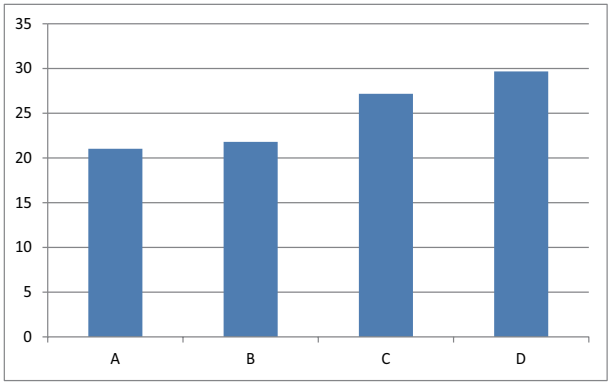

図 5　Mira での GTCP-C の Weak scaling 結果 (単位:秒)

3.4　Scalasca の適用

Scalasca では，表 11 で示した結果に対して，特に，各ノードではどのような分布となっているか GUI で確認する．

図 4 では，条件 C(8192 ノード実行) の適用結果を示す．ネットワークトポロジーの形状は，3 次元形状の 32x16x16 を指定した．図 4 の右にある 3 次元図形は，field 関数内の MPI_Sendrecv のノード別データ送信量の分布を示している．データ送信量は，図 4 下部の色彩によって示され，赤くなるほど多く，青くなるほど少なくなる．上下限の範囲は，5 倍に収まる範囲を指定している．図 4 からは，データ送信量の負荷の上限付近のノード，下限付近のノードが特定の範囲に偏在していることがわかる．GTCP-C の他の主要関数の最大差異が大きな通信関数についても，同様な偏在があることを確認した．

4.　考察と今後の課題

詳細プロファイラおよび Scalasca でコード解析した結果，二つの課題があることが分かった．第一に，相対的に粒度の小さい 1 対 1 通信で，ノード間の負荷バランスが悪いことが挙げられる．これにより，ノード数が比較的少ないときには，影響は小さいものの，ノード数が増加するにつれて各ノードの待ち時間に対する影響が増加していると考えられる．また，現在の通信処理は，計算とオーバーラップする構造とはなっていないため，今後の並列化効率改善を考える場合，アルゴリズムの見直しが必要である．

第二に，ノード内では，SIMD 化率が 5% 以下とほとんど SIMD 化できていない．特にロード・ストアや，pushi，field 以外の主要関数では，1% を大きく下回っており，このことが，全体実行効率が 4% 以下に留まっている一因になっていると考えられる．

GTCP-C は，TOP500 [6] の 1,2,4 位 (2013 年 5 月現在) に位置する Titan, Sequoia, Mira で動作実績がある．図 5 に Blue Gene/Q システム Mira での実行結果を示す．図 5 の実行条件は，表 2 と同じである．なお，1MPI プロセスあたり 64 スレッド実行としている．

図 5 でも，図 2 と同様に，ノード数が増加するにつれて，

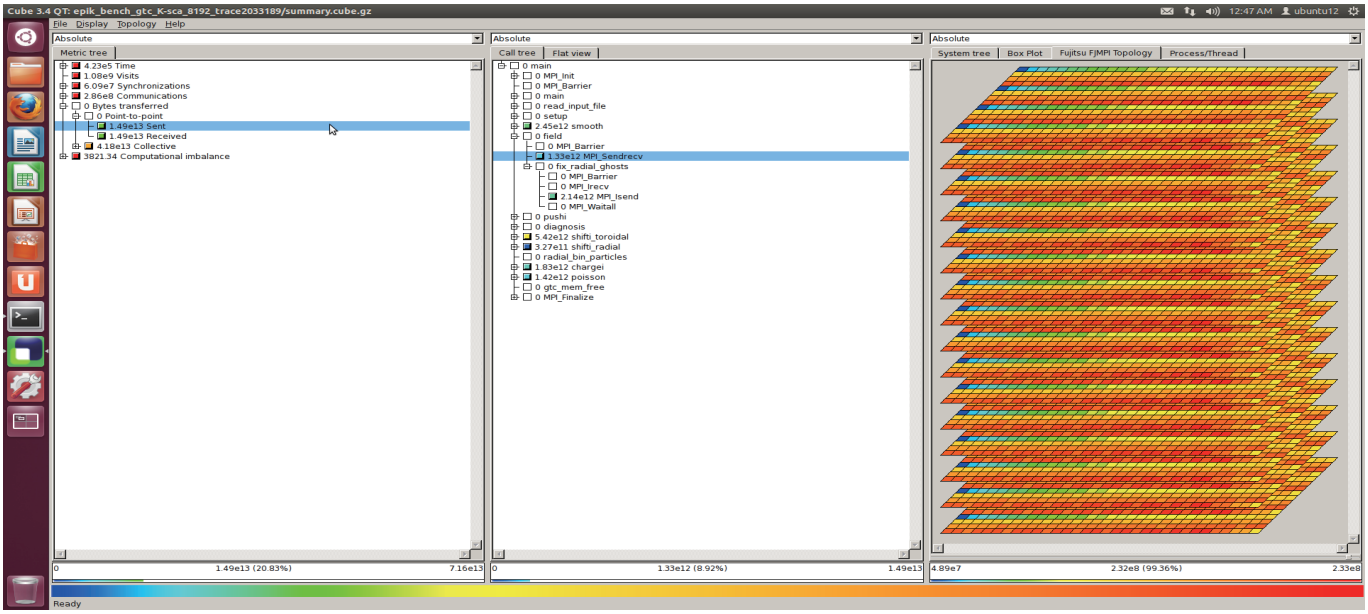

図 4　条件 C(8192 ノード実行) での Scalasca 適用結果

実行時間が徐々に増加する傾向があることから，Mira での実行でも，通信に関しては，京のときと同様の課題が発生していると考えられる．

以上のことから，今後の課題としては，通信負荷バランスと SIMD 化に着目したアルゴリズムの改善が挙げられる．

5. まとめ

「京」のシステム標準のプロファイラを使用して，GTCP-C のコード解析を実施した．その結果，現状の GTCP-C の全体および主要関数別の性能特性と二つのプログラム上の課題を提示した．また，Scalasca を使用して，各実行関数別の負荷バランスなどの性能特性を確認した．

謝辞　本論文の結果の一部は，理化学研究所のスーパーコンピュータ「京」を利用して得られたものです．本論文に対して，ご支援いただいた関係各位に感謝します．

参考文献

[1] S. Ethier, W. Tang, and Z. Lin: Gyrokinetic particle-in-cell simulations of plasma microturbulence on advanced computing platforms, Journal of Physics:Conference Series, 16:1-5 (2005).
[2] S. Ethier, M. Adams, J. Carter, and L. Oliker: Petascale Parallelization of the Gyrokinetic Toroidal Code, VECPAR'10: 9th International Meeting High Performance Computing for Computational Science (2010).
[3] The ITER project: http://www.iter.org/.
[4] 高瀬亮，横川三津夫 (編)：特集：スーパーコンピュータ「京」，情報処理, Vol.53, No.8 (2012).
[5] The Scalasca Development Team: *Scalasca 1.4 User Guide*, http://www.scalasca.org/download/documentation (2011).
[6] Top500 Supercomputer Sites: http://www.top500.org/.