Vol.2009-HPC-120 No.3
2009/6/12

姫野ベンチマークの GPU マルチノード実行における 通信と演算のオーバーラップによる高速化 ～ 32GPU で 700GFLOPS 超を達成 ～

加藤季広[†] 青木尊之[††] 額田彰[††]
遠藤敏夫[†††] 松岡聡[††] 長谷川篤史[§]

GPU マルチノードシステム上でプログラムを実行する際は，GPU-GPU 間の直接通信が行えないため，CPU を介してのノード間通信が必要となる．CPU-GPU 間の通信が頻繁に生じ，スケーラビリティを向上させる上でのボトルネックとなりうる．姫野ベンチマークを対象に，通信と演算を同時に実行することで通信時間を隠蔽し，スケーラビリティの向上を試みた．サイズ XL の，配列サイズが最長となる次元をZ方向からX方向に変更した問題について測定を行った. NVIDIA Tesla S1070 32GPU を使用し，709GFLOPS の実効性能（実効メモリバンド幅 1.17TByte/s に相当）を実現した．通信と演算のオーバーラップ実行による性能向上は 55%超となる．この手法は，通信と演算を同時に処理できる他の問題についても適用可能であり，GPU マルチノードシステム上での高並列化実行について，その可能性を示すことができた．

Acceleration of Himeno Benchmark on Multi-node GPU System by Overlapping Communication with Calculation
- Over 700 GFLOPS of Sustained Performance is Achieved with 32 GPUs -

Toshihiro Kato[†], Takayuki Aoki[††], Akira Nukada[††],
Toshio Endo[†††], Satoshi Matsuoka[††]
and Atsushi Hasegawa[§]

When programs run on GPU multi-node system, communication between CPU and GPU occurs frequently because direct communication between GPU and GPU is unusable and inter-node communication via CPU is inevitable. This can become a bottleneck in improving scalability. We tried to improve scalability of Himeno benchmark by overlapping communication with calculation and hiding communication time behind calculation time. As a result, 709 GFLOPS of sustained performance (corresponding to 1.17 TByte/s of sustained memory bandwidth) with 32 GPUs of NVIDIA Tesla S1070 is achieved about the problem that longest dimension is exchanged from Z-direction to X-direction based on size XL. Increase in performance by such a overlapping is over 55%. Such an approach can be applied to other problems that its communication and calculation can be operated simultaneously. We've succeeded in showing potential of high scalability execution on GPU multi-node systems.

1. はじめに

近年，GPU(Graphics Processing Unit)を汎用計算に用いるGPGPU(General-Purpose computation on GPU)[1]が注目されている．NVIDIAがGPU上で汎用計算を行なうためのフレームワークであるCUDA(Compute Unified Device Architecture)[2]をリリースする等，GPGPU利用のための環境が整いつつある．

GPGPUが注目される理由としては，非常に高い単精度演算性能と，デバイスメモリの高バンド幅が挙げられる．例えば，本論文中の測定で使用したNVIDIA Tesla S1070は，1GPUあたり 1.04TFLOPSの単精度演算性能，102.4GByte/sのデバイスメモリバンド幅を持つ[3].

しかし，その活用にあたっては，まだいくつかの克服すべき問題点がある．その一つとして, PCIバス経由で行われるCPU側のメインメモリとGPU上のデバイスメモリ間のデータ通信がCPU側のメインメモリへのアクセスに対して相対的に遅いため，このような通信が頻繁に生じるアプリケーションではその性能を発揮できないということが挙げられる．CPU-GPU間の通信時間が支配的となり，GPUの使用がアプリケーション全体の性能向上に結びつかない[4]．GPGPUに限らず，アクセラレータを用いた計算一般に伴う問題である．

上記のような性質は, 複数の GPU を用いた大規模計算を行う際にも問題となりうる. MPI を用いたマルチノード並列化を適用する際は，GPU-GPU 間の直接通信が行えないため CPU を介してのノード間 MPI 通信が必要となり, 必然的に CPU-GPU 間通信を

[†] 日本電気株式会社 HPC事業部
HPC Division, NEC Corporation
[††] 東京工業大学 学術国際情報センター
Global Scientific Information and Computing Center, Tokyo Institute of Technology
[†††] 東京工業大学 情報理工学研究科
Graduate School of Information Science and Engineering, Tokyo Institute of Technology
[§] 株式会社NEC情報システムズ 先端技術ソリューション事業部
Advanced Technology Solutions Division, NEC Informatec Systems, Ltd.

伴うためである．前述の「CPU-GPU 間通信が頻繁に生じる」状況となる．何らかの施策を講じ，CPU-GPU 間通信の影響を緩和しなければならない．

本論文では，Poisson方程式をJacobi反復法で解く場合の主要ループを模したベンチマークコード，姫野ベンチマーク[5]を対象に，CPU-GPU間通信およびMPI通信をGPU上での演算と同時に行い，演算時間中に通信時間を隠蔽することで，GPUマルチノードシステム上でのスケーラビリティ向上を試みた．測定に用いたGPUマルチノードシステムは，GPUを搭載した大規模計算機システム，東京工業大学学術国際情報センター TSUBAMEスーパーコンピュータ[6]である．

姫野ベンチマークについては，これまでにもGPU上での性能評価結果が複数報告されている．主なものとして，小川らによる単一ノード内最大 4GPUでの性能評価結果[7]や，単一GPU上での最適化手法を追求した成瀬らによる成果[8]が挙げられる．しかし，GPUマルチノードシステムにおける評価結果はこれまでに報告されていない．

今後，さらなる高並列化を試みる上でマルチノードでの性能評価は必須であり，ここで得られる知見は，姫野ベンチマークだけでなく他のアプリケーションに対しても適用できる可能性がある．

2. 姫野ベンチマークの概要および問題サイズ

姫野ベンチマークにおける主要ループでは，ループ 1 回あたり 34 回の浮動小数点演算と 31 要素のメモリread，1 要素のメモリwriteが必要となる．しかし，圧力の配列pについては，18 点の隣接要素参照があり，ここを適宜再利用することで 13 要素のメモリread，1 要素のメモリwriteに減少させることができる[7].

上記の計算を 3 次元格子の各点について行う．既定の問題サイズとして，表 1に示す 4 種類が用意されている．

表 1 姫野ベンチマークの既定問題サイズ

サイズ	X×Y×Z
S	65×65×129
M	129×129×257
L	257×257×513
XL	513×513×1025

今回対象としたのは，サイズXLにおける長さが最長となる次元をZ方向からX方向に変更したサイズXL相当（表 2）である．

表 2 評価対象とした問題サイズ

サイズ	X×Y×Z
XL 相当	1025×513×513

GPU マルチノードシステムにおける並列化のため，MPI を利用した分散メモリ並列化を適用する．上記の 3 次元格子をプロセス数だけの小領域に分割し，各小領域を各プロセスに割り当てる．今回の測定においては，X 方向のみを分割した．

メモリ上でデータが連続していない X 方向のサイズを拡大し，分割したのは以下のような理由による．

- 分割数を増加させてもメモリ連続方向のループ長が保たれる．
- MPI 通信の対象となる断面が，Y×Z=513×513 となり，他の方向で分割する場合よりも通信量が小さい．

X 方向のみでの分割を行ったため，通信の対象となる境界領域のサイズは並列数を増加させても減少しない．しかし，各 GPU 上で行われる計算の量は並列数にほぼ反比例して減少するため，並列数をある程度増やした段階で通信時間が全体の実行時間に対して支配的になる．通信時間をどれだけ隠蔽できるかが，スケーラビリティを向上させる上での最重要課題となる．

演算については，単一 GPU での性能を向上させるため，主に以下のような最適化を適用した．

(1) Shared memory の使用

Shared memory とは，1 つの SM (Streaming Multiprocessor)内の 8 つの SP (Streaming Processor)で共有される，高速，小容量のメモリ領域である．Shared memory のアクセス速度はレジスタ並み，サイズは SM あたり 16KByte である．

隣接要素の参照がある配列pについて，適宜データを再利用するためにShared memoryを利用した[7].

3 次元格子各点で行われる計算を，X×Y×Z=4×8×64 の小領域に分割して行い，そこで参照される配列 p のデータを Shared memory 上にプリフェッチした．各小領域の計算を行う際に必要な Shared memory のサイズは，領域サイズ各次元の両端に 1 つの隣接要素を加えた(4+2)×(8+2)×(64+2)×4 [Byte]，約 15.5Kbyte となり 16Kbyte に収まる．

(2) コアレスアクセス化

デバイスメモリへのアクセスを高速に行うためには，スレッド間の協調メモリアセス，コアレスアクセスを用いる必要がある．

コアレスアクセスを発動させるためには様々な条件を満たす必要があるが，今回問題となったのは，アクセス開始領域のアドレスが 64Byte（単精度実数型で 16 要素）

の倍数でなければならないという制限である[9].

コアレスアクセスを発動させるため，配列における連続方向（Z 方向）のサイズが 16 の倍数になるように冗長な要素を挿入した．サイズ XL 相当における Z 方向のサイズは 513 であり，16 の倍数とならない．15 個の冗長要素を挿入し，サイズを 528 とした．

3. 通信と演算のオーバーラップ実行による通信時間の隠蔽

通信と演算のオーバーラップ実行を実現するため，姫野ベンチマークをGPUに移植した直後の，オーバーラップ実行を考慮していないコードを元に，ループ 1 回の処理を図 1のように変更した．

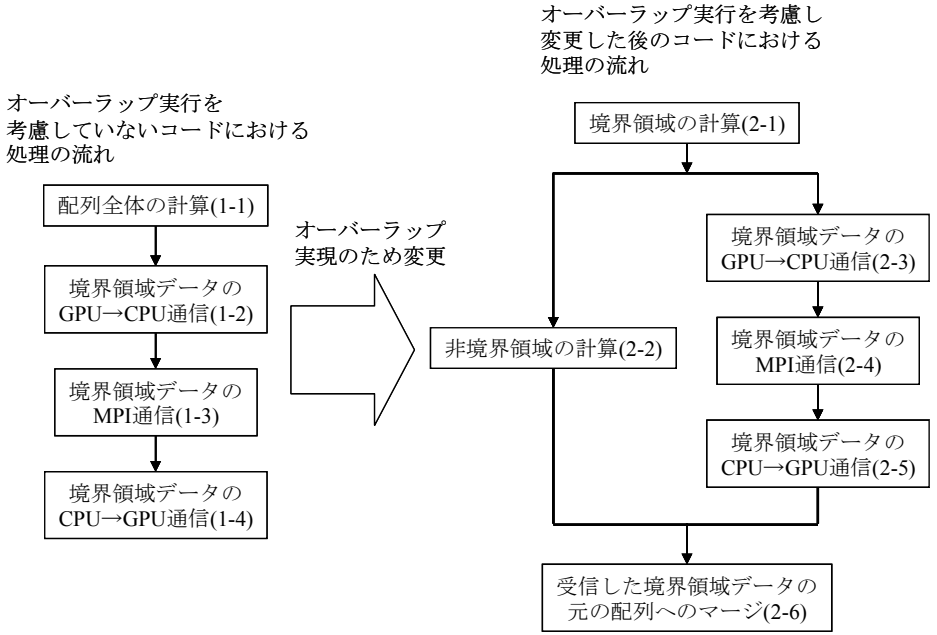

図 1 オーバーラップ実現のための変更

配列全体の計算(1-1)を，MPI 通信の対象となる境界領域の計算(2-1)とその他非境界領域の計算(2-2)に分けて行う．結果として，境界領域の計算(2-1)の処理が終わり次第，一連の通信処理(2-3～2-5)を開始することができ，これらは非境界領域の計算(2-2)と同時に行うことができる．

図 2は，このステップにおいて，GPU，CPUそれぞれで行われる処理の流れを示したタイムチャートである．GPU上での計算に相当する処理は黒の矢印，CPU-GPU間通信を赤の矢印，MPI通信を青の矢印で示している．図中の数値は，サイズXL相当を 16GPUで実行した際の実行時間を示している．（詳細は第5節で示す表 3を参照．）非境界領域の計算（図中(2-2)）と一連の通信処理（図中(2-3)～(2-5)）が同時に行われている．非境界領域の計算時間が通信時間に対して十分長く，通信時間を隠蔽可能である状況を表している．

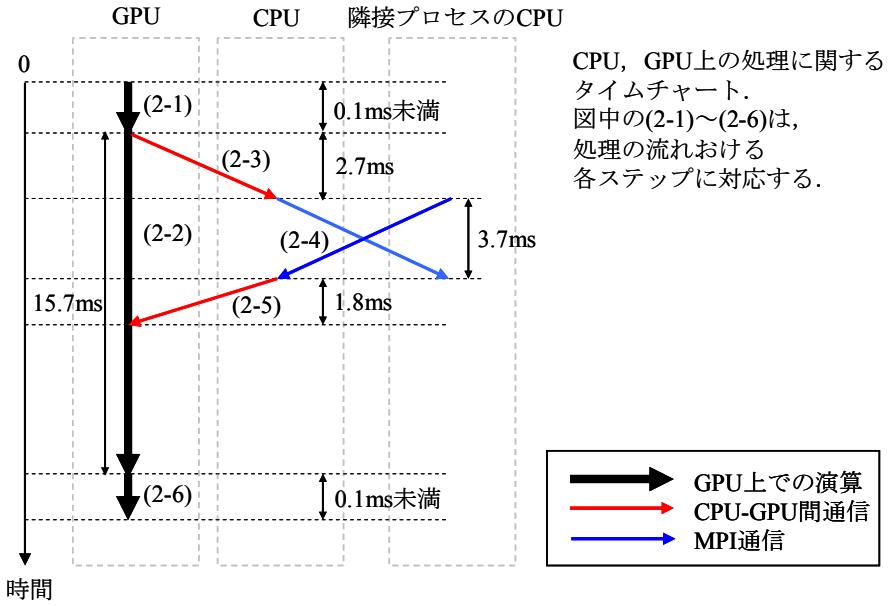

図 2 処理の流れに関するタイムチャート（16GPU 使用時）

CUDA対応GPUで通信と演算を同時に行うためにはstream機能[9]を利用する．Streamは依存関係のある一連の処理を定義するものであり，複数作成可能である．CPU-GPU間通信(2-3 および 2-5) については，CUDAでサポートされている非同期メモリコピー命令cudaMemcpyAsyncを使用し，非境界領域の計算(2-2)と別のstreamに投入することによって，これらの処理の間に依存関係がないことを示す．非同期通信が可能となり，

CPU-GPU間通信は非境界領域の計算と同時実行される．

GPU 上での演算は CPU 上での処理とは非同期で実行されるため，MPI 通信(2-4)は非同期で行われる．

4. 測定結果

測定環境を以下に示す．

東京工業大学学術国際情報センター TSUBAMEスーパーコンピュータ[6]
ホストノード: Sun Fire X4600 最大 32 ノード
Dual-Core Opteron 2.4GHz×8
ネットワーク: InfiniBand 4x SDR（10Gbps 双方向）×2
GPU: NVIDIA Tesla S1070（4GPU 搭載）
単精度浮動小数点演算ピーク性能: 1GPU あたり 1.04TFLOPS
デバイスメモリ: 1GPU あたり 4GByte
デバイスメモリバンド幅: 1GPU あたり 102.4GByte/s
ホストノードの仕様により PCI-Express 1.1 x8 モード(2GByte/s)で接続

最大 32GPU までの測定を行った. 測定環境では単一のホストノードに 2GPU が接続されているが，1 ホストノードあたり 1GPU のみを使用した．例えば，4GPU の測定ではホストノード 4 ノードを使用している．

サイズXL相当（表 2参照）について，通信と演算のオーバーラップ実行を行った場合と行わなかった場合，それぞれにおける測定結果を図 3に示す．

通信時間に対して演算時間が十分長い 4GPU の場合では，通信と演算のオーバーラップ実行による性能向上は 15%程度に止まるが，演算時間が短縮される 16GPU，32GPU の場合では 55%超の性能向上が得られている．

プロセス数の増加に関わらず MPI 隣接通信の時間が一定となるような理想的な状態では，並列数の増加にかかわらず通信時間が一定となる．オーバーラップ非適用版では，通信時間が逐次実行時間として直接表れるためスケーラビリティは低い．4GPU から 32GPU での性能向上は，ピーク性能 8 倍に対して 3.45 倍に止まっている．

オーバーラップ適用版については，4GPU から 8GPU ではピーク性能 2 倍に対して性能向上は約 1.92 倍となり，よくスケールしている．しかし，8GPU から 16GPU では性能向上は 1.72 倍，16GPU から 32GPU では性能向上は 1.35 倍と，スケーラビリティが悪化している．

4GPU から 32GPU での性能向上は 4.66 倍，32GPU 使用時で実効性能 709GFLOPS を実現した．実効メモリバンド幅は 1.17TByte/s となる．

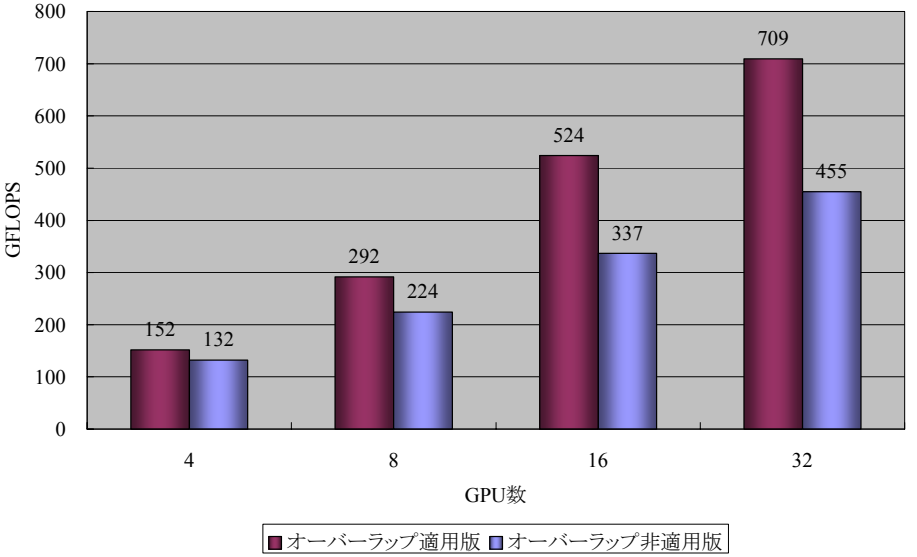

図 3 サイズ XL 相当測定結果

5. 問題規模の拡大

通信時間の影響を見るため，通信に相当する処理を削除した版についても測定を行い，オーバーラップ適用版と比較した．結果を図 4に示す．通信時間が完全に隠蔽できれば通信を削除した場合と同等の性能が得られるが，理想値としての通信削除版との乖離を見ると，並列数の増加に伴って乖離が大きくなっている．32GPUでは理想値との乖離が 30%を超えている．

第3節で述べた 1 サイクル内の各処理における実行時間表 3に示す．通信に相当する項目については実効バンド幅も示す．

境界領域の計算(2-1)，非境界領域の計算(2-2)，GPU→CPU 通信(2-3)，CPU→GPU 通信(2-5)および元の配列へのマージ(2-6)については，コード中にタイマを挿入して測定を行った．

MPI 通信(2-4)については，実際の状況を模したテストプログラムを作成し，測定を行った．各プロセスが，隣接する 2 プロセスとの間でそれぞれ 1MByte のデータ送受信を行う．4～32 並列の場合について測定を行ったが，全てのケースでほぼ同等の結果が得られた．バンド幅を算出する際は，送受信される 4MByte を測定時間で割るこ

とで算出した．

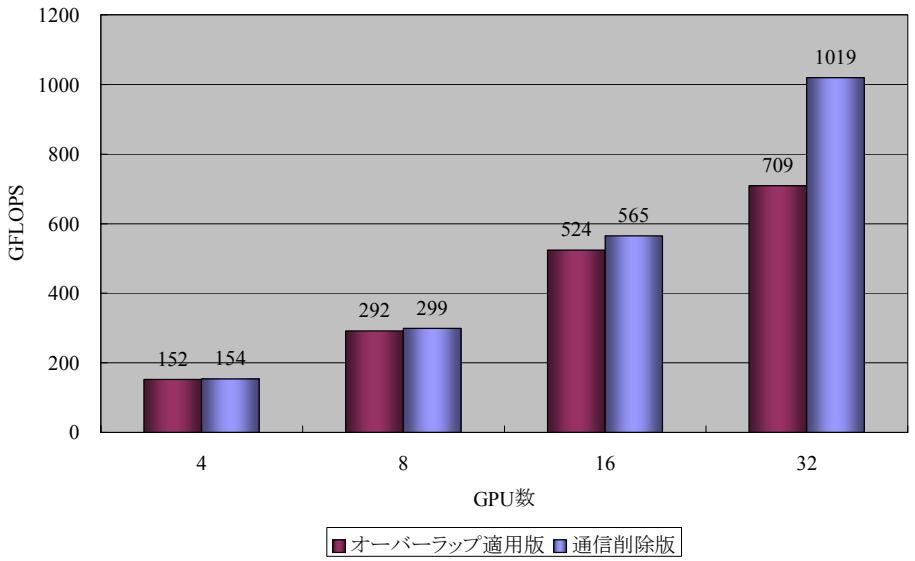

図 4 サイズ XL 相当における理想値との乖離

表 3 各処理の実行時間

処理の内容		実行時間	実効バンド幅
境界領域の計算(2-1)		0.1ms 未満	-
非境界領域の計算(2-2)	16GPU 使用時	15.7ms	-
	32GPU 使用時	8.6ms	-
境界領域データの GPU→CPU 通信(2-3)		2.7ms	0.72GByte/s
境界領域データの MPI 通信(2-4)		3.7ms	1.09GByte/s
境界領域データの CPU→GPU 通信(2-5)		1.8ms	1.10GByte/s
受信した境界領域データの元の配列へのマージ(2-6)		0.1ms 未満	-

2-3～2-5 の通信時間は合計 8.2ms程度となる．この通信時間を隠蔽すべき非境界領域の計算時間(2-2)は，32GPU実行時で 8.6msとなり通信時間と同程度となる．図 4に示した理想値との乖離を見ても，32GPU時は通信時間が全体の実行時間に表れていると考えられる．

通信時間を隠蔽するのに十分な演算量が保たれるような状況では，理想値に近い性能が得られるはずである．確認のため，サイズXL相当についてX方向を 2 倍に拡大したサイズXXL（表 4）を定義し，最大 32GPUまでの測定を行った．結果を図 5に示す．比較のため，図 3におけるサイズXL相当の性能も併せて示している．

表 4 独自に定義したサイズ XXL の問題サイズ

サイズ	X×Y×Z
XXL	2049×513×513

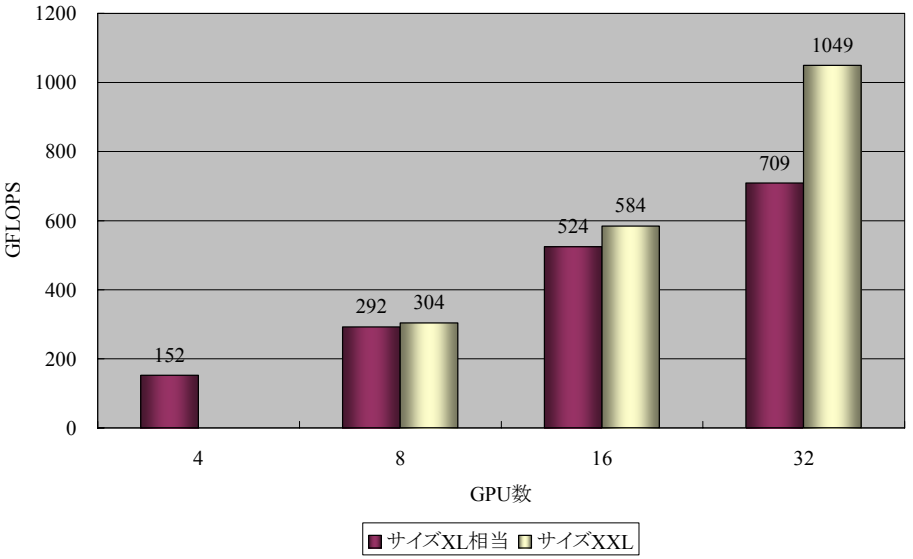

図 5 サイズ XXL 測定結果

16GPU から 32GPU での性能向上が 1.79 倍と，サイズ XL 相当における 1.35 倍と比較して改善している．32GPU 使用時で実効性能 1.05TFLOPS を実現した．実効メモリバンド幅は 1.73TByte/s となる．

サイズXXLについても，通信を削除した理想値との比較を行った．結果を図 6に示す．

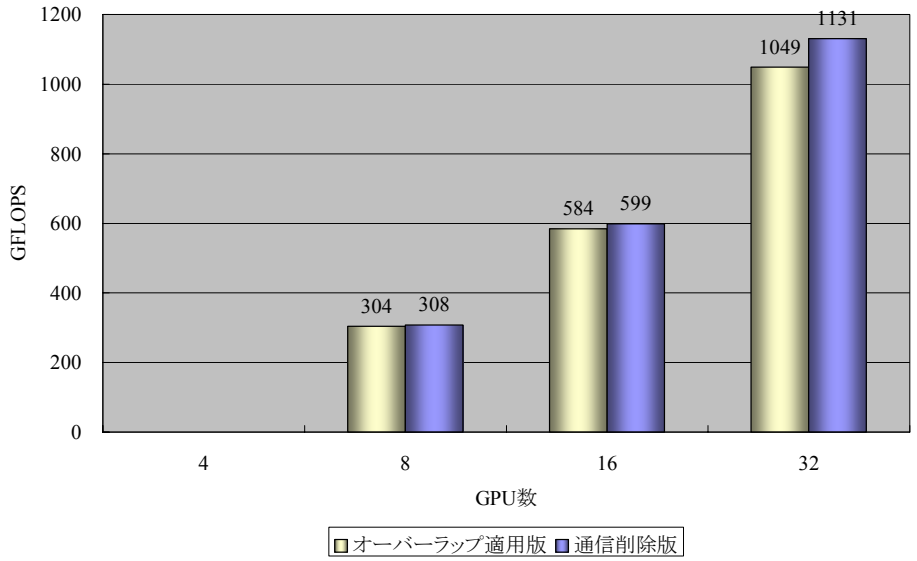

図 6 サイズ XXL における理想値との乖離

サイズ XL の場合，32GPU での理想値との乖離は 30%超となっていたが，通信時間を隠蔽するに十分な演算が行われるサイズ XXL では，理想値との乖離は 7%程度に止まっている．

6. おわりに

姫野ベンチマークについて，マルチノード最大 32GPU での性能を評価した．

通信と演算のオーバーラップ実行で通信時間を演算時間で隠蔽することによって，高並列時に 55%超の性能向上が見られ，並列数の増加に応じた性能向上が得られた．「サイズ XL 相当では 32GPU 使用時に 709GFLOPS（実効メモリバンド幅 1.17TByte/s に相当）」を実現した．

サイズ XL 相当では 32GPU 使用時に大幅なスケーラビリティの低下が見られたが，これは並列数の増加に伴う演算時間の短縮によって，通信時間の影響が表れたためである．サイズ XXL における測定で確認されたように，演算量と通信量の比率を調整することでより高い実効性能が得られる．

サイズ XL 相当ではまず通信時間の改善が必要となる．以下に示す施策により，更なる性能向上の可能性がある．

- 今回使用した環境では，PCI-Express 1.1 x8 に GPU および InfiniBand SDR のネットワークカードを接続して使用している．現在の最新ハードウェアを使用し，PCI-Express 2.0 x16 に InfiniBand QDR のネットワークカードを接続して使用した場合，通信時間が支配的になっている状況を改善することができる．
- 現状では，サイズXL相当，32GPU実行時の非境界領域の演算性能は 1GPUあたり 34GFLOPS程度であり，実効メモリバンド幅は 56GByte/s程度である．NVIDIA Tesla S1070 の 1GPUあたりのデバイスメモリバンド幅は 102.4GByte/sであり，効率は 54%程度に過ぎない．単一GPUでの最適化により，メモリバンド幅の実効効率 81%を実現したという報告[8]があり，さらなる最適化の可能性が残されている．

本論文では最大 32GPU までの評価を行ったが，今後さらなる高並列化を目指す際は，2 次元分割が必須となる．分割する次元が増加すると，メモリアクセスのストライド幅が大きい境界領域計算が加わることや，通信と演算のオーバーラップ実行によって隠蔽できない処理が増加することが考えられ，3 次元分割が有効となる状況は限られる．

本論文で述べた通信と演算のオーバーラップ実行による高速化は，通信と演算を同時に処理できる格子 QCD 等の問題についても応用可能であり，GPU マルチノードシステム上での高並列化実行について，その可能性を示すことができた．

参考文献

1) General-Purpose Computation on Graphics Hardware: http://gpgpu.org/
2) NVIDIA CUDA Zone: http://www.nvidia.com/object/cuda_home.html
3) NVIDIA Tesla S1070: http://www.elsa-jp.co.jp/products/hpc/tesla/s1070/index.html
4) 青木尊之: フル GPU による CFD アプリケーション，情報処理 Vol.50 No.2 Feb. 2009, pp.107-115 (2009)
5) Himeno Benchmark: http://accc.riken.jp/HPC/HimenoBMT/index.html
6) 松岡聡: TSUBAME の飛翔：ペタスケールへ向けた「みんなのスパコン」の構築，情報処理学会研究報告，2006-HPC-107，pp.37-42 (2006)
7) 小川慧，青木尊之: CUDA による定常反復 Poisson ベンチマークの高速化，情報処理学会研究報告，2008-HPC-115，pp.19-23 (2008)
8) 成瀬彰，住元真司，久門耕一: GPGPU 上での流体アプリケーションの高速化手法，情報処理学会研究報告，2008-HPC-117，pp.49-54 (2008)
9) NVIDIA: CUDA Programming Guide 2.0: http://www.nvidia.com/object/cuda_develop.html