Vol.2010-HPC-124 No.18
2010/2/23

GPU クラスタにおける科学技術計算の自動最適化

渡辺祐也[†1]　遠藤敏夫[†1]　松岡聡[†1 †2]

高性能計算においてグラフィックプロセッサ(GPU)の利用が，その高い計算性能とメモリバンド幅のために注目されている．さらにクラスタやスパコンに GPU を多数搭載し，並列利用する試みを数多く行われている．一方で，そのような複雑化したシステムにおいてはチューニングパラメータが増大するため，自動チューニングの必要性が増す．本稿では，GPU を用いた三次元熱拡散方程式プログラムを題材とする．対象とするプログラムは，GPU と CPU を併用する，および効率化のために PCI-Express 通信と MPI 通信の並列化を行う，という特徴を持つ．このプログラムについて自動チューニングを行った予備評価の結果と最大 32GPU で実行した場合の結果を示す．

Auto-Tuning of a Scientific Application on GPU clusters

Yuya Watanabe[†1] , Toshio Endo[†1] and Satoshi Matsuoka[†1 †2]

Graphics processors (GPUs) recently attract much a ttention i n high performance computing area, for the ir ex cellent pe rformance an d memory band width. Also, ther e have b een many attempts that use a lo t of GPUs on clus ters o r su percomputers for parallel applicat ions. Fo r su ch p urposes, auto-tuning methodology is getting more important, since tuning parameters on such systems are increasing. This paper picks up a three-dimensional heat equ ation p rogram as the tar get of tuning. The pr ogram h as the following p roperties: it cooperatively u ses GPUs and C PUs fo r computation, and PCI-Express co mmunication and M PI communication are done in parallel to reduce overhead. This p aper shows results of p reliminary exper iments with auto-tuning and performance of parallel execution with up to 32 GPUs.

1. はじめに

近年，GPU による科学技術計算が広く注目されている．グラフィック処理用に開発されてきた GPU は，今では CPU を上回る高い理論ピーク性能とメモリバンド幅をもつようになった．GPU を開発している NVIDIA や AMD も GPU を使った汎用計算(GPGPU) に注目し，特に NVIDIA が CUDA プログラミング環境を発表して以降，GPU を用いた汎用計算に関する研究報告が増加している．その対象は BLAS, FFT, LAPACK などのカーネル計算から物理シミュレーション，データベース，暗号演算などのアプリケーションに至るまで多岐にわたる．

クラスタやスパコンを用いた HPC 分野においても，限られた電力内で高い性能を発揮するために，複数 GPU による GPGPU は大きな注目を集めている．現時点で GPU を搭載したスーパーコンピュータとして，次の事例が挙げられる．東工大のスーパーコンピュータ TSUBAME は NVIDIA Tesla S1070 GPU を 680 デバイス(170 ユニット)備えている．国外においても，中国の天河スーパーコンピュータに AMD の GPU が搭載されている．また Oak Ridge National Laboratory は次世代の GPU を搭載した大型クラスタの構築を計画している (a)．このような，汎用 CPU をベースとしつつ，GPU の搭載により省電力性や高性能を実現するシステムは増加すると予想される．しかし，そのような環境ではシステムのコンポーネントが増加し，それに伴うチューニングパラメータの増大は，性能チューニングをますます困難とする．たとえば TSUBAME において CPU と GPU を併用した Linpack ベンチマークの性能評価が報告されている[1]が，その多数のパラメータのチューニングは手動で成されている．Linpack よりも複雑なアプリケーションではチューニングはさらに困難になるであろう．以上の議論から GPU を搭載した並列計算環境における性能チューニング，特に自動チューニングに関する知識の蓄積が求められている．

本研究では，多数の GPU 搭載ノードを用いた三次元熱拡散方程式プログラムを，MPI および CUDA を用いて記述し，チューニングの対象とする．対象とするプログラムは，複数ノードにまたがった GPU と CPU の両方を併用する，および効率化のために GPU-ホスト間通信と MPI 通信の並列化を行う，という特徴を持つ．ステンシル計算における自動チューニングの研究も行われている[2,3,4,5,6]が，多くは GPU 単体もしくはノード内のチューニングにとどまっている．

本研究のようなプログラムのチューニングにおいては，GPU 単体および CPU 単体

[†1] 東京工業大学
Tokyo Institute of Technology
[†2] 国立情報学研究所
National Institute of Informatics

(a)Oak Ridge National Laboratory、NVIDIA の「Fermi」アーキテクチャ搭載のスーパーコンピュータを計画
http://www.nvidia.co.jp/object/pr_oakridge_093009_jp.htm

で発生するチューニング項目だけでなく，並列化のための選択肢が多数存在する．まず，CPU と GPU の間の適切な負荷分散が必要となる．この選択においては，どちらか一種のみ使うのが最適である場合も存在することに注意する．さらに，システムが性能が不均一な GPU を含む場合には, GPU どうしの負荷分散も考慮する必要がある．

さらには，MPI 並列化に伴う多数の選択肢が存在する．たとえばマルチコア CPU を用いた MPI プログラムにおいては，1 プロセス 1 スレッドとするか(フラット MPI)，1 プロセスに複数スレッドを割り当てるか(ハイブリッド MPI)の選択肢があるが，GPU を用いる場合にも，1 プロセス 1GPU とするか，1 プロセス複数 GPU とするかが考えられる(ただし本稿に含まれる実験は全て前者とした)．また CPU を併用する場合に，システムが持つ CPU コアのうちどれだけを計算に使うべきかも自明ではない．GPU クラスタにおける並列計算では GPU とホスト間の PCI-Express を介した通信が頻発するが，その通信のためにも CPU リソースが使われることを考慮する必要がある．

以上のように GPU クラスタでは多数のチューニングパラメータが存在する. 本研究では，現状ではその一部に焦点をしぼり，予備実験結果を示す．実験は，GPU 単体におけるパラメータと性能の関係, 不均一な GPU を持つマシンでの負荷バランスと性能の関係を示す．また 3.2 節で提案する，GPU-ホスト間通信と MPI 通信の並列化技法の性能評価を，TSUBAME スパコン上で多数 GPU を用いて評価する．

2. 関連研究

2.1 ステンシル計算のチューニング

本論文では 3 次元のステンシル計算に基づく熱拡散方程式プログラムを対象とする．ここではステンシル計算のチューニングに関する研究について簡単に述べる．

3 次元熱拡散方程式に対するオートチューニングを用いた研究として Datta らによる研究がある[2]. マルチコア CPU, Cell, GPU のそれぞれを自動チューニングの対象とし，階層的ブロッキング，NUMA 環境に適したメモリ初期化手法，SIMD 化，プリフェッチなどの手法を適応し，それぞれのプロセッサにオートチューニングが有効であることを示した．本研究ではそれらを参考にしつつも，CPU と GPU を併用した場合の最適化を行う．

Venkatasubramanian らは CPU と GPU を併用して 2 次元の熱拡散方程式を計算している[3]．彼らはホストと GPU 間の同期コストを削減する手法を提案し，GPU と CPU の併用により GPU 単独の実行よりも高い性能を実現した. 彼らは 1 ノードで実験しているのに対し，本研究ではマルチノードの結果も示す．

Micikevicius は三次元のステンシル計算をマルチノード/マルチ GPU 上で実装した[4]．GPU の非同期データ転送機能と計算順序を工夫し，4GPU までのスケーラビリティを評価した．

Meng らは GPU 上のステンシル計算の精緻な性能モデルを構築し評価している[5]. 彼らの焦点は，シェアードメモリにデータを搭載したまま複数回のイテレーションを実行すること(時間的ブロッキング)である．この手法はデバイスメモリへのアクセスを大幅に削減できる一方，計算に冗長性が生まれその増加はイテレーションの回数に対して急速に増加する，というトレードオフがある．彼らの結果によると，計算対象領域が 1 次元と 2 次元の場合には良好な性能を示したが，3 次元においては冗長計算の増加のため有効ではなかった．本研究では 3 次元計算を行うため，この手法は採用しなかった．

加藤ら[6]はマルチノード/マルチ GPU 環境を対象としている．別ノードにある GPU 間の通信には 3hop の通信が必要であり，その大きなコストを，計算と通信のオーバラップにより隠ぺいすることに焦点を当てている．本研究では，通信の並列化により 3hop から 1hop に削減する手法を述べる．

2.2 オートチューニング

計算機システムの複雑さに対応するため，オートチューニングに関する研究が，広くなされている．著名な研究として ATLAS や FFTW が挙げられるが，これらは特定のライブラリやアルゴリズムに対して，システムの特性に適合させるためのチューニングを自動で行うものである．一方で片桐らはより汎用に使うことのできる自動チューニングソフトウェアの開発を支援ツールとなる ABCLibScript を提案している[7]. 最適化のタイミングとしてインストール時と実行直前, 実行中と選択できる.ループアンローリングの段数などを自動的に最適化する事ができ，パラメータの探索も全数探索などを指定することができる．

3. チューニング対象のプログラム

3.1 熱拡散方程式プログラムの構成

我々がチューニングの対象とする，GPU クラスタ上の熱拡散方程式プログラムについて述べる．

熱拡散方程式は，熱の変化を計算する偏微分方程式である．離散化した偏微分方程式の反復解法はいくつも知られているが，今回は単純なヤコビ法による離散化を用いた. 3 次元熱拡散方程式は 7 ポイントステンシル計算になり，1 回のイテレーションごとに 1 要素当たり 8 回の浮動小数点演算を行う．ステンシル計算とは計算対象の近隣の点を参照して次のタイムステップの要素を決定する計算方法である．今回の 3 次元熱拡散方程式の計算式は，

$$A'[0,0,0]=C0\times[0,0,0]+C1\times$$
$$(A[+1,0,0]+A[-1,0.0]+A[0,+1,0]+A[0.-1.0]+A[0.0.+1]+A[0,0,-1]$$

となる．

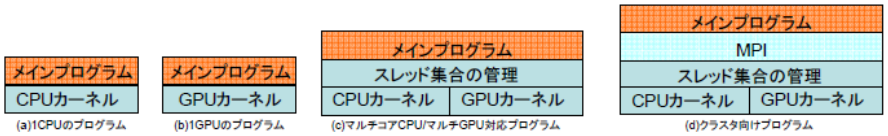

図 1　プログラムの構成

本プログラムは，マルチノード/マルチ GPU 環境において以下のように動作する．ノード間通信は MPI により実現し，3 次元の計算対象領域は各 MPI プロセスに分割される．現在の実装では 1 次元分割となっている．さらに各 MPI プロセスは複数スレッドから成り，プロセスが持つ領域は，さらに各スレッドに 1 次元分割される．そして各スレッドは GPU 上のカーネルか CPU 上のカーネルのどちらかを実行する．

プログラムは，以下のようにモジュール化されており，図 1 のように異なるマシン環境にも容易に対応可能になっている．

- カーネル：熱方程式を計算する．CPU 用と GPU 用の 2 種類用意し，共通のインターフェースをもたせる
  - CPU カーネル：担当領域をループ実行により計算する．単純には三重ループで実現可能だが，ブロック化のために六重ループとなる．
  - GPU カーネル：担当領域(サイズ lx×ly×lz とする)は，複数の CUDA スレッドにより並列に計算される．スレッドブロック中のスレッド数は bx×by(これは後述のチューニングパラメータ)とし，スレッドブロック数は (lx/bx)×(ly/by)となる．各スレッドは，Z 方向に向かって lz 個の点の計算を行う．CUDA プログラムの効率化においては，shared memory によるデータ再利用が効果的であるが，本プログラムにおいても，スレッドブロック内のスレッドたちが，担当する配列 A のコピーを shared memory に載せる．
- スレッド集合の管理：カーネルを複数のスレッドで管理する．MPI とマルチコア CPU/シングル or マルチ GPU 用では異なる実装を用意した．

並列ステンシル計算においては，ホストと GPU 間のデータ通信および，ノード間の通信が必要となる．理由は，上述のステンシル計算では，各点の計算のために隣りの点のデータが必要となるためで，各カーネル担当領域の境界部分の計算のためには，隣りを担当するカーネルとのデータ交換が必要である．本実装においては，ホストと GPU 間の通信は，GPU カーネルを担当するスレッドが cudaMemcpy を呼ぶことにより実装した．またノード間通信のための MPI 関数は，メインスレッドが呼ぶこととした．

以上の構成方法によりノート PC のようなマシンからデスクトップ PC，小規模クラスタなどに対応することができるプログラムを比較的容易に作成できた（図 1）．

3.2　CPU/GPU 併用時の通信最適化

マルチノード環境において，GPU どうしでの通信を行うには，図 2 左に示すように，3hop の通信が必要となる．つまり，(1)ノード A の GPU からホストへ，cudaMemcpy により PCI-Express 通信を行う，(2)ノード A からノード B へ MPI 通信を行う，(3)ノード B のホストから GPU へ PCI-Express 通信を行う．CPU-GPU 間のバンド幅やレイテンシの制限から，この 3hop 分のコストが性能に直接影響するのは好ましくない．

そこでここでは，CPU/GPU 併用時の通信の並列化手法について提案する．各 MPI プロセス中は，GPU カーネルスレッドに加え，必ず 2 つの CPU 計算スレッドを持つこととした．各プロセスにおいて，2 つの CPU 計算領域は GPU 計算領域をはさむようにした．これにより，GPU どうしが直接隣接する領域を持つことがなくなり，GPU 間での 3hop のデータ転送が不要になるように工夫した．これにより，PCI-Express 通信と MPI 通信の間の依存関係がなくなり，図 2 右のように，それらを並列に実行することができる．

本論文ではこれ以降，3hop かけて GPU どうしがデータを交換する場合を Comm(1)，通信を並列に実行するように工夫した場合を Comm(2)と呼ぶ．

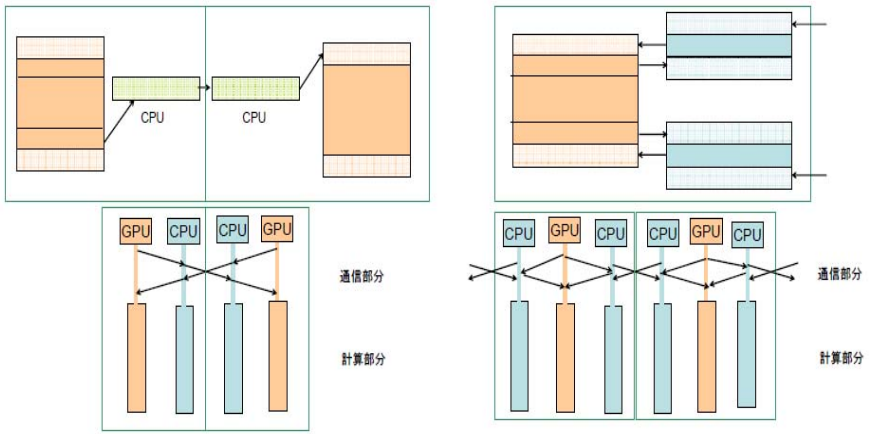

図 2　MPI 通信のパターン
（左）[Comm(1)] GPU→CPU→CPU→GPU と通信する場合　途中で CPU を経由して通信する（右）[Comm(2)] GPU が他のノードと通信しない場合　GPU の通信がノードの

内部に限定される

3.3 チューニング項目

我々が対象とするプログラムの性能には，数多くのパラメータが影響する．大きくわけて，CPU カーネル内部のパラメータ，GPU カーネル内部のパラメータ，並列化に伴うパラメータに分類できる．以下では代表的なものを述べるが，CPU 内部や GPU 内部についてのより網羅的なパラメータについては Datta らの調査[2]を参照されたい．

- CPU カーネル内部
  - ブロッキングサイズ
  - ループアンロール段数
- GPU カーネル内部
  - ブロックサイズ
  - ループアンロール段数
- 並列化に関連するパラメータ
  - MPI プロセス間の負荷分散
  - GPU/CPU 間の負荷分散
  - 領域分割次元数
  - 通信方式(3.2 節で提案した手法を採用するか否か)

現状では全てのチューニングは間に合っておらず，ループアンロールは未対応，領域分割次元数については 1 次元で固定としている．また上記の一覧の他にも，各 CUDA スレッドが計算する領域なども変更が考えられるため，探索空間は膨大となる．

上述のうち，スレッドブロックサイズについて補足する．CUDA では,スレッドをブロックという単位に分けて管理している．1 ブロック当たりのスレッド数(ブロックサイズ)とブロック数は実行時に指定可能な引数であり，GPU カーネル呼び出し時にプログラムで指定可能である．今回の GPU によるステンシル計算の実装では，1 スレッドが x,y 平面上の 1 点を担当し,z 軸上の全ての点をそのスレッドが更新する．したがって，ブロックサイズが決定すれば,ブロック数も決まる．CUDA ではブロックサイズとして 2 次元の値を，たとえば(bx,by)=(16,4)のように指定でき，本プログラムでは，スレッドブロックは XY 平面のうちそのサイズの領域を(16×4×lz のように)担当して計算する．なお，境界部分の計算を可能とするため，隣接領域を含め各ブロックは合計で(bx+2)×(by+2)×(lz+2)個の要素をデバイスメモリから shared me mory へコピーする．

このブロックサイズの設定については以下のようなトレードオフがある：もし小さすぎると，配列 A のデータを shared me mory にコピーする際に，一緒にコピーが必要な隣接領域のサイズが相対的に大きくなってしまう．一方，この値が大きすぎると，各スレッドブロックが必要な shared me mory サイズが必要となり，GPU 上で同時に動作可能なスレッドブロックの数が少なくなることになる．このとき，ハードウェアスレッド切り替えによるメモリアクセス隠ぺいがうまく働くなり，性能低下が予想される．以上の議論から，このパラメータ設定には注意が必要であり，5.1 節で議論する．

4. 自動チューニングツール

今回のプログラムの自動チューニングに向けて，ABCLibScript などを参考にして自動チューニングツールを開発中である．このツールは C 言語を対象とし，(1)#define で定義する変数の初期値などをさまざまに変化させる，(2) ループのアンローリング段数の違うコードを生成する，(3)ソースコードの一部分のアルゴリズムを複数のものからひとつのアルゴリズムを選択する，という処理が可能である．それらの選択肢についてそれぞれ，コード生成，コンパイル，結果のデータベースへの保存を行う．現状においては，パラメータ空間の探索は全数探索にしか対応していないなどの理由により，本研究では現在このツールを，小規模な予備実験にのみ利用している．

5. 評価

5.1 GPU 単体のチューニング

ここでは GPU カーネル単体の性能チューニングについて述べる．上述のスレッドブロックのサイズに注目し，それを変えたときどの程度性能に影響があるのか調べた結果を述べる．実験に利用した PC の性能を表 1 に示す．搭載した 2 台の GPU のうち，GTX 275 を用いた(表 2)

表 1　実験に用いた PC の性能

CPU Phen	om-II X4 810
Memory 8GB	
OS Open	SUSE 11.0
GPU	NVIDIA Geforce GTX275, Tesla C1070
PCL-Express	2.0 x16 レーン
CUDA version	2.1

表 2　GTX 275 と C1060 の性能諸表

	GTX 275	C1060
GFLOPS(単精度)	1010.88	933
バンド幅(GB/S)	127.0	102.8
SP のクロック数(MHz)	1404	1300
SP 数	240	240

実行時間の比較を図 3 に示す．ここでは，計算領域サイズを 256×256×256 とし，100 イテレーション行った．3.3 節で議論したように，ブロックサイズは小さくても大きくても性能は低下する現象が見られる．今回比較したうちでは，32×4 の場合が最も高速であった．ついで，16×8, 16×16, 32×8 が良好な性能となっている．4×4 とした場合には性能は大きく低下し，最良のケースの約 5 倍の実行時間となっている．このように，ブロックサイズは性能に大きく影響し，かつ最良のケースを実行前に見つけるのは自明ではない．そのため，自動チューニングを用いるのは適していると言える．

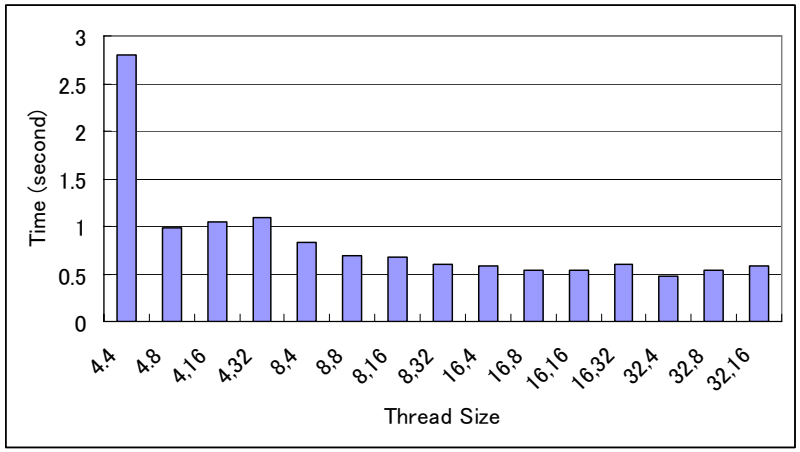

図 3 1 ブロック当たりのスレッド数の変化による性能の変化

5.2 不均一 GPU でのチューニング

5.1 節と同じ PC において，GTX275 と Tesla C1060 の性能の違う 2 種類の GPU を用いた場合の性能を調査した. なお 2 枚の異なる GPU のうち一つをグラフィック用に用いもう一つはグラフィック処理を助けるための物理演算用に用いることができるので, パーツの段階的なアップグレードなどに伴い, 不均一な GPU 環境は昔ほど珍しくはないと考える. 両者の性能は表 の通りであり，GT X 275 の方が性能が高くなっている．

利用したプログラムは 3 節で述べた MPI で並列化されたものであり，GPU と CPU を併用する．2 つの MPI プロセスを起動し，それぞれ 1GPU に対応させる．この前提でのチューニングパラメータは，MPI プロセス間の負荷分散および，GPU と CPU の負荷分散である．予備実験により CPU カーネルの速度は GPU カーネルより 2 桁近く遅いことが分かっており，CPU カーネルの計算領域の z 方向サイズとしては，2, 1, 0(CPU カーネルを用いない)について調査した．また 3.2 節で説明した 2 種類の通信方式を比較する. なおこの実験は 1 台の PC で行ったものであるが、MPI を用いている．

結果を図 4 に示す．ここでは 512×512×512 のサイズの計算を 100 イテレーション実行している．横軸が 0 の場合は， GT X 275 と C1060 に均等にタスクが割り振られている．右に行くほど C1060 により多く，左に行くほど GTX 275 により多く負荷を割り当てた場合を表す．CPU(n)の n の値は CPU カーネルに割り当てられた計算量を示す．Comm(1), Co mm(2)は 3.2節で述べた 2 種類の通信方式を表す．たとえば,横軸+20 のところの Comm(1) CPU(2)では,GPU が他の GPU のデータとデータ交換をする通信パターンであり,CPU:GTX 275:CPU:CPU:C1060:CPU = 2: (252-20) : 2 : 2 : (252+20) : 2 の比でタスクが分配されている．CPU(0)は CPU にタスクを割り当てない場合である．

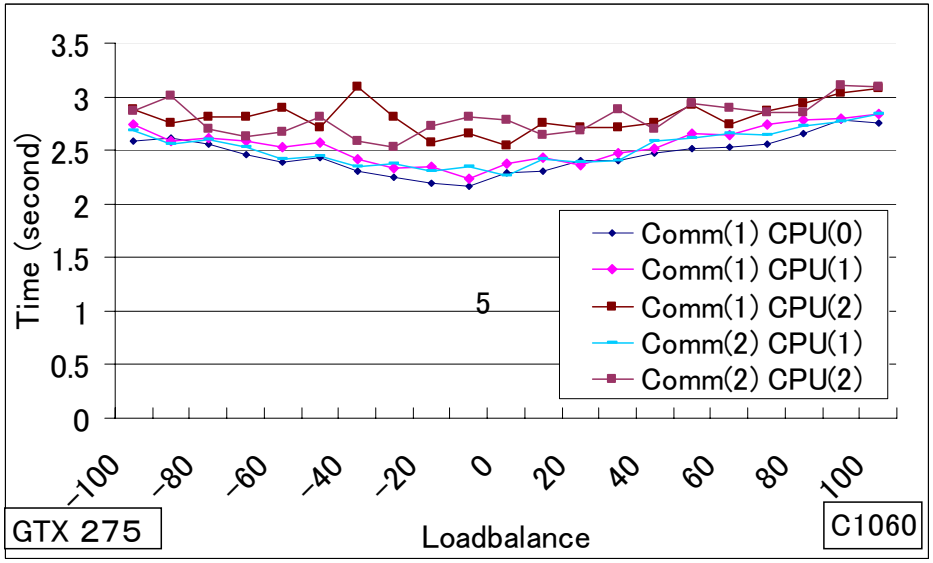

図 4 異種 GPU を用いた実験 正の方向に行くほど C1060 に割り振られるタスク量が増え,負の方向に行くほど GTX 275 に割り振られるタスク量が増える．Comm(1)と Comm(2)については3.2 節参照．CPU の後の数値は CPU に割り当てられるタスク量.

グラフの各折れ線において, 負荷バランスは-10 から-30 のときに最良となっており，これは GTX 275 の方が高速であることと一致する. 最も性能の良かったのは Comm(1) GPU(0)の負荷バランス-10 であった．CPU(2)の場合は，全てのケースで性能が悪くなっており，これは CPU の仕事が多すぎてボトルネックになっていると考えられる．な

お CPU(2)の場合を除き，グラフは下に凸型のグラフとなっており，この点に関しては網羅的な探索の代わりに，二分探索法などでも最良の点を見つけることができると予想される．なお本節の結果では CPU を併用しないほうが性能が良かったが，次節の多数 GPU 利用時では逆の結果となっており，その原因については究明中である．

5.3 GPU クラスタでの実験

表 3 TS UBAME ノードの情報

CPU	Opetron 880 (dual core) x 8
Memory DD	R 32GB
OS	Suse Linux Enterprise 10
GPU	NVIDIA Tesla S1060 (2 デバイス)
PCL-Express	1.1 x 8 レーン
ネットワーク	InifiniBand 4x SDR
MPI V	oltare MPI
CUDA version	2.2

3.2節で述べた通信方式について GPU クラスタである東工大 TSUBAME を用いて実験を行った． TS UBAME の各ノードの情報は表 の通りである．今回の実験では 1 ノードごとに 1MPI プロセスを起動し，各 MPI プロセスごとに 1GPU を用いた．いくつかの問題サイズについて，以下の 2 つのケースを比較した．

- G: GPU カーネルのみを用い，通信に 3hop かける場合 (C omm(1))
- G+C: GPU と CPU を用い，通信の並列化を行う場合 (C omm(2))

なお後者においては，各プロセスにおける CPU:GPU:CPU の計算量の比を 1:126:1 とした．なおプロセスあたりの負荷が小さい(z 方向サイズが 128 未満)の場合には，CPU カーネルの負荷(z 方向サイズ)は 1 とした．

最大 32 ノード (32 GPU)を用いて実験を行い，その結果を図 5 に示す．グラフの縦軸は速度 (GFlops) を示す． 問題サイズとしては， 512x512x512， 1024x1024x1024, 2048x2048x2048 を用いた．なお問題サイズが大きい場合には少数の GPU ではメモリ不足で実行できない．2048 の 3 乗のケースは 32GPU の場合でのみ実行に成功した．

問題サイズ 2048x2048x2048 の場合において提案した通信並列化は良好な性能を示し，GPU のみの Comm(1)利用時に比べ 20%の速度向上を示した．1024x1024x1024 の場合には，8GPU 利用時までは提案手法のほうが高速であるものの，16GPU 以上では GPU のみのケースより遅くなっている．この原因は，プロセスあたりの計算量が小さいにも関わらず，CPU には z 方向 1 の仕事を割り当てる必要があり，その CPU の処理がボトルネックになっているためである．このボトルネックを回避するためには現在の 1 次元分割から 2 次元以上の分割に変更することが有効と考えられる．

なお CPU カーネルのボトルネックの影響を軽減するために, 512x512x4096 という z 方向に長い領域についても測定した．それによると，全ての GPU 数において，提案する CPU 併用・通信並列化を行ったほうが高性能という結果となっている．

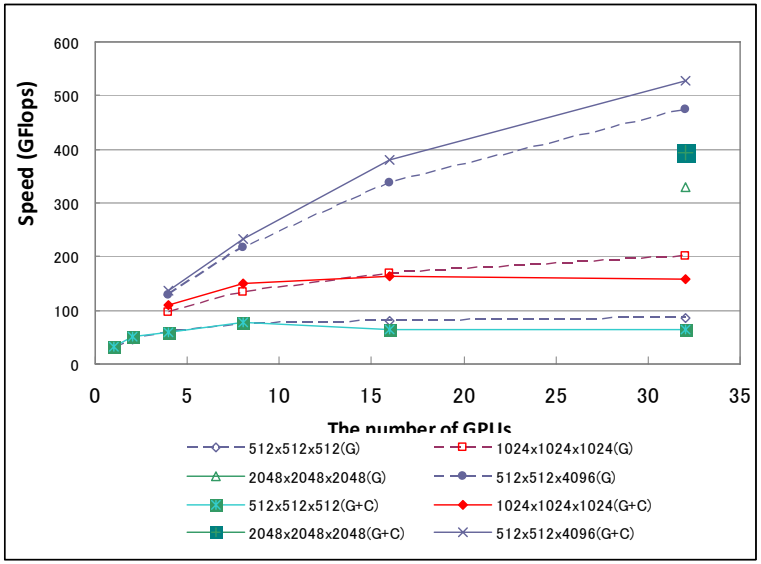

図 5 GPU クラスタにおける実験 横軸が GPU 数,縦軸が GFLOPS. (G)は GPU のみを利用した場合(G+C)は GPU と CPU を用い並列通信(Comm(2))を用いた場合．通信方式については3.2 節参照.

6. おわりに

本論文では，GPU クラスタ上における熱拡散方程式プログラムを題材に，チューニングの自動化へ向けた実験結果について述べた．また PCI-Express 通信と MPI 通信を並列化する技法を提案し，32GPU を用いた実験によりその効果を示した．この手法に

ついては，問題サイズに対してプロセス数が大きいときに，CPU でのステンシル計算がボトルネックとなることが確認された. この点については, 現状の一次元分割から，二次元以上の分割することにより境界部分の計算オーバヘッドを小さくするなどにより，問題を軽減できると考えられる．また，通信どうしの並列化に加え，通信と計算のオーバラップ技法[6]も用いて, GPU クラスタにおける大きな課題である通信オーバヘッド削減を行っていきたい.

今後はチューニングの自動化へ向けて，より総合的にチューニングを行うことも課題である．本稿の実験では固定としたパラメータでも，網羅的なパラメータ探索により, 違う選択が望ましかったケースがある可能性がある. 今回はまだ考慮していない，各カーネルのアンローリング，プロセスあたりの GPU 数および CPU スレッド数，スレッドと CPU コアのバインド, MPI 実装の選択も対象としたい. さらにより大規模な環境を考慮すると，ネットワークトポロジーやネットワークの性能特性を考慮したデータ割り当てのチューニングが必要となる．このような総合的・網羅的な自動チューニングのためのシステム構築が今後の課題である.

謝辞

本研究の一部は，JST-CREST「ULP-HPC:次世代テクノロジのモデル化・最適化による超低消費電力ハイパフォーマンスコンピューティング」および，科学研究費補助金特定領域研究 (1804 9028) の補助による

参考文献

1) 遠藤敏夫，額田彰，松岡聡，丸山直也． 異種アクセラレータを持つヘテロ型スーパーコンピュータ上の Linp ack の性能向上手法 . 並列/分散/協調処理に関するサマーワークショップ (SWoPP2009)，情報処理学会研究報告, 2009-HPC-121, No.24, 8 pages, 2009.
2) Kaushik Datta, Mark Murphy, Vasily Volkov, Samuel Williams,Jonathan Carter, Leonid Oliker, David Patterson, John Shalf, and Katherine Yelick. Stencil computation optimization and autotuning on state-of-the-art multicore architectures. SC Conference, Vol. 0, pp. 1–12, 2008.
3) Sundaresan Venkatasubramanian, Richard W. Vuduc,. Tuned and wildly asynchronous stencil kernels for hybrid cpu/gpu systems. In ICS '09: Proceedings of the 23rd international conference on Supercomputing, pp. 244–255, New York, NY, USA, 2009. ACM.
4) Micikevicius, P. . 3D finite difference computation on GPUs using CUDA. In Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units (Washington, D.C., March 08 - 08, 2009). GPGPU-2, vol. 383.
5) Meng, J. and Skadron, K.. Performance modeling and automatic ghost zone optimization for iterative stencil loops on GPUs. In Proceedings of the 23rd international Conference on Supercomputing (Yorktown Heights, NY, USA, June 08 - 12, 2009). ICS '09.
6) 加藤季広，青木尊之，額田 彰，遠藤敏夫，松岡 聡，長谷川篤史． 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 〜 . 情報処理学会研究報告, 2009-HPC-120 No.3, 6 pages, 2009.
7) 片桐孝洋, 吉瀬謙二, 本多弘樹, 弓場敏嗣：自動チューニング処理記述用ディレクティブ ABCLibScript の設計と実装, 先進的計算基盤システムシンポジウム SAC SIS2004 論文集, pp. 43--52 (2004)