Vol.2020-HPC-177 No.10
2020/12/21

GPU におけるマルチグリッド前処理付き共役勾配法の最適化

山岸孝輝[1]　松村義正[2]　羽角博康[2]

概要：マルチグリッド前処理付き CG 法を数値海洋モデルのポワソン方程式ソルバとして GPU 実装した．MPI 通信用袖領域を分離したデータ構造を採用し，通信と演算をオーバラップさせる事並びに荒い格子での処理を複数 GPU から 1GPU に集約することで通信を隠蔽または削減した．これによりソルバのコストの 26%を削減でき，数値海洋モデル全体では CPU 比較で 4 倍の性能向上と 64GPU まで良いスケーリング性能を得た．

1. はじめに

我々のグループでは，海洋の働きが海洋資源や人為起源物質の輸送・拡散に与える影響や気象・気候との関係について GPU を搭載したスーパーコンピュータを用いた大規模シミュレーションで調べるために，効率的な計算手法の開発に取り組んでいる．GPU はデータ並列性が高く，メモリアクセスが規則的で，同一の命令が多数のデータに発行される場合に高い性能を示すが，数値海洋シミュレーションはこの特徴に合致している．

数値海洋シミュレーションは，海流を表現するのは一般の CFD アプリケーションと同じく数値流体力学であり，特に我々が開発した海洋モデルの基本方程式は適切な近似を元にして 3 次元ポワソン方程式に帰着し，それをいかに大規模問題にて高速に解くかは主要な問題の一つである．3 次元ポワソン方程式のソルバとしてマルチグリッド前処理付き CG 法（以下 MGCG 法）を用いているが，複数 GPU への実装事例は少なく，最適化とその詳細な評価は未だ行われていない．

本稿では，我々のグループが MGCG 法の GPU 実装にこれまでに主に演算部分に最適化を施してきたものに対して，今回新たに施した通信に関係する処理の最適化についてまとめる．

2. 非静力学数値海洋モデル kinaco と MGCG 法

本研究にて用いた数値海洋モデル kinaco と，kinaco にて海洋の力学場を表現する 3 次元ポアソン方程式のソルバの概要をまとめる．

(1) 非静力学数値海洋モデル kinaco

松村らによって開発された非静力学数値海洋モデル kinaco[1]は，海洋の力学場を 3 次元のナビィエストークス方程式を元に算出する．その際鉛直方向の非静力学過程を考慮することで，圧力を近似の元で見積もることをせずに陽に算出している．これにより，細かいスケールの海流をより詳細に解析することが可能となる．運動方程式である 3 次元のナヴィエストークス方程式と連続の式とから，圧力項の 3 次元ポワソン方程式に帰着する．CPU での実行を基本環境として開発が進められており，言語は Fortran90 である．海洋に関する問題を扱うには複数プロセッサによる大規模計算が必須で，計算領域の分割に伴うプロセッサ間の通信は MPI ライブラリを用いる．ポワソン方程式の求解のコストが全体のおよそ半分を占め，それの高速化と複数プロセッサ利用時のスケーリング性能の改善が求められている．

(2) MGCG 法とポワソン方程式

3 次元ポワソン法指定式のソルバにマルチグリッド前処理付き CG 法（MGCG 法）を採用している．MGCG 法は大規模問題においても収束までの反復回数が増加しないことが特徴で，kinaco においてもスーパーコンピュータ「京」にて 1024 ノード（8192MPI 並列）まで反復回数が増加せず良い弱スケーリング性能を示すことが確認されている[2].

3. GPU 実装と基本評価

(1) 評価環境と実験設定

評価環境は東京大学情報基盤センターのスーパーコンピュータ Reedbush-H であり，コンパイラ等の設定は表 1 にまとめた．1GPU あたりの格子点数は約 200 万（256*256*32）に設定し，海底地形は複雑な地形はとらずに平坦なものとし，対称的な外部強制を与えた理想的な条件の下で複数プロセッサ利用時の性能を弱スケーリングの条件で評価した（表 1）．

表 1　評価環境と実験設定

	項目	内容
ハード	Reedbush-H	SGI Rackable C1102-GP8
	CPU	Xeon E5-2695v4 [605GF, 2.1GHz, 18core], 2socket
	GPU	Tesla P100 with NVLink [5.3TF, 16GB], 2board
	ノード間接続	InfiniBand FDR 4x2 [56 Gbps x2]
ソフト	F90	PGI 18.7
	MPI	OpenMPI 2.1.2 GPUDirect 対応版
	CUDA	CUDA 9.1.85
実験	実験設定	Deep convection/Visbeck1996
	格子点数	256x256x32 cells/processor[弱スケー

1 高度情報科学技術研究機構
Research Organization for Information Science and Technology
2 東京大学大気海洋研究所
Atmosphere and Ocean Research Institute, The University of Tokyo

		リング]
	CPU・GPU 数	2, 4, 8, 16, 32, 64
	時間ステップ	120sec*300step=10hours

(2) GPU 実装

PGI CUDA Fortran を用いて GPU の実行モデルに合わせた実装を行っている．コアレスアクセスを促進するための配列インデックスの交換，レジスタを活用したデータの再利用の促進，スレッドブロックに調整による L1 cache の活用等の最適化を取り込んでいる[3][4]．GPU 間の通信では，CUDA 対応版 OpenMPI と Reedbush-H に備わっている GPUDirect RDMA の機能を利用することで，通信オーバヘッドを抑えている[4].

(3) 基本評価

16 並列（16CPU/16GPU）での基本的な性能評価を示す(図 1)．我々がこれまでに施した最適化[2, 3]により，通信を除いた演算部分では GPU は CPU 比で約 4 倍高速であるが，通信コストは増加して約 0.4 倍の速度である．特に MGCG 法での通信コストの増加が大きく，MGCG 法の通信コストは CPU 実装では全体の 3%程度が GPU 実装では 23%程度を占め，その結果 MGCG 法は全体に対して CPU 実装では 57%程度のコストであったものが GPU 実装では 73%程度を占め, 数値モデル全体の高速化の制約となっている. GPU による演算部分の高速化を活かすためには，MGCG 法の通信コスト増加の問題は避けて通れない.

通信コスト増加の要因として，MGCG 法に含まれる粗い格子での小さいサイズの通信にて CPU に比べてオーバヘッドが大きいことが考えられる.

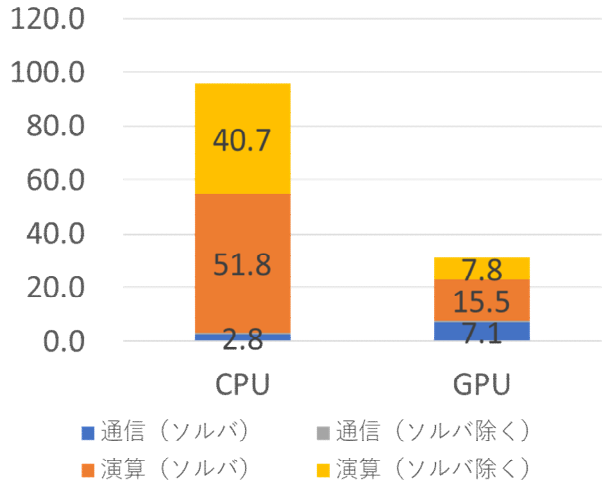


図 1 基本性能比較（16 並列）．縦軸：経過時間[秒].

(4) 通信コストの削減策

本研究では MGCG 法の通信コスト増加の問題に対して，MPI 通信用袖領域を分離したデータ構造の採用，演算とのオーバラップによる通信コストの隠蔽と，マルチグリッド前処理における粗い格子の集約の 3 案によってコストを削減することを試みた．以下ではそれら 3 つについて概要をまとめる.

4. MPI 通信用袖領域を分離したデータ構造の採用と評価

Kinacoの計算領域は水平一般直交座標・鉛直z座標系を採用した3次元構造格子で，水平面を領域分割して領域間はMPIライブラリを用いて通信を行っている．運動方程式を離散化した際に必要とする隣接格子までは最大で2格子分となっている．予報変数Aが各次元にてn1, n2, n3個の要素を有する場合, 袖領域は各次元で2個必要となり, A(-1:n1+2, -1:n2+2, -1:n3+2)と設定してメモリを確保している．MPIライブラリに袖領域を渡す際，通信のオーバヘッドを抑えるために, 予報変数とは別に確保した2次元配列に袖領域を詰めた上でMPIライブラリにて縁辺部通信を行っている．袖領域の処理を含む演算と通信の処理の流れを示す.

1．隣接通信
2．袖領域の詰め替え（2次元配列→3次元配列）
3．演算処理
4．袖領域の詰め替え（3次元配列→2次元配列）
5．隣接通信

この流れにおいて，2次元配列を3次元配列の袖領域へと詰め替え（2）と，3次元配列の袖領域から2次元配列への詰め替え（4）については，同じデータを別々のアドレスに保持しており非効率である. そこで袖領域を3次元配列には含まずにA(1:n1, 1:n2, 1:n3)としてメモリを確保し，通信に用いる袖領域の2次元配列を直接（3）の演算処理にて扱うこととした．（2）から（4）までをまとめて処理するプログラム例を示す（図 2）.

```
! ［GPU のスレッドを設定］
  i=threadidx%x+blockdim%x*(blockidx%x-1)
  j=threadidx%y+blockdim%y*(blockidx%y-1)
  k=threadidx%z+blockdim%z*(blockidx%z-1)

! ［東側の袖をスカラ変数 y_ip1 に格納］
  if(i.eq.n1)then
    y_ip1 = r_e(j,k)
  else
    y_ip1 = y(i+1,j,k)
  endif
  …
! ［その他 3 方向の袖を同様にスカラ変数に格納］
    y_im1 = r_w(j,k)
  …
    y_jp1 = r_n(i,k)
  …
    y_jm1 = r_s(i,k)
  …

! ［メインの演算処理］
  out_r = out(i,j,k)                &
        + b(i,j,k,-3) * y(i,j,k-1) &
        + b(i,j,k,-2) * y_jm1      &
        + b(i,j,k,-1) * y_im1      &
        + b(i,j,k, 0) * y(i,j,  k) &
        + b(i,j,k, 1) * y_ip1      &
        + b(i,j,k, 2) * y_jp1      &
        + b(i,j,k, 3) * y(i,j,k+1)

  out(i,j,k) = out_r

! ［東側の袖を送信用 2 次元配列に格納］
  IF(i.eq.n1)then
     s_e(j,k) = out_r
  ENDIF
  …
! ［その他 3 方向の袖を同様に送信用配列に格納］
     s_w(j,k) = out_r
  …
     s_n(j,k) = out_r
  …
     s_s(j,k) = out_r
  …
```

図 2　袖領域の2次元配列を演算処理にて直接アクセスするCUDAカーネルの例.

後述するマルチグリッド法を用いて前処理を行うMGCG法の仕様上，水平方向の次元の大きさn1, n2らは2の階乗と設定する場合がほとんどであるが，袖領域を外した事で，グローバルメモリへのアクセスでアライメントが揃うこととなり，メインの演算処理にてグローバルメモリへのアクセスで無駄なトランザクション発生を抑えることができる．MPI通信にて転送されてきた袖領域と，（3）の演算処理での結果をスカラ変数に格納することで，グローバルメモリへの冗長なメモリアクセスを削減することを図っている．また，（2）から（4）を同一カーネルに融合したことで，GPUの同じSMX内の同じスレッドブロックにて処理されることから，キャッシュの活用も期待できる．

本評価では，NVIDIA CUDA ToolKitに含まれるプロファイラnvprofを用い，グローバルメモリへのロードストア効率とトランザクション数を，コスト上位に位置する行列ベクトル積カーネルにて調べた[表 2].

表 2　グローバルメモリのロードストア効率とトランザクション数

metrics	ASIS	TUNED
gld_efficiency	66.74	85.91
gst_efficiency	88.89	100

metrics	ASIS	TUNED
global memory transactions	12,472,832	11,835,392

グローバルメモリへのロード, ストア共に効率が上がり，アライメントの調整が改善されたこと，それに伴いメモリのトランザクションも減少したことが確認できる．ソルバ単体のコストでは，2から64並列での平均で17.9秒とASIS版の20.0秒から約11%のコストを削減できている．

5. 隣接通信の演算との非同期処理による隠蔽

MGCG法における袖領域の通信を演算と非同期に実行することで通信コストの隠蔽を行った．非同期処理の実装に伴うオーバヘッドが発生するため，演算量が通信を隠蔽するに十分多い事が必要である．それ故今回の実装では，CG法の処理の他にマルチグリッド前処理で最も格子が細かいレベルにおいて，非同期処理を行った．CUDAによる実装としては，依存関係の無い演算・通信処理を，それぞれを異なるCUDAストリームに割り当てて非同期に処理を行い，適切なタイミングで同期をとって終了する（図 3a）．演算と通信に依存関係がある場合は，通信と関係する縁辺部の演算のみを通信と逐次に処理し，内部領域は通信と非同期に処理させている（図 3b）．数値流体シミュレーションのGPU実装において，通信と演算の非同期実行については[5][6]の先行事例があるが，本稿ではMGCGソルバに適用した上に最適化を施して性能評価したことが新しい点である．

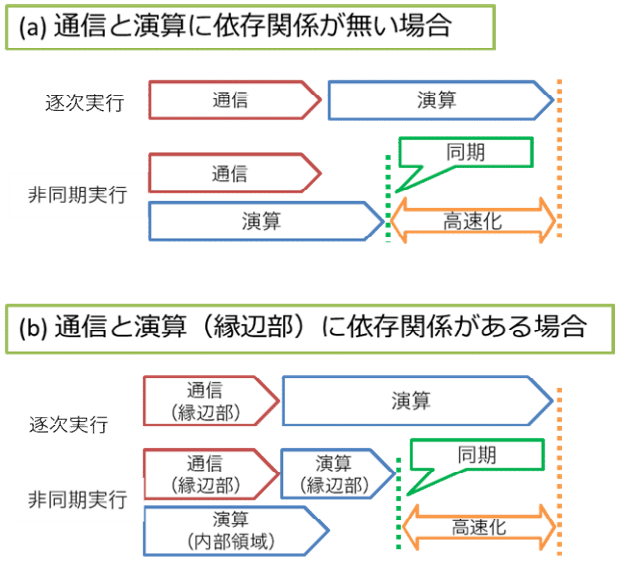


図 3　通信と演算の非同期実行による通信コストの隠蔽

図 4はMGCGソルバの処理の流れであるが，内積の計算，ベクトルのノルムの計算では袖領域の情報を必要としないため，縁辺部通信とは非同期に処理が行える．マルチグリッド前処理にてコストの大きい処理である，行列とベクトルの演算や格子数の変換処理は縁辺部通信に依存する．それ故内部領域のみを分離させた上で，縁辺部通信と非同期に処理させた．

```
M: 係数行列Lの近似逆行列
r_0 = q - L*p_0; u_0 = M*r_0; rho_0 = (M*r_0, r_0)
DO n = 0,1,2,,,
  alpha = rho_n/(u_n, L*u_n)
   // 内積計算
  p_n+1 = p_n + alpha*p_n
  r_n+1 = r_n - alpha*L*u_n
  if |r_n+1| accurate enough then quit
   // ベクトルノルム計算
  e_n+1 = M*r_n+1  // マルチグリッド前処理
   //solve for e_n+1 in L*e_n+1 = r_n+1
  rho_n+1 = (M*r_n+1, r_n+1)
   // 内積計算
  beta = rho_n+1/rho_n
  u_n+1 = e_n+1 + beta*u_n
END DO
```

図 4　MGCG 法の擬似コード

図 5はNVIDIA Visual Profilerにて採取したタイムラインの一例であるが，両方の手法で通信コストの隠蔽並びに性能向上が確認できた．

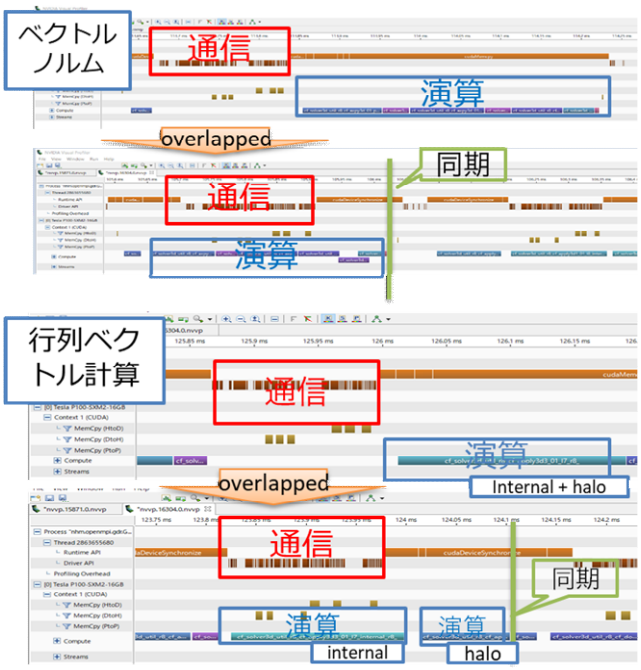


図 5　演算と通信の非同期処理実施時のタイムライン

ソルバ単体のコストでは，先述のデータ構造の変更の効果と併せると，2から64並列での平均で17.1秒とASIS版の20.0秒から約15%のコストを削減できている．

6. マルチグリッド前処理における荒い格子の集約

数値海洋シミュレーションでは高い解像度での実行が必要で，数千を超える複数ノードでの効率的な並列化が求められており，格子点数増加に対して反復回数が増加しない解法として，kinacoではMGCG法を採用している．前処理のマルチグリッド法ではV-cycleを採用し，最も細かい格子レベルをlevel=1として，荒くなるにつれてレベルが増加する．最も荒い格子で格子数が1となるように設定し，全プロセスの情報をMPIのrankが0となるGPUに集約して，そこから改めてマルチグリッド法で処理を続けていた[図 6]．先行研究[7]にて集約後の処理をCoarse Grid Solverと呼ぶが，GPU並列数が多くなると，全体通信による1GPUへの集約後のマルチグリッド前処理での格子数とレベルが多くなる．その結果Coarse Grid Solverの負荷が増大し，全体の格子点数増加に対するスケーリング性能が悪化する．

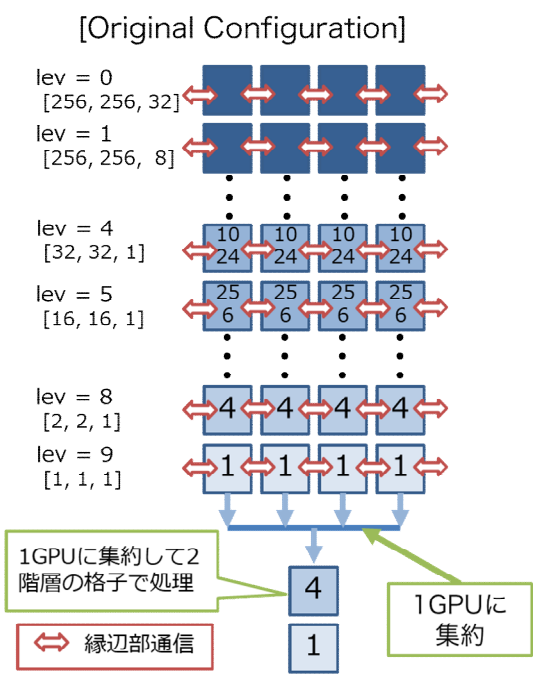


図 6　マルチグリッド前処理（4GPU 並列時）の格子情報と格子の集約

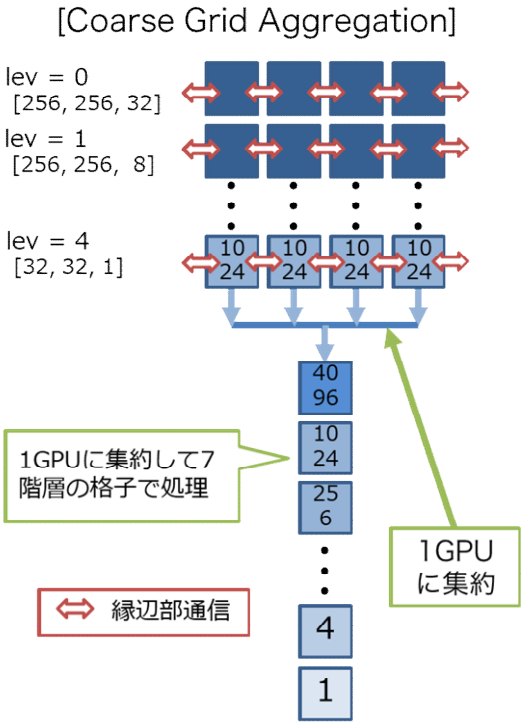


図 7　マルチグリッド前処理（4GPU 並列時）の格子情報と CGA を採用した粗い格子の集約

GPUでの荒い格子での処理では，大半が通信コストであるが，演算処理それぞれの単独でのコストが少ないため，非同期処理を行う際のオーバヘッドの影響が大きく，前節で用いた演算と非同期に通信させて通信コストを隠蔽する手法は使えない．荒い格子での通信コストを減らす方法として，先行研究[8]では，Coarse Grid Aggregation（CGA）法を提案した．これは，V-cycleのレベルが小さく格子が細かい段階で集約してCoarse Grid Solverにて処理を行う．これにより，通信コストの削減が期待できるが，集約に伴う全体通信コストとCoarse Grid Solverの負荷の増大が生じる．先行研究[8]では，Coarse Grid Solverにて各MPIプロセス上の全てのコアをOpenMP並列によって負荷分散させて高速にCoarse Grid Solverを処理しているが，GPUの場合はCPUよりもより多く数千もの演算コアを含むため，CPUよりも多くの格子を集約できる可能性がある．その一方，GPUはCPUよりも大きい全体通信のオーバヘッドを抱える問題も存在する．

本稿ではCGAをGPU実装されたMGCG法に適用し，縁辺部通信コストの減少，全体通信コストの増加とCoarse Grid Solverの負荷の増加ついて評価した結果をまとめる．

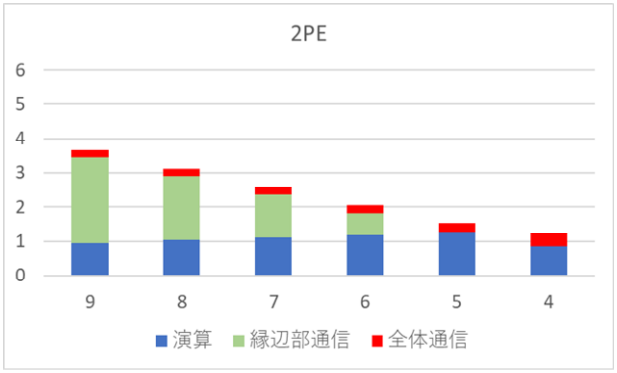
今回の評価では，1CPU/GPUあたり[256, 256, 32]の格子点を割り当て，level0から最大でlevel12まで処理を行う（表 3）．Level0からlevel9までは全ての並列数で共通の格子点数を各GPUそれぞれにて処理を行う．level10以上がCoarse Grid Solverに相当し，level9における全ての格子を全体通信で集約後，CPUまたはGPU単独で処理を行う．並列数が多くなるに従い，Coarse Grid Solverで扱う格子点数とlevelの数が多くなり，負荷が増大する．

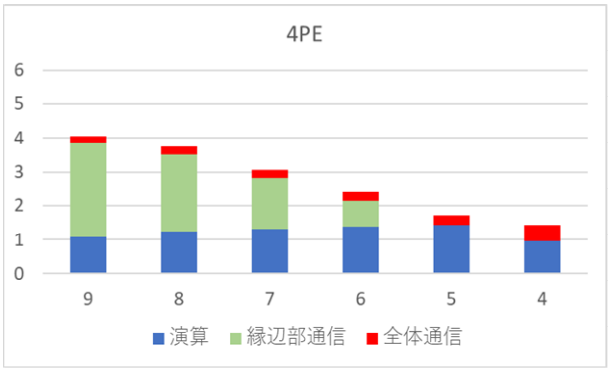
表 3　マルチグリッド前処理の格子情報

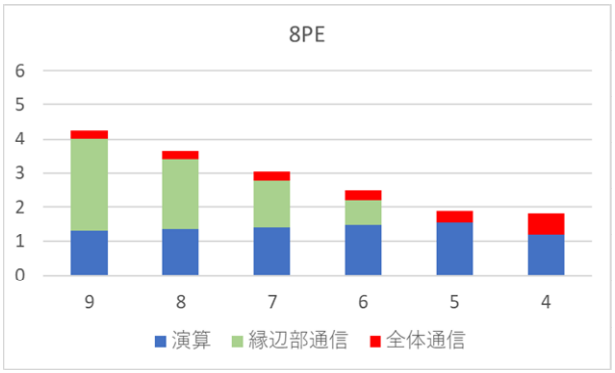
		並列数					
		2	4	8	16	32	64
Multi Grid level	0	[256, 256, 32](2,097,152)					
	1	[256, 256, 8](524,288)					
	2	[128, 128, 4](65,536)					
	3	[64, 64, 2](8,192)					
	4	[32, 32, 1](1,024)					
	5	[16, 16, 1](256)					
	6	[8, 8, 1](64)					
	7	[4, 4, 1](16)					
	8	[2, 2, 1](4)					
	9	[1, 1, 1](1)					
	10	[2, 1, 1]	[2, 2, 1]	[4, 2, 1]	[4, 4, 1]	[8, 4, 1]	[8, 8, 1]
	11			[2, 1, 1]	[2, 2, 1]	[4, 2, 1]	[4, 4, 1]
	12					[2, 1, 1]	[2, 2, 1]

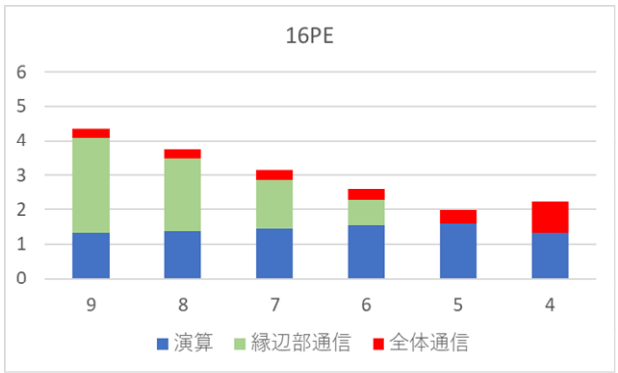
今回の実装では，格子点数が2次元となるlevel4から標準の実装で集約を行うlevel9までの間の各レベルにて集約を

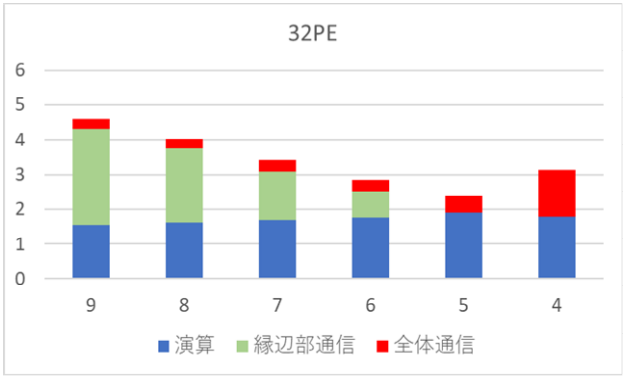
適用し，性能を評価した．図 8は，ソルバのマルチグリッド前処理のうち，2次元となる部分（level4以上）のみを抽出し，通信・演算のコストを2から64並列まで比較したものである．











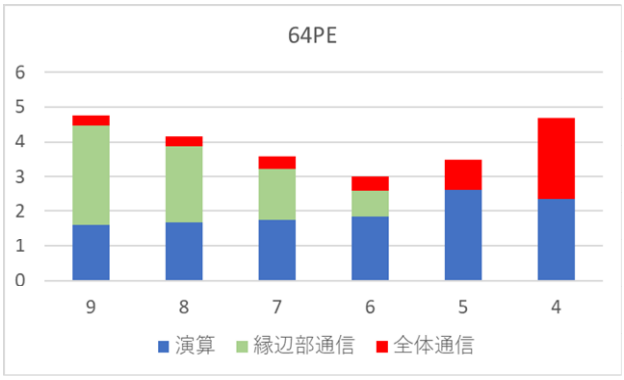


図 8　マルチグリッド前処理（level4 以上）の通信・演算コスト　縦軸：経過時間[秒]　横軸：格子を集約する level

格子の集約するlevelが9となる場合が，CGAを行わない標準の設定に相当する．ソルバのマルチグリッド前処理のうち，2次元となる部分（level4以上）では通信コストが大半を占めることを確認できる．

すべての並列数で，格子を集約するlevelを小さくする（よりたくさんの格子点を1GPUにまとめる）ことで，縁辺部通信が減少することを確認できる．その一方，集団通信のコストが増大しており，それは格子を集約するレベルと並列数に依存している．

演算部分については，level9（[1, 1, 1]）での集約からlevel4（[32, 32, 1]）まで一定または増加の傾向を示すが，通信に比べると少ない変化である．先述したとおり，GPUは数千を超えるコアで並列計算を行うプロセッサであるために，最大で64並列でleve4（[32, 32, 1]）程度での格子の集約にて演算処理のコストに大きな違いが発生しないと考えられる．

今回の評価環境と実験設定では，2, 4, 8並列ではlevel4，16, 32並列ではlevel5，そして64並列ではlevel6での集約が最善となった．どの並列数とlevelの組み合わせが最善となるかに一般性のある解はないが，ここで挙げた3要素を検討する必要性と検討する際に注目すべき内容を示せたといえる．

64を超える並列数の場合は，すべての値を1つのGPUに集約すると，全体通信のコストと演算のコストが増大する可能性がある．対策としては，[9]にて用いられた，hCGA法（Hierarchical Coarse Grid Aggregation）が有効な手段と考え

ている．これは，段階的に格子を集約させて，複数GPUそれぞれにてCoarse Grid Solverを処理する方法で，集団通信のコストを抑えつつ演算部分を負荷分散することで高並列におけるスケーリング性能を改善することが可能と考えている．

MPI通信用袖領域を分離したデータ構造の採用，演算とのオーバラップによる通信コストの隠蔽と，マルチグリッド前処理における粗い格子の集約の効果を，2から64並列においてMGCGソルバにて比較した[図 9]．全ての並列数において本稿で検討した3つの施策がMGCGソルバのコスト削減に概ね効果があることがわかった．

CGAの効果はソルバ単体のコストでは，先述のデータ構造の変更と通信隠蔽の効果と併せると，2から64並列での平均で14.8秒とASIS版の20.0秒から約26%のコストを削減できている．

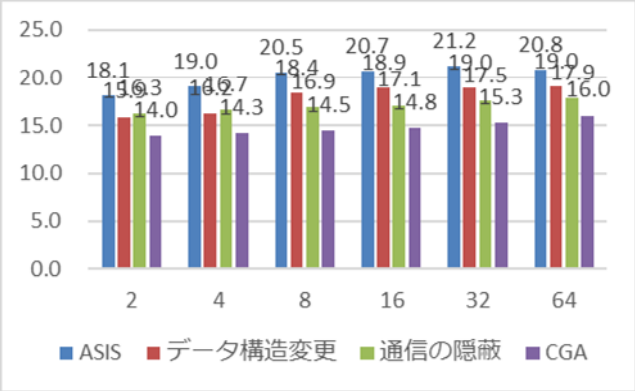


図 9　3つの施策それぞれのMGCGソルバのコストへの影響　縦軸：経過時間[秒]　横軸：並列数

7. 最適化を施した MGCG ソルバの数値海洋モデルへの適用と性能評価

本稿に示したMGCG法の最適化手法を数値海洋モデルkinacoに適用した．KinacoはMGCGソルバ以外もGPU実装されており，最適化されたMGCGソルバと共に2から64の弱スケーリングにおいて，良いスケーリング性能とCPU比での高い性能を達成できた．

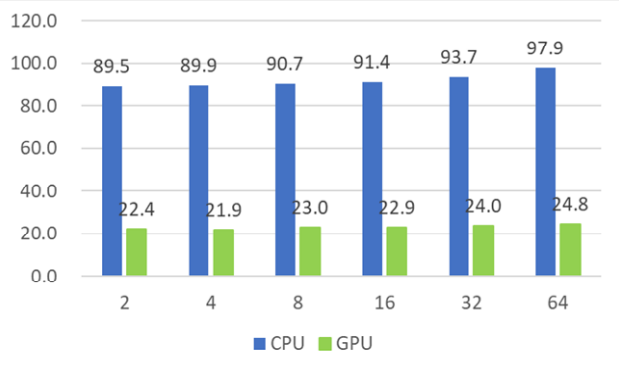


図 10　kinaco 全体のコスト．CPU と GPU の比較．縦軸：経過時間[秒]　横軸：並列数

8. まとめと今後

我々のグループでは，数値海洋シミュレーションの大規模問題での高速化を実現すべく，開発した非静力学数値海洋モデルkinacoの複数GPUへの実装と最適化を行っており，その内の一つである，MGCGソルバの通信処理の高度化と評価を紹介した．MPI通信用袖領域を分離したデータ構造の採用，演算とのオーバラップによる通信コストの隠蔽と，マルチグリッド前処理における粗い格子の集約を施すことによって，これによりソルバのコストの26%を削減でき，数値海洋モデル全体ではCPU比較で4倍の性能向上と64GPUまで良いスケーリング性能を得た．

今後はさらなる大規模実行でのhCGAの適用と評価を通じて，数百以上のGPUにて高い性能を実現し，海洋の働きが海洋資源や人為起源物質の輸送・拡散に与える影響や気象・気候との関係について詳細に調べる事を目指すと共に，広くCFDアプリケーションのGPU適用に応用できるような知見の蓄積を目指している．

9. 参照文献

[1] Matsumura, Y., H. Hasumi, A non-hydrostatic ocean model with a scalable multigrid Poisson solver., Ocean Modelling 24(1–2): 15-28., 2008.
[2] 松村義正，羽角博康，富山英治，山岸孝輝，井上孝洋，井上俊介，南一生，大島慶一郎. (2012). 非静力学モデルによる海洋微小スケールプロセスシミュレーション．京コンピュータ・シンポジウム 2012 .
[3] Yamagishi, T., Y. Matsumura, GPU Acceleration of a Non-hydrostatic Ocean Model with a Multigrid Poisson/Helmholtz Solver, Procedia Computer Science 80: 1658-1669., 2016.
[4] Yamagishi, T., Y. Matsumura , H. Hasumi, Multi-GPU Accelerated Non-Hydrostatic Numerical Ocean Model with GPUDirect RDMA Transfers, Supercomputing Conference 2018, 2018.
[5] Micikevicius, P. 3D finite difference computation on GPUs using CUDA, Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, ACM, 2009.
[6] Shimokawabe, T., T. Aoki, C. Muroi, J. Ishida, K. Kawano, T. Endo, A. Nukada, N. Maruyama , S. Matsuoka, An 80-Fold Speedup, 15.0 TFlops Full GPU Acceleration of Non-Hydrostatic Weather Model ASUCA Production Code, Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, IEEE Computer Society, 2010.
[7] Nakajima, K. New strategy for coarse grid solvers in parallel multigrid methods using OpenMP/MPI hybrid programming models. Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores. New Orleans, Louisiana, Association for Computing Machinery: 93–102. 2012.
[8] Nakajima, K. OpenMP/MPI Hybrid Parallel Multigrid Method on Fujitsu FX10 Supercomputer System. 2012 IEEE International Conference on Cluster Computing Workshops. 2012.
[9] Nakajima, K. Optimization of serial and parallel communications for parallel geometric multigrid method. 2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS). 2014.