Vol.2015-HPC-149 No.4
2015/6/26

SSD 搭載クラスタを用いた大規模ステンシル計算のための Out-of-core アルゴリズム

丹　英之[†1]　　緑川　博子[†1]

クラスタの総メモリサイズを越える大規模ステンシル計算を行うことを目的とし，Out-of-core アルゴリズムを提案する．総メモリサイズを越えた問題を扱うため，ドメインデータをクラスタの各ノードに搭載された SSD に分散し，MPI と非同期 I/O を用いることで処理を行う．初期評価では並列効率が 90%以上で，SSD の総容量までの規模のステンシル計算が可能となった．

The Out-of-core Algorithm for Large-scale Stencil Computation using SSD equipped Cluster

HIDEYUKI TAN[†1]　HIROKO MIDORIKAWA[†1]

We propose an out-of-core algorithm for to do large-scale stencil computation that exceeds the total memory size of cluster system. To deal with a huge problem larger than total memory size, this method divides the domain data into each nodes local SSD of the cluster, using MPI for internodal buffer exchange, and using asynchronous I/O for data transfer in SSD-DRAM. The parallel efficiency by the initial evaluation was more than 90%. The stencil calculation of the scale to total capacity of the SSD was enabled.

1. はじめに

ステンシル計算は, 流体シミュレーションなどでよく用いられる重要な計算カーネルである．格子点の値をその近傍格子点の値から求めることを格子系全体に渡って，時間ステップ毎に行う．このため，大規模，高解像度の問題を解くには大きなサイズの記憶装置が必要になる．記憶装置には，一般的に DRAM が用いられているが，計算機に搭載可能なメモリサイズには制限がある．

これまで我々は，メインメモリ拡張のため，今後普及すると想定される不揮発性メモリを「アクセス性能は劣るが大容量なメモリ」として扱うことを企図し，中でも入手し易い SSD を対象に，その使い方について検討してきた[1][2][3]．また，ステンシル計算を対象に，テンポラルブロッキングによりデータ参照局所性を高め, DRAM と SSD によるメモリ階層を効率よく利用する手法を検討してきた[4][5][6][7]．SSD をメインメモリとする手法には，1)スワップデバイスとして利用する swap 方式, 2)ストレージとして利用しファイルシステム上のファイルをマッピングする mmap 方式，3)ブロックデバイスを直接参照し，問題のアルゴリズム中でデータの入出力を非同期 I/O(AIO)で明示的に行う aio 方式の 3 手法がある．その中で aio 方式が他の 2 手法に比べ，よい結果を得ることができていた[4][5].

しかし, これら検討は 1 台の計算機を用いた評価であり，更なる問題の大規模化へ対応するには，問題を分割して処理を行う仕組みを導入する必要がある．そこで，aio 方式を用いた 1 ノードでのメインメモリ拡張の手法を元に，マルチノードに適用できるアルゴリズムを構築した．

本報告では，aio 方式による SSD を用いたメモリ拡張とテンポラルブロッキングを組み合わせた，ステンシル計算のマルチノード対応アルゴリズム，そして初期実装と簡単な評価について述べる．

2. マルチノードに拡張したテンポラルブロッキング

ここでは，Out-of-Core アルゴリズムである，メモリ階層構造を考慮したテンポラルブロッキングを，マルチノード向けに拡張したアルゴリズムについて述べる．

2.1 問題のノード分割

問題をマルチノードで処理するために，問題を各ノードに分割する必要がある．この問題格子を各ノードに分割する方法を図 1 に示す．図中 Domain-Buffer が対象とする問題の格子に相当し，このサイズを問題サイズとする．この格子は仮想的なもので，実際のデータは，各ノードに分割された状態で保持される(図中 Sub-Buffer)．つまり，計算開始前の初期化の段階で，問題格子のデータを各ノードに配置しておくことになる．計算は，3 次元格子データにおける近傍 7 点ステンシルを対象とし，問題格子の各次元サイズを(nx, ny, nz)とすると，境界条件の考慮で Domain-Buffer 格子は(nx+2, ny+2, nz+2)となる．

各ノードは問題格子の担当する部分を計算することになるが，ステンシル計算では境界条件が発生するため，各ノード間の「のりしろ」を考慮しなければならない．各ノー

†1 成蹊大学，JST CREST
Seikei University, JST CREST

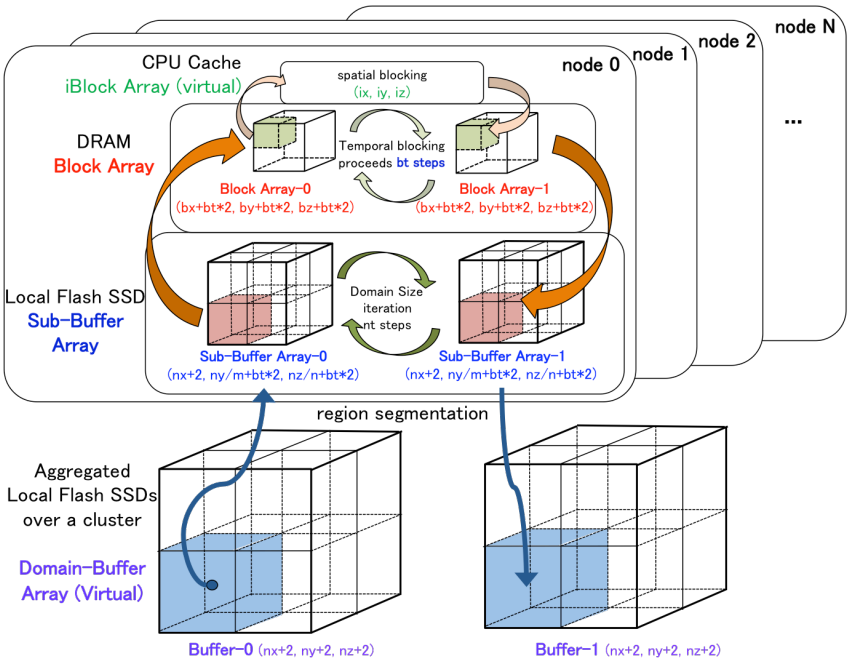

図 1　マルチノード向けテンポラルブロッキングのデータレイアウト

Figure 1　The data layout of temporal blocking for multi-node systems

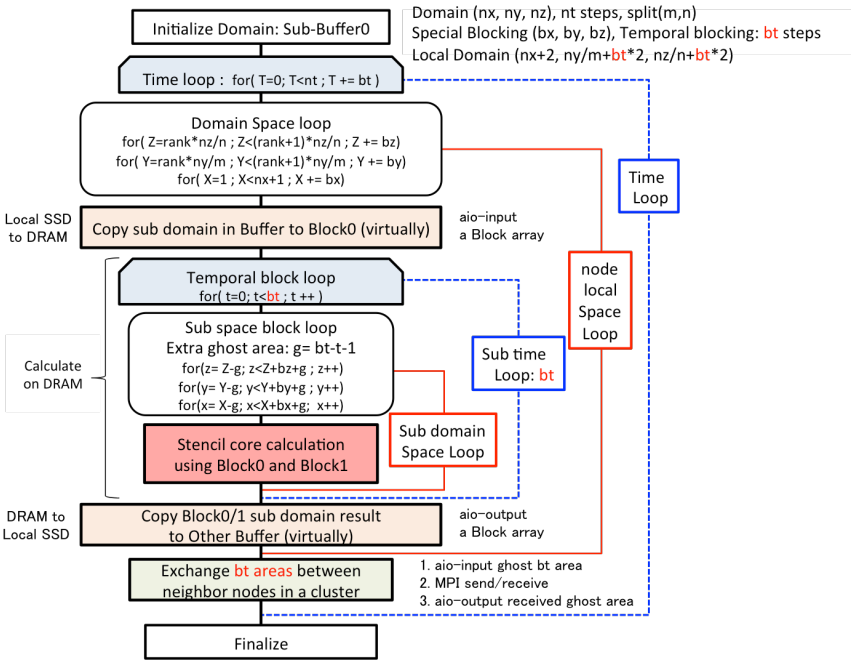

図 2　ブロッキングを用いた複数ノードでのステンシル計算

Figure 2　The stencil calculation using blocking for multi-node systems

ド内部の計算においてデータ参照の局所性を高めるためテンポラルブロッキングを行っている．このため，テンポラルブロッキングを行うステップ数 bt 分の計算に必要となる幅分の格子を「のりしろ」として各ノード間で冗長して確保する．ノード数 N のクラスタにて，問題格子を y, z 軸方向の 2 次元(m, n)で分割(この場合，m*n=N)すると，各ノードの担当する Sub-Bufffer 格子は，(nx+2, ny/m+bt*2, nz/n+bt*2)となる．この Sub-Buffer について各ノードの計算が終了した後，「のりしろ」部分をノード間で交換する．

2.2 ノード内ステンシル計算とノード間バッファ交換

前節で述べた，マルチノード対応のステンシル計算における全体の処理の流れを図 2 に示す．処理は(1)初期化，(2)終了ステップ nt までの時間ループ，(2-1)各ノード内でのステンシル更新計算，(2-2)隣接ノード間でのバッファ交換，から成り，これらを順に述べる．

(1) 初期化

与えられた問題格子(nx, ny, nz)を，指定された分割数(m, n)で分割し，各ノードの SSD に Sub-Buffer としてデータを配置する．

(2) 終了ステップ nt までの時間ループ

(2-1) 各ノード内でのステンシル更新計算

時間ループの中で，各ノードの担当分の領域を空間ブロック格子(bx, by, bz)毎にステンシルの更新処理を行う．

1. 空間ブロックのステンシル計算では，まず図 1 の Sub-Buffer Array-0 から Block0 領域に格子データを入力する．この入力には，Linux ネイティブ AIO(非同期 I/O)を用い SSD からマルチコア並列 I/O で DRAM へデータを読み込む．
2. この空間ブロックを対象に，ステンシル更新の演算を時間ブロッキング分 bt ステップ先に進める．片方の Block を参照し，もう片方の Block に演算結果を代入することを繰り返す．図 1 の Block Array の部分に相当する．また，更なる性能向上を企図し内部ループにおいて CPU の L3 Cache を考慮した空間ブロッキングを行っている．
3. bt ステップ分の更新が終了したら，計算結果のある Block 領域から，入力元とは異なる図 1 の Sub-Buffer Array へ格子データを出力する．この出力には，入力と同様，AIO を用い DRAM からマルチコア並列 I/O で SSD へデータを書き込む．

この空間・時間のブロッキングによりデータ参照局所性を高めることで，DRAM と SSD の性能差による処理時間への影響を小さくすることができる．

Linux ネイティブで非同期 I/O を実現するには，ストレージのオブジェクトを open(O_DIRECT)で開くことが必要である．このため，I/O バッファの開始アドレス，I/O サイズ，ストレージ上のオブジェクトを，ブロックデバイスサイズ(blkdev_size)境界に合わせる必要がある．このため，問題格子の nx と空間ブロック格子の bx に制約を設け，境界条件

を含めた x 軸の配列サイズを，blkdev_size の倍数となるようにした[6].

(2-2) 隣接ノード間でのバッファ交換

時間ブロッキングのステップ幅 bt 分 Sub-Buffer のステンシル更新計算が，全てのノードにおいて完了したら，隣接ノード間にて冗長して確保している bt ステップ分の Sub-Buffer の計算結果を交換する．計算結果は SSD に格納されている．そこで，MPI 通信バッファとして(2-1)でステンシル計算に用いた DRAM 領域である図 1 の Block Array を利用する．AIO を用いて SSD から DRAM に読み込み，MPI_Isend()/MPI_Irecv()を行い，通信完了後，再び AIO を用いて DRAM から SSD へ書き込む．詳細は，次節で述べる．これを Z 軸方向，Y 軸方向の隣接ノードの順に行う．

これら(2-1), (2-2)の操作を，時間ループの終了まで bt 毎に繰り返す．

2.3 隣接ノード間のバッファ交換

全てのノードで Sub-Buffer の更新が完了した後に行う，隣接ノード間のバッファ交換を図 3 に示す．バッファ交換は，Z 軸方向，Y 方向の順で行う．まず各ノードにて，計算結果がある Sub-Buffer から，Z 軸方向での両端 bt ステップ分の計算結果を読み込む(SSD to DRAM)．これを Z 軸方向での隣接ノード(左右ノード，図中(1)の rank *j-1* と rank *j+1*)に MPI_Isend()する．また，左右ノードから計算結果を受け取るため MPI_Irecv()する．通信完了後，受け取ったデータを計算結果のある Sub-Buffer，つまり次のステップの計算時に参照する Sub-Buffer の，ノード間「のりしろ」であるテンポラルブロッキング計算用の bt 分の冗長領域に書き込む(DRAM to SSD)．この操作が全ノードで完了した後，Y 軸方向でのバッファ交換を行う．Z 軸方向での交換と同様，各ノードにて計算結果のある Sub-Buffer から Y 軸方向での両端 bt テップ分のデータを読み込み，Y 軸方向での隣接ノード(上下ノード，図中(2)の rank *i* と rank *k*)に MPI_Isend()，上下ノードからの受信に MPI_Irecv()する．通信完了後，計算結果のある Sub-Buffer の bt 分の冗長領域に書き込む．この操作が全ノードで完了した後，各ノード内にて次の bt ステップのステンシル更新計算が開始される．

問題格子を 1 次元 z 軸方向だけで n 分割している場合(1, n)は，Z 軸方向のみのバッファ交換を行う．同様に，y 軸方向だけで m 分割している場合(m, 1)は，y 軸方向のみバッファ交換を行う．1 次元分割に比べ，2 次元分割ではバッファ交換の回数が増えるが，問題の形状に合わせた形で分割するノードの追加を見込める．すなわち，利用可能な DRAM, SSD が増えることで，より大規模な問題の処理が可能となる．

前節で述べたとおり，隣接ノード間のバッファ交換は，時間ループの中で時間ブロック bt ステップ毎に行われるので，bt が大きくなる程，ノード間通信の回数は低減される．そして時間ループ中のステンシル更新計算は，各ノード内で閉じており，完全に独立している．つまり，時間ステップ bt が，(1)ノード間通信，(2)ノード内 SSS-DRAM 間の I/O，の両方でデータ参照局所性抽出のパラメータとなっている．

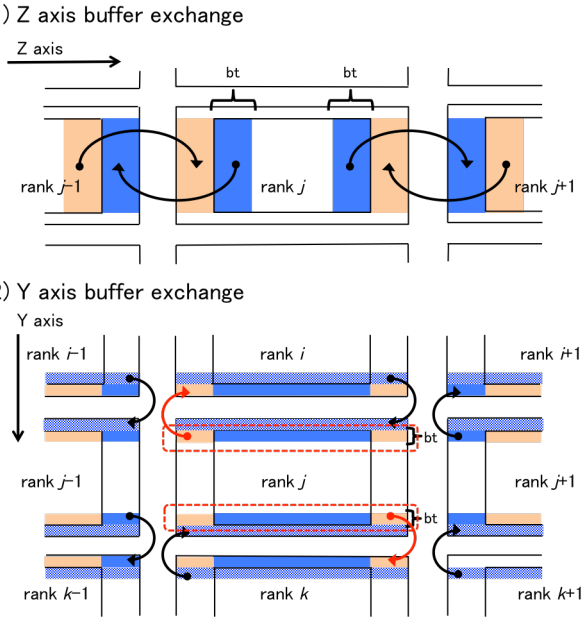

図 3　隣接ノード間のバッファ交換

Figure 3　Buffer exchanges between neighbor nodes

表 1　実験環境

Table 1　Experimental Environment

node	1	2	3	4
CPU	Intel Xeon CPU E5-2687W 3.1GHz (8cores) x 2			
Memory	8GiB DDR3 1600MHz ECC Reg			
	x16 (128GiB)	x8 (64GiB)	x8 (64GiB)	x8 (64GiB)
OS Kernel	CentOS7.1.1503, kernel 3.19.5			
Compiler	gcc 4.8.3 20140911 (option: -O3)			
Network	FDR InfiniBand x1 (56Gbps) Mellanox ConnectX-3			
MPI	MVAPICH2-2.0.1 (Thread Support Level = MPI_THREAD_MULTIPLE)			
Flash Storage	fusion-io ioDrive2(PCIe2.0x4)		Samsung XP941 MZHPU512HCGL (PCIe2.0x4) 512GB	
	1.2TB	785GB		

3. マルチノード対応ステンシル計算の初期実験

マルチノードに対応した拡張したテンポラルブロッキングのステンシル計算の効果を確認するため 4 ノードのクラスタシステムにて実験を行った．実験環境を表 1 に示す．

3.1 4 ノードクラスタでの性能実験

近傍 7 点ステンシル計算について，格子サイズ 2046x4096x8192 (1TiB), 256 タイムステップの問題を，空間ブロック 2046x1024x1024 格子, 時間ブロック 128 ステップ, 内部ループブロック 2048x2x30 のパラメータで，問題サイズを固定しノード数を変更して実行した．各ノードの SSD はブロックデバイスを直接 open(2)し，raw device として利用した．処理に要した時間を図 4 に，得られた相対性能比

を図 5 に示す．4 ノードでは，z 軸方向の 1 次元で分割した(1, 4)と，y, z 軸方向 2 次元で分割した(2, 2)を計測した．ノード数が増加すると共に，性能が向上している．並列効率は 2 ノードでは 98%，4 ノードでは，(1, 4)が 92%, (2, 2)では 95%であった．

2 次元分割では，バッファ交換が 2 回行われることから，全ノードの同期待ち回数も増加し，1 次元分割に比べ性能が低下することを予想していたが，4%性能が向上していた．ノード間のバッファ交換に要する時間が，ステンシル更新計算に要する時間に比べ，非常に短く，バッファ交換の影響が小さい．

1 ノードの担当する問題規模を一定にした場合についても計測した．1 ノードあたりの問題格子を 2Kx4Kx2K (256GiB)とし，z 軸方向に問題サイズを増加させた．ノード数 1, 2, 4 の順に，問題格子は 2Kx4kx2K (256GiB), 2Kx4Kx4K (512GiB), 2Kx2Kx8K (1TiB)となる．ブロッキングのパラメータは，問題サイズ 1TiB 固定の計測と同じ値を指定した．処理に要した時間を図 6 に，得られた相対性能を図 7 に示す．1 ノードと比較すると，2 ノードではコア実行時間の 3%，4 ノード(2, 2)ではコア実行時間の 5%がノード間のバッファ交換に要した時間となる．

この実験では，最大ノード数が 4 までであった．しかし，並列効率は 95%であり，各ノードに接続された SSD のサイズを増やすことで，更に大きな規模のステンシル計算を実施できることを示唆している．

3.2 実行時間の内訳

時間ループ中，各ノードにおけるそれぞれの処理に要する時間を計測した．問題格子のサイズを 4094x4096x4096 (1TiB)とし，これを 1 ノードと 4 ノード(2, 2)で実行した．ブロキングのパラメータは，問題サイズ 1TiB 固定と同じ値を指定した．1 ノード，4 ノードでの実行時間に対する各処理の時間内訳をそれぞれ図 8，図 9 に示す．1 ノードでのステンシル更新演算時間(図中ラベル calc)が，ノード間で 4 等分されていることが伺える．1 ノードでの初期化を除いたコア実行時間は，10323 秒，4 ノードでの各ランクの初期化を除いた平均コア実行時間は，2775 秒であった．ステンシル更新処理の前の Sub-Buffer から Block0 にデータを読み込む部分(図中ラベル calc_R)は，ステンシル演算と AIO のオーバラップをしておらず，1 ノード，4 ノード共に処理時間の 13%，更新処理後の Block から Sub-Buffer へデータを書き込む部分(図中ラベル calc_W)は，1 ノード，4 ノード共に 5%程度であった．ステンシルの更新演算時間(図中ラベル calc)は，1 ノードが処理時間の 83%，4 ノードでは 78%であった．

マルチノード向けアルゴリズムでは，MPI でのバッファ交換自体の処理の他に，バッファ交換前後でノード間のバリア同期を取る必要がある．ノード間でのステンシル演算に要する時間，SSD への I/O に要する時間のばらつきがある場合は，バリア同期に時間がかかる(図中ラベル barrier)場合があり，総実行時間に影響を及ぼす．バッファ交換にかかった時間(図中ラベル comm_W, comm, comm_R)は，コア実行時間の 3%未満であり，ステンシル更新処理がコア実行時間の 95%を占めることは，前節での 1 次元分割と 2 次元分割で性能差に大きな違いが見られなかった理由を裏

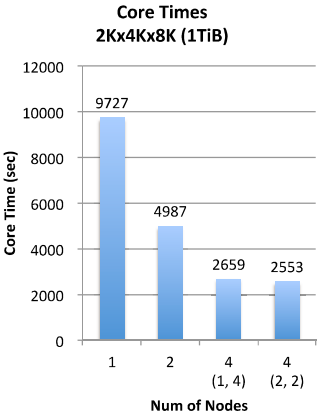

図 4　1TiB 問題コア実行時間
Figure 4　Core Times (1TiB)

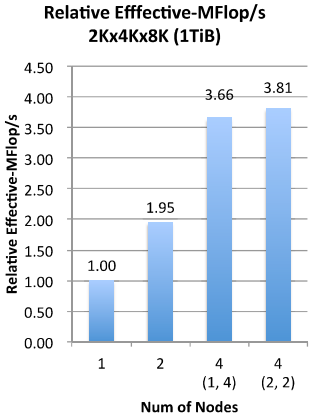

図 5　1TiB 問題相対性能
Figure 5　Relative MFLOPS (1TiB)

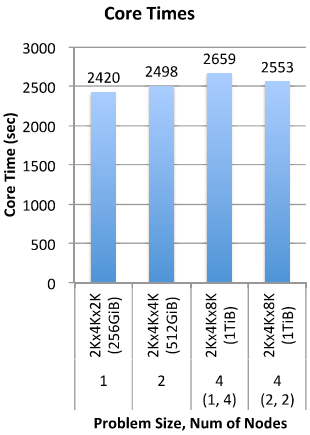

図 6　コア実行時間
Figure 6　Core Times

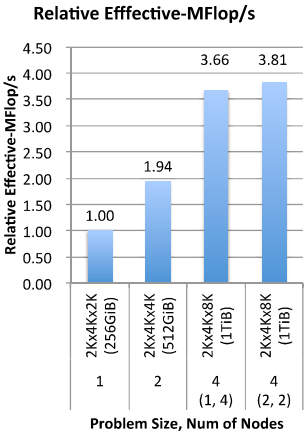

図 7　相対性能
Figure 7　Relative MFLOPS

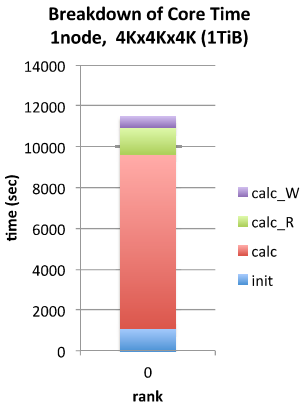

図 8　1 ノード時間内訳
Figure 8　The breakdown of Execution Time (1node)

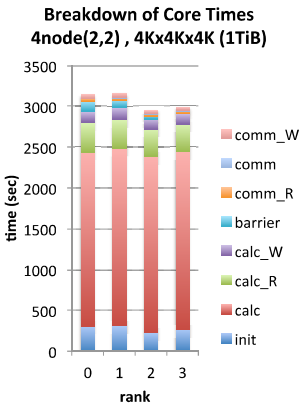

図 9　4 ノード時間内訳
Figure 9　The breakdown of Execution Times (4node)

付けるものである．

4. 多数ノードでのスケーラビリティ

前章ではマルチノード対応ステンシル計算を，4 ノードのクラスタで実行した際の性能評価について述べた．本章では，更にノード数の多い環境下でのスケーラビリティについて述べる．

実験には，SSD が搭載されているクラスタシステムの TSUBAME2.5[8]を用いた．Intel Xeon 5670 2.93HGz (6cores) x 2, PC3-10600 96GiB, SSD 240GB, QDR InfiniBand x2 のハードウエアで稼働する，キューS96 のジョブとして実行した．コンパイラは gcc 4.3.4(オプション-O3)でビルドし，MPI は MVAPICH2-2.0.1 を用いた．TSUBAME で利用できる SSD は，ext3fs で各ノードのローカルストレージとしてマウントされており，ブロックデバイスを直接参照することは出来ない．そこで，各ノードの Sub-Buffer 初期化時に SSD のマウントポイント上にファイルを生成し，それを AIO で参照するようにした．また，SATA 接続の一般的な SSD を RAID0 構成にしたもので容量が小さい．したがって， SSD/DRAM 比も小さくなり大容量のメモリ拡張実験としてではなく，多数ノードの実験環境として利用した．

予備実験として，簡単なベンチマークを行い SSD の性能を調査した．前章で用いたクラスタに搭載されている SSD と TSUBAME に搭載されている SSD の I/O バンド幅を図 10 に示す．アクセスパターンはシーケンシャル，16 スレッド(TSUBAME は 12 スレッド)でそれぞれ 9216MiB のデータを 12MiB の単位で読み込み，4096MiB のデータを 8MiB 単位で書き込んだ時のバンド幅である．4.4〜5.8 倍の性能差がある．

4.1 多数ノードクラスタでの性能実験

近傍 7 点ステンシル計算について，格子サイズ 4094x4096x4096 (1TiB), 256 タイムステップの問題を，空間ブロック 4094x1024x512 格子，時間ブロック 32 ステップ，内部ブロックループ 2048x2x30 のパラメータで，ノード数を 8(2, 4 分割), 16(4, 4 分割), 32(4, 8 分割)と変更して実行した．処理に要した時間を図 11 に，得られた相対性能を図 12 に示す．8 ノードを 1 とした場合の相対並列効率は 16 ノードで 97%, 32 ノードで 91%であった．この 1TiB の問題サイズは，SSD の容量の制約(実際にユーザが利用できるサイズの上限は約 153GiB)から 7 ノード未満では実行できない．このため，1 ノード実行時との比較はできていない．

1 ノードの担当する問題規模を一定にした場合についても計測した．1 ノードあたりの問題格子を 4Kx2Kx1K (128GiB)とし，ノード数に合わせ問題サイズを大きくした．ノード数 4, 8, 16 の順に，問題格子は 4Kx4Kx2K (512GiB), 4Kx4Kx4K (1TiB), 4Kx4Kx8K (2TiB)となる．ブロッキングのパラメータは，問題サイズ 1TiB 固定と同じ値を指定した．処理に要した時間を図 13 に，得られた相対性能比を図 14 に示す．ノード数が増すにつれ，実行時間は長くなる傾向が見られるが，8 ノード，16 ノードどちらとも，4 ノードと比較して全体の約 5%程度の増加が見られた．ノード数が増えるほど，アルゴリズム中の全ノードでバリア同期を取る部分の影響が現れていると考えられる．しかし，少数ノード時の実験と同様，ノード内のステンシル計算がコ

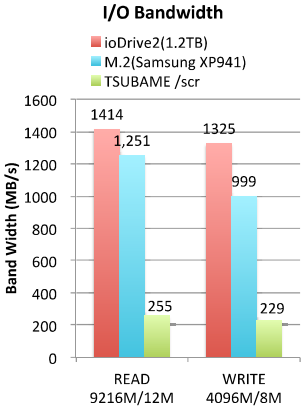

図 10 I/O バンド幅
Figure 10 I/O Bandwidth

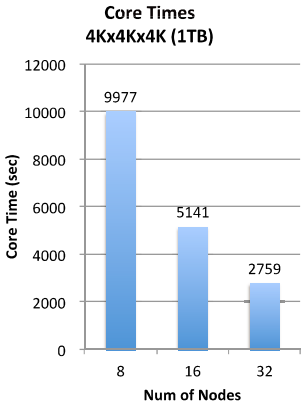

図 11 1TiB 問題コア実行時間
Figure 11 Core Times (1TiB)

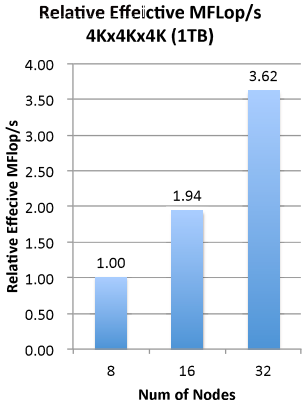

図 12 1TiB 問題相対性能
Figure 12 Relative MFLOPS (1TiB)

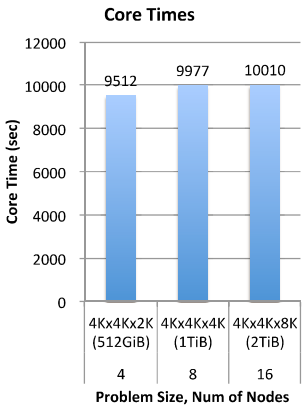

図 13 コア実行時間
Figure 13 Core Times

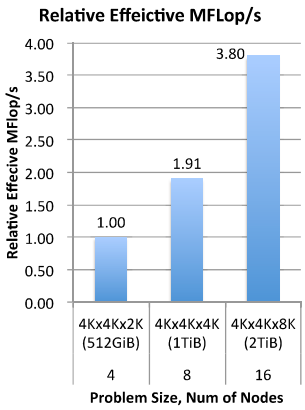

図 14 相対性能
Figure 14 Relative MFLOPS

ア実行時間の大部分を占めており，ノード間のバッファ交換に要する時間が相対的に短い．各ノードの担当する問題サイズを大きくすることで，実行時間に対するバッファ交換の影響を低減できると考えられる．

これら結果は，多数のノードを利用することにより，シングルノードでは実行できない規模のステンシル計算を，極端な性能低下を伴わずに実施できること示している．

4.2 問題格子の形状と分割方法

前節での実験を実施するにあたり，各ノードが担当する分割された問題のサイズは，クラスタのリソースの範囲内であることを考慮しておかなければならない．すなわち，問題格子のサイズと形状，そしてノードに搭載された SSD のサイズにより決まる問題分割方法について検討した．

8 ノードにおいて，問題の形状が異なるがデータサイズは同じ 1TiB のステンシル計算を 2 パターンで行うことにした．但し，各ノードが担当する問題サイズは，上記で述べたとおり，SSD の容量の制約があるので，ブロッキングパラメータを調整することで対応した．これら問題格子，ブロッキングパラメータとコア実行時間を表 2 に示す．形状が立方体の問題を 4Kx4Kx4K とし，8 ノードで(2, 4)分割すると，1 ノード担当分である Sub-Buffer は，4Kx1Kx2K となる．一方，直方体の問題を 2Kx2Kx16K として，8 ノードで(1, 8)分割すると，Sub-Buffer は，2Kx2Kx2K となる．Sub-Buffer には，時間ブロック bt ステップ分のノード間「のりしろ」が分割方向の両端に追加されるが，当然 Sub-Buffer のサイズは，ノードに搭載された SSD のサイズ以内である必要がある．このため，直方体の問題では z 軸方向のみの「のりしろ」だけなので bt を 128 ステップとしたが，立方体の問題では y, z 軸方向の「のりしろ」が追加されるため，bt を 32 ステップへ縮小した．

表 2　サイズが同じ問題のパラメータと実行時間
Table 2　Same Size Problem Parameter and Core Time

shape	cube	cuboid
node	8	8
Problem Size (GiB)	1024	1024
Domain-Buffer	4Kx4Kx4K	2Kx2Kx16K
calc step (nt)	256	256
temporal block (bt)	32	128
time loop iteration	8	2
split	(2, 4)	(1, 8)
Sub-Buffer	4Kx2Kx1K	2Kx2Kx2K
Sub-Buffer Size(GiB)	140	136
Block	4Kx1Kx512	2Kx1Kx1K
Block Size(GiB)	38	50
core time (sec)	9977	5902
Effective-MFlop/s	17624	29779

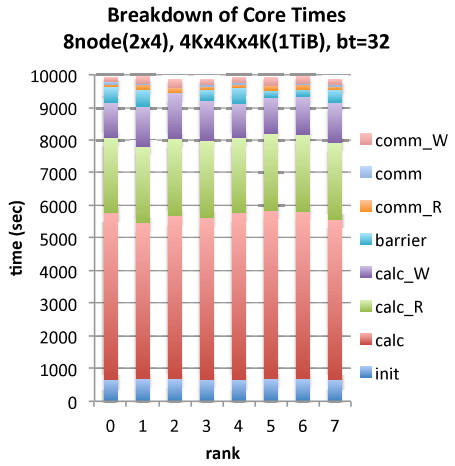

図 15　立方体問題の時間内訳
Figure 15　The breakdown of Core Times (cube)

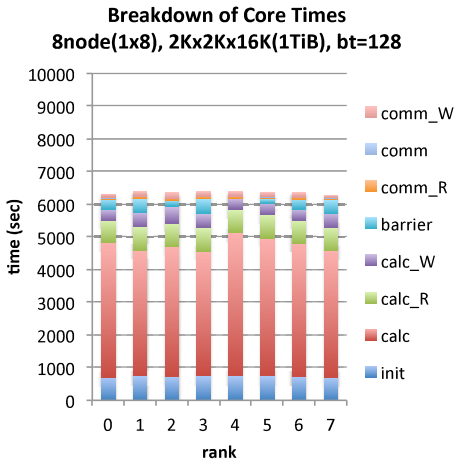

図 16　直方体問題の時間内訳
Figure 16　The breakdown of Core Times (cuboid)

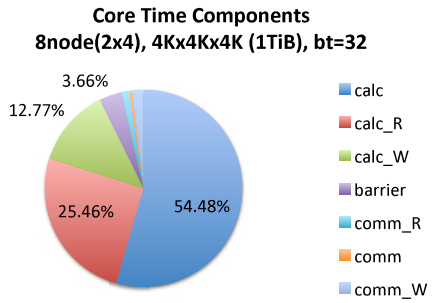

図 17　立方体問題コア実行時間成分
Figure 17　Core Time Components (cube)

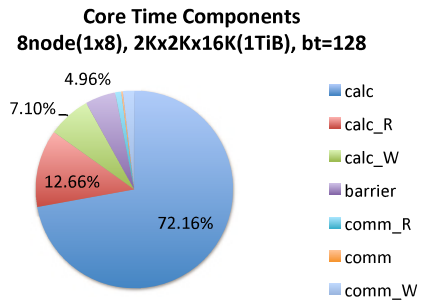

図 18　直方体問題コア実行時間成分
Figure 18　Core Time Components (cuboid)

これら問題のコア実行時間を比較すると，同じ問題サイズを扱っているにもかかわらず，立方体では，直方体の 1.7 倍の時間を要した．この差は，時間ブロックの縮小したことで，時間ループの繰り返しが 2 回から 4 倍の 8 回に増加したことに依るもので，MPI 通信によるバッファ効果に依るものではない．

立方体, 直方体の問題実行時の時間内訳をそれぞれ図 15, 図 16 に示す．両者を比較すると，立方体では，ステンシル更新での Sub-Buffer, Block 間での I/O の時間(図中ラベル calc_R, calc_W)が約 3 倍増加しており，時間ループの繰り返し回数が増えることで SSD-DRAM 間の I/O 回数も増え，その分，実行時の性能低下を引き起こしていることがわかる．また，各ノードでステンシル更新の演算時間(図中ラベル calc)が約 20%増加している．これもテンポラルブロッキングの冗長計算が，時間ループの繰り返し回数分積算されたものである．これら結果から，問題実行時の時間ブロックパラメータ bt は，慎重に決定する必要があり，如何に bt を大きくしてステンシル更新演算処理におけるテンポラルブロッキングによるデータ参照局所性を高めるか，が重要であることがわかる．

立方体問題，直方体問題でのコア実行時間における各処理に要した時間内訳を，それぞれ図 17, 図 18 に示す．直方体では，Sub-Buffer, Block 間での I/O(図中ラベル cacl_R, calc_W)がコア実行時間の 20%であるが，立方体では 38%と，コア実行時間に対し大きな割合を占めている．直方体では時間ブロッキングが大きい分，ステンシル更新の演算を高速化できていることがわかる．ノード間の各処理に要する時間のばらつきがノード間の同期時間(図中ラベル barrier)となる．ノード間のバッファ交換は，どちらの形状でもコア実行時間の 4%と，ステンシル更新処理に比べ，非常に小さい割合であり，分割方法によるバッファ交換回数の違いは性能に大きな影響を及ぼさない．

ステンシル更新の処理は独立性が高く，ノードが増加した場合でも，並列効率はあまり低下しないことを示唆している．

5. おわりに

メモリ階層構造を考慮したテンポラルブロッキングによるステンシル計算を，マルチノード向けに拡張したアルゴリズムを設計し，実装と初期評価を行った．

その結果，ノード間同期によるオーバーヘッドで性能低下が伴うが，並列効率は 90%以上であり，更にノードに搭載される SSD 容量を大きくすることで, 大規模のステンシル計算を実施できることが示唆された．

時間ブロック bt において，最適値のある DRAM-Cache 間におけるテンポラルブロッキング適用時とは異なり，SSD-DRAM 間のテンポラルブロッキング適用においては，bt ステップを最大限大きくすることが重要であることが改めて確認された[5].

シングルノードにおける各種ブロッキング自動パラメータ選択アルゴリズムは既に構築している[7]. マルチノードシステムにおいては，新たに MPI 通信が必要になるため，MPI 通信バッファとの兼ね合いも考慮する必要がある．今後は，与えられた問題と，それを実施する環境から，最適なブロッキングパラメータを決めるチューニング手法を確立したい．

コア時間の内訳を見ると，テンシルの更新演算が最も時間を要していることがわかった．この部分に対しては，GPGPU によるアクセラレーションを実施したい．

次期 TSUBAME など，今後増えると予想される各ノードに大容量の PCI バス接続型 SSD が搭載された多階層メモリのクラスタでは，データ参照局所性を高めたアルゴリズムとの併用により，SSD をメモリ拡張として用いることが十分効果のあることが示された．

参考文献

1) 緑川博子,丹英之:"メモリサイズを越えるデータ処理を目的としたバス接続型 SSD の性能評価",情報処理学会，ハイパフォーマンス研究会,Vol.2013-HPC-140, No.44, pp.1-6, (2013.8)

2) 丹英之,緑川博子:"フラッシュ向け Linux スワップシステムの評価",電子情報通信学会,コンピュータ・システム研究会 Vol.113, No.282, pp.61-66, (2013,11)

3) 丹英之,緑川博子:"フラッシュ SSD をメモリセマンティクス API で用いるための予備調査",ハイパフォーマンスコンピューティングと計算科学シンポジウム HPCS2014, HPCS2014 論文集, (2014,1)

4) Hiroko Midorikawa, Tan,Hideyuki, Toshio Endo, ”An Evaluation of the Potential of Flash SSD as Large and Slow Memory for Stencil Computations”, IEEE The 12th International Conference on High Performance Computing & Simulation, HPCS2014, 2014, pp.268-277,

5) 緑川博子，丹英之: "大規模ステンシル計算のための Flash SSD 向けテンポラルブロッキングの性能評価",情報処理学会, ハイパフォーマンスコンピューティング研究会,Vol.2014-HPC-145, No.22, pp.1-9, (2014.7)

6) 丹英之,緑川博子:"ブロックデバイス非同期 I/O によるフラッシュストレージを用いたステンシル計算の性能評価",電子情報通信学会，コンピュータ・システム研究会 CPSY2014-52, pp.31-36, （2014,10/10）

7) Hiroko Midorikawa, Hideyuki Tan "Locality-Aware Stencil Computations using Flash SSDs as Main Memory Extension", Proc. of IEEE/ACM International Symp. on Cluster, Cloud and the Grid Computing CCGrid2015, pp.1163-1168, 2015-5 (DOI 10.1109/CCGrid.2015.126)

8) TSUBAME2.5, http://tsubame.gsic.titech.ac.jp