Flash を用いた Out-of-core ステンシル計算のための最適ブロッキングパラメタ自動チューニングシステム

緑川 博子[†1]　丹 英之[†2]

概要： 主メモリサイズを超えるような大規模サイズのステンシル計算を行う際に，flash SSD を主メモリ拡張として用い，時間，空間ブロッキング手法を導入してデータアクセス局所性を高めることで，実用上，十分な性能で処理ができることを明らかにしてきた．しかし，ユーザの計算機環境や，問題サイズなどのパラメタの違いにより，最適なブロッキングパラメタは異なるため，適切なパラメタを設定するには，事前の計算や手動によるパラメタ設定が必要であった．本報告で提案する *Blk-Tune* は，実行時に計算機のハードウエア情報を自動的に抽出し，これをもとに，flash と DRAM との間の明示的な IO のデータ転送量を最小にするブロッキングパラメタを自動的に探索計算し最適値を選定するシステムである．多くの自動チューニングシステムとは異なり，利用する計算プラットフォームでの事前実行や自動チューニングに必要なパラメタ取得のための多数回にわたる評価実行などを必要としない．また，プログラムの静的情報のみを利用したコンパイラなどによる最適化手法と異なり，実際のステンシル計算プログラムのフロントエンドとして稼働し，実行時のプラットフォーム環境と問題パラメタに応じた最適ブロックサイズを即時選択して，後段のステンシルプログラムに提供することを特徴とする．これにより，メモリ，キャッシュサイズ，CPU ソケット，コア数などのハードウエア情報と，問題サイズやステップ数に応じて，ユーザの介入なしに，最適なパラメタを計算して実行することができるようになり，flash への入出力を最小にすることで最大性能を得ることが可能になった．

キーワード：自動選定システム，自動チューニング，不揮発性メモリ，フラッシュメモリ，メモリ階層，タイリング，テンポラルブロッキング，ステンシル，アウトオブコア，非同期入出力，mmap，ブロック，主メモリ拡張

1. はじめに

ステンシル計算は，多くの科学技術，工学分野において広く用いられる重要な計算カーネルの一つである．ステンシル計算は, メモリアクセス逐次性があり, 規則的なため，古くから，データ空間を小さい領域に分割し，メモリアクセス局所性を高めて高速化されている．さらに繰り返し計算である特徴を生かし，空間ブロック化に加え，時間軸における時間ブロック化を行うテンポラルブロック手法についても様々な研究が行われている．これまで，このようなブロック化手法は，キャッシュと主メモリ（DRAM）間への適用が主であり，CPU ホストメモリと GPU メモリ間，クラスタにおけるノード間などにも適用されている．筆者らは，主メモリサイズを超えるような大規模サイズのステンシル計算を行う際に，flash SSD を主メモリ拡張として用い，時間，空間ブロッキング手法を導入してデータアクセス局所性を高めることで，実用上，十分な性能で処理ができることを明らかにしてきた [2-5].

しかし，時間，空間における適切なブロックサイズは，ユーザの計算機環境や，問題サイズパラメタなどにより異なるため, 最適なブロックサイズパラメタを決定するには，多くの場合，事前の予備実行や手動によるパラメタ設定の試行錯誤などが必要であった．我々が提案する *Blk-Tune*[6] は, 実行時に計算機のハードウエア情報（主メモリサイズ，キャッシュサイズ，CPU ソケット数，コア数など）を自動的に抽出し，これをもとに，flash と DRAM との間の入出力（IO）データ量を最小にするブロッキングパラメタを自動的に探索計算し最適値を選定するシステムである．多くの自動チューニングシステムとは異なり，異なる複数のパラメタによる事前実行や事前の情報収集などを必要としない．また，プログラムの静的情報のみを利用したコンパイラなどによる最適化手法とは異なり，実際のステンシル計算プログラムのフロントエンドとして稼働し，実行時のプラットフォーム環境と問題パラメタに応じた最適ブロックサイズを即時選択して，後段のステンシルプログラムに提供することを特徴とする．これにより，ユーザの介在を一切なしに，実行時に利用するハードウエア情報と問題設定に最適なパラメタを選定し，即時実行することが可能となり，flash デバイスへの入出力量を最小にすることで最大の性能を得ることができる．

2. 主メモリ拡張のための flash 利用手法

flash と DRAM のアクセス遅延時間の差は，1000 倍程度あり，DRAM とキャッシュ間の差に比べると非常に大きい．このため，同じテンポラルブロッキングアルゴリズムを用いる場合でも, DRAM-キャッシュ間でのパラメタ選択や手法とは状況が大きく異なる．

これまで, 主メモリ拡張として flash を用いる手法として以下の３つを調査してきた[2-3]. 1) Linux における fast-swap（OpenNVM）[7-8]を利用し flash をスワップデバイスとして用いる swap 方式[2]，2)flash をファイルシステムとして利用し，ファイルをメモリマップで利用する mmap 方式，　3) flash をブロックデバイスとして用い，Linux に新

†1 成蹊大学　Seikei University.
†2 (株)アルファシステムズ　Alpha Systems,Inc.

ⓒ2016 Information Processing Society of Japan

たに導入されたマルチコア対応の IO サブシステム[9-10]の利点を生かし，高並列非同期入出力を用いる aio 方式である．

swap 方式はユーザプログラムに完全な透過性があるが，性能は低く，不安定である．メモリ枯渇状態で起動するスワップデーモンの使命は，古くからプロセスや OS を守るための緊急対処という色合いが強く，データの退避，時にはプロセス強制終了などを伴い，安定かつ高速に応用プログラムを実行する環境としては，未だ適さない．

mmap 方式は，メモリアクセスを前提に書かれた既存プログラムの書き換えがほとんど不要である点で優位性があるが，ファイルをメモリに展開した際のページキャッシュの置き換えはすべて OS カーネルが制御しており，汎用であるがゆえに，個々の応用に応じた適切な制御ができないという欠点がある．また，OS カーネルの種類やバージョンにより，どのような動きをするのかの予測も難しい．

aio 方式で，明示的な flash への IO はユーザや応用に合わせて制御可能である点が大きな利点であり，事実，3 手法のうち，最も高性能が得られる．前述の mmap 方式では，OS によるページキャッシュ領域として，主メモリを一定容量分，空けておく必要性があり，その利用可能メモリ容量の大小により mmap 利用時の性能が変わってくる[4-5]．一方，aio 方式では，原則として，OS 稼働用の最低限の主メモリを確保すれば，応用プログラムが主メモリのほとんどを DRAM 上のデータとして利用することが可能となる．

3. Flash 利用テンポラルブロッキングステンシルアルゴリズム

flash 向けテンポラルブロッキングアルゴリズムにおけるデータ構造，疑似コード，処理概要をそれぞれ，図 1，図 2，図 3 に示す．テンポラルブロッキングアルゴリズムは，冗長計算あり冗長計算なしの 2 つに分類できるが，図 2，図 3 は冗長計算なしアルゴリズム，図 4 は冗長計算ありアルゴリズム[2][3]の場合を示す．

最適ブロックサイズの選定に関わる Flash 向けテンポラルブロッキングアルゴリズムの概要を述べる．たとえば，3 次元格子データ　(nx, ny, nz)を nt 時間ステップ，近傍格子データ（7 点，19 点，27 点など）を用いて更新する問題の場合，図 2 に示すように，空間ブロックサイズとして(bx, by, bz) ，時間ブロックサイズとして bt　ステップを選び，時間・空間ループをそれぞれ 2 レベルに分割して，内部ループの計算においてメモリアクセス局所性を高めようとするアルゴリズムである．ここでは，図 1 に示すように，全体の問題サイズは主メモリサイズを超えており flash に格納されている．flash のソースバッファにあるデータの一部を，主メモリ（DRAM）に入るサイズのブロックアレイに読み出し，DRAM 上で bt ステップの更新計算をした後，結果を flash のデスティネーションバッファへ書き込む．

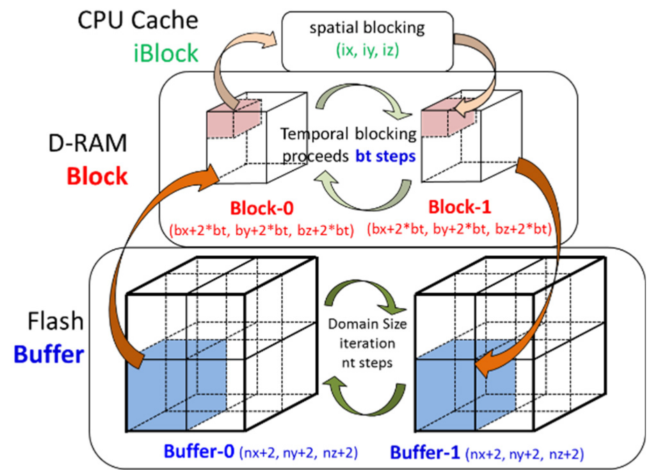

図 1　各メモリ階層におけるデータ構造

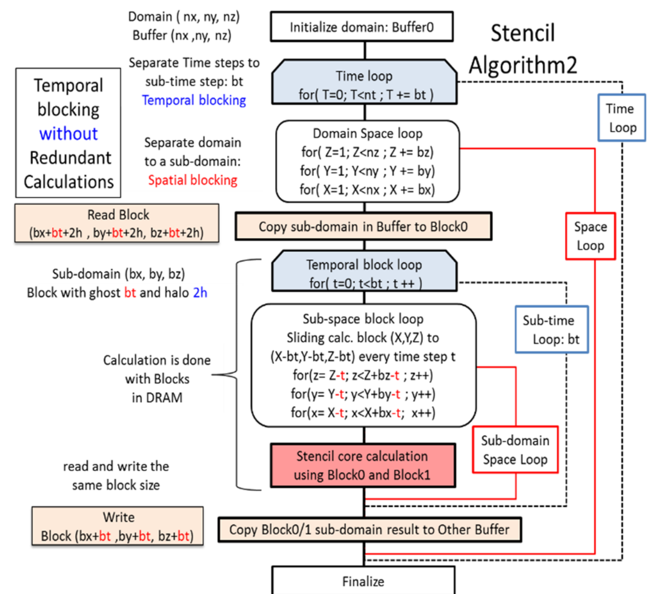

図 2　冗長計算なしテンポラルブロッキングアルゴリズム

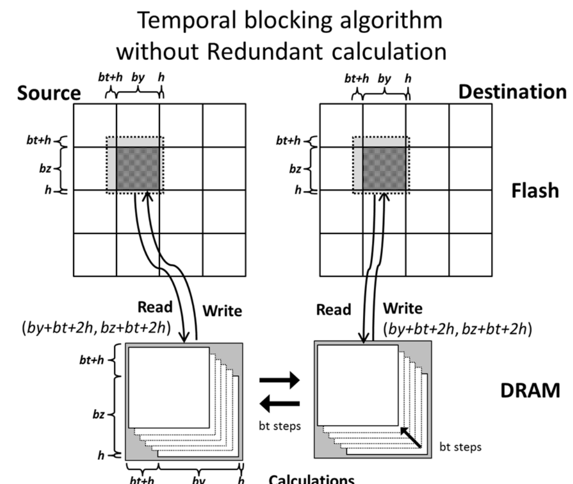

図 3　冗長計算なしテンポラルブロッキングアルゴリズムにおける DRAM 上での計算と Flash の読み書き

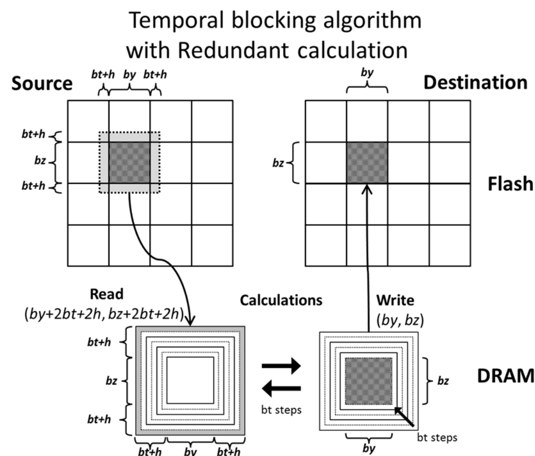

図 4 冗長計算ありテンポラルブロッキングアルゴリズムにおける DRAM 上での計算と Flash の読み書き

テンポラルブロッキングアルゴリズムでは，図 3, 図 4 に示すように，もともとの空間ブロックサイズよりも，時間ブロックサイズ *bt* 分，大きいサイズの領域を flash から DRAM に読み出し，*bt* ステップ数分の計算を行ってから，flash へ書き戻す．冗長計算ありの場合は，読み出しサイズと書き込みサイズは異なり（図 4)，冗長計算なしの場合には，同じサイズを読み書きする(図 3).

それぞれのアルゴリズムに応じたサイズのブロックアレイを用い，*bt* 時間ステップ分の計算を DRAM 上で行う．このため，用いる計算プラットフォームの DRAM 容量に応じて，適切な空間ブロックサイズと時間ブロックサイズを選定することが処理高速化に必要である．空間ブロックサイズは DRAM 容量を最大限利用する大きさにしたほうが，flash 入出力回数を低減させ，マルチコアによる処理も効率的である．一方，全データ領域を走査する回数は，問題時間ステップ数 *nt* を時間ブロックサイズ *bt* で除した数であるため，*bt* が大きければ，よりデータアクセス局所性を高めることができる．このため，決められた容量の主メモリをどのように空間・時間ブロックサイズで分け合うか最適な点を見つけるのは難しい. 空間ブロックサイズでは，ブロックアレイの体積（容量）だけではなく，各空間次元の長さなど，ブロックアレイ形状も性能に影響を与える．問題データパラメタやデータの境界条件などによっても，効率的な空間ブロック分割の仕方が変わってくる場合もあり，問題設定と計算プラットフォームによって，最適な空間・時間ブロックサイズが変化する．

さらに，冗長計算ありアルゴリズムの場合には，時間ブロックサイズ *bt* が大きくなるとそれに応じて，冗長計算が増加するため，冗長計算オーバヘッドとアクセス局所性増加による高速化の間でトレードオフも存在する．冗長計算なしアルゴリズムの場合には，Flash と DRAM 間のデータ転送量を最小にする空間・時間ブロックサイズの組み合わせを最適値として選ぶことができる．冗長計算ありアルゴリズムにおいては, 冗長計算時間の増加と flash とのデータ転送時間の低減との間でトレードオフ判定が必要なため，用いる flash のアクセスバンド幅と CPU 性能指標を利用する．以降の節では，主に冗長計算なしアルゴリズムに対する空間・時間ブロックサイズの最適パラメタ選定の方式とそのシステムについて説明する．

aio 方式を用いるステンシルアルゴリズムでは，図 1 の DRAM における 3 次元ブロックアレイのメモリレイアウトとして，これまでの評価[5]から，図 5 に示すような構造を用いている．NUMA システムにおける性能低下を防ぎ，並列非同期入出力 aio における制約を満たしつつ，高性能な IO ができるように，1 つの IO 操作は，図 5 の xy 平面を単位としている．各 xy 平面の先頭アドレスとサイズは，いずれも flash のブロックサイズの倍数になっている．また，問題データ（図 1 のバッファアレイ）の x 次元のサイズ *nx* とブロックアレイの *bx* は等しくするという前提を設けている．

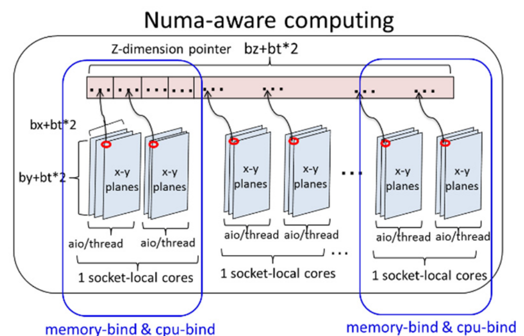

図 5 NUMA システムにおける aio 方式向けのブロックアレイのレイアウト

4. ブロッキングパラメタ自動選定システム

本報告では，aio 方式を用いた flash 向けテンポラルブロッキングステンシル計算において，計算プラットフォームの主メモリ（DRAM）容量，コア数，ソケット数などに応じた適切な空間，時間ブロックサイズの最適値を自動選定するシステムについて述べる．

flash を用いた Out-of-core 向けアルゴリズムでは，空間・時間ブロックサイズの違いによる flash への総入出力(IO)量の差が性能上の指標になる．そこで，与えられた問題 (*nx, ny, nz*，*nt*) に対し，ブロックサイズ(*bx, by, bz*，*bt*)を決めたときの flash との IO 量をコスト関数とし，これを最小化するパラメタを計算し抽出するシステムを構築した．

4.1 ブロッキングパラメタ自動選定システムの概要

図 6 は，ブロックパラメタ自動選定システム（*Blk-Tune*）の全体概要を示す．このシステムでは，図 1 に示す flash, DRAM, cache の 3 階層のメモリに対応して，ブロッキングパラメタを自動選定する． *Blk-Tune* は，ステンシル計算実

行時に Portable Hardware Locality (hwloc) [1]を用いて計算プラットフォームの各種ハードウエア情報をリアルタイムで取得する．hwloc は広範囲の OS と CPU アーキテクチャをサポートしており，ポータビリティが高い．これにより，各メモリ階層容量, コア数, ソケット数などの情報を得て，実行時にプラットフォームに応じた最適なブロックサイズを自動計算し，後段のステンシル計算への入力パラメタとする．すなわち，*Blk-Tune* は，ステンシル計算プログラムのフロントエンドとして利用することが可能で，Just-In-Time で，最適ブロックパラメタを自動設定する．多くの自動チューニングシステムが前提としている利用プラットフォームにおける事前実行や，自動チューニングに必要なパラメタ取得のための多数回にわたる評価実行などが不要である．また，*Blk-Tune* は後段ステンシル計算と切り離し，オフラインで単体利用することも可能で，問題サイズなどを変えて事前評価や分析に利用することもできる．また，ハードウエア情報の自動取得をオフにして手動で主メモリ容量などの情報を入力することで，利用プラットフォーム以外のシステムで評価や分析を行うこともできる．

Blk-Tune を利用したステンシル計算のユーザコマンドは，たとえば，問題サイズ(nx,ny,nz)，時間ステップ数(nt)　と flash のパスを以下のように入力するだけである．

```
./stencil7p -n 4094 4096 2048 -t 1000 -d /dev/sdc
```

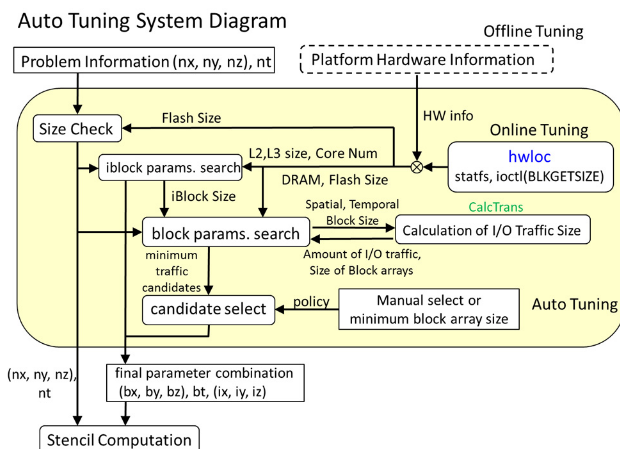

図 6　ブロッキングパラメタ自動選定システム

図 6 のシステムにある CalTrans は，(*bx, by, bz, bt*)における flash への IO 総量（コスト）を計算する．この計算は，図 3，図 4 のアルゴリズムにより異なり，問題ごとの境界条件など影響もすべて考慮する．*Blk-Tune* は，空間・時間ブロックサイズ(*bx, by, bz, bt*)により形成されるブロックアレイの容量が主メモリサイズに入るという拘束条件の下で，大域的にコストが最小となるブロックサイズの組み合わせを探索し出力する．同コスト値の候補が複数存在した場合には，手動選択あるいは，主メモリの専有容量が最小のものを選択する．*Blk-Tune* は，主メモリ-flash 間のブロックサイズだけでなく, DRAM-キャッシュ間の空間ブロックサイズについても，ヒューリスティックな手法で，準最適値を出力する．

4.2 問題定義：自動選定システムへの入出力と前提

本システムの入出力と前提について以下に示す．

(1) 自動選定システムへの入力

- ハードウエア情報：

  各メモリ階層の容量と CPU 情報

  Sc2: レベル 2 キャッシュ，*Sc3*: レベル 3 キャッシュ，

  Sm: 主メモリ容量,

  Sf: flash デバイス容量，*Sb:* flash デバイスブロックサイズ,

  Nc: システム全コア数, *Ns*: CPU ソケット数

- 問題情報：

  nx, ny, nz：問題サイズ，　*nt:* 時間ステップ

(2) 自動選定システムからの出力

- flash 向け　最適　空間・時間ブロックサイズ

  bx, by, bz：空間ブロックサイズ，*bt:* 時間ブロックサイズ

- キャッシュ向け 準最適　空間ブロックサイズ

  ix, iy, iz：空間ブロックサイズ（図 1 の *iBlock* サイズ）

(3) 前提（flash デバイスに問題データが入る）

$Sf > nx * ny * nz * k * Esz$

k: ダブルバッファアレイなら 2.

Esz：データ要素サイズ（バイト）

nx　：*Sb* の倍数（非同期入出力 aio の制約）

5. 最適ブロッキングパラメタ選定手法

次に，本システムで用いている最適ブロックサイズ探索のための手法を示す．

5.1 ステップ 1： *iBlock* サイズ（*ix, iy, iz*）の選定

これまでの実験結果を基にしたヒューリスティック手法により， *iBlock* サイズ（*ix, iy, iz*）は，以下の拘束条件を満たし，L2 キャッシュ総量に近い体積になるものを選択する．マルチコアによるワークシェアは，z 方向の並列化を行っているので[5]，*iz* のサイズは，コア数の倍数とする．

図 7 に，ブロックアレイ(*Block*)と *iBlock*　の関係を示す．

- 目的関数

$$min\ (\ |\ k*iBsz(\ ix,\ iy,\ iz)*Esz-Sc2*Nc\ |\ ) \qquad (1a)$$

ただし，$iBsz(ix,iy,iz)=ix*iy*iz$.

- 拘束条件

  1. *iz*　は．*Nc* の倍数
  2. *ix*　は．*nx* と等しい

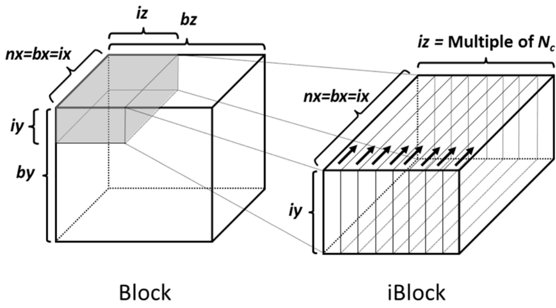

図 7　*iBlock* サイズ（*ix, iy, iz*）の選定

5.2 ステップ 2:　ブロックサイズ（*bx, by, bz, bt*）の選定

最適な空間・時間ブロックサイズ（*bx, by, bz, bt*）を見つける．以下の拘束条件を満たし，flash への IO 総量を最小とする組み合わせを選ぶ.

- 目的関数

$$\min(Total_A) = \min\left(\sum_{i}^{T_c} Ai\right)$$
$$= \min\left(\sum_{i}^{T_c}\sum_{X,Y,Z}^{nx,ny,nz} B(X,Y,Z,i)\right) \quad (2a)$$

Total_A は，*nt* 時間ステップにおける IO 転送量の総和で，*Tc* は，図 1 のバッファで示される問題データアレイ全体を走査する回数を示し，図 2 における外側の時間ステップループの繰り返し数（走査数）に対応する．以下の *Ai* は，i 回目の走査時の IO 総量を示す.

$$T_c = \ ceiling\left(\frac{nt}{bt}\right), \qquad T_f = \ floor\left(\frac{nt}{bt}\right) \qquad (2b)$$

$$Ai = \sum_{X,Y,Z}^{nx,ny,nz} B(X,Y,Z,i) \qquad (2c)$$

ただし $1 \le i \le T_c$,
$1 \le X \le ny$　かつ　X はbxの倍数,
$1 \le Y \le ny$　かつ　Y はbyの倍数,
$1 \le Z \le nz$　かつ　Z はbzの倍数.

B(X, Y, Z, i)　は，バッファアレイ中の(X,Y,Z)から始まるブロックの i 回目の走査時の IO 転送量である．*(2c)* 式の総和は，図 2 の外側の空間ループX,Y,Zに対応する.

$$\sum_{i}^{T_c} Ai =$$

$$\sum_{X,Y,Z}^{nx,ny,nz}\left(B(X,Y,Z,1) * T_f + B(X,Y,Z,T_c) * \left(T_c - T_f\right)\right) \quad (2d)$$

*(2d)*式の第一項は，初回走査時の *bt* ステップ更新における 1 ブロックの IO 量を走査回数分T_f.で乗じた値で，第二項は，*nt / bt* が割り切れなかった場合に加えられる最終走査時の IO 量を表している．走査回数と時間ステップの関係は以下の図 8 に示す.

$$bt_i = \ bt, \qquad \text{ただし} \quad 1 \le i \le T_f$$
$$bt_i = \ nt \ \text{modulo} \ bt, \quad \text{ただし} \ i = T_c, \quad T_c \ \neq \ T_f.$$

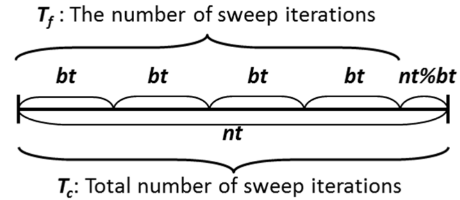

図 8　*Tc*　と　T_f. の関係

- 拘束条件

1. *bz* は，*iz* の倍数
2. *by* は，*iy* の倍数
3. *bx* は，*nx* と等しい
4. *bt* は，空間ブロック各次元サイズの半分以下

$$2 \le bt \le nt, \ \ bt \le \frac{bx}{2}, \ bt \le \frac{by}{2}, \ and \ \ bt \le \frac{bz}{2}$$

bt = 2　は，時間ブロック化のための最小サイズ

5. ブロック容量は主メモリ容量 *Sm* を超えない

$$k * Bsz_max \ * Esz \ < Sm - Km$$

ただし,
Km:　OS 用にとっておくメモリ容量，*k*:定数
Bsz_max:　最大 ブロックサイズ

$$Bsz_max \quad =$$
$$Max_{X,Y,Z}\left(Max\left(Bsz_{w(X,Y,Z,1)}, Bsz_{r(X,Y,Z,1)}\right)\right)$$

拘束条件 1, 2 は，図 7 に示す *iBlock* サイズの倍数でになるような値を選ぶことで，マルチコアへ均一な計算ワークロードが配当されることを目的とする．拘束条件 3. は，ステップ 1 の拘束条件 2 と同様に，aio 手法を用いたアルゴリズムで，3 次元データの読み込みの際に，*x* 方向を分割せずに，一度の IO で行う際に必要となる条件である．また，この場合 4. 2 節の「前提」から，IO サイズ，IO のスタートアドレスともに flash のブロックサイズ *Sb* の倍数になる.

5.3 ブロック当たりの flash への入出力量：$B(X,Y,Z,i)$

5.3.1　典型的なブロック形状と flash への IO 転送量

ステンシル計算における近傍データの範囲，袖領域幅を *h* とすると，１ブロック計算当たりの flash への IO 量（リード，ライト）は，以下のようになる．図 3，図 4 には，*x* 次元をブロックで分割しないという条件の下に，二次元

配列で表した場合のリード・ライト要素数を示している．冗長計算有(アルゴリズム 1)，冗長計算なし(アルゴリズム 2)のステンシル計算時のリード・ライト要素数はそれぞれ以下のようになる．

● 冗長計算ありアルゴリズム

$$Bsz1_r = (bx + 2*h + 2*bt)*(by + 2*h + 2*bt) * (bz + 2*h + 2*bt)$$

$$Bsz1_w = (bx)*(by)*(bz)$$

● 冗長計算なしアルゴリズム

$$Bsz2_r = \ Bsz2_w = (\mathrm{bx} + 2*\mathrm{h} + \mathrm{bt})*(\mathrm{by} + 2*\mathrm{h} + \mathrm{bt})*(\mathrm{bz} + 2*\mathrm{h} + \mathrm{bt})$$

袖領域幅 h を 1 とした場合の典型的なリード・ライトの IO 量（バイト）は，それぞれ以下になる．

$$B1(X,Y,Z,1)typical = ((nx + 2\ )*\ (\ by + 2 + 2*bt\ )*\ (\ bz + 2 + 2*bt\ ) + (nx + 2\ )*\ (\ by + 2\ )*\ (\ bz + 2\ ))*\ Esz \quad (2f')$$

$$B2(X,Y,Z,1)typical = 4*(nx + 2)*(by + 2 + bt)*(bz + 2 + bt)*Esz \quad (2g')$$

5.3.2 データ領域分割手法と境界界条件の影響

flash にある問題データを主メモリサイズにあわせてブロック化してリード・ライトする場合に，ブロックアレイに必要な DRAM メモリサイズやその IO 量は，ブロックごとに異なる．したがって，複数のブロックのうち最大のブロックサイズ *Bsz_max* に合わせて DRAM 上にアレイを確保する必要がある．また，IO 量の総和計算では，各ブロックの体積を正確に積算することが必要となる． 5.3.1 節で述べた典型的な冗長計算なしアルゴリズムのブロックの形状を，図 9 の右端に示す．しかし，図 9 左側に示す分割例では，いずれも典型的なブロックサイズにはならず，ブロック体積も IO 量も小さくなることがわかる． 図 9 のように *bt* が小さい場合は，典型的サイズと大きな差がないように見えるが，*bt* が一定量を超えると，典型的ブロック形状なら主メモリに入らない場合でも，境界条件などを加味した形状であれば主メモリに格納できる場合もあり，主メモリを最も効率的に用いるブロックサイズを選ぶことができる．さらに図 10,11 は，同じ形状の問題データを同じ数のブロック数で分割する場合に，DRAM に確保しなければならないブロックアレイの体積 *Bsz_max* が異なる場合があることを示している．図 10 では，上の分割法に比べ，下の均等分割のほうが，主メモリにおけるブロックサイズを小さくすることができ．必要な主メモリサイズが小さくても処理できることを示す．一方，図 11 の例では，下の均等分割に比べ，上の不均一分割のほうが，*bt* の幅を考慮すると，ブロックアレイに必要な DRAM サイズを小さくすることができる．このように，問題データの境界条件，分割形状などによる影響も，*Blk-Tune* の CalcTrans 部では考慮しコストを計算している．

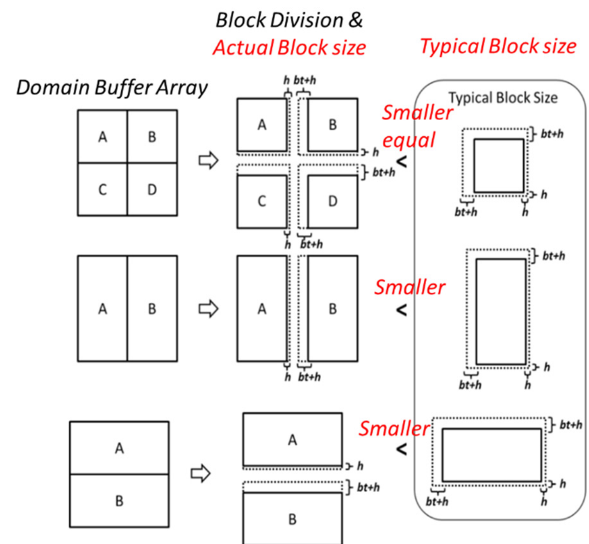

図 9 右側の典型的ブロックサイズに比べ，左の実際のブロックサイズは小さい場合がある

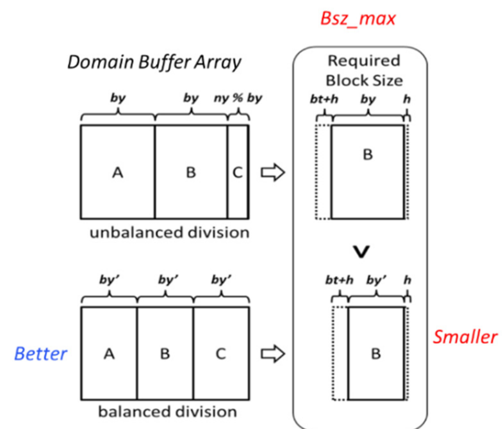

図 10 均等分割（下）が不均一分割（上）よりも最大ブロックサイズ（右側）を小さくできる例

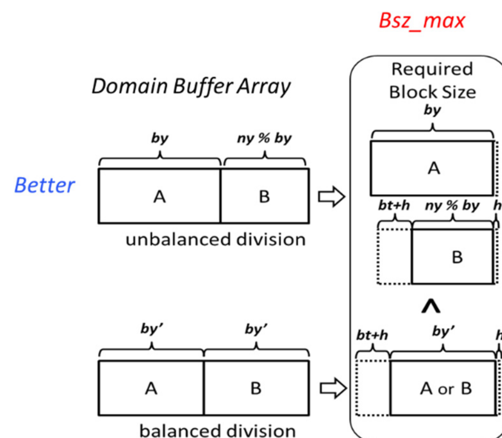

図 11 不均一分割（上）が均等分割（下）よりも最大ブロックサイズ（右側）を小さくできる例

5.4 ステップ 3：最終ブロックサイズの選定

flash への IO 転送量が同じブロックパラメタセットの候補が複数ある場合，DRAM における最大ブロックサイズ *Bsz_max* のが小さいほうを候補として選択する．あるいは，手動による選択も可能である．

6. ブロックサイズ選定システムの効果

6.1 計算環境とステンシルアルゴリズム

ブロックサイズ選定システムの効果を，以下の表 1 に示す 3 種の計算サーバにおいて評価した．

用いたのは冗長計算なしステンシルアルゴリズムで，flash への IO をなるべく少なくなるように工夫した 2 つの実装（aio3y と aio5y）である．aio3y は，y 方向に隣あうブロックアレイの処理で，前のブロック処理の終了時に flash へライトする部分と，次のブロック処理の開始時に flash からのリードする部分で共通なデータ部分を，DRAM 内（ブロック内）でコピーするようにし，flash の IO 量を低減させた実装である．aio5y は，aio3y に加え，図 3 におけるデスティネーションバッファへのリード・ライトを減らすための工夫を加えているが, 処理は複雑になっている．いずれも，ステンシル計算カーネル部分のコードは別ファイルになっており，7 点，19 点，27 点ステンシルをコンパイル時に指定できる． *Blk-Tune* で最適値の判定に用いる flash IO 量の比較において，ステンシル計算カーネルの種別や，aio3y と aio5y の違いは，同一実装で比較する限り影響を与えないため，冗長計算なしの CalcTans を用いて最適値を選んでいる．また，次節以降の実行時間測定には 7 点ステンシル計算を用いている．

表 1 評価に用いた計算サーバ

server	L1 cache (KiB)	L2 cache (KiB)	L3 cache (MiB)	Phys Mem (GiB)	Flash Mem (TiB)	CPU Xeon E5, (GiHz)	Total cores	socket	cores/ socket
crest0	32	256	20	32	1.2	2650, (2)	8	1	8
crest4	32	256	20	64	0.785	2687W, (3.1)	16	2	8
crest6	32	256	25	128	1	2687W v3,(3.1)	20	2	10

6.2 計算環境と問題に応じた最適値の即時選定

Blk-Tune は，計算プラットフォーム環境と問題パラメタ（データ空間サイズ *nx, ny, nz* や時間ステップ数 *nt*）に応じて，最適なブロックサイズを選定する．表 2 は，128GiB 問題（2048x2048x2048, 500 ステップ）と 256GiB 問題（2048x2048x4096, 1000 ステップ）に対し，表 1 に示す 3 つのサーバで処理する場合の *Blk-Tune* が出力した最適ブロックサイズである．flash と主メモリ向けの空間，時間ブロックサイズ（*bx=nx，by, bz, bt*）と，DRAM とキャッシュ向けの *iBlock* の空間サイズ（*ix=nx, iy, iz*）を示す．

計算サーバは，主メモリサイズ，キャッシュサイズ，ソケット数，コア数が異なるので，同じ問題であっても，最適ブロッキングサイズが異なることがわかる．*by* と *bz* を比較すると，いずれも，*bz* に比べ *by* が小さくなっているが，これは 6.1 で述べたように，隣合う y 方向のブロック計算に関し，flash の IO を一部省略して，DRAM 内でコピーできる部分があるためである．すなわち，y 方向の分割数が増加しても(by が小さくても)，z 方向の分割にくらべ，flash への IO 量が増えないという *Blk-Tune* 内の正確なコスト計算の結果が反映されている．

最適ブロックサイズは，人手では考えつかないような値も選ばれており，単に，人手設定の手間や時間を減らせるというだけでなく，空間と時間の両方を加味して大域的に最小の IO 量のブロックサイズ組み合わせが選ばれるという点でも，非常に有効である．

表 2 *Blk-Tune* により選択されたブロックサイズ例

Problem size	nx	ny	nz	nt		
128GiB-problem	2048	2048	2048	500		
Server (DRAM, cores)	bx, ix	by	bz	bt	iy	iz
crest6 (128GiB, 20)	2048	1025	2048	500	1	160
crest4 (64GIB, 16)	2048	1025	1280	500	1	128
crest0 (32GiB, 8)	2048	513	1152	250	1	64

Problem size	nx	ny	nz	nt		
256GiB-problem	2048	2048	4096	1000		
Server (DRAM, cores)	bx, ix	by	bz	bt	iy	iz
crest6 (128GiB, 20)	2048	1025	2240	500	1	160
crest4 (64GIB, 16)	2048	683	1536	334	1	128
crest0 (32GiB, 8)	2048	513	832	250	1	64

6.3 *Blk-Tune* と手動設定の性能比較

ここでは，*Blk-Tune* による最適ブロックサイズと，人間による手動設定サイズの場合の実行時間を比較する．

ここで用いた「手動設定」とは，コンピュータを使わないでも計算できる程度のサイズ選択で，具体的には，（1）主メモリ容量に入る最大限の空間ブロックサイズで，*by, bz* は共に 2 の累乗の値とする（*bx* は問題サイズの *nx* とする）ただし，*by* と *bz* は等しいか，他方が一方の 2 倍までの値とする．主メモリ容量は通常 2 の累乗なので，これであれば，人間でも計算できる．（2）次に上記空間ブロックアレイの体積を差し引いて，残った主メモリサイズに入ると思われる *bt* を問題の時間ステップ *nt* 以下の値で選ぶ．ただし, *bt* を含む正確なブロックアレイの計算はできないので，余裕をもって *nt* の約数などを選ぶ．

図 12 と図 13 はそれぞれ，2 つの計算サーバ（主メモリ 64GiB の crest4 と主メモリ 128GiB の crest6）における手動選択と *Blk-Tune*（自動選択）のブロックパラメタを利用したときのステンシル計算の実行時間を示している．各図の上部の 2 つのグラフは， aio3y と aio5y 実装における手動と自動の比較である．下部のグラフは，aio3y よりも高速な aio5y における手動設定と自動（Blk-Tune）の場合の時間成分を示す．問題サイズ（ny, nz, nt） と自動，手動で選ばれた空間ブロックサイズ(*by* x *bz*)，時間ブロックサイズ *bt* も横軸に示されている．

図 12 の crest4 の場合は，手動時実行時間は，自動実行時間の，5.6～7.7%増（aio5y），7.3%～13.5%増（aio3y）である．

図 13 の crest6 の場合は，13.4～20.6%増（aio5y），12.9%～41.9%増（aio3y）で，crest4 に比べ，自動と手動の時間差が大きい．いずれも，自動設定の効果は，aio5y より aio3y のほうが大きいが，絶対性能は，aio5y のほうが高い．

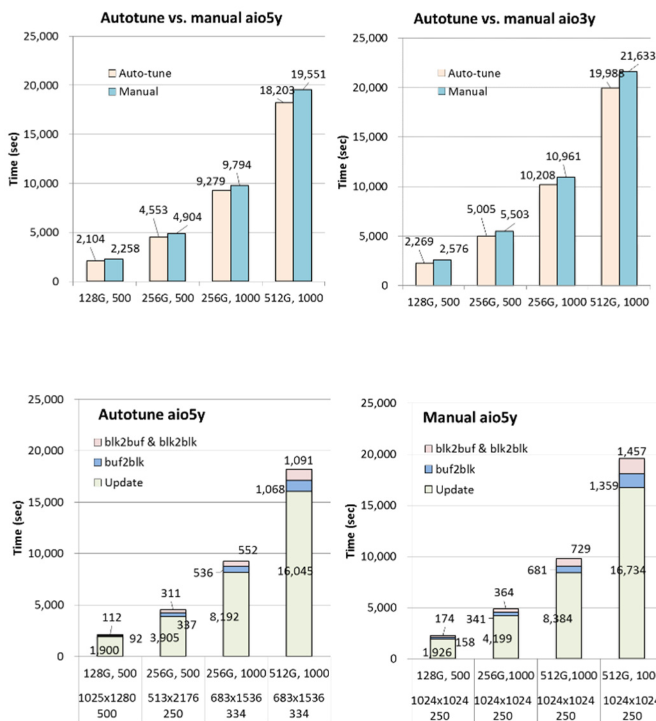

図 12 様々なサイズの問題に対する手動選択と Blk－Tune による選択の実行時間比較（crest4，主メモリ 64GiB）

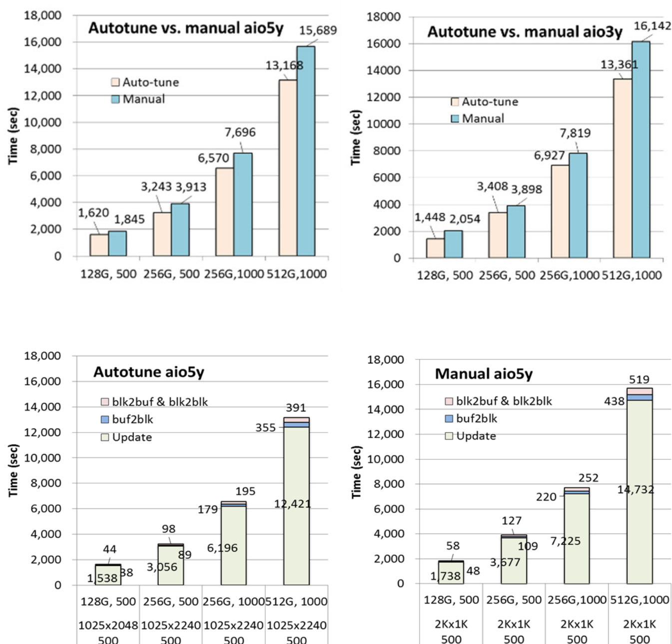

図 13　様々な問題サイズに対する手動選択と Blk-Tune による選択の実行時間比較（crest6，主メモリ 128GiB）

図 12,13 の下のグラフに示す時間成分をみると，いずれのサーバでも，計算時間（update）が主要であるものの，flash ライト（blk2buf，ただし blk2blk の値は非常に小さい）と flash リード（buf2blk）の値が，自動設定では手動の場合の 58%～78%に減少している．

6.4 *Blk-Tune* における最適ブロックサイズ選定時間

Blk-Tune は，５節で述べた拘束条件のもと，コストを最小にすることを目的関数とし，大域的最小値探索を行っている．本報告で用いた Blk-Tune のバージョンでは，探索手法にほとんど工夫をしていないため，図 14 に示すように，問題サイズが大きくなるのに応じて，探索時間が長くなる．*Blk-Tune* の高速化のために，様々な改良探索手法も適用可能であるが，現在の単純なアルゴリズムであっても，後段のステンシル計算時間に比較すると，処理時間は 0.1%程度（128GiB~512GiB 問題において）と無視できる程度なので，高速化の効果は見えにくい．

ただし，オフライン利用として，後段のステンシル計算と切り離して，単体で，分析に場合には，高速化の意義がある．

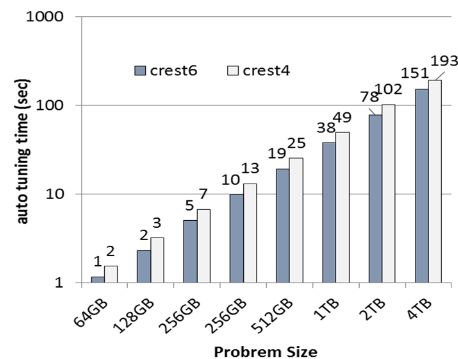

図 14 様々な問題サイズに対する Blk-Tune の最適ブロックサイズ選定時間（crest4, crest6）

7. 終わりに

flash を用いた Out-of-core ステンシル計算のための最適ブロッキングパラメタ自動選定システムとその選定手法を提案し，人手を介さずに最適ブロッキングサイズを実行時に自動選定するシステムの有用性を示した．

本報告では，冗長計算なしステンシル計算を中心に述べたが，冗長計算ありステンシル計算のための最適ブロッキング自動選定システムもすでに構築し，利用している．*Blk-Tune* システムにおけるコスト計算部の CalcTrans として，それぞれのアルゴリズムに合わせたものを複数用意しておき，選ぶことにより利用が可能となる．冗長計算ありのアルゴリズムでは，利用する flash のアクセスバンド幅と CPU 性能の指標を事前に用意する必要があるが，現在用いているテストカーネルによる計測をステンシル計算実行時に組

み込むことも可能であると考えている．

また，ここで紹介した 1 ノード向けの垂直方向のメモリ階層を対象としたテンポラルブロッキングアルゴリズムを，さらにマルチノード向けに水平方向に拡張し，ローカル SSD を装備したクラスタにおいて，クラスタノードの総メモリ容量を超えるサイズのステンシル計算を行う実装も稼働しており，ここでもマルチグリッド間の最適ブロックサイズの自動選定を行うアルゴリズムを構築している．今後，これらを組み合わせたシステムを構築したいと考えている．

参考文献

[1] Portable Hardware Locality (hwloc). https://www.open-mpi.org/projects/hwloc/
[2] H. Midorikawa, H.Tan, T.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,
[3] 緑川，丹: "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",情処学会，HPC 研究会, Vol.2014-HPC-145, No.22, pp.1-9, (2014.7)
[4] H. Midorikawa, H.Tan "Locality-Aware Stencil Computations using Flash SSDs as Main Memory Extension",IEEE/ACM CCGrid2015, pp.1163-1168, 2015-5
[5] 緑川，丹："フラッシュ SSD を用いた out-of-core ステンシル計算の性能向上手法とその効果"，電子情報通信学会、コンピュータシステム研究会 信学技報 CPSY2015-41, pp.241-246, (2015, 8/6)
[6] H. Midorikawa: "Blk-Tune: Blocking Parameter Auto-Tuning to Minimize Input-Output Traffic for Flash-based Out-of-Core Stencil Computations", The 11th International Workshop on Automatic Performance Tuning (iWAPT2016) , Proc. of IPDPSW2016, pp.1516-1526 (10.1109/IPDPSW.2016.48), (2016.5)
[7] Improve Linux swap for High speed Flash Storage http://events.linuxfoundation.org/sites/events/files/lcjpcojp13_shaohua.pdf.
[8] OpenNVM, FusionIO, http://opennvm.github.io.
[9] M Bjørling, J Axboe, D Nellans, P Bonnet,"Linux block IO: introducing multi-queue SSD access on multi-core systems”, Proc.of the 6th International Systems and Storage Conference (SYSTOR '13)
[10] “Linux Block IO: Introducing Multiqueue SSD Access on Multicore Systems”, M. Bjorling et.al, 2013, http://bjorling.me/blkmq-slides.pdf

ⓒ2016 Information Processing Society of Japan