Formura DSL による temporal blocking の PEZY-SC2 での実装と性能評価

Implementation and Performance of temporal blocking on PEZY-SC2 with Formura DSL

○ 田中 英行, 株式会社 Exascaler, 東京都千代田区神田小川町 2-1, E-mail: tanaka@pezy.co.jp
石原 陽平, 京都大学/理研 R-CCS, 京都府京都市左京区吉田本町, E-mail:youhei.ishihara@yukawa.kyoto-u.ac.jp
坂本 亮, 株式会社 PEZY Computing, 東京都千代田区神田小川町 1-11, E-mail: sakamoto@pezy.co.jp
中村 孝史, 株式会社 PEZY Computing, 東京都千代田区神田小川町 1-11, E-mail: nakamura@pezy.co.jp
木村 耕行, 株式会社 PEZY Computing, 東京都千代田区神田小川町 1-11, E-mail: yasuyuki@pezy.co.jp
似鳥 啓吾, 理研 R-CCS, 兵庫県神戸市中央区港島南町 7-1-26, E-mail:keigo@riken.jp
坪内 美幸, 理研 R-CCS, 兵庫県神戸市中央区港島南町 7-1-26, E-mail:miyuki.tsubouchi@riken.jp
牧野 淳一郎, 神戸大学/理研 R-CCS, 兵庫県神戸市灘区六甲台町 1-1, E-mail:jmakino@people.kobe-u.ac.jp
Hideyuki Tanaka, ExaScaler Inc., 2-1 Ogawa-machi, Chiyoda-ku, Tokyo, 101-0052, Japan
Youhei Ishihara, Kyoto University/RIKEN R-CCS, Kitashirakawa Oiwakecho, Sakyo-ku, Kyoto, 606-8502, Japan
Ryo Sakamoto, PEZY Computing K. K., 1-11 Ogawa-machi, Chiyoda-ku, Tokyo, 101-0052, Japan
Takashi Nakamura, PEZY Computing K. K., 1-11 Ogawa-machi, Chiyoda-ku, Tokyo, 101-0052, Japan
Yasuyuki Kimura, PEZY Computing K. K., 1-11 Ogawa-machi, Chiyoda-ku, Tokyo, 101-0052, Japan
Keigo Nitadori, RIKEN R-CCS, 7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, 650-0047, Japan
Miyuki Tsubouchi, RIKEN R-CCS, 7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, 650-0047, Japan
Jun Makino, Kobe University/RIKEN R-CCS, 1-1, Rokkodai-cho, Nada-ku, Kobe, 657-8501, Japan

We present the basic idea, implementation and achieved performance of our DSL for stencil computation, Formura, on systems based on PEZY-SC2 manycore processor. Formura generates, from high-level description of the differential equation and simple description of finite difference stencil, the simulation code with MPI parallelization, overlapped communication and calculation, temporal blocking and parallelization for many-core processors. Achieved performance is 4.78PFlops, or 21.5% of the theoretical peak performance for an explicit scheme for compressive CFD. For a slightly modified implementation of the same scheme, efficiency was slightly lower (17.5%) but actual calculation time per one timestep was faster by 25%. Temporal blocking improved the performance by up to 70%. We have demonstrated that automatic generation of the code with temporal blocking is a quite effective way to make use of processors with low memory bandwidth for large-scale CFD calculations.

この論文では我々の開発した構造格子計算 DSL「Formura」の基本的な構造と導入，またこれを用いてメニーコア計算機である PEZY-SC2 上で実際にステンシル計算を行った際の性能，最適化の詳細について述べる．

Formura は，微分方程式の高レベルの記述や有限差分スキームの記述ら，高度なテンポラルブロッキングにより通信・計算をオーバーラップし，MPI 並列化された，メニーコアプロセッサ用の完全なシミュレーションコードを生成する．

我々は空間 4 次時間 3 次精度の圧縮性 CFD の陽的スキームの計算で，理論ピーク値の 21.5%である 4.87PFlops の性能を達成した．同じスキームで僅細な変更を加えたバージョンでは，効率は 17.5%と僅かに下がったが，実際の 1 タイムステップごとの計算時間は 25%減少した．また，テンポラルブロッキングは性能を 70%まで改善した．PEZY-SC2 における B/F 値は低く，0.02 程度であるが，大きなメモリバンド幅を持つ「京」コンピュータのようなスーパーコンピュータに最適化した CFD コードと比較しても同程度の性能を達成した．

これらより，テンポラルブロッキングを用いたコード自動生成は，低メモリバンド幅での超大規模マシンにおける大規模 CFD 計算に，非常に効果的であると確認された．

1. イントロダクション

ステンシル計算は多くの HPC アプリケーション，例えば電磁波，音波，地震波等の計算に用いられる FDTD 法等の数値流体力学計算において頻用される数値計算法であり，重要な位置を占めている．

数値流体力学計算におけるいくつかの重要な分野で非圧縮性近似と陰解法が使われている．陰解法は情報伝搬の速度によって決まるタイムステップについての CFL リミットに左右されないためである．例えば，2016 年 Gordon Bell 賞受賞の Yang. 等 (1) の非静力学地球大気変動の計算においても陰解法が用いられており，Sunway Taihulight 上での実行効率 7%を達成している．浮動小数点演算数はタイムステップ・グリッドポイント毎に 9.3×10^4 で，その他の陽解法より 20-40 倍大きい（表 1）．演算数が大きい理由は，陰解法では各ステップにおける圧力を計算するポアソン方程式を解くために反復が必要となるからである．彼等は非常に洗練された DD-MF 法を開発したが，それでも，格子数が 8 倍に増えると，反復数はだいたい 2 倍になる．

最近では，音速抑制法（RSS 法）(2), (3) において，陽解法を用い物理的な CFL リミットを超えてタイムステップのサイズを増加させることが可能になってきている．音速抑制法の基本的な考え方は，非圧縮性近似を逆方向に適用することである．亜音速流は違なるマッハ数を持つ流れで近似出来ることが知られている．例えば同じ非圧縮性流れを $M = 0.01$，$M = 0.1$，$M = 0.5$ で近似できる．従って，$M = 0.01$ の流れを $M = 0.5$ の流れに近似でき，よって CFL 条件は 50 倍緩和される．

1 格子点・1 タイムステップごとの浮動小数点演算の数は，陽解法の方が陰解法に比べ何桁も小さい．更に言う

と，音速抑制法を使用することは，以下のような理由から，現代の HPC システムにおいて高い優位性があるといえる．

- 通信バンド幅・レイテンシの必要要件が陽解法の方が陰解法よりも少ない．陽解法は隣のノードとの通信を必要とするのみであるため．陰解法においては，マルチグリッド法を使った場合にはタイムステップごとに遠くのノードとの通信を必要とし，大きな通信量が必要となるため，結果的に大きな通信量のオーバーヘッドが発生する．
- 必要なメモリバンド幅も陰解法より陽解法の方が少ない．適切なキャッシュブロッキングを適用することで，陽解法における物理変数のリード＆ライトは 1 タイムステップあたり 1 度だけにできるが，陰解法においては 1 タイムステップ内で各反復ごとにメインメモリをリード＆ライトすることが必要となる．
- 音速抑制法を使う場合，タイムステップの制限における陰解法と陽解法の大きな違いは無い．

音速抑制法のようなアイディアはほかのさまざまな課題ーマントル対流 (4) や輻射輸送等 (5) ーのような，今まで陰解法が用いられてきた問題にも用いることが出来る．よって我々は，テンポラルブロッキングと陽解法を組み合わせて高い効率を実現することは，今後の大規模 HPC 開発において非常に重要かつ効果の大きなものであると考える．テンポラルブロッキングの基本的なアイディアは複数のタイムステップをローカルの小さなグリッドに適用することで，メインメモリへのアクセスを減少させられるというものである．

理論的には，テンポラルブロッキング無しの陽解法でのパフォーマンス上限 $F_{\max}$ は式（1）で与えられる．

$$F_{\max} = \min\left(F, \frac{C_e G}{2H_e}\right), \tag{1}$$

ここで C_e，H_e は格子点（グリッドポイント）あたりの計算量と通信量で，F は 1 秒あたりの計算スループット，G はメインメモリバンド幅で 1 秒あたりの語数を単位としたものである．テンポラルブロッキングを使うと，この制限は式（2）の通り緩和される (6)．

$$F_{\max,\mathrm{TB}} = \min\left[F, \frac{C_e G}{2H_e}\left(\frac{1}{N_t} + \frac{2dN_s}{N_T}\right)^{-1}\right], \tag{2}$$

ここで，N_t はテンポラルブロッキングで結合されたタイムステップ数，N_T はラストレベルキャッシュに適応した一次元方向のグリッド数，N_s はステンシルの幅，d は空間次元数である．

テンポラルブロッキングありの陽解法の効率を決定する重要なパラメータは $2dN_s/N_T$ である．テンポラルブロッキングの効果は N_s が小さく，N_T が大きいほど大きい．ラストレベルキャッシュ（LLC）の物理サイズが N_T を決める．テンポラルブロッキングを有効にするために N_s を出来る限り小さくするべきである．1 次元の簡単な 3 点ステンシルの場合は $N_s = 1$，多くの FDTD アプリケーションで使われているスタガードグリッド 4 点の場合には $N_s = 3$ となる．

また時間積分の効果も考慮しなければならない．CFD アプリケーションで広く使われているルンゲークッタ法では，ステンシルの実効的なサイズは $N_s s$ になる．ここで，s はルンゲークッタ法の段数である．それゆえ，$N_s = 3$ のステンシルで，$s = 4$ のルンゲークッタ法を使った場合，実効的に幅 12 のステンシルを持つことになる．これは数メガバイトのサイズを持つキャッシュでは，テンポラルブロッキングの効果が完全に失われることを意味すると言える．

よって，テンポラルブロッキングによるメモリアクセスを減らすためには小さな N_s と s のスキームを使うことが非常に重要である．しかし，こういった数値スキームの必要性がそれほど認識されていないため，このような方向の研究はまだそれほどされていない．

これらより，単純な 7 点ステンシルの拡散方程式では，テンポラルブロッキングは明らかに有用であることが示されているが，実際のアプリケーションではその有用性はいまだ示されていない．

表 1 では，地球科学や宇宙物理学での亜音速流の大規模 CFD シミュレーションについての最新の陰的・陽的ソルバをまとめた．堀田等は 4 点スタガードグリッドの空間差分と 4 段ルンゲークッタ法を使った．それゆえ，テンポラルブロッキングを用いることによって得られるポテンシャル（潜在的効果）は小さい．次のセクションでは，我々が開発した新しい有限差分スキームについて簡単に述べる．

京コンピュータ上の 2 つの計算 (7)(8) におけるタイムステップごとの浮動小数点演算の数は 2000 程度で，物理変数の数は 5 である．式（1）より，理論ピークスピードで正規化されたメモリバンド幅によって決定される演算速度によって定義された効率の上限を式（3）として表現することが出来る．

$$\eta_{\max} = \frac{B}{8F}\frac{C_e}{2H_e}, \tag{3}$$

ここで B はメモリバンド幅（1 秒あたりバイト）で，更に，1 語は 8 バイトと想定している．$G = B/8$ と言い換えられる．この記法では，式（3）の B/F は，広く使われている B/F と同じ形である．京コンピュータの一つのプロセッサの理論ピーク性能は 128GFlops で，実効的なメモリバンド幅は 50GB/s 程度である．これより，$B/F \sim 0.4$，$C_e \sim 2000$，$H_e = 5$ と設定する．よって $\eta_{\max} \sim 10$ である．これらの計算の効率は実際にはメインメモリバンド幅ではない何かによって制限されていることがわかる．

この所見は少々驚くべき結果である．従来の常識では，現代のキャッシュベースアーキテクチャ上での陽解法の効率はメインメモリバンド幅で制限されるからである．京コンピュータ上で十分にチューニングされたコード群の効率が理論限界よりも非常に低いのにはいくつかの理由がある．一つは単純にこれらのコードは式（1）の導出での仮定を満たさない方法で書かれていることである．我々は個々の変数は 1 タイムステップあたり 1 度のみのリード＆ライトを仮定しており，更に，すべての中間結果はキャッシュにストアされ，メインメモリには流れないとしている．4 段ルンゲークッタ法を使用する場合には，通常の実装ではステージ毎に 1 回のリードとライトを要求する．その為，1 変数あたり 4 回のリードと 4 回のライト動作が要求される．理論的には，これらの追加のリード＆ライトはキャッシュブロッキングによって防ぐことができうる．しかし，それは実効的にテンポラルブロッキングの実装を要求する．

他の困難は，京コンピュータのプロセッサでは，理論通りの効率を達成するためには，最内側ループで処理するデータのサイズが比較的大きく（256 以上）なくてはならないことである．一方，テンポラルブロッキングの効率的な実装は LLC でリードされるブロックの 1 次元方向のサイズが 16 や 32 のように非常に小さいことを要求する．それゆえ，式（1）で定義される効率を達成することは最近のコンピュータでは簡単ではない．

本論文では圧縮性 CFD 計算で高い実行効率を出すためのテンポラルブロッキングを用いての新しいアプローチ一高レベルで記述された流体方程式や空間差分スキームから実際のプログラムを自動生成しテンポラルブロッキングを用いて高い実行効率を達成する新しい CFD スキームについて述べる．

すでに記したように，広く使われている 4 点スタガードグリッドと 4 段ルンゲークッタ法の組み合わせではテンポラルブロッキングによる性能向上はほとんど不可能である．テンポラルブロッキングによる意味のあるスピードアップを達成するには時間方向の段数の小さい高次ス

application	Yashiro et. al.	Yang et. al.	Hotta et.al.	This work
reference	(7)	(1)	(8)	–
method	explicit	implicit	explicit	explicit
# grids	5.5×10^{12}	1.29×10^{11}	2.61×10^{11}	3.97×10^{12}
WCT per Timestep (seconds)	12.4	9	0.254	3.37
# grids processed per second	4.6×10^{11}	8.6×10^{10}	1.03×10^{12}	1.18×10^{12}
flop per mesh per timestep	1888	93023	2505	4246
Computer	K computer	TaihuLight	K computer	Gyoukou
Efficiency(%)	10.2	7	24.3	21.5

Tab. 1: Comparison of state-of-the-arts high-resolution mesh schemes

キームと空間方向に狭いステンシルサイズが必要である．我々はコロケーショングリッドとエルミート積分をベースとした新しいスキーム SL4TH3 を開発した．SL4TH3 は，タイムステップ毎に空間微分を計算することで空間 4 次・時間 3 次を実現した．広く使われているスキームでは $N_s = 12$ であるが，SL4TH3 では $N_s = 2$ である．

SL4TH3 スキームにおける一つの実際上の問題は，形式的な微分方程式をもとにしているために，多くの項が現れ非常に複雑なことである．このことは必ずしも計算コストが高いことを指してはいない．なぜならこのスキームでは，タイムステップあたり 1 回の微分計算を要求するのに対し，4 段ルンゲークッタ法では 4 回の微分計算が必要となるからである．しかしながら，プログラムは非常に長く複雑になり，人間がコードを書いたり，デバックをすることは実際的ではない．このため，数式処理プログラムによるステンシルの自動生成，また Formura DSL でのテンポラルブロッキングによる実行コードの自動生成・ステンシルからの並列化をおこなった．

我々は Formura により生成されたコードの性能を 2 つの PEZY-SC2 プロセッサを用いたシステム，Gyoukou と Shobu System-B で測定した．このシステム選定には 2 つの理由があり，一つはワットあたりの実行効率が高く，それゆえもっとも効率的な HPC システムの一つと考えられること，もう一つの理由は，Top500 や Green500 にランクインするシステムの中でおそらく最もメモリバンド幅（B/F 比）が低いことである．これらより，もし PEZY-SC2 をベースとしたシステムで，我々のアプローチがうまく行く事を示すことができれば，将来的に有効な手段と考えられる．なぜなら，未来の HPC システムの多くはおそらく現在の HPC システムに比べても低い B/F を持つからである．

測定された実行効率は 21.5%であり，この数値は京コンピュータでの高度に最適化された CFD 計算と同程度である．公表されている京コンピュータの B/F 値は 0.5 で，PEZY-SC2 の 20 倍である．Formura を用い，熟慮された有限差分スキームから自動生成され，テンポラルブロッキングを用いたシミュレーションプログラムは，低メモリバンド幅の計算機上で非常に高い実行効率を達成することが示されたと言える．

本論文では，以下，セクション 2 において新しいアプローチについて説明し，セクション 3 では性能測定結果について述べ，セクション 4 ではこれまで述べた我々の研究成果についてまとめる．

2. 新しいアプローチ

2.1 SL4TH3 スキーム

先述の通り，テンポラルブロッキングを用いて高い実行効率を達成するためには，小さなサイズのステンシルと少ない段数で高精度を実現できるスキームを使う必要がある．空間高次精度は，高次の空間差分オペレータを使うことによって達成することが出来る．しかしながら，ステンシルサイズを大きくすることなく，時間高次を達成する必要がある．つまり，タイムステップあたり，最小の数の微分計算を要求する高次の時間積分スキームが必要である．

タイムステップあたり 1 回の微分によって時間高次を達成する明らかな方法は予測子・修正子法を使うことである．予測子・修正子法では，最初に新しい時刻での解を，過去の幾つかの時間ステップでの解から多項式補間により予測し，新しい時刻での微分を計算し，通常は計算された微分を用いて，新しい解の補正を適用する．これを PEC モードと呼ぶ．時に安定性のために，微分の計算と補正を複数回行う．これを $\mathrm{P(EC)}^n$ mode という．

残念ながら，n-step（段）の予測子・修正子法では，$n-1$ 個の過去の時間ステップの微分を保持する必要があるため，その結果，メモリ使用量を増加させ，テンポラルブロッキングの効率を下げることとなる．

1 タイムステップあたり 1 回の微分計算によって高次精度の時間積分を達成するほかの方法では，空間の高階の微分から高階の時間微分を直接計算する．我々は元の偏微分方程式の時間微分を行い，新しい方程式の右辺の時間微分をもとの方程式を使って空間微分に置き換える．このアプローチは IDO スキーム [11] で用いられている．しかしながら，高階の時間微分の計算コストは特に 3 次元の場合には非常に大きくなる．それゆえ，我々は予測子・修正子法とテイラー展開のハイブリッド法であるエルミート法（スキーム）を採用した [12]．

エルミート法では，我々は空間微分から元の方程式の右辺の 1 階の時間微分を計算する．そして，2 ステップの予測子・修正子法を使って，時間高次を達成する．通常の予測子・修正子法の場合，p-step 法の精度は p であるが，エルミート法の p 段予測子・修正子法の精度は $2p$ である．それゆえこの方法は，粒子系の高精度の時間積分法として広く使われている．4 次精度を達成するためにはエルミート法は 2 段でなければならないが，予測子の多項式精度は低くても良い．この場合，4 次のエルミート法は実効的に 1 段の 4 次精度スキームになる [13]．そのためテンポラルブロッキングを使うのには理想的である．次に連続の方程式の場合の公開の時間微分について説明する．

$$\rho_t = -(u_x + v_y + w_z)\rho - u\rho_x - v\rho_y - w\rho_z, \quad (4)$$

ここで，ρ, u, v, w は，密度，x，y，z 方向の速度を表している．下添え字の α は α での偏微分で，$\partial/\partial\alpha$ で表される．両辺を t で微分することによって，式（5）を得る．

$$\begin{aligned}\rho_{tt} = &-(u_x + v_y + w_z)\rho_t - (u_{tx} + v_{ty} + w_{tz})\rho \\ &- u\rho_{tx} - v\rho_{ty} - w\rho_{tz} - \rho_x u_t - \rho_y v_t - \rho_z w_t. \quad (5)\end{aligned}$$

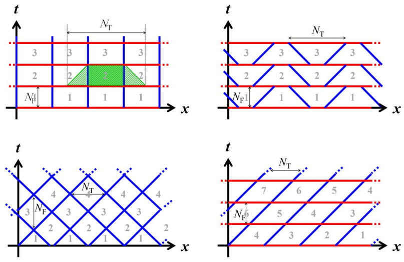

Fig. 1: Various methods of temporal blocking. The gray numbers indicate the order of regions that can be processed in parallel.

これは密度の 2 階の時間微分を与えるもので，まだ ρ_{tx} のような項があり，式（6）のように書ける．

$$\rho_{tx} = -(u_x + v_y + w_z)\rho_x - (u_{xx} + v_{xy} + w_{xz})\rho - u\rho_{xx} - v\rho_{xy} - w\rho_{xz} - \rho_x u_x - \rho_y v_x - \rho_z w_x. \quad (6)$$

これで，ρ_{tt} を高階の空間微分で表すことができた．これまで見てきたように，導出はシンプルで単純であるにも関わらず，結果の方程式は非常に多くの項を含んでいるため，元の方程式から時間の 2 階微分を導出するシンプルなプログラムを実装し，そのプログラム内でステンシル計算のインナーカーネルを生成させた．空間微分については 125 点（5×5×5）の有限差分式を使う．一次元方向の微分に関しては，我々は 4 次の多項式補間から係数を導いた．最終形を導出するために，交差偏導関数については，我々は単純に，有限差分演算子を順番に 3 つの方向に適用した．我々は得られたスキームを多くのテスト問題を用い試験し，非常に良い結果を得た．この論文では，このスキームを Spatial Lagrange interpolation with forth order and temporal Hermite interpolation with third order の略語として SL4TH3 と呼ぶ．

2.2 PEZY-SC2 と Gyoukou

このセクションでは，13312 の PEZY-SC2 プロセッサからなる暁光（Gyoukou）に焦点を当て記述する．

1 つの PEZY-SC2 プロセッサチップは物理的には 1984 の要素プロセッサ（PE）からなる．それぞれの PE は 1 サイクル当たり FP64 では 1 回，FP32 では 2 回の積和演算を行うことができる．この論文では正確性を保つため，FP64 計算のみを扱う．クロックスピードは 700MHz で，理論限界性能は 2.8TFlops である．それぞれの PEZY-SC2 プロセッサは 4 チャネルの DDR メモリを持ち，76.8GB/s のピークスループットである．以上より B/F は 0.027 である．LLC のトータルサイズは 40MB である．またキャッシュメモリについては，それぞれのプロセッサは異なるアドレス空間にローカルメモリ 20KB を持つ．PEZY-SC2 プロセッサは非コヒーレントで共有されたキャッシュメモリを 3 レベルで持つ．それぞれの PE は独自の L1D を持ち，16PE で L2D を共有し，すべての PE は LLC を共有する．それぞれの PE は 8 スレッドを同時に実行することが出来る．このうち少なくとも 4 スレッドを使えば，演算器をフルに稼働させることができる．従って，算術ユニットと L1D のレイテンシを隠すことは比較的容易である．暁光システムにおいては，8 つの PEZY-SC2 チップを PLX PCIe スイッチチップを通し，1 つの Xeon-D1571 プロセッサに接続されている．暁光システムの理論ピーク性能は 37PFlops であるが，我々が性能測定を行ったタイミングでは，最大で 8000 チップ，22.2PFlops のピーク性能であった．

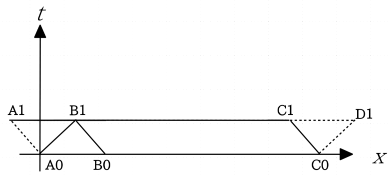

Fig. 2: Calculation on one processor. For simplicity, we show one-dimensional case. One processor initially has region B0-C0. In order to start calculation, it generates a dummy “wall” C0-D1, and calculation proceeds from right to left. During the calculation, the data of region A0B0 is transferred from the neighboring processor. At the end of one blocked calculation, the valid region is B1C1.

2.3 テンポラルブロッキングの実装

我々の以前の実装 (9)(10) においては，理論的には効率的な台形を基本とした複雑なテンポラルブロッキングを用いた．台形でのブロッキングの一つの問題としては，相互作用の間にループサイズが変化し，多数の PEZY-SC2 のプロセッサを効果的に使用することが難しいことである．よって，平行四辺形によるよりシンプルなテンポラルブロッキングを使用することにした（図 1 の右側）．

現在の我々の実装では，すべての PE は LLC のサイズによって制限される領域のサイズを出来る限り大きくするように一つの領域を計算する．1 タイムステップにおいて，1 プロセッサチップが複数の領域を順次引き受ける．それぞれの領域は (nx, ny, nz) のサイズを持ち，1 チッププロセスは $N_x \times N_y \times N_z$ の領域を計算する．平行四辺形ブロッキングの一つの問題は，計算が右から左にしかできず (表 1 の場合)，計算スタートが出来ないことである．よって，我々は 1 プロセッサの全体領域に長方形ブロッキングを使うことで，この問題を“解決”した．

図 2 に現在の実装を示す．1 つのプロセッサは最初領域 B0C0 を持ち，1 ブロック計算の最後に 1 プロセッサは領域 B1C1 を持つことになる．よって，A0B0 のデータは左側の隣接プロセスから移動されるべきである．計算をスタートするために，ダミー“壁”C0D1 を生成し，そこから計算をはじめて左に進む．計算中に通信も行う．従って，通信と計算は自然かつ最大限にオーバーラップしている．このスキームでは，三角形 C0C1D1 の計算は使用せず，A0B0B1 も冗長である．よって，不必要な計算はゼロではないが，$N_t/(N_x n_y)$ に比例するので実際には十分に小さいと言える．

2.4 空間差分

我々はすべての空間微分に 5 点中心差分公式を採用した．それらは

$$\begin{aligned}
q &= q^0 \\
q_x &= \frac{1}{12h}(q^{-2} - 8q^{-1} + 8q^{+1} - q^{+2}) + O(h^4) \\
q_{xx} &= \frac{1}{12h^2}(-q^{-2} + 16q^{-1} - 30q^0 \\
&+ 16q^{+1} - q^{+2}) + O(h^4) \\
q_{xxx} &= \frac{1}{2h^3}(-q^{-2} + 2q^{-1} - 2q^{+1} + q^{+2}) + O(h^2) \\
q_{xxxx} &= \frac{1}{h^4}(q^{-2} - 4q^{-1} + 6q^0 - 4q^{+1} \\
&+ q^{+2}) + O(h^2) \qquad (7)
\end{aligned}$$

で与えられる．q^i は $q(x + i\Delta x)$ を採用し，$q(x)$ はある位置 x での物理量を表す．多次元空間の場合，交差偏

導関数の有限差分近似が必要である．それらは式（7）を各次元に適用することで得られる．例えば $\partial^2/\partial x \partial y$ の有限差分近似は式（8）のように与えられる．

$$
\begin{aligned}
q_{xy} = \quad & \big[q^{-2,-2} - 8q^{-2,-1} + 8q^{-2,+1} - q^{-2,+2} \\
& -8(q^{-1,-2} - 8q^{-1,-1} + 8q^{-1,+1} - q^{-1,+2}) \\
& +8(q^{+1,-2} - 8q^{+1,-1} + 8q^{+1,+1} - q^{+1,+2}) \\
& -q^{+2,-2} - 8q^{+2,-1} + 8q^{+2,+1} - q^{+2,+2}\big] \\
& \frac{1}{144h^2}. \qquad (8)
\end{aligned}
$$

ここで交差偏導関数について得られた式は採用した有限差分演算子の順序に依存しないことに留意する．

計算に必要な高次の有限差分項を計算するには，幾つかの異なる方法がありえる．3 次元空間では，3 個の 1 回微分と，6 個の 2 回微分と，11 個の 3 回微分，15 個の 4 回微分が存在する．単純には，すべての計算を 125 点（$5 \times 5 \times 5$ の立方体）のデータから直接計算する方法がある．この方法の利点は，異なる格子点についての計算が独立しているため，容易に異なるスレッドや異なるプロセスに計算を割り当てることが出来ることである．更に，ひとつの微分項の計算に必要な中間変数以外の中間変数のためのストレージを必要としない．よって，この方法が PEZY-SC2 のようなメニーコアプロセッサで高い性能を達成することが期待できる．

一方，このアプローチにはいくつかの欠点がある．ひとつはこの方法で導かれた実際の計算コードは多くの冗長な演算を含むことである．例えば，q_x は各グリッドポイントで 1 回計算されるが，式（8）からわかるように，q_{xy} の計算は異なるつのグリッドポイントの q_x の計算を含んでいる．

他の問題は，L1D のヒットレートが PEZY-SC2 の場合には低いと思われることである．なぜならば，1 グリッドポイントにおける 1 タイムステップあたりの読み込みデータ量は L1D のサイズより非常に大きいからである．個々のグリッドポイントは 5 個の変数を持つので，空間微分の計算には $5 \times 125 = 625$ 個の変数もしくは 5.4KB のデータにアクセスする必要があるが，一方で，PEZY-SC2 の個々のプロセッサは 2KB の L1D しか持たず，この L1D は 4 もしくは 8 スレッドで共有されている．その為に L1D は 1 つのグリッドポイントのデータを保持するには小さ過ぎる．もしひとつのプロセッサが 625 グリッドポイントを保持出来れば，次の反復で 500 グリッドポイントのデータを再利用できる．もし隣接格子点を扱うなら，この目的を達成するために，L1D はグリッドデータや計算された空間微分，他の高階の時間微分計算のための中間項を保持するのに十分大きくなければならない．合計サイズは約 10KB 程度だろう．それゆえ，もし L1D のサイズが 10KB/スレッドより大きければ，L1D ヒットレートはもっと良くなるだろう．

PEZY-SC プロセッサのユニークな特徴は，個々のプロセッサが離れたアドレス空間でローカルメモリを持つことである．PEZY-SC2 の場合，そのサイズは 20KB なので，4 スレッド使うのであれば，個々のスレッドは 5KB のローカルメモリまで使える．残念ながら，それでも，グリッドデータを保持するのに十分な大きさとは言えない．

以下のようにすれば，冗長な計算を消去することは出来る．最初にすべてのグリッドポイントについて x 方向のすべての有限差分を計算しそれらをメモリにストアする．その後すべてのグリッドポイントについて，y 方向の差分を含む全ての項を先ほど計算した x 項を使って計算する．最後に z 座標の差分を含む全ての項を先ほど計算した xy 項を使って計算する．

この方法は演算数の観点から最適である．しかし，実際的ではない．なぜなら，この方法は高階の微分を保持するために莫大なメモリ量を要求するので，キャッシュフレンドリーではない．

現在のところで我々は 2 つのアプローチを実装している．最初のアプローチはすべての差分項を独立に計算するものである．2 番目の方法は，z 座標を含む項の計算にのみ冗長な計算を消去するものである．2 番目の方法では個々のグリッドポイントについて，x，y，xx，xy 項のような z の有限差分を含まない項を計算する．計算は z 方向に進んで行く．よって，z を含まない項は グリッドポイント k（k は z 軸方向のグリッド数）での z を含まない計算の後，我々はグリッドポイント $k-2$ の z 項を計算できる．これらの微分はキャッシュメモリにストアされず，ローカルメモリにストアされる．それゆえ，我々はグリッドデータへのアクセスを減らすことが出来るはずである．

```
r[t,x,y,z]_t = -u[t,x,y,z]*r[t,x,y,z]_x - v[t,x,y,z]*r[t,x,y,z]_y
          - w[t,x,y,z]*r[t,x,y,z]_z
          - r[t,x,y,z]*(u[t,x,y,z]_x + v[t,x,y,z]_y + w[t,x,y,z]_z)
u[t,x,y,z]_t = -u[t,x,y,z]*u[t,x,y,z]_x - v[t,x,y,z]*u[t,x,y,z]_y
           - w[t,x,y,z]*u[t,x,y,z]_z
           - p[t,x,y,z]_x/r[t,x,y,z] + c*vis1[t,x,y,z]/r[t,x,y,z]
v[t,x,y,z]_t = -u[t,x,y,z]*v[t,x,y,z]_x - v[t,x,y,z]*v[t,x,y,z]_y
          - w[t,x,y,z]*v[t,x,y,z]_z
          - p[t,x,y,z]_y/r[t,x,y,z] + c*vis2[t,x,y,z]/r[t,x,y,z]
w[t,x,y,z]_t = -u[t,x,y,z]*w[t,x,y,z]_x - v[t,x,y,z]*w[t,x,y,z]_y
          - w[t,x,y,z]*w[t,x,y,z]_z
          - p[t,x,y,z]_z/r[t,x,y,z] + c*vis3[t,x,y,z]/r[t,x,y,z]
p[t,x,y,z]_t = -u[t,x,y,z]*p[t,x,y,z]_x - v[t,x,y,z]*p[t,x,y,z]_y
                - w[t,x,y,z]*p[t,x,y,z]_z
           - gm*p[t,x,y,z]*(u[t,x,y,z]_x + v[t,x,y,z]_y + w[t,x,y,z]_z)
           - c2*(u[t,x,y,z]*vis1[t,x,y,z] + v[t,x,y,z]*vis2[t,x,y,z]
           + w[t,x,y,z]*vis3[t,x,y,z])
```

Fig. 3: The input differential equation.

この論文中では以後最初のアプローチを単純差分とし，2 つ目の方法を改良型差分と呼ぶこととする．単純差分の演算数は 2849（空間差分演算についてのみ）で，改良型差分の場合は 1415 である．これらふたつの方法は丸め誤差を除いて一致することに注意する．

2.5 コード生成

我々の現在の実装では，ステンシルカーネルのみが微分方程式から生成されており，他のすべてのコードはあらゆるステンシル計算に使うことができる．カーネルコードは有限差分スキームの記述（セクション 2.1）から FORMURA システム [9][10] によって生成される．有限差分スキーム自体は形式微分と空間微分のテンプレートによって元の流体スキームから生成される．グリッドポイントをアップデートするために生成されたカーネルは 400 行程度で，いくつかの行は空間微分のナイーブアプローチの場合，100 程度の項を含む．

3. 性能測定

性能を測定するために，我々は 1 ステップあたりの時間を測定した．ウォールクロックタイムのばらつきは非常に小さく，実行時間をウォールクロックタイマーによって精度よく測定することができた．演算数は相互作用数から計算された．我々は冗長な演算や，セクション 2.3 で述べた，使われない領域についての演算は除いた．

最初に，暁光システムで測定した単純アプローチでの並列性能とスケーラビリティを示す．次に，改善されたアプローチでのシングルノードでの性能を示す 2 番目のアプローチでの暁光システム上でのパフォーマンスを示さないのは，暁光システムが 2018 年 3 月 31 日に停止したため，改善されたアプローチの実行ができなかったからである．

3.1 暁光上での性能とスケーラビリティ

ナイーブアプローチによる空間微分の計算は 2849 浮動小数点演算を要求する．一方，運動方程式と，他の計算は 1415 浮動小数点演算を要求する．これらを足し上げると，グリッドポイントあたり 4264 浮動小数点演算が必要と分かる．我々は，浮動小数点演算の合計数の計算にこの演算数を使った．暁光システムの停止によりストロングスケーリングテストを終了させることが出来なかった．その為，この論文では，ウィークスケーリングテストの結果のみ報告する．ウィークスケーリングの測定では，我々は MPI プロセスあたり 792^3 グリッドポイント

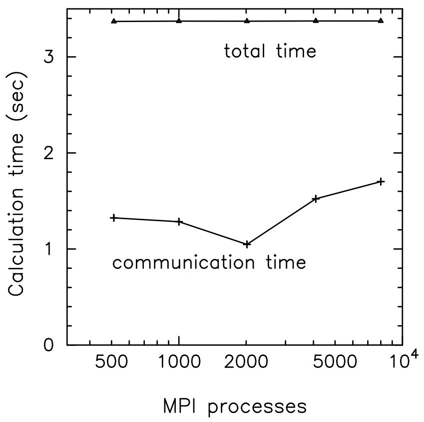

Fig. 4: Time per timestep in second for the weak-scaling test. The horizontal axis is the number of MPI processes. Triangles and crosses show the total time per timestep and communication time per timestep, respectively.

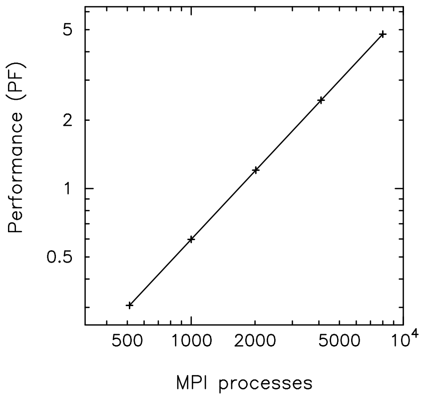

Fig. 5: Performance in petaflops for weak-scaling test. The calculation speed in Petaflops is plotted as a function of the number of MPI processes.

を使用してシミュレーションを行った．プロセス数は 512 から 8000 まで変化させた．暁光でのナイーブアプローチのすべての測定に対して，$N_t = 4$ とした．改善されたスキームの N_t の効果の詳細については次のセクションで議論することとする．

図 4 に 1 タイムステップあたりの時間を示す．タイムステップあたりの合計時間はあまり変動を見せない．一方，通信時間はやや大きな変動を見せ，MPI プロセスが多くなると通信時間が増える傾向にある．セクション 2.3 で説明したように，通信は計算と完全にオーバーラップしている．よって，通信時間は計算時間を超えない限り，合計時間は一定である．

図 5 は，PFlops 単位での計算速度を示している．計算速度はプロセス数の増加に対して完全な線形での増加を見せている．

これらの結果をまとめると，8000 MPI プロセスの計算性能は，4.78PFlops，もしくは理論ピーク性能の 21.5% である．

3.2 「改善された」アプローチ

このセクションでは，「改善された」 アプローチでの性能を述べる．セクション 2.4 で議論したように，空間差分についてのナイーブアプローチの性能は非常に小さい L1 キャッシュを持ったマシンでは比較的低い．実際に，L1D のキャッシュミスレートは 40%近くである．同時に，

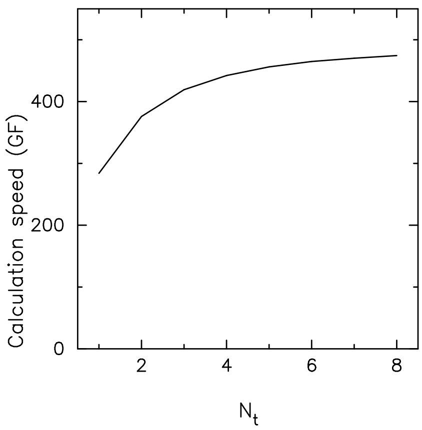

Fig. 6: Calculation speed plotted as a function of the size of temporal blocking N_t. The improved scheme is used.

演算数も比較的大きい．

改善されたスキームでは，1 グリッドポイントがアクセスする合計の語数とユニークワーズ（1 ステップでアクセスする語数（同じ語に複数回アクセスしても 1 と数える））数は大きく減る．ユニークアクセスの場合の係数は 5 で，625 から 125 になる．合計語数も同程度の割合で減少する．今後，更に良い効率を達成することが期待できる．現在の性能はやや低いが，空間差分の冗長な計算数が減るので実際の計算速度は 25%程度改善した．

図 6 はテンポラルブロッキングの効果を示している．N_t が増加すると，計算時間が大きく改善していることが分かる．

我々がテンポラルブロッキングを用いることによって大きく計算スピードアップを達成したという事実は，大きな N_t を計算した時の我々の実装した計算速度はもはやメインメモリバンド幅で制限されないことを意味している．更に，20%近い実行効率を達成したが，これはメモリバンド幅が非常に大きな計算機で達成された効率と同程度である．図 7 では N_t を変えた時に計算時間とメインメモリアクセス時間がどのように変化するかを表している．$N_t = 1$ の時（テンポラルブロッキングを使っていない場合），メインメモリアクセスは合計計算時間の 60%を占めている．しかし，$N_t = 8$ の時は 30%しか占めていない．これは合計時間から観測される明確なスピードアップの理由である．

$N_t = 8$ の場合に合計時間の 70%が計算に使われていることは全体の効率をあと 20%程度上げることが出来ることを意味する．この乖離の理由はまだ調査中である．

暁光で 1 秒間に更新できるグリッド数は 1.18×10^{12} である．もし暁光が稼働していたら，改善されたスキームでは 1.5×10^{12} と予測される．

4. 議論とまとめ

この論文では，高効率の陽的スキームである SL4TH3 をテンポラルブロッキングと組み合わせ，スーパーコンピュータ暁光上での圧縮性流体のシミュレーションについての性能と実装について述べてきた．

測定した性能は 8000 個の PEZY-SC2 チップで，15840^3 グリッドを用いた亜音速の乱流シミュレーションで 4.78PFlops もしくは理論ピーク性能の 21.5%であった．このスピードは 1 秒あたり 1.18×10^{12} のグリッドポイントのアップデートに対応する．この測定は，“ ナイーブスキーム “ によって行われており，改善版スキームでは，1.5×10^{12} アップデートできると予測される．改善版スキームの場合，テンポラルブロッキングの効果は，$N_t = 8$ の場合，1.7 倍程度である．これらより，テンポラルブロッキングによって非常に大きなスピードアップを達成したと言える．

達成性能より，たとえ京コンピュータの 1/20 のメモリ

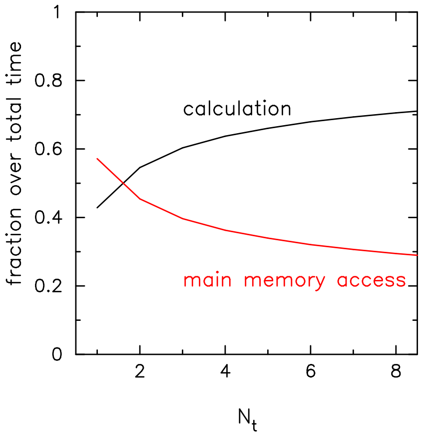

Fig. 7: Ratios of times spent in calculation and main memory access plotted against the size of temporal blocking N_t. The improved scheme is used.

バンド幅の PEZY-SC2 でも，京コンピュータと同程度の効率を達成できること，更にテンポラルブロッキングの適切な使用により合計実行時間が減ることを示した．我々の結果は，規則格子上での陽的解法の計算で比較的メモリバンド幅の低い HPC システムでも効率よく使用できることを意味している．また更なるパフォーマンス性能の向上のために，テンポラルブロッキングのアイディアを適用することが出来る．

我々のアプローチでは，セクション 2.5 で述べたように，並列化とテンポラルブロッキングの実装と微分方程式についての有限差分法の実装は完全に分離されている．そして，通信と計算のオーバーラップなどの最適化は並列化フレームワークの側で引き受けている．有限差分スキームの実装は図 1 の右下の図の 1 ブロックのアップデートを実行するためのコードを提供している．解くべき問題のすべての物理と数学はこのパートに置かれ，並列化フレームワークには存在しない．Formura フレームワークは規則格子のあらゆる陽解法に適用することが出来る．テンポラルブロッキングについても（SL4TH3 スキームを使用した時よりも効率は良くないかもしれないが）あらゆる陽解法での使用が可能である．

陽解法，特に音速抑制法のような十分な時間スケール変換と組み合わせた場合に，陽解法は古典的な陰解法より簡単に効率を上げることが出来る．ゆるされる最大の時間刻みはそれほど大きく変わらない一方，要求されるメインメモリバンド幅と通信レイテンシは大きく違うためである．

我々は本研究結果より，古典的な陰的で反復を必要とする解法の複雑でクレバーな方法から，陽的解法で物理量をスケールし CFL 条件のような制限を緩和するような，賢明な方法にシフトしていくことが大切であると信じる．更に，高いメモリバンド幅や計算効率を必要としない新しい高次精度の陽的スキームを開発・改善していくことが肝要である．

謝辞

本研究開発の中核部分に大きく貢献した故村主崇行氏に深い感謝を捧げます．

本研究は文部科学省次世代領域研究開発（高性能汎用計算高度利用事業費補助金）「ヘテロジニアス・メニーコア計算機による大規模計算科学」の補助を受けています．また一部には理化学研究所，海洋研究開発機構，株式会社 PEZY Computing，株式会社 Exascaler からも助力をいただきました．

参考文献

(1) C. Yang, W. Xue, H. Fu, H. You, X. Wang, Y. Ao, F. Liu, L. Gan, P. Xu, L. Wang et al., “10m-core scalable fully-implicit solver for nonhydrostatic atmospheric dynamics,” in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Press, (2016), p. 6.

(2) M. Rempel, “Solar Differential Rotation and Meridional Flow: The Role of a Subadiabatic Tachocline for the Taylor-Proudman Balance,” ApJ, 622 (2005), pp. 1320–1332

(3) ——, “Flux-Transport Dynamos with Lorentz Force Feedback on Differential Rotation and Meridional Flow: Saturation Mechanism and Torsional Oscillations,” ApJ, 647 (2006), pp. 662–675

(4) K. Takeyama, T. R. Saitoh, and J. Makino, “Variable inertia method: A novel numerical method for mantle convection simulation,” New Astronomy, 50 (2017), pp. 82–103

(5) N. Y. Gnedin and T. Abel, “Multi-dimensional cosmological radiative transfer with a Variable Eddington Tensor formalism,” New Astronomy, 6 (2001), pp. 437–455

(6) T. Muranushi and J. Makino, “Optimal temporal blocking for stencil computation,” Procedia Computer Science, 51 (2015), pp. 1303–1312

(7) H. Yashiro, M. Terai, R. Yoshida, S.-i. Iga, K. Minami, and H. Tomita, “Performance analysis and optimization of nonhydrostatic icosahedral atmospheric model (nicam) on the k computer and tsubame2.5,” in Proceedings of the Platform for Advanced Scientific Computing Conference, ser. PASC ’16. New York, NY, USA: ACM, (2016), pp. 3:1– 3:8. [Online]. Available: http://doi.acm.org/10.1145/2929908.2929911

(8) H. Hotta, M. Rempel, and T. Yokoyama, “High-resolution calculations of the solar global convection with the reduced speed of sound technique. i. the structure of the convection and the magnetic field without the rotation,” The Astrophysical Journal, 786 (2014), no. 1, p. 24

(9) T. Muranushi, H. Hotta, J. Makino, S. Nishizawa, H. Tomita, K. Nitadori, M. Iwasawa, N. Hosono, Y. Maruyama, H. Inoue et al., “Simulations of below-ground dynamics of fungi: 1.184 pflops attained by automated generation and autotuning of temporal blocking codes,” in High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for. IEEE, (2016), pp. 23–33.

(10) T. Muranushi, S. Nishizawa, H. Tomita, K. Nitadori, M. Iwasawa, Y. Maruyama, H. Yashiro, Y. Nakamura, H. Hotta, J. Makino et al., “Automatic generation of efficient codes from mathematical descriptions of stencil computation,” in Proceedings of the 5th International Workshop on Functional High-Performance Computing. ACM, (2016), pp. 17–22.

(11) T. Aoki, “Interpolated differential operator (ido) scheme for solving partial differential equations,” Computer Physics Communications, 102 (1997), no. 1, pp. 132 – 146

(12) J. Makino, “Optimal order and time-step criterion for aarseth-type nbody integrators,” ApJ, 369 (1991), pp. 200–212

(13) J. Makino and S. J. Aarseth, “On a hermite integrator with ahmad-cohen scheme for gravitational many-body problems,” PASJ, 44 (1992), pp. 141–151