Physis フレームワークにおける 性能モデルに基づく通信の自動最適化に向けて

河村知輝[†] 丸山直也[†,††] 松岡聡[†,††]

1. 研究背景

流体運動や熱伝導などの偏微分方程式を数値計算で解く際にステンシル計算が頻出する．各セルに行う計算は全て独立に行うことができ，メモリアクセスなども規則的であるため，高い並列性を有している．そこで，多くのコアを搭載することで高い演算性能を持つ GPU を使う研究が多くなされている．

その一つとして，ステンシル計算のみを対象とした，GPU クラスタ向け自動並列化フレームワーク（Physis）の研究[1] が行われている．このフレームワークを使うと，ユーザーは本質的に必要である計算カーネルの記述のみで，複数ノード環境で実行できるコードを生成することができる．

しかし，現在この Physis では通信と計算をオーバーラップさせる最適化は施されているが，ステンシル計算特有の最適化などはまだされていない．

2. 提案

本研究ではステンシル計算の最適化として，既存手法であるテンポラルブロッキング[2] を Physis に適用することを提案する．

2.1 提案達成に向けた課題

テンポラルブロッキングとは袖領域の量を増加させることにより，一度の通信で複数回の計算を行う手法である．これにより，同じ計算を複数回行う領域が発生し計算コストが増加してしまうが，通信回数が減るため通信レイテンシを抑えることができる．一度の通信で何回計算を行うかにより，計算コストの増加量と通信コストの減少量が変化するため，この最適化手法には計算回数というパラメータが存在する．さらに，本研究では通信と計算をオーバーラップさせる最適化手法と組み合わせることになる．そのため，Physis に適用するためには，テンポラルブロッキングとオーバーラップを組み合わせた場合の性能検証や，最適な計算回数を定める機構の実装などが課題となる．

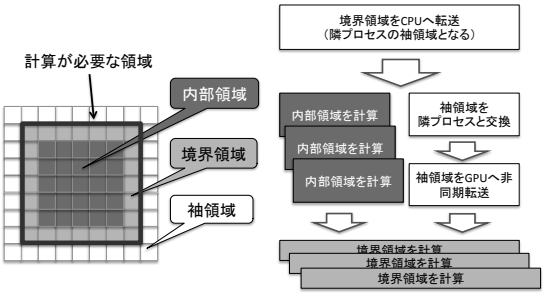

図 1 オーバーラップ+テンポラルブロッキングの流れ

本稿ではこれらの課題のうち，二つの最適化手法を組み合わせた場合の性能検証と，最適な計算回数を導出する方法の一つとして性能モデルを構築し，評価を行う．

2.2 性能モデル構築

オーバーラップとテンポラルブロッキングを両方適用すると 1 ステップでの処理は**図 1** のようになる．袖領域を必要とする領域（境界領域）と袖領域を必要としない領域（内部領域）に分けて計算を行うことで，袖領域通信と内部領域計算を同時に実行することができる．

次に性能モデル構築について説明をする．1 通信あたりの計算回数を k 回，GPU から CPU への通信時間を T_{G2C}，MPI による CPU 間の通信時間を T_{MPI}，CPU から GPU への通信時間を T_{C2G}，k 回の内部領域計算時間を T_{inner}，k 回の境界領域計算時間を $T_{boundary}$ とすると，1 ステップあたりの実行時間 $t_{1step}(k)$ は次のようになる．

$$\max(T_{inner}, T_{G2C} + T_{MPI} + T_{C2G}) + T_{boundary}$$

ただし，各処理の実行時間は，各種アーキテクチャの理論性能から計算するのではなく，実行環境のマシンで別途に個別に測定したものを使用している．

さらに，全計算回数を $loop$ 回とすれば，全体のモデルは次のような k に関する最小化問題になる．

$$\text{Minimize} : \frac{loop}{k} \times t_{1step}(k)$$

† 東京工業大学
Tokyo Institute of Technology
†† 独立行政法人科学技術振興機構 CREST
JST CREST

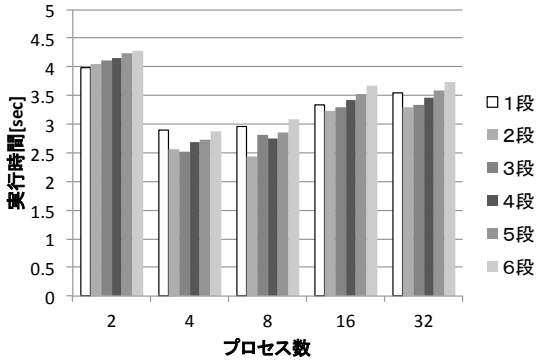

図 2 手動最適化による性能評価（ストロングスケール）

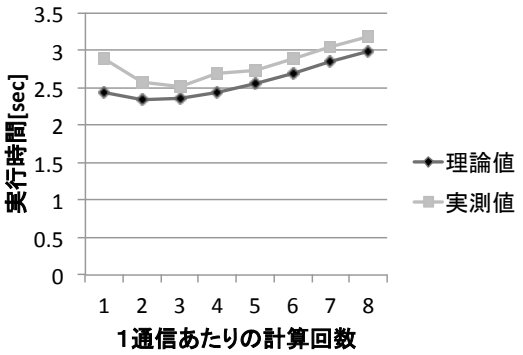

図 3 性能モデル評価（プロセス数を 4 に固定）

3. 実験・評価

Physis への実装の準備段階として，手動最適化コードによる評価を行なっていく．ステンシル計算は 3 次元の 7 点ステンシルを対象とし，MPI により Z 軸方向 1 次元にデータ分割を行なっている．

まずは追加最適化の性能評価を TSUBAME2.0 上で行った．全体の問題サイズは $256 \times 256 \times 256$ とし，GPU 数で分割して計測したグラフが**図 2** となる．図中の段数が 1 通信あたりの計算回数を表しており，1 段のグラフがオーバーラップのみを表している．追加最適化版ではオーバーラップのみ版に比べてプロセス数 8，計算回数 2 回のときに最大で約 17%の性能向上した．

図 3 は性能モデルと実際の実行時間を，計算回数で変化させたときの推移を表している．性能モデルによる理論値と，実プログラムによる実測値での誤差は 6%から 15%となった．二つの間に誤差はあるが，二つのグラフの挙動は非常に似ており最適な計算回数の値を絞り込むことは十分可能であるといえる．

4. 考　　察

図 2 で各処理の実行時間を計測したところ，プロセス数が 2 のときのみ計算時間が通信時間より大きいことがわかった．テンポラルブロッキングでは計算コストは増加するため，プロセス数 2 のときのみ性能向上していない理由は説明できる．また，8 プロセス以降で全体の実行時間が減少していない．1 次元分割を行った場合には通信を行うデータ量が分割数によって変化しないため理論的にも増加はしないが，実験結果では実行時間が増加してしまっている．これは，今回の MPI 実装ではノンブロッキング通信を使うことで通信の順番などを MPI 側に任せているため，並列数増加に伴い通信が複雑になり，結果として実行時間が増加してしまっていると考えられる．

理論値を算出するために MPI 通信を単独で行ったときや，手動最適化コードを実行した際，実行時間に最大で 10%程度のばらつきが出た．これは実験環境である TSUBAME2.0 ではプログラムを実行する際に使用されるノードの割り当てがランダムに行われるためと考えられる．そのため現在は複数回試行をし，その中で最小の値を採用する形をとっている．

5. まとめと今後の課題

本稿では Physis にテンポラルブロッキングを適用させる準備段階として，オーバーラップとテンポラルブロッキングを組み合わせた最適化版を実装し，オーバーラップのみの最適化版と比べて最大で 17%の性能向上を確認した．また，テンポラルブロッキングの計算回数の最適回数を得るために性能モデルを構築，評価を行った．その結果，誤差が 6%から 15%になったものの，計算回数変化による性能の上下の挙動は非常に似ており最適値の絞り込みは十分可能であるとわかった．

今後の課題として，Physis フレームワークに最適化の実装を施すことが挙げられる．また，並列数を増加させた場合に通信データ量を抑えることができる 2 次元，3 次元分割での実装や，性能モデルにより絞り込んだ範囲から最適な計算回数を定める機構を作ることも挙げられる．

参 考 文 献

1) Naoya Maruyama, Tatsuo Nomura, Kento Sato, and Satoshi Matsuoka. Physis: an implicitly parallel programming model for stencil computations on large-scale gpu-accelerated supercomputers. SC '11.
2) M. Wittmann, G. Hager, and G. Wellein. Multicore-aware parallel temporal blocking of stencil codes for shared and distributed memory. april 2010.