適合格子細分化法による細粒度マルチスレッドライブラリの評価

中島 潤[†] 田浦 健次朗[†]

1. 背景

計算機のマルチコア化やノード数の増加，ノード間通信の高速化などの技術発展の恩恵を受け，並列計算環境の規模や，そのハードウェアがもつ並列度は今後ますます大きくなると予測されている[1]．このような大規模並列計算環境の性能を十分に発揮するには，そのハードウェアがもつ，あらゆる要素において並列性を引き出すことを考慮してアプリケーションを記述しなければならない．

それに加えて，より大規模な計算を高精度に，かつ高速に行うことに対する要請を満たすため，近年の科学技術計算においては，適合格子細分化法や高速多重極展開法 といった，計算の手順を部分領域ごとに動的に変更することで，計算量の増加を抑えつつ計算精度を向上させる手法が利用されている．これらの手法は部分領域ごとの計算負荷が変動するため，効率的な並列実行のためには計算負荷を動的に分散することが必要である．

しかし，これらすべての要素を考慮してアプリケーションを記述することは極めて煩雑である．

2. MassiveThreads の提案

計算ノード内における動的な負荷分散を達成するための手法として，Cilk[2] などの処理系が提供している，細粒度なスレッドが存在している．これを利用するアプリケーションでは，アプリケーションの並列性に応じた数のスレッドを動的に作成・削除するだけで，処理系の機能を利用して動的な負荷分散を達成することができる．しかし，これら既存の処理系は計算がノード内で完結することを前提としており，他のノードとの協調のために各スレッドが I/O によって通信を行うことなどは想定されていない．

そこで我々は，細粒度スレッドによる動的な負荷分散と，各スレッドからの I/O の効率的な実行を両立するマルチスレッド処理系，MassiveThreads を提案している．MassiveThreads や，それを基盤ソフトウェアとして利用した処理系により，アプリケーションの並列性に応じてスレッドを動的に作成・削除し，それらが自由に通信を行う，という直感的な形で高性能な並列分散アプリケーションを記述できるようになることが期待される．

MassiveThreads は，既存の処理系に用いられている手法を拡張した実装により先に挙げた要件を両立し，さらに OS 標準のマルチスレッド処理系と同様に利用可能なインターフェースを備えている[3]．本稿では，スレッドの軽量性と動的負荷分散能力を実アプリケーションで確認するために，適合格子細分化法による移流計算を MassiveThreads により実装し，性能評価を行なった結果について述べる．

† 東京大学
The University of Tokyo

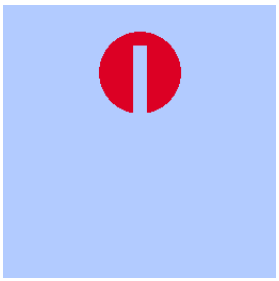

図 1 Zalesak 問題の初期状態
Fig. 1 Initial state of Zalesak problem

3. 適合格子細分化法による評価

3.1 問題設定・実装

問題としては Zalesak 問題[4] の移流計算を用いた．これは**図 1** に示すような濃度分布を角速度一定の流れのもとで回転させた際の濃度分布の変化を計算するものである．本実験においては，計算領域を 256×256 の格子に離散化し，これに対して濃度勾配を指標とした最大 2 レベルの適合格子細分化法を適用して計算を行なった．移流計算の手法としては Cubic セミラグランジュ法を用いた．

並列化は，16×16 格子点を単位として計算領域を

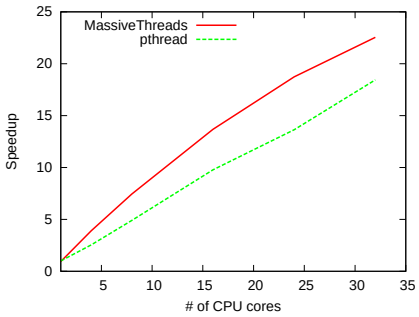

図 2 性能向上率
Fig. 2 Speedup

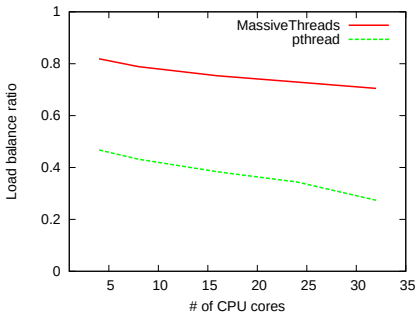

図 3 負荷分散の効率
Fig. 3 Load balance efficiency

分割し，それぞれの領域に MassiveThreads の 1 スレッドを割り当てることによって行なった．計算にはグローバルな同期をとる必要のある箇所が存在しているが，これは同期の直前で全スレッドの終了待ちを行い，その後スレッドを起動しなおすことによって実現している．比較対象としては，Linux 標準の pthread によってスレッドを CPU コア数だけ作成し，領域を 1 次元に分割してそれらに割り振り並列化したものを用いた．

実験環境としては，Opteron 8354(2.2GHz)8 ソケットからなる 32 コアの計算ノードを，コンパイラには GCC4.4.1 を用いた．

3.2 実験結果

pthread による実装を 1 コアのみ使用して実行した際の時間を基準にした性能向上率を**図 2** に，使用するコア数を変化させた際の負荷分散の効率を**図 3** に示す．動的負荷分散の効果により，MassiveThreads による実装は pthread による実装よりも高い性能向上率と，良好な負荷分散を示している．

なお，負荷分散の効率は N_C を使用された CPU コア数 (スレッド数)，$n_u(t,i)$ をタイムステップ t において i 番目のコアが計算を行なった格子点の数として，以下の式で計算した．

$$\sum_t \max_i(n_u(t,i)) \div \left(\sum_t \sum_i (n_u(t,i)) \div N_c \right)$$

4. 今後の課題

4.1 局所性に配慮したスケジューリング

現在の MassiveThreads の実装はデータの局所性に配慮したスケジューリングを行っていない．そこで NUMA アーキテクチャのような，データの局所性に配慮することが性能を発揮するために重要になるアーキテクチャ上でよりスケーラブルに動作するように，データの局所性に対して配慮したスケジューリングや，それを補助するユーザーインターフェースを実装する．

4.2 計算と I/O 処理の共存に関する検討・評価

MassiveThreads が想定しているアプリケーションの記述では，多くのスレッドが存在し，それぞれが自由に計算や I/O 呼び出しを行う．しかし，このような場合に性能を最大化するためにとるべきスケジューリング方針は自明ではない．

アプリケーション全体として最大の性能を得るために，どのようにスレッドをスケジュールすれば最適な性能が得られるのかを検討し，実際のアプリケーションを用いてその効果を確認する必要がある．

参 考 文 献

1) Dongarra, J., Beckman, P., Cappello, F., Lippert, T., Matsuoka, S., Messina, P., Aerts, P., Trefethen, A. and Valero, M.: The International Exascale Software Project Roadmap, University of Tennessee EECS Technical Report (2010).
2) Blumofe, R. D., Joerg, C. F., Kuszmaul, B. C., Leiserson, C. E., Randall, K. H. and Zhou, Y.: Cilk: an efficient multithreaded runtime system, *SIGPLAN Not.*, Vol. 30, No. 8, pp. 207–216 (1995).
3) 中島潤, 田浦健次朗: 高効率な I/O と軽量性を両立させるマルチスレッド処理系, 情報処理学会 論文誌 プログラミング (PRO), Vol. 4 No. 1, pp. 13–26 (2011).
4) Zalesak, S. T.: Fully Multidimensional Flux-Corrected Transport Algorithms for Fluids, *Journal of Computational Physics*, Vol. 31, No. 3, pp. 335 – 362 (1979).