TSUBAME 共同利用　平成 26 年度　学術利用　成果報告書

利用課題名　高性能と高生産性を両立する並列分散ランタイムシステム
英文：Parallel and Distributed Runtime System Realizing High Performance and High Productivity

田浦健次朗
Kenjiro Taura

東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo

邦文抄録（300 字程度）

当研究グループでは、大規模並列計算機において高性能な並列プログラムを容易に開発できるようにすることを目的として、タスク並列プログラミングモデルに関する研究を行っている。本研究課題では、昨年度から引き続いて、当研究グループで開発している分散メモリ並列計算機向けタスク並列処理系 MassiveThreads/DM の性能改善・機能追加を実施し、TSUBAME 上での動作確認と性能評価を行った。結果として、二分木上再帰タスク生成、Unbalanced Tree Search、N クイーン問題の 3 つのベンチマークプログラムについて良好な負荷分散性能が得られることを確認した。

英文抄録（100 words 程度）

Our group has studied task-parallel programming models to improve productivity in developing high performance parallel programs on large-scale distributed memory machines. In this project, we conducted performance improvement for MassiveThreads/DM, a task-parallel library for distributed memory machines, and performance evaluation on the TSUBAME supercomputer. As a result, we confirmed that MassiveThreads/DM achieved good scalability in three task-parallel benchmark programs (Binary Task Creation, Unbalanced Tree Search, NQueens).

Keywords: Task parallelism; lightweight multithreading; thread migration; distributed work stealing; remote direct memory access

背景と目的

近年、並列計算機の複雑化が進んでいる。特に分散メモリ並列計算機は、単に構成する計算ノードの数が増えているだけでなく、計算ノード内に複数の CPU を持ち、さらに CPU 内にプロセッサコアを複数持つような複雑な構成も珍しくなくなっている。このような並列計算機の性能を最大限引き出すためには、複数のメモリ・ネットワーク階層について熟知し、その知識に基づいて並列プログラムを記述しなければならない。

当研究グループでは、この問題を解決するために、(1)分割統治形式による通信効率の良い並列アルゴリズム（cache-oblivious アルゴリズム）を設計し、(2)そのアルゴリズムによって再帰的に分解されるタスク・データを、各メモリ・ネットワーク階層に適切に配置するようなタスク並列処理系の実現を目指している。このようなタスク並列処理系は、共有メモリ並列計算機では軽量マルチスレッド処理系（Cilk, Java fork/join framework など）という形で実現され、広く使われるようになってきている。

当研究グループでは、その一環として、次の二つのシステムから構成される分散メモリ並列計算機向けタスク並列処理系 MassiveThreads/DM を提案する。MassiveThreads/DM は以下の要素技術で構成される：

1. 分散メモリ並列計算機において、ノード間負荷分散を実現するタスク並列ライブラリ：
   従来のタスク並列ライブラリの多くは共有メモリ計算機向けであり、ノード内での動的な負荷分散機能を提供する一方で、ノード間で負荷分散を行う機能を提供しているものは数少ない。また、ノード間負荷分散機能を提供している処理系についても、

コード変換を用いてタスクのマイグレーションを実装しており、利用するためには専用のコンパイラを使う必要があるなど、利便性の点で問題がある。

この問題に対して、提案するタスク並列ライブラリでは、タスクのもつスタックをノード間で転送することにより、コード変換を用いない C ライブラリとしての実装を実現する。

2. タスク並列処理系からの利用に特化した大域アドレス空間ライブラリ:

タスク並列モデルと大域アドレス空間モデルを組み合わせるにあたっては、タスク並列モデルの「計算機のもつ並列性と比べて非常に多くのタスクが生成される」という特徴が問題となる。既存の大域アドレス空間モデルにおいては、大域メモリにアクセスする際に、アクセスするメモリ領域の実体が存在するノードとの通信が発生する。これは、前述のタスク並列モデルの性質と組み合わせたときに、大量に生成されたタスクがそれぞれ通信を行うこととなり、(1)通信の準備にかかるオーバーヘッドがタスクの数だけ発生し、また(2)データ量の少ない通信によって通信帯域を活用できないといった点で、計算機のもつ性能を最大限活用する上での障害となる。

提案する大域アドレス空間ライブラリでは、この問題を解決するため、既存の大域アドレス空間モデルに対して、次の二つの機能を新たに導入する。

(a) キャッシュ機能: タスク間で共有されるキャッシュ領域を提供する。この機能を利用することによって、あるタスクがまとめてデータを読み込み、他のタスクは通信なしでキャッシュからデータを読むだけで済むようにすることが可能になる。

(b)大域メモリの配置を動的に変更する機能(ページマイグレーション): タスクがプロセス間を移動した際に、タスクのアクセスする大域メモリも同時に移動させることができる。

概要

本利用課題では、昨年度から引き続いて TSUBAME を用いた MassiveThreads/DM の設計・開発を実施した。今年度は主に軽量マルチスレッド機能の高性能化に取り組んだ。

MassiveThreads/DM の軽量マルチスレッド機能は、生成した軽量スレッドをノード間で移動させることによって、動的な負荷分散(ワークスティーリング)を行っている。これまでの軽量マルチスレッド実装は、既存のスレッドマイグレーション手法である iso-address 法 [Antoniu et al.,1999] を用いて実装していたが、iso-address 法には負荷分散のスケーラビリティに関して次の二つの制約が存在する:

(1) スレッドのスタック領域を割り当てるために、プロセッサ数に比例した仮想アドレス空間を1ノードごとに予約しておく必要がある。例えば、300万並列の環境で、1ワーカ上で同時に存在しうるスレッド数を8192個、スレッドスタックのサイズを16KBとすると、必要な仮想アドレス空間のサイズは $2^{22+13+14}$ バイトとなる。これは現在の x86-64プロセッサの仮想アドレス空間の上限である 2^{48} バイトを越えており、このような環境ではiso-address法を適用することができない。

(2) スケーラブルなワークスティーリングを実現するためには、Remote Direct Memory Access (RDMA) を用いた片側タスクスティーリングが重要である [Dinan et al., 2009]。RDMA を用いてスレッドマイグレーションを実装するためには、各スレッドのもつスタック領域を物理メモリに固定する必要があるが、iso-address 法によるスレッドマイグレーションでは、スレッドのスタック領域が広大な仮想アドレス空間のどこかに割り当てられるようになっているため、スレッドスタックとして使われる領域をすべて物理メモリに固定することは不可能である。

我々は、これらの問題を解決し、スケーラブルな負荷分散を実現するため、仮想メモリ使用量を逐次実行時と同程度にまで削減するスレッドマイグレーション方式 uni-address 法を新しく提案した。Uni-address 法は、iso-address 法が「すべてのスレッドが常に正しい仮想アドレス上に配置されている」という仮定を保つように設計されているのに対して、「すべてのスレッドが少なくと

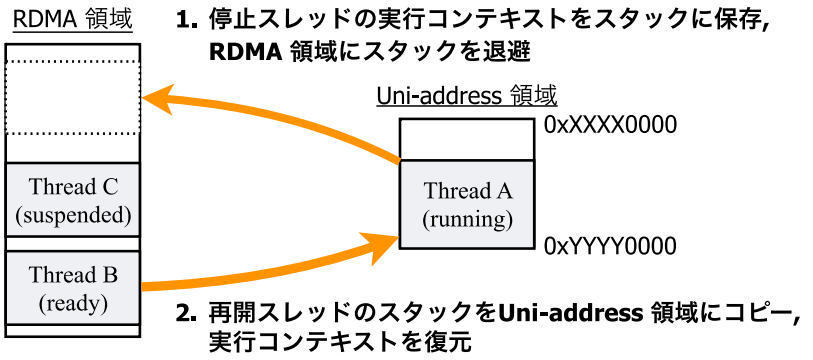

図 1. Uni-address 法におけるコンテキストスイッチの動作

も実行中に正しい仮想アドレス上に配置されていればよい」という考えに基づいてスレッドマイグレーションを実現している。

以下、uni-address 法の基本動作を説明する。まず、スタックを配置する領域を uni-address 領域と RDMA 領域の二つに分ける。Uni-address 領域は実行中スレッドを配置する領域で全ノードで同じ仮想アドレスを割り当てておく。RDMA 領域は停止中のスレッドのスタックを退避するための領域で、任意の仮想アドレスを割り当ててよいものとする。Uni-address 法では、すべてのスレッドの仮想アドレスを uni-address 領域の仮想アドレス上に割り当てる。そして、あるスレッドにコンテキストスイッチする際には、実行中スレッドのスタック領域を RDMA 領域に退避したのち、実行したいスレッドのスタック領域を uni-address 領域にロードした上でスレッドのコンテキストを復元する（図 1）。ノードをまたがってスレッドを移動させる際には、移動元ノードの RDMA 領域から移動先の uni-address 領域へスレッドスタックを RDMA 転送したのち、そのスレッドのコンテキストを復元すればよい。このようにスレッドを管理することで、実行中のスレッドについて常に正しい仮想アドレス上に配置することができる。

Uni-address 法では、コンテキストスイッチのたびに RDMA 領域へのスタックの退避が必要で、従来の軽量スレッド実装と比較してスレッド操作のオーバーヘッドが大きくなる。そこで我々は、典型的なワークスティーリングスケジューラについて、uni-address 領域に複数のスレッドを同時に配置できるようにすることで RDMA 領域への退避を削減する手法を開発した。この手法を用いると、スタックコピーの回数をワークスティーリング

Uni-address threads (server)	125 cycles
Uni-address threads (polling)	286 cycles
MassiveThreads	125 cycles

図 2. スレッド生成オーバーヘッド

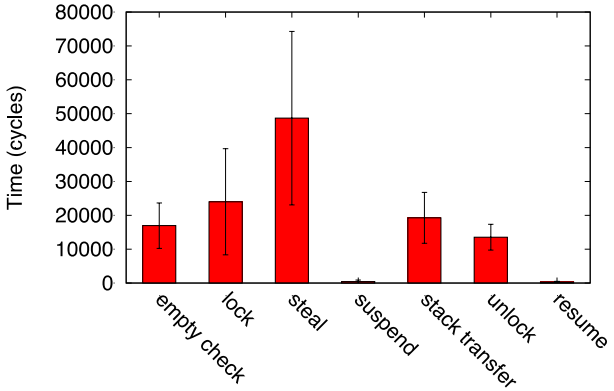

図 3. ワークスティーリングにかかる時間の内訳

回数を N としたとき O(N)回にまで削減できる。

我々は、uni-address 法を用いたスレッドマイグレーションを提供する軽量マルチスレッドライブラリ uni-address threads を TSUBAME 上に実装した。実装にあたっては、RDMA 通信ライブラリである GASNet を用いて行った。GASNet は RDMA READ/WRITE をサポートしている一方で、ワークスティーリングキューの実装に必要な RDMA fetch-and-add などのアトミック命令をサポートしていない。そこで我々は、RDMA fetch-and-add を 2 つの方法を用いてソフトウェア実装した。一つは 1 計算コアごとに 1 通信コアを割り当て、通信コアに fetch-and-add を処理させる方法で、RDMA のもつ、通信相手との同期を必要としない通信方式を再現する

ものである(server ベース実装)。もう一つは、計算の最中に定期的にfetch-and-addリクエストが到着しているかどうか確認し、fetch-and-add 処理を実行する方式である(polling ベース実装)。こちらは通信用のコアを用意する必要がなく、すべてのコアを計算に用いることができる。

結果および考察

本利用課題では、uni-address threads の性能評価を行った。具体的には、スレッド生成オーバーヘッドの評価、1 回のワークスティーリングにかかる時間の評価、3 つの負荷分散ベンチマークによる仮想メモリ使用量と負荷分散性能の評価を行った。

スレッド生成にかかるオーバーヘッドを図 1 に示す。図 2 を見ると、提案手法のスレッド生成オーバーヘッドは 125 cycles と十分小さく、既存の共有メモリ環境向け軽量マルチスレッドライブラリである MassiveThreads と同等のオーバーヘッドとなっている。polling ベース実装のスレッド生成オーバーヘッドが 285 cycles と server ベース実装に対して大きくなっているのは、fetch-and-add リクエストの到着確認のコスト(gasnet_AMPoll 関数の呼び出し）が原因である。

図 3 に示すのは 1 回のワークスティーリングにかかる時間の内訳である。図中の suspend と resume が uni-address 法によるオーバーヘッドを示しており、ワークスティーリング全体にかかる時間に占める割合は十分小さいことが確認できる。

Binary Task Creation、Unbalanced Tree Search、NQueens ソルバの 3 つの負荷分散ベンチマークを実行したときの uni-address threads の並列性能を図 4 に示す。どのベンチマークについても、RDMA ワークスティーリングを再現している server ベース実装については良好なスケーラビリティが得られている。一方で、ワークスティーリング時に計算コア間で同期が必要な polling ベース実装では一部スケーラビリティの低下を確認した。

これらのベンチマークは、スタック領域の仮想メモリ使用量が 104KB 以下で動作することを確認している。

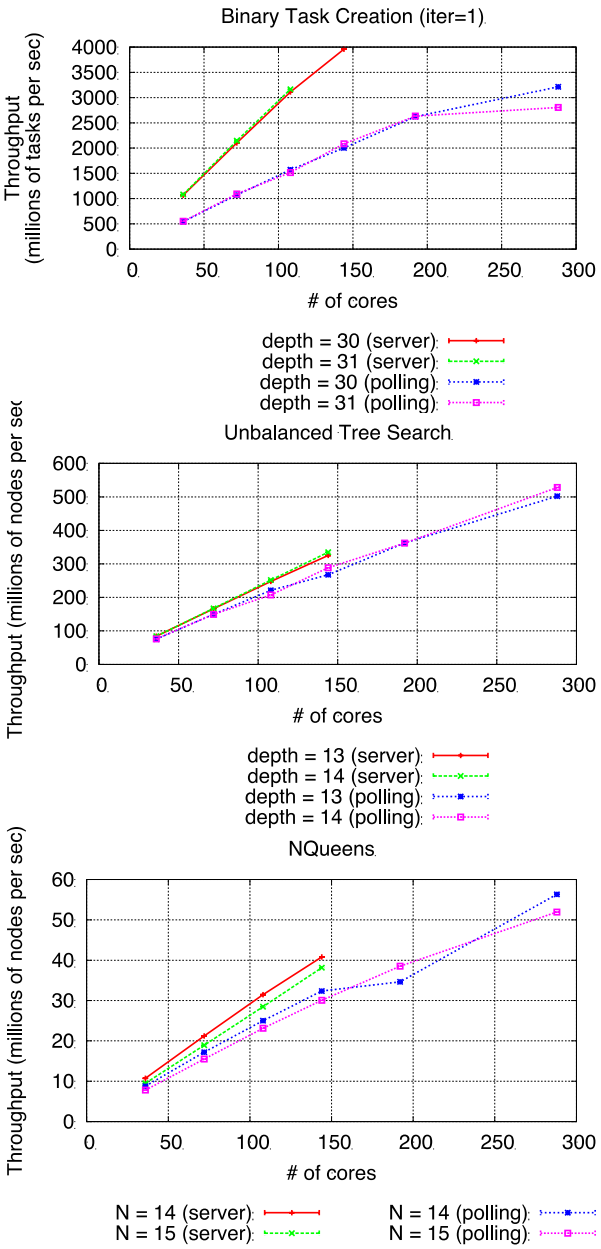

図 4. 各負荷分散ベンチマークのスケーラビリティ

まとめ、今後の課題

本利用課題では、分散メモリ環境向けタスク並列ライブラリ MassiveThreads/DM の設計・実装を進めた。本年度は、特に軽量マルチスレッド機能の負荷分散性能の改善に取り組んだ。従来のスレッドマイグレーション手法に基づくワークスティーリング実装では RDMA を用いた片側タスクスティーリングを実現できないため、負荷分散性能に課題があった。我々は、RDMA 通信でスレッドを転送できる新しいスレッドマイグレーション手法である uni-address 法を新しく開発し、RDMA を用いたスケーラブルなワークスティーリングを実現した。今後は、より多くのコアを用いた実験を行い、高並列時の性能評価を進める予定である。