TSUBAME 共同利用　平成 27 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）
並列計算において、並列計算機を構成する各プロセッサに適切にタスクを割り合て負荷分散することは、ハードウェアのもつ性能を最大限活用する上で非常に重要な仕事の一つである。本研究では、TSUBAME のような 300 ノード規模（S キューの最大構成）のクラスタ型スーパーコンピュータにおいて、自動的なノード間動的負荷分散を提供する軽量マルチスレッドライブラリ Uni-address threads を設計・実装する。本稿では、Uni-address threads におけるスケーラブルなノード間スレッドマイグレーション方式 uni-address と、その TSUBAME 上での実装・評価について報告する。

英文抄録（100 words 程度）
In parallel computing, it is one of the most important work to assign tasks to each processor consisting a parallel computer and do load balancing. In this work, we design and implement uni-address threads, a lightweight multithread library supporting automatic inter-node dynamic load balancing. This paper reports uni-address, a scalable inter-node thread migration scheme, and the implementation and evaluation on the TSUBAME supercomputer.

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

背景と目的

タスク並列処理は、マルチコア環境や NUMA 環境のような共有メモリ型の並列計算環境において、手軽に並列化を行うための手段として広く用いられるようになっている。タスク並列処理では、プログラマはプログラム中の並列実行可能な部分(=タスク)を指定するだけで、処理系が自動的に各プロセッサにタスクを割当て、負荷分散を行う。つまり、タスクの各計算資源への配置を自動化できる。タスク並列機構の実装方式の一つである軽量マルチスレッドは、各タスク(=並列単位)が固有のコールスタックをもち、プログラム上の任意の地点で停止・再開させることができるという点で、他の実装方式と比較して種々のタスクスケジューリングを実装可能ということや、特殊なコードトランスレータを必要とせず、従来のプログラムに容易に組み込み可能ということなどの特長を持っている。

本研究課題では、これまで共有メモリ環境で主に用いられてきた軽量マルチスレッド機構を、分散メモリ環境上に実装する。分散メモリ環境におけるタスク並列処理フレームワークは、X10, Chapel, Scioto など提案されてきているが、これらは、(1) ノード間負荷分散をサポートしていない、もしくは、(2) タスクをスレッドとして実装しておらず、スケジューリングの柔軟性に欠ける，など機能上の制約がある。

我々は、昨年度より、TSUBAME のような大規模分散メモリ環境においてスケーラブルなノード間負荷分散をサポートする軽量マルチスレッドライブラリ Uni-address threads を提案，実装技術の検討を行い、昨年度は最大 24 ノードまで用いた性能評価を行ってきた。 本年度は、Uni-address threads の開発を継続し、S キューの最大構成である 300 ノード規模で、より詳細な性能評価を行うことができた。

結果として、提案する実装方式である uni-address の有効性を示すとともに、動的負荷分散(ワークスティーリング)の TSUBAME (Infiniband クラスタ) 上での実装方式として、「RDMA を用いた実装」が「メッセージング

機構を用いた実装」よりもスケーラブルであることを確認した。この事実は、既存研究にて指摘されていることではあるが、実験的に「RDMA を用いた実装」がスケーラブルであることを示したのは本研究が初めてである。また、「メッセージング機構を用いた実装」のスケーラビリティ低下の主因が、プロセッサ数の増加につれてワークスティーリング要求メッセージの処理時間が増大することによることを確認した。

概要

本利用課題では、昨年度から引き続き、スケーラブルなノード間負荷分散をサポートする軽量スレッドライブラリ Uni-address threads の TSUBAME における設計・実装を進めた。 Uni-address threads は、uni-address と呼ぶ手法を用いてノード境界をまたいだスレッドマイグレーションを実装し、動的負荷分散を行っている。以下では、昨年度の利用課題において開発した uni-address 法の概要と、本年度に進めてきた TSUBAME 向けの実装について報告する。

まず、 uni-address 法について説明する。ノード境界をまたがったスレッドマイグレーションを実装するにあたっては、スレッドのもつコールスタック内部にあるポインタが無効にならないようにコールスタックを移動させる必要がある。既存研究としては iso-address [Antoniu et al., 1999] と呼ばれる手法が存在する。iso-address 法は、各コールスタックのアドレスをクラスタを構成するすべてのノードで一意に割り合てることによって、コールスタックの移動時に、移動前と移動先で同一のアドレス上にスタックを配置できるようにする。一方、iso-address 法には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とし、全ノードで同一の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とし、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

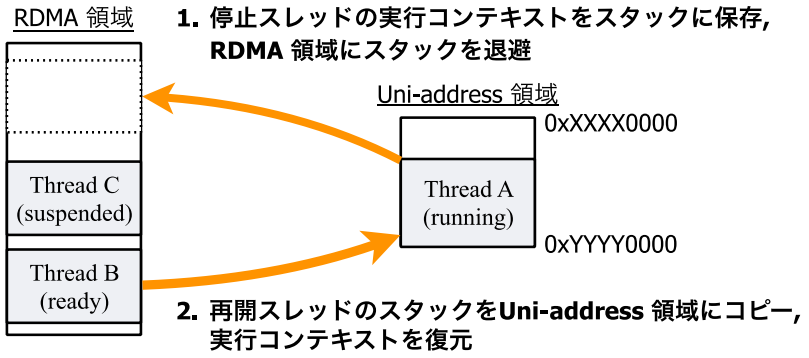

図 1. Uni-address 法におけるコンテキストスイッチの動作

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。紙面の都合によりここでは詳述しないが、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、下位通信ライブラリとして Infiniband をサポートする GASNet を用い、RDMA 機能を活用して Uni-address threads における通信処理、すなわちスレッドマイグレーションを含むワークスティーリング処理を実装した。

GASNet の提供する RDMA のための API 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的に実装した。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

表 1. スレッド生成オーバーヘッド

	時間
Uni-address threads (RDMA-emulated)	184 cycles
Uni-address threads (AM-based)	353 cycles
MassiveThreads	122 cycles

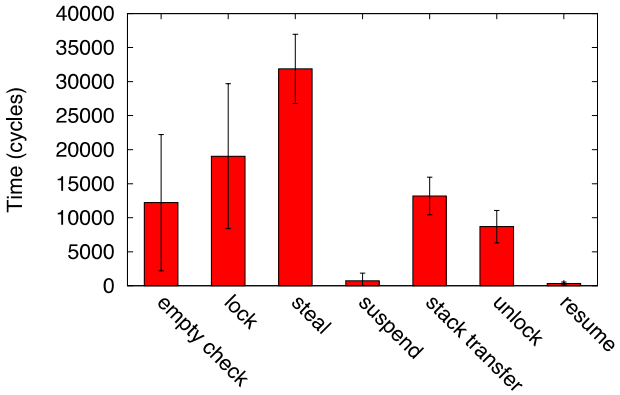

図 2. ワークスティーリングにかかる時間の内訳

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(RDMA-emulated 実装)。もう一つは、メッセージングを用いた実装方法で、計算の最中に定期的に fetch-and-add 要求メッセージ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(AM-based 実装)。こちらは通信用のコアを計算コアと別に用意する必要がなく、ハードウェアのもつ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

Uni-address threads のTSUBAMEにおける性能を評価することを目的として、性能評価実験を行った。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表 1 に示す。表 1 を見ると、RDMA-emulated 実装のスレッド生成オーバーヘッドは 184 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 と比較しても遜色ないオーバーヘッドとなっている。AM-based 実装のスレッド生成オーバーヘッドが 353 cycles と RDMA-emulated 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（gasnet_AMPoll 関数の呼び出し）をスレッド生成ごとに行っていることが原因である。

RDMA-emulated 実装における 1 回のワークスティーリングにかかる時間とその内訳を図 2 に示す。1 回のワークスティーリングにかかる時間は合計で 81K サイクル程度、uni-address 法を用いることによるオーバーヘッド(suspend, resume)は、1.9K サイクル程度であり、残りの時間はワークスティーリングキューの操作に占められている。このことから、uni-address 法を用いることによるワークスティーリングにかかる時間への影響は十分小さいことがわかる。

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 の負荷分散スケーラビリティ(計算コア数を変化させたときの 1 秒あたり実行タスク・ノード数の推移)を図 3 に示す。結果として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ソフトウェア的に実装している RDMA-emulated 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 AM-based 実装では台数効果の低下を確認した。これは、計算コア数が増加するにつれて、fetch-and-add 要求メッセージの処理数が増え、メッセージ処理にかかる時間が増大していることが主な原因であるということを確認している。この結果は、RDMA を用いたワークスティーリング実装がスケーラブルである一方で、メッセージングを用いたワークスティーリング実装方式のスケーラビリティに課題があることを示している。

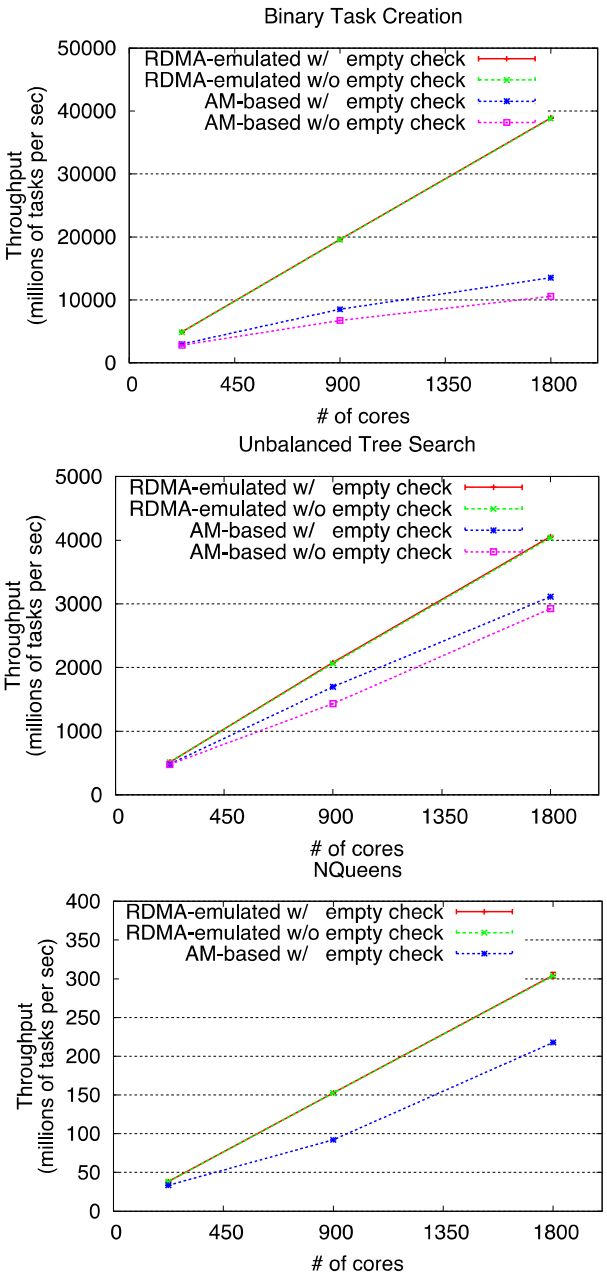

図 3. 各負荷分散ベンチマークのスケーラビリティ

まとめ

本利用課題では、分散メモリ環境においてノード境界をまたがった動的負荷分散を提供する軽量マルチスレッドライブラリ Uni-address threads の設計・実装を行った。今年度は、昨年度から引き続き Uni-address threads の TSUBAME 向け実装に取り組み、最大 1800 計算コアでの性能評価を行うことができた。結果として、RDMA を活用した実装を用いて 1800 コア規模で良好な台数効果が得られること、メッセージングベー

スの実装では負荷分散スケーラビリティに課題がある
ことを確認できた。