An Implementation of Inter-node Communication System with Efficient Light-weight Thread Scheduling

xSIG'19 (May 27-29, 2019)

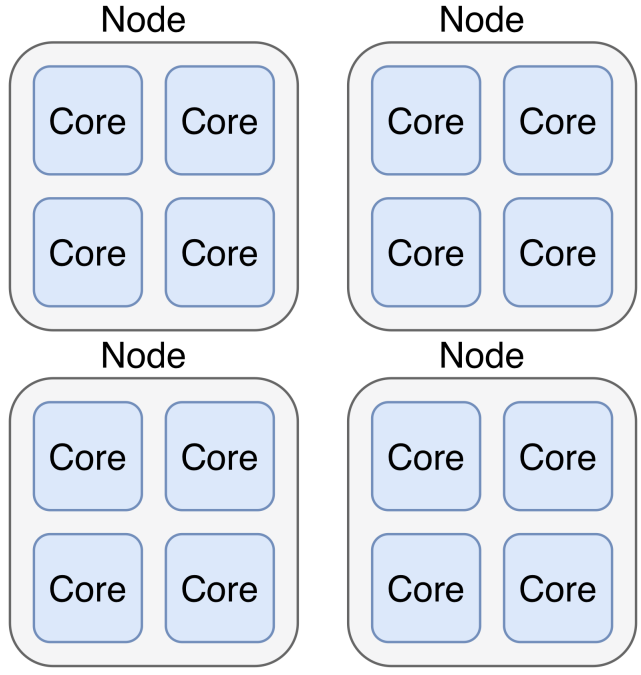
Takuya Fukuoka, Wataru Endo, Kenjiro Taura

The University of Tokyo

はじめに

- 1つのCPUチップの中に非常に多くのコアが搭載される時代
- ノード間とノード内の並列を効率的に組み合わせた並列化の需要
- 新しいノード間ノード内並列のためのシステムである(MPI+myth)を提唱

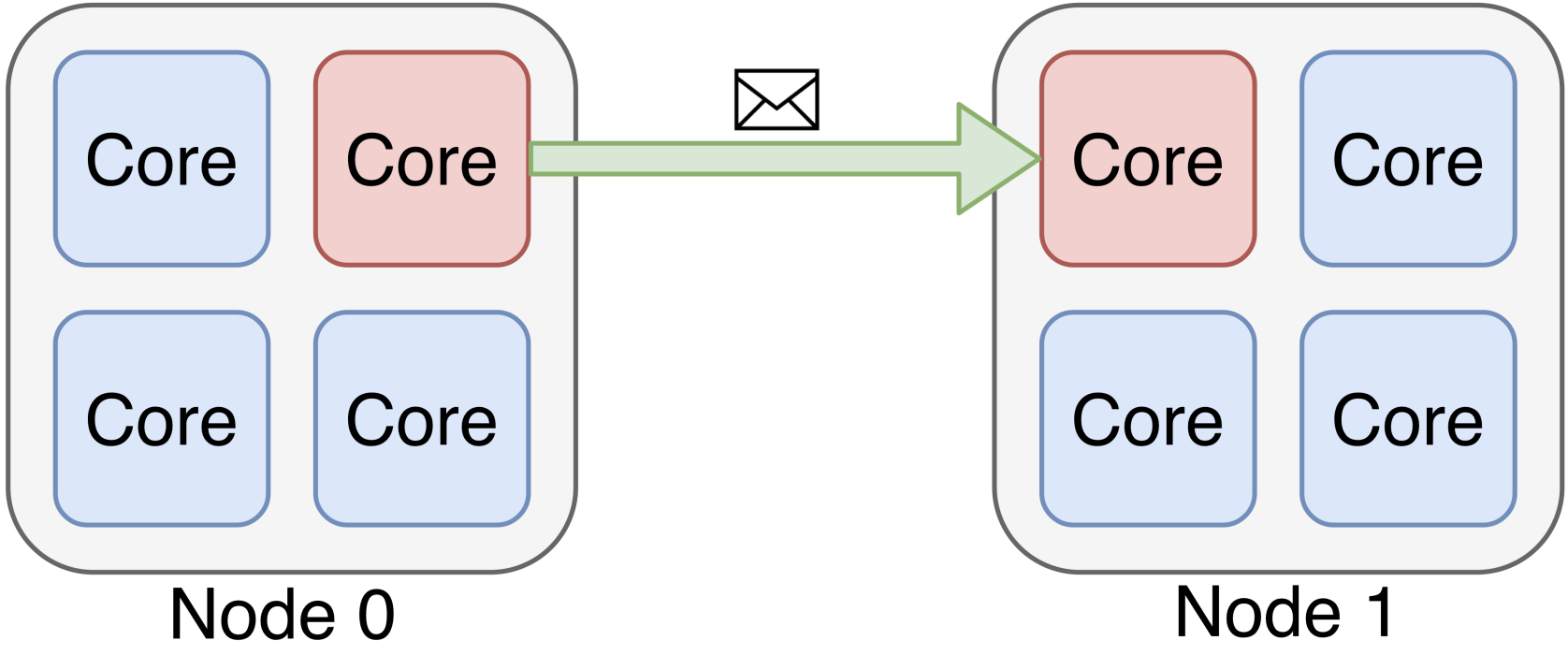
スパコンの構成



スパコン京のシステムラック(https://www.bsc.es/sites/default/files/public/mare_nostrum/hpc-events/4307.pdf)]

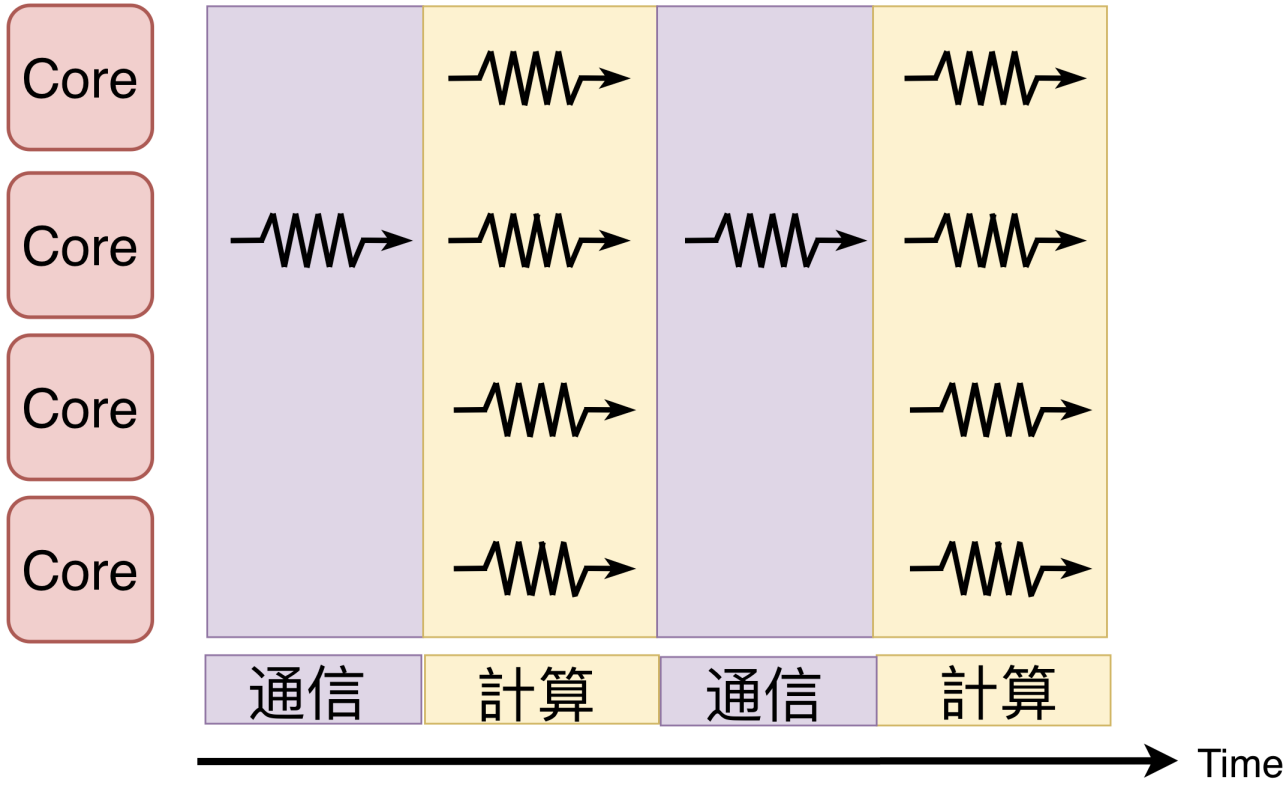
- 現在のスパコンは複数のNodeから構成されている
- Node内に計算資源としてのCPUのCoreが複数存在してメモリを共有
- Node内ではCoreの数だけスレッドを同時に実行できる

MPI (Message Passing Interface)



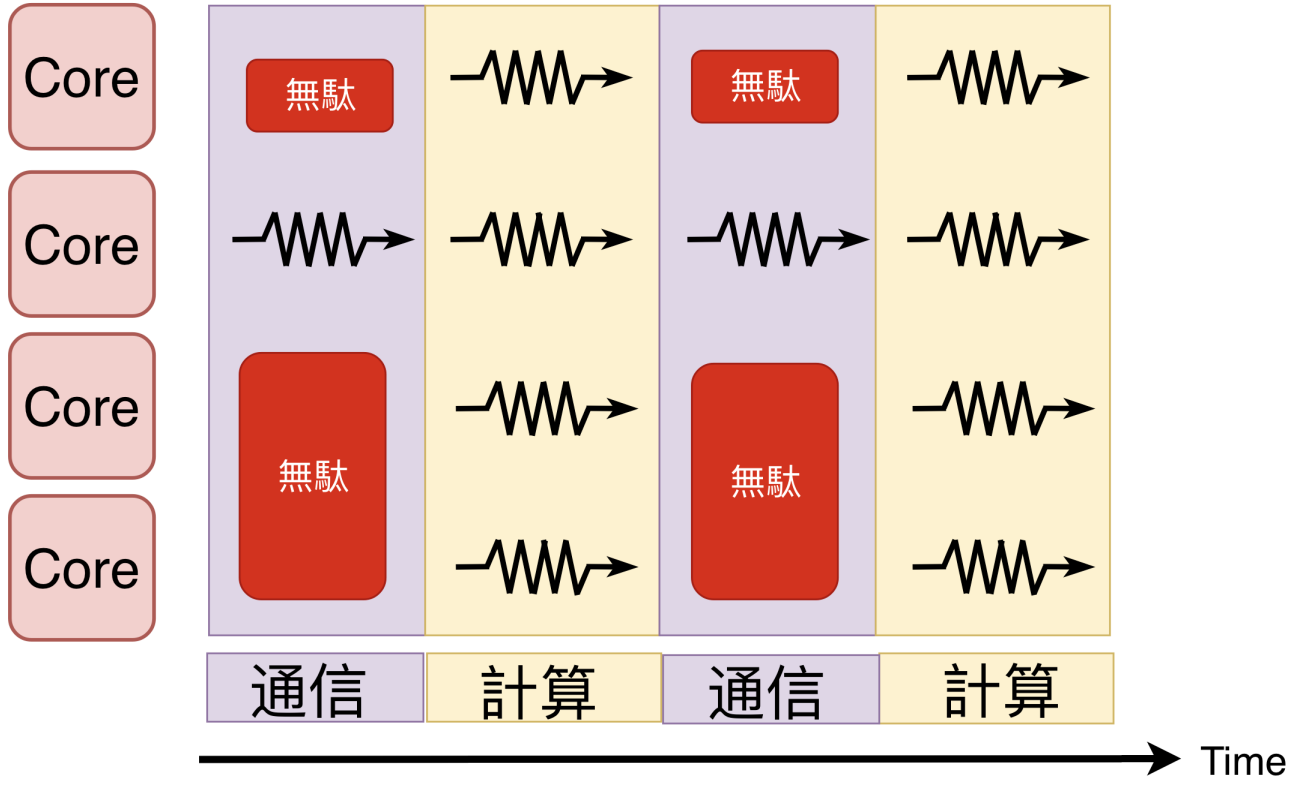
- Node間の通信に用いるインターフェース
- 通信が終了するまでブロックするブロッキングAPIと、命令を発行すると即座に返るノンブロッキングAPIが存在する
- 直感的に書けることからブロッキングAPIの利用が標準的

スパコン上での通常の並列計算



- Node間通信フェーズとNode内計算フェーズが分離
- 通信フェーズではマスタースレッドのみがMPIを発行
- 計算フェイズでは共有メモリを使ってノード内並列

スパコン上での通常の並列計算の問題点

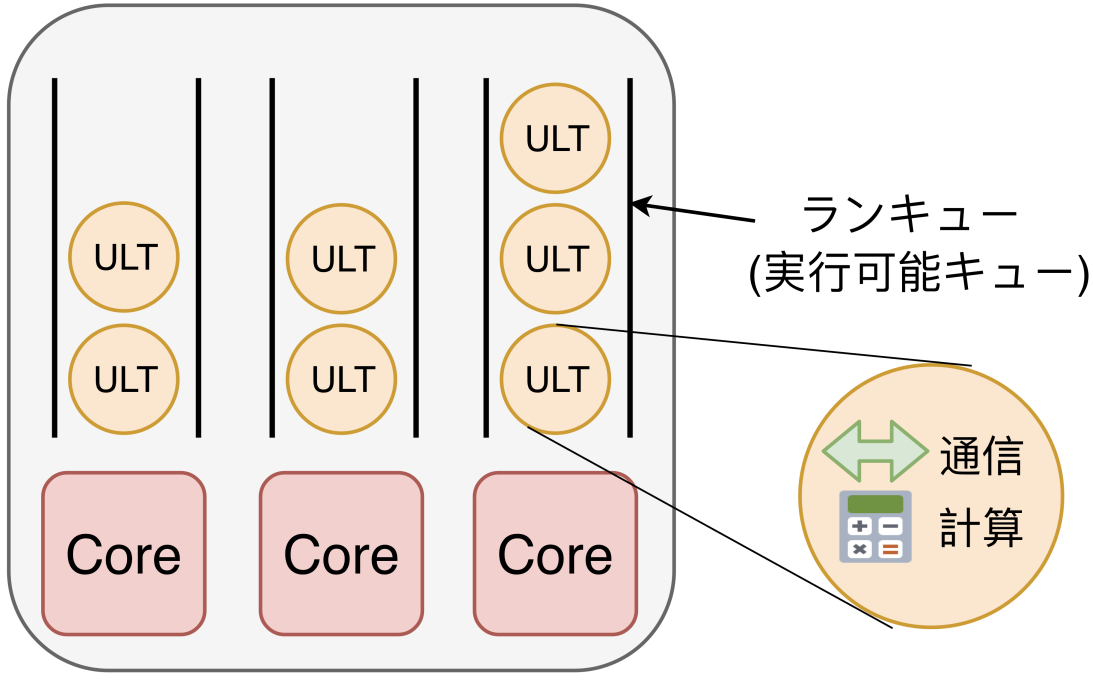


- 通信時にマスタースレッドを担当していないコアの資源の無駄
- これらのコアの有効活用のためには依存性を考慮して計算と通信のオーバーラップを記述する必要があってプログラマの負担が重い

研究の目的

- ノード間通信にMPI、ノード内並列にULT(ユーザーレベルスレッド, 後述）を使用したハイブリッド並列システムであるMPI+mythの提案
- MPI+mythでは、プログラマはULTライブラリであるMassiveThreadsのAPIと、MPIのブロッキングAPIのみを用いてアプリケーションを記述
- プログラマに負担をかけずに通信と計算のオーバーラップを達成

ユーザーレベルスレッド(ULT)

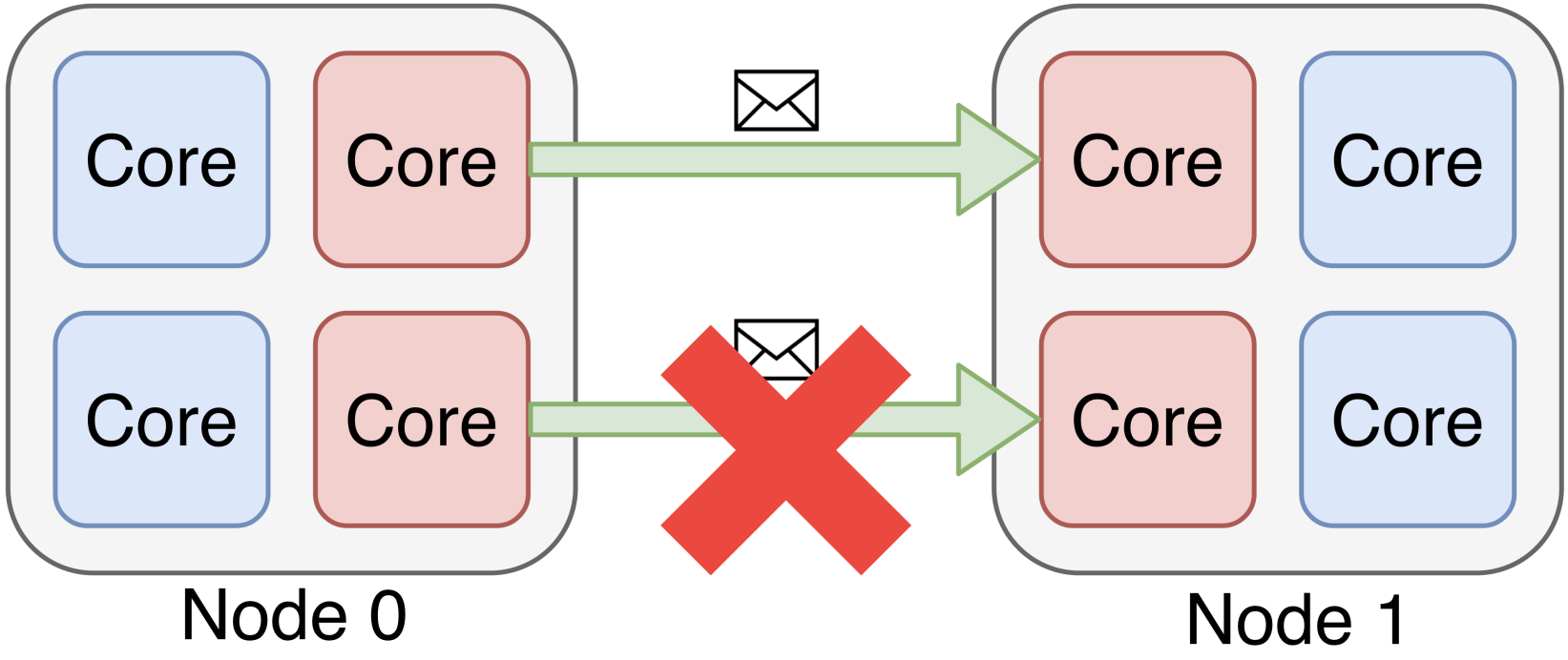


- ULTはOSの(カーネル)スレッド上で動作する仮想的なスレッド
- カーネルスレッドと比較して生成やコンテクストスイッチが2桁ほど高速に可能
- コアごとにULTを格納する1つのランキューを持ちスケジューリングを行う

MPI+mythの実装の2つの特徴

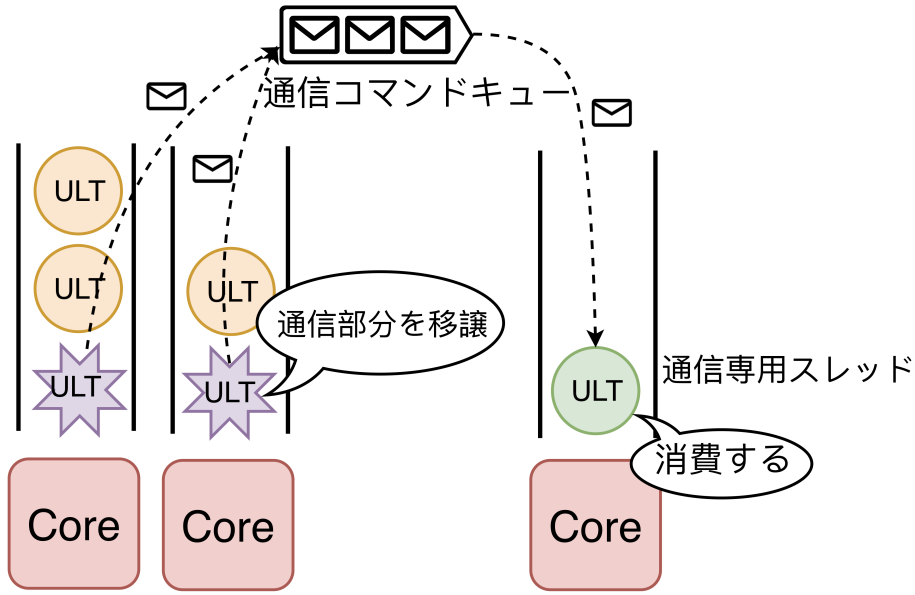
- ソフトウェアオフローディングを導入することでMPIのマルチスレッドモードを回避して性能向上
- ブロックしたULTをランキューから外すスケジューリングを使用することで効率的な負荷分散を達成

MPIのマルチスレッド呼び出しの制限



- 通常のモードでは、MPIをマルチスレッドで同時に呼び出すことはできない
- マルチスレッドでMPIを呼び出すにはそれ専用のモード(MPI_THREAD_MULTIPLE)を使う必要があるが、内部で複雑で重い排他制御を用いていて性能が非常に悪い

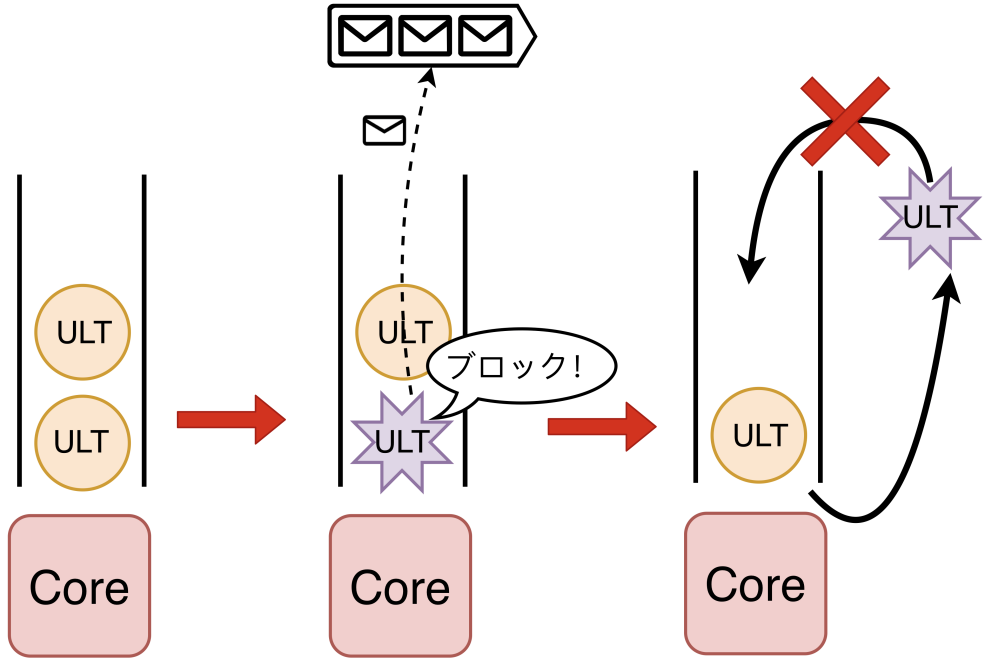
MPI+mythの特徴１：ソフトウェアオフローディング



- MPI呼び出しを通信専用スレッドに移譲してまとめて処理することで、MPIのマルチスレッドモードを回避する手法 [1]
- 通信専用スレッドとコマンドキューの導入が必要
- ULTが通信を発行すると、通信コマンドキューに通信の関数の名前と引数をまとめたものを挿入

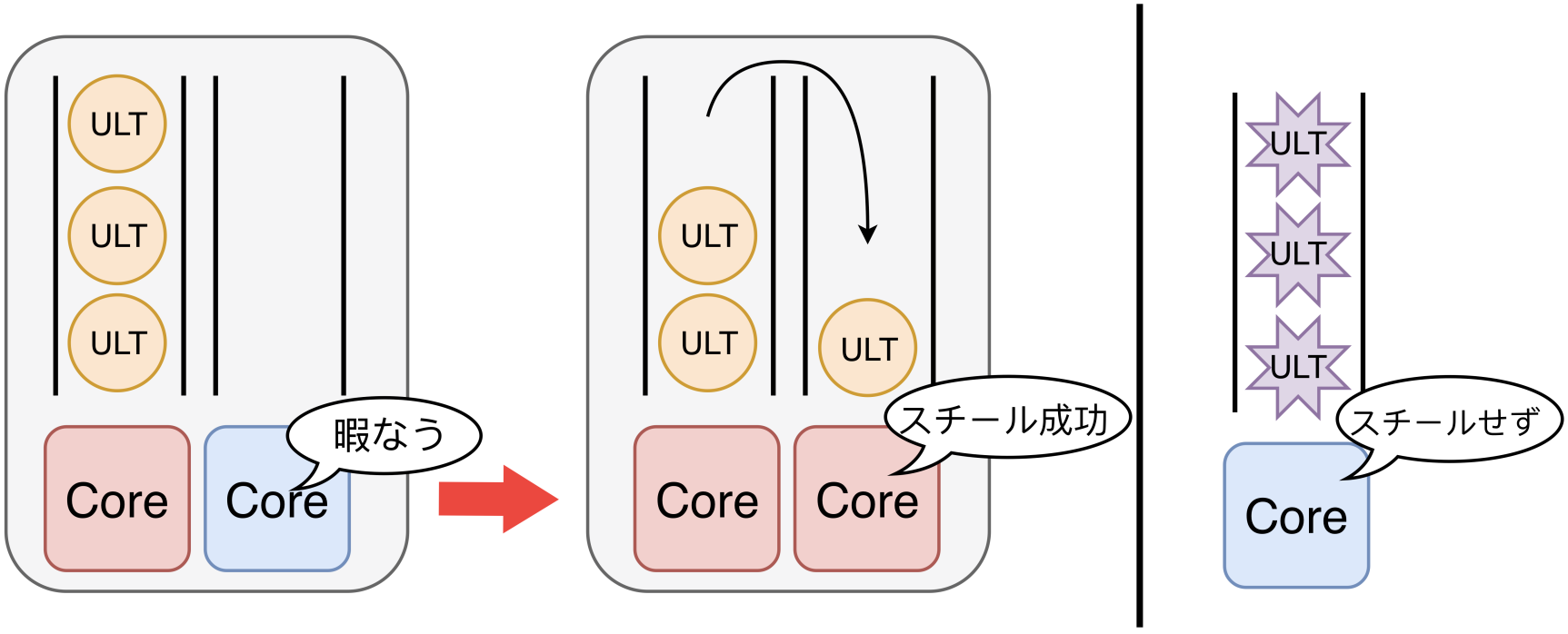
1. Vaidyanathan, K. et al. (2015). Improving Concurrency and Asynchrony in Multithreaded MPI Applications using Software Offloading - SC'15.

MPI+mythの特徴２：ランキューから除外するスケジューリング



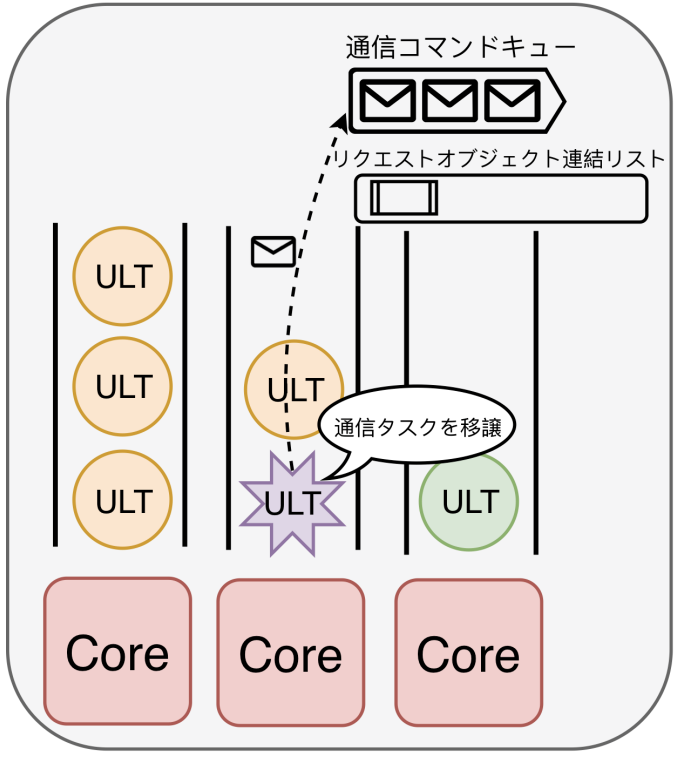
- 通信でブロックしたULTをランキューから除外する手法
- 通信が終了したら即座にランキューに戻す
- 従来のランキューに戻す手法と比較して、負荷分散を効率的に行うことができる

従来の手法で負荷分散がうまく働かない原因



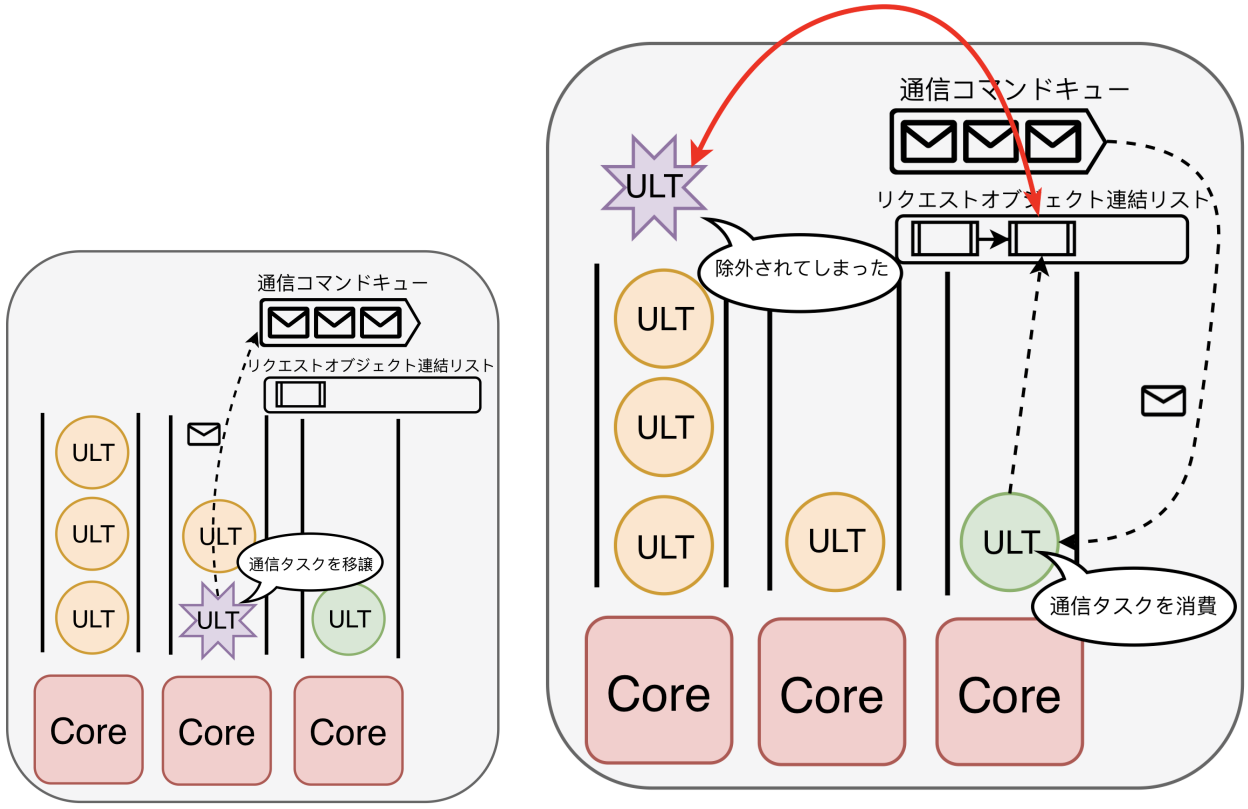
- ULTライブラリは処理するべきULTがキューになくなると、他のコアからULTを盗む(スチールする)機能がある(左図)
- ブロックしたULTでランキューが埋まると、スチール機能がうまく機能しない (右図)

MPI+mythの実際の流れ (その1)



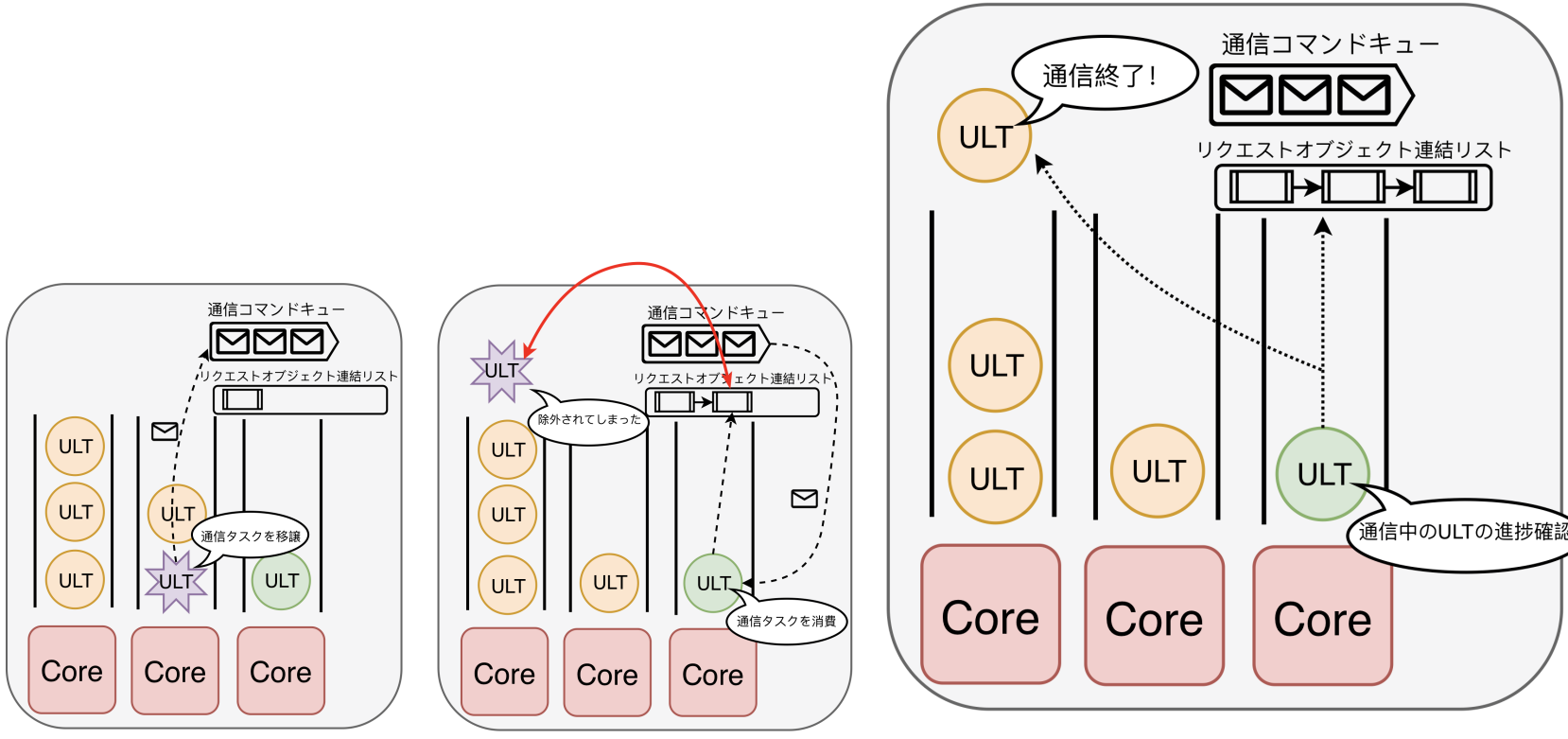
- あるULTが通信を発行すると、通信命令を発行する代わりにコマンドキューに通信呼び出しを入れる
- その後、ULTはランキューから外される

MPI+mythの実際の流れ (その2)



- 通信専用スレッドが通信をコマンドキューから取り出して対応するノンブロッキングAPIを実行
- ブロックしたULTを通信進捗管理用のオブジェクト(リクエストオブジェクト)と紐づける

MPI+mythの実際の流れ (その3)



- それと同時に、通信専用スレッドは発行した通信の進捗を確認する
- 通信が終了していたらULTをランキューに戻す

関連研究 (1)

- HCMPI (MPI+Habanero-C) [2]
  - async句でタスク(ULT)を生成して、finish句でタスクの同期を取る
  - MPI関数は1つのタスク(ULT)として切り出して、通信専用スレッドに処理させる
  - 提案システム(MPI+myth)と異なり、明示的な同期処理が必要

2. S. Chatterjee, et al. (2013) Integrating Asynchronous Task Parallelism with MPI - IPDPS'13.

関連研究 (2)

- MPI+ULT [3]
  - 本研究と同様にMPIとULTを組み合わせて通信と計算のオーバーラップに焦点
  - シングルカーネルスレッドでのULTの使用に焦点

3. H. Lu, et al. (2015). MPI+ULT: Overlapping communication and computation with user-level threads - HPCC'15

関連研究との比較

	MPIのマルチスレッドモードのオーバーヘッド	明示的な同期処理
MPI/SMPSs [4]	サポートせず	不要
HCMPI	なし	必要
MPIQ [5]	あり	不要
MPI+ULT	あり	不要
MPI+myth	なし	不要

4. V.Marjanović, et al. (2010) Overlapping Communication and Computation by Using a Hybrid MPI/SMPSs Approach - ICS

5. D. Stark, et al. (2014) Early Experiences Co-Scheduling Work and Communication Tasks for Hybrid MPI+X Applications

評価

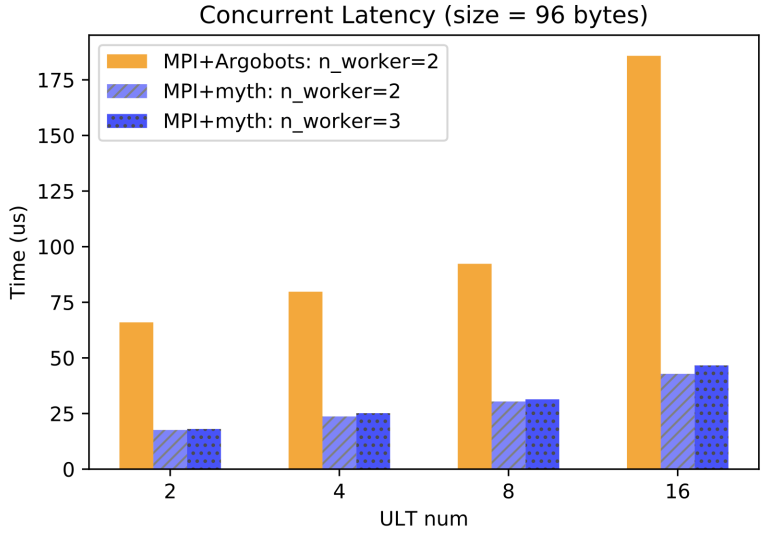
- マイクロベンチマークで既存の並列との比較
  - MPI+Argobots [6] (MPI+ULTの公開実装）との比較
  - MPI+Pthread+Offloadingとの比較
- 通信と計算のオーバーラップの評価
- 既存のスケジューリング手法との比較
- アプリケーション(miniFE)での評価

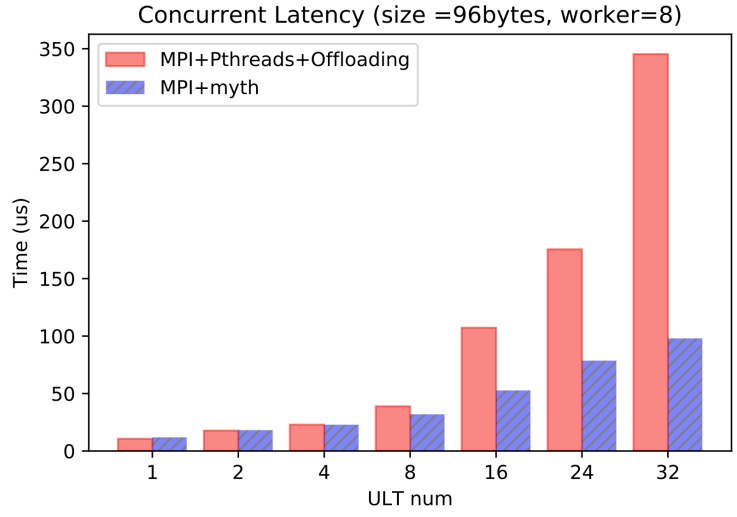
6. https://wiki.mpich.org/mpich/index.php/MPI+Argobots

実験環境

システム	Reedbush-U
インターコネクト	InfiniBand EDR 4x (100 Gbps)
プロセッサ名	Intel Xeon E5-2695v4 (Broadwell-EP)
プロセッサ数(コア数)	2 (36)
メモリ	256 GB

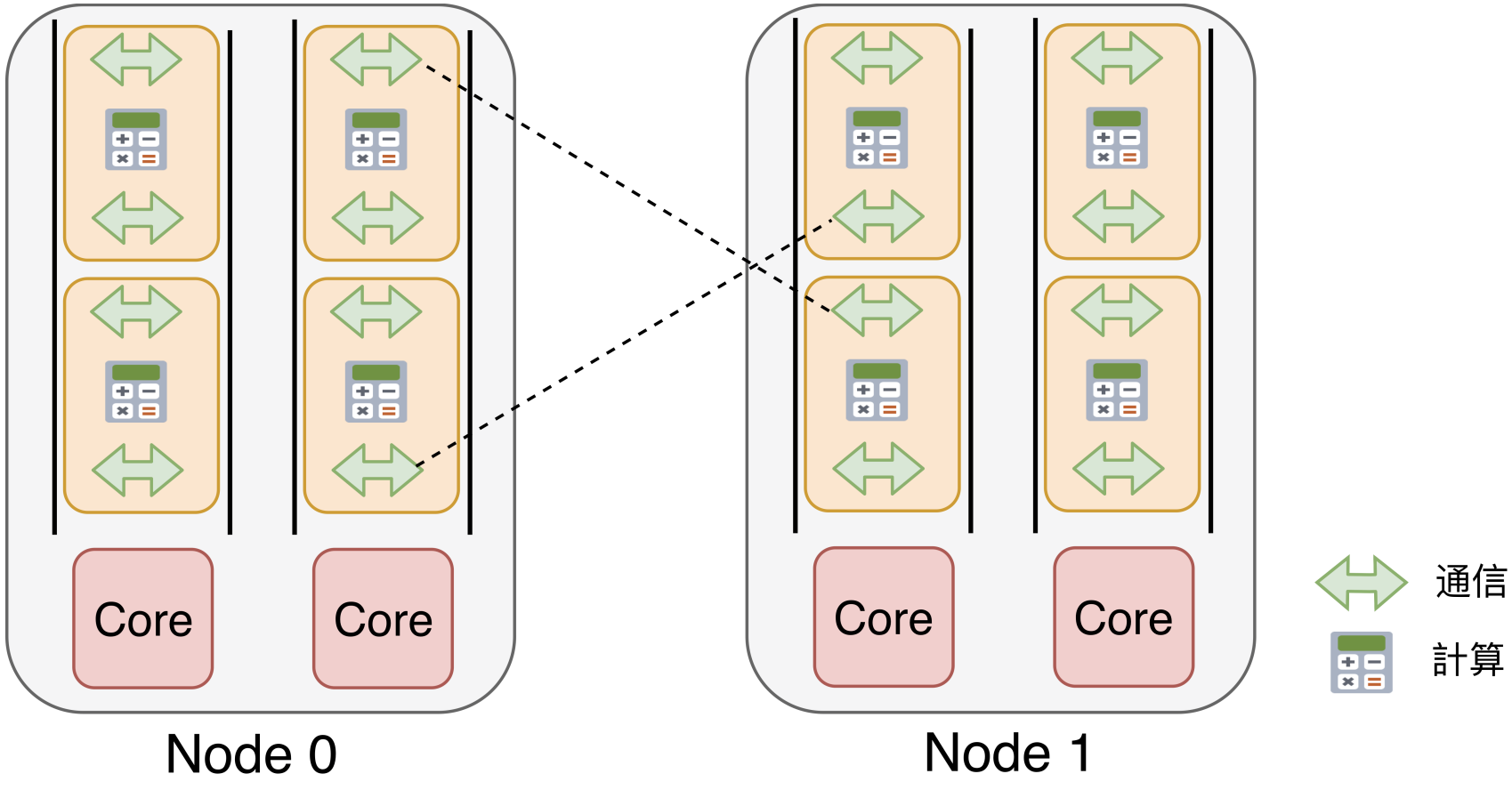
既存の並列との比較（マイクロベンチ)





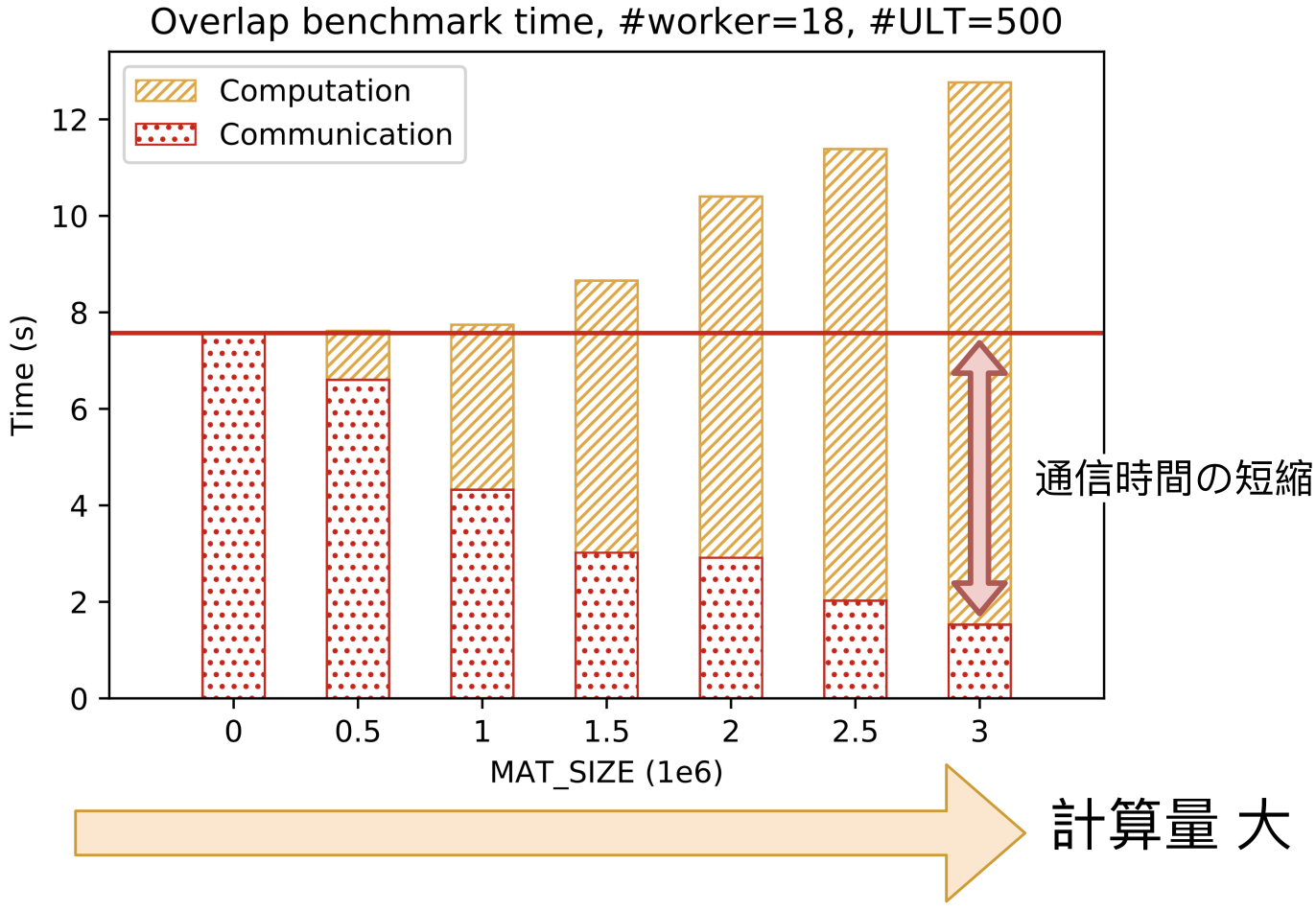
- 提案システムと、MPI+Argobots, MPI+Pthread+Offloadingという2つの既存の並列システムとConcurrent Latencyで比較
- MPI+Argobotsとの比較では、 ソフトウェアオフローディングでのMPIのマルチスレッドモードの回避によって性能が向上
- MPI+Pthreads+Offloadingとの比較では、カーネルスレッドよりULTの方がスケジューリングが軽量であることから性能が向上

Overlapベンチマーク



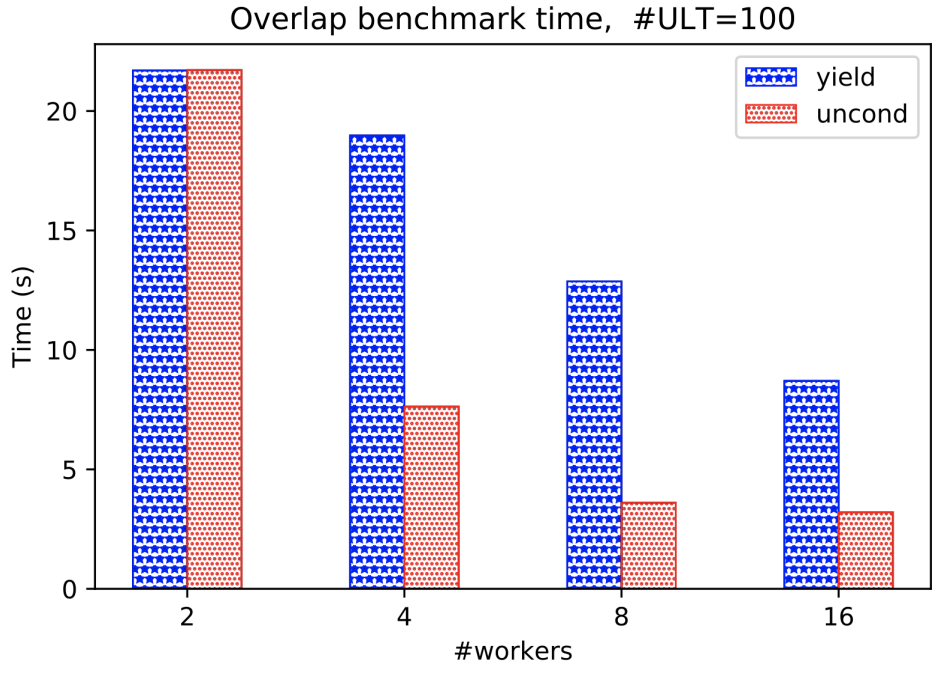
- 2つのMPIプロセスを立ててお互いに通信を行う
- 1つのULTが通信、計算、通信の順序で実行
- 通信にはブロッキング通信関数を使用

Overlapベンチマーク



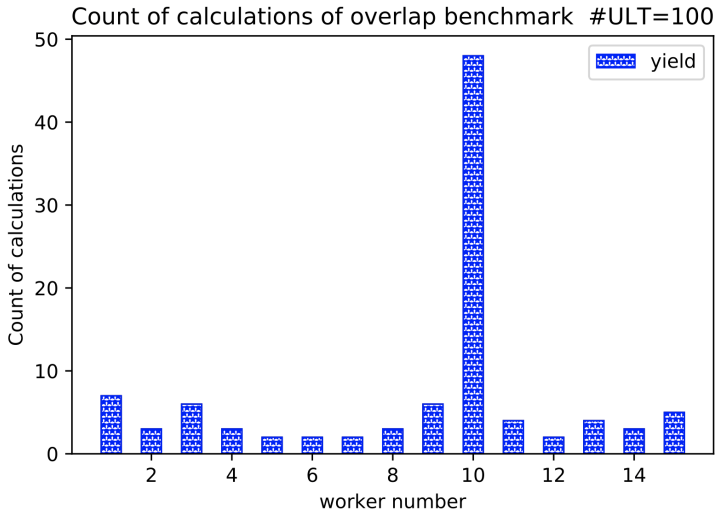
- 通信の分量を固定して計算の分量を変化させる
- 実質的に通信にかかる時間が短くなり、オーバーラップを達成

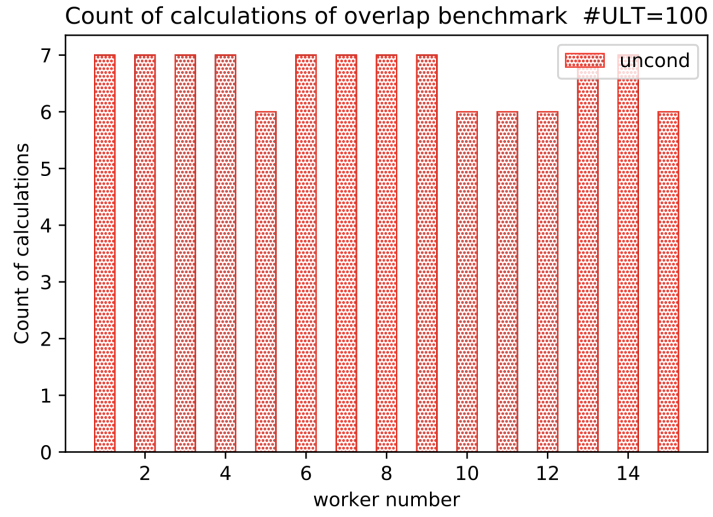
スケジューリング手法の比較



- 青はブロックしたULTをランキューに戻す場合で、赤はブロックしたULTをランキューに戻さない場合
- 戻さない場合の方が性能が良い

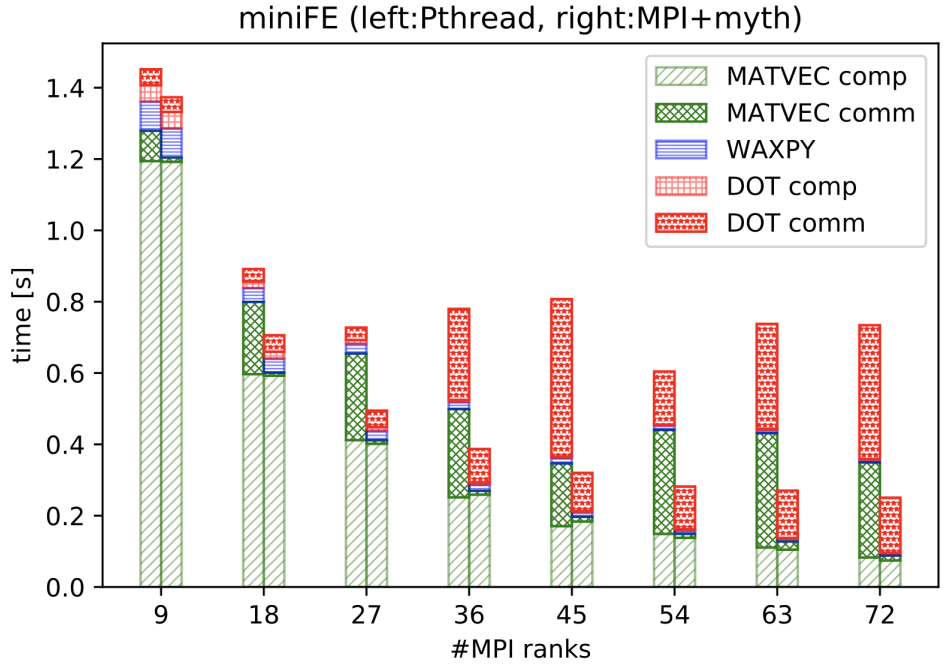
計算部分の負荷分散の様子





- それぞれのコアがULTの計算部分を処理した個数をカウント
- ランキューに戻す場合には、負荷分散がうまくいっていない

既存の並列との比較（アプリケーション)



- 有限要素法を用いたminiFE [1]で評価
- 隣のノードとのデータの交換を、交換すべきノードの数だけスレッドを立てて行うようにする
- ソフトウェアオフローディングの使用でMPIのマルチスレッドモードの使用回避によって、通信時間の短縮を実現した

1. https://github.com/Mantevo/miniFE

評価のまとめ

- 提案システムが既存のシステムと比較してConcurrent Latencyが小さいことを示した
- オーバーラップベンチマークで、MPI+mythで通信と計算のオーバーラップが可能であることを示した
- ブロックしたULTをランキューに戻さないスケジューリングがそうでない場合と比較して負荷分散の面で優れていることを示した
- アプリケーションの評価を通して、既存の並列の手法と比較してMPI+mythの方が性能が優れていることを示した

結論

- プログラマに負担をかけずに通信と計算をオーバーラップできるノード内ノード間並列の実装として、MPI+mythを提案
- MPI+mythでは、プログラマはULTライブラリであるMassiveThreadsのAPIと、MPIのブロッキングAPIのみを用いてアプリケーションを記述
- MPI+mythの実装面の特徴は、ソフトウェアオフローディングとランキューにブロックしたULTを戻さないスケジューリングの２つである
- マイクロベンチマークとアプリケーション両方においてMPI+mythは既存の並列と比較して性能が優れている