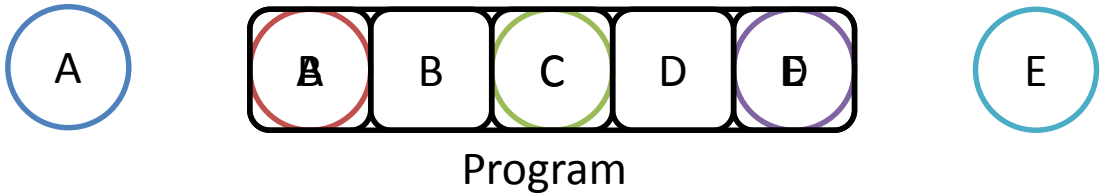
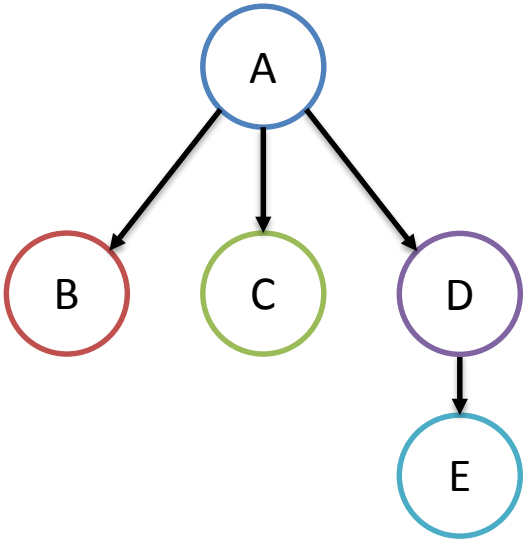
マルチノード・マルチGPU上のコレスキー分解に対するデータドリブン手法

辻田 裕紀
東京工業大学 遠藤研究室
並列コンピューティング

並列プログラミングにおける データ・ドリブン実行





Process 1:

Process 2:

データ・ドリブンアルゴリズムにおいて スケジューリングが重要!

SDPARA: ターゲットアプリケーション

- SDPARA GPU ver.[藤澤 et al. 2011]
  - SDP(SemiDefinite Program、半正定値計画問題)を解くアプリケーション
  - 密コレスキー分解が重要カーネルのひとつ
    - 主にMMやMVM計算
  - → コレスキー分解はGPU向き

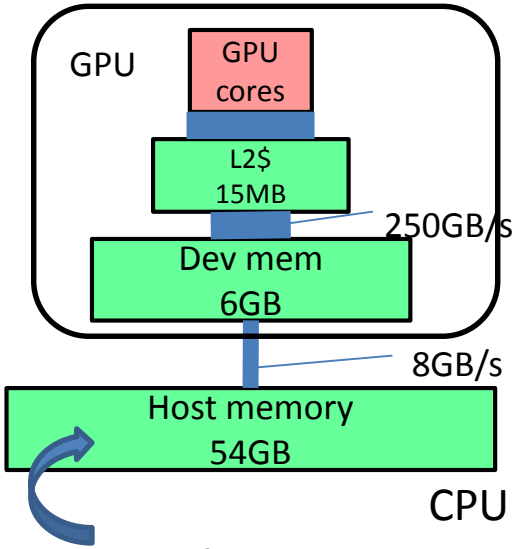
GPGPUとその問題

- General-purpose computing on GPU(GPGPU)
  - GPUは非常に高いバンド幅と計算性能をもつ
  - GPUを汎用計算に利用
- GPUメモリ容量 << CPUメモリ容量

  →GPUメモリ容量超の問題サイズにも対応したい
- CPU-GPUメモリ間でデータ転送をすればよい

CPU-GPU間のPCIe通信がボトルネックに!

TSUBAME 1ノードの
メモリアーキテクチャ



GPUコアにとって
CPUメモリはPCIe(8GB/s)の向こうの
大きなメモリ

モチベーション

- 高性能
  - マルチノード・マルチGPUによる計算
- 大規模な問題サイズにも対応
  - ホストメモリを利用
- より高い性能へ
  - 提案手法によりPCIe通信量を減らす

目的と成果

- 目的
  - CPU-GPU間のデータ転送量をへらすことによりマルチノード・マルチGPUコレスキー分解の計算性能の向上
- アプローチ
  - データドリブンアルゴリズムによるデータ転送量の削減
    - 適切なタスクスケジューリング
    - GPUメモリの再利用性を高めるようなタスク選択
  - コレスキー分解の性質を利用

- 最大で85%までPCIe通信量を削減
- 16ノード上で13.9TFlopsを達成

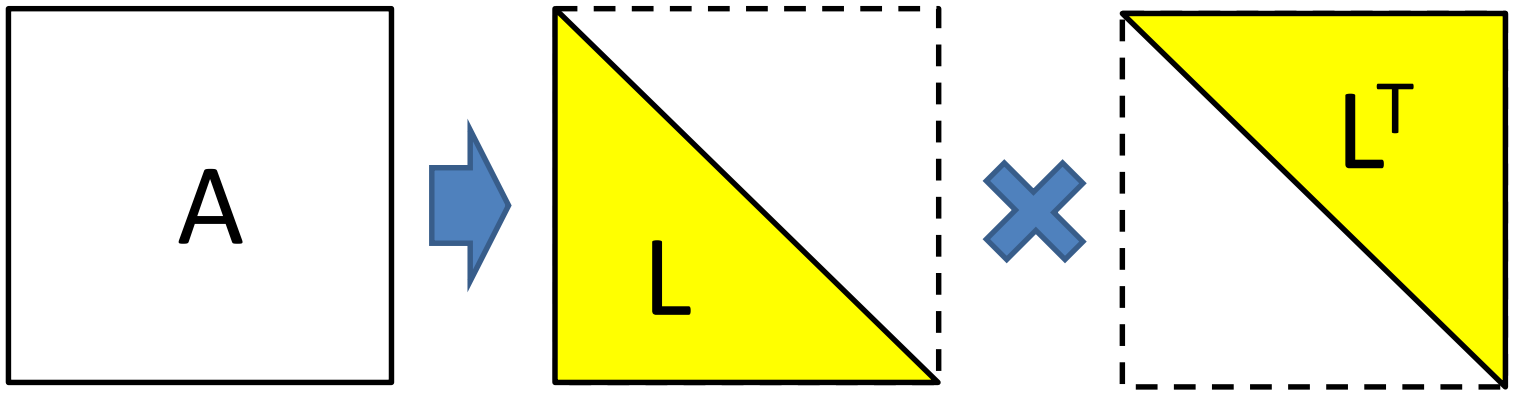
対象アプリケーション

- 密行列コレスキー分解
  - マルチノード・マルチGPUアプリケーション
  - Out-of-coreサポート
  - 入力データはタイルに分割されていると仮定
- タイルは入力データの一部分をもつ
  - タイル内での局所性が良くなるように配列を並び替え
  - 下三角部分のみ持つ

コレスキー分解とは何か？

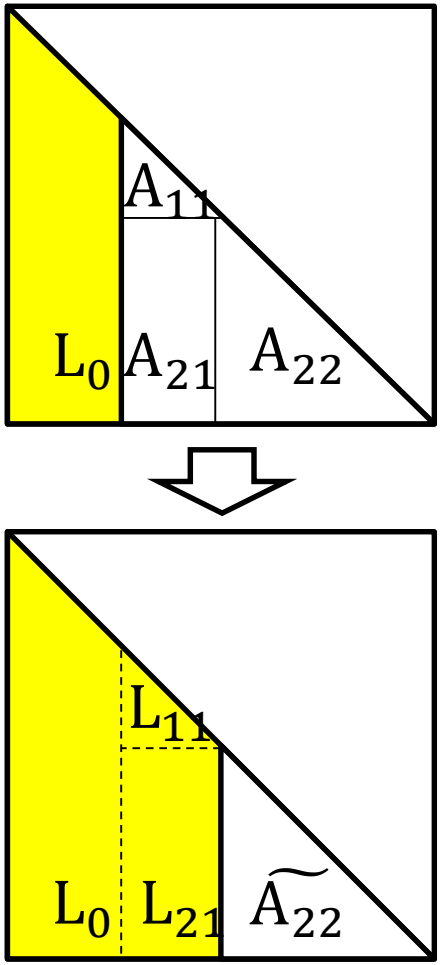
- コレスキー分解は半正定値実対称行列を下三角行列とその転置行列の積に分解する計算
- Statement

$$A=LL^{T}(A\in R^{n\times n})$$



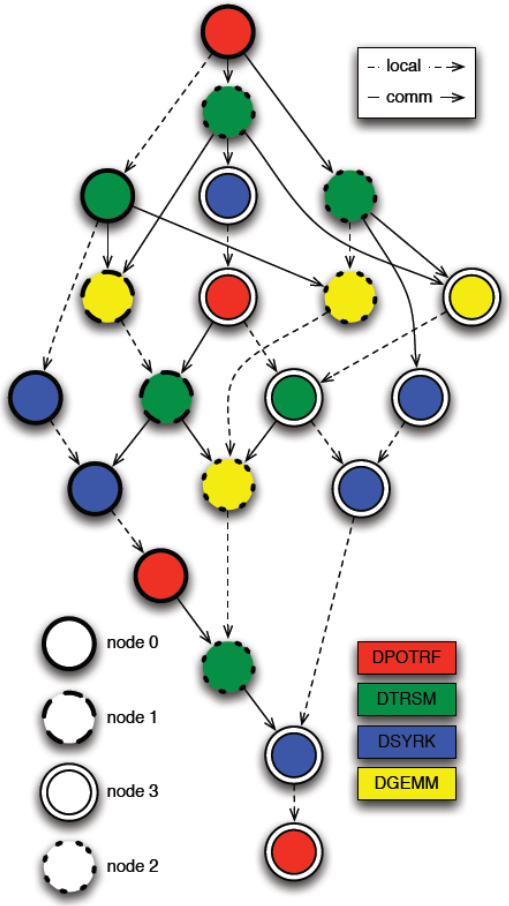
ブロックコレスキー分解

- ブロックコレスキー分解
  - 並列計算手法
  - データをブロック単位に分割
    - 計算はブロックごとに行われる
  - ブロックは“タイル”と同義
  - タイルは二次元ブロックサイクリック分割により割り当てられている。
    - プロセスは割り当てられたデータのみを実行



コレスキー分解における データードリブンアプローチ

- ４つのカーネル
  - DPOTRF, DTRSM, DSYRK, DGEMM
- カーネルをタスクとみなす
- コレスキー分解ではカーネルは互いに依存関係をもつ
  - 依存関係に注意してタスクを管理する必要



The DAG of the Cholesky decomposition [G.Bosilca 2010]

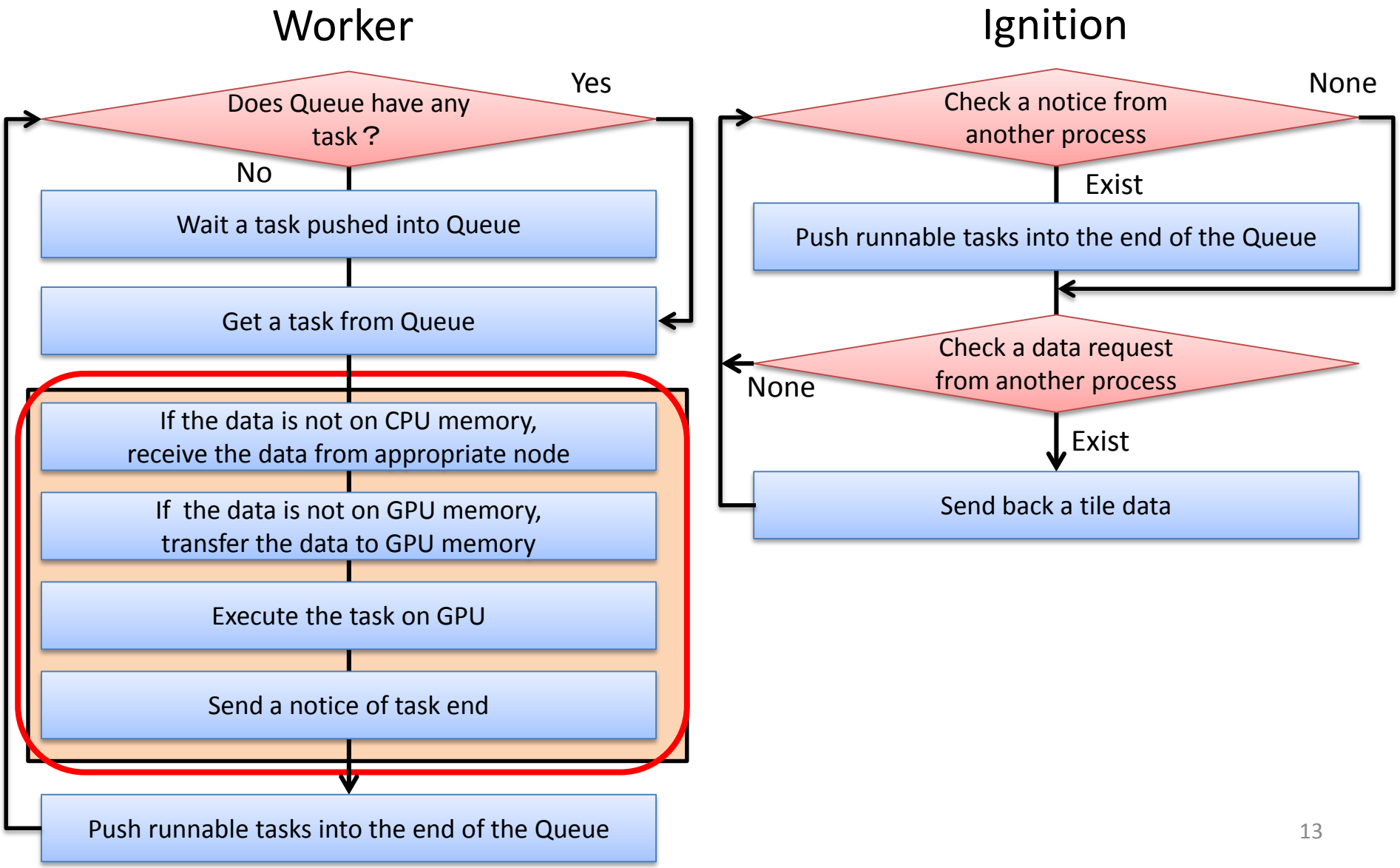
MPIベースのデータドリブン実装

- 行列はMPIプロセス間で分割されていてCPUメモリにおかれる
  - タイルのタスクはオーナープロセスにより実行
  - 他のノードのタイルのデータが必要なときはMPI通信を行う
- データドリブンアルゴリズム
  - GPUメモリの再利用性を考慮したスケジューリング
  - 4つのタスク選択戦略
- MPI通信
  - ポイント・ツー・ポイントの非同期通信
- スワップによるGPUメモリ管理
  - 不必要なデータを選択
  - LRU戦略により選択

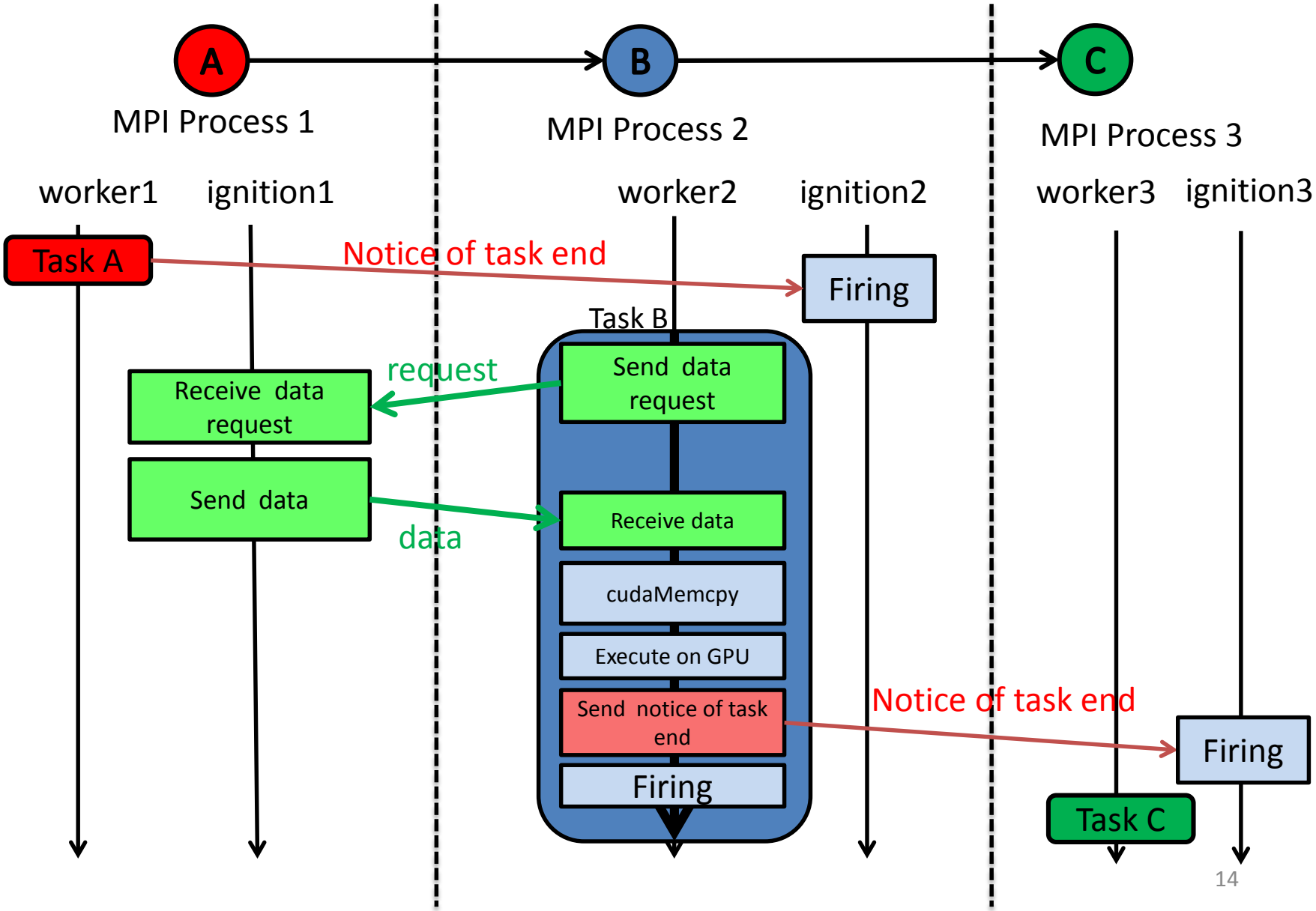
Workerスレッド & Ignitionスレッド

- MPIプロセスは複数のworkerスレッドと一つのignitionスレッド
- Worker
  - タスクを実行する
  - プロセスは計算とPCIe通信やMPI通信のオーバーラップをするために2-3のworkerをもつ
- Ignition
  - 他のノードからのnoticeを監視
  - 他のノードからのデータリクエストを管理
- MPIプロセス上のすべてのスレッドは一つのタスクキューを共有している

スレッドフローチャート



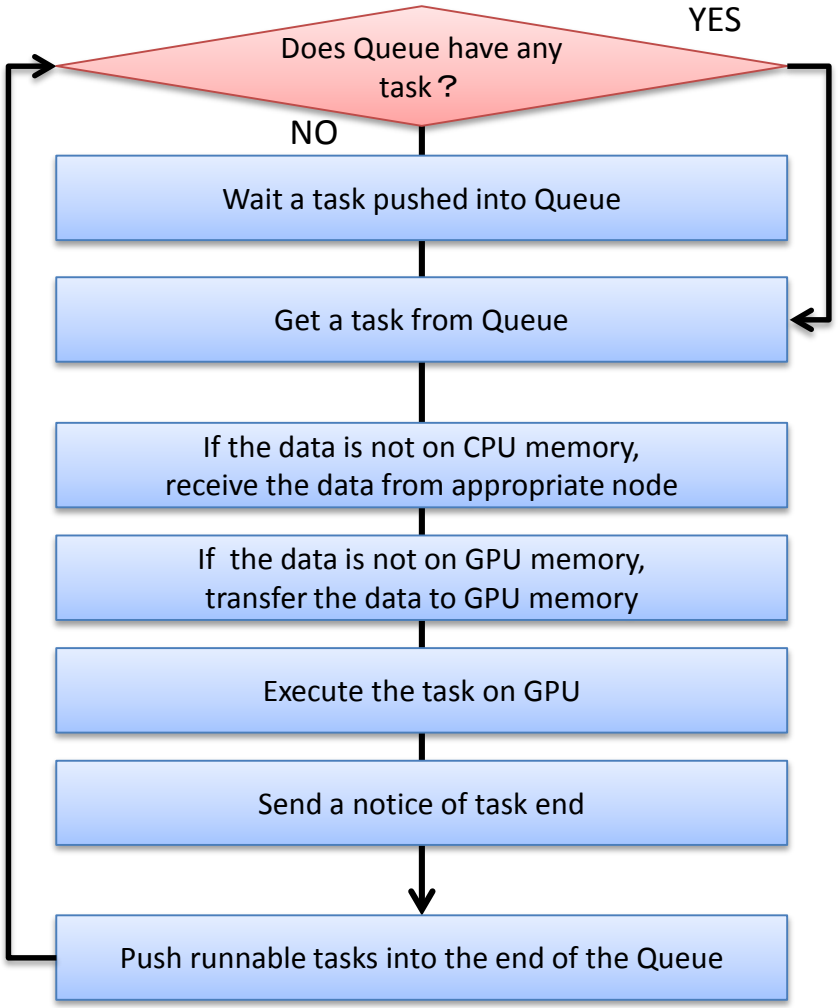
タスク実行パターン


A
B
C
MPI Process 1
MPI Process 2
MPI Process 3
worker1
ignition1
worker2
ignition2
worker3
ignition3
Task A
Notice of task end
Firing
Task B
request
Receive data request
Send data request
Send data
Receive data
data
cudaMemcpy
Execute on GPU
Send notice of task end
Notice of task end
Firing
Firing
Task C

GPUバッファ

- GPUメモリ上のデータはスワップにより管理
  - GPUメモリを最大限利用するため
- タイルにGPUメモリへの紐付け情報を与える
  - 紐付け情報を見ることで冗長な通信を削減する
- どのデータが掃き出されるかにより性能が変わる
  - victimはLRUにより選択

タスク選択戦略



- タイルの情報は利用可能
- DAGのようなタスクフローに関する情報は利用不可能
- 4つの戦略
  - FIFO Strategy
  - LIFO Strategy
  - RANDOM Strategy
  - BYIJ Strategy

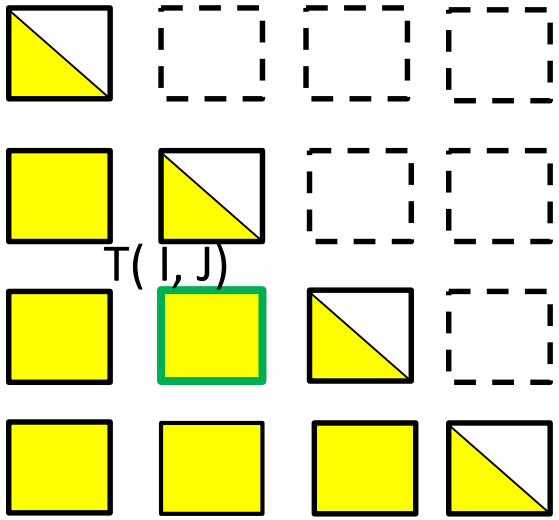
タスク選択戦略

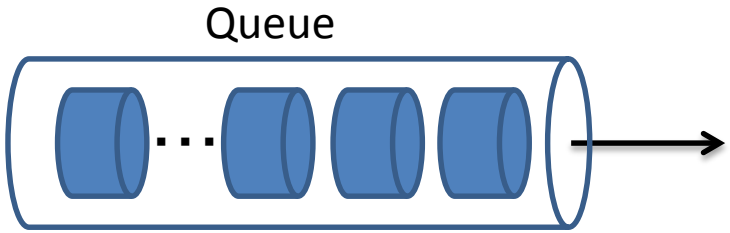
- BYIJ戦略
  - コレスキー分解では同じ行・同じ列にあるデータは強い依存関係をもつ
  - 前のタスクの情報を保持それを利用して次のデータを選択

BYIJ戦略

- 次のタスクは以下のように選ばれる

```
IF ( ( task=Find( T( I ,・) )) != NULL )
        THEN Select( task )
ELSE IF ( ( task=Find( T( ・, J ) )) != NULL )
        THEN Select( task )
ELSE
        Select( QUEUE_HEAD )
```

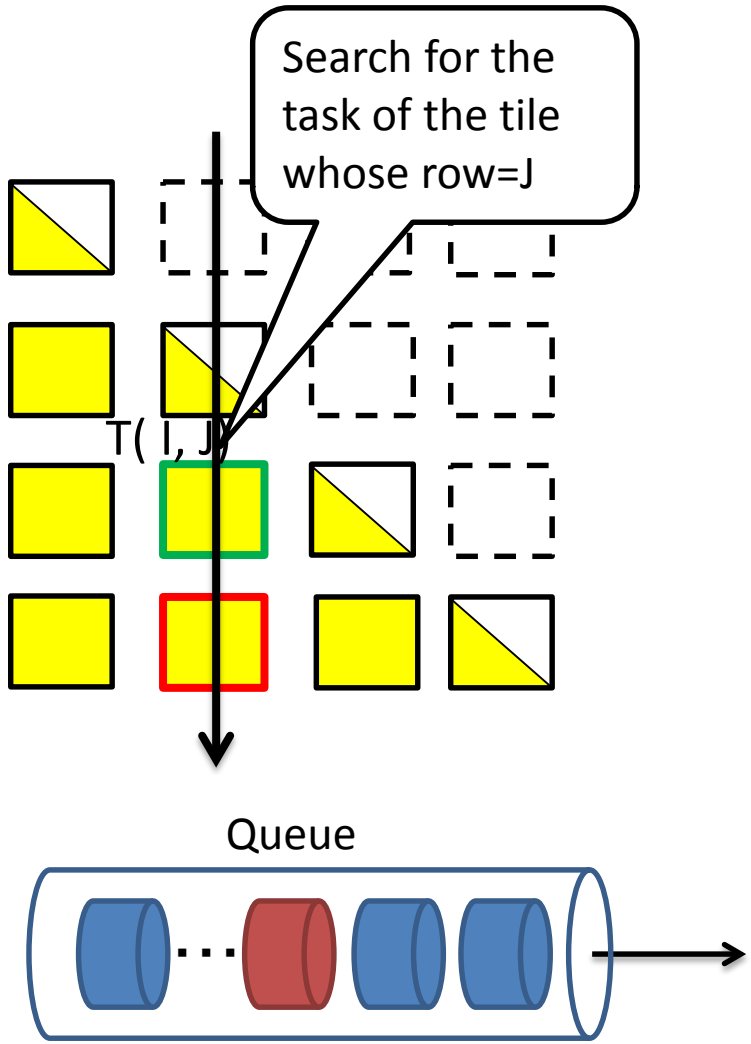




BYIJ戦略

- 次のタスクは以下のように選ばれる

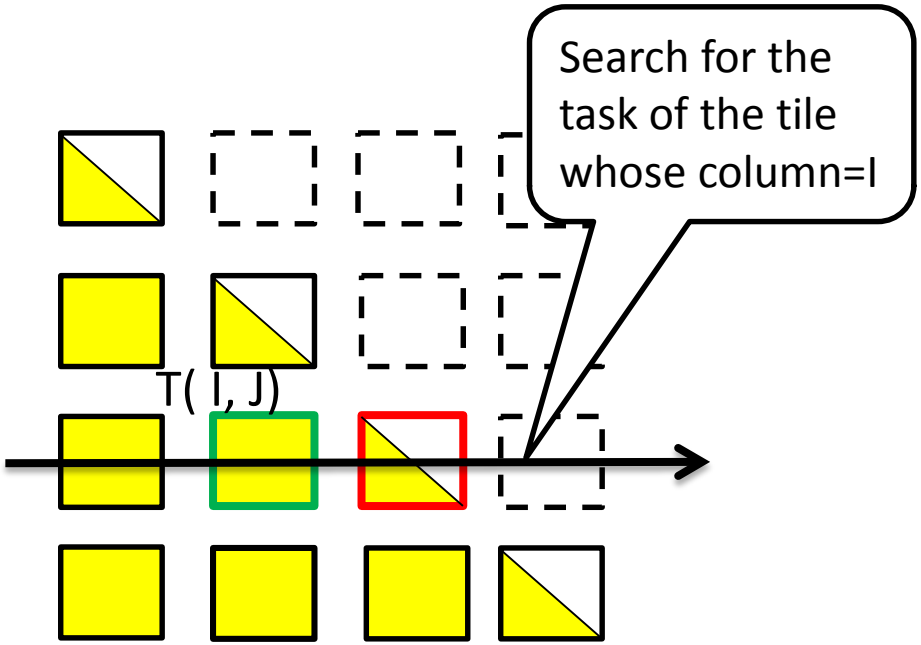
```
IF ( ( task=Find( T( I ,・) )) != NULL )
        THEN Select( task )
ELSE IF ( ( task=Find( T( ・, J ) )) != NULL )
        THEN Select( task )
ELSE
        Select( QUEUE_HEAD )
```

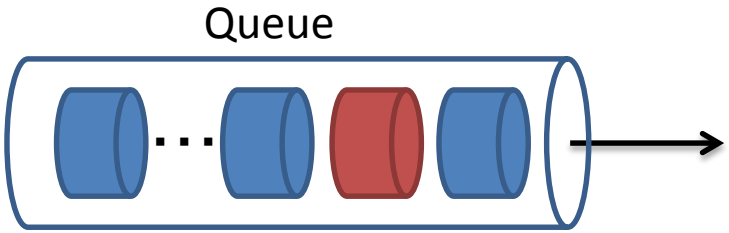


BYIJ戦略

- 次のタスクは以下のように選ばれる

```
IF ( ( task=Find( T( I ,・) )) != NULL )
        THEN Select( task )
ELSE IF ( ( task=Find( T( ・, J ) )) != NULL )
        THEN Select( task )
ELSE
        Select( QUEUE_HEAD )
```

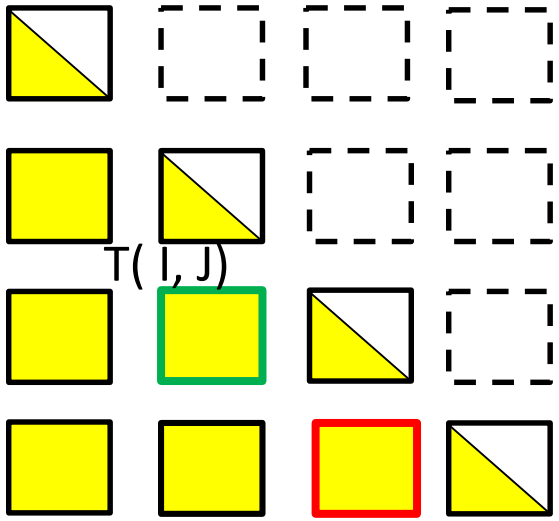


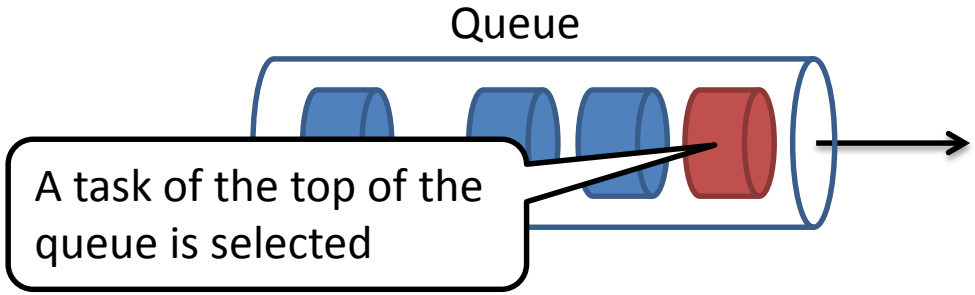


BYIJ戦略

- 次のタスクは以下のように選ばれる

```
IF ( ( task=Find( T( I ,・) )) != NULL )
        THEN Select( task )
ELSE IF ( ( task=Find( T( ・, J ) )) != NULL )
        THEN Select( task )
ELSE
        Select( QUEUE_HEAD )
```





実験環境

- TSUBAME2.5 Thin node
  - 1ノードあたり1GPU
  - 1MPIプロセスあたり3worker
- タイルサイズ:2,048 x 2,048
- GPUメモリ:5,000MiB per a GPU
- NVIDIA CUDA 6.0とCUBLAS 6.0を利用

TSUBAME 2.5 Thin node architecture	
CPU	Intel Xeon 2.93 GHz (6 cores) x 2
CPU memory	54GiB
GPU	NVIDIA Tesla K20X × 3
GPU memory	6GiB

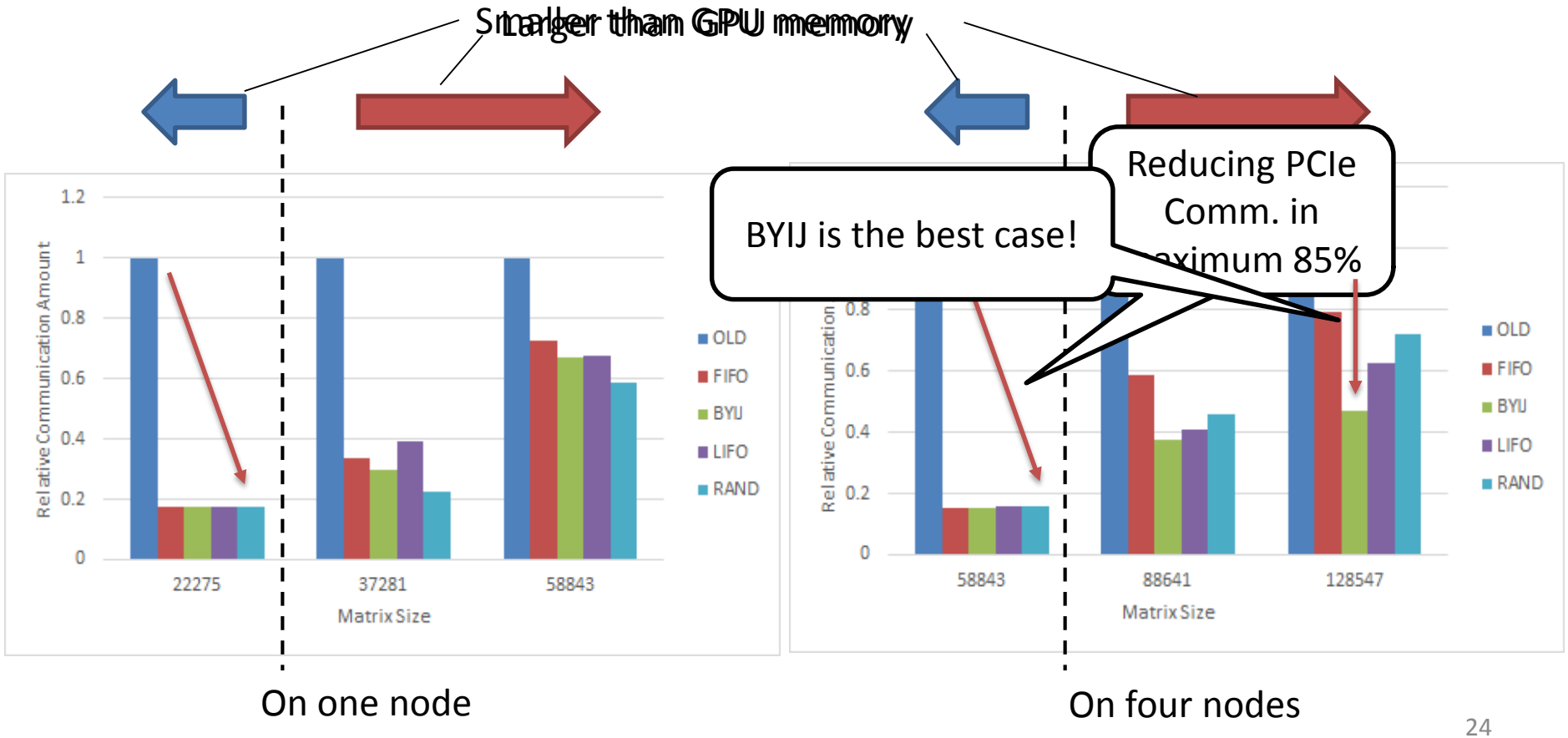
評価実験

- 評価内容
  - PCIe通信量
  - Weak Scaling
  - Strong Scaling
- 比較アルゴリズム
  - 既存のSDPARA GPU ver. (OLD)
    - Scalapackライクのブロックアルゴリズム
    - 各ブロックイテレーションでCPUからGPUへデータを転送する
  - 提案手法(NEW)

    FIFO, BYIJ, LIFO, RAND

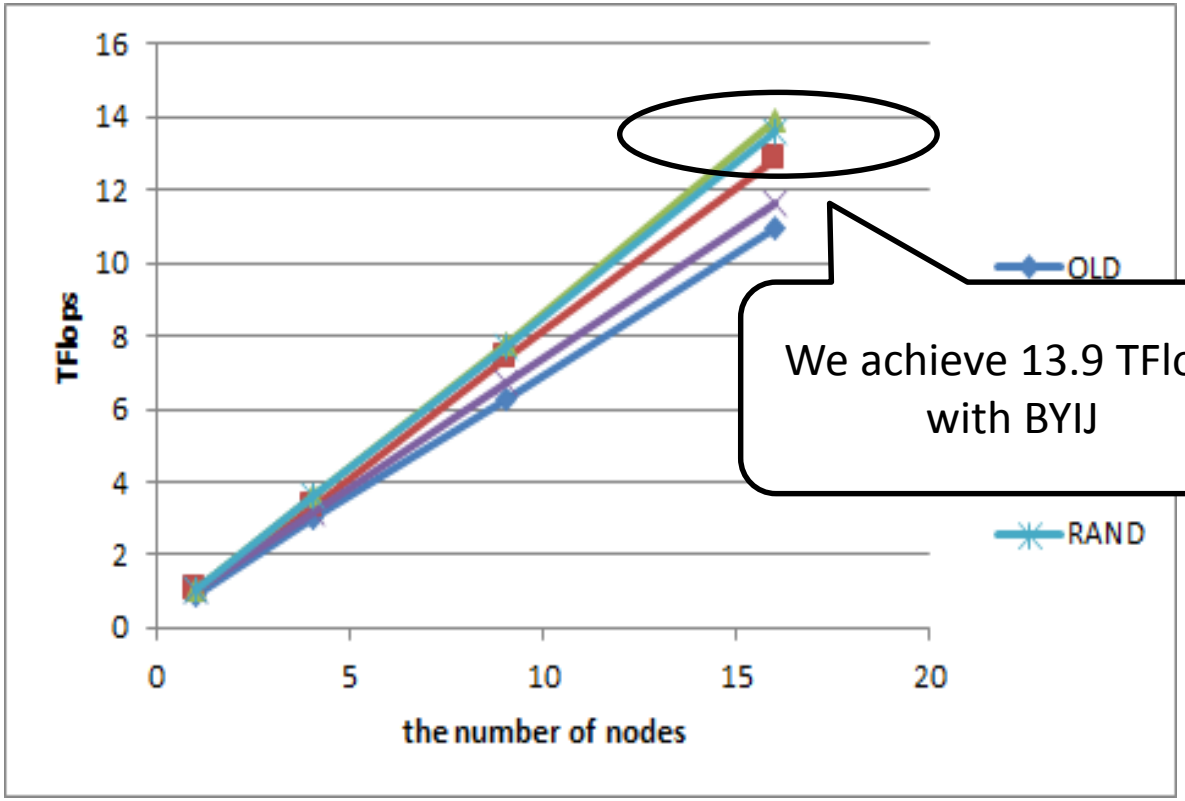
PCIe通信量

- OLDを1としたときのPCIeの相対通信量



Weak Scaling

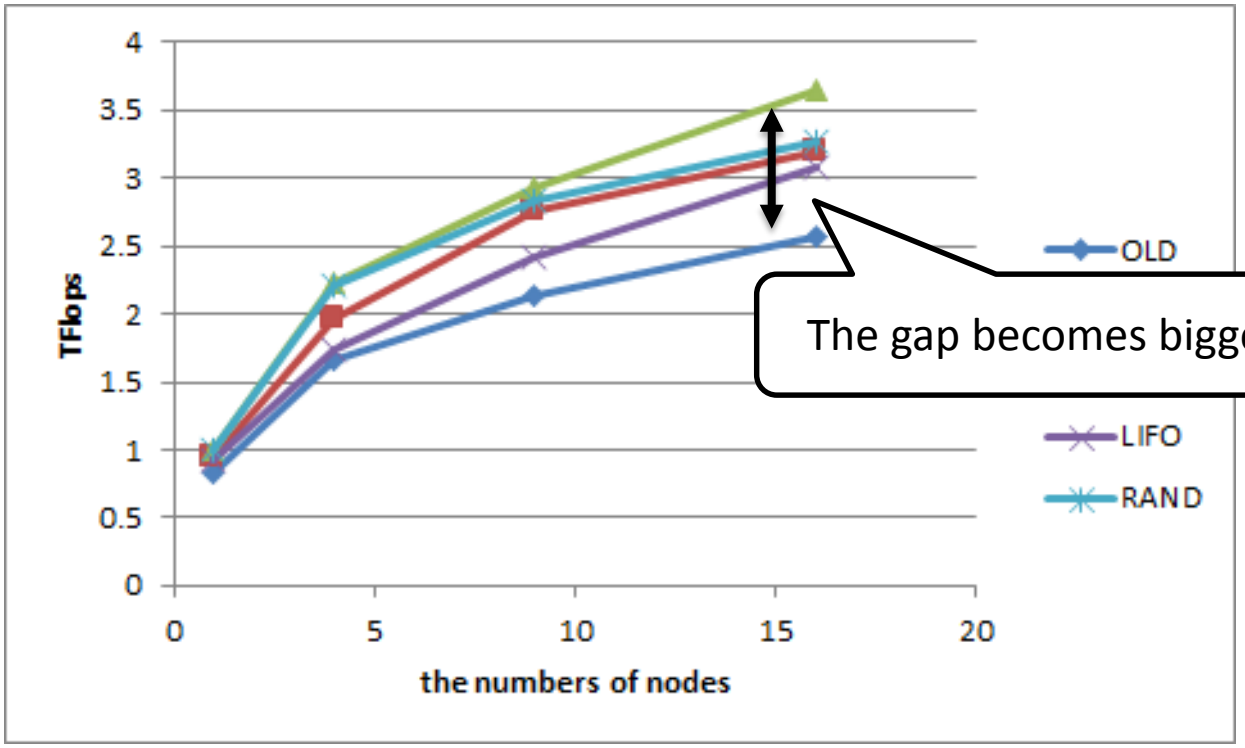
1ノードあたりの行列サイズを固定 (58,000 per node)



- すべての戦略で線形にスケール
- すべてのNEW戦略でOLDより良い性能を観測

Strong Scaling

行列サイズを47,142



- Weak Scalingほどのスケーラビリティはでない

NEW戦略はOLDよりも非常に高い性能を実現

結論

- コレスキー分解にデータドリブンアルゴリズムを適応
  - PCIe通信量を最大で85%削減
  - 16GPUを利用して13.9TFlops達成
    - 16GPU時のピーク性能の66.3%
  - 計算の性質を利用することでよりよい性能を出すことができる

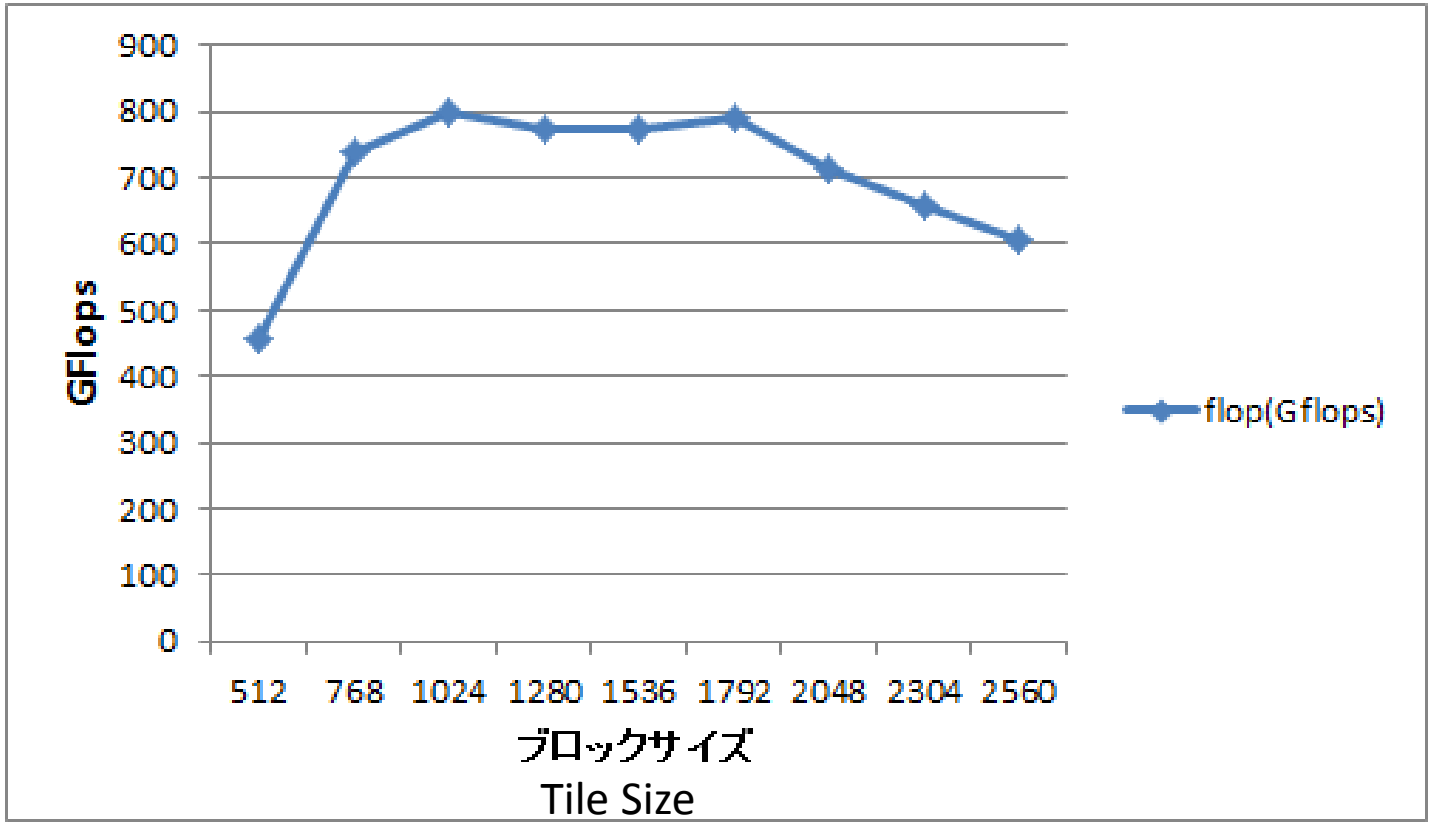
ありがとうございました！

予備スライド

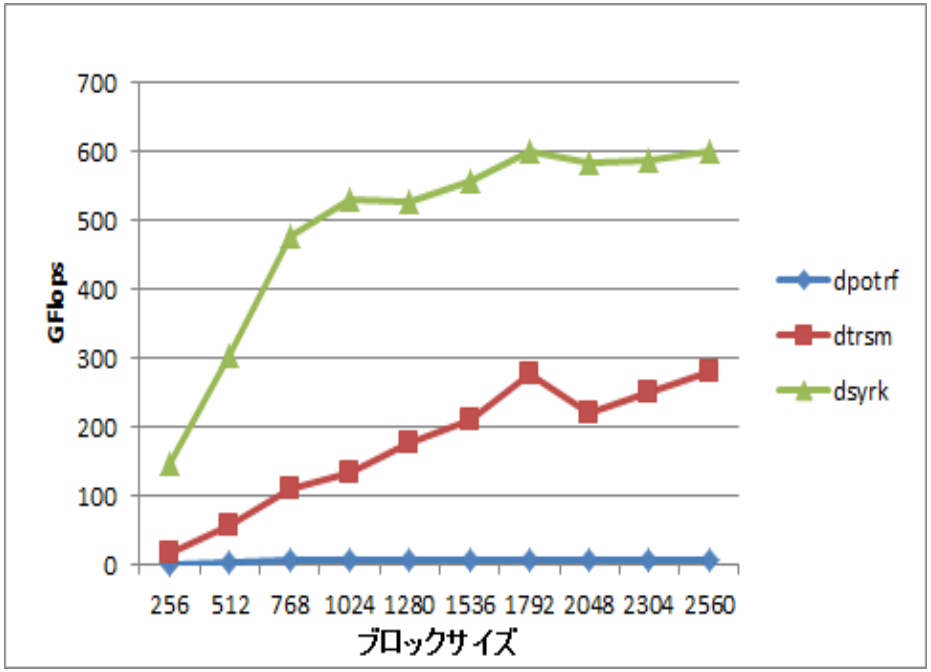
APPENDIX

The Performance Influence of Tile Size

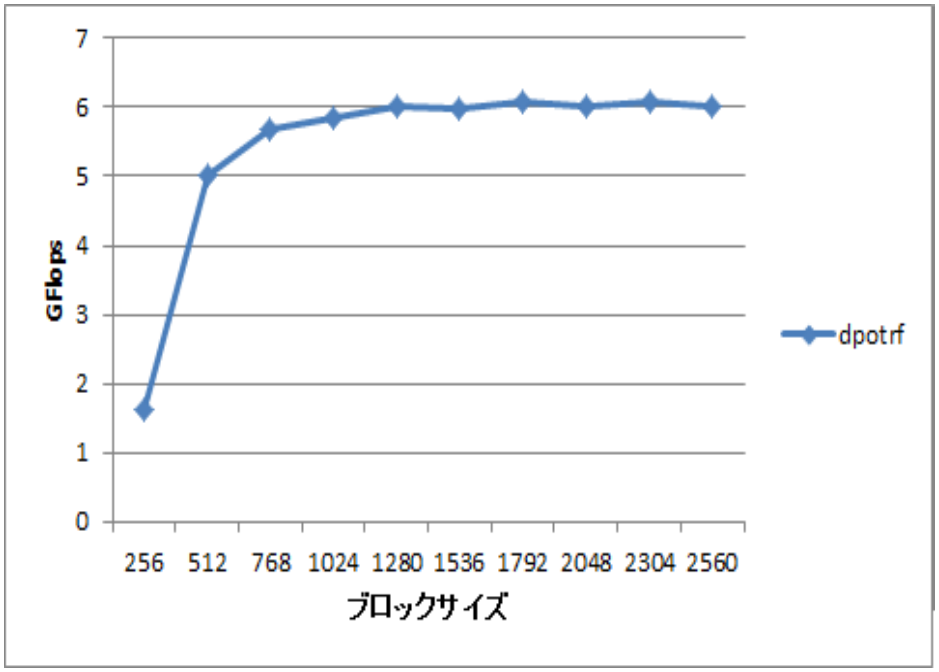
One node
GPU memory size : 5500MiB
Strategy : ByIJ-LRU



The Performance Influence of Tile Size



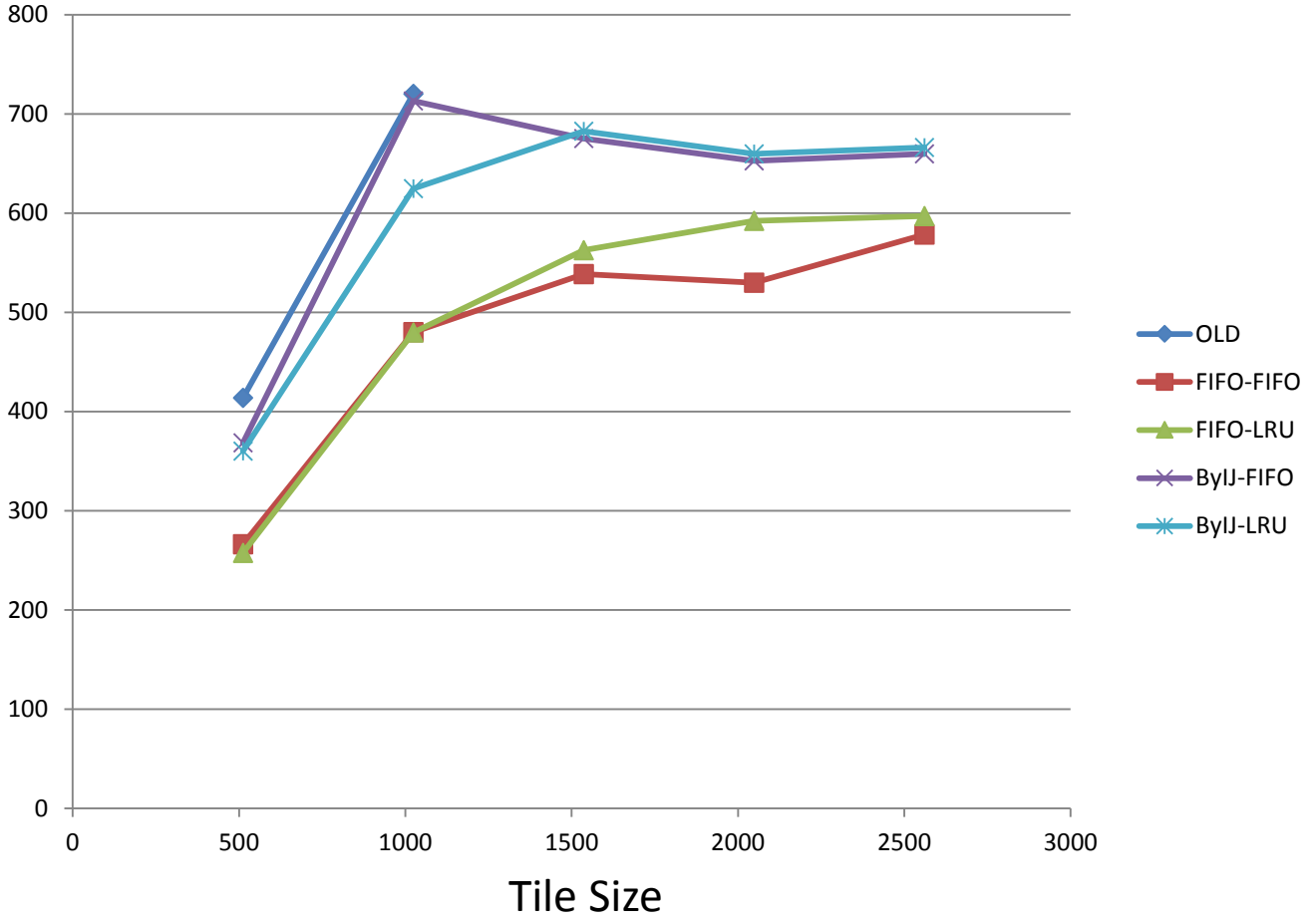
Tile Size



Tile Size

The influence of the granularity of the task

- Matrix size : 61440
- One process
- GPU memory size : 5500MiB



The influence of the granularity of the task

- Matrix size : 122880
- Four processes
- GPU memory size : 5500MiB