Coordinating the Use of GPU and CPU for Improving Performance of Compute Intensive Applications

10M37108 白幡 晃一
数理・計算科学専攻 松岡研究室

出典

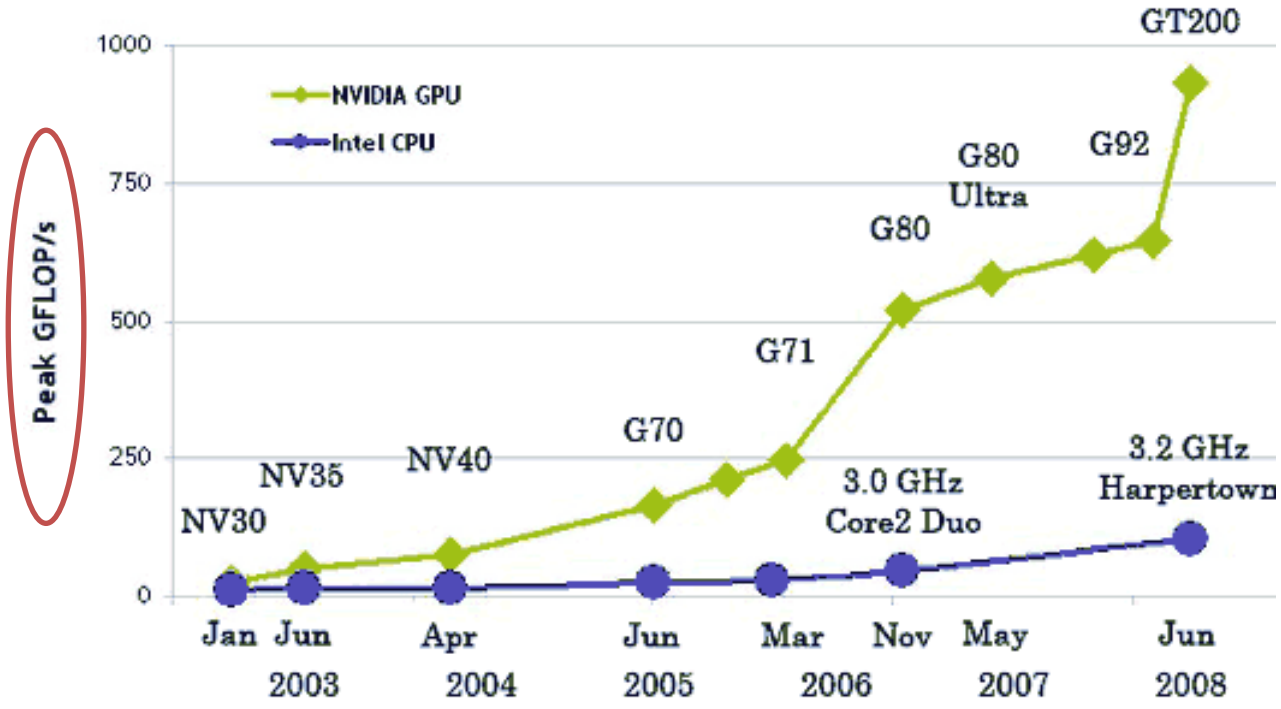
- Coordinating the Use of GPU and CPU for Improving Performance of Compute Intensive Applications
  - G.Teodoro, R.Sachetto, O.Sertel
  - Cluster Computing and Workshops, 2009. CLUSTER '09.

背景

- 不均質環境 (CPUs and GPUs) の出現
- CPUのマルチコア化、メニーコア化
- GPGPU
  - CPUに比べ高いピークパフォーマンス
  - 高い並列度

背景

- GPUが常に速いわけではない
- CPUとGPUはそれぞれ単独で使用される



目標

- 不均質な環境を対象
  - マルチCPU / GPU
- CPU, GPUの効率的な協調
  - 両者の特徴を活かすタスクスケジューリング
- プログラミングの高レベルの抽象化

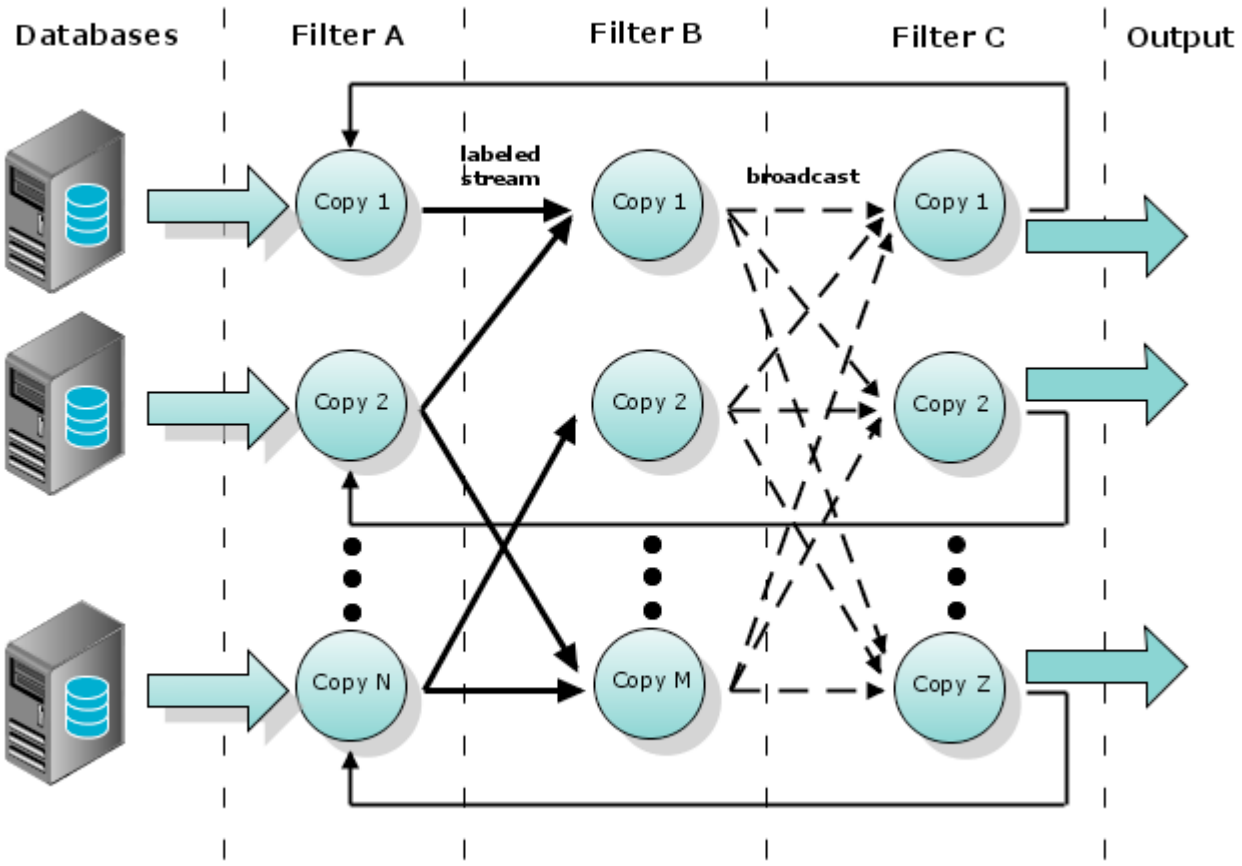
流れ

- Anthill
  - プログラミングの抽象化
  - 不均質環境のサポート
- アプリケーション
  - 神経芽細胞腫
- 評価
- まとめ

Anthill

- フィルタストリーミングモデルがベース (DataCutter)
  - アプリケーションをフィルタ (処理単位) に分解
  - ストリームを介して通信
    - フィルタ間通信の抽象化
  - 透過的なインスタンスのコピー
    - ユーザはコピー処理について気にする必要がない
  - データフローモデル
    - 並列化を容易にする
  - 2種類の並列化
    - タスク並列
    - データ並列

Anthill


Databases
Filter A
Filter B
Filter C
Output
labeled stream
broadcast
Copy 1
Copy 2
Copy N
Copy 1
Copy 2
Copy M
Copy 1
Copy 2
Copy Z

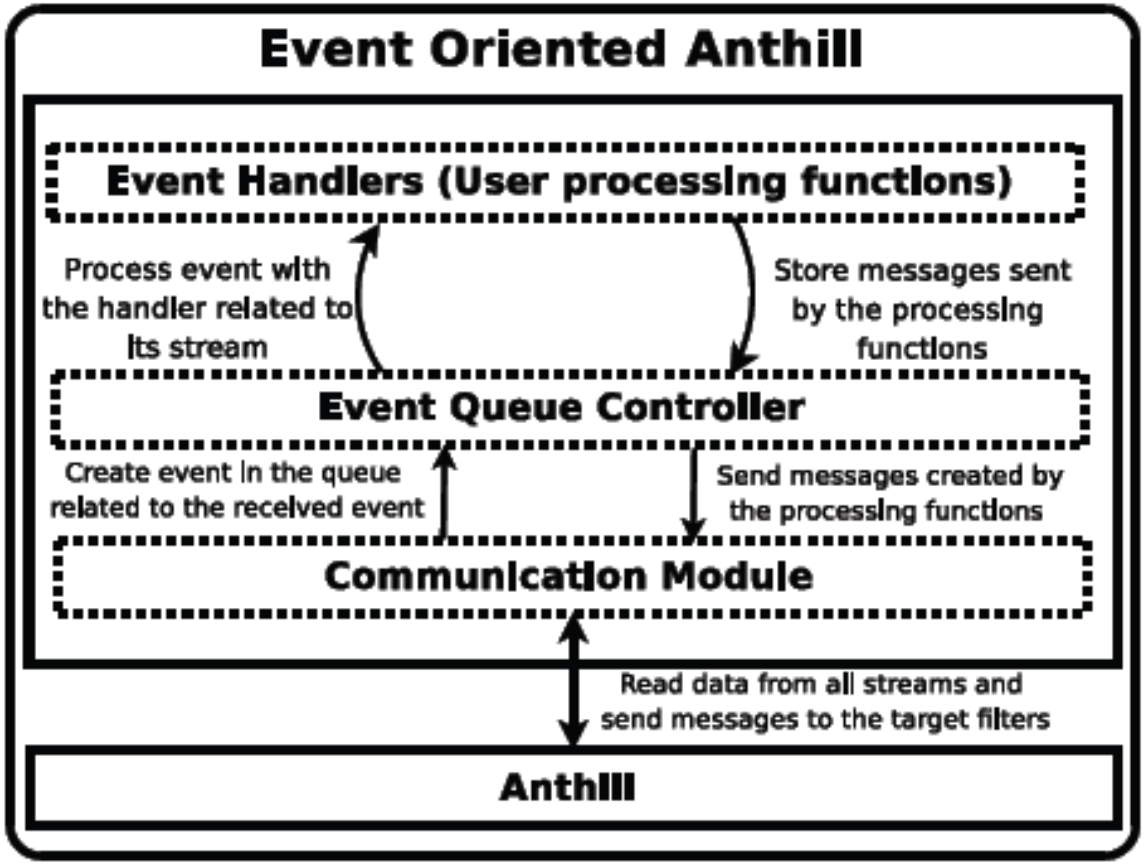
フィルタプログラミングの抽象化

- イベントドリブンなインターフェイス
  - データフローモデルと協調
- ユーザはデータ処理の関数のみを記述
  - 各データユニットに対して順に実行
- システムがユーザ関数の起動を制御
  - 依存関係の解析
  - 並列化

フィルタの構造

- コミュニケーションモジュール
  - 入力ストリーム監視, フィルタへのメッセージ送信
- イベントキューコントローラ
  - フィルタに関連するイベントキューの制御
- イベントハンドラ
  - ユーザが定義したアプリケーションの関数を実行

フィルタの構造


Event Oriented Anthill
Event Handlers (User processing functions)
Process event with the handler related to its stream
Store messages sent by the processing functions
Event Queue Controller
Create event in the queue related to the received event
Send messages created by the processing functions
Communication Module
Read data from all streams and send messages to the target filters
AnthIII

イベントハンドラ

- ユーザが提供する関数を実行
- データオブジェクトに対して動作
  - イベントが始動すると関数を実行
  - データ処理を終了後に結果を返す

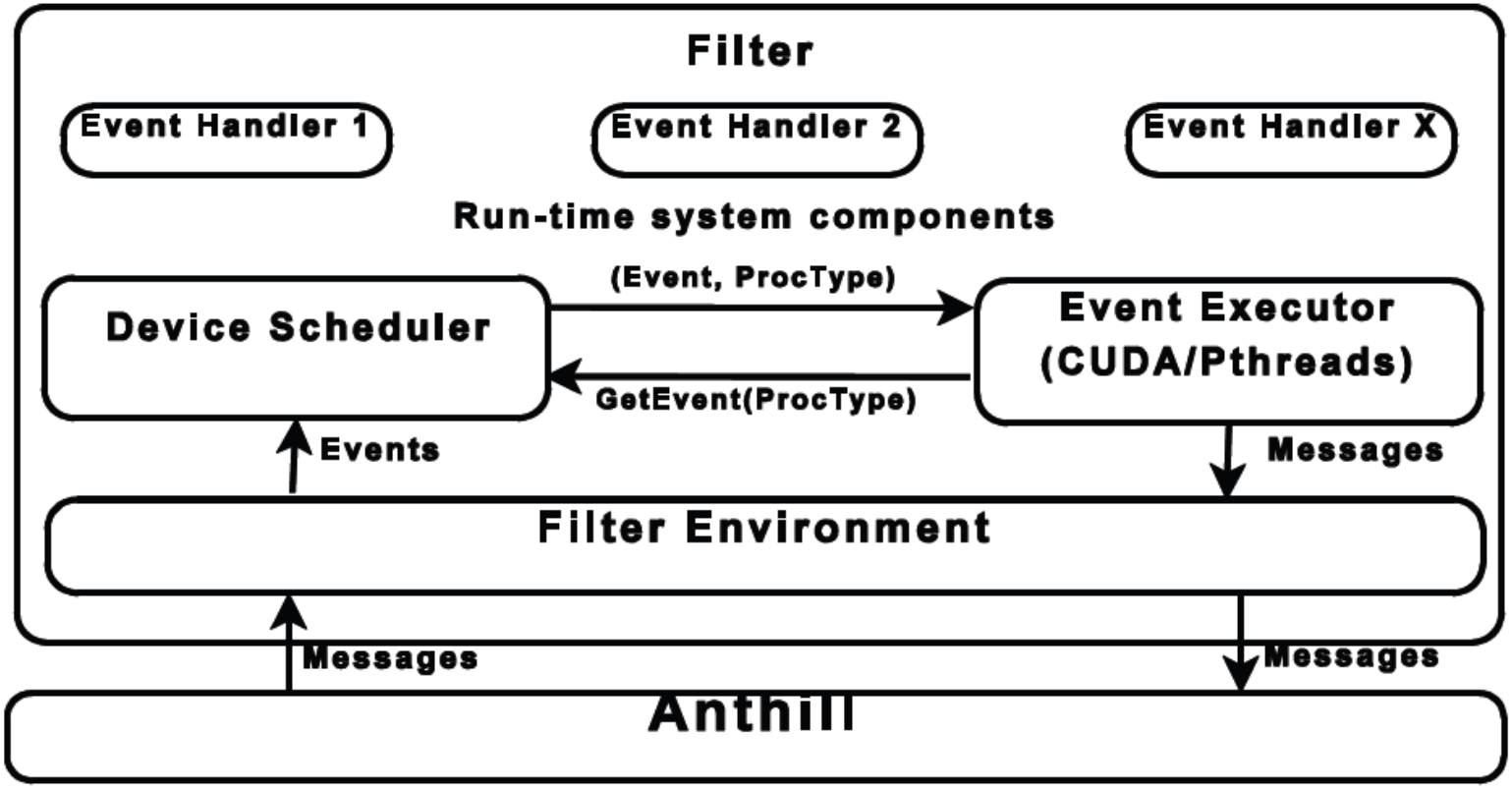
不均質環境のサポート

- 複数のデバイスにイベントハンドラを実装
  - 各フィルタを適切なデバイス向けに実装
- 複数のデバイスを並列に使用
- Anthillのランタイムが各イベントに対し、使用するデバイスを選択

不均質環境のサポートの概要

- デバイススケジューラ
  - 各イベントハンドラを監視
- Event Executor
  - ユーザが提供するコードに対し、ハードウェアへの依存をなくすシステムソフトウェア

不均質環境のサポートの概要



デバイススケジューラ

- 仮定
  - 各イベントは独立
  - Out-of-Order実行
- スケジューリングの方針
  - FCFS (First Come First Served)
  - DWRR (Dynamic Wighted Round Robin)
    - イベントを各デバイスでのパフォーマンスに応じて順序づけ
    - 各デバイスは最も加速するイベントを選択して実行

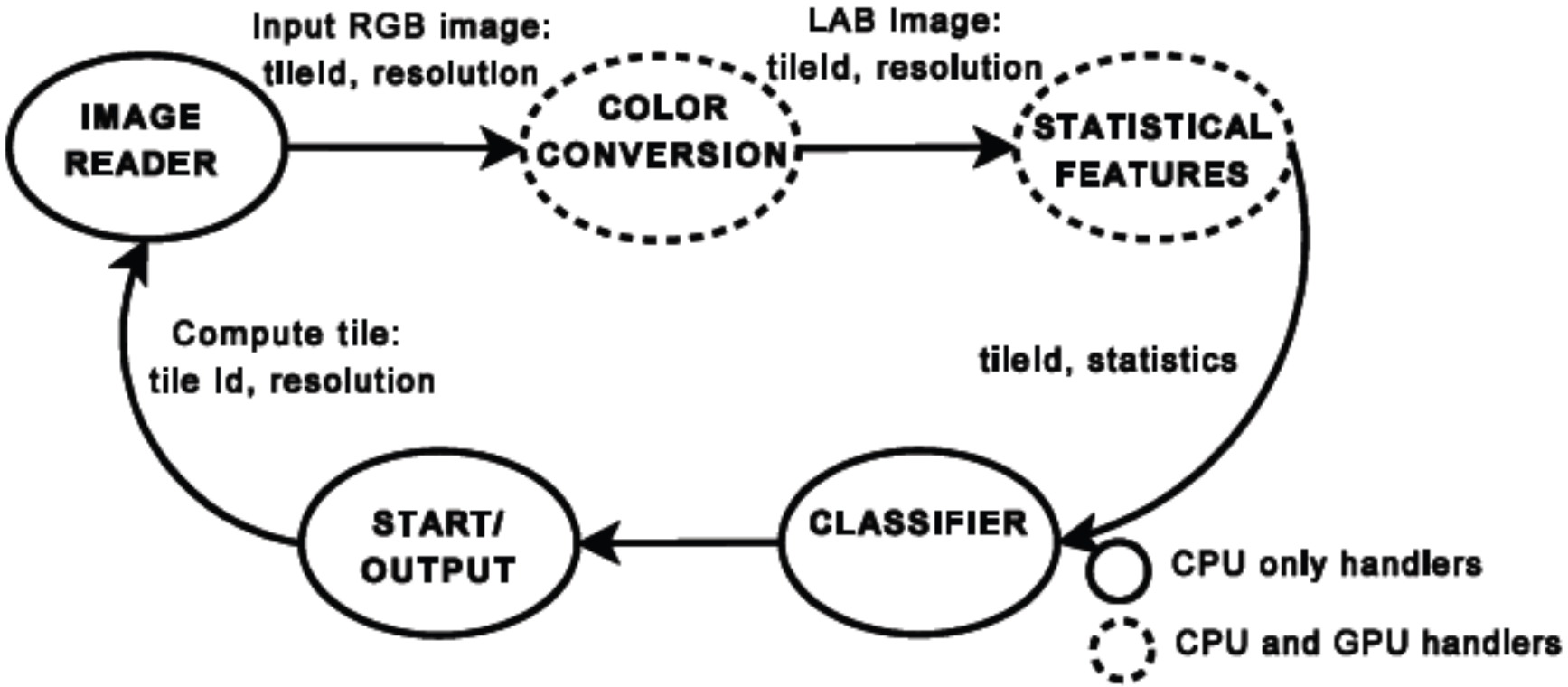
アプリケーション

- 病理学　(Histopathology)
  - 病気の原因、発生のメカニズムの解明
  - がん患者の予後診断
    - 神経芽細胞腫
- 画像解析
  - より正確で一貫性のある腫瘍形態 (tumor morphology) の把握
  - 視覚による検査 (疎粒度, 細粒度)

神経芽細胞腫 (Neuroblastoma) の画像解析システム

- 各細胞を予後の重要性について分類
- 解像度が高い
  - 小さいタイルに分割
- 多重解析度の画像解析
  - 低い解像度から高い解像度へ
    - 病理学者が調べる方法と類似

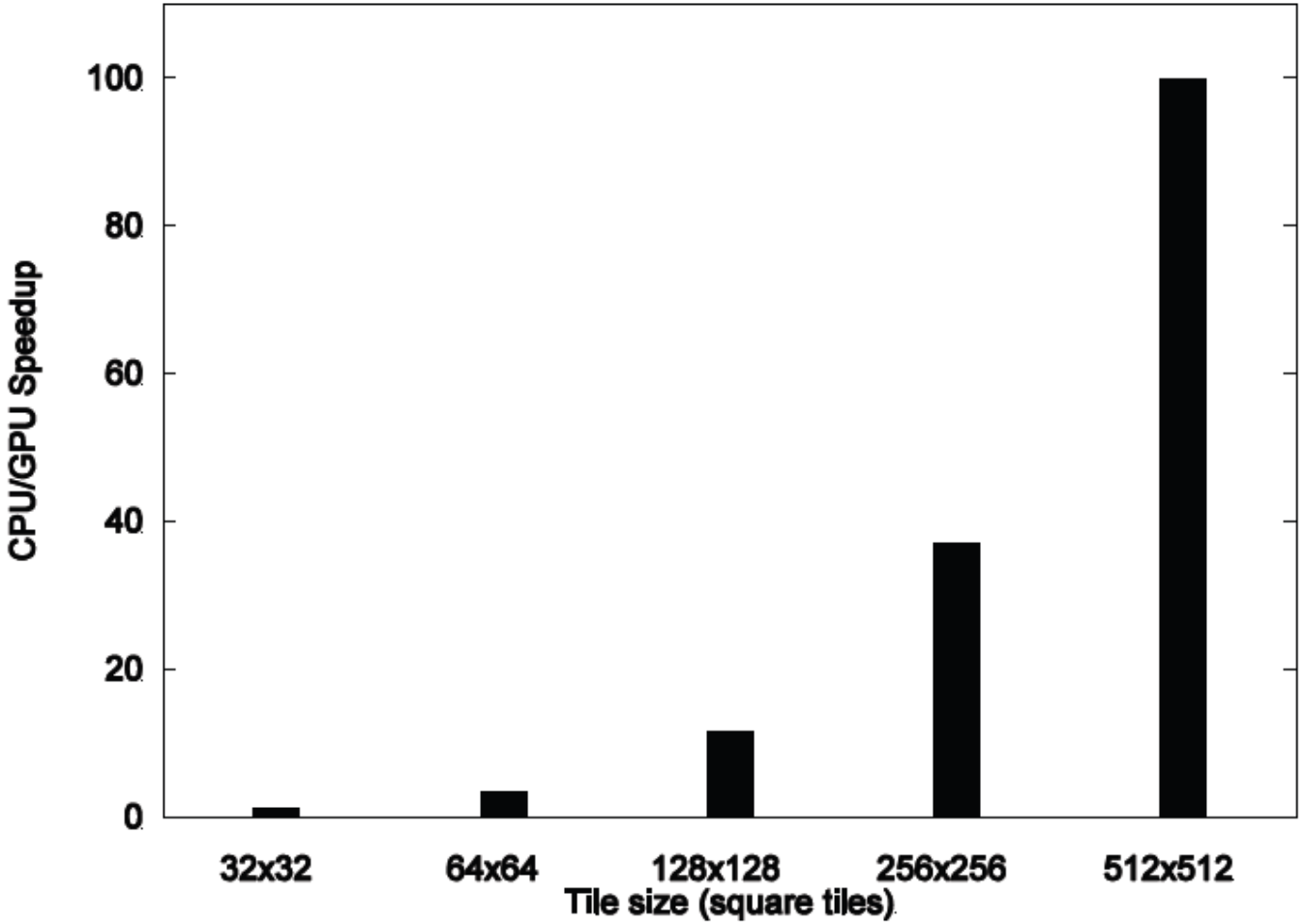
Anthillへの実装



評価

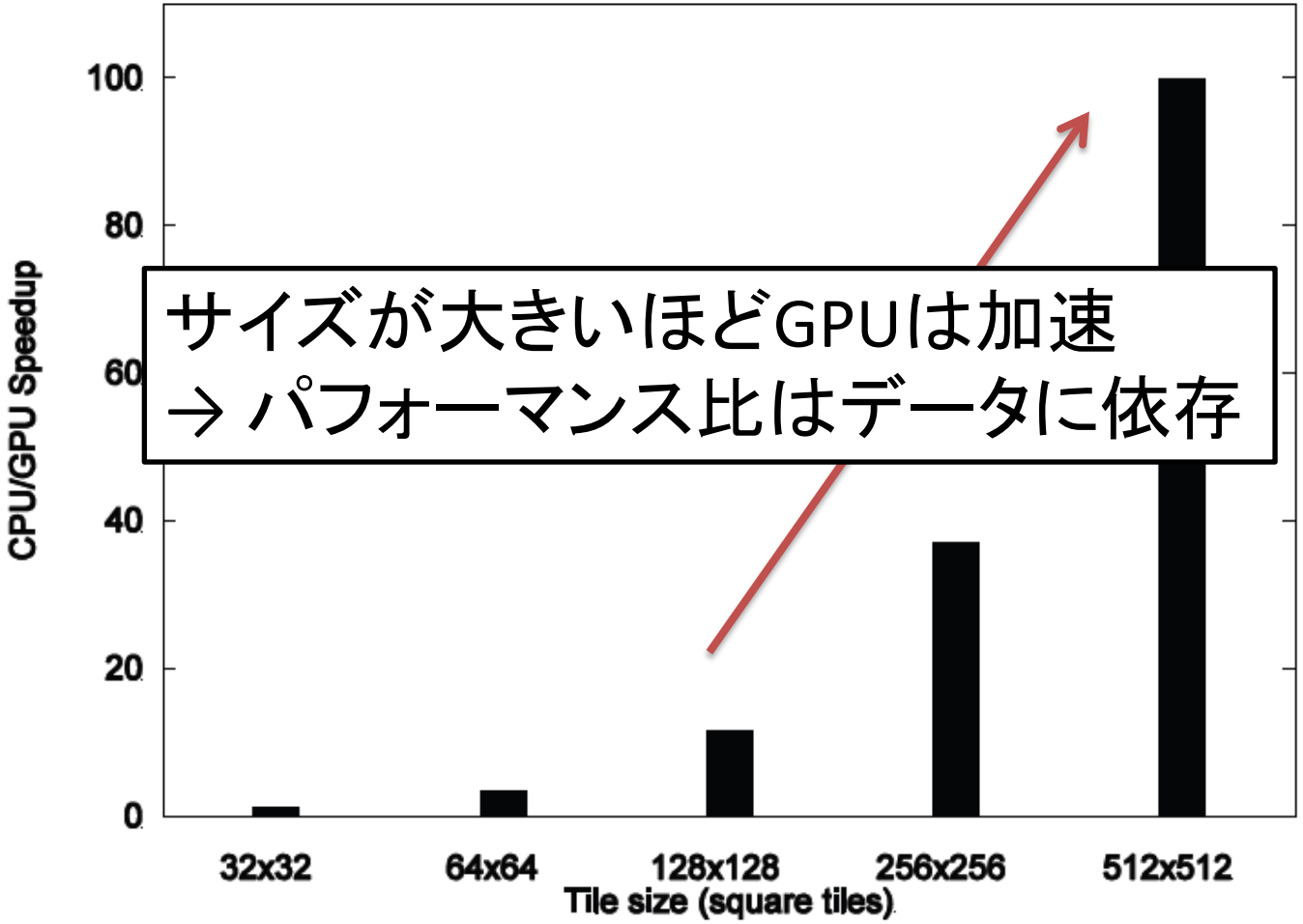
- セットアップ
  - 10台, Intel Core 2 Duo CPU 2.13GHz / NVIDIA GeForce 8800GT GPU
  - 4台, dual quad-core AMD Opteron 2.00GHz processor / NVIDIA GeForce GTX 260
  - 入力データ: 26,742個のタイルの画像, 二種類の解像度: 32x32, 512x512

NBIAタスクの分析



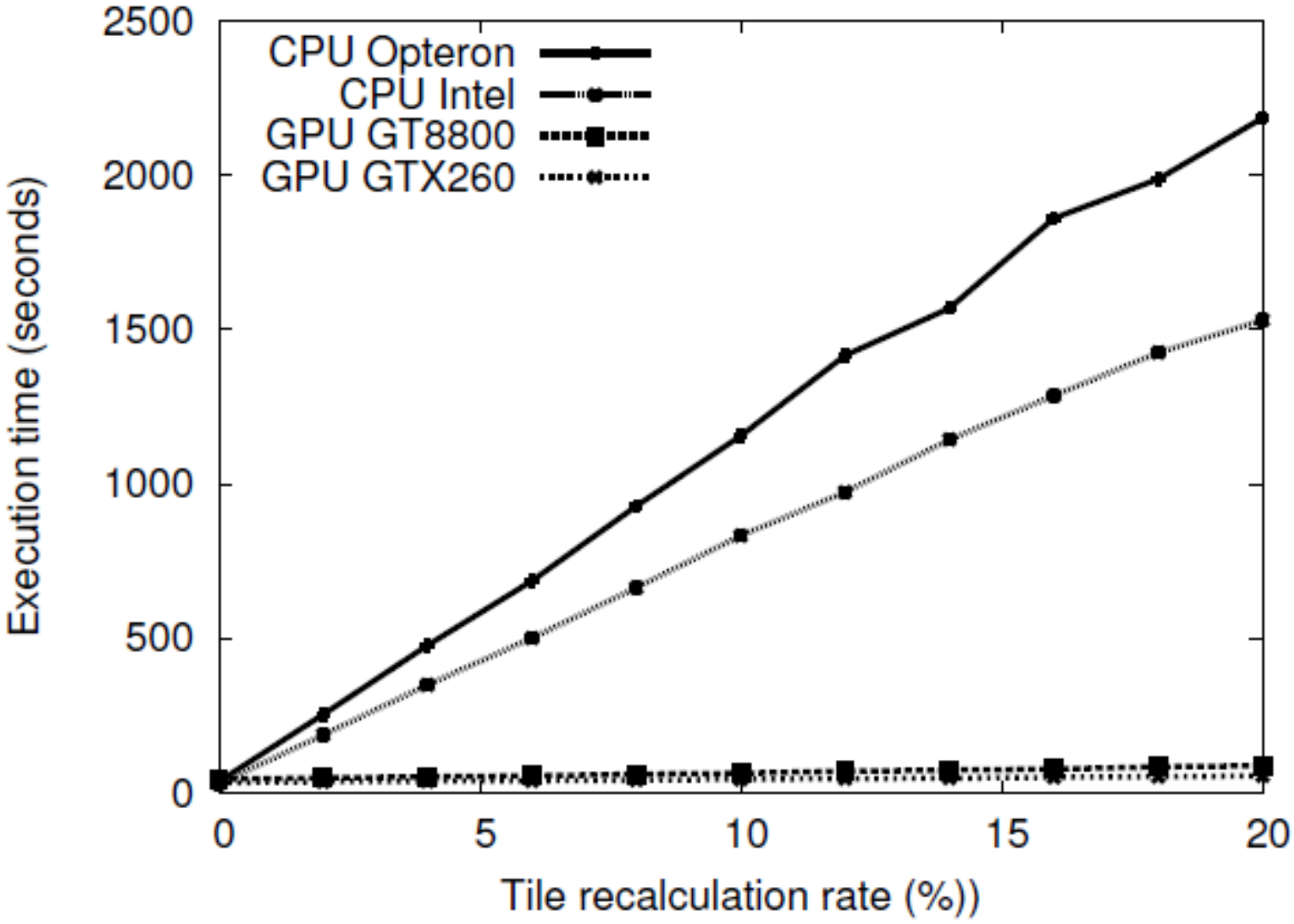
Dual quad-core AMD Opteron 2.00GHz/NVIDIA GeForce GTX260

NBIAタスクの分析

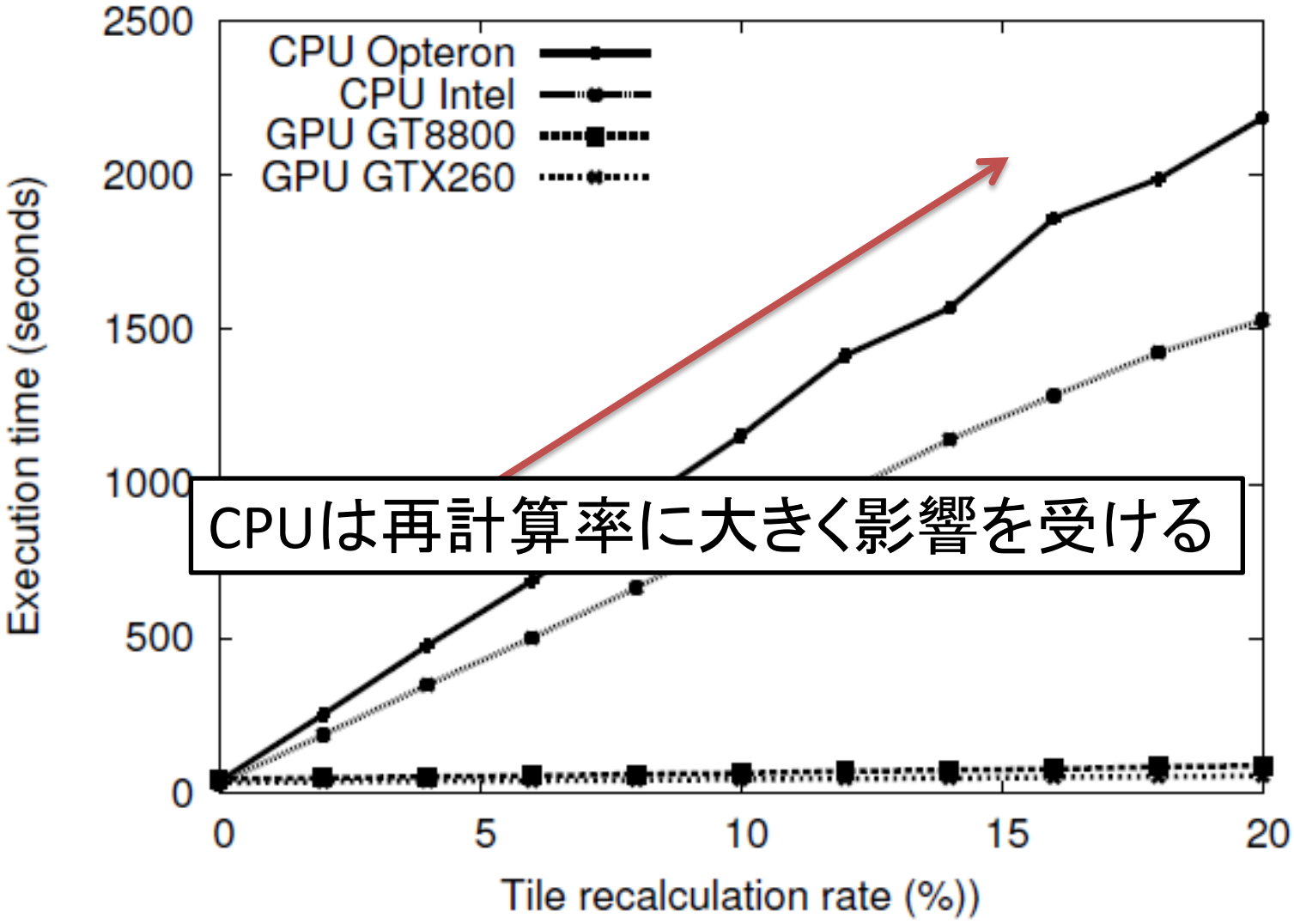


Dual quad-core AMD Opteron 2.00GHz/NVIDIA GeForce GTX260

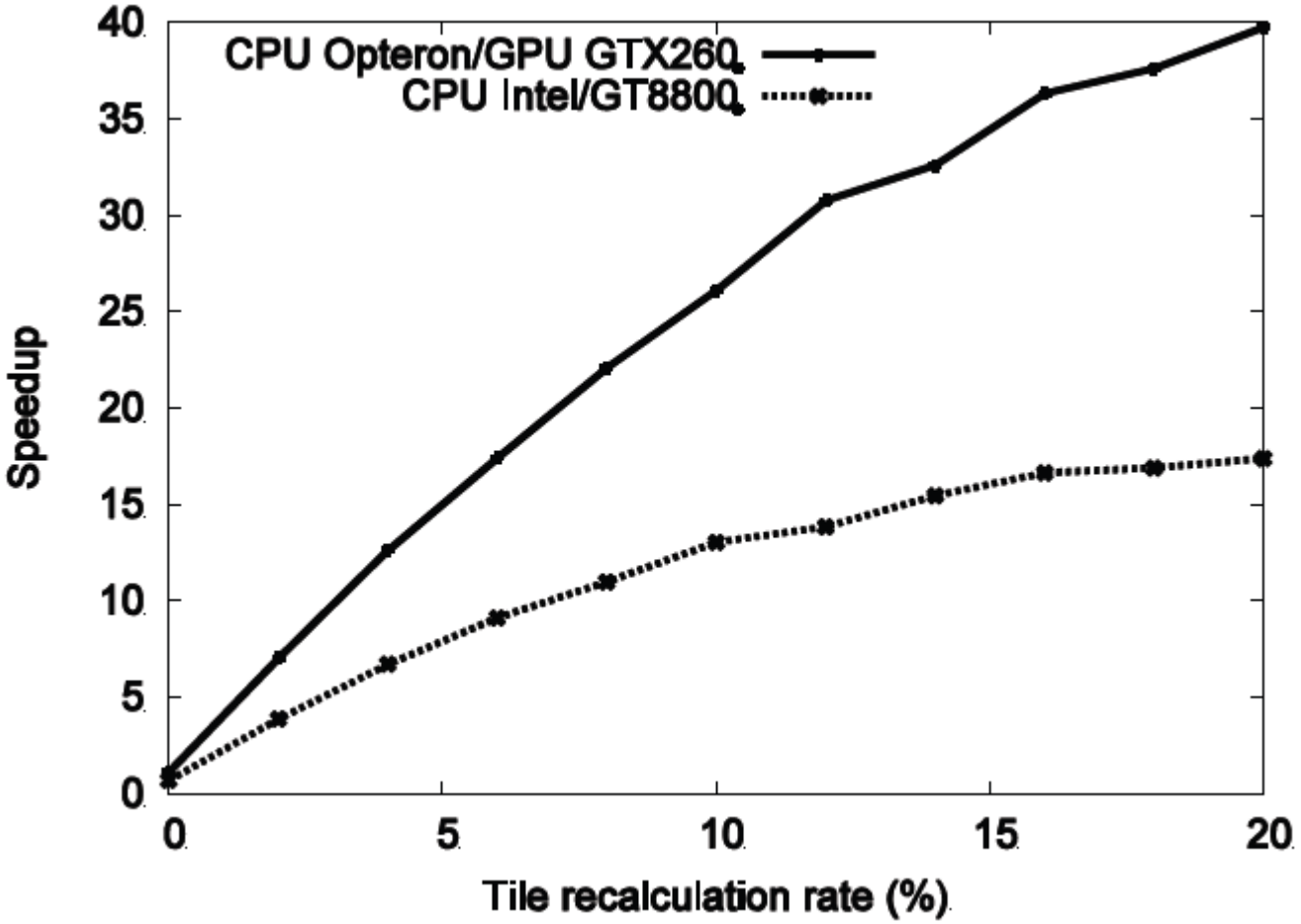
各デバイスの実行時間


CPU Opteron
CPU Intel
GPU GT8800
GPU GTX260
Execution time (seconds)
2500
2000
1500
1000
500
0
0
5
10
15
20
Tile recalculation rate (%))

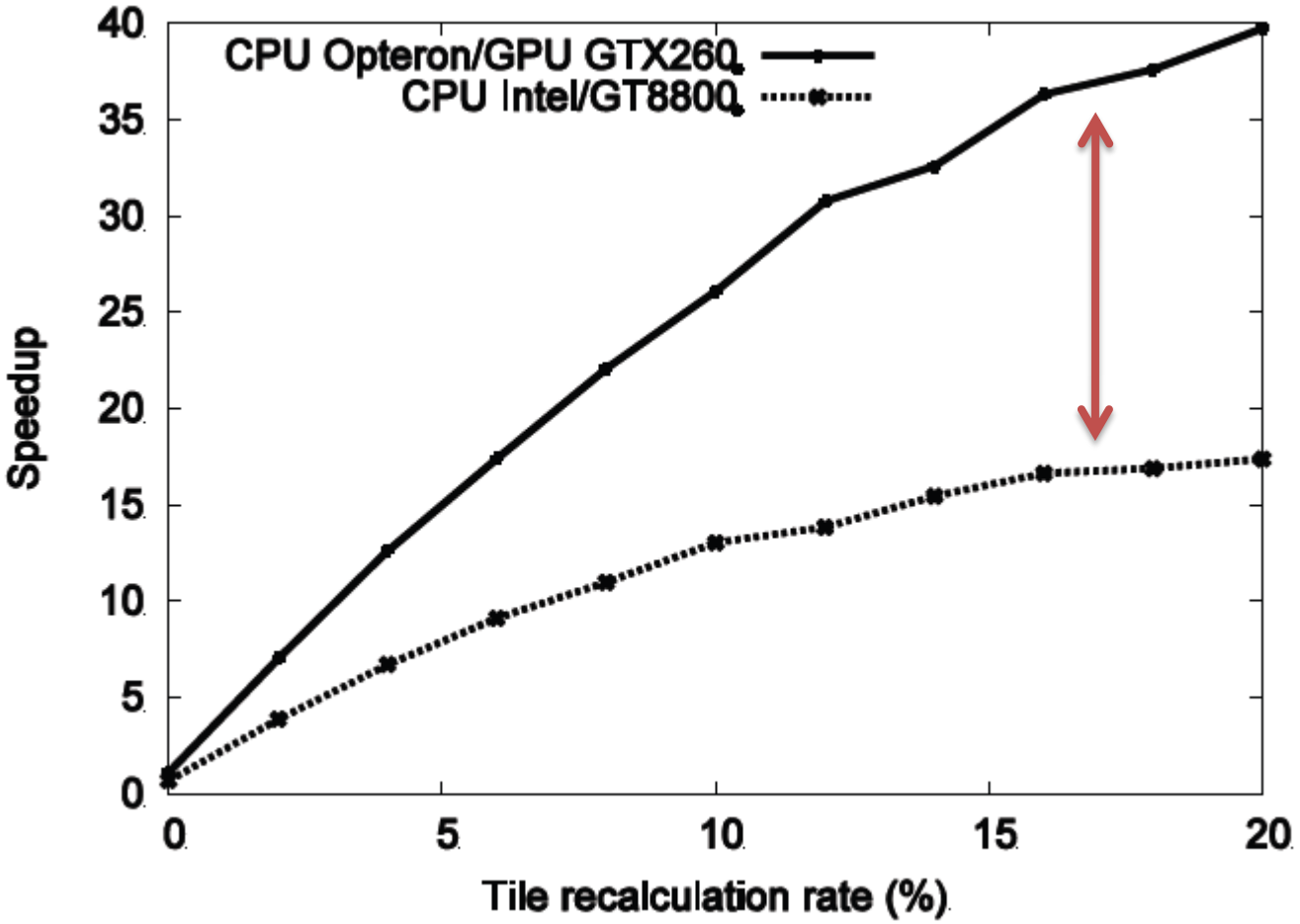
各デバイスの実行時間


CPU Opteron
CPU Intel
GPU GT8800
GPU GTX260
Execution time (seconds)
2500
2000
1500
1000
500
0
0
5
10
15
20
Tile recalculation rate (%))
CPUは再計算率に大きく影響を受ける

ハイブリッド実行時の加速

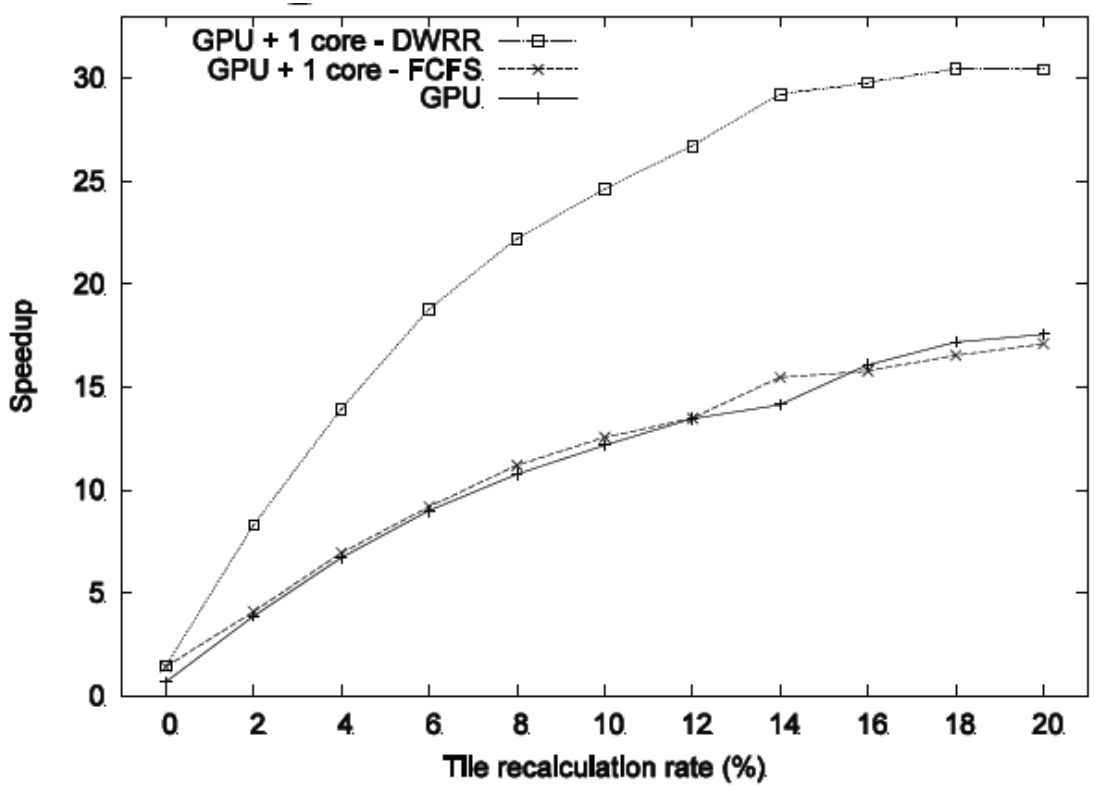


ハイブリッド実行時の加速



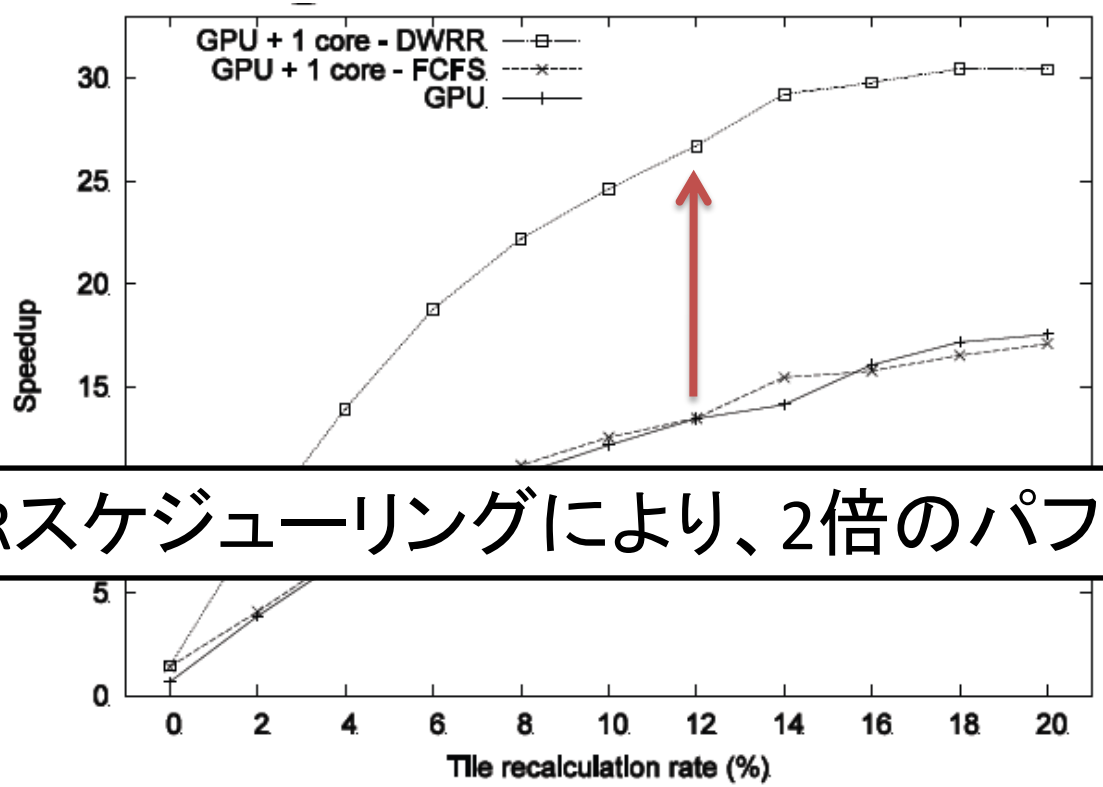
CPUとGPUの組み合わせによって加速度が異なる

不均質スケジューリングの分析



Recalc (%)	0		12	
Resolution	Low	High	Low	High
1 CPU core- FCFS	16049	0	263	215
1 CPU core- DWRR	15978	0	21592	4

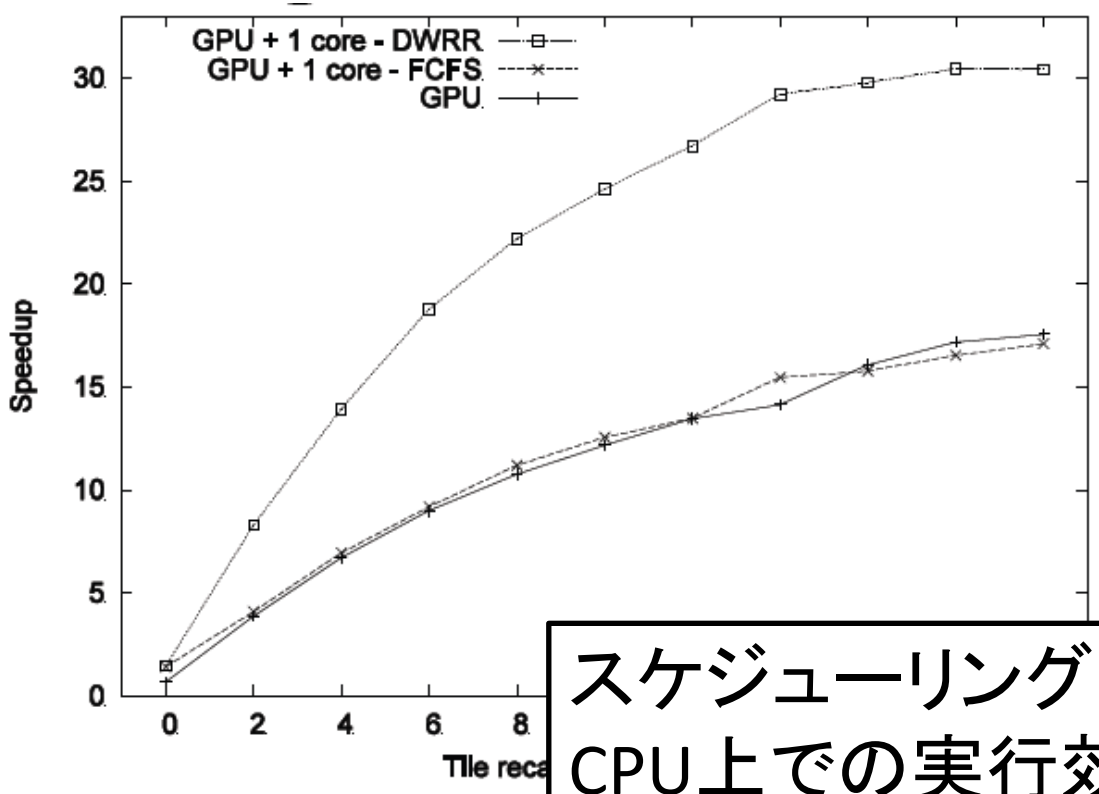
不均質スケジューリングの分析



DWRRスケジューリングにより、2倍のパフォーマンス

Recalc (%)	0		12	
Resolution	Low	High	Low	High
1 CPU core- FCFS	16049	0	263	215
1 CPU core- DWRR	15978	0	21592	4

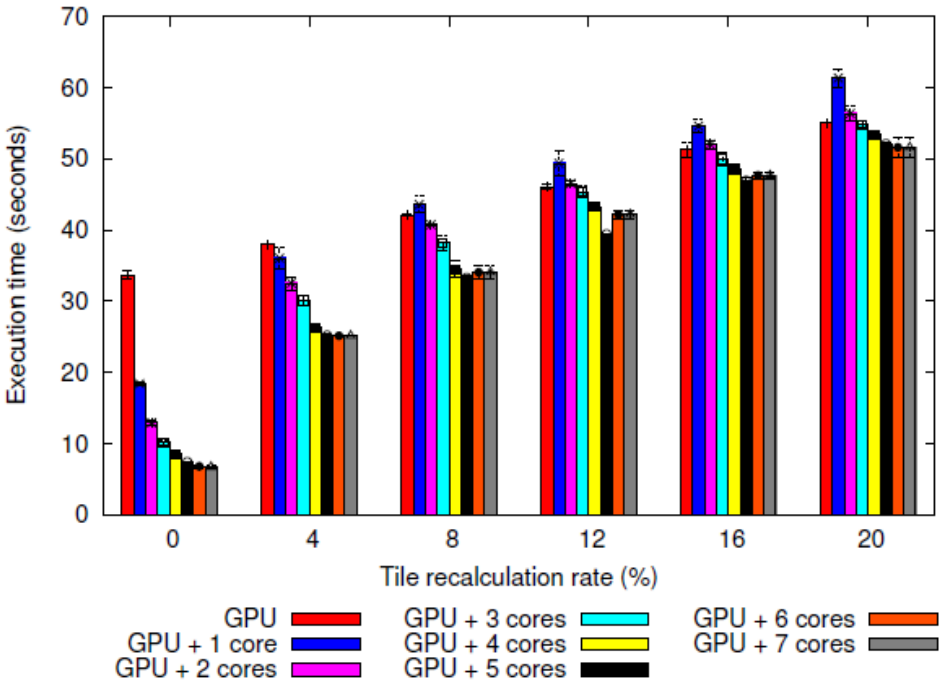
不均質スケジューリングの分析



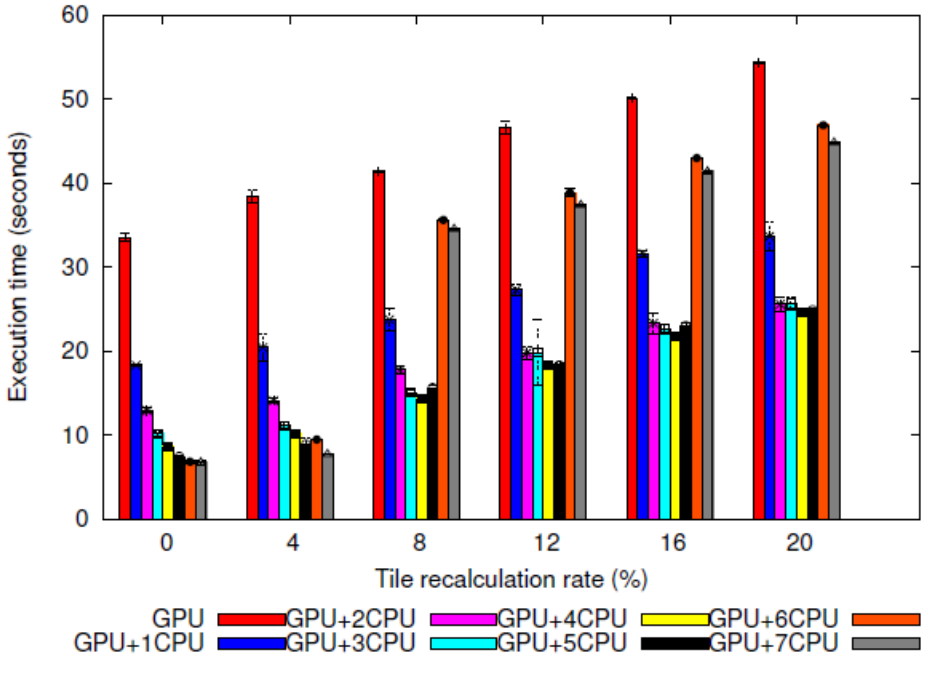
スケジューリング手法による、
CPU上での実行効率の違い

Recalc (%)	0		12	
Resolution	Low	High	Low	High
1 CPU core- FCFS	16049	0	263	215
1 CPU core- DWRR	15978	0	21592	4

不均質スケジューリングの分析

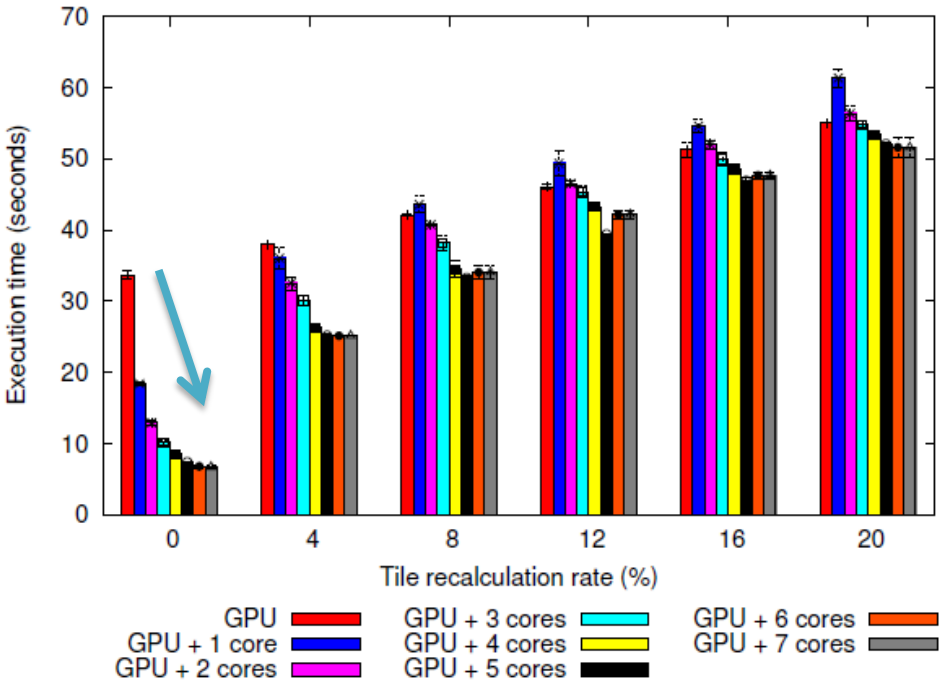


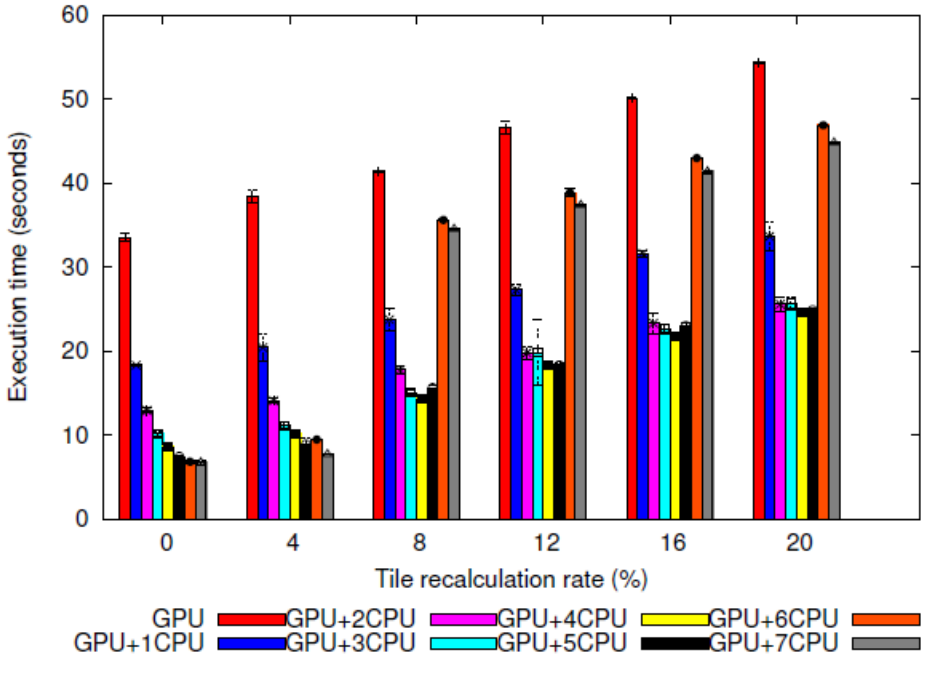
FCFS



DWRR

不均質スケジューリングの分析
実行時間

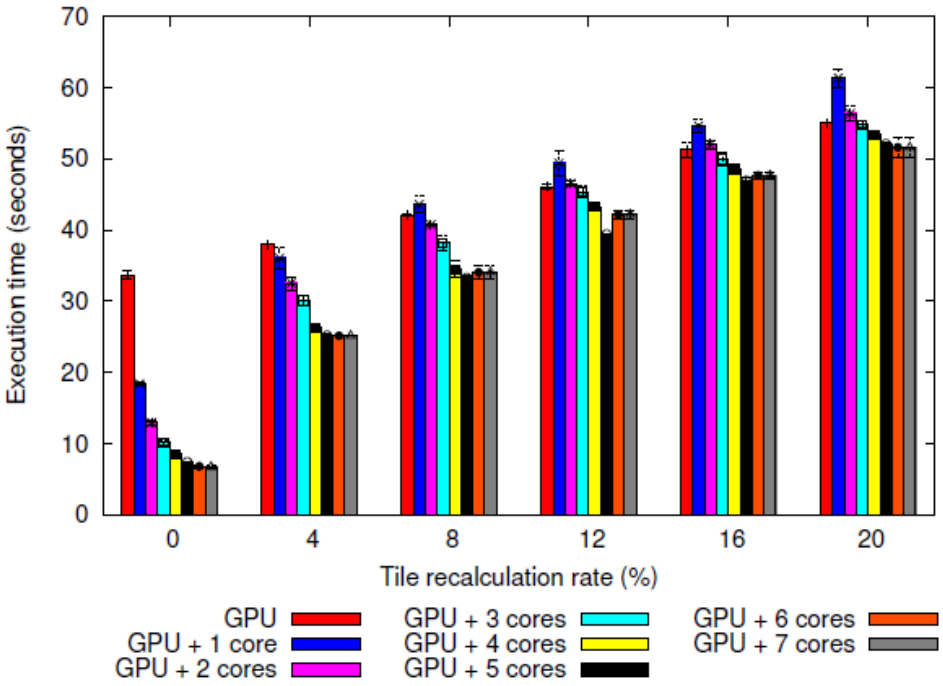


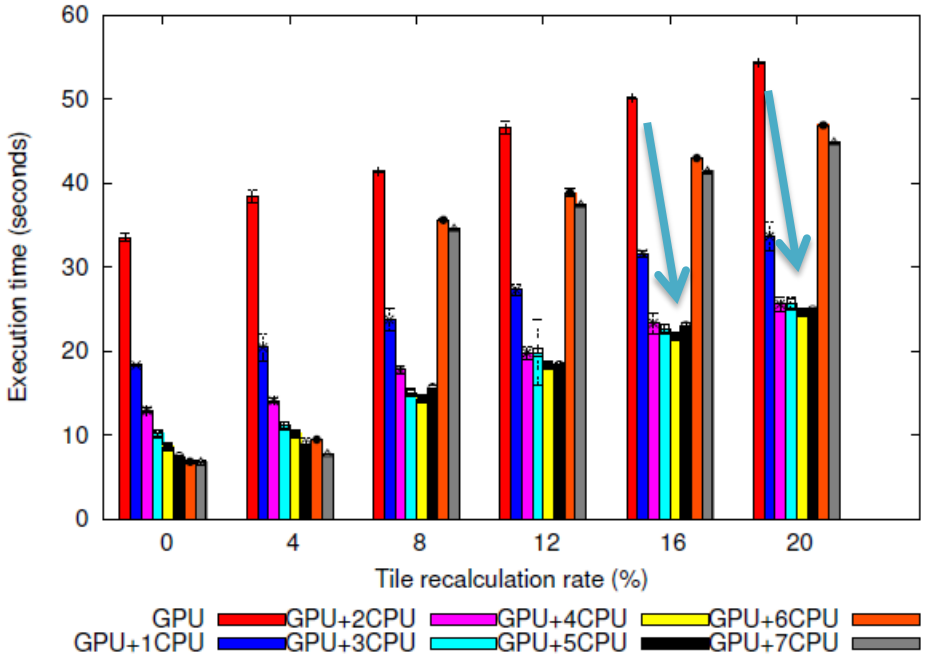


FCFS　　DWRR

再計算率が小さいほど
CPUコア数による性能向上大

不均質スケジューリングの分析
実行時間

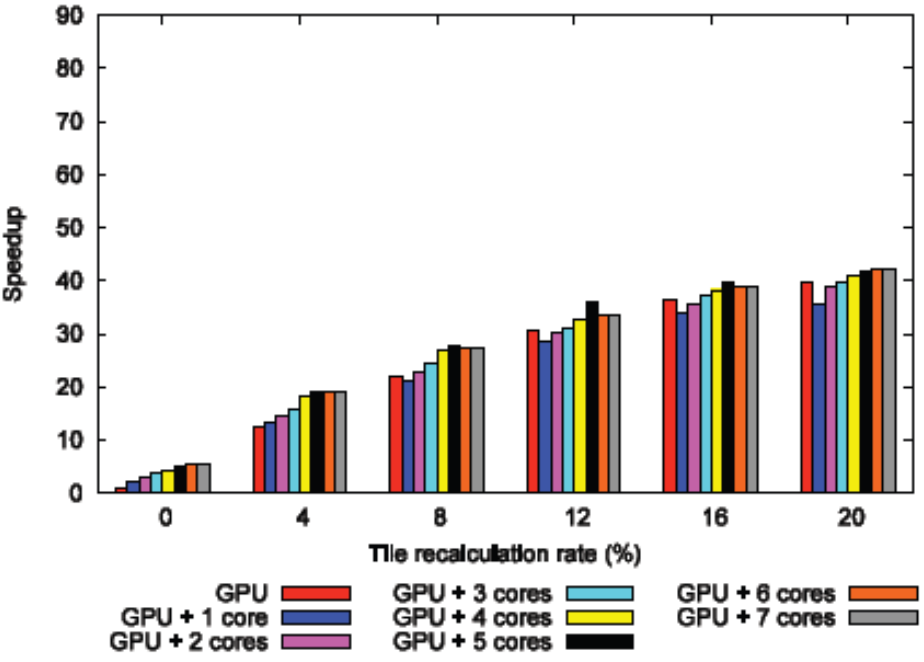




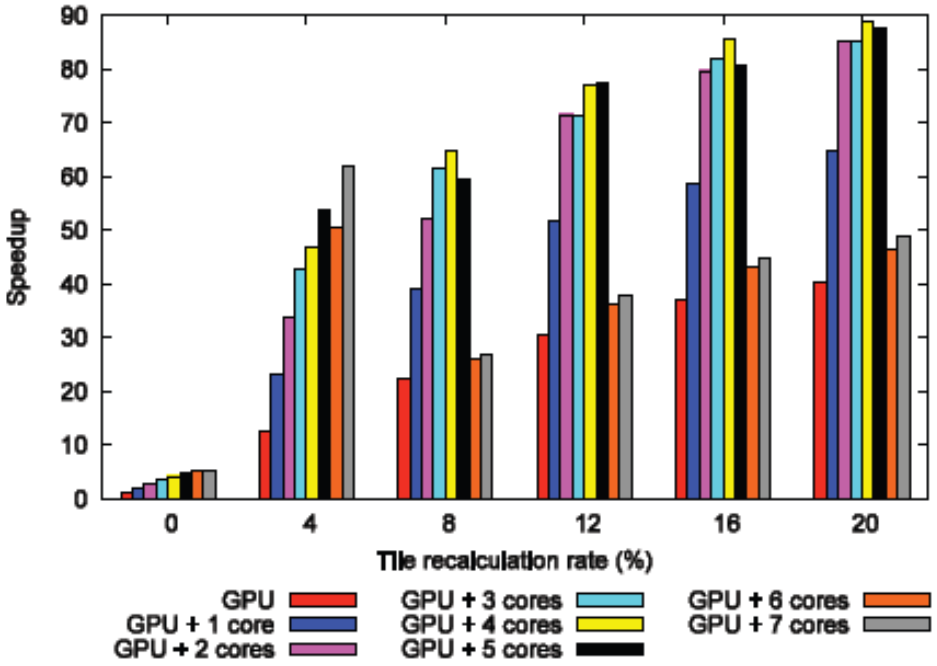
FCFS　　DWRR

DWRRでは再計算率が大きいとき、
CPUコア数の増加による著しい性能向上

不均質スケジューリングの分析
スピードアップ

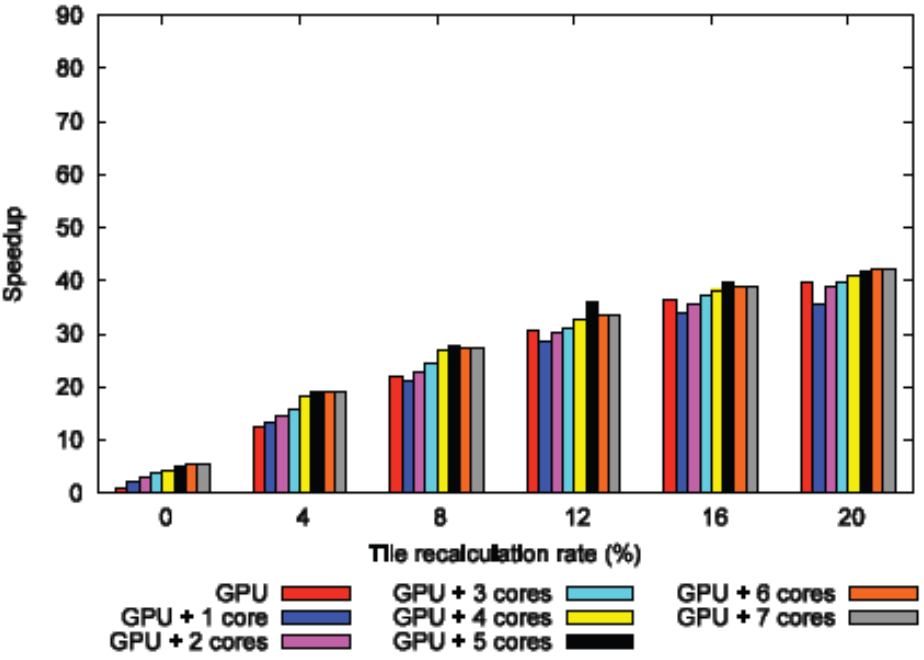


FCFS

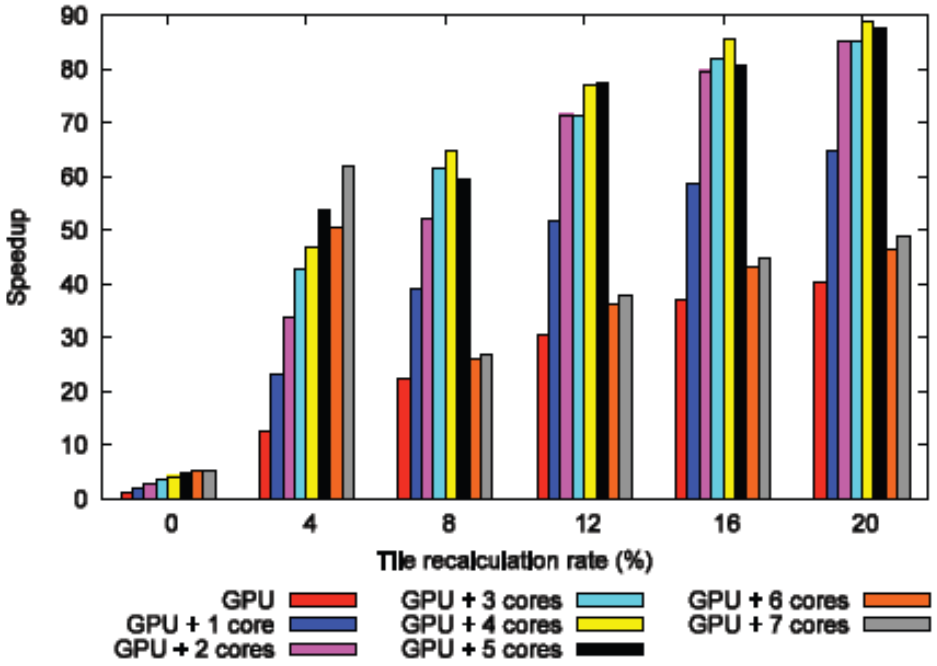


DWRR

不均質スケジューリングの分析
スピードアップ



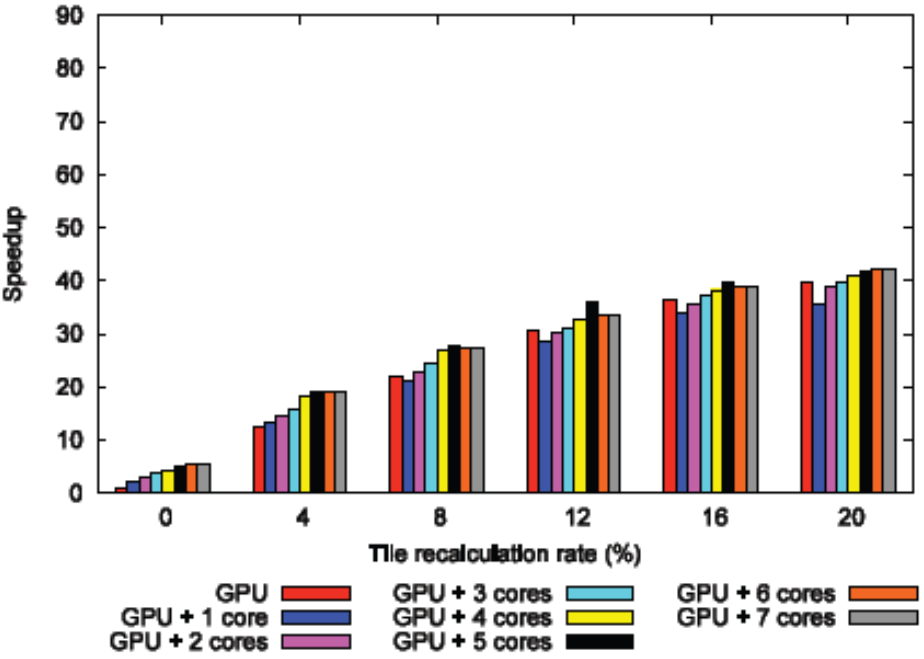
FCFS

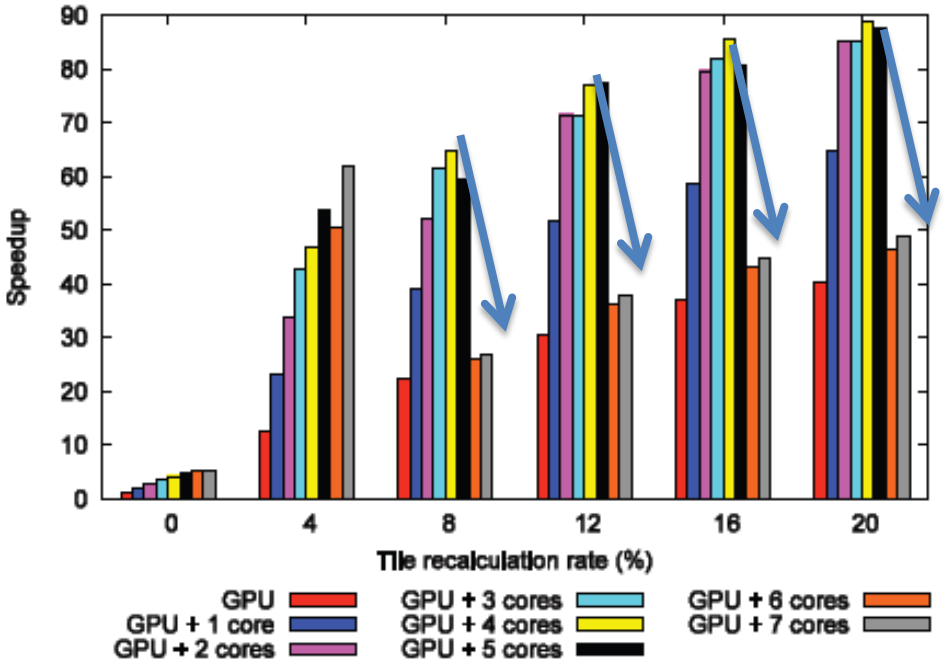


DWRR

DWRRではFCFSの2倍近い性能

不均質スケジューリングの分析
スピードアップ





FCFS　　DWRR

DWRRで、CPUが6コアを超えてから性能低下
→ I/Oなどとの競合が原因

不均質スケジューリングの分析

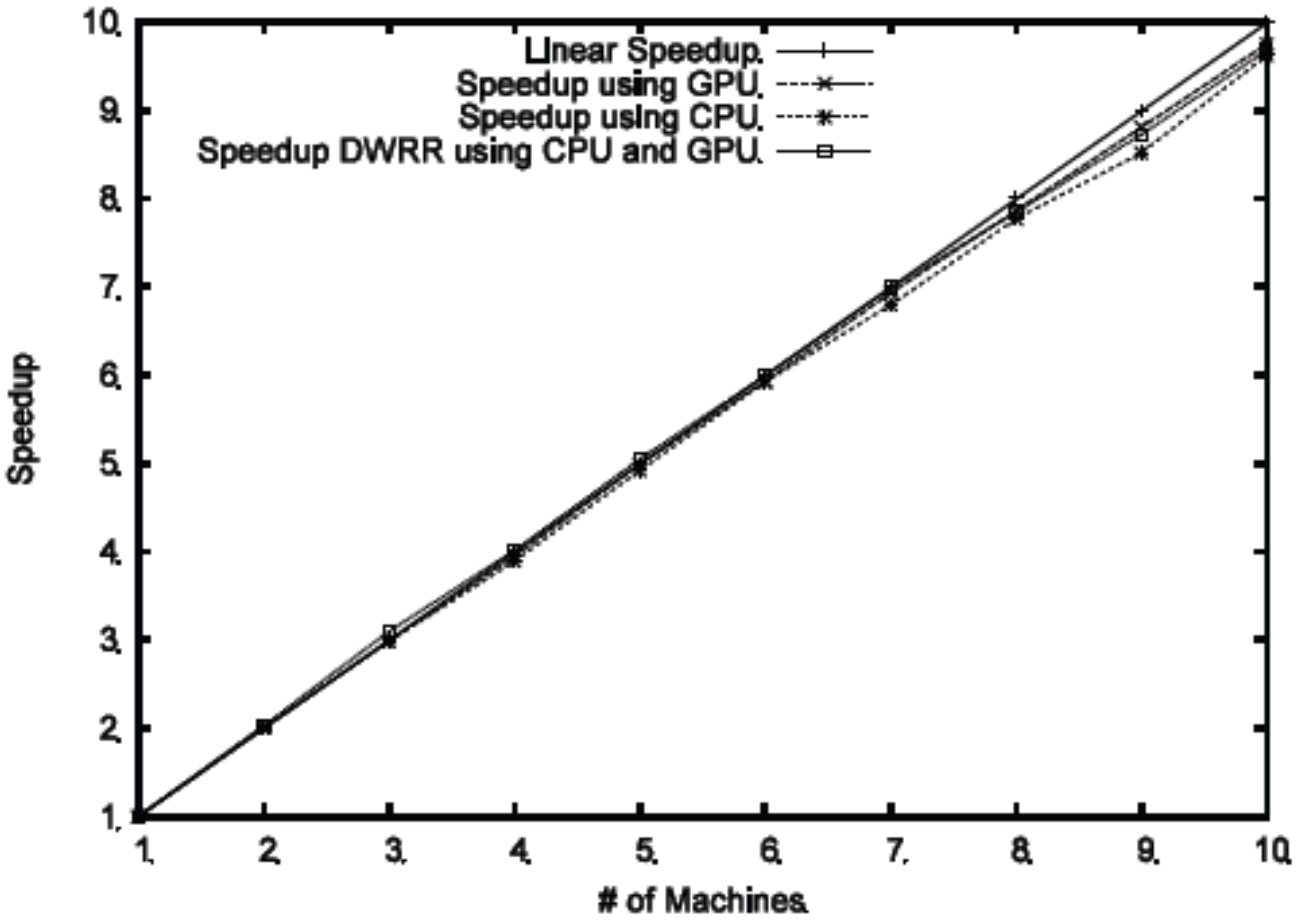
# of CPU cores	FCFS		DWRR	
	Low	High	Low	High
1	637	58	10714	1
2	1177	133	15748	2
3	1925	173	18614	5
4	2090	219	18634	28
5	2872	286	20070	40
6	3819	393	20417	76
7	4726	478	20266	57

不均質スケジューリングの分析

# of CPU cores	FCFS		DWRR	
	Low	High	Low	High
1	637	58	10714	1
2	1177	133	15748	2
3	1925	173	18614	5
4	2090	219	18634	28
5	2872	286	20070	40
6	3819	393	20417	76
7	4726	478	20266	57

スケジューリング手法によるタスクの実行効率の違い

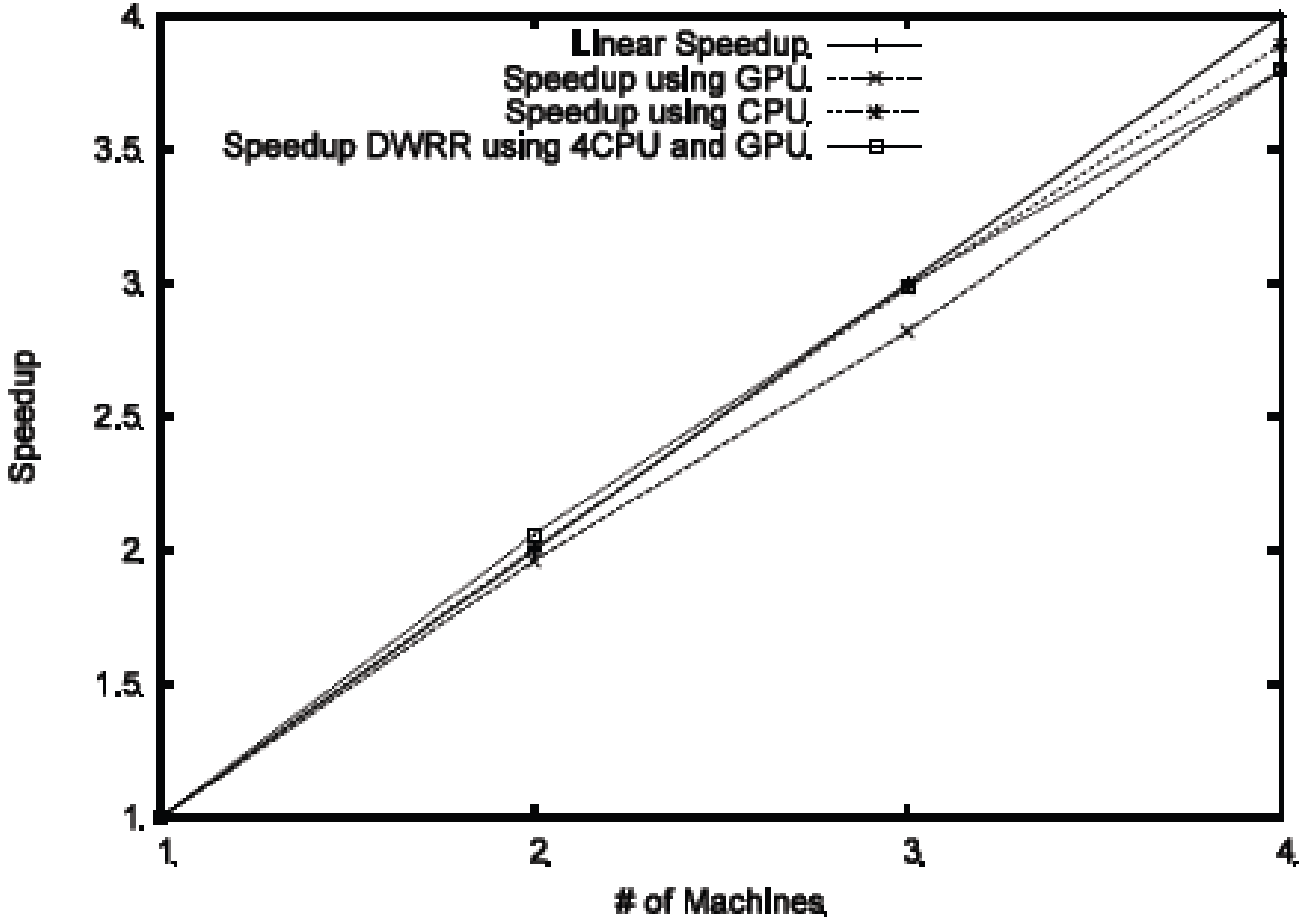
分散環境での実験



Intel dual core /8800GT cluster

ノード数の増加に対し、ほぼ線形にスケール

分散環境での実験



Two Opteron Quad-core / GTX260 cluster

こちらの場合もほぼ線形にスケール

関連研究

- データフロー計算モデル
  - MapReduce, DataCutter, SPC, Dryad
  - 不均質環境を考慮していない
- GPUの活用
  - Mars, Merge framework
  - 1ノードでの実行しか考慮していない
- 既存のスケジューリング手法との違い
  - GPUとの不均質環境
  - 入力に柔軟に対応

まとめ

- CPU, GPUのパフォーマンス比はデータに依存
- 不均質環境へのスケジューリングの適用により、アプリのパフォーマンスが2倍に増加
- GPUのみではなく、CPUも使用すべき
- データフローは並列化のために重要

感想

- 興味深かった点
  - DWRRではFCFSの2倍程度のパフォーマンスを発揮できており、スケジューリングの手法によってパフォーマンスが大きく異なる
- 疑問点
  - DWRRの詳細なスケジューリングの計算方法
  - 他のアプリケーションへの適用時にどのような挙動をするか
    （DWRRでFCFSに対しどの程度加速するか）