並列プログラムを
メモリ階層利用可能とする
ランタイム

とその応用

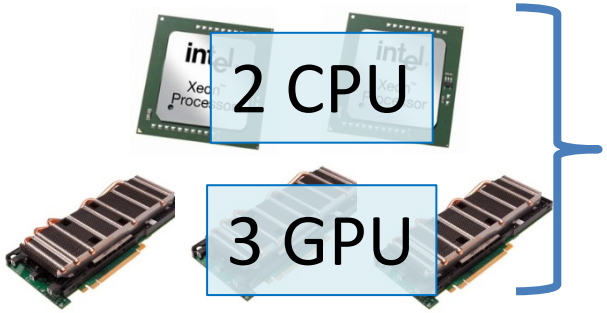
の予備評価

東京工業大学
遠藤敏夫

JST-CREST「ポストペタスケール時代のメモリ階層の深化に
対応するソフトウェア技術」

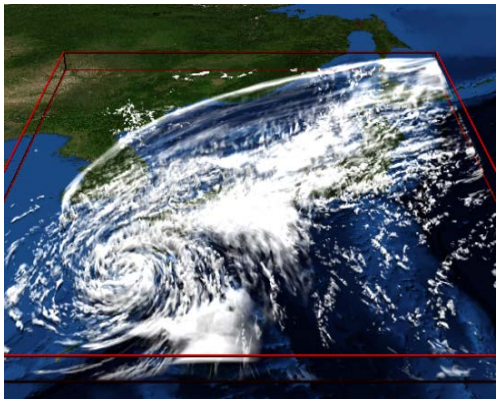
ハイブリッドスパコンと様々なシミュレーション

東工大TSUBAME2.0スパコン

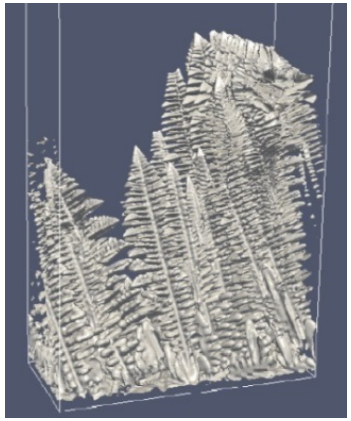


× 1408 =

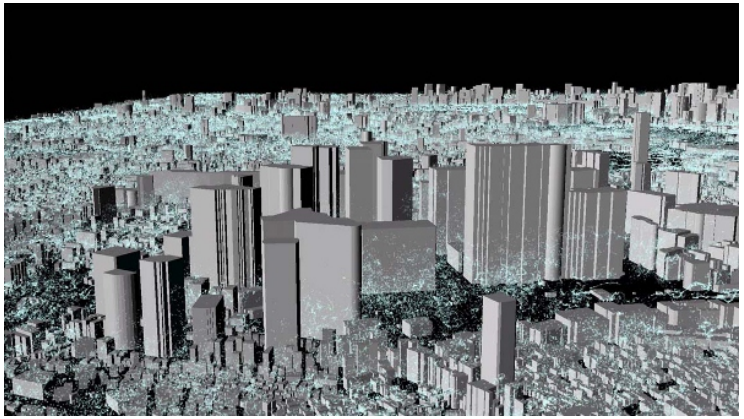
TSUBAME2.0上での大規模アプリ例 (東工大青木ら)



気象コードASUCA



Phase-Field計算
(2011 Gordon Bell賞)

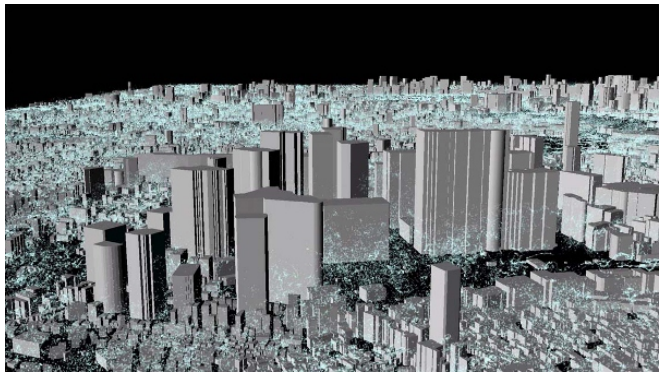


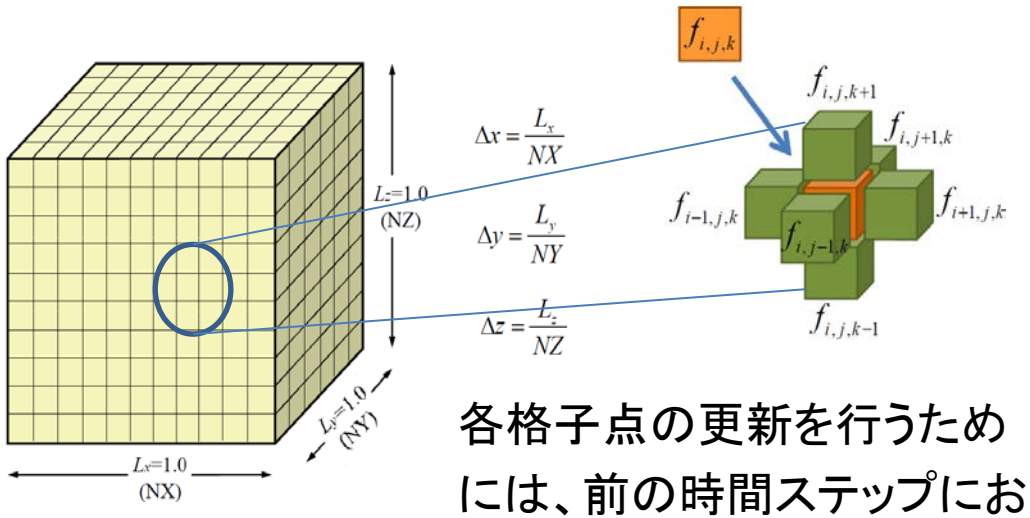
都市気流シミュレーション
(HPCS 2013 最優秀論文)

本研究ではステンシル計算に注目し、その問題規模を議論

ステンシル計算が ハイブリッドスパコンで成功する理由

ステンシル計算：連続体シミュレーションを中心とする様々な科学分野計算において重要なカーネル



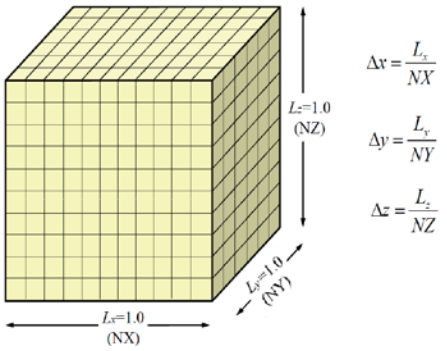


各格子点の更新を行うためには、前の時間ステップにおける近傍の値を必要.

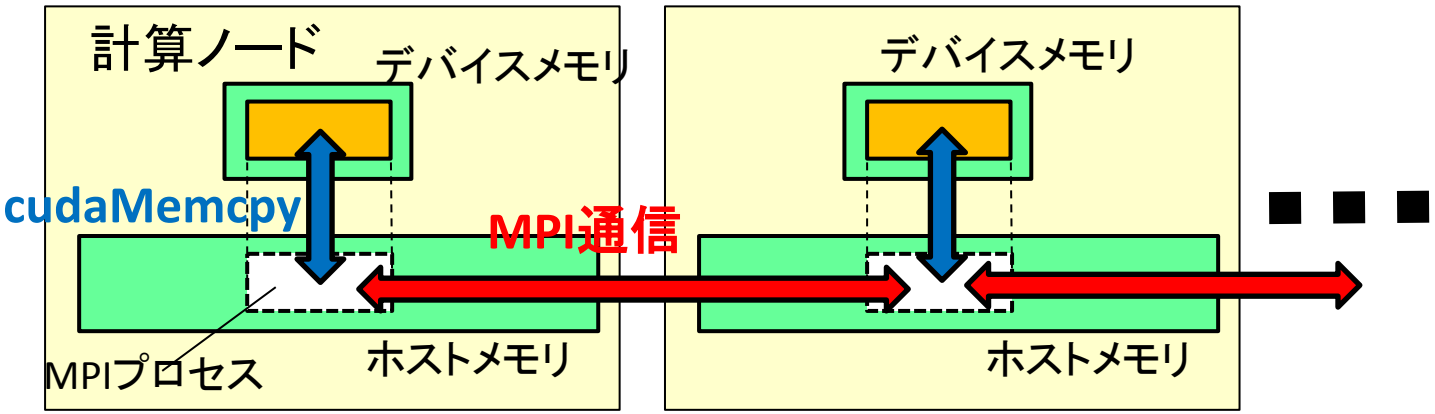
- GPUアクセラレータの特徴利用
  - 高いメモリバンド幅: 150GB/s × 4200
  - 高い演算速度: 1030GFlops(SP) × 4200
- マルチGPU時でも良好な計算・通信比
  - 領域分割⇒隣接のみの通信なので、通信量のオーダーは一つ低い

デバイスメモリ容量による限界

- 格子の細かさはGPUデバイスメモリ容量により限定される
  - TSUBAME2.0: 3GB × 4200 = 12TB
  - シミュレーションの精度を上げるため、もっと格子を細かくしたい！

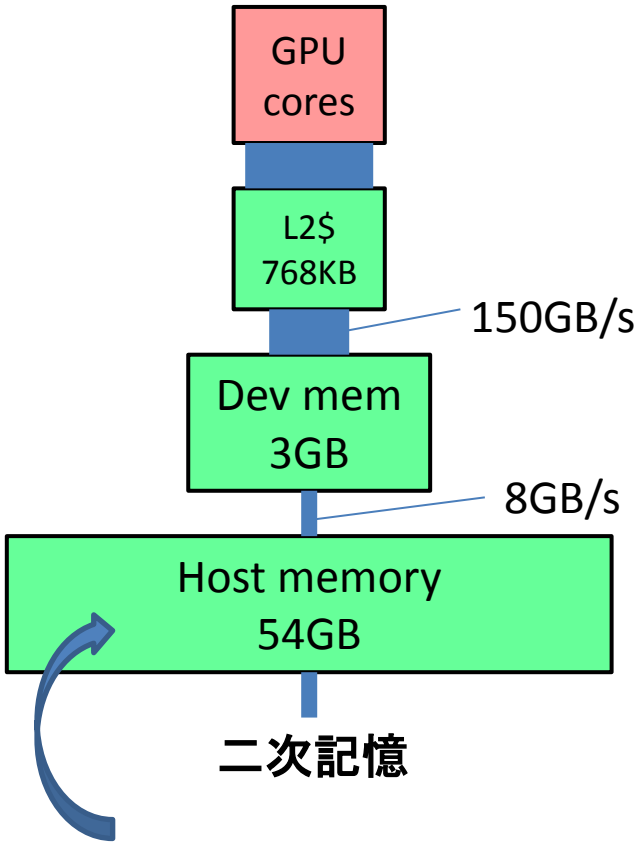


MPI+CUDAでかかれた典型的ステンシルプログラム実行中の様子



⇒ **ホストメモリの容量(TSUBAME2.0では計100TB)を活用できれば、さらに高精細なシミュレーション可能！**

メモリ階層としての デバイスメモリとホストメモリ



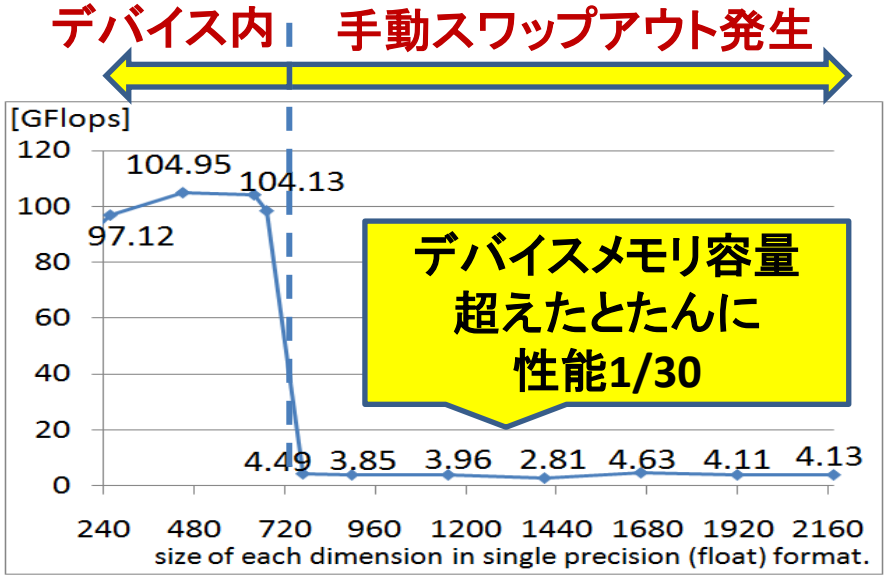
GPUコアから見ると、PCIe(8GB/s)の向こうにある大きいメモリ

デバイスメモリ⇔ホストメモリ間でスワップ機能があれば大容量使えそう

しかし問題！！

- [機能面] 現状GPUにスワップ機能なし
- [性能面] ステンシルで単純にスワップすると性能がひどいことに!

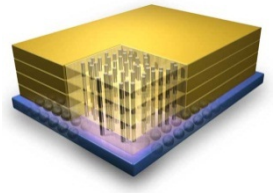
1GPUで3D7点ステンシル



エクサ時代へ向けたメモリ階層の深化
ハイブリッドスパコンだけの問題ではない

Hybrid Memory Cube (HMC)

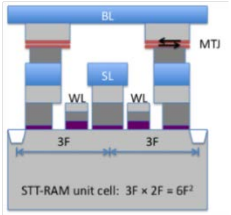
- DRAMチップの3D積層化による高帯域化
- DDRより電力あたり容量は不利
- Micron/Intelなど



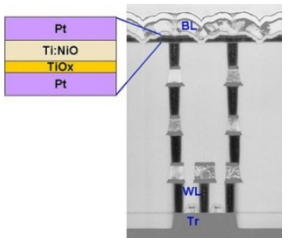
次世代 不揮発メモリ(NVRAM)

- DRAMと異なる記憶方式
- アクセス速度・密度・write耐性まちまち

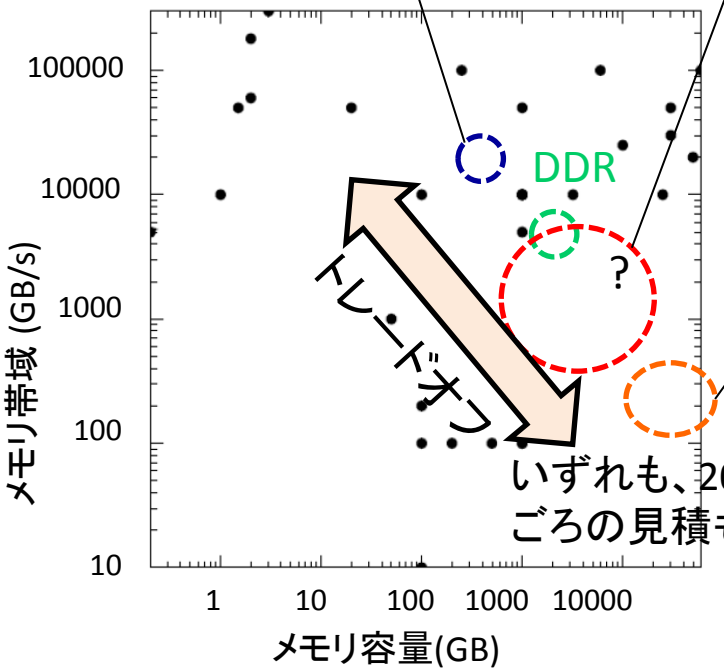
STT-MRAM



ReRAM



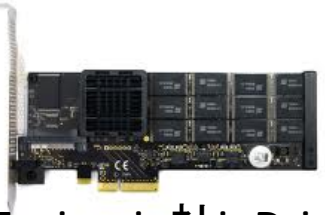
他、PCM, FeRAM・・・



いずれも、2018年ごろの見積もり

高速Flashメモリ

- PCI-Express直接接続・デバイス並列化によりO(GB/s)の帯域
- Solid State Accelerator(SSA)とも



Fusion-io社ioDrive

本研究のゴール

既存マルチGPUコード

＋

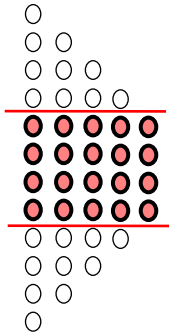
局所性向上技術

＋

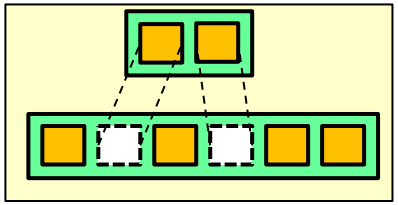
スワップ対応ランタイム

高性能+大容量+高生産性

- MPI+CUDAで記述
- デバイスメモリ内で動作すると仮定



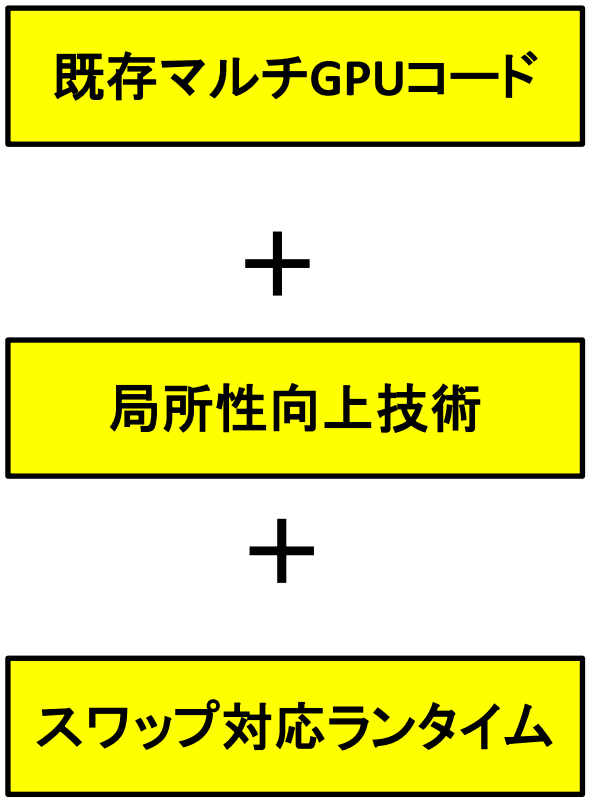
ステンシルでは、**テンポラルブロッキング**が重要。さらなる最適化も
[金・遠藤・松岡 AsHES 13]



HHRT/MC
ライブラリ

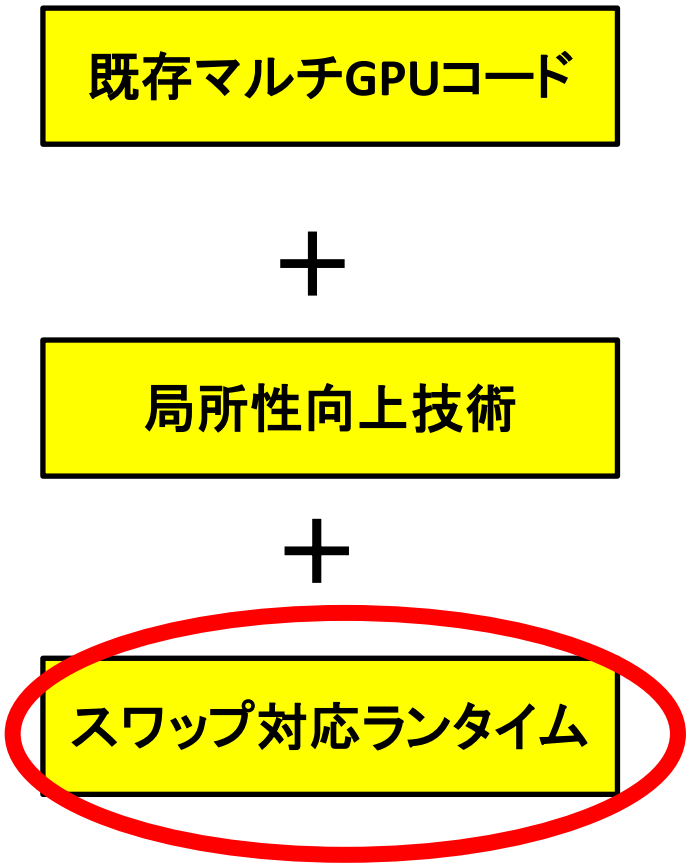
発表の流れ

- HHRT/MCランタイム
  - 実行モデル
  - プロトタイプ実装
- ステンシル計算によるケーススタディ
  - 特に、手動最適化バージョンとの比較
- まとめ



発表の流れ

- HHRT/MCランタイム
  - 実行モデル
  - プロトタイプ実装
- ステンシル計算によるケーススタディ
  - 特に、手動最適化バージョンとの比較
- まとめ



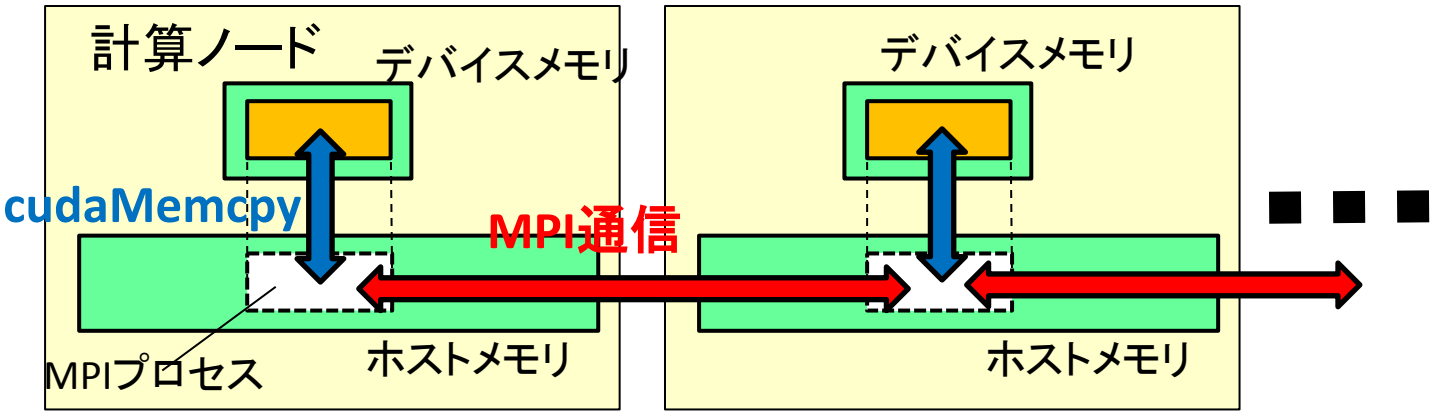
HHRT/MC: メモリ階層を利用可能とするランタイムライブラリ

Hybrid-Hierarchical RunTime on MPI + CUDA

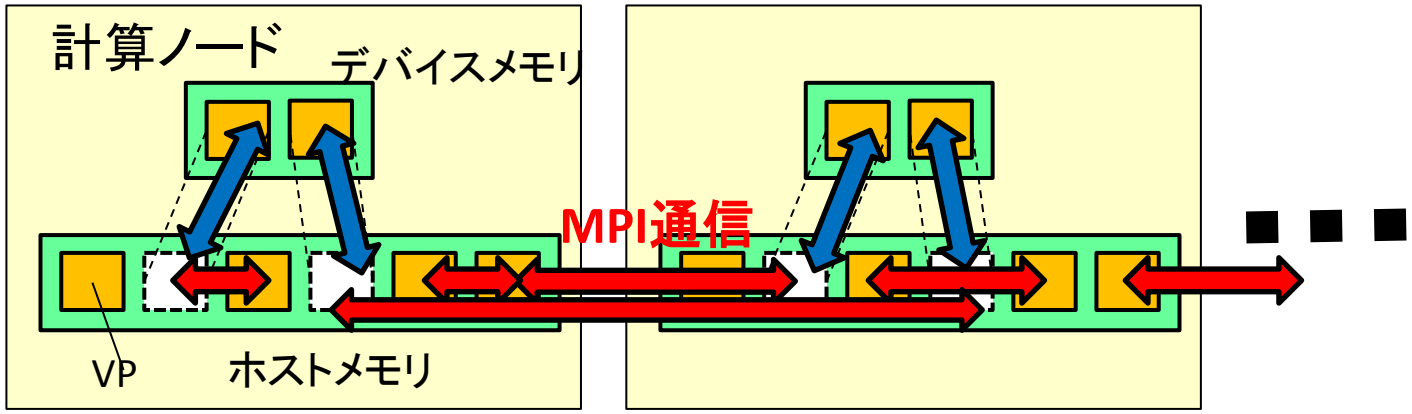
- *何をするか*:
  - MPI + CUDAで記述された既存並列プログラムが、最小限の変更にて動作する
    - 無変更でも動くのが理想だが…
  - 仮想プロセス(VP)によるGPUの多重化
  - スワップ機能により、広いメモリ領域を利用可能に
    - デバイスメモリ – ホストメモリの階層。将来FusionIOも
- *何をしないか*:
  - 新しいプログラミングモデルやDSLの提案ではない
  - ランタイム自身が局所性向上を行うわけではない

HHRT/MCの実行モデル (1)

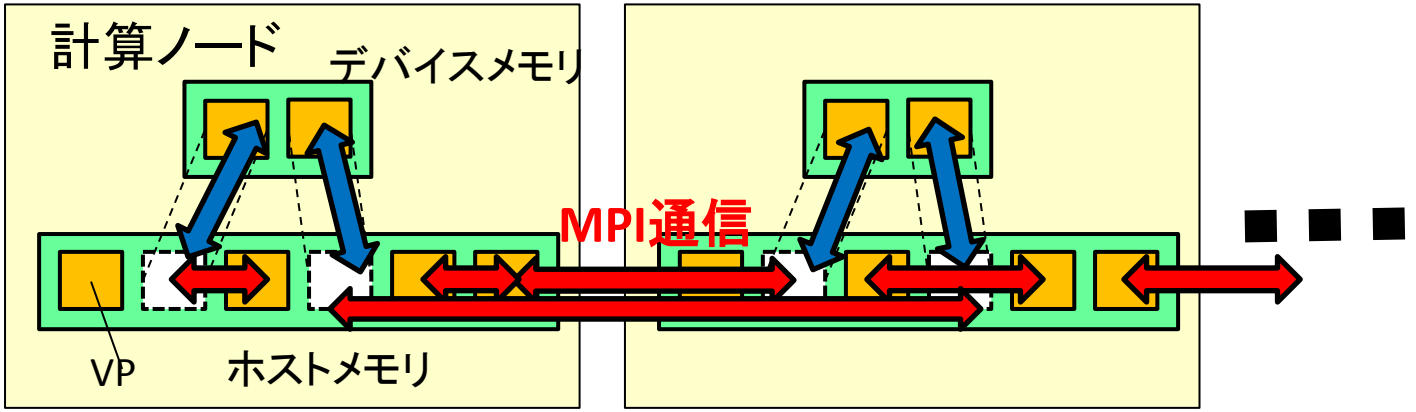
通常のMPI+CUDA実行モデル



HHRT/MCの実行モデル



HHRT/MCの実行モデル (2)



- 処理の単位は仮想プロセス(VP)
- 複数VPが1GPUを共有(多重化)
  - 各VPの利用メモリは、デバイスメモリ内に収まることを前提
- VPは、通常モードまたはスワップアウトモードのいずれかにある
  - スワップアウトされると、利用中メモリはホストメモリへ
  - スワップアウトモードでは、プロセスは進行しない
  - つまり、ページ単位のスワップではない。**VP単位のスワップ**

スワップのルール

VP aは、下記のいずれかが成り立つときスワップアウトされ、スワップアウトモードになる

- VP aがMPI関数を呼び出してブロックした && 条件(#)
- VP aがHH_yield()を呼び出した && 条件(#)
- VP aがcudaMalloc()等を呼び出し、かつメモリ不足を検知

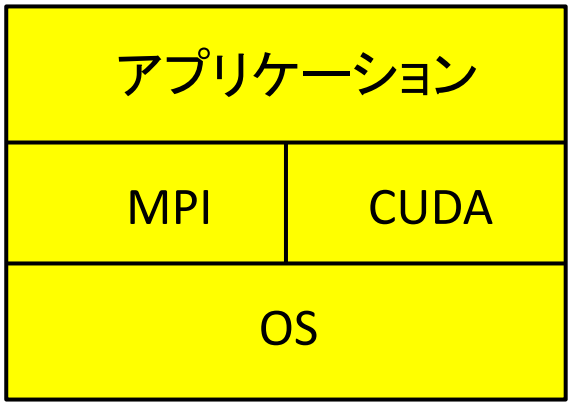
条件(#): VP aと同一GPUを共有するVPのうち、スワップアウトモードにあり、かつ走行可能なVPが存在する。

のちに、十分なメモリの空きができたときにスワップインされ、通常モードに戻り走行再開

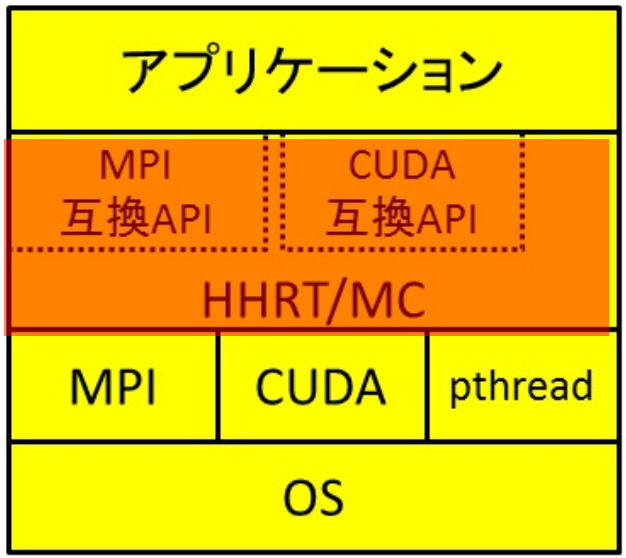
HHRT/MCプロトタイプ版の実装

- アプリからのMPI, CUDA呼び出しをフック
  - MPI, CUDA関数呼び出しの際にスワップの可能性あり
- HHRT/MC自身は、(本物の)MPI, CUDAランタイムAPIを無変更で利用
- VP = pthreadとして実装
  - MPIメッセージのノード内メッセージ配送部分なども実装

通常実行時



HHRT/MC上での実行時



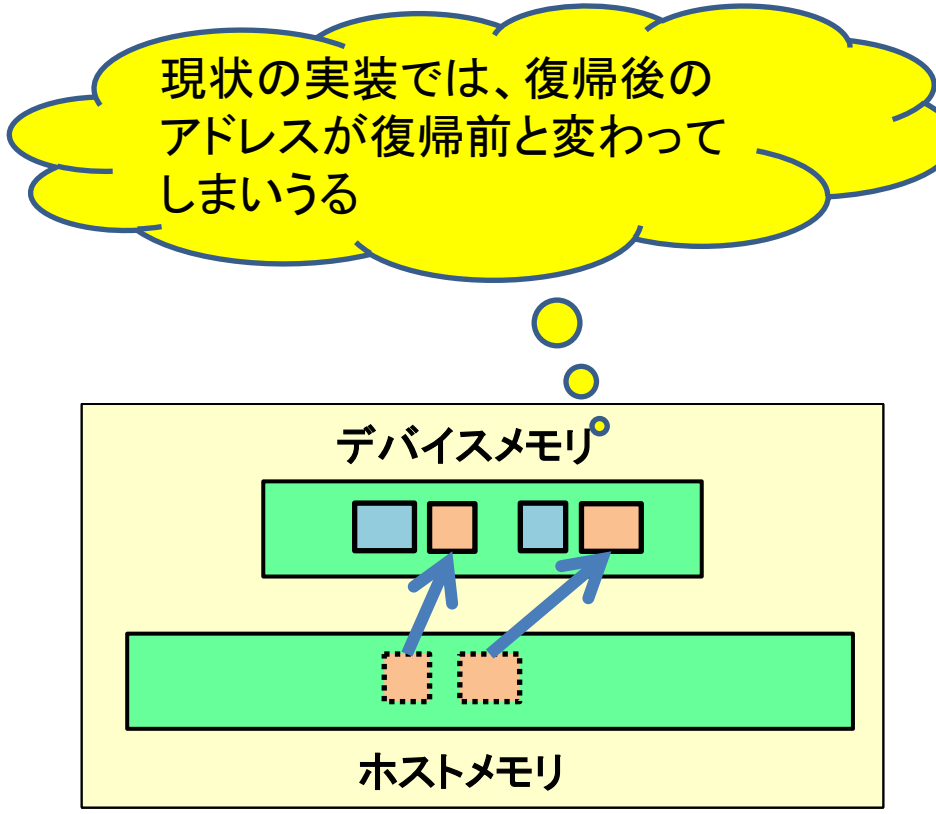
現状のスワップの実装

目的：VPがこれまでに確保したデバイスメモリ領域を、ホストメモリへ追い出し、デバイスメモリ容量を空ける

(1) HHRT/MC内部で各VPが確保した領域リストを管理

(2) VPスワップアウト時
- cudaMemcpyで各領域の内容をホストメモリへ退避
- cudaFreeで容量を空ける

(3) VPスワップイン時
- cudaMallocで必要な容量確保
- cudaMemcpyで各領域の内容をデバイスメモリへ復帰



現状のスワップの実装の問題

- ユーザプログラムから見ると、いつのまにかメモリ領域のアドレスが変わってしまう
- 現状ではユーザの協力を得て復帰
  - HH_registerDevmem(int id, void *ptr)

    cudaMalloc直後に呼んでもらい、メモリ領域と、ユーザの決める整数IDを紐づけ
  - void *HH_findDevmem(int id)

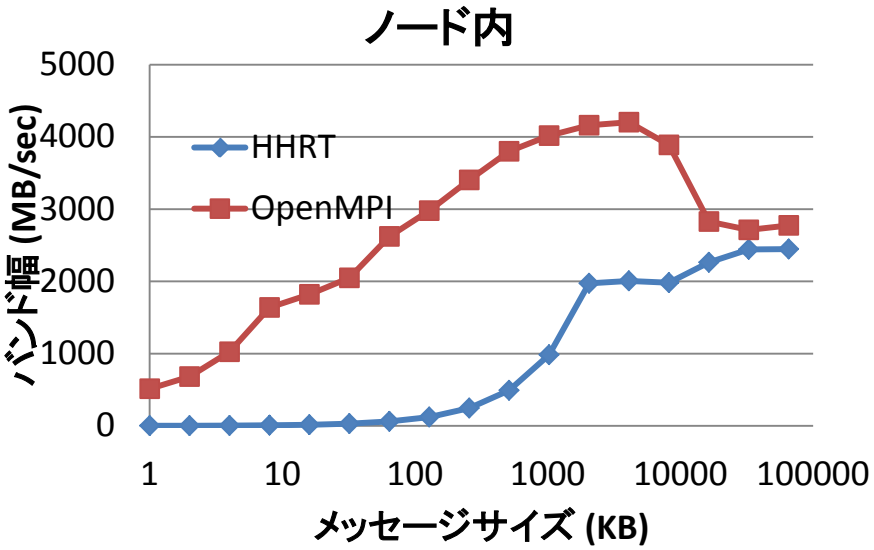
    アドレスが変更されうる箇所の後に呼んでもらい、整数IDからアドレスを取得

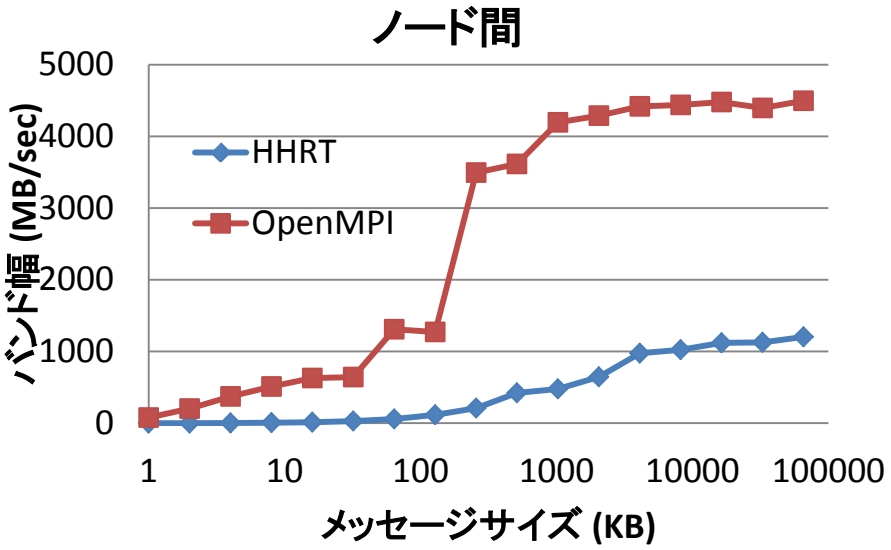
現状必要なユーザプログラムへの変更

- MPI_xx, cudaXxx ⇒ HHMPI_xx, HHcudaXxxに名称変更
- main ⇒ HHmain
- 大域変数に__threadを付ける
- ストリーム0⇒HH_STREAM0
- HH_registerDevmem, HH_findDevmem

MPI通信性能の予備評価

- TSUBAME2.0の1~2ノード利用
- InfiniBand 4xQDR 二本 (理論値8GB/s)
- OpenMPI 1.6.3

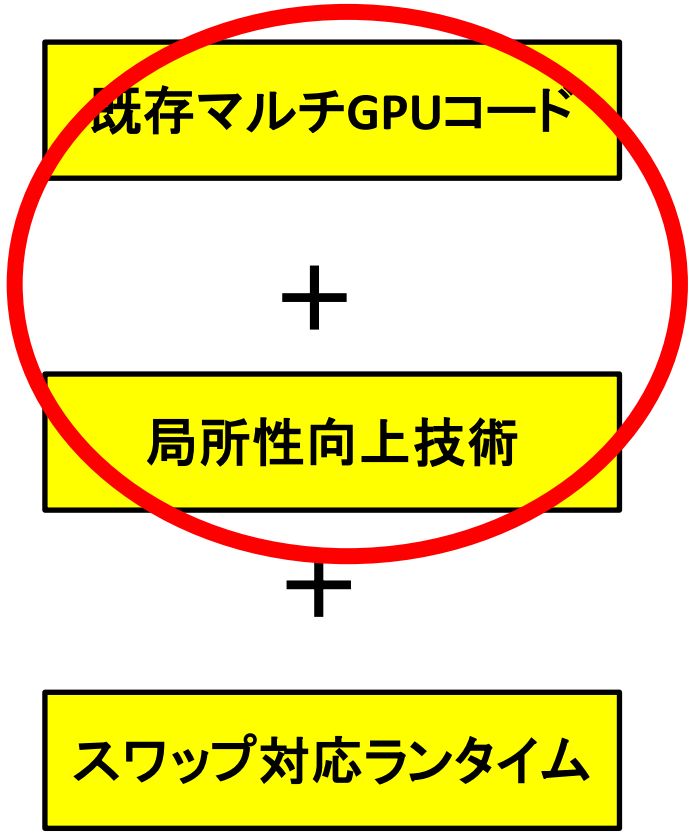




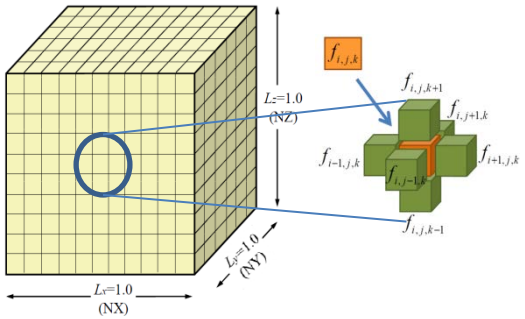
まだまだまだ実装の改良は必要

発表の流れ

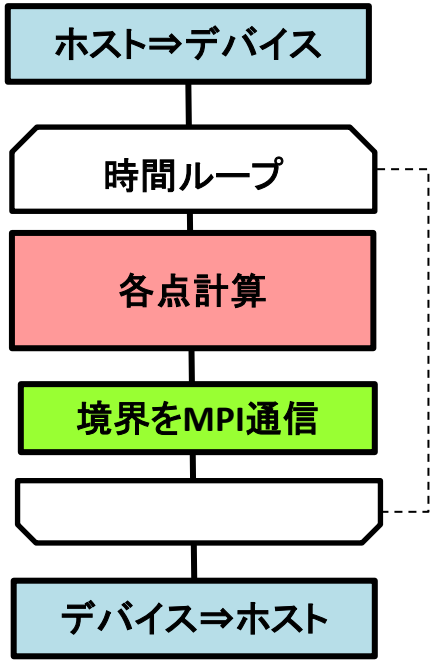
- HHRT/MCランタイム
  - 実行モデル
  - プロトタイプ実装
- ステンシル計算によるケーススタディ
  - 特に、手動最適化バージョンとの比較
- まとめ



単純なステンシル計算 (1)

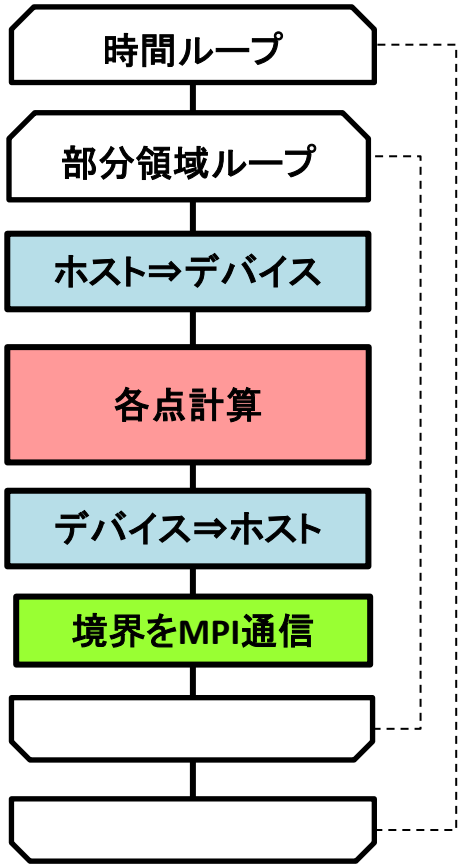


典型的な処理の流れ

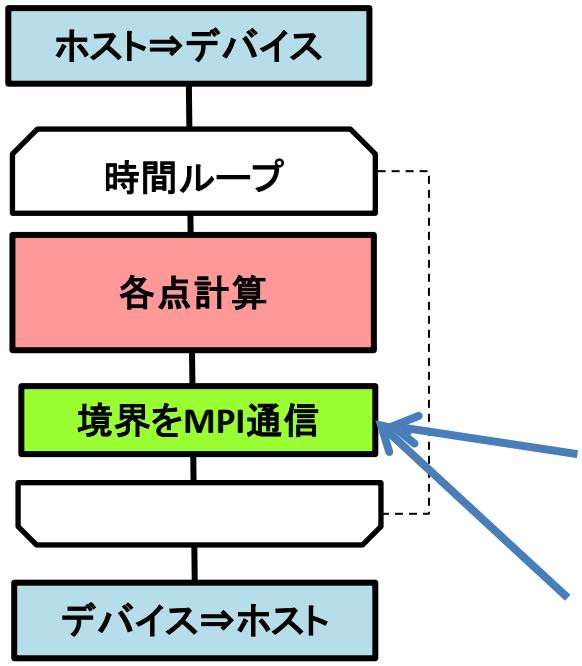


ハンドコーディングで大容量対応するには？

・ 各ランクの担当領域をさらに細切れにし、ちょっとずつGPUへ送って計算



単純なステンシル計算(2)



HHRT/MC上では、図のプログラムが「(ほぼ)そのままで動き」、「大容量対応」可能！

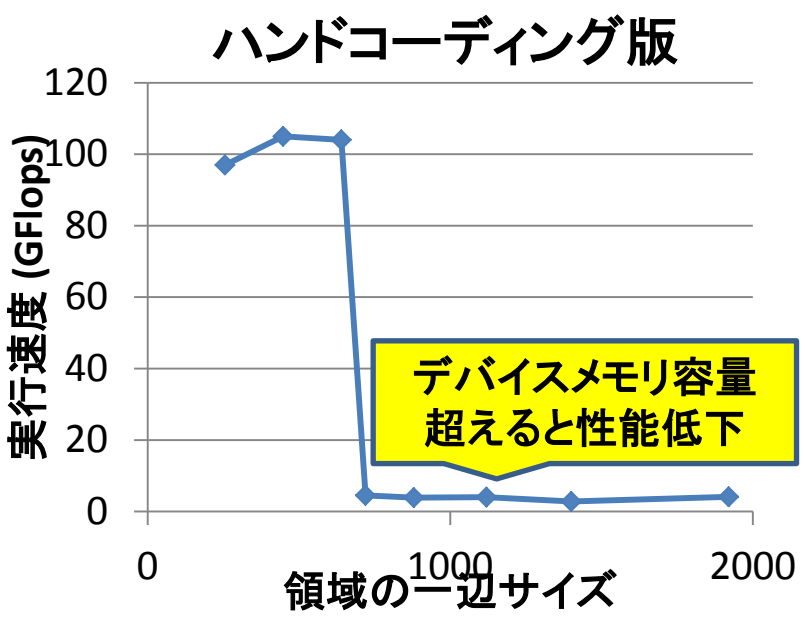
- 実行時にVP数を調整して、

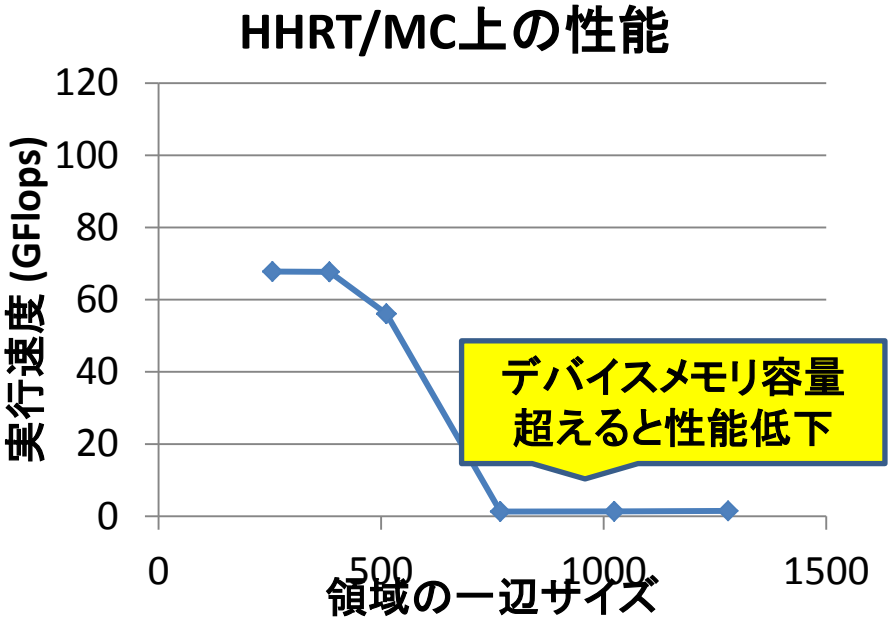
各VPの担当領域＜デバイスメモリサイズにするだけ

- ただし、同一GPU間のVPであってもMPI通信コストはかかる
- MPI通信のときに、こっそりスワップアウトが起こっている

単純なステンシル計算の性能

- TSUBAME2.0上のTesla M2050 GPUを1GPU利用
- CUDA 5.0
- 3D立方体領域で7点ステンシルを実行



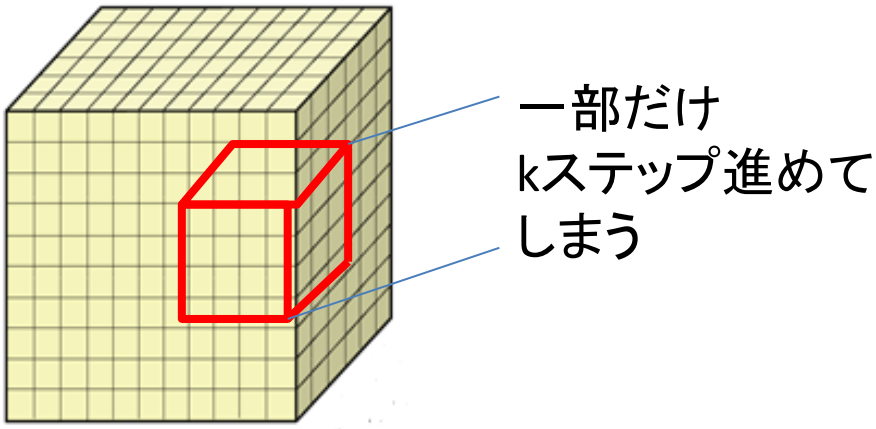


※VP数(多重度)は4~16で良いものを選択

- 単純なステンシルの局所性の低さが問題
- 局所性向上技術との組み合わせが必要

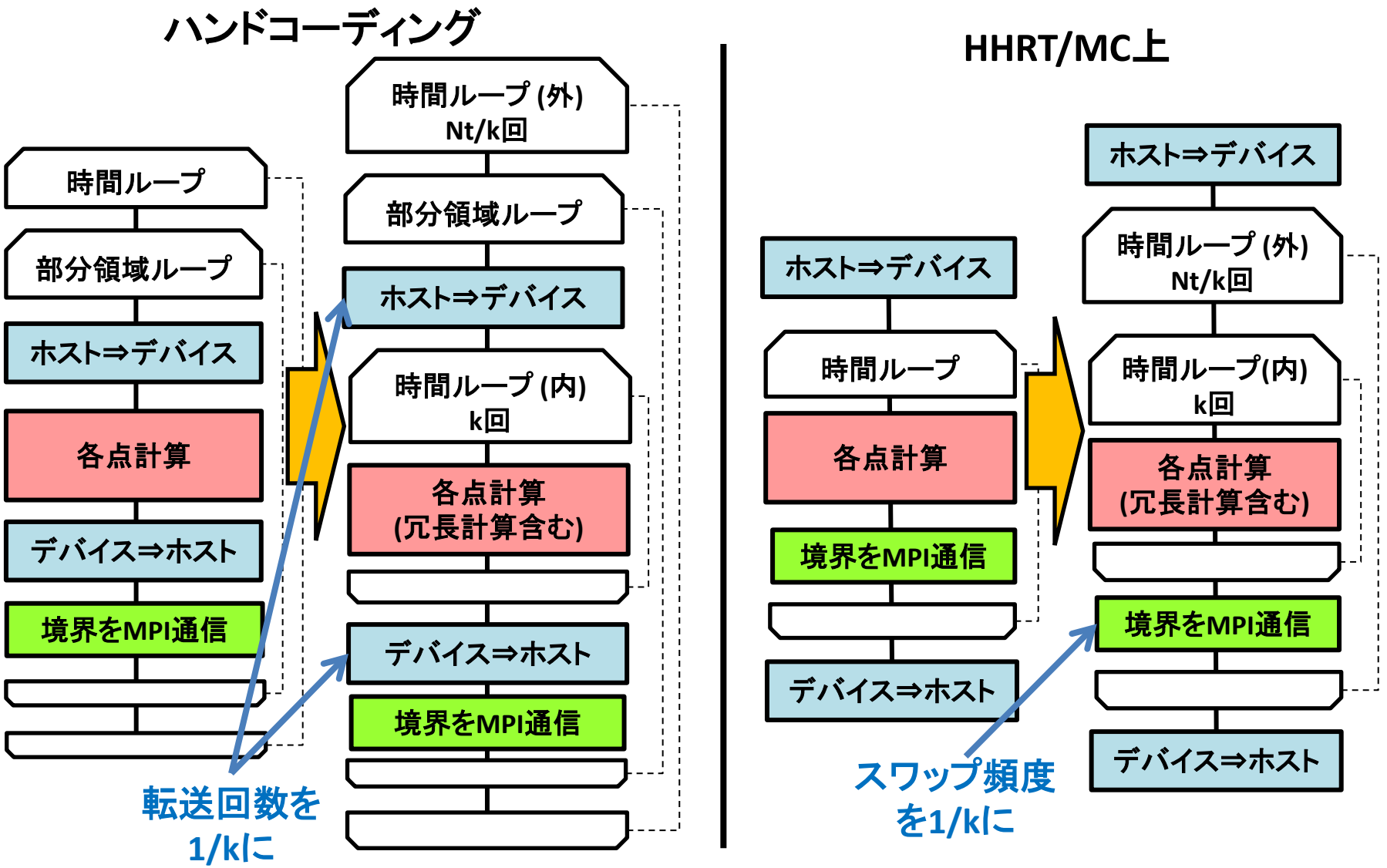
テンポラルブロッキングによる局所性向上

- データがO(1)回だけアクセスされて、デバイスメモリから追い出されるのが問題
- テンポラルブロッキング: 部分領域に対し、複数回時間ステップを進めてしまう [Wittmann 10] [Mattes 10] など
  - 以下、ブロッキング段数をkとする



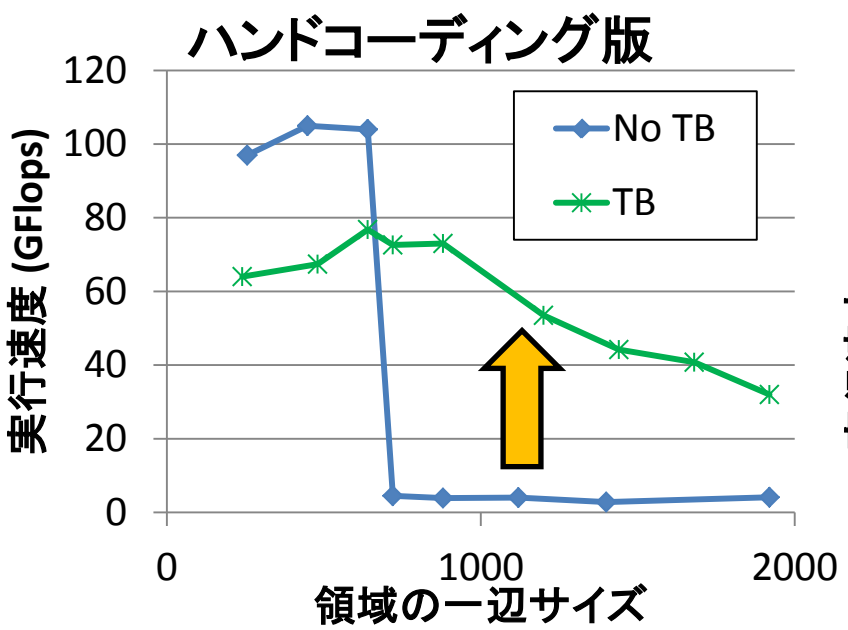
- 密行列演算では古典的な考え方だが、ステンシルならではの難しさあり(後述)

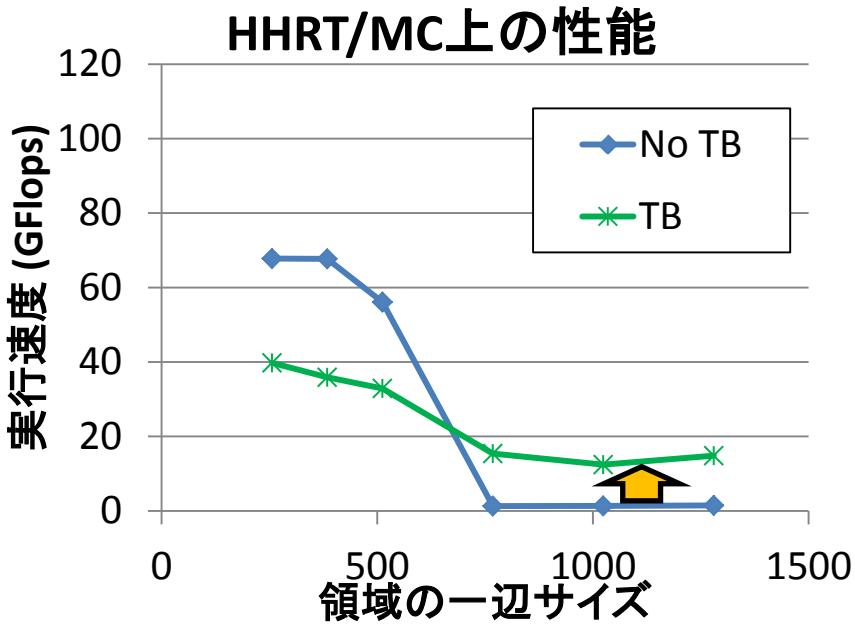
テンポラルブロッキング導入に伴うユーザプログラム変更



テンポラルブロッキング導入時の性能

- Tesla M2050を1GPU用い、3D立方体領域で7点ステンシル
- ブロッキング段数kについては、各条件のパラメータスイープより決定

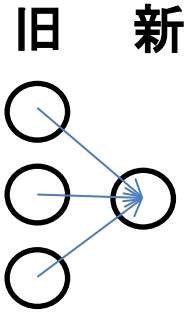




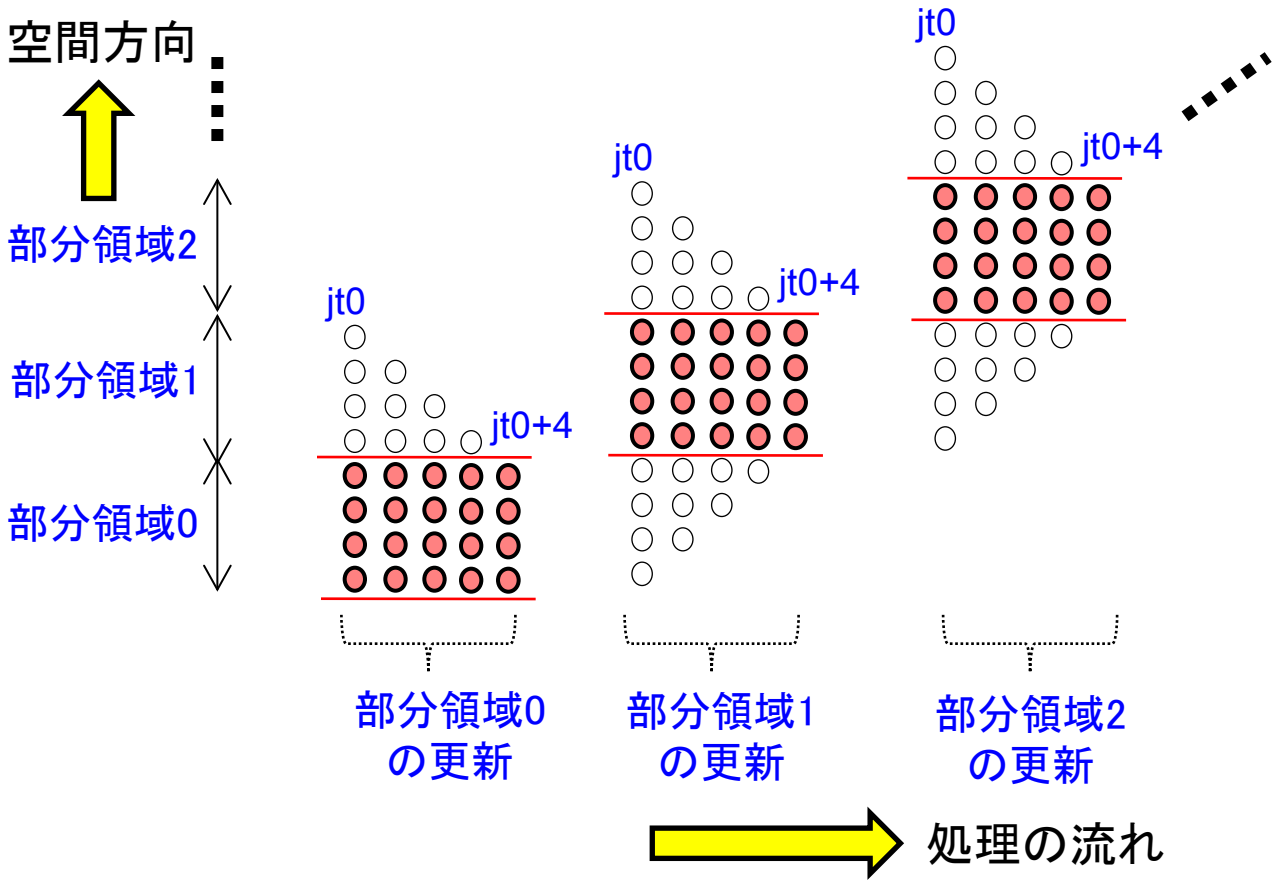
- TBにより、大規模時の性能大幅アップ
  - TBなし・小規模時より遅い理由は次のページにて
- ただし、HHRT/MC上の性能向上がまだまだまだ
  - 必要以上のスワップ回数など観測。解析中

テンポラルブロッキングのコスト

各点の更新処理は、
隣接点に依存



簡単のため、
空間方向は
一次元とした図



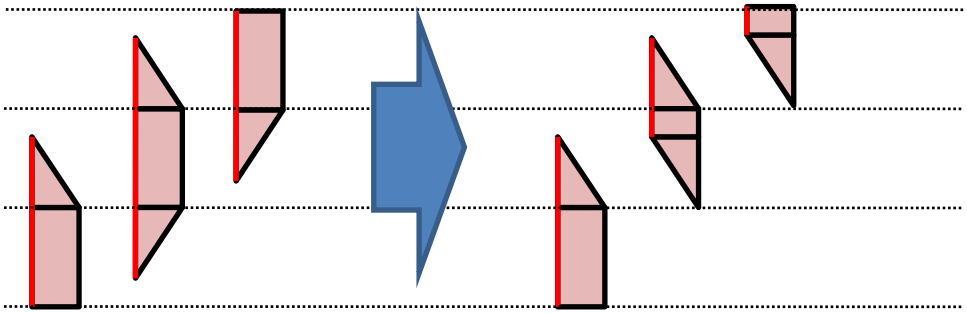
⇒　局所的に複数ステップ計算すると、依存元データと
冗長計算が増大！

キャッシュ⇔メモリの効率利用の場合はk=2~8だが、
PCIeコストを隠ぺいするにはk=30~300の必要

テンポラルブロッキングの最適化手法

[金・遠藤・松岡Hokke12, AsHES13]

- 前の部分領域計算の途中結果を再利用した、冗長計算除去



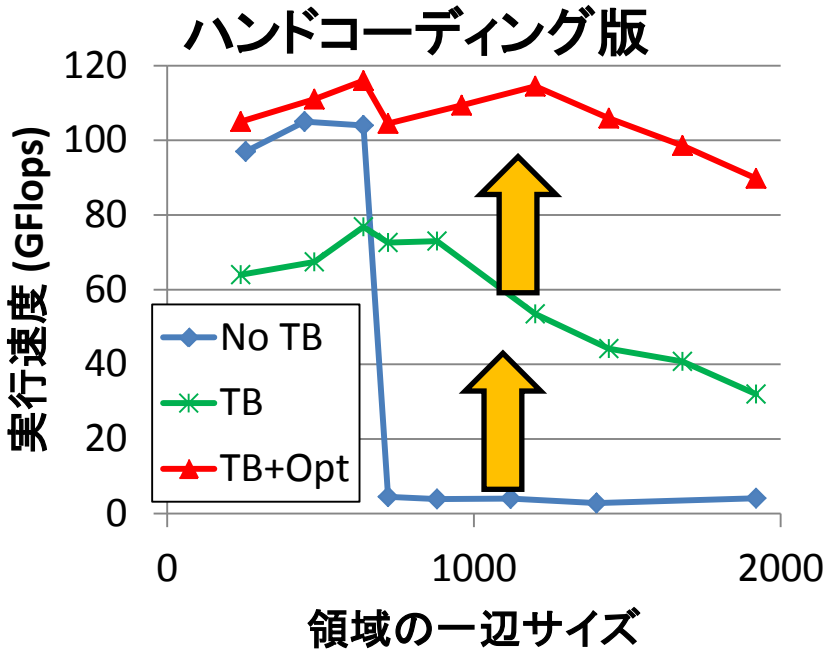
キャッシュ局所性の文脈で、先行研究あり

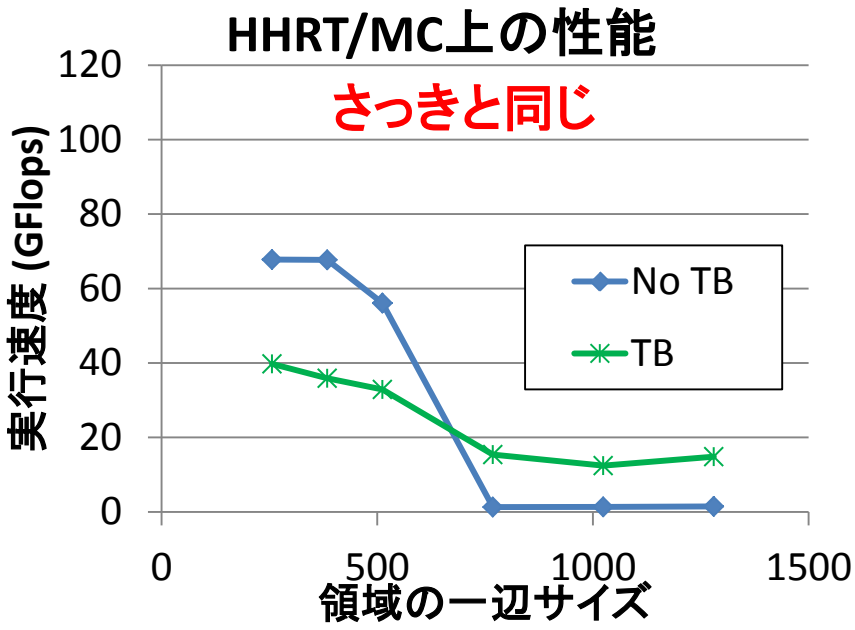
- バッファ利用をシフトすることにより使用メモリ節約

HHRT/MC上では、現在実現困難　→ 今後の課題

最適化版の性能

- Tesla M2050を1GPU用い、3D立方体領域で7点ステンシル
- ブロック段数kについては、各条件のパラメータスイープより決定





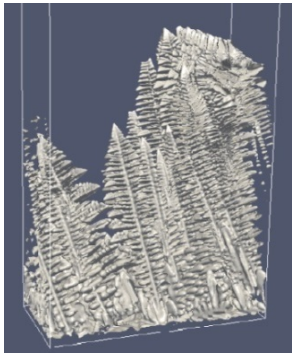
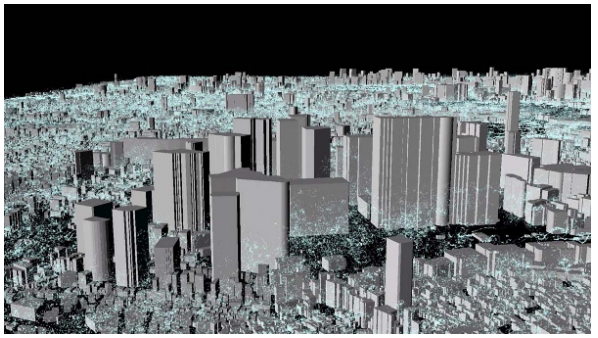
- ハンドコーディング版では更なる高速化がなされ、小規模のケースと遜色ない性能が可能
- HHRT/MC版はまだまだまだまだ

ここまでのまとめ

ゴールは、高性能+大容量+高生産性

- マルチGPUアプリの、問題サイズを大規模化可能とするために、HHRT/MCランタイムを提案
  - GPUデバイスメモリ-ホストメモリ間のスワップにより、大容量ホストメモリを利用可能に
- ケーススタディとして7点ステンシル計算
  - ハンドコーディングでは、高性能と大容量が両立可能
    - マルチGPU・ハンドコーディング版についてはCluster2013で発表予定
  - HHRT/MCの利用により、高生産性をめざしているが、高性能とのトレードオフについて議論した
    - HHRT/MC内部と、ユーザプログラム双方の改良が必要

実アプリケーションへの展開に向けて(1)

- 今後、実用ステンシルアプリへの適用を適用
  - 多数種のカーネル、データ配列
  - 複雑な境界条件

  HHRT/MCでは、時間ループ周りの改造で済むので、改造しやすいことを期待
- テンポラル(iteration)ブロッキングができないケースあり
  - 共役勾配法のように、大域reductionして次ステップに用いるアルゴリズムは×

実アプリケーションへの展開に向けて(2)

Asanovicらによる Motif分類で議論

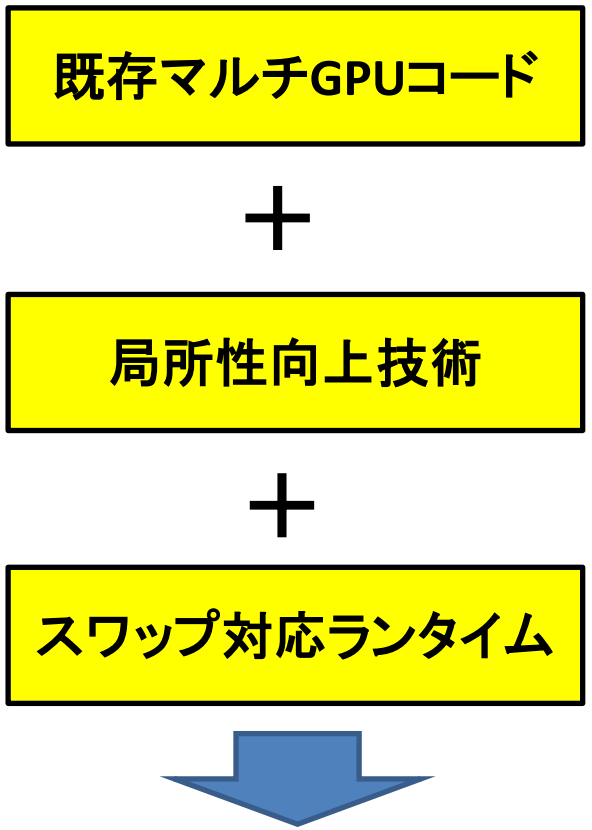
Motif	対応可能性
規則格子 (ステンシル)	○ 共役勾配法など大域依存があると×
不規則格子	上と同様
疎行列・グラフ	上と同様
密行列	多くは局所性高く、○ Hetero Linpack(ハンドコーディング)など
粒子系	大規模メモリを使うケース少？
スペクトル法・FFT	きびしそう
モンテカルロ	各タスクの性質次第

関連研究

- Adaptive MPI on Charm++ [Huang 03]
  - Charm++: 細粒度並列オブジェクトシステム
  - 既存MPIプログラムを、CPU多重化させる
  - それ自身がメモリ階層を考慮したものではない
  - 目的は計算通信のオーバラップ、動的負荷分散、耐故障…

  → HHRTでも上記の利点を活用できると期待
- Physis [丸山11]
  - 並列ステンシル計算のためのDSL。Forallタイプ
  - 既存アプリがすでにある場合は、Physis向けに書き直し必要
  - Physis内に局所性向上技法を組み込むのは有望
- Communication Avoiding Algorithm [Demmelグループ]
  - 種々の計算について、データ移動削減・局所性向上の試み
  - CAQRアルゴリズム
  - テンポラルブロッキングもこの範疇

おわりに



高性能+大容量+高生産性

今回は、下記を対象に評価

- アプリ：ステンシル
- モデル：MPI+CUDA
- システム：GPUクラスタ
- メモリ：GDDR+DDR

将来、ポストペタスケール時代に向け様々な環境に対応

PGAS, OpenACC,

Xeon Phi, 汎用CPU,

Flash, HMC, ReRAM…