# 第５回先進スーパーコンピューティング環境研究会（ASE 研究会）開催報告

東京大学情報基盤センター　特任准教授　片桐孝洋

2009 年 8 月 21 日（金）14 時 00 分から 17 時 30 分まで、東京大学情報基盤センター大会議室にて、第 5 回先進スーパーコンピューティング環境研究会（ASE 研究会）が開催されました。国内の大学、研究機関、および企業からの参加者が 23 名ありました。活発な議論がなされました。

第 5 回 ASE 研究会の招待講演として、電気通信大学情報システム学研究科の大島聡史　博士をお呼びしました。大島博士は、GPU を用いた汎用処理、および並列プログラミング言語 OpenMP から GPU で実行可能なコードを自動生成するコンパイラの研究をなされております。講演は大変興味深い内容でした。また、会場から活発な質疑応答がなされました。なお、講演内容の詳細につきましては、添付の参考資料をご覧ください。

同時に、東京大学情報基盤センターの中島研吾教授による「拡張階層型領域間境界分割に基づく並列前処理手法」の講演、そして著者らによる「Xabclib ： 汎用な自動チューニングインターフェース OpenATLib を利用した疎行列反復解法ライブラリ」の講演があり、双方とも活発な質疑応答がなされました。

ASE 研究会の開催情報はメーリングリストで発信をしております。研究会メーリングリストに参加ご希望の方は、ASE 研究会幹事の片桐（katagiri@cc.u-tokyo.ac.jp）までお知らせください。


図１　会場の様子

# 第5回先進スーパーコンピューティング環境研究会（ASE研究会）発表資料

ASE研究会幹事　片桐孝洋

2009年8月21日（金）14時00分から17時30分まで、東京大学情報基盤センター大会議室にて、第5回先進スーパーコンピューティング環境研究会（ASE研究会）が開催されました。本号では、大島博士が発表した内容を資料として掲載させていただきます。

大島博士の講演は、「GPUとスーパーコンピュータを用いた高性能計算」と題しまして、近年注目が高まっているGPUを用いた汎用演算に関する講演でした。現在では、小型モバイルノートからスーパーコンピュータまで様々な計算機にGPUが搭載されており、GPGPU（GPUによる汎用計算処理）を利用可能な計算機環境の普及が進んでいます。

特にGPGPUプログラミング環境での計算機言語CUDAを用いてアプリケーションを高速化する研究が多数なされており、プログラミングの難しさや手間を削減するためのGPGPUプログラミング環境に関する研究も進められています。この講演では、GPGPUの研究動向について、また、大島博士らが研究を進めているGPGPUプログラミング環境の研究について講演がなされました。さらに、計算機センター等のスーパーコンピュータにGPUを導入する際の課題や展望について述べられました。

ASE研究会では、GPGPU環境のような先進スーパーコンピューティング環境を支援する研究話題の提供を目的に、年数回の活動を計画しております。これからもご支援のほどを、よろしくお願い申し上げます。

# GPUとスーパーコンピュータを用いた高性能計算

大島 聡史

電気通信大学 情報システム学研究科 博士研究員
JST CREST



## アウトライン

1.GPGPUの概要と研究動向
- GPUとGPGPUの概要
- GPGPUの研究事例やプログラミング方法

2.GPGPUプログラミング環境に関する研究
- 我々が行っている研究の紹介

3.GPU(GPGPU)とスーパーコンピュータ
- 現在および近い将来の課題や展望



## アウトライン

1.GPGPUの概要と研究動向
- GPUとGPGPUの概要
- GPGPUの研究事例やプログラミング方法

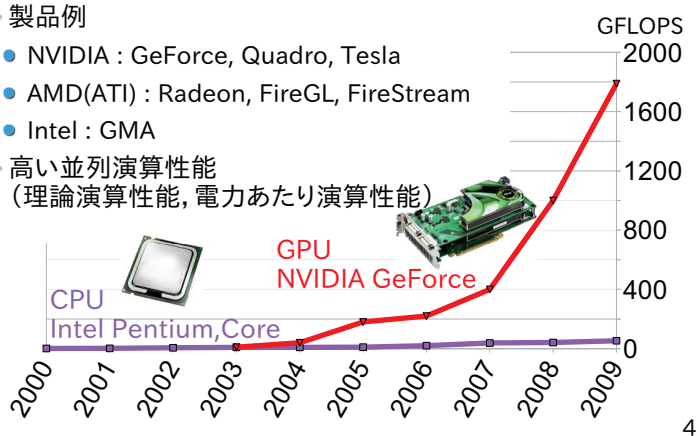
2.GPGPUプログラミング環境に関する研究
- 我々が行っている研究の紹介

3.GPU(GPGPU)とスーパーコンピュータ
- 現在および近い将来の課題や展望



## GPU(Graphics Processing Unit)

- 画像処理用のハードウェア：LSIチップ, ビデオカード
- 製品例
  - NVIDIA : GeForce, Quadro, Tesla
  - AMD(ATI) : Radeon, FireGL, FireStream
  - Intel : GMA
- 高い並列演算性能
  (理論演算性能, 電力あたり演算性能)

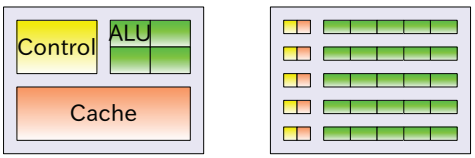





## CPUとGPUの違い(2008年後半時点 ハイエンド製品)

| | CPU<br>Intel Xeon X5482 | GPU<br>NVIDIA GeForceGTX280 |
|---|---|---|
| 演算器(コア)数 | 4 | 240 |
| クロック周波数 | 3.20 GHz | 1296 MHz |
| 搭載メモリ容量 | — | 1 GB |
| メモリ帯域幅 | 12.8 GB/s | 141.7 GB/s |
| 最大消費電力 | 150 W | 236 W |
| 理論演算性能 | 51.2 GFLOPS | 933 GFLOPS |
| ワット当たり性能 | 0.34 GFLOPS/W | 3.95 GFLOPS/W |
| CPUとの接続 | — | PCIe x16 Gen2(16GB/s) |

CPUとGPUの
トランジスタ配分
(CUDA Programming
Guide より)



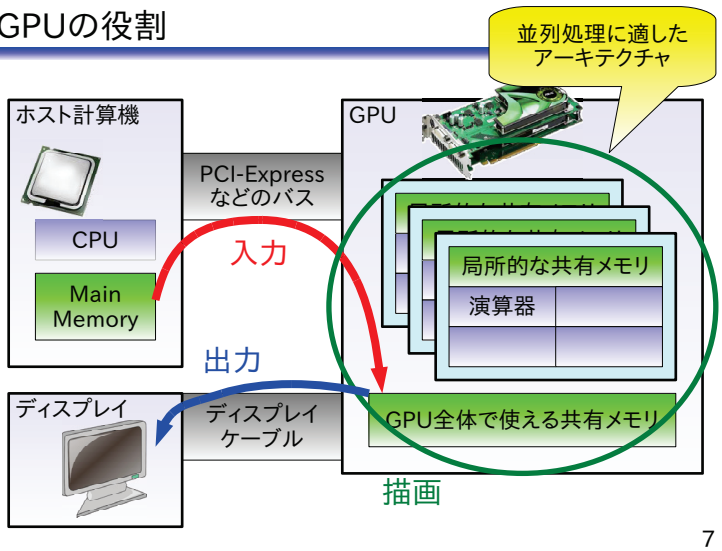



## CPUとGPUの違い(2009年前半時点 ハイエンド製品)

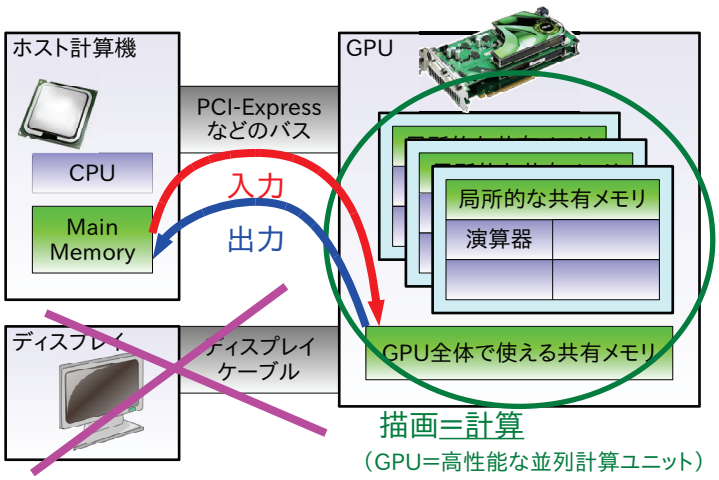
| | CPU<br>Intel Core i7-975 | GPU<br>NVIDIA GeForceGTX295 |
|---|---|---|
| 演算器(コア)数 | 4 (HT 8) | 480 (240*2) |
| クロック周波数 | 3.33GHz | 1242 MHz |
| 搭載メモリ容量 | — | 1792 MB (896MB*2) |
| メモリ帯域幅 | 25.6 GB/s | 223.8 GB/s |
| 最大消費電力 | 130 W | 289 W |
| 理論演算性能 | 53.28 GFLOPS ※ | 1788.48 GFLOPS |
| ワット当たり性能 | 0.41 GFLOPS/W | 6.19 GFLOPS/W |
| CPUとの接続 | — | PCIe x16 Gen2(16GB/s) |

※HTを無視した場合の値

## GPUの役割






## GPGPU（General-Purpose computation using GPUs）

- 『GPUの高い演算性能を，画像処理以外の処理にも活用する』
- GPGPUに関する研究事例
  - アプリケーションの高速化
    - 数値計算：行列積やFFTなど
    - シミュレーション：流体や粒子など
    - マルチメディア処理：動画エンコードなど
    - 可視化を伴うもの：画像処理と非画像処理の組み合わせ
    - （ゲーム：高度な描画処理と物理計算などの組み合わせ）
  - システム
    - 性能のモデル化，性能解析，省エネルギー最適化：マルチコアCPUやCell B.E.などとの比較も
    - GPGPUプログラミング環境：プログラミング言語，ライブラリ

☆高い並列性を持つもの
☆可視化を伴うもの



## GPGPUの利用方法（プログラミング環境）

- 従来手法：グラフィックスプログラミング
  - グラフィックスAPI（DirectX, OpenGL）
    ＋シェーダ言語（HLSL, GLSL）
    - 描画用の記述を用いて汎用演算を記述
      → 直接的でない，性能チューニングが難しい，記述量が多い
- 現在の主流：CUDA (CUDA Unified Device Architecture, NVIDIA)
  - NVIDIA GPU向けのアーキテクチャとプログラミング環境
    - GPUのアーキテクチャを低いレイヤで定義・公開
    - C言語を拡張した記述によるプログラミングが可能
- etc.：ストリーミング言語など
  - GPUを対象としたさまざまな並列化プログラミング言語
    - Brook+, SPRAT, ATI Stream, etc.
    - マルチコアCPUやCell/B.Eと共通に利用可能なものも



## GPGPUプログラムの例 1/2：シェーダ言語(OpenGL+GLSL)

GPU上で動作（計算内容）

シェーダ言語を用いた
配列加算（vc = a*va + b*vb）の例

```
void gpumain(){
  vec4 ColorA = vec4(0.0, 0.0, 0.0, 0.0);
  vec2 TexA = vec2(0.0, 0.0); vec2 TexB = vec2(0.0, 0.0);
  TexA.x = gl_FragCoord.x;     TexA.y = gl_FragCoord.y;
  TexB.x = gl_FragCoord.x;     TexB.y = gl_FragCoord.y;
  ColorA = texRECT( texUnit0, TexA ); ColorB       = texRECT( texUnit1, TexB );
  gl_FragColor          = F_ALPHA*ColorA + F_BETA*ColorB;
}
```

シェーダ言語（GLSL)プログラム

CPU上で動作（GPUの制御やCPU-GPU間の通信など）

```
void main(){
  glutInit( &argc, argv );
  glutInitWindowSize(64,64);glu
  glGenFramebuffersEXT(1, &g
  glBindFramebufferEXT(GL_FR
  glGenTextures(4, g_nTexID); /
  glBindTexture(opt1, texid);
  glTexParameteri(opt1, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(.....);
  glTexImage2D(opt1, 0, opt2, width, height, 0, GL_RGBA, GL_FLOAT, 0);
    ……（以下省略）
```

- グラフィックスプログラミングの知識やGPUアーキテクチャに関する知識が必要
- 記述量が多い
- 描画処理と対象問題との対応付けが重要

OpenGLを用いたC言語プログラム



## GPGPUプログラムの例 2/2：CUDA

GPU上で動作（計算内容）

CUDAを用いた
配列加算（vc = a*va + b*vb）の例

```
__global__ void gpumain
 (float* fOut, float* fIn1, float *fIn2, int nSize, float a, float b){
   int i;
   for(i=0; i<nSize; i++){
      fOut[i] = a*fIn1[i] + b*fIn2[i];
   }
}
```

- 言語仕様やGPUアーキテクチャに関する知識は必要
- 記述量はそれなり
- 並列度を上げることや階層的なメモリの活用が重要

CPU上で動作（GPUの制御やCPU-GPU間の通信など）

```
void main(){
   CUT_DEVICE_INIT();
   cudaMalloc((void**)&d_sd1, n);
   cudaMemcpy(d_sd1, h_sd1, n, cudaMemcpyHostToDevice);
   test<<< grid, threads >>>(d_sd2, d_sd1, d_sd2, nSize, alpha, beta);
   cudaMemcpy(h_sd2, d_sd2, n, cudaMemcpyDeviceToHost);
}
```

## アウトライン

1.GPGPUの概要と研究動向
- GPUとGPGPUの概要
- GPGPUの研究事例やプログラミング方法

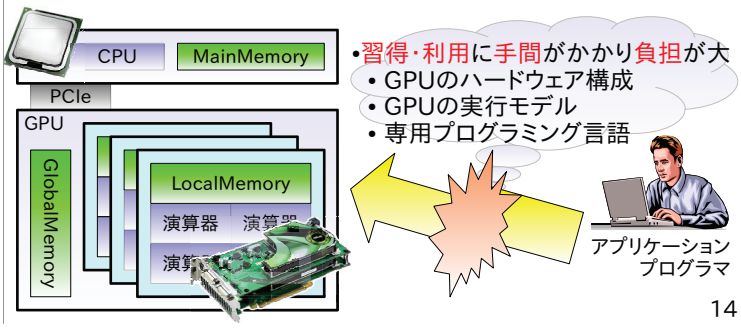
2.GPGPUプログラミング環境に関する研究
- 我々が行っている研究の紹介

3.GPU（GPGPU）とスーパーコンピュータ
- 現在および近い将来の課題や展望



## GPGPUの課題

- プログラム作成（GPGPUプログラミング）の難しさや手間
  - GPGPUによる並列高速化の事例が増加
    ⇒ さらなる普及には難しさや手間の削減が必要
  - 原因：ハードウェアアーキテクチャの違い，ハードウェアアーキテクチャに依存した専用の開発環境（言語・ライブラリ）






## 研究の目的とアプローチ

- 目的（要求）
  - アプリケーションプログラマがGPGPUを容易に活用できるようにする
- アプローチ
  - GPGPU向けの並列化プログラミング言語を作成する
    - ○ 設計と実装次第で良い性能が期待できる
      （処理速度，使いやすさ，etc.）
    - △ 習得の手間が大きく使いにくい，アーキテクチャによる影響を受けやすい（実装も大変？）
  - CPU向けの並列化プログラミングに利用されている汎用プログラミング環境（言語・ライブラリ）をGPGPUにも使えるようにする
    - ○ 習得の手間が小さく使いやすい，CPUプログラムとGPUプログラムを同じ記述方法で書ける
    - △ 適当な環境があるか，性能が出るか

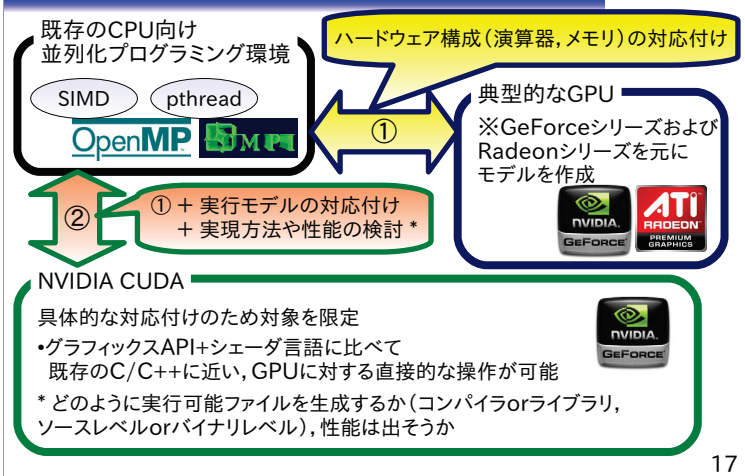


## 既存の並列化プログラミング環境の確認

- 要求：既存のCPU向け並列化プログラミング環境を用いてGPGPUプログラミングを行いたい
  - （容易になったとはいえ）CUDAプログラミングは難しく手間もかかる，習得・利用の手間を削減したい
- 現状：異なる環境が提供・利用されている
  - 既存の並列処理ハードウェア
    - SIMD対応CPU，SMP（マルチコアCPU），PCクラスタなど
  - 既存のCPU向け並列化プログラミング環境
    - SIMD関数，pthread，OpenMP，MPIなど
  - GPGPUプログラミング環境
    - グラフィックスAPI+シェーダ言語，CUDAなど
- 問題：ハードウェアアーキテクチャや実行モデルが異なる
  - 対応付けの検討や処理系の実装が必要



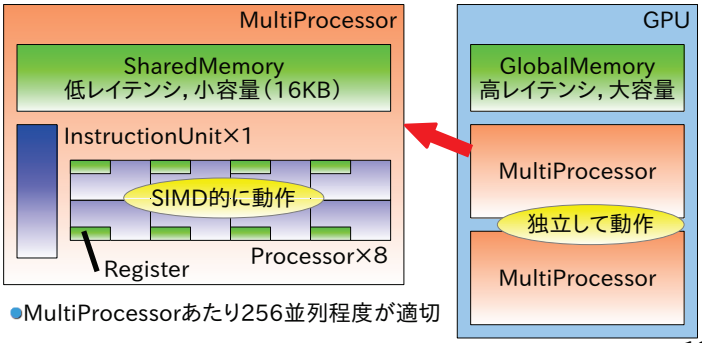
## 既存のCPU向け並列化プログラミング環境とGPGPU・CUDAの対応付け 1/2





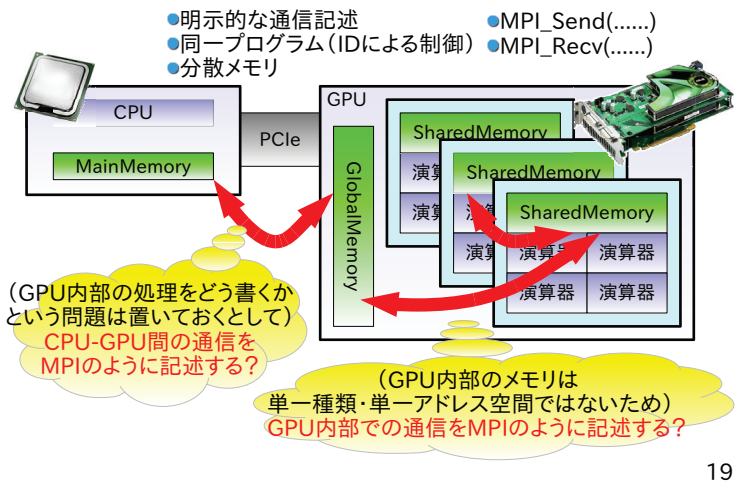

## CUDAの特徴

- アーキテクチャ：演算器とメモリの階層性
- プログラミング：Cを拡張した言語仕様，IDを用いた実行制御，階層性・複数メモリの活用，高い並列度での実行




- MultiProcessorあたり256並列程度が適切

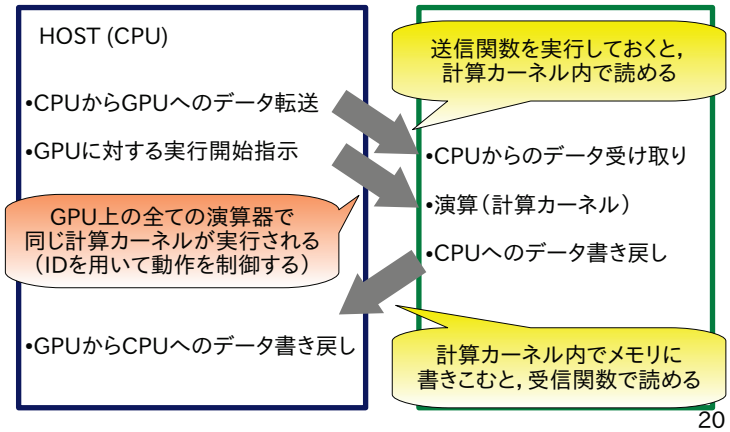
## 対応付け検討の例：MPI と CUDA 1/2





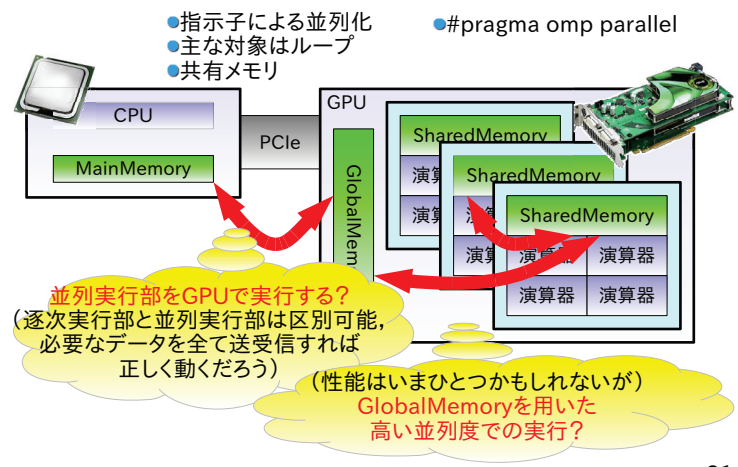

## 対応付け検討の例：MPI と CUDA 2/2

▶ 問題：実行モデルの不一致



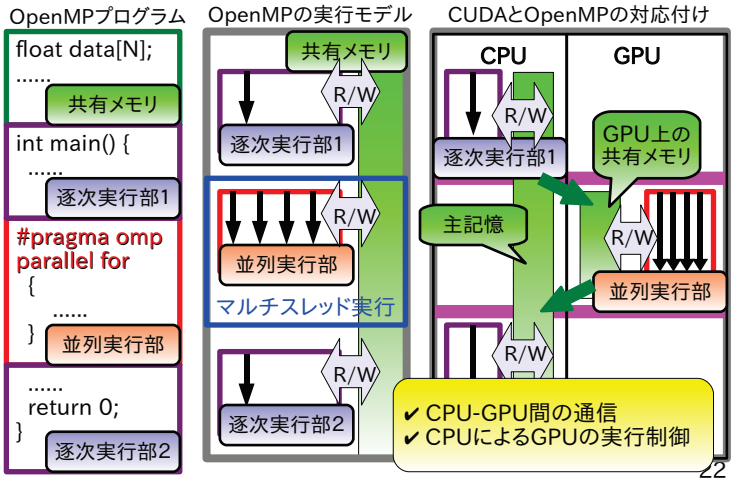



## 対応付け検討の例：OpenMP と CUDA 1/2






## 対応付け検討の例：OpenMP と CUDA 2/2






## OMPCUDAの提案と実装

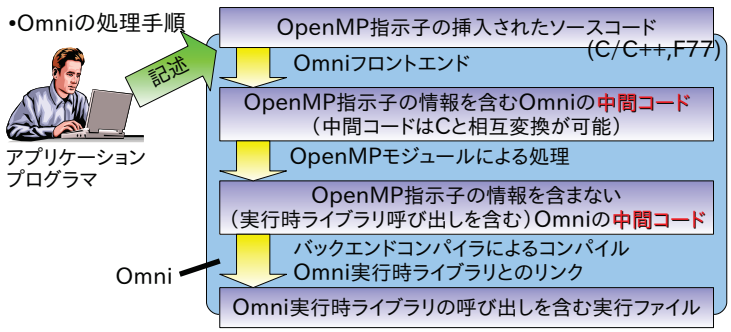
▶ OpenMPを用いたGPGPUプログラミング
- OMPCUDA：OpenMP for CUDA
- 対応付けに基づく「OpenMPを用いて記述されたプログラムの並列実行部をGPU（CUDA）上で実行可能とする処理系」

▶ 対象とする範囲・方針
- 典型的なOpenMPプログラムへの対応を最優先
  - ループの並列化：for指示子，DO指示子
    - 高い並列性（forループを高い並列度で実行）による高性能を目指す
  - リダクション演算（総和計算）：reduction指示子



## OMPCUDAの実装 1/2（全体の構成）

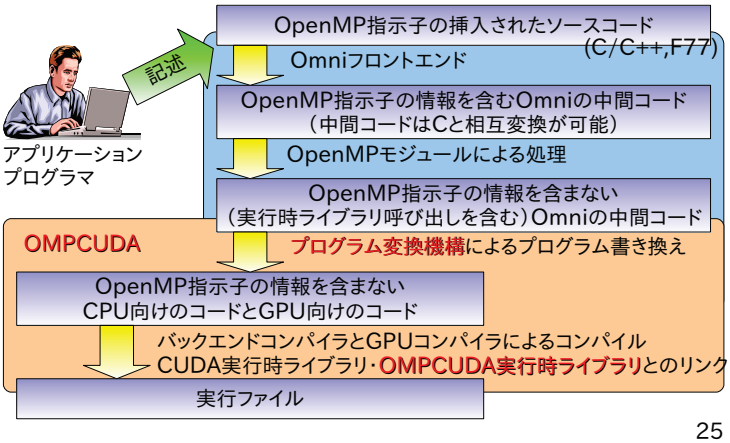
▶ Omni OpenMP Compiler† （以下Omni）をベースに実装




†M.Sato, S.Satoh, K.Kusano, and Y.Tanaka. Design of OpenMP Compiler for an SMP Cluster. In EWOMP '99, pp. 32–39 (1999).

## OMPCUDAの実装 1/2（全体の構成）

•OMPCUDAの処理手順






## OMPCUDAの実装 2/2

1.プログラム変換機構

- 並列実行部で利用する共有変数を確認し，スレッド割り当て部分を書き換えてデータの送受信やGPU呼び出しに書き換える
  - アプリケーションプログラマはCPU−GPU間のデータ通信等を気にする必要無し（手間の削減）

2.OMPCUDA実行時ライブラリ

- Omniの実行時ライブラリが行う処理をCUDA向けに再実装
- 逐次実行部
  - スレッドの制御：プログラム変換機構によって書き換え済
  - 初期化処理：OMPCUDA実行時ライブラリの機能として実装
- 並列実行部：OMPCUDA実行時ライブラリの機能として実装
  - forループのスケジューリング（実行時のループパラメタ書き換え）
  - リダクション演算：一部処理を逐次実行部で行う最適化の余地も確認



## 評価実験

▶ 評価項目

1.並列実行の効果が得られるか（性能確認）

2.利用の手間に関する評価

▶ 評価環境

| | |
|---|---|
| CPU | Intel Xeon E5345 2.33GHz 4コア |
| メインメモリ | 4.0GB (DDR2-800MHz) |
| GPU | NVIDIA GeForce GTX 280 240SP<br>コアクロック 604MHz, SPクロック 1.30GHz |
| ビデオメモリ | 1.0GB (GDDR3-1107MHz) |
| GPU接続バス | PCI-Express Gen2 x16 |
| OS | CentOS 5.0 (kernel 2.6.18) |
| コンパイラ | GCC 3.4.6<br>Omni OpenMP compiler version 1.6<br>nvcc 2.0 v0.2.1221 |



## 評価対象とする問題

▶ 位置づけ

- 並列性の高いプログラムを容易に並列高速化できるかの確認

▶ 対象問題：行列積

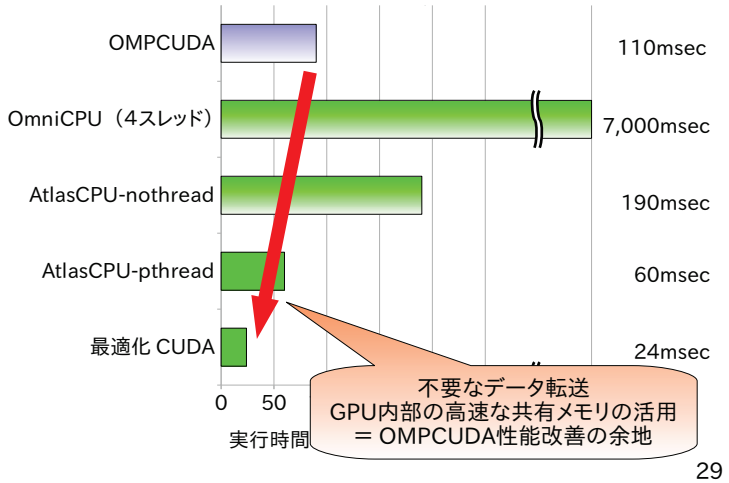
- 単純な3重ループプログラムの外側2ループを展開し，2重ループ化したもの（並列度を上げるため）

▶ 比較条件

- 実行時間測定範囲
  - GPU（CUDA）の初期化は含まない（1秒程度必要）
  - CPU-GPU間の通信を含む
- 比較対象
  - OmniCPU（CPUでのOpenMP並列化，OMPCUDAと同一ソース）
  - AtlasCPU（ATLASを使用）
  - 最適化CUDA（CUDAサンプルより，GPU上の高速共有メモリを活用）

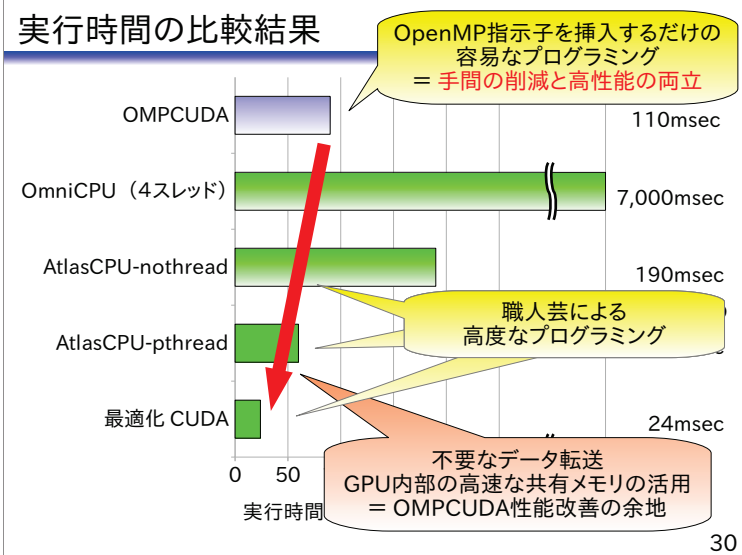


## 実行時間の比較結果






## 実行時間の比較結果



## GPGPUプログラミング環境に関する研究:まとめ

- 提案:既存のCPU向け並列化プログラミング環境を用いたGPGPUプログラミング
  - GPUが並列処理による高い並列性能を持つこと,CPU向けの並列化プログラミング環境の研究が進んでいることに着目
  - 既存のCPU向け並列化プログラミング環境とGPGPU・CUDAとの対応付けを検討
- 実装:OMPCUDA(OpenMP for CUDA)
  - OpenMPプログラムの並列実行部をGPU上で高速実行
- 今後の課題
  - 性能向上:特に高速共有メモリ(SharedMemory)の活用
  - CPUとGPUによる問題分割・並列実行



## CPUとGPUによる問題分割・並列実行

- CPUとGPUは同時に演算を行うことが可能
  - 問題分割・並列実行 → 「CPUのみ」や「GPUのみ」より高性能
  - データ規模が小さい場合やCPUとGPUのバランスによって実行対象プロセッサの選択が必要 → 利用の手間
- 提案:CPU単体で実行されるライブラリと同様のAPIを持ち,CPUとGPUを用いて問題分割・並列実行を行うライブラリ
  - 行列積和演算(SGEMM)を行うライブラリを実装
  - 問題設定やハードウェアの性能バランスを気にすることなく容易に高性能を得られることを確認
- OMPCUDAへの適用
  - OpenMP(のループ並列化)は制御の並列性を指示,共有メモリが前提

    → CPUとGPUにメモリが分散,対応付けが容易でない,対応を検討中



## アウトライン

1. GPGPUの概要と研究動向
   - GPUとGPGPUの概要
   - GPGPUの研究事例やプログラミング方法
2. GPGPUプログラミング環境に関する研究
   - 我々が行っている研究の紹介
3. GPU(GPGPU)とスーパーコンピュータ
   - 現在および近い将来の課題や展望



## GPU・GPGPUの現状と課題,今後の展望

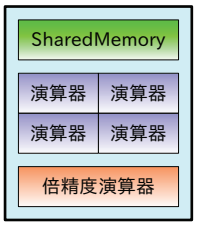
- 現在および近い将来の動向の確認
- 今後のGPGPU研究の方向性や課題の見積もり
- 解決・改善方法の検討

1. GPUの性能やハードウェアアーキテクチャ
2. GPUの普及
3. GPGPUの対象アプリケーション
4. スーパーコンピュータ(ベクトル計算機,計算機センタ)とGPGPU
5. GPGPUのプログラム開発環境



## 1. GPUの性能やハードウェアアーキテクチャ 1/4

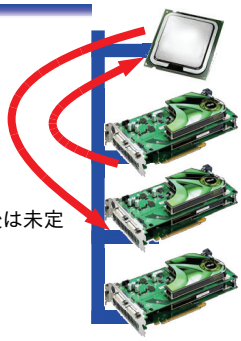
- 演算精度(倍精度への対応,CPUとの演算精度差)
  - (特に数値計算において)GPGPUにおける最大の問題点の1つ
  - 倍精度への対応が行われた(2007年末以降)
    - 廉価なGPUでは非対応
    - 低性能(単精度の5〜10分の1)
    - 疑問:本当にCPUと同じ計算結果が得られるか
  - 展望
    - (HPC向けGPU製品の)倍精度演算の性能向上を期待する
    - 混合精度計算
      - 混合精度計算の技術が注目される可能性
      - NVIDIA GPU : 単精度計算と倍精度計算の同時実行による性能向上






## 1. GPUの性能やハードウェアアーキテクチャ 2/4

- GPUに搭載されているメモリの容量
  - 4GB/1Card搭載モデルが登場(昨年)
    - 大規模な計算が可能に
  - 複数GPUによる大規模計算が容易に
    - GPU間の転送がオーバヘッド
      - 現時点ではGPU同士で直接転送が不可能,今後は未定
      - アプリケーション個々のチューニングが重要
- GPUの(理論最大)演算性能
  - 多数の単純なコアによる超並列計算
    - 超並列・微細化・クロック周波数向上により1TFLOPSを達成したが一→(1GPUあたりの演算性能は,CPUと同様に)限界となる可能性
    - CPUはマルチコアに移行したが,GPUはそもそもマルチコア
    - 複数GPUによる更なる性能向上への注目(→GPU間転送の問題)



## 1. GPUの性能やハードウェアアーキテクチャ 3/4

- 現在のGPUアーキテクチャ
  - （事実上）NVIDIA vs AMD
- GPUのアーキテクチャ研究
  - （GPUベンダ以外では）あまり行われていない
    - （グラフィックスの時代は）内部アーキテクチャまで注目されなかった？
  - 展望
    - GPUの内部構造が（ある程度）明らかになったため，今後発展する可能性がある
      - スケジューラ，メモリの構成，シミュレータ†，etc.

† Sylvain Collange, David Defour, David Parello. Barra, a Modular Functional GPU Simulator for GPGPU. Technical Report hal-00359342, Université de Perpignan, 2009.



## 1. GPUの性能とハードウェアアーキテクチャ 4/4

- CPUとGPUの統合
  - CPUとGPUが1チップに統合されようとしている（早ければ今年）
  - GPUがさらに普及，CPUとGPUの併用など有効活用が重要
    → 2. GPUの普及，5. GPGPUのプログラム開発環境
- GPUに近いハードウェア（SIMD型アクセラレータ）の動向
  - Cell/B.E., PowerXCell　→ 4. スーパーコンピュータとGPGPU
    - RoadRunner（ロスアラモス研究所）
    - GPUと比較して：倍精度演算が高性能，プログラミングの制限が緩い
  - ClearSpeed Advanced Accelerator(CSX)
    - TSUBAME（東工大）
  - Larrabee
    - x86ベースの並列プロセッサ，今年～来年ごろに登場予定
  - GRAPE, etc.



## 2. GPUの普及

- モバイルからスーパーコンピュータまで広がるGPU
  - CPUと同様にどこにでもあるプロセッサへ
  - 積極的な利用が可能になっている・求められている
- モバイルとGPU
  - Tegra(NVIDIA)：プログラマブルシェーダ対応の携帯端末用GPU
    - （省電力と動画再生支援が注目されているが）
      携帯端末の演算性能底上げや可視化処理の性能向上に期待
    - 展望：マルチメディア処理への活用（動画像の解析・変換），超低消費電力HPC（搭載メモリ容量の小ささが問題）
- スーパーコンピュータとGPU
  - アクセラレータによるヘテロジニアス計算環境
    - RoadRunner：Opteron + PowerXCell
    - TSUBAME：Opteron + Xeon + CSX + Tesla
      → 4. スーパーコンピュータとGPGPU



## 3. GPGPUの対象アプリケーション

- GPGPU黎明期 ： 基本的なアルゴリズムの試験実装
  - 線形計算，ソーティング，物理シミュレーションなど
- 現在 ： 様々なアプリケーションへの応用が始まっている
  - 流体計算（東工大青木），FFT（東工大額田），N体問題（長崎大濱田）
  - コア部分をGPUへ移植 → より広い部分をGPUへ，複数GPU
- 課題と展望
  - 高速化知識・技術の集積
    - さらなるアプリケーションの高速化やミドルウェア実装へ活用する段階
  - 新たなアプリケーションの発掘
    - 気づかせる努力＋積極的に広める
      - GPGPU技術者とアプリケーションエンジニアの交流が必要
      - 並列計算による高速化に興味があるユーザの取り込み
    - （WindowsVista/7, Snow Leopard ： OSがキラーソフト?）



## 4. スーパーコンピュータ（ベクトル計算機，計算機センタ）とGPGPU 1/5

- GPU＝ベクトル計算機？
  - GPUはベクトル計算機を置き換えるか（復権させるか）
- 計算機センタへのGPU導入はどうか
  - GPUの特徴：高い並列度，高いピーク性能，大きなメモリ帯域，分岐に弱い
    - ベクトル計算機に近い，アプリケーションによっては高い性能
  - 性能に期待が持てる一方で制限や不安も

  →利点・課題を検討



## 4. スーパーコンピュータとGPGPU 2/5

- GPUの利点1（運用者）
  - 消費電力あたり・占有空間あたりの演算性能が高い
    - TSUBAMEでは筐体の隙間にGPU(Tesla)を増設し性能向上
    - 計算機センタの課題の1つである空間の制約に有利
  - 導入が容易
    - ハードウェア：PCI-Expressバスと電源の余裕があれば良い
    - ソフトウェア：LinuxやWindowsなどに容易にインストール可能
- GPUの利点2（利用者）
  - 手元のPCと計算機センタで同じ（近い）アーキテクチャ
    - GPUは汎用PCに普及済，開発環境も揃えやすいため，デバッグやチューニングが行いやすい
      - GPU向けのデバッグ技術やデバッガの普及は始まったばかり

## 4. スーパーコンピュータとGPGPU 3/5

- GPUの欠点（課題となりえる点）
  - 信頼性（大規模・長期での運用実績が不足）
  - 利用率（GPUは多くのユーザに活用されるのか）
    - 授業や特定アプリケーションでの利用は期待できる
      - ユーザの確保が見込めるなら導入の価値あり
      - 新たなユーザに有効活用してもらうためには課題多し
    - 計算機センタ利用者はGPUを利用するか/できるか/挑戦するか
    - 例えば計算機センタにGPUを導入したとして……
      - 「GPUとは何ですか?」
      - 「どう使うんですか?」
      - 「デバッグはどうやるんですか?」
      - 「移植は容易ですか?」
      - 「私のアプリケーションは速くなりますか?」
      - 「自動ベクトル化してくれますか?」

※同様の状況は GPU以外のアクセラレータでも（並列計算機でも）起こると考えられる



## 4. スーパーコンピュータとGPGPU 4/5

- 開発環境（特に初期実装の手間）の問題
  - ベクトル計算機の利用シナリオ

    Step1.汎用PC向けのプログラムが（ある程度）動く
    Step2.自動ベクトル化で性能向上
    Step3.チューニングして超高速化を目指す

  - GPUの利用シナリオ

    Step1.汎用PC向けのプログラムが（GPUでは）動かない
    Step2.努力して正しく並列実行可能にする
    Step3.チューニングして超高速化を目指す
    - 課題：初期実装の手間の削減，チューニング後の性能の見積もり



## 4. スーパーコンピュータとGPGPU 5/5

- 教育による解決
  - CUDAの授業・講習会
    - 一定の成果は期待できる
      - （GPU等のアクセラレータを導入するなら必須）
    - 問題：CUDAがいつまで主流か
      - OpenMPやMPI同様に長く利用されるのか，長く利用されて良いのか（→OpenCL）
- 展望：システムによる解決
  - 計算機システムによる解決

    → 5. GPGPUのプログラム開発環境
  - 利用形態による解決
    - 産学共同研究のプロジェクトなど，並列計算への関心が高い層への働きかけ



## 5. GPGPUのプログラム開発環境 1/2

- 現在：高級言語が整備され始めている
  - CUDA(NVIDIA), ATI Stream(AMD), ストリーミング言語, etc.
- 近い将来：共通化へ
  - 候補：OpenCL（Khronos, CUDAに近い言語仕様）
- 課題と展望
  - GPU（のアーキテクチャ）への依存度が高い
    - アーキテクチャ習得の手間と難しさが増加してしまう
  - CPUの並列度も向上，新たな並列化プログラミング環境の研究が始まっている
    - →互換性（FORTRANなど）
  - CPU, GPU, Cell/B.E., etc.で共通の開発環境が必要
    - （OMPCUDAを含めて）プログラミング環境に関する研究が重要
    - プログラミング言語を研究するグループとの連携



## 5. GPGPUのプログラム開発環境 2/2

- さらなる展望：CPUとGPUの活用を促進する仕組み（性能と利用率を上げる）
  - CPUとGPUによる負荷分散（演算を行うプロセッサの選択）
    - 現在：プログラマがCPUとGPUを明示的に選択
    - 将来：システムが選択を支援，システムが自動的に選択
  - 空いているプロセッサの活用（CPUとGPUの併用・有効活用）
    - CPU・GPUの負荷の偏りを是正（CPUの処理をGPUへ移動）
      - （GPUで動くプログラムは基本的にCPUでも動く）
      - 対象問題の特性（CPU向けかGPU向けか）や資源の利用・予約状況を元に振り分ける
      - 複数種類のCPU処理をまとめてGPU上で実行
  - 異なるプロセッサ向けの演算を共通記述する言語，振り分ける実行時システム，性能予測やチューニング（のためのモデル化），etc.　（→アーキテクチャへのフィードバック）



## 整理：課題と展望のまとめ

- 今後注目されるであろう研究
  - GPU向け混合精度演算技術
  - GPU向けアプリケーションの発掘
    - （計算機センタの）ユーザとの連携
    - 高速化知識・技術の蓄積とミドルウェアへの活用
  - GPUアーキテクチャとプログラム開発環境・開発支援
    - 蓄積された高速化知識・技術を取り込み，使いやすさと性能を備えた開発環境を提供（OMPCUDAの改良・拡張など）
    - （GPUにこだわらず）計算機・プロセッサをさらに活用する仕組み

## おわりに（全体のまとめ）

1.GPGPUの概要と研究動向

2.GPGPUプログラミング環境に関する研究

- 我々が進めている「汎用プログラミング環境を用いたGPGPUに関する研究」を紹介した
  - 実用的なツールを目指す
    ＋ 問題点の指摘, 新しいプログラミング環境作りのベース

3.GPU(GPGPU)とスーパーコンピュータ

- GPGPU関連で近年注目されている話題などを紹介した
  - アプリケーションやユーザの発掘がまだまだ必要
  - 開発支援が重要



## ご清聴ありがとうございました