TSUBAME2.0におけるGPUの活用方法

東京工業大学学術国際情報センター
丸山直也

第9回GPUコンピューティング講習会
2011年8月3日

目次

1. TSUBAMEのGPU環境
2. プログラム作成
3. プログラム実行
4. 性能解析、デバッグ

サンプルコードは
/work0/GSIC/seminars/gpu-2011-08-03
からコピー可能です

1. TSUBAMEのGPU環境

計算ノード

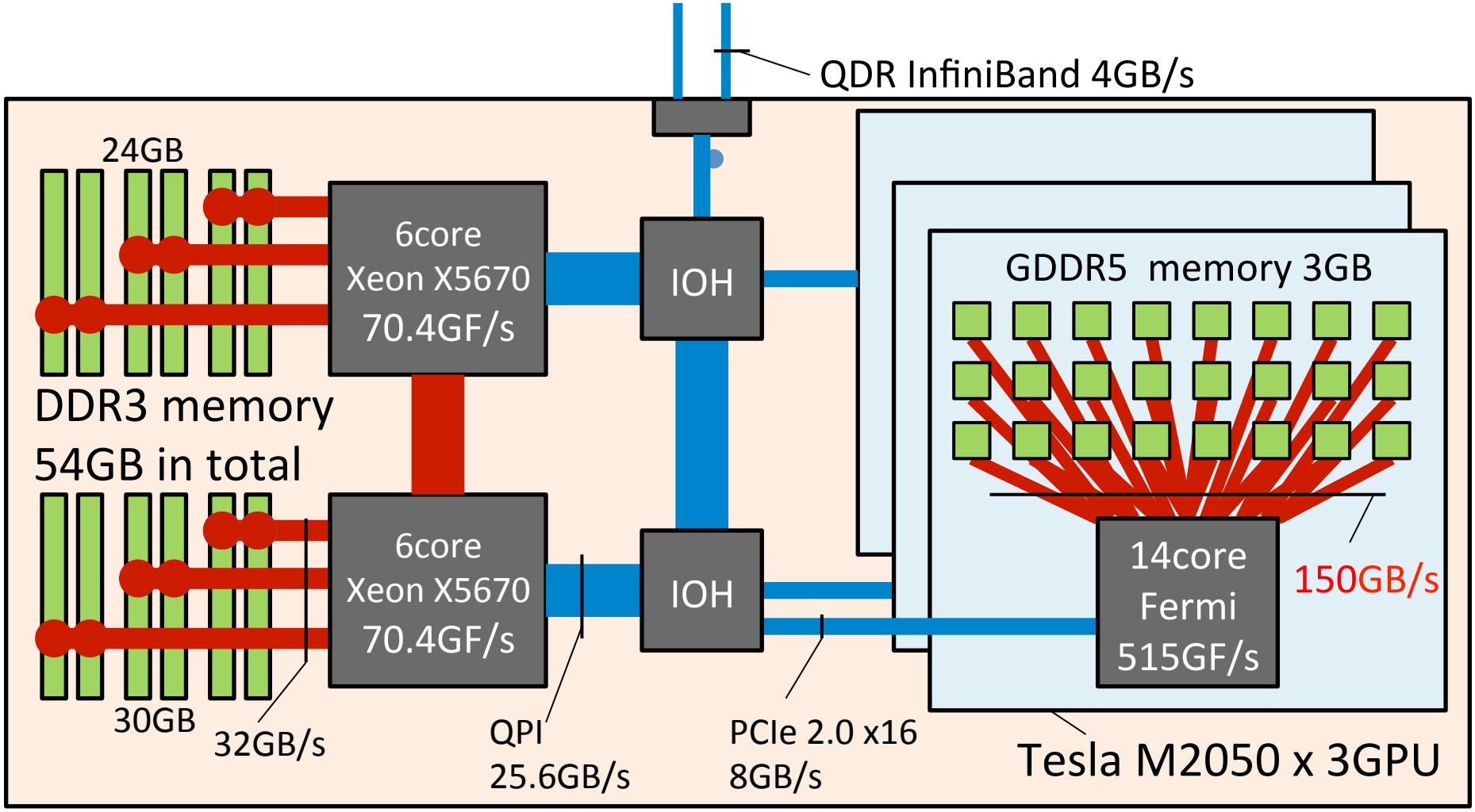
- 1408 Thin nodes + 24 Medium nodes + 10 Fat nodes

- Thin node: HP Proliant SL390s G7
  - CPU: Intel Xeon 2.93GHz 6core x 2CPU=12 cores
  - GPU: NVIDIA Tesla M2050 3GPU
    CPU 140GF + GPU 1545GF = 1685GF
  - Memory: 54GB
  - SSD: 120GB
  - Network: QDR InfiniBand x 2 = 80Gbps

NVIDIA Tesla M2050

- 448コア、3GBメモリ
- 1030 GFLOPS (SP), 515 GFLOPS (DP)
- メモリバンド幅 148 GB/s
- Fermi（フェルミ）アーキテクチャ
  - ハードウェアキャッシュ
  - C++サポート
  - ECC
  - その他のFermi GPU
    - Tesla 2070/2090 シリーズ
    - GeForce GTX 480/580 GTX

計算ノード構成（Thin node）



TSUBAME 2.0 全体概要

TSUBAME2.0: A GPU-centric Green 2.4 Petaflops Supercomputer

Tsubame 2.0: "Tiny" footprint, very power efficient
- Floorspace less than 200m² (2,100 ft²)
- Top-class power efficient machine on the Green 500

System
(42 Racks)
1408 GPU Compute Nodes,
34 Nehalem "Fat Memory" Nodes

Rack
(8 Node Chassis)

Node Chassis
(4 Compute Nodes)

Compute Node
(2 CPUs,3 GPUs)

Chip
(CPU ,GPU)

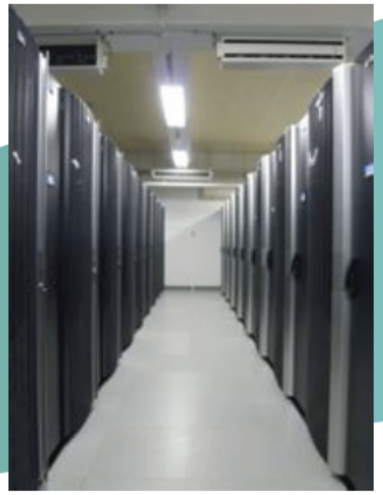
CPU(Westmere EP)
76.8 GFLOPS

GPUs(Tesla M2050)
515 GFLOPS
3 GB

1.6 TFLOPS
55 GB/103 GB

6.7 TFLOPS
220 GB/412 GB

53.6 TFLOPS
1.7 TB/3.2 TB



2.4 PFLOPS
80 TB

Integrated by NEC Corporation

ソフトウェア環境

	TSUBAME 2.0
Linux OS	SUSE Linux Enterprise Server 11 SP1
Windows OS	Windows HPC Server 2008 R2
Job Scheduler for Linux	PBS Professional
Job Scheduler for Windows	Windows HPC Server

- Windows OSを新規にサポート
- ジョブスケジューラが変更されたため、バッチジョブ投入オプションがTsubame1と大きく変わります

コンパイラ・ライブラリなど

	TSUBAME 2.0
Compiler	Intel Compiler 11.1.072 (標準) PGI CDK 10.6 gcc 4.3.4
MPI	OpenMPI 1.4.2 (標準) MVAPICH2 1.5.1
CUDA	3.2 (夏季メンテンナンス以降4.0に)
CPU用BLAS/ LAPACK/FFT	MKL (http://tsubame.gsic.titech.ac.jp/docs/guides/ tsubame2/html/programming.html#id4)
GPU用BLAS	CUBLAS 3.2 (CUDA Toolkit 付属)
GPU用LAPACK	CULA (http://tsubame.gsic.titech.ac.jp/docs/guides/ tsubame2/html/programming.html#cula)

2. GPUプログラム作成

GPUプログラミング

- CUDA C/Fortranを利用
- OpenCLを利用
- PGIアクセラレータコンパイラを利用

CUDA Cプログラム開発

- コンパイラ
  - nvcc
  - /opt/cuda ディレクトリ以下にバージョンごとにインストールされています
  - 現在のデフォルトバージョンは 3.2 です（2011年8月3日現在）
  - 2011年8月15日よりデフォルトバージョンが4.0にアップデートされます
- デバッガ
  - CUDA標準の cuda-gdb が利用可能です
  - cuda-memcheck: メモリエラーチェック

CUDA Cプログラム開発実習

- 以下のコマンドをターミナルから入力し、CUDAプログラムのコンパイル、実行を確認してください
  - “$” はコマンドプロンプトです

```
$ cd
$ cp /work0/GSIC/seminars/
gpu-2011-08-03/test.cu .
$ nvcc test.cu –o test
$ ./test
```

CUDA Fortranプログラム開発

- コンパイラ
  - CUDA Fortranコンパイラが利用可能
    - PGIコンパイラがサポート
    - 通常のPGI Fortranコンパイラによりコンパイル可能

```
$ cd
$ cp /work0/GSIC/seminars/
gpu-2011-08-03/fortran/matmul.CUF .
$ pgfortran matmul.CUF –o matmul
$ ./matmul
```

OpenCLプログラム開発

- NVIDIA GPU用OpenCL開発ツールキットは CUDAツールキットおよびGPUドライバに付属
- OpenCLヘッダーファイル、ライブラリ
  - /opt/cuda/3.2/include/CL 以下
  - /usr/lib64/libOpenCL.so
- コンパイル方法
  - “-I/opt/cuda/3.2/include”
- リンク方法
  - “-lOpenCL”

PGIアクセラレータプログラム開発

- PGIアクセラレータ拡張
  - OpenMPのような指示文により一部をGPU実行
    - OpenMPでは指示文によりループを並列実行
    - PGIアクセラレータ拡張ではループをGPUにより並列実行
- PGIコンパイラによりコンパイル
  - コンパイルオプションに “-ta=nvidia” を追加

PGI指示文サンプルコード

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
int main( int argc, char* argv[] ) {
   int n = 10000;      /* size of the vector */
   float *restrict a;  /* the vector */
   float *restrict r;  /* the results */
   float *restrict e;  /* expected results */
   int I;
   a = (float*)malloc(n*sizeof(float));
   r = (float*)malloc(n*sizeof(float));
   e = (float*)malloc(n*sizeof(float));
   for( i = 0; i < n; ++i ) a[i] = (float)(i+1);
```

続く

PGI指示文サンプルコード

```
  #pragma acc region
  {
    for( i = 0; i < n; ++i ) r[i] = a[i]*2.0f;
  }
  /* compute on the host to compare */
    for( i = 0; i < n; ++i ) e[i] = a[i]*2.0f;
  /* check the results */
  for( i = 0; i < n; ++i )
    assert( r[i] == e[i] );
  printf( "%d iterations completed\n", n );
  return 0;
}
```

PGIアクセラレータコンパイラ実習

- 必須→PGIコンパイラに –ta=nvidia オプションを追加
- 推奨→ -Minfo オプションによりコンパイラによるGPUコード生成の情報を表示

```
$ cd
$ cp /work0/GSIC/seminars/
gpu-2011-08-03/pgi_acc/c1.c .
$ pgcc c1.c –ta=nvidua -Minfo –o c1
$ ./c1
```

PGIアクセラレータコンパイラ実習

- コンパイル時のメッセージ

```
t2a006173:tmp$ pgcc c1.c -ta=nvidia -Minfo -o ci1
main:
    23, Generating copyin(a[0:n-1])
        Generating copyout(r[0:n-1])
        Generating compute capability 1.0 binary
        Generating compute capability 1.3 binary
    25, Loop is parallelizable
        Accelerator kernel generated
        25, #pragma acc for parallel, vector(256)
            CC 1.0 : 3 registers; 20 shared, 36 constant, 0 local memory bytes; 100 occupancy
            CC 1.3 : 3 registers; 20 shared, 36 constant, 0 local memory bytes; 100 occupancy
```

3. GPUプログラム実行

テスト実行（無料）

- インタラクティブノード上で実行
  - 制限： 実行時間30分まで、並列度4プロセス、メモリ6GB
  - GPUの利用に関しては時間以外に制限なし
  - コマンドラインで直接プログラムを実行可能

- 無料キューで実行
  - 制限： 2ノード、10分まで
  - ノード内プロセス数・メモリ利用量に制限なし
  - GPU利用に関しても制限なし
  - バッチキューにジョブを投入して実行
    - キュー： S、グループ： 無指定
    - 例： `t2sub -q S -l` 他のオプション ジョブスクリプト

制限を超えた利用は他の利用者の迷惑になるため注意

バッチキューの使い方
t2subコマンドの基本

- ~/testにあるmyprogというプログラムを、Sキューで実行する場合

(1) スクリプトファイルを作っておく (たとえばjob.shというファイル)

```
#!/bin/sh
cd $HOME/test
./myprog
```

job.shファイル

(2) t2subコマンドで投入

```
t2sub –W group_list=xxx –q S ./job.sh
```

-q xxx: キュー名を指定

-W group_list=xxx: TSUBAMEグループ番号を指定

本実行用キュー（有料）

- Sキュー
  - 指定した台数のノードを専有して利用
  - システムが順番にリクエストされたジョブを処理
- Hキュー
  - Sキューと同様に指定した台数のノードを専有して利用
  - ただし、バッチキュー形式ではなくTSUBAMEポータルより利用したい日付・台数を予約して利用（カレンダー予約）
    - http://portal.g.gsic.titech.ac.jp/ →「ノード予約」
  - 予約が入れば指定した日に確実に利用可能
  - 多数ノードを利用する場合に最適
  - 利用料Sキューの1.25倍
- Gキュー
  - 各ノードの3台のGPUおよびCPU4コア（ハードウェアスレッド数８）のみ利用
    - 残り8コアは仮想マシンにて利用（Vキュー）し、CPUジョブとGPUジョブを共存
  - Sキューの半額

節電・ピークシフト運用

- 節電のため一部変則的な運用
- Yキュー
  - 200ノード
  - Sキューと同様の構成（GPU利用可）
  - ただし、実行時間1時間まで
  - さらに、電力使用量を緊急に削減する必要がある場合はジョブを強制終了しノードをシャットダウン
  - 利用料はSの0.8倍

キュー稼働状況 → http://mon.g.gsic.titech.ac.jp/summary/

有料キュー利用シナリオ

- GPUのみを用いる場合
  - Gキューがおすすめ
  - 利用料金Sの半額
  - ただし、CPUコアは4コアのみ
- CPUもそれなりに用いる場合
  - SまたはYがおすすめ
  - 短い実行時間のジョブならばYの方がお得
- 大規模実行（数百ノード）の場合
  - Hキューで予約する方法が確実
  - ただし、最低利用時間が1日のため短時間利用の場合には利用料金的に非効率
  - Sキューが混んでいる場合、急ぎで確実に実行したい場合も有効

4. 性能解析、デバッグ

プロファイラ

- “Compute Visual Profiler”
  - NVIDIA CUDAツールキット付属
  - /opt/cuda/3.2/computeprof/bin/computeprof
- CUDAおよびOpenCLプログラムの性能解析をサポート
  - 実行時間
  - PCIデータ転送サイズ
  - メモリアクセス回数
  - 分岐回数
  - 実行命令数
  - キャッシュミス回数
  - など

CUDA用デバッガ

- cuda-gdb
  - NVIDIAによるGDBの拡張
  - Linux専用（CUDA 4.0よりOS Xもサポート）
  - CUDA toolkit付属
  - TSUBAMEで利用可能
- Parallel Nsight
  - NVIDIAによるVisual Studio用プラグイン
  - 無料
  - 性能解析等を含む非常に豊富な機能を搭載
- TotalView
  - 商用
  - TSUBAMEで利用可能
- DDT
  - 商用

CUDA-GDB

- GDB
  - Linux標準のデバッガ
  - 標準的なデバッガの機能を搭載
    - シングルステップ実行、ブレイクポイント、など
- CUDA-GDB
  - GDBをベースにGPU上のプログラム実行のデバッグをサポート
  - ホストコードは通常のGDBと同様にデバッグ可能
  - カーネル関数のシングルステップ実行やブレイクポイントの設定が可能
- 両者ともコマンドラインインターフェイスのみ
  - TotalViewなどはより使いやすいGUIを提供