Computing Platforms
– GPU Computing Platforms

Introducing collaboration members – Korea University (KU)
ALICE TPC online tracking algorithm on a GPU

Joohyung Sun
Prof. Hyeonjoong Cho

ALICE Collaboration
Korea University, Sejong

4th ALICE ITS upgrade, MFT and O2 Asian Workshop 2014 @ Pusan

Introduction

- Collaboration Institute, Korea University
- Research goal
- ALICE TPC online tracking algorithm on a GPU
- Specification of benchmark platform

Introducing Korea University

Prof. Hyeonjoong Cho, Embedded Systems and Real-time Computing Laboratory

Meeting of June 19th 2014 in KISTI

- Proposal of contribution of KISTI and the Korea University to the ALICE O2
- Participants from KISTI, Korea University, and CERN
- One of the suggested possible collaborations
  - ✓ Benchmarking of detector-specific algorithms on some agreed hardware platforms
  - ✓ Multi-cores CPU, many-cores CPU, ***GPGPU***, etc.

Our Research Goal

Prof. H. Cho and J. Sun, Korea University, Republic of Korea

- **Collaboration institute**
  - Prof. H. Cho, Institute Team Leader, Korea University, Sejong, Republic of Korea
  - J. Sun, Deputy, Korea University, Sejong, Republic of Korea

- **Application benchmark on a modern GPU**
  - Benchmarking different types of processors
    - ✓ Kepler- and Maxwell-based architecture GPU
    - ✓ Maxwell GPU is the successor to the Kepler and is the latest GPU in this year
  - Reengineering detector data processing algorithms (GPU tracker)
    - ✓ Apply NVIDIA Kepler's technologies

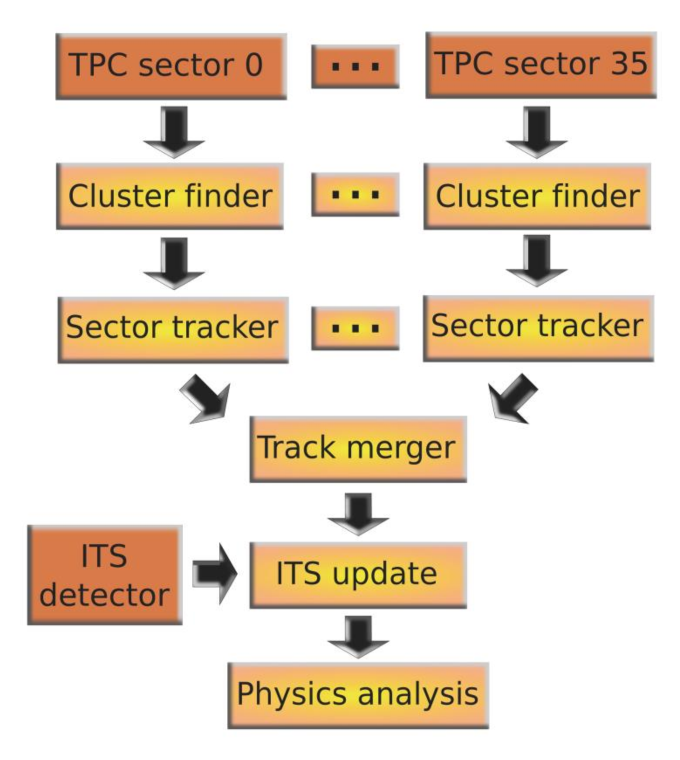
      Hyper-Q and Dynamic parallelism

ALICE TPC Online Tracking Algorithm on a GPU

Detector-specific algorithms with parallel frameworks

The online event reconstruction

- Performs by the High-Level Trigger
- The most complicated algorithm
- Adapted to GPUs
  - ✓ GPU evolves into a general-purpose, massively parallel processor
  - ✓ NVIDIA Fermi, CUDA, and AMD OpenCL



HLT reconstruction scheme (Reference: David Rohr, CHEP2012)

Our Research Goal

Benchmarking platform

Specification of benchmark platform

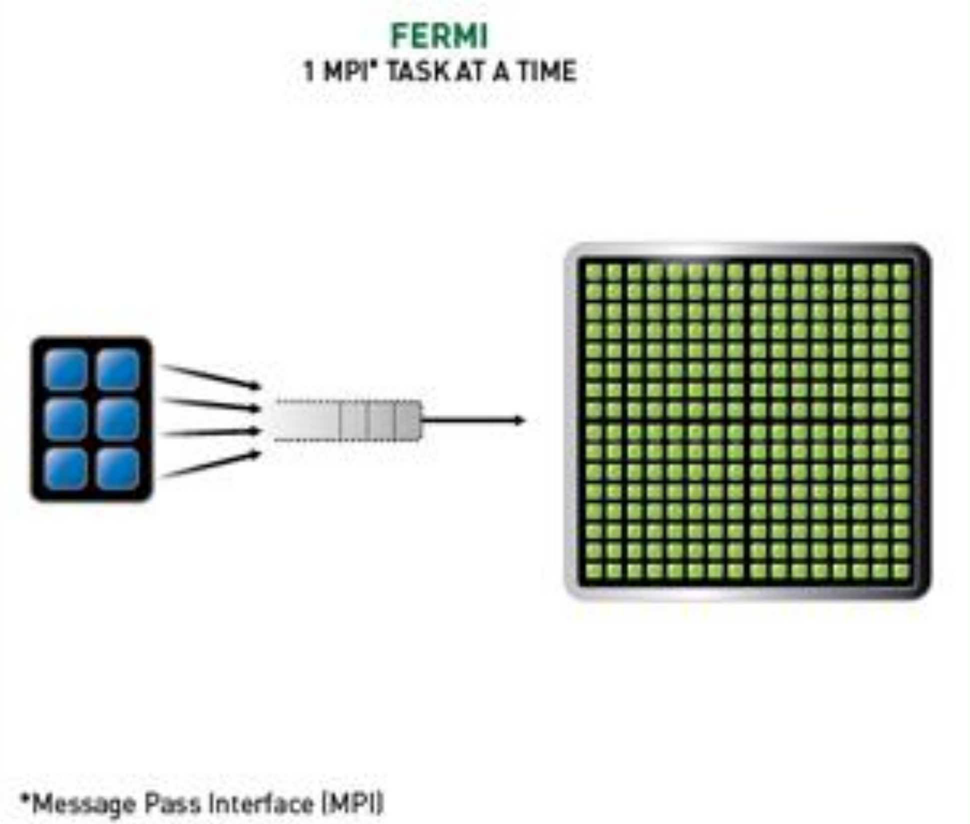
- CPU: Intel i7-4770 CPUs @ 3.4 GHz, 4-cores (HT, 8-cores)

- GPU: NVIDIA Tesla K20c GPU
  - ✓ Kepler-based architecture
  - ✓ 13 Multiprocessors
  - ✓ 192 CUDA cores per multiprocessor
  - ✓ 706 MHz (0.71 GHz) GPU Clock rate
  - ✓ 2600 MHz Memory Clock rate
  - ✓ 320-bit Memory Bus Width
  - ✓ Maximum number of threads per multiprocessor: 2048
  - ✓ Maximum number of threads per block: 1024
  - ✓ Concurrent copy and kernel execution: Yes with 2 copy engines

Fermi and Previous Generation GPUs

Low the usage of GPU resources

- **Only one work queue**
  - It can execute a work at a time
  - CPUs are not able to fully utilize GPU resources

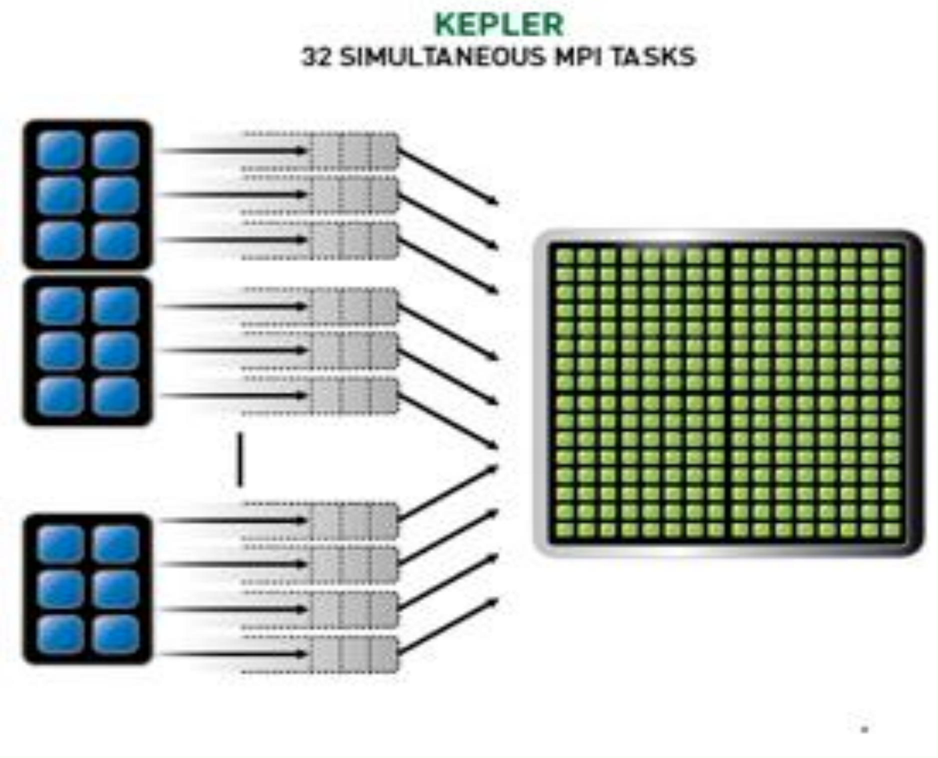


- **Low usage of GPU resources**
  - Even though the GPU has plenty of computational resources

Hyper-Q

Maximizing the usage of GPU resources

- **Enabling multiple CPU cores to launch work on a single GPU simultaneously**
  - Increasing GPU utilization
  - Slashing CPU idle times

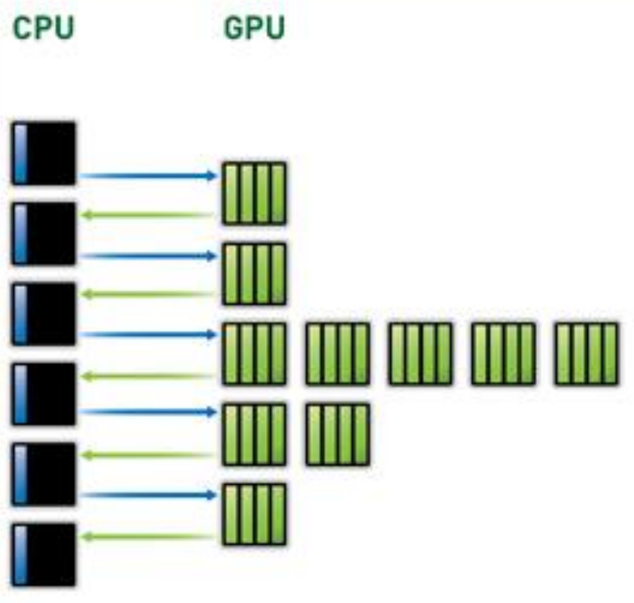


- **32 work queues**
  - ✓ Fully scheduled, synchronized, and managed all by itself
- **GPUs receive works from queues at the same time**
  - ✓ All of the works is being done concurrently

Previous CUDA programming model

The communication between host and device

- **Previous CUDA programming model**

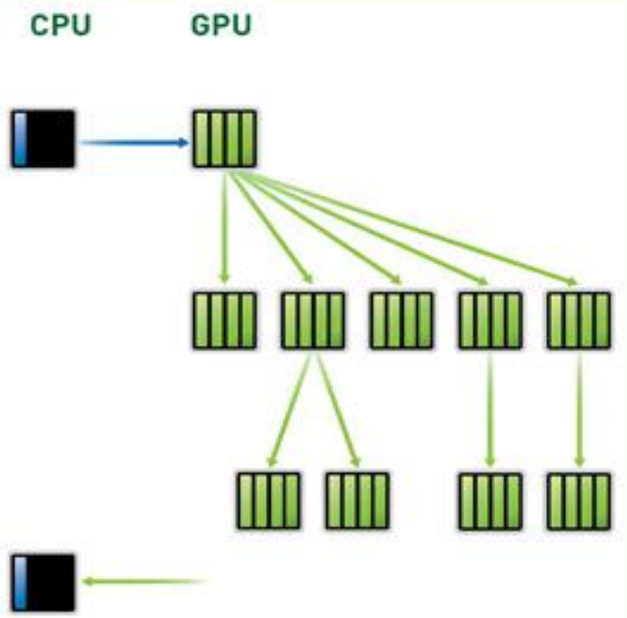


- **The communications between CPU and GPU**
  - Can affect the application's performance
  - Each cost as a time is not negligible

Dynamic Parallelism

Creating work on-the-fly

- **CUDA programming model in Kepler**



- **Effectively allows to be run directly on GPU**
  - Saving the time for communications

- **Enabling GPU to dynamically spawn new threads**
  - By adapting to the data
  - Without going back to the host CPU

Progress

- Previous works
- Current progress
- Optimization with NVIDIA Visual Profiler

Previous Works

Benchmarking HLT tracker

- **Some results of benchmarking HLT tracker on each GPU**
  - NVIDIA Fermi (current version) 174 ms
  - NVIDIA GTX780 (Kepler) 155 ms
  - NVIDIA Titan (Kepler) 146 ms
  - AMD GCN 160 ms

Reference: P. Buncic and et al., "O2 Project", ALICE LHCC Referee Meeting

Our Current Progress

ALICE TPC online tracking algorithm on a GPU

❖ Application benchmark

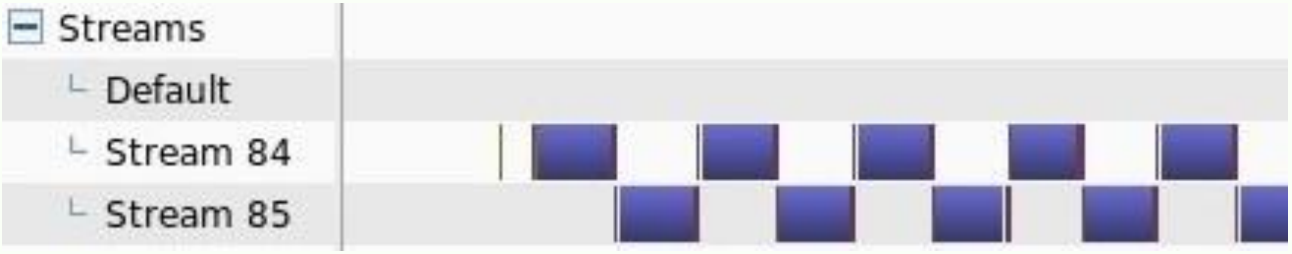
- Tested on Kepler-based architecture GPU
  - ✓ Maxwell-based architecture GPU will be benchmarked
- To fully utilize the compute and data movement capabilities of the GPU
  - ✓ Optimization
  - ✓ Hyper-Q is applied for enabling concurrent copy and ker execution
  - ✓ Dynamic parallelism will be applied for reducing the number of communications between CPU and GPU

Comparison of Hyper-Q between GTX650 and K20c

NVIDIA Visual Profiler (nvvp)

Profiling GeForce GTX650 with 2 streams

- Works are managed by one work queue



Kernel execution of algorithm

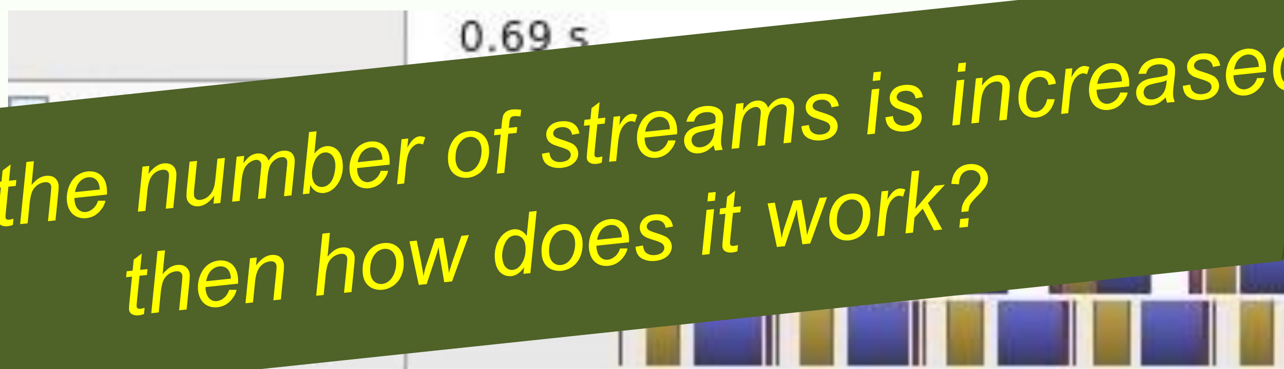
Memory copy execution from CPU to GPU

- All other copy and kernel executions wait for previous executions

Comparison of Hyper-Q between GTX650 and K20c

NVIDIA Visual Profiler (nvvp)

- **Profiling Tesla K20c with 2 streams**
  - 32 work queues for concurrent executions



If the number of streams is increased, then how does it work?

Kernel execution of algorithm

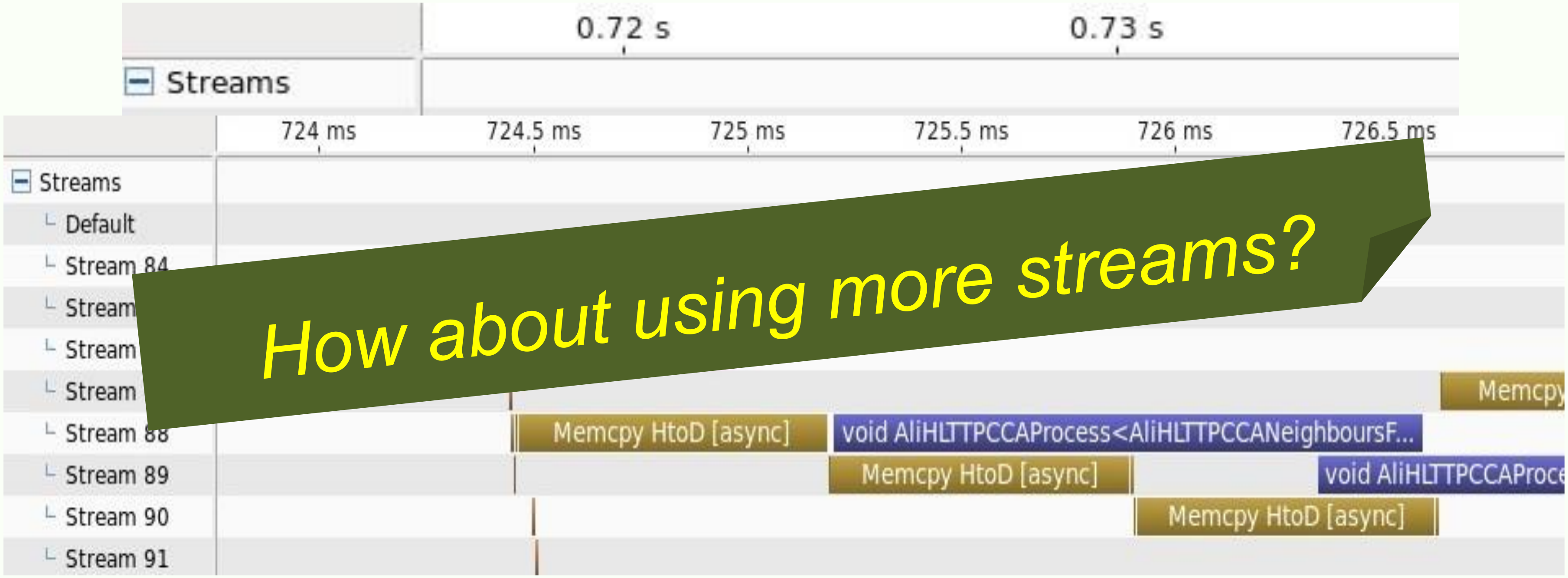
Memory copy execution from CPU to GPU

  - Copy and kernels are executed concurrently

More Concurrent Executions

Tesla K20c, 8 streams

Tesla K20c with 8 streams

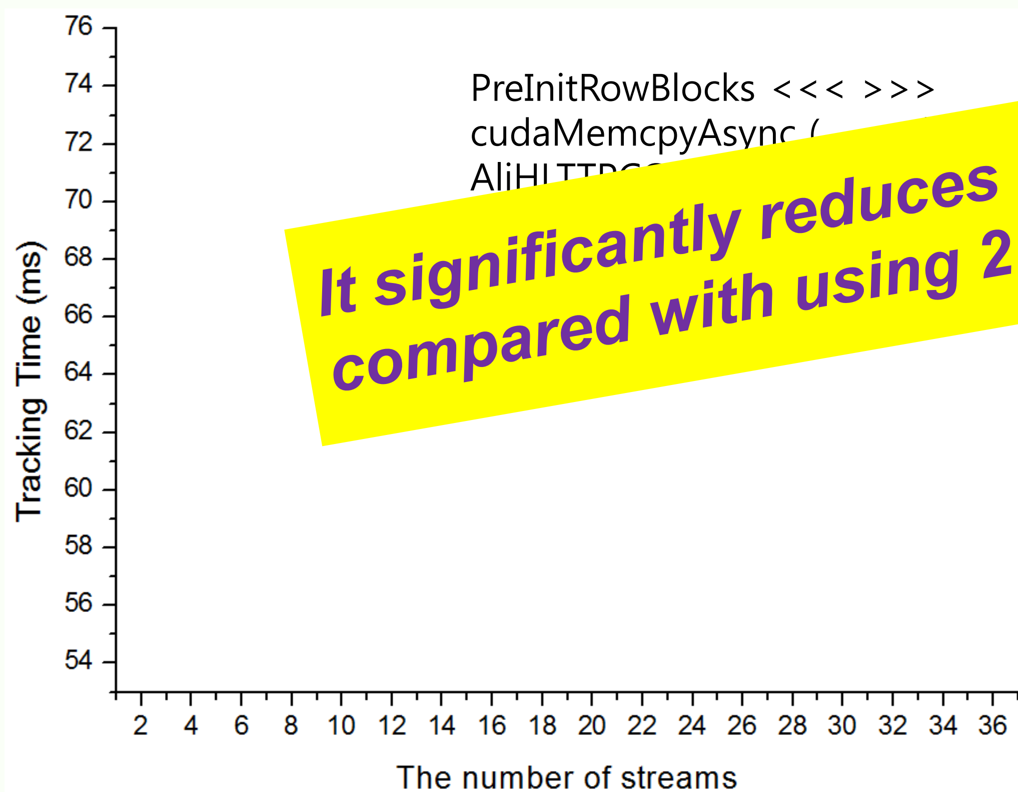


- Copy and kernels in some of the streams more than two are executed concurrently

Observation from Multiple Streams

The number of streams

- **Measuring specific compute kernels' time per the number of streams**
  - The number of streams: 2~36



It significantly reduces the tracking time compared with using 2 streams

Possible Reasons for Observation

The key for optimization

- **The number of copy engines in GPU**
  - E.g. Tesla K20c has only 2 copy engines
  - Limit as the number of works can be executed concurrently
- **Too short kernel execution time**
  - It could be finished before another kernel execution is arrived
  - The longest kernel execution time
    - ✓ Only about 2 ms during this test
- ***This observation will be a key***
  - For optimizing Hyper-Q

Summary

Next research plans

- **Korea University**
  - Prof. Hyeonjoong Cho, Institute Team Leader, Korea University, Sejong, Republic of Korea
  - Joohyung Sun, Deputy, Korea University, Sejong, Republic of Korea

- **Next research plans**
  - Benchmarking Maxwell-based architecture GPU
    - ✓ GeForce GTX 980, about $ 549
  - Efficiently applying GPU's technologies
    - ✓ Hyper-Q with scheduling of streams
    - ✓ Dynamic parallelism with device memory management

Appendix. Actual Code for Dynamic Parallelism

Creating work on-the-fly

LU decomposition (Fermi)

CPU code

```
dgetrf(N,N) {
    for j=1 to N
        for i=1 to 64
            idamx <<< >>>
            memcpy
            dswap <<< >>>
            memcpy
            dscal <<< >>>
            dger <<< >>>
        next i

        memcpy
        dlaswap <<< >>>
        dtrsm <<< >>>
        dgemm <<< >>>
    next j
}
```

GPU code

```
idamx ();

dswap ();

dscal ();
dger ();


dlaswap ();
dtrsm ();
dgemm ();
```

Appendix. Actual Code for Dynamic Parallelism

Creating work on-the-fly

- **LU decomposition (Kepler)**

```
dgetrf(N,N) {
    dgetrf <<< >>>

    synchronize();
}
```

CPU is Free

CPU code

```
dgetrf(N,N) {
    for j=1 to N
        for i=1 to 64
            idamx <<< >>>
            dswap <<< >>>
            dscal <<< >>>
            dger <<< >>>
        next i
        dlaswap <<< >>>
        dtrsm <<< >>>
        dgemm <<< >>>
    next j
}
```

GPU code

Appendix. Example of LU Decomposition

Profiling LU Decomposition using NVIDIA Visual Profiler (nvvp)

- **Tesla K20c, Context 1 (CUDA)**

Memcpy

cgetrf_cdpentry <<< >>>