グリッドコンピューティング

2011/10/31
小嶋秀徳 (11M37123 松岡研究室)

紹介論文

Dynamic Load Balancing on Single- and Multi-GPU Systems

IPDPS 2010
(International Parallel and Distributed Processing Symposium)

Long Chen†, Oreste Villa‡, Sriram Krishnamoorthy‡, Guang R. Gao†

† Department of Electrical & Computer Engineering University of Delaware
‡ High Performance Computing Pacific Northwest National Laboratory

Abstract

- The GPUs computational power for many applications.
- The current programming techniques are not sufficient for irregular, and unbalanced workload.
- The serious performance problem with concurrent multiple GPUs execution.
- Task-based dynamic load-balancing solution for single and multi-GPU systems.
- A Finer granularity than current GPU programming APIs
- Micro-benchmarks, Molecular dynamics application
- On single- GPU systems, the solution is more efficiently than the CUDA scheduler for unbalanced workload.
- On multi- GPU systems, the solution achieves near-linear speedup, load balance, and significant performance improvement over techniques based on standard CUDA APIs.

Agenda

1. Introduction
2. Related Work
3. CUDA Architecture
4. System Design
5. Implementation and Microbenchmarks
6. Case Study: Molecular Dynamics
7. Conclusion and Future Work
8. References

Agenda

1. Introduction
2. Related Work
3. CUDA Architecture
4. System Design
5. Implementation and Microbenchmarks
6. Case Study: Molecular Dynamics
7. Conclusion and Future Work
8. References

Introduction

Problem

- Load balancing and GPU resource utilization are not enough, with the current GPU programming paradigm.
- Conventional GPU programming does not provide sufficient mechanisms to exploit task parallelism in applications

Introduction

Solution

- Task-based fine-grained execution scheme
  - Dynamically balance workload on individual GPUs and among GPUs
  - Utilize the underlying hardware more efficiently
- Mechanisms to enable correct and efficient CPU-GPU interactions while the GPU is computing, based on the current CUDA technology.
- Task queue scheme enables dynamic load balancing at a finer granularity than what is supported in existing CUDA programming paradigm.
- Optimal memory sub-system locations for the queue data structures.
- Implementation of the queue scheme with CUDA.
  - Concurrent host enqueue and device dequeue
  - Wait-free dequeue operations on the device.

Introduction

Result

- Case study: molecular dynamics application
  - Single-GPU: effective utilization of the hardware than the CUDA scheduler, for unbalanced problems
  - Multi-GPU: nearly linear speedup, load balance, and significant performance improvement over alternative implementations based on the canonical CUDA paradigm

Agenda

1. Introduction
2. Related Work
3. CUDA Architecture
4. System Design
5. Implementation and Microbenchmarks
6. Case Study: Molecular Dynamics
7. Conclusion and Future Work
8. References

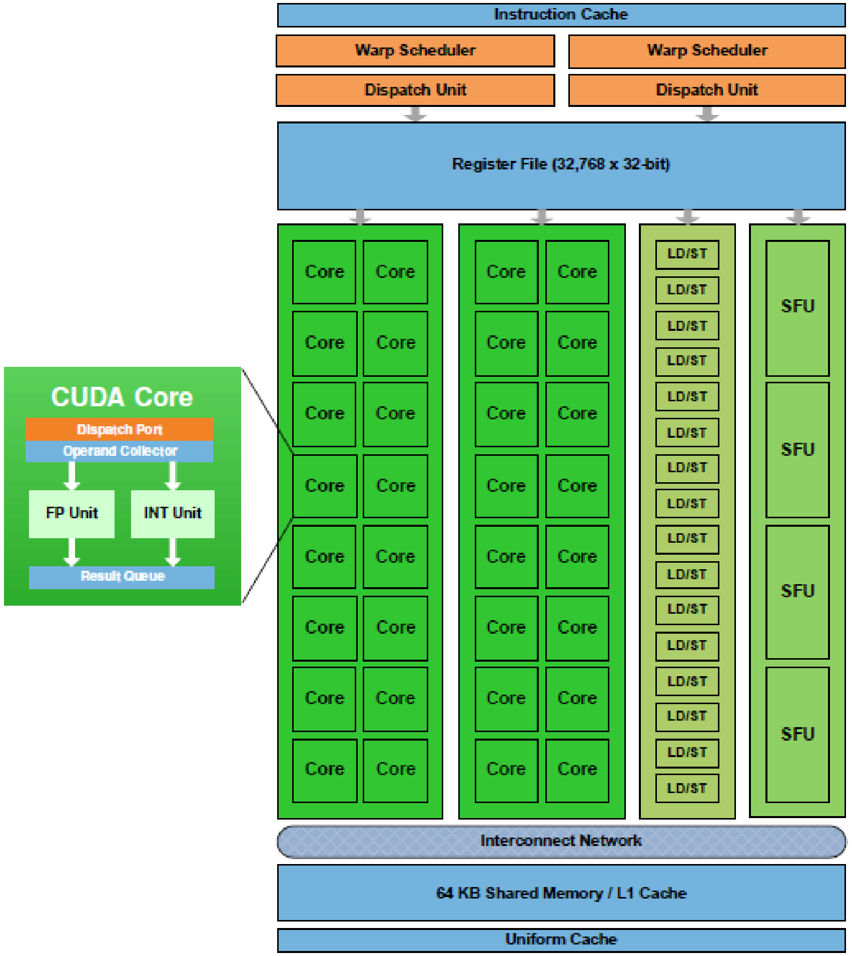
Related Work

- T. Foley and J. Sugerman. Kd-tree acceleration structures for a gpu raytracer. In HWWS’05, pages 15–22, New York, NY, USA, 2005.
- M. Mller, C.and Strengert and T. Ertl. Adaptive load balancing for raycasting of non-uniformly bricked volumes. Parallel Computing, 33(6):406 – 419, 2007. Parallel Graphics and Visualization.
- D. Cederman and P. T. On Dynamic Load Balancing on Graphics Processors. In GH 2008, pages 57–64, 2008.
- M. Guevara, C. Gregg, and S. K. Enabling task parallelism in the cuda scheduler. In PEMA 2009, 2009.
- M. D. Linderman, J. D. Collins, H. Wang, and T. H. M. Merge: a programming model for heterogeneous multi-core systems. SIGPLAN Not., 43(3):287–296, 2008.
- C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. In Euro-Par 2009, pages 863–874, Delft, Netherlands, 2009.

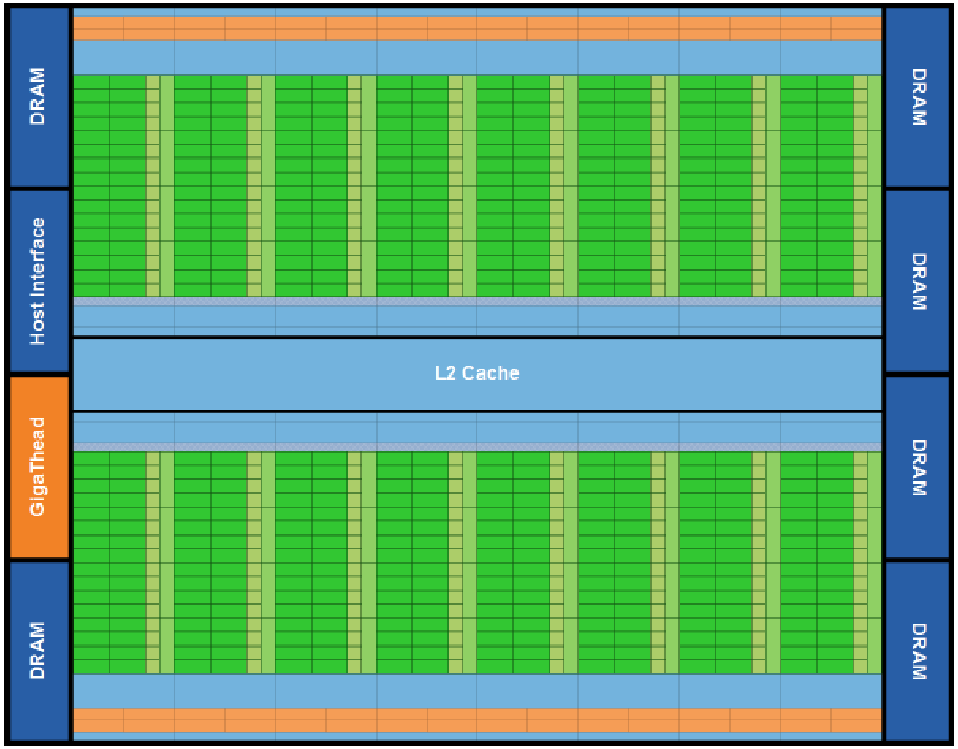
Agenda

1. Introduction
2. Related Work
3. CUDA Architecture
4. System Design
5. Implementation and Microbenchmarks
6. Case Study: Molecular Dynamics
7. Conclusion and Future Work
8. References

CUDA Architecture [1]



Fermi Streaming Multiprocessor (SM)



Agenda

1. Introduction
2. Related Work
3. CUDA Architecture
4. System Design
5. Implementation and Microbenchmarks
6. Case Study: Molecular Dynamics
7. Conclusion and Future Work
8. References

System Design

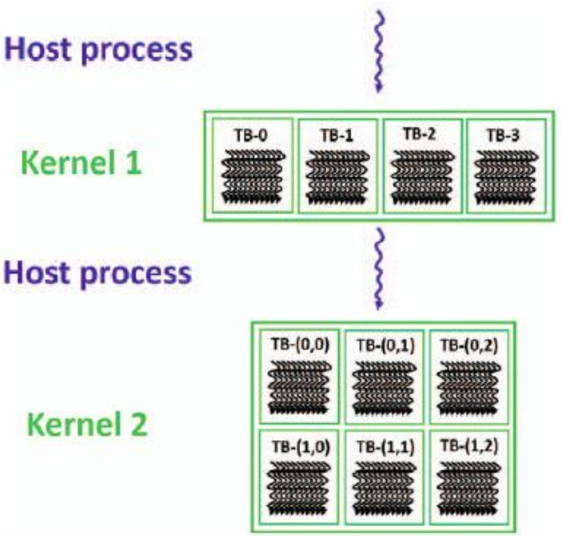


Figure 1: CUDA programming paradigm

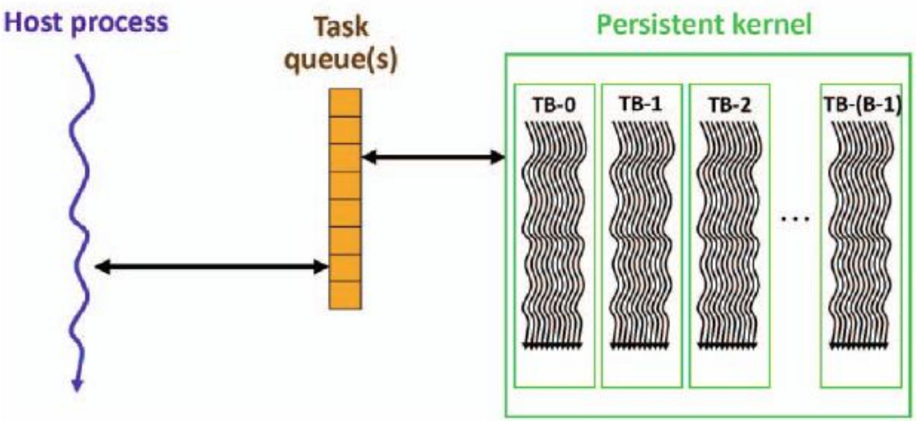
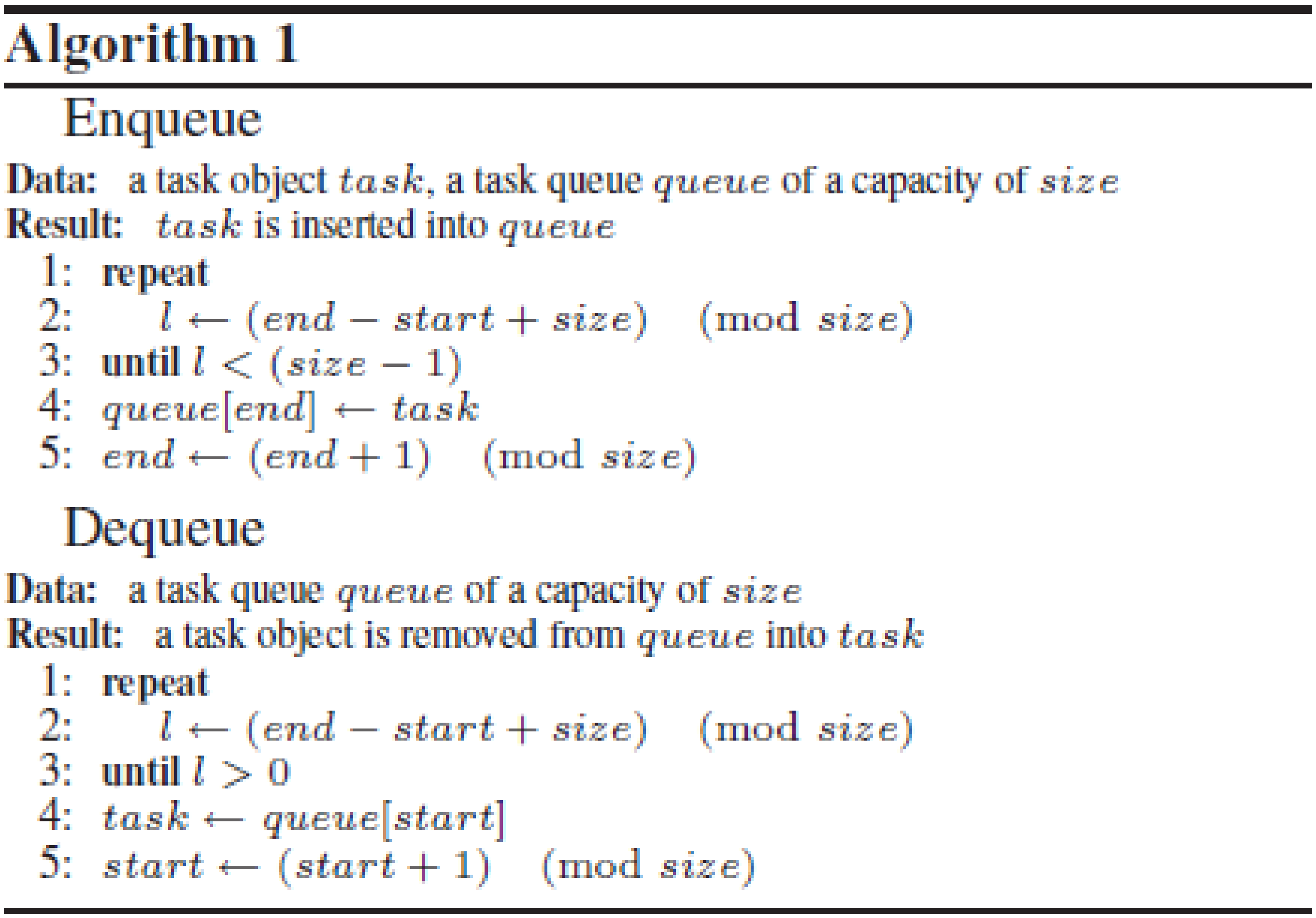


Figure 2: Task queue paradigm

System Design

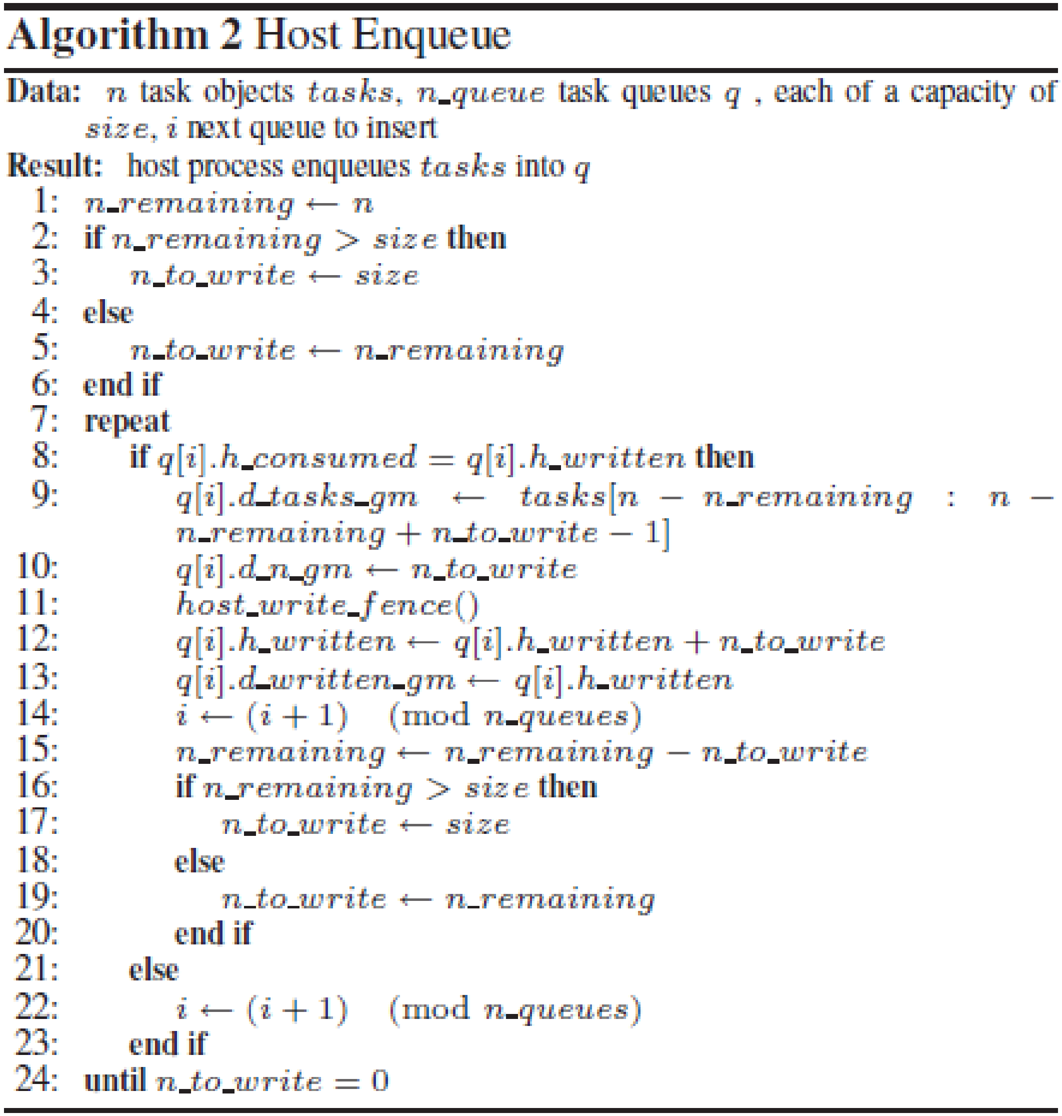


Algorithm 1

```
Enqueue
Data: a task object $task$, a task queue $queue$ of a capacity of $size$
Result: $task$ is inserted into $queue$
repeat
    $l \leftarrow (end - start + size) \pmod{size}$
until $l < (size - 1)$
$queue[end] \leftarrow task$
$end \leftarrow (end + 1) \pmod{size}$

Dequeue
Data: a task queue $queue$ of a capacity of $size$
Result: a task object is removed from $queue$ into $task$
repeat
    $l \leftarrow (end - start + size) \pmod{size}$
until $l > 0$
$task \leftarrow queue[start]$
$start \leftarrow (start + 1) \pmod{size}$
```

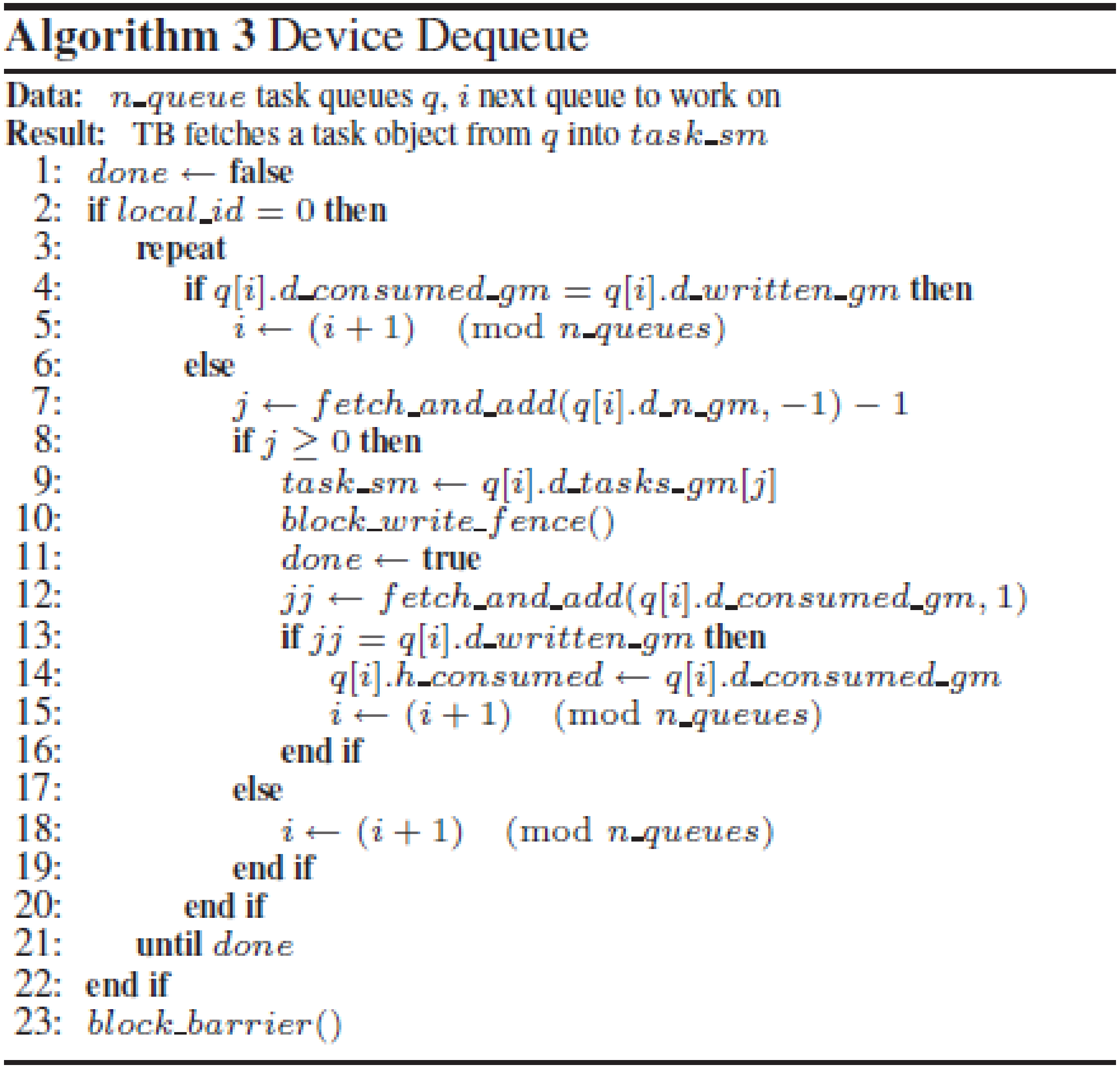
System Design



Algorithm 2 Host Enqueue

```
Data: $n$ task objects $tasks$, $n\_queue$ task queues $q$ , each of a capacity of $size$, $i$ next queue to insert
Result: host process enqueues $tasks$ into $q$
$n\_remaining \leftarrow n$
if $n\_remaining > size$ then
    $n\_to\_write \leftarrow size$
else
    $n\_to\_write \leftarrow n\_remaining$
end if
repeat
    if $q[i].h\_consumed = q[i].h\_written$ then
        $q[i].d\_tasks\_gm \leftarrow tasks[n - n\_remaining : n - n\_remaining + n\_to\_write - 1]$
        $q[i].d\_n\_gm \leftarrow n\_to\_write$
        $host\_write\_fence()$
        $q[i].h\_written \leftarrow q[i].h\_written + n\_to\_write$
        $q[i].d\_written\_gm \leftarrow q[i].h\_written$
        $i \leftarrow (i + 1) \pmod{n\_queues}$
        $n\_remaining \leftarrow n\_remaining - n\_to\_write$
        if $n\_remaining > size$ then
            $n\_to\_write \leftarrow size$
        else
            $n\_to\_write \leftarrow n\_remaining$
        end if
    else
        $i \leftarrow (i + 1) \pmod{n\_queues}$
    end if
until $n\_to\_write = 0$
```

System Design



Algorithm 3 Device Dequeue

```
Data: $n\_queue$ task queues $q$, $i$ next queue to work on
Result: TB fetches a task object from $q$ into $task\_sm$
$done \leftarrow$ false
if $local\_id = 0$ then
    repeat
        if $q[i].d\_consumed\_gm = q[i].d\_written\_gm$ then
            $i \leftarrow (i+1) \pmod{n\_queues}$
        else
            $j \leftarrow fetch\_and\_add(q[i].d\_n\_gm, -1) - 1$
            if $j \geq 0$ then
                $task\_sm \leftarrow q[i].d\_tasks\_gm[j]$
                $block\_write\_fence()$
                $done \leftarrow$ true
                $jj \leftarrow fetch\_and\_add(q[i].d\_consumed\_gm, 1)$
                if $jj = q[i].d\_written\_gm$ then
                    $q[i].h\_consumed \leftarrow q[i].d\_consumed\_gm$
                    $i \leftarrow (i+1) \pmod{n\_queues}$
                end if
            else
                $i \leftarrow (i+1) \pmod{n\_queues}$
            end if
        end if
    until $done$
end if
$block\_barrier()$
```

System Design

Problems

- Copies between the host memory and device memory without interrupting the kernel execution
- Where to keep the queue and associated index variables
- How to guarantee the correctness of the queue operations in this host-device situation
- How to guarantee the correctness on accessing shared objects, if we allow dynamic load balance on the device

System Design

Solutions

- Asynchronous concurrent execution[2]: overlap the host-device data transfer with kernel execution
- Mapped host memory: enable the light-weight queue polling without generating host-device traffic
- Event: asynchronously monitor the device’s progress
- Atomic instructions: enable the non-blocking synchronization

Agenda

1. Introduction
2. Related Work
3. CUDA Architecture
4. System Design
5. Implementation and Microbenchmarks
6. Case Study: Molecular Dynamics
7. Conclusion and Future Work
8. References

System Environment

- 1 quad-core AMD Phenom II X4 940 processor
- 4 NVIDIA Tesla C1060 GPUs
- 64-bit Ubuntu version 8.10
- NVIDIA driver version 190.10
- CUDA Toolkit version 2.3
- CUDA SDK version 2.3
- GCC version 4.3.2

Implementation and Microbenchmarks

Host-device data transfer

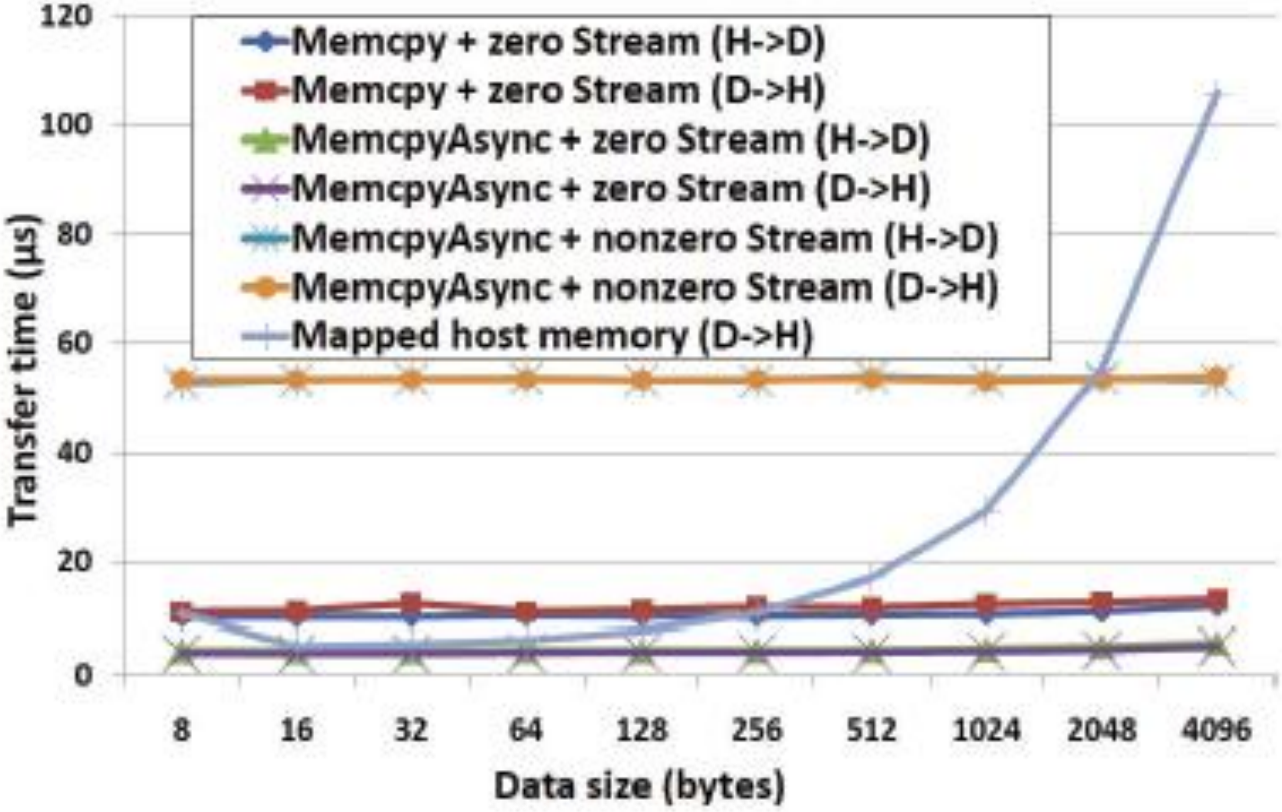


Figure 3: Data transfer time

Implementation and Microbenchmarks

Barrier and fence

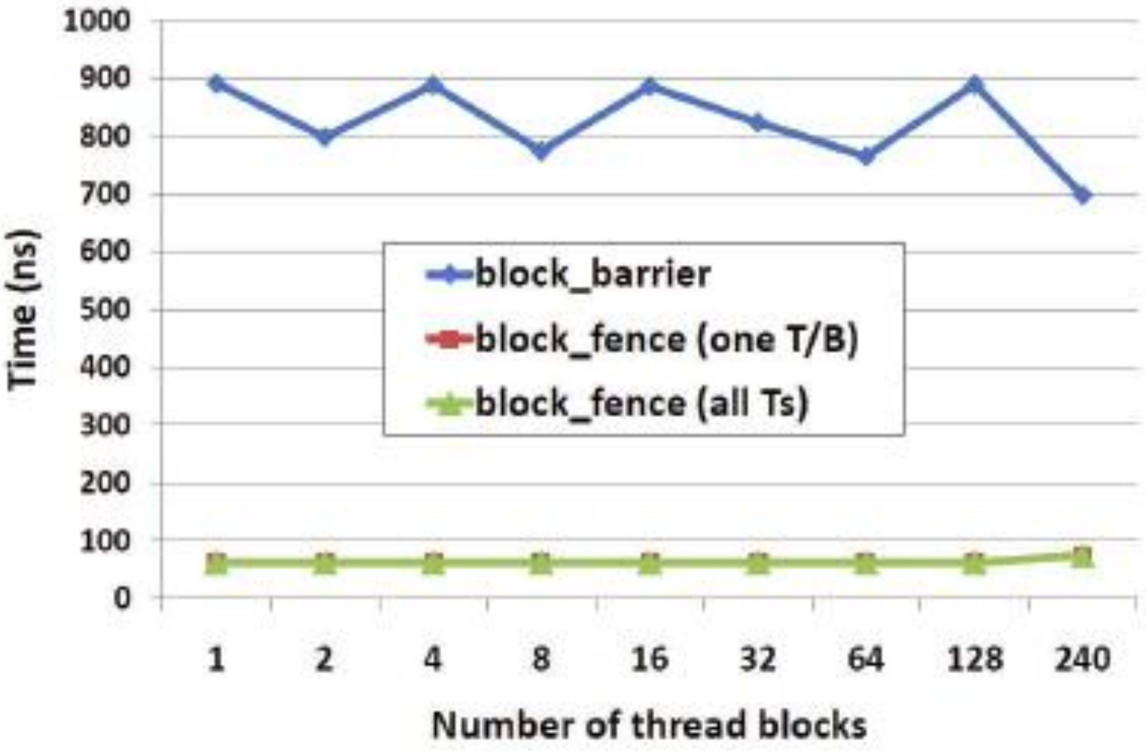


Figure 4: Barrier and fence functions (128Ts/B)

Implementation and Microbenchmarks

Atomic instructions

- One thread in each TB
  - A large number of *fetch-and-add* function -(access)→ Global memory address

327 [ns]

Implementation and Microbenchmarks

Task queue operations

- Average enqueue time: 114.3 [μs]

  2 PCIe transactions: 110 [μs] (95%)

  (120 tasks)

- Average dequeue time: 0.4 [μs]

  (128 threads/TB, 120 TBs)

Agenda

1. Introduction
2. Related Work
3. CUDA Architecture
4. System Design
5. Implementation and Microbenchmarks
6. Case Study: Molecular Dynamics
7. Conclusion and Future Work
8. References

Case Study: Molecular Dynamics

Molecular Dynamics

Newtonian dynamics of molecule/atom behavior

Gaussian distribution of helium atoms in a 3D box

Case Study: Molecular Dynamics

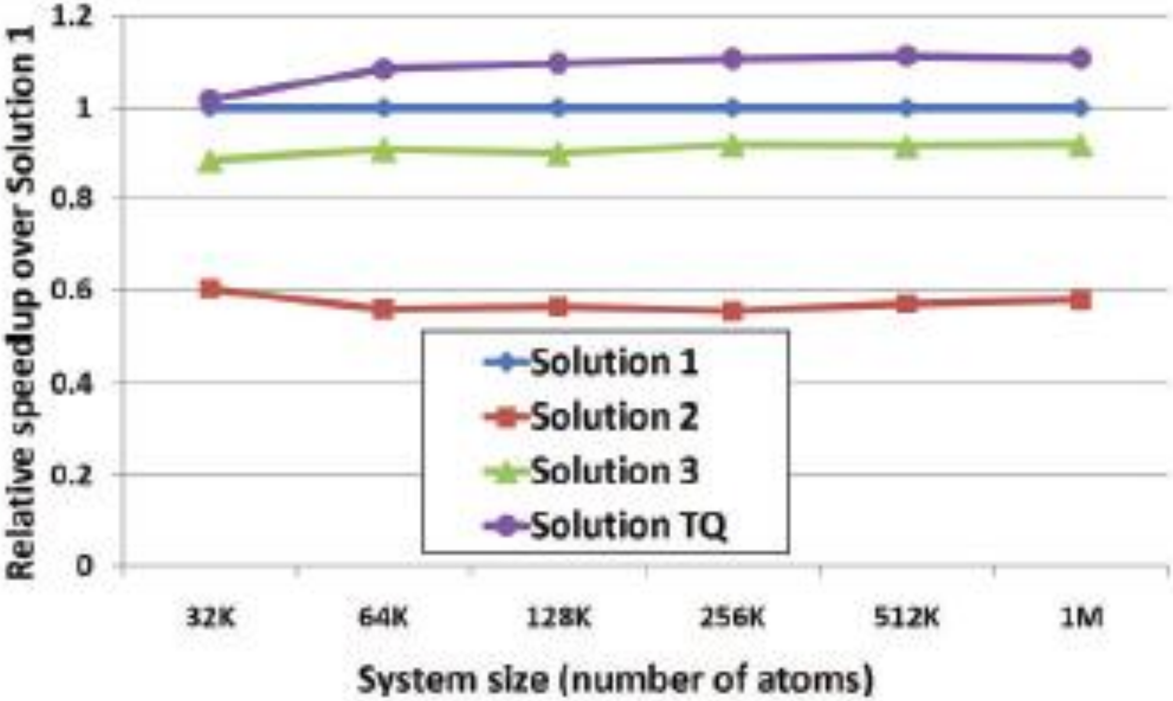


Figure 5: Relative speedup over Solution 1 versus system size (1 GPU)

Case Study: Molecular Dynamics

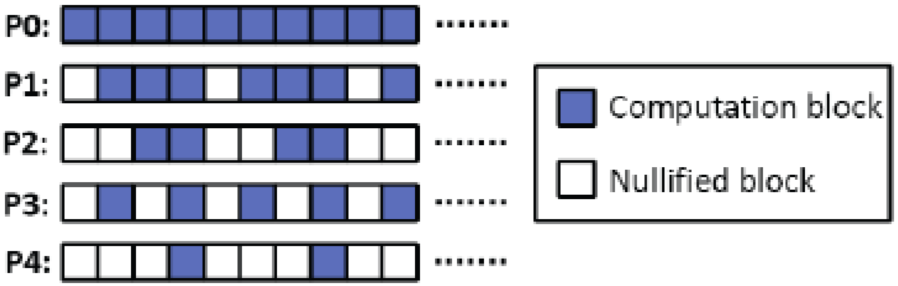


Figure 6: Workload patterns

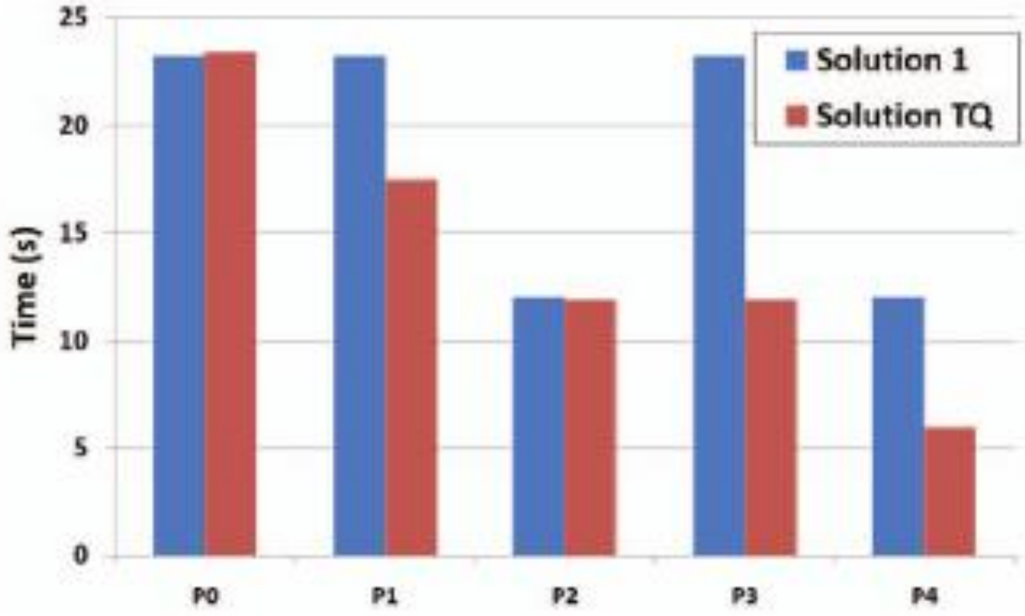


Figure 7: Runtime for different load patterns

Case Study: Molecular Dynamics

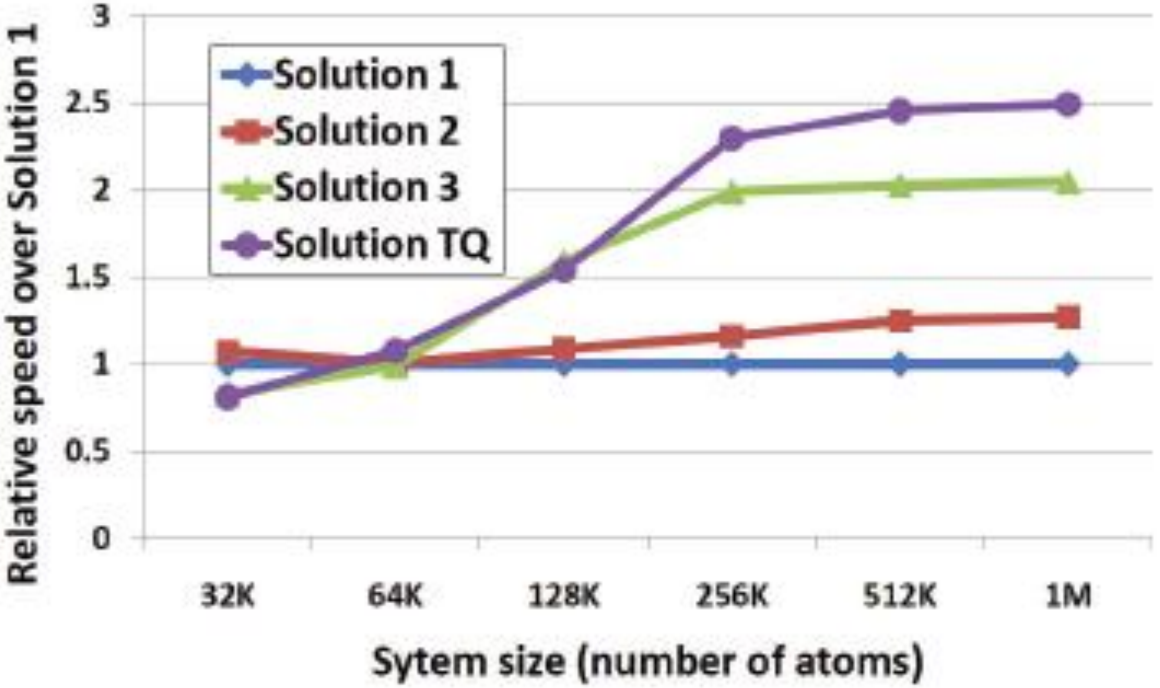


Figure 8: Relative speedup over Solution 1 versus system size (4 GPUs)

Case Study: Molecular Dynamics

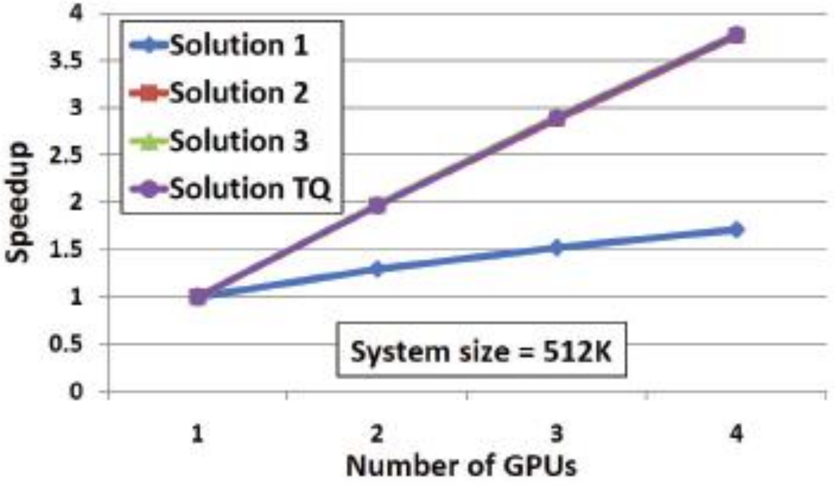


Figure 9: Speedup versus number of GPUs

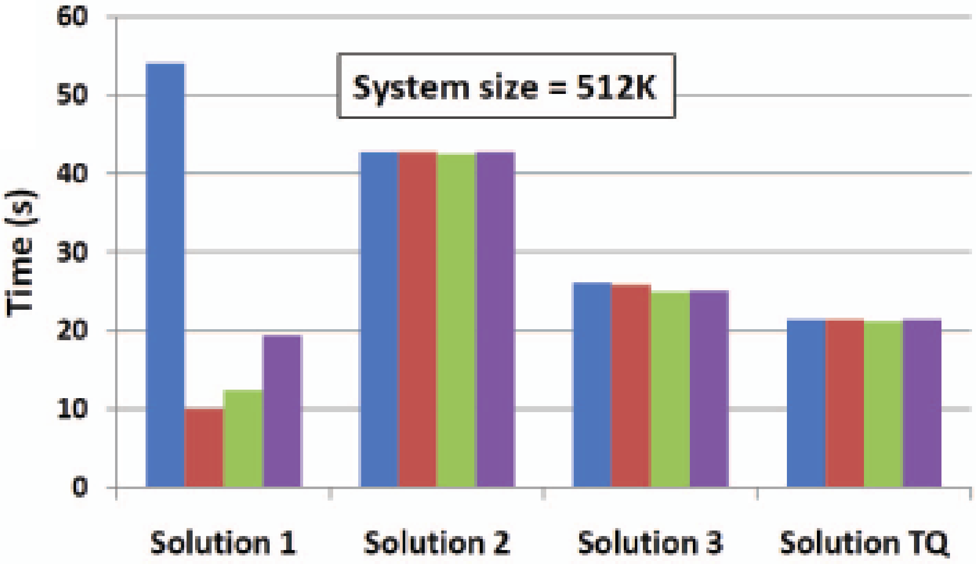


Figure 10: Dynamic load on GPUs

Agenda

1. Introduction
2. Related Work
3. CUDA Architecture
4. System Design
5. Implementation and Microbenchmarks
6. Case Study: Molecular Dynamics
7. **Conclusion and Future Work**
8. References

Conclusion and Future Work

Conclution

- the design of a dynamical load balance task queue scheme on single- and multi-GPU systems

- excellent speedup and performance improvement at a molecular dynamics application.

Future Work

- beneficial to other load imbalanced problems on GPU-enabled systems.

Comments

- Solution TQ have no defect ?
  What are the points we should consider ?
- How about the multi-CPU, multi-GPU situation ?
- OSS queue library should be available for further researches.
- Does CUDA have any plans to support these kind of queue schema ?

Agenda

1. Introduction
2. Related Work
3. CUDA Architecture
4. System Design
5. Implementation and Microbenchmarks
6. Case Study: Molecular Dynamics
7. Conclusion and Future Work
8. References

References

[1] Whitepaper NVIDIA’s Next Generation CUDA Compute Architecture Fermi http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

[2] NVIDIA CUDA C Programming Guide http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/NVIDIA_CUDA_C_ProgrammingGuide_3.1.pdf