A Tuned, Concurrent-Kernel Approach to Speed Up the APSP Problem

Hector Ortega-Arranz, Yuri Torres, Diego R. Llanos, and Arturo Gonzalez-Escribano

Universidad de Valladolid, Spain

{hector|yuri.torres|diego|arturo}@infor.uva.es

13th CMMSE International Conference, Almería, june 2013

Introduction

- Many real-world problems compute **shortest paths** from any source to any destination.
- The All-Pair Shortest-Path (**APSP**) problem is a well-known problem in graph theory whose objective is to find the shortest paths between any pair of nodes.
- The application of **GPGPU** computing to accelerate problems related with shortest-path problems have increased during the last years.
- The use of advanced optimizations as the correct choice of the ┃**threadBlock size**┃ and the use of ┃**concurrent kernels**┃ can improve even more the **GPU performance**.
- **Our goal**: To **squeeze** the performance of the GPU solution [1] for a real-life problem (APSP), following the recommendations of CUDA [2] and the guidelines described in [3].

Fermi architecture

Parameter	Fermi GF110
Number of SPs (per-SM)	32
Max. number of blocks (per-SM)	8
Max. number of threads (per-SM)	1 536
Max. number of threads (per-block)	1 024
Max. concurrent kernel supported	16
Max. Occupancy block sizes recommended by CUDA [2]	192, 256, 384, 512, 768
Block sizes for scatter access patterns recommended by [3]	64, 96, 128

GPU Dijkstra and the *relax kernel*

```
1: <<<initialize>>>(U, F, δ);
2: while (Δ ≠ ∞) do
3:    <<<relax>>>(U, F, δ);
4:    Δ =<<<minimum>>>(U, δ);
5:    <<<update>>>(U, F, δ, Δ);
6: end while
```

```
 1: tid = thread.Id;
 2: if (F[tid] == TRUE) then
 3:    for all suc successor of tid do
 4:       if (U[suc] == TRUE) then
 5:          BEGIN ATOMIC REGION
 6:          δ[suc] = min{δ[suc], δ[tid] + w(tid,suc)};
 7:          END ATOMIC REGION
 8:       end if
 9:    end for
10: end if
```

- U: Set of unsettled nodes
- F: Set of frontier nodes
- $δ$: Vector of tentative distances
- $Δ$: Iteration threshold

Optimization 1: ThreadBlock size

- **Not always Maximum Occupancy (MO)**: A common optimization to hide the memory latencies is the use of MO block sizes but not always achieves the best performance.
- **Kernel characterization**: $\uparrow \frac{\#low\_coalesced\_accesses}{\#instruc\_per\_thread}$

  - Best performance obtained with medium-occup. block sizes.
  - Medium-occup. block sizes alleviate the memory bottleneck and these blocks are evicted quicker than MO blocks.

- ┃**Hypothesis:**┃ *Relax kernel* performance would be improved using threadBlock sizes that lead to SM medium-occupancy.
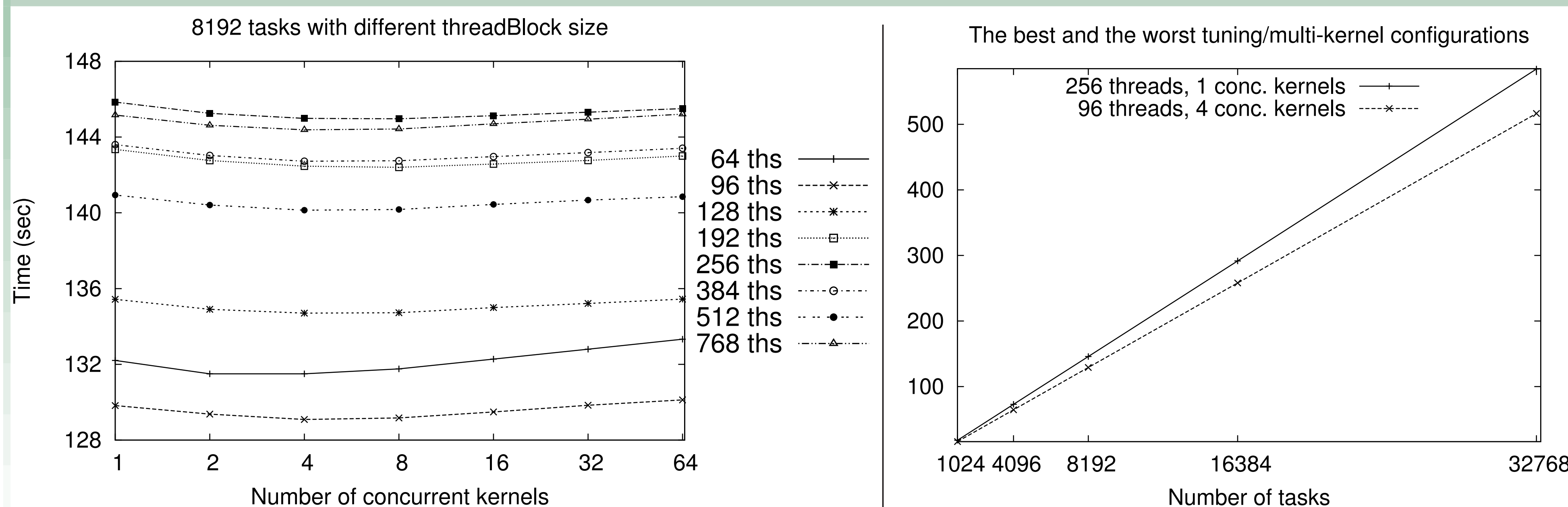
Optimization 2: Concurrent kernels

- Feature released since the 2nd CUDA architecture generation.
- Introduces a new level of parallelism automatically managed by the CUDA driver.
- Good performance for **small size kernels**.
  - Hardware resources are shared between concurrent kernels.
- Kernels with **bigger sizes** than available resources are queued, but they are **already launched**.
- ┃**Hypothesis:**┃ Queued kernels could take profit from the L1/L2 data-cache reutilization and the better block/warp dispatcher exploitation.

Experimental Setup

- Exhaustive simultaneous evaluation of **threadBlock size** and **concurrent kernel** optimization techniques on the GPU implementation described in [1].
- ThreadBlock sizes tested: ┃**192, 256, 384, 512 and 768**┃ recommended by CUDA and ┃**64, 96 and 128**┃ recomended by [3].
- We use sparse graphs with 1 049 088 nodes (multiple of recommended values).
- Due to the amount of computational load, we have reduced the APSP problem to 1 024, 4 096 and 8 192-source-node to all.
- Number of concurrent kernels tested: ┃**1, 2, 4, 8 and 16**┃ (maximum number supported by Fermi) and ┃**32, 64**┃ to observe an stressed concurrent environment.
- The worst and best configurations are tested with 16 384, 32 768-source-node to all.

Results

8192 tasks with different threadBlock size

64 ths, 96 ths, 128 ths, 192 ths, 256 ths, 384 ths, 512 ths, 768 ths

The best and the worst tuning/multi-kernel configurations

256 threads, 1 conc. kernels
96 threads, 4 conc. kernels

- Always, the best configuration for *relax kernel* is reached with 96 threads and 4 concurrent kernels.
- There are performance improvements from using 1 kernel until 4 - 8 kernels.
- Concurrent kernels better exploit the data-cache and block warp dispatchers.
- The use of more than 4 - 8 concurrent kernels leads to more memory bottlenecks and cache thrashing.
- The **performance gain** between the worst configuration and the best one is ┃**11.5%**┃.

Conclusions and future work

- We have **squeezed the performance of GPU architecture** for the *relax kernel* in a **11.5%**.
- The CUDA recommended configurations **do not always reach the best results**.
- The results corroborate the conclusion described in [3]:
  - Smaller block sizes than the smallest MO size present better performance.
  - Smaller blocks can be evicted from the SM quicker alleviating the memory bottleneck.
- We will test all L1 cache configurations to better exploit the memory hierarchy.
- Additionally, we want to extend the used techniques to optimize the rest of APSP kernels.

References

[1] Hector Ortega-Arranz and Yuri Torres and Diego R. Llanos and Arturo Gonzalez-Escribano A New GPU-based Approach to the Shortest Path Problem. To appear in Proceedings of High Performance Computing and Simulation (HPCS) 2013.

[2] David B. Kirk and Wen-mei W. Hwu Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann. Feb, 2010 ISBN: 978-0-12-381472-2.

[3] Yuri Torres, Arturo Gonzalez and Diego R. Llanos uBench: exposing the impact of CUDA block geometry in terms of performance The Journal of Supercomputing, pp. 1-14, 2013.