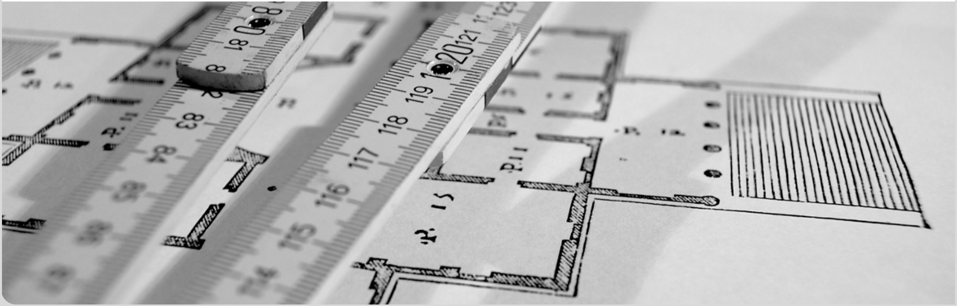
Application-independent Autotuning for GPUs

Martin Tillmann, Thomas Karcher, Carsten Dachsbacher, Walter F. Tichy

KARLSRUHE INSTITUTE OF TECHNOLOGY



Introduction

- Autotuning is an effective technique for optimizing parallel applications.
- Most work in the autotuning area has concentrated on application specific tuning parameters.
- On graphic processing units (GPUs), there are various tunable parameters that are *application independent*.
  - Number of threads per block
  - Loop unrolling
  - Workload per thread

$\Rightarrow$ Application-independent Autotuning for GPUs

Online Autotuning - AtuneRT

- Our autotuner AtuneRT uses a feedback loop of
  1. measuring execution time of a program section and
  2. adjusting the tuning parameter configuration.
- A search-based algorithm such as Nelder-Mead guides the process.
- As a general-purpose run-time autotuner, AtuneRT does *not* use application-specific knowledge about tuning parameters.

AtuneRT - API

AtuneRT is controlled with the following three functions:

- `addParameter(&pararm, range, default)`
- `startMeasurement()`
- `stopMeasurement()`

Autotuning on the GPU

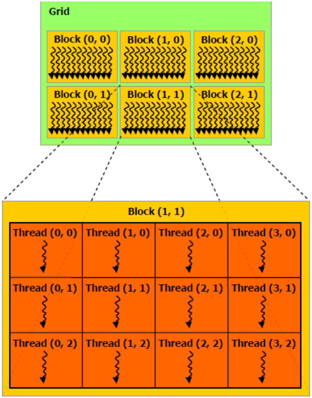
- Modern GPUs offer massive parallelism for a wide range of algorithms.
- Optimizing GPUs is hard.
  - Align memory access patterns
  - Minimize control flow costs
  - Balance workloads

Even though GPU compilers got better, most optimization is still done *by hand using trial and error*.

Autotuning on the GPU - CUDA

Karlsruhe Institute of Technology

In our work we employed NVIDIA's CUDA architecture.
CUDA is parallel programming model that organizes threads in a hierarchy.



Autotuning on the GPU - Block Size

- The number of overall threads is dictated by the number of data elements.
- The number of threads per block (**block size**) is *variable*.
- Due to limited resources for each block not all threads can be active.

Common approach:
Trial and error hand-optimized code for one GPU generation.

Autotuning on the GPU

Online autotuning removes the need for hand-tuning and has multiple advantages:

- No knowledge of the GPUs specification is required.
- React to changes at run-time.
- Tune multiple non-independent parameters.
- Fast and easy to use.

Results

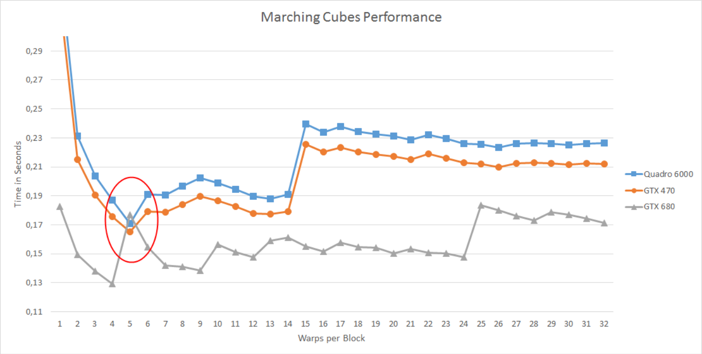
Evaluation on three GPUs:

Model	Generation
Geforce GTX 680	GK104
Geforce GTX 470	GF100
Quadro 6000	GF100GL

We examined the performance of four applications with all possible tuning configurations.
We also measured the iterations it took AtuneRT to reach the optimal values and resulting speed-up.

Results - Marching Cubes

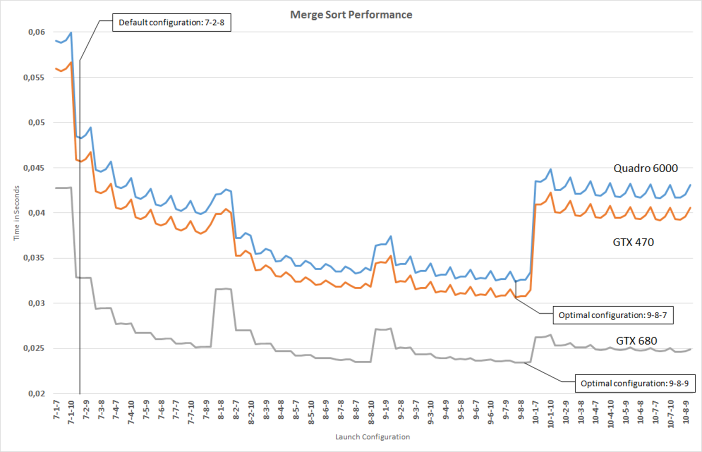
Marching Cubes - block size of one kernel - complete parameter space



What is optimal for one GPU can be the worst case for another.

Results - Merge Sort

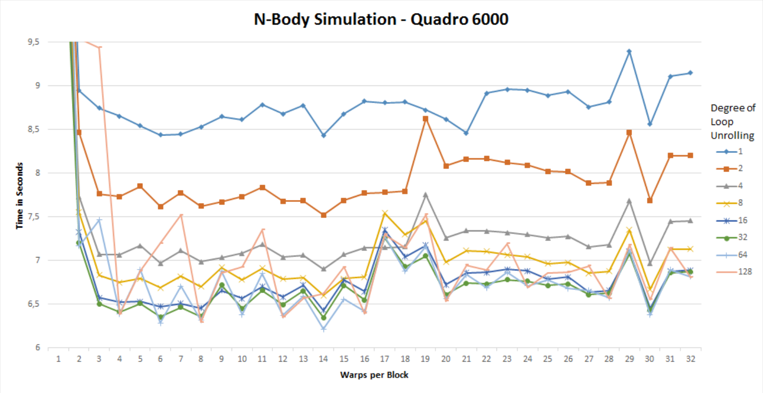
Merge Sort - block sizes of three kenerls - complete parameter space



Even with similar behavior the optimum is different.

Results - N-Body Simulation

N-Body Simulation - block size and degree of loop unrolling



Tuning parameters (here: **degree of loop unrolling** and **block size**) are not independent of each other.

Optimization matters – over 20% speed-up only through loop unrolling.

Results - Thrust

- Thrust is a C++ template library for CUDA, similar to the Standard Template Library (STL).
- Thrust's algorithms either use hard-coded values or simple heuristics to determine the **block size** of the CUDA kernels.

	time in seconds	block size
GTX 680, no tuner	2.566	224
GTX 680, with tuner	2.223	128
Quadro 6000, no tuner	10.025	224
Quadro 6000, with tuner	10.026	224

- Tuning the `thrust::inclusive_scan`-function results in a speed-up of 13% on the GTX 680.
- Thrust optimizes for older generation GPUs like the Quadro 6000.

Conclusion

Karlsruhe Institute of Technology

- Tuning application independent parameters is important.
- Autotuning is feasible for optimizing GPU applications on multiple platforms.
- Preparing the applications for AtuneRT was easy: Three calls to the tuner sufficed.
- The tuner could be integrated in the kernel API call.
- No knowledge of hardware specifications is required.

We expect autotuning to become an essential part in determining tuning parameters at run-time on the GPU.

References I

Jason Ansel et al. "Language and compiler support for auto-tuning variable-accuracy algorithms". In: *Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization*. IEEE Computer Society. 2011, pp. 85–96.

"CUDA C Best Practices Guide". In: 2012, p. 39.

Mark Harris et al. "Optimizing parallel reduction in CUDA". In: *NVIDIA Developer Technology* 2 (2007).

Thomas Karcher and Victor Pankratius. "Run-time automatic performance tuning for multicore applications". In: *Euro-Par 2011 Parallel Processing*. 2011, pp. 3–14.

Anna Morajko, Tomàs Margalef, and Emilio Luque. "Design and implementation of a dynamic tuning environment". In: *Journal of Parallel and Distributed Computing* 67.4 (2007), pp. 474–490.

References II

- John A. Nelder and Roger Mead. “A simplex method for function minimization”. In: *The computer journal* 7.4 (1965), pp. 308–313.
- Frank Otto et al. “A language-based tuning mechanism for task and pipeline parallelism”. In: *Proceedings of the 16th international Euro-Par conference on Parallel processing: Part II*. Euro-Par’10. Ischia, Italy, 2010, pp. 328–340. ISBN: 3-642-15290-2, 978-3-642-15290-0. URL: `http://dl.acm.org/citation.cfm?id=1885276.1885309`.
- Cristian Ţăpuş, I-Hsin Chung, Jeffrey K Hollingsworth, et al. “Active harmony: towards automated performance tuning”. In: *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*. IEEE Computer Society Press. 2002, pp. 1–11.

References III

Yuri Torres, Arturo Gonzalez-Escribano, and Diego R Llanos. "Understanding the impact of CUDA tuning techniques for Fermi". In: *High Performance Computing and Simulation (HPCS), 2011 International Conference on*. IEEE. 2011, pp. 631–639.

Vasily Volkov. "Better performance at lower occupancy". In: *Proceedings of the GPU Technology Conference, GTC*. Vol. 10. 2010.

Clint Whaley, Antoine Petitet, and Jack J. Dongarra. "Automated empirical optimizations of software and the ATLAS project". In: *Parallel Computing* 27.1 (2001), pp. 3–35.

Henry Wong et al. "Demystifying GPU microarchitecture through microbenchmarking". In: *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*. 2010, pp. 235–246. DOI: 10.1109/ISPASS.2010.5452013.

Results - Merge Sort

	default time	optimal time	speed-up	autotuning iterations
Quadro 6000	0.0483	0.0324	32.81%	17
GTX 470	0.0457	0.0307	32.85%	20
GTX 680	0.0328	0.0234	28.64%	20

Execution times in seconds. Speed-up of the optimal configuration relative to the default parameters.

Autotuning on the GPU - Time measurement

Measuring time on the GPU is not accurate. GPU computation is initiated via driver calls (with internal scheduling).

Time is measured by inserting events in the execution pipeline.