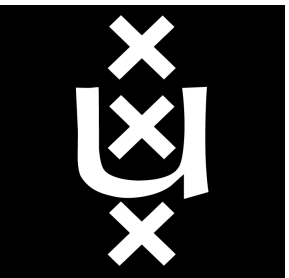
An Empirical Evaluation of GPGPU Performance Models

S. Madougou, A. Varbanescu, C. de Laat and R. van Nieuwpoort

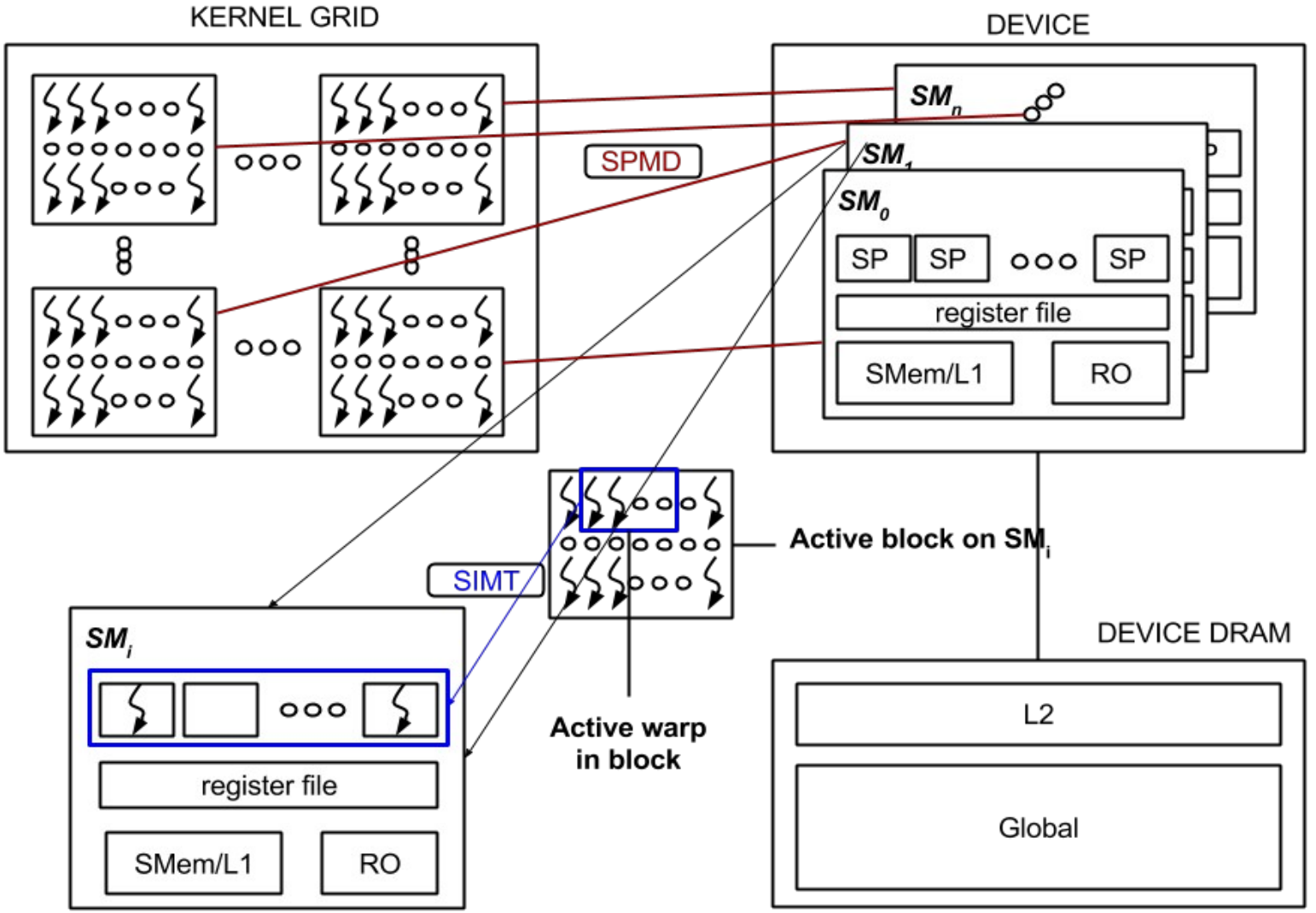
Hetero-Par 2014, Porto, Portugal



Motivation

- Ubiquity of parallel hardware (multicore, manycore, clusters, grid, clouds)
- Promise of very high performance but very challenging to achieve
  - Peak performance requires hardware specific features
  - Exploration of large design and optimization space
- Performance modeling to help high performance “affordability”
  - Systematic and portable vs in-house expertise and per case
- GPUs as the most common parallel architectures

GPU execution model



GPU performance factors

- Maximize parallel execution
  - More independent work in a thread (ILP)
  - More concurrent threads (TLP)
  - More independent memory accesses (MLP)
  - Good utilization of the hardware (occupancy)
- Maximize memory throughput
  - Memory coalescing and access patterns
  - Shared memory bank conflicts and access patterns
  - Caching effects
- Maximize instruction throughput
  - Instruction mix, instruction serialization

GPU performance modeling

- Model = application model + hardware model
- Accuracy of prediction
- Evaluation speed
- Easy model construction and evaluation
- Capture of salient performance factors
- Performance bottlenecks highlighting

Evaluating 7 GPGPU models

- Trend setters and/or promising
- Simple benchmark: dense matrix multiplication
  - From CUDA SDK for MxM square matrices
  - Uses BxB block matrices, M multiple of B
  - Optimized by use of shared memory
  - Memory-bound kernel

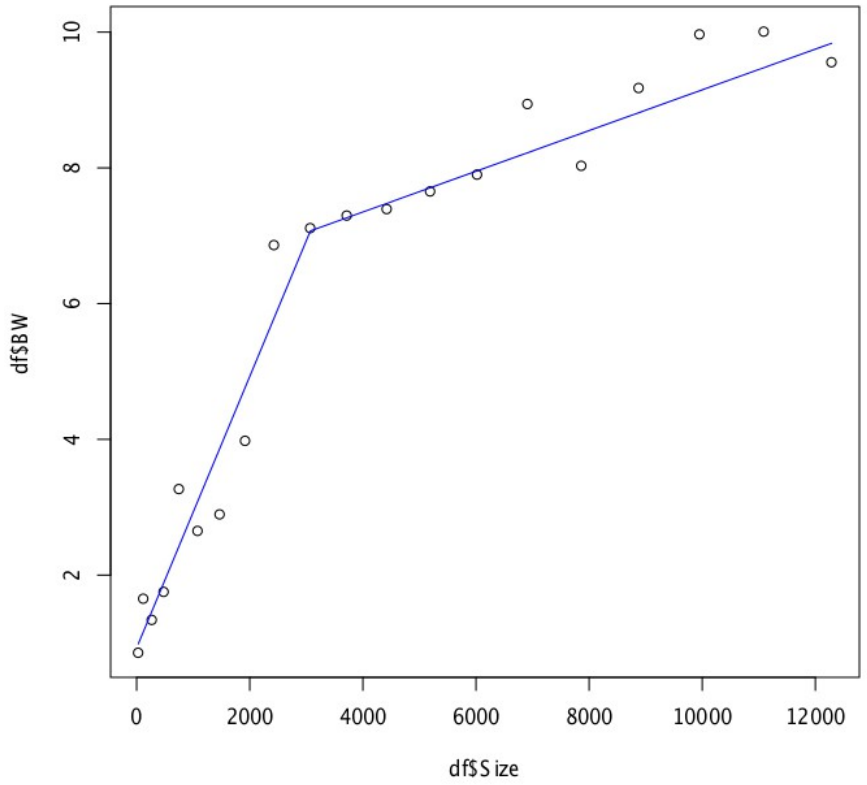
PMAC framework [1]

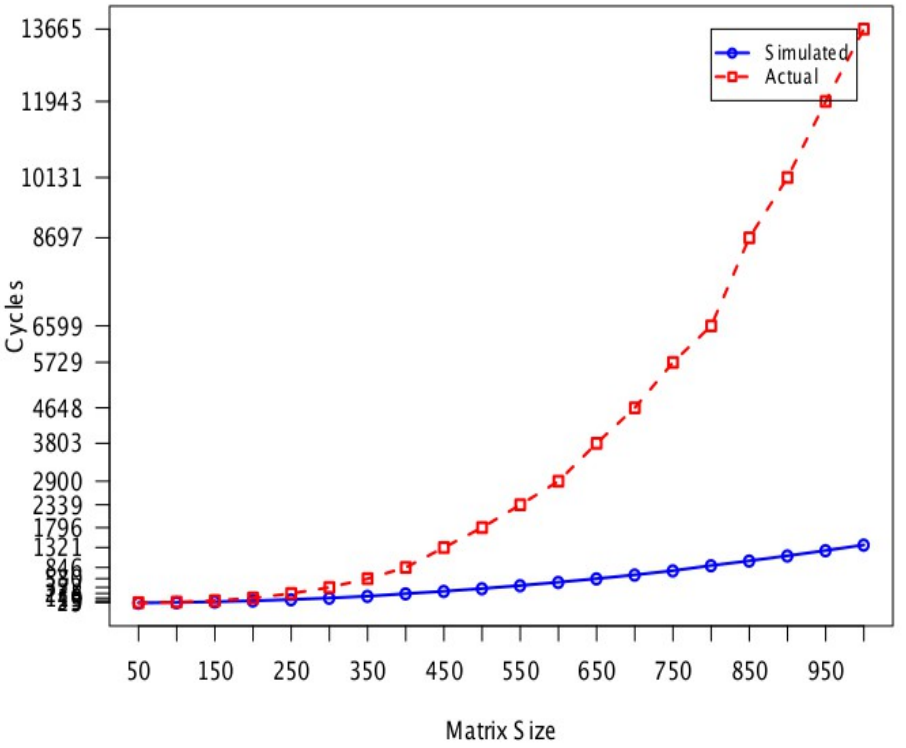
- PMAC: performance analysis of distributed systems
- Extension with tools to handle heterogeneity
- Based on *idioms* recognition and modeling
- Uses micro-benchmarking and binary instrumentation
- Tested on a few idioms on GPUs, acc 80-90%

PMAC evaluation I

- PMAC estimates only memory operations
- $$MemTime = \sum_{i}^{allBB} \frac{MemRef_{i,j} \times RefSize}{MemBW_{stream}} \quad (1)$$
- $MemBW_{stream}$ regression model built per accelerator using micro-benchmarking
- $$MemBW_{stream}(s) = -0.0020 \times max(0, 3072 - s) + 0.0003 \times max(0, s - 3072) + 7.0709$$

PMAC evaluation II



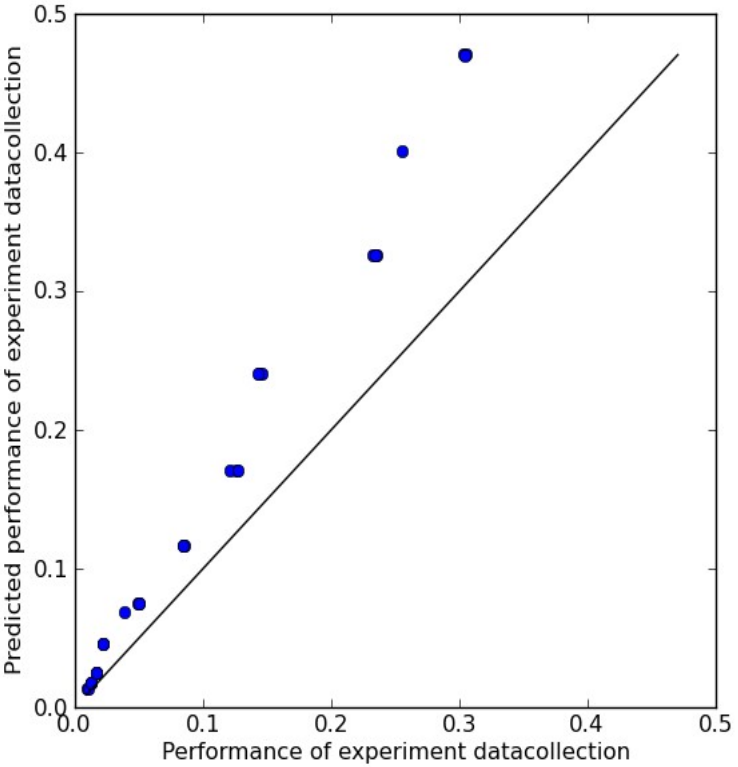


Eiger framework [2]

- Automated statistical methodology to model program behavior on different architectures (CPU+GPU)
- Synthesizes performance models through:
  - Experimental data acquisition and DB construction
  - Series of data analysis passes (PCA)
  - Model selection and construction
- Captures major performance factors (47 metrics)
- Software toolchain (simulator) poorly documented
- Validation on 12 benchmarks from CUDA SDK

Eiger evaluation

Eiger metric	Performance counter
Memory efficiency	(gld_eff+gst_eff) / 2
Memory intensity	ldst_exec / inst_exec
Memory sharing	Code analysis
Activity factor	CUDA occupancy
SIMD/MIMD	Exec configuration
DMA size	Code analysis



STARGAZER framework [3]

- Automated GPU performance exploration
  - Sparsely and randomly samples the parameter values of the full GPU design space
  - Simulates or measures values for each parameter
  - Uses stepwise regression to find the most influential parameters to performance
- Interactions between parameters modeled
- Validation with benchmarks (accuracy ~99%)

STARGAZER evaluation

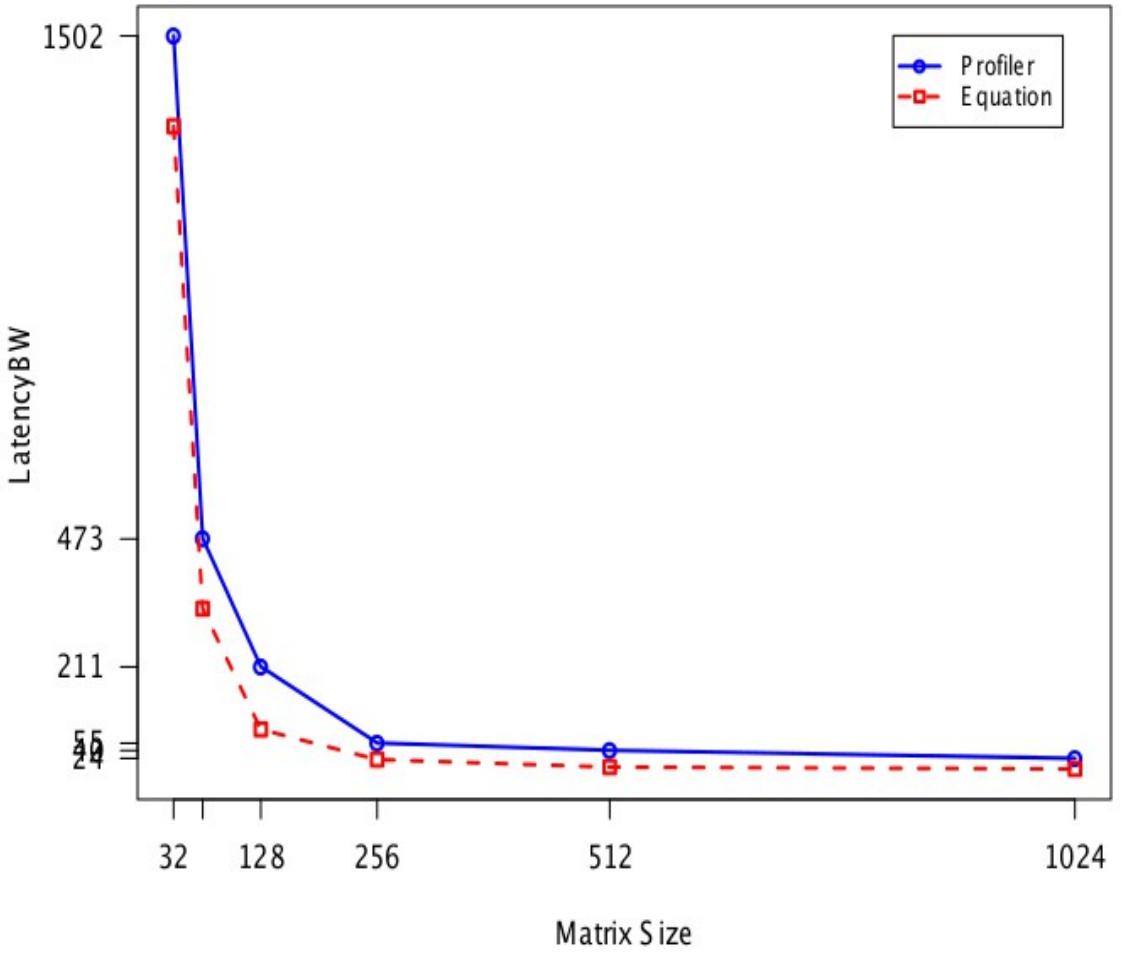
- STARGAZER only considers hardware characteristics (design space pruning)
- No application metrics s.a. bank conflicts nor flow divergence
- Uses GPGPU-Sim to collect parameter values
  - It can take days to gather experimental data
- Direct measurements as alternative but challenging
- Predicts GPGPU-Sim times reasonably well
  - Simulated times order of magnitude different from actual times

WFG modeling tool [4]

- Kernel execution time based on its work flow graph (WFG)
- The WFG is built from kernel dependence graph (both control flow + data dependence)
- Both transition and dependence arcs are labeled with cycles estimates
- Captures major performance factors (-caching)
- Validation with 4 kernels with good accuracy

WFG evaluation

WFG metric	Performance counter
LatencyBW	(1-sm_eff) x stall_data_req / (warps x cyc_sm)
CYCcompute	inst_wp x CPI
NUMmem	gld_req + gst_req
CYCmem	NUMmem x WS x bw_sm / warps



MWP-CWP analytical model [5]

- Performance model built from 2 metrics
  - Memory warp parallelism (MWP)
  - Compute warp parallelism (CWP)
- Model based on 17 hardware and application parameters
- Parameters extracted either from hardware spec or source and PTX code
- Validation on 2 Nvidia GPUs

MWP-CWP evaluation

- Evaluation attempt of the model on newer GPU
- 5 of the 17 parameters require micro-benchmarking but benchmark suite deprecated by authors
- Using approximation of those parameters leads to far off results
- Recalibration for new hardware and in-depth analysis for new applications code
- The model only predicts execution time, no bottlenecks highlighting

GPU *à la* PRAM [6]

- Analytical mode based on BSP and PRAM
- The model uses 7 platform parameters and 6 application parameters
- Execution time estimated by “mapping” the dataset on the threads and evaluating cycles
- Application characterization (cycles per thread) done by calibration or source code analysis
- Validation on few applications with good accuracy

GPU *à la* PRAM evaluation

- Evaluation on a GTX480 with good accuracy (3-10% error)
- Evaluation on a GT-Titan with bad accuracy (30-70% error)
- Calibration of the model for new applications is tedious
- So is analyzing applications with complex data items to threads mapping

A quantitative analysis [7]

- The model first measures *everything* about the hardware
- Then, application model expressed in terms of consumption of the hardware resources
- Bottlenecks are detected by hardware resources usage within execution time
- Application model is a detailed breakdown of the instructions in the code
- Validation on benchmarks with accuracy within 15%

A quantitative analysis evaluation

- Evaluation of the model requires the benchmark suite not available
- Shared and global memory analysis performed on non-cache architectures -> code instrumentation cache-aware?
- The model gives insight into causes of performance behavior
- Unsure whether the approach will work when caches plays an important role

Conclusion and future work

- An overview of current GPGPU performance modeling landscape
- Description and evaluation of 7 models
- Results certainly improvable with proper doc
- Extension into a comprehensive survey using benchmark suite

References

[1] Allan Snavely, Laura Carrington, Nicole Wolter, Jesus Labarta, Rosa Badia, and Avi Purkayastha. A framework for performance modeling and prediction. In Proceedings of SC '02, pages 1{17, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press

[2] Andrew Kerr, Eric Anger, Gilbert Hendry, and Sudhakar Yalamanchili. Eiger: A framework for the automated synthesis of statistical performance models. In Proceedings of WPEA 2012, 2012.

[3] Wenhao Jia, K.A. Shaw, and M. Martonosi. Stargazer: Automated regression-based gpu design space exploration. In ISPASS 2012, pages 2{13, April 2012.

[4] Sara S. Baghsorkhi, Matthieu Delahaye, Sanjay J. Patel, William D. Gropp, and Wen-mei W. Hwu. An adaptive performance modeling tool for gpu architectures. SIGPLAN Not., 45(5):105{114, January 2010.

[5] Sunpyo Hong and Hyesoon Kim. An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness. SIGARCH Comput. Archit. News, 37(3):152{163, June 2009.

[6] K. Kothapalli, R. Mukherjee, M.S. Rehman, S. Patidar, P. J. Narayanan, and K. Srinathan. A performance prediction model for the cuda gpgpu platform. In HiPC 2009, pages 463{472, Dec 2009.

[7] Yao Zhang and J.D. Owens. A quantitative performance analysis model for gpu architectures. In HPCA 2011, pages 382{393, Feb 2011.