A cache-aware performance prediction framework for GPGPU computations

The 8th Workshop on UnConventional High Performance Computing 2015

Alexander Pöppl, Alexander Herz

August 24th, 2015

Agenda

Introduction

- Motivation
- Contributions
- Example

Model

- Execution Time Computation
- Memory Transfer
- Empty Kernels
- Workgroup Size
- Basic Operations
- Memory accesses

Evaluation

- Qualitative Evaluation
- Quantitative Evaluation

Further Work

Introduction

Motivation

- OpenCL is used for running heterogeneous HPC applications
- It is low level, fairly explicit, and has manual task management

[1] Cédric Augonnet et al. "StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures". English. In: **Euro-Par 2009 Parallel Processing**. Ed. by Henk Sips, Dick Epema, and Hai-Xiang Lin. Vol. 5704. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 863–874. ISBN: 978-3-642-03868-6. DOI: 10.1007/978-3-642-03869-3_80. URL: http://dx.doi.org/10.1007/978-3-642-03869-3_80.

[2] Gregory F. Diamos and Sudhakar Yalamanchili. "Harmony: An Execution Model and Runtime for Heterogeneous Many Core Systems". In: **Proceedings of the 17th International Symposium on High Performance Distributed Computing**. HPDC '08. Boston, MA, USA: ACM, 2008, pp. 197–200. ISBN: 978-1-59593-997-5. DOI: 10.1145/1383422.1383447. URL: http://doi.acm.org/10.1145/1383422.1383447.

Introduction

Motivation

- OpenCL is used for running heterogeneous HPC applications
- It is low level, fairly explicit, and has manual task management
- Hence runtime systems with schedulers, such as StarPU[1] or Harmony[2] have been developed
- These schedule tasks onto heterogeneous hardware based on expected runtime.
- High-quality estimations crucial for efficient schedules.

[1] Cédric Augonnet et al. "StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures". English. In: **Euro-Par 2009 Parallel Processing**. Ed. by Henk Sips, Dick Epema, and Hai-Xiang Lin. Vol. 5704. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 863–874. ISBN: 978-3-642-03868-6. DOI: 10.1007/978-3-642-03869-3_80. URL: http://dx.doi.org/10.1007/978-3-642-03869-3_80.

[2] Gregory F. Diamos and Sudhakar Yalamanchili. "Harmony: An Execution Model and Runtime for Heterogeneous Many Core Systems". In: **Proceedings of the 17th International Symposium on High Performance Distributed Computing**. HPDC '08. Boston, MA, USA: ACM, 2008, pp. 197–200. ISBN: 978-1-59593-997-5. DOI: 10.1145/1383422.1383447. URL: http://doi.acm.org/10.1145/1383422.1383447.

Introduction

Motivation

- Performance Prediction models already exist, and work well with earlier GPU architectures.
- Introduction of Caches complicate predictions.
- GPU memory Hierarchy needs to be considered.

Introduction

Contributions

- Categorization of memory accesses into classes with distinct performance characteristics.
- Fully static OpenCL computation prediction model.
- Evaluation using randomly generated OpenCL kernels shows that a cache-aware model improves predictions.

Introduction

Example

- Popular operation: **Stencil operations**
- Array of size: $n * m$

$$b(i,j) = a(i,j)^2 - a(1,j)$$

Introduction

Example

$$n_{\mathrm{WI}} = m * n$$

$$mem_{GPU}^{input} \leftarrow device.alloc(n_{\mathrm{WI}} * s_{\mathrm{WI}})$$

$$mem_{GPU}^{output} \leftarrow device.alloc(n_{\mathrm{WI}} * s_{\mathrm{WI}})$$

$$\text{copyDataToGPU}(\rightarrow mem_{GPU}^{input})$$

$$device.kernel(n_{\mathrm{WI}}, n_{\mathrm{WG}}, m, n)$$

$$\Rightarrow \forall id \in \{0, .., n_{\mathrm{WI}}\}.\ sq\_mod(mem_{GPU}^{input}, mem_{GPU}^{output}, m, n)$$

$$\text{copyDataFromGPU}(\rightarrow mem_{GPU}^{output})$$

Introduction

Example

```
kernel void sq_mod(global float *matrix,
                   global float *res,
                   unsigned int m, unsigned int n) {
  size_t current_pos = get_global_id(0);
  unsigned int current_row = current_pos / n;
  unsigned int current_col = current_pos % n;
  res[current_pos] = matrix[current_row * n
                            + current_col]
       * matrix[current_row * n + current_col]
       - matrix[current_col];
}
```

Model

Execution Time Computation — Computation of the Runtime

$$t(n_{\mathrm{WI}}, s_{\mathrm{WI}}, n_{\mathrm{WG}}) = t_{\mathrm{Transfer}}(n_{\mathrm{WI}}, s_{\mathrm{WI}}) + t_{\mathrm{Kernel}}(n_{\mathrm{WI}}, n_{\mathrm{WG}})$$

$$t_{\mathrm{Kernel}}(n_{\mathrm{WI}}, n_{\mathrm{WG}}) = \frac{t_{\mathrm{Base}}(n_{\mathrm{WI}}) + \sum_{Op \in \mathrm{Expr.\text{-}Types}} W_{\mathrm{Op}}(n_{\mathrm{Op}}) t_{\mathrm{Op}}(n_{\mathrm{WI}})}{U(n_{\mathrm{WG}}, n_{\mathrm{XU}})}$$

n_{WI}	Number of work-items
n_{WG}	Number of work-items per work-group
s_{WI}	Size of a work-item in bytes
n_{XU}	Number of execution units on the GPU

Model

Memory Transfer

- GPUs have a dedicated portion of memory for their computations
- Time for memory transfer governed by two variables
  - bw Bandwidth
  - l_{prop} Propagation latency

Model

Memory Transfer

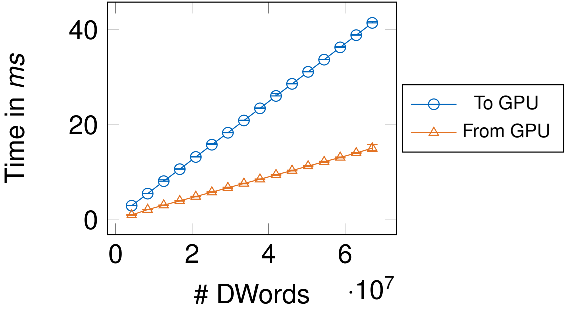


Figure: Memory Transfer times

- $t_{\text{trans}}^{\text{to}}(n_{\text{WI}}) = bw_{\text{to}}^{-1}\ n_{\text{WI}} + l_{\text{to}}$
- $t_{\text{trans}}^{\text{from}}(n_{\text{WI}}) = bw_{\mathit{from}}^{-1}\ n_{\text{WI}} + l_{\text{from}}$

Model

Empty Kernels

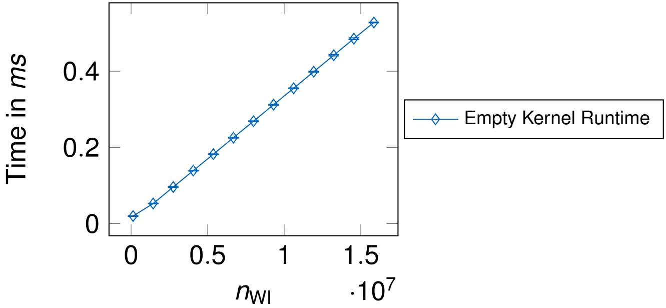
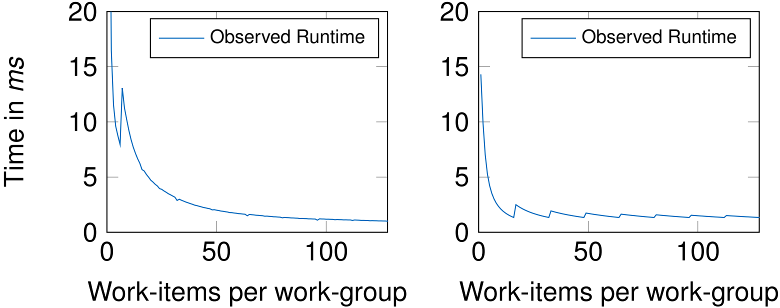


Figure: Execution times for empty kernels.

- $t_{\text{Base}}(n_{\text{WI}}) = c_{\text{Base}}\ n_{\text{WI}} + c^{\text{fixed}}_{\text{Base}}$

Model

Workgroup Size



(a) NVidia GT-650M

(b) Intel HD Graphics 4000

Figure: Execution time for different work-group sizes. The kernel we used to evaluate this behavior performs one read from and write to the global memory, and one floating point division.

Model

Workgroup Size — Modelling the behavior

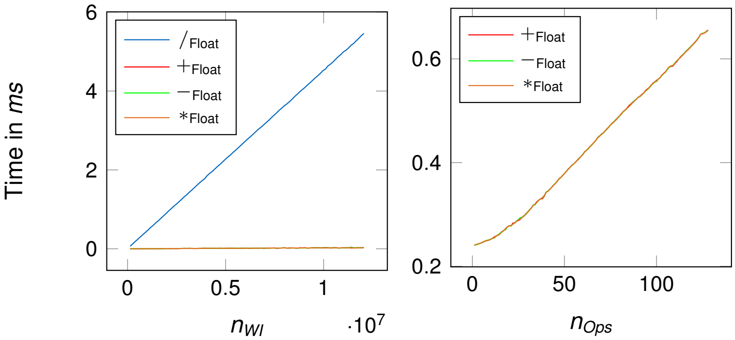
- Periodic spikes in execution time.
- Especially visible on the HD 4000.

Influence of Work-Group size

$$U(n_{\mathrm{WG}}, n_{\mathrm{XU}}) = \underbrace{\frac{\left\lfloor \frac{n_{\mathrm{WG}}}{n_{\mathrm{XU}}} \right\rfloor}{\left\lceil \frac{n_{\mathrm{WG}}}{n_{\mathrm{XU}}} \right\rceil}}_{A} + \underbrace{\frac{n_{\mathrm{WG}} \bmod n_{\mathrm{XU}}}{n_{\mathrm{XU}}} \frac{\left\lceil \frac{n_{\mathrm{WG}}}{n_{\mathrm{XU}}} \right\rceil - \left\lfloor \frac{n_{\mathrm{WG}}}{n_{\mathrm{XU}}} \right\rfloor}{\left\lceil \frac{n_{\mathrm{WG}}}{n_{\mathrm{XU}}} \right\rceil}}_{B}$$

Model

Basic Operations



(a) One operation per work-item

(b) Multiple Operations per work-item

Figure: Progression of the execution time for basic operations.

Model

Basic Operations

$$W_{\text{op}}^{\text{type}}(n_{\text{Ops}}) = \begin{cases} a\, n_{\text{Ops}}^{b} + c & : n_{\text{Ops}} \leq n_{\text{Ops}}^{\text{sat}} \\ a'\, n_{\text{Ops}} + c' & : n_{\text{Ops}} > n_{\text{Ops}}^{\text{sat}} \end{cases}$$

$$t_{\text{op}}^{\text{type}}(n_{\text{WI}}) = c_{\text{op}}^{\text{type}}\, n_{\text{WI}}$$

- a, a', b, c, c' are obtained by fitting $W_{\text{op}}^{\text{type}}(n_{\text{Ops}})$ to 4b
- $c_{\text{op}}^{\text{type}}$ is obtained by fitting $t_{\text{op}}^{\text{type}}(n_{\text{WI}})$ to 4a.

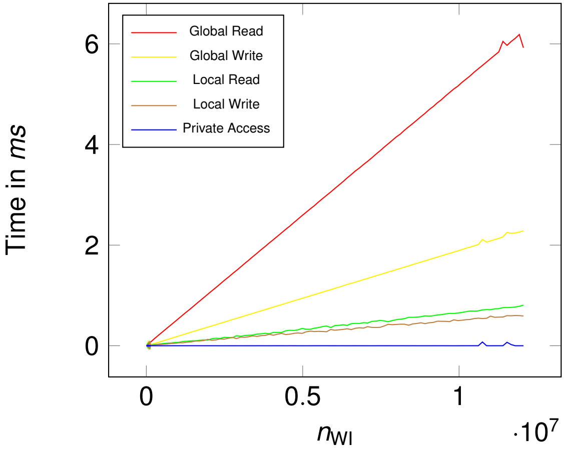
Model

Memory accesses

- In OpenCL, **3** different kinds of memory accesses are available
  - **private:** Used for local variables, parameters.
  - **local:** Shared between work-items within a work-group
  - **global:** Shared amongst all work-items
- Usually implemented using different kinds of memory.

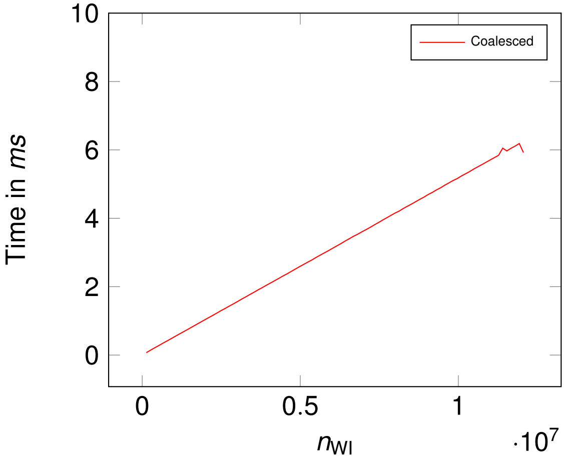
Model

Memory accesses



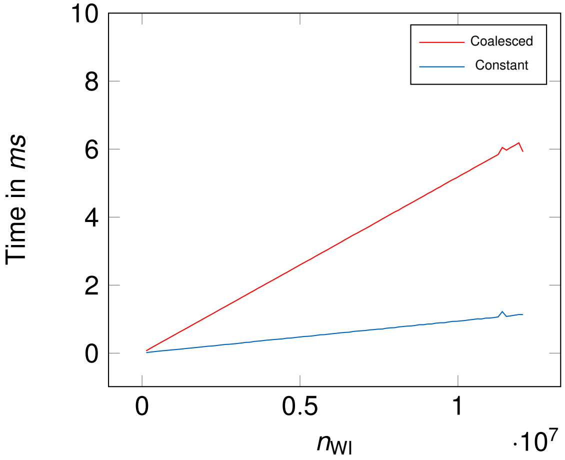
Model

Memory accesses — Coalesced Accesses



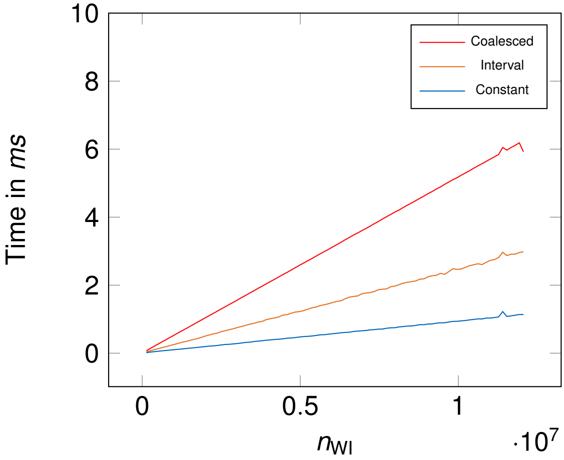
Model

Memory accesses — Constant Accesses



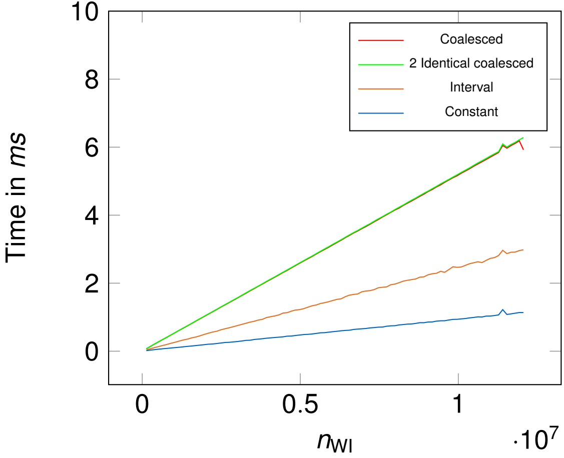
Model

Memory accesses — Interval Accesses



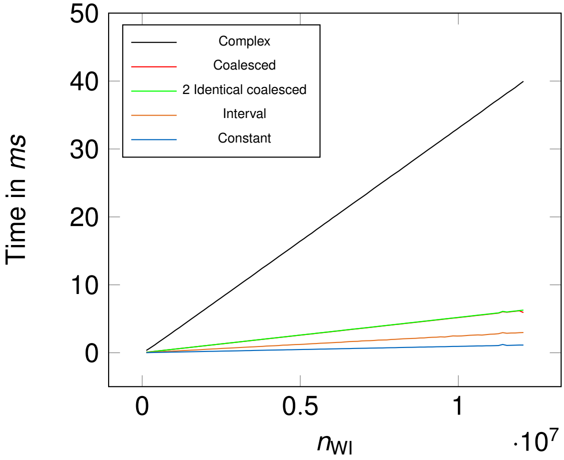
Model

Memory accesses — Two Identical Accesses



Model

Memory accesses — Complex Accesses



Evaluation

Qualitative Evaluation

- Static prediction of the execution time given the following data:
  - Kernel Source Code
  - Data about GPU characteristics
  - Number of work-items n_{WI}

Evaluation

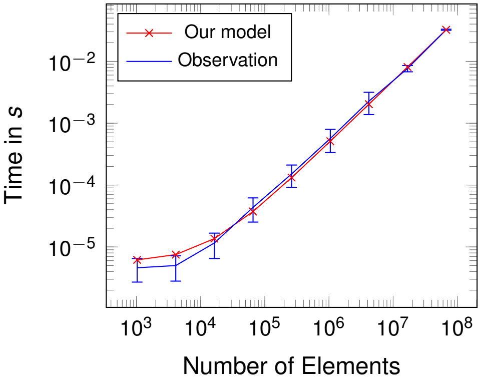
Qualitative Evaluation

- Static prediction of the execution time given the following data:
  - Kernel Source Code
  - Data about GPU characteristics
  - Number of work-items n_{WI}

Cost Type	# in Kernel	Time in μs
$-_{float}$	1	74.16
$*_{float}$	1	74.54
$+_{int}$	1	55.13
$*_{int}$	7	81.04
$/_{int}$	4	1506
private access	1	0.0
interval global read access	1	770.9
continuous global read access	1	2335
base cost	1	3191
work-group size	1024	-
final prediction		**8089**

Evaluation

Qualitative Evaluation



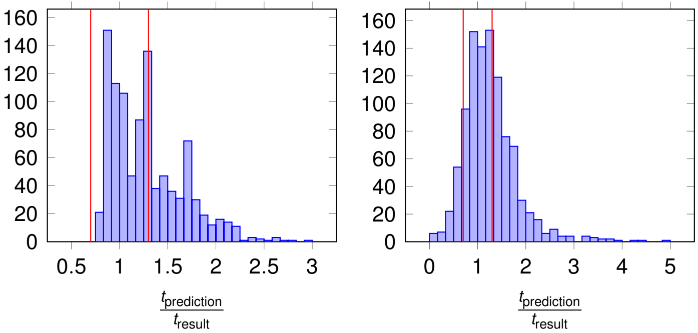
Evaluation

Quantitative Evaluation

- Quantitative evaluation through generated OpenCL Kernels
- 2 Sets of kernels, **Unrestricted** and **“Realistic”**
- Unrestricted Set
  - Little restrictions on complexity
  - Complex memory access patterns possible
  - `((xxx[((y + x) + 454) & 0x7F] / (matrix[x][y] * x)) - (matrix[x][y] + (matrix[x][y] + ((matrix[(44190 * (20 + x)) % HEIGHT][1094 % WIDTH] - xxx[71632 & 0x7F]) - ((162.82883f * (x - y)) + (785.19073f / (((((matrix[x][y] - matrix[x][y]) - xxx[(y * x) & 0x7F]) + 77.578835f) + matrix[x][y]) + 550.7608f))))))) + xxx[x & 0x7F]`
- Realistic Set
  - Complexity restricted, limited number of nodes in syntax tree
  - No overly complex memory access patterns
  - `((x / (xxx[x & 0x7F] / (matrix[1 % HEIGHT][361 % WIDTH] * matrix[x][y]))) * xxx[y & 0x7F]) + 747.18744f`

Evaluation

Quantitative Evaluation — GT-650M

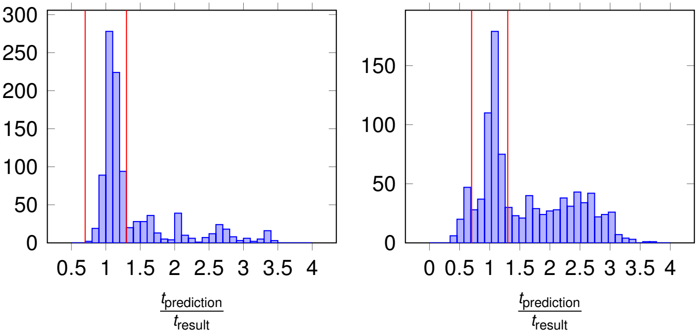


(a) Realistic Set **(b)** Unrestricted Set

- $0.7 < \frac{t_{prediction}}{t_{result}} < 1.3$ for 63% of all samples for the restricted set.
- $0.7 < \frac{t_{prediction}}{t_{result}} < 1.3$ for 50% of all samples for the unrestricted set.

Evaluation

Quantitative Evaluation — Quadro K4000

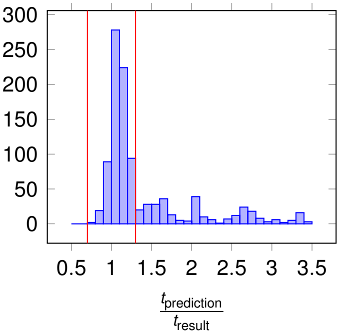


(c) Realistic Set **(d)** Unrestricted Set

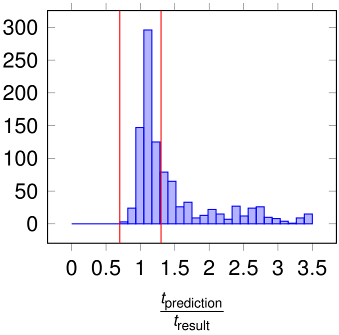
- $0.7 < \frac{t_{\text{prediction}}}{t_{\text{result}}} < 1.3$ for 71% of all samples for the restricted set.
- $0.7 < \frac{t_{\text{prediction}}}{t_{\text{result}}} < 1.3$ for 43% of all samples for the unrestricted set.

Evaluation

Quantitative Evaluation — Comparison



(e) Cache-Aware Model



(f) Simple Model

- $0.7 < \frac{t_{\text{prediction}}}{t_{\text{result}}} < 1.3$ for 71% of all samples for out model.
- $0.7 < \frac{t_{\text{prediction}}}{t_{\text{result}}} < 1.3$ for 61% of all samples for the simpler model.

Further Work

- Improve predictions, expand onto more architectures
- Support more language constructs, e.g. `if` or `for`
- Support intrinsic operations, e.g. `sin()`, `sqrt()`

Thank you for your attention

Acknowledgements This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre "Invasive Computing" (SFB/TR 89).