Towards Performance-Portable, Scalable, and Convenient Linear Algebra

Philippe Tillet[1,3], Karl Rupp[2], Siegfried Selberherr[1], Chin-Teng Lin[3]

[1]Institut für Mikroelektronik, TU Wien, Austria
[2]MCS Division, Argonne National Laboratory, USA
[3]National Chiao Tung University, Taiwan

HotPar'13, June 24th, 2013

GPUs: Library Aspects

Challenge: The architectural barrier

Challenge: The memory barrier

Challenge: The usability barrier

The architectural barrier

Tackling the portability issue

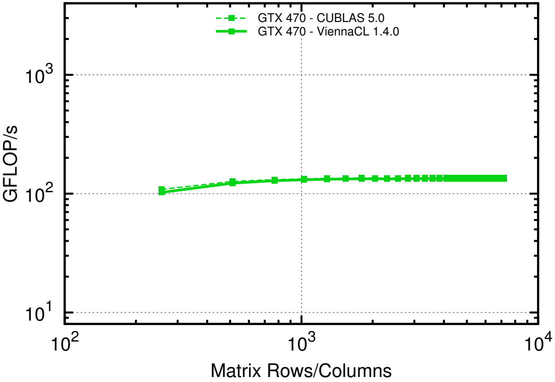
OpenCL - A unified programming model for parallel devices

- Includes:
  - a low-level API
  - a low-level programming language
- Compiled Just-In-Time

Write once, run everywhere

- Currently available for CPUs, GPUs, MICs...
- Soon available on FPGAs, DSPs...

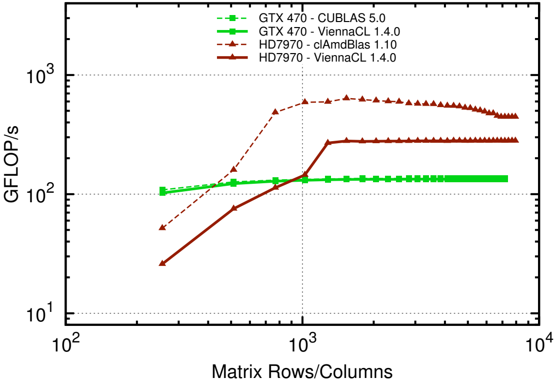
The good old matrix-matrix multiplication problem



We obtain same performance as CuBLAS 5.0

What if we use this kernel on AMD Hardware?

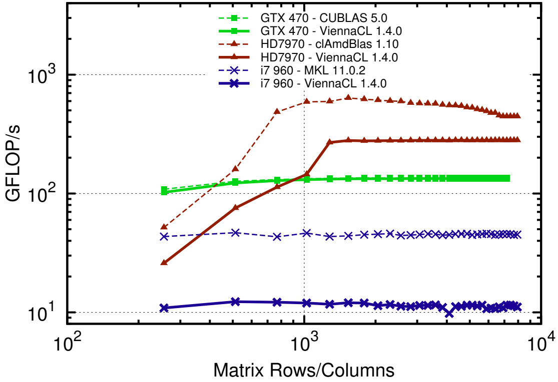
The good old matrix-matrix multiplication problem



We get 60% of the clAmdBlas performance!

What if we use this kernel on a CPU?

The good old matrix-matrix multiplication problem

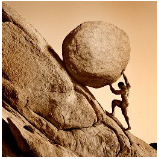


We get 20% of the MKL performance!

Code portability does not imply performance portability

The manual approach

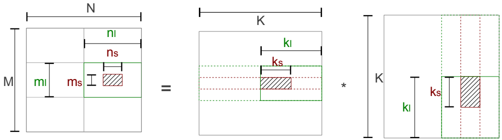
- Learn about the architecture
- Write, debug, profile, optimize your code
- Repeat the process whenever a new architecture is released



The automatic approach

- Build a compute kernel generator
- Build parameterized compute kernels for your algorithms
- Run an automatic tuning procedure

Potentially huge parameter space



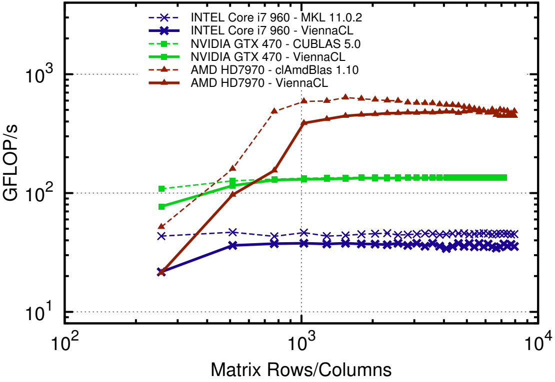
$30\text{k} \times 4$ layouts $\times\ 4$ transpositions $= 480\text{k}$ kernels

Big area of future work

The parameter space is still architecture-dependent

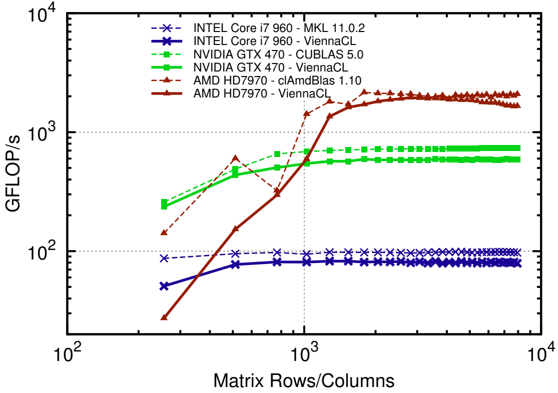
The brute-force approach executes in $10 \times 4 \times 4$ hours...

The good old matrix-matrix multiplication problem - revisited



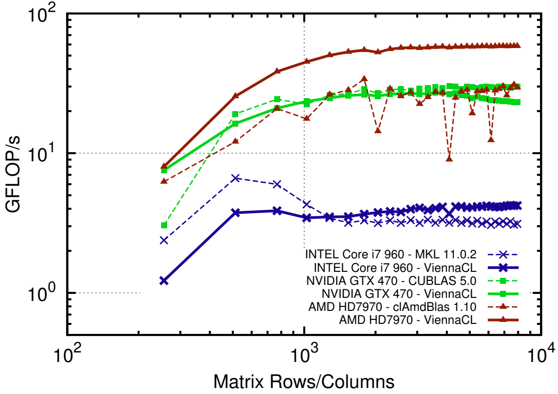
What about single precision?

The good old matrix-matrix multiplication problem - revisited



What about less compute-intensive tasks?

The Matrix-Vector multiplication problem



OpenCL code *can* be performance-portable

How to run

BLAS Level 1 operations

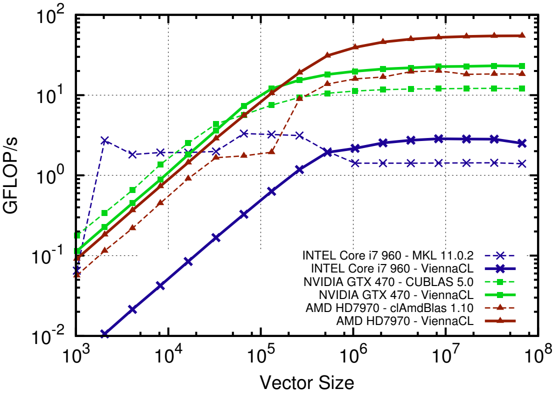
```
using namespace viennacl;
vector<double> x, y;
scalar<double> beta;
/* Fill x, y here */
custom_operation op;
op.add(beta = inner_prod(x, 2*x + y));
op.execute();
```

Multi-Matrix multiplication

```
using namespace viennacl;
add_all_available_devices(CL_DEVICE_TYPE_GPU);
multi_matrix<float> C, A, B;
/* Fill A, B here */
C = prod(A + B, A - B);
finish();
```

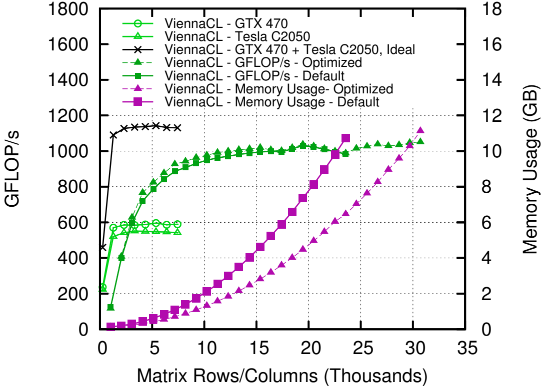
What you get

BLAS Level 1 operations



What you get

Distributed SGEMM



Summary

Auto-Tuning Framework

- Adapts the underlying hardware
- The whole BLAS Standard is supported
- At least 75% of vendor-tuned libraries performance
- Supports multiple GPUs

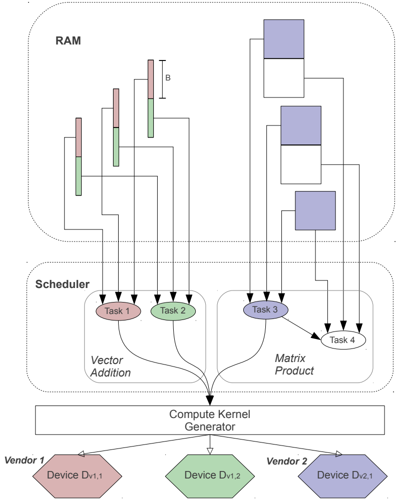
Future Work

- Improve the auto-tuning procedure
- Fully transparent use

Open-Source

- Will be released in ViennaCL 1.5.0
- `http://viennacl.sourceforge.net`

Global Structure



Generator Scheme