One OpenCL to Rule Them All? *

Romain Dolbeau (CAPS),
François Bodin (Irisa),
Guillaume Colin de Verdière (CEA)
7 September 2013

*This work has been partially funded by the H4H-PerfCloud ITEA2 project http://www.h4h-itea2.org

UMR IRISA cea CAPS

Introduction

- OpenCL is available on a large set of computing platforms
  - Parallel model fits GPU and CPU constraints
  - Aims at avoiding machine specific development
- Code development performance-wise issues
  - Assessing portability of a code
  - Deciding development strategy and technology
    - e.g. auto-tuning, self-adapting code

Target Platforms for this Study

- AMD7970
  - 947 GFlops DP peak
  - 264 GB/s theoretical memory bandwidth (TMB)
  - 32 “compute units”, each with 4 “SIMDs”, having16 scalar FPUs
- Nvidia K20C
  - 1170 GFlops DP peak
  - 208 GB/s TMB
  - 13 “SMX” for a total of 2496 “CUDA cores”
- Intel Xeon Phi SE10P
  - 1070 GFlops DP peak
  - 352 GB/s TMB
  - 61 conventional superscalar in-order x86 64 cores, each capable of running 4 threads via HyperThreading

Same class of devices

OpenCL Portability

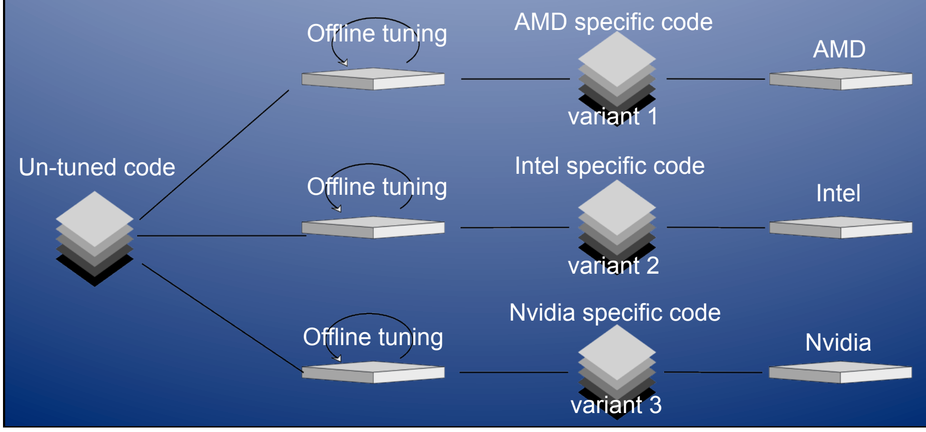
- Syntax
  - High thanks to standard definition
- Functional
  - Not 100% due to explicit resources uses
    - e.g. local memory, threads block size, …
- Performance
  - ???

Which Development Strategy?

1. Optimize for each target independently
2. Search of a tradeoff performing well on all targets
3. Generate codes that can adapt to the target / runtime context

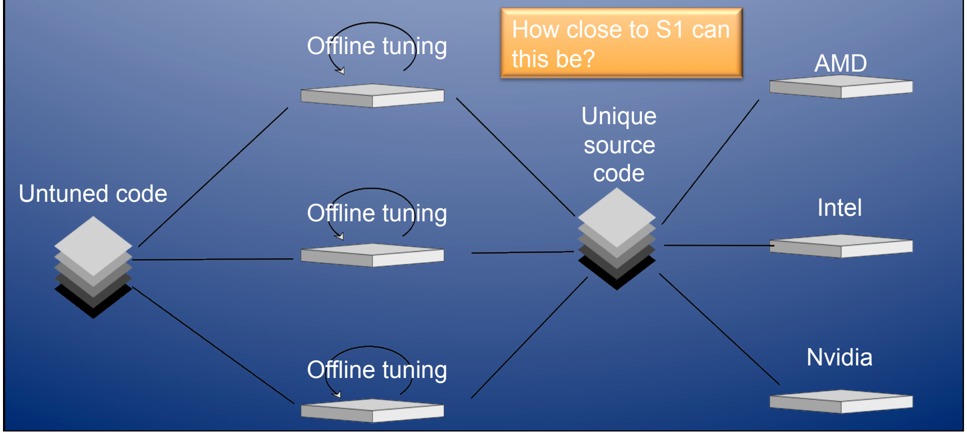
Development Strategy 1

- Produce one code for each target
  - Pros: target fitting, highest performance, predictability
  - Cons: Maintenance cost, application deployment



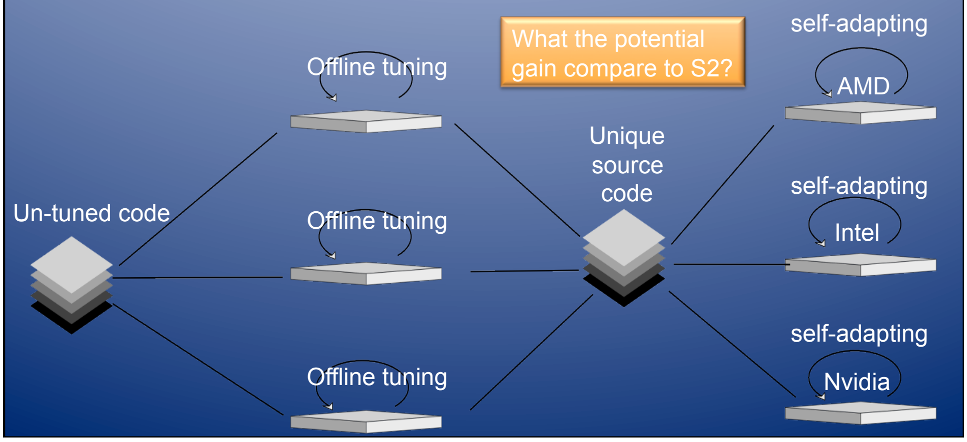
Development Strategy 2

- Offline tuning only
  - Pros: One code to deploy, maintenance cost, predictability
  - Cons: Performance averaging



Development Strategy 3

- Offline and online tuning strategy
  - Pros: Strategy 2 pros + performance, input set adaptability
  - Cons: Advanced technology, runtime overhead, debugging



Loss of Performance

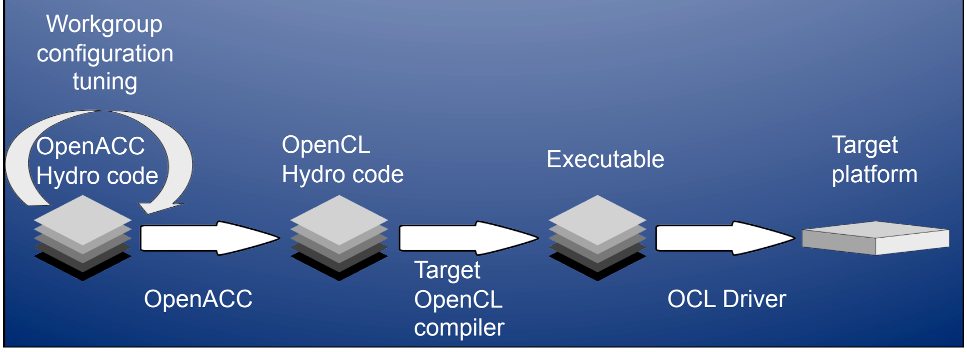
- Definition
  - Efficiency Loss = $\frac{ExecTime - BestExecTime}{BestExecTime}$
  - 0% indicates that the variant reaches the best performance
- Pro
  - Measures efficiency of an approach compare to the best known code for a given target
- Con
  - Finding the best execution time is complicated and expensive

Experimental Application

- Hydro
  - Mini-application built from the RAMSES used to study large-scale structures and galaxy formation
  - Classical algorithms found in applications codes for Tier-0 systems
- Main properties for this study
  - Extensively studied on many targets (i.e. we know the best performance)
  - *Non trivial with 22 OCL kernels (1.1)*
  - Not communication bound
  - Code mainly sensitive to work-groups (threads grid) configuration

Code Generation Process

- Using CAPS OpenACC compiler
  - Simplifies tuning → Workgroup configurations
  - Hide implementation details for each targets
    - Simpler scan of the optimization space



Code Generation Process Assessment

- We compared the performance of
  - the generated OpenCL codes
  - to the native OpenCL version that was previously developed for Hydro
- The OpenCL generated code is
  - As fast as the hand-written version on the Phi
  - Slightly faster on the K20C
  - CUDA code instead of OpenCL for the K20C does not improve performance on this code
- Two work-groups settings
  - Same work-group configuration for all kernels
  - Kernel specific work-group sizes

Kernel Specific Work-group Sizes

- AMD7970
  - did not show much improvement (1.53%)
- SE10P
  - did not show much improvement (1.41%)
- K20C
  - showed an improvement of 8.39%
  - when optimizing for the K20C (or the SE10P) the code will no longer run on the AMD7970 as the work-group size used are too large

Efficiency Loss Results

WG	AMD	Nvidia	Intel	Avg.	Max
16x4	*6.5%*	*12%*	*23.5%*	*14%*	*23.5%*
16x8	*5%*	*5.5%*	*34.5%*	*15%*	*34.5%*
16x16	*7.5%*	*2.5%*	*38.5%*	*16%*	*38.5%*
32x2	*4.5%*	*22%*	*8%*	*11.5%*	*22%*
32x4	*3%*	*12.5%*	*11%*	*9%*	***12.5%***
32x8	***0%***	*6%*	*18%*	*8%*	*18%*
64x1	*9.5%*	*34%*	*3.5%*	*15.5%*	*33.5%*
64x2	*9%*	*20.5%*	*4.5%*	*11.5%*	*20.5%*
64x4	*4.5%*	*13%*	*6%*	*8%*	*13%*
128x1	*13%*	*27.5%*	*1%*	*13.5%*	*27.5%*
128x2	*7.5%*	*20.5%*	*2%*	*10%*	*20.5%*
256x1	*9.5%*	*28.5%*	***0%***	*12.5%*	*28.5%*
16x32	***N/A***	***0%***	*43.5%*	*21.5%*	***43.5%***

Data Highlights

	AMD	Nvidia
256x1	9.5%	28.5%
16x32	***N/A***	***0%***

code non functional on AMD

Intel

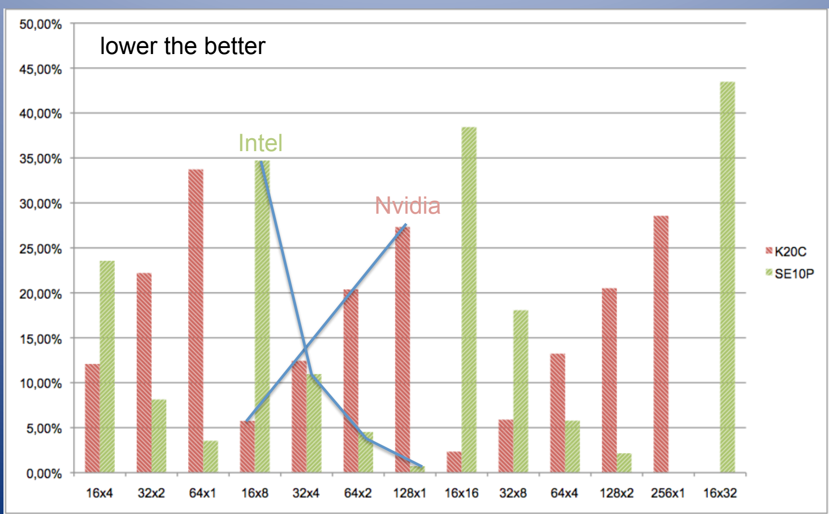
Best configuration

	AMD	Nvidia	Intel		
32x4	*3%*	*12.5%*	*11%*	*9%*	***12.5%***

Best for Nvidia worst for Intel

	AMD	Nvidia	Intel		
16x32	***N/A***	***0%***	*43.5%*	*21.5%*	***43.5%***
32x8	***0%***	*6%*	*18%*	*8%*	*18%*
256x1	*9.5%*	*28.5%*	***0%***	*12.5%*	*28.5%*

Performance correlation between Nvidia and Intel Devices



Conclusion

- OpenCL a very interesting alternative for programming accelerator
- Code tuning remains an issue
- Performance tradeoff has to be explicitly researched
  - Performance portability can be achieved to an extent
  - No target can be used as a performance Oracle
- Auto-tuning techniques necessary to explore the optimization space
- Runtime techniques need to be very efficient to be worthwhile