Demonstrating Performance Portability of a Custom OpenCL™ Data Mining Application to the Intel® Xeon Phi™ Coprocessor

Alexander Heinecke (TUM),

Dirk Pflüger (Universität Stuttgart),

Dmitry Budnikov, Michael Klemm, Sergey Lyalin,
Arik Narkis, Maxim Shevtsov, Ayal Zaks (Intel)

IWOCL
INTERNATIONAL WORKSHOP ON OPENCL

OFFICIAL DATE & VENUE:
May 13–14, 2013
Georgia Institute of Technology

Legal Disclaimer & Optimization Notice

Introduction

- We use OpenCL™ as an API of choice, which allows keeping OpenCL codebase the same for the different devices
- Out-of-box OpenCL performance of Data Mining Application is reasonable for both devices under consideration: Intel® Xeon Phi™ coprocessor and NVIDIA* Tesla* K20X
- We applied algorithmic improvement to the core data mining routine that significantly boosts performance on both devices
- By applying relatively simple device-specific optimizations Intel Xeon Phi coprocessor provides best measured performance

Optimization Notice

Agenda

- Intel® Many Integrated Core (Intel® MIC) Architecture and Intel® Xeon Phi™ Coprocessor
- Data mining + sparse grids
- Optimizations
  - ✓ Intel Xeon Phi coprocessor specific
  - ✓ General
- Performance data
- Conclusions

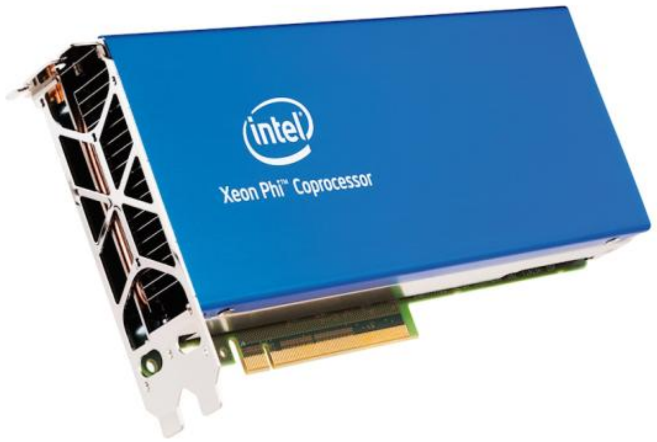
Intel® Many Integrated Core (Intel® MIC) Architecture

- Targeted at highly parallel applications
  - ✓ Physics, Chemistry, Biology, Financial Services
- General Purpose Programming Environment
  - ✓ Runs Linux* (full service, open source OS)
  - ✓ Runs applications written in Fortran, C, C++, OpenMP*, OpenCL™ …
  - ✓ Runs the x86 ISA + new SIMD extension
  - ✓ Supports x86 coherent memory model, IEEE 754
  - ✓ x86 collateral (libraries, compilers, Intel® VTune™ Amplifier XE, debuggers, etc.)

The Intel® Xeon Phi™ Coprocessor SE10P

http://ark.intel.com/search?q=PHI

- The first product based on Intel® Many Integrated Core architecture
- 61 cores with 4 HW threads, each at 1.1 GHz
- 32 KB L1$ - data and instruction, 512 KB L2$, per core
- L2$ are kept coherent through a fast on-chip interconnect ring
- 512 bit wide vector instructions, throughput of one per cycle per core
- ~1 TFLOPS Peak DP, ~2 TFLOPS Peak SP
- 8 GB GDDR5, 512 bit interface, 320 GB/s
- PCIe 2.0 Host-Interface



Optimization Notice

Data Mining

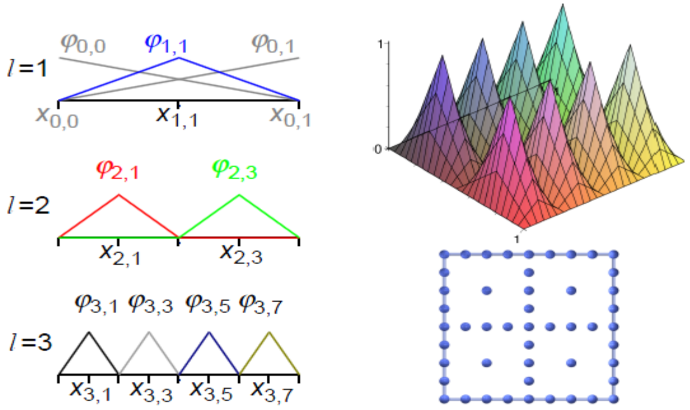
- Classification and regression tasks
- Generalize known data and predict properties for a new (unknown) data
- Reconstruct underlying function having M data points $\vec{x}_m$ and y_m target values

$$f_N(\vec{x}) = \sum_{j=1}^{N} \alpha_j \varphi_j(\vec{x}) \,.$$

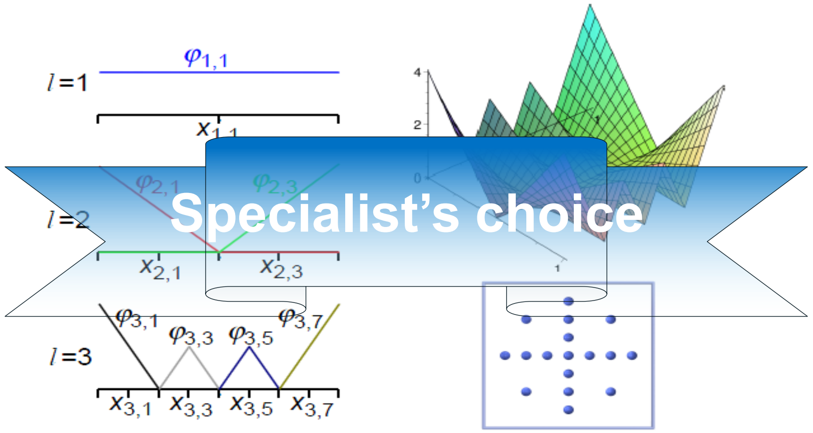
- Solve regularized least squares problem to find coefficients vector $\vec{\alpha}$
- Classical finite element scheme – basis functions centered at grid points

Sparse Grids

- Curse of Dimensionality: $O(N^d)$ grid points
- Therefore: Sparse Grids
  - ✓ Reduce cost to $O(N \log(N)^{d-1})$
  - ✓ Similar accuracy, if problem sufficiently smooth
- Hierarchical basis functions in 1D – linear and modified linear



$\varphi_{0,0}$ $\varphi_{1,1}$ $\varphi_{0,1}$
$l=1$
$x_{0,0}$ $x_{1,1}$ $x_{0,1}$
$\varphi_{2,1}$ $\varphi_{2,3}$
$l=2$
$x_{2,1}$ $x_{2,3}$
$\varphi_{3,1}$ $\varphi_{3,3}$ $\varphi_{3,5}$ $\varphi_{3,7}$
$l=3$
$x_{3,1}$ $x_{3,3}$ $x_{3,5}$ $x_{3,7}$

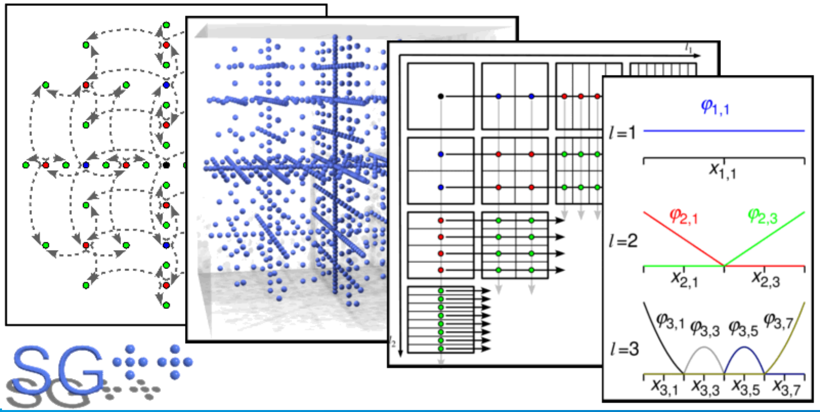


$\varphi_{1,1}$
$l=1$
$x_{1,1}$
$\varphi_{2,1}$ $\varphi_{2,3}$
$l=2$
$x_{2,1}$ $x_{2,3}$
$\varphi_{3,1}$ $\varphi_{3,3}$ $\varphi_{3,5}$ $\varphi_{3,7}$
$l=3$
$x_{3,1}$ $x_{3,3}$ $x_{3,5}$ $x_{3,7}$
Specialist's choice

- 4-fold if{}else{} control flow in the kernel: determines if the processed point is on the level 1, on the left or right boundary or a grid point on the inner grid

SG++ Sparse Grids Implementation

- Open source http://www5.in.tum.de/SGpp/releases/index.html
- SG++ based application originally optimized for CPU
  - ✓ Implemented using compiler vector intrinsics
  - ✓ OpenMP* parallelization
- The first OpenCL™ implementation was optimized for GPUs
- Now also optimized for both Intel® Xeon Phi™ coprocessor and GPUs
  - ✓ Difference mostly in runtime parameters like local size
  - ✓ More on the next foils



$\varphi_{1,1}$
$l=1$
$x_{1,1}$
$\varphi_{2,1}$
$\varphi_{2,3}$
$l=2$
$x_{2,1}$
$x_{2,3}$
$\varphi_{3,1}$ $\varphi_{3,3}$ $\varphi_{3,5}$ $\varphi_{3,7}$
$l=3$
$x_{3,1}$ $x_{3,3}$ $x_{3,5}$ $x_{3,7}$

Optimization Notice

Implementation Details

```
Grid_update_loop( ~6 … 8 iterations ) //Adjust grid configuration to better fit training data
{
        Conjugate_gradient_loop(~200 … 250 iterations)    //Solver loop – adjust basis functions coefficients (alpha)
        {
                OCL_NDRange_loop_transposed_mult(1500…20000 iter.) //over grid
                {
                        inner_loop_inside_OCL_kernel (~260K iter.) //over data
                        {
                                4-fold if{}else{} control flow
                                // dimensions loop unrolled by JIT
                        }
                }
                Grid_tail_processing(); //Tail not divisible on WGS – tail processed on the host
                OCL_NDRange_loop_mult(~260K iter.) //over data
                {
                        inner_loop_inside_OCL_kernel (1500…20000 iter.) //over grid
                        {
                                4-fold if{}else{} control flow
                                // dimensions loop unrolled by JIT

                        }
                }
        }
        Rebuild_grid();
}
```

OpenCL device code time is >90% of overall

***Host code**

*****OpenCL device code***

Optimization Notice

The First Intel® Xeon Phi™ Coprocessor Optimization: Local Memory/Barriers Avoidance

- Original implementation was targeted for GPU
- Intel® Xeon Phi™ coprocessor doesn't distinguish between local and global memory
- Barriers and local memory adds additional overhead

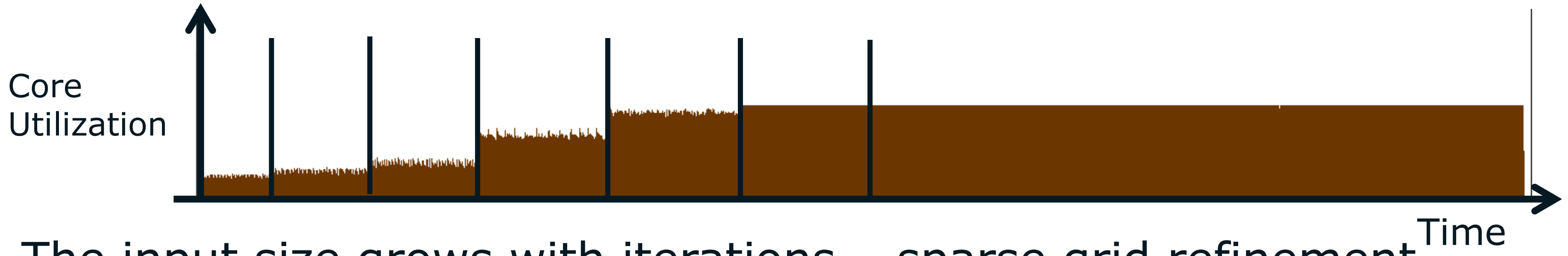
```
int GIdx = get_global_id(0);
int LIdx = get_local_id(0);
__local double locData[64];
__local double locSource[64 ];

for(int i = 0; i < sourceSize; i+= 64 )
{
   locData[LIdx] = ptrData[i+LIdx];
   locSource[LIdx] = ptrSource[i+LIdx];
   barrier(CLK_LOCAL_MEM_FENCE);
   for(int k = 0; k < 64 ; k++)
   {
       myResult += DoWork(
                         locSource[k],
                         locData[k],
                         ptrLevel[GIdx]
                         );
   }
   barrier(CLK_LOCAL_MEM_FENCE);
}
ptrResult[globalIdx] = myResult;
```

```
int GIdx = get_global_id(0);
int LIdx = get_local_id(0);


for(int i = 0; i < sourceSize; i++)
{




    myResult += DoWork(
                         ptrSource[i],
                         ptrData[i],
                         ptrLevel[GIdx]
                         );


}
ptrResult[globalIdx] = myResult;
```

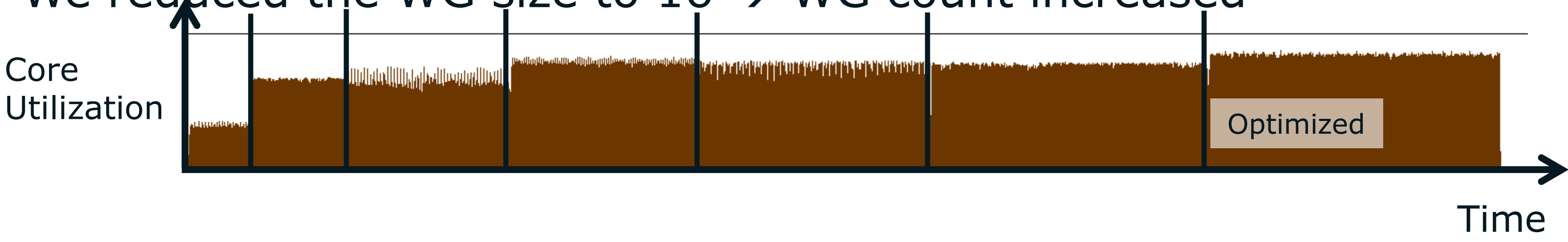
Optimization Notice

Now the Important Intel® Xeon Phi™ Optimization: Saturating HW Threads

- Iterative algorithm



- The input size grows with iterations – sparse grid refinement
- With original WG size of 64 (2 GPU warps), there are not enough WGs for Intel® Xeon Phi™ coprocessor
  - ✓ Especially in the first iterations (need to feed 240 HW threads)
  - ✓ Each WG is assigned to individual HW thread
- We reduced the WG size to 16 → WG count increased



[1] Intel® VTune™ Amplifier XE graphs: http://software.intel.com/en-us/intel-vtune-amplifier-xe

Workgroup Vectorization: Diverged Branches

- Implementation tips:
  - ✓ Parallelize WGs on the HW threads (prev. foil)
  - ✓ Vectorize WIs on the SIMD unit
  - ✓ Vectorize over NDRange innermost loop (dimension zero)

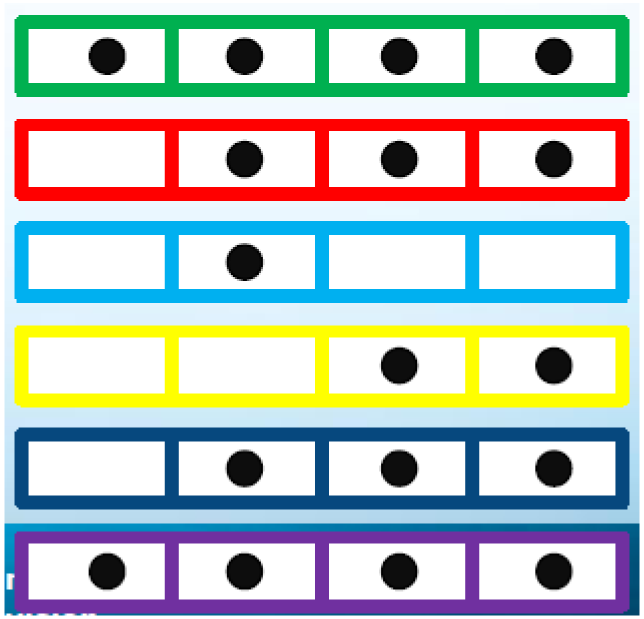
Uniform Branch	Diverged Branch
The compiler can prove that all the WIs within the vector take the same branch	global_id(0) dependent branch The compiler cannot prove that the branch is uniform

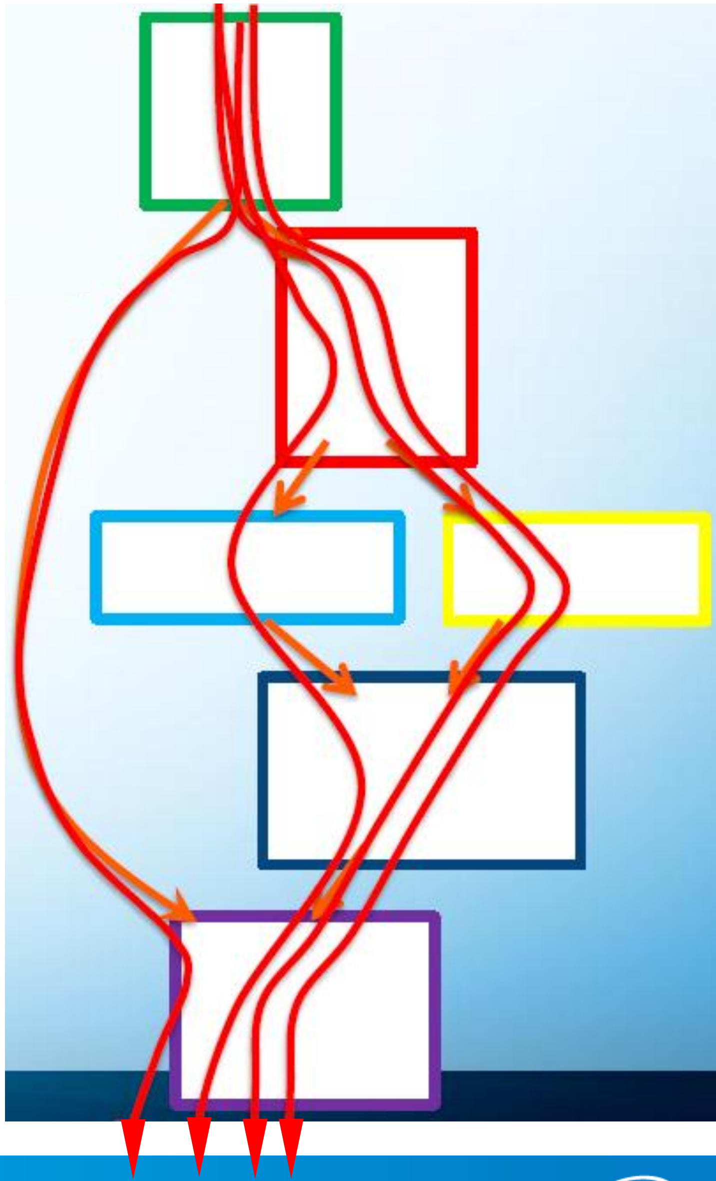
//isSimple is a kernel argument
int GID = get_global_id(0);
if (isSimple == 0)
*   res = buff[GID];*
	int GID = get_global_id(0);
	if (GID == 0)
	* res = -1;*

- One of the kernels contains branches dependent on global_id(0)
- Diverged branches impacts compiler vectorization effectiveness

Predicating (Flattening) Diverged Branches

- Predication flattens the control flow (CF) and executes both the „then“ and „else“
- Diverging CF reduces the utilization of vector instructions
- Predication adds masking-overhead





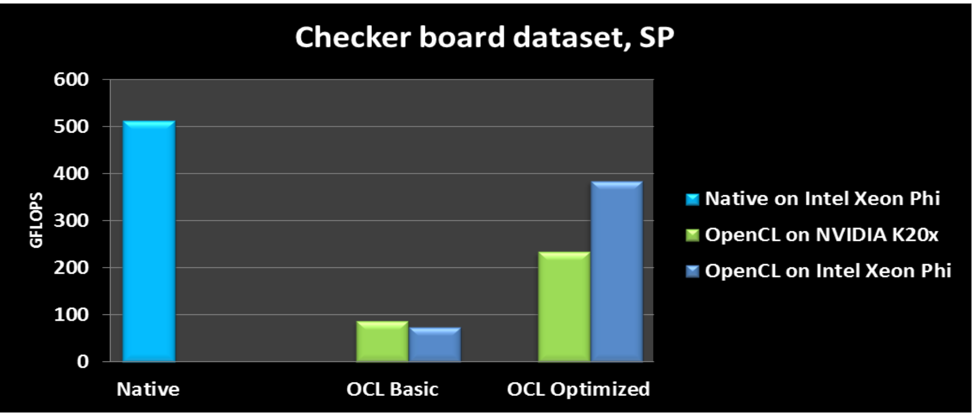
The General Optimization: Avoid Control Flow

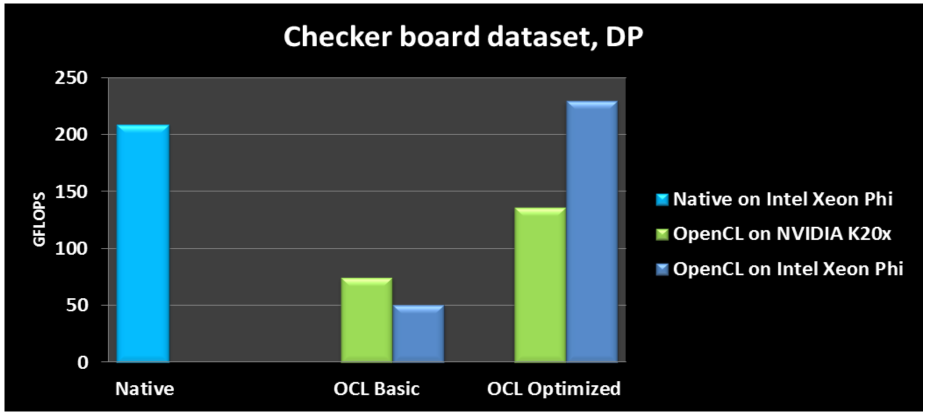
```
if (pow2_Level[]== 2.0)                    /* ORIGINAL CODE SNIPPET */
{
        curSupport *= 1.0;
}
else if (Index[] == 1.0)
{
        curSupport *= max(2.0 - ( (pow2_Level[]) * (data) ), 0.0) ;
}
else if (Index[] == (pow2_Level[] - 1.0) )
{
        curSupport *= max(( (pow2_Level[]) * (data) ) - (Index[]) + 1.0, 0.0);
}
else
{
        curSupport *= max(1.0 - fabs( ( (pow2_Level[]) * (data) ) - (Index[]) ), 0.0);
}
```

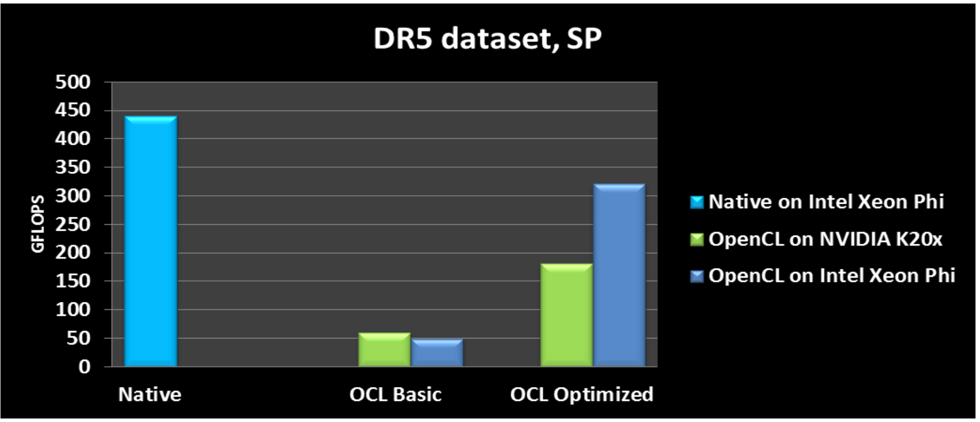
```
max(as_float(as_uint(Alpha[] * data + Beta[]) | Gamma[] ) + Delta[], 0.0);
```

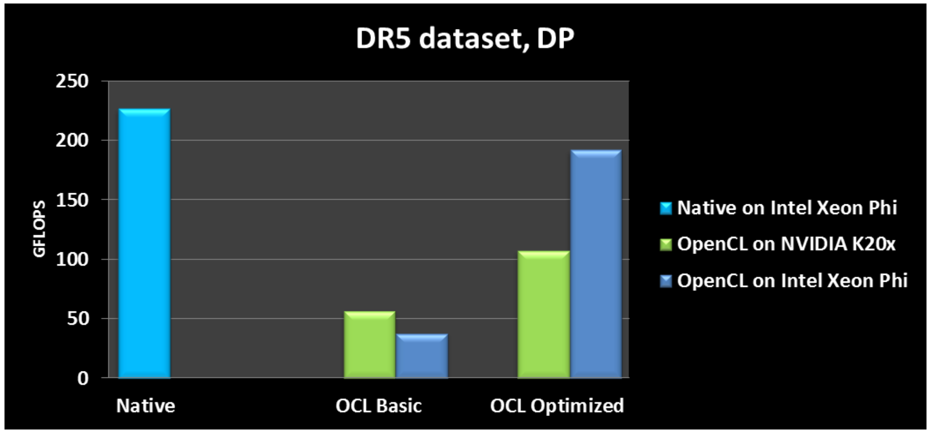
- Loop invariant hoisting optimization
- Alpha Beta Gamma Delta parameters describing grid topology can be precalculated on the host once for each grid refinement

OpenCL™ Performance of Intel® Xeon Phi™ Coprocessor vs. NVIDIA* Tesla* K20X









Results as presented in the article: "**Demonstrating Performance Portability of a Custom OpenCL Data Mining Application to the Intel® Xeon Phi™ Coprocessor" by Heinecke et al.**

System configuration used: Supermicro R X9DRG-HF baseboard with 2S Intel Xeon processor E5-2670 (128GB DDR3 with 1600MHz, Red Hat Enterprise Linux 6.3) and single Intel R C600 IOH, Intel Xeon Phi coprocessor with B0 ES2 silicon (GDDR with 5.5GT/sec, driver v2.1.4982, ash v2.1.05.0375, device OSv2.6.38.8-g32944d0), Intel R Composer XE 2013 U1, Intel R SDK for OpenCL Applications XE 2013 Beta v3.0.56860, and NVIDIA Tesla K20Xm (GDDR with 5.2GHz, driver 310.32, CUDA v5.0 with OpenCL support).

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.
For more information go to http://www.intel.com/performance

Optimization Notice

Conclusions

- We use OpenCL™ as an API of choice, which allows
  - ✓ Keeping OpenCL codebase the same for the different devices
  - ✓ Allows easy accommodation to the device-specific run time parameters (like optimal workgroup size) via simple host-side logic
- Out-of-box OpenCL performance of SG++ is reasonable for both Intel® Xeon Phi™ coprocessor and NVIDIA* Tesla* K20X devices
  - ✓ We applied algorithmic improvement to the core data mining routine that significantly boosts performance of both devices.
- Finally by applying relatively simple device-specific optimizations on Intel Xeon Phi coprocessor provides best measured performance
  - ✓ Also managed to match the speed of the full-blown Native code on Intel Xeon Phi coprocessor for DP case which is in primary use

This work demonstrates performance portability with OpenCL and the Intel Xeon Phi Coprocessor

Optimization Notice

Credits

Intel:

Anat Shemer

Yariv Aridor

Orna Etzion

Zvi Rackover

Ohad Shacham

Arnon Peleg

Mikhail Letavin

Igor Martynov

Optimization Notice

Resources

www.intel.com/software/opencl-xe

A. Heinecke and D. Pflüger.
Multi- and Many-Core Data Mining with Adaptive Sparse Grids. In Proc. Of the 8th ACM Intl. Conf. on Computing Frontiers,
pages 29:1 - 29:10, New York, USA, May 2011.

A. Heinecke, M. Klemm, D. Pflüger, A. Bode, and
H.-J. Bungartz.
Extending a Highly Parallel Data Mining Algorithm to the Intel(R) Many Integrated Core Architecture.
In The 4th Workshop on UnConventional High Performance Computing 2011, Bordeaux, France, August 2011.

Thank You!

Questions?

Optimization Notice

Legal Disclaimer & Optimization Notice