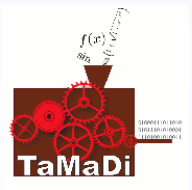
# Solving the Table Maker's Dilemma on multi-core CPUs and on the Xeon Phi

Christophe Avenel, Pierre Fortin, Mourad Gouicem, Samia Zaidi

Université Pierre et Marie Curie - LIP6

ANR TaMaDi meeting
October 07-08, 2013 - Lyon, France




# Outline

1. Table Maker's Dilemma & SIMD units
2. C programming with Intel compiler
3. OpenCL programming with Intel OpenCL SDK on multi-core CPUs
4. OpenCL programming with Intel OpenCL SDK on the Xeon Phi
5. Conclusion & future work

# Outline

- Table Maker's Dilemma solving with Lefèvre algorithm
- Two steps
  1. Approximation generation: massively parallel and regular control flow
  2. HR-case search: massively parallel but irregular control flow
     - performance penalty on GPU due to partial SIMD (Single Instruction, Multiple Data) execution (SIMD width = 32 on NVIDIA Fermi GPU)
     - new "regular" HR-case search
       $\rightarrow$ up to 3.5 times faster than Lefèvre HR-case search on GPU
     - Regular HR-case search >2.5 times faster on K20 w.r.t. C2070
       $\Rightarrow$ well-suited for new GPU architectures

| GPU | Year | CUDA cores | Peak perf. |
|---|---|---|---|
| C2070 | 2010 | 448 @1.15 GHz | 1030 SGflop/s - 515 DGflop/s |
| K20 | 2012 | 2496 @0.706 GHz | 3520 SGflop/s - 1170 DGflop/s |

- What about SIMD units on CPU?

| CPU | Year | CPU cores | SIMD | Peak perf. |
|---|---|---|---|---|
| X5650 | 2010 | 6 @2.67 GHz | SSE (128 bits) | 128.16 SGflop/s - 64.08 DGflop/s |
| E5-2660 | 2012 | 8 @2.20 GHz | AVX (256 bits) | 281.6 SGflop/s - 140.8 DGflop/s |

  - How to harness their computational power?
  - Is the new regular HR-case search more efficient on such SIMD units?

# SIMD units in multi-core CPUs

- Various SIMD instruction sets on multi-core CPUs
  - SSE (Intel, AMD), AltiVec (IBM) : 128 bits
  - AVX (Intel Sandy/Ivy Bridge) : 256 bits (but no integer operations)
  - AVX2 (Intel Haswell - 2013) : 256 bits with integer operations
  - Intel Xeon Phi (or MIC - Many Integrated Core) 5110P :
    - 60 x86 cores (@1.053 GHz)
    - 4-way SMT : 240 hardware threads
    - in-order execution
    - cache-coherent architecture (L2)
    - 512-bit SIMD units (with mask registers):
      $\rightarrow$ 16 floats or 32-bit integers
      $\rightarrow$ 8 doubles or 64-bit integers
    - peak performance: 2021.76 SGflop/s - 1010.88 DGflop/s
    - currently not stand-alone: requires CPU host + PCI bus
    - parallel programming: MPI, OpenMP, Intel TBB, Intel Cilk Plus, OpenCL...
- Increasing width of SIMD units $\Rightarrow$ increasing computation power
- HPC applications must harness these SIMD units in order to fully exploit the hardware computation power
  - 32-bit computation without SIMD on Xeon Phi: only $\frac{1}{16}$ of the Xeon Phi computation power

# Programming SIMD instructions

- ★ Manual programming with intrinsics
  - ▸ tedious
  - ▸ padding possibly required
  - ▸ must be re-written when moving to another SIMD instruction set (e.g. SSE → AVX → Xeon Phi)
- ★ Intel C compiler (w/o or w/ `#pragma` compiler directives)
  - ▸ the compiler has to first extract the available parallelism (w/o compiler directives)
  - ▸ limitations depending on the compiler[1]
- ★ SPMD (*Single Process/Program, Multiple Data*) programming with implicit vectorization: OpenCL, ISPC (*Intel SPMD Program Compiler*)...
  - ▸ SPMD programming: main source of parallelism given by the programmer
  - ▸ "Only" need to merge scalar control flows in SIMD instructions
  - ▸ No SIMD width written in the source code → SIMD width fixed at compile time (depending on the targeted hardware)

[1] See: *A Guide to Vectorization with Intel C++ Compilers*, M. Deilmann, K. Kuah, M. Corden, M. Sabahi, Intel ; *Requirements for Vectorizable Loops [with #pragma SIMD]*, M. Corden, Intel.

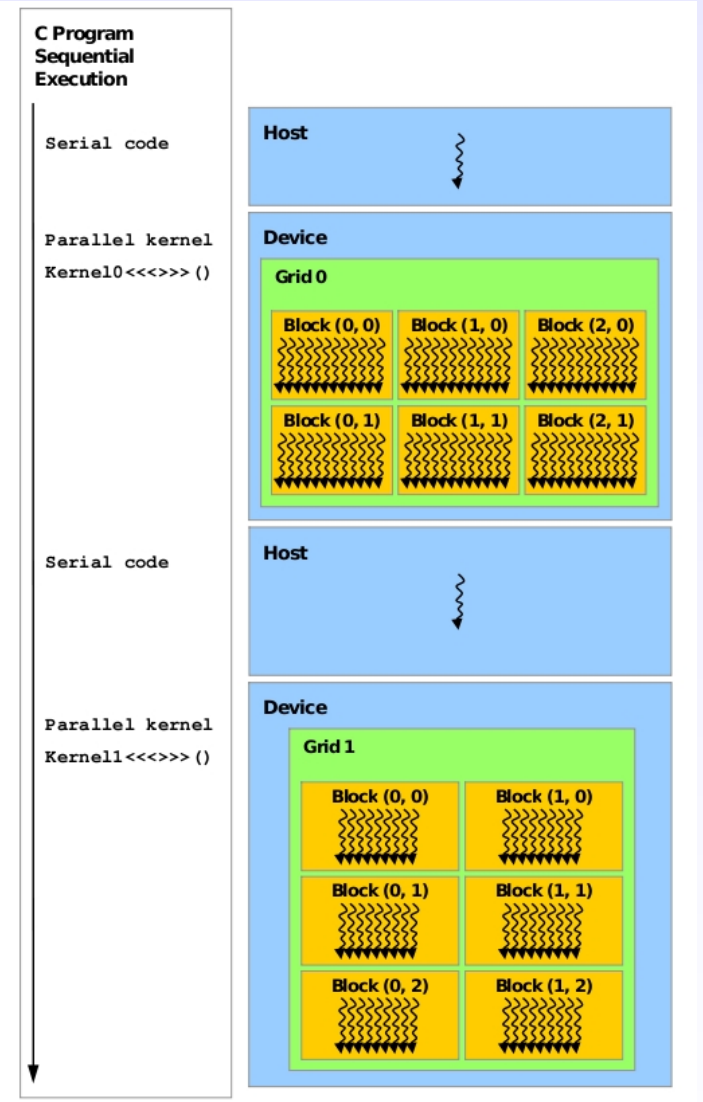
# OpenCL

## CUDA programming

- ★ *grid* of *blocks* of *threads*
- ★ Restricted to NVIDIA GPUs

## OpenCL programming

- ★ *NDRange* of *work-groups* of *work-items*
- ★ NVIDIA & AMD GPUs, Intel & AMD CPUs, Intel Xeon Phi, IBM (Cell)...
- ★ On multi-core CPUs :
  - ▸ 1 thread per core
  - ▸ each thread processes 1 work-group at a time (dynamic load balancing)
  - ▸ possible use of SIMD instructions to process multiple consecutive work-items at a time




(CUDA Programming Guide 2.3)

# OpenCL vectorization

- ★ Explicit vectorization possible
- ★ Implicit CPU vectorization module in Intel OpenCL SDK
  - ▸ handling control flow divergence in consecutive work-items[2]:
    - for SSE → requires explicit masking
    - easier with Xeon Phi SIMD instructions → use mask registers
  - ▸ heuristics used to determine whether the implicit vectorization is worthwhile (depending on the control flow divergence among work-items)
- ★ Performance tests performed with *Intel OpenCL SDK 1.5* and *Intel SDK for OpenCL Applications 2013*

⇒ Lefèvre & regular HR-case searches are challenging applications for the Intel OpenCL implicit vectorization module

[2] Remark: on GPU, control flow divergence mainly handled by the hardware.

# Outline

# C vectorization (1)

- Auto-vectorization for SSE with icc (Intel C/C++ Compiler) version 12.1
- Without & with `#pragma SIMD`

## Approximation generation:

- Not yet studied
- Problems with GMP? (internal use of SSE instructions for multi-precision)

## Regular HR-case search:

- Goal: vectorize multiple iterations of the external for loop on $2^{25}$ intervals (all iterations are independent)
- Remark: problems for (64-bit) integer division and modulo on SIMD instructions... ($\rightarrow$ emulation or “subtractive division” algorithm)

# C vectorization (2)

## Regular HR-case search (cont.):

- ★ Phase #3 only (exhaustive search):
  - 2 inner for loops → merged in 1 for loop
  - `icc` only:
    - Dependence among iterations in the inner for loop (update of approximation coefficients)
    - **Vectorization failure** for inner loop ⇒ **vectorization failure** for outer loop
    - Use of `#pragma ivdep` (ignore vector dependencies) does not help...
  - `icc` with `#pragma simd`: **successful vectorization**!
    But prohibitive CPU times with exhaustive search only...
    ⇒ **and requires further correctness checking...**

- ★ Phases #1-2-3:
  - `icc` only:
    - Function call → macro, `goto+return` → boolean tests in loops
    - **Vectorization failure** for both Lefèvre & Regular HR-case searches
  - `icc` with `#pragma simd`:
    - **Vectorization failure** (even with subtractive division to avoid integer divisions and modulos)
    - Possible reasons: while loops and/or too many levels of nested loops (and integer divisions and modulos)

# Outline

# OpenCL vectorization of approximation generation

- ★ 3 kernels
- ★ Regular control flow: only `for` loops, no while loop
- ★ No division
- ★ Internal function → macro
- ★ Fixed multi-precision additions hand-coded in OpenCL with 32-bit integers

⇒ **Successful vectorization** of the 3 kernels for SSE4.2, AVX, AVX2

| Version | Time(s) | Speedup / Sequential C | Vectorization speedup |
|---|---|---|---|
| Sequential C (with GMP) | 5.54 | - | - |
| OpenCL 12 cores<br>no SMT<br>no vectorization | 0.53 | 10.45 | - |
| OpenCL 12 cores<br>SMT<br>no vectorization | 0.48 | 11.54 | - |
| OpenCL 12 cores<br>SMT<br>vectorization<br>(4 work-items/work-group) | 0.19 | 29.16 | 2.53 |

Approximation generation for double precision $\exp(x)$ on $[1; 1 + 2^{-13}]$,
using 12 CPU cores with 2-way SMT (2 Intel X5650), and Intel OpenCL SDK 1.5.

- ★ C2070 GPU: 1.94 times faster than 2 X5650 → 3.88 times faster than 1 X5650 CPU

# OpenCL vectorization for HR-case search

## OpenCL kernel for phase #3 (exhaustive search)

- ⋆ most regular phase
- ⋆ but: while loop (for loop possible), conditional statements and atomic operations

- ⋆ **Vectorization failure** for SSE4.2, AVX, AVX2
- ⋆ But: **successful vectorization** when replacing 64-bit integers by 32-bit integers for SSE4.2, AVX, AVX2 (but wrong computation...)

## OpenCL kernels for phases #1 & #2

- ⋆ more control flow divergence than in phase #3
- ⋆ with: multiple nested while and for loops, conditional statements and atomic operations

- ⋆ Modifications tried:
  - 64-bit $\rightarrow$ 32-bit integers
  - `return` $\rightarrow$ boolean tests in loops
  - prefer subtractive division to avoid integer divisions and modulos
  - only 1 interval per work-item to simplify the memory access pattern

$\Rightarrow$ **Vectorization failure** in all cases with SDK 2013 for SSE/AVX/AVX2

# Outline

# On the Xeon Phi

- ⋆ **Successful vectorization of all kernels!**

# On the Xeon Phi

- **Successful vectorization of all kernels!**
- Problem with first Xeon Phi card

# On the Xeon Phi

- ⋆ **Successful vectorization of all kernels!**
- ⋆ Problem with first Xeon Phi card
- ⋆ But low performance gain for the HR-case search
  $\Rightarrow$ due to 64-bit integers for Xeon Phi vector instructions

# On the Xeon Phi

- ⋆ **Successful vectorization of all kernels!**
- ⋆ Problem with first Xeon Phi card
- ⋆ But low performance gain for the HR-case search
  $\Rightarrow$ due to 64-bit integers for Xeon Phi vector instructions
- ⋆ Approximation generation (32-bit integers): currently limited performance gain but still under investigation...

# Outline

# Conclusion & future work

- OpenCL implementation of Lefèvre & regular HR-case searches + approximation generation
  - Performance results similar to CUDA implementation on NVIDIA GPU
- On current parallel architectures: TMD more efficiently solved on GPU architectures

- Investigate implicit vectorization on future Xeon Phis? (with arithmetic vector instructions for 64-bit integers)
  - Will the Xeon Phi handle divergence as efficiently as GPUs?
  - With emulation for 64-bit division and modulo?
- Study other hardware with SIMD units:
  - AMD GPU Tahiti HD 7970 (SIMD width = 64)
  - FPGA? (OpenCL 2.0?)