IPCC 2015 „ICE-DIP”

25/02/2015

Przemysław Karpiński
CERN Openlab

This research project has been supported by a Marie Curie Early European Industrial Doctorates Fellowship of he European Community’s Seventh Framework Programme under contract number (PITN-GA-2012-316596-ICE-DIP)”

Agenda

- ICE-DIP overview
- Experiments at CERN
- LHCb Trigger system
- Computing at CERN
- Many-core research

ICE-DIP Overview

Theme	WP	ESR	Challenge	Research
Silicon Photonics	WP1	ESR1 (Santa-clara, US)	Need affordable, high throughput, radiation tolerant links	Design, manufacture, test under stress a Si-photonics link
Reconfi-gurable Logic	WP2	ESR2 (Munich, Germany)	Reconfigurable logic is used where potentially more programmable CPUs could be proposed	A hybrid CPU/FPGA data pre-processing system
DAQ networks	WP3	ESR3 (Gdańsk, Poland	Bursts in traffic are not handled well by off-the-shelf networking equipment	Loss-less throughput up to multiple Tbit/s with new protocols
High performance data filtering	**WP4**	ESR4 (Munich, Germany)	Accelerators need network data, but have very limited networking capabilities	Direct data access for accelerators (network-bus-devices-memory)
		ESR5 (Paris, France)	**Benefits of new computing architectures are rarely fully exploited by software**	**Find and exploit parallelization opportunities and ensure forward scaling in DAQ networks**

Technical work

Activity	Status	Measurables
VCL library port for KNC	Completed?	- Good cooperation with VCL author Agner Fog (TUD, Copenhagen) - Code published in public domain (GPL) - Measurements gathered, - Article pending for acceptance.
LLVM as large code optimization platform	Hardware manufacturers need to put effort	- Possible methodology for both industry and academia,
HEP benchmarking suite	Under development	- New benchmark suite and algorithm library for HEP available in public domain (permissive license)
Blog on Many-core	Continuous work	- cern.ch/manycore

CERNopenlab

Computation at CERN

ONLINE:

Gaudi

WLCG
Worldwide LHC Computing Grid

ALFA

OFFLINE:

Geant
The best Geant ever

POOL

ROOT
Data Analysis Framework

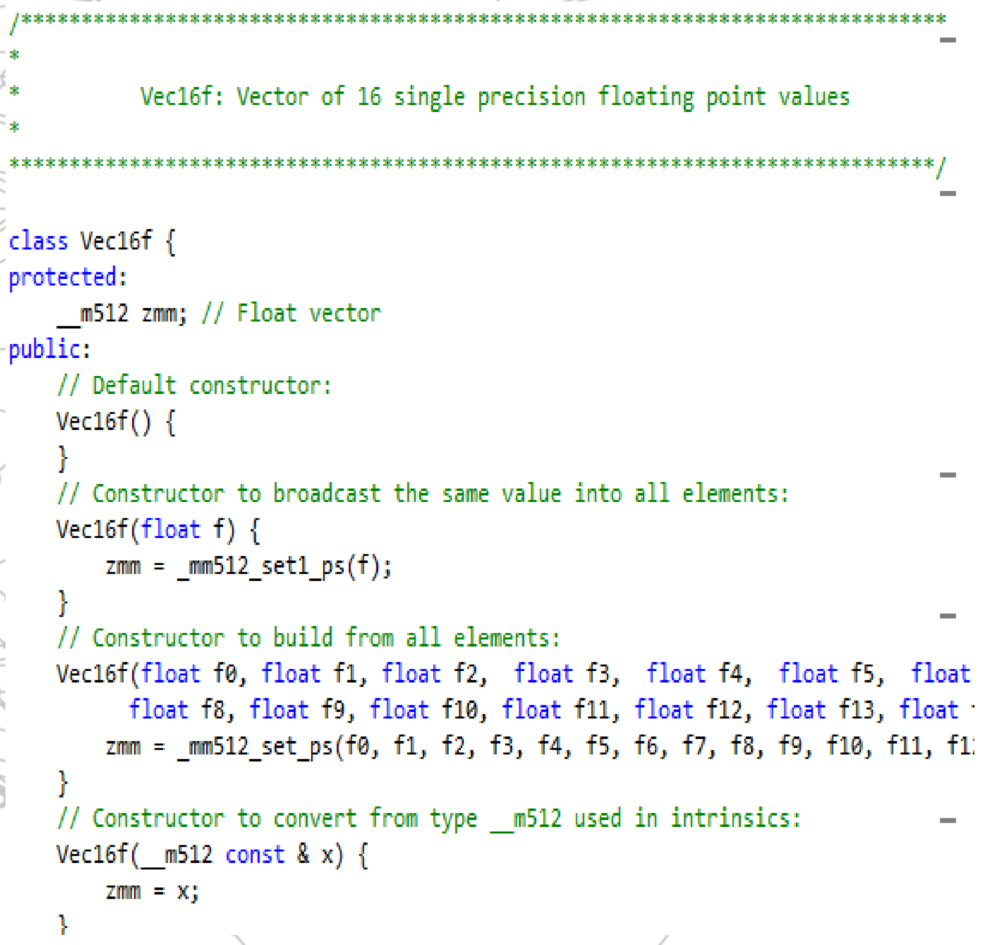
Wrong questions asked?

- *How do I measure performance?*
  - *Do you know what the metric is?*
- *How do I increase performance?*
  - *What are your hot-spots?*
- *How do I make my solution scalable?*
  - *What is your definition for scaling?*

Better questions?

- *How do I increase performance with minimal effort?*
  - *#pragma ...*
  - *Compile with –O3 –fastmath*
  - *Use faster library*
- *How do I choose proper metric?*
  - *Measure throughput*
  - *Measure latency*
  - *Measure memory utilization*
- *How do I create specification of my software?*
  - *Code IS the specification*

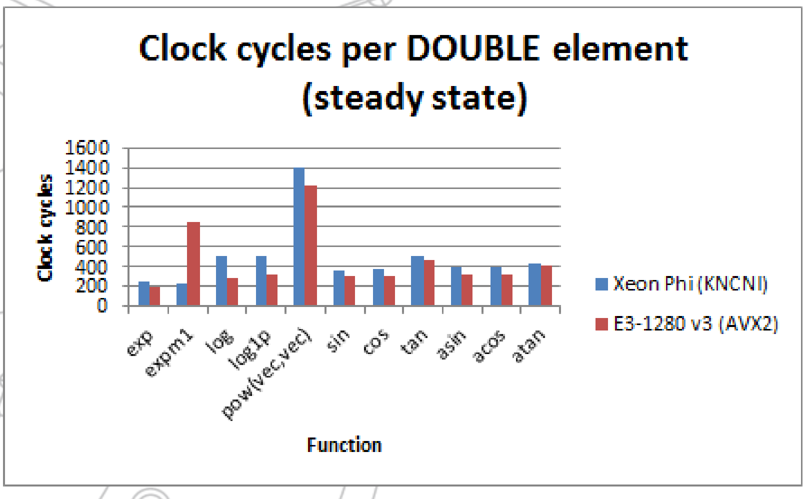
VCL and VCLKNC

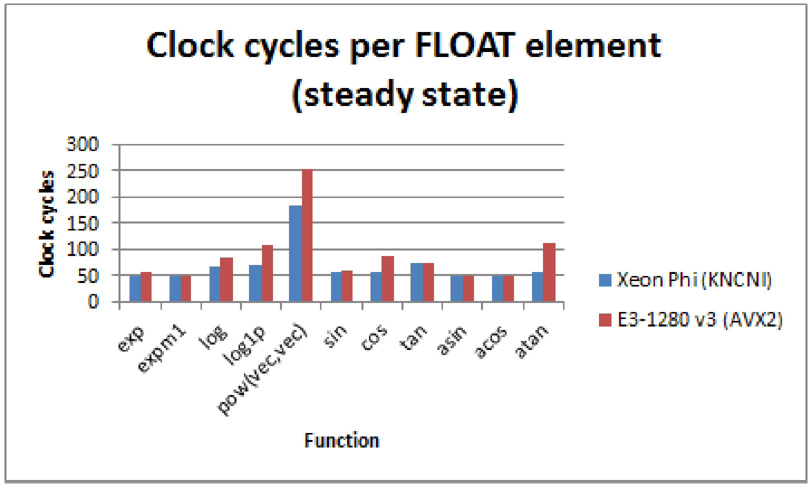


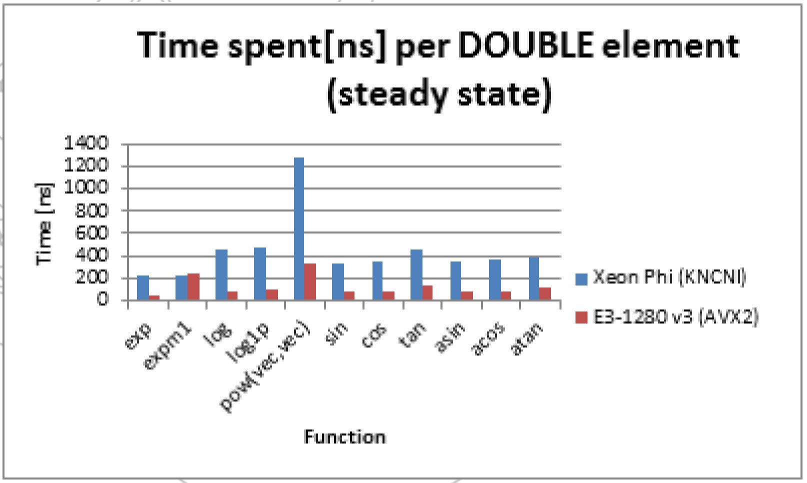
```
/*****************************************************************************
*
*          Vec16f: Vector of 16 single precision floating point values
*
*****************************************************************************/

class Vec16f {
protected:
    __m512 zmm; // Float vector
public:
    // Default constructor:
    Vec16f() {
    }
    // Constructor to broadcast the same value into all elements:
    Vec16f(float f) {
        zmm = _mm512_set1_ps(f);
    }
    // Constructor to build from all elements:
    Vec16f(float f0, float f1, float f2,  float f3,  float f4,  float f5,  float
          float f8, float f9, float f10, float f11, float f12, float f13, float
        zmm = _mm512_set_ps(f0, f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f1
    }
    // Constructor to convert from type __m512 used in intrinsics:
    Vec16f(__m512 const & x) {
        zmm = x;
    }
```

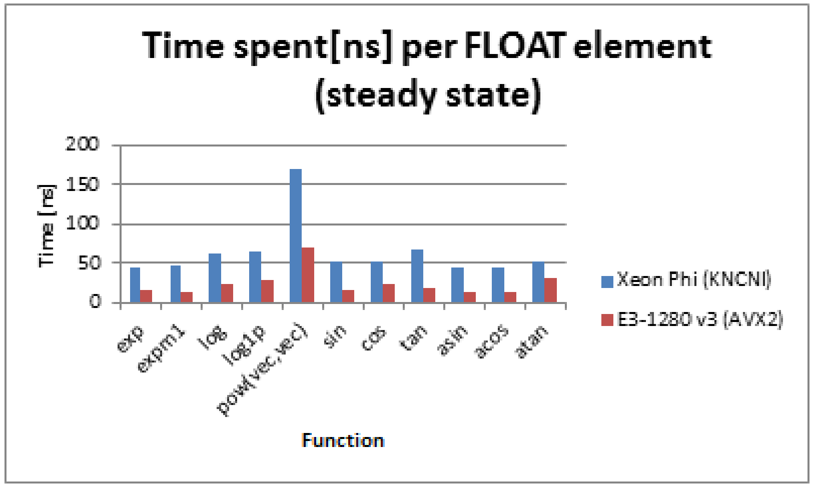
- SIMD vector abstraction layer
- Based on VCL library by Agner Fog (TUD, Copenhagen) www.agner.org
- Classes hiding SSE,AVXx
- VCLKNC – extension for IMCI (KNC)
- Invaluable learning materials!
- https://bitbucket.org/veclibknc/vclknc (GPL, proprietary licensing possible)

VCL: KNC vs XEON

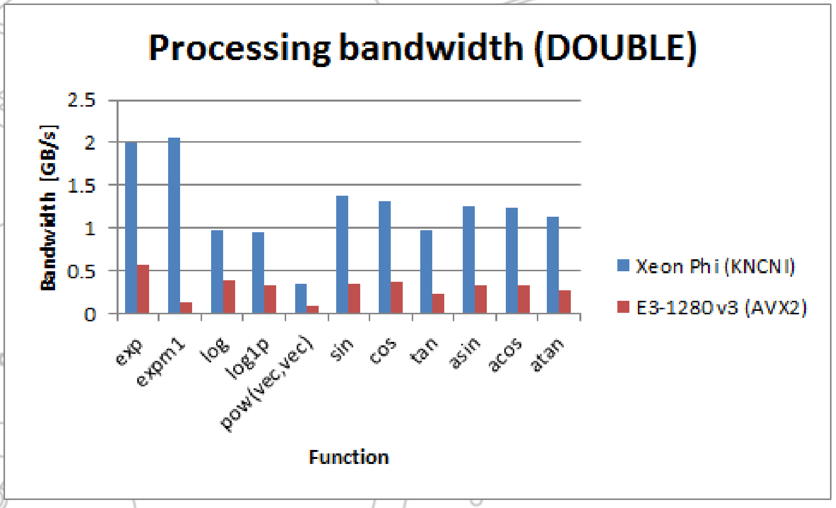


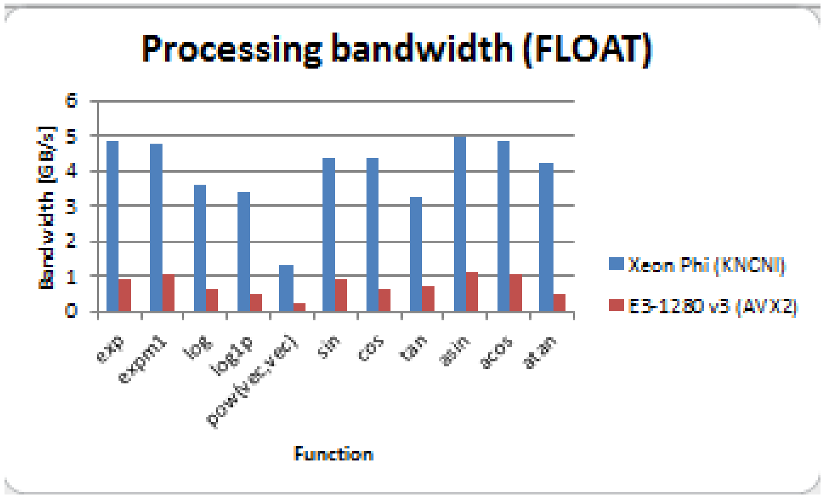






VCL: KNC vs XEON





Conclusions:

- Intrinsics are not that complicated (but tricky sometimes)
- KNC core microarchitecture is not that bad
- Can we get higher frequency?
- Use floats instead of doubles!
- Throughput is promising.

We need systematic revolution

1) Write simplest code that solves your problem
   - We ALWAYS underestimate complexity!
   - Not sure what is proper SPECIFICATION before writing code down
   - Early optimization is overkill

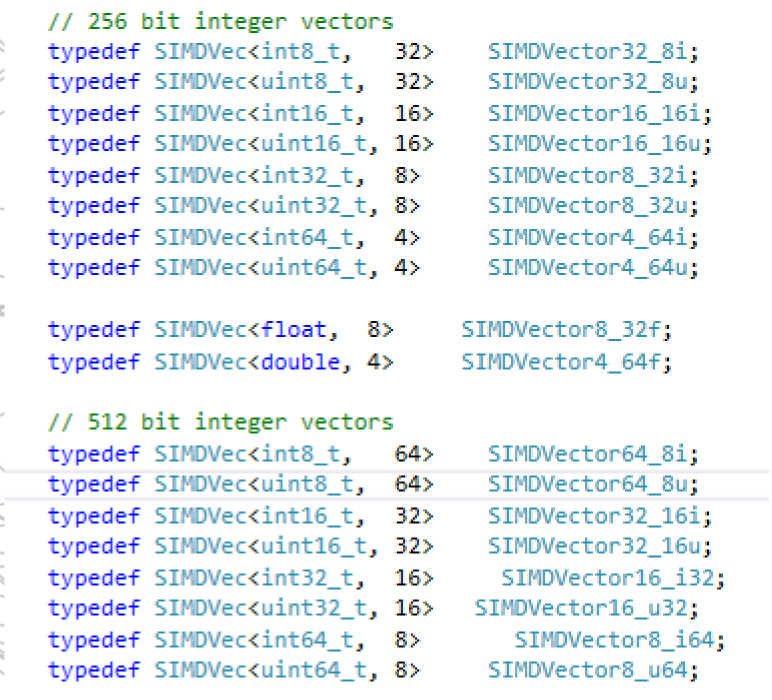
2) Identify „hot spots” and optimize them
   - Hot spot is not only a function: it can be algorithm or structure

3) Repeat 2) until it is REASONABLE!

4) Write version 2 and start from the beginning
   - Don’t be afraid to do that! You how knowledge you didn’t have at stage 1)
   - Some components can and should be re-used

UME – Unified Multi/Manycore Environment

SIMD abstraction layer:



```
// 256 bit integer vectors
typedef SIMDVec<int8_t,   32>    SIMDVector32_8i;
typedef SIMDVec<uint8_t,  32>    SIMDVector32_8u;
typedef SIMDVec<int16_t,  16>    SIMDVector16_16i;
typedef SIMDVec<uint16_t, 16>    SIMDVector16_16u;
typedef SIMDVec<int32_t,  8>     SIMDVector8_32i;
typedef SIMDVec<uint32_t, 8>     SIMDVector8_32u;
typedef SIMDVec<int64_t,  4>     SIMDVector4_64i;
typedef SIMDVec<uint64_t, 4>     SIMDVector4_64u;

typedef SIMDVec<float,  8>     SIMDVector8_32f;
typedef SIMDVec<double, 4>     SIMDVector4_64f;

// 512 bit integer vectors
typedef SIMDVec<int8_t,   64>    SIMDVector64_8i;
typedef SIMDVec<uint8_t,  64>    SIMDVector64_8u;
typedef SIMDVec<int16_t,  32>    SIMDVector32_16i;
typedef SIMDVec<uint16_t, 32>    SIMDVector32_16u;
typedef SIMDVec<int32_t,  16>     SIMDVector16_i32;
typedef SIMDVec<uint32_t, 16>   SIMDVector16_u32;
typedef SIMDVec<int64_t,  8>       SIMDVector8_i64;
typedef SIMDVec<uint64_t, 8>     SIMDVector8_u64;
```

- VCL, VC, Boost::SIMD
- Library selection at compile time
- Uniform interface chosen after analysis of libraries
- Vector symetry resolved by emulation
- Possible to „plug-in” other libraries

UME – Unified Multi/Manycore Environment

Next steps:

- „Other” abstraction layers
- Integrated benchmarking capabilities
- Microbenchmarking platform characteristics
  - Before or even during application compilation
- Canonical design of HEP algorithms
  - Ability to select parameters of the algorithm based on the platform specifics
  - Ability to re-use the algorithm for other applications
  - Example algorithms: Hough Transform Kalman Filter,
  - Canonical algorithm FORCES input data structures layout!!!
- Autotuning based on runtime information
  - It’s difficult to do „real” autotuning

Q&A

?

Thank you for attention!