# Exploring Intel Xeon Phi and NVIDIA GPUs for Nuclear Physics Simulations

Ümit V. Çatalyürek
Collaborator: Erik Saule, Kamer Kaya

THE OHIO STATE UNIVERSITY

Department of Biomedical Informatics
Department of Electrical and Computer Engineering
Department of Computer Science and Engineering (courtesy)

umit@bmi.osu.edu

International Conference on Nuclear Theory in the Supercomputing Era
Ames, Iowa
May 16, 2013

# Exploring Intel Xeon Phi for Nuclear Physics Simulations

Ümit V. Çatalyürek
Collaborator: Erik Saule, Kamer Kaya

THE OHIO STATE UNIVERSITY

Department of Biomedical Informatics
Department of Electrical and Computer Engineering
Department of Computer Science and Engineering (courtesy)

umit@bmi.osu.edu

International Conference on Nuclear Theory in the Supercomputing Era
Ames, Iowa
May 16, 2013

# Outline

# What is Intel MIC?

Intel Many integrated Core (MIC) is a coprocessor architecture developped by Intel as a response to GPUs becoming popular in High Performance Computing.

## What GPUs do well?

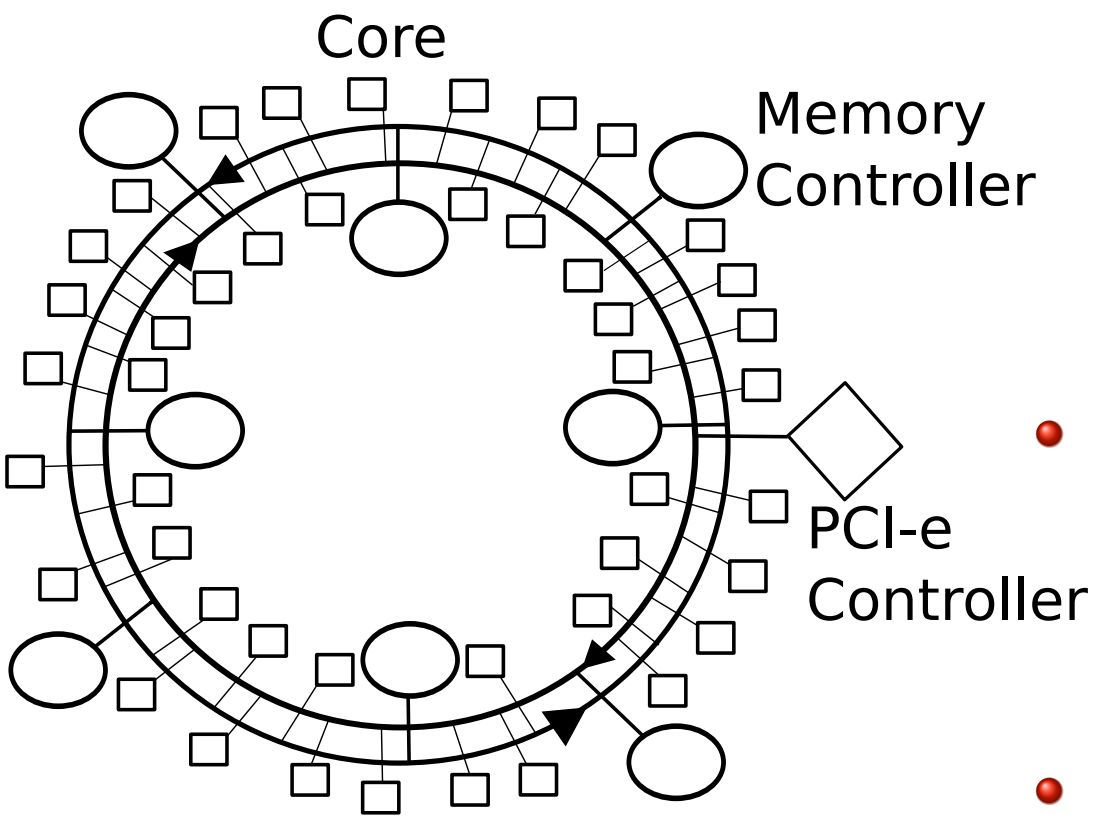
- Get a lot of GFlops by using hundreds to thousands of "cores"
- SIMD-like execution
- Hide memory latency by using "free" context switch

## What GPUs do not do well?

- Alien to program
- Poor support for legacy applications
- Inter thread communications
- Branching

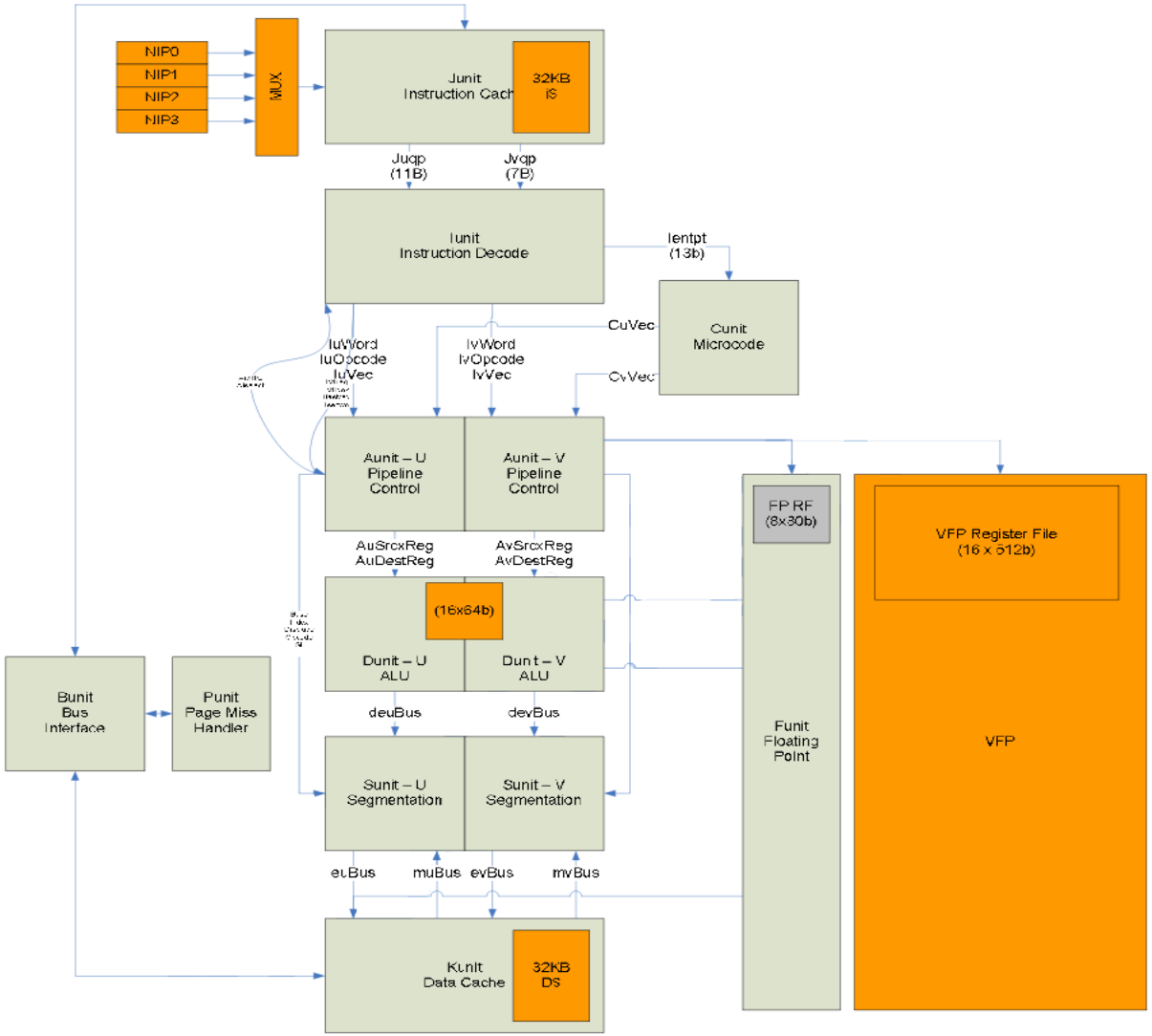
Goal of Intel Xeon Phi: do all of it well!

# MIC Overall Architecture (numbers for Xeon Phi SE10P)




- 8 memory controllers
  - GDDR5
  - 2 channels (32-bit)
  - 5.5GT/s
  - 352GB/s aggregated peak
    - The ring bus limits to 220GB/s
    - 180GB/s in practice
- 61 cores
  - 32KB of L1 cache
  - 512KB of L2 cache
  - LRU
  - 8-way associative
- 1 PCI-e controller
  - to the host (2GB/s guaranteed to memory)

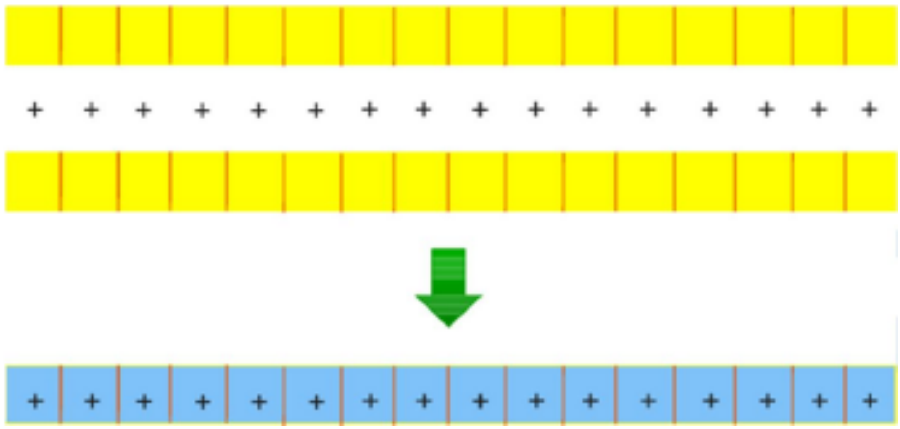
# MIC Core architecture (numbers for Xeon Phi SE10P)




Source: Intel Xeon Phi System Software Developer's Guide

- clocked at 1.1GHz
- 64-bit
- 4 hardware threads
  - no context switching cost
  - 1 thread cannot execute instructions in back-to-back cycles
- A vectorial unit
  - (more details on the next slides)
- Two instruction pipes:
  - 2 ALU ops
  - ALU + MEM ops
  - ALU + VFP ops
  - VFP + MEM ops
- In-order execution

# Vectorial Unit (SIMD)



32 512-bits registers (16x32, 8x64)

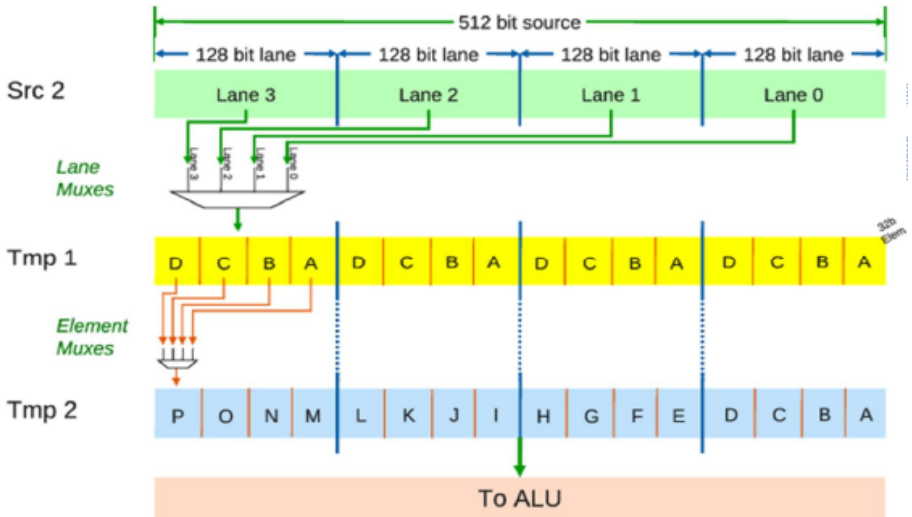
**Unary math operators (2 cycles)**

recip, sqrt, rsqrt, log, exp, exp2, log2

**Binary operators (1 cycle)**

mul, div, add, sub, and, xor, or, fma

# Vectorial Unit (SIMD)




32 512-bits registers (16x32, 8x64)

## Unary math operators (2 cycles)

recip, sqrt, rsqrt, log, exp, exp2, log2

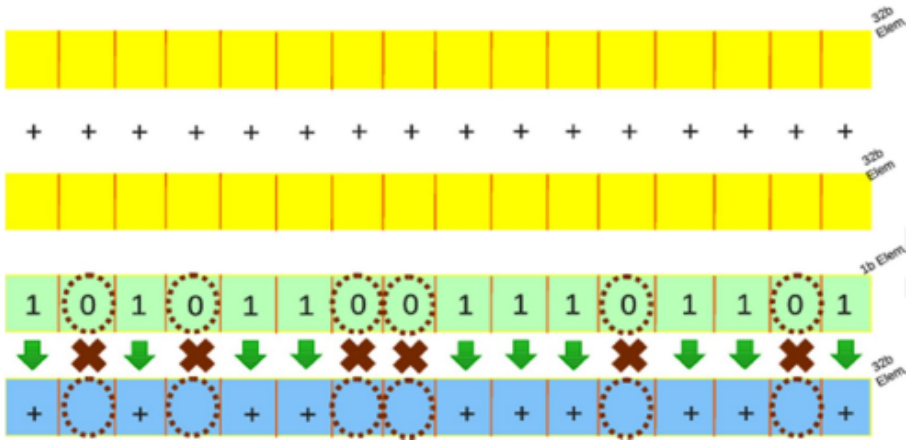
## Binary operators (1 cycle)

mul, div, add, sub, and, xor, or, fma

## Swizzling

Registers are organized in 4 lanes. Almost all instructions support recombining lanes and elements.

# Vectorial Unit (SIMD)




32 512-bits registers (16x32, 8x64)

## Unary math operators (2 cycles)

recip, sqrt, rsqrt, log, exp, exp2, log2

## Binary operators (1 cycle)

mul, div, add, sub, and, xor, or, fma

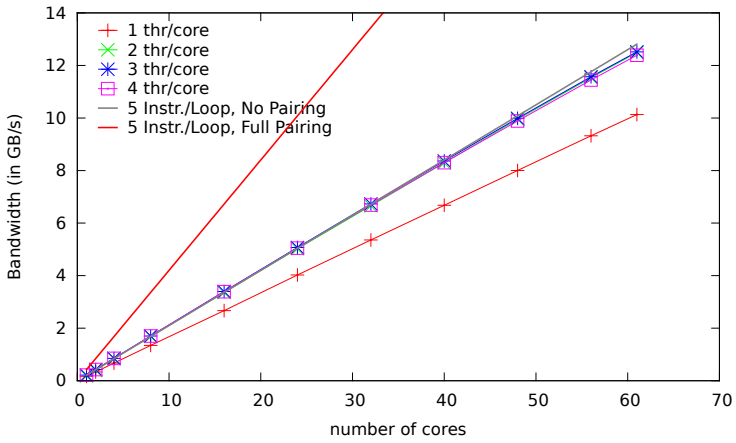
## Swizzling

Registers are organized in 4 lanes. Almost all instructions support recombining lanes and elements.

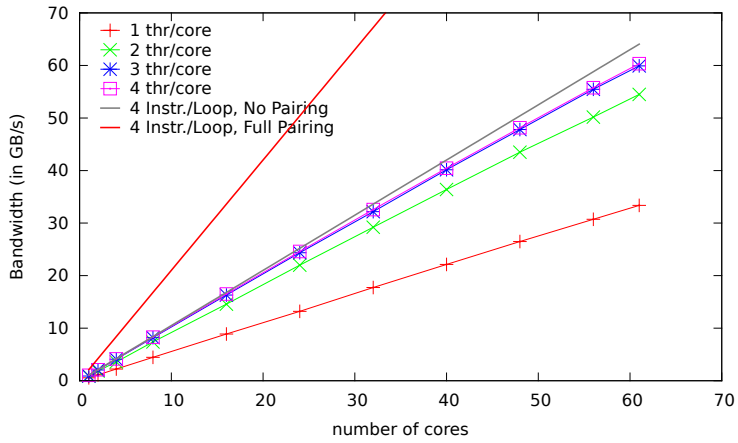
## Masking

8 masking registers of 16 bits. Specify which elements to be written.

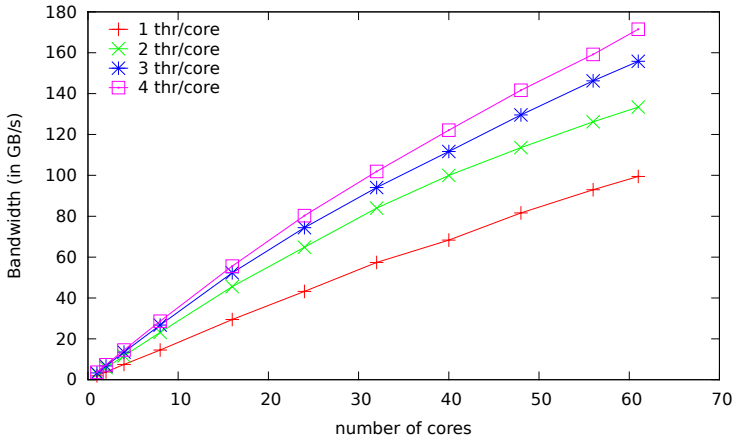
# SIMD and Threads are key (Read bandwidth)



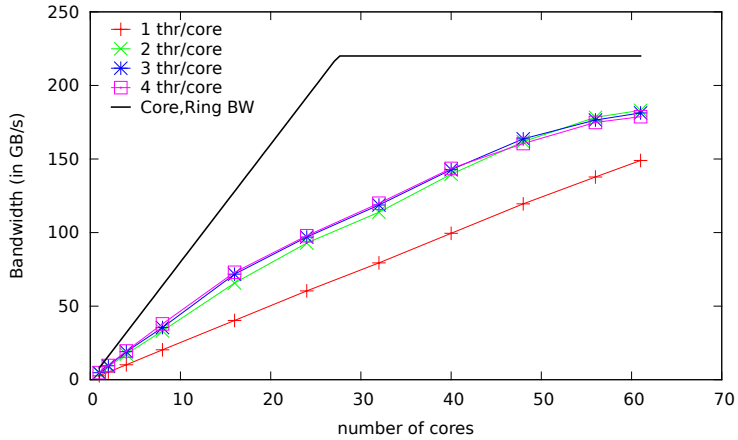

(a) Simple sum, `char`, `-O1`




(b) Simple sum, `int`, `-O1`




(c) Vectorization




(d) Vectorization + prefetching

# When Should I Use Intel MIC ?

## Key points

- Large memory bandwidth (about 180GB/s in practice)
- 50+ cores with mandatory use of hardware threading
- 512-bit wide SIMD registers:
  - FMA: up to 2x16 SP Flop/c (2x8 DP Flop/c)
  - otherwise: up to 16 SP Flop/c (8 DP Flop/c)
- On a 61-core configuration at 1.1Ghz (SE10P):
  - FMA: 2x16x61x1.1Ghz = 2 TFlop/s SP (1TFlop/s DP)
  - otherwise: 16x61x1.1Ghz = 1T TFlop/s SP (500GFlop/s DP)

## Problem Requirement

- Parallel scaling with at least 100 concurrent threads
- Some use of SIMD operations

# How to program Intel Xeon Phi

## Software

The card runs GNU/Linux. The instruction set is incompatible with x86_64. Intel Compiler supports Intel Xeon Phi's instruction set (supports C, C++, Fortran). Support in GCC is coming. OpenMP, pthread, CILK, TBB...

## Native

Log on the card with SSH. IP connexion to the host over PCI-express to deal with the outside world.

## Offload

A set of pragma that declare a block of code should be executed on Intel Xeon Phi. (Somewhat similar to OpenACC)

## MPI

MPI support with MVAPICH. You can allocate one rank or multiple ranks to an Intel Xeon Phi card. Communications between cards and nodes are handled by MPI.

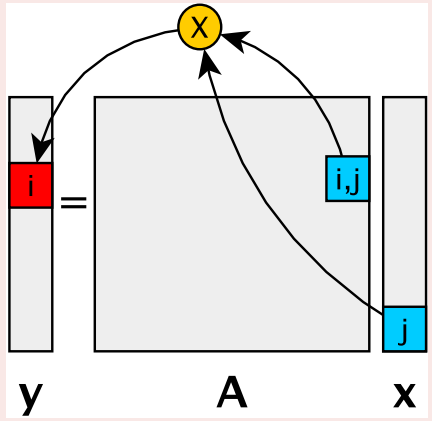
# Outline

# Sparse Matrix Multiplication Kernels

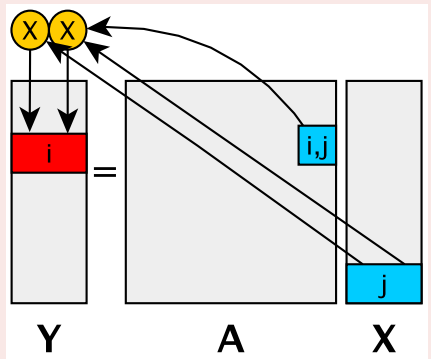
They appear in many scientific computations:

- Sparse Matrix Eigensolvers
  - Lanzcos (SpMV)
  - LOBPCG (SpMM)
- Social Network Analysis
  - PageRank (SpMV)
  - PersPageRank (SpMV, SpMM)
- Preconditioners
  - Scaling (SpMV, tSpMV)

## SpMV




## SpMM



# Setting

## Xeon Phi SE10P

(Pre-release hardware.) 61 cores, 1.1 GHz, 8GB DDR5.

## Compare with

2 dual CPU architectures (impl. native)

- Intel Xeon X5680 (Westmere)
- Intel Xeon E5-2670 (Sandy Bridge)

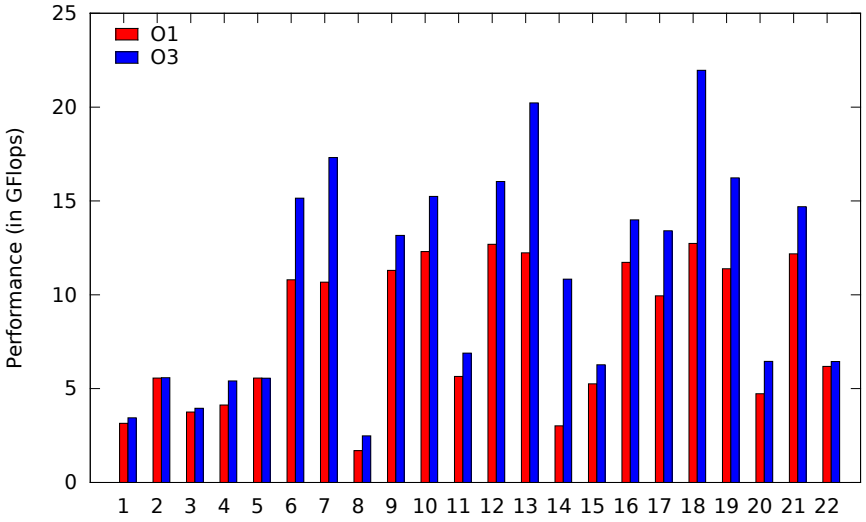
2 GPU architectures (impl. cuSparse)

- NVIDIA Tesla C2050
- NVIDIA Kepler K20

| # | name | #row | #nonzero |
|---|---|---|---|
| 1 | *shallow_water1* | 81,920 | 204,800 |
| 2 | *2cubes_sphere* | 101,492 | 874,378 |
| 3 | *scircuit* | 170,998 | 958,936 |
| 4 | *mac_econ* | 206,500 | 1,273,389 |
| 5 | *cop20k_A* | 121,192 | 1,362,087 |
| 6 | *cant* | 62,451 | 2,034,917 |
| 7 | *pdb1HYS* | 36,417 | 2,190,591 |
| 8 | *webbase-1M* | 1,000,005 | 3,105,536 |
| 9 | *hood* | 220,542 | 5,057,982 |
| 10 | *bmw3_2* | 227,362 | 5,757,996 |
| 11 | *pre2* | 659,033 | 5,834,044 |
| 12 | *pwtk* | 217,918 | 5,871,175 |
| 13 | *crankseg_2* | 63,838 | 7,106,348 |
| 14 | *torso1* | 116,158 | 8,516,500 |
| 15 | *atmosmodd* | 1,270,432 | 8,814,880 |
| 16 | *msdoor* | 415,863 | 9,794,513 |
| 17 | *F1* | 343,791 | 13,590,452 |
| 18 | *nd24k* | 72,000 | 14,393,817 |
| 19 | *inline_1* | 503,712 | 18,659,941 |
| 20 | *mesh_2048* | 4,194,304 | 20,963,328 |
| 21 | *ldoor* | 952,203 | 21,723,010 |
| 22 | *cage14* | 1,505,785 | 27,130,349 |

# Outline

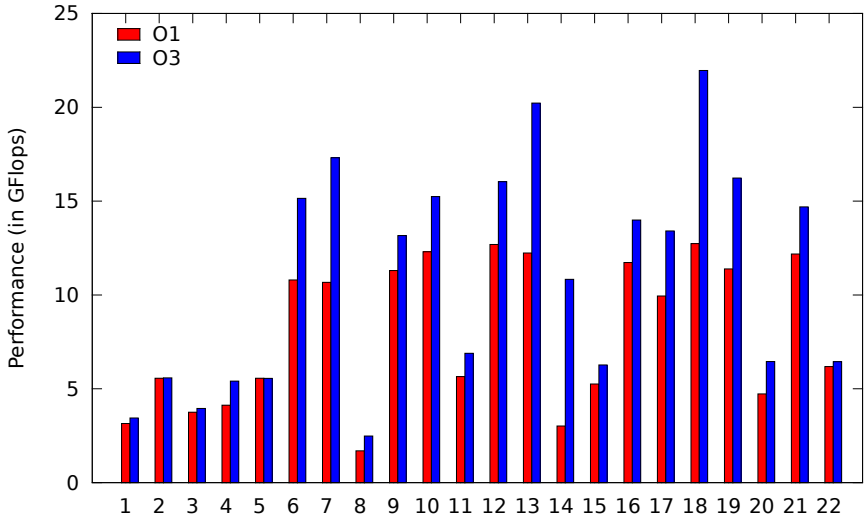
# Vectorization and SpMV

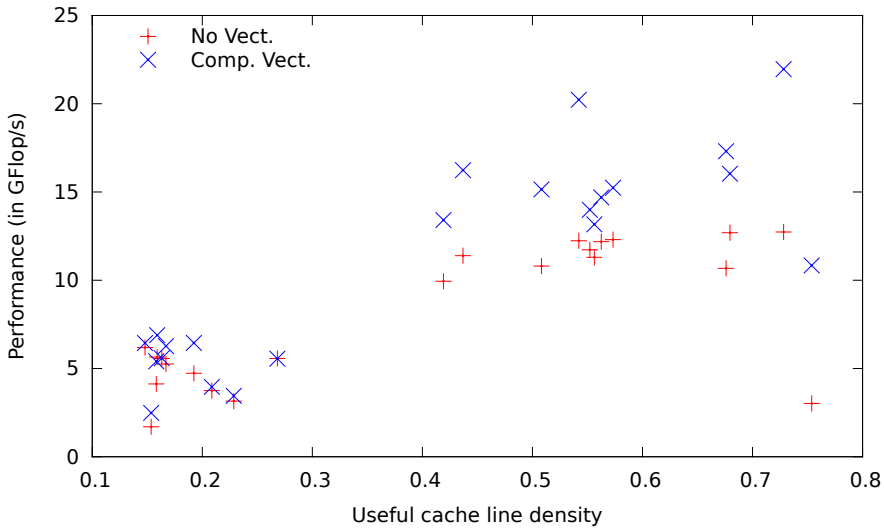



## vgatherd

Vectorized code uses the `vgatherd` instruction. Given a SIMD vector of 16 addresses, fetch the entries from the same cacheline (512bits/64 bytes) in a single instruction.

# Vectorization and SpMV




## vgatherd

Vectorized code uses the `vgatherd` instruction. Given a SIMD vector of 16 addresses, fetch the entries from the same cacheline (512bits/64 bytes) in a single instruction.

## Useful Cache Line Density

For each row, compute the ratio of used vector entries over the number brought to cache. (Even accessing a single entry cause the whole cache line to be fetched.)

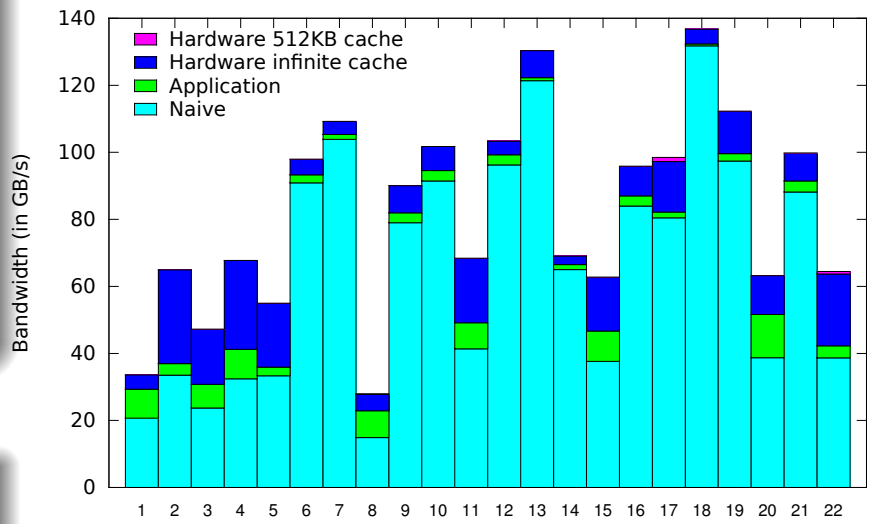
# Bandwidth of SpMV

## How it is computed

- Naive: $\frac{matrix\ \ size}{runtime}$
- Application: $\frac{matrix\ \ size+vector\ \ sizes}{runtime}$
- Hardware Infinite Cache:
  - assuming a distribution of the rows
  - $\frac{memory\ \ touched}{runtime}$
- Hardware 512KB Cache:
  - same with finite cache

## Remarks

- Vector size should not be ignored
- Multiple cache transfers are significant
- Small caches are not a problem

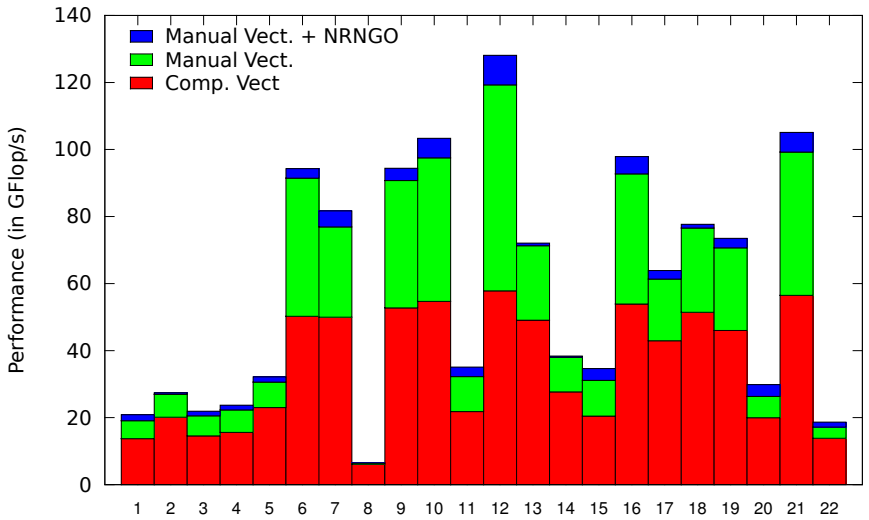


# Outline

# Vectorization for SpMM

## Three implementations

- **Compiler vectorized** uses arbitrary number of vectors.
- **Manually vectorized** is specialized for number of vectors multiple of 8. It uses Fused Multiply Add.
- **Manually vectorized + NRNGO** bypasses the Read For Ownership protocol and allows arbitrary order for committing to memory.



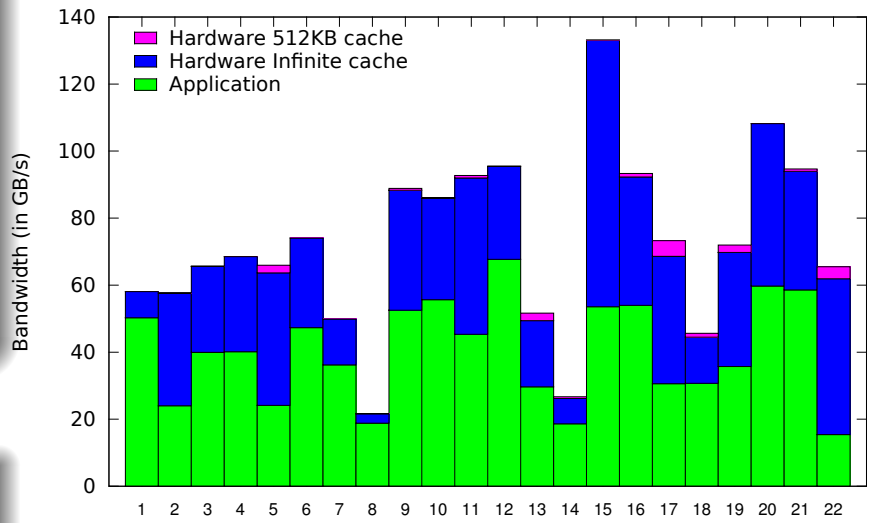
# Bandwidth of SpMM

## How it is computed

- Application: $\frac{matrix\ \ size + vectors\ \ sizes}{runtime}$
- Hardware Infinite Cache:
  - assuming a distribution of the rows
  - $\frac{memory\ \ touched}{runtime}$
- Hardware 512KB Cache:
  - same with finite cache

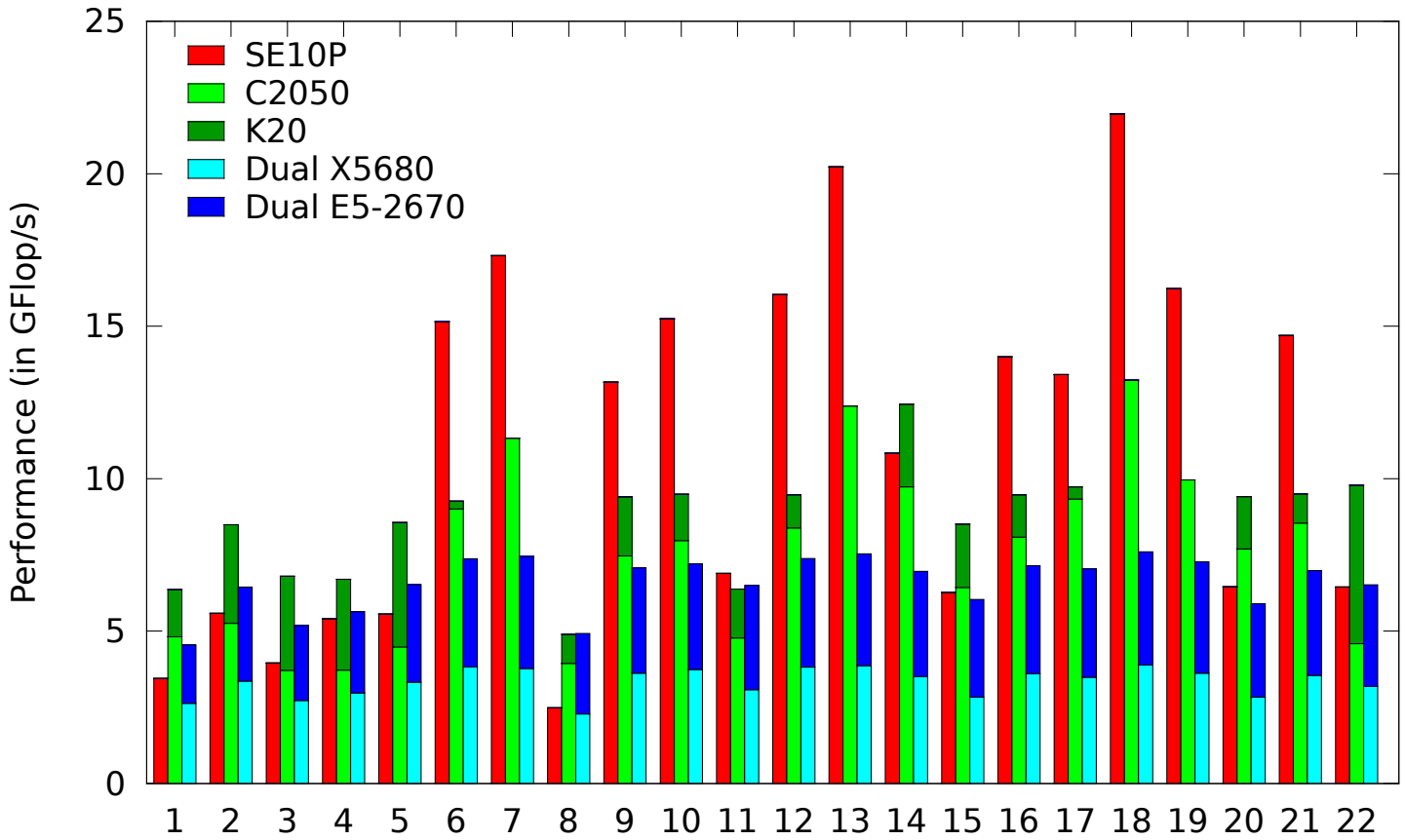
## Remarks

- Multiple cache transfers are significant
- Small caches are not a problem



# Outline

# Architectural Comparison - SpMV

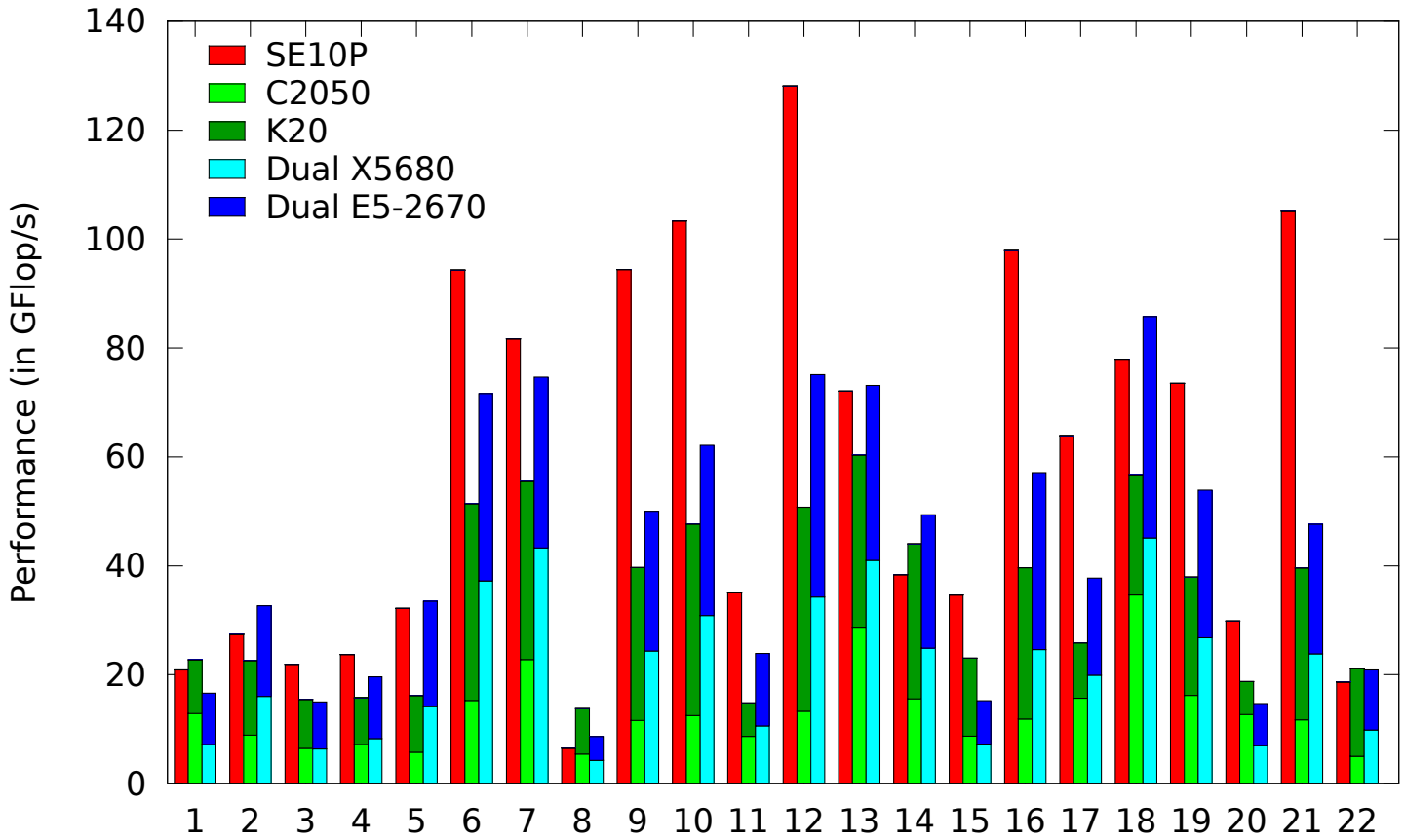



Xeon Phi is best in 12 cases, obtains $\geq$ 15GFlop/s on 7 cases, peaks at 22GFlop/s

NVIDIA K20 is best in 9 cases (incl. the 5 smallest), never passes 13.2GFlop/s.

Dual Sandy Bridge is best in 1 case, never passes 7.6GFlop/s.

# Architectural Comparison - SpMM



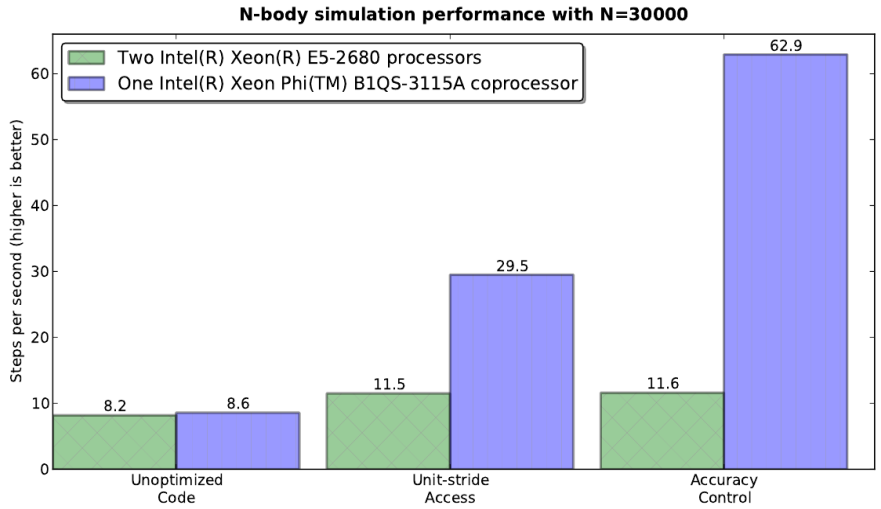

Xeon Phi is best in 14 cases, reaches $\geq$60GFlop/s 9 times, peaks at 128GFlop/s.
NVIDIA K20 is best in 3 cases, peaks at 60GFlop/s.

Dual Sandy Bridge is best in 5 cases, reaches $\geq$ 6 GFlop/s on 6 cases.

# Outline

# Numerical iterative N-body simulation

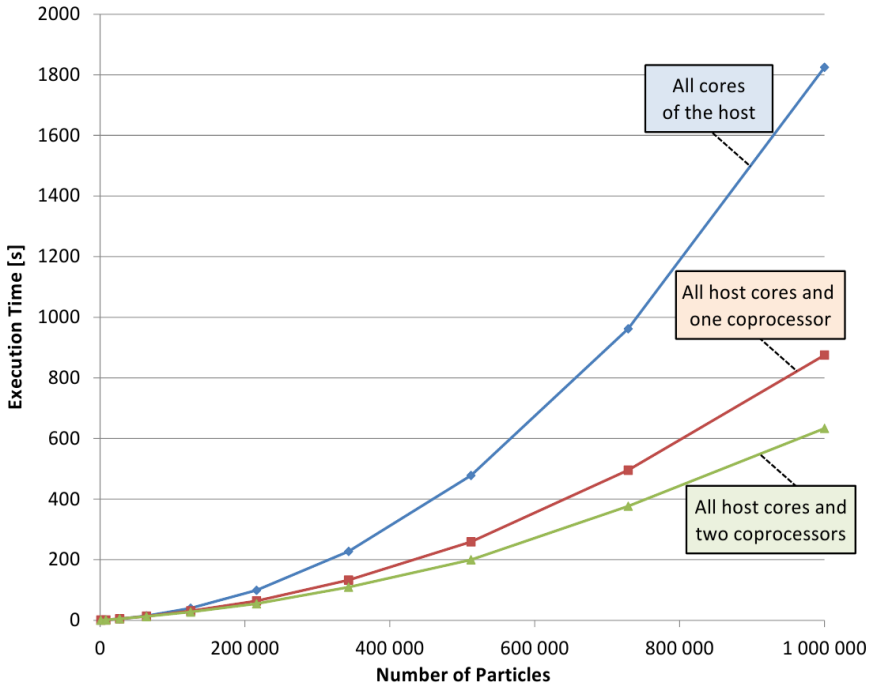
- Unit stride access enables more vectorization. (Going from array-of-structure to structure-of-array.)
- Relaxing IEEE 754 floating point compliance to compute $\frac{1}{\sqrt{x}}$ faster.
- Comparison with 2 octo-core Sandy Bridge processor at 2.7 GHz. Xeon Phi is 5.6 times faster.




Source: [Andrey Vladimirov and Vadim Karpusenko. *Test-driving Intel Xeon Phi coprocessors with a basic N-body simulation*. Tech. Rep., Colfax International, January 2013.]

# Particle in Cell Simulation

- Using TBB-like construct, share work between host and device.
- System: Two six-core Intel Xeon Westmere CPUs (X5680, 12M Cache, 3.33 GHz, Hyper-Threading) and two Xeon Phi coprocessors.
- 1 MIC = 2 Westmere




Source: [Jirí Dokulil, Enes Bajrovic, Siegfried Benkner, Sabri Pllana, Martin Sandrieser, and Beverly Bachmayer. *Efficient hybrid execution of c++ applications using Intel Xeon Phi coprocessor*. CoRR, abs/1211.5530, 2012.]

# Outline

# Conclusions

## Intel Xeon Phi can benefit physics computation

- Sparse Linear Algebra
- Particle in Cell
- N-body numerical simulation

## Key programming points

- Compatible with legacy code
  - but might take some work to optimize
- Must have lots of threads (at least 2 per core, so more than 120)
- Must use SIMD operations

# Thank you

## Support

Intel for allowing us to use pre-release Xeon Phi prototypes. NVIDIA for providing K20 cards. OSC for providing computation infrastructure.

## More information

contact : umit@bmi.osu.edu

visit: `http://bmi.osu.edu/hpc/` or `http://bmi.osu.edu/~umit`

## Research at HPC lab is supported by


Erik Saule, Kamer Kaya, and Ümit V. Çatalyürek.
Performance evaluation of sparse matrix multiplication kernels on intel xeon phi.
Technical Report arXiv:1302.1078, ArXiv, February 2013.