Intel Xeon Phi Processor

Kevin Olson Ning Li

May 3, 2017

nag®

Experts in numerical algorithms
and HPC services

Roadmap

Roadmap for these webinars

1. *Introduction to Xeon Phi*
2. Parallelism with OpenMP
3. Vectorization with OpenMP
4. Hands-on exercise
5. Xeon Phi Tuning - Part 1
6. Xeon Phi Tuning - Part 2
7. Hands-on exercise

Purpose of This Course

- To introduce the Intel Xeon Phi processor architecture, ways to program it, and optimization techniques to use its unique features
- This course is designed to help those doing technical computing
- Some examples of technical computing:
  - Machine learning
  - Data analytics
  - Physics simulations
  - Many more...(you know what you're doing!)

Purpose of This Course

- The key property of technical computing is its workload
- It uses a lot of arithmetic and it uses a lot of memory
- Most importantly: great potential for parallelism

Purpose of This Course

- Hardware vendors realize this and have made special tools for it.
- Example: Xeon processors
- Here you will learn how to fully utilize this, new generation of processors
- But first some historical context:

Some History

- Pre 2000s: CPUs improved by upping transistor count and clock frequency
- Post 2000s: Unable to continue frequency and transistor number scaling
  - First: added vector instructions to add arithmetic throughput
  - Next: Increased core count for parallelism, cores share memory, all on a single chip

Some History

- Core counts have increased from 8 to 28, and continue to increase.
- And vector widths within cores can range from 64 bits to 512 bits
  - We will learn more about this later
  - But for now: wider vector width means higher arithmetic throughput

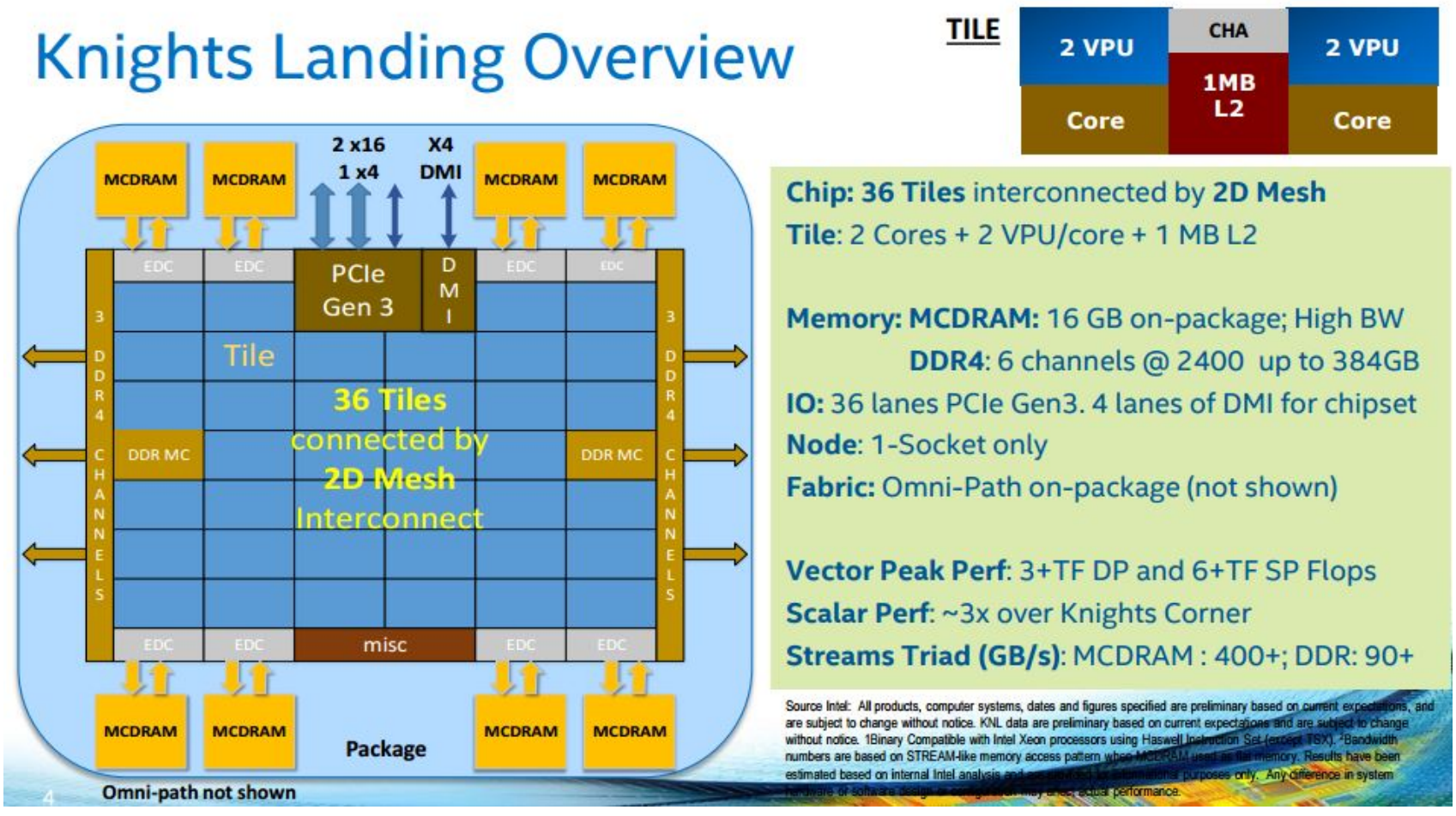
Some History

- Intel took this trend to the next logical step: even more cores
- Made special chip with 60+ cores (Xeon Phi processor).
- Gave it effective 1024 bit wide vector units
- Today we are on second generation of this throughput monster

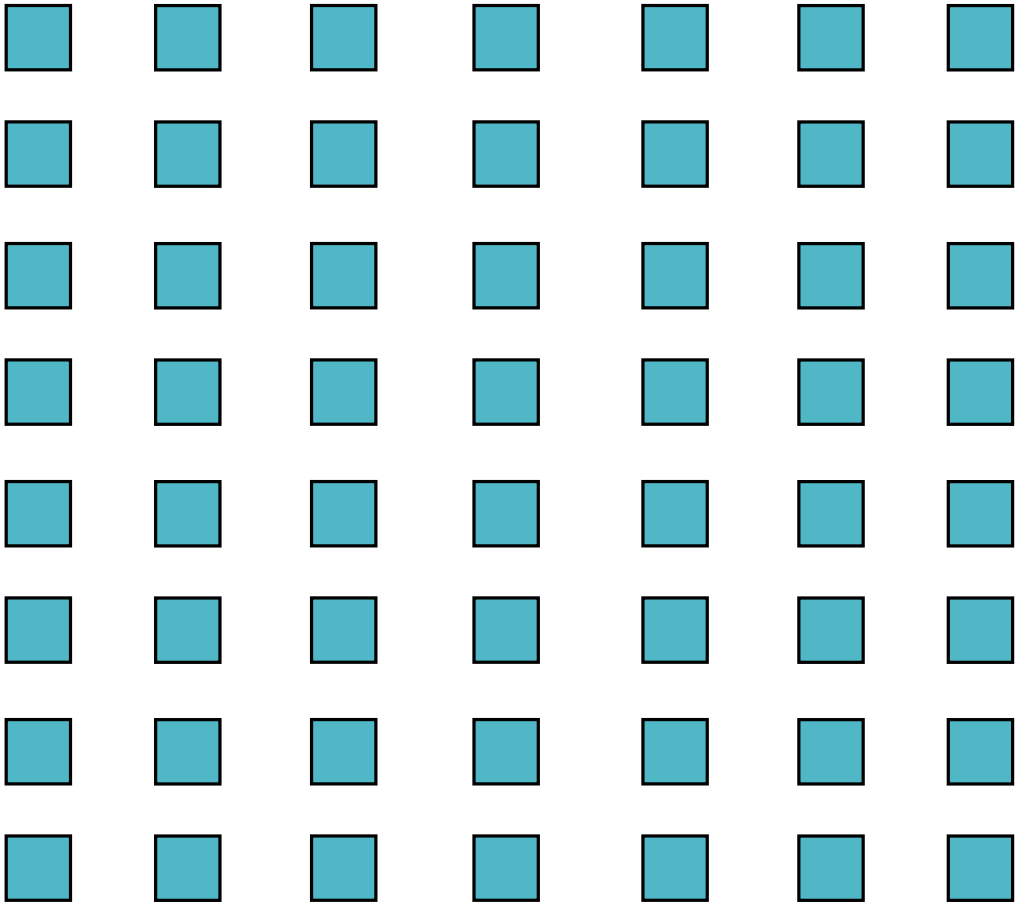
Xeon Phi processor

- Second generation Intel many-integrated-core (MIC) architecture
- NO offloading necessary as for the previous generation, Xeon Phi coprocessor
- Boots off-the-shelf OS's
- Runs single-threaded well (but optimize)
- Binary compatible
- Designed to accelerate applications needing raw throughput
- It has a high degree of parallelism
  - Because of core count and large vector lanes

Xeon Phi Processor Architecture Overview

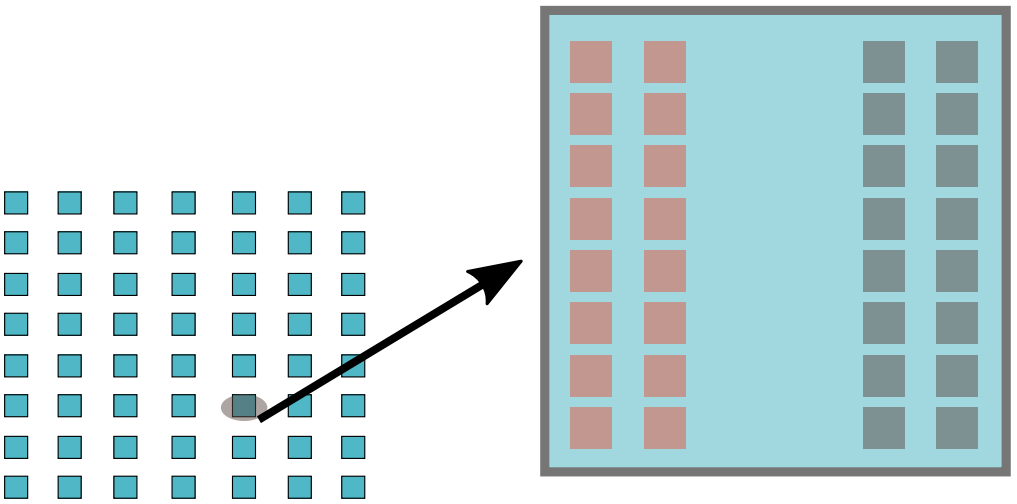


Xeon Phi Processor



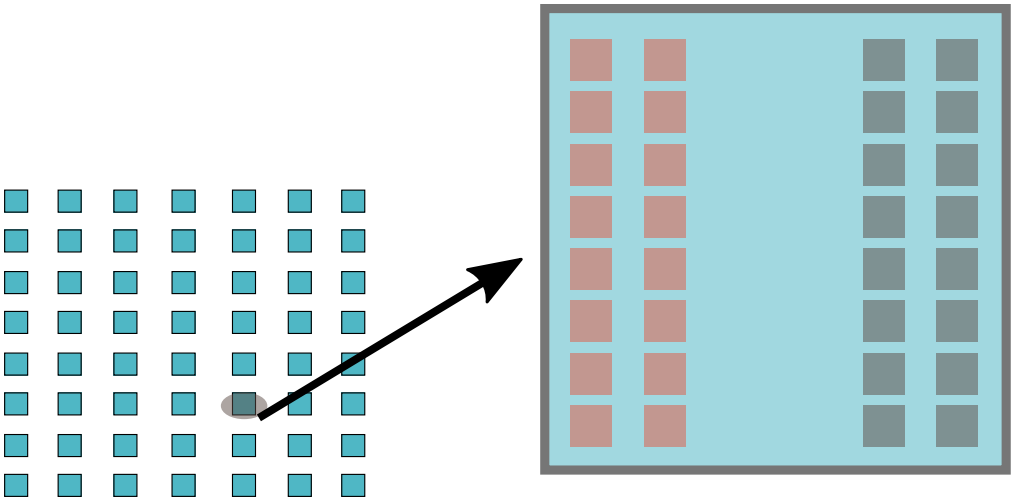
Xeon Phi processor has up to 36 tiles with 2 cores each, potentially getting a 72X speedup over serial

Xeon Phi Processor



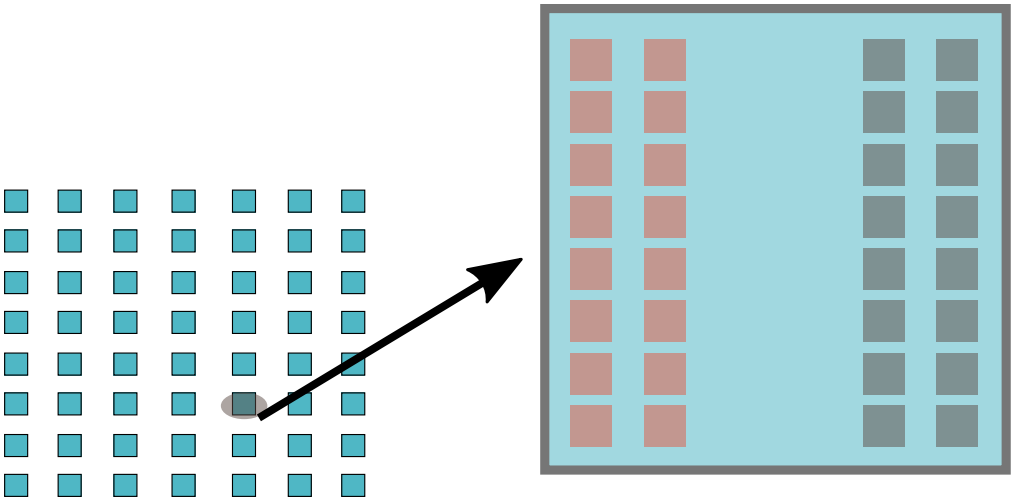
Furthermore, each core has two vector units.

Xeon Phi Processor



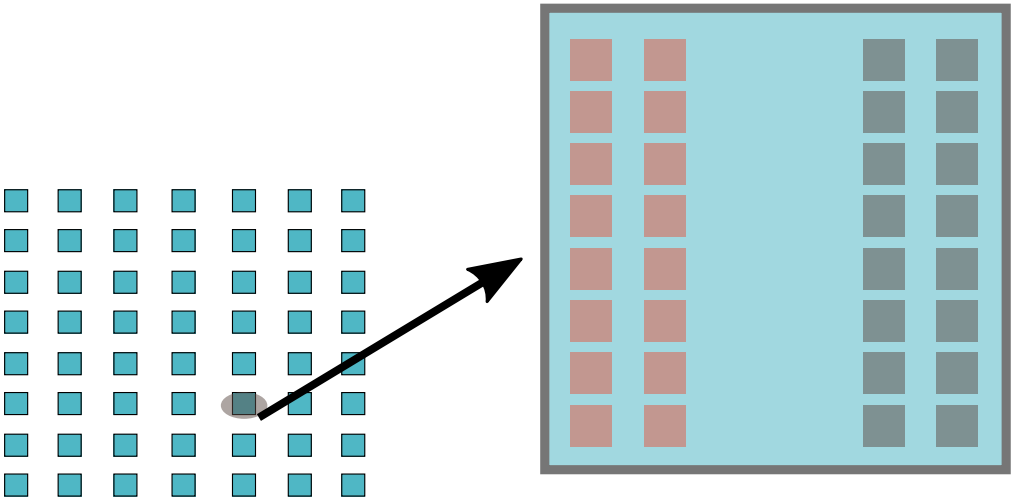
On 32 bit math, vector units each yield 16X more parallelism

Xeon Phi Processor



Properly used, this means further 32X speedup of code.

Xeon Phi Processor



That is on top of the 72X, so total potential speedup:
72*32=2304X

Xeon Phi Processor

- Big part of this course is parallelism
- With vector + multicore, you need a lot of it
- We will emphasize OpenMP to express this parallelism.

Metric Terms Defined

- Peak FLOPS: Theoretical floating point operations throughput
- Peak Bandwidth: Theoretical memory transfer rate
- TDP: Thermal Design Power
  - Gives a sense for power requirement

NOTE: These do not fully characterize the capabilities, but they are useful for inter-chip comparison

Metric Comparisons

While potential 2304X speedup is impressive, this is only relative to single core of the Xeon Phi processor. It is important to compare against another, actual processor.

- Xeon Phi Processor
  - Cores : up to 72
  - Peak flops : 6 TFLOPS (single), 3 TFLOPS (double)
  - Peak bandwidth : 450 GB/s
  - TDP : 200 watts
- 2 socket Haswell
  - Model : 2670 v3
  - Cores : 2x12 = 24
  - Peak flops : 1.5 TFLOPS
  - Peak bandwidth : 110 GB/s
  - TDP : 2x120 = 240 watts

Meaning

- The big win for accelerators is high bandwidth and flops per watt
- But Xeon Phi processor introduces new features also
- These make it more like normal CPU than external accelerator

Xeon Phi Processor Features: Self-hosting

- Means it does compute and OS.
- This considerably simplifies executing code.
- Just run it like a normal executable (no offloading necessary).
- Its NOT a separate co-processor like the previous generation Xeon Phi.
- It runs the entire "software stack", and is binary compatible.

Xeon Phi Processor Features: MCDRAM

- Xeon Phi Processor has transparent high-bandwidth memory called Multi-Channel DRAM or MCDRAM integrated on-package.
- 8 MCDRAM devices, each is 2 GB, total 16 GB.
- Can achieve up to 450 GM/s aggregate bandwidth.
- "transparent" means it lives in the same memory space as normal RAM (DDR4)
  - ☐ We will explain how this works

Xeon Phi Processor Features: AVX-512

- Xeon Phi Processor uses new AVX-512 vector instructions
- We won't use them directly
- Instead will instruct compiler when to use
- Result: potential 32X speedup of arithmetic codes

Xeon Phi Processor Features: AVX-512

Code such as

```
1 for(int i=0;i<32;i++){
2   z[i]=z[i]+x[i]*y[i];
3 }
```

potentially executed in a single clock cycle (fused multiply-add)

Xeon Phi Processor Features: "heavyweight" cores

- Each core capable of decent single-threaded performance
- This lets you see improvements right away after parallelizing
- Compare to "lightweight" which requires more tuning/rewriting

Xeon Phi Processor Features: "heavyweight" cores

With lightweight cores, the code

```
1  int x1=y1;
2  int x2=y2;
3  z1=x1*z1;
4  if(z1>5)
5    z2=x2*y2
```

could perform significantly worse than

```
1  int x1=y1;
2  z1=x1*z1;
3  int x2=y2;
4  int grtr5=(1^((unsigned int)(5-z2)>>255));
5  z2=(1-grtr5)*z2 + grtr5*z2*x2;
```

But with good hardware features, the simpler version should be competitive.

Using Xeon Phi Processor Features

- Xeon Phi processor features lighten burden on developers
- But they still must learn to use it
- This mostly means exposing sufficient parallelism
- But also important are tuning strategies to manage memory hierarchy (L1, L2 and L3 Caches, MCDRAM, and DDR4)

Using Xeon Phi Processor Features

- Xeon Phi processor more like a normal intel chip than a separate device
- Parallelism and tuning strategies are portable
- Meaning: Good code on Xeon Phi Processor should also be good on a Xeon Haswell
- Here you will learn the essentials of achieving this

Learning Objectives

- Parallelizing code with OpenMP
- Vectorizing code with OpenMP
- Xeon Phi Processor specific tuning
  - MCDRAM usage
  - Cache usage
  - Low level details

Learning Objectives

- The focus here will be more than just Xeon Phi Processor
- Also understanding performance issues in general
- Writing code to perform great on past,current, and future architecture

Practical Exercise 1

Logging Into Test Cluster

- `ssh userid@knl1-login.stampede.tacc.utexas.edu`
- You can compile code now, but NOT run executables.
- Upload the example codes from the practicals directory using sftp
- Start an interactive shell using the command `idev`. This will start an interactive shell on one of the Xeon Phi processors in the cluster.
- From here you can compile and run executables.

A Simple Hello World

- First exercise is ubiquitous "hello world"
- It is just like normal CPU code

A Simple Hello World

```
#include <iostream>
int main(int argc,char**argv){
  std::cout<<"Hello, world!\n";
  return 0;
}
```

A Simple Hello World

```
>> icpc helloworld.cpp
>> ./a.out
```

```
Hello, world!
```

Parallelizing Hello World

- This exercise shows the simplest way to utilize multiple CPU cores
- It foreshadows the next lecture: Parallelism with OpenMP
- Only use 4 cores here to limit output

Parallelizing Hello World

```
1  #include <iostream>
2  #include <omp.h>
3  int main(int argc,char**argv){
4
5  #pragma omp parallel num_threads(4)
6    {
7      int id=omp_get_thread_num();
8      std::cout<< "Hello␣from␣thread␣"<<id<<"\n";
9    }
10
11   return 0;
12 }
```

Parallelizing Hello World

```
>> icpc -qopenmp helloworld.cpp
>> ./a.out
```

```
Hello from thread Hello from
thread Hello from thread Hello
from thread 3102
```

Parallelizing Hello World

- Weird output because all 4 cores writing to stdout at once
- You can optionally fix this with a "critical region"
- Left as exercise

Vector add

- Next exercise works "out of box"
- First parallelism is demonstrated
- Then vectorization

Vector add

- All examples are in the "practicals" directory
- Much of the code there also measures execution time
- Here are just snippets showing the logic

Vector add

```
1   for(int i=0;i<n;i++){
2     z[i]=x[i]+y[i];
3   }
```

A serial (meaning: single core and no vectorization) vector-add

Vector add

```
1 #pragma omp parallel for num_threads(4)
2   for(int i=0;i<n;i++){
3     z[i]=x[i]+y[i];
4   }
```

Parallel vector add targeting 4 of the 64 Xeon Phi processor cores.

Vector add

```
1 #pragma omp parallel for simd num_threads(4)
2   for(int i=0;i<n;i++){
3     z[i]=x[i]+y[i];
4   }
```

Vectorized vector add targeting the 32 vector lanes of a single Xeon Phi processor core.

Vector Add: Building

```
>> icpc -O2 -no-vec vecadd.cpp
>> icpc -O2 -qopenmp \
 -no-vec vecadd_par.cpp
>> icpc -O2 -qopenmp vecadd_vec.cpp
```

Note: -no-vec added to stop compiler from automatically generating vector instructions as we will compare code with and without vectorization.

Vector Add: Timing Results

	Time	Speedup
One core, No vectorization	1.2s	1.0
Four cores, No vectorization	0.3s	4.0
One core, With vectorization	0.4s	3.0

Discussion

- Theoretical speedup suggests vectorization should do better than four-core
- This is a memory bandwidth issue
  - Cost dominated by memory reads and writes
- After this course you will be able to diagnose and solve such problems

Discussion

- These examples show pitfalls of parallelism and vectorization
- The following lectures will show how to write correct parallel+vectorized code
  - Without sacrificing portability or performance

Going Forward

- Xeon Phi processor offers accelerator performance with easier development
- Modern software tools enable this portability
- This course will teach the essentials of doing so