



Introdução à Programação Paralela em Clusters Heterogêneos

Silvio Stanzani, Jefferson Fialho , Raphael Cóbe , Rogério Iope
Núcleo de Computação Científica – UNESP

[silvio, fialho, rmcobe, rogerio]@ncc.unesp.br

ERAD-SP 2018

UNESP Center for Scientific Computing

- Consolidates scientific computing resources for São Paulo State University (UNESP) researchers
 - It mainly uses Grid computing paradigm
- Main users
 - UNESP researchers, students, and software developers
 - SPRACE (São Paulo Research and Analysis Center)
 - physicists and students
- Caltech, Fermilab, CERN
- São Paulo CMS Tier-2 Facility

VECPAR

HOME

COMMITTEES

CALL

REGISTRATION

VENUE

PROGRAMME

KEYNOTE SPEAKERS

SPONSORS

VECPAR 2018

13th International Meeting on
High Performance Computing for Computational Science

17-19 SEPTEMBER 2018
SÃO PEDRO – SP – BRAZIL

<https://vecpar2018.ncc.unesp.br>

13th International Meeting on High Performance
Computing for Computational Science

Agenda

- Parallel Architectures
- Parallelism Levels
- Xeon and Xeon Phi
- Heterogeneous Cluster @ NCC
- Offload Programming
- **Optimization Examples**
- Hands-on

Agenda

- Parallel Architectures
- Parallelism Levels
- Xeon and Xeon Phi
- Heterogeneous Cluster @ NCC
- Offload Programming
- Optimization Examples
- Hands-on

Agenda

- **Parallel Architectures**
- Parallelism Levels
- Xeon and Xeon Phi
- Heterogeneous Cluster @ NCC
- Offload Programming
- Optimization Examples
- Hands-on

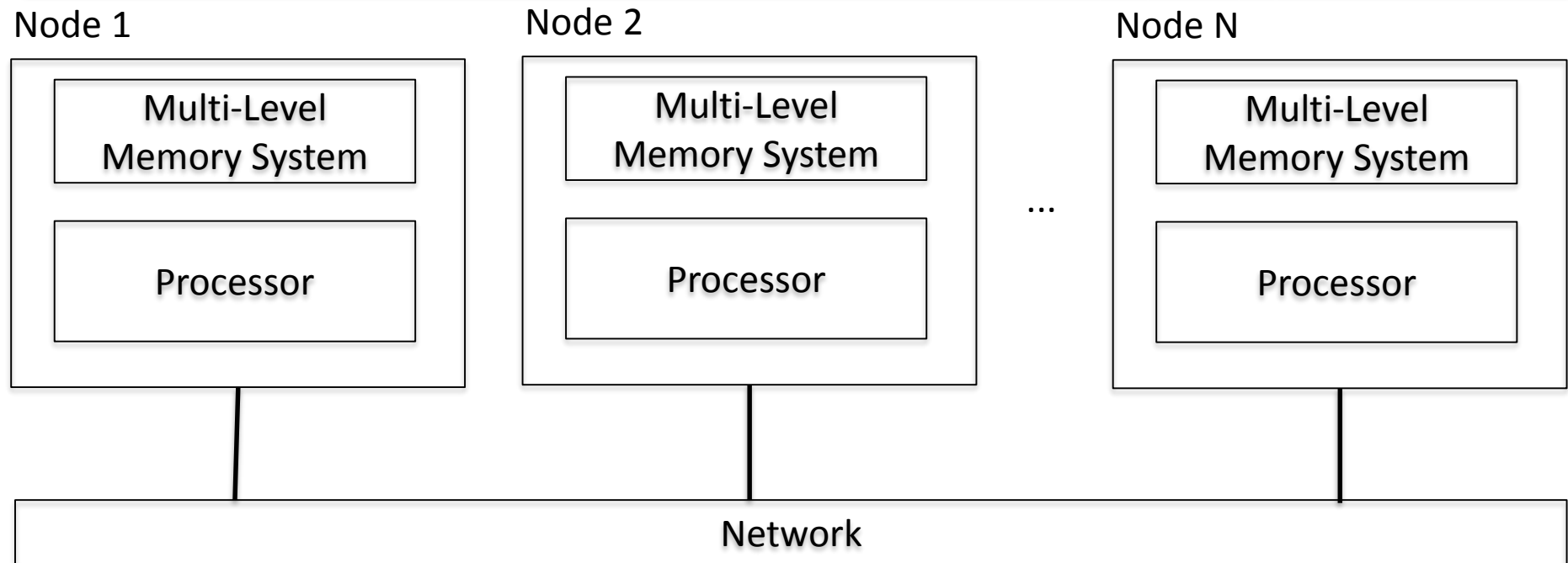
Parallel Processing

- ❑ A parallel computer is a computer system that uses multiple processing elements simultaneously in a cooperative manner to solve a computational problem
- ❑ Parallel processing includes techniques and technologies that make it possible to compute in parallel
 - ❑ Hardware, networks, operating systems, parallel libraries, languages, compilers, algorithms, tools, ...
- ❑ Parallel computing is an evolution of serial computing
 - ❑ Parallelism is natural
 - ❑ Computing problems differ in level / type of parallelism

Parallelism

- Concurrency = Different units of work that can be executed in parallel
 - Ex: loop iterations
- Parallelism = concurrency + parallel hardware
 - Find concurrent execution opportunities
 - Develop application to execute in parallel
 - Run application on parallel hardware
- Motivations for parallelism
 - Faster time to solution
 - Solve bigger computing problems
 - Effective use of machine resources
- Serial machines have inherent limitations
 - Processor speed, memory bottlenecks, ...

Homogeneous Parallel Architectures

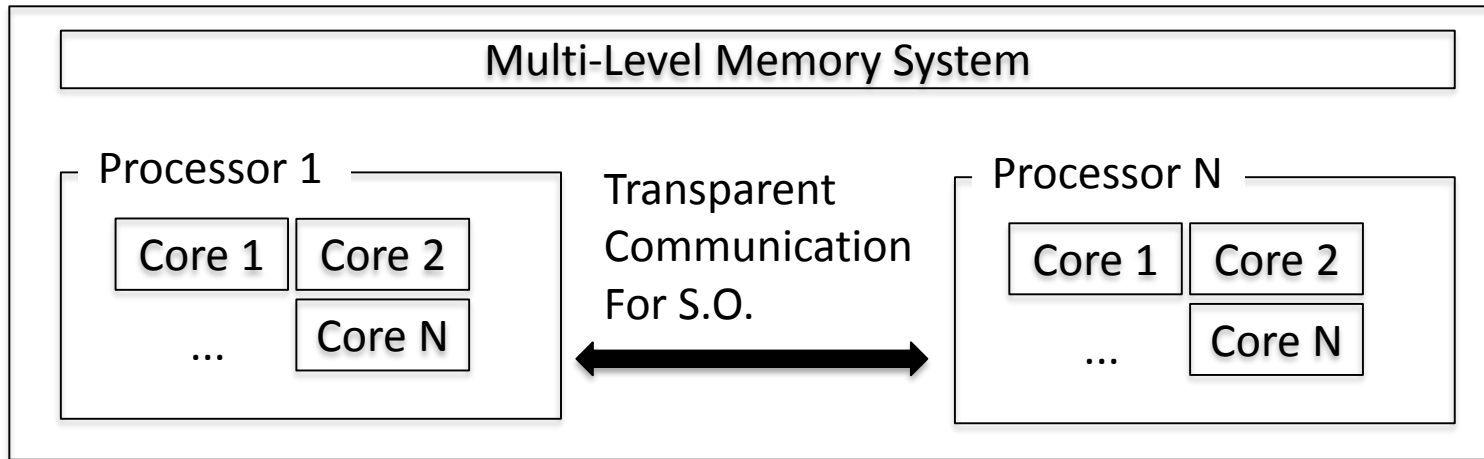


- **First initiatives:**

- Identical Nodes interconnected;
- Parallel architectures is deployed as a group of serial computers;

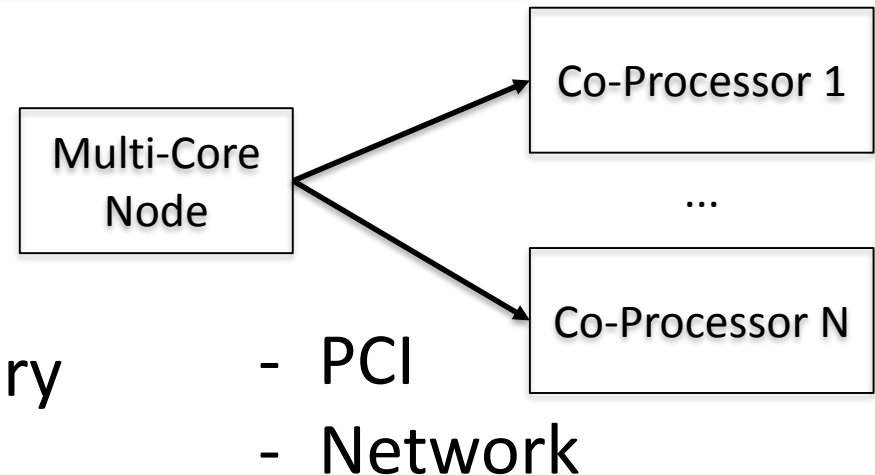
Heterogeneous Parallel Architectures

Multi-Core Node



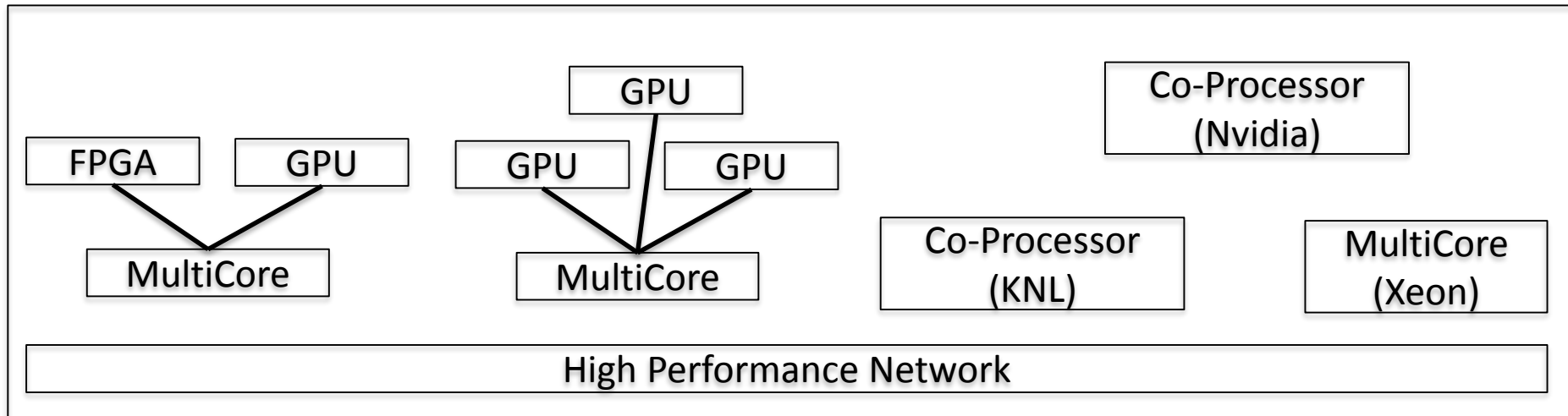
Multi-Core Architecture

- Multi Processors
- Multiple Cores
- **NUMA** (Non-Uniform Memory Access)



Heterogeneous Parallel Architectures

- Several families and architectures:
- Hybrid Parallel nodes
 - Coprocessors and accelerators;
- Multi-level parallelism:
 - Processing core;
 - Chip multiprocessor;
 - Computing node;
 - Computing cluster;



Agenda

- Parallel Architectures
- **Parallelism Levels**
- Xeon and Xeon Phi
- Heterogeneous Cluster @ NCC
- Offload Programming
- Optimization Examples
- Hands-on

Exploiting the parallel universe

Instruction Level Parallelism

- Single thread (ST) performance
- Automatically exposed by HW/tools
- Effectively limited to a few instructions

Data Level Parallelism

- Single thread (ST) performance
- Exposed by tools and programming models
- Operate on 4/8/16 elements at a time

Vectorization

Task Level Parallelism

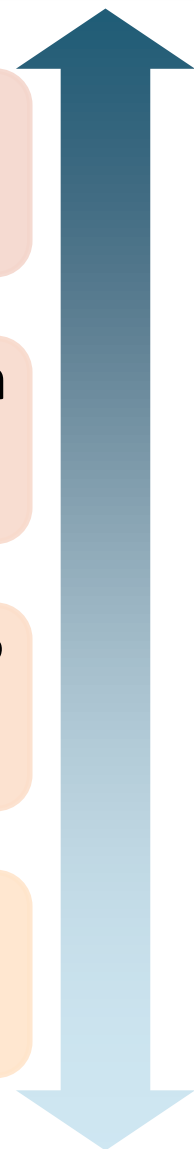
- Multi thread/task (MT) performance
- Exposed by programming models
- Execute tens/hundreds/thousands task concurrently

OpenMP

Process Level Parallelism

- Multi Process (MP) performance
- Exposed by programming models
- Execute tens/hundreds/thousands of process concurrently across several nodes

MPI

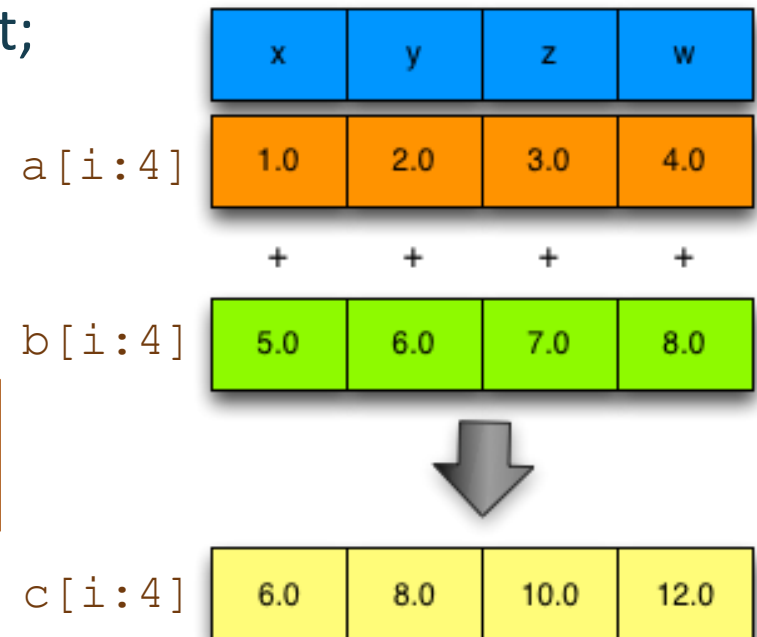


Vectorization

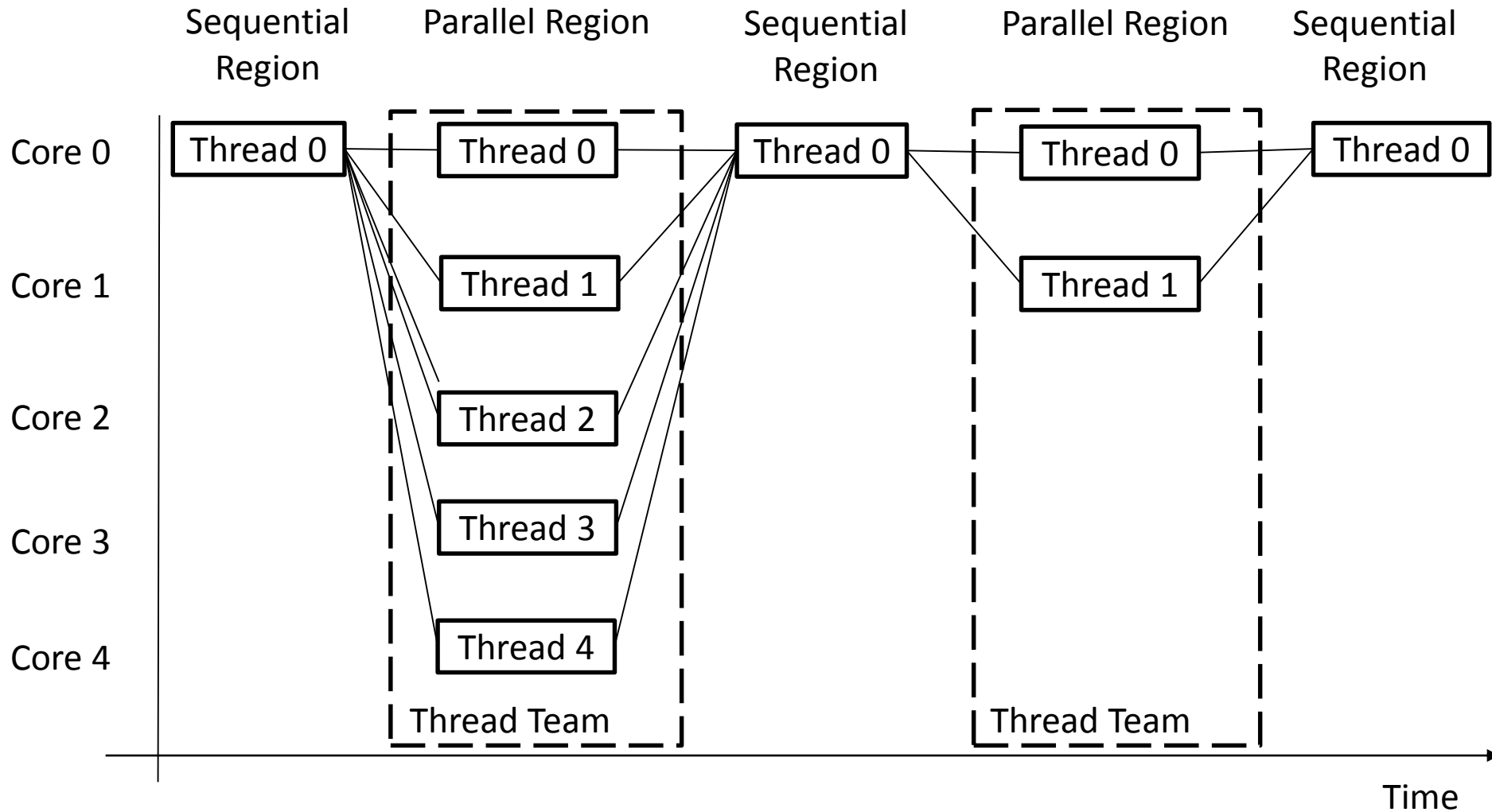
- Vectorization
 - Loading data into cache accordingly;
 - Store elements on SIMD registers or vectors;
 - Apply the same operation to a set of Data at the same time;
 - Iterations need to be independent;
 - Usually on inner loops.

Scalar loop

```
for (int i = 0; i < N; i++)  
    c[i] = a[i] + b[i];
```

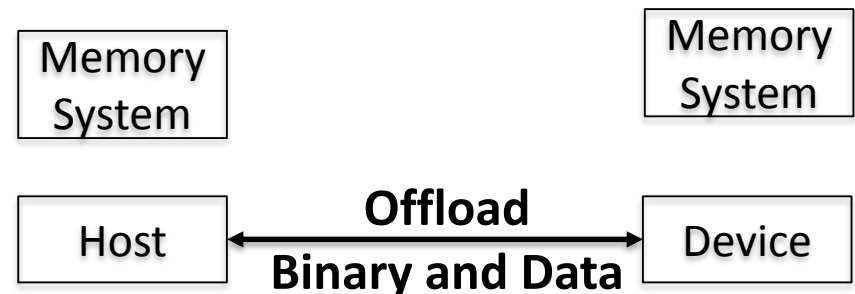


OpenMP



Offload

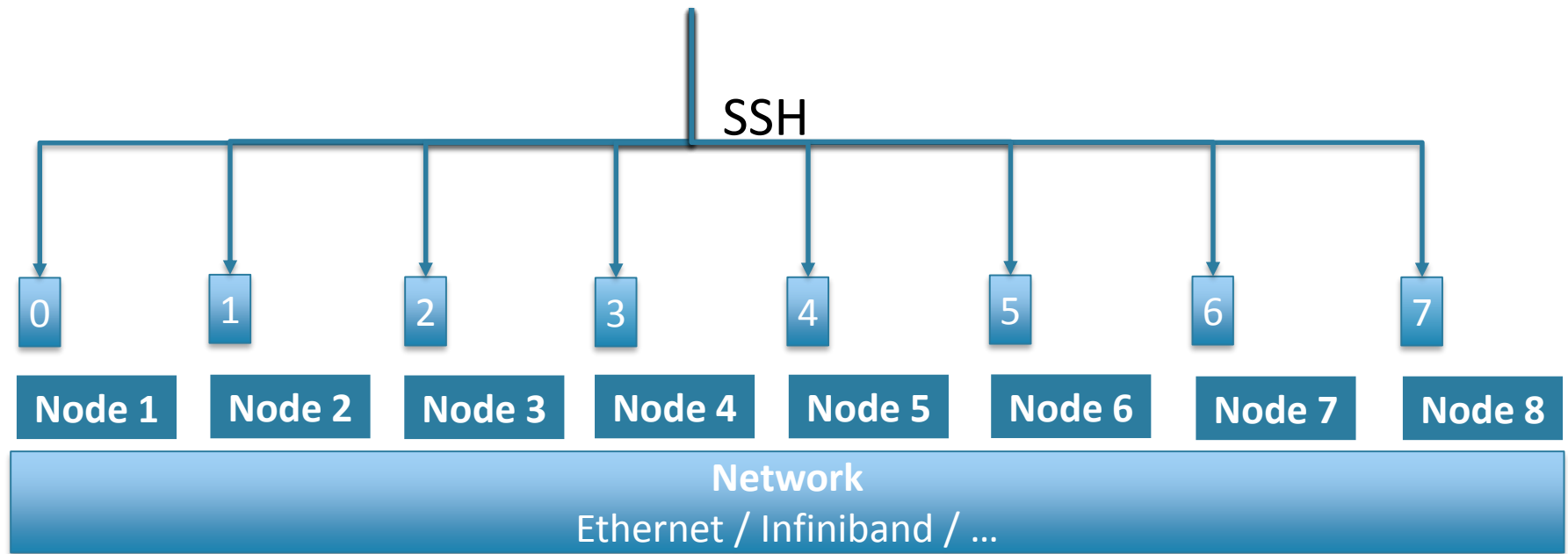
- Regions of code is marked to be executed in a device attached to the main processor
 - Communication using PCI Express or Network
- Main Processor and Device has independent Memory system
 - Developer need to control data transfer of data structures needed by application
 - Can Present Delay



Message Passing Interface (MPI)

Execution of a MPI application across several nodes using SSH (Secure Shell)

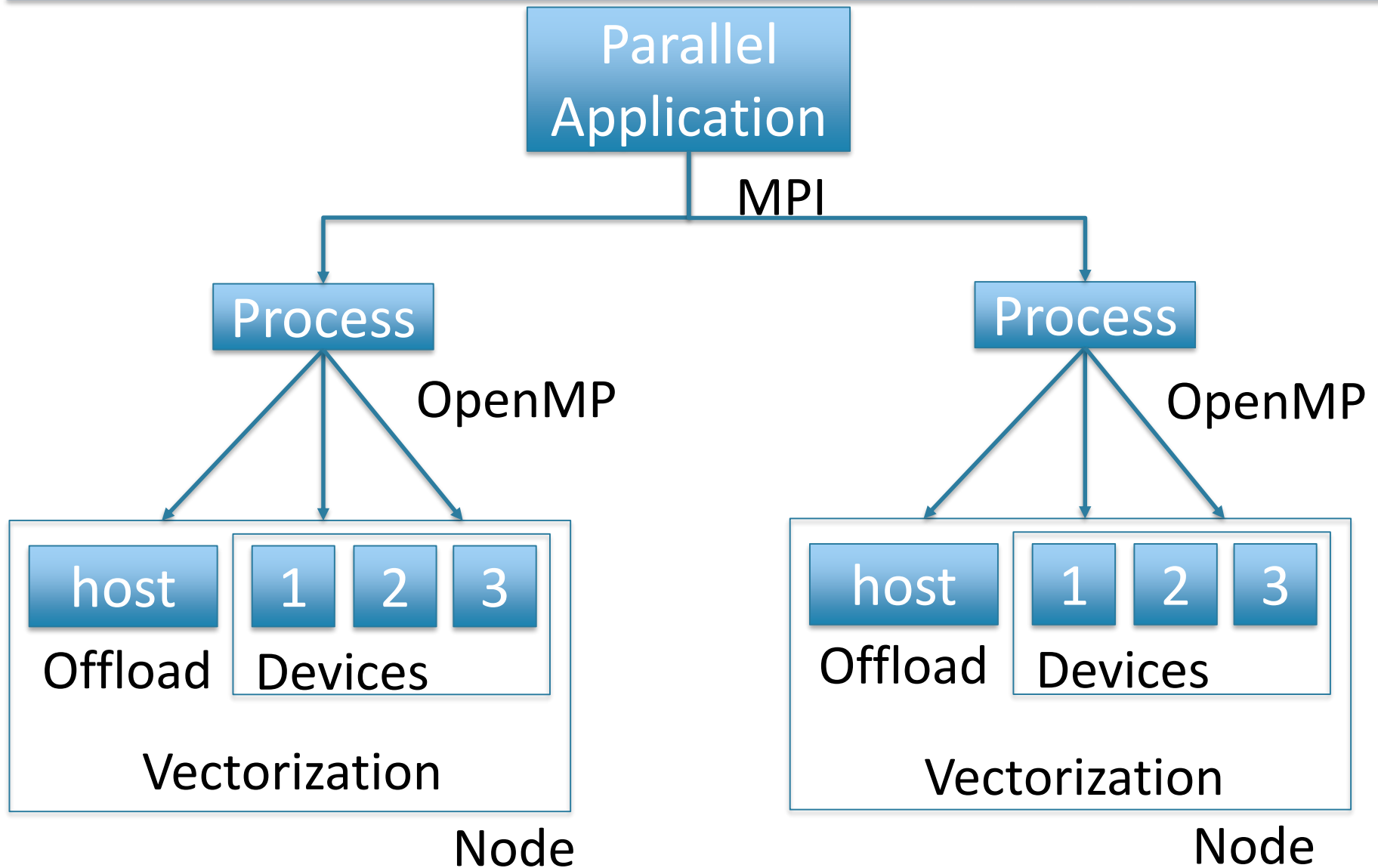
```
mpirun -n 8 ./myapp
```



Infiniband: computer-networking communications standard used in high-performance computing

- Very high throughput;
- Very low latency.

Combining Levels

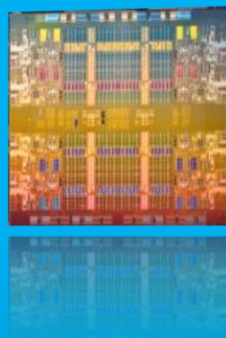


Agenda

- Parallel Architectures
- Parallelism Levels
- **Xeon and Xeon Phi**
- Heterogeneous Cluster @ NCC
- Offload Programming
- Optimization Examples
- Hands-on

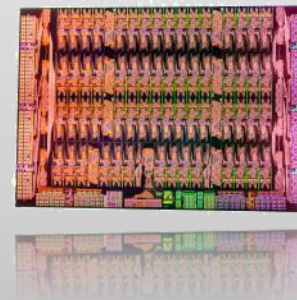
Intel Xeon and Intel® Xeon Phi™ Overview

Intel® Multicore Architecture



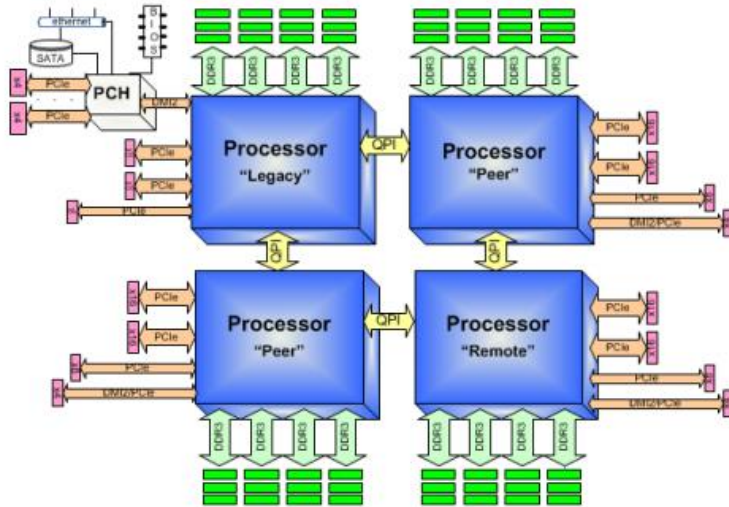
- ❖ Foundation of HPC Performance
- ❖ Suited for full scope of workloads
- ❖ Focus on fast single core/thread performance with “moderate” number of cores

Intel® Many Integrated Core Architecture

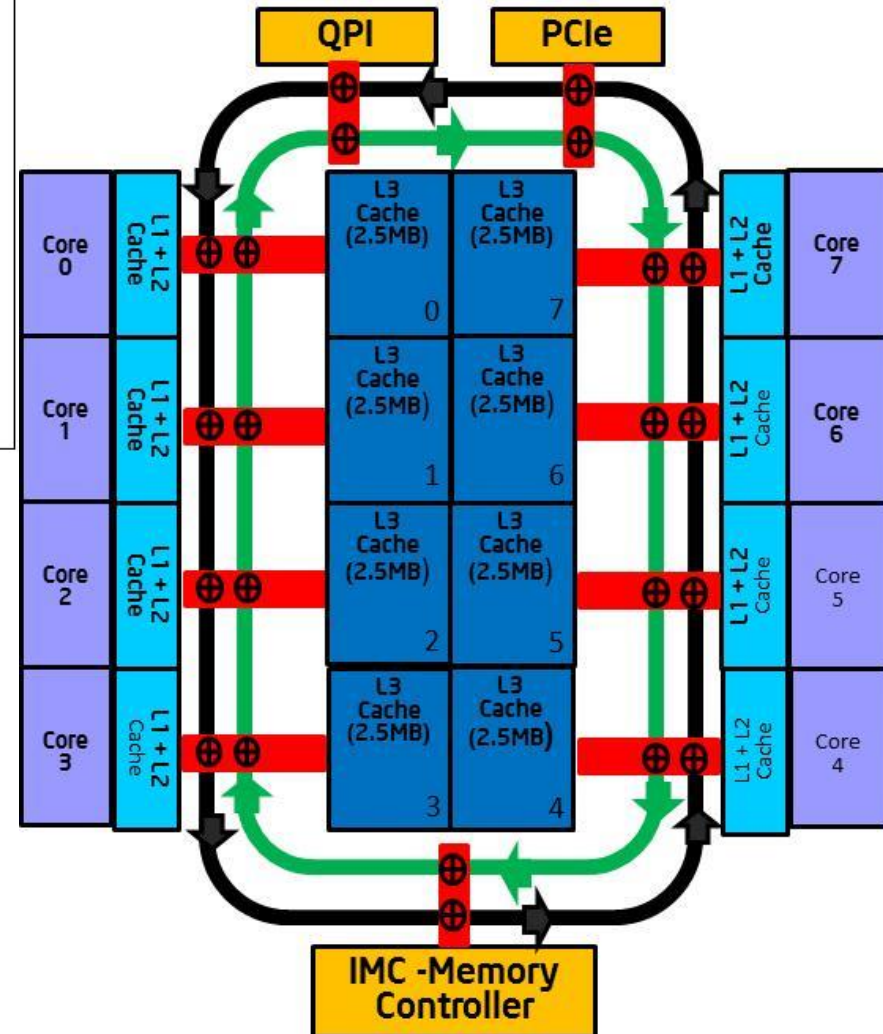


- ❖ Performance and performance/watt optimized for highly parallelized compute workloads
- ❖ IA extension to Manycore
- ❖ Many cores/threads with wide SIMD

Intel Xeon Architecture Overview



- Socket: mechanical component that provides mechanical and electrical connections between a microprocessor and a printed circuit board (PCB).
- QPI (Intel QuickPath Interconnect): high speed, packetized, point-to-point interconnection, that stitch together processors in distributed shared memory and integrated I/O platform architecture.



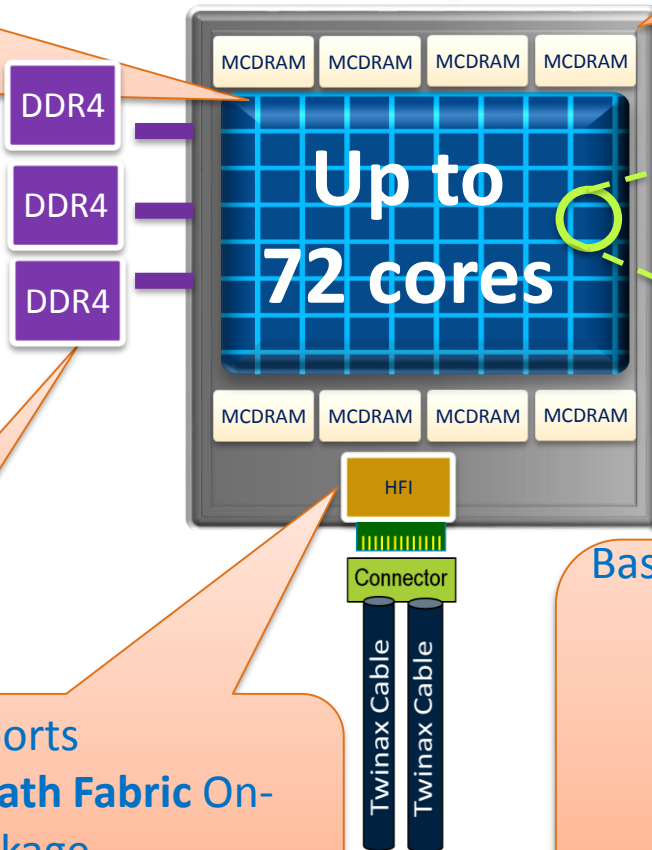
Knights Landing (KNL)

Over 3 TF DP peak

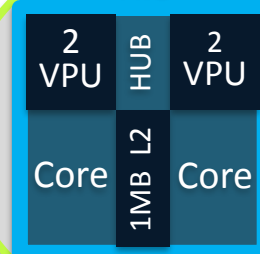
Full Xeon ISA compatibility through AVX-512
~3x single-thread vs. compared to Knights
Corner

Up to 16GB high-bandwidth on-package
memory (MCDRAM)
Exposed as NUMA node
~500 GB/s sustained BW

Up to 72
cores
2D mesh
architecture



Tile



2x 512b VPU
per core
(Vector
Processing
Units)

6 channels
DDR4
Up to
384GB

2 ports
Intel Omni-Path Fabric On-
package
50 GB/s bi-directional

Based on Intel® Atom Silvermont processor with
many HPC enhancements
Deep out-of-order buffers
Gather/scatter in hardware
Improved branch prediction
4 threads/core
High cache bandwidth

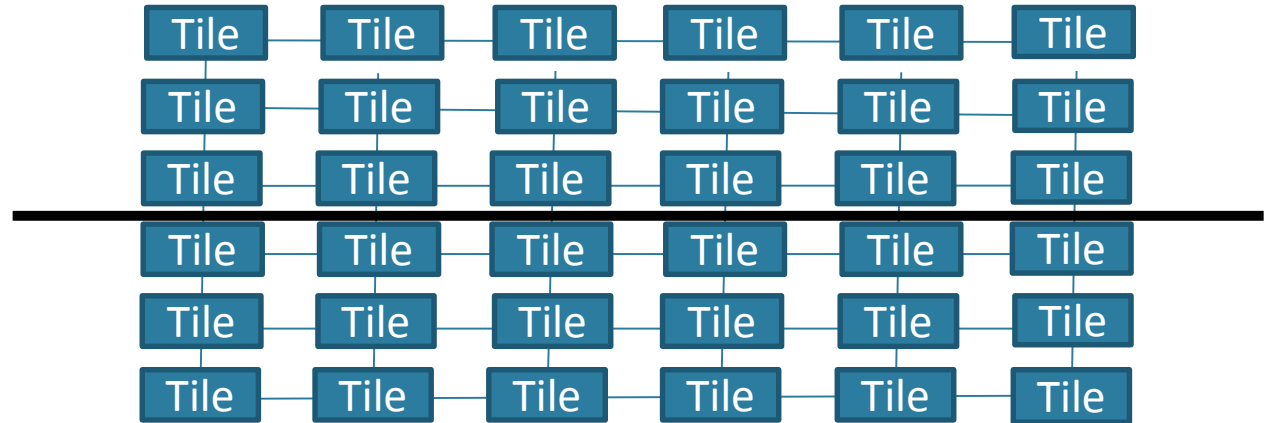
Cluster modes

One single space address

Hemisphere:

the tiles are divided into two parts called hemisphere

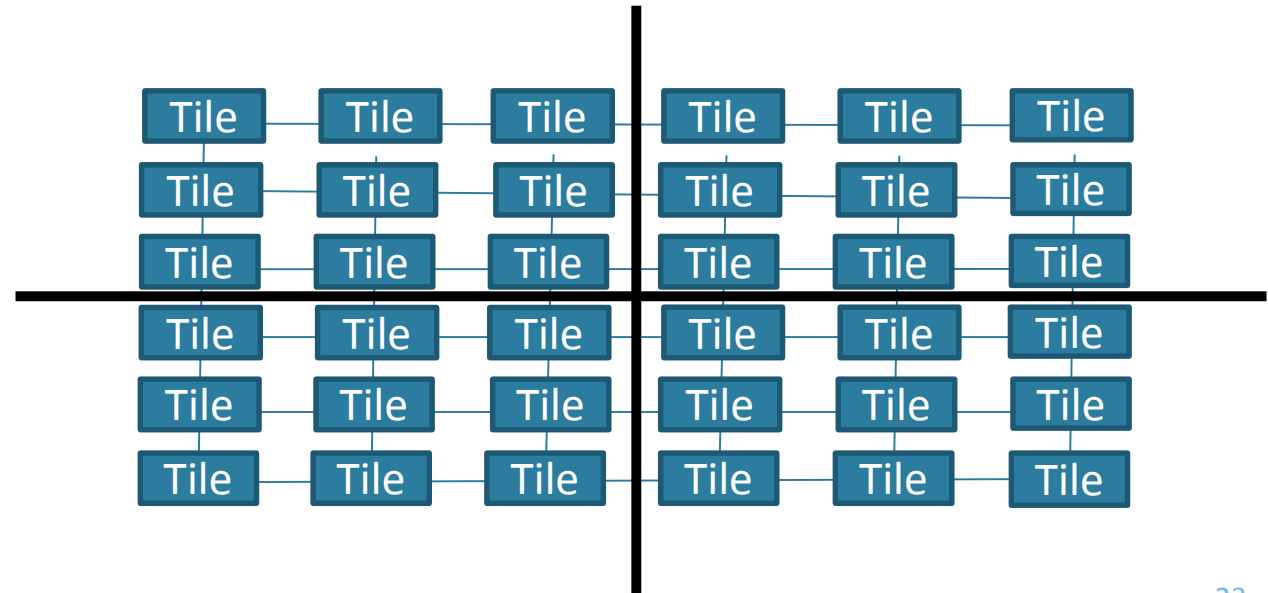
Node 0



Quadrant:

tiles are divided into two parts called hemisphere or into four parts called quadrants

Node 0

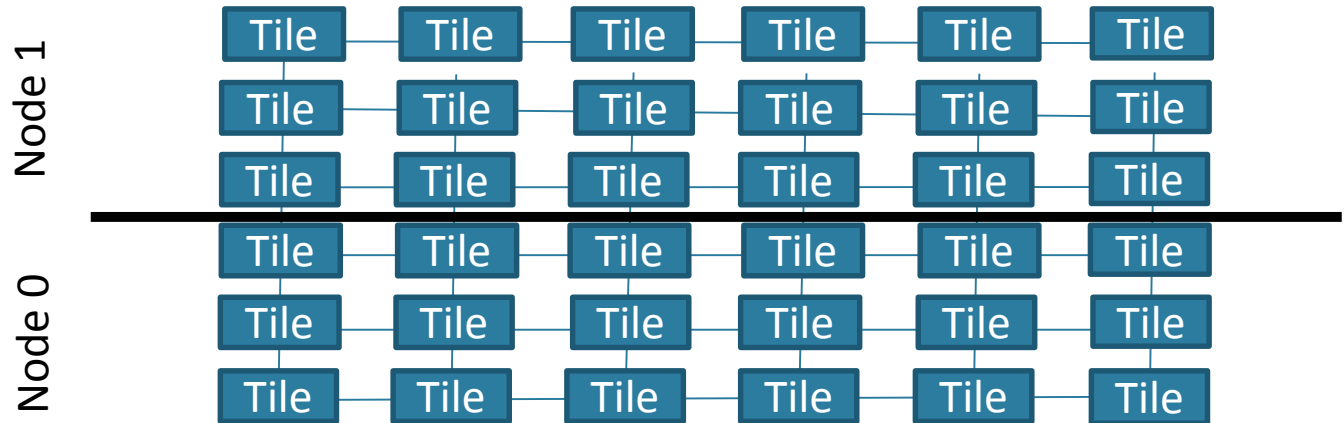


Cluster modes

Cache data are isolated in each sub numa domain

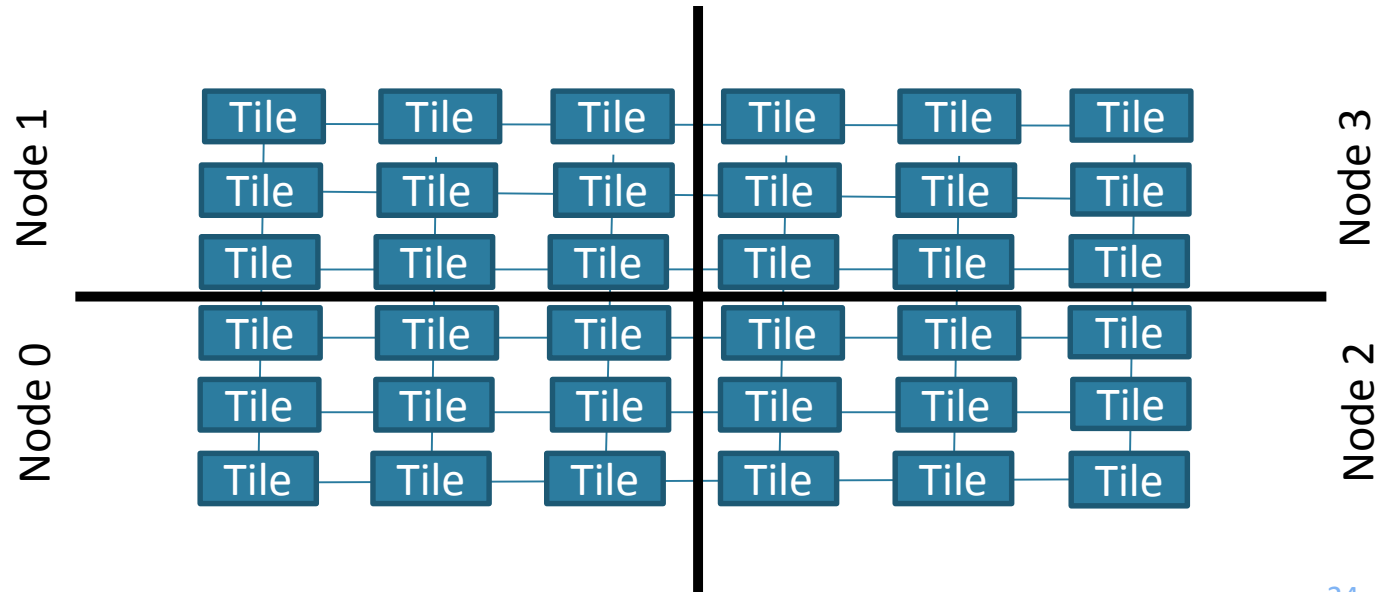
SNC-2:

the tiles are
divided into two
Numa Nodes

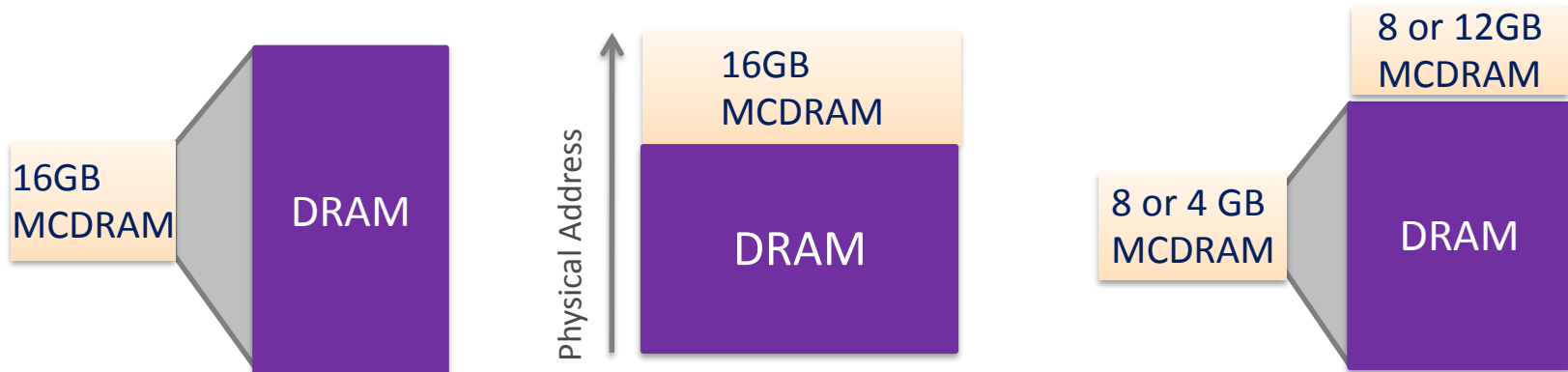


SNC-4:

the tiles are
divided into two
Numa Nodes



Integrated On-Package Memory Usage Models

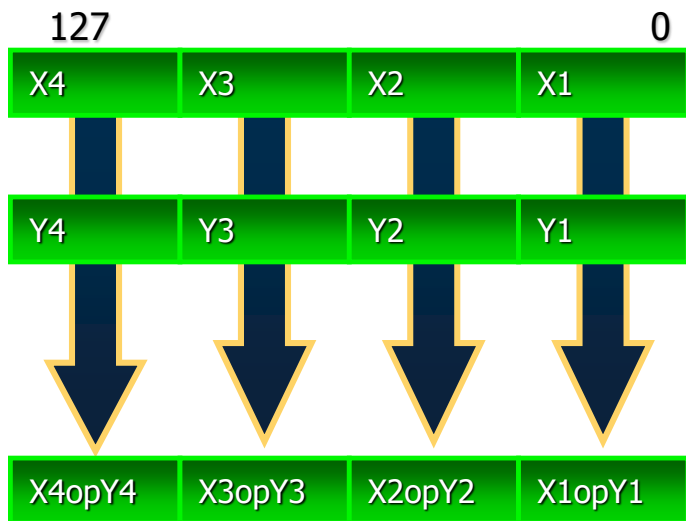


Split Options:
25/75% or 50/50%

Cache Model	Flat Model	Hybrid Model
Hardware automatically manages the MCDRAM as a “L3 cache” between CPU and ext DDR memory	Manually manage how the app uses the integrated on-package memory and external DDR for peak perf	Harness the benefits of both Cache and Flat models by segmenting the integrated on-package memory
<ul style="list-style-type: none"> ▪ App and/or data set is very large and will not fit into MCDRAM ▪ Unknown or unstructured memory access behavior 	<ul style="list-style-type: none"> ▪ App or portion of an app or data set that can be, or is needed to be “locked” into MCDRAM so it doesn’t get flushed out 	<ul style="list-style-type: none"> ▪ Need to “lock” in a relatively small portion of an app or data set via the Flat model ▪ Remaining MCDRAM can then be configured as Cache

Vectorial processing Intel® Architecture

MMX	Intel® SSE	Intel® AVX / AVX2	Intel® MIC / AVX-512
Vector size: 64bit Data types: 8, 16 and 32 bit integers VL: 2,4,8	Vector size: 128bit Data types: 8,16,32,64 bit integers 32 and 64bit float VL: 2,4,8,16	Vector size: 256bit Data types: 32 and 64 bit floats VL: 4, 8, 16 Sample: Xi, Yi 32 bit int or float	Vector size: 512bit Data types: 32 and 64 bit integers 32 and 64bit float VL: 8,16

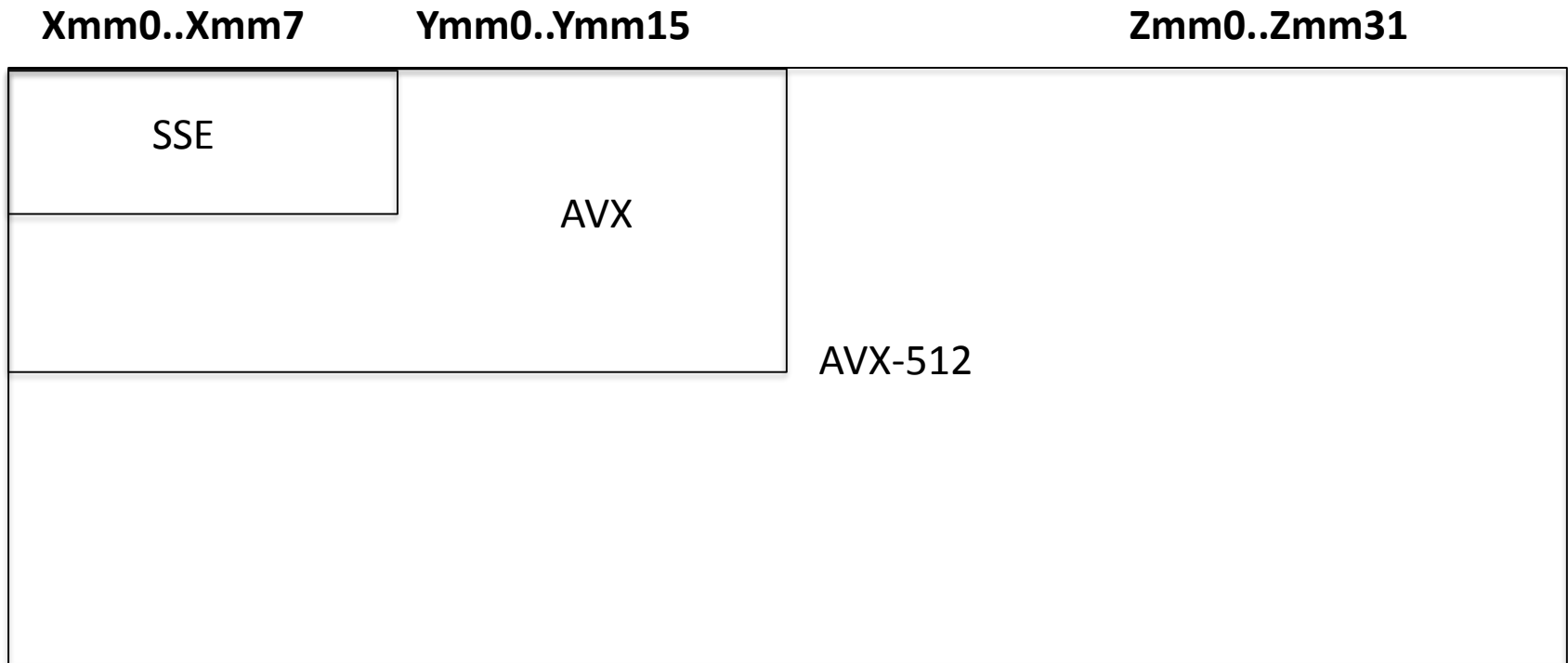


AVX -512

Exponential & Reciprocal Instructions (ERI)
Prefetch Instructions (PFI)
Foundation instructions (FI)
Conflict Detection Instructions (CDI)
Byte & Word Instructions (BWI)
Double-/Quad-word Instructions (DQI)
Vector Length Extensions (VLE)

AVX-512 - Foundation

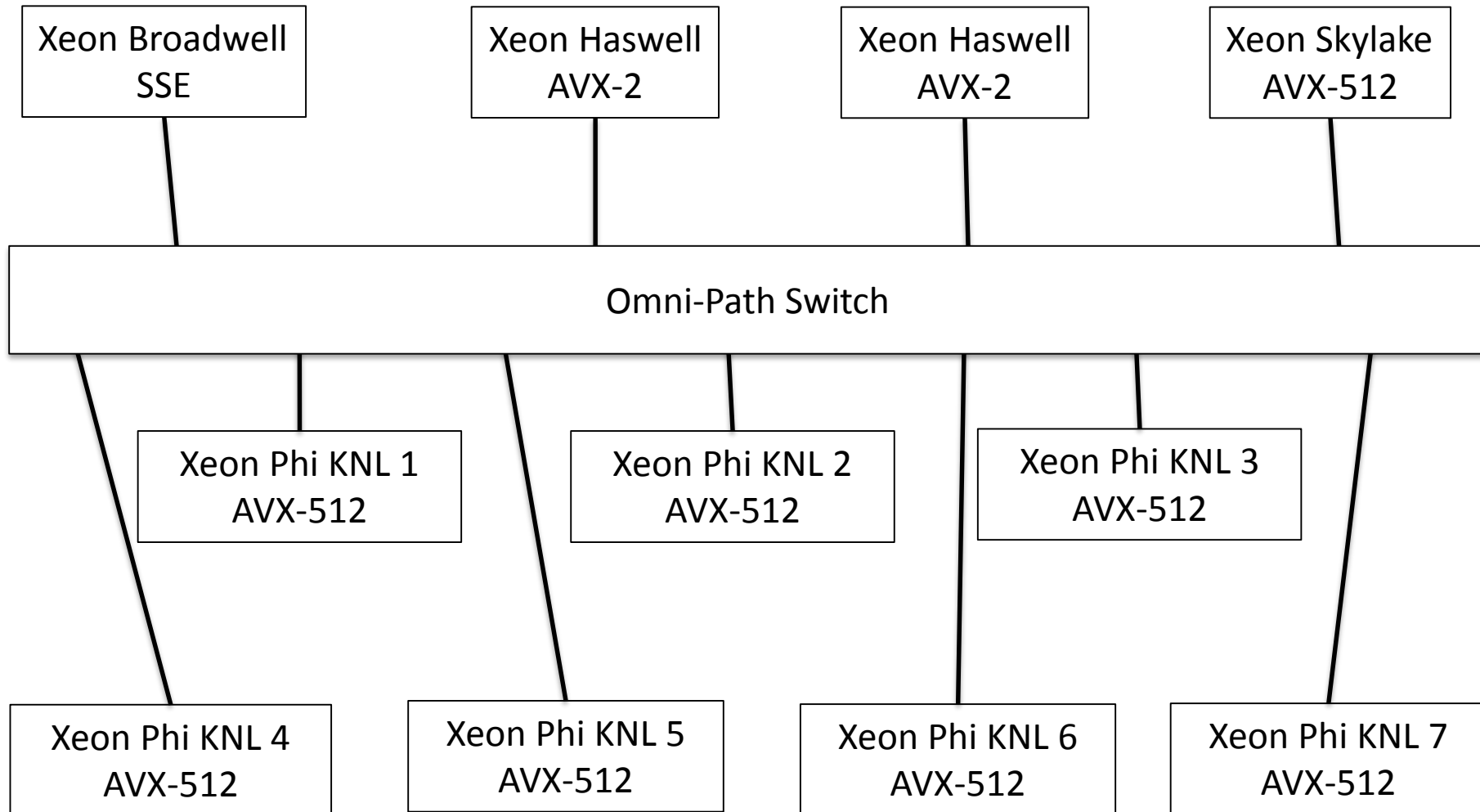
- Implements new version of AVX-2 functions using 512-bit registers (ZMM);



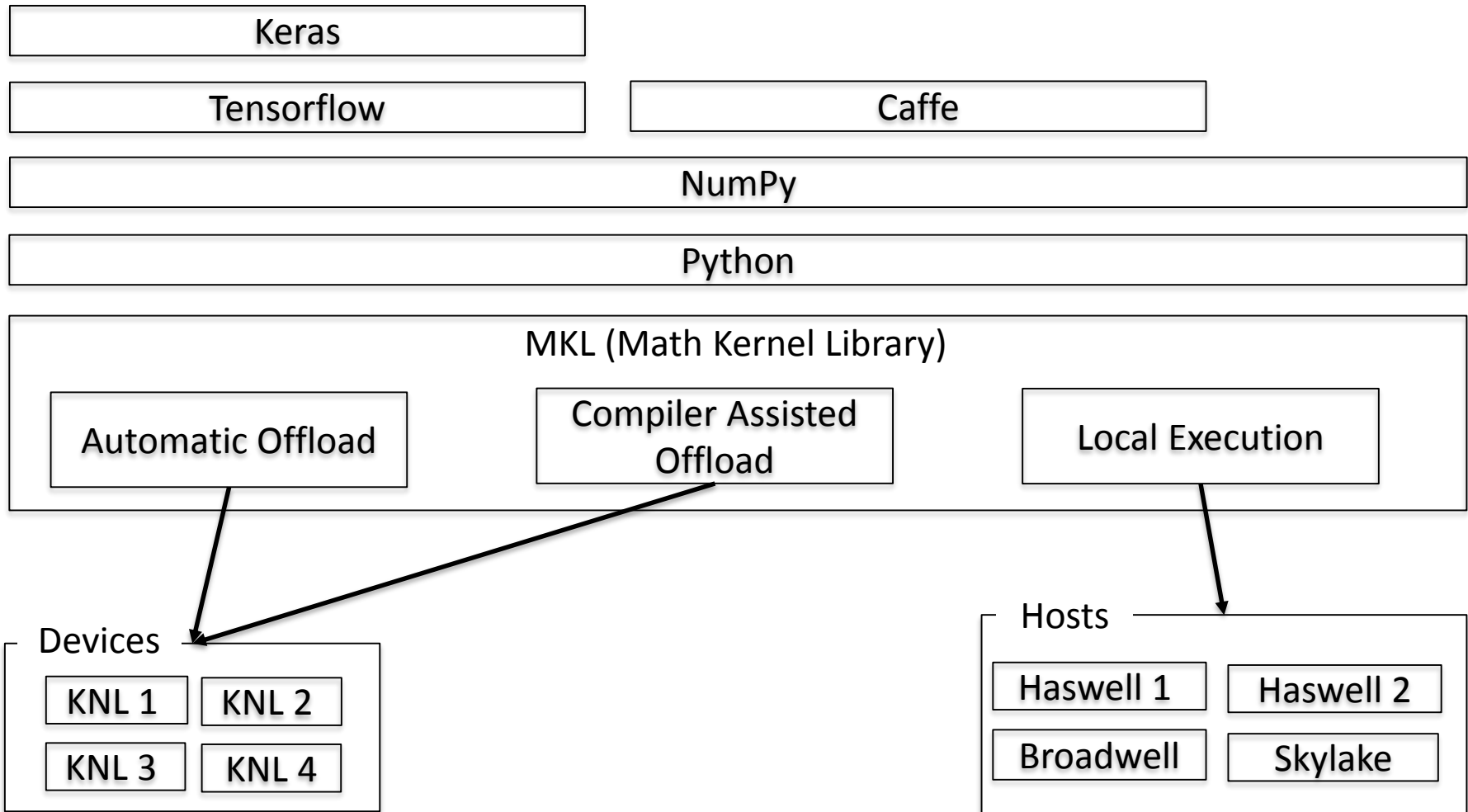
Agenda

- Parallel Architectures
- Parallelism Levels
- Xeon and Xeon Phi
- **Heterogeneous Cluster @ NCC**
- Offload Programming
- Optimization Examples
- Hands-on

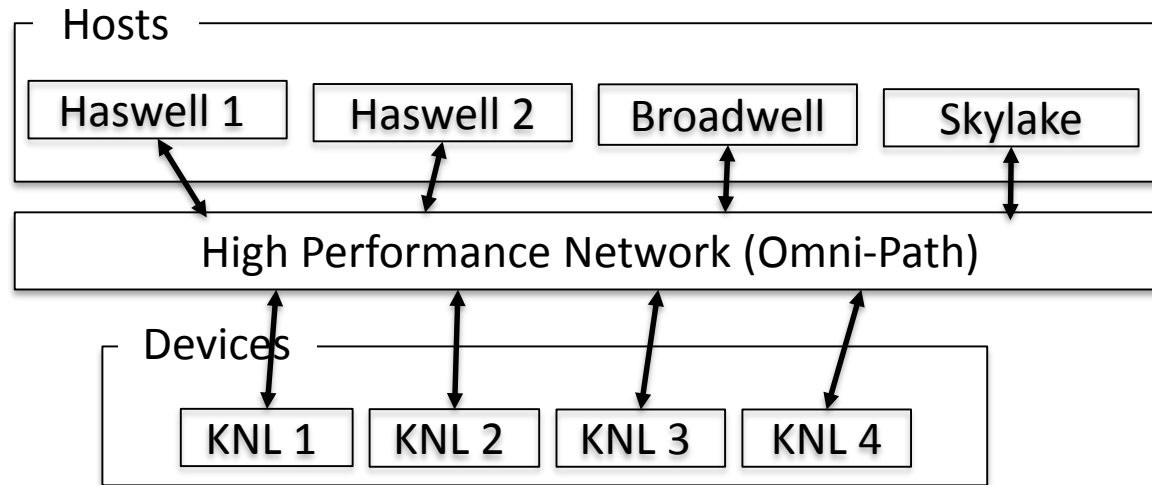
Heterogeneous Cluster @ NCC



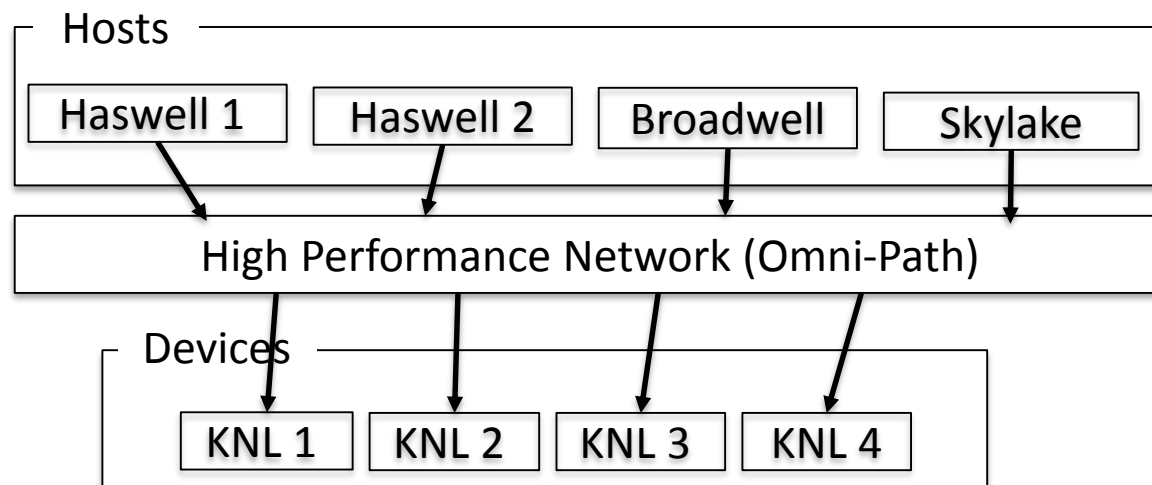
Parallelism Model - MKL



Parallelism Model – MPI x Offload



- **MPI (Message Passing Interface)**
 - High Performance Communication using MPI Functions
 - Every node send and receive messages



- **Offload**
 - One Host node can offload code to one or more devices
 - Hosts only send code regions for execution
 - Devices only execute code regions

Offload Over Fabric

- Offload over Fabric (OOF) enables offloading from servers with Intel Xeon processor to servers with Intel Xeon Phi x200 processor within a high-speed network, such as Intel® Omni-Path Fabric.
- *OFFLOAD_NODES* environment variable define nodes available for the offloading operation from the offload host.
 - *OFFLOAD_NODES* is a comma-separated list of nodes' names or IP addresses.

Offload Over Fabric

```
icc -qopenmp testHW.c -o testHW
```

```
export OFFLOAD_NODES=10.0.0.5  
./testHW
```

Running on host: phi01.ncc.unesp.br

Running on target: phi05

```
export OFFLOAD_NODES=10.0.0.6  
./testHW
```

Running on host: phi01.ncc.unesp.br

Running on target: phi06

Agenda

- Parallel Architectures
- Parallelism Levels
- Xeon and Xeon Phi
- Heterogeneous Cluster @ NCC
- **Offload Programming**
- Optimization Examples
- Hands-on

OpenMP 4.0 Offload

- **target:** transfers the control flow to the target device
 - Transfer is sequential and synchronous
 - Transfer clauses control data flow
- **target data:** creates a scoped device data environment
 - Does not include a transfer of control
 - Transfer clauses control data flow
 - The device data environment is valid through the lifetime of the target data region
- **target update:** request data transfers from within a target data region
- **omp declare target:** creates a structured-block of functions that can be offloaded.

OpenMP 4.0 Offload Report

- **OFFLOAD REPORT:**
 - Measures the amount of time it takes to execute an offload region of code;
 - Measures the amount of data transferred during the execution of the offload region;
 - Turn on the report: `export OFFLOAD_REPORT=2`
- **[Var]** The name of a variable transferred and the direction(s) of transfer.
- **[CPU Time]** The total time measured for that offload directive on the host.
- **[MIC Time]** The total time measured for executing the offload on the target.
- **[CPU->MIC Data]** The number of bytes of data transferred from the host to the target.
- **[MIC->CPU Data]** The number of bytes of data transferred from the target to the host.

Pragma omp declare target

- Creates a structured-block of functions that can be offloaded.
- Syntax
 - `#pragma omp declare target [clause[[,] clause],...]`
declaration of functions
 - `#pragma omp end declare target`

Pragma omp target

- Transfer control [and data] from host to device
- Syntax
 - `#pragma omp target [data] [clause[,] clause],...]`
structured-block
- Clauses
 - `device(scalar-integer-expression) :`
 - ❑ `device to offload code;`
 - `map(alloc | to | from | tofrom: list) :`
 - ❑ `map variables to device;`
 - `if(scalar-expr) :`
 - ❑ `test an expression before offload:`
 - o True executes on device;
 - o False executes on host;
 - `Nowait`
 - ❑ `Execute the data transfer defined in map asynchronously;`

Pragma omp target

- Map clauses:
 - alloc : allocate memory on device;
 - to : transfer a variable from host to device;
 - from : transfer a variable from device to host;
 - tofrom :
 - ❑ transfer a variable from host to device before start execution;
 - ❑ transfer a variable from device to host after finish execution;

Offloading - omp target

```
Int main() {  
  Printf("begin");  
  int N=25;  
  int b =2;  
  int l = 0;
```

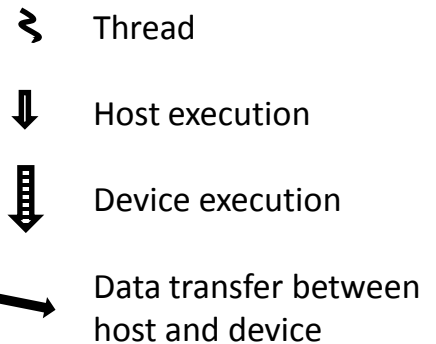
*Offload:
Copy variable:
N,b,l and **a** to device*

```
#pragma omp target map(N,b,l,a)  
{  
  for (i=0; i<N; i++) a[i] = 2;  
  for (i=0; i<N; i++) a[i] = a[i] + b;  
}
```

```
for (i=0; i<N; i++)  
  printf("%d",a[i]);  
...  
return(0);  
}
```

Host

Device



Pragma omp target example

```
#pragma omp target device(0) map(a[0:NUM][0:NUM])  
map(b[0:NUM][0:NUM]) map(c[0:NUM][0:NUM])  
{  
    #pragma omp parallel for collapse (2)  
    for(i=0; i<msize; i++) {  
        for(k=0; k<msize; k++) {  
            #pragma omp simd  
            for(j=0; j<msize; j++) {  
                c[i][j] = c[i][j] + a[i][k] * b[k][j];  
            }  
        }  
    }  
}
```

Agenda

- Parallel Architectures
- Parallelism Levels
- Xeon and Xeon Phi
- Heterogeneous Cluster @ NCC
- Offload Programming
- **Optimization Examples**
- Hands-on

N-Body Simulation

- An N-body simulation [1] aims to approximate the motion of particles that interact with each other according to some physical force;
- Used to study the movement of bodies such as satellites, planets, stars, galaxies, etc., which interact with each other according to the gravitational force;
- Newton's second law of motion can be used in a N-body simulation to define the bodies' movement.

[1] AARSETH, S. J. Gravitational n-body simulations. [S.l.]: Cambridge University Press, 2003. Cambridge Books Online.

N-Body Algorithm

- Bodies struct:
 - 3 matrix represents velocity (x,y and z)
 - 3 matrix represents position (x,y and z)
 - 1 matrix represent mass
- A loop calculate temporal steps:
 - At each temporal step new velocity and position are calculated to all bodies according to a function that implements Newton's second law of motion

N-Body - Parallel version (host only)

```
function Newton(step)
{
    #pragma omp for
    for each body[x] {
        #pragma omp simd
        for each body[y]
            calc force exerted from body[y] to body[x];
        calc new velocity of body[x]
    }
    #pragma omp simd
    for each body[x]
        calc new position of body[x]
}

Main() {
    for each temporal step
        Newton(step)
}
```

N-Body - Parallel version (Load balancing)

- The temporal step loop remains sequential
- The N-bodies are divided among host and devices to be executed using Newton
- OpenMP offload pragmas are used to
 - Newton function offloading to devices
 - Transfer data (bodies) between host and devices

N-Body - Parallel version (Load balancing)

```
function Newton(step, begin_body, end_body, deviceId)
{
    #pragma omp target device (deviceId) {
        #pragma omp for
        for each body[x] from subset(begin_body, end_body) {
            #pragma omp simd
            for each body[y] from subset(begin_body, end_body)
                calc force exerted from body[y] to body[x];
            calc new velocity of body[x]
        }
        #pragma omp simd
        for each body[x]
            calc new position of body[x]
    }
}
```

N-Body - Parallel version (Load balancing)

for each temporal step

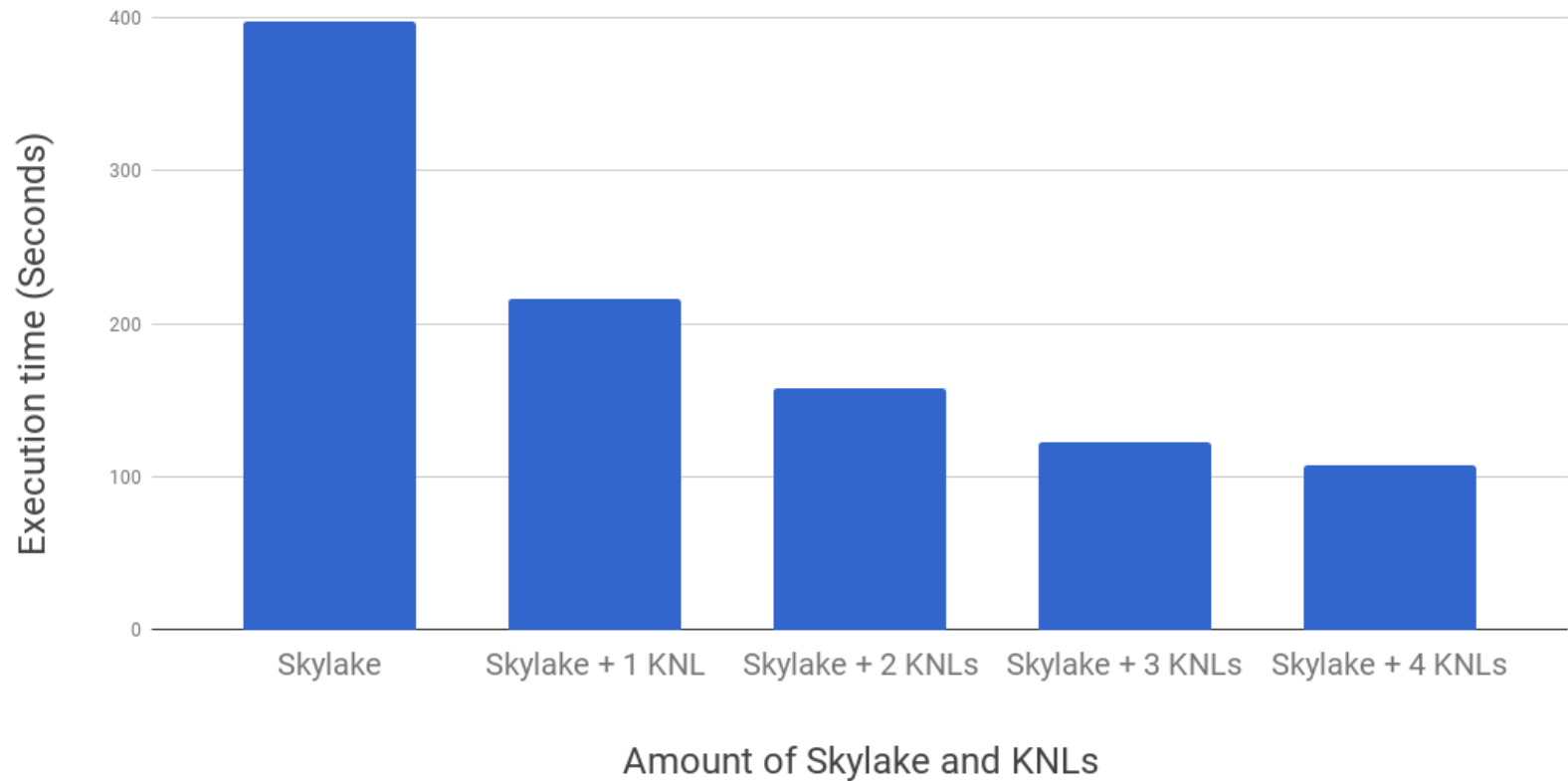
Divide the amount of bodies among host and devices;

```
#pragma omp parallel  
{  
    #pragma omp target data device ( tid ) to(bodies[begin_body:  
end_body])  
    {  
        Newton(step, begin_body, end_body, deviceId)  
        #pragma omp target update device ( tid ) (from:bodies)  
        #pragma omp barrier  
        #pragma omp target data device ( tid )  
to(bodies[begin_body: end_body])  
    }  
}
```


N-Body

N-Body Evaluation using offload over fabric (Xeon - Xeon Phi and Omni-Path)

Amount of Particles in Simulation: 30000



Agenda

- Parallel Architectures
- Parallelism Levels
- Xeon and Xeon Phi
- Heterogeneous Cluster @ NCC
- Offload Programming
- Optimization Examples
- **Hands-on**

VECPAR

HOME

COMMITTEES

CALL

REGISTRATION

VENUE

PROGRAMME

KEYNOTE SPEAKERS

SPONSORS

VECPAR 2018

13th International Meeting on
High Performance Computing for Computational Science

17-19 SEPTEMBER 2018
SÃO PEDRO – SP – BRAZIL

<https://vecpar2018.ncc.unesp.br>

13th International Meeting on High Performance
Computing for Computational Science