Multithreading and Vectorization on Intel® Xeon™ and Intel® Xeon Phi™ architectures using OpenMP

Silvio Stanzani , Raphael Cóbe , Rogério Iope

UNESP - Núcleo de Computação Científica

silvio@ncc.unesp.br , rmcobe@ncc.unesp.br , rogerio@ncc.unesp.br

VECPAR 2016

12th International Meeting on
High Performance Computing for Computational Science

Agenda

- NCC/UNESP Presentation
- Parallel Architectures
- Intel Xeon and Intel Xeon Phi
- OpenMP
- Thread Affinity
- Vectorization
- Offloading
- Thread League
- N-body Simulation

Reference Material

- **Source-code, slides and book chapter (in Portuguese):**

https://github.com/intel-unesp-mcp/talks-source-code/tree/master/OpenMP4

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

UNESP Center for Scientific Computing

SPRACE - LHC/CMS Tier2 Facility

- 96 worker nodes
  - Physical CPUs: 128
  - Logical CPUs (cores): 1152
  - HEPSpec06: 17456
  - 128 cores: 3GB/core
  - 1024 cores: 4GB/core
- 02 head nodes
- 13 storage servers
  - 1 PB (effective)
- Network
  - LAN: 1 Gbps & 10 Gbps
  - WAN: 2x 10 Gbps , 2x 40 Gbps (1x 100G in Q3 2016)

GridUnesp - HPC infrastructure

- Campus Grid
  - 1 central cluster + 6 secondary clusters (deployed in different Unesp campi at São Paulo State)
- Worker nodes @ NCC
  - Physical CPUs: 256 (2009)
  - Logical CPUs (cores): 2048 - 2GB/core
- 1 head node
- 1 storage server
  - 132 TB (effective)
- Network
  - LAN: 1 Gbps
  - WAN: 2x 10 Gbps

Unesp / Intel Collaborative Efforts

- IPCC (Intel Parallel Computing Center)
  - Vectorization & Parallelization of Geant (**GE**ometry **AN**d **T**racking)

- Intel Modern Code
  - Workshops and Tutorials
    - ❑ High Performance Computing (HPC)
    - ❑ Data Science / Big Data Analytics
  - HPC Consultancy

Agenda

- NCC Presentation
- Parallel Architectures
- Intel Xeon and Intel Xeon Phi
- OpenMP
- Thread Affinity
- Vectorization
- Offloading
- Thread League
- N-body Simulation

Parallel Architectures

- Heterogeneous computational systems:
  - Multicore processors

Vector Instructions (SIMD)

A7	A6	A5	A4	A3	A2	A1	A0
			+				
B7	B6	B5	B4	B3	B2	B1	B0
			=				
A7+B7	A6+B6	A5+B5	A4+B4	A3+B3	A2+B2	A1+B1	A0+B0

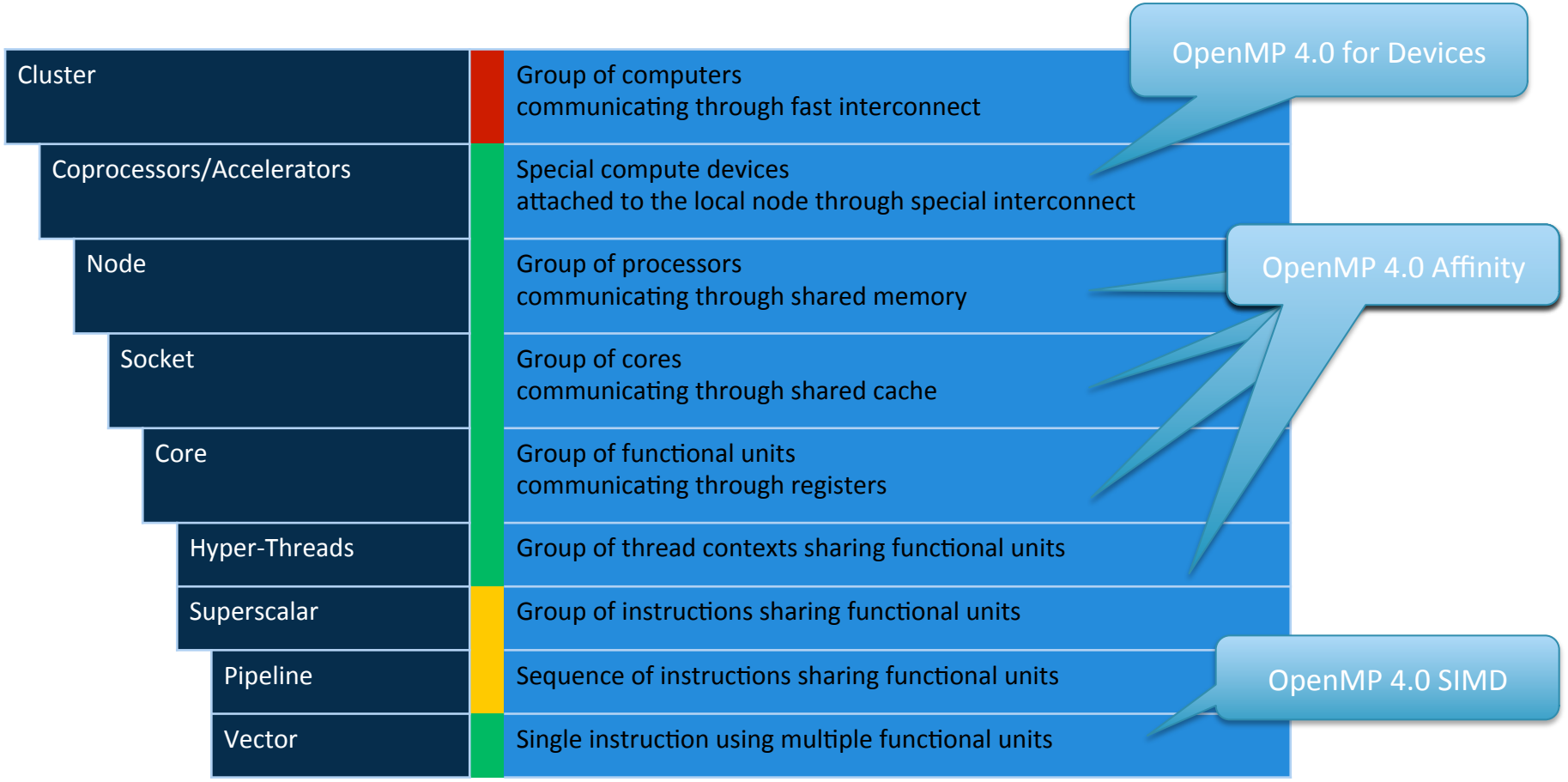
Scalar Instructions

A
+
B
=
A+B

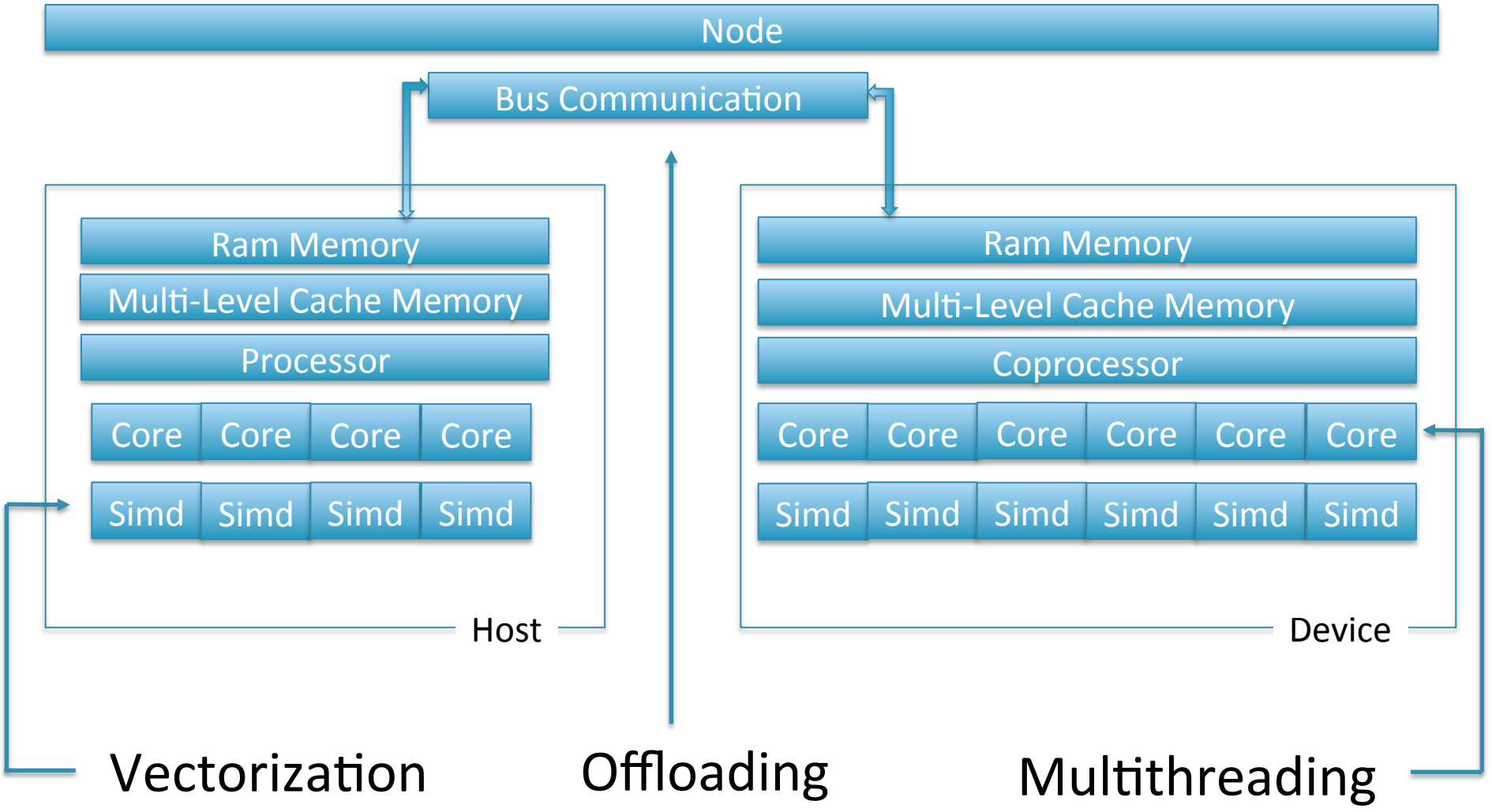
  - Multi-level memory
    - Ram Memory;
    - Multi-level Cache.

Processor 1			Processor 2		
Core 1	Core 2	Core N	Core 1	Core 2	Core N
L1	L1	L1	L1	L1	L1
L2	L2	L2	L2	L2	L2
L3			L3		
Ram					

Multi Level Parallelism



Hybrid Parallel Architectures



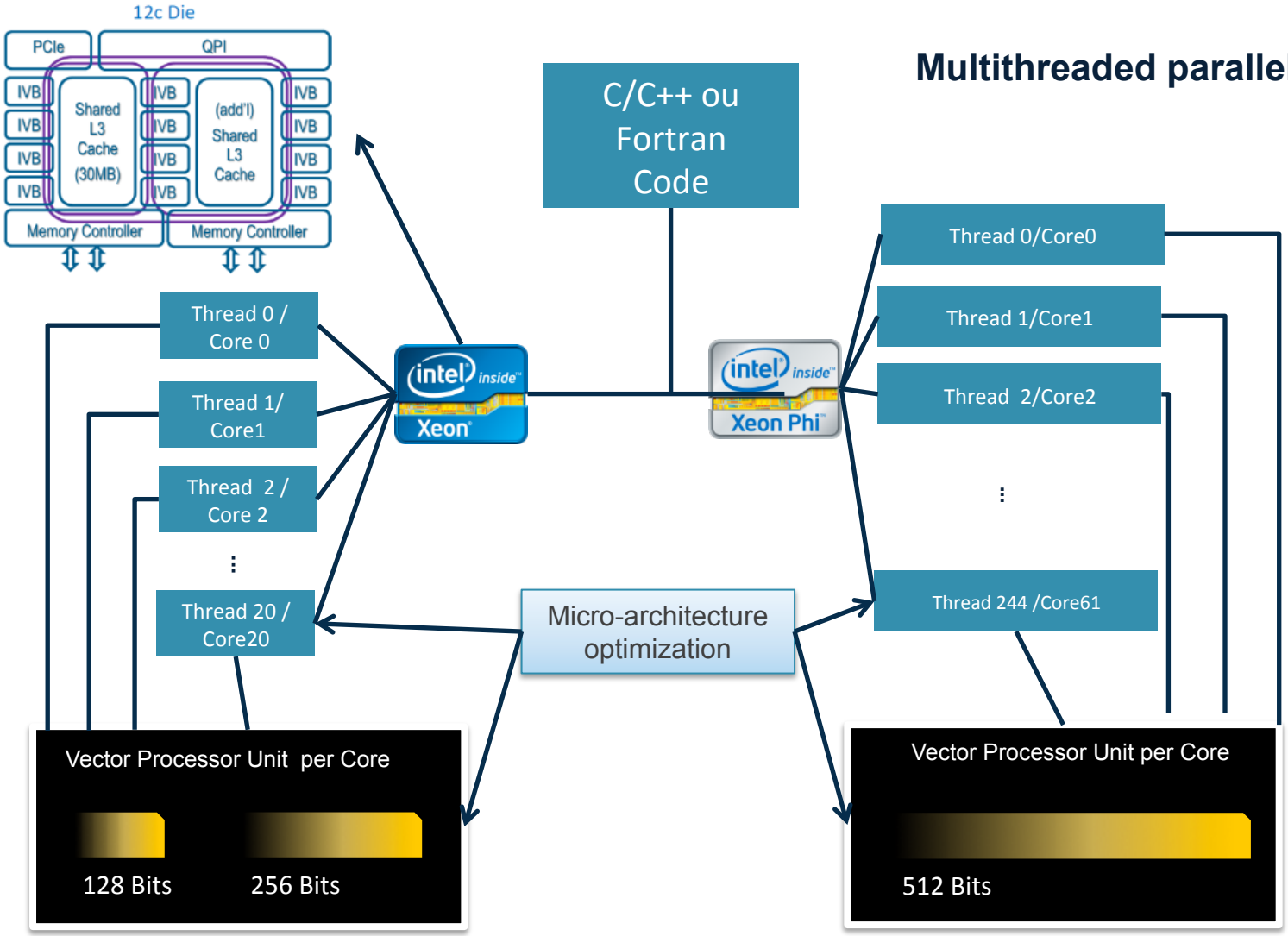
Hybrid Parallel Architectures

- Exploring parallelism in hybrid parallel architectures
  - Multithreading
  - Vectorization
    - ❑ Auto vectorization
    - ❑ Semi-auto vectorization
    - ❑ Explicit vectorization
  - Offloading
    - ❑ Offloading code to device
- OpenMP 4.0
  - Supports vectorization and offloading on hybrid parallel architectures

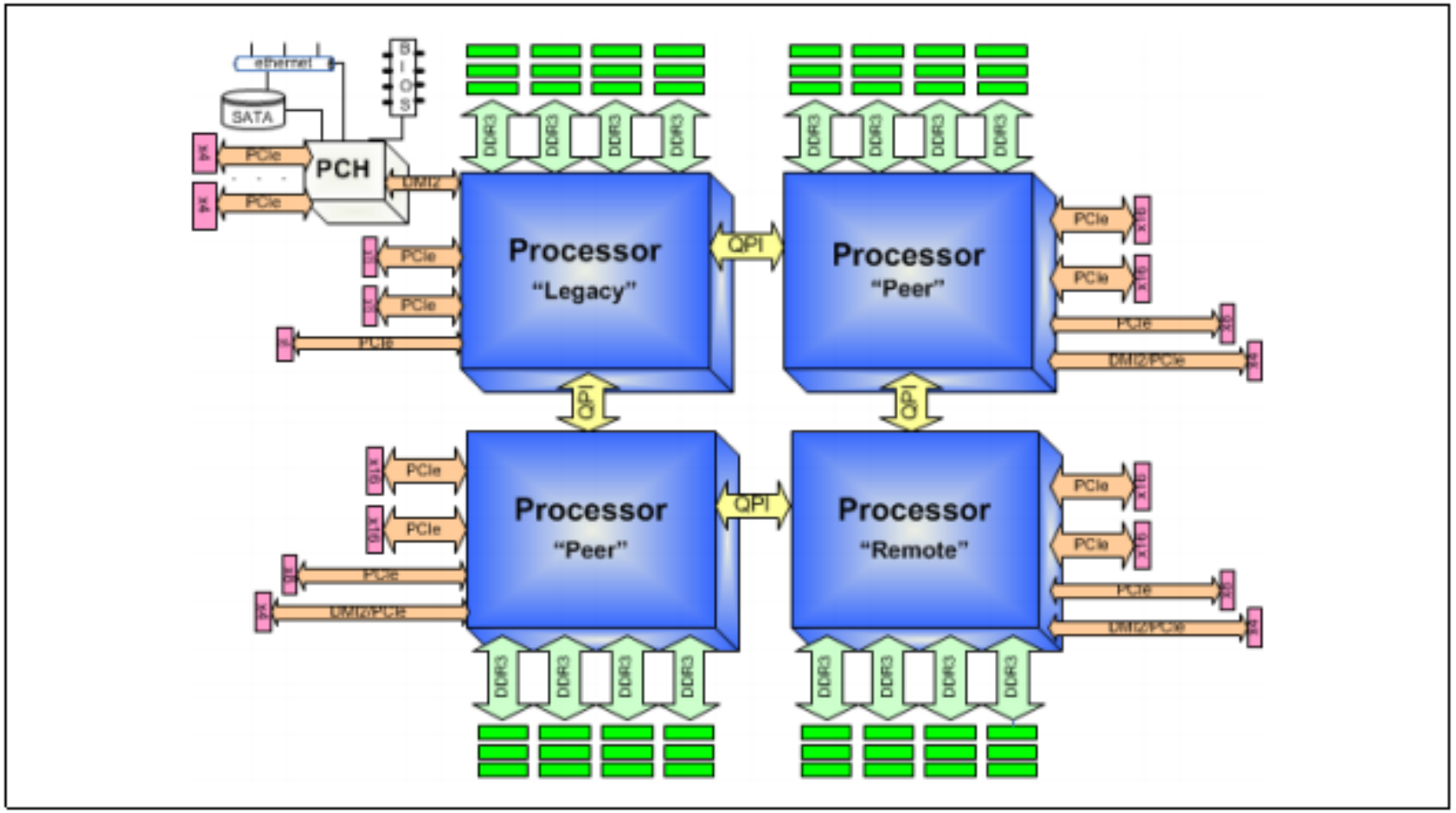
Agenda

- NCC Presentation
- Parallel Architectures
- Intel Xeon and Intel Xeon Phi
- OpenMP
- Thread Affinity
- Vectorization
- Offloading
- Thread League
- N-body Simulation

Data Level Parallelism via SIMD



Intel Xeon Architecture Overview

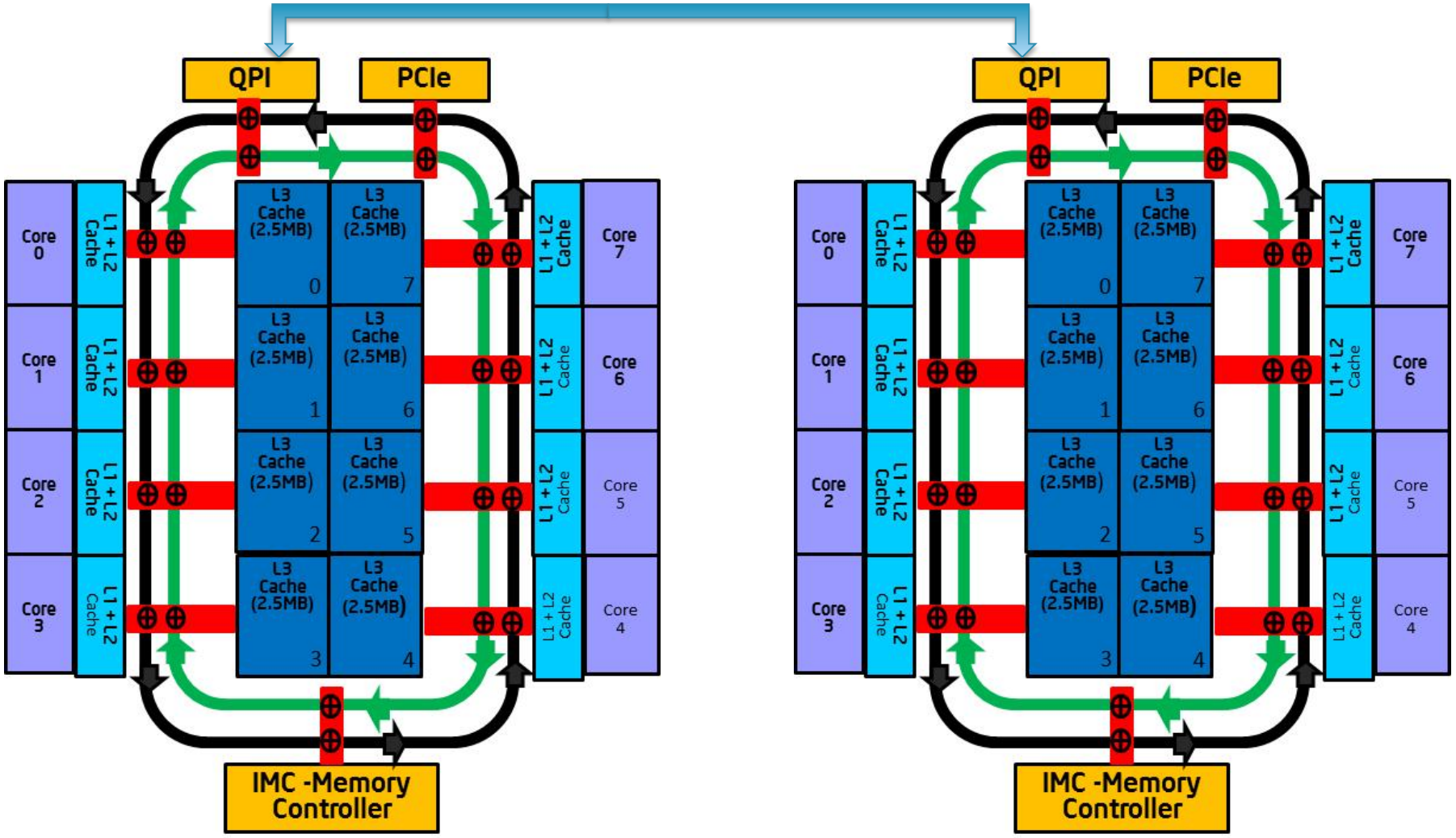


Intel Xeon Architecture Overview

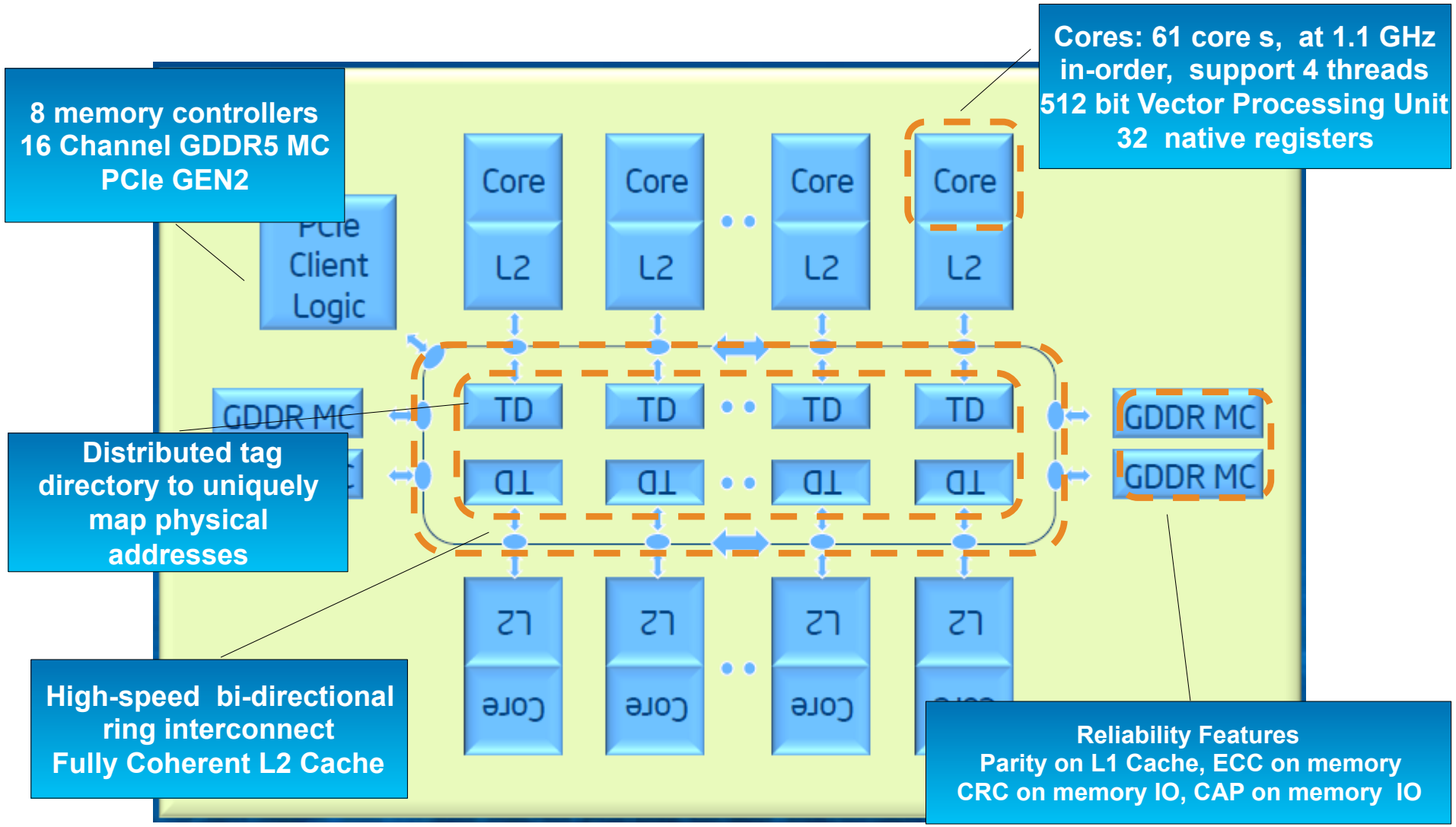
- Socket: mechanical component that provides mechanical and electrical connections between a microprocessor and a printed circuit board (PCB).

- 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.

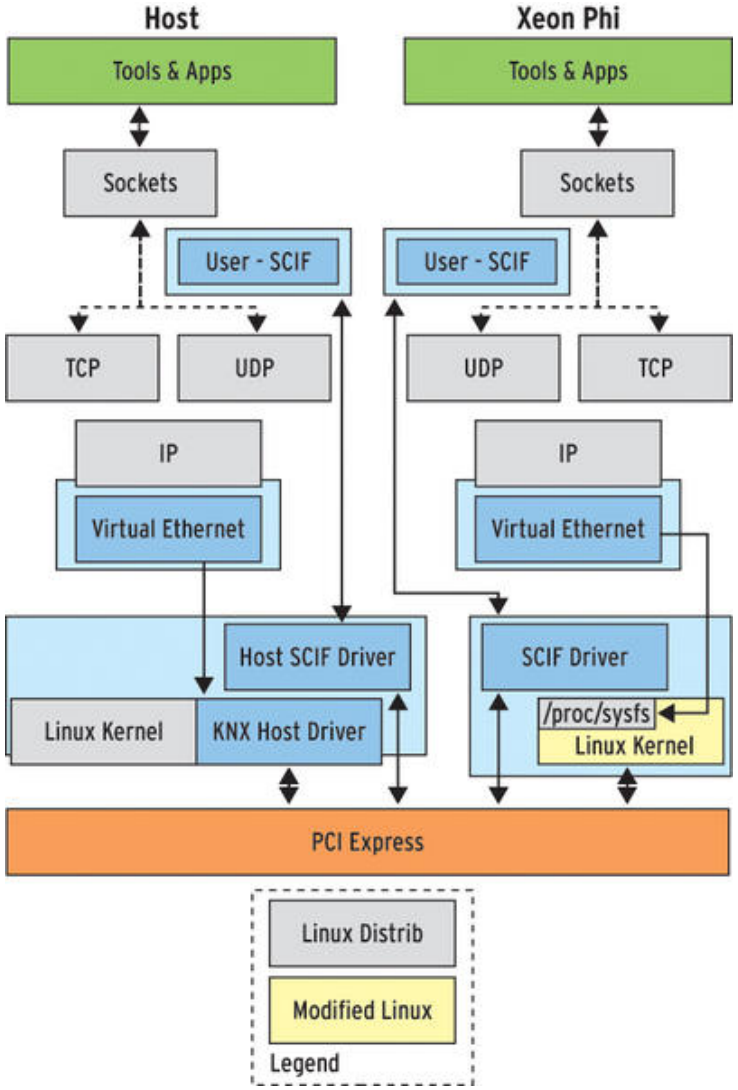
Intel® Xeon Architecture Overview



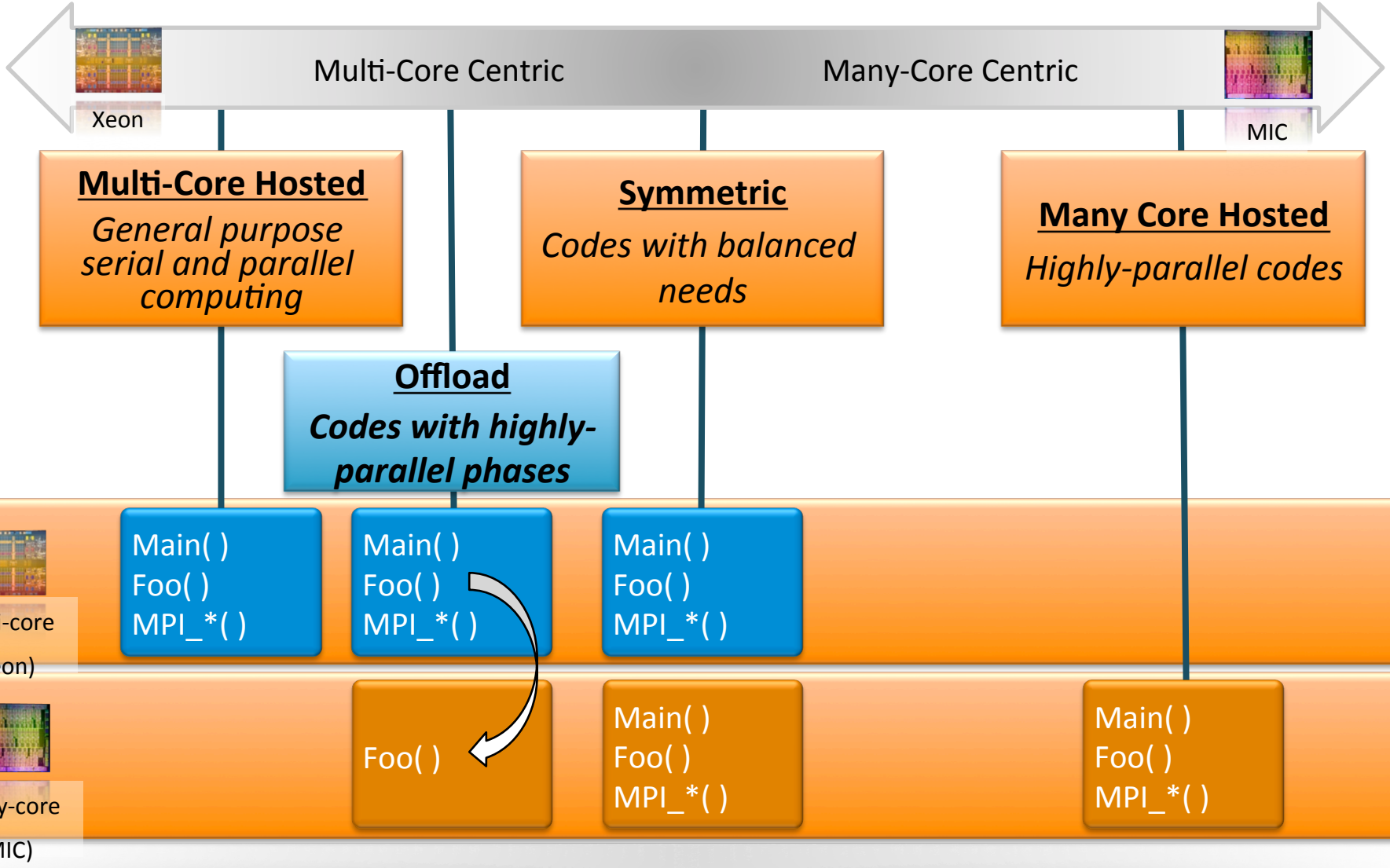
Intel® Xeon Phi™ Architecture Overview



Intel® Xeon and Intel® Xeon Phi™



Programming Models



Range of models to meet application needs

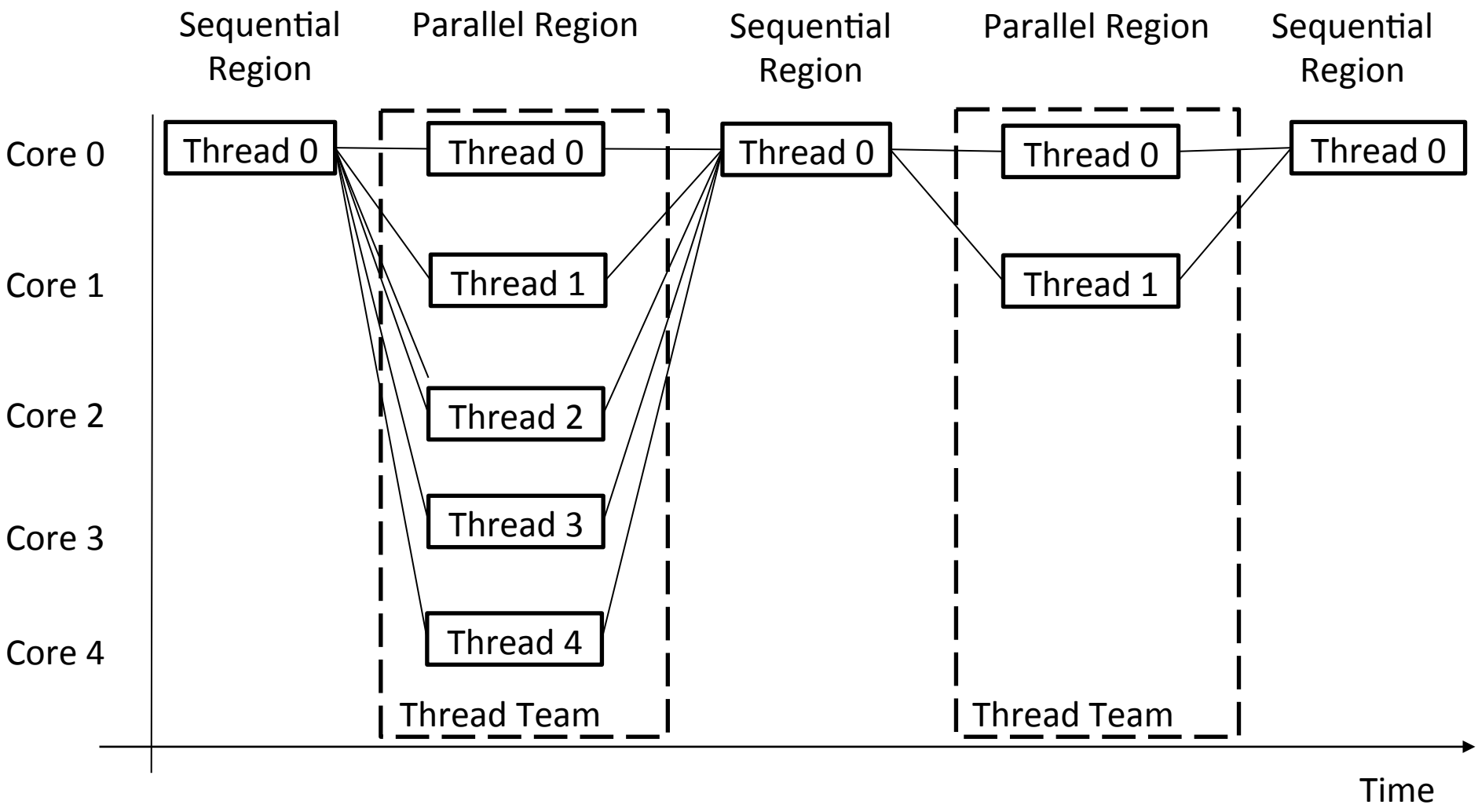
Agenda

- NCC Presentation
- Parallel Architectures
- Intel Xeon and Intel Xeon Phi
- OpenMP
- Thread Affinity
- Vectorization
- Offloading
- Thread League
- N-body Simulation

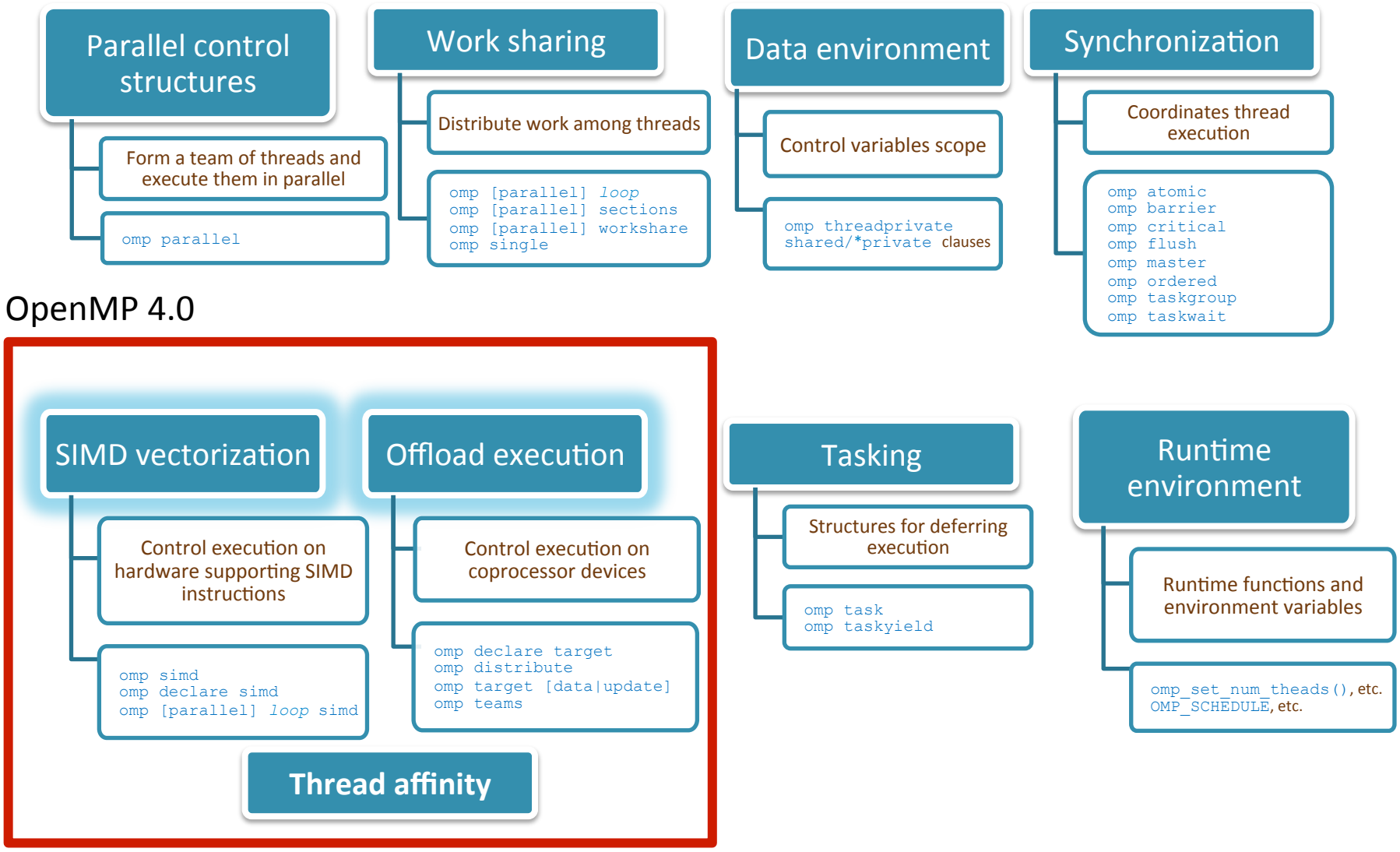
OpenMP

- OpenMP is an acronym for Open Multi-Processing
- An Application Programming Interface (API) for developing parallel programs in shared memory architectures
- Three primary components of the API are:
  - Compiler Directives
  - Runtime Library Routines
  - Environment Variables
- De facto standard - specified for C / C++ and FORTRAN
- http://www.openmp.org/
  - Specification, examples, tutorials and documentation

OpenMP


Sequential Region
Parallel Region
Sequential Region
Parallel Region
Sequential Region
Core 0
Core 1
Core 2
Core 3
Core 4
Thread 0
Thread 0
Thread 1
Thread 2
Thread 3
Thread 4
Thread 0
Thread 0
Thread 1
Thread 0
Thread Team
Thread Team
Time

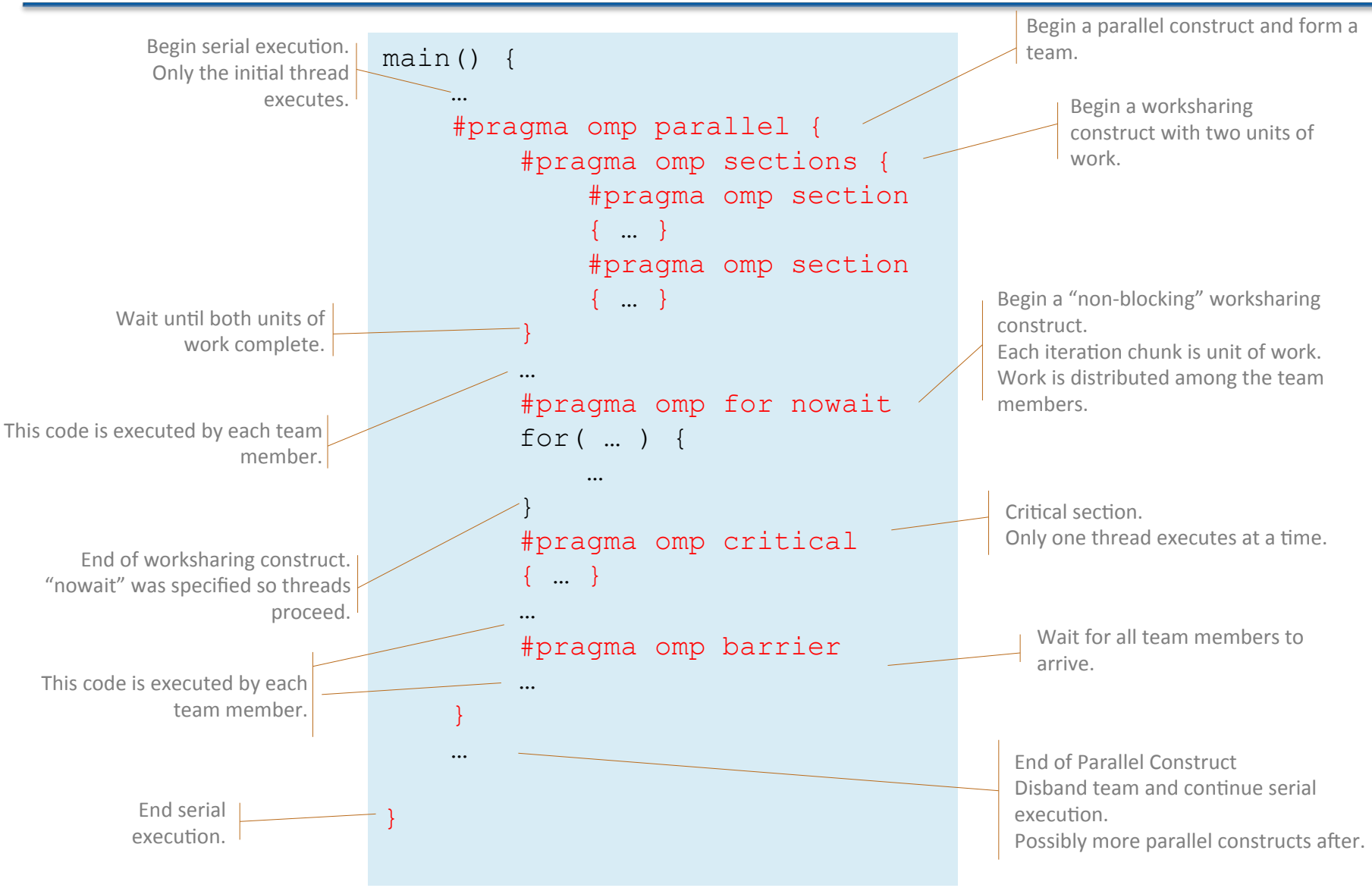
OpenMP - Core elements



OpenMP - Release Notes

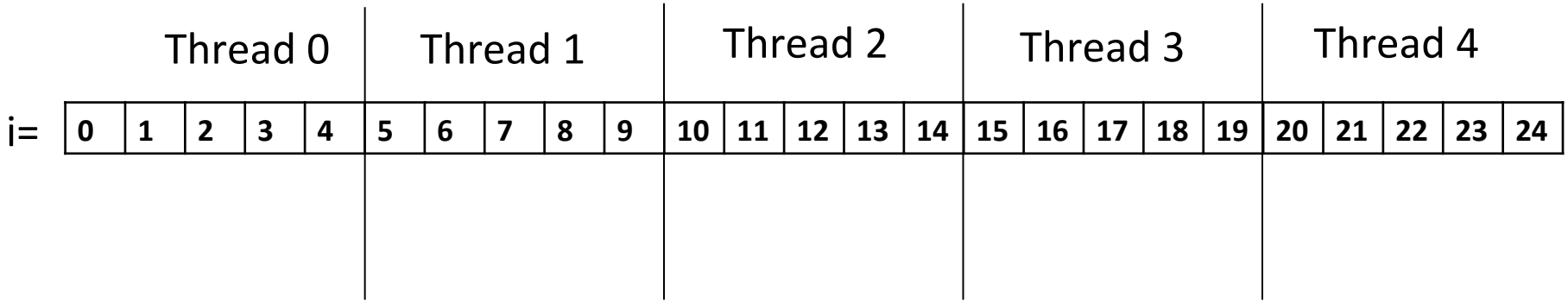
- OpenMP 4.0
  - Support for accelerators
  - SIMD constructs to vectorize both serial as well as parallelized loops
  - Thread affinity
- OpenMP 4.5
  - Improved support for devices
  - Thread affinity support
  - SIMD extensions

OpenMP Parallel Processing Model



OpenMP Sample Program

```
N=25;
#pragma omp parallel for
for (i=0; i<N; i++)
    a[i] = a[i] + b;
```



OpenMP Sample Program

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <unistd.h>

int main() {
  int thid; char hn[600], i;
  double res, p[100];

  #pragma omp parallel
  {
    gethostname(hn,600);
    printf("hostname %s\n",hn);
  }
  res = 0;

  #pragma omp for
  for ( i = 0 ; i < 100 ; i++ ) {
    p[i] = i/0.855;
  }

  #pragma omp for
  for ( i = 0 ; i < 100 ; i++ ) {
    res = res + p[i];
  }

  printf("sum: %f", res);
}
```

Compiling and running an OpenMP application

#Build the application for Multicore Architecture (Xeon)

icc <*source-code*> -o <*omp_binary*> -fopenmp

#Build the application for the ManyCore Architecture (Xeon Phi)

icc <*source-code*> -o <*omp_binary*>.mic -fopenmp -mmic

#Launch the application on host

./*omp_binary*

#Launch the application on the device from host

micnativeloadex ./*omp_binary.mic* -e "LD_LIBRARY_PATH=/opt/intel/lib/mic/"

Compiling and running an OpenMP application

```
export OMP_NUM_THREADS=10
./OMP-hello
```

```
hello from hostname phi02.ncc.unesp.br
hello from hostname phi02.ncc.unesp.br
hello from hostname phi02.ncc.unesp.br
hello from hostname phi02.ncc.unesp.br
hello from hostname phi02.ncc.unesp.br
hello from hostname phi02.ncc.unesp.br
hello from hostname phi02.ncc.unesp.br
hello from hostname phi02.ncc.unesp.br
hello from hostname phi02.ncc.unesp.br
hello from hostname phi02.ncc.unesp.br
```

Launch the application on the Coprocessor from host

```
micnativeloadex ./OMP-hello.mic -e
"OMP_NUM_THREADS=10 LD_LIBRARY_PATH=/
opt/intel/lib/mic/"
```

```
hello from hostname phi02-mic0.ncc.unesp.br
hello from hostname phi02-mic0.ncc.unesp.br
hello from hostname phi02-mic0.ncc.unesp.br
hello from hostname phi02-mic0.ncc.unesp.br
hello from hostname phi02-mic0.ncc.unesp.br
hello from hostname phi02-mic0.ncc.unesp.br
hello from hostname phi02-mic0.ncc.unesp.br
hello from hostname phi02-mic0.ncc.unesp.br
hello from hostname phi02-mic0.ncc.unesp.br
hello from hostname phi02-mic0.ncc.unesp.br
sum of vector elements: 5789.473684
```

Agenda

- NCC Presentation
- Parallel Architectures
- Intel Xeon and Intel Xeon Phi
- OpenMP
- Thread Affinity
- Vectorization
- Offloading
- Thread League
- N-body Simulation

Thread Affinity

- Thread affinity:
  - Restricts execution of certain threads to a subset of the physical processing units in a multiprocessor computer;
  - OpenMP runtime library has the ability to bind OpenMP threads to physical processing units.

Thread Affinity - KMP_AFFINITY

- KMP_AFFINITY:
  - Environment variable that control the physical processing units that will execute threads of an application

- Syntax:

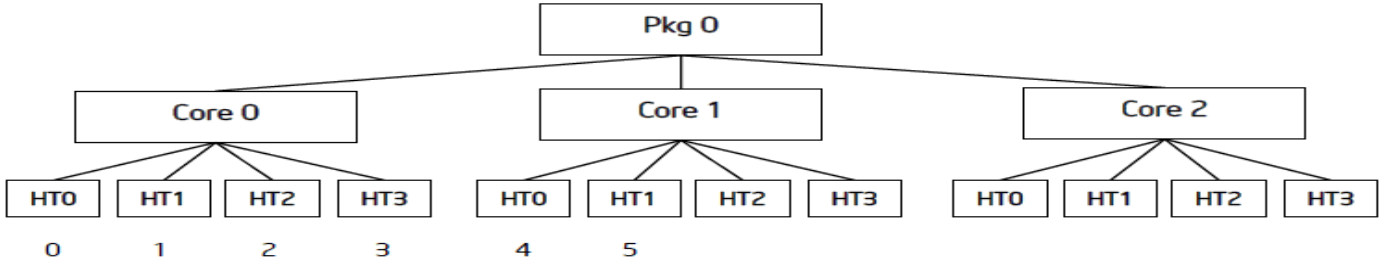
```
KMP_AFFINITY=
    [<modifier>,...]
    <type>
    [,<permute>]
    [,<offset>]
```

Example:

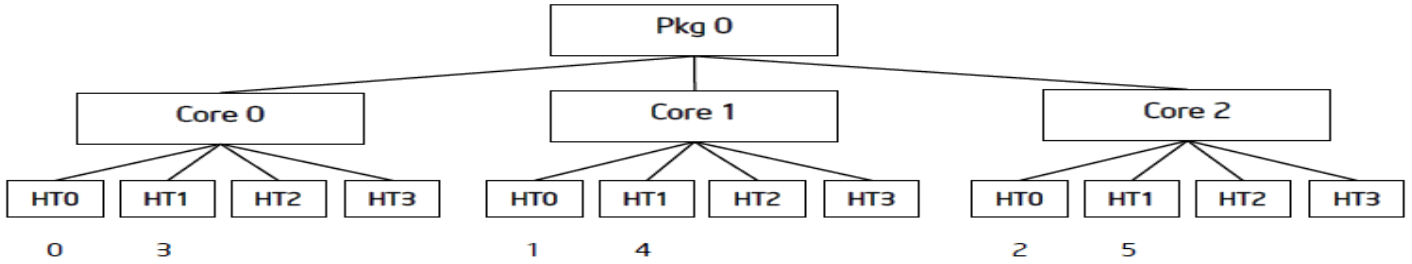
```
export KMP_AFFINITY=scatter
```

KMP_AFFINITY - Types

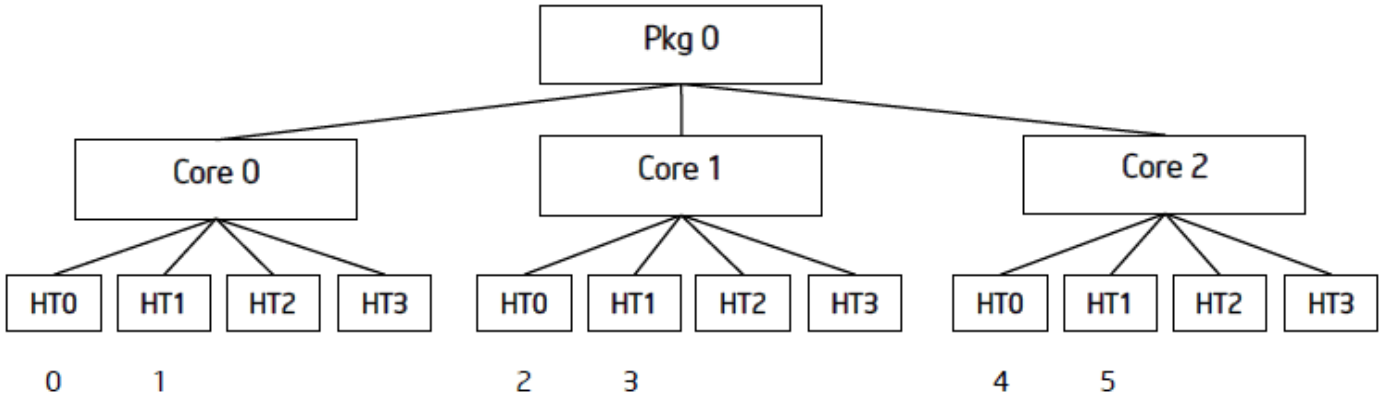
- Compact



- Scatter



- Balanced



Thread Affinity Examples

compact xeon

export KMP_AFFINITY=compact,verbose
./OMP_hello

compact xeon phi

micnativeloadex ./OMP-hello.mic -e "KMP_AFFINITY=compact,verbose OMP_NUM_THREADS=10 LD_LIBRARY_PATH=/opt/intel/lib/mic/"

scatter xeon

export KMP_AFFINITY=scatter,verbose
./OMP_hello

scatter xeon phi

micnativeloadex ./OMP-hello.mic -e "KMP_AFFINITY=scatter,verbose OMP_NUM_THREADS=10 LD_LIBRARY_PATH=/opt/intel/lib/mic/"

balanced xeon phi

micnativeloadex ./OMP-hello.mic -e "KMP_AFFINITY=balanced,verbose OMP_NUM_THREADS=10 LD_LIBRARY_PATH=/opt/intel/lib/mic/"

Thread Affinity Physical Resources Mapping

OMP: Info #156: KMP_AFFINITY: 72 available OS procs

OMP: Info #179: KMP_AFFINITY: 2 packages x 18 cores/ pkg x 2 threads/core (36 cores)

OS proc to physical thread map:

OS proc 0 maps to package 0 core 0 thread 0

OS proc 36 maps to package 0 core 0 thread 1

OS proc 1 maps to package 0 core 1 thread 0

OS proc 37 maps to package 0 core 1 thread 1

OS proc 2 maps to package 0 core 2 thread 0

OS proc 38 maps to package 0 core 2 thread 1

OS proc 18 maps to package 1 core 0 thread 0

OS proc 54 maps to package 1 core 0 thread 1

OS proc 19 maps to package 1 core 1 thread 0

OS proc 55 maps to package 1 core 1 thread 1

OS proc 20 maps to package 1 core 2 thread 0

OS proc 56 maps to package 1 core 2 thread 1

OS proc 21 maps to package 1 core 3 thread 0

Processor 1						**Processor 2**			
Core 0		Core 1		…		Core 0		Core 1	
Thread 0	Thread 1	Thread 0	Thread 1	…	…	Thread 0	Thread 1	Thread 0	Thread 1
Proc 0	Proc 36	Proc 1	Proc 37			Proc 18	Proc 54	Proc 19	Proc 55

Thread Affinity compact

OMP: Info #242: KMP_AFFINITY: pid 68487 thread 0 bound to OS proc set {0,36}
OMP: Info #242: KMP_AFFINITY: pid 68487 thread 1 bound to OS proc set {0,36}
OMP: Info #242: KMP_AFFINITY: pid 68487 thread 2 bound to OS proc set {1,37}
OMP: Info #242: KMP_AFFINITY: pid 68487 thread 3 bound to OS proc set {1,37}
OMP: Info #242: KMP_AFFINITY: pid 68487 thread 4 bound to OS proc set {2,38}
OMP: Info #242: KMP_AFFINITY: pid 68487 thread 5 bound to OS proc set {2,38}
OMP: Info #242: KMP_AFFINITY: pid 68487 thread 6 bound to OS proc set {3,39}
OMP: Info #242: KMP_AFFINITY: pid 68487 thread 7 bound to OS proc set {3,39}
OMP: Info #242: KMP_AFFINITY: pid 68487 thread 8 bound to OS proc set {4,40}
OMP: Info #242: KMP_AFFINITY: pid 68487 thread 9 bound to OS proc set {4,40}

Thread Affinity scatter

OMP: Info #242: KMP_AFFINITY: pid 69401 thread 0 bound to OS proc set {0,36}
OMP: Info #242: KMP_AFFINITY: pid 69401 thread 1 bound to OS proc set {18,54}
OMP: Info #242: KMP_AFFINITY: pid 69401 thread 2 bound to OS proc set {1,37}
OMP: Info #242: KMP_AFFINITY: pid 69401 thread 3 bound to OS proc set {19,55}
OMP: Info #242: KMP_AFFINITY: pid 69401 thread 4 bound to OS proc set {2,38}
OMP: Info #242: KMP_AFFINITY: pid 69401 thread 5 bound to OS proc set {20,56}
OMP: Info #242: KMP_AFFINITY: pid 69401 thread 6 bound to OS proc set {3,39}
OMP: Info #242: KMP_AFFINITY: pid 69401 thread 7 bound to OS proc set {21,57}
OMP: Info #242: KMP_AFFINITY: pid 69401 thread 8 bound to OS proc set {4,40}
OMP: Info #242: KMP_AFFINITY: pid 69401 thread 9 bound to OS proc set {22,58}

Thread Affinity balanced

OMP: Info #242: KMP_AFFINITY: pid 17662 thread 9 bound to OS proc set {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,143,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,164,165,166,167,168,169,170,171,172,173,174,175,176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,225,226,227,228,229,230,231,232,233,234,235,236,237,238,239}

OMP: Info #242: KMP_AFFINITY: pid 17662 thread 0 bound to OS proc set {1}

OMP: Info #242: KMP_AFFINITY: pid 17662 thread 8 bound to OS proc set {33}

OMP: Info #242: KMP_AFFINITY: pid 17662 thread 3 bound to OS proc set {13}

OMP: Info #242: KMP_AFFINITY: pid 17662 thread 4 bound to OS proc set {17}

OMP: Info #242: KMP_AFFINITY: pid 17662 thread 5 bound to OS proc set {21}

OMP: Info #242: KMP_AFFINITY: pid 17662 thread 9 bound to OS proc set {37}

OMP: Info #242: KMP_AFFINITY: pid 17662 thread 1 bound to OS proc set {5}

OMP: Info #242: KMP_AFFINITY: pid 17662 thread 6 bound to OS proc set {25}

OMP: Info #242: KMP_AFFINITY: pid 17662 thread 7 bound to OS proc set {29}

OMP: Info #242: KMP_AFFINITY: pid 17662 thread 2 bound to OS proc set {9}

Agenda

- NCC Presentation
- Parallel Architectures
- Intel Xeon and Intel Xeon Phi
- OpenMP
- Thread Affinity
- Vectorization
- Offloading
- Thread League
- N-body Simulation

Vectorization

- Instructs the compiler to enforce vectorization of loops (**Semi-auto vectorization**)

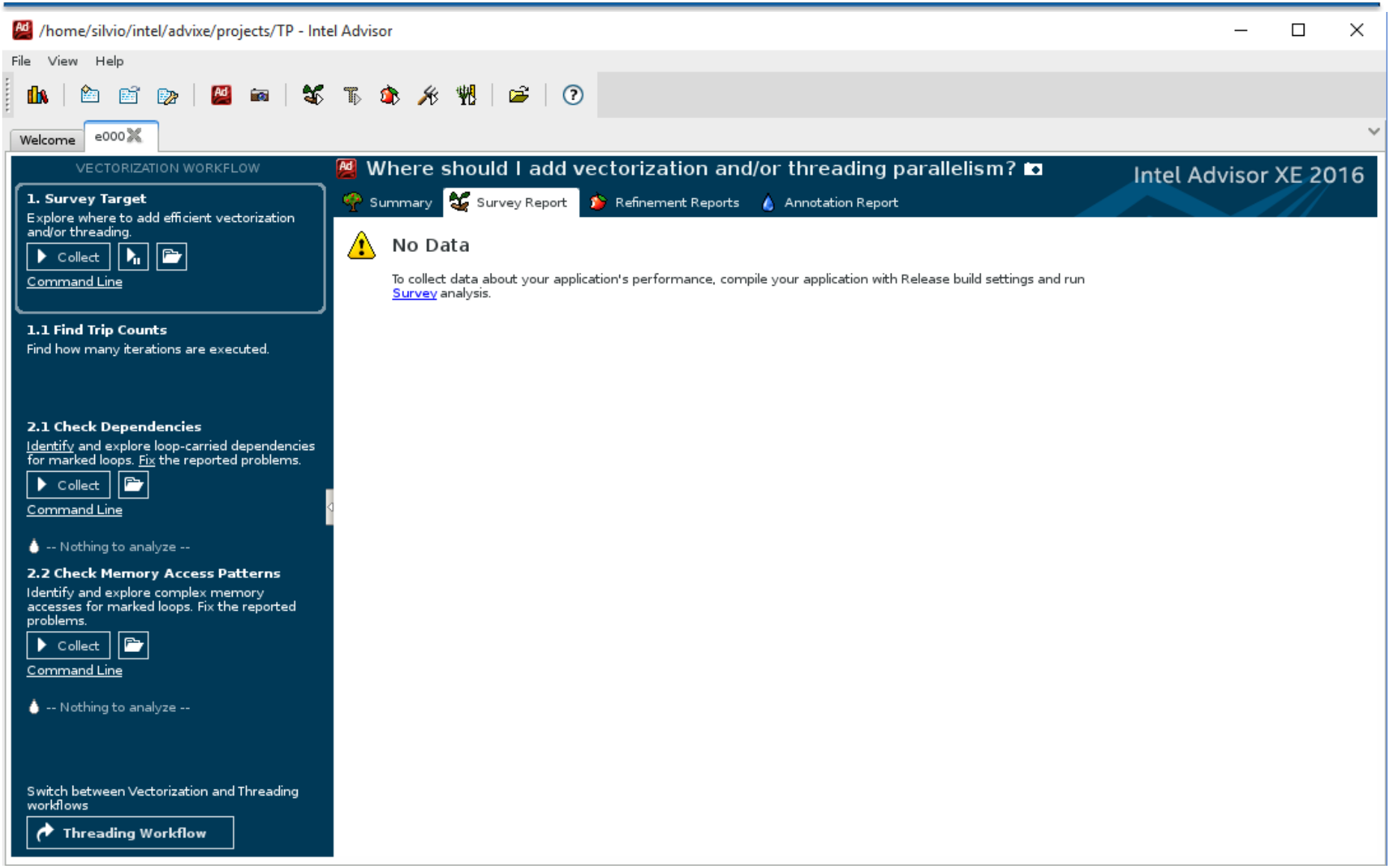
- omp simd
  - marks a loop to be vectorized by the compiler
- omp declare simd
  - marks a function that can be called from a SIMD loop to be vectorized by the compiler
- omp parallel for simd
  - marks a loop for thread work-sharing as well as SIMDing

Intel Advisor

- Evaluate multi-threading parallelization

- Intel® Advisor XE
  - ❑ Performance modeling using several frameworks for multi-threading in processors and co-processors:
    - OpenMP, Intel® Cilk ™ Plus, Intel® Threading Bulding Blocks
    - C, C++, Fortran (OpenMP only) and C# (Microsoft TPL)
  - ❑ Identify parallel opportunities
    - Detailed information about vectorization;
    - Check loop dependencies;
  - ❑ Scalability prediction: amount of threads/performance gains
  - ❑ Correctness (deadlocks, race condition)

Intel Advisor



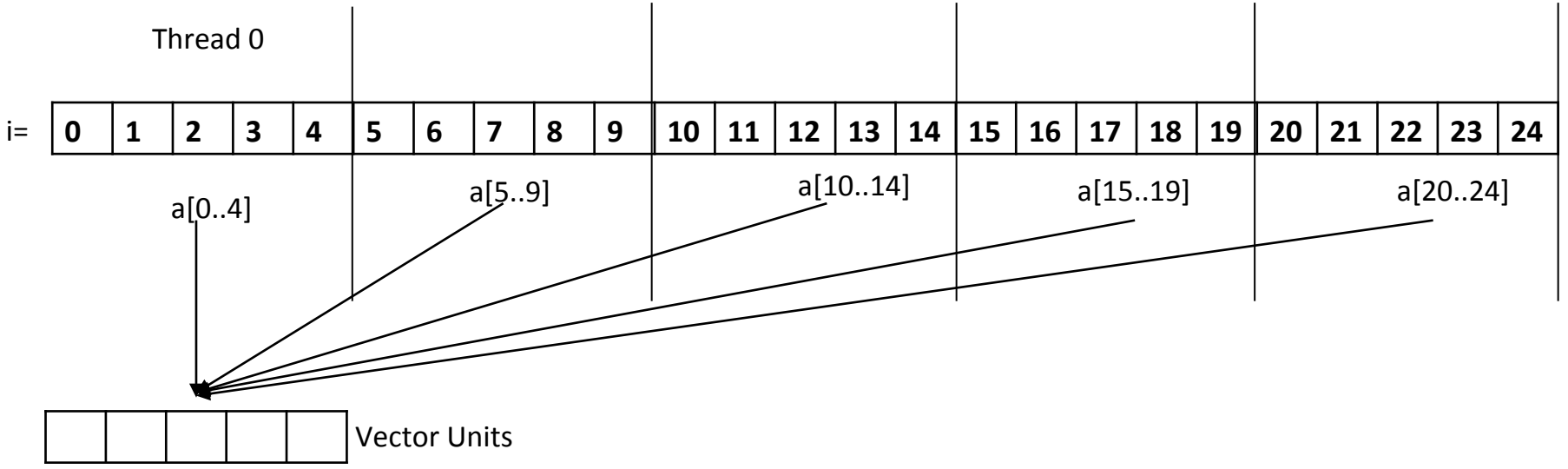
Pragma omp simd

- Vectorize a loop nest
  - Cut loop into chunks that fit a SIMD vector register
  - No parallelization of the loop body

- Syntax

  `#pragma omp simd [clause[[,] clause],…]`
  for-loops

```
N=25;
#pragma omp simd
for (i=0; i<N; i++)
   a[i] = a[i] + b;
```



Data Sharing Clauses

- Specifies that each thread has its own instance of a variable:
  - private(*var-list*): uninitialized vectors for variables in *var-list*
  - firstprivate(*var-list*): Initialized vectors for variables in *var-list*
  - lastprivate(*var-list*):
    - similar to private clause
    - Private copy of last iteration is copied to the original variable
  - reduction(*op*:*var-list*): create private variables for *var-list* and apply reduction operator *op* at the end of the construct

SIMD Loop Clauses

- simdlen (*length*)
  - generate function to support a given vector length
- safelen (*length*)
  - Maximum number of iterations that can run concurrently without breaking a dependence
- linear (*list*[:*linear-step*])
  - The variable's value is in relationship with the iteration number

  $x_i = x_{orig} + i * \text{linear-step}$
- aligned (*list*[:*alignment*])
  - Specifies that the list items have a given alignment
  - Default is alignment for the architecture
- collapse (*n*)
  - Groups two or more loops into a single loop

Pragma omp simd - Example 1

```
#pragma omp parallel for colapse (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] ;
      }
   }
}
```

OMP SIMD - Vectorization Report

Compiler could not automatically vectorize loop on line 228, because of “assumed dependency”

Loops	Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?
[loop in __kmp_launch_thread at kmp_runtime.c:5900]		0.393s 58.1%	0.730s	Scalar	
[loop in multiply3ompparallel_for@225 at multiply.c:226]	1 Assumed dependency present	0.011s	0.347s	Scalar	vector dependence: assumed dependence between lines
[loop in __libc_csu_init]		0.000s	0.020s	Scalar	
[loop in __libc_start_main]		0.000s	0.070s	Scalar	
[loop in start_thread]		0.000s	0.730s	Scalar	
[loop in _INTERNAL_16_offload_host_cpp_ad9271c5::__offload_init_library_once]		0.000s	0.020s	Scalar	
[loop in func@0x5b810]	1 Data type conversions present	0.000s	0.010s	Scalar	
[loop in func@0x5b740]	1 System function call(s) present	0.000s	0.010s	Scalar	
[loop in func@0x5b740]		0.000s	0.010s	Scalar	
[loop in [OpenMP worker] at z_Linux_util.c:786]		0.000s	0.730s	Scalar	
[loop in multiply3ompparallel_for@225 at multiply.c:228]	**1 Assumed dependency pre..**	**0.336s**	**0.336s**	**Scalar**	**vector dependence: assumed dependence betwe...**

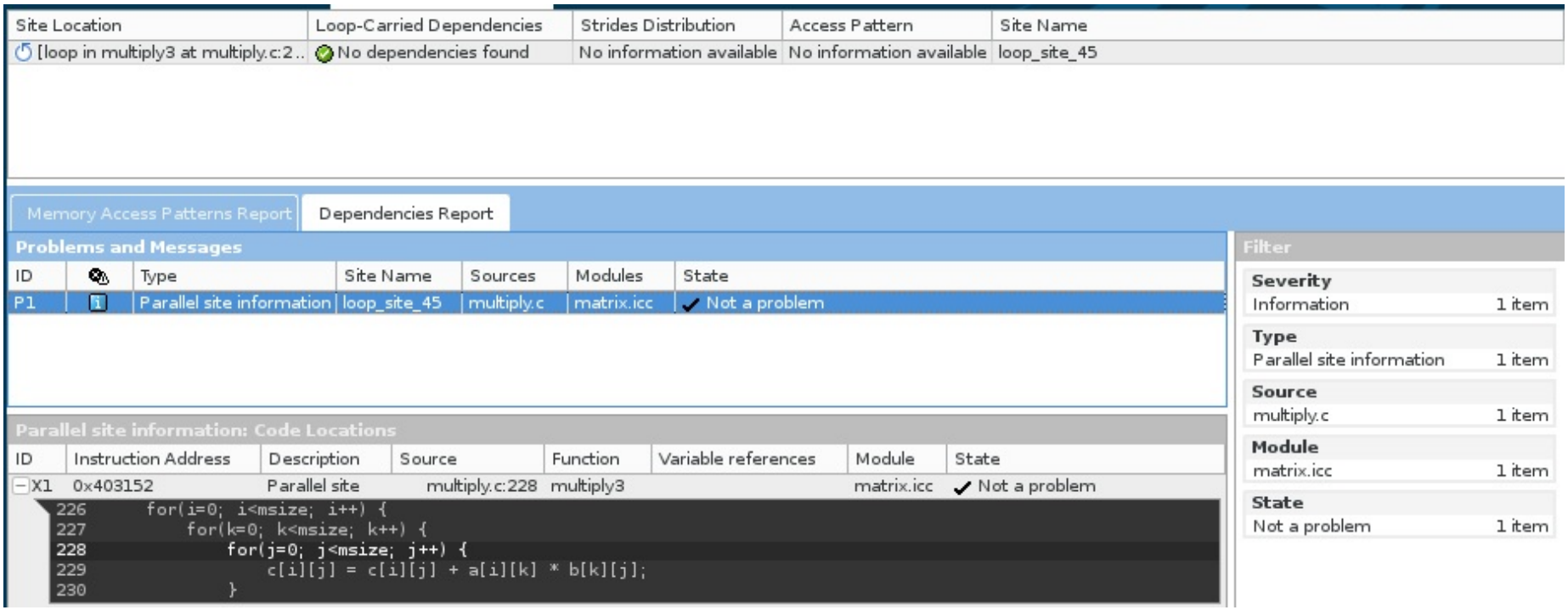
Source | Top Down | Loop Assembly | Recommendations | Compiler Diagnostic Details

File: multiply.c:228 multiply3ompparallel_for@225

Line	Source	Total Time	%	Loop Time	%	
218	void multiply3(int msize, int tidx, int numt, TYPE a[][NUM], TYPE b[][NUM], TYPE c[][NUM], TYPE t[][NUM])					
219	{					
220						
221	//#pragma omp target device(0) map(a[0:NUM][0:NUM]) \					
222	map(b[0:NUM][0:NUM]) map(c[0:NUM][0:NUM])					
223	//{					
224	int i,j,k;					
225	#pragma omp parallel for collapse (2) //num threads(60)	20.000ms				
226	for(i=0; i<msize; i++) {	11.343ms		347.104ms		Divisions
	[Scalar loop in multiply3ompparallel_for@225 at multiply.c:226] Scalar Loop. Not vectorized: vector dependence: assumed dependence between lines Remainder loop					
227	for(k=0; k<msize; k++) {					
228	for(j=0; j<msize; j++) {	40.118ms		335.761ms		
	[Scalar loop in multiply3ompparallel_for@225 at multiply.c:228] Scalar Loop. Not vectorized: vector dependence: assumed dependence between lines Loop was unrolled by 2					
229	c[i][j] = c[i][j] + a[i][k] * b[k][j];	295.644ms				FMA
230	}					
231	}					
232	}					
233	//}					
234	}					

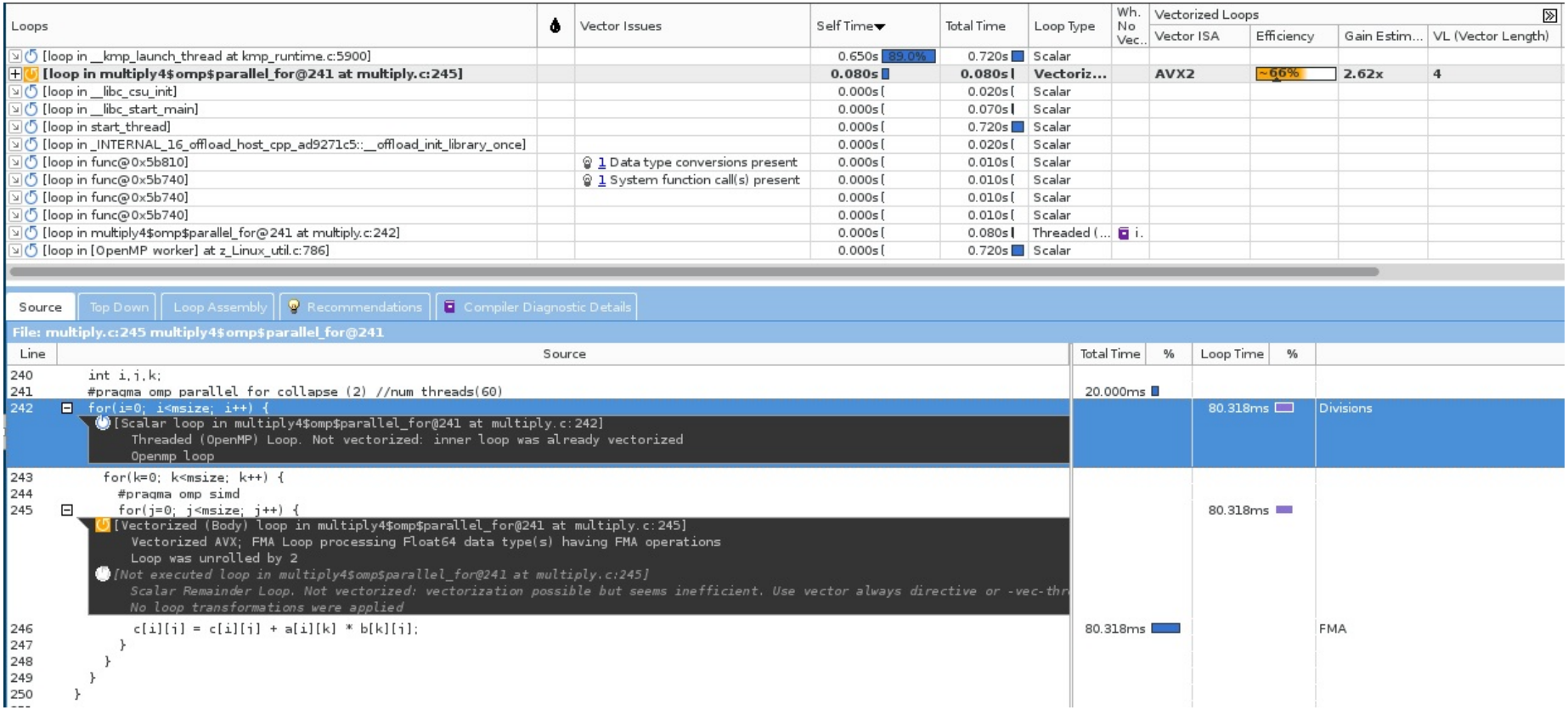
OMP SIMD - Vectorization Report

Check dependency analysis shows that it is safe to enforce the vectorization of this loop



OMP SIMD - Vectorization Report

#pragma omp simd guided the compiler to vectorize loop using AVX2



Loops	Vector Issues	Self Time	Total Time	Loop Type	Wh. No Vec..	Vectorized Loops: Vector ISA	Efficiency	Gain Estim...	VL (Vector Length)
[loop in __kmp_launch_thread at kmp_runtime.c:5900]		0.650s 89.0%	0.720s	Scalar					
[loop in multiply4ompparallel_for@241 at multiply.c:245]		**0.080s**	**0.080s**	**Vectoriz...**		**AVX2**	**~66%**	**2.62x**	**4**
[loop in __libc_csu_init]		0.000s	0.020s	Scalar					
[loop in __libc_start_main]		0.000s	0.070s	Scalar					
[loop in start_thread]		0.000s	0.720s	Scalar					
[loop in _INTERNAL_16_offload_host_cpp_ad9271c5::__offload_init_library_once]		0.000s	0.020s	Scalar					
[loop in func@0x5b810]	1 Data type conversions present	0.000s	0.010s	Scalar					
[loop in func@0x5b740]	1 System function call(s) present	0.000s	0.010s	Scalar					
[loop in func@0x5b740]		0.000s	0.010s	Scalar					
[loop in func@0x5b740]		0.000s	0.010s	Scalar					
[loop in multiply4ompparallel_for@241 at multiply.c:242]		0.000s	0.080s	Threaded (...	i.				
[loop in [OpenMP worker] at z_Linux_util.c:786]		0.000s	0.720s	Scalar					

Source | Top Down | Loop Assembly | Recommendations | Compiler Diagnostic Details

File: multiply.c:245 multiply4ompparallel_for@241

Line	Source	Total Time	%	Loop Time	%	
240	int i,j,k;					
241	#pragma omp parallel for collapse (2) //num threads(60)	20.000ms				
242	for(i=0; i<msize; i++) {			80.318ms		Divisions
	[Scalar loop in multiply4ompparallel_for@241 at multiply.c:242] Threaded (OpenMP) Loop. Not vectorized: inner loop was already vectorized Openmp loop					
243	for(k=0; k<msize; k++) {					
244	#pragma omp simd					
245	for(j=0; j<msize; j++) {			80.318ms		
	[Vectorized (Body) loop in multiply4ompparallel_for@241 at multiply.c:245] Vectorized AVX; FMA Loop processing Float64 data type(s) having FMA operations Loop was unrolled by 2 *[Not executed loop in multiply4ompparallel_for@241 at multiply.c:245]* *Scalar Remainder Loop. Not vectorized: vectorization possible but seems inefficient. Use vector always directive or -vec-thr* *No loop transformations were applied*					
246	c[i][j] = c[i][j] + a[i][k] * b[k][j];	80.318ms				FMA
247	}					
248	}					
249	}					
250	}					

Pragma omp simd - Example 2

```
void vec3(float *a, float *b, int off, int len)
{
  int i;
  #pragma omp simd aligned(a:64, b:64) simdlen(64)
  for(i = 0; i < len; i++)
  {
    a[i] = (sin(cos(a[i])) > 2.34) ?
    a[i] * atan(b[i]) :
    a[i] + cos(sin(b[i]));
  }
}
```

OMP SIMD Example 2 - Vectorization Report

Assumed dependency prevents automatic vectorization;

Loops		Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?
[loop in main at OMP-function.c:102]		4 Assumed dependency present	1.470s	8.810s	Scalar	vector dependence prevents vectorization
[loop in vec3 at omp-func.c:27]		**2 Assumed dependency present**	**1.430s**	**67.319s**	**Scalar**	**vector dependence prevents vectorization**
[loop in __libc_start_main]			0.000s	76.129s	Scalar	
[loop in main at OMP-function.c:98]			0.000s	76.129s	Scalar	loop with function call not considered an optimization ...

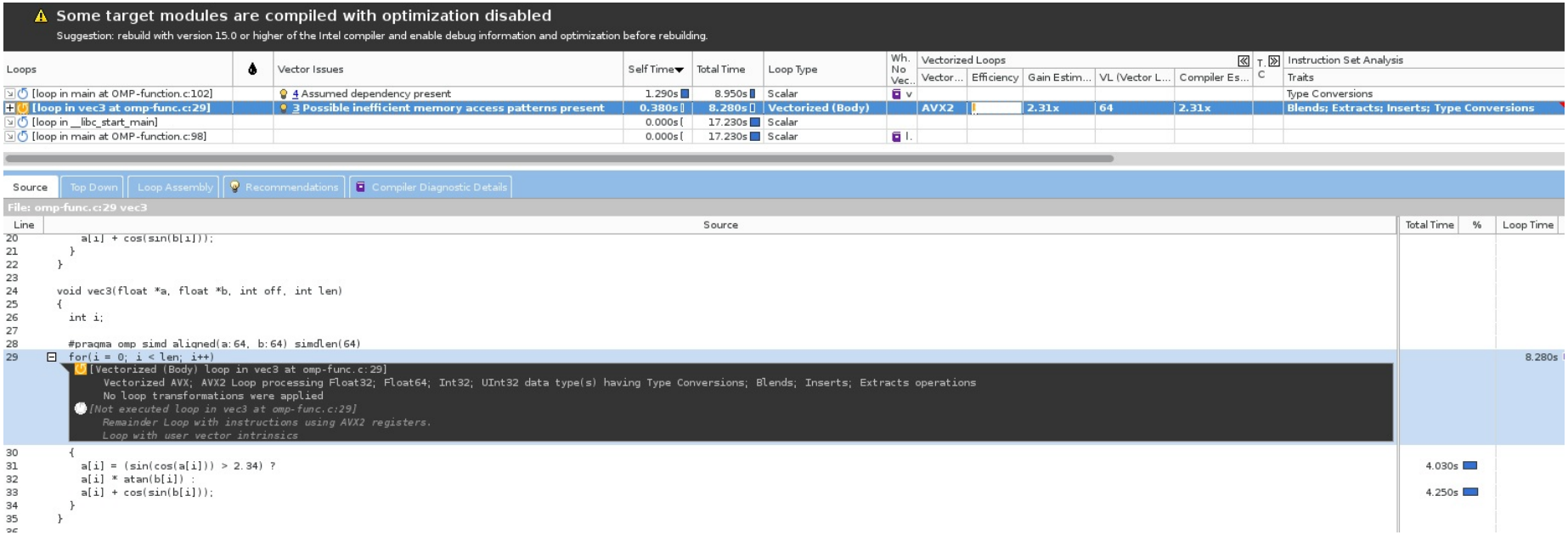
Source | Top Down | Loop Assembly | Recommendations | Compiler Diagnostic Details

File: omp-func.c:27 vec3

```
Line  Source
11      //#pragma omp simd aligned(a:64, b:64) simdlen(64)
12    void vec2(float *a, float *b, int off, int len)
13    {
14      int i;
15      #pragma omp simd
16      for(i = 0; i < len; i++)
17      {
18        a[i] = (sin(cos(a[i])) > 2.34) ?
19        a[i] * atan(b[i]) :
20        a[i] + cos(sin(b[i]));
21      }
22    }
23
24    void vec3(float *a, float *b, int off, int len)
25    {
26      int i;
27      for(i = 0; i < len; i++)
          [Scalar loop in vec3 at omp-func.c:27]
            Scalar Loop. Not vectorized: vector dependence prevents vectorization
            No loop transformations were applied
28      {
29        a[i] = (sin(cos(a[i])) > 2.34) ?
30        a[i] * atan(b[i]) :
31        a[i] + cos(sin(b[i]));
32      }
33    }
```

OMP SIMD Example 2 - Vectorization Report

aligned 64 and simdlen 64 guided the compiler to vectorize loop using AVX2;



SIMD Function Vectorization

- Declare one or more functions to be compiled for calls from a SIMD-parallel loop

- Syntax (C/C++):

```
#pragma omp declare simd [clause[[,] clause],…]
[#pragma omp declare simd [clause[[,] clause],…]]
[…]
function-definition-or-declaration
```

SIMD Function Vectorization

- uniform (*argument-list*)
  - argument has a constant value between the iterations of a given loop
- inbranch
  - function always called from inside an if statement
- notinbranch
  - function never called from inside an if statement

- simdlen (*argument-list[:linear-step]*)
- linear (*argument-list[:linear-step]*)
- aligned (*argument-list[:alignment]*)
- reduction (*operator*:list)

Pragma omp declare simd

```
#pragma omp declare simdlen (SIMD_LEN)
int FindPosition(double x) {
   return (int)(log(exp(x*steps)));
}

#pragma omp declare simd uniform (vals)
double Interpolate(double x, const point*
vals)
{
   int ind = FindPosition(x);
   ...

   return res;
}
```

```
int main ( int argc , char argv [] )
{
   . . .
  for ( i=0; i <ARRAY_SIZE;++ i ) {
      dst[i] = Interpolate( src[i], vals ) ;
   }
   . . .
}
```

George M. Raskulinec, Evgeny Fiksman "Chapter 22 - **SIMD functions via OpenMP**", In High Performance Parallelism Pearls, edited by James Reinders and Jim Jeffers, Morgan Kaufmann, Boston, 2015, Pages 171-190, ISBN 9780128038192

Vectorization report without OpenMP - Main loop

LOOP BEGIN at main.c(126,5)

remark #15382: vectorization support: call to function Interpolate(double, const point *) cannot be vectorized [main.c(127,18)]

remark #15344: loop was not vectorized: vector dependence prevents vectorization

LOOP END

Vectorization report with OpenMP - Main loop

```
LOOP BEGIN at main.c(126,5)
    remark #15388: vectorization support: reference src has aligned access   [ main.c(127,18) ]
    remark #15388: vectorization support: reference dst has aligned access   [ main.c(127,9) ]
    remark #15305: vectorization support: vector length 8
    remark #15399: vectorization support: unroll factor set to 2
    remark #15309: vectorization support: normalized vectorization overhead 0.013
    remark #15300: LOOP WAS VECTORIZED
    remark #15448: unmasked aligned unit stride loads: 1
    remark #15449: unmasked aligned unit stride stores: 1
    remark #15475: --- begin vector loop cost summary ---
    remark #15476: scalar loop cost: 107
    remark #15477: vector loop cost: 14.500
    remark #15478: estimated potential speedup: 7.370
    remark #15484: vector function calls: 1
    remark #15488: --- end vector loop cost summary ---
    remark #15489: --- begin vector function matching report ---
    remark #15490: Function call: Interpolate(double, const point *) with simdlen=8, actual parameter types:
(vector,uniform)   [ main.c(127,18) ]
    remark #15492: A suitable vector variant was found (out of 4) with ymm2, simdlen=4, unmasked, formal
parameter types: (vector,uniform)
    remark #15493: --- end vector function matching report ---
  LOOP END
```

Vectorization report with OpenMP - Interpolate

```
Begin optimization report for: Interpolate..._simdsimd3__H2n_v1_s1.P(double, const point *)

   Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED   [ main.c(74,48) ]
================================================================================

Begin optimization report for: Interpolate..._simdsimd3__H2m_v1_s1.P(double, const point *)

   Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED   [ main.c(74,48) ]
================================================================================

Begin optimization report for: Interpolate..._simdsimd3__L4n_v1_s1.V(double, const point *)

   Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED   [ main.c(74,48) ]
remark #15415: vectorization support: gather was generated for the variable pnt:  indirect access, 64bit indexed   [ main.c(78,26) ]
remark #15415: vectorization support: gather was generated for the variable pnt:  indirect access, 64bit indexed   [ main.c(78,36) ]
================================================================================

Begin optimization report for: Interpolate..._simdsimd3__L4m_v1_s1.V(double, const point *)

   Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED   [ main.c(74,48) ]
remark #15415: vectorization support: gather was generated for the variable pnt: masked, indirect access, 64bit indexed   [ main.c(78,26) ]
remark #15415: vectorization support: gather was generated for the variable pnt: masked, indirect access, 64bit indexed   [ main.c(78,36) ]
```

Vectorization report with OpenMP - FindPosition

egin optimization report for: FindPosition.._simdsimd3__H2n_v1.P(double)

Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED [main.c(70,28)]
==========

Begin optimization report for: FindPosition.._simdsimd3__H2m_v1.P(double)

Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED [main.c(70,28)]
==========

Begin optimization report for: FindPosition.._simdsimd3__L4n_v1.V(double)

Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED [main.c(70,28)]
==========

Begin optimization report for: FindPosition.._simdsimd3__L4m_v1.V(double)

Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED [main.c(70,28)]
==========

Analysis of function Interpolate

- Without uniform clause ./main 0m36.828s
- Using uniform clause ./main 0m16.926s
- OpenMP parameter uniform enabled the compiler to use the “fused multiply and add” instruction

```
69    #endif
70    int FindPosition(double x) {
71        return (int)(log(exp(x*steps)));
72    }
73
74    double Interpolate(double x, const point* vals){      2.3%
75
76        int ind = FindPosition(x);                        0.2%
77        const point* pnt = &vals[ind];
78        double res = log(exp(pnt->c0*x+pnt->c1));         13.1%
79
80        return res;                                       0.8%
81    }
82
83
```

```
0x40157b        Block 2:
0x40157b  77    vpmovsxdq %xmm0, %ymm0
0x401580  74    vmovq %r15, %xmm1
0x401585  77    vpsllq $0x4, %ymm0, %ymm2
0x40158a  78    vmovupdy  (%rsp), %ymm0
0x40158f  74    vpbroadcastq %xmm1, %ymm3
0x401594  77    vpaddq %ymm3, %ymm2, %ymm6
0x401598  78    vpcmpeqd %ymm5, %ymm5, %ymm5
0x40159c  78    vxorpd %ymm7, %ymm7, %ymm7
0x4015a0  78    vmovdqa %ymm5, %ymm4
0x4015a4  78    vxorpd %ymm1, %ymm1, %ymm1
0x4015a8  78    vgatherqpdq  %ymm4, (,%ymm6,1), %ymm7
0x4015b2  78    vgatherqpdq  %ymm5, 0x8(,%ymm6,1), %ymm1
0x4015bc  78    vfmadd213pd %ymm1, %ymm7, %ymm0
0x4015c1  78    callq  0x403490 <__svml_exp4>
```

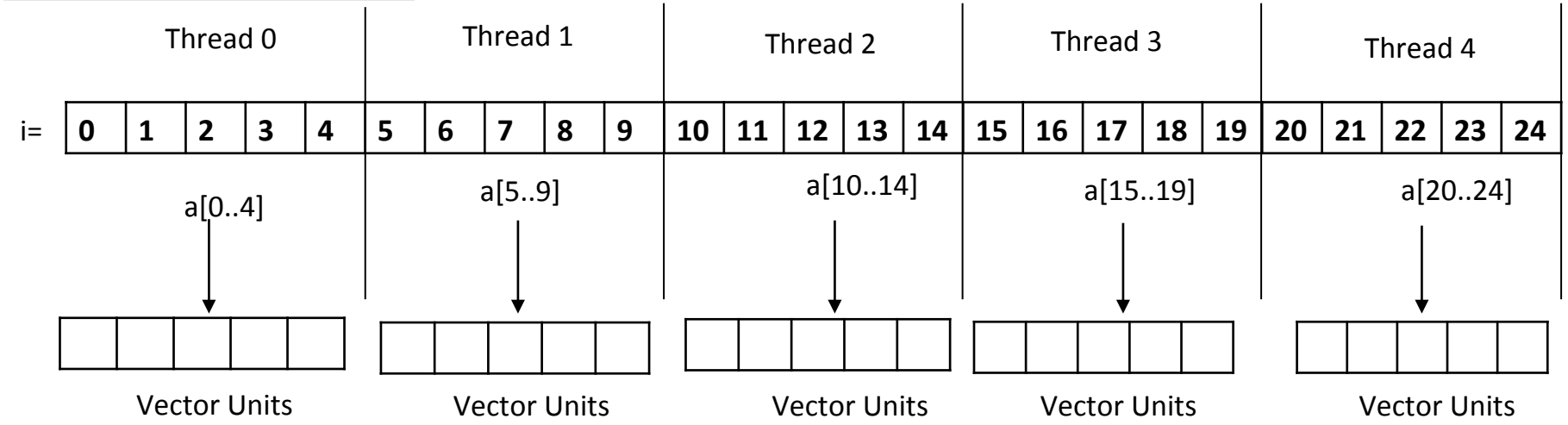
Pragma omp for simd

- Parallelize and vectorize a loop nest
  - Distribute a loop's iteration space across a thread team
  - Subdivide loop chunks to fit a SIMD vector register

- Syntax

```
#pragma omp for simd [clause[[,] clause],…]
for-loops
```

```
N=25;
#pragma omp for simd
for (i=0; i<N; i++)
  a[i] = a[i] + b;
```



Pragma omp for simd

```
#pragma omp parallel for simd
for(i=0; i<msize; i++) {
    a[i][j] = distsq(a[i][j], b[i][j])-auxrand;
    b[i][j] += min(a[i][j], b[i][j])+auxrand;
    c[i][j] = (min(distsq(a[i][j], b[i][j]), a[i][j]))/auxrand;
}
```

Agenda

- NCC Presentation
- Parallel Architectures
- Intel Xeon and Intel Xeon Phi
- OpenMP
- Thread Affinity
- Vectorization
- Offloading
- Thread League
- N-body Simulation

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`

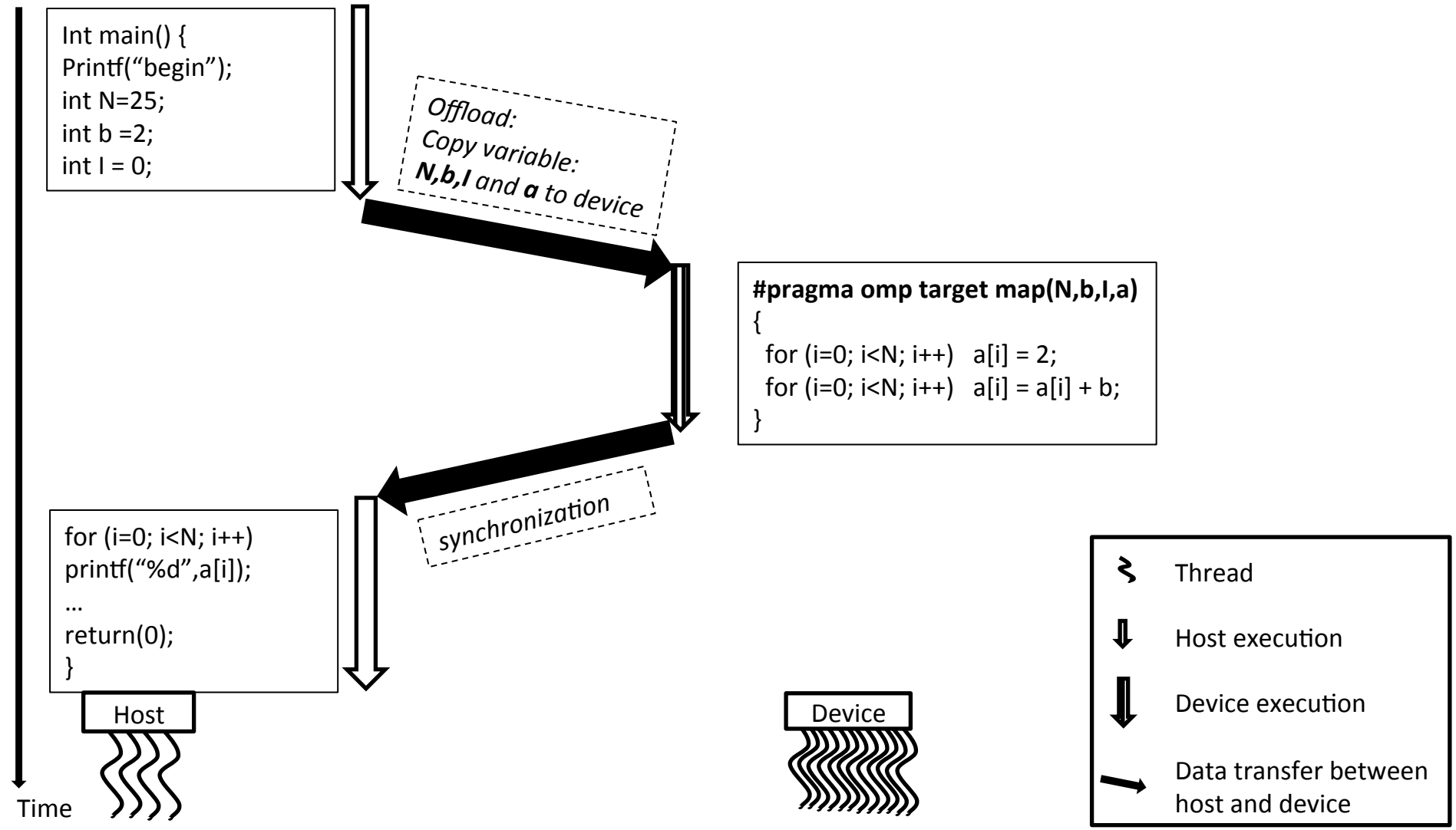
Pragma omp target

- 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:
      - True executes on device;
      - 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



Pragma omp target example

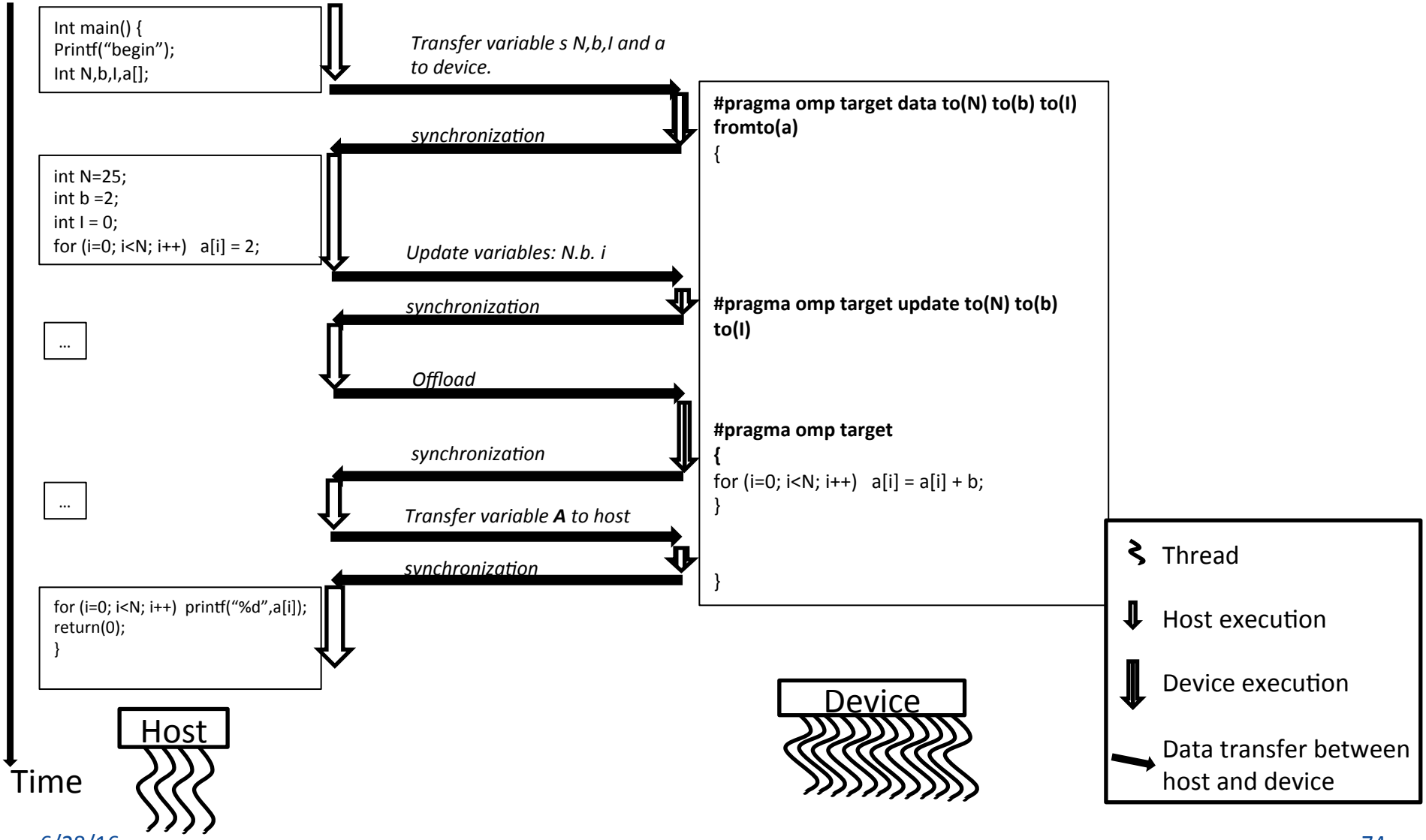
```
#pragma omp target device(0) map(a[0:NUM][0:NUM])
map(b[0:NUM][0:NUM]) map(c[0:NUM][0:NUM])
{
   #pragma omp parallel for collapse (2)
   for(i=0; i<msize; i++) {
      for(k=0; k<msize; k++) {
         #pragma omp simd
         for(j=0; j<msize; j++) {
            c[i][j] = c[i][j] + a[i][k] * b[k][j];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

Pragma omp target example

```
[Offload] [MIC 0] [File]                ../src/multiply.c
[Offload] [MIC 0] [Line]                256
[Offload] [MIC 0] [Tag]                 Tag 0
[Offload] [HOST]  [Tag 0] [CPU Time]        3.705509(seconds)
[Offload] [MIC 0] [Tag 0] [CPU->MIC Data]   402653212 (bytes)
[Offload] [MIC 0] [Tag 0] [MIC Time]        3.246152(seconds)
[Offload] [MIC 0] [Tag 0] [MIC->CPU Data]   402653188 (bytes)
```

- Ellapsed time:
  - Execution time: 16 s;
  - Data transfer (400 MB): 3 s.

Offloading - target data



Pragma omp target data example

```
#pragma omp target data map(to:a[0:NUM][0:NUM]) map(i , j ,k)
map(to:b[0:NUM][0:NUM]) map(tofrom:c[0:NUM][0:NUM])
{
   #pragma omp target
   {
      #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];
         }
      }
   }
}
```

Pragma omp target data example

```
[Offload] [MIC 0] [File]                ../src/multiply.c
[Offload] [MIC 0] [Line]                297
[Offload] [MIC 0] [Tag]                 Tag 0
[Offload] [HOST]  [Tag 0] [CPU Time]       1.594387(seconds)
[Offload] [MIC 0] [Tag 0] [CPU->MIC Data]   402653220 (bytes)
[Offload] [MIC 0] [Tag 0] [MIC Time]       0.000158(seconds)
[Offload] [MIC 0] [Tag 0] [MIC->CPU Data]   0 (bytes)

[Offload] [MIC 0] [File]                ../src/multiply.c
[Offload] [MIC 0] [Line]                299
[Offload] [MIC 0] [Tag]                 Tag 1
[Offload] [HOST]  [Tag 1] [CPU Time]       2.166915(seconds)
[Offload] [MIC 0] [Tag 1] [CPU->MIC Data]   36 (bytes)
[Offload] [MIC 0] [Tag 1] [MIC Time]       3.374661(seconds)
[Offload] [MIC 0] [Tag 1] [MIC->CPU Data]   4 (bytes)

[Offload] [MIC 0] [File]                ../src/multiply.c
[Offload] [MIC 0] [Line]                312
[Offload] [MIC 0] [Tag]                 Tag 2
[Offload] [HOST]  [Tag 2] [CPU Time]       0.014292(seconds)
[Offload] [MIC 0] [Tag 2] [CPU->MIC Data]   56 (bytes)
[Offload] [MIC 0] [Tag 2] [MIC Time]       0.000068(seconds)
[Offload] [MIC 0] [Tag 2] [MIC->CPU Data]   134217740 (bytes)
```

Pragma omp target update

- Update Data between host and device

- Syntax

```
#pragma omp target update [clause[[,] clause],…]
structured-block
```

- Clauses

```
device(scalar-integer-expression)
map(alloc | to | from | tofrom: list)
if(scalar-expr)
```

Pragma omp target update example

```
#pragma omp target data map(to:a[0:NUM][0:NUM]) map(i , j ,k)
map(to:b[0:NUM][0:NUM]) map(to:c[0:NUM][0:NUM])
{
   #pragma omp target
   {
    #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];
      }
     }
    }
   }
   #pragma omp target update from(c[0:NUM][0:NUM])
}
```

Pragma omp target update example

```
[Offload] [MIC 0] [File]                ../src/multiply.c
[Offload] [MIC 0] [Line]                300
[Offload] [MIC 0] [Tag]                 Tag 0
[Offload] [HOST]  [Tag 0] [CPU Time]    1.621304(seconds)
[Offload] [MIC 0] [Tag 0] [CPU->MIC Data]   402653220 (bytes)
[Offload] [MIC 0] [Tag 0] [MIC Time]    0.000151(seconds)
[Offload] [MIC 0] [Tag 0] [MIC->CPU Data]   0 (bytes)

[Offload] [MIC 0] [File]                ../src/multiply.c
[Offload] [MIC 0] [Line]                302
[Offload] [MIC 0] [Tag]                 Tag 1
[Offload] [HOST]  [Tag 1] [CPU Time]    18.781722(seconds)
[Offload] [MIC 0] [Tag 1] [CPU->MIC Data]   36 (bytes)
[Offload] [MIC 0] [Tag 1] [MIC Time]    29.251363(seconds)
[Offload] [MIC 0] [Tag 1] [MIC->CPU Data]   4 (bytes)

[Offload] [MIC 0] [File]                ../src/multiply.c
[Offload] [MIC 0] [Line]                314
[Offload] [MIC 0] [Tag]                 Tag 2
[Offload] [HOST]  [Tag 2] [CPU Time]    0.013202(seconds)
[Offload] [MIC 0] [Tag 2] [CPU->MIC Data]   0 (bytes)
[Offload] [MIC 0] [Tag 2] [MIC Time]    0.000000(seconds)
[Offload] [MIC 0] [Tag 2] [MIC->CPU Data]   134217728 (bytes)

[Offload] [MIC 0] [File]                ../src/multiply.c
[Offload] [MIC 0] [Line]                315
[Offload] [MIC 0] [Tag]                 Tag 3
[Offload] [HOST]  [Tag 3] [CPU Time]    0.002192(seconds)
[Offload] [MIC 0] [Tag 3] [CPU->MIC Data]   56 (bytes)
[Offload] [MIC 0] [Tag 3] [MIC Time]    0.000078(seconds)
[Offload] [MIC 0] [Tag 3] [MIC->CPU Data]   12 (bytes)
```

Agenda

- NCC Presentation
- Parallel Architectures
- Intel Xeon and Intel Xeon Phi
- OpenMP
- Thread Affinity
- Vectorization
- Offloading
- Thread League
- N-body Simulation

Thread League

- **omp teams:** creates a league of thread teams
  - #pragma omp teams [clause [[,] clause] . . .]
    - num_teams(amount) : define the amount of thread teams
    - thread_limit(limit) : define the highest amount of threads that can be created in each team;
- **omp distribute:** distributes a loop over the teams in the league
  - #pragma omp distribute [clause [[,] clause] . . .]
    - dist_schedule (static[block size]):

Thread League

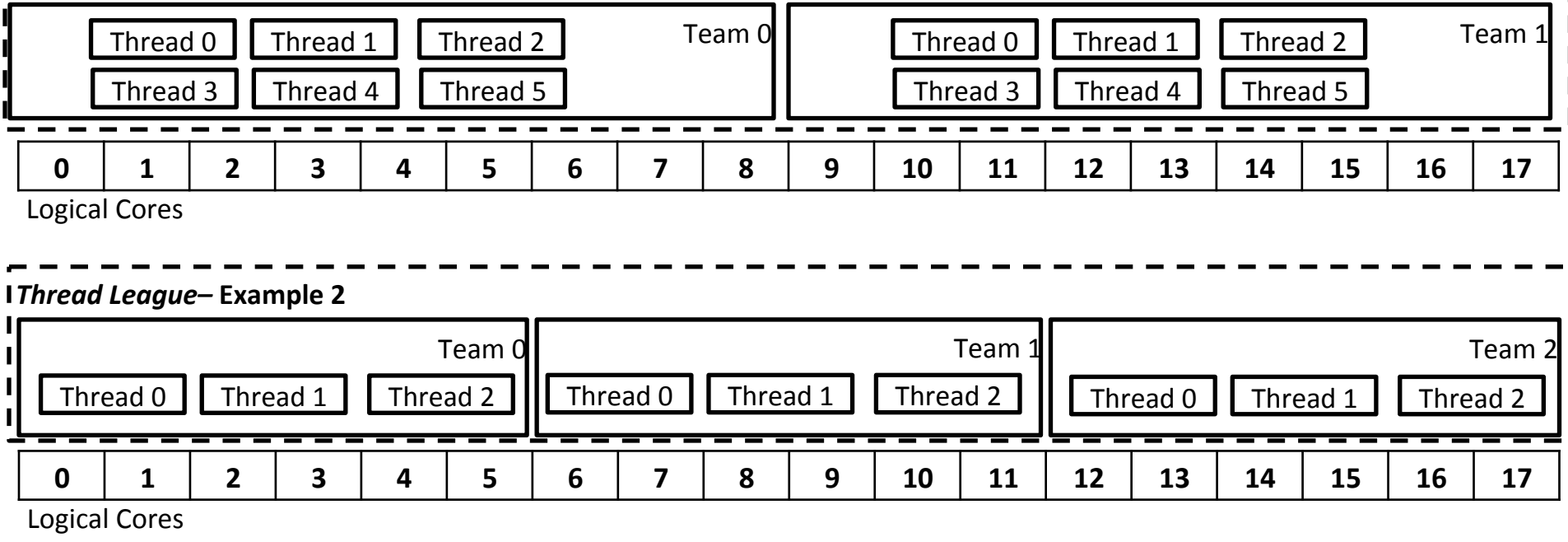
```
#pragma omp target teams num_teams (2) thread_limit (6)
{
   int i , N, teams , idteam , idthread ; int sum; N=20;
   #pragma omp distribute parallel for reduction (+:sum)
    for ( i =0; i<N; i ++) sum += i ;
}
```

Example1

```
#pragma omp target teams num_teams (3) thread_limit (3)
{
   int i , N, teams , idteam , idthread ; int sum; N=20;
   #pragma omp distribute parallel for reduction (+:sum)
    for ( i =0; i<N; i ++) sum += i ;
}
```

Example2

***Thread League*– Example 1**



***Thread League*– Example 2**

Thread League - Example 1

```
#pragma omp target teams num_teams (2) thread_limit( 3 )
{
   int i, N, teams, idteam , idthread;
   int sum;
  N=20;

   #pragma omp distribute parallel for reduction (+: sum)
   for ( i =0; i <N; i ++) {
      sum += i ;
      idthread = omp_get_thread_num ();
      idteam = omp_get_team_num () ;
      teams = omp_get_num_teams () ;
      printf("i %d n %d idteam %d idthread %d teams %d \ n" , i ,N, idteam ,
idthread , teams ) ;
   }
}
```

Thread League - Example 2

```
#pragma omp target data device (0) map (i , j , k) map ( to : a[0:NUM] [0:NUM] ) map ( to : b [ 0 :NUM] [ 0 :NUM] ) map ( tofrom : c [ 0 :NUM] [ 0 :NUM] )
{
   #pragma omp target teams distribute parallel for collapse (2) num_teams (2) thread_limit (30)
   for ( i =0; i <NUM; i ++) {
      for ( k =0; k<NUM; k++) {
         #pragma omp simd
         for ( j =0; j <NUM; j ++) {
            c[i][j] = c[i][j] + a [i][k]  b[k][j] ;
         }
      }
   }
}
```

Thread League - Example 2

```
[Offload] [MIC 0] [File]                ../src/multiply.c
[Offload] [MIC 0] [Line]                277
[Offload] [MIC 0] [Tag]                 Tag 0
[Offload] [HOST]  [Tag 0] [CPU Time]       1.593593(seconds)
[Offload] [MIC 0] [Tag 0] [CPU->MIC Data]   402653220 (bytes)
[Offload] [MIC 0] [Tag 0] [MIC Time]       0.000147(seconds)
[Offload] [MIC 0] [Tag 0] [MIC->CPU Data]   0 (bytes)

[Offload] [MIC 0] [File]                ../src/multiply.c
[Offload] [MIC 0] [Line]                279
[Offload] [MIC 0] [Tag]                 Tag 1
[Offload] [HOST]  [Tag 1] [CPU Time]       3.759050(seconds)
[Offload] [MIC 0] [Tag 1] [CPU->MIC Data]   44 (bytes)
[Offload] [MIC 0] [Tag 1] [MIC Time]       5.854270(seconds)
[Offload] [MIC 0] [Tag 1] [MIC->CPU Data]   12 (bytes)

[Offload] [MIC 0] [File]                ../src/multiply.c
[Offload] [MIC 0] [Line]                288
[Offload] [MIC 0] [Tag]                 Tag 2
[Offload] [HOST]  [Tag 2] [CPU Time]       0.039104(seconds)
[Offload] [MIC 0] [Tag 2] [CPU->MIC Data]   56 (bytes)
[Offload] [MIC 0] [Tag 2] [MIC Time]       0.000073(seconds)
[Offload] [MIC 0] [Tag 2] [MIC->CPU Data]   402653196 (bytes)
```

Agenda

- NCC Presentation
- Parallel Architectures
- Intel Xeon and Intel Xeon Phi
- OpenMP
- Thread Affinity
- Vectorization
- Offloading
- Thread League
- N-body Simulation

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])
      }
   }
```

Thank you for your attention (Obrigado!)

Silvio Stanzani , Raphael Cóbe , Rogério Iope

UNESP - Núcleo de Computação Científica

silvio@ncc.unesp.br , rmcobe@ncc.unesp.br , rogerio@ncc.unesp.br

VECPAR 2016

12th International Meeting on
High Performance Computing for Computational Science