Multithreaded programming on hybrid parallel architectures

Silvio Stanzani , Raphael Cóbe , Rogério Iope, Jefferson Fialho
UNESP - Núcleo de Computação Científica
silvio@ncc.unesp.br , rmcobe@ncc.unesp.br , rogerio@ncc.unesp.br , jfialho@ncc.unesp.br

Agenda

- Hybrid Parallel Architectures
- Intel HPC Architectures
- OpenMP
- Thread Affinity
- Profiling with Intel Advisor (Threading Workflow)

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

Open Positions

email to: cs-jobs[at]ncc unesp br

- High Performance Heterogeneous Computing
- Network Engineer/Architect
- Software-Defined Networking (SDN).
- Grid & Cloud Computing
- Data Science (Machine Learning)

intel

HUAWEI

BR PETROBRAS

Hybrid Parallel Architectures

- Heterogeneous computational systems:
  - Multicore processors;
  - Multi-level memory sub-system;

- Multi-level parallelism:
  - Processing core;
  - Chip multiprocessor;
  - Computing node;
  - Computing cluster;

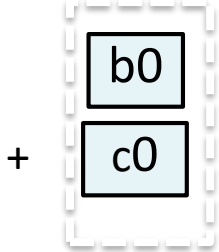
- Hybrid Parallel architectures
  - Coprocessors and accelerators;

Scalar and Vector Instructions

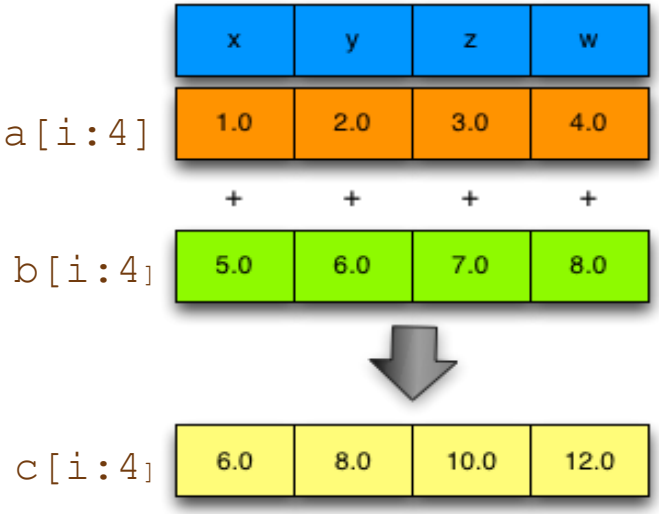
- **Scalar** Code computes this one-element at a time.
- **Vector (or SIMD)** Code computes more than one element at a time.
  - SIMD stands for **S**ingle **I**nstruction **M**ultiple **D**ata.
- Vectorization
  - Loading data into cache accordingly;
  - Store elements on SIMD registers or vectors;
  - Iterations need to be independent;
  - Usually on inner loops.

```
float *A, *B, *C;
for(i=0;i<n;i++){
  A[i] = B[i] + C[i];
}
```

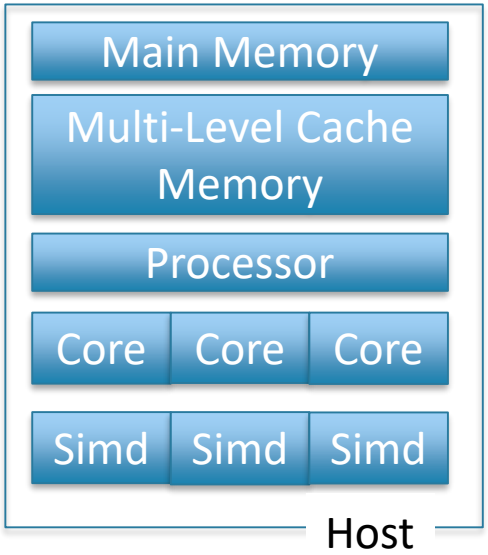
- Scalar



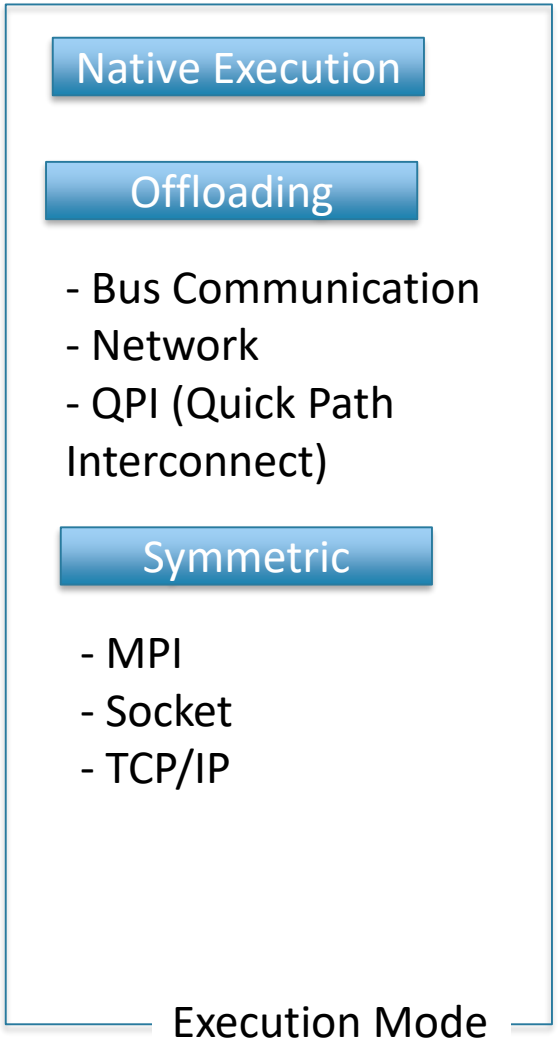
- SIMD



Hybrid Parallel Architectures



- Heterogeneous Multilevel Memory System
- Multilevel parallelism:
  - Instruction Level;
  - Vectorization;
  - Multithreading;
  - Multiprocessing;

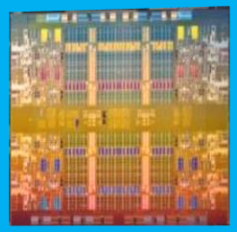


Agenda

- Hybrid Parallel Architectures
- Intel HPC Architectures
- OpenMP
- Thread Affinity
- Profiling with Intel Advisor (Threading Workflow)

Intel Xeon and Intel® Xeon Phi™ Overview

Intel® Multicore Architecture

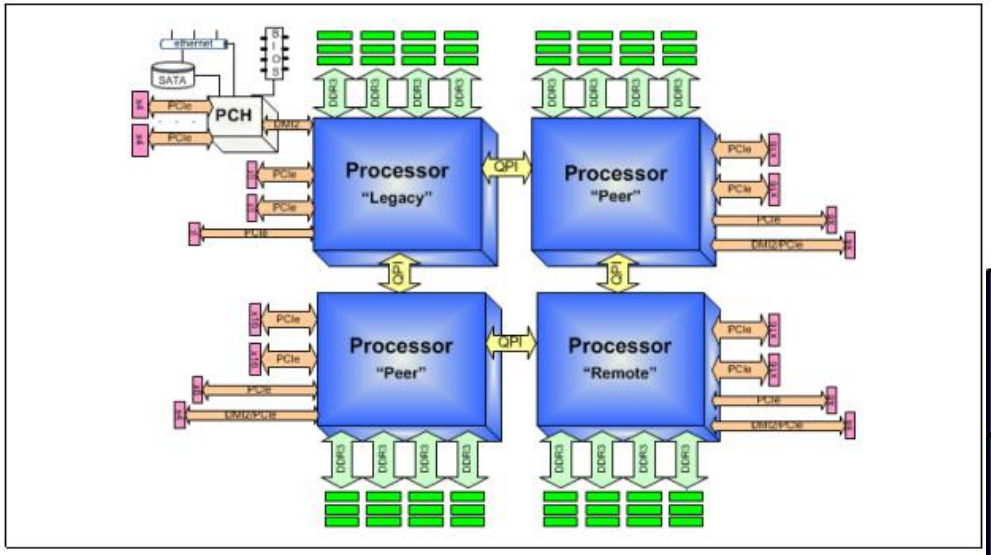


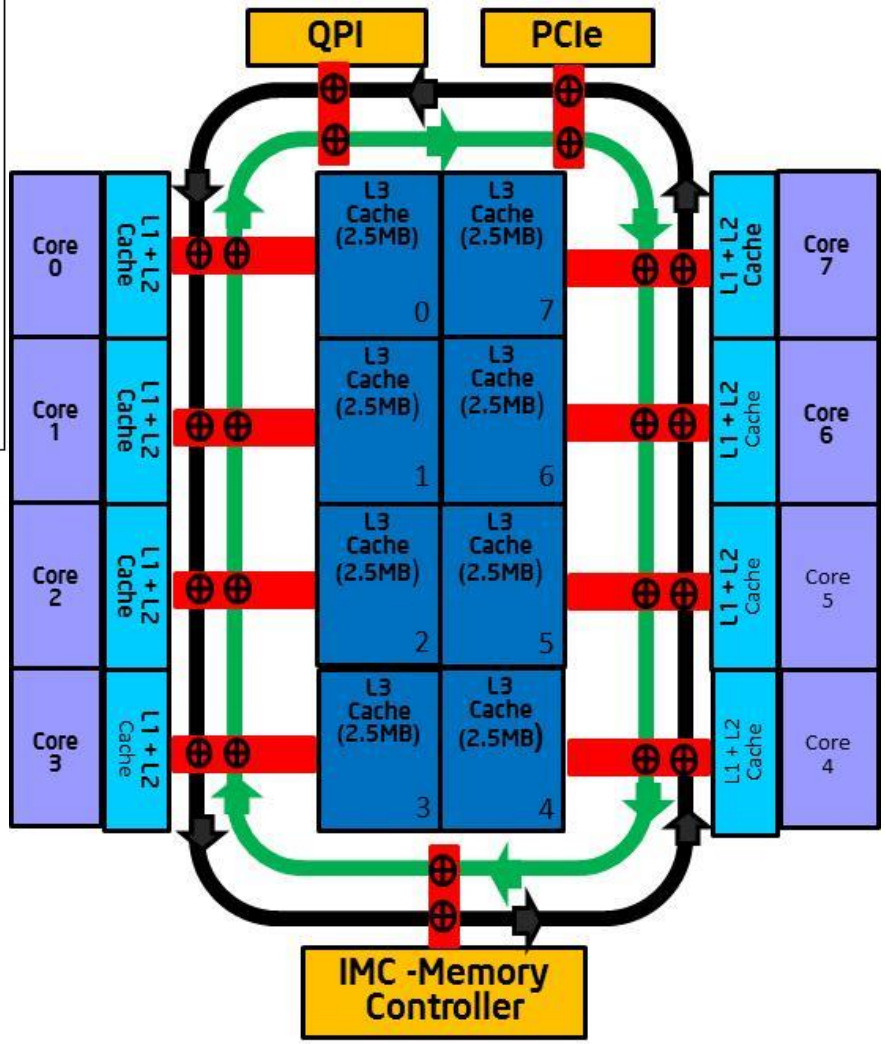
- Foundation of HPC Performance
- Suited for full scope of workloads
- Focus on fast single core/thread performance with “moderate” number of cores

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



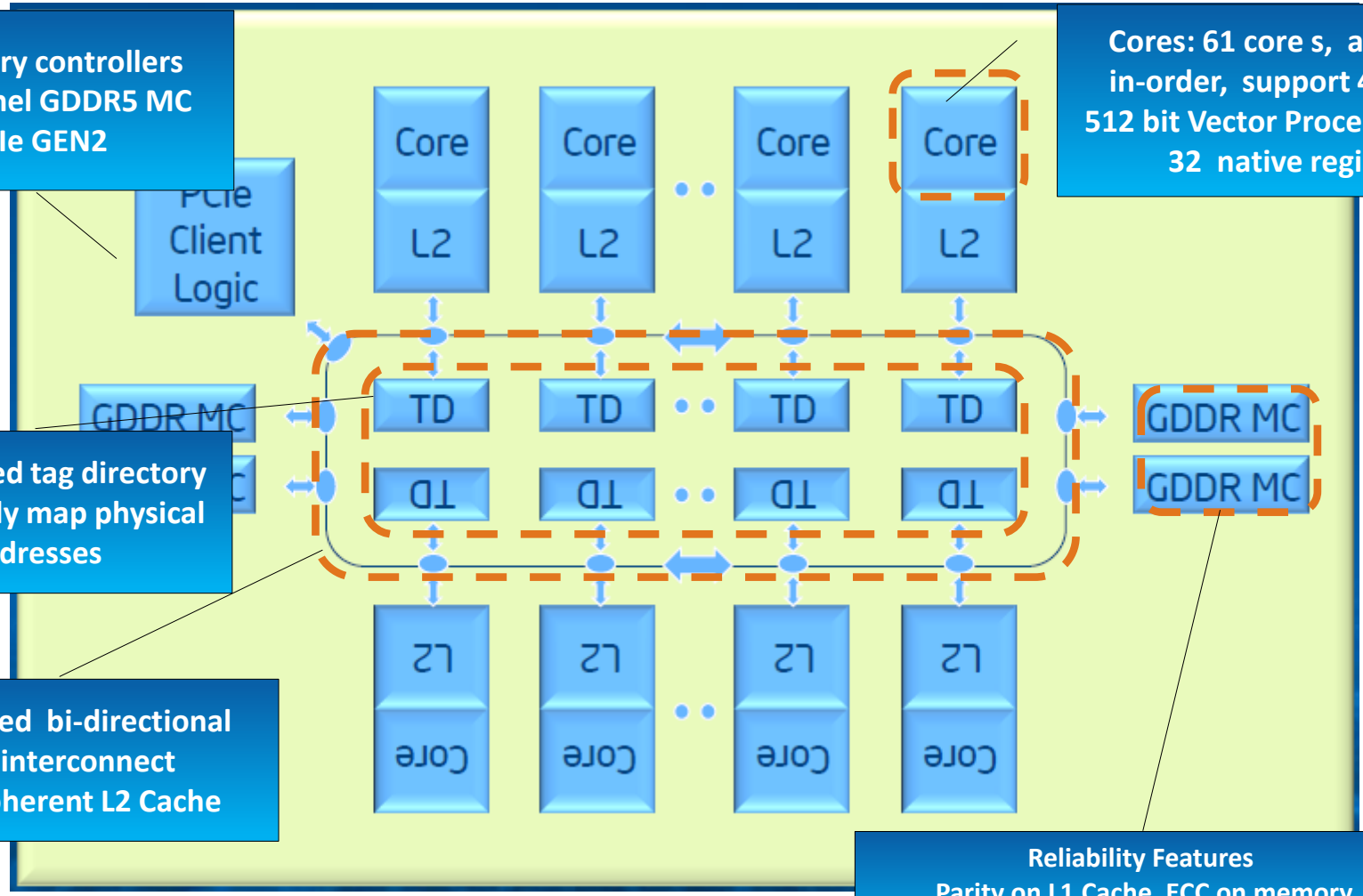


•Socket: mechanical component that provides mechanical and electrical connections between a microprocessor and a printed circuit board (PCB).

•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.

Intel® Xeon Phi™ Architecture Overview

- Knights Core (KNC)



8 memory controllers
16 Channel GDDR5 MC
PCIe GEN2

Cores: 61 core s, at 1.1 GHz
in-order, support 4 threads
512 bit Vector Processing Unit
32 native registers

Distributed tag directory
to uniquely map physical
addresses

High-speed bi-directional
ring interconnect
Fully Coherent L2 Cache

Reliability Features
Parity on L1 Cache, ECC on memory
CRC on memory IO, CAP on memory IO

Intel® Xeon Phi™ Coprocessor Arch – System SW Perspective

- Large SMP UMA machine – a set of x86 cores
  - 4 threads
    - 32 KB L1 I/D
    - 512 KB L2 per core
  - Supports loadable kernel modules
  - VM subsystem, File I/O

- Virtual Ethernet driver
  - supports NFS mounts from Intel® Xeon Phi™ Coprocessor
  - Support bridged network

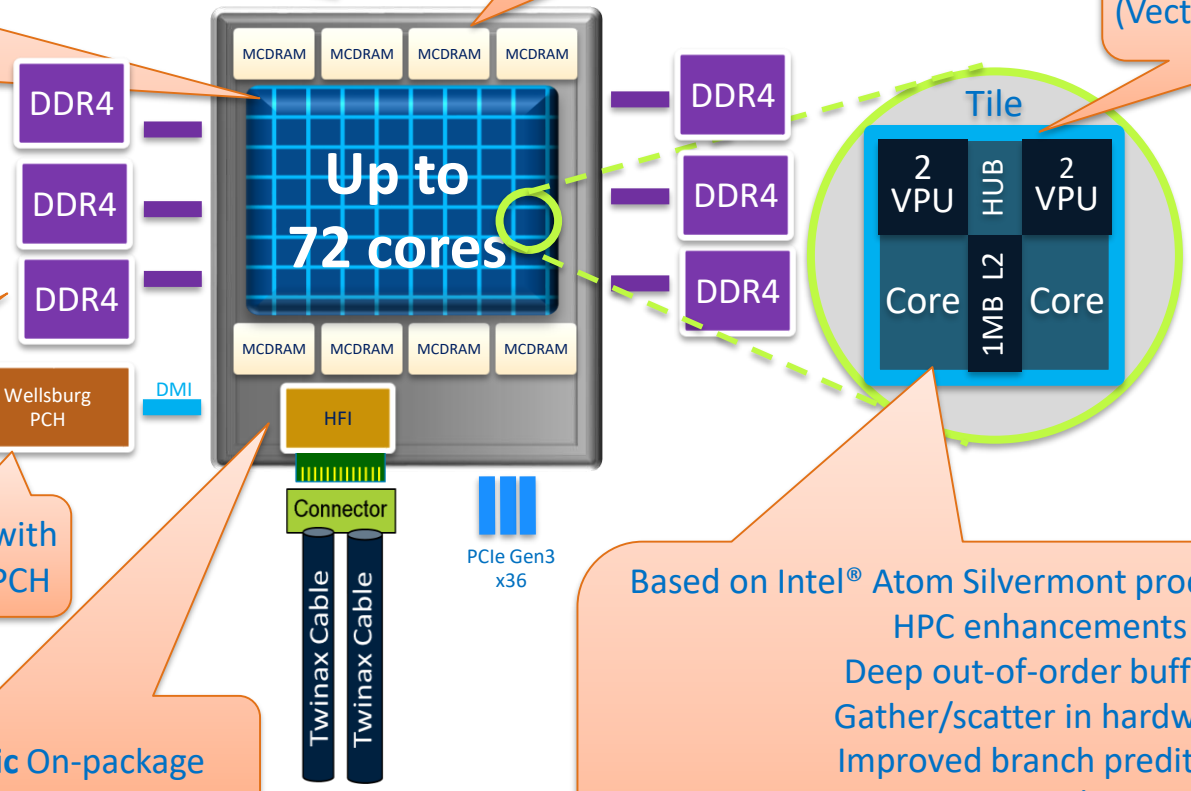
Knights Landing (KNL)

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

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

Up to 72 cores
2D mesh architecture



6 channels
DDR4
Up to
384GB

Common with
Grantley PCH

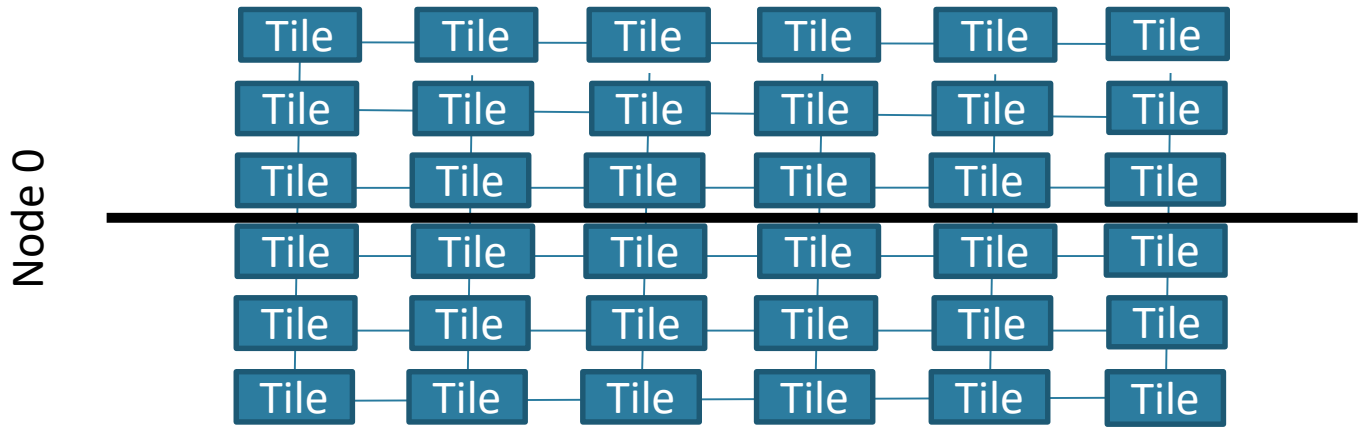
2 ports
Intel Omni-Path Fabric On-package
50 GB/s bi-directional

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

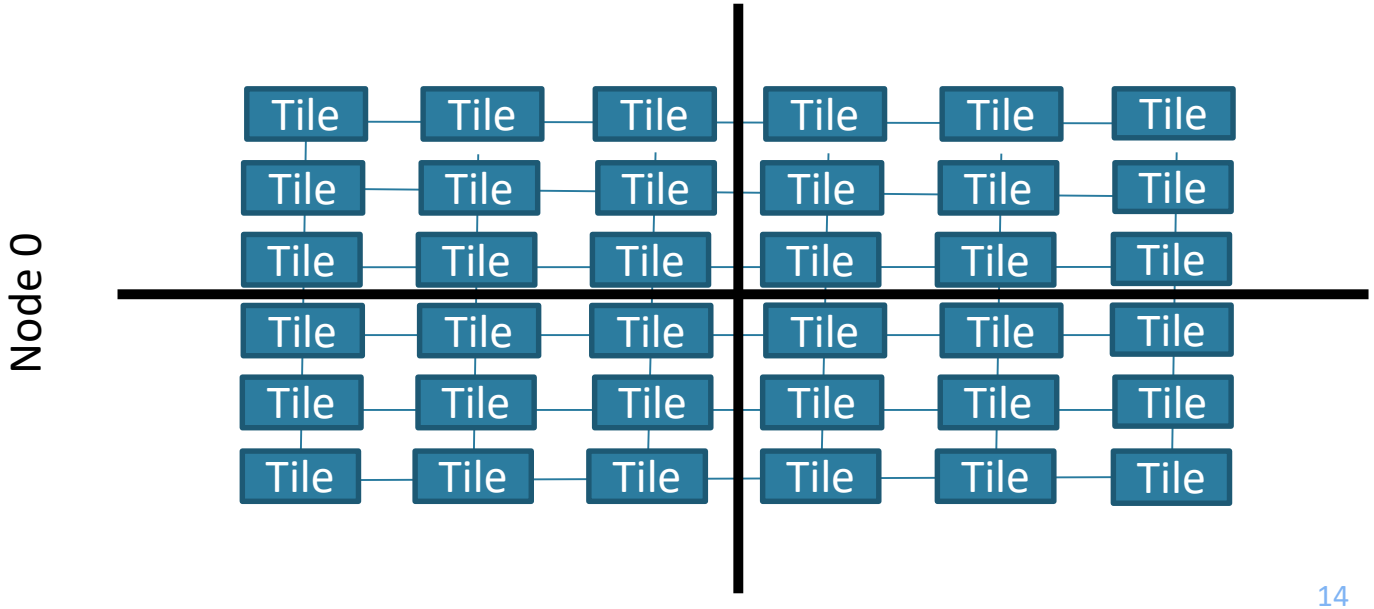
Cluster modes

One single space address

Hemisphere:
the tiles are divided into two parts called hemisphere



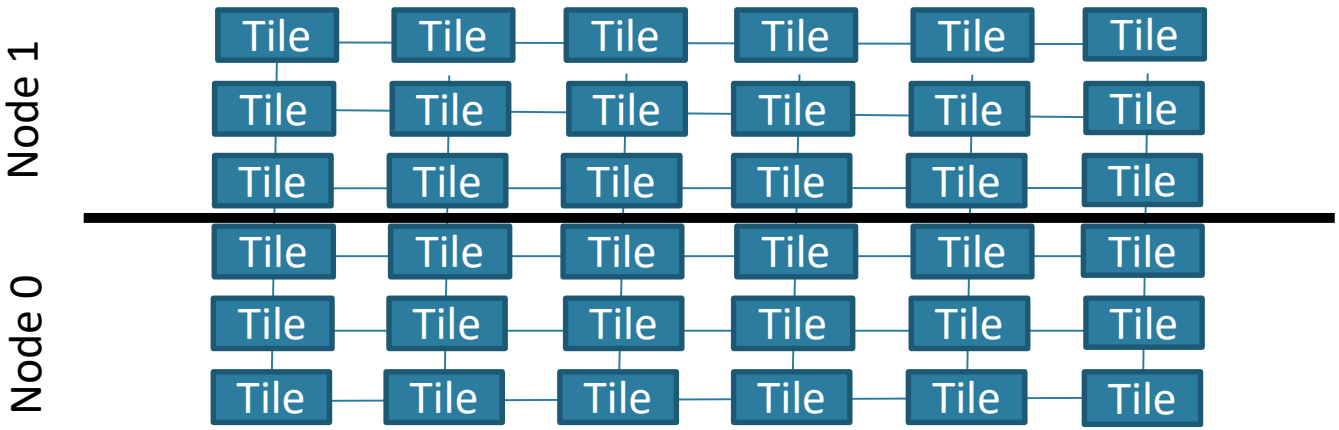
Quadrant:
tiles are divided into two parts called hemisphere or into four parts called qudrants



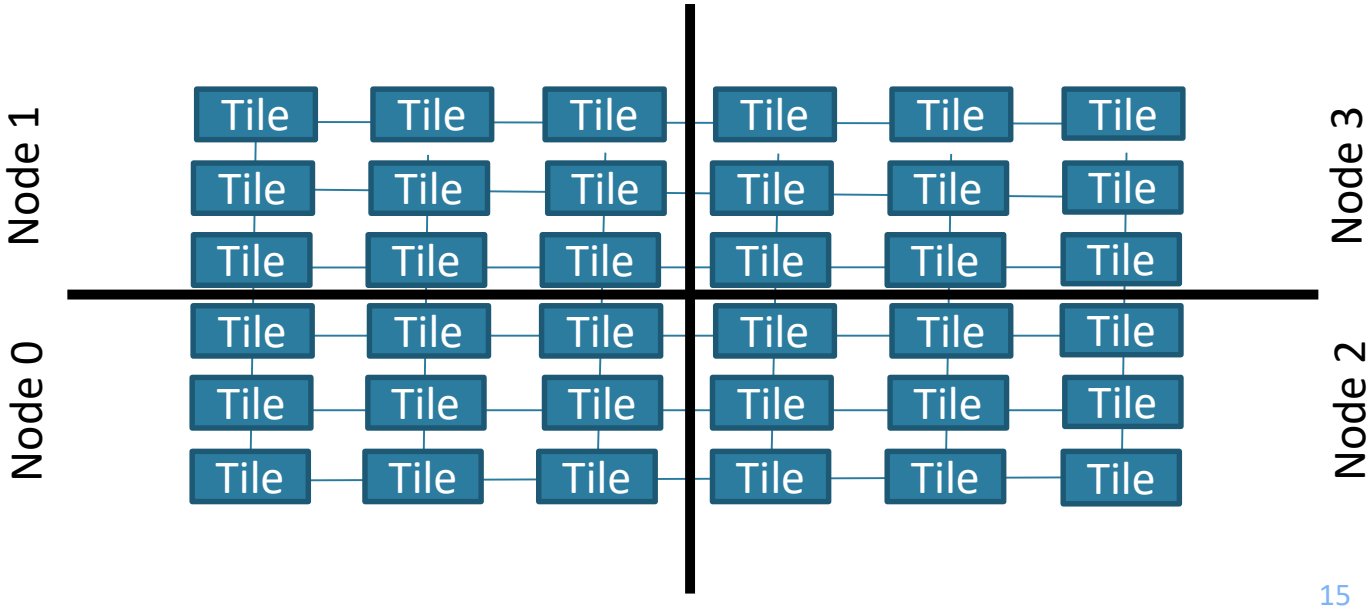
Cluster modes

Cache data are isolated in each sub numa domain

SNC-2:
the tiles are divided into two Numa Nodes

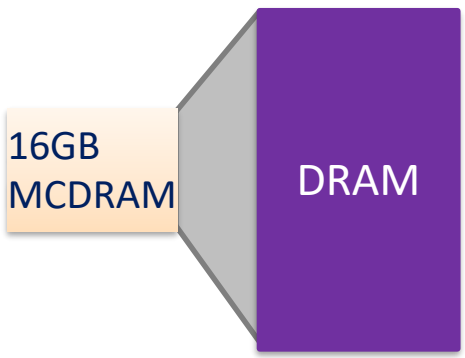


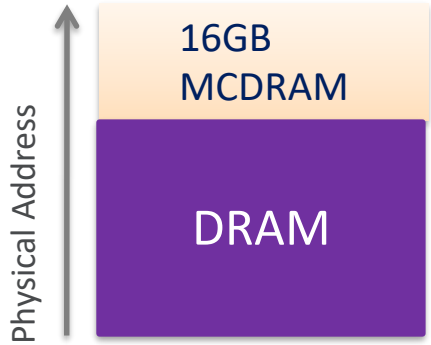
SNC-4:
the tiles are divided into two Numa Nodes

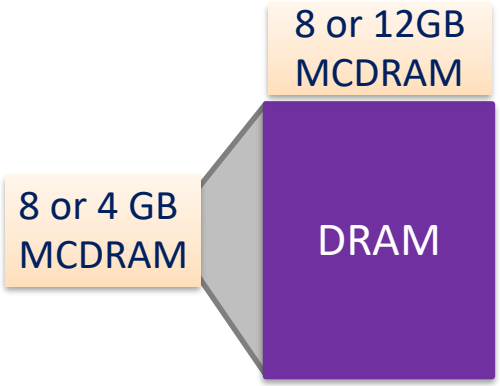


Integrated On-Package Memory Usage Models

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
▪ App and/or data set is very large and will not fit into MCDRAM ▪ Unknown or unstructured memory access behavior	▪ 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	▪ 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

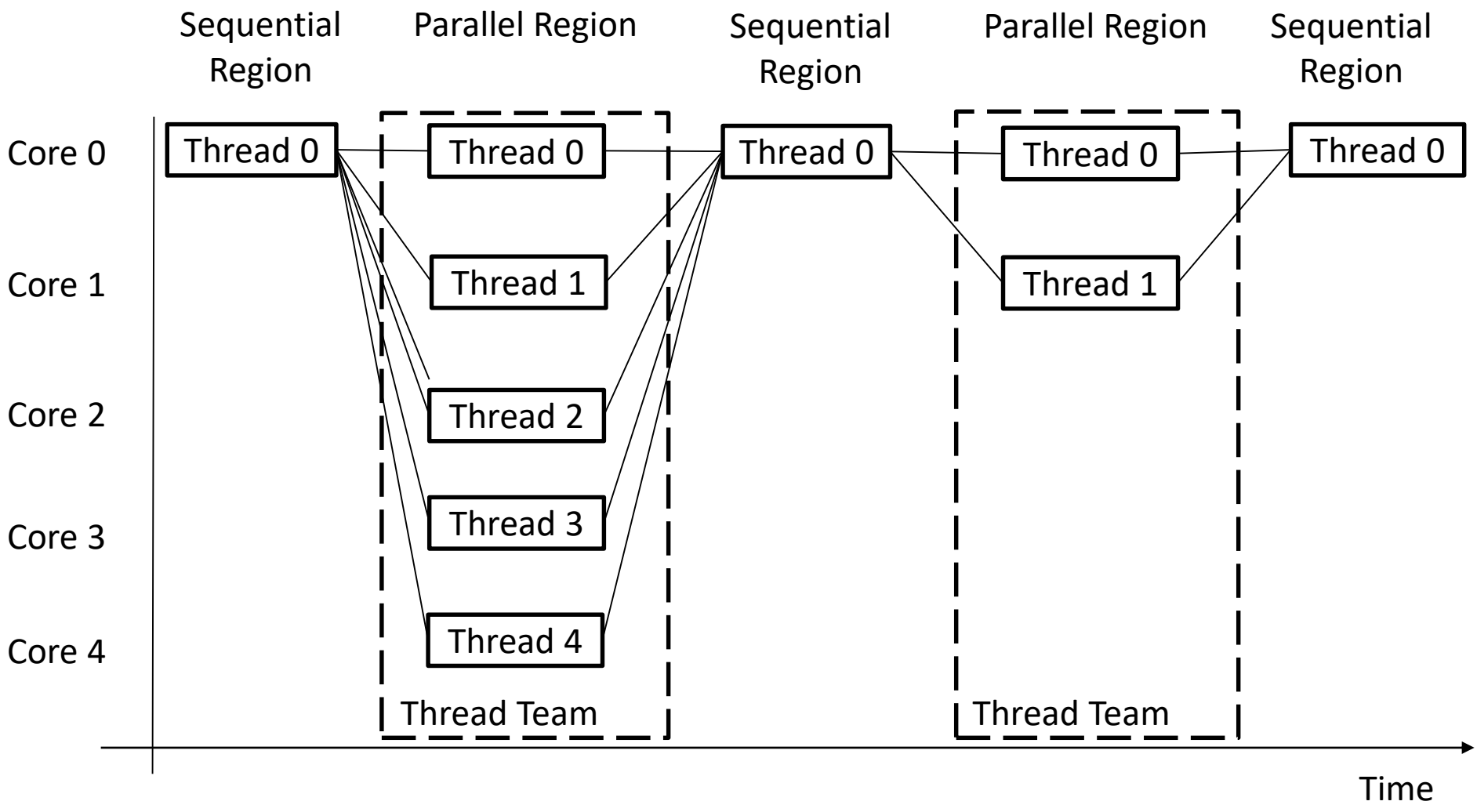
Agenda

- Hybrid Parallel Architectures
- Intel HPC Architectures
- OpenMP
- Thread Affinity
- Profiling with Intel Advisor (Threading Workflow)

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 Team
Thread 0
Thread 0
Thread 1
Thread Team
Thread 0
Time

OpenMP - Core elements

Parallel control structures

- Form a team of threads and execute them in parallel
- `omp parallel`

Work sharing

- Distribute work among threads
- `omp [parallel] loop`
  `omp [parallel] sections`
  `omp [parallel] workshare`
  `omp single`

Data environment

- Control variables scope
- `omp threadprivate`
  `shared/*private` clauses

Synchronization

- Coordinates thread execution
- `omp atomic`
  `omp barrier`
  `omp critical`
  `omp flush`
  `omp master`
  `omp ordered`
  `omp taskgroup`
  `omp taskwait`

OpenMP 4.0

SIMD vectorization

- Control execution on hardware supporting SIMD instructions
- `omp simd`
  `omp declare simd`
  `omp [parallel] loop simd`

Offload execution

- Control execution on coprocessor devices
- `omp declare target`
  `omp distribute`
  `omp target [data|update]`
  `omp teams`

Thread affinity

Tasking

- Structures for deferring execution
- `omp task`
  `omp taskyield`

Runtime environment

- Runtime functions and environment variables
- `omp_set_num_theads()`, etc.
  `OMP_SCHEDULE`, etc.

OpenMP Sample Program

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

	Thread 0					Thread 1					Thread 2					Thread 3					Thread 4				
i=	**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**

OpenMP Sample Program

```
#include <stdio.h>
int main(){
   int r, c, i, j, *a , *b , *sum;
   char hn[600];

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

   r=40000;
   c=40000;

   a = (int*)malloc(r*c*sizeof(double));
   b = (int*)malloc(r*c*sizeof(double));
   sum = (int*)malloc(r*c*sizeof(double));

   #pragma omp parallel for
      for(i=0; i<r; ++i)
         for(j=0; j<c; ++j) {
           a[i*r + j]=i+j;
           b[i*r + j]=i-j;
         }

      #pragma omp parallel for
      for(i=0;i<r;++i)
         for(j=0;j<c;++j)
            sum[i*r+j] = a[i*r+j] + b[i*r+j];

      free(a);
      free(b);
      free(sum);

      return 0;
}
```

Compiling and running an OpenMP application

#Build the application for Multicore Architecture (Xeon)

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

#Launch the application on host

./*omp_binary*

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

Agenda

- Hybrid Parallel Architectures
- Intel HPC Architectures
- OpenMP
- Thread Affinity
- Profiling with Intel Advisor (Threading Workflow)

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:

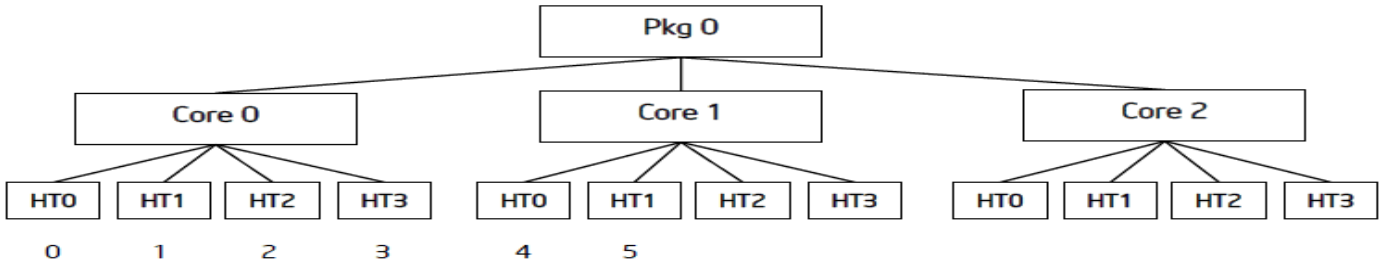
```
KMP_AFFINITY=
    [<modifier>,...]
    <type>
    [,<permute>]
    [,<offset>]
```

Example:

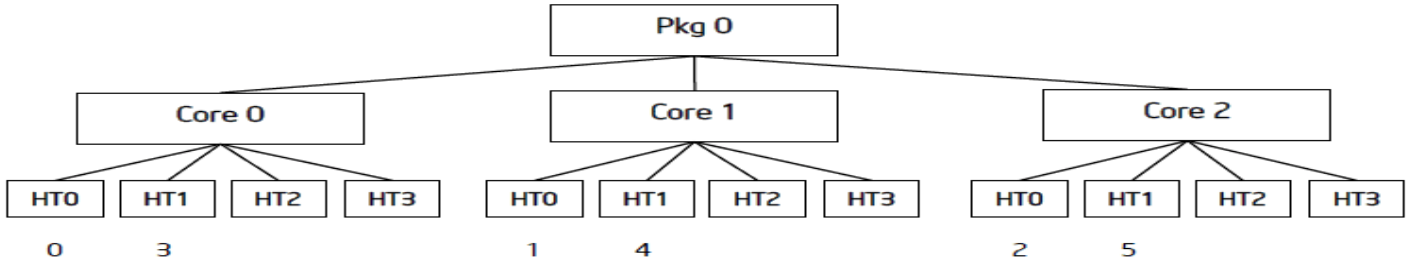
export KMP_AFFINITY=scatter

KMP_AFFINITY - Types

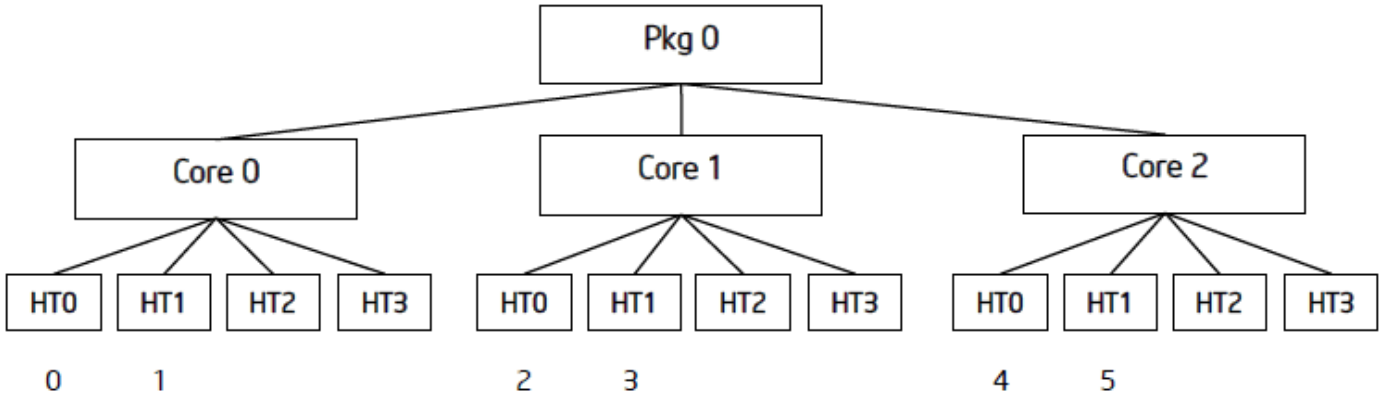
- Compact



- Scatter



- Balanced



Thread Affinity Examples

compact xeon

```
export KMP_AFFINITY=compact,verbose
./OMP_hello
```

scatter xeon

```
export KMP_AFFINITY=scatter,verbose
./OMP_hello
```

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 x scatter

thread 0 bound to OS proc set {0,36}
thread 1 bound to OS proc set {0,36}
thread 2 bound to OS proc set {1,37}
thread 3 bound to OS proc set {1,37}
thread 4 bound to OS proc set {2,38}
thread 5 bound to OS proc set {2,38}
thread 6 bound to OS proc set {3,39}
thread 7 bound to OS proc set {3,39}
thread 8 bound to OS proc set {4,40}
thread 9 bound to OS proc set {4,40}

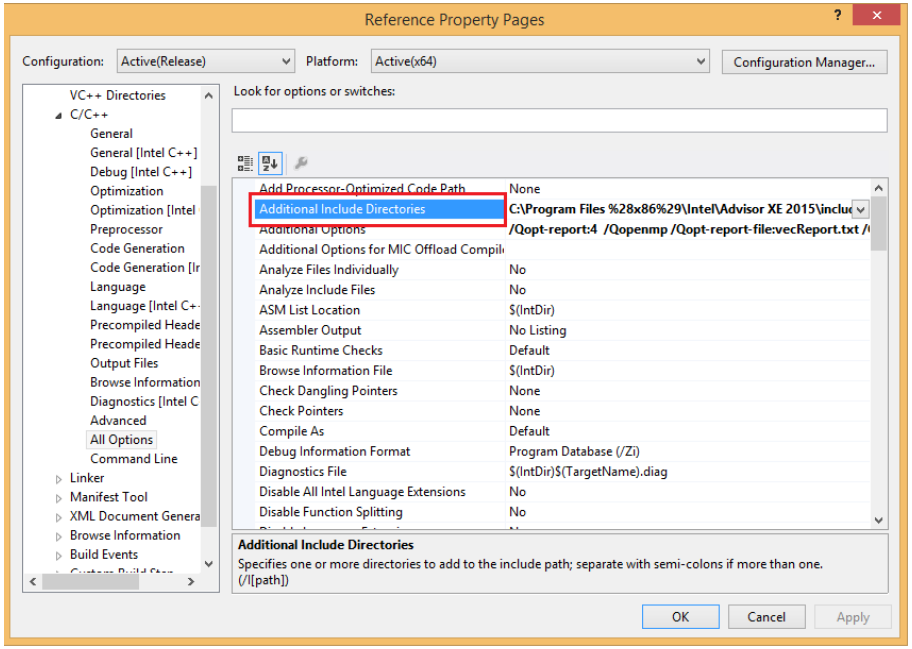
thread 0 bound to OS proc set {0,36}
thread 1 bound to OS proc set {18,54}
thread 2 bound to OS proc set {1,37}
thread 3 bound to OS proc set {19,55}
thread 4 bound to OS proc set {2,38}
thread 5 bound to OS proc set {20,56}
thread 6 bound to OS proc set {3,39}
thread 7 bound to OS proc set {21,57}
thread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

Agenda

- Hybrid Parallel Architectures
- Intel HPC Architectures
- OpenMP
- Thread Affinity
- Profiling with Intel Advisor (Threading Workflow)

Identifying Parallelization Opportunities

- Intel Advisor steps:
  - 1º - Include headers
  - #include "advisor-annotate.h"
  - 2º - add include reference ; link library



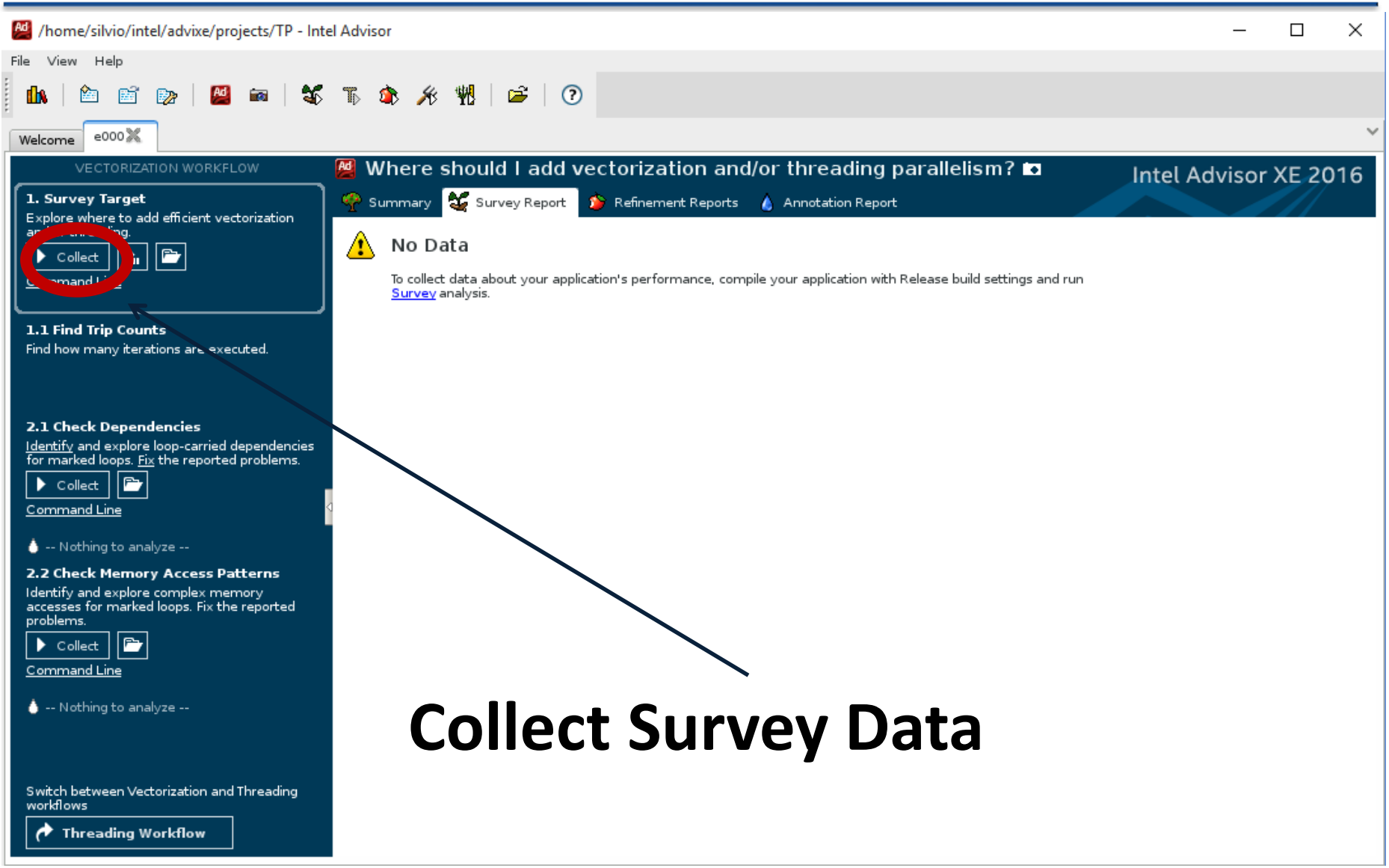
<u>Linux – compiling / link with Advisor</u>

icpc -O2 -openmp 02_ReferenceVersion.cpp -o 02_ReferenceVersion -I/opt/intel/advisor/include/ -L/opt/intel/advisor/lib64/

Identifying Parallelization Opportunities

- Intel Advisor Analysis:
  - Survey
    - Vectorization of loops: detailed information about vectorization;
    - Total Time: elapsed time in each loop considering the time involved in internal loops;
    - Self Time: elapsed time in each loop without internal loops;
  - Suitability
    - Speedup gains obtained parallelizing annotated loops;

Intel Advisor - Survey Data



Collect Survey Data

Intel Advisor - Survey Data

Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Type	Why No Vectorization?
[loop in multiply3 at multiply.c:228]		2 Assume ...	0.170s	0.170s	Scalar	vector dependence prevents vectorization
[loop in __libc_csu_init]			0.000s	0.000s	Scalar	
[loop in _INTERNAL_16_offload_host_cpp_ad92 ...			0.000s	0.000s	Scalar	
[loop in func@0x5b810]		2 Data typ ..	0.000s	0.000s	Scalar	
[loop in main at matrix.c:144]		2 Data typ ..	0.000s	0.000s	Scalar	inner loop was already vectorized
[loop in multiply3 at multiply.c:227]		2 Assume ...	0.000s	0.170s	Scalar	vector dependence prevents vectorization
[loop in multiply3 at multiply.c:226]		2 Assume ...	0.000s	0.170s	Scalar	vector dependence prevents vectorization
[loop in main at matrix.c:144]		1 Data typ ..	0.000s	0.000s	Vectorized (B ...	

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

File: multiply.c:228 multiply3

```
Line  Source
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)
226         for(i=0; i<msize; i++) {
            [loop in multiply3 at multiply.c:226]
                Scalar loop. Not vectorized: vector dependence prevents vectorization
                No loop transformations applied
227             for(k=0; k<msize; k++) {
            [loop in multiply3 at multiply.c:227]
                Scalar loop. Not vectorized: vector dependence prevents vectorization
                Remainder loop
228                 for(j=0; j<msize; j++) {
            [loop in multiply3 at multiply.c:228]
                Scalar loop. Not vectorized: vector dependence prevents vectorization
                Loop was unrolled by 2
229                     c[i][j] = c[i][j] + a[i][k] * b[k][j];
230                 }
231             }
232         }
233       //}
234     }
235
```

Matrix Multiplication

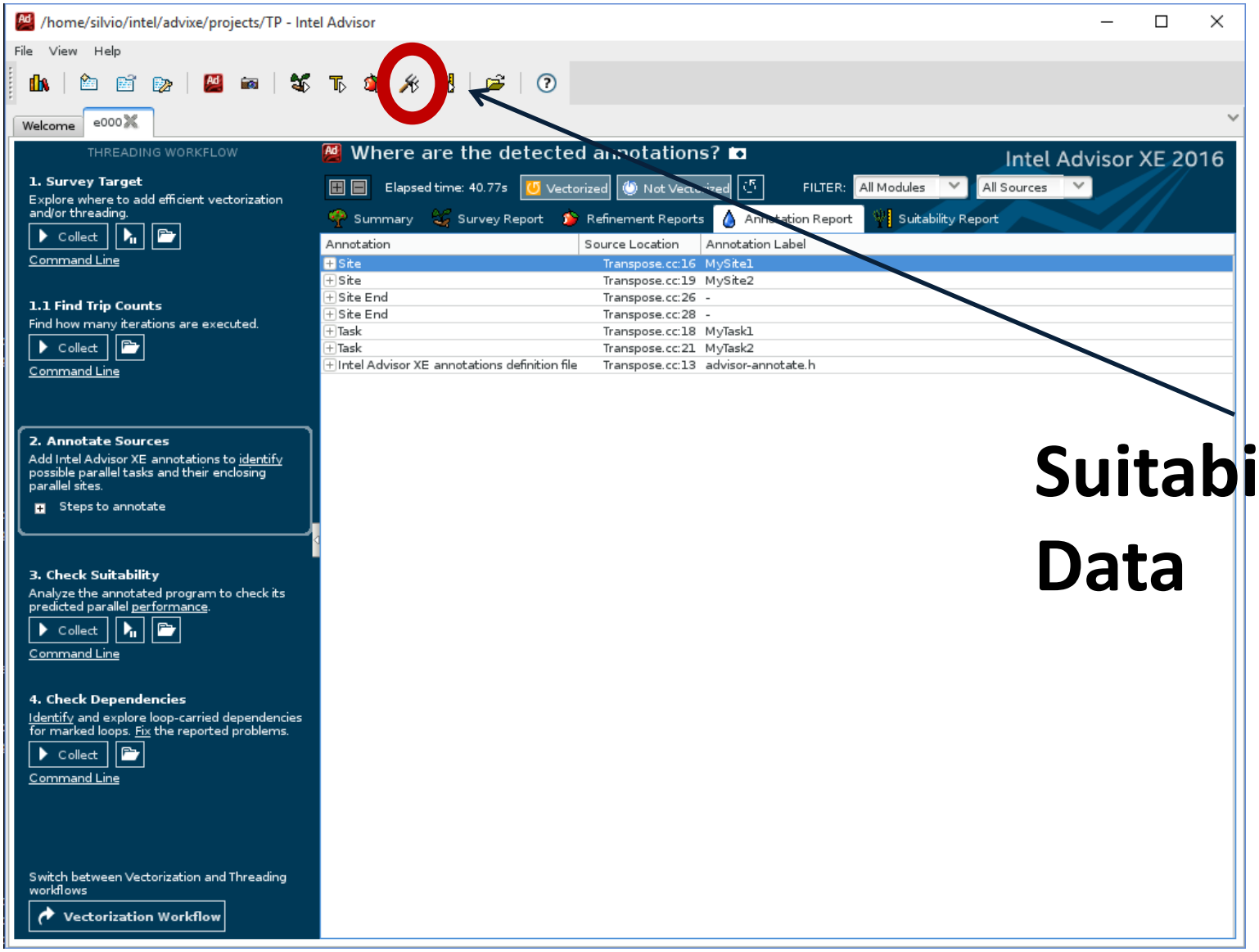
```
for(i=0; i<msize; i++) {
    for(j=0; j<msize; j++) {
     for(k=0; k<msize; k++) {
            c[i][j] = c[i][j] + a[i][k] * b[k][j];
        }
     }
  }
```

Intel Advisor – Check Suitability

- Inserting advisor **Annotations key words** for Check Suitability:
  - **ANNOTATE_SITE_BEGIN(id)**: before beginning of loop;
  - **ANNOTATE_ITERATION_TASK(id)**: first line inside the loop;
  - **ANNOTATE_SITE_END()**: after end of loop;

```
ANNOTATE_SITE_BEGIN( MySite1 );
for(i=0; i<msize; i++) {
   ANNOTATE_ITERATION_TASK( MyTask1 );
   for(k=0; k<msize; k++)
      for(j=0; j<msize; j++) {
         c[i][j] = c[i][j] + a[i][k] * b[k][j];
}
ANNOTATE_SITE_END();
```

Intel Advisor – Check Suitability



Suitability Data

Intel Advisor – Check Suitability

Instances of task MyTask3 are too small. Suitability data may be unreliable.

View Source

Maximum Program Gain For All Sites: 7.85x

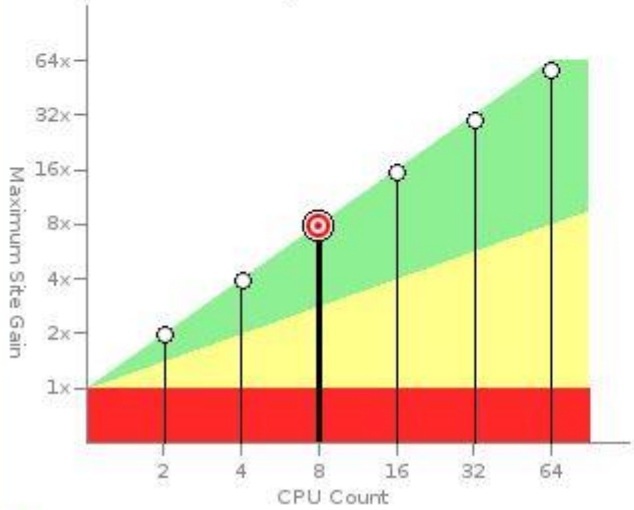
Serial time: 16.231s
Predicted Parallel time: 2.067s

Target System: CPU | Threading Model: Intel TBB | CPU Count: 8

Site Label	Source Location	Impact to Program Gain	Combined Site Metrics, All Instances		
			Total Serial Time	Total Parallel Time	Site Gain
MySite1	multiply.c:179	7.82x	16.21s	2.056s	7.88x
MySite2	multiply.c:182	6.95x	16.21s	2.314s	7.01x
MySite3	multiply.c:185	0.15x	15.75s	104.47s	0.15x

Site Performance Scalability | Site Details

Scalability of Maximum Site Gain



Loop Iterations (Tasks) Modeling

Avg. Number of Iterations (Tasks): 1024

0.008x
0.040x
0.200x
1x (1024)
5x
25x
125x

Avg. Iteration (Task) Duration: 0.016s

0.008x
0.040x
0.200x
1x (0.016s)
5x
25x
125x

Apply

Runtime Modeling

Type of Change | Gain Ber

- Reduce Site Overhead
- Reduce Task Overhead
- Reduce Lock Overhead
- Reduce Lock Contention
- Enable Task Chunking

Agenda

- Hybrid Parallel Architectures
- Intel HPC Architectures
- OpenMP
- Thread Affinity
- Profiling with Intel Advisor (Threading Workflow)

Critical section

- In concurrent programming, concurrent accesses to shared resources can lead to unexpected or erroneous behavior;

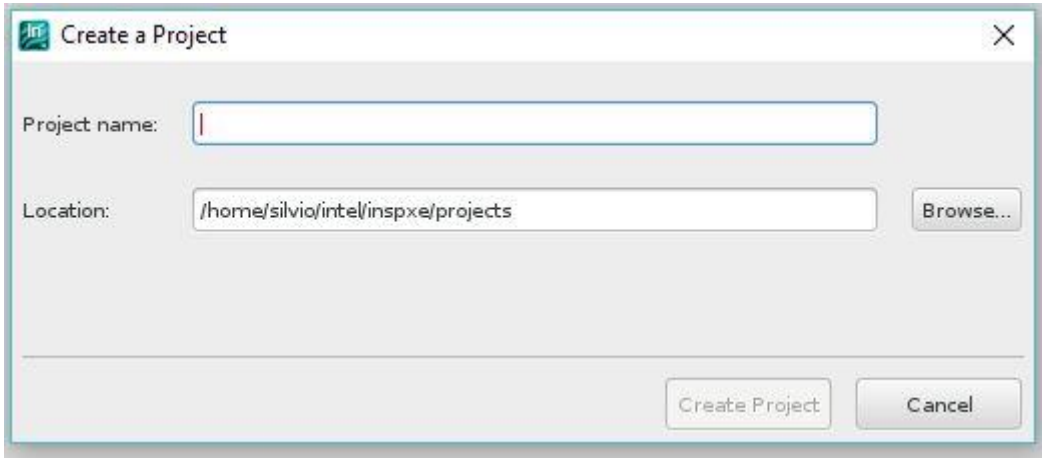
- Regions of code where the shared resource is accessed, has to be protected against concurrent access and is known as critical section;

- Challenge:
  - Identify critical sections;
  - Impose synchronization without loss of performance.

Intel Inspector

- Intel Inspector is a debugger tool that performs dynamic analysis. It is capable of identify the following errors:
  - **Memory Errors**
    - Memory leaks;
    - Memory corruption;
    - Allocation / de-allocation API mismatches
    - Inconsistent memory API usage;
    - Illegal memory access;
    - Uninitialized memory read;
  - **Threading Errors**
    - Data races:
      - Heap races;
      - Stack races;
    - Deadlocks;

Intel Inspector



Intel Inspector

multi-thread - Project Properties

Target | Suppressions | Binary/Symbol Search | Source Search

Specify and configure your analysis target: an application or a script to execute. Press F1 for more details.

No application executable (target) file specified.

Application: Browse...

Application parameters: Modify...

Use application directory as working directory

Working directory: Browse...

User-defined environment variables:

Modify...

Store result in the project directory: /home/silvio/intel/inspxe/projects/multi-thread

Store result in (and create link file to) another directory

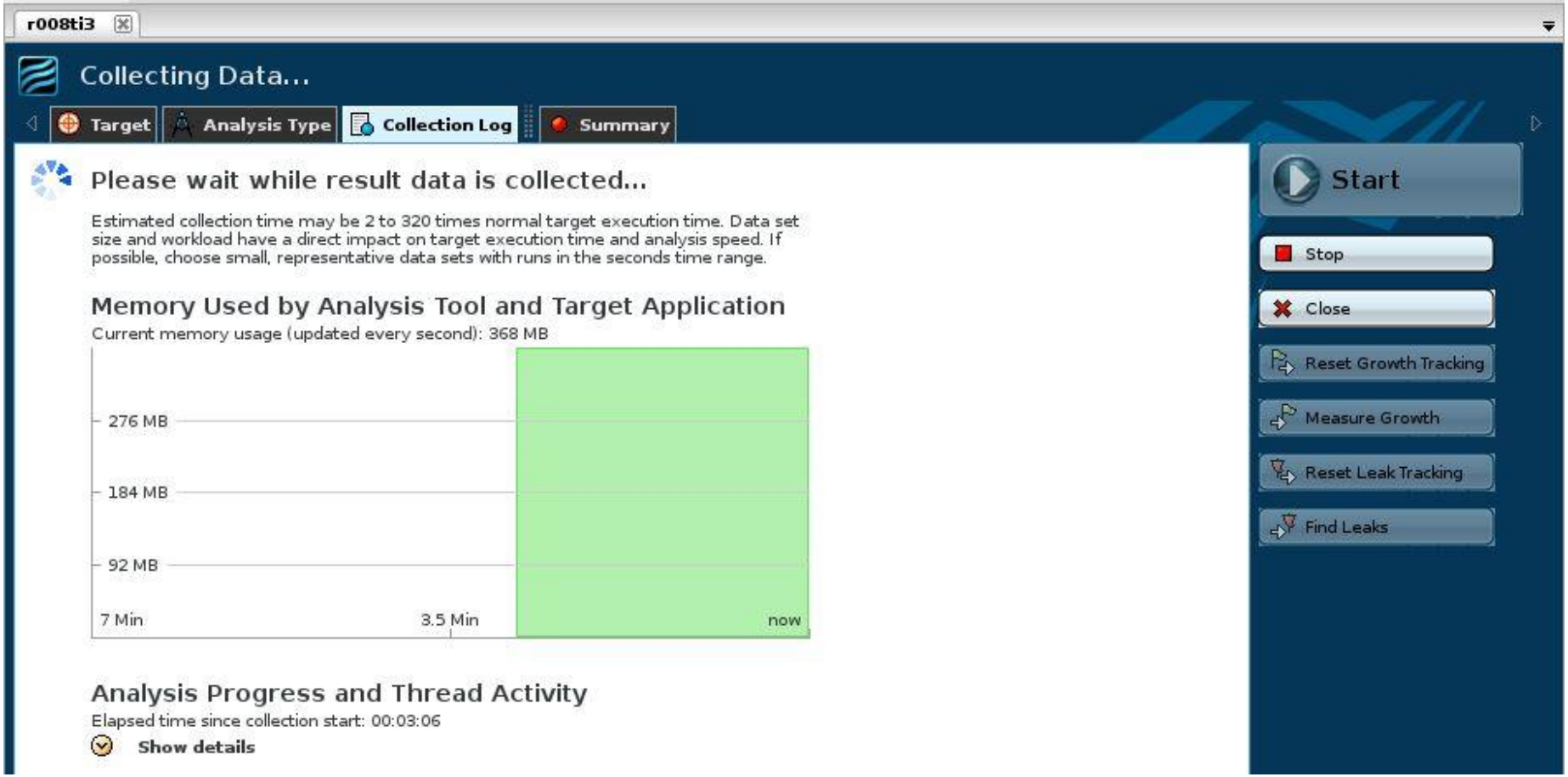
/home/silvio/intel/inspxe/projects/multi-thread Browse...

Result location:
/home/silvio/intel/inspxe/projects/multi-thread/r@@@{at}

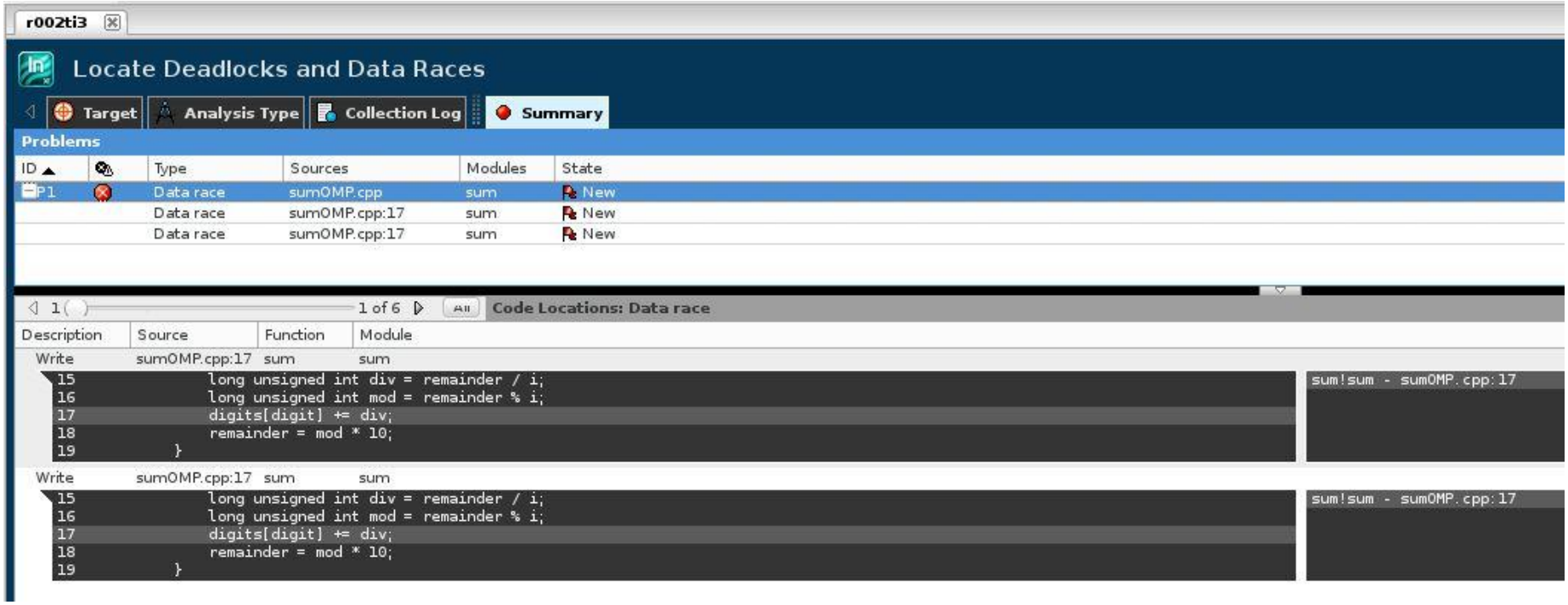
Advanced

OK Cancel

Intel Inspector



Intel Inspector



Tachyon

- **Tachyon:** a parallel/multiprocessor ray tracing software.
- Variable col is declared as global, but used by several threads;

r000ti3

Locate Deadlocks and Data Races

Target | Analysis Type | Collection Log | Summary

Problems

ID	Type	Sources	Modules	State
P1	Data race	find_and_fix_threading_errors.cpp	tachyon.find_and_fix_threading_errors	New
	Data race	find_and_fix_threading_errors.cpp:105	tachyon.find_and_fix_threading_errors	New
	Data race	find_and_fix_threading_errors.cpp:105; find_a ...	tachyon.find_and_fix_threading_errors	New
	Data race	find_and_fix_threading_errors.cpp:105; find_a ...	tachyon.find_and_fix_threading_errors	New
	Data race	find_and_fix_threading_errors.cpp:105; find_a ...	tachyon.find_and_fix_threading_errors	New
P2	Data race	xvideo.cpp	tachyon.find_and_fix_threading_errors	New
	Data race	xvideo.cpp:264	tachyon.find_and_fix_threading_errors	New
	Data race	xvideo.cpp:264	tachyon.find_and_fix_threading_errors	New

Filters

Severity
Error

Type
Data race

Source
find_and_fix_threading_errors.cpp
xvideo.cpp

Module
tachyon.find_and_fix_threading_errors

State
New

1 18 of 18 All Code Locations: Data race

Description	Source	Function	Module
Read	find_and_fix_threading_errors.cpp:149	render_one_pixel	tachyon.find_and_fix_threading_errors

```
147    else if (G < 0) G = 0;
148
149    B=(int) (col.b*255);  //Threading Error: see comments near line 104
150    if (B > 255) B = 255;
151    else if (B < 0) B = 0;
```

```
tachyon.find_and_fix_threading_errors!render_one_pixel - find_and_fix_threading_errors.cpp:149
tachyon.find_and_fix_threading_errors!operator() - find_and_fix_threading_errors.cpp:202
tachyon.find_and_fix_threading_errors!run_body - parallel_for.h:102
tachyon.find_and_fix_threading_errors!execute<tbb::interface6::internal::start_for<tbb::blocked_range2d<int, int>, parallel_task,
tachyon.find_and_fix_threading_errors!execute - parallel_for.h:108
```

Write	find_and_fix_threading_errors.cpp:105	render_one_pixel	tachyon.find_and_fix_threading_errors

```
103    primary.scene = &scene;
104
105    col=trace(&primary);  //Threading Error: col is a global variable declared at line 80
106    //2 ways to fix this threading error
107        //    1) Make col a local variable
```

```
tachyon.find_and_fix_threading_errors!render_one_pixel - find_and_fix_threading_errors.cpp:105
tachyon.find_and_fix_threading_errors!operator() - find_and_fix_threading_errors.cpp:202
tachyon.find_and_fix_threading_errors!run_body - parallel_for.h:102
tachyon.find_and_fix_threading_errors!execute<tbb::interface6::internal::start_for<tbb::blocked_range2d<int, int>, parallel_task,
tachyon.find_and_fix_threading_errors!execute - parallel_for.h:108
```

Tachyon

- Solution with synchronization;

- Eliminating concurrency
  - Variable is not used outside function;
  - chaging variable from global to local eliminates Syncronization;