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

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

Task Level Parallelism

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

Process Level Parallelism

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

Agenda

- OpenMP
- Profiling
- Thread Affinity
- Vectorization
- Offloading
- N-body Simulation

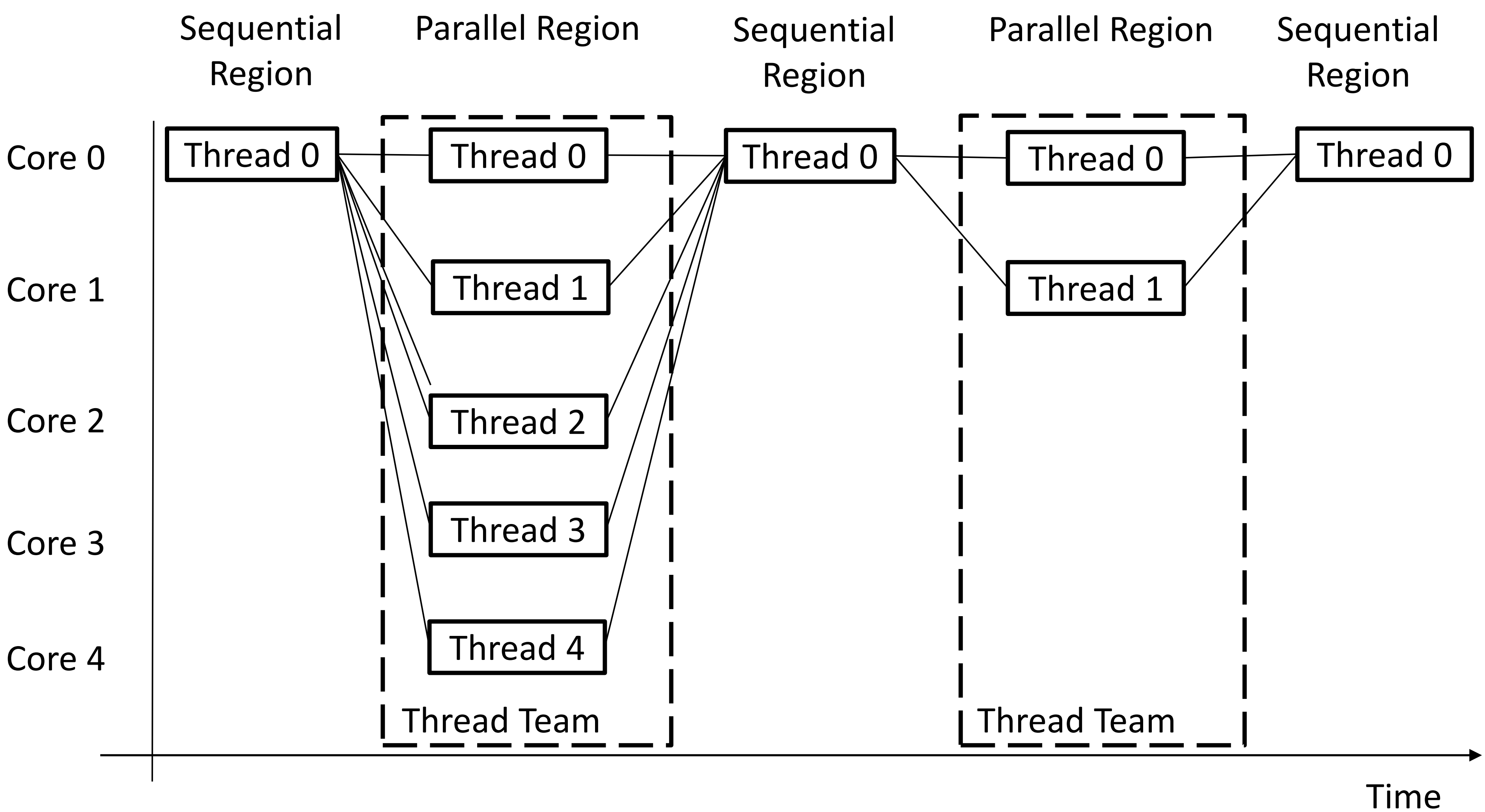
Agenda

- OpenMP
- Profiling
- Thread Affinity
- Vectorization
- Offloading
- 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 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>
#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/"

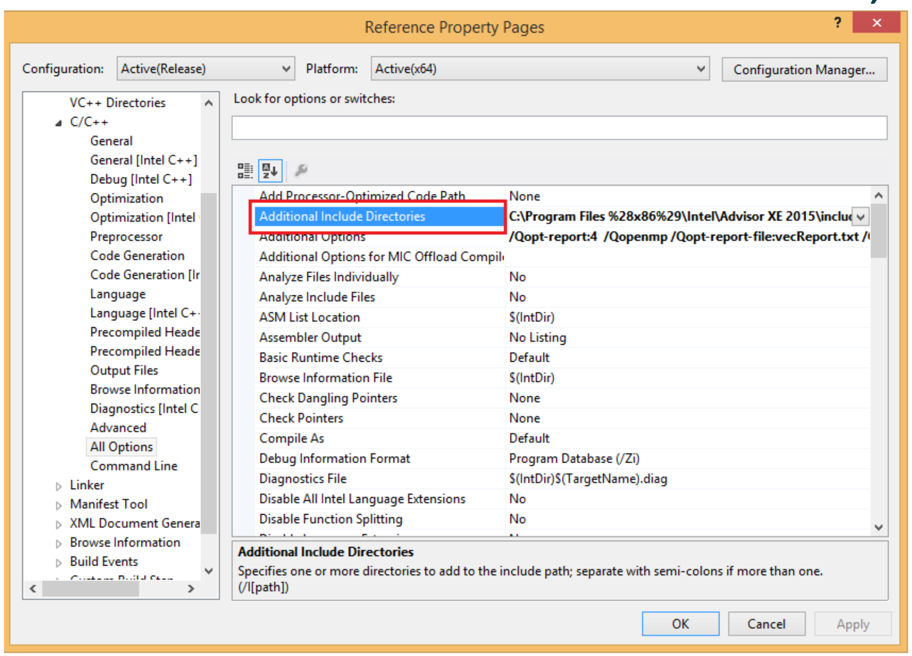
hello from hostname phi02-mic0.ncc.unesp.br
hello from hostname phi02-mic0.ncc.unesp.br
hello from hostname phi02-mic0.ncc.unesp.br
hello from hostname phi02-mic0.ncc.unesp.br
hello from hostname phi02-mic0.ncc.unesp.br
hello from hostname phi02-mic0.ncc.unesp.br
hello from hostname phi02-mic0.ncc.unesp.br
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

- OpenMP
- Profiling
- Thread Affinity
- Vectorization
- Offloading
- N-body Simulation

Identifying Parallelization Opportunities

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



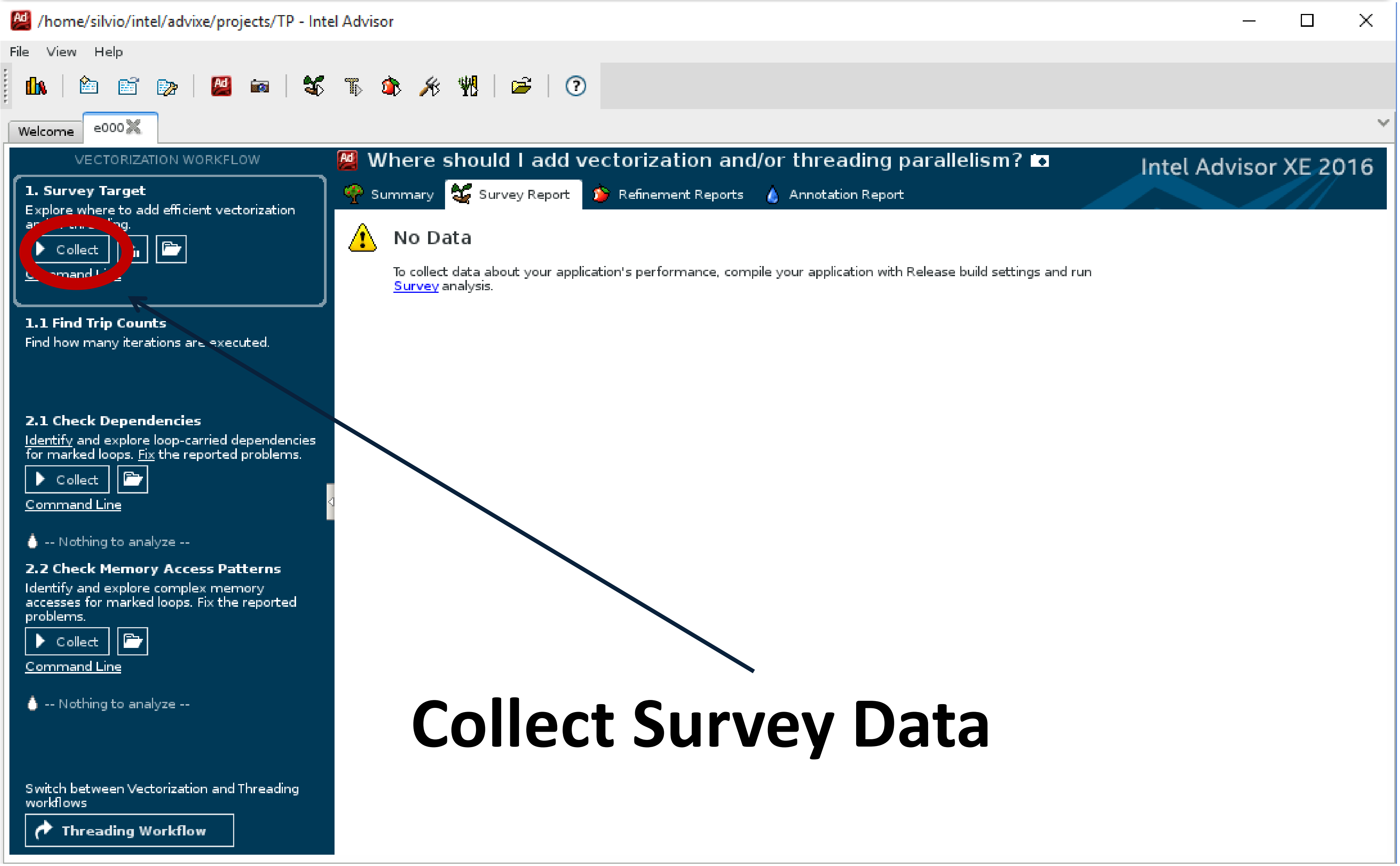
<u>Linux – compiling / link with Advisor</u>

icpc -O2 -openmp 02_ReferenceVersion.cpp -o 02_ReferenceVersion -I/opt/intel/advisor_xe/include/ -L/opt/intel/advisor_xe/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 – Check Suitability

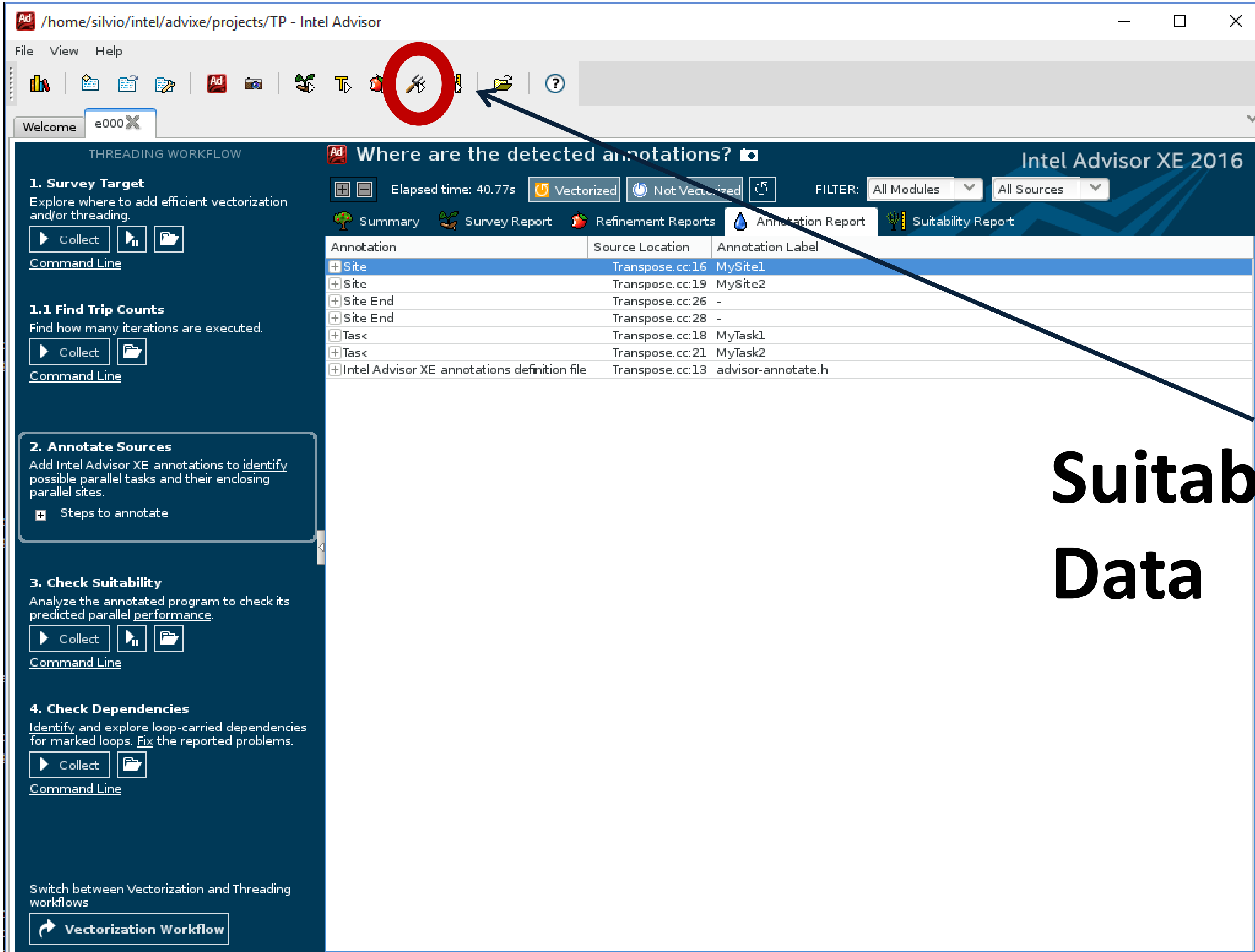
- Inserting advisor **Annotations key words** for Check Suitability:
  - **ANNOTATE_SITE_BEGIN(id)**: before beginning of loop;
  - **ANNOTATE_ITERATION_TASK(id)**: first line inside the loop;
  - **ANNOTATE_SITE_END()**: after end of loop;

- Example:

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

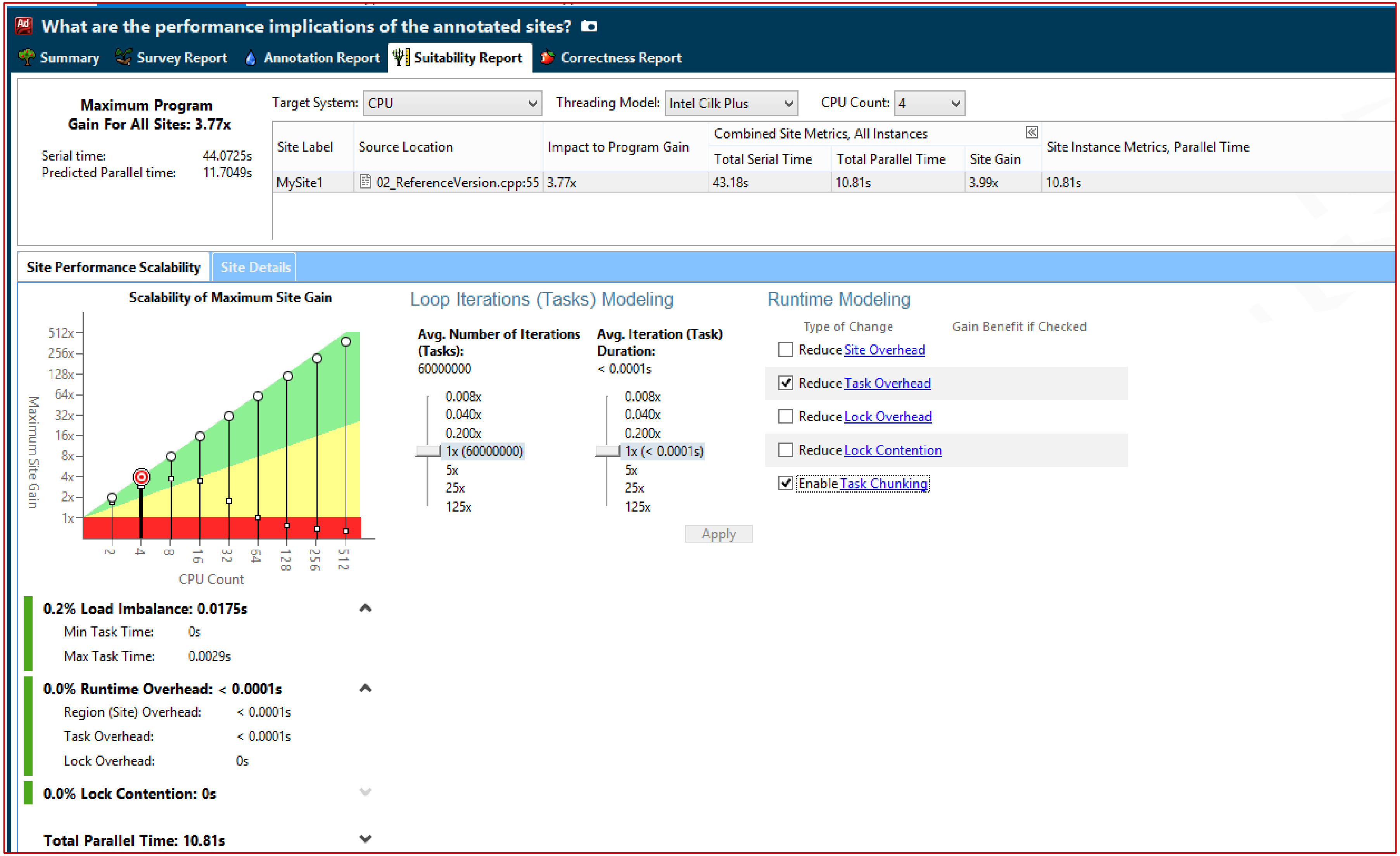
- Recompile application;

Intel Advisor – Check Suitability



Suitability Data

Intel Advisor – Check Suitability



Agenda

- OpenMP
- Profiling
- Thread Affinity
- Vectorization
- Offloading
- 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:

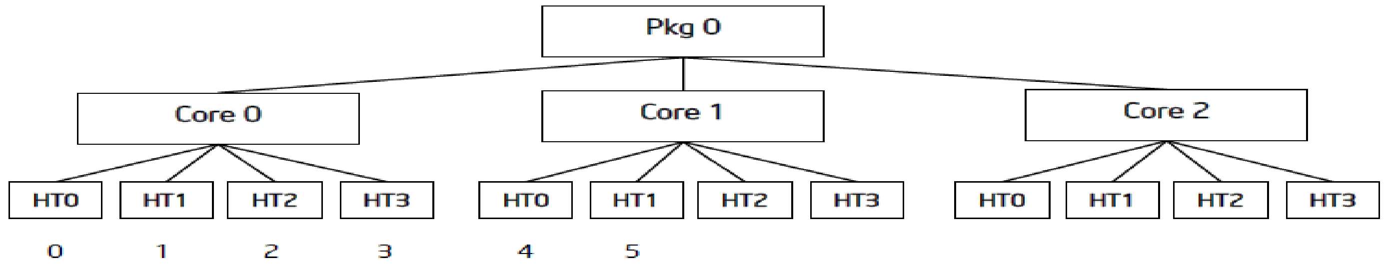
```
KMP_AFFINITY=
    [<modifier>,...]
    <type>
    [,<permute>]
    [,<offset>]
```

Example:

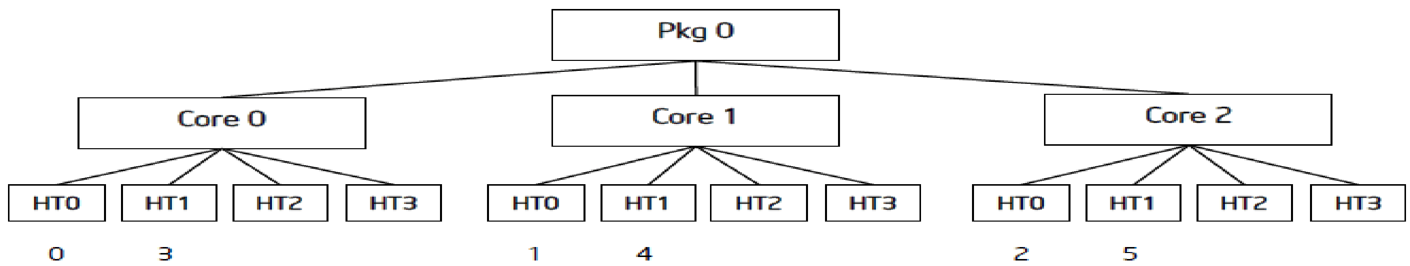
```
export KMP_AFFINITY=scatter
```

KMP_AFFINITY - Types

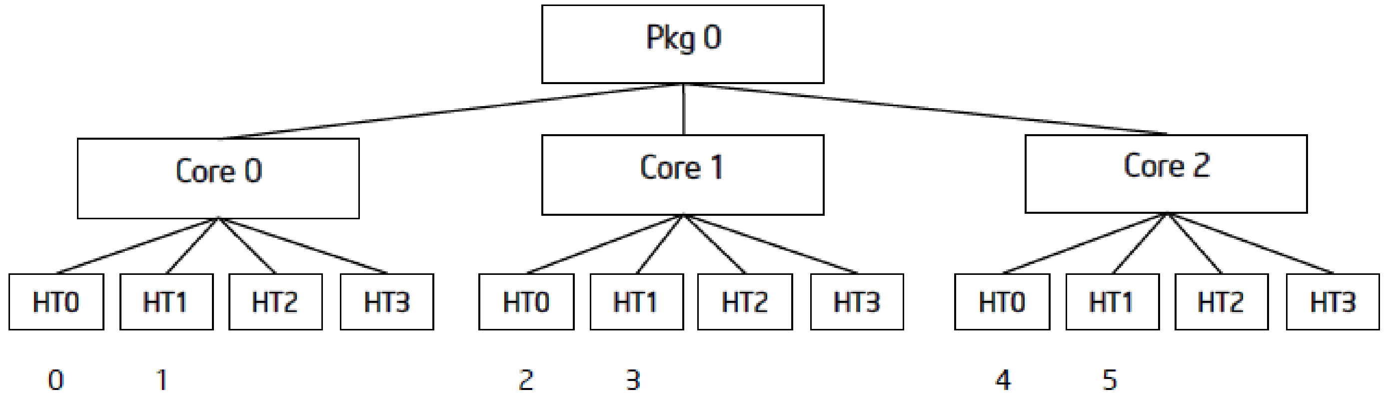
- Compact



- 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 x scatter

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

- OpenMP
- Profiling
- Thread Affinity
- Vectorization
- Offloading
- 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

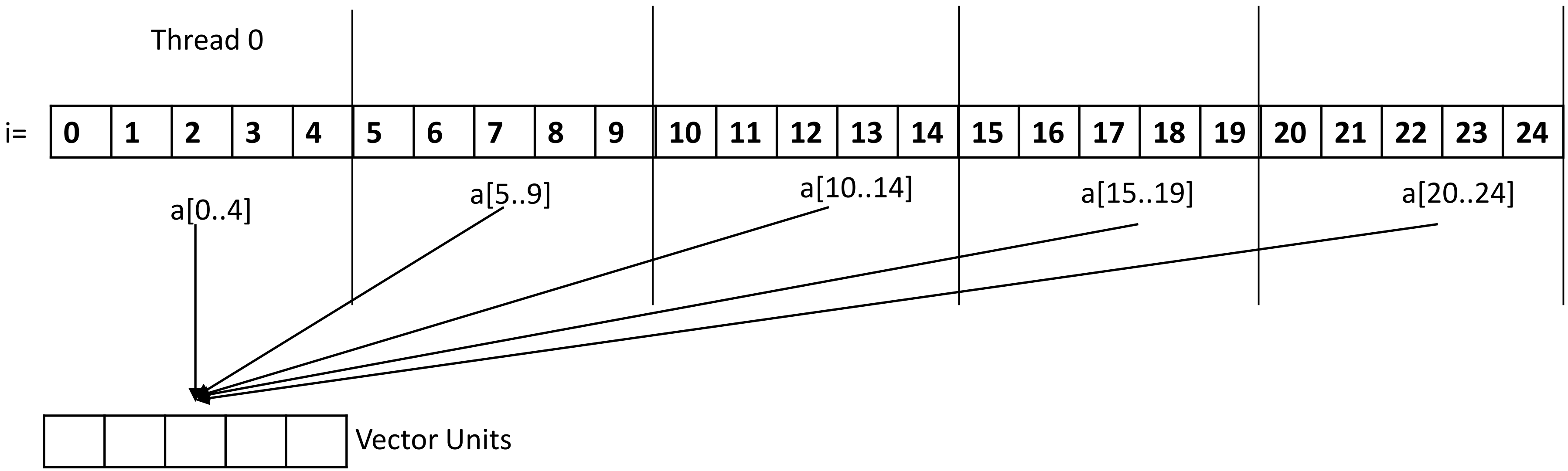
Pragma omp simd

- 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;
```



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

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)

Interpolation

```
#pragma omp declare
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 [] )
{
   . . .
  #pragma omp parallel for
  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 - 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 - 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)]
===

Agenda

- OpenMP
- Profiling
- Thread Affinity
- Vectorization
- Offloading
- 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.

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

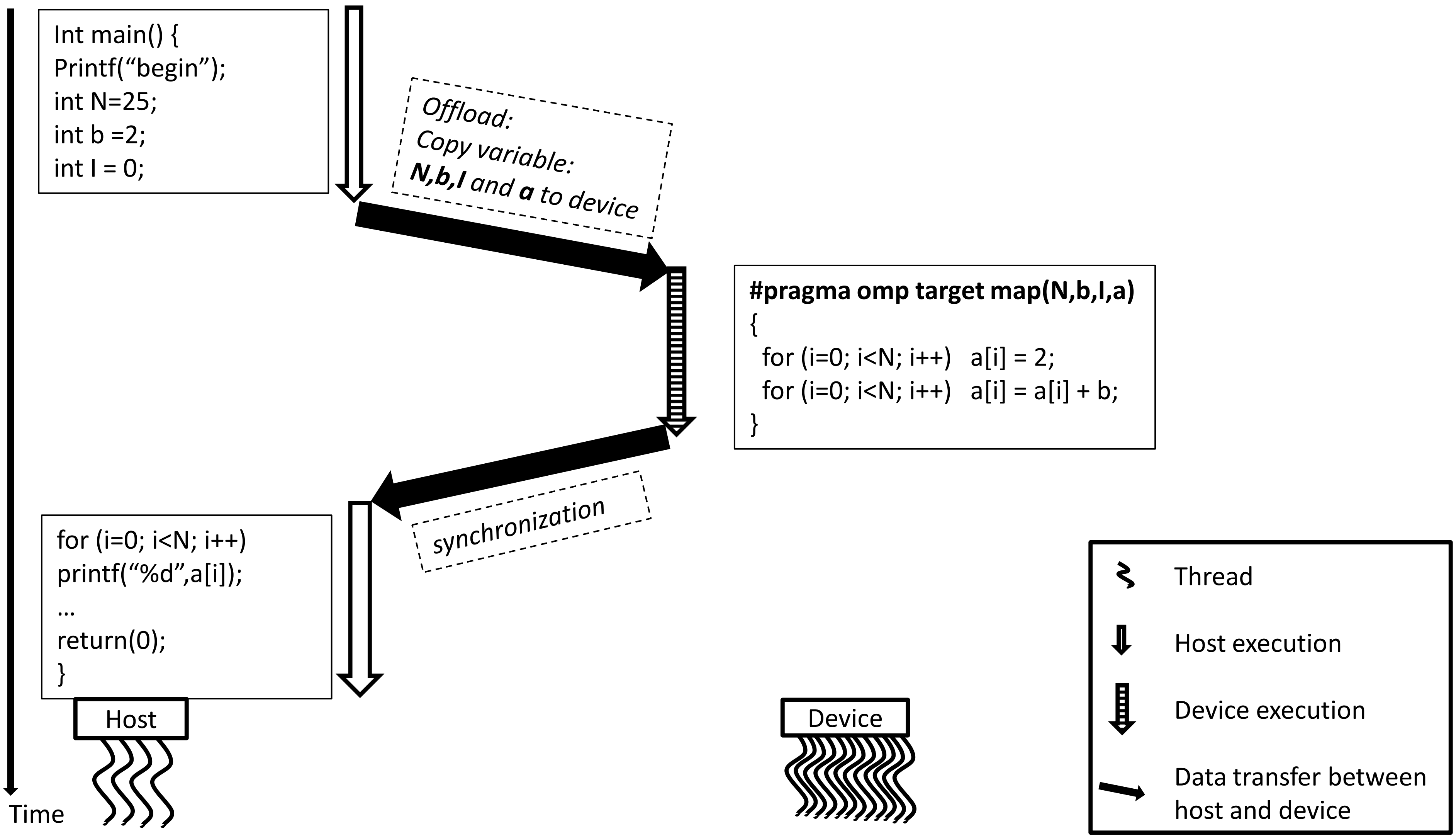
- Syntax
  - `#pragma omp target [data]` *`[clause[[,] clause],…]`*
    `structured-block`

- 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



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 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)
```

Matrix – load balancing

- Matrix Size: 10240;
- Strategy: Half of the iterations of outer loop to host and the other half to devices;
- Starts one thread to each device and one for the host
- The threads offloads the matrix to devices and start the multiplication
- The last thread start the multiplication on the host

Agenda

- OpenMP
- Profiling
- Thread Affinity
- Vectorization
- Offloading
- 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])
        }
    }
```