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
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
• 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)
• 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 (GEometry AND Tracking)

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

<table>
<thead>
<tr>
<th>A7</th>
<th>A6</th>
<th>A5</th>
<th>A4</th>
<th>A3</th>
<th>A2</th>
<th>A1</th>
<th>A0</th>
</tr>
</thead>
<tbody>
<tr>
<td>B7</td>
<td>B6</td>
<td>B5</td>
<td>B4</td>
<td>B3</td>
<td>B2</td>
<td>B1</td>
<td>B0</td>
</tr>
</tbody>
</table>

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

### Vector Instructions (SIMD)

- Vector Instructions (SIMD)
  - A7 + B7
  - A6 + B6
  - A5 + B5
  - A4 + B4
  - A3 + B3
  - A2 + B2
  - A1 + B1
  - A0 + B0

### Scalar Instructions

- Scalar Instructions
  - A + B

### Multi-level memory

- Ram Memory;
- Multi-level Cache.

<table>
<thead>
<tr>
<th>Processor 1</th>
<th>Processor 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 1</td>
<td>Core 1</td>
</tr>
<tr>
<td>Core 2</td>
<td>Core 2</td>
</tr>
<tr>
<td>Core N</td>
<td>Core N</td>
</tr>
<tr>
<td>L1</td>
<td>L1</td>
</tr>
<tr>
<td>L1</td>
<td>L1</td>
</tr>
<tr>
<td>L1</td>
<td>L1</td>
</tr>
<tr>
<td>L2</td>
<td>L2</td>
</tr>
<tr>
<td>L2</td>
<td>L2</td>
</tr>
<tr>
<td>L2</td>
<td>L2</td>
</tr>
<tr>
<td>L3</td>
<td>L3</td>
</tr>
<tr>
<td>L3</td>
<td>L3</td>
</tr>
<tr>
<td>Ram</td>
<td>Ram</td>
</tr>
</tbody>
</table>
## Multi Level Parallelism

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster</td>
<td>Group of computers communicating through fast interconnect</td>
</tr>
<tr>
<td>Coprocessors/Accelerators</td>
<td>Special compute devices attached to the local node through special interconnect</td>
</tr>
<tr>
<td>Node</td>
<td>Group of processors communicating through shared memory</td>
</tr>
<tr>
<td>Socket</td>
<td>Group of cores communicating through shared cache</td>
</tr>
<tr>
<td>Core</td>
<td>Group of functional units communicating through registers</td>
</tr>
<tr>
<td>Hyper-Threads</td>
<td>Group of thread contexts sharing functional units</td>
</tr>
<tr>
<td>Superscalar</td>
<td>Group of instructions sharing functional units</td>
</tr>
<tr>
<td>Pipeline</td>
<td>Sequence of instructions sharing functional units</td>
</tr>
<tr>
<td>Vector</td>
<td>Single instruction using multiple functional units</td>
</tr>
</tbody>
</table>

*OpenMP 4.0 for Devices, OpenMP 4.0 Affinity, OpenMP 4.0 SIMD*
Hybrid Parallel Architectures

Node

Bus Communication

Node

Host

Device

Vectorization

Offloading

Multithreading

Ram Memory

Multi-Level Cache Memory

Processor

Core

Core

Core

Core

Simd

Simd

Simd

Simd

Ram Memory

Multi-Level Cache Memory

Coprocessor

Core

Core

Core

Core

Core

Core

Simd

Simd

Simd

Simd

Simd

Simd

Simd
• 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

Multithreaded parallelism

Micro-architecture optimization

C/C++ ou Fortran Code

Thread 0/Core0
Thread 1/Core1
Thread 2/Core2
...Thread 244/Core61

Vector Processor Unit per Core
128 Bits 256 Bits

Vector Processor Unit per Core
512 Bits

Thread 0 / Core 0
Thread 1 / Core 1
Thread 2 / Core 2
...Thread 20 / Core20

128 Bits 256 Bits

Micro-architecture optimization

Vector Processor Unit per Core
512 Bits
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

- Cores: 61 cores, at 1.1 GHz in-order, support 4 threads
- 512 bit Vector Processing Unit
- 32 native registers
- Reliability Features: Parity on L1 Cache, ECC on memory, CRC on memory IO, CAP on memory IO
- High-speed bi-directional ring interconnect
- Fully Coherent L2 Cache
- 8 memory controllers
- 16 Channel GDDR5 MC
- PCIe GEN2
- Distributed tag directory to uniquely map physical addresses
Programmable Models

- Multi-Core Hosted: General purpose serial and parallel computing
- Symmetric: Codes with balanced needs
- Many Core Hosted: Highly-parallel codes

Offload: Codes with highly-parallel phases

Range of models to meet application needs
• 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/](http://www.openmp.org/)
  - Specification, examples, tutorials and documentation
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`

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

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

Thread affinity
• 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
main() {
    ...
    #pragma omp parallel {
        #pragma omp sections {
            #pragma omp section
            { ... }
            #pragma omp section
            { ... }
        }
        ...
        #pragma omp for nowait for( ... ) {
            ...
        }
        #pragma omp critical
        { ... }
        ...
        #pragma omp barrier
    ...
    }
    ...
}
N=25;
#pragma omp parallel for
for (i=0; i<N; i++)
    a[i] = a[i] + b;
#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);
    }

    #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);
}
# 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/"`
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

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

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.
• **KMP_AFFINITY:**
  – Environment variable that control the physical processing units that will execute threads of an application

• **Syntax:**

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

Example:

```bash
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/"
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 |

<table>
<thead>
<tr>
<th>Processor 1</th>
<th>Core 0</th>
<th>Core 1</th>
<th>...</th>
<th>Processor 2</th>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 0</td>
<td>Thread 1</td>
<td>Thread 0</td>
<td>Thread 1</td>
<td>...</td>
<td>...</td>
<td>Thread 0</td>
</tr>
</tbody>
</table>
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}
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}
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}

Thread Affinity balanced
• 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
• 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 Building 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)
Where should I add vectorization and/or threading parallelism?

No Data

To collect data about your application's performance, compile your application with Release build settings and run Survey analysis.
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

Thread 0

\[
\begin{align*}
N &= 25; \\
#pragma omp simd \\
&\text{for } (i=0; i<N; i++) \\
&a[i] = a[i] + b;
\end{align*}
\]

Vector Units

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

i=

a[0..4]
a[5..9]
a[10..14]
a[15..19]
a[20..24]
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
• 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
    \[ x_i = x_{\text{orig}} + i \times \text{linear-step} \]
• 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 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] ;
        }
    }
}

6/28/16
OMP SIMD - Vectorization Report

Compiler could not automatically vectorize loop on line 228, because of “assumed dependency”
Check dependency analysis shows that it is safe to enforce the vectorization of this loop.
#pragma omp simd guided the compiler to vectorize loop using AVX2
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;

```
// #pragma omp simd aligned(a:64, b:64) simdlen(64)
void vec2(float *a, float *b, int off, int len)
{
    int i;
    #pragma omp simd
    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]));
    }
}

void vec3(float *a, float *b, int off, int len)
{
    int i;
    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]));
    }
```
aligned 64 and simdlen 64 guided the compiler to vectorize loop using AVX2;
• Declare one or more functions to be compiled for calls from a SIMD-parallel loop

• Syntax (C/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)
```c
#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 ) ;
    }
    ...
}
```

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
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..._simd3__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..._simd3__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..._simd3__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..._simd3__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

begin 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) ]
===========================================================================

6/28/16
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
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

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

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 3</th>
<th>Thread 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4</td>
<td>5 6 7 8 9</td>
<td>10 11 12 13 14</td>
<td>15 16 17 18 19</td>
<td>20 21 22 23 24</td>
</tr>
</tbody>
</table>
#pragma omp 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;
}

• NCC Presentation
• Parallel Architectures
• Intel Xeon and Intel Xeon Phi
• OpenMP
• Thread Affinity
• Vectorization
• Offloading
• Thread League
• N-body Simulation
• **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.
• 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.
• 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;
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;
```c
Int main() {
    printf("begin");
    int N = 25;
    int b = 2;
    int l = 0;
    
    #pragma omp target map(N, b, l, a)
    {
        for (i = 0; i < N; i++)
            a[i] = 2;
        for (i = 0; i < N; i++)
            a[i] = a[i] + b;
    }
    
    offload:
    Copy variable:
    N, b, l and a to device
    
    #pragma omp target map(N, b, l, a)
    {
        for (i = 0; i < N; i++)
            printf("%d", a[i]);
    }
    ...
    return(0);
}
```

**Thread**
- Host execution
- Device execution
- Data transfer between host and device
```c
#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

```c
int main() {
    printf("begin");
    int N, b, l, a[];
    int N = 25;
    int b = 2;
    int l = 0;
    for (i = 0; i < N; i++)
        a[i] = 2;
    ...

    #pragma omp target data to(N) to(b) to(l)
    fromto(a)
    {
    #pragma omp target update to(N) to(b)
    to(l)
    #pragma omp target
    {
        for (i = 0; i < N; i++)
                a[i] = a[i] + b;
    }
    }

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

Update variables: N, b, i

Transfer variable A to host

Transfer variable s N, b, l and a to device.

#pragramomp target

synchronization

synchronization

synchronization

synchronization

Thread

Host execution

Device execution

Data transfer between host and device
#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 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] 0.000078(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
• **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] );
#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;
}

Example 1

#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;
}

Example 2

**Thread League—Example 1**

<table>
<thead>
<tr>
<th>Team 0</th>
<th>Team 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 0</td>
<td>Thread 1</td>
</tr>
<tr>
<td>Thread 2</td>
<td></td>
</tr>
<tr>
<td>Thread 3</td>
<td>Thread 4</td>
</tr>
<tr>
<td>Thread 5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Example 2**

<table>
<thead>
<tr>
<th>Team 0</th>
<th>Team 1</th>
<th>Team 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 0</td>
<td>Thread 1</td>
<td></td>
</tr>
<tr>
<td>Thread 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread 0</td>
<td>Thread 1</td>
<td></td>
</tr>
<tr>
<td>Thread 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
#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 );
    }
}
#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 ( to/from : 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
• 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.

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
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)

    
}
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
function Newton(step, begin_body, end_body, deviceld)
{
    #pragma omp target device (deviceld) {
        #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;

```c
#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