C-DAC Four Days Technology Workshop

ON

Hybrid Computing – Coprocessors/Accelerators
Power-Aware Computing – Performance of Applications Kernels

hyPACK-2013
Mode 3 : Intel Xeon Phi Coprocessors

Lecture Topic :
Intel Xeon-Phi Coprocessor OpenMP – An Overview

***Venue** : CMSD, UoHYD ; **Date** : October 15-18, 2013*

An Overview of Multi-Core Processors

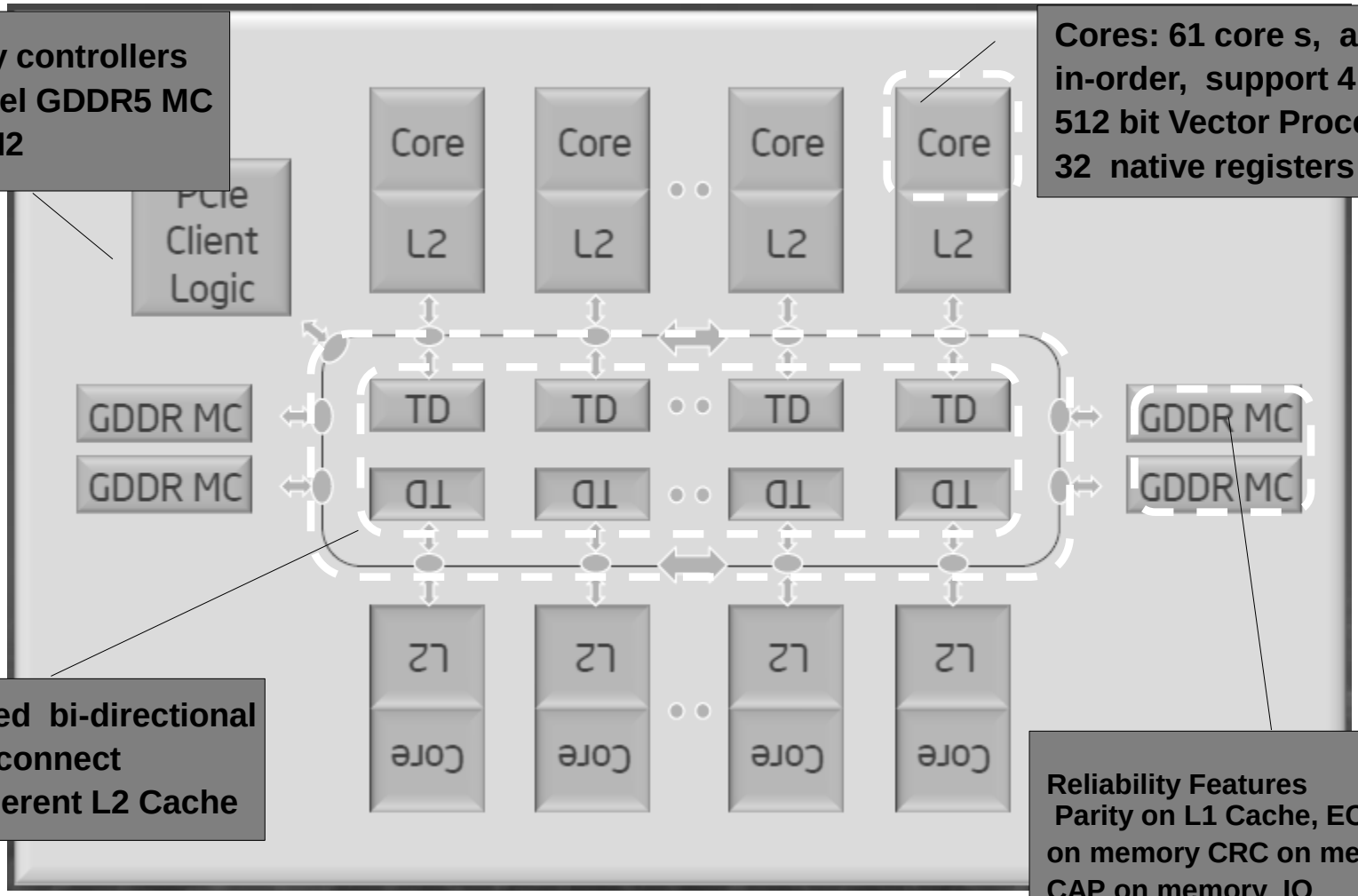
Lecture Outline

Following topics will be discussed

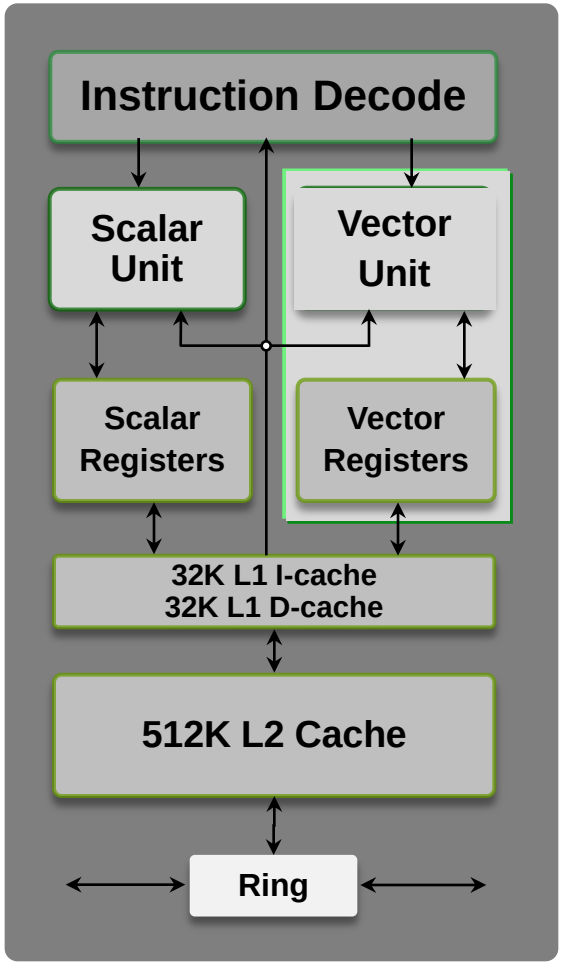
- Understanding of Multi-Core Architectures
- Programming on Multi-Core Processors - OpenMP
- Tuning & Performance – Software Threading

Intel Xeon-Phi Coprocessor : Architrecture

Intel® Xeon Phi™ Architecture Overview



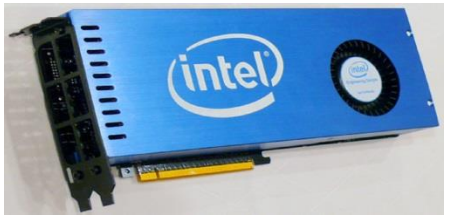
Core Architecture Overview



- 60+ in-order, low power IA cores in a ring interconnect
- Two pipelines
  - Scalar Unit based on Pentium® processors
  - Dual issue with scalar instructions
  - Pipelined one-per-clock scalar throughput
- SIMD Vector Processing Engine
- 4 hardware threads per core
  - 4 clock latency, hidden by round-robin scheduling of threads
  - Cannot issue back to back inst in same thread
- Coherent 512KB L2 Cache per core

Xeon Phi Hardware

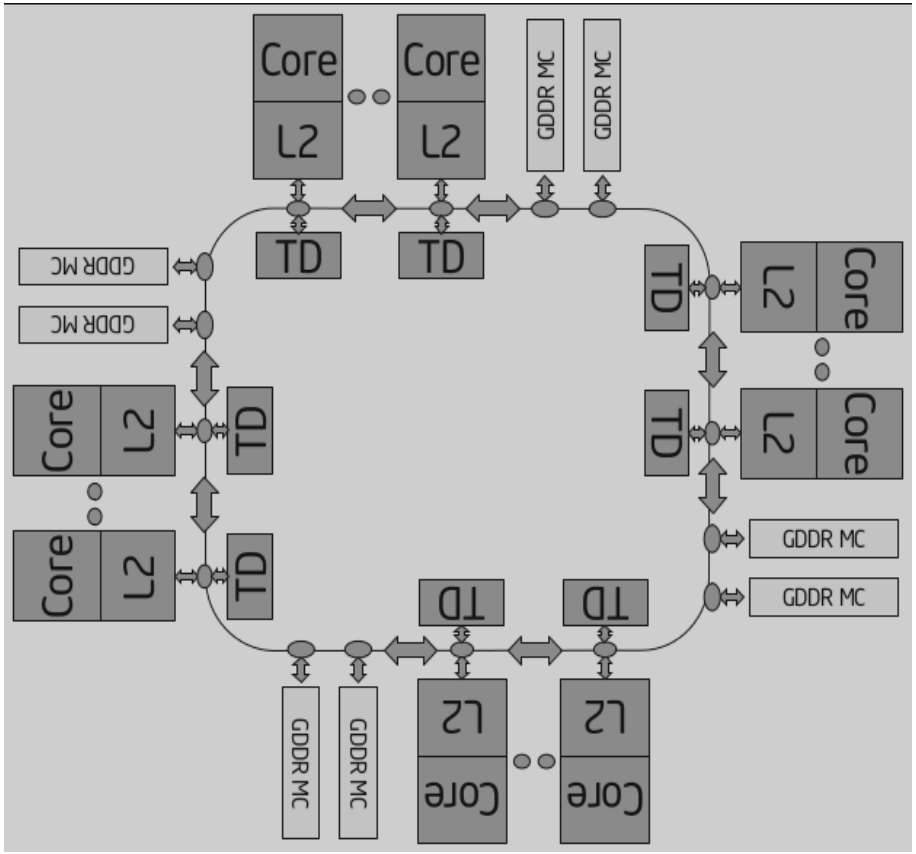
- X16 PCIe 2.0 card in Xeon host system
  - Up to 60 cores, bi-directional ring bus
  - 1-2GB GDDR5 main memory
- CPU cores
  - 1.2GHz, 4-way threading
  - 512-bit SIMD vector unit
  - 32KB L1, 256KB L2



Source : References & Intel Xeon-Phi; http://www.intel.com/

- Xeon-Phi coprocessor capacity 8GB;
  - processor :Xeon Phi 5110P; memory channel interface speed: 5.0 Giga Transfer/ Sec (GT/s); 8 memory controllers, each accessing two memory channels, used on co-processor

MIC Intel Xeon Phi Ring



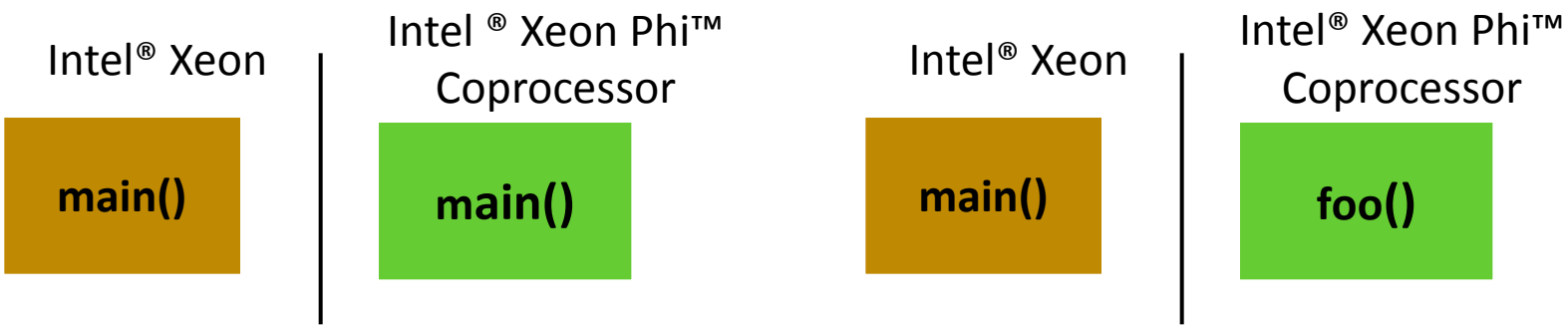
- Each microprocessor core is a fully functional, in-order core capable of running IA instructions independently of the other cores.
- Hardware multi-threaded cores
- Each core can concurrently run instructions from four processes or threads.
- The Ring Interconnect connecting all the components together on the chip

Source : References & Intel Xeon-Phi; http://www.intel.com/

Intel® Xeon Phi™ Coprocessor Arch – System SW Perspective

- Large SMP UMA machine – a set of x86 cores to manage
  - 4 threads and 32KB L1I/D, 512KB L2 per core
  - Supports loadable kernel modules – we'll talk about one today
- Standard Linux kernel from kernel.org
  - 2.6.38 in the most recent release
  - Completely Fair Scheduler (CFS), VM subsystem, File I/O
- Virtual Ethernet driver– supports NFS mounts from Intel® Xeon Phi™ Coprocessor
- New vector register state per thread for Intel® IMCI
  - Supports "Device Not Available" for Lazy save/restore
- Different ABI – uses vector registers for passing floats
  - Still uses the x86_64 ABI for non-float parameter passing (rdi, rsi, rdx ..)

Execution Modes



Native

- Card is an SMP machine running Linux
- Separate executables run on both MIC and Xeon
  - e.g. Standalone MPI applications
- No source code modifications most of the time
  - Recompile code for Xeon Phi™ Coprocessor
- Autonomous Compute Node (ACN)

Offload

- "main" runs on Xeon
- Parts of code are offloaded to MIC
- Code that can be
  - Multi-threaded, highly parallel
  - Vectorizable
  - Benefit from large memory BW
- Compiler Assisted vs. Automatic
  - #pragma offload (…)

Optimization Framework

A collection of methodology and tools that enable the developers to express parallelism for Multicore and Manycore Computing

Objective: Turning unoptimized program into a scalable, highly parallel application on multicore and manycore architecture

Step 1: Leverage Optimized Tools, Library

Step 2: Scalar, Serial Optimization /Memory Access

Step 3: Vectorization & Compiler

Step 4: Parallelization

Step 5: Scale from Multicore to Manycore

Source : References & Intel Xeon-Phi; http://www.intel.com/

Keys to Productive Performance on Intel ® MIC Architecture

- Choose the right Multi-core centric or Many-core centric model for your application
- Vectorize your application (today)
  - Use the Intel Vectorizing compiler
- Parallelize your application (today)
  - With MPI (or other multi-process model)
  - With threads (via Intel (R) CilkTM Plus, OpenMP*, Intel (R) Threading Buildig Blocks, Pthreads, etc.)
- Go asynchronous

Source : References & Intel Xeon-Phi; http://www.intel.com/

Xeon Phi : Programming Environment

Performance

- Vectorization is key for performance
  - Sandybridge, MIC, etc.
  - Compiler hints
  - Code restructuring
- Many-core nodes present scalability challenges
  - Memory contention
  - Memory size limitations

Source : References & Intel Xeon-Phi; http://www.intel.com/

Xeon Phi : Programming Environment

- Shared Address Space Programming (Offload, Native, Host)

  OpenMP, Inetl TBB, Cilk Plus, Pthreads

- Message Passing Programming

  (Offload – MIC Offload /Host Offload)

  (Symmetric & Coprocessor /Host)

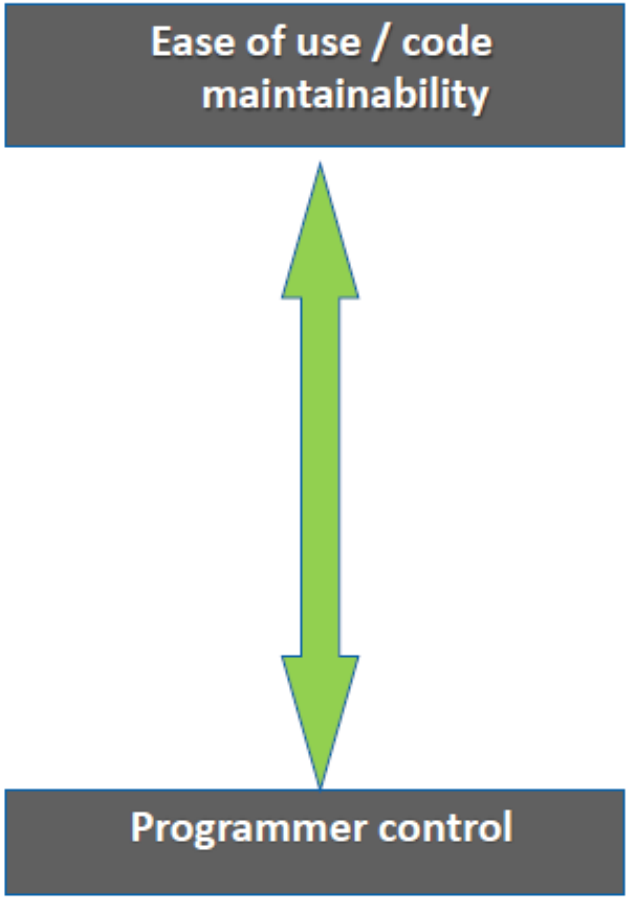
- Hybrid Programming

  (MPI – OpenMP, MPI Cilk Plus MPI-Intel TBB)

Source : References & Intel Xeon-Phi; http://www.intel.com/

Options for Thread Parallelism

- Intel® Math Kernel Library
- Intel® Threading Building Blocks
  Intel® Cilk™ Plus
- OpenMP*
- Pthreads* and other threading libraries



Source : References & Intel Xeon-Phi; http://www.intel.com/

Programming on Systems with Co-processors & Accelerators – An Overview of OpenMP

Xeon Phi : Programming Environment

- **Shared Address Space Programming (Offload, Native, Host)**

  OpenMP, Inetl TBB, Cilk Plus, Pthreads

- **Message Passing Programming (Offload – MIC Offload /Host Offload)**

  (Symmetric & Coprocessor /Host)

- **Hybrid Programming**

  (MPI – OpenMP, MPI Cilk Plus MPI-Intel TBB)

Source : References & Intel Xeon-Phi; http://www.intel.com/

MPI Prog. Models for Xeon systems with MIC

Offload

- Intel ® MIC Architecture or host CPU as an accelerator

MIC Offload (direct acceleration)

- MPI ranks on the host CPU only
- Messages into/out of the host CPU
- Intel ® MIC Architecture as an accelerator

Host Offload (reverse acceleration)

- MPI ranks on the MIC CPU only
- Messages into/out of the MIC CPU
- Host CPU as an accelerator

MPI

- MPI ranks on several co-processors and/or host nodes
- Messages to/from any core

Co-processor-only

- MPI ranks on the MIC CPU only
- Messages into/out of the MIC CPU c/o host CPUs
- Threading possible

Symmetric

- MPI ranks on the MIC and host CPUs
- Messages into/ out of the MIC and host CPUs
- Threading possible

Source : References & Intel Xeon-Phi; http://www.intel.com/

Offload Code Examples

C/C+ Offload Pragma

```
#pragma offload target (mic)
#pragma omp parallel for reduction(+:pi)
 for (i = 0; i<count; i++) {
      float t = (float) (i+0.5/count);
      pi += 4.0/(1.0t*t);
 }
pi/ = count;
```

C/C++ Offload Pragma

```
#pragma offload target(mic)
      in(transa, transb, N, alpha, beta) \
      in(A:length(matrix_elements)) \
      in(B:length(matrix_elements)) \
      inout(C:length(matrix_elements))
         sgemm(&transa, &transb, &N, &N, &N,
& alpha, A, &N, B, & N, &beta, C &N);
```

Fortran Offload Directives

```
!dir$ omp offload target(mic)
!$omp parallel do
     do i = 1, 10
                A(i) = B(i) * C(i)
     enddo
```

C/C++ Language Extension

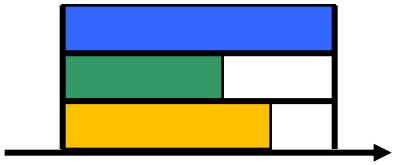
```
class_Cilk_Shated common {
    int data1;
    int *data2;
    class common *next;
    void process();
}
_Cilk_Shared class common obj1, obj2;
_Cilk_spawn _offload obj1.process();
_Cilk_spawn  _offload  obj2.process();
```

Source : References & Intel Xeon-Phi; http://www.intel.com/

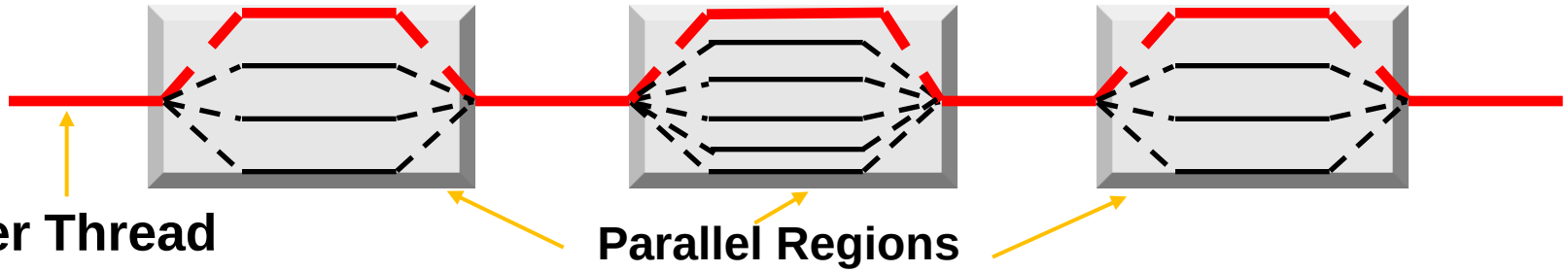
Options for Parallelism – OpenMP*

- Compiler directives/pragmas based threading constructs
  - Utility library functions and Environment variables
- Specify blocks of code executing in parallel

```
#pragma omp parallel sections
{
    #pragma omp section
    task1();
    #pragma omp section
    task2();
    #pragma omp section
    task3();
```



- Fork-Join Parallelism:
  - Master thread spawns a team of worker threads as needed
  - Parallelism grow incrementally



Source : References & Intel Xeon-Phi; http://www.intel.com/

OpenMP* Pragmas and Extensions

- OpenMP* pragmas in C/C++:

  `#pragma omp construct [clause [clause]…]`

- Large robust specification that includes
  - Parallel sections and tasks
  - Parallel loops
  - Synchronization points
    - critical sections
    - barriers
  - Atomic and ordered updates
  - Serial sections within parallel code

```
#pragma omp parallel sections
                {
            #pragma omp section

BinomialTemplate<F32vec8, float>(callResult, S,
              X, T, R, V, N, NUM_STEPS)
                }
            #pragma omp section
          #pragma offload  target(mic:0)
            in(S, X, T:length(N))
  out(MICExpected, MICConfidence:length(N))
                {
montecarlo(S, X, T, MICExpected, MICConfidence);
                }
```

- Extension to support offloading – OpenMP* 4.0 RC2
  - Use `#pragma omp target` or `#pragma offload` from Intel LEO
  - Either syntax works, no performance differences

Source : References & Intel Xeon-Phi; http://www.intel.com/

OpenMP* Worksharing Construct

Sequential code

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

OpenMP* parallel region

```
#pragma omp parallel
{
   int id, i, Nthrds, istart, iend;

   id = omp_get_thread_num();
   Nthrds = omp_get_num_threads();
   istart = id * N / Nthrds;
   iend = (id + 1) * N / Nthrds;

   for (i = istart; i < iend; i++)
      a[i] = a[i] + b[i];
}
```

OpenMP* worksharing construct

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

Source : References & Intel Xeon-Phi; http://www.intel.com/

Shared, Private and Reduction Variables

- Default Rules
  - Variables defined outside the parallel region are `shared`
  - Variables defined inside the parallel region are `private`
- Override the defaults
  - The `private(var)` clause creates a local copy of var for each thread
  - Loop indices in a parallel for are private by default
- The `reduction(op:list)` clause is a special case of shared
  - Variables in "list" must be shared in the enclosing parallel region
    - A local copy of each reduction variable is created and initialized based on the *op* (0 for "+")
    - Compiler finds reduction expressions containing *op* and uses them to update the local copy
    - Local copies are reduced to a single value and combined with the original global value

```
#pragma omp parallel reduction(+ : sum_delta) reduction(+ : sum_ref)
{
   float local_sum_delta = 0.0f;
   for(int i = 0; i < OptPerThread; i++)
   {
      ref   = callReference;
      delta = fabs(callReference - CallResult[i]);
      local_sum_delta += delta;
      sum_ref   += fabs(ref);
   }
   sum_delta += local_sum_delta;
}
```

Source : References & Intel Xeon-Phi; http://www.intel.com/

OpenMP* Performance, Scalability Issues

- Manage Thread Creation Cost
  - Create threads as early as possible, Maximize the work for worker threads
  - IA threads take some time to create, But once they're up, they last till the end
- Take advantage of memory locality, use NUMA memory manager
  - Allocate the memory on the thread that will access them later on.
  - Try not to allocate the memory the worker threads use in the main thread
- Ensure your OpenMP* program works serially, compiles without openmp*
  - Protect OpenMP* API calls with _OPENMP,
  - Make sure serial works before enable OpenMP* (e.g. compile with –openmp)
- Minimize the thread synchronization
  - use local variable to reduce the need to access global variable

Source : References & Intel Xeon-Phi; http://www.intel.com/

```
#pragma omp parallel for
for (int k = 0; k < RAND_N; k++)
   h_Random[k] = cdfnorminv ((k+1.0)/(RAND_N+1.0));

#pragma omp parallel for
for(int opt = 0; opt < OPT_N; opt++)
{
   CallResultList[opt]     = 0;
   CallConfidenceList[opt] = 0;
}
```

```
#pragma omp parallel
{
   #ifdef _OPENMP
   int threadID = omp_get_thread_num();
   #else
   int threadID = 0;
   #endif

   float *CallResult = (float *) scalable_aligned_malloc
                                  (mem_size, SIMDALIGN);
   float *PutResult  = (float *) scalable_aligned_malloc
                                  (mem_size, SIMDALIGN);

}
```

```
#ifdef _OPENMP
int ThreadNum = omp_get_max_threads();
omp_set_num_threads(ThreadNum);
#else
int ThreadNum = 1;
#endif
```

OpenMP* Offload Environment Variables

- Set/Get the number of coprocessor threads from the host
  - Notice that `omp_get_max_thread_target()` return 4*(ncore-1)
  - Use `omp_set_num_threads_target()` `omp_get_num_threads_target()`
  - Protect under #ifdef __INTEL_OFFLOAD,
- Access coprocessor environment variables from the host processor
  - First define `MIC_ENV_PREFIX=MIC`
  - Issue export `MIC_OMP_NUM_THREADS=240` on the host
  - OpenMP sets the coprocessor max threads to 240
  - Host OpenMP threads still take the cues from `OMP_NUM_THREADS`
- Initial Stack Size on the device is default to be 12MB
  - Use `MIC_STACKSIZE` to override the default size for main threads in coprocessor
  - Use `MIC_OMP_STACKSIZE` to change the default stack size for worker threads
- Compile OpenMP codes by adding `-openmp` compiler flag.
- Use `KMP_AFFINITY="compact,granularity=fine"` for thread pinning.

Source : References & Intel Xeon-Phi; http://www.intel.com/

Intel Xeon-Phi : OpenMP I Prog. Model

- OpenMP parallelization on an "**Intel Xeon + Xeon Phi coprocessor machine**" can be applied in **four** different programming models.

  - **Realized with Complier Options**

Intel Xeon-Phi : OpenMP I Prog. Model

- **Four** Models with different programming models
  - Native OpenMP on the Xeon host
  - Serial Xeon host with OpenMP offload
  - OpenMP on the Xeon Host with OpenMP offload
  - Native OpenMP on the Xeon Phi coprocessor

Intel Xeon-Phi : OpenMP I Prog. Model

❖ **Remark :**

- OpenMP threads on Xeon Host and OpenMP threads on Xeon Phi do not interface each other and when an offload/pragma section of the code is encountered

- Offloaded as a **Unit** and uses a number of threads based on available resources on **Xeon Phi** coprocessor

- Usual semantics of **OpenMP** Constructs apply on Xeon host and Xeon-Phi Coprocessor

Intel Xeon-Phi : OpenMP I Prog. Model

❖ **Remark :**

- Offload to the **Xeon Phi** coprocessor can be done at any time by multiple host CPUs until the filling of the available resources.

- If there are **no** free threads, the task meant to be offloaded may be done on the **host**.

- For offload schemes, the **maximal amount** of threads that can be used on the Xeon Phi coprocessor is **4 times** the total number of cores **minus one**, because **one core** is reserved for the **OS** and its **services**.

Intel Xeon-Phi : OpenMP I Prog. Model

❖ Threading and affinity

- **Important Considerations** for OpenMP threading and affinity are the total number of threads that should be utilized and the scheme for binding threads to processor cores.
- The Xeon Phi coprocessor supports 4 threads per core.
- Using more than one core is recommended.
- When running applications natively on the Xeon Phi the full amount of threads can be used.
- On Xeon host, benefit from hyper-threading exists.

Intel Xeon-Phi Coprocessor : MPI on Cluster

Threading and affinity : Settings :

Settings	Description
OpenMP on host without HT	1 x ncore-host
OpenMP on host with HT	2 x ncore-host
OpenMP on Xeon Phi in native mode	4 x ncore-phi
OpenMP on Xeon Phi in offload mode	1 x ncore-phi-1

- If OpenMP regions exist on the **host** and on the part of the code **offloaded** to the Xeon Phi, **two** separate OpenMP runtimes exist.

Intel Xeon-Phi : OpenMPI Prog. Model

❖ **Threading and affinity**

- Environment variables for controlling OpenMP behavior are to be set for both runtimes

For example

- the **KMP_AFFINITY** variable which can be used to assign a particular thread to a particular physical node. For
- Intel Xeon Phi it can be done like this:
- **export MIC_ENV_PREFIX=MIC**

Intel Xeon-Phi : OpenMPI Prog. Model

❖ Threading and affinity

```
export MIC_ENV_PREFIX=MIC

#specify affinity for all cards
export MIC_KMP_AFFINITY=...

#specify number of threads for all cards
export MIC_OMP_NUM_THREADS=120

#specify the number of threads for card #2
export MIC_2_OMP_NUM_THREADS=200

#specify number of threads and affinity for card #3
export MIC_3_ENV="OMP_NUM_THREADS=60 |
                  KMP_AFFINITY=balanced"
```

Intel Xeon-Phi : OpenMPI Prog. Model

Threading and affinity

One can also use special **API calls** to set the environment for the coprocessor only, **e.g.**

```
omp_set_num_threads_target()
```

```
omp_set_nested_target()
```

Intel Xeon-Phi : OpenMPI Prog. Model

Loop Scheduling

- OpenMP accepts four different kinds of loop scheduling - **`static`**, **`dynamic`**, **`guided`** & **`auto`**.

- The **`schedule`** clause can be used to set the loop scheduling at compile time.

- Another way to control this feature is to specify schedule(runtime) in your code and select the loop scheduling at runtime through setting the **`OMP_SCHEDULE`** environment variable

Intel Xeon-Phi : OpenMPI Prog. Model

Scalability

- Use **`-collapse`** directive to specify how many for-loops are associated with the OpenMP loop construct

- Another way to improve scalability is to reduce barrier synchronization overheads by using the **`nowait`** directive.

- Another way to control this feature is to specify schedule(runtime) in your code and select the loop scheduling at runtime through setting the **`OMP_SCHEDULE`** environment variable

Intel Xeon Phi Coprocessor – OpenMP framework

The easiest and (arguably) the most productive way to exploit threading parallelism with an Intel Xeon Phi is to use OpenMP.

Intel Xeon Phi Coprocessor – OpenMP framework

- OpenMP – Programming on Shared Address Space Platform
- Cluster of Multi-Core Processor Systems (Nodes) – Use MPI & OpenMP programming
- Each Node has single /multiple Intel Xeon Phi Co-processor to use OpenMP.
- **Node Topology :** Each node contains a shared-memory environment with several processor sockets, each socket containing several physical cores which, in turn, are divided into several logical cores.

Intel Xeon Phi Coprocessor – OpenMP framework

- Each memory bank is usually attached to processor socket
- Each memory bank usually resides closer to some of the cores in the topology and therefore access to data laying in a memory bank attached to another socket is generally more expensive.
- Uniform Memory Access (SMP) and Non-Uniform Memory Access (NUMA) systems exist. (NUMA) can create performance issues if threads are allowed to migrate from one logical core to another during their execution.

Intel Xeon Phi Coprocessor – OpenMP framework

- Each memory bank is usually attached to processor socket
- Each memory bank usually resides closer to some of the cores in the topology and therefore access to data laying in a memory bank attached to another socket is generally more expensive.
- Uniform Memory Access (SMP) and Non-Uniform Memory Access (NUMA) systems exist. (NUMA) can create performance issues if threads are allowed to migrate from one logical core to another during their execution.

Intel Xeon Phi Coprocessor – OpenMP framework

- **Thread Affinity :** Consider binding OpenMP threads to logical and physical cores across different sockets on one compute node.

- The layout of this binding in respect to the node topology has performance implications depending on the computational task and is referred as thread affinity.

- The thread affinity interface of the Intel runtime library can be controlled by using the **`KMP_AFFINITY`** environment variable or by using a proprietary Intel API.

Intel Xeon Phi Coprocessor – OpenMP framework

- **KMP_AFFINITY**

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

modifier default=noverbose, respect, granularity=core

type default=none

balanced,compact,disabled,explicit,none,scatter

permute default=0.

>=0,integer.Not valid with offset default=0.

Offset default=0

>=0,integer.Not valid with type=explicit,none,disabled

The most important affinity types supported by Intel Xeon Phi are *balanced*, *compact*, *none*, *scatter*

Example: Thread Affinity

```
!$OMP PARALLEL DO DEFAULT(NONE) &
!$OMP SHARED(A,B,C) &
!$OMP PRIVATE(i,j,k) &
DO j=1,n
  DO k=1,n
    DO i=1,n
      C(i,j)=C(i,j)+A(i,k)*B(k,j)
    END DO
  END DO
END DO
!$OMP END PARALLEL DO
```

Programming on Systems with Co-processors & Accelerators – Results & Performance Issues

Intel Xeon-Host : Benchmarks Performance

Host : Xeon (Memory Bandwidth (BW) - Xeon: 8 bytes/channel * 4 channels * 2 sockets * 1.6 GHz = 102.4 GB/s)

Xeon Phi Co-Processor Bandwidth

Xeon-Phi coprocessor capacity 8GB; processor Xeon Phi 5110P; memory channel interface speed: 5.0 Giga Transfer/ Sec (GT/s); 8 memory controllers, each accessing two memory channels, used on co-processor. each memory transaction to GDDR5 memory is 4 bytes of data, resulting in 5.0 GT/s * 4 bytes or 20 GB/s per channel.

PARAM YUVA-II Intel Xeon- Node Benchmarks(*)

Xeon Node Memory Bandwidth :
8 bytes/channel * 4 channels * 2 sockets * 1.6 GHz = 102.4 GB/s)
Experiment Results : Achieved Bandwidth : 70 % ~75 % Effective bandwidth can be improved in the range of 10% to 15% with some optimizations

Node : Intel-R2208GZ; Intel Xeon E52670; Core Frequency : 2.6GHz; Cores per Node : 16 ; Peak Performance /Node : 2.35 TF; Memory : 64 GB;

Data Size (MegaBytes)	No. of Cores (OpenMP)	Sustained Bandwidth (GB/sec)
1024	16	72.64

(*) = Bandwidth results were gathered using untuned and unoptimized versions of benchmark (In-house developed) and Intel Prog. Env

Source : http://www.intel.com; Intel Xeon-Phi books, conferences, Web sites, Xeon-Phi Technical Reports

http://www.intel.in/content/dam/www/public/us/en/documents/performance-briefs/xeon-phi-product-family-performance-brief.pdf

PARAM YUVA-II Intel Xeon- Phi Benchmarks(*)

PARAM YUVA-II Xeon Phi Co-Processor Bandwidth

- Xeon-Phi coprocessor (PARAM YUVA-II) capacity 8GB; processor Xeon Phi 5110P; memory channel interface speed: 5.0 Giga Transfer/ Sec (GT/s); 8 memory controllers, each accessing two memory channels, used on co-processor. Each memory transaction to GDDR5 memory is 4 bytes of data, resulting in 5.0 GT/s * 4 bytes or 20 GB/s per channel.
- **Peak Electrical bandwidth 320 GB/s.** (16 total channels provide a maximum transfer rate 320 GB/s)
- Our experiments indicated that 40% of the peak is achieved. Effective bandwidth in the range of 50 to 60% of peak memory bandwidth can be achieved with some optimization.

(*) = Bandwidth results were gathered using untuned and unoptimized versions of benchmark (in-house developed) and Intel Prog. Env

Source : http://www.intel.com; Intel Xeon-Phi books, conferences, Web sites, Xeon-Phi Technical Reports

http://www.intel.in/content/dam/www/public/us/en/documents/performance-briefs/xeon-phi-product-family-performance-brief.pdf

PARAM YUVA-II Intel Xeon- Phi Benchmarks(*)

Bandwidth : Peak Electrical bandwidth 320 GB/s. (16 total channels provide a maximum transfer rate 320 GB/s)

Experiment Results : Achieved bandwidth is **40%** of the peak & it can be increased in the range of **50% to 60%** of peak memory bandwidth.

Data Size (Mega bytes)	No. of Cores (OpenMP)	Sustained Bandwidth (GB/sec)(*)
1024	8	39.47
	16	68.59
	30	98.23
	40	118.22
	50	136.56
	60	138.22

(=No optimizations are carried-out to use OpenMP threads & Intel Prog. Env)*

(*) = Bandwidth results were gathered using untuned & unoptimized versions of benchmark (in-house developed) and Intel Prog. Env

Source : http://www.intel.com; Intel Xeon-Phi books, conferences, Web sites, Technical Reports **http://www.intel.in/content/dam/www/public/us/en/documents/performance-briefs/xeon-phi-product-family-performance-brief.pdf**

PARAM YUVA-II Intel Xeon- Phi Benchmarks(*)

Bandwidth : Peak Electrical bandwidth 320 GB/s. (16 total channels provide a maximum transfer rate 320 GB/s)

Experiment Results : Achieved bandwidth is **40%** of the peak & it can be increased in the range of **50% to 60%** of peak memory bandwidth on some nodes of PARAM YUVA (ycn213, ycn210, ycn212)

Data Size (Megabytes)	No. of Cores (MPI & 120 OpenMP threads)	Sustained Bandwidth (GB/sec)(*)
2048	ycn213(mic-0)	137.108
	ycn213(mic-1)	137.654
	ycn210 (mic-0)	138.697
	ycn210 (mic-1)	137.712
	ycn212 (mic-0)	137.819
	ycn212 (mic-1)	132.085

(=No optimizations are carried-out to use OpenMP threads & Intel Prog. Env) CDAC P-COMS software is used.)*

(No optimizations are carried-out to use Intel MPI & OpenMP threads ***Prog. Env***

(*) = Speedup results were gathered using untuned and unoptimized versions of benchmark (in-house developed) and Intel Prog. Env

http://www.intel.in/content/dam/www/public/us/en/documents/performance-briefs/xeon-phi-product-family-performance-brief.pdf

PARAM YUVA-II Intel Xeon- Phi Benchmarks(*)

Peak Performance : Single Precision : 2129.47 Gflops/s

Peak Perf : 1.1091 GHz X 60 cores X 16 lanes X 2

No. of Cores = 60

Peak Perf. of Single Core = 35.49 GigaFlop/s

Experiment Results for Single Precision Addition of Two Vectors(*)		
Type of Optimization	**No. of Cores OpenMP threads**	**Sustained Perf in Gflops**
No Vectorization	1	0.195
Vectorization	1	17.256
1	1 (4)	28.435

(*=No optimizations are carried-out to use OpenMP threads & Intel Prog. Env) Intel MKL Libraries are used.)

(No optimizations are carried-out to use OpenMP threads & Intel Prog. Env)

(*) = Speedup results were gathered using untuned and unoptimized versions of benchmark (in-house developed) and Intel Prog. Env

PARAM YUVA-II Intel Xeon- Phi Benchmarks(*)

Peak Performance : Single Precision : 2129.47 Gflops/s
No. of Cores = 60

Experiment Results for Single Precision Addition of Two Vectors(*)		
No. of Cores / OpenMP threads	**Thread Affinity**	**Sustained Perf in Gflops**
4	COMPACT	66.7
8	COMPACT	133.69
16	COMPACT	266.89
32	COMPACT	482.85
64	COMPACT	1001.84
120	COMPACT	1804.25
240	COMPACT	1892.66

(=No optimizations are carried-out to use OpenMP threads & Intel Prog. Env) Intel MKL Libraries are not used*

(No optimizations are carried-out to use OpenMP threads & Intel Prog. Env)

(*) = Speedup results were gathered using untuned and unoptimized versions of benchmark (in-house developed) and Intel Prog. Env

PARAM YUVA-II Intel Xeon- Phi Benchmarks(*)

Peak Performance : Single Precision : 2129.47 Gflops/s

No. of Cores = 60

Experiment Results for Single Precision Addition of Two Vectors(*)		
No. of Cores / OpenMP threads	**Thread Affinity**	**Sustained Perf in Gflops**
4	SCATTER	66.69
8	SCATTER	133.69
16	SCATTER	231.60
32	SCATTER	480.29
64	SCATTER	947.53
120	SCATTER	1795.33
240	SCATTER	1893.56

(=No optimizations are carried-out to use OpenMP threads & Intel Prog. Env) Intel MKL Libraries are not used .)*

(No optimizations are carried-out to use OpenMP threads & Intel Prog. Env)

(*) = Speedup results were gathered using untuned and unoptimized versions of benchmarks (in-house developed) and Intel Prog. Env

References & Acknowledgements

References :

1. Theron Voran, Jose Garcia, Henry Tufo, University Of Colorado at Boulder National Center or Atmospheric Research, TACC-Intel Hihgly Parallel Computing Symposium, Austin TX, April 2012
2. Robert Harkness, Experiences with ENZO on the Intel R Many Integrated Core (Intel MIC) Architecture, National Institute for Computational Sciences, Oak Ridge National Laboratory
3. Ryan C Hulguin, National Institute for Computational Sciences, Early Experiences Developing CFD Solvers for the Intel Many Integrated Core (Intel MIC) Architecture, TACC-Intel Highly Parallel Computing Symposium April, 2012
4. Scott McMillan, Intel Programming Models for Intel Xeon Processors and Intel Many Integrated Core (Intel MIC) Architecture, TACC-Highly Parallel Comp. Symposium April 2012
5. Sreeram Potluri, Karen Tomko, Devendar Bureddy , Dhabaleswar K. Panda, Intra-MIC MPI Communication using MVAPICH2: Early Experience, Network-Based Computing Laboratory, Department of Computer Science and Engineering The Ohio State University, Ohio Supercomputer Center, TACC-Highly Parallel Computing Symposium April 2012
6. Karl W. Schulz, Rhys Ulerich, Nicholas Malaya ,Paul T. Bauman, Roy Stogner, Chris Simmons, Early Experiences Porting Scientific Applications to the Many Integrated Core (MIC) Platform ,Texas Advanced Computing Center (TACC) and Predictive Engineering and Computational Sciences (PECOS) Institute for Computational Engineering and Sciences (ICES), The University of Texas at Austin ,Highly Parallel Computing Symposium ,Austin, Texas, April 2012
7. Kevin Stock, Louis-Noel, Pouchet, P. Sadayappan ,Automatic Transformations for Effective Parallel Execution on Intel Many Integrated, The Ohio State University, April 2012
8. http://www.tacc.utexas.edu/
9. Intel MIC Workshop at C-DAC, Pune April 2013
10. First Intel Xeon Phi Coprocessor Technology Conference iXPTC 2013 New York, March 2013
11. Shuo Li, Vectorization, Financial Services Engineering, software and Services Group, Intel ctel Corporation;
12. Intel® Xeon Phi™ (MIC) Parallelization & Vectorization, Intel Many Integrated Core Architecture, Software & Services Group, Developers Relations Division

References & Acknowledgements

References :

13. Intel® Xeon Phi™ (MIC) Programming, Rama Malladi, Senior Application Engineer, Intel Corporation, Bengaluru India April 2013
14. Intel® Xeon Phi™ (MIC) Performance Tuning, Rama Malladi, Senior Application Engineer, Intel Corporation, Bengaluru India April 2013
15. Intel® Xeon Phi™ Coprocessor Architecture Overview, Dhiraj Kalamkar, Parallel Computing Lab, Intel Labs, Bangalore
16. Changkyu Kim,Nadathur Satish ,Jatin Chhugani ,Hideki Saito,Rakesh Krishnaiyer ,Mikhail Smelyanskiy ,Milind Girkar, Pradeep Dubey, Closing the Ninja Performance Gap through Traditional Programming and Compiler Technology , Technical Report Intel Labs , Parallel Computing Laboratory , Intel Compiler Lab, 2010
17. Colfax International Announces Developer Training for Intel® Xeon Phi™ Coprocessor, Industry First Training Program Developed in Consultation with Intel SUNNYVALE, CA, Nov, 2012
18. Andrey Vladimirov Stanford University and Vadim Karpusenko , Test-driving Intel® Xeon Phi™ coprocessors with a basic N-body simulation Colfax International January 7, 2013 Colfax International, 2013 http://research.colfaxinternational.com/
19. Jim Jeffers and James Reinders,Intel® Xeon Phi™ Coprocessor High-Performance Programming by Morgann Kauffman Publishers Inc, Elsevier, USA. 2013
20. Michael McCool, Arch Robison, James Reinders, Structured Parallel Programming: Patterns for Efficient Computation, Morgan Kaufman Publishers Inc, 2013.
21. Dan Stanzione, Lars Koesterke, Bill Barth, Kent Milfeld by Preparing for Stampede: Programming Heterogeneous Many-Core Supercomputers. TACC, XSEDE 12 July 2012
22. John Michalakes, Computational Sciences Center, NREL, & Andrew Porter, Opportunities for WRF Model Acceleration, WRF Users workshop, June 2012
23. Jim Rosinski , Experiences Porting NOAA Weather Model FIM to Intel MIC, ECMWF workshop On High Performance Computing in Meteorology, October 2012
24. Michaela Barth, KTH Sweden , Mikko Byckling, CSC Finland, Nevena Ilieva, NCSA Bulgaria, Sami Saarinen, CSC Finland, Michael Schliephake, KTH Sweden, Best Practice Guide Intel Xeon Phi v0.1, Volker Weinberg (Editor), LRZ Germany March 31 ,2013

References & Acknowledgements

References :

25. Barbara Chapman, Gabriele Jost and Ruud van der Pas, Using OpenMP, MIT Press Cambridge, 2008
26. Peter S Pacheco, An Introduction Parallel Programming, Morgann Kauffman Publishers Inc, Elsevier, USA. 2011
27. Intel Developer Zone: Intel Xeon Phi Coprocessor,
28. http://software.intel.com/en-us/mic-developer
29. Intel Many Integrated Core Architecture User Forum,
30. http://software.intel.com/en-us/forums/intel-many-integrated-core
31. Intel Developer Zone: Intel Math Kernel Library, http://software.intel.com/en-us
32. Intel Xeon Processors & Intel Xeon Phi Coprocessors – Introduction to High Performance Applications Development for Multicore and Manycore – Live Webinar, 26.-27, February .2013,
33. recorded http://software.intel.com/en-us/articles/intel-xeon-phi-training-m-core
34. Intel Cilk Plus Home Page, http://cilkplus.org/
35. James Reinders, Intel Threading Building Blocks (Intel TBB), O'REILLY, 2007
36. Intel Xeon Phi Coprocessor Developer's Quick Start Guide,
37. http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-developers-quick-start-guide
38. Using the Intel MPI Library on Intel Xeon Phi Coprocessor Systems,
39. http://software.intel.com/en-us/articles/using-the-intel-mpi-library-on-intel-xeon-phi-coprocessor-systems
40. An Overview of Programming for Intel Xeon processors and Intel Xeon Phi coprocessors,
41. http://software.intel.com/sites/default/files/article/330164/an-overview-of-programming-for-intel-xeon-processors-and-intel-xeon-phi-coprocessors_1.pdf
42. Programming and Compiling for Intel Many Integrated Core Architecture,
43. http://software.intel.com/en-us/articles/programming-and-compiling-for-intel-many-integrated-core-architecture
44. Building a Native Application for Intel Xeon Phi Coprocessors,
45. http://software.intel.com/en-us/articles/

References & Acknowledgements

References :

46. Advanced Optimizations for Intel MIC Architecture, http://software.intel.com/en-us/articles/advanced-optimizations-for-intel-mic-architecture
47. Optimization and Performance Tuning for Intel Xeon Phi Coprocessors - Part 1: Optimization Essentials, http://software.intel.com/en-us/articles/optimization-and-performance-tuning-for-intel-xeonphi-coprocessors-part-1-optimization
48. Optimization and Performance Tuning for Intel Xeon Phi Coprocessors, Part 2: Understanding and Using Hardware Events, http://software.intel.com/en-us/articles/optimization-and-performance-tuning-for-intel-xeon-phi-coprocessors-part-2-understanding
49. Requirements for Vectorizable Loops,
50. http://software.intel.com/en-us/articles/requirements-for-vectorizable-
51. R. Glenn Brook, Bilel Hadri, Vincent C. Betro, Ryan C. Hulguin, Ryan Braby. Early Application Experiences with the Intel MIC Architecture in a Cray CX1, National Institute for Computational Sciences. University of Tennessee. Oak Ridge National Laboratory. Oak Ridge, TN USA
52. http://software.intel.com/mic-developer
53. Loc Q Nguyen , Intel Corporation's Software and Services Group , Using the Intel® MPI Library on Intel® Xeon Phi™ Coprocessor System,
54. Frances Roth, System Administration for the Intel® Xeon Phi™ Coprocessor, Intel white Paper
55. Intel® Xeon Phi™ Coprocessor, James Reinders, Supercomputing 2012 Presentation
56. Intel® Xeon Phi™ Coprocessor Offload Compilation, Intel software

References

1. Andrews, Grogory R. (2000), Foundations of Multithreaded, Parallel, and Distributed Programming, Boston, MA : Addison-Wesley
2. Butenhof, David R (1997), Programming with POSIX Threads , Boston, MA : Addison Wesley Professional
3. Culler, David E., Jaswinder Pal Singh (1999), Parallel Computer Architecture - A Hardware/Software Approach , San Francsico, CA : Morgan Kaufmann
4. Grama Ananth, Anshul Gupts, George Karypis and Vipin Kumar (2003), Introduction to Parallel computing, Boston, MA : Addison-Wesley
5. Intel Corporation, (2003), Intel Hyper-Threading Technology, Technical User's Guide, Santa Clara CA : Intel Corporation Available at : http://www.intel.com
6. Shameem Akhter, Jason Roberts (April 2006), Multi-Core Programming - Increasing Performance through Software Multi-threading , Intel PRESS, Intel Corporation,
7. Bradford Nichols, Dick Buttlar and Jacqueline Proulx Farrell (1996), Pthread Programming O'Reilly and Associates, Newton, MA 02164,
8. James Reinders, Intel Threading Building Blocks – (2007) , O'REILLY series
9. Laurence T Yang & Minyi Guo (Editors), (2006) High Performance Computing - Paradigm and Infrastructure Wiley Series on Parallel and Distributed computing, Albert Y. Zomaya, Series Editor
10. Intel Threading Methodology ; Principles and Practices Version 2.0 copy right (March 2003), Intel Corporation
11. William Gropp, Ewing Lusk, Rajeev Thakur **(1999),** Using MPI-2, Advanced Features of the Message-Passing Interface, The MIT Press..
12. Pacheco S. Peter, **(1992),** Parallel Programming with MPI, , University of Sanfrancisco, Morgan Kaufman Publishers, Inc., Sanfrancisco, California
13. Kai Hwang, Zhiwei Xu, (**1998**), Scalable Parallel Computing (Technology Architecture Programming), McGraw Hill New York.
14. Michael J. Quinn (**2004**), Parallel Programming in C with MPI and OpenMP McGraw-Hill International Editions, Computer Science Series, McGraw-Hill, Inc. Newyork
15. Andrews, Grogory R. **(2000),** Foundations of Multithreaded, Parallel, and Distributed Progrmaming, Boston, MA : Addison-Wesley

References

16. SunSoft. Solaris multithreaded programming guide. SunSoft Press, Mountainview, CA, **(1996),** Zomaya, editor. Parallel and Distributed Computing Handbook. McGraw-Hill,
17. Chandra, Rohit, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, and Ramesh Menon, **(2001)**,Parallel Programming in OpenMP San Fracncisco Moraan Kaufmann
18. S.Kieriman, D.Shah, and B.Smaalders **(1995),** Programming with Threads, SunSoft Press, Mountainview, CA. 1995
19. Mattson Tim, **(2002),** Nuts and Bolts of multi-threaded Programming Santa Clara, CA : Intel Corporation, Available at : http://www.intel.com
20. I. Foster **(1995,** Designing and Building Parallel Programs ; Concepts and tools for Parallel Software Engineering, Addison-Wesley (1995
21. J.Dongarra, I.S. Duff, D. Sorensen, and H.V.Vorst **(1999),** Numerical Linear Algebra for High Performance Computers (Software, Environments, Tools) SIAM, 1999
22. OpenMP C and C++ Application Program Interface, Version 1.0". **(1998),** OpenMP Architecture Review Board. October 1998
23. D. A. Lewine. *Posix Programmer's Guide:* **(1991),** Writing Portable Unix Programs with the Posix. 1 Standard. O'Reilly & Associates, 1991
24. Emery D. Berger, Kathryn S McKinley, Robert D Blumofe, Paul R.Wilson, *Hoard : A Scalable Memory Allocator for Multi-threaded Applications* ; The Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX). Cambridge, MA, November (**2000**). Web site URL : http://www.hoard.org/
25. Marc Snir, Steve Otto, Steyen Huss-Lederman, David Walker and Jack Dongarra, (**1998**) *MPI-The Complete Reference: Volume 1, The MPI Core, second edition* [MCMPI-07].
26. William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir (**1998**) *MPI-The Complete Reference: Volume 2, The MPI-2 Extensions*
27. A. Zomaya, editor. Parallel and Distributed Computing Handbook. McGraw-Hill, **(1996)**
28. OpenMP C and C++ Application Program Interface, Version 2.5 (**May 2005**)", From the OpenMP web site, URL **: http://www.openmp.org/**
29. Stokes, Jon 2002 Introduction to Multithreading, Super-threading and Hyper threading *Ars Technica*, October **(2002)**

References

30. Andrews Gregory R. 2000, Foundations of Multi-threaded, Parallel and Distributed Programming, Boston MA : Addison – Wesley (**2000)**
31. Deborah T. Marr , Frank Binns, David L. Hill, Glenn Hinton, David A Koufaty, J . Alan Miller, Michael Upton, "Hyperthreading, Technology Architecture and Microarchitecture", Intel (**2000-01)**
32. http://www.erc.msstate.edu/mpi
33. http://www.arc.unm.edu/workshop/mpi/mpi.html
34. http://www.mcs.anl.gov/mpi/mpich
35. The MPI home page, with links to specifications for MPI-1 and MPI-2 standards : http://www.mpi–forum.org
36. Hybrid Programming Working Group Proposals, Argonne National Laboratory, Chiacago (2007-2008)
37. TRAC Link : https://svn.mpi-forum.org/trac/mpi-form-web/wiki/MPI3Hybrid
38. Threads and MPI Software, Intel Software Products and Services 2008 - 2009
39. Sun MPI 3.0 Guide November 2007
40. Treating threads as MPI processes thru Registration/deregistration –Intel Software Products and Services 2008 – 2009
41. Intel MPI library 3.2 - http://www.hearne.com.au/products/Intelcluster/edition/mpi/663/
42. http://www.cdac.in/opecg2009/
43. PGI Compilers http://www.pgi.com

Thank You

Any questions ?