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 An Overview

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

An Overview of Xeon Phi Coprocessor

Lecture Outline

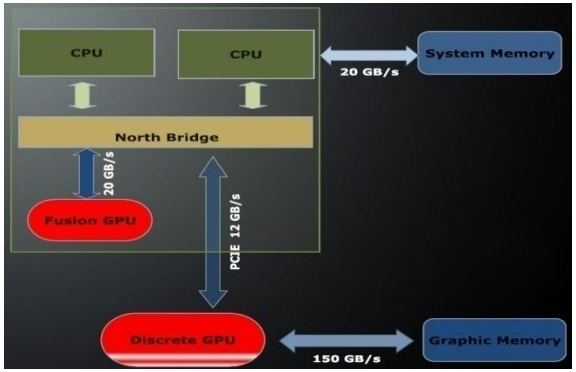
Following topics will be discussed

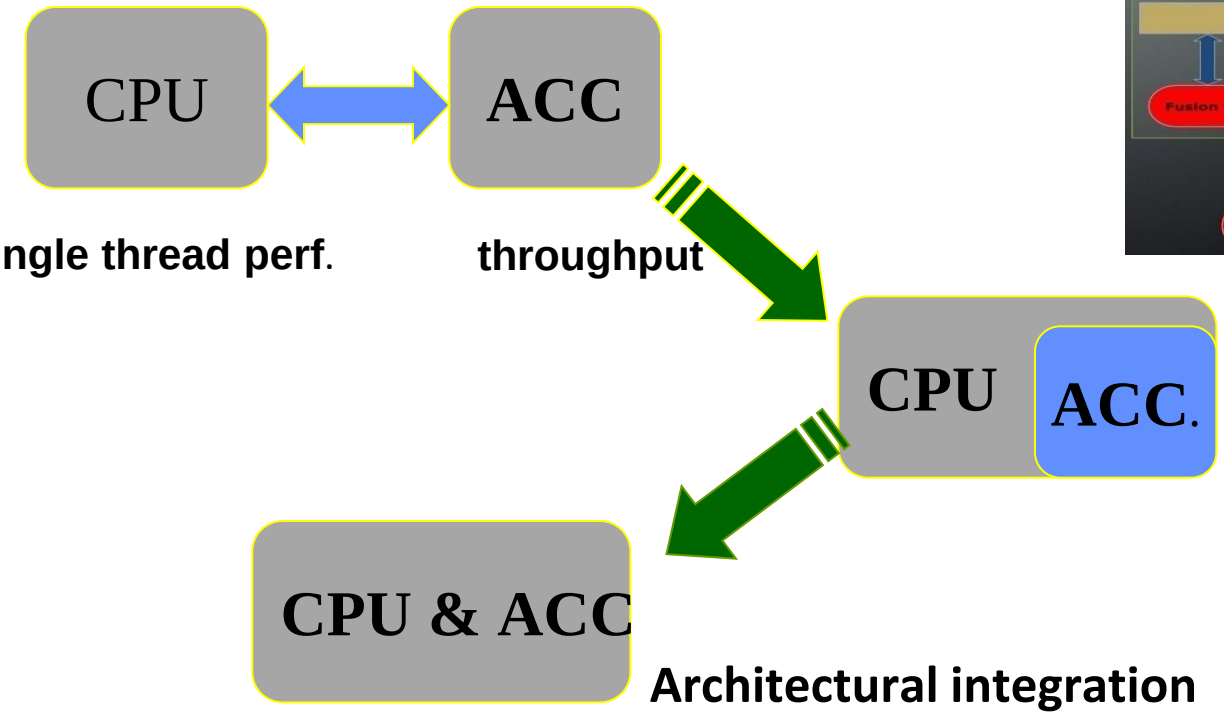
- Understanding of Xeon –Phi Architectures
- Programming Environment on Xeon-Phi Architectures – Compilation
- Tuning & Performance – Compilation & Vectorization

Programming Environment
Compilation
& Performance Issues

Systems with Accelerators

A set (one or more) of very simple execution units that can perform few operations (with respect to standard CPU) with very high efficiency. When combined with full featured CPU (CISC or RISC) can accelerate the “nominal” speed of a system.



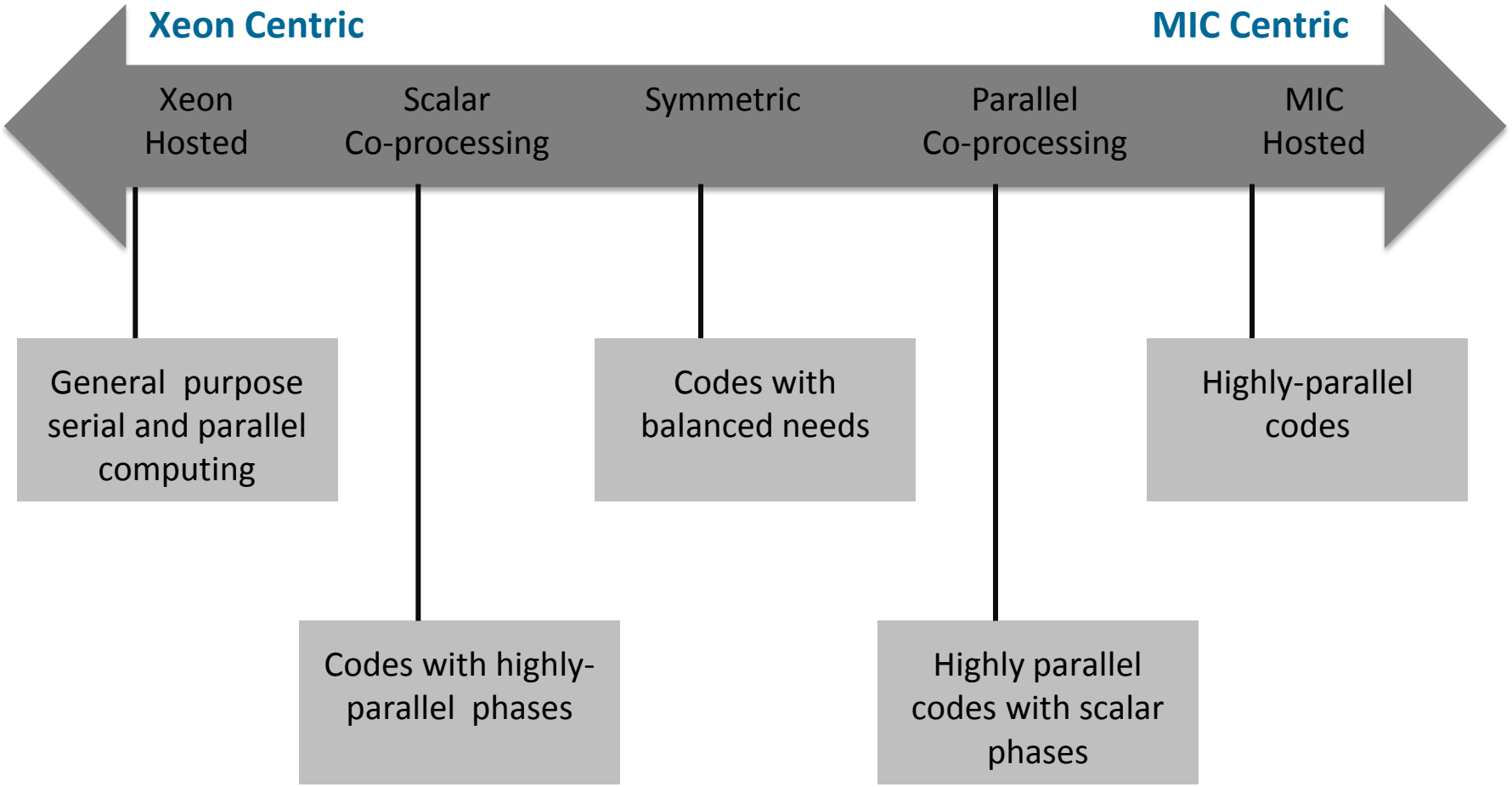


CPU : Control Path & Data Path
ALU : Arithmetic and logic units
ACC: Accumulator
PC : Program counter
Micro-Instructions (Register transfer)

Source : NVIDIA, AMD, SGI, Intel, IBM Alter, Xilinux References

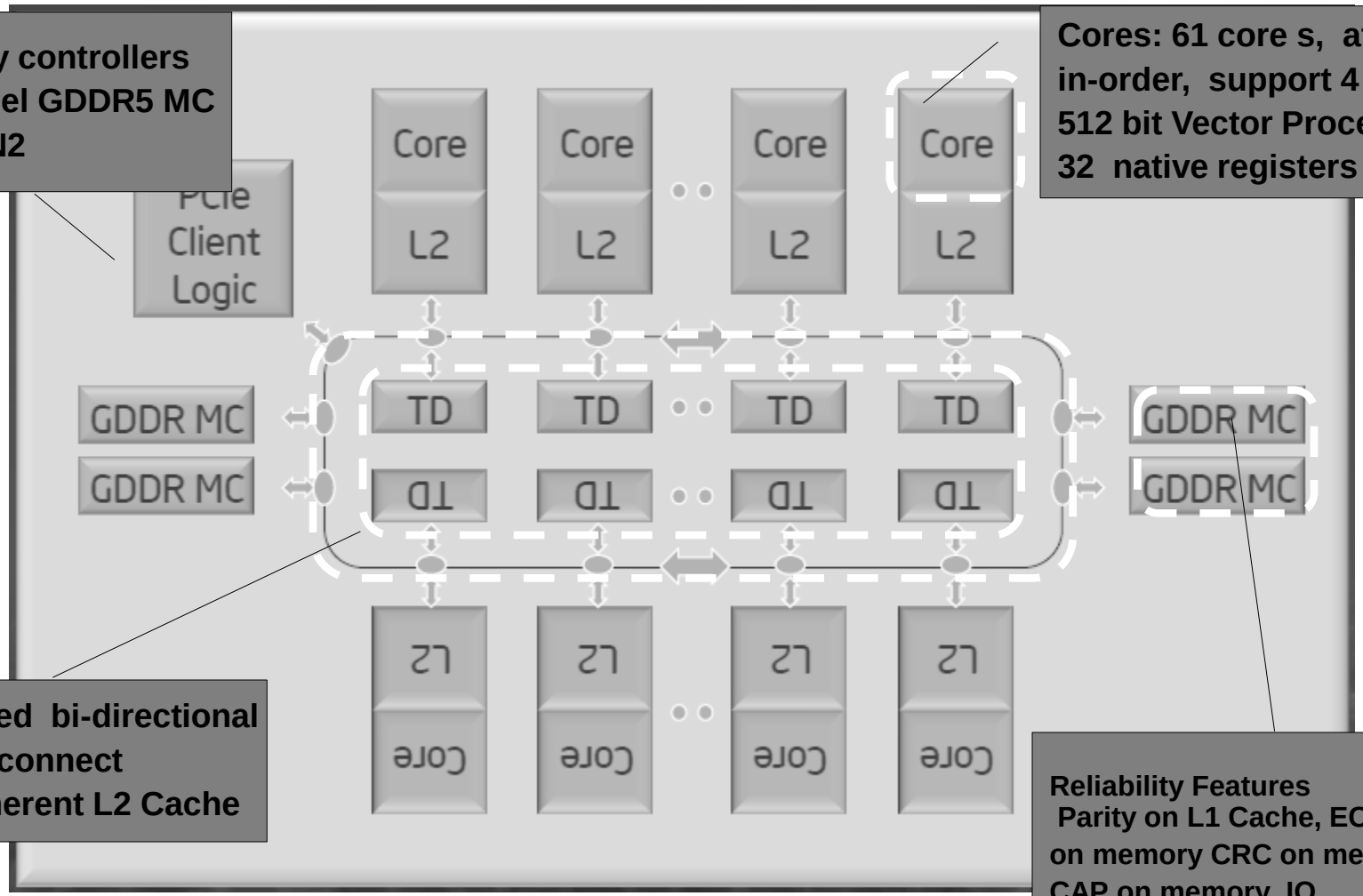
MIC Architecture, System Overview

Compute modes vision



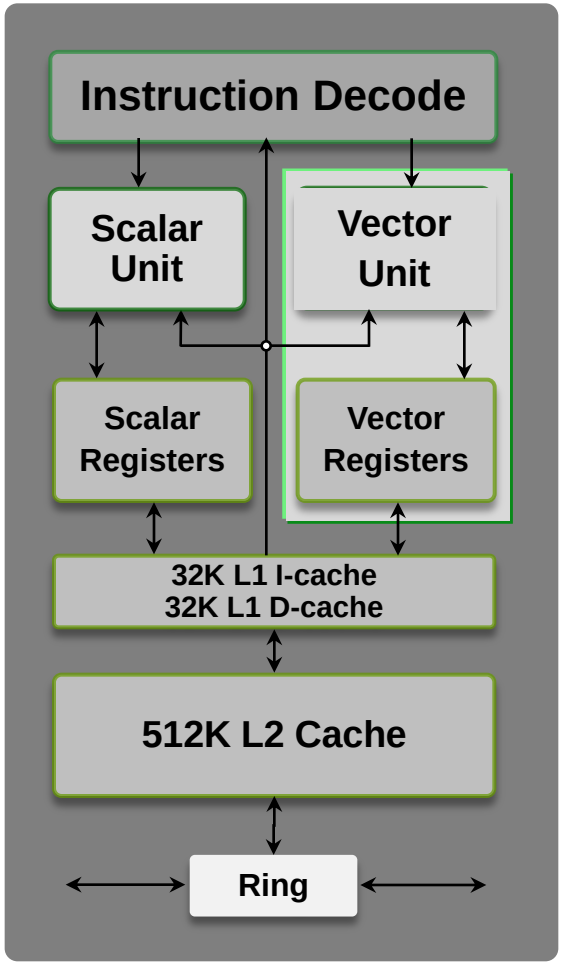
Source : References & Intel Xeon-Phi; http://www.intel.com/

Intel® Xeon Phi™ Architecture Overview



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

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

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

Intel Xeon-Phi Coprocessor architecture Overview

Quick Glance*

- The Intel Xeon Phi coprocessor Architecture Overview (Core, VPU, CRI, Ring, SBOX, GBOX, PMU)
- The Cache hierarchy (Details of L1 & L2 Cache)
- Network Configuration (MPSS) : (Obtain the information can be obtained by running the micinfo program on the host.)
- System Access

Remark : Root privileges are necessary for the destination directories (Required for availability of some library usage for codes such MKL)

(= Useful for tuning and Performance)*

Intel Xeon-Phi Coprocessor architecture Overview

- The Intel Xeon Phi coprocessor consists of up to 61 cores connected by a high performance on-die bidirectional interconnect.
- The coprocessor runs a full service Linux operating system
- The coprocessor supports all important Intel development tools, like C/C++ and Fortran compiler, MPI and OpenMP
- To Coprocessor support s high performance libraries like MKL, debugger and tracing tools like Intel VTune Amplifier XE.

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

Intel Xeon-Phi Coprocessor architecture Overview

- The Intel Xeon Phi coprocessor The coprocessor is connected to an Intel Xeon processor - the "host" - via the PCI Express (PICe) bus.
- The implementation of a virtualized TCP/IP stack allows to access the coprocessor like a network node.

Remark : Summarized information can be found In the following MIC architecture from the System Software Developers Guide and other references

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

Intel Xeon-Phi Coprocessor System Access

Quick Glance:

- Details about the system startup and the network configuration can be found in Intel Xeon-Phi documentation coming with MPSS
- To start the Intel Manycore Platform Software Stack (Intel MPSS) and initialize the Xeon Phi coprocessor the following command has to be executed as root or during host system start-up:

```
hypack-root@mic-0:~> sudo service mpss start
```

Remark : The above command has to be executed as a root

Intel Xeon-Phi Coprocessor System Access

Quick Glance:

- To start the Intel Manycore Platform Software Stack (Intel MPSS) and initialize the Xeon Phi coprocessor the following command has to be executed as root or during host system start-up:

```
hypack-root@mic-0:~> sudo service mpss start
```

Remark : The above command has to be executed as a root.

Details about the system startup and the network configuration can be found in Intel Xeon-Phi documentation coming with MPSS. For other necessary commands, refer Intel Xeon Phi documentation

Intel Xeon-Phi Coprocessor System Access

Quick Glance:

- Deafault IP addresses ???•?? •?•??? , ???•?? •?•???, etc. are assigned to the attached Intel Xeon Phi coprocessors. The IP addresses of the attached coprocessors can be listed via the traditional **ifconfig** Linux program.

```
hypack-root@mic-0:~> /sbin/ifconfig
```

Further information can be obtained by running the **micinfo** program on the host.

```
hypack-root@mic-0:~>/sudo/opt/intel/mic/bin/micinfo
```

Intel Xeon-Phi Coprocessor System Access

Quick Glance:

```
hypack-root@mic-0:~>/sudo/opt/intel/mic/bin/micinfo
System Info
 Host OS : Linux
 OS Version : 3.0.13-0.27-default
 Driver Version : 4346-16
 MPSS Version : 2.1.4346-16
 Host Physical Memory : 66056 MB
 ……
 Device No: 0, Device Name: Intel(R) Xeon Phi(TM) coprocessor
 ……
 Version
 ……..…
 Board
 …….
```

Intel Xeon-Phi Coprocessor System Access

Quick Glance:

```
hypack-root@mic-0:~>/sudo/opt/intel/mic/bin/micinfo
 Device No: 0, Device Name: Intel(R) Xeon Phi(TM) coprocessor
 ……..
 Core
 ……..…
 Thermal
 ……..
 GGDR
 ……..
 Device No: 1, Device Name: Intel(R) Xeon Phi(TM) coprocessor
 ……..
 ……..
 ……..
```

Intel Xeon-Phi Coprocessor System Access

Quick Glance:

```
hypack-root@mic-0:~>/sudo/opt/intel/mic/bin/micinfo
 Device No: 0, Device Name: Intel(R) Xeon Phi(TM) coprocessor
 ……..
 Core
 ……..…
```

Intel Xeon-Phi Coprocessor System Access

Quick Glance:

Users can log in directly onto the Xeon Phi coprocessor via ssh. User can get basic information abbot Xeon-Phi by executing the following commands.

```
[hypack01@mic-0]$ ssh mic-0
[hypack01@mic-0]$ hostname
.....
[hypack01@mic-0]$ cat /etc/issue
 Intel MIC Platform Software Stack release 2.1
```

To get further information about the cores, memory etc. can be obtained from the virtual Linux /proc or /sys filesystems:

```
[weinberg@knf1-mic0 weinberg]$
[hypack01@mic-0]$ tail -n26 /proc/cpuinfo
.....
```

Intel Xeon-Phi : Performance-Tips

Performance on Xeon Phi using different prog.

- **Rule of thumb :** An application must scale well past one hundred threads on Intel Xeon processors to profit from the possible higher parallel performance offered with e.g. the Intel Xeon Phi coprocessor.

- The scaling would profit from utilising the highly parallel capabilities of the MIC architecture, you should start to create a simple performance graph with a varying number of threads (from one up to the number of cores)

Intel Xeon-Phi : Performance-Tips

Performance on Xeon Phi using different prog.

- **What we should know from programming point of view :** We treat the coprocessor as a 64-bit x86 **SMP-on-a-chip** with an high-speed bi-directional **ring** interconnect, (up to) **four** hardware threads per core and **512-bit SIMD** instructions.

- With the available number of cores, we have easily 200 hardware threads at hand on a single coprocessor.

Intel Xeon System & Xeon-Phi

Performance on Xeon Phi using different prog.

About Hyper-Threading

- hyper-threading hardware threads can be switched off and can be ignored.

About Threading on Xeon-Phi Coprocessor

- The multi-threading on each core is primarily used to hide latencies that come implicitly with an in-order microarchitecture. Unlike hyper-threading these hardware threads cannot be switched off and should never be ignored.
- In general a minimum of **three** or **four** active threads per cores will be needed.

Programming Environment
Native Compilation
& Intel Complier Offload Pragmas

Intel Xeon-Phi Coprocessor System Access

Quick Glance:

- In native mode an application is compiled on the host using the compiler switch **-mmic** to generate code for the MIC architecture. The binary can then be copied to the coprocessor and has to be started there.
- Vector-Vector-Multiplication

```
[hypack01@mic-0]$ icc -O3 -mmic vv.c -o vv
[hypack01@mic-0]$ scp vv mic0:
    program 100% 10KB 10.2KB/s 00:00
[hypack01@mic-0]$ ssh mic0 ~/run
      vector-vector Multiplication = 16.00
```

Intel Xeon-Phi Coprocessor System Access

Quick Glance:

In native mode an application is compiled on the host using the compiler switch **-mmic** to generate code for

the MIC architecture. The binary can then be copied to the coprocessor and has to be started there.

```
[hypack01@mic-0]$ icc -O3 -mmic test.c -o test

[hypack01@mic-0]$ scp test mic0:
    program 100% 10KB 10.2KB/s 00:00

[hypack01@mic-0]$ ssh mic0 ~/test
                  hello world
```

Intel Xeon Phi Coprocessor :Native Compilation

To achieve good Performance - Following information should be kept in mind.

- Data should be **aligned to 64 Bytes (512 Bits)** for the MIC architecture, in contrast to 32 Bytes (256 Bits) for AVX and 16 Bytes (128 Bits) for SSE.
- Due to the large SIMD width of 64 Bytes **vectorization is even more important for the MIC architecture than for Intel Xeon!**
- The MIC architecture offers **new instructions** like
  - **gather/scatter,**
  - **fused multiply-add,**
  - **masked vector instructions etc.**

  which allow more loops to be parallelized on the coprocessor than on an **Intel Xeon based host**.

Intel Xeon Phi Coprocessor : Native Compilation

To achieve good Performance - Following information should be kept in mind.

Use pragmas like

- `#pragma ivdep,`
- `#pragma vector always,`
- `#pragma vector aligned,`
- `#pragma simd`

etc. to achieve autovectorization.

Autovectorization is enabled at default optimization level `-O2`. Requirements for vectorizable loops can be found references.

Intel Xeon Phi Coprocessor : Native Compilation

To achieve good Performance - Following information should be kept in mind.

- Let the compiler generate vectorization reports using the compiler option **`-vecreport2`** to see if loops were vectorized for MIC (Message "*MIC* Loop was vectorized" etc).

- The options –**`opt-report-phase hlo`** (High Level Optimizer Report) or

  –**`opt-report-phase ipo_inl`** (*Inlining* report) may also be useful.

Intel Xeon Phi Coprocessor :Native Compilation

To achieve good Performance - Following information should be kept in mind.

- Explicit vector programming is also possible via Intel Cilk Plus language extensions (C/C++ array notation, vector elemental functions, ...) or the new SIMD constructs from OpenMP 4.0 RC1.
- Vector elemental functions can be declared by using `__attributes__((vector))`. The compiler then generates a vectorized version of a scalar function which can be called from a vectorized loop.

Intel Xeon Phi Coprocessor : Native Compilation

To achieve good Performance - Following information should be kept in mind.

- One can use intrinsics to have full control over the vector registers and the instruction set.
- Include `<immintrin.h>` for using intrinsics.
- **Hardware prefetching** from the L2 cache is enabled per default.
- In addition, **software prefetching** is on by default at compiler optimization level `-O2` and above. Since Intel Xeon Phi is an `inorder` architecture, care about prefetching is more important than on `out-of-order` architectures.

Intel Xeon Phi Coprocessor : Native Compilation

To achieve good Performance - Following information should be kept in mind.

- The compiler prefetching can be influenced by setting the compiler switch **`-opt-prefetch = n`**.

- Manual prefetching can be done by using intrinsics **`(_mm_prefetch`**`()` or

  pragmas (#**`pragma prefetch var`**).

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

- Simply add OpenMP-like pragmas to C/C++ or Fortran code to mark regions of code that should be offloaded to the Intel Xeon Phi Coprocessor and be run there.

- This approach is quite similar to the accelerator pragmas introduced by the

  - NVIDIA - PGI compiler,
  - CAPS HMPP or
  - OpenACC to offload code to GPGPUs.

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

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

❖ Work done – Compiler's Offload

1. When the Intelcompiler encounters an offload pragma, it generates code for both the coprocessor and the host.
2. Code to transfer the data to the coprocessor is automatically created by the compiler,
3. The programmer can influence the data transfer by adding data clauses to the offload pragma.

Details can be found under "**Offload Using a Pragma**" in the Intel compiler documentation.

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

A simple example how to offload a **matrix-matrix computation** to the coprocessor. (*No function or subroutine) is included*

```
main(){
  double *a, *b, *c;
  int i,j,k, ok, n=100;

  // allocated memory on the heap aligned to 64 byte boundary
  ok = posix_memalign((void**)&a, 64, n*n*sizeof(double));
  ok = posix_memalign((void**)&b, 64, n*n*sizeof(double));
  ok = posix_memalign((void**)&c, 64, n*n*sizeof(double));

// initialize matrices
  ...
```

Code "***Simple example for matrix-matrix computation***" – may not give good performance on all cores

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

```
//offload code
#pragma offload target(mic) in(a,b:length(n*n))
inout(c:length(n*n))
{
//parallelize via OpenMP on MIC
#pragma omp parallel for
  for( i = 0; i < n; i++ ) {
    for( k = 0; k < n; k++ ) {
#pragma vector aligned
#pragma ivdep
 for( j = 0; j < n; j++ ) {
   //c[i][j] = c[i][j] + a[i][k]*b[k][j];
     c[i*n+j] = c[i*n+j] + a[i*n+k]*b[k*n+j];
         }
       }
     }
   }
}
```

Code "***Simple example for matrix-matrix computation***" – may not give good performance on all cores

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

Summary of Example Program

1. Shows how to offload the matrix computation to the coprocessor using the `#pragma offload target(mic)`.

2. One could also specify the specific coprocessor `num` in a system with multiple coprocessors by using `#pragma offload target(mic:num)`

3. Matrices have been dynamically allocated using `posix_memalign()`, their sizes must be specified via the `length()` clause.

   It is recommended that for Intel Xeon Phi data is 64-byte aligned

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

Summary of Example Program

1. Shows how to offload the matrix computation to the coprocessor using the `#pragma offload target(mic)`.

1. `#pragma vector aligned` tells the compiler that all array data accessed in the loop is properly aligned.

2. `#pragma ivdep` discards any data dependencies assumed by the compiler

Offloading is enabled per default for the Intel compiler. Use `-no-offload` to disable the generation of offload code.

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

Obtain Offload Information about the following

Using the compiler option **-vec-report2,** one can see which loops have been vectorized on the host & the MIC coprocessor:

```
[hypack01@mic-0]$ icc -vec-report2 -openmp offload.c

offload.c(57): (col. 2) remark: loop was not vectorized:
           vectorization possible but seems inefficient.

...
offload.c(57):(col. 2) remark: *MIC* LOOP WAS VECTORIZED.
offload.c(54):(col. 7) remark: *MIC* loop was not
                  vectorized: not inner loop.
offload.c(53): (col. 5) remark: *MIC* loop was not
                  vectorized: not inner loop.
```

Mind the **C99** restrict keyword that specifies that the vectors do not overlap. (Compile with **-std=c99**)

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

Obtain Offload Information about the following

By setting the environment variable **OFFLOAD_REPORT** one can obtain information about per.& data transfers at runtime:

```
[hypack01@mic-0]$ export OFFLOAD_REPORT=2
[hypack01@mic-0]$ ./a.out
[Offload] [MIC 0] [File] offload2.c
[Offload] [MIC 0] [Line] 50
[Offload] [MIC 0] [CPU Time] 12.853562 (seconds)
[Offload] [MIC 0] [CPU->MIC Data] 9830416 (bytes)
[Offload] [MIC 0] [MIC Time] 12.208636 (seconds)
[Offload] [MIC 0] [MIC->CPU Data] 3276816 (bytes)
offload.c(53): (col. 5) remark: *MIC* loop was not
                vectorized: not inner loop.
```

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

A simple example how to offload a **matrix-matrix computation** to the coprocessor. (*No function or subroutine) is included*

If a function is called within the offloaded code block, this function has to be declared with

```
__attribute__((target(mic)))
```

to disable the generation of offload code.

Code " ***Simple example for matrix-matrix computation***" – may not give good performance on all cores

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

A simple example how to offload a **matrix-matrix computation** a *subroutine and call that routine within an offloaded block region:*

```
attribute__((target(mic))) void mxm( int n, \
      double *restrict a, double * restrict b, \
      double *restrict c ){
   int i,j,k;
   for( i = 0; i < n; i++ ) {
   ...
}
main(){
...
#pragma offload target(mic) \
     in(a,b:length(n*n)) inout(c:length(n*n))
  {
   mxm(n,a,b,c);
  }
}
```

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

Syntax of Programs

Pragma	Syntax	Semantic
	C++	
Offload pragma	`#pragma offload <clauses> <statement>`	Allow next statement to execute on coprocessor or host CPU
Variable/function offload properties	`__attribute__ ((target(mic)))`	Compile function for, or allocate variable on, both host CPU and coprocessor
Entire blocks of data/code defs	`#pragma offload_attribute(push, target(mic)) ... #pragma offload_attribute(pop)`	Mark entire files or large blocks of code to compile for both host CPU and coprocessor

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

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

Syntax of Programs

Pragma	Syntax	Semantic
Fortran		
Offload directive	`!dir$ omp offload <clauses> <statement>`	Execute OpenMP parallel block on coprocessor
Variable/function offload properties	`!dir$ attributes offload:<mic> :: <ret-name> OR <var1,var2,…>`	Compile function or variable for CPU and coprocessor
Entire code blocks	`!dir$ offload begin <clauses> ... !dir$ end offload`	Mark entire files or large blocks of code to compile for both host CPU and coprocessor

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

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

Syntax of Programs

The following clauses can be used to control data transfers:

Clause	Syntax	Semantic
Multiple coprocessors	`target(mic[:unit])`	Select specific coprocessors
Inputs	`in(var-list modifiers)`	Copy from host to coprocessor
Outputs	`out(var-list modifiers)`	Copy from coprocessor to host
Inputs & Outputs	`inout(var-list modifiers)`	Copy host to coprocessor and back when offload completes
Non-copied data	`nocopy(var-list modifiers)`	Data is local to target

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

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

Syntax of Programs

The following (optional) modifiers are specified:

Modifier	Syntax	Semantic
Specify copy length	`length(N)`	Copy N elements of pointer's type
Coprocessor memory allocation	`alloc_if ( bool )`	Allocate coprocessor space on this offload (default: TRUE)
Coprocessor memory release	`free_if ( bool )`	Free coprocessor space at the end of this offload (default: TRUE)
Control target data alignment	`align ( N bytes )`	Specify minimum memory alignment on coprocessor
Array partial allocation & variable relocation	`alloc (array-slice ) into ( var-expr )`	Enables partial array allocation and data copy into other vars & ranges

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

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

Explicit Worksharing

```
#pragma omp parallel
  {
#pragma omp sections
    {
#pragma omp section
     {
//section running on the coprocessor
#pragma offload target(mic) in(a,b:length(n*n)) inout(c:length(n*n))
 {
   mxm(n,a,b,c);
 }
      }
#pragma omp section
      {
//section running on the host
mxm(n,d,e,f);
      }
   }
}
```

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

Persistent data on the coprocessor

- The main bottleneck of accelerator based programming are data transfers over the slow PCIe bus from the **host** to the accelerator and vice versa.
- To increase the performance one should minimize data transfers as much as possible and keep the data on the coprocessor between computations using the same data.
- Defining the following macros

```
#define ALLOC alloc_if(1)
#define FREE free_if(1)
#define RETAIN free_if(0)
#define REUSE alloc_if(0)
```

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

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

Persistent data on the coprocessor

- The main bottleneck of accelerator based programming are data transfers over the slow PCIe bus from the **host** to the accelerator and vice versa.
- one can simply use the following notation: to allocate data and keep it for the next offload

```
#pragma offload target(mic)in (p:length(l) ALLOC RETAIN)
```

- to reuse the data and still keep it on the coprocessor

```
#pragma offload target(mic)in (p:length(l) REUSE RETAIN)
```

- to reuse the data again and free the memory. (`FREE` is the default, and does not need to be explicitly specified)

```
#pragma offload target(mic) in (p:length(l) REUSE FREE)
```

More information can be found in the section "Managing Memory Allocation for Pointer Variables" under "Offload Using a Pragma"

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

Optimised Offloaded Code

- Optimizing offloaded code
- The implementation of the matrix-matrix multiplication can be optimized by defining appropriate ROWCHUNK and COLCHUNK chunk sizes.
- Rewrite the code with 6 nested loops (using OpenMP col-apse for the 2 outermost loops) and some manual loop unrolling

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

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

Optimizing Offloaded Code

```
#define ROWCHUNK 96
#define COLCHUNK 96
#pragma omp parallel for collapse(2) private(i,j,k)
  for(i = 0; i < n; i+=ROWCHUNK ) {
    for(j = 0; j < n; j+=ROWCHUNK ) {
 for(k = 0; k < n; k+=COLCHUNK ) {
  for (ii = i; ii < i+ROWCHUNK; ii+=6) {
   for (kk = k; kk < k+COLCHUNK; kk++ ) {
#pragma ivdep
#pragma vector aligned
       for ( jj = j; jj < j+ROWCHUNK; jj++){
  c[(ii*n)+jj] += a[(ii*n)+kk]*b[kk*n+jj];
  c[((ii+1)*n)+jj] += a[((ii+1)*n)+kk]*b[kk*n+jj];
  c[((ii+2)*n)+jj] += a[((ii+2)*n)+kk]*b[kk*n+jj];
  c[((ii+3)*n)+jj] += a[((ii+3)*n)+kk]*b[kk*n+jj];
  c[((ii+4)*n)+jj] += a[((ii+4)*n)+kk]*b[kk*n+jj];
  c[((ii+5)*n)+jj] += a[((ii+5)*n)+kk]*b[kk*n+jj];
                }
              }
            }
          }
      }
    }
}
```

Intel Xeon Phi : Coprocessors – Intel Compiler's Offload Programs

Optimised Offloaded Code

Tuning & Performance :

- Using intrinsics with manual data prefetching and register blocking can still considerably increase the performance.

- Try to get a suitable vectorization and write cache and register efficient code, i.e. values stored in registers should be reused as often as possible in order to avoid cache and memory access.

Intel Xeon-Phi Compilation & Vectorization

Use Compiler Optimization Switches

Optimization Done	Linux*
Disable optimization	-O0
Optimize for speed (no code size increase)	-O1
Optimize for speed (default)	-O2
High-level loop optimization	-O3
Create symbols for debugging	-g
Multi-file inter-procedural optimization	-ipo
Profile guided optimization (multi-step build)	-prof-gen -prof-use
Optimize for speed across the entire program	-fast (same as: -ipo –O3 -no-prec-div -static -xHost)
OpenMP 3.0 support	-openmp
Automatic parallelization	-parallel

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

Intel Xeon Phi : Coprocessors – Native Compilation
Complier Offload Pragmas

Compiler Reports – Optimization Report

Compiler switch:

`-opt-report-phase[=phase]`

`phase` can be:

- **`ipo_inl`** - Interprocedural Optimization Inlining Report
- **`ilo`** — Intermediate Language Scalar Optimization
- **`hpo`** — High Performance Optimization
- **`hlo`** — High-level Optimization
- **`all`** — All optimizations (not recommended, output too verbose)

Control the level of detail in the report:

`-opt-report[0|1|2|3]`

If you do not specify the option, no optimization report is being generated; if you do not specify the level (i.e. -opt-report) level 2 is being used by the compiler.

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

Hints to Compiler for Vectorization

#pragma	Semantics
#pragma ivdep	Ignore vector dependences unless they are proven by the compiler
#pragma vector always [assert]	If the loop is vectorizable, ignore any benefit analysis If the loop did not vectorize, give a compile-time error message via assert
#pragma novector	Specifies that a loop should never be vectorized, even if it is legal to do so, when avoiding vectorization of a loop is desirable (when vectorization results in a performance regression)
#pragma vector aligned / unaligned	instructs the compiler to use aligned (unaligned) data movement instructions for all array references when vectorizing
#pragma vector temporal / nontemporal	directs the compiler to use temporal/non-temporal (that is, streaming) stores on systems based on IA-32 and Intel® 64 architectures; optionally takes a comma separated list of variables

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

Intel Xeon Phi : Coprocessors – Native Compilation
Complier Offload Pragmas

Get Your Code Vectorized by Intel Compiler

- Data Layout, AOS -> SOA
- Data Alignment (next slide)
- Make the loop innermost
- Function call in treatment
  - Inline yourself
  - inline! Use `__forceinline`
  - Define your own vector version
  - Call vector math library - SVML
- Adopt jumpless algorithm
- Read/Write is OK if it's continuous
- Loop carried dependency

Array of Structures		
S0	X0	T0
S1	X1	T1
…	…	…

Structure of Arrays		
S0	S1	…
X0	X1	…
S0	S1	…

Not a true dependency

```
for(int i = TIMESTEPS; i > 0; i--)
        #pragma simd
        #pragma unroll(4)
  for(int j = 0; j <= i - 1; j++)
cell[j]=puXDf*cell[j+1]+pdXDf*cell[j];
CallResult[opt] = (Basetype)cell[0];
```

A true dependency

```
for (j=1; j<MAX; j++)
a[j] = a[j] + c * a[j-n];
```

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

Intel Xeon Phi : Coprocessors – Native Compilation
Complier Offload Pragmas

Compiler VECreport

- Indicates whether each loop is vectorized
  - Vectorized ≠ efficient
- Different levels
  - -vec-report1, for high-level triage of large code
  - -vec-report2, when you want reasons for not vectorizing
  - -vec-report6, for even more detail, e.g. misalignment
- Indicates reasons for not vectorizing
  - Unsupported datatype → rewrite to use 32b indices vs. 64b
- Line numbers may not be what you expect
  - Inlining
  - Loop distribution, interchange, unrolling, collapsing

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

Compiler-Based Autovectorization

- Compiler recreate vector instructions from the serial Program
- Compiler make decisions based on some assumption
- The programmer reassures the compiler on those assumptions
  - The compiler takes the directives and compares them with its analysis of the code
- Compiler checks for
  - Is "*p" loop invariant?
  - Are a, b, and c loop invariant?
  - Does a[] overlap with b[], c[], and/or sum?
  - Is "+" operator associative? (Does the order of "a
  - Vector computation on the target expected to be faster than scalar code?

```
#pragma simd reduction(+:sum)
     for(i=0;i<*p;i++) {
       a[i] = b[i]*c[i];
       sum = sum + a[i];
             }
```

- Compiler Confirms this loop :
  - "*p" is loop invariant
  - a[] is not aliased with b[], c[], and sum
  - sum is not aliased with b[] and c[]
  - "+" operation on sum is associative (Compiler can reorder the "add"s on sum)
  - Vector code to be generated even if it could be slower than scalar code

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

Intel Xeon Phi : Coprocessors – Native Compilation
Complier Offload Pragmas

Compiler OPT report - contents

- Control over static reports
  - -opt-report [n=0-3] enables varying levels of detail
  - -opt-report-phase=[several options] enables specific detail
- Reveals info on various compiler optimization
  - Offloaded variables, –opt-report-phase=offload
  - Inlining, Vectorization
  - OpenMP parallelization, auto-parallelization
  - Loop permutations, loop distribution, loop distribution
  - Multiversioning of loops performed by compiler
    - Dynamic dependence checking, unit-stride for assumed shape arrays, trip-count checks, etc.
  - Prefetching
  - Blocking, unrolling, jamming
  - Whole-program optimization

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

Summary: Tricks for Performance

- Use asynchronous data transfer and double buffering offloads to overlap the communication with the computation
- Optimizing memory use on Intel MIC architecture target relies on understanding access patterns
- Many old tricks still apply: peeling, collapsing, unrolling, vectorization can all benefit performance

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

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

An Overview of Multi-Core Processors

Conclusions

- An Overview of Intel Xeon-Phi Architectures, Programming on Xeon-Phi using Native Compilation and Compilation & Vectorization Techniques

Thank You

Any questions ?