C-DAC Four Days Technology Workshop

ON

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

hyPACK-2013
Mode 3 : Intel Xeon Phi Coprocessors

Lecture Notes :
Intel Xeon Phi Coprocessor - An Overview

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

An Overview of Prog. Env on Intel Xeon-Phi

Lecture Outline

Following topics will be discussed

- Understanding of Intel Xeon-Phi Coprocessor Architecture
- Programming on Intel Xeon-Phi Coprocessor
- Performance Issues on Intel Xeon-Phi Coprocessor

Intel Xeon Host : An Overview of Xeon - Multi-Core and Systems with Devices

Part-I

Background : Xeon Host - Multi-Core & Devices

Programming paradigms-Challenges
Large scale data Computing – Current trends

How to run Programs faster ?

You require Super Computer

Era of Single - Multi-*to*-Many Core - Heterogeneous Computing

Sequential Computing

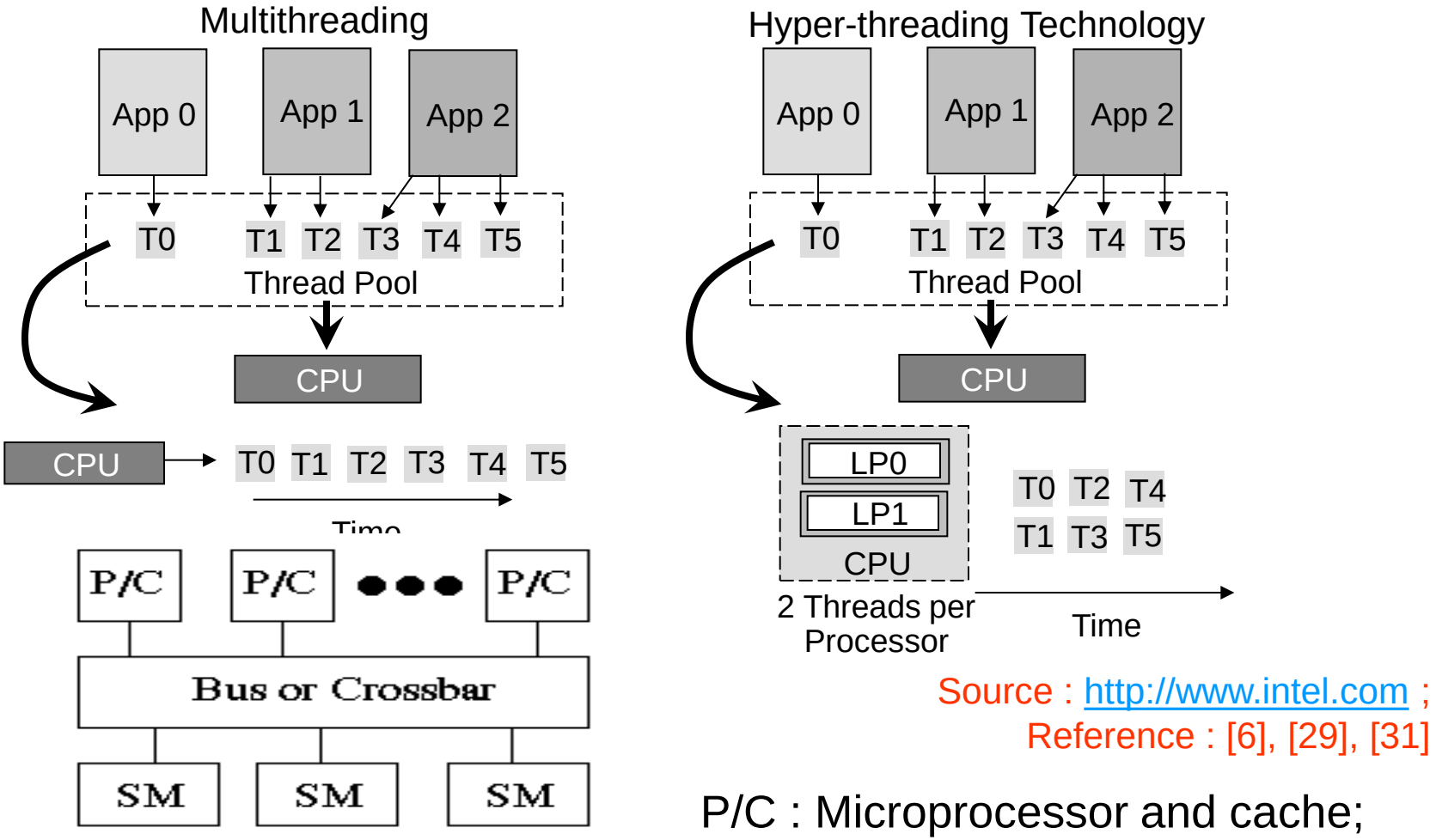
- Fetch/Store
- Compute

How to run Programs faster ?

- Fast Access of data
- Fast Processor
- More Memory to Manage data

Multi-threaded Processing using Hyper-Threading Technology

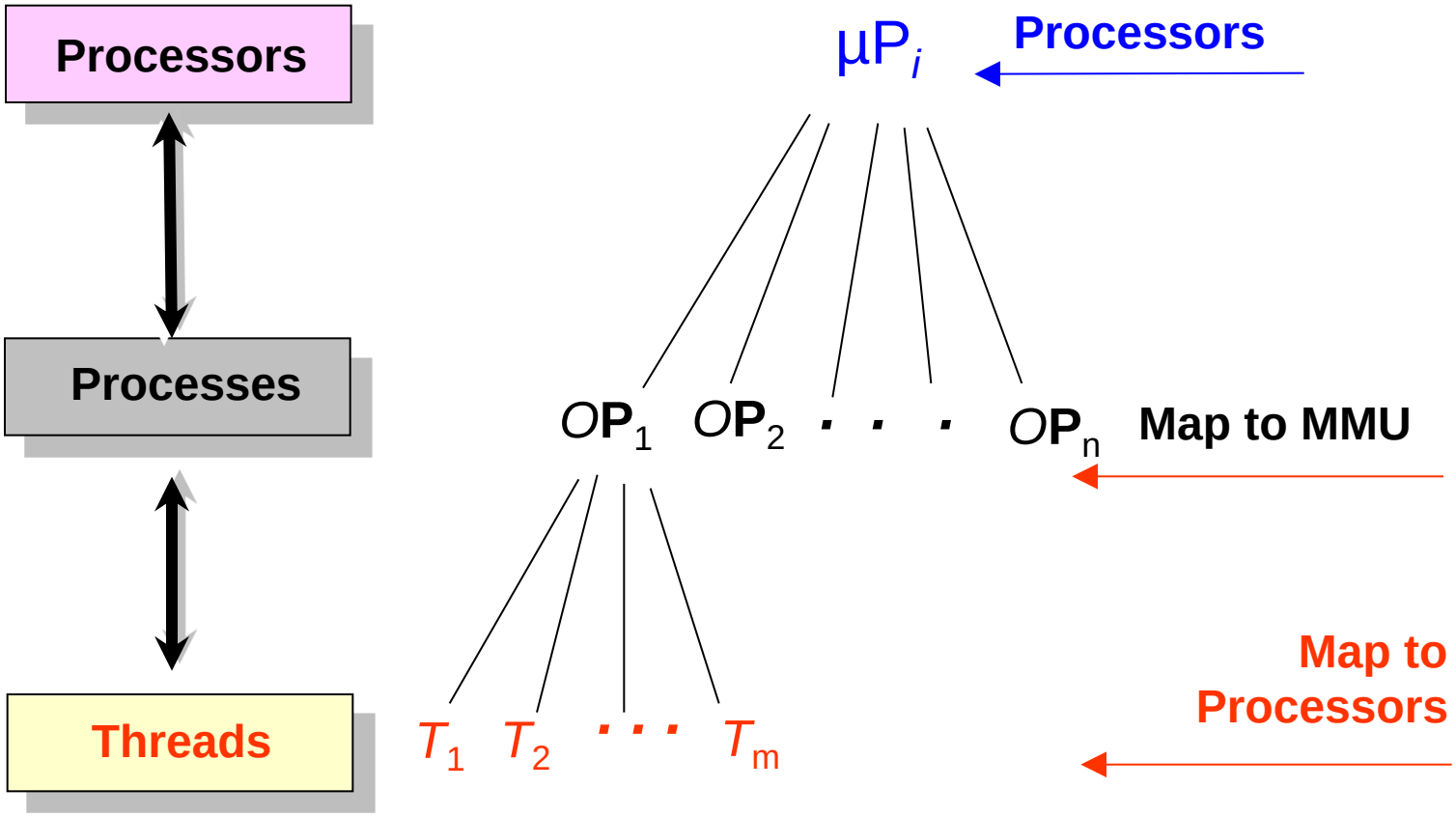
- Time taken to process *n* threads on a single processor is significantly more than a single processor system with HT technology enabled.



Source : http://www.intel.com ;
Reference : [6], [29], [31]

P/C : Microprocessor and cache;
SM : Shared memory

Relationship among Processors, Processes, &Threads



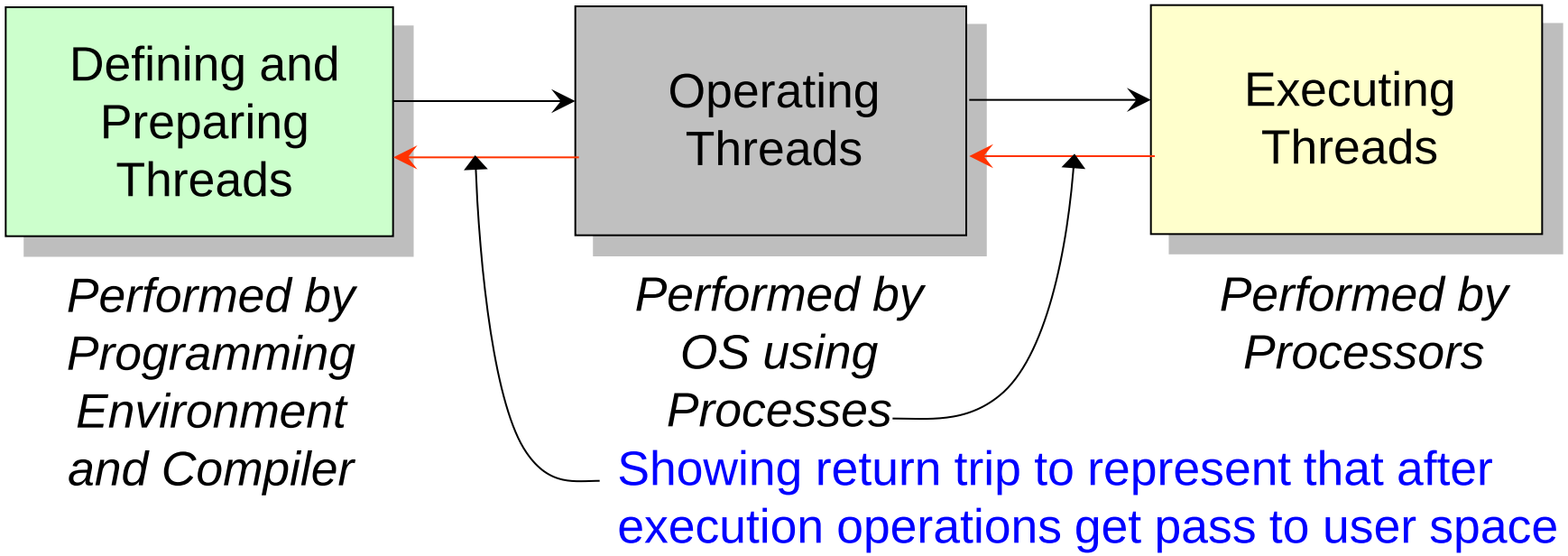
μP_i : Processor OP_1 : Process T_1 : Thread **MMU** : Main Memory Unit

Source : Reference [4],[6], [7]

System View of Threads

Threads Above the Operating System

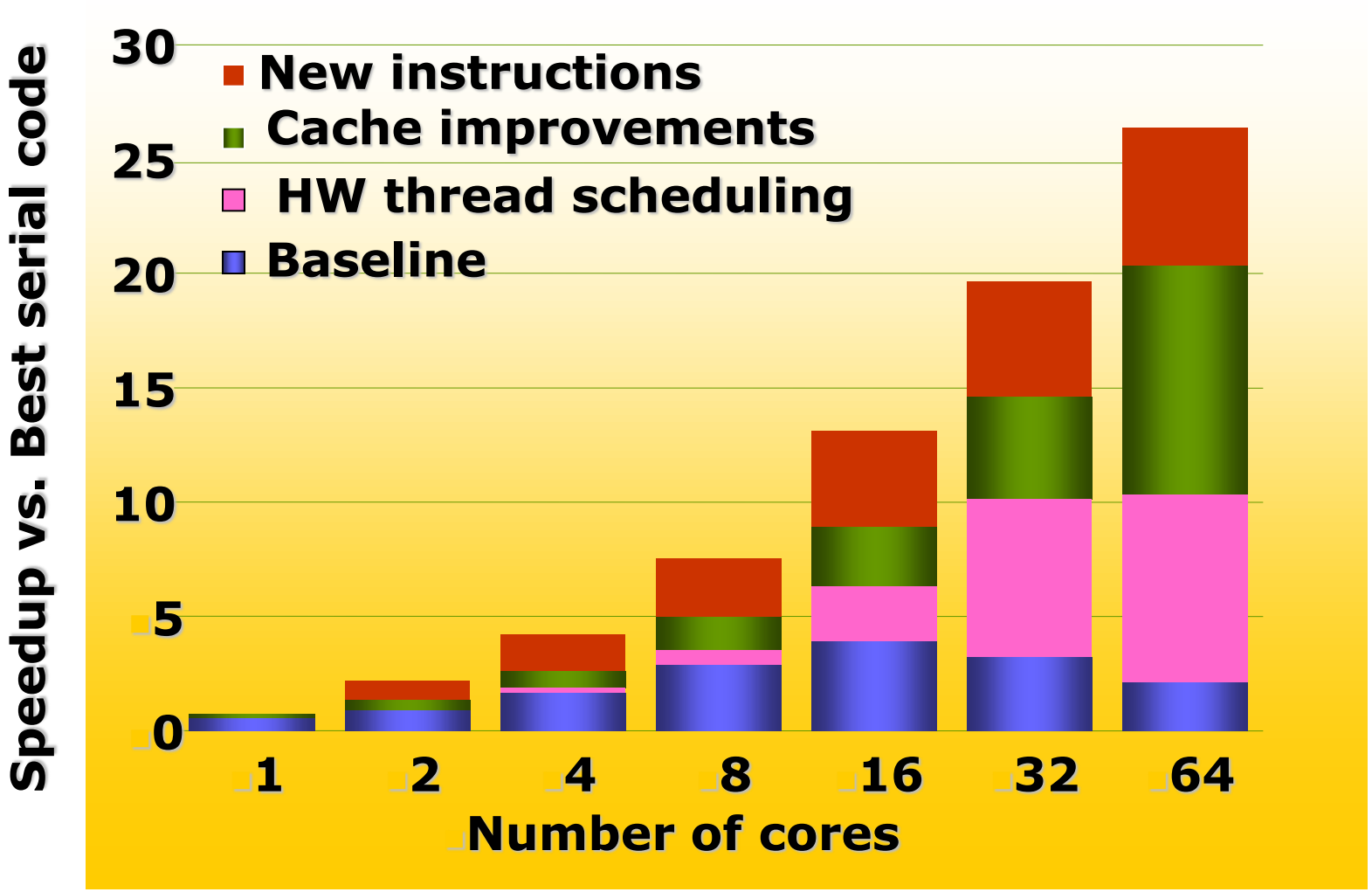
- Understand the problems - Face using the threads – Runtime Environment



Flow of Threads in an Execution Environment

Source : Reference [4],[6], [7]

Architecture-Algorithm Co-Design



Source : http://www.intel.com

Intel Xeon-Host : system configuration

System 1 : Intel Sandy Bridge Server

- Intel Software Development Platform (Intel SDP) MAK F1 Family.
- **Platform** **:** Intel (r) Many Integrated Core Architecture
- **Platform Code Name** : Knights Ferry
- **CPU Chipset Codename** **:** Westmere EP/ Tylersburg UP
- **Board Codename** **:** Sandy Core.
- **CPU** **:** Intel Xeon X5680 Westmere 3.33GHz 12MB L3 Cache LGA 1366 130W Six-Core Server Processor BX80614X5680

Source : www.cdac.in/ Intel

Intel Xeon-Host : system configuration

System 2 : Super Micro SYS-7047GR-TPRF Server

- **Chipset :** Intel C602 Chipset,
- **Mother board :** Super X9DRG-QF,
- CPU : Intel Xeon processor E5-2643 (quad core) (up to 150W TDP), Support for Xeon Phi - 5110P.
- **Memory :** 32 GB DDR3 ECC Registered memory(1600 MHz ECC supported DDR3 SDRAM 72-bit, 240-pin gold-plated DIMMs),
- **Expansion slot :** with 4x PCI-E 3.0 x16 (double-width), 2x PCI-E x8),
- IPMI : Support for IPMI (Support for Intelligent Platform Management Interface v.2.0, IPMI 2.0 with virtual media over LAN and KVM-over-LAN support),
- **Power :** 1620W high-efficiency redundant power supply w/PMBus.
- **Storage :** SATA 3.0 6Gbps with RAID 0,1 support ,1 TB SATA Hard Disk,
- **Network :** Intel i350 Dual Port Gigabit Ethernet withsupport of Supports 10BASE-T, 100BASE-TX, and1000BASE-T, RJ45 output and 1x Realtek RTL8201N\PHY (dedicated IPMI port)

Intel Xeon-Host : Benchmarks Performance

Systems 3 : Host : Xeon (Memory Bandwidth (BW) - Xeon: 8 bytes/channel * 4 channels * 2 sockets * 1.6 GHz = 102.4 GB/s)

PARAM YUVA-II Intel Xeon- Node

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

Source : www.cdac.in/ Intel

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

PARAM YUVA 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

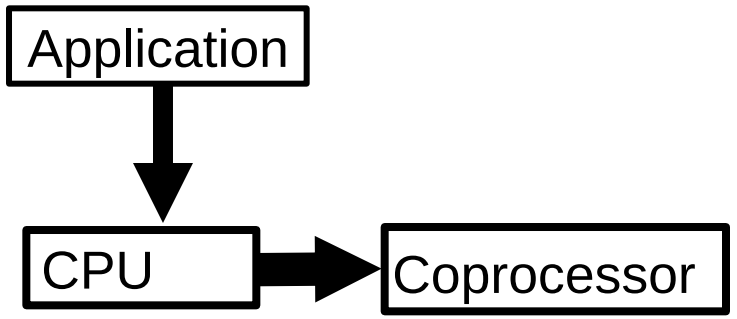
Source : http://www.intel.com; Intel Xeon-Phi books, conferences, Web sites, Xeon-Phi Technical Reports http://www.cdac.in/

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

Computing on Coprocessors : Think in Parallel

Performance = parallel hardware + scalable parallel prog.

- Computing drives new applications
  - Reducing "Time to Discovery"
  - 100 x Speedup changes science & research methods

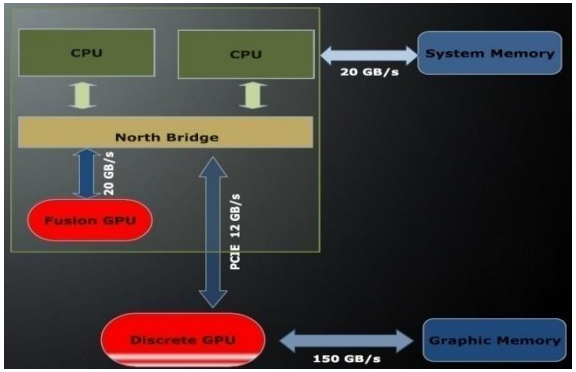


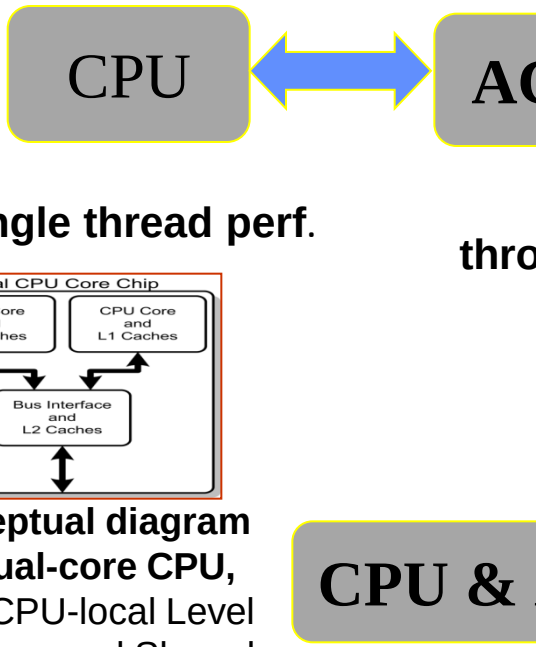
- New applications drive the future of Co-processors & GPUs
  - Drives new GPU /Coprocessor capabilities
  - Drives hunger for more performance

Source : NVIDIA, AMD,References

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.

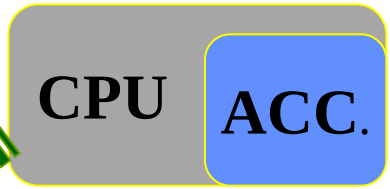




Single thread perf.

throughput

Conceptual diagram of a dual-core CPU, with CPU-local Level 1 caches, and Shared, on-chip Level 2 caches



Physical integration

Architectural integration

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

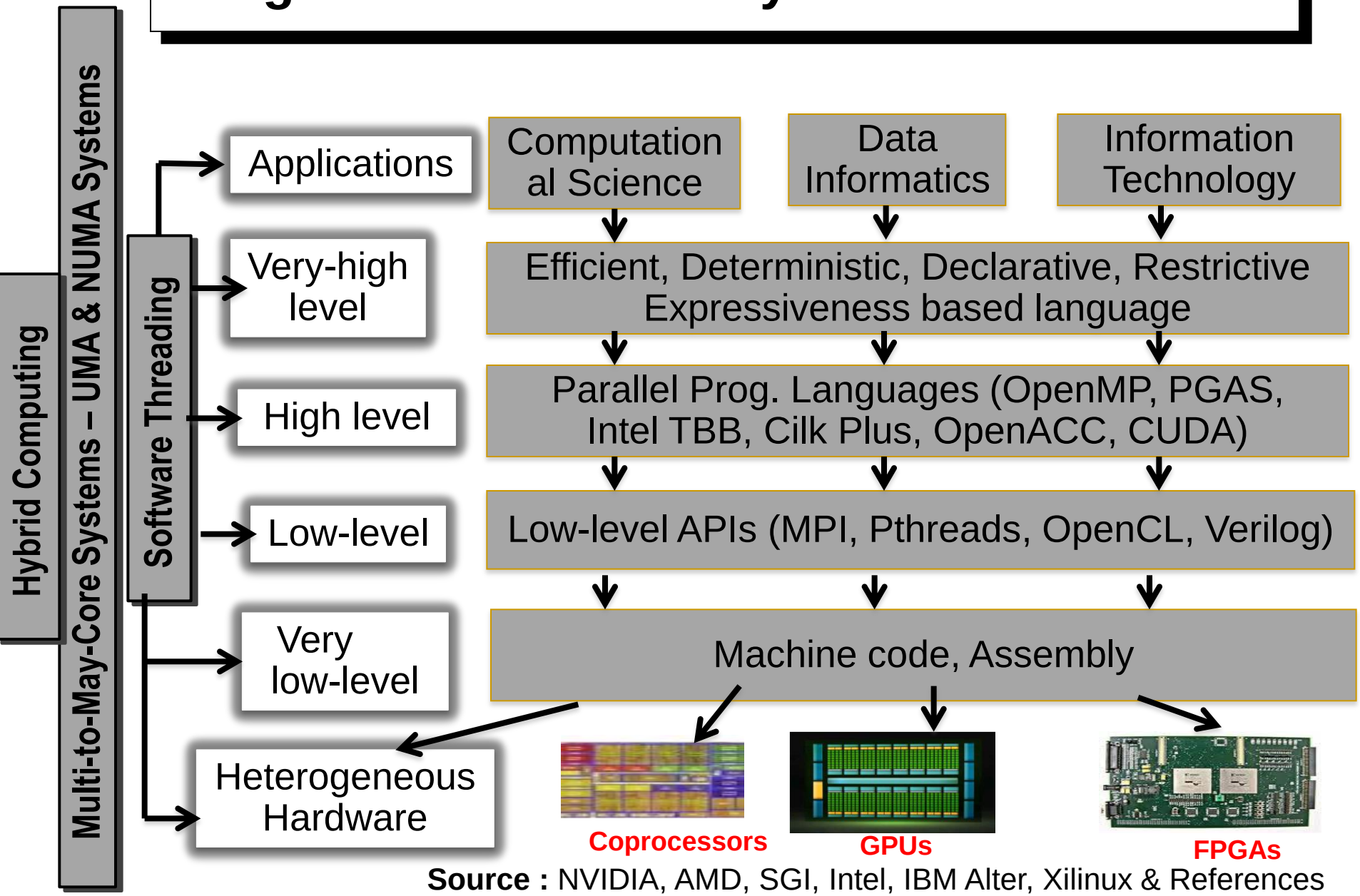
Source : NVIDIA, AMD, SGI, Intel, IBM Alter, Xilinux References

Multi-Core Systems with Accelerator Types

- **FPGA**
  - Xilinx, Alter
- **GPU**
  - Nvidia (Kepler),
  - AMD Trinity APU
- **MIC (Intel Xeon-Phi)**
  - Intel Xeon-Phi (MIC)

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

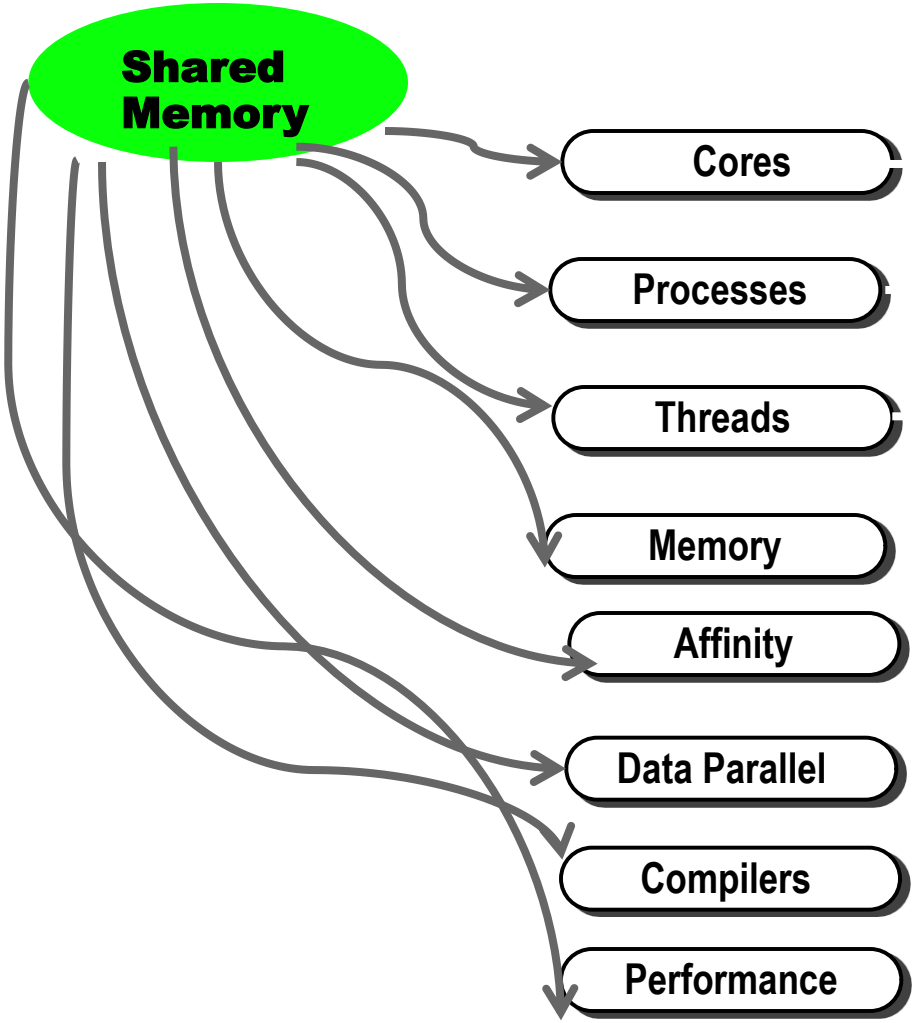
Prog.API - Multi-Core Systems with Devices



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

Prog.API - Multi-Core Systems with Devices

Typical UMA /NUMA Computing Systems



System updates & Performance

Improvements
Quantify impacts prior to implementation

Small prototype available
What will be the performance of system ?

System available for measurement
What will be performance for App

Resources Availability
Application Scaling – Resources Available

High Level APIs : HParallel Prog. Languages (OpenMP, PGAS, Intel TBB, Cilk Plus, OpenACC, CUDA)

Low-level APIs : MPI, Pthreads, OpenCL, FPGA-Verilog:

Prog.API - Multi-Core Systems with Devices

- DO Parallel
- DO Synchronize
- Get Maximum of all values
- Give to Compiler

- DO TRANSFER from Host to Device
- Perform C omp. On Device

- Use as Linux OS
- DO TRANSFER from Host to Device

Parallel Prog. Languages (OpenMP, PGAS, Intel TBB, Cilk Plus, OpenACC, CUDA)

Programming with High Level APIs

Host : Multi-Core Systems OR ARM Multi-core Systems

CPU- Multi-Core Sys With

FPGA (Xilinx, Alter)

GPU - Nvidia (Kepler), AMD Trinity APU

MIC (Intel Xeon-Phi)

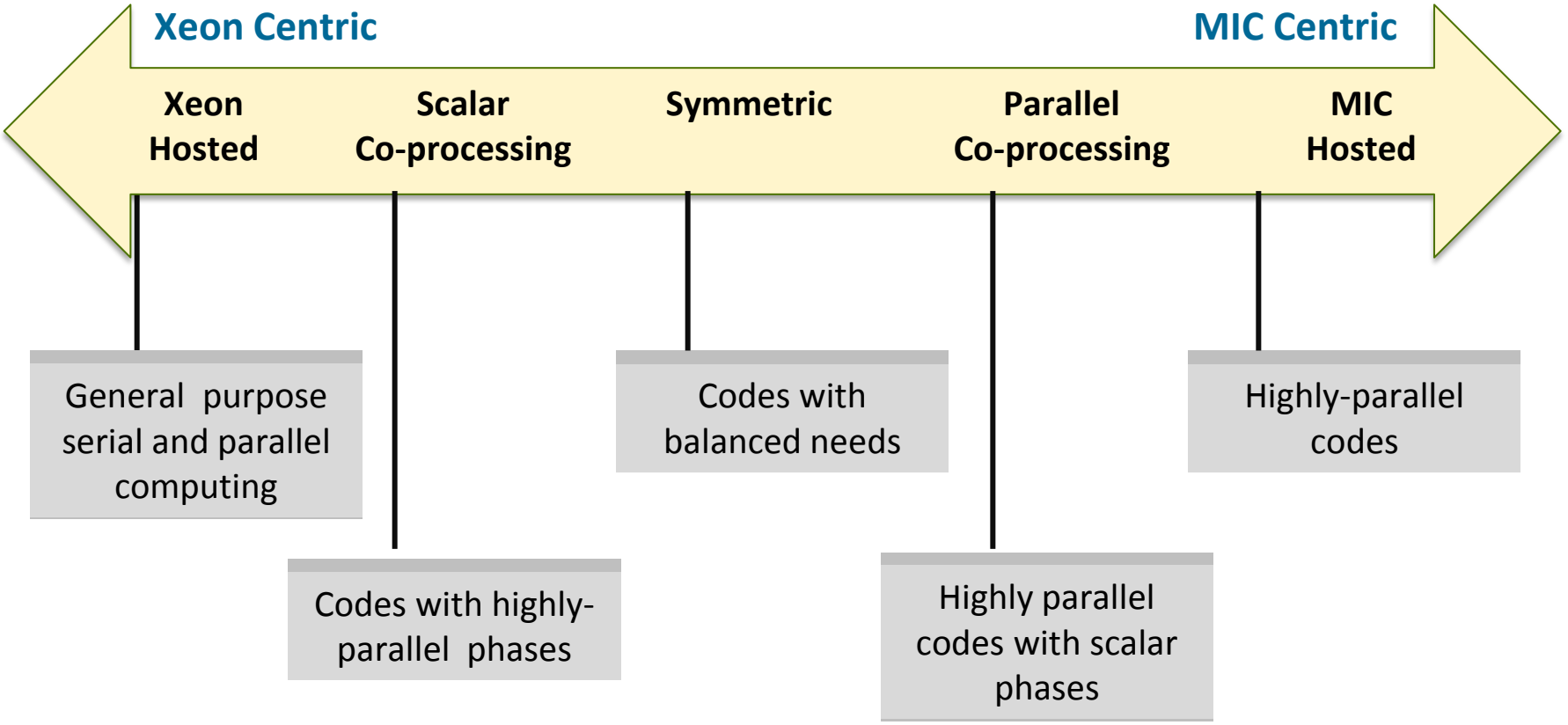
Source : NVIDIA, AMD, SGI, Intel, IBM Alter, Xilinux References

Intel Xeon Host : An Overview of Xeon - Multi-Core and Systems with Devices

Part-I

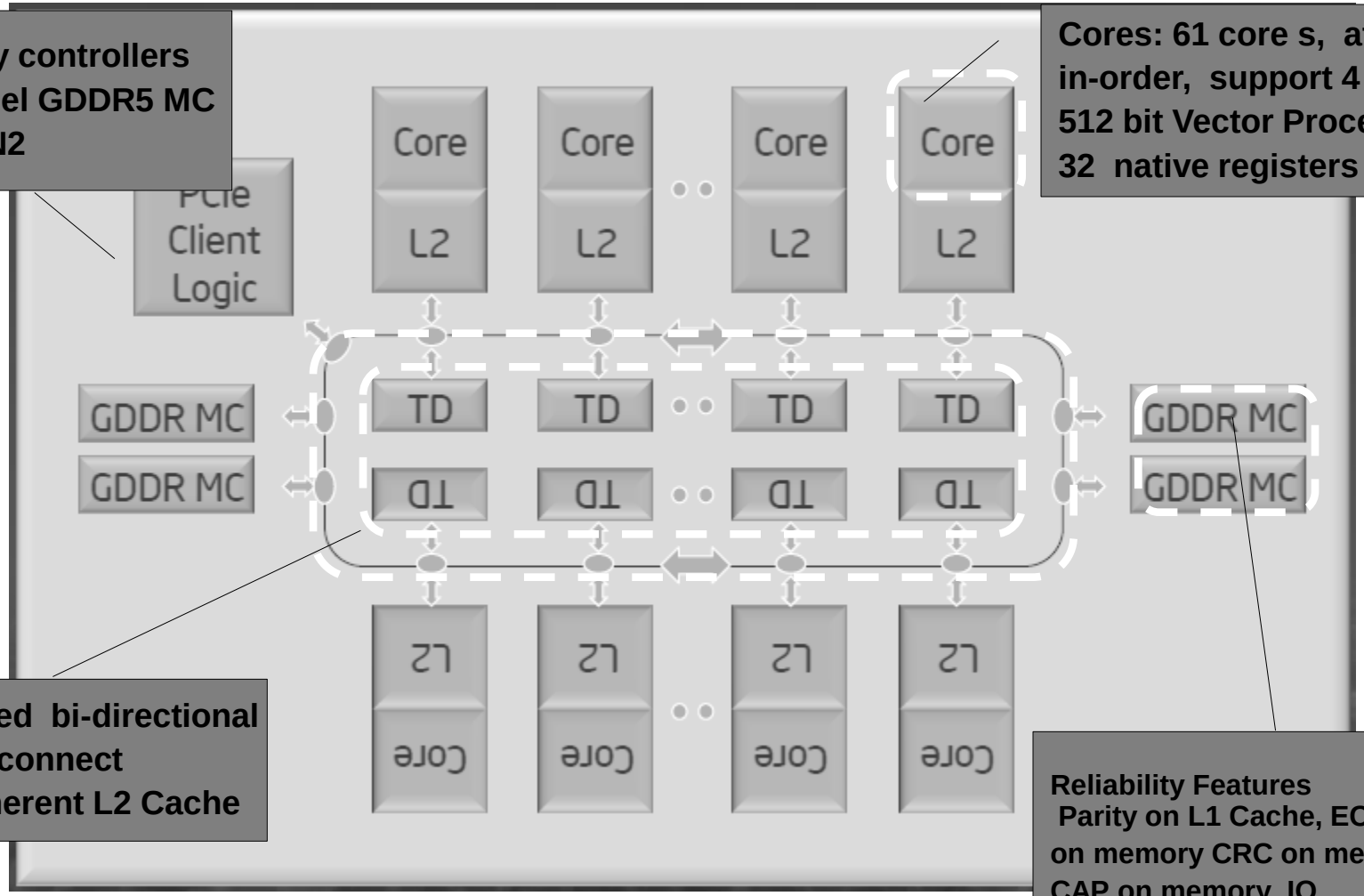
Xeon Phi Architecture & system Software

Prog.- Multi-Core Systems with Coprocessors



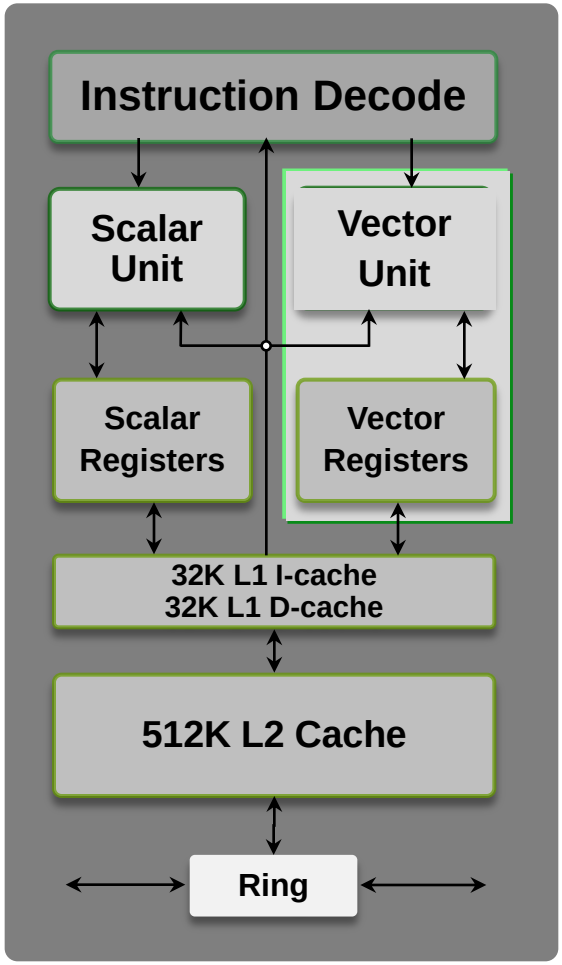
Source : References & Intel Xeon-Phi; http://www.intel.com/

Intel® Xeon Phi™ Architecture Overview



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

Intel Xeon Phi 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/

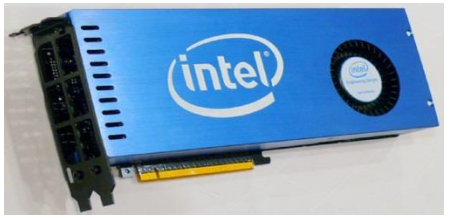
MIC (Xeon Phi) Architecture

- MIC : Many Integrated Core
- Knight Corner co-processor
- Intel Xeon Phi co-processor
  - 22 nm technology
  - > 50 Intel Architecture cores
  - connected by a high performance on-die bi-directional interconnect.
  - I/O Bus: PCIe
  - Memory Type: GDDR5 and >2x bandwidth of KNF
  - Memory size: 8 GB GDDR5 memory technology
  - Peak performance: >1 TFLOP (DP)
  - Single Linux image per chip

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

(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

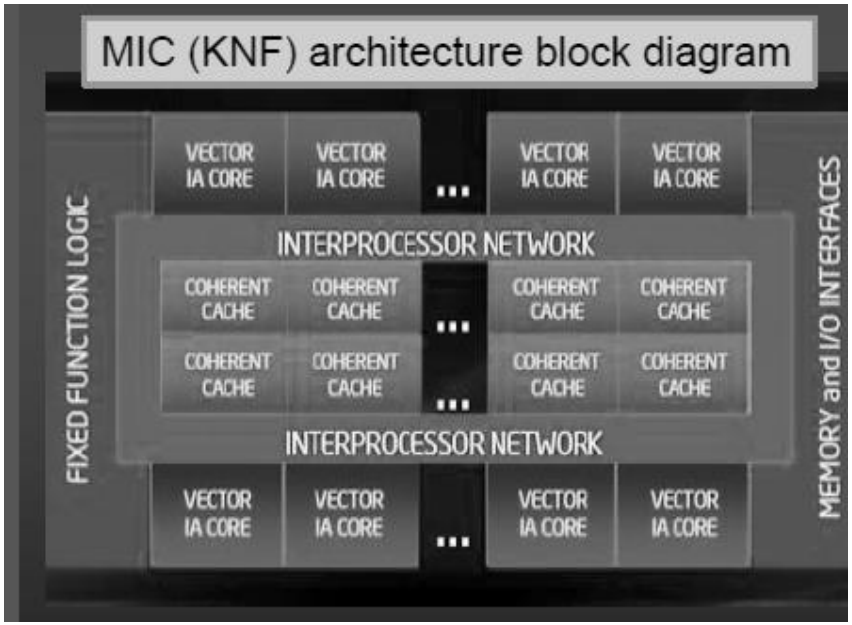


Source : References & Intel Xeon-Phi; http://www.intel.com/

- 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 Many Integrated Core Architecture

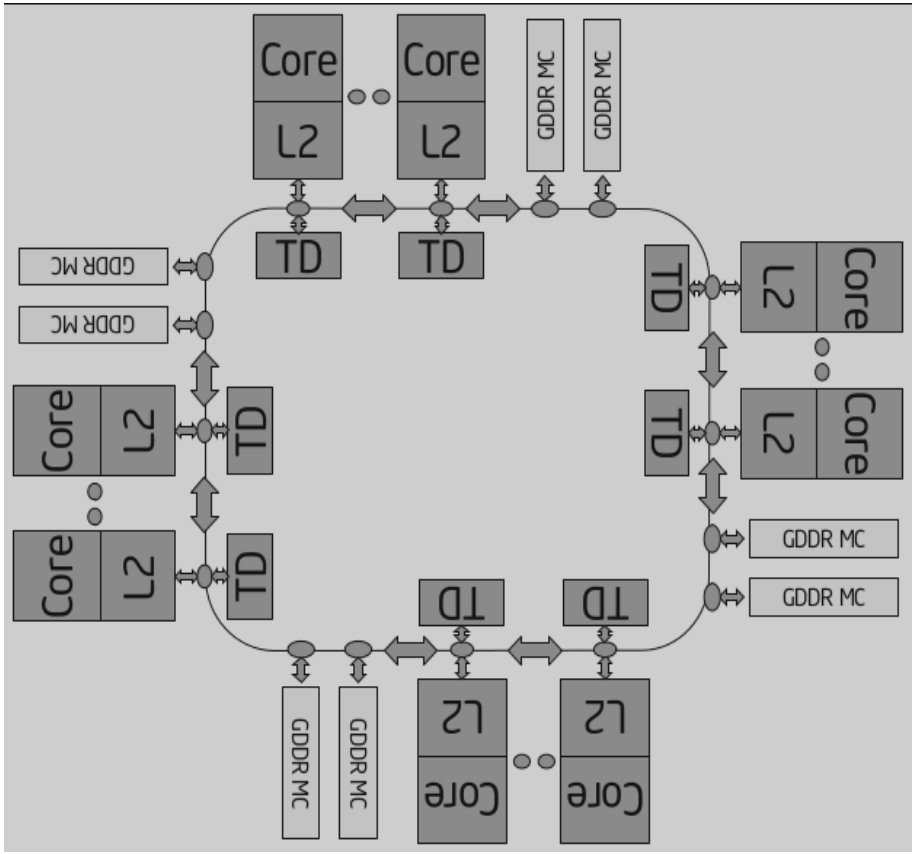
- **MIC Architecture**
  - Mnay cores on the die
  - L1 and L2 cache
  - Bidirectional ring network
  - Memory and PCIe connection



- **Knights Ferry-SW Dev Platform**
  - Up to 32 cores
  - 1.2 GB of GDDRs RAM
  - 512-bit wide SIMD registers
  - L1/L2 cahces
  - Multiple threads (up to 4) per core
  - Slow operation in double precision
- **Xeon PHI (Was Knights Corener)**
  - Firsr product
  - Used in Stampede
  - 50+cores
  - Increased amount of RAM
  - Details are under NDA
  - 22nm technology

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

MIC Intel Xeon Phi Ring

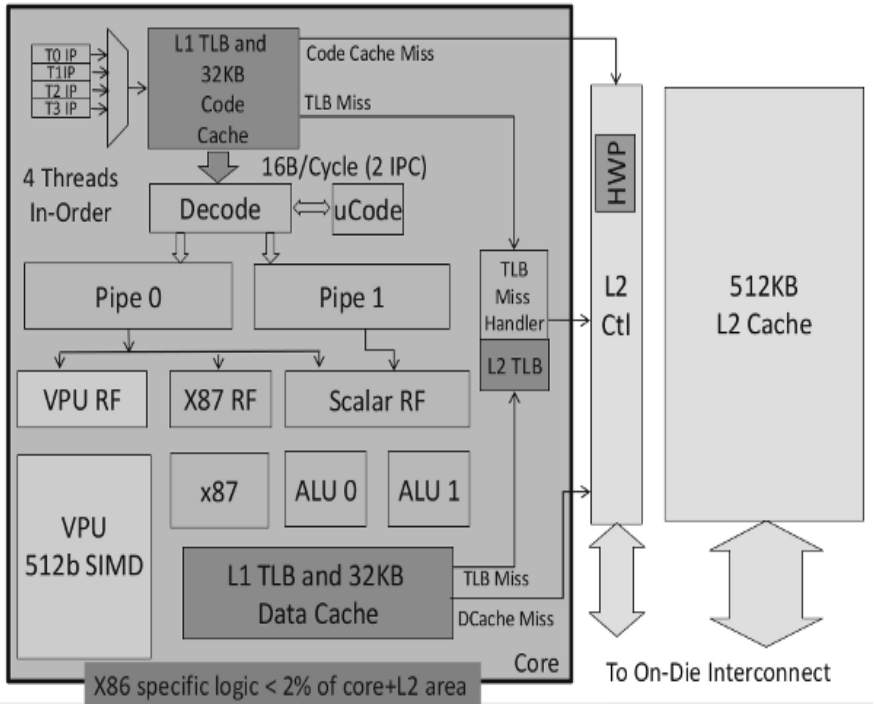


- Each microprocessor core is a fully functional, in-order core capable of running IA instructions independently of the other cores.
- Hardware multi-threaded cores
- Each core can concurrently run instructions from four processes or threads.
- The Ring Interconnect connecting all the components together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

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

The Processor Core

- Fetches and decodes instructions from four hardware thread execution contexts
- Executes the x86 ISA, and Knights Corner vector instructions
- The core can execute 2 instructions per clock cycle, one per pipe - 32KB, 8-Way set associative L1 Icache & Dcache
- Core Ring Interface (CRI)
- L2 Cache



- Memory controllers (which access external memory devices to read and write data)
- PCI Express client: is the system interface to the host CPU or PCI Express switch,

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

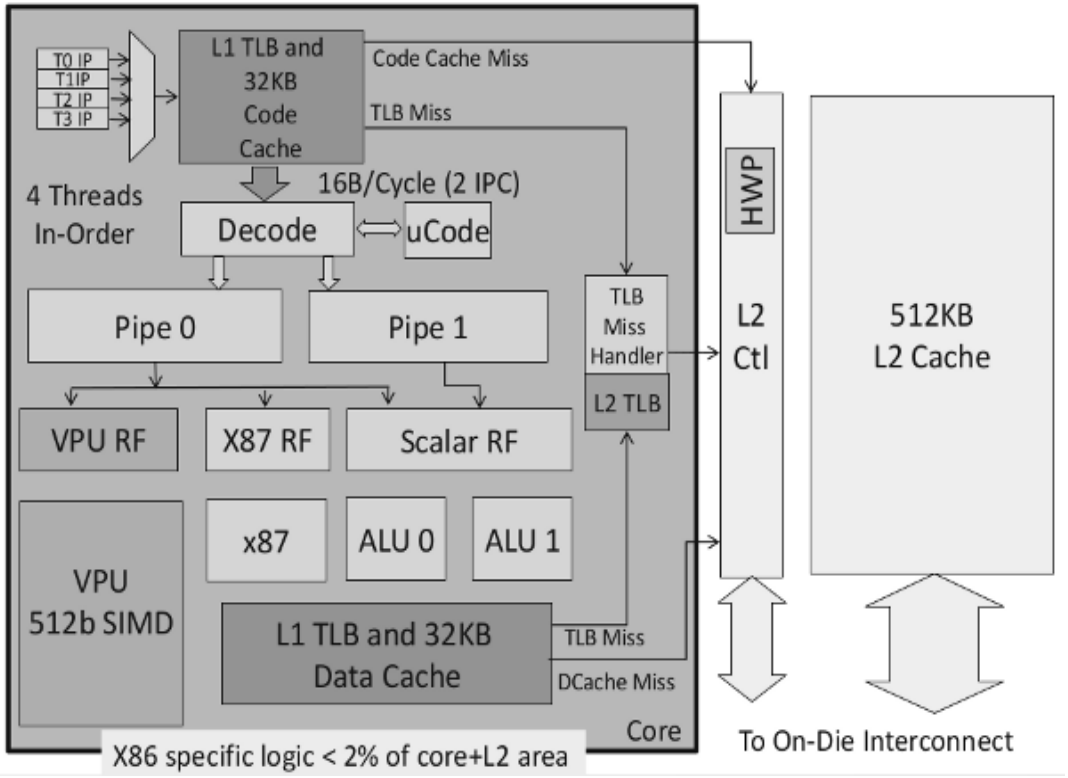
Intel Xeon Phi :Coprocessor - Cache Overview

The L2 Cache

- Each core has a 512 KB L2 cache
- The L2 cache is part of the Core-Ring Interface block
- The L2 cache is private to the core: each core acts as a stand-alone core with 512 KB of total L2 cache space
- Other cores can not directly use them as a cache
- 512 KB x > 50 cores → > 25 MB L2 on Knight Corner
- Tag Directory (**TD**) on each core, not private to the core
- A simplified way to view the many cores in Knights Corner is as a chip-level symmetric multiprocessor (SMP) and > 50 such cores share a high-speed interconnect on-die.

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

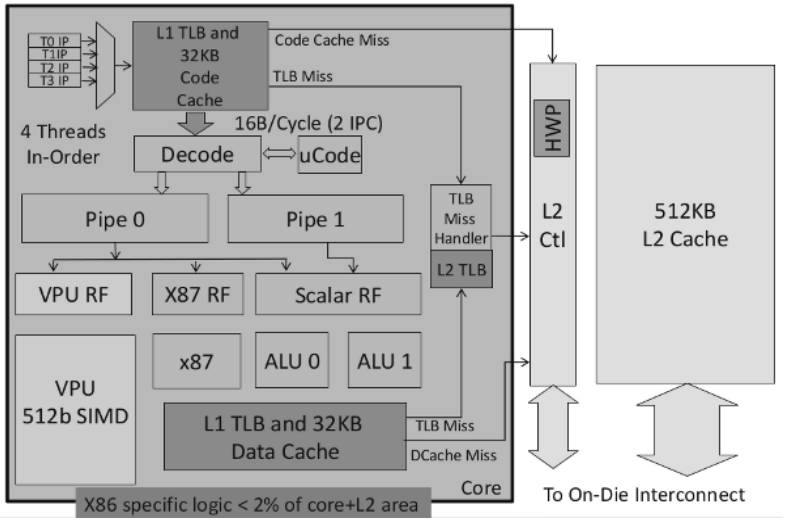
The vector processing unit

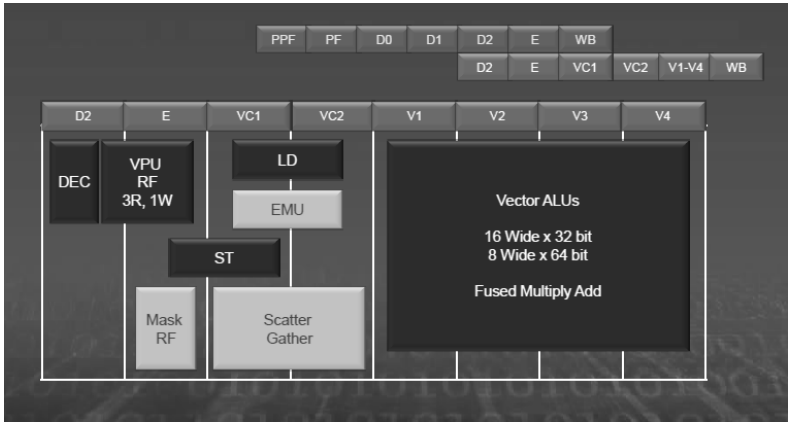


- Vector processing unit (VPU) associated with each core.
- This is primarily a sixteen-element wide SIMD engine, operating on 512-bit vector registers.
- Gather / Scatter Unit
- Vector Mask

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

Xeon Phi : The Vector Processing Unit





- Vector processing unit (VPU) associated with each core.
- This is primarily a sixteen-element wide SIMD engine, operating on 512-bit vector registers.
- Gather / Scatter Unit
- Vector Mask
- Fetches and decodes instructions fr four hardware thread execution contexts
- Executes the x86 ISA, and Knights Corner vector instructions
- The core can execute 2 instructions per clock cycle, one per pipe - 32KB, 8-Way set associative L1 Icache & Dcache
- Core Ring Interface (CRI)
- L2 Cache

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

Intel Xeon Phi - Software

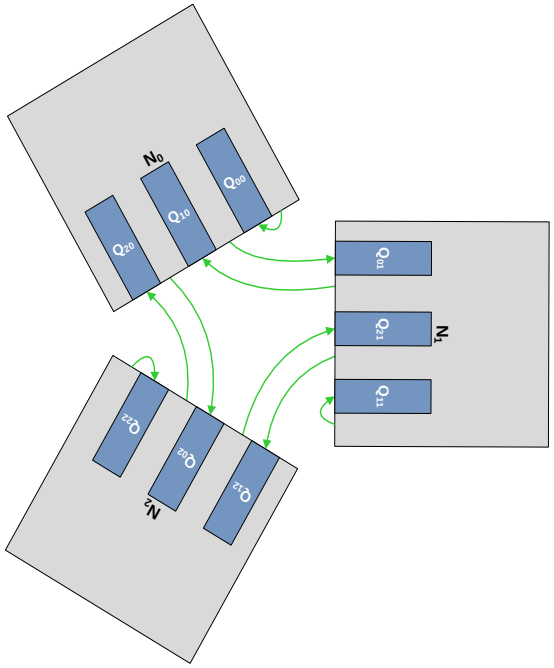
- The System SW Stack
  - Card OS
  - Symmetric Communications Interface (SCIF)
- Compiler Runtimes
  - Coprocessor Offload Infrastructure (COI)
- Coprocessor Communication Link (CCL)
  - IB-SCIF
  - MPI Dual-DAPL

Intel Xeon Phi : SCIF Introduction

- Primary goal: Simple, efficient communications interface between "nodes"
  - Symmetric across Xeon host and Xeon Phi™ Coprocessor cards
  - User mode (ring 3) and kernel mode (ring 0) APIs
    - Each has several mode specific functions
    - Otherwise virtually identical
  - Expose/leverage architectural capabilities to map host/card mapped memory and DMA engines
- Support a range of programming models
- Identical APIs on Linux and Windows

Intel Xeon Phi : SCIF Introduction(2)

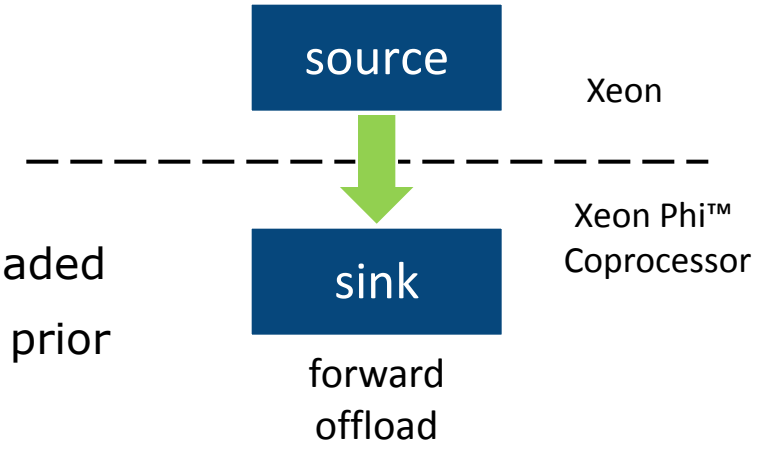
- Fully connected network of SCIF nodes
  - Each SCIF node communicates directly with each other node through the PCIe root complex
- Nodes are physical endpoints in the network
  - Xeon host and Xeon Phi™ Coprocessor cards are SCIF nodes
- SCIF communication is *intra-platform*
- Key concepts:
  - SCIF drivers communicate through dedicated queue pairs
  - one “ring0 QP” for each pair of nodes
  - A receive queue (Qij) in each node is directly written to from the other node.
  - Interrupt driven, relatively low latency



Intel Xeon Phi : COI Terminology

Coprocessor Offload Infrastructure (COI)

- COI allows commands to be sent from a "source" to a "sink"
  - Commands are asynchronous function invocations ("run functions")
  - "Source" is where "run functions" are *initiated*
  - "Sink" is where "run functions" are *executed*
- A typical COI application is comprised of a source application and a sink offload *binary*
- The sink binary is a complete executable
  - Not just a shared library
  - Starts executing from main when it is loaded
- COI automatically loads dependent libraries prior to starting the offload binary on the sink
- COI has a *coi_daemon* that spawns sink processes and waits for them to exit



Intel Xeon Phi : COI APIs

- COI exposes four major abstractions:
  - Use the simplest layer or add additional capabilities with more layers as needed
  - Each layer intended to interoperate with other available lower layers (e.g. SCIF)
- Enumeration: COIEngine, COISysInfo
  - Enumerate HW info; cards, APIC, cores, threads, caches, dynamic utilization
- Process Management: COIProcess (requires COIEngine)
  - Create remote processes; loads code and libraries, start/stop
- Execution Flow: COIPipeline (requires COIProcess)
  - COIPipelines are the RPC-like mechanism for flow control and remote execution
  - Can pass up to 32K of data with local pointers
- Data and Dependency Management: COIBuffer, COIEvent (requires COIPipeline)
  - COIBuffers are the basic unit of data movement and dependence managment
  - COIEvent optionally used to help manage dependences
  - COIBuffers and COIEvents are typically used with Run Functions executing on COIPipelines

Intel Xeon Phi : Coprocessor Communication Link (CCL)

An Overview

- OFED is the industry standard code used for messaging on high-end HPC clusters
  - Supports Intel MPI and all open source MPIs
  - Is in Linux and all the various Linux distributions
- RDMA over SCIF (IB-SCIF) – RDMA within the platform between the host and KNC or multiple KNCs
- Intel ® Xeon Phi ™ Coprocessor Communication Link (CCL) Direct
  - Direct access to InfiniBand HCA from Intel® Xeon Phi ™
  - Lowest latency data path
- Intel ® Xeon Phi ™ Coprocessor Communication Link (CCL) Proxy
  - Pipeline data through host memory to InfiniBand network
  - Higher bandwidth data path for some platform configurations
- Intel MPI dual-DAPL support
  - Uses best data path, direct path for small messages, and proxy path for large messages for best overall MPI performance

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

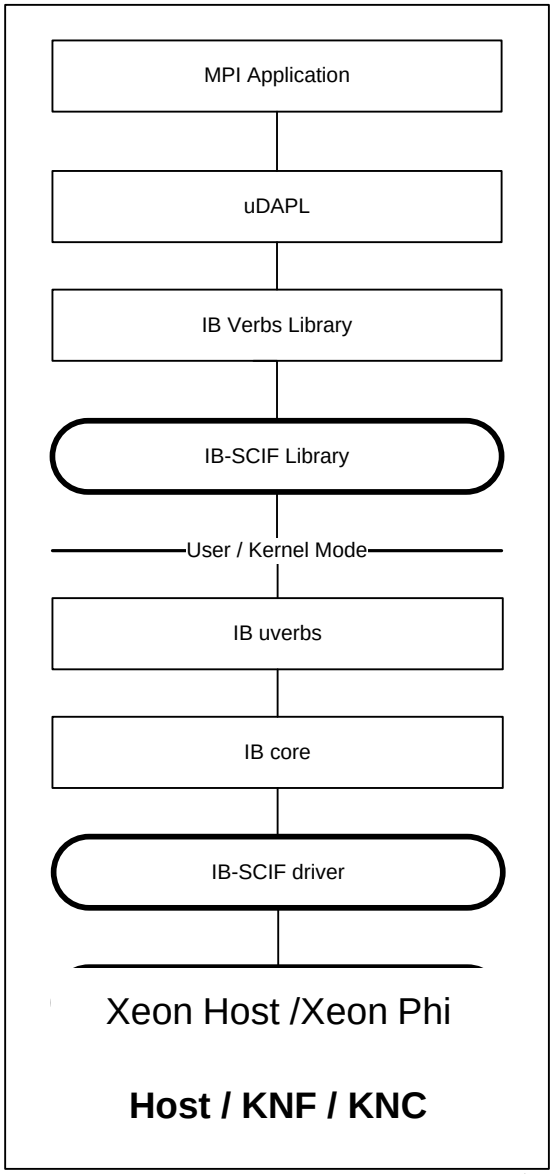
Intel® Xeon Phi Coprocessor : CCL Direct Software

- **CCL-Direct**
  - Allows access to an HCA directly from the Xeon Phi™ Coprocessor using standard OFED interfaces using PCI-E peer-to-peer transactions
  - Provides the lowest latency data path
  - For each hardware HCA, a unique vendor driver has to be developed.
    - e.g., mlx4, mthca, Intel® True Scale ™ hca etc
    - Currently support for Mellanox HCAs (mlx4) exists and is shipping in MPSS
    - Support for Intel® TrueScale™ InfiniBand NICs via PSM is under development, expected release in early 2013
- **Implementation Limitations**
  - Intel® Xeon Phi™ Coprocessor CCL Direct only supports user space clients, e.g. MPI
  - Peak bandwidth is limited on some platforms and configurations
  - CCL-Direct 1 byte latency is in the 2.5us range for Host-KNC, and 3.5-4us range for KNC-KNC across an InfiniBand HCA, peak BW varies depending on the Xeon platform

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

Intel Xeon Phi : RDMA over IB-SCIF

- OFED for Intel® Xeon Phi™ Coprocessor uses the core OFA software modules from the Open Fabrics Alliance
- IB-SCIF is a new hardware specific driver and library that plugs into the OFED core mid-layer
  - SCIF is the lowest level in the SW stack as we saw earlier
  - Provides standard RDMA verbs interfaces within the platform, i.e., between the Intel® Xeon™ and Intel® Xeon Phi ™ Coprocessor cards within the same system.
  - IBSCIF 1 byte latency is in the 13us range, (host-KNC), peak BW is in the 6GB/s per sec. range



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

Intel Xeon Phi Coprocessor – 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 ..)

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

Intel Xeon Phi :Coprocessor Offload Programming

Offload Compiler
(compiler assisted offload)

```
__declspec(target(mic)) int numFloats = 100;

__declspec(target(mic)) float input1[100], input2[100];

__declspec(target(mic)) float output[100];

pragma offload target(mic) \
in(input1, input2, numFloats) out (output) {
    for(int j=0; j<numFloats; j++)  {

          output[j] = input1[j] + input2[j];

         }

}
```

COI Runtime

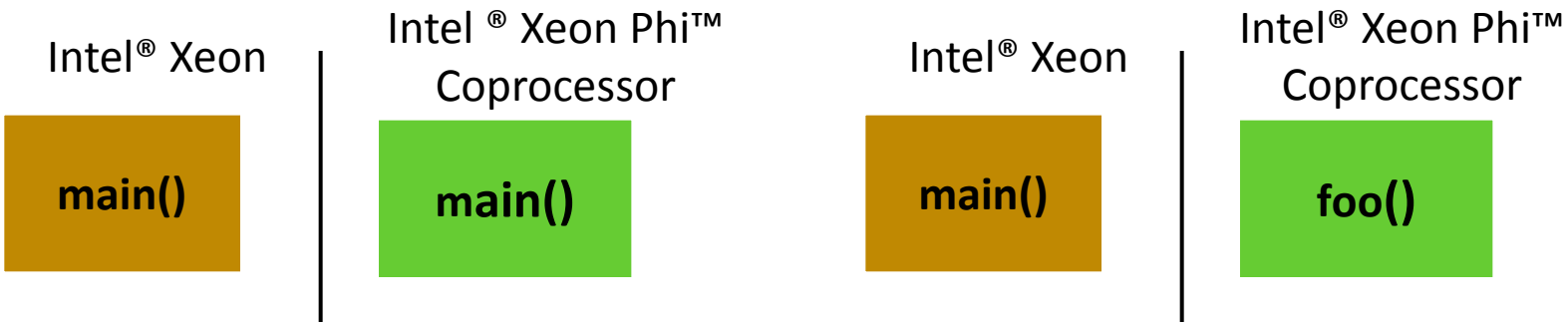
```
COIProcessCreateFromFile( … );

COIBufferCreate( … );

…
COIPipelineRunFunction ( … );
```

SCIF

```
scif_vwriteto( … );
scif_send( … );
scif_recv( … );
scif_vreadfrom( … );
```

Xeon Phi : Programming Environment

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 (…)

Xeon Phi : Programming Environment

Execution Modes

- 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/

Xeon Phi : Data Access Semantics

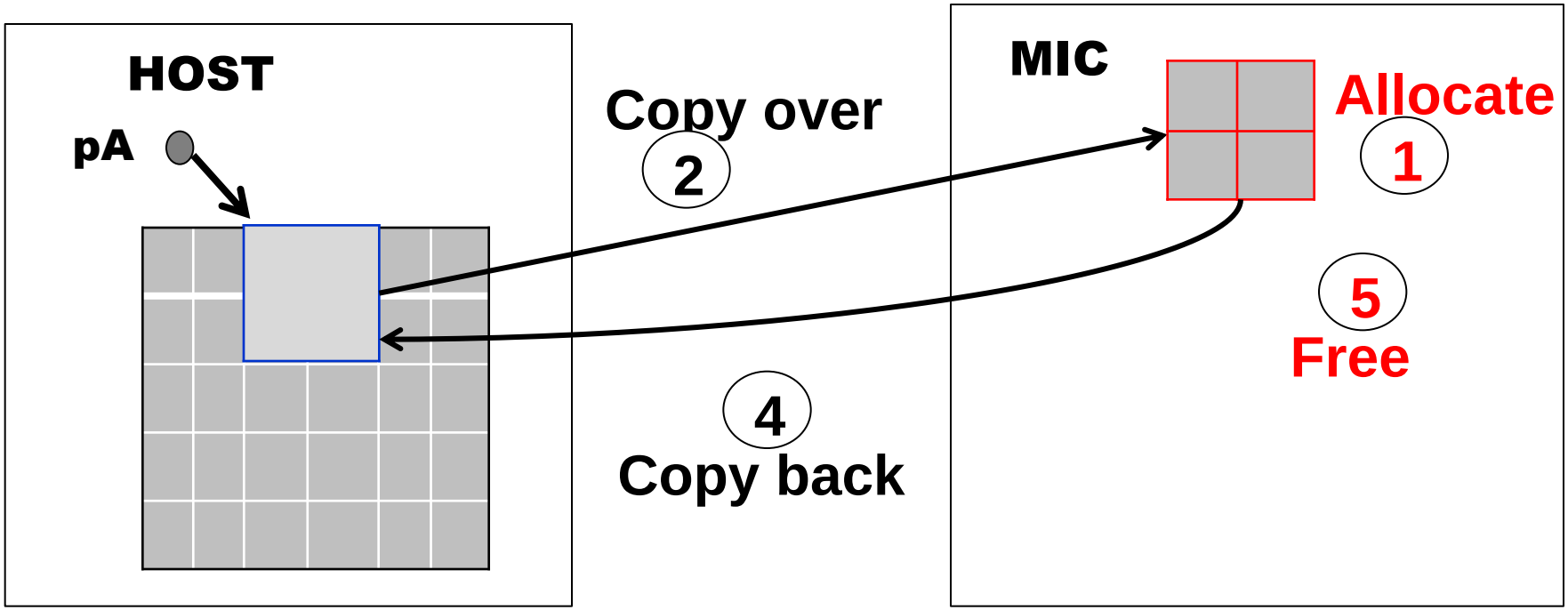
- **Data Access Semantics**
  - Explicit Offloading
  - Implicit Offloading

- Complier Data Transfer Overview
  - The host CPU and the Intel Xeon Phi coprocessor do not share physical or virtual memory in hardware
  - Two offload transfer models are : **Explicit Copy** and **Implicit Copy**

Xeon Phi : Data Access Semantics

- Two offload transfer models are : **Explicit Copy** and **Implicit Copy**
- **Explicit Copy :**
  - Programmer designates variables that need to be copied between **host** and **card** in the *offload directive*
  - **Syntax:** Pragma/directive-based
  - **C/C++ Example:** #pragma offload target(mic) in(data:length(size)) (OpenMP, Pthreads, Intel TBB, MPI with OpenMP/Pthreads/Intel TBB)

Compiler : Offload using Explicit Copies – Data movement



- Default treatment of **`in/out`** variables in a **`#pragma`** offload statement

Compiler : Offload using Explicit Copies – Data movement

- Default treatment of **`in/out`** variables in a **`#pragma`** offload statement
  - At the start of an offload:
    - Space is allocated on the coprocessor
    - **`in`** variables are transferred to the coprocessor
  - At the end of an offload:
    - **`out`** variables are transferred from the coprocessor
    - Space for both types (as well as **`inout`**) is **`deallocated`** on the coprocessor

Compiler : Offload using Explicit Copies

	C/CC+ Syntax	Semantics
Offload pragma	#pragma offload <clauses> <statement block>	Allow next statement block to execute on Intel MIC Arch or host CPU
Keyword for variable & function definitions	_attribute_((target(mic)))	Compile for, or allocate variable on, both CPU and Intel MIC Arch.
Entire Blocks of Code	#pragma offload_attribute(push, target(mic))	Mark entire files or large blocks of code for generation on both host CPU
Data Transfer	#pragma offload_transfer target(mic)	Initiates asynchronous data transfer, or initiates and completes synchronous data
Synchronization	#pragma offload_wait signal(signal_slot)	Wait asynchronous offload processes to complete

Compiler : Offload using Explicit Copies

	Fortran	Semantics
Offload directive	Offload directive !dir$ omp offload <clause> <OpenMP construct>	Execute next OpenMP* parallel construct on Intel® MIC Architecture
	!dir$ offload <clauses> <statement>	Execute next statement (function call) on Intel® MIC Architecture
Keyword for variable/function definitions	!dir$ attributes offload:<MIC> :: <rtn-name>	Compile function or variable for CPU and Intel® MIC Architecture

Data Transfer	#pragma offload_transfer
	target(mic)
		Initiates asynchronous data
		transfer, or initiates and
		completes synchronous data

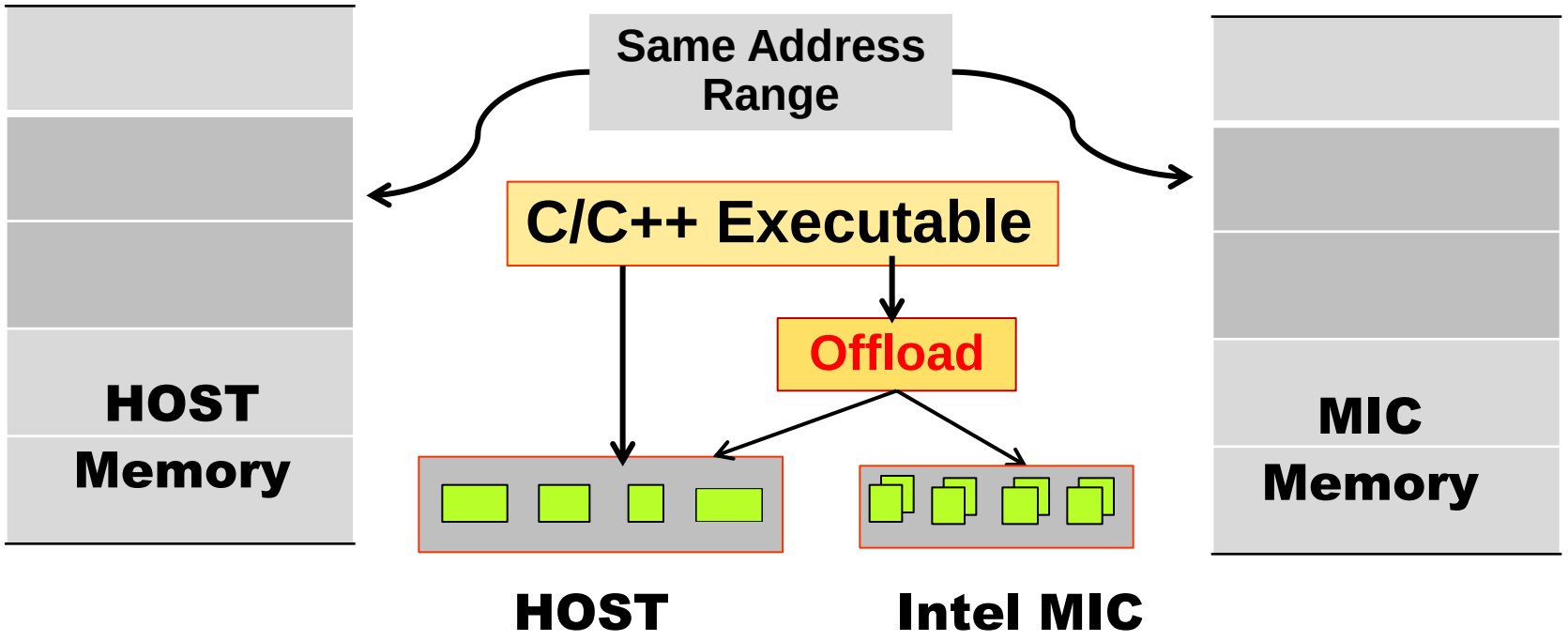
Xeon Phi : Data Access Semantics

- **Data Access Semantics**
  - Implicit Offloading

- Section of memory maintained at the same virtual address on both the host and Intel MIC Architecture coprocessor

- Reserving same address range on both devices allows
  - Seamless sharing of complex pointer-containing data structures
  - Elimination of user marshaling and data management
  - Use of simple language extensions to C/C++

Compiler : Offload using Explicit Copies – Data movement



Heterogeneous Compiler : Offload using Implicit Copies

- When “shared” memory is synchronized
  - Automatically done around offloads (so memory is only synchronized on entry to, or exit from, an offload call)
  - Only modified data is transferred between CPU and coprocessor
- Dynamic memory you wish to share must be allocated with special functions: **`_Offload_shared_malloc`**, **`_Offload_shared_aligned_malloc`**, **`_Offload_shared_free`**, **`_Offload_shared_aligned_free`**
- Allows transfer of C++ objects
  - Pointers are no longer an issue when they point to “shared” data
- Well-known methods can be used to synchronize access to shared data and prevent data races within offloaded code
  - E.g., locks, critical sections, etc.
- This model is integrated with the Intel Cilk Plus Parallel Extensions

  Supported in C /C++ Languages Only

Compiler : Data Transfer Overview Compiler

- Two offload transfer models are : **Explicit Copy** and **Implicit Copy**
- **Implicit Copy :**
  - Programmer makes variables that need to be shared between **host** and **mic** card
  - The same variable can be used in both **host** and **coprocessor** code
  - Runtime automatically maintains coherence at the beginning and end of offload statements
  - **Syntax:** keyword extensions based
  - **Example:**

```
_Cilk_shared double foo;
_Offload func(y);
```

Intel Xeon Phi Coprocessors :

Compilation and Vectorization

Part-2

Vectorization Methodlogy

What is meant by Vectorization ?

Vectorization is the process of converting an algorithm from a **scalar** implementation to a **vector process**.

Scalar : an operation one pair of operands at a time

Vector : A process in which a single instruction can refer to a vector (series of adjacent values)

- it adds a form of parallelism to software in which one instruction or operation is applied to multiple pieces of data.
- Efficient Processing of Data Movement is required to get improvement in performance.

What is meant by Vectorization ?

- Many general-purpose microprocessors support SIMD (single-instruction-multiple-data) parallelism

- When the hardware is coupled with C/ C++ compilers that support it, developers have an easier time delivering more efficient, better performing software

- Types of Vector Computations in Applications
  - Multi-media Applications
  - Scientific and Engineering Applications
  - Graphic Computations
  - Computational Finance
  - Information Science Applications

What is meant by Vectorization ?

Compilers :

- Performance or efficiency benefits from **vectorization** depend on the code structure.
- Automatic & near automatic techniques (**Auto-Vectorization feature**) introduced below are most productive in delivering improved performance or efficiency.
- **SIMD** Support
  - Intel C++ Compilers
  - Intel Fortran 90 Compilers
  - Compliers supporting SIMD Instructions
- Intel Compilers supporting the **Intel Streaming SIMD Extensions** (**Intel SSE**) & Intel Advanced Vector Extensions (**Intel AVX**) on both IA-32 and Intel 64 processors.

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

What is meant by Vectorization ?

Compilers :

- **Auto-vectorization :** Performance or efficiency benefits from **vectorization** depend on the both compilers do auto-vectorization, generating Intel SIMD code to automatically vectorize parts of application software when certain conditions are met.

- **Portability Problems :** Because no source code changes are required to use auto-vectorization, there is no impact on the portability of your application.

- To take advantage of auto-vectorization, applications must be built at default optimization settings (-O2) or higher. No additional or special switch setting is needed using packed SIMD instructions

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

What is meant by Vectorization ?

Advantage of Intel MKL and Intel IPP

- Intel Math Kernel Library (**MKL**)
- Intel® Integrated Performance Primitives (**IPP**) is another library for C and C++ developers,

- Another easy way to take advantage of vectorization is to make calls in your applications to the vectorized forms of functions in the Intel® Math Kernel Library. Much of Intel MKL is threaded and supports auto-vectorization to help you get the most of today's multi-core processors. **Intel MKL** functions are also fully thread-safe, so multiple calls for different threads will not conflict with one another.
- **Intel IPP** offers libraries that can be called for multimedia, data processing, and communications applications

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

About Vectorization

- Whenever possible, instructions on data arrays are processed in an assembly line manner, where several pieces of data are undergoing different parts of an operation simultaneously

Vector Registers

- The vector computers get most of their speed through vector operations. This means that a single type of instruction on multiple data. This is uniquely accomplished through the use of vector registers.

Vector Chaining

- Vector chaining is a way to decrease vector start-up time. On the C90 a functional unit can begin processing data as soon as the first elements are in the registers.

About Vectorization

- **About Vectorization :**
  - High performance is dependent on the vectorization of long loops. Poor performance can result from the inhibition of this vectorization.
- **Types of Computation in Applications**
  - Loop Not Innermost
  - Vector Dependencies
  - Other Not Vectorizable Constructs
  - Memory Conflicts
  - I/O Optimization

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

About Vectorization

Loop No nnermost

Problem

- Only innermost loop can be vectorized at the machine instruction level. However, it may be more efficient to vectorize the operations in the outer loops instead. This could be the case if:
  - The inner loop is inhibited from vectorization
  - The outer loop has a longer vector length than the inner loop
  - The outer loop does more work than the inner loop

Solution

- The solution is to make the outer loop innermost. Depending on the structure of the loops, there are three ways to do this:
  - Swap the loops
  - Split the outer loop
  - Unwind the inner loop

About Vectorization

Vector Dependencies

Problem : Dependencies occur when each iteration of a loop is dependent on the result of previous iterations.

- There are three kinds of dependency:
  (1) Result not ready (recurrence or recursion)
  (2) Value destroyed (3) Ambiguous subscript

Solution :

Result Not Ready

- The solution is to restructure the loop to remove the dependency. Sometimes, this is difficult and requires rethinking the algorithm. Often, however, you can do it by:
  - Swapping loops **OR** Splitting the dependent work out of the loop

Value Destroyed

- You generally do not have to worry about this kind of dependency. The compiler handles it by saving the values in a temporary array.

Ambiguous Subscript

- The solution is to use an IVDEP directive to tell the compiler that there is not dependency (if that is in fact the case!)

About Vectorization

Other Non-vectorizable Constructs

Problem

- There are a number of other constructs that prevent vectorization. These include:
  - I/O statements (These generate calls no library subroutines)
  - CHARACTER data and functions
  - STOP and PAUSE
  - Assigned GOTO (obsolete, anyway)

Solution

- The only solution is to move these constructs out of the loop, either by splitting or by recording so that the constructs are unnecessary

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

About Vectorization

Typical Vector Computer Features

- Fast Clock Speed.
- Segmented, Vector Functional Units
- Independent Functional Units
- Register-to-Register Operations
- Shared, Banked Memory
- No Virtual Memory Fast I/O

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

Vectorization and SIMD Execution

- SIMD
  - Flynn's Taxonomy: Single Instruction, Multiple Data
  - CPU perform the same operation on multiple data elements
- SISD
  - Single Instruction, Single Data
- Vectorization
  - In the context of Intel® Architecture Processors, the process of transforming a scalar operation (SISD), that acts on a single data element to the vector operation that that act on multiple data elements at once(SIMD).
  - Assuming that setup code does not tip the balance, this can result in more compact and efficient generated code
  - For loops in "normal" or "unvectorized" code, each assembly instruction deals with the data from only a single loop iteration

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

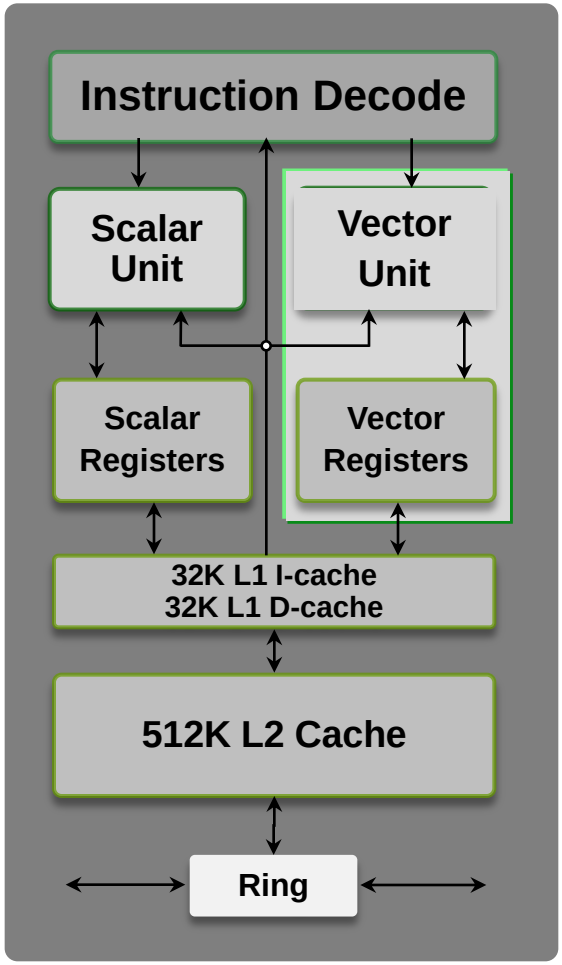
Intel Xeon Phi : Vector Unit

Understand floating point arithmetic Unit

- Vector Processing Unit executing vector FP instruction
- X87 unit also exist can execute FP Instruction as well
- Compiler choose which place to use for FP operation
- VPU is preferred place because of its speed
  - VPU can make the FP results reproducible as well
- Use X87 should be used for two reasons
  - Reproduce the same results 15 years ago, right or wrong
  - Need generate FP exceptions for debugging purpose
- Intel Compiler default to VPU the user can override with
  `-fp-model strict`
- Vectorized, high precision of division, square root and transcendental functions from libsvml
  `-fp-model-precise -no-prec-div -no-prec-sqrt -fast-transcendentals -fimf-precision=high`

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

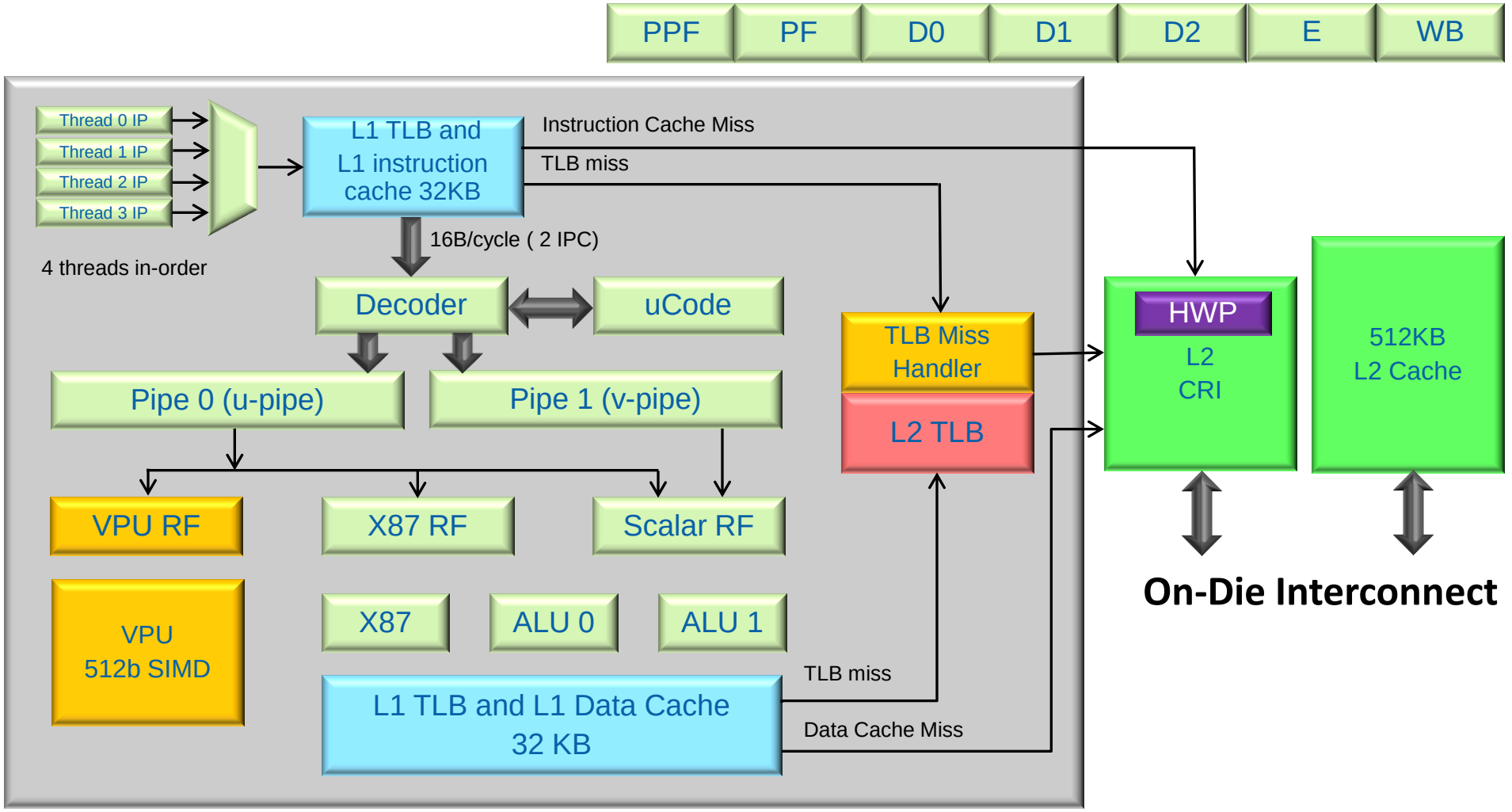
Core Architecture Overview



- 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

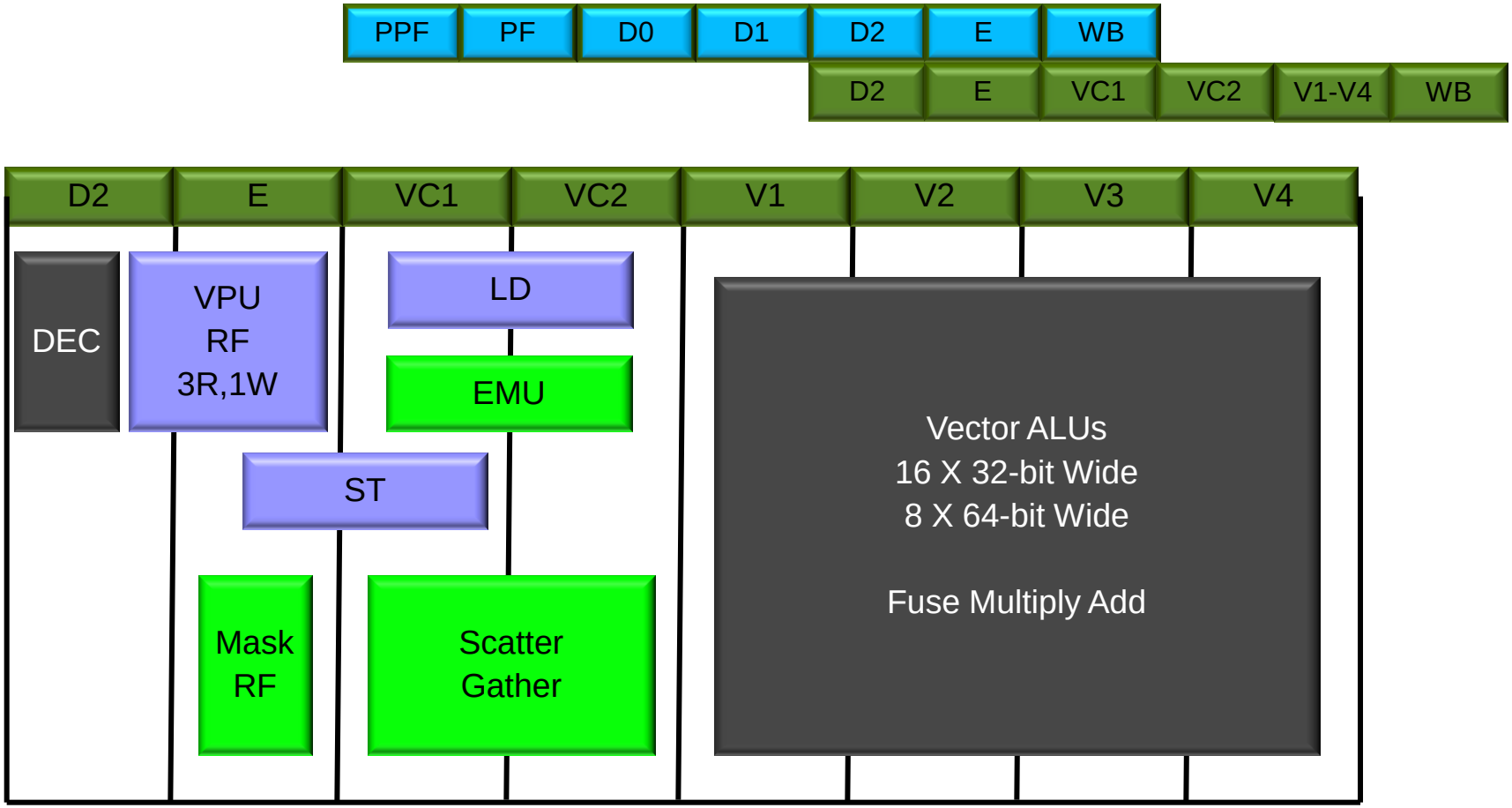
Source : References & Intel Xeon-Phi; http://www.intel.com/

Vector Processing Unit Extends the Scalar IA Core



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

Core extension Vector Processing Unit



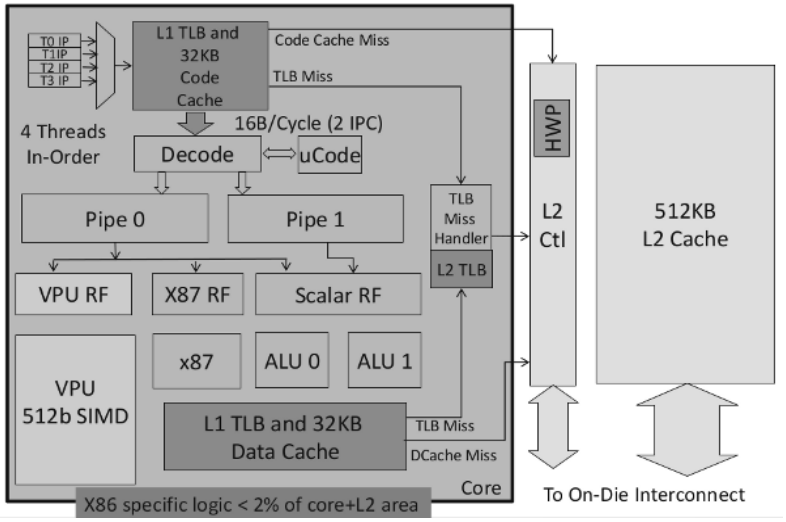
Source : References & Intel Xeon-Phi; http://www.intel.com/

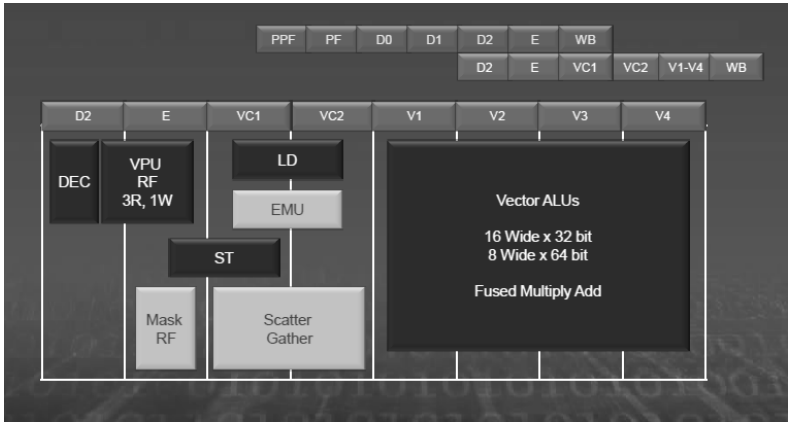
Vector Processing Unit and Intel® IMCI

- Vector Processing Unit Execute Intel® IMCI
  - Intel® Initial Many Core Instructions
- 512-bit Vector Execution Engine
  - 16 lanes of 32-bit single precision and integer operations
  - 8 lanes of 64-bit double precision and integer operations
  - 32 512-bit general purpose vector registers in 4 thread
  - 8 16-bit mask registers in 4 thread for predicated execution
- Read/Write
  - One vector length (512-bits) per cycle from/to Vector Registers
  - One operand can be from the memory free
- IEEE 754 Standard Compliance
  - 4 rounding Model, even, 0, +∞, -∞
  - Hardware support for SP/DP denormal handling
  - Sets status register VXCSR flags but not hardware traps

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

Xeon Phi : The Vector Processing Unit





- Vector processing unit (VPU) associated with each core.
- This is primarily a sixteen-element wide SIMD engine, operating on 512-bit vector registers.
- Gather / Scatter Unit
- Vector Mask
- Fetches and decodes instructions fr four hardware thread execution contexts
- Executes the x86 ISA, and Knights Corner vector instructions
- The core can execute 2 instructions per clock cycle, one per pipe - 32KB, 8-Way set associative L1 Icache & Dcache
- Core Ring Interface (CRI)
- L2 Cache

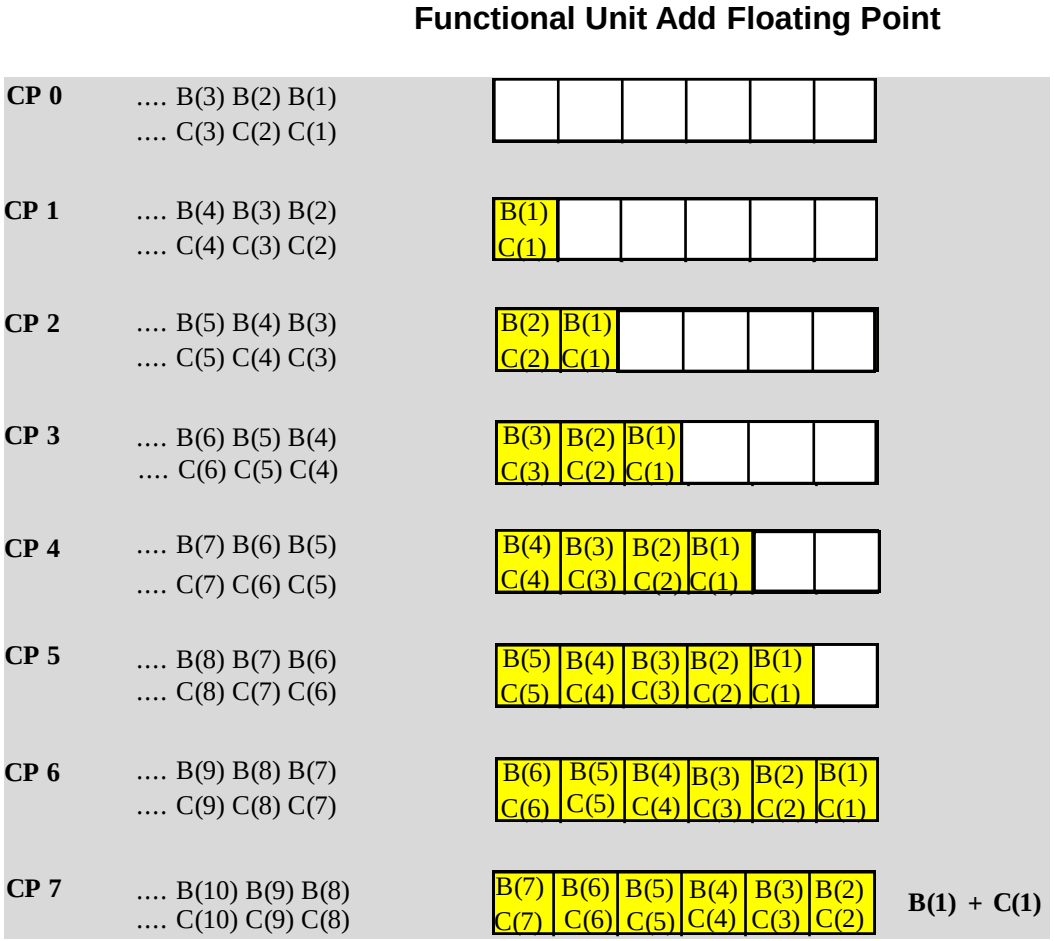
Source : References & Intel Xeon-Phi; http://www.intel.com/

Intel Xeon Phi : Vector Instruction Performance

Vector processing

```
do i = 1, N
   A(i) = B(i)+C(i)
end do
```

V0 ← V1 + V2



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

Intel Xeon Phi : Vector Instruction Performance

- VPU contains 16 SP ALUs, 8 DP ALUs,
- Most VPU instructions have a latency of 4 cycles and TPT 1 cycle
  - Load/Store/Scatter have 7-cycle latency
  - Convert/Shuffle have 6-cycle latency
- VPU instruction are issued in u-pipe
- Certain instructions can go to v-pipe also
  - Vector Mask, Vector Store, Vector Packstore, Vector Prefetch, Scalar

Intel Xeon Phi : Vector Instruction 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/

Demand vectorization by annotation - #pragma simd

- Syntax: `#pragma simd [<clause-list>]`
  - Mechanism to force vectorization of a loop
  - Programmer: asserts a loop ought to be vectorized
  - Compiler: vectorizes the loop or gives an error

Clause	Semantics
No clause	Enforce vectorization of innermost loops; ignore dependencies etc
`vectorlength` *(n_1[, n_2]…)*	Select one or more vector lengths (range: 2, 4, 8, 16) for the vectorizer to use.
`private` *(var_1, var_2, …, var_N)*	Scalars private to each iteration. Initial value broadcast to all instances. Last value copied out from the last loop iteration instance.
`linear` *(var_1:$step_1$, …, var_N:$step_N$)*	Declare induction variables and corresponding positive integer step sizes (in multiples of vector length)
`reduction` *(operator:var_1, var_2,…, var_N)*	Declare the private scalars to be combined at the end of the loop using the specified reduction operator
[`no`*]*`assert`	Direct compiler to assert when the vectorization fails. Default is to assert for SIMD pragma.

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

SIMD Abstraction – Vectorization/SIMD

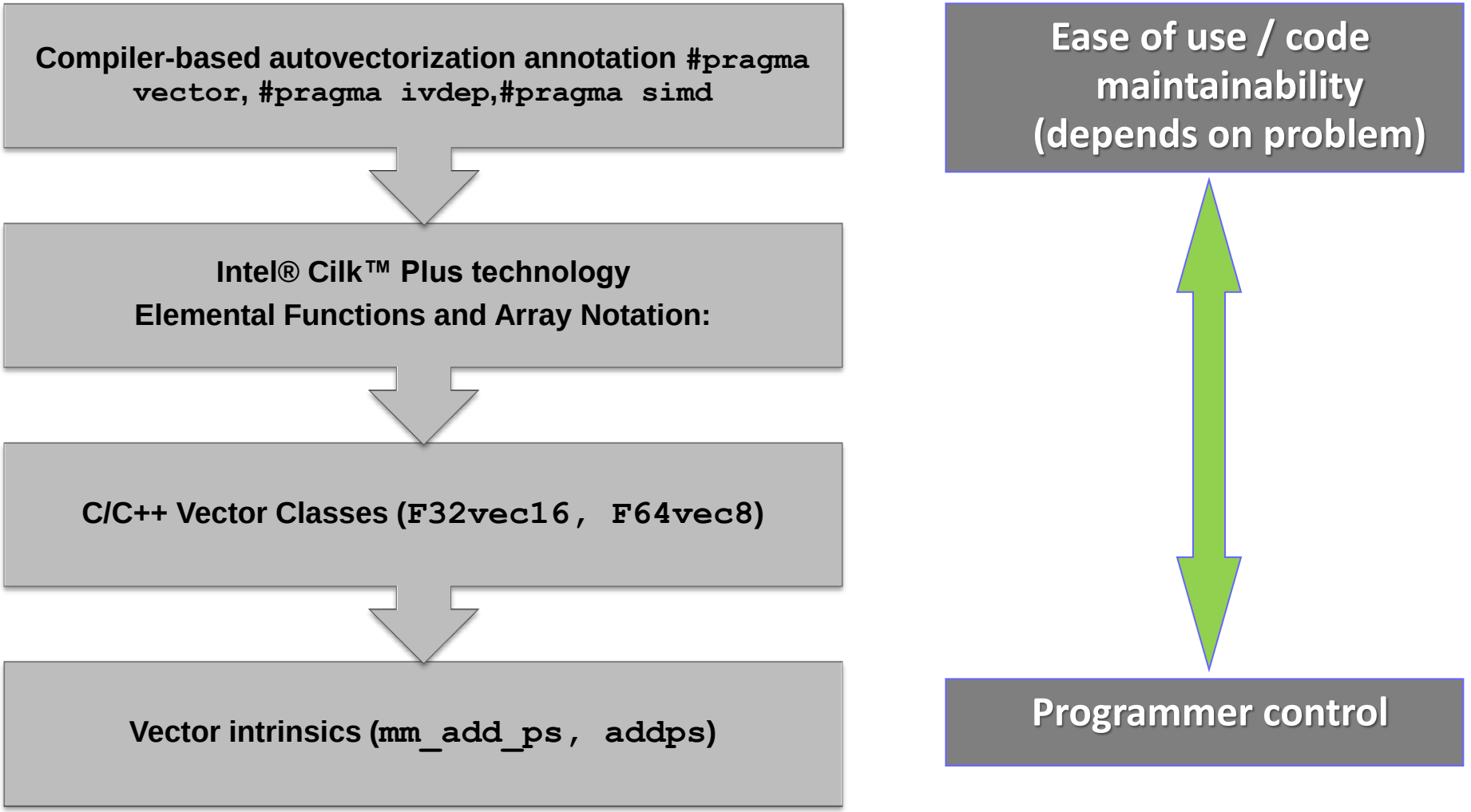
```
for (i = 0; i < 15; i++)
    if (v5[i] < v6[i])
        v1[i] += v3[i];
```

SIMD can simplify your code and reduce the jumps, breaks in program flow control

Note the lack of jumps or conditional code branches

```
v5 = 0 4 7 8 3 9 2 0 6 3 8 9 4 5 0 1
v6 = 9 4 8 2 0 9 4 5 5 3 4 6 9 1 3 0
vcmppi_lt k7, v5, v6
k7 = 1 0 1 0 0 0 1 1 0 0 0 0 1 0 1 0

v3 = 5 6 7 8 5 6 7 8 5 6 7 8 5 6 7 8
v1 = 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
vaddpi v1{k7}, v1, v3
v1 = 6 1 8 1 1 1 8 9 1 1 1 1 6 1 8 1
```

SIMD Abstraction – Options Compared



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

Demand vectorization by annotation - #pragma simd

- Syntax: **#pragma simd [<clause-list>]**
  - Mechanism to force vectorization of a loop
  - Programmer: asserts a loop ought to be vectorized
  - Compiler: vectorizes the loop or gives an error

Clause	Semantics
No clause	Enforce vectorization of innermost loops; ignore dependencies etc
vectorlength (n_1[, n_2]…)	Select one or more vector lengths (range: 2, 4, 8, 16) for the vectorizer to use.
private (var_1, var_2, …, var_N)	Scalars private to each iteration. Initial value broadcast to all instances. Last value copied out from the last loop iteration instance.
linear (var_1:$step_1$, …, var_N:$step_N$)	Declare induction variables and corresponding positive integer step sizes (in multiples of vector length)
reduction (operator:var_1, var_2,…, var_N)	Declare the private scalars to be combined at the end of the loop using the specified reduction operator
[no]assert	Direct compiler to assert when the vectorization fails. Default is to assert for SIMD pragma.

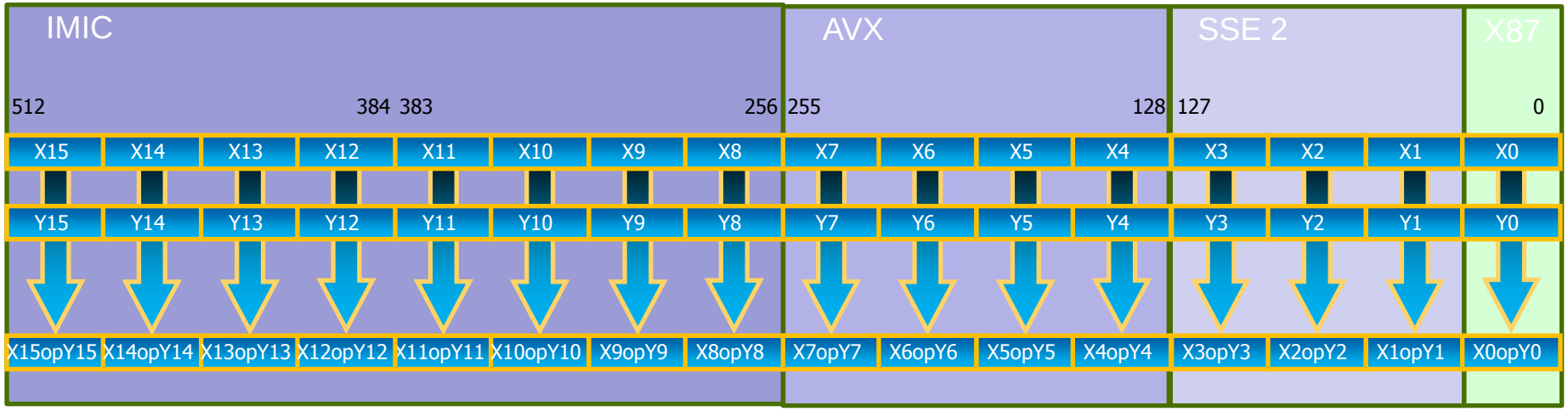
Source : References & Intel Xeon-Phi; http://www.intel.com/

Software Behind the Vectorization

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

Vector (or SIMD) Code computes more than one element at a time.

- [SSE2] 4 elems at a time
  `addps xmm1, xmm2`
- [AVX] 8 elems at a time
  `vaddps ymm1, ymm2, ymm3`
- [IMCI] 16 elems at a time
  `vaddps zmm1, zmm2, zmm3`



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

Hardware resources behind Vectorization

- CPU has lot of computation power in form of SIMD unit.

- XMM (128bit) can operate
  - 16x chars
  - 8x shorts
  - 4x dwords/floats
  - 2x qwords/doubles/float complex

- YMM (256bit) can operate
  - 32x chars
  - 16x shorts
  - 8x dwords/floats
  - 4x qwords/doubles/float complex
  - 2x double complex

- Intel® Xeon Phi™ Coprocessor (512bit) can operate
  - 16x chars/shorts (converted to int)
  - 16x dwords/floats
  - 8x qwords/doubles/float complex
  - 4x double complex

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

Intel Xeon Phi Coprocessors : Compilation and Vectorization

Part-2

Compilation

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 : Compilation and Vectorization

Part-2

Compiler-based 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/

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/

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

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

- 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 "add"s matter?)
  - Vector computation on the target expected to be faster than scalar code?

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

Compiler-Based Autovectorization

- **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 "add"s matter?)
  - Vector computation on the target expected to be faster than scalar code?
- **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

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 "add"s matter?)
  - 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/

Hints to Compiler for Vectorization Opportunities

#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)

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

Hints to Compiler for Vectorization Opportunities

#pragma	Semantics
#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/

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/

Compiler VEC report

- 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 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/

Use Compiler Optimization Switches

```
#include <math.h>
void quad(int length, float *a, float *b, float *c,  \
                float *restrict x1, float *restrict x2)
{
    for (int i=0; i<length; i++) {
       float s = b[i]*b[i] - 4*a[i]*c[i];
        if ( s >= 0 ) {
                s = sqrt(s) ;
           x2[i] = (-b[i]+s)/(2.*a[i]);
           x1[i] = (-b[i]-s)/(2.*a[i]);
    }
   else {
         x2[i] = 0.;
        x1[i] = 0.;
    }
 }
}
```

```
>cc -c -restrict -vec-report2 quad.cpp
```

> quad5.cpp(5) (col. 3): remark: LOOP WAS VECTORIZED.

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/

Prefetch on Intel Multicore and Manycore

- **Objective:** Move data from memory to L1 or L2 Cache in anticipation of CPU Load/Store
- More import on in-order Intel Xeon Phi Coprocessor
- Less important on out of order Intel Xeon Processor
- Compiler prefetching is on by default for Intel® Xeon Phi™ coprocessors at –O2 and above
- Compiler prefetch is not enabled by default on Intel® Xeon® Processors
  - Use external options `-opt-prefetch[=n] n = 1.. 4`
- Use the compiler reporting options to see detailed diagnostics of prefetching per loop
  - Use `-opt-report-phase hlo -opt-report 3`

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

Automatic Prefetches

Loop Prefetch

- Compiler generated prefetches target memory access in a future iteration of the loop
- Target regular, predictable array and pointer access

Interactions with Hardware prefetcher

- Intel® Xeon Phi™ Comprocessor has a hardware L2 prefetcher
- If Software prefetches are doing a good job, Hardware prefetching does not kick in
- References not prefetched by compiler may get prefetched by hardware prefetcher

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

Explicit Prefetch

- **Use Intrinsics**
  - `_mm_prefetch((char *) &a[i], hint);`
    See xmmintrin.h for possible hints (for L1, L2, non-temporal, …)
  - But you have to specify the prefetch distance
  - Also gather/scatter prefetch intrinsics, see zmmintrin.h and compiler user guide, e.g. _mm512_prefetch_i32gather_ps
- **Use a pragma / directive (easier):**
  - #pragma prefetch a [:hint[:distance]]
  - You specify what to prefetch, but can choose to let compiler figure out how far ahead to do it.
- **Use Compiler switches:**
  - `-opt-prefetch-distance=n1[,n2]`
  - specify the prefetch distance (how many iterations ahead, use n1 and prefetches inside loops. n1 indicates distance from memory to L2.

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

Memory Alignment

- Allocated memory on heap
  - `_mm_malloc(int size, int aligned)`
  - `scalable_aligned_malloc(int size, int aligned)`
- Declarations memory:
  - `__attribute__((aligned(n))) float v1[];`
  - `__declspec(align(n)) float v2[];`
- Use this to notify compiler
  - `__assume_aligned(array, n);`
- Natural boundary
  - Unaligned access can fault the processor
- Cacheline Boundary
  - Frequently accessed data should be in 64
- 4K boundary
  - Sequentially accessed large data should be in 4K boundary

Instruction	Length	Alignment
SSE	128 Bits	16 Bytes
AVX	256 Bits	32 Bytes
IMCI	512 Bits	64 Bytes

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

Streaming Store

- Avoid read for ownership for certain memory write operation
- Bypass prefetch related to the memory read
- Use `#pragma vector nontemporal(v1,…)` to drop a hint to compiler
- Without Streaming Stores 448 Bytes read/write per iteration
  - With Streaming Stores, 320 Bytes read/write per iteration
  - Relief Bandwidth pressure; improve cache utilization
  - –vec-report6 displays the compiler action

```
bs_test_sp.c(215): (col. 4) remark: vectorization support: streaming store was generated for CallResult.
bs_test_sp.c(216): (col. 4) remark: vectorization support: streaming store was generated for PutResult.
```

```
for (int chunkBase = 0; chunkBase < OptPerThread; chunkBase += CHUNKSIZE)
{
#pragma simd vectorlength(CHUNKSIZE)
#pragma simd
#pragma vector aligned
#pragma vector nontemporal (CallResult, PutResult)
      for(int opt = chunkBase; opt < (chunkBase+CHUNKSIZE); opt++)
      {
         float CNDD1;
         float CNDD2;
         float CallVal =0.0f, PutVal  = 0.0f;
         float T = OptionYears[opt];
         float X = OptionStrike[opt];
         float S = StockPrice[opt];
                        ……

         CallVal  = S * CNDD1 - XexpRT * CNDD2;
         PutVal  = CallVal  +  XexpRT - S;
         CallResult[opt] = CallVal ;
         PutResult[opt] = PutVal ;
      }
}
```

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

Data Blocking

- Partition data to small blocks that fits in L2 Cache
  - Exploit data reuse in the application.
  - Ensure the data remains in the cache across multiple uses
  - Using the data in cache remove the need to go to memory
  - Bandwidth limited program may execute at FLOPS limit
- Simple case of 1D
  - Data size DATA_N is used WORK_N times from 100s of threads
  - Each handles a piece of work and have to traverse all data

Without Blocking

- 100s of thread pound on different area of DATA_N
- Memory interconnet limit the performance

```
#pragma omp parallel for
for(int wrk = 0; wrk < WORK_N; wrk++)
{
     initialize_the_work(wrk);
     for(int ind = 0; ind < DATA_N; ind++)
     {
        dataptr  datavalue = read_data(dataind);
        result = compute(datavalue);
        aggregate = combine(aggregate, result);
     }
     postprocess_work(aggregate);
}
```

With Blocking

- Cacheable BSIZE of data is processed by all 100s threads a time
- Each data is read once kept reusing until all threads are done with it

```
for(int BBase = 0; BBase < DATA_N; BBase += BSIZE)
{
#pragma omp parallel for
    for(int wrk = 0; wrk < WORK_N; wrk++)
    {
         initialize_the_work(wrk);
         for(int ind = BBase; ind < BBase+BSIZE; ind++)
         {
            dataptr  datavalue = read_data(ind);
            result = compute(datavalue);
            aggregate[wrk] = combine(aggregate[wrk], result);
          }
          postprocess_work(aggregate[wrk]);
    }
}
```

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/

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

Conclusions

- An Overview of Intel Xeon-Phi Compilation & Vectorisation techniques are discussed

Intel Xeon Phi - Coprocessors : An Overview

Shared Address Space Programming –

Part-3

MKL (Math Kernel Library)

Intel Xeon-Phi Coprocessors (Intel MKL)

Simple way to Jobs using Intel MKL (Math Kernel Library)

Details on using MKL (11.0) with Intel Xeon Phi co-processors can be found in references. Also the MKL developer zone contains useful information.

Intel MKL 11.0 Update 2 the following functions are highly optimized for the Intel Xeon Phi coprocessor:

- BLAS Level 3, and much of Level 1 & 2
- Sparse BLAS:
- Some important LAPACK routines (LU, QR, Cholesky)
- Fast Fourier Transformations
- Vector Math Library
- Random number generators in the Vector Statistical Library

Remark : All functions can be used on the Xeon Phi, however the optimization level for wider 512-bit SIMD instructions differs.

Intel Xeon-Phi Coprocessors (Intel MKL)

- On Xeon Phi coprocessor, the following usage models of MKL are available :
  - **Automatic Offload**
  - **Compiler Assisted Offload**
  - **Native Execution**

To know more about the availability of various functions for above usage models, Please refer MKL documents

Intel Xeon-Phi Coprocessors (Intel MKL)

Automatic Offload (AO) :

- In the case of automatic offload the user does not have to change the code at all.
- For automatic offload enabled functions the runtime may automatically download data to the Xeon Phi coprocessor and execute (all or part of) the computations there.
- The data transfer and the execution management is completely automatic and transparent

Remark : The matrix sizes for which MKL decides to offload the computation should be **indicated in function statement.** Refer Intel MKL documents

Intel Xeon-Phi Coprocessors (Intel MKL)

Automatic Offload (AO) :

- **Approach 1 :** call the function **`mkl_mic_enable()`** within the source code
- **Approach 2 :** Set the environment variable **`MKL_MIC_ENABLE =1`**

The data transfer and the execution management is completely automatic and transparent

Remark : If **no** Xeon Phi coprocessor is detected the application runs on the host without penalty.

Intel Xeon-Phi Coprocessors (Intel MKL)

Automatic Offload (AO) : To build a program for automatic offload, the same way of building code as on the **Xeon host** is used:

```
icc -O3 -mkl file.c -o file
```

By default, the MKL library decides when to offload and also tries to determine the optimal work division between the host and the targets . In case of the BLAS routines the user can specify the work division between the host and the coprocessor by calling the routine

```
mkl_mic_set_Workdivision(MKL_TARGET_MIC,0,0.5)
```

or by setting the environment variable

```
MKL_MIC_0_WORKDIVISION=0.5
```

Both examples specify to offload 50% of computation only to the 1st card (card #0).

Intel Xeon-Phi Coprocessors (Intel MKL)

Compiler Assisted Offload (CAO) : In this mode of MKL the offloading is explicitly controlled by compiler pragmas or directives.

Advantage :

1. A big advantage of this mode is that it allows for data persistence on the device.
2. All MKL function can be offloaded in CAO-mode. (In contrast to the automatic offload mode.)

Remarks :

- For Intel compilers it is possible to use AO and CAO in the same program, however the work division must be explicitly set for AO in this case. Otherwise, all MKL AO calls are executed on the host.
- MKL functions are offloaded in the same way as any other offloaded function.

Intel Xeon-Phi Coprocessors (Intel MKL)

Compiler Assisted Offload (CAO) : To build a program for compiler assisted offload, the following command is recommended by Intel:

```
#pragma offload target(mic) \
        in(transa, transb, N, alpha, beta) \
        in(A:length(N*N)) in(B:length(N*N)) \
        in(C:length(N*N)) \
        out(C:length(N*N) alloc_if(0))
{
   sgemm(&transa, &transb, &N, &N, &N, \
            &alpha, A, &N, B, &N, &beta, C, &N);
}
```

Remarks :. Refer Intel MKL documents

Intel Xeon-Phi Coprocessors (Intel MKL)

Compiler Assisted Offload (CAO) : To build a program for compiler assisted offload, the following command is recommended by Intel:

```
icc -O3 -openmp -mkl \
-offload-option,mic,ld, \
 "-L$MKLROOT/lib/mic -Wl,\
 --start-group -lmkl_intel_lp64 \
 -lmkl_intel_thread \
 -lmkl_core -Wl,--end-group" \
  hello.c -o file
```

Remarks : Setting larger pages by the environment setting `MIC_USE_2MB_BUFFERS=16K` usually increases performance. It is also recommended to exploit data persistence with CAO. Refer Intel MKL documents

Intel Xeon-Phi Coprocessors (Intel MKL)

Native Execution : In this mode of MKL the Intel Xeon Phi coprocessor is used as an independent compute node.

To build a program for native mode, the following compiler settings should be used:

```
icc -O3 -mkl -mmic file.c -o file
```

Example code : Example code can be found under

`$MKLROOT/examples/mic_ao` and

`$MKLROOT/examples/mic_offload`

Remarks : The binary must then be manually copied to the coprocessor via **ssh** and directly started on the coprocessor or Cluster environment automatically copy the data

Intel Xeon-Phi Coprocessors (Intel MKL)

Native Execution : In this mode of MKL the Intel Xeon Phi coprocessor is used as an independent compute node.

To build a program for native mode, the following compiler settings should be used:

```
icc -O3 -mkl -mmic file.c -o file
```

Example code : Example code can be found under

`$MKLROOT/examples/mic_ao` and

`$MKLROOT/examples/mic_offload`

Remarks : The binary must then be manually copied to the coprocessor via **ssh** and directly started on the coprocessor or Cluster environment automatically copy the data

Intel Xeon-Phi Coprocessors (Intel MKL)

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/

An Overview of Intel Xeon-Phi Coprocessors

Conclusions

- An Overview of Intel Xeon-Phi Architecture; Tuning & Performance of Software threading- using MKL

This slide is intentionally kept Blank

Intel Xeon Phi - Coprocessors : An Overview

Shared Address Space Programming –

Part-3
POSIX Threads

Prog.API - Multi-Core Systems with Devices

Options for Parallelism – pthreads*

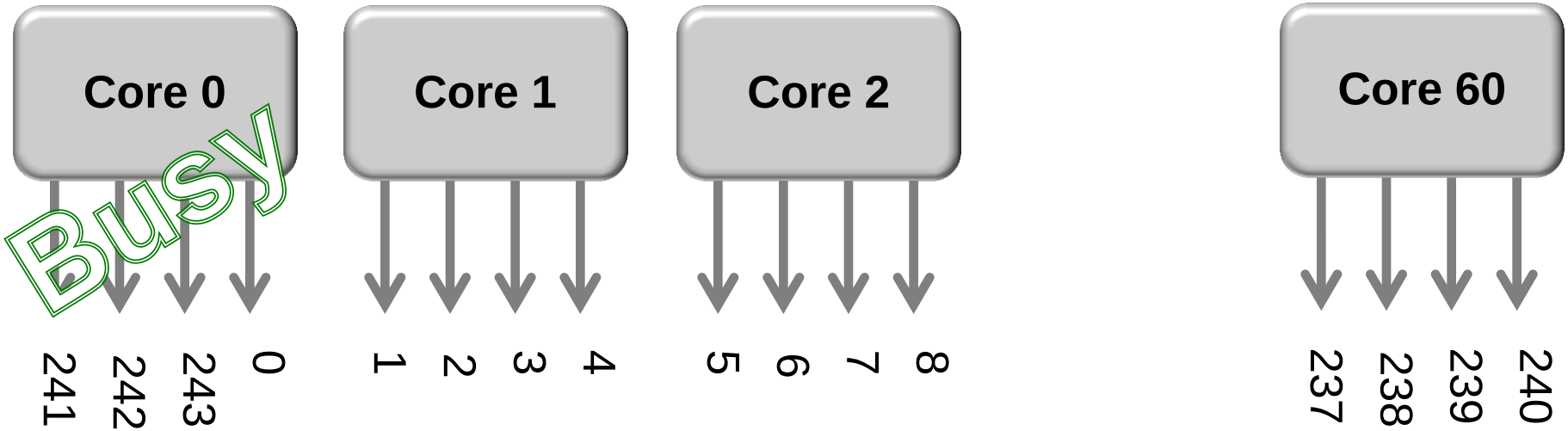
- POSIX* Standard for thread API with 20 years history
- Foundation for other high level threading libraries
- Independently exist on the host and Intel® MIC
- No extension to go from the host to Intel® MIC
- Advantage: Programmer has explicit control
  - From workload partition to thread creation, synchronization, load balance, affinity settings, etc.
- **Disadvantage:** Programmer has too much control
  - Code longevity
  - Maintainability
  - Scalability

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

Intel Xeon-Phi : Programming Env.

Thread Affinity using pthreads*

- Partition the workload to avoid load imbalance
  - Understand static vs. dynamic workload partition
- Use pthread API, define, initialize, set, destroy
  - Set CPU affinity with `pthead_setaffinity_np()`
  - Know the thread enumeration and avoid core 0
  - Core 0 boots the coprocessor, job scheduler, service interrupts



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

Intel Xeon Phi - Coprocessors : An Overview

Shared Address Space Programming –

Part-3

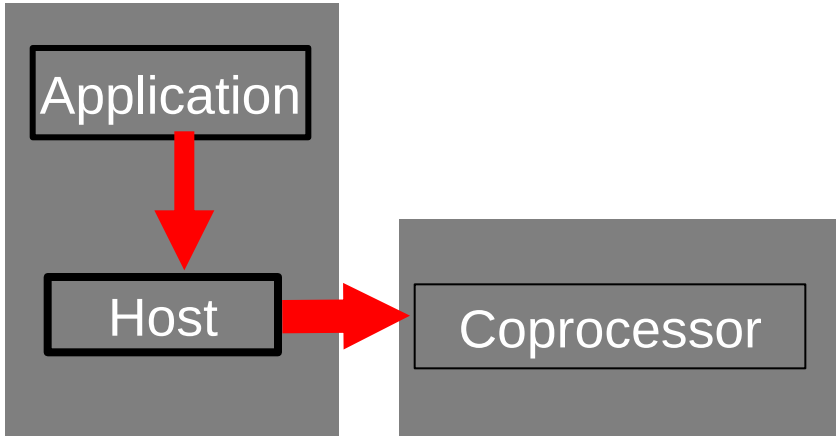
OpenMP

Intel Xeon-Phi : OpenMP I Prog. Model

OpenMP

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

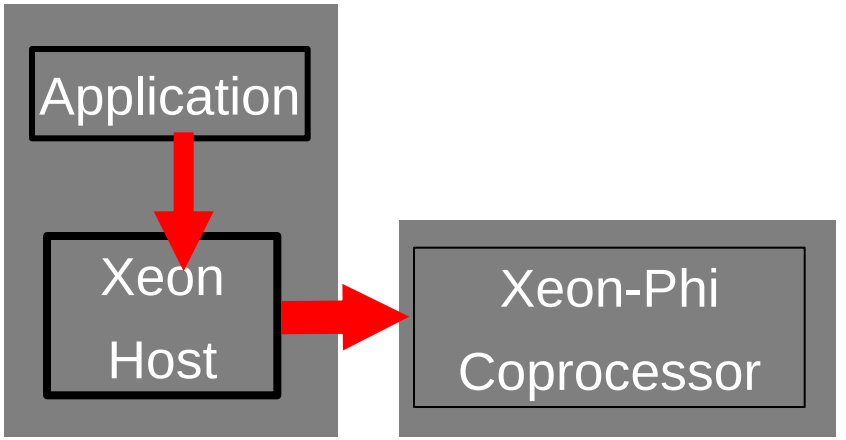
  - **Realized with Complier Options**



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
- Native OpenMP on the Xeon Phi coprocessor
- OpenMP on the Xeon Host with OpenMP offload

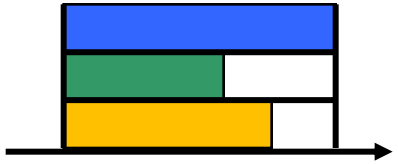


Intel Xeon-Phi : OpenMP I Prog. Model

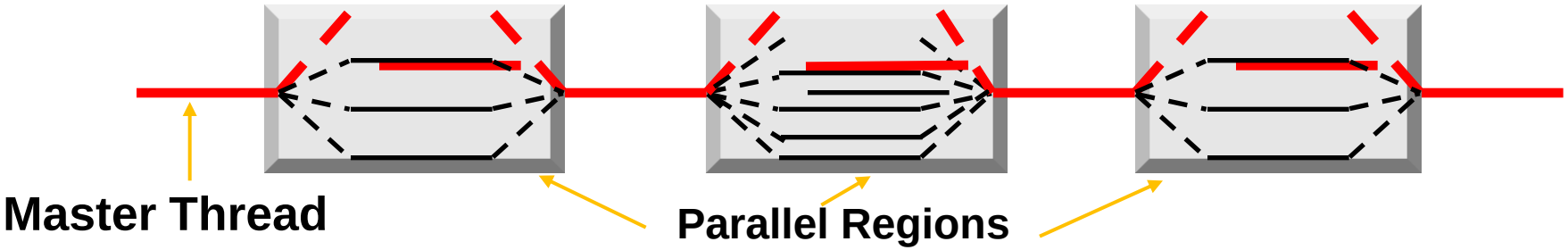
Options for Parallelism – OpenMP*

- 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



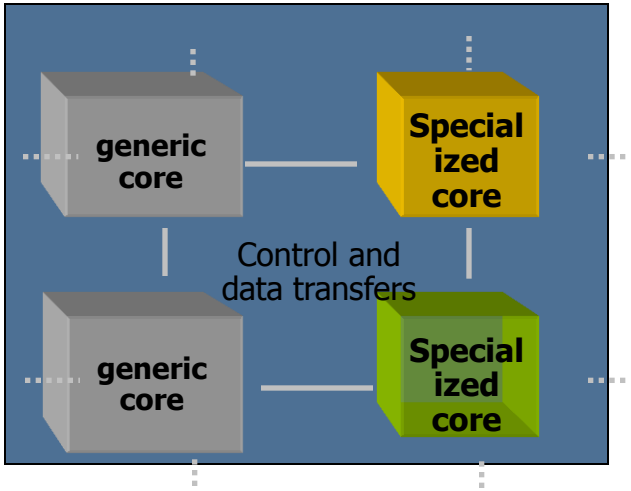
Source : References & Intel Xeon-Phi; http://www.intel.com/

Intel Xeon-Phi : OpenMP I Prog. Model

OpenMP Evolution Beyond 1,00,000 cores

- OpenMP language committee is actively working toward the expression of locality and heterogeneity
  - And to improve task model to enhance asynchrony

- How to identify code that should run on a certain kind of core?
- How to share data between host cores and other devices?
- How to minimize data motion?
- How to support diversity of cores?



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 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 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 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 : OpenMP 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 : OpenMP Prog. Model

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 : OpenMP 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**

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

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 : OpenMP 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 : OpenMP 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 : OpenMP 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

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

Intel Xeon-Phi : MPI Prog. Model

Setting Up the MPI Environment

The following commands have to be executed to set up the MPI environment:

copy MPI libraries and binaries to the card (as root)

only copying really necessary files saves memory

```
scp /opt/intel/impi/4.1.0.024/mic/lib/* mic0:/lib
scp /opt/intel/impi/4.1.0.024/mic/bin/* mic0:/bin
```

setup Intel compiler variables

```
. /opt/intel/composerxe/bin/compilervars.sh intel64
```

setup Intel MPI variables

```
. /opt/intel/impi/4.1.0.024/bin64/mpivars.sh
```

Intel Xeon-Phi : Hybrid MPI/OpenMP

Programming Models

Two Major Approaches

1. A **MPI offload** approach (MPI ranks reside on the host CPU and work is offloaded to the Xeon Phi Coprocessor

1. A **symmetric** approach in which MPI ranks reside both on the CPU and on the Xeon Phi.

AMPI program can be structured using either model

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

Intel Xeon-Phi : Hybrid MPI/OpenMP

Programming Models : Threading of MPI ranks

1. For **hybrid OpenMP/MPI** applications use the **thread safe** version of the Intel MPI Library by using the `-mt_mpi` compiler driver option.
2. A desired process pinning scheme can be set with the `I_MPI_PIN_DOMAIN` environment variable. It is recommended to use the following setting:

```
$exportI_MPI_PIN_DOMAIN = omp
```

By using this, one sets the process pinning domain size to be `OMP_NUM_THREADS.` In this way, every MPI process is able to create `$OMP_NUM_THREADS` number of threads that will run within the corresponding domain.

Intel Xeon-Phi : Hybrid MPI/OpenMP

Programming Models : Threading of MPI ranks

It is recommended to use the following setting:

```
$exportI_MPI_PIN_DOMAIN = omp
```

- Using this, one sets the process pinning domain size to be **OMP_NUM_THREADS.** Every MPI process is able to create **$OMP_NUM_THREADS** no. of threads that will run within the corresponding domain.
- If this variable is not set, each process will create a number of threads per MPI process equal to the no. of cores (treated as a separate domain.)
- To pin OpenMP threads within a particular domain, one could use the **KMP_AFFINITY** environment variable

Intel Xeon-Phi : MPI Programming Model

Setting up the MPI environment :

Details about using the Intel MPI library on Xeon Phi coprocessor systems can be found in references

The following commands have to be executed to set up the MPI environment:

copy MPI libraries and binaries to the card (as root)

only copying really necessary files saves memory

```
scp /opt/intel/impi/4.1.0.024/mic/lib/* mic0:/lib
scp /opt/intel/impi/4.1.0.024/mic/bin/* mic0:/bin
```

setup Intel compiler variables

```
. /opt/intel/composerxe/bin/compilervars.sh intel64
```

setup Intel MPI variables

```
. /opt/intel/impi/4.1.0.024/bin64/mpivars.sh
```

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

Intel Xeon-Phi Coprocessor : MPI on Cluster

Network Fabric : The following network fabrics are available for the Intel Xeon Phi coprocessor (Refer C-DAC PARAM YUVA Cluster)

Fabric Name	Description
shm	Shared-memory
tcp	TCP/IP-capable network fabrics, such as Ethernet and InfiniBand (through IPoIB)
ofa	OFA-capable network fabric including InfiniBand (through OFED verbs)
dapl	DAPL–capable network fabrics, such as InfiniBand, iWarp, Dolphin, and XPMEM (through DAPL)

The Intel MPI library tries to automatically use the best available network fabric detected (usually shm for in-tra-node communication and InfiniBand (dapl, ofa) for inter-node communication).

The default can be changed by setting the I_MPI_FABRICS environment variable to I_MPI_FABRICS=<fabric> or I_MPI_FABRICS=<intra-node fabric>:<inter-nodes fab-ric>. The availability is checked in the following order: shm:dapl, shm:ofa, shm:tcp.

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

Intel Xeon Phi - Coprocessors : An Overview

Shared Address Space Programming –

Part-3

Intel TBB

Intel Xeon-Phi : Intel TBB 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)

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

Intel Xeon-Phi : Intel TBB 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 : Intel TBB 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

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.

Intel Xeon-Phi : Intel TBB Prog.

Intel TBB Advantages

- Intel TBB Generic Programming
- Intel TBB is easy to start
- Intel TBB obeys to logical parallelism:
- Intel TBB is compatible with other programming models:
- The Intel TBB template-based approach – Performance gain can be achieved.

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

Intel Xeon-Phi : Intel TBB Prog.

Intel TBB : Using TBB NATIVELY

A minimal C++ TBB example looks as follows:

```
#include “tbb/task_scheduler_init.h”
#include “tbb/parallel_for.h”
#include "tbb/blocked_range.h"
   using namespace tbb;
   int main() {
   task_scheduler_init init;
   return 0;
}
```

Intel Xeon-Phi : Intel TBB Prog.

Intel TBB : Using TBB NATIVELY

- Scalable parallelism can be achieved by **parallelizing** a loop of iterations that can each run independently from each other. The **`parallel_for`** template function replaces a serial loop

- A typical example would be to apply a function **`MatAdd`** on all elements of an array over the iterations space of type **`size_t`** going from **`0`** to **`n-1`**

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

Intel Xeon-Phi : Intel TBB Prog.

Intel TBB : Using TBB NATIVELY

```
void SerialApplyMatAdd(float a[], size_t n) {
for( size_t i=0; i!=n; ++i )
     MatAdd(a[i]);
}
```

becomes

```
void  ParallelApplyMatAdd(float  a[],size_t  n)
{
     parallel_for(size_t(0),n,[=](size_ti)
     {MatAdd(a[i]);});
}
```

Compiling programs that employ TBB constructs, link in the Intel TBB shared library with `-ltbb`.

```
icc -mmic -ltbb foo.cpp
```

Intel TBB : Using TBB OFFLOAD

The Intel TBB header files are not available on the Intel MIC target environment by default (the same is also true for Intel Cilk Plus). To make them available on the coprocessor the header files have to be wrapped with

#pragma offloaddirectives as demonstrated in the example below:

```
#pragma offload_attribute (push,target(mic))
#include “tbb/task_scheduler_init.h”
#include “tbb/parallel_for.h”
#include "tbb/blocked_range.h"
#pragma offload_attribute (pop)
```

Intel Xeon-Phi : Intel TBB Prog.

Intel TBB : Using TBB NATIVELY

Functions called from within the offloaded construct and global data required on the Intel Xeon Phi coprocessor should be appended by the special function **`__attribute__((target(mic)))`**.

Codes using Intel TBB with an offload should be compiled with **`-tbbflag`** instead of **`-ltbb`**.

Compiling programs that employ TBB constructs, link in the Intel TBB shared library with **`-ltbb`**.

```
icc -mmic -ltbb foo.cpp
```

An Overview of Multi-Core Processors

Conclusions

- An Overview of Xeon-Phi Architectures, Programming on based on Shared Address Space Platforms – OpenMP, MPI, Intel TBB, Performance of Software threading are discussed.

Intel Xeon Phi - Coprocessors : An Overview

Shared Address Space Programming –

Part-3

Cilk Plus

Intel Xeon-Phi - for Parallelism: Intel® Cilk™ Plus

- MIT Cilk – The original research project from MIT , culminating in Cilk-5.4.6. MIT Cilk was implemented as a source-to-source translator that converts Cilk code to C and them compiled the resulting C source.
  - Only supported C code with Cilk keywords
  - All parallel functions had to be marked with a cilk keyword
  - Cilk functions had to be spawned, not called

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

Intel Xeon-Phi - for Parallelism: Intel® Cilk™ Plus

- Cilk++ - Compilers developed by Cilk Arts, Inc. Cilk Arts licensed the Cilk technology from MIT.
  - Only supported C++ code
  - Used a non-standard calling convention, meaning you had to use a cilk::context to
  - call Cilk functions from C or C++
  - Cilk files used the .cilk extension
  - Released by Intel as unsupported software through the WhatIf site

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

Intel Xeon-Phi - for Parallelism: Intel® Cilk™ Plus

- Cilk++ - Intel Cilk Plus – Fully integrated into the Intel C/C++ compiler
  - Supports both C and C++
  - Uses standard calling conventions
  - Includes both task and data parallelism

Intel Xeon-Phi - for Parallelism: Intel® Cilk™ Plus

- **Why to use it ?**
  - Intel® Cilk™ Plus is the easiest, quickest way to harness the power of both multicore and vector processing.
- **What is it ?**
  - Intel Cilk Plus is an extension to the C and C++ languages to support data
- **Primary Features :**

HPC

  - In efficient work-stealing scheduler provides nearly optimal scheduling of parallel tasks
  - Vector support unlocks the performance that's been hiding in your processors
  - Powerful hyperobjects allow for lock-free programming

Intel Xeon-Phi - for Parallelism: Intel® Cilk™ Plus

Primary Features :

- **Easy to Learn**
  - Only 3 new keywords to implement task parallelism
  - Serial semantics make understanding and debugging the parallel program easier
  - Array Notations provide a natural way to express data parallelism

- **Easy to Use**
  - Automatic load balancing provides good behaviour in multi-programmed environments
  - Existing algorithms easily adapted for parallelism with minimal modification
  - Supports both C and C++ programmers

Intel Xeon-Phi - for Parallelism: Intel® Cilk™ Plus

Primary Features :

Keywords : Simple, powerful expression of task parallelism:

- **cilk_for -** Parallelize for loops
- **cilk_spawn -** Specifies that a function can execute inparallel with the remainder of the calling function
- **cilk_sync -** Specifies that all spawned calls in a function must complete before execution continues

Other Options for Parallelism: Intel® Cilk™ Plus

C/C++ extension for fine-grained task parallelism. 3 keywords:

`_Cilk_spawn`

- Function call ***may*** be run in parallel with caller – up to the runtime

`_Cilk_sync`

- Caller waits for all children to complete

`_Cilk_for`

- Iterations are structured into a work queue
- Busy cores do not execute the loop
- Idle cores steal work items from the queue
- Countable loop Granularity is N/2, N/4, N/8, for trip count of N
- Intended use:
  - when iterations are not balanced, or
  - When overall load is not known at design time

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

Intel Xeon-Phi - for Parallelism: Intel® Cilk™ Plus

Primary Features :

Keywords : Simple, powerful expression of task parallelism:

- **Reducers:** Eliminate contention for shared variables among tasks by automatically creating views of them as needed and "reducing" them in a lock free manner.
- **Array Notation :** Data parallelism for arrays or sections of arrays.
- **Elemental Functions :** Define functions that can be vectorized when called from within an array notation expression or a **`#pragma simd`** loop
- **`#pragma simd:`** Specifies that a loop is to be vectorized

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

Intel Xeon-Phi : Prog. Env. Cilk Plus

Cilk Plus Keywords

cilk_spawn and cilk_sync:

Example of Fibonacci number calculator program

❖ Sequential

```
int fib(int n)
{
  if (n < 2)
      return n;
      int x = fib(n-1);
      int y = fib(n-2);
      return x + y;
 }
```

❖ (With Cilk Plust Key Words)

```
int fib(int n)
{
  if (n < 2)
      return n;
      int x = cilk_spwan fib(n-1);
      int y = fib(n-2);
      cilk_sync;
      return x + y;
 }
```

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

Intel Xeon-Phi : Programming Env.

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

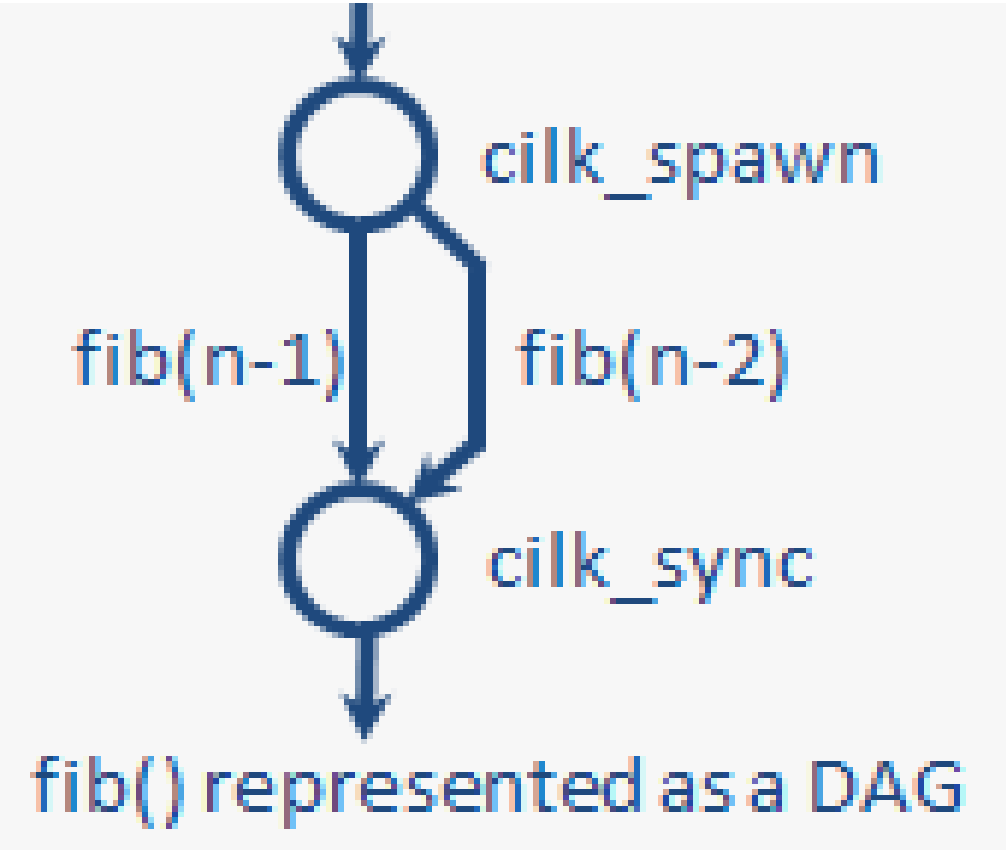
```
class_Cilk_Shared common {
    int data1;
    int *data2;
    class common *next;
    void process();
}
_Cilk_Shared class common obj1, obj2;
_Cilk_spawn _offload obj1.process();
_Cilk_spawn  _offload  obj2.process();
```

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

Intel Xeon-Phi : Prog. Env. Cilk Plus

Cilk Plus Keywords

cilk_spawn and cilk_sync:



p://www.intel.com/

Intel Xeon-Phi : Prog. Env. Cilk Plus

Cilk Plus Keywords

cilk_spawn and cilk_sync:

Example of Fibonacci number calculator program

❖ Sequential

```
int fib(int n)
{
  if (n < 2)
      return n;
      int x = fib(n-1);
      int y = fib(n-2);
      return x + y;
 }
```

❖ (With Cilk Plust Key Words)

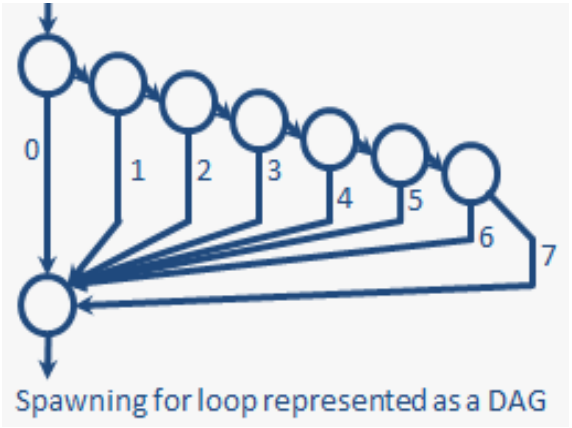
```
int fib(int n)
{
  if (n < 2)
      return n;
      int x = cilk_spwan fib(n-1);
      int y = fib(n-2);
      cilk_sync;
      return x + y;
 }
```

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

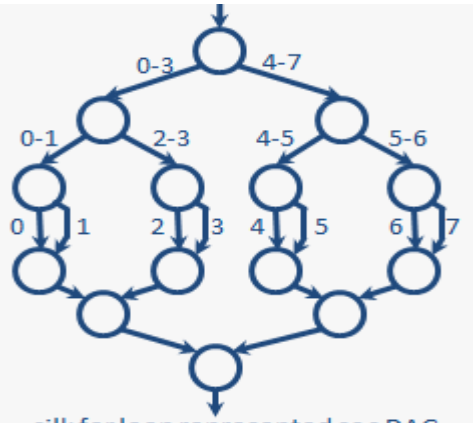
Intel Xeon-Phi : Prog. Env. Cilk Plus

Cilk Plus Keywords

Advantage of cilk_for over cilk_spawn



Spawning for loop represented as a DAG



cilk for loop represented as a DAG

cilk_spawn code

```
for (int i = 0; i < 8; ++i)
 {
   cilk_spawn do_work(i);
 }
cilk_sync;
```

cilk_for code

```
cilk_for (int i = 0; i < 8; ++i)
{    do_work(i);
}
```

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

Intel Xeon-Phi - for Parallelism: Intel® Cilk™ Plus

Cilk Plus Keywords

❖ **Features of cilk_spawn:**

- **cilk_spawn** permits parallelism. It does not command it. cilk_spawn does not create a thread. It allows the runtime to steal the **continuation** to execute in another worker thread.
- A **strand** is a sequence of instructions that starts or ends on a statement which will change the parallelism.
- Permitting parallelism instead of commanding it is an aspect of the **serial semantics** of a deterministic **Intel Cilk Plus** application.
- It is always possible to run an **Intel Cilk Plus** application with a single worker, and it should give identical results to the **serialization** of that program

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

Intel Xeon-Phi - for Parallelism: Intel® Cilk™ Plus

Cilk Plus Reducers : The Cilk Plus Reducer Library :

Lists

reducer_list_append: Creates a list by adding elements to the back.

reducer_list_prepend: Creates a list by adding elements to the front.

Min/Max

reducer_max: Calculates the maximum value of a set of values.

reducer_max_index: Calculates the maximum value and index of that value of a set of values.

reducer_min: Calculates the minimum value of a set of values.

reducer_min_index: Calculates the minimum value and index of that value of a set of values.

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

Intel Xeon-Phi - for Parallelism: Intel® Cilk™ Plus

Cilk Plus Reducers : The Cilk Plus Reducer Library :

Math Operators

reducer_opadd: Calculates the sum of a set of values.

Bitwise Operators

- reducer_opand: Calculates the binary AND of a set of values.
- reducer_opor: Calculate the binary OR of a set of values.
- reducer_opxor: Calculate the binary XOR of a set of values.

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

Intel Xeon-Phi - for Parallelism: Intel® Cilk™ Plus

Cilk Plus Reducers : The Cilk Plus Reducer Library :

```
void locked_list_test()
{  mutex m;
    std::list<char>letters;
  // Build the list in parallel
    cilk_for(char ch = 'a'; ch <= 'z'; ch++)
    {  simulated_work();
        m.lock();
        letters.push_back(ch);
        m.unlock();   }
 // Show the resulting list
    std::cout << "Letters from locked list: ";
    for(std::list<char>::iterator i = letters.begin(); i != letters.end(); i++)
    { std::cout << " " << *i;
    }std::cout << std::endl;}
```

Letters from locked list: y g n d t a w x e z q j o h b u f v c k i r p l m s

Intel Xeon-Phi - for Parallelism: Intel® Cilk™ Plus

Cilk Plus Reducers : The Cilk Plus Reducer Library :

```
void reducer_list_test()
{  cilk::reducer_list_append<char> letters_reducer;
  // Build the list in parallel
    cilk_for(char ch = 'a'; ch <= 'z'; ch++)
    {  simulated_work();
       letters_reducer.push_back(ch);
    }
// Fetch the result of the reducer as a standard STL list
    const std::list<char> &letters = letters_reducer.get_value();
  // Show the resulting list
    std::cout << "Letters from reducer_list:";
    for(std::list<char>::const_iterator i = letters.begin(); i != letters.end(); i++)
    { std::cout << " " << *i;
    }std::cout << std::endl;}
```

Letters from reducer_list: a b c d e f g h i j k l m n o p q r s t u v w x y z

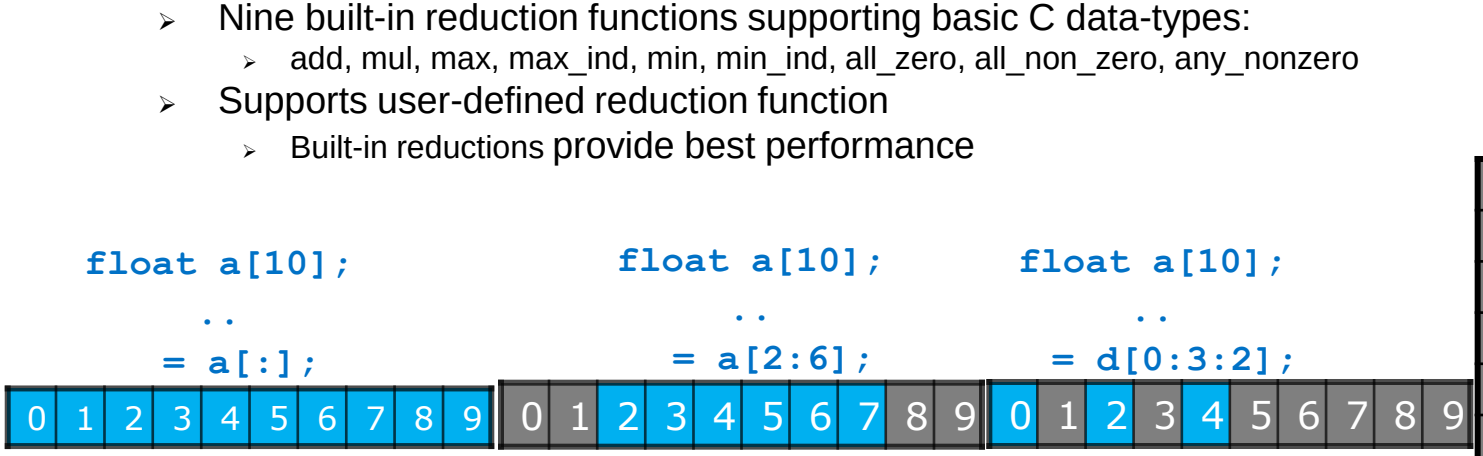
Source : References & Intel Xeon-Phi; http://www.intel.com/

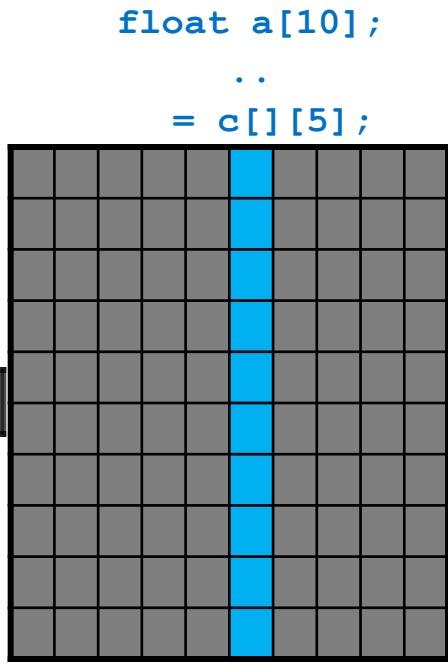
Intel® Cilk™ Plus Array Notation

- C/C++ Language extension supported by the Intel® Compiler
- Based on the concept of array-section notation:

```
<array>[<low_bound> : <len> : <stride>] [<low_bound> : <len> :
  <stride>]…
```

- C/C++ Operators / Function Calls
  - `d[:] = a[:] + (b[:] * c[:])`
  - `b[:] = exp(a[:]); // Call exp() on each element of a[]`
- Reductions combine array section elements to generate a scalar result
  - Nine built-in reduction functions supporting basic C data-types:
    - add, mul, max, max_ind, min, min_ind, all_zero, all_non_zero, any_nonzero
  - Supports user-defined reduction function
    - Built-in reductions provide best performance





Intel Xeon-Phi - for Parallelism: Intel® Cilk™ Plus

Cilk Plus Array Notation

- Intel Cilk Plus includes extensions to C and C++ that allows for parallel operations on arrays.
- The intent is to allow users to express high-level vector parallel array operations. –
- This helps the compiler to effectively vectorize the code.
- Array notation can be used for both static and dynamic arrays.
- The extension has parallel semantics that are well suited for per-element operations that have no implied ordering and are intended to execute in data-parallel instructions.

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

Intel Xeon-Phi - for Parallelism: Intel® Cilk™ Plus

Cilk Plus Array Notation

A new operator [:] delineates an array section:

```
array-expression[lower-bound : length : stride]
```

- Length is used instead of upper bound as C and C++ arrays are declared with a given length.
- The three elements of the array notation can be any integer expression. The user is responsible for keeping the section within the bounds of the array.
- Each `argument to [:]` may be omitted if the default value is sufficient.
  - The default lower-bound is `0.`
  - The default length is the length of the array. If the length of the array is not known, length must be specified.
  - The default stride is 1. If the stride is defaulted, the second ":" may be omitted.

Intel Xeon-Phi - for Parallelism: Intel® Cilk™ Plus

User-mandated Vectorization(pragma simd)

- SIMD (Single Instruction, Multiple Data) vectorization uses the **`#pragma simd`** pragma to enforce loop vectorization.
- Consider an example in C++ where the function **`add_floats()`** uses too many unknown pointers, preventing automatic vectorization. You can give a data-dependence assertion using the **`auto-vectorization`** hint via **`#pragma ivdep`** and let the compiler decide whether the **`auto-vectorization`** optimization should be applied to the loop. Or you can now enforce vectorization of this loop by using **`#pragma simd`**.

```
void add_floats(float *a, float *b, float *c,
                float *d, float *e, int n)
{  int i;
   #pragma simd
   for (i=0; i<n; i++){
     a[i] = a[i]+ b[i]+c[i]+d[i] + e[i];}
}
```

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

Intel Xeon-Phi - for Parallelism: Intel® Cilk™ Plus

Cilk Plus : User-mandated Vectorization(pragma simd)

- The one big difference between using the SIMD pragma and **auto-vectorization** hints is that with the SIMD pragma, the compiler generates a warning when it is unable to vectorize the loop. With **auto-vectorization** hints, actual vectorization is still under the discretion of the compiler, even when you use the **#pragma vector** always hint.

- If a **#pragma simd** annotated loop is not vectorized by the compiler, the loop holds its serial semantics.

- By default "**#pragma simd**" is set to "**noassert**".and compiler will issue a warning if the loop fails to **vectorize**.

- To direct the compiler to assert an error when the **#pragma simd** annotated loop fails to vectorize , add the "assert" clause to the **#pragma simd**

Intel® Cilk™ Plus Technology: Elemental Function

- Allow you to define data operations using scalar syntax
- Compiler apply the operation to data arrays in parallel, utilizing both SIMD parallelism and core parallelism

Programmer	Build with Intel Cilk Plus Compiler
1. Writes a standard C/C++ scalar syntax 2. Annotate it with **__declspec**(vector) 3. Use one of the parallel syntax choices to invoke the function	1. Generates vector code with SIMD Instr. 2. Invokes the function iteratively, until all elements are processed 3. Execute on a single core, or use the task scheduler, execute on multicores

```
__declspec (vector)
double BlackScholesCall(double S,
                        double K,
                        double T)
                  {
 double d1, d2, sqrtT = sqrt(T);
d1 = (log(S/K)+R*T)/(V*sqrtT)+0.5*V*sqrtT;
            d2 = d1-(V*sqrtT);
 return S*CND(d1) - K*exp(-R*T)*CND(d2);
                  }
```

```
Cilk_for (int i=0; i < NUM_OPTIONS; i++)
     call[i] = BlackScholesCall(SList[i],
                                KList[i],
                                TList[i]);
```

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

Intel Xeon-Phi - for Parallelism: Intel® Cilk™ Plus

Elemental Functions

- An elemental function is a regular function, which can be invoked either on scalar arguments or on array elements in parallel. You define an elemental function by adding

  "__declspec(vector)" (on Windows*) and

  "__attribute__((vector))" (on Linux*) before

- the function signature:

```
__declspec (vector)
double  ef_add (double x, double y)  {return x + y;}
```

When you declare a function as elemental the compiler generates a short vector form of the function, which can perform your function's operation on multiple arguments in a single invocation.

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

Intel Xeon-Phi - for Parallelism: Intel® Cilk™ Plus

Elemental Functions

The vector form of the function can be invoked in parallel contexts in the following ways:

1. From a for-loop. It gets auto-vectorized; a loop that only has a call to an elemental function is always vectorizable, but the **`auto-vectorizer`** is allowed to apply performance heuristics and decide not to vectorize the function.
2. From a for-loop with **`pragma simd`**. If the elemental function is called from a loop with "**`pragma simd`**", the compiler no longer does any performance heuristics, and is guaranteed to call the vector version of the function.
3. From a **`cilk_for`**
4. From an array notation syntax. .

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/

Data Access Semantics

❖ **Data Access Semantics**

- ➢ Explicit Offloading
- ➢ Implicit Offloading

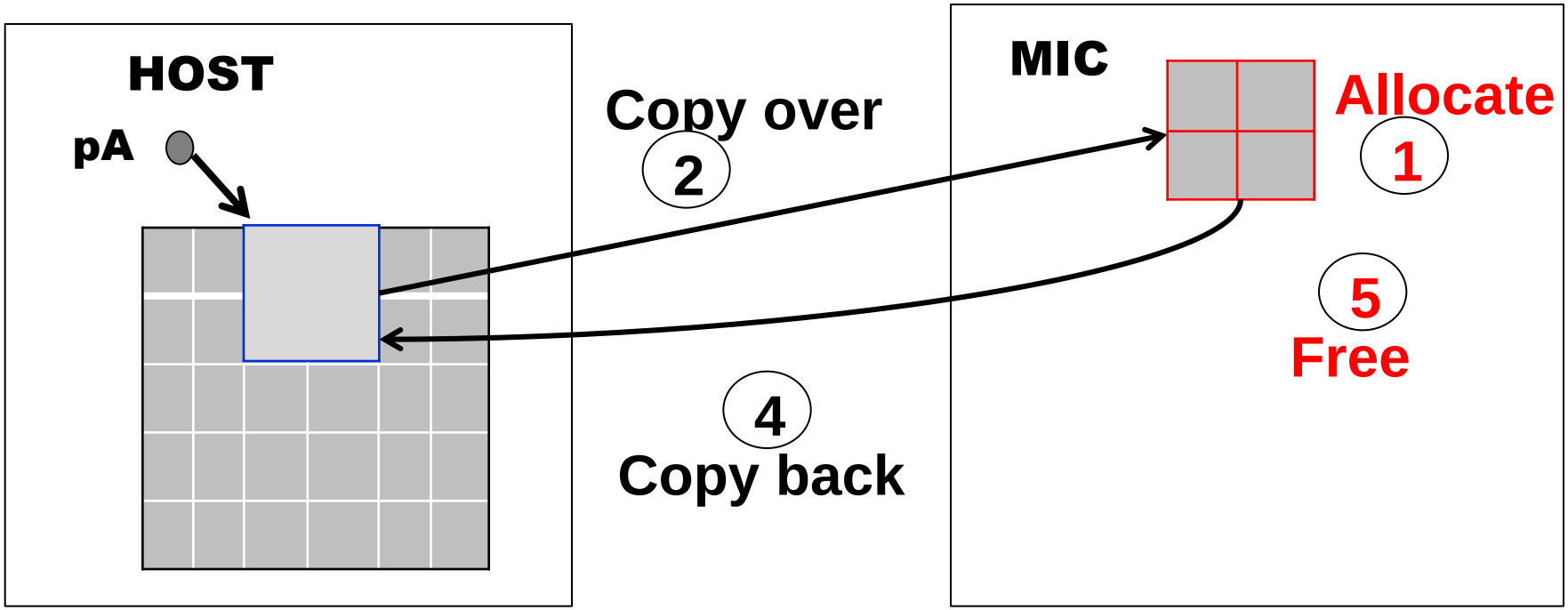
❖ Complier Data Transfer Overview

- ➢ The host CPU and the Intel Xeon Phi coprocessor do not share physical or virtual memory in hardware
- ➢ Two offload transfer models are : **Explicit Copy** and **Implicit Copy**

Data Access Semantics

- Two offload transfer models are : **Explicit Copy** and **Implicit Copy**
- **Explicit Copy :**
  - Programmer designates variables that need to be copied between **host** and **card** in the *offload directive*
  - **Syntax:** Pragma/directive-based
  - **C/C++ Example:** #pragma offload target(mic) in(data:length(size)) (OpenMP, Pthreads, Intel TBB, MPI with OpenMP/Pthreads/Intel TBB)

Compiler : Offload using Explicit Copies – Data movement



- Default treatment of **`in/out`** variables in a **`#pragma`** offload statement

Compiler : Offload using Explicit Copies – Data movement

- Default treatment of **`in/out`** variables in a **`#pragma`** offload statement
  - At the start of an offload:
    - Space is allocated on the coprocessor
    - **`in`** variables are transferred to the coprocessor
  - At the end of an offload:
    - **`out`** variables are transferred from the coprocessor
    - Space for both types (as well as **`inout`**) is **`deallocated`** on the coprocessor

Data Access Semantics

❖ **Data Access Semantics**

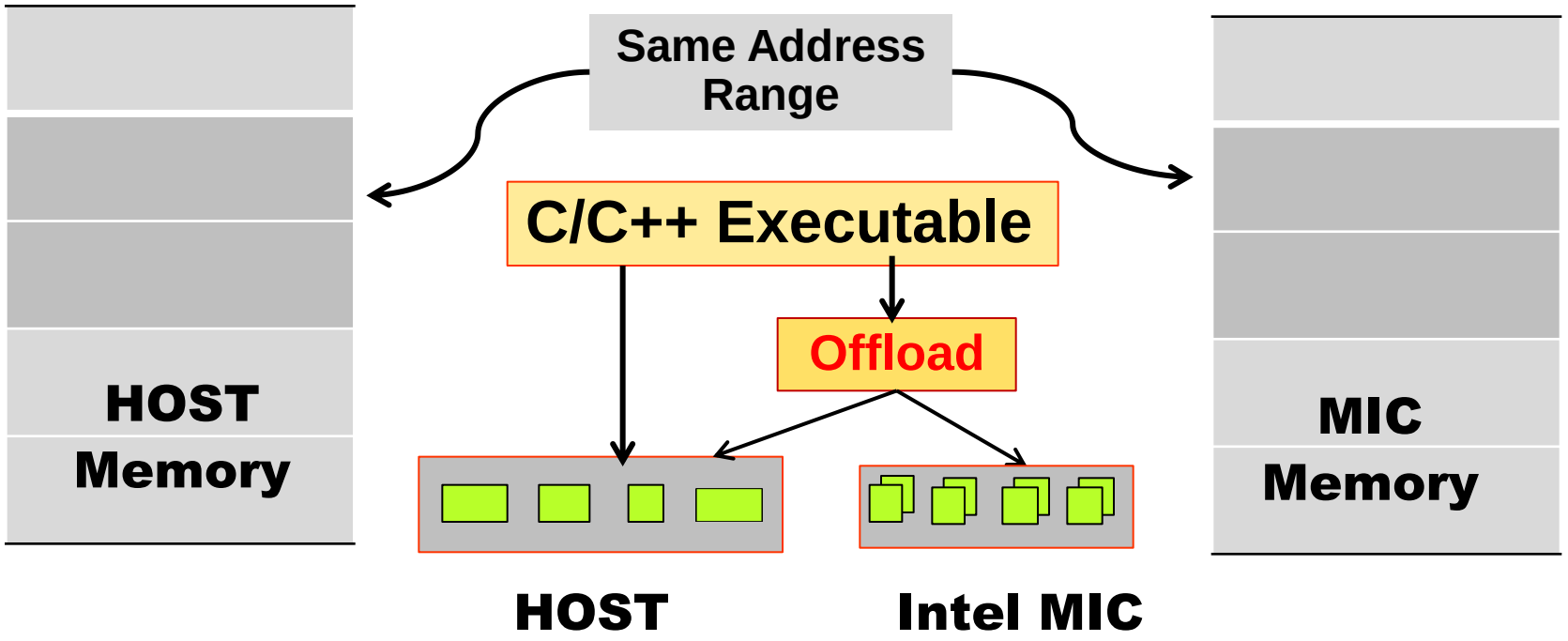
- ➢ Implicit Offloading

❖ Section of memory maintained at the same virtual address on both the host and Intel MIC Architecture coprocessor

❖ Reserving same address range on both devices allows

- Seamless sharing of complex pointer-containing data structures
- Elimination of user marshaling and data management
- Use of simple language extensions to C/C++

Compiler : Offload using Explicit Copies – Data movement



Compiler : _Cilk_shared / _Cilk_offload

- Use this extension when data exchanged between CPU and coprocessor is complex
- Data movement is automatic
- Markup is more extensive but richer class of C/C++ programs can be handled
  - Functions and statically allocated data need _Cilk_shared attribute
  - Dynamically data is allocated using “shared” malloc
- This model is available in C/C++ only

Heterogeneous Compiler : Offload using Implicit Copies

- When “shared” memory is synchronized
  - Automatically done around offloads (so memory is only synchronized on entry to, or exit from, an offload call)
  - Only modified data is transferred between CPU and coprocessor
- Dynamic memory you wish to share must be allocated with special functions: **`_Offload_shared_malloc`**, **`_Offload_shared_aligned_malloc`**, **`_Offload_shared_free`**, **`_Offload_shared_aligned_free`**
- Allows transfer of C++ objects
  - Pointers are no longer an issue when they point to “shared” data
- Well-known methods can be used to synchronize access to shared data and prevent data races within offloaded code
  - E.g., locks, critical sections, etc.
- This model is integrated with the Intel Cilk Plus Parallel Extensions

  Supported in C /C++ Languages Only

Compiler : Data Transfer Overview Compiler

- Two offload transfer models are : **Explicit Copy** and **Implicit Copy**
- **Implicit Copy :**
  - Programmer makes variables that need to be shared between **host** and **mic** card
  - The same variable can be used in both **host** and **coprocessor** code
  - Runtime automatically maintains coherence at the beginning and end of offload statements
  - **Syntax:** keyword extensions based
  - **Example:**

```
_Cilk_shared double foo;

_Offload func(y);
```

Heterogeneous Compiler : Offload using Intel Cilk Plus

- Intel C/C++ Compiler extension with new offloading key words
- Provide the appearance of shared memory using virtual Shared-memory technology

Feature	Example	Description
Offloading a function call	`x = _Cilk_shared _Cilk_offload func(y);`	func can executes on Intel MIC
Offloading asynchronously	`x = _Cilk_spawn _Cilk_offload func(y);`	Non blocking offload
Data available on both sides	`_Cilk_shared int x = 0;`	Allocated in the shared memory area, can be synchronized
Function available on both sides	`int _Clik_shared f(int x) { return x+1}`	The function can execute on either side
Offload a parallel for loop (Requires Cilk on Intel MIC)	`_Cilk_offload _Cilk_for (i = 0; i < N; i++) { a[i] = b[i] + c[i]; }`	Loop executes in parallel on Intel MIC. The loop is implicitly outlined as a function call. (borrow inside the loop disallowed)
Offload array expressions	`_Offload a[:] = b[:] <op> c[:]; _Offload a[:] = elemental_func(b[:]);`	Array operations execute in parallel on Intel MIC.

An Overview of Multi-Core Processors

Conclusions

- An Overview of Xeon-Phi Architectures, Programming on based on Shared Address Space Platforms – Cilk Plus, Performance of Software threading are discussed.

Intel Xeon Phi - Coprocessors : An Overview

Explicit Message Passing - Programming

Part-3
MPI

Intel Xeon-Phi : MPI Programming Model

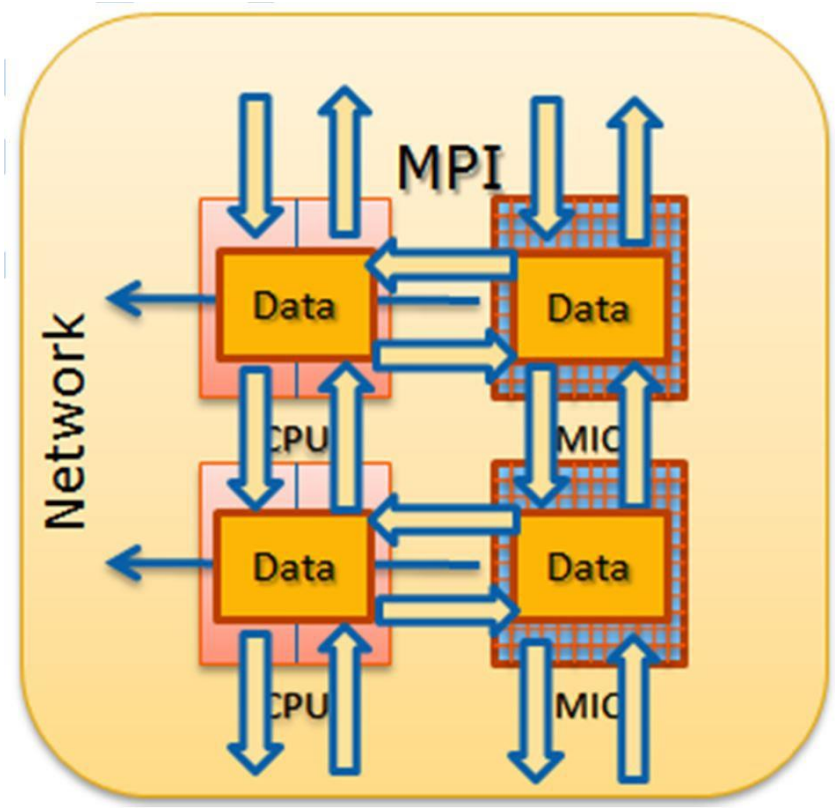
- Intel MPI for the Xeon Phi coprocessors offers various MPI programming models:
  - **Symmetric model :** The MPI ranks reside on both the host and the coprocessor. Most general MPI case.
  - **Coprocessor-only model :** All MPI ranks reside only on the coprocessors
  - **Host-only model All :** MPI ranks reside on the host. The coprocessors can be used by using offload pragmas. (Using MPI calls inside offloaded code is not supported

Intel Xeon-Phi : MPI Programming Model

- Intel MPI for the Xeon Phi coprocessors offers various MPI programming models:
  - **Symmetric model :** The MPI ranks reside on both the host and the coprocessor. Most general MPI case.
  - **Coprocessor-only model :** All MPI ranks reside only on the coprocessors
  - **Host-only model All :** MPI ranks reside on the host. The coprocessors can be used by using offload pragmas. (Using MPI calls inside offloaded code is not supported

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

Symmetric Model



MPI on Host Devices and Co-processors

- The MPI processes reside on both the host and the MIC devices
- This model involves both the host CPUs and the co-processors into the execution of the MPI processes and the related MPI communications.
- Message passing is supported inside the co-processor, inside the host node, and between the co-processor and the host. environment variable
- Most general MPI view of an essentially heterogeneous cluster.

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

Intel Xeon-Phi : MPI Programming Model

❖ **Symmetric model** To build and run an application in **host-only** mode, the following commands have to be executed:

compile the program for the host (offloading is enabled per default

```
mpiicc –mmic -o hello.MIC hello.c
```

launch MPI jobs on the host “ycn-0”,the MPI process will offload code for acceleration

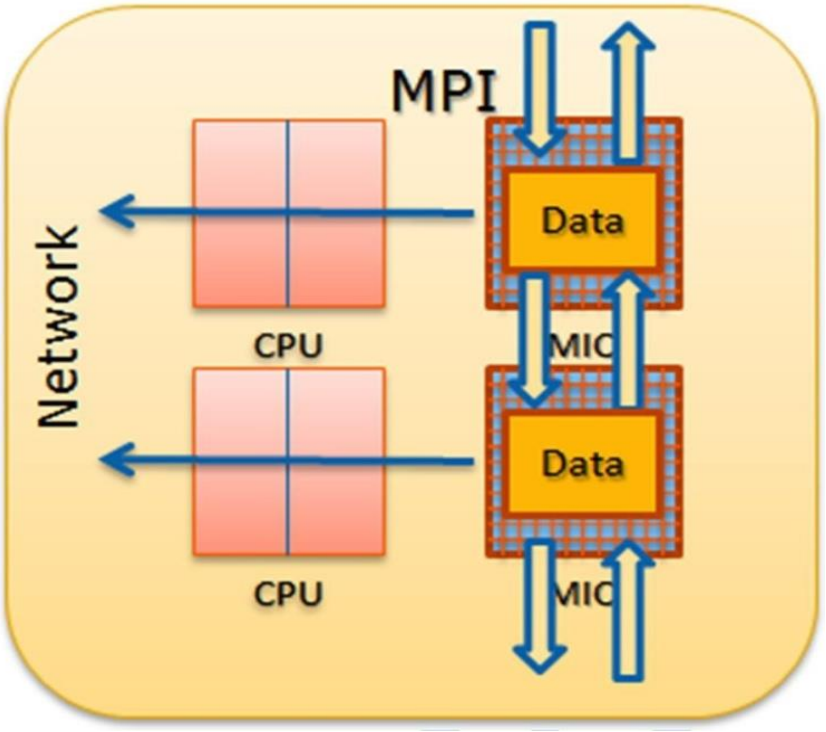
```
mpirun –host ycn-1 -n 1 ./hello
```

Intel Xeon-Phi : MPI Programming Model

- **Coprocessor-only model** To build and run an application in coprocessor-only mode, the following commands have to be executed:

```
# compile the program for the coprocessor (-mmic)
mpiicc -mmic -o hello.MIC hello.c

#copy the executable to the coprocessor
scp hello.MIC mic0:/tmp
#set the I_MPI_MIC variable
export I_MPI_MIC=1
#launch MPI jobs on the coprocessor mic0 from the host
#(alternatively one can login to the coprocessor and
   run mpirun there)
mpirun -host mic0 -n 2 /tmp/hello.MIC
```

Co-processor-only Model (or MPI Native)



MPI on Co-processors

- The MPI processes reside on the MIC co-processor only .
- MPI libraries, the application, and other needed libraries are uploaded to the co-processors.
- An application can be launched from the host or the co-processor.
- This can be seen as a specific case of the symmetric model

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

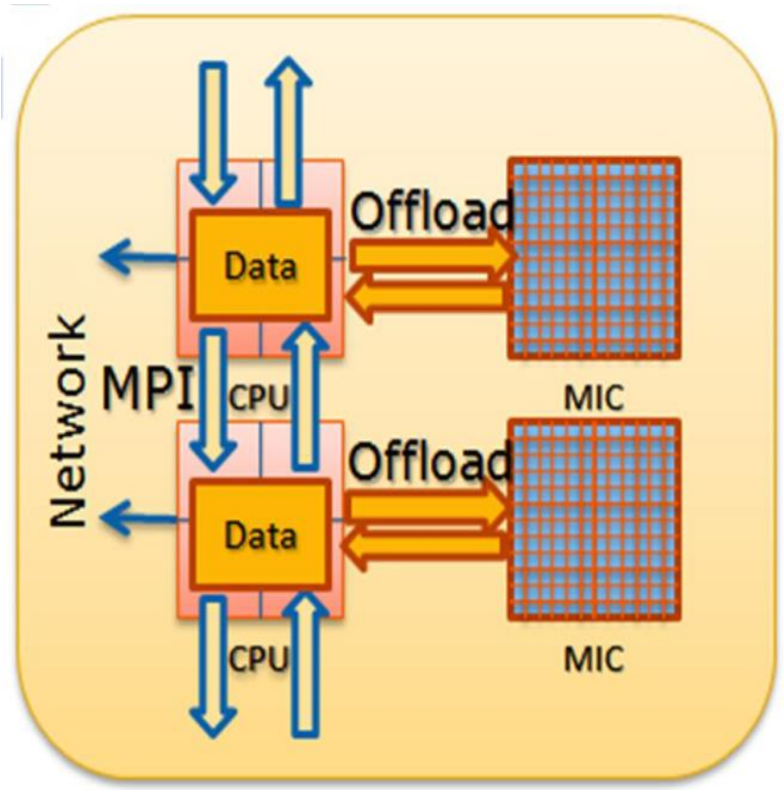
- MPI ranks on the MIC CPU only
- Messages into/out of the MIC CPU c/o host CPUs
- Threading possible

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 Model



MPI on Host Devices with Offload to Co-processors

- This model is characterized by the MPI communications taking place only between the host processors.
- The co-processors are used exclusively thru the offload capabilities of the products like Intel C, C++, and Fortran Compiler for Intel MIC Architecture, Intel Math Kernel Library (MKL), etc.
- This mode of operation is already supported by the Intel MPI Library for Linux OS

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

Intel Xeon-Phi : MPI Programming Model

❖ **Host-only model** To build and run an application in **host-only** mode, the following commands have to be executed:

compile the program for the host (offloading is enabled per default

```
mpiicc -o hello.MIC hello.c
```

launch MPI jobs on the host “ycn-0”,the MPI process will offload code for acceleration

```
mpirun –host ycn-1 -n 1 ./hello
```

Intel Xeon-Phi : MPI Programming Model

Simple way to Launch MPI jobs :

Instead of specifying the hosts and coprocessors via **`-n hostname`** one can also put the names into a **`hostfile`** and launch the jobs via

```
mpirun -f hostfile -n 4 ./hello
```

Note that the executable must have the same name on the hosts and the coprocessors in this case. If one sets **`export`**

```
I_MPI_POSTFIX=.mic
```

the **`.mic`** postfix is automatically added to the executable name by **`mpirun`**, so in the case of the example above test is launched on the host and **`hello.mic`** on the coprocessors.

Intel Xeon-Phi : MPI Programming Model

Simple way to Launch MPI jobs :

```
mpirun -f hostfile -n 4 ./hello

I_MPI_POSTFIX=.mic
```

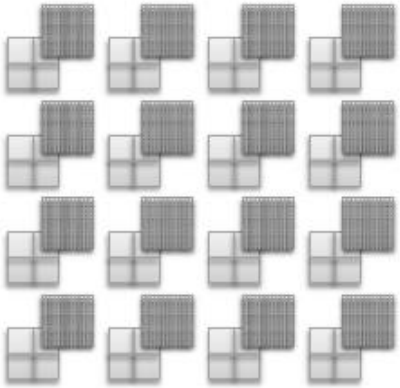
It is also possible to specify a prefix using

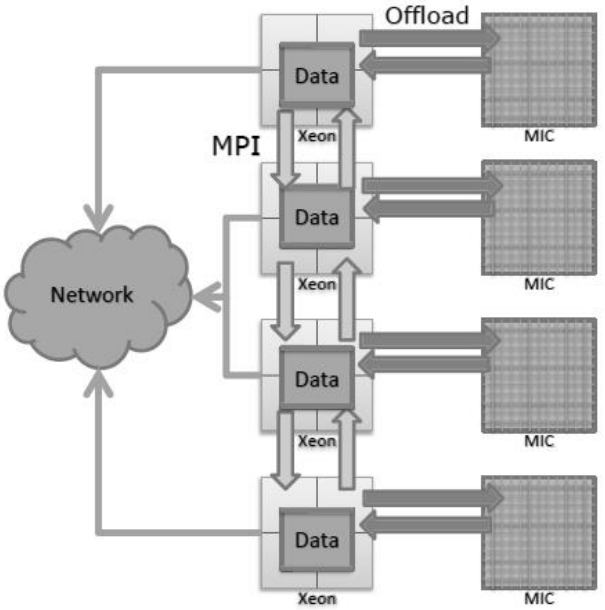
```
export I_MPI_PREFIX=./MIC/
```

In this case `./MIC/hello` will be launched on the coprocessor. This is specially useful if the host and the coprocessors share the same NFS file system

Programming Intel MIC-based Systems MPI+Offload

- MPI ranks on Intel ® Xeon ® processor (only)
- All messages into/out of processors
- Offload models used to accelerate MPI ranks
- Intel CilkTM Plus, Open MP*, Intel Threading Building Blocks, Pthreads* within Intel ®MIC
- Homogenous network of hybrid nodes:

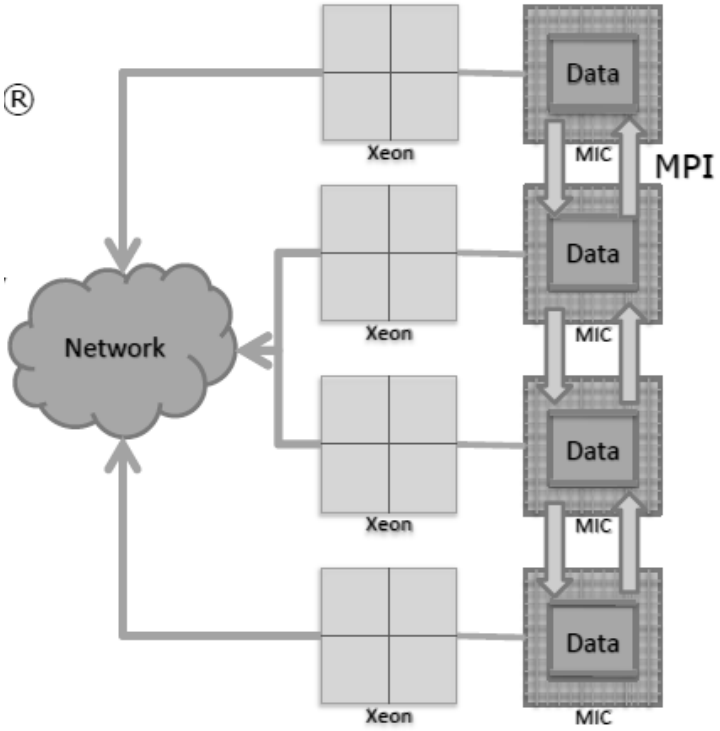


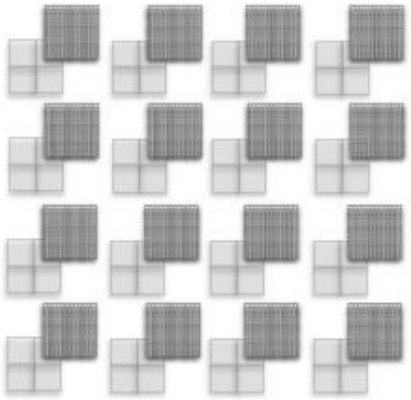


Source : References & Intel Xeon-Phi; http://www.intel.com/

Programming Intel ® MIC-based Systems *Many-core Hosted*

- MPI ranks on Intel ® MIC (only)
- All messages into/out of Intel ® MIC
- Intel ® CilkTM Plus, Open MP*, Intel ® Threading Building Blocks, Pthreads* used directly within MPI processes
- Programmed as homogenous network of many-core

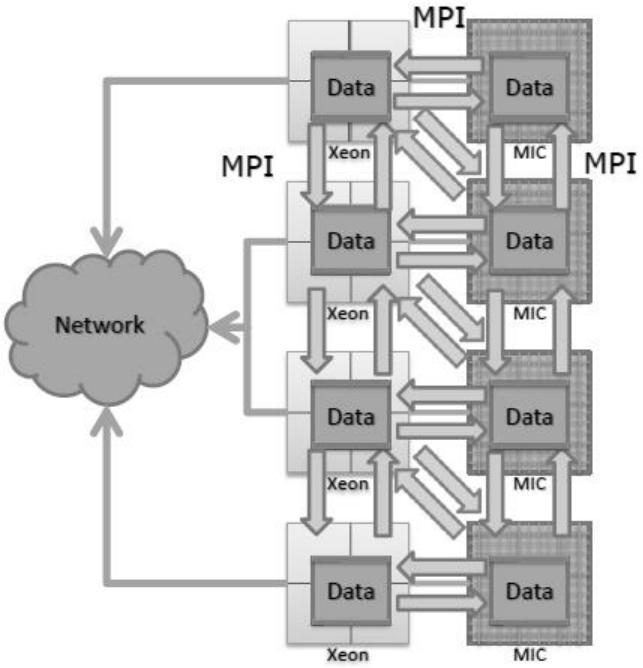


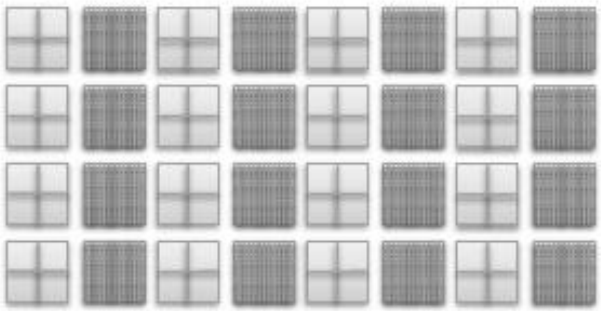


Source : References & Intel Xeon-Phi; http://www.intel.com/

Programming Intel ® MIC-based Systems *Symmetric*

- MPI ranks on Intel ® MIC Intel ® Xeon ® processors
- Messages into/out anu core
- Intel ® CilkTM Plus, Open MP*, Intel ® Threading Building Blocks, Pthreads* used directly within MPI processes
- Programmed as heteregenous network of homogenous





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/

An Overview of Multi-Core Processors

Conclusions

- An Overview of Xeon-Phi Architectures, Programming on based on Shared Address Space Platforms – OpenMP, MPI, Intel TBB, Performance of Software threading are discussed.

Intel Xeon Phi - Coprocessors : An Overview

Shared Address Space Programming –

Part-3

OpenMP 4.0

Programming on Systems with Co-processors - OpenMP 4.0

OpenMP : SIMD Constructs

`simd` construct

Summary

The **`simd`** construct can be applied to a loop to indicate that the loop can be transformed into a SIMD loop (that is, multiple iterations of the loop can be executed concurrently using SIMD instructions).

Syntax

C/C++

The syntax of the **`simd`** construct is as follows:

```
#pragma omp simd [clause[[,] clase] … ] new-line
        for-loops
```

Source : NVIDIA, PGI & References given in the presentation

OpenMP : SIMD Constructs

Syntax

C/C++

The syntax of the **`simd`** construct is as follows:

```
#pragma omp simd [clause[[,] clause] …] new-line
        for-loops
```

where *clause* is one of the following:

`safelen`*(length)*
`linear`(list[:linear-step])
`aligned`(list[:alignment])
`private`(list)
l**astprivate**(list)
reduction(reduction-identifier:list)
collapse(n)

The **`simd`** directive places restrictions on the structure of the associated *for-loops*. Specifically, all associated *for-loops* must have canonical loop form

Source : http:/www.openmp.org; References of OpenMP

OpenMP : `declare simd` Construct

Summary

The **`declare simd`** construct can be applied to a function (C, C++ and Fortran) or a subroutine (Fortran) to enable the creation of one or more versions that can process multiple arguments using SIMD instructions from a single invocation from a SIMD loop. The **`declare simd`** directive is a declarative directive. There may be multiple **`declare simd`** directives for a function (C, C++, Fortran) or subroutine (Fortran 90).

Source : http:/www.openmp.org; References of OpenMP

OpenMP : `declare simd` Construct

Syntax

C/C++

The syntax of the **`declare simd`** construct is as follows:

```
#pragma omp declare simd [clause[[,] clause] …] new-line
/#pragma omp declare simd [clause[[,] clause] …] new-line
[…]
        function definition or declaration
```

where *clause* is one of the following:

- **`simdlen`***(length)*
- **`linear`***(argument-list[:constant-linear-step])*
- **`aligned`***(argument-list[:alignment])*
- **`private`***(argument-list)*
- **`inbranch`**
- **`notinbranch`**

Source : http:/www.openmp.org; References of OpenMP

OpenMP : `Loop SIMD` Constructs

Summary

The loop SIMD construct specifies a loop that can be executed concurrently using SIMD instructions and that those iterations will also be executed in parallel by threads in the team.

Syntax

C/C++

Description

The loop SIMD construct will first distribute the iterations of the associated loop(s) across the implicit tasks of the parallel region in a manner consistent with any clauses that apply to the loop construct. The resulting chunks of iterations will then be converted to a SIMD loop in a manner consistent with any clauses that apply to the **`simd`** construct. The effect of any clause that applies to both constructs is as if it were applied to both constructs separately.

Source : http:/www.openmp.org; References of OpenMP

OpenMP : `Device` Construct

Summary

Create a device data environment for the extent of the region.

Syntax

C/C++

The syntax of the **`target data`** construct is as follows:

```
#pragma omp target data [clause[[,] clase] … ] new-line
        structured-block
```

Where *clause* is one of the following:

`device`(integer-expression)

`map`(*[map-type :] list*))

`if` (*scalar-expression*)

OpenMP `Device` Construct

Binding

The binding task region for a **`target data`** construct is the encountering task. The target region binds to the enclosing parallel or task region.

Description

When a **`target data`** construct is encountered, a new device data environment is created, and the encountering task executes the **`target data`** region. If there is no **`device`** clause, the default device is determined by the *default-device-var* ICV. The new device data environment it constructed form the enclosing device data environment, the data environment of the encountering task and any data-mapping clauses on the construct. When an **`if`** clause is present and the **`if`** clause expression evaluates to false, the device is the host.

Restrictions

A program must not depend on any ordering of the valuations of the clauses of the **`target data`** directive, or any side effects of the evaluations of the clauses.

At most one **`device`** clause can appear on the directive. The **`device`** expression must evaluate to a non-negative integer value.

At most one **`if`** clause can appear on the directive

OpenMP `target` Construct

Summary

Create a device data environment and execute the construct on the same device.

Syntax

C/C++

The syntax of the **`target`** construct is as follows:

```
#pragma omp target data [clause[[,] clase] … ] new-line
      structured-block
```

Where *clause* is one of the following:

`device`(*integer-expression*)

`map`(*[map-type :] list*))

`if` (*scalar-expression*)

Source : http:/www.openmp.org; References of OpenMP

OpenMP `target` Construct

Binding

The binding task region for a **`target`** construct is the encountering task. The target region binds to the enclosing parallel or task region.

Description

When a **`target`** construct provides a superset of the functionality and restrictions provided by the **`target data`** directive. The functionality added to the **`target`** directive is the inclusion of an executable region to be executed by a device. That is, the **`target`** directive is an executable directive. The encountering task waits for the device to complete the target region. When an **`if`** clause is present and the **`if`** clause expression evaluated to *false*, the target region is executed by the host device.

Restrictions

- If a **`target, target update`**, or **`target data`** construct appears within a target region then the behaviour is unspecified.
- The result of an **`omp_set_default_device, omp_get_default_device`**.

OpenMP `target update` Construct

Summary

The **`target update`** directive makes the corresponding list item in the device data environment consistent with their original list items, according to the specified motion clauses. The **`target update`** construct is a stand-alone directive

Syntax

C/C++

The syntax of the **`target`** construct is as follows:

```
#pragma omp target data [clause[[,] clase] … ] new-line
```

Where *motion-clause* is one of the following:

`to`(*list*)

`from`(*list*)

and where clause is motion-clause or one of the of following:

`device`(*integer-expression*)

`from`(*scalar-expression*)

OpenMP `target update` Construct

Binding

The binding task for a **`target update`** construct is the encountering task. The **`target update`** directive is a stand-alone directive.

Description

For each list item in a **`to`** or **`from`** clause there is a corresponding list item and an original list item. If the corresponding list item is not present in the device data environment, the behaviour is unspecified. Otherwise, each corresponding list item in the device data environment has an original list item in the current task's data environment.

For each list item in a **`from`** clause the value of the corresponding list item is assigned to the original list item.

The list items that appear in the to or from clauses may include array sections.

The device is specified in the **`device`** clause. If there is no **`device`** clause, the device is determined by the *default-device-var* ICV. When an **`if`** clause is present and the **`if`** clause expression evaluates to false then no assignments occur.

Restrictions

- A program must bod depend on any ordering of the evaluations of the clauses of the **`target update`** directive, or on any side effects of the evaluations of the clauses.

OpenMP `Declare target` Directive

Summary

The **`declare target`** directive specifies that variables, functions (C, C++ and Fortran), and subroutines (Fortran) are mapped to a device. The **`declare target`** directive is a declarative directive.

Syntax

C/C++

The syntax of the **`declare target`** directive is as follows:

```
#pragma omp declare target new-line
declarations-definition-seq
#pragma omp end declare target new-line
```

Source : http:/www.openmp.org; References of OpenMP

Description

OpenMP `Declare target` Directive

C/C++

Variable and routine declarations that appear between the **`declare target`** and **`end declare target`** directives form an implicit list where each list item is the variable or function name.

C/C++

Fortran

If a declare target does not have an explicit list, then an implicit list of one item is formed from the name of the enclosing subroutine, subprogram, function subprogram or interface body to which it applies.

Fortran

If a list item is a function (C, C++, Fortran) subroutine (Fortran) then a device-specific version of the routine is created that can be called from a target region.

If a list item is a variable then the original variable is mapped to a corresponding variable in the initial deivce data environment for all devices. If the original variable is initialized the corresponding variable in the device data environment is initialized with the same value.

Restrictions

- A threadprivate variable cannot appear in a **`declare target`** directive
- A variable declared in a **`declare target`** directive must have a mappable type.

OpenMP teams Construct

Summary

The **teams** construct creates a league of thread teams and the master thread of each team executes the region.

Syntax

The syntax of the **team** construct is as follows:

C/C++

#pragma omp teams*[clause[[,] clase] …] new-line*
structured-block

and where clause is one of the of following:

num_teams(*integer-expression*)

thread_limit(*integer-expression*)
default(***shared*|*none***)
private (list)
firstprivate (list)
shared (list)
reduction (reduction-identifier : list)

OpenMP teams Construct

Binding

The binding thread set for a **teams** region is the encountering thread.

Description

When a thread encounters a teams construct, a league of thread teams is created and the master thread of each thread team executes the teams region.

The number of teams created in implementation defined, but is less than or equal to the value specified in the **num_teams** clause.

The maximum number of threads participating in the contention group that each team initiates is implementation defined, but is less than or equal to the value specified in the **thread_limit** clause.

Once the **teams** are created, the number of teams remains constant for the duration of the teams region.

Within a teams region, team numbers uniquely identify each team. Team numbers are consecutive whole numbers ranging from zero to one less than the number of teams. A thread may obtain its own team number by a call to the **omp_get_team_num** library routine.

The threads other than the master thread do not begin execution until the master thread encounters a **parallel** region.

After the teams have completed execution of the **teams** region, the encountering thread resumes execution of the enclosing **target** region. There is no implicit barrier at the end of a **teams** construct.

OpenMP `distribute` Construct

Summary

The **`distribute`** construct specifies that the iterations of one or more loops will be executed by the thread teams in the context of their implicit tasks. The iterations are distributed across the master threads of all teams that execute the **`teams`** region to which the **`distribute`** region binds..

Source : http:/www.openmp.org; References of OpenMP

OpenMP `distribute` Construct

Syntax

C/C++

The syntax of the **`distribute`** construct is as follows:

```
#pragma omp distribute [clause[[,] clase] … ] new-line
      for-loops
```

Where *clause* is one of the following:

`private` (*list*)
`firstprivate` (*list*)
`collaspe` (*n*)
`dist_schedule` (*kind[, chunk_size]*)

All associated for – loops must have the canonical form described in Section 2.6 on page 51

OpenMP `distribute` Construct

Binding

The binding thread set for a **`distribute`** region is the set of master threads created by a **`teams`** construct. A **`distribute`** region binds to the innermost enclosing **`teams`** region. Only the threads executing the binding **`teams`** region participate in the execution of the loop iterations.

Description

The **`distribute`** construct is associated with a loop nest consisting of one or more loops that follow the directive.

There is no implicit barrier at the end of a **`distribute`** construct.

The **`collapse`** clause may be used to specify how many loops are associated with the **`distribute`** construct. The parameter of the **`collapse`** clause must be a constant positive integer expression. If no **`collapse`** clause is present, the only loop that is associated with the **`distribute`** construct is the one that immediately follows the **`distribute`** construct.

OpenMP `distribute` Construct

If more that one loop is associated with the **`distribute`** construct, then the iteration of all associated loops are collapsed into one larger iteration space. The sequential execution of the iterations in all associated loops determines the order of the iterations in the collapsed iteration space.

If **`dist_schedule`** is specified kind must be static. If specified, iterations are divided into chunks of size *chunk_size*, chunks are assigned to the teams of the league in a round-robin fashion in the order of the team number. When no *chunk_size* is specified, the iteration space is divided into chunks that are approximately equal in size, and at most one chunk is distributed to each team of the league. Note that the size of the chunks is unspecified in this case.

When no **`dist_schedule`** clause is specified, the schedule is implementation defined.

OpenMP `distribute simd` Construct

Summary

The **`distribute simd`** construct specifies a loop that will be distributed across the master threads of the teams region and executed concurrently using SIMD instructions.

Syntax

The syntax of the **`team`** construct is as follows:

C/C++

```
#pragma omp distribute simd[clause[[,] clase] … ]
      for-loops
```

Where clause can be any of the clauses accepted by the **`distribute`** or **`simd`** directives with identical meaning and restrictions:

C/C++

Fortran

```
!$omp distribute simd[clause[[,] clase] … ]
      do-loops
[ !$omp and distribute simd ]
```

OpenMP `distribute simd` Construct

Description

The **`distribute simd`** construct will first distribute the iterations of the associated loop(s) according to the semantics of the **`distribute`** construct and any clauses that apply to the distribute construct. The resulting chunks of iterations will then be converted to a SIMD loop in a manner consistent with any clauses that apply to the **`simd`** construct. The effect of any clause that applies to both constructs is as if it were applied to both constructs separately.

Restrictions

The restrictions for the **`distribute`** and **`simd`** constructs apply.

Cross References

- **`simd`** construct, see Section 2.8.1 on page 68
- **`distribute`** construct, see Section 2.9.6 on page 88
- Data attribute clauses

OpenMP Distribute Parallel Loop Construct

Summary

The distribute parallel loop construct specifies a loop that can be executed in parallel by multiple threads that are members of multiple teams.

Syntax

The syntax of the distribute parallel loop construct is as follows:

C/C++

```
#pragma omp distribute parallel for[clause[[,] clause] … ]
      for-loops
```

Where clause can be any of the clauses accepted by the **`distribute`** or parallel loop directives with identical meaning and restrictions:

C/C++

Source : http:/www.openmp.org; References of OpenMP

OpenMP `distribute simd` Construct

Description

The distribute parallel loop construct will first distribute the iterations of the associated loop(s) according to the semantics of the **`distribute`** construct and any clauses that apply to the **`distribute`** construct. The resulting loops will then be distributed across the threads contained within the **`teams`** region to which the **`distribute`** construct binds in a manner consistent with any clauses that apply to the parallel loop construct. The effect of any clause that applies to both the **`distribute`** and parallel loop constructs is as if it were applied to both constructs separately.

Restrictions

The restrictions for the **`distribute`** and parallel loop constructs apply.

Cross References

- **`distribute`** construct, Parellel loop construct, Data attribute clauses

OpenMP `Distribute Parallel Loop SIMD` Construct

Summary

The distribute parallel loop SIMD construct specifies a loop that can be executed in concurrently using SIMD instruction in parallel by multiple threads that are members of multiple teams.

Syntax

The syntax of the distribute parallel loop SIMD construct is as follows:

C/C++

```
#pragma omp distribute parallel for simd [clause[[,] clause] … ]
    for-loops
```

Where *clause* can be any of the clauses accepted by the **`distribute`** or parallel loop SIMD directives with identical meaning and restrictions:

C/C++

OpenMP `Distribute Parallel Loop SIMD` Construct

Description

The distribute parallel loop construct will first distribute the iterations of the associated loop(s) according to the semantics of the **`distribute`** construct and any clauses that apply to the **`distribute`** construct. The resulting loops will then be distributed across the threads contained within the **`teams`** region to which the **`distribute`** construct binds in a manner consistent with any clauses that apply to the parallel loop construct. The resulting chunks of iterations will then be converted to a SIMD loop in a manner consistent with any clauses that apply to the **`simd`** construct. The effect of any clause that applies to both the **`distribute`** and parallel loop SIMD constructs is as if it were applied to both constructs separately.

Source : http:/www.openmp.org; References of OpenMP

OpenMP Combined Construct

Description

Combined constructs are shortcuts for specifying one construct immediately nested inside another construct. The semantics of the combined constructs are identical to that of explicitly specifying the first construct containing one instance of the second construct and no other statements.

Some combined constructs have clauses that are permitted on both constructs that were combined. Where specified, the effect is as if applying the clauses to one or both constructs. If not specified and applying the clause to one to one construct would result in different program behaviour than applying the clause to the other construct then the program's behaviour is unspecified.

Source : NVIDIA, PGI & References given in the presentation

OpenMP `Parallel for` Loop Construct

Summary

The parallel loop construct is a shortcut for specifying a **`parallel`** construct containing one or more associated loops and no other statements.

Syntax

The syntax of the parallel loop construct is as follows:

C/C++

```
#pragma omp parallel for [clause[[,] clause] … ] new-line
        for-loops
```

where *clause* can be any of the clauses accepted by the **`parallel`** or **`for`** directives, except the **`nowait`** clause, with identical meanings and restrictions.

C/C++

Fortran

OpenMP `parallel section` Construct

Summary

The parallel sections construct is a shortcut for specifying a **`parallel`** construct containing one **`sections`** construct and no other statements.

Syntax

The syntax of the **`parallel sections`** construct is as follows:

C/C++

```
#pragma omp parallel for [clause[[,] clause] … ] new-line
      {
       [#pragma omp section new-line]
           structured-block
       [#pragma omp section new-line
           structured-block]
      …
      }
```

where *clause* can be any of the clauses accepted by the **`parallel`** or **`sections`** directives, except the **`nowait`** clause, with identical meanings and restrictions.

C/C++

OpenMP : `parallel workshare` Construct

Syntax

The syntax of the **`parallel workshare`** construct is as follows:

```
!$omp parallel for [clause[[,] clause]
        structured-block
!$omp end parallel workshare
```

where *clause* can be any of the clauses accepted by the **`parallel`** directives, with identical meanings and restrictions. **`nowait`** may not be specified on an **`end parallel workshare`** directive.

Description

The semantics are identical to explicitly specifying a **`parallel`** directive immediately followed by a **`workshare`** directive, and an **`end workshare`** directive immediately followed by an **`end parallel`** directive.

OpenMP `Parallel for SIMD` Loop Construct

Summary

The parallel loop SIMD construct is a shortcut for specifying a **`parallel`** construct containing one loop SIMD construct and no other statement.

Syntax

C/C++

```
#pragma omp parallel for simd [clause[[,] clause] … ] new-line
        for-loops
```

where *clause* can be any of the clauses accepted by the **`parallel`**, **`for`** or **`simd`** directives, except the **`nowait`** clause, with identical meanings and restrictions.

C/C++

Fortran

OpenMP `Parallel for SIMD` Loop Construct

Summary

The semantics of the parallel loop SIMD construct are identical to explicitly specifying a **`parallel`** directive immediately followed by a loop SIMD directive. The effect of any clause that applies to both constructs is as if it were applied to the loop SIMD construct and not to the **`parallel`** construct.

Source : NVIDIA, PGI & References given in the presentation

OpenMP `target teams` Construct

Summary

The target teams construct is a shortcut for specifying a **`target`** construct containing a **`teams`** construct

Syntax

The syntax of the target teams construct is as follows:

C/C++

```
#pragma omp parallel for [clause[[,] clause] … ]
      structured-block
```

where *clause* can be any of the clauses accepted by the **`target`** or **`teams`** directives with identical meanings and restrictions

C/C++

OpenMP `teams distribute` Construct

Summary

The **`teams distribute`** construct is a shortcut for specifying a **`team`** construct containing a **`distribute`** construct

Syntax

The syntax of the **`teams distribute`** construct is as follows:

C/C++

```
#pragma omp team distribute [clause[[,] clause] … ]
        for-loops
```

where *clause* can be any of the clauses accepted by the **`teams`** or **`distribute`** directives with identical meanings and restrictions

C/C++

OpenMP `teams distribute simd` Construct

Summary

The **`teams distribute simd`** construct is a shortcut for specifying a **`teams`** construct containing a **`distribute simd`** construct

Syntax

The syntax of the **`teams distribute simd`** construct is as follows:

C/C++

```
#pragma omp team distribute [clause[[,] clause] … ]
        for-loops
```

where *clause* can be any of the clauses accepted by the **`teams`** or **`distribute simd`** directives with identical meanings and restrictions

C/C++

Fortran

```
!$omp teams distribute simd [clause[[,] clause] … ]
        for-loops
[!$ompand end teams distribute simd]
```

OpenMP `teams distribute simd` Construct

`!$omp teams distribute simd` *[clause[[,] clause] …]*
do-loops
*[***`!$ompand end teams distribute simd`***]*

where *clause* can be any of the clauses accepted by the **`teams`** or **`distribute simd`** directive with identical meanings and restrictions.

If an **`end teams distribute`** directive is not specified, an **`end teams distribute`** directive is assumed at the end of the *do-loops*.

Fortran

Description

The semantics are identical to explicitly specifying a **`teams`** directive immediately followed by a **`distribute simd`** directive. Some clauses are permitted on both constructs.

Source : http:/www.openmp.org; References of OpenMP

OpenMP `target teams distribute` Construct

Summary

The **`target teams distribute`** construct is a shortcut for specifying a **`target`** construct containing a **`teams distribute`** construct

Syntax

The syntax of the **`target teams distribute`** construct is as follows:

C/C++

```
#pragma omp target team distribute [clause[[,] clause] … ]
      for-loops
```

where *clause* can be any of the clauses accepted by the **`target`** or **`team distribute`** directives with identical meanings and restrictions

C/C++

OpenMP `target teams distribute simd` Construct

Summary

The **`target teams distribute`** construct is a shortcut for specifying a **`target`** construct containing a **`teams distribute`** construct

Syntax

The syntax of the **`target teams distribute`** construct is as follows:

C/C++

```
#pragma omp target teams distribute simd [clause[[,] clause] … ]
      for-loops
```

where *clause* can be any of the clauses accepted by the **`target`** or **`team distribute simd`** directives with identical meanings and restrictions

C/C++

OpenMP `teams distribute parallel for` Construct

Summary

The teams distribute parallel loop construct is a shortcut for specifying a **`teams`** construct containing a distribute parallel loop construct.

Syntax

The syntax of the teams distribute parallel loop construct is as follows:

C/C++

```
#pragma omp teams distribute parallel for [clause[[,] clause] … ]
      for-loops
```

where *clause* can be any of the clauses accepted by the **`teams`** or **`distribute parallel for`** directives with identical meanings and restrictions

C/C++

Source : http:/www.openmp.org; References of OpenMP

OpenMP : Target Teams Distribute Parallel Loop Construct

Summary

The target teams distribute parallel loop construct is a shortcut for specifying a **`target`** construct containing a teams distribute parallel loop construct.

Syntax

The syntax of the target teams distribute parallel loop construct is as follows:

C/C++

```
#pragma omp target teams distribute parallel for [clause[[,] clause] … ]
    for-loops
```

where *clause* can be any of the clauses accepted by the **`target`** or **`teams distribute parallel for`** directives with identical meanings and restrictions

C/C++

Source : http:/www.openmp.org; References of OpenMP

OpenMP : Target Teams Distribute Parallel Loop Construct

Summary

The **`teams distribute`** parallel construct is a shortcut for specifying a **`teams`** construct containing a distribute parallel loop SIMDconstruct

Syntax

The syntax of the **`teams distribute simd`** construct is as follows:

C/C++

```
#pragma omp team distribute [clause[[,] clause] … ]
      for-loops
```

where *clause* can be any of the clauses accepted by the **`teams`** or **`distribute parallel for simd`** directives with identical meanings and restrictions

C/C++

Fortran

```
!$omp teams distribute parallel do simd [clause[[,] clause] … ]
      for-loops
[!$omp and end teams distribute parallel do simd]
```

OpenMP : Target Teams Distribute Parallel Loop Construct

Summary

The **`teams distribute`** parallel construct is a shortcut for specifying a **`teams`** construct containing a distribute parallel loop SIMD construct

Syntax

The syntax of the **`teams distribute simd`** construct is as follows:

C/C++

```
#pragma omp team distribute parallel for simd [clause[[,] clause] … ]
        for-loops
```

where *clause* can be any of the clauses accepted by the **`teams`** or **`distribute parallel for simd`** directives with identical meanings and restrictions

C/C++

Source : http:/www.openmp.org; References of OpenMP

OpenMP:Target Teams Distribute Parallel Loop SIMD Construct

Summary

The target **`teams`** distribute parallel loop SIMD construct is a shortcut for specifying a **`target`** construct containing a teams distribute parallel loop SIMD construct.

Syntax

The syntax of the target teams distribute parallel loop SIMD construct is as follows:

C/C++

```
#pragma omp team distribute [clause[[,] clause] … ]
        for-loops
```

where *clause* can be any of the clauses accepted by the **`target`** or **`teams distribute parallel for simd`** directives with identical meanings and restrictions

C/C++

OpenMP Tasking Construct

Summary

The **`teams`** construct defines an explicit task.

Syntax

The syntax of the target teams distribute parallel loop SIMD construct is as follows:

C/C++

```
#pragma omp task [clause[[,] clause] … ] new-line
        structured-block
```

where *clause* is one of the following:

`if` (*scalar-expression*)
`final` (*scalar-expression*)
`untied`
`default`(**`shared | none`**)
`mergeable`
`private` (*list*)
`firstprivate` (*list*)
`shared` (*list*)
`depend` (*dependence-type : list*)

OpenMP :Tasking Construct

Description

The encountering thread may immediately execute the task, or defer its execution. In the latter case, any thread in the team may be assigned the task. Completion of the task can be guaranteed using task synchronization constructs. A **`task`** construct may be nested inside an outer task, but the **`task`** region of the inner task is not a part of the **`task`** region of the outer task.

When an **`if`** clause is present on a **`task`** construct, and the **`if`** clause expression evaluates to *false*, an undeferred task is generated, and the encountering thread must suspend the current task region, for which execution cannot be resumed until the generated task is completed. Note that the use of a variable in an **`if`** clause expression of a **`task`** construct causes an implicit reference to the variable in all enclosing constructs.

When a **`final`** clause is present on a **`task`** construct and the **`final`** clause expression evaluates to true, the generated task will be a final task. All **`task`** constructs encountered during execution of a final task will generate final and included tasks. Note that the use of a variable in a **`final`** clause expression of a **`task`** construct cause an implicit reference to the variable in all enclosing constructs.

OpenMP : `depend` Clause

Summary

The **`depend`** clause enforces additional constraints on the scheduling of tasks. These constraints establish dependences only between sibling tasks. The clause consists of a *dependence-type* with one or more list items.

Syntax

The syntax of the target teams distribute parallel loop SIMD construct is as follows:

```
depend( dependence-type : list )
```

Description

Task dependences are derived from the dependence-type of a **`depend`** clause and its list items, where *dependence-type* is one of the following:

OpenMP : `depend` Clause

Description

Task dependences are derived from the dependence-type of a **`depend`** clause and its list items, where *dependence-type* is one of the following:

The **`in`** *dependence-type*. The generated task will be a dependent task of all previously generated sibling tasks that reference at least one of the list items in an **`out`** or **`inout`** dependence-type list.

The **`out`** and **`inout`** dependence-types. The generated task will be a dependent task of all previously generated sibling tasks that reference at least one of the list items in an **`in`**, **`out`**, or **`inout`** dependence-type list.

The list items that appear in the **`depend`** clause may include array sections.

Note – The enforced task dependence establishes a synchronization of memory accesses performed by a dependent task with respect to accesses performed by the predecessor tasks. However, it is the responsibility of the programmer to synchronize properly with respect to other concurrent accesses that occur outside of those tasks.

OpenMP : `taskyield` Clause

Summary

The **`taskyield`** construct specifies that the current task can be suspended in favour of execution of a different task. The taskyield construct is a stand-along directive.

Syntax

C/C++

The syntax of the **`taskyield`** construct is as follows:

```
#pragma omp taskyield new-line
```

C/C++

Fortran

The syntax of the **`taskyield`** construct is as follows:

```
!$omp taskyield
```

OpenMP : `task Scheduling` Clause

Whenever a thread reaches a task scheduling point, the implementation may cause it to perform a task switch, beginning or resuming execution of a different task bound to the current team. Task scheduling points are implied as the following locations:

- the point immediately following the generation of an explicit **`task`** region
- in a **`taskyield`** region
- in a **`taskwait`** region
- at the end of a **`taskgroup`** region
- in an implicit and explicit **`barrier`** region
- the point immediately following the generation of a **`target`** region
- at the beginning and end of a **`target data`** region
- in a **`target update`** region

Source : http:/www.openmp.org; References of OpenMP

OpenMP : Data Environment

This section presents a directive and several clauses for controlling the data environment during the execution of **`parallel`**, **`task`**, **`simd`**, and worksharing regions.

- how the data-sharing attributes of variables referenced in **`parallel, task, simd`**, and worksharing regions are determined.
- The **`threadprivate`** directive, which is provided to create threadprivate memory
- Clauses that may be specified on directives to control the data-sharing attributes of variables referenced in **`parallel, task, simd`** or worksharing constructs
- Clauses that may be specified on directives to copy data values from private or threadprivate variables on one thread to the corresponding variables on other threads in the team
- Clauses that may be specified on directives to map variables to devices

OpenMP : `threadprivate` Directive

Summary

The **`threadprivate`** directive specifies that variable are replicated, with each thread having its own copy. The **`threadprivate`** directive is a declarative directive.

Syntax

C/C++

The syntax of the **`threadprivate`** directive is as follows:

```
#pragma omp threadprivate (list) new-line
```

Where list is a comma-separated list of file-scope, namespace-scope, or static block-scope variables that do not have incomplete types.

C/C++

OpenMP : Data-Sharing Attribute Clauses

Several constructs accept clauses that allow a user to control the data-sharing attributes of variables referenced in the construct. Data-sharing attribute clauses apply only to variables for which the names are visible in the construct on which the clause appears.

Not all of the clauses listed in this section are valid on all directives. The set of clauses that is valid on a particular directive is described with the directive.

Most of the clauses accept a comma-separated list of list items (see Section 2.1 on age 26). All list items appearing in a clause must be visible, according toe the scoping rules of the base language. With the exception of the default clause, clauses may be repeated as needed. A list item that specifies a give variable may not appear in more than one clause on the same directive, except that a variable may be specified in both firstprivate and lastprivate clauses.

C/C++

If a variable referenced in a data-sharing attribute clause has a type derived from a template, and there are no other references to that variable in the program, then any behaviour related to that variable is unspecified

C/C++

OpenMP : Data Copying Clauses

This section describes the **`copyin`** clause (allowed on the **`parallel`** directive and combined parallel worksharing directives) and the **`copyprivate`** clause (allowed on the **`single`** directive).

These clauses support the copying of data values from private or threadprivate variables on one implicit task or thread to the corresponding variables on other implicit tasks or threads in the team.

The clauses accept a comma-seperated list of list items (see Section 2.1 on page 26). All list items appearing in a clause must be visible, according to the scoping rules of the base language. Clauses may be repeated as needed, but a list item that specifies a given variable may not appear in more than one clause on the same directive.

Source : http:/www.openmp.org; References of OpenMP

OpenMP : `copypriavate` clause

Summary

The **`copyprivate`** clause provides a mechanism to use a private variable to broadcast a value from the data environment of one implicit task to be data environments of the other implicit tasks belonging to the **`parallel`** region.

To avoid race conditions, concurrent reads or updates of the list item must be synchronized with the update of the list item that occurs as a result of the **`copyprivate`** clause.

Syntax

The syntax of the **`copyprivate`** clause is as follows:

```
copyprivate (list)
```

Description

The effect of the **`copypriate`** clause on the specified list items occurs after the execution of the structured block associated with the single construct (see Section 2.7.3 on page 63), and before any of the threads in the team have left the barrier at the end of the construct.

OpenMP : `map` clause

Summary

The **`map`** clause maps a variable from the current task's data environment to the device data environment associated with the construct.

Syntax

The syntax of the **`copyprivate`** clause is as follows:

```
map (list)
```

Description

The list items that appear in a **`map`** clause may include array sections.

For list items that appear in a **`map`** clause, corresponding new list items are created in the device data environment associated with the construct.

The original and corresponding list items may share storage such that write to either item by one task followed by a read or write of the other item by another task without intervening synchronization can result in data races.

OpenMP : `declare reduction` Directive

Summary

The following section describes the directive for declaring user-defined reductions. The **`declare reduction`** directive declares a reduction-identifier that can be used in **`reduction`** clause. The **`declare reduction`** directive is a declarative directive.

Syntax

C/C++

`#pragma omp declare reduction` *(reduction-identifier : typename-list : combiner) [initializer-clause] new-line*

where:

- *reduction-identifier* is either a base language identifier or one of the following operators **`+, -, *, &, |, ^, &&`** and **`||`**
- *typename-list* in list of type names
- *combiner* is an expression
- *Initializer-clause* is initializer (*initializer-expr*) where i*nitializer-expr*r is omp_priv * *initializer* or *function-name* (*argument-list*)

C

Conclusions

- Discussed An overview of Programming for Multi-Core Systems with Coprocessors OpenMP 4.0

Source : OpenMP Web Sites and References

Intel Xeon Phi - Coprocessors : An Overview

Hybrid Programming

Part-3

Hybrid Programming – MPI & OpenMP

Prog. on Intel Xeon Phi : Hybrid Prog. MPI /OpenMP

- For hybrid OpenMP/MPI programming there are two major approaches:
  - An MPI offload approach, where MPI ranks reside on the host CPU and work is offloaded to the Xeon Phi coprocessor and
  - a symmetric approach in which MPI ranks reside both on the CPU and on the Xeon Phi. Messages into/out and on the Xeon Phi
- An MPI program can be structured using either model

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

Prog. on Intel Xeon Phi : Threading of MPI ranks

- For hybrid OpenMP/MPI applications use the thread safe version of the Intel MPI Library by using the **`-mt_mpi`** compiler driver option.

- A desired process pinning scheme can be set with the **I_MPI_PIN_DOMAIN** environment variable.

- **It is recommended to use the following setting:**

```
$ export I_MPI_PIN_DOMAIN = omp
```

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

Prog. on Intel Xeon Phi : Threading of MPI ranks

```
$ export I_MPI_PIN_DOMAIN = omp
```

- By using this, one sets the process pinning domain size to be OMP_NUM_THREADS.

  In this way, every MPI process is able to create **$OMP_NUM_THREADS** number of threads that will run within the corresponding domain.

Remark : If this variable is not set, each process will create a number of threads per MPI process equal to the number of cores, because it will be treated as a separate domain.

Further, to pin OpenMP threads within a particular domain, one could use the **KMP_AFFINITY** environment variable.

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

Prog. on Intel Xeon Phi : Threading of MPI ranks

❖ **MPI programming models :** Intel MPI for the Xeon Phi coprocessors offers various MPI programming models:

- **Symmetric model :** The MPI ranks reside on both the host and the coprocessor. Most general MPI case.
- **Coprocessor-only model :** All MPI ranks reside only on the coprocessors
- **Host-only model :** All MPI ranks reside on the host. The coprocessors can be used by using offload pragmas. (Using MPI calls inside offloaded code is not supported.)

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

Intel Xeon Phi - Coprocessors : An Overview

Heterogeneous Programming

Part-3

Heterogeneous Programming – OpenCL

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

OpenCL tries to Standardize Parallel Programming

Background & Challenging Objectives :

- OpenGL: Open Graphics Library
  - Widely supported application programming interface (API) for graphics ONLY
- OpenCL: "CL" Stands for Computing Language
  - providing an API library
  - Modifies C and C++ parallel programming
  - New Initiatives for other programming languages(Fortran)

Aim: to standardize general purpose parallel programming for any application

Source : Intel, NVIDIA, Khronos AMD, References

The OpenCL Standard

❖ **Challenging Objectives :**

- OpenCL C is a restricted version of the C99 language with extension appropriate for executing data-parallel code on a variety of heterogeneous devices.

- Aimed for full support for the IEEE 754 formats

- Programming language, well suited to the capabilities of current heterogeneous platforms

Source : Intel, NVIDIA, Khronos AMD, References

The OpenCL Standard

❖ **Challenging Objectives :**

- The model set forth by OpenCL creates portable, vendor- and device-independent programs that are capable of being accelerated on many different platforms.
  - The OpenCL API is C wit h a C++ Wrapper API that is defined in terms of the C-API.
  - There are third-party bindings for many languages, including Java, Python, and .NET
  - The code that executes on an OpenCL device, which in general is not the same device as the host-CPU, is written in the OpenCL C language.

Source : Intel, NVIDIA, Khronos AMD, References

OpenCL Design Requirements

- **Use all computational resources in system**
  - Program GPUs, CPUs and other processors as peers
  - Support both data- and task- parallel compute models
- **Efficient c-based parallel programming model**
  - Abstract the specified of underlying hardware
- **Abstraction is low-level, high-performance but device-portable**
  - Approachable –but primarily targeted at expert developers
  - Ecosystem foundation – no middleware or “convenience” functions
- **Implementation on a range of embedded, desktop, and server systems**
  - HPC desktop, and handheld profiles in on specification
- **Drive future hardware requirements**
  - Floating point precision requirements
  - Application to both consumer and HPC applications

Source : Intel, NVIDIA, Khronos AMD, References

OpenCL Design Requirements

- **Efficient c-based parallel programming model**
  - Abstract the specified of underlying hardware
- **Abstraction is low-level, high-performance but device-portable**
  - Approachable –but primarily targeted at expert developers
  - Ecosystem foundation – no middleware or "convenience" functions

Source : Intel, NVIDIA, Khronos AMD, References

Conceptual Foundations of OpenCL

An Application for a heterogeneous platform must carry out the following steps.

- Discover the completion that make-up the heterogeneous system
- Probe the characteristics of these components, so that the software can adapt to specific features of different hardware elements
- Create the blocks of instructions (Kernels) that will run on the platform

Source : Intel, NVIDIA, Khronos AMD, References

Conceptual Foundations of OpenCL

An Application for a heterogeneous platform must carry out the following steps.

- Set up and manipulate memory objects involved in the computation.
- Execute the kernels in the right order and on the right components of the system
- Collect the final results
  - Above steps are accomplished through a series of APIs inside OpenCL plus a programming environment for the kernels

Source : Intel, NVIDIA, Khronos AMD, References

Conceptual Foundations of OpenCL

An Application for a heterogeneous platform must carry out the following steps.

- Discover the completion that make-up the heterogeneous system
- Probe the characteristics of these components, so that the software can adapt to specific features of different hardware elements
- Create the blocks of instructions (Kernels) that will run on the platform

Source : Intel, NVIDIA, Khronos AMD, References

Conceptual Foundations of OpenCL

An Application for a heterogeneous platform must carry out the following steps.

- Set up and manipulate memory objects involved in the computation.
- Execute the kernels in the right order and on the right components of the system
- Collect the final results
  - Above steps are accomplished through a series of APIs inside OpenCL plus a programming environment for the kernels

Source : Intel, NVIDIA, Khronos AMD, References

The OpenCL Specification – Models

- The OpenCL specification is defined in four parts, called models, that can be summarized as follows.
  - Platform Model
  - Execution Model
  - Memory Model
  - Programming Model

Source : NVIDIA, Khronos AMD, References

The OpenCL Specification – Models

❖ OpenCL Software Stack

- **Platform Layer**
  - ➢ Query and select computer devices in the system
  - ➢ Initialize a compute device(s)
  - ➢ Create compute contexts and work-queues
- **Runtime**
  - ➢ Resource management
  - ➢ Execute compute kernels
- **Compiler**
  - ➢ A subset of ISO C99 with appropriate language additions
  - ➢ Compile and build compute program executable
  - ➢ Online or offline

Source : Intel, NVIDIA, Khronos AMD, References

The OpenCL Specification – Models

- The OpenCL specification is defined in four parts, called models, that can be summarized as follows.

  - Platform Model
    - High Level description of the heterogeneous system

  - Execution Model
    - An abstract representation of how stream of instructions execute on the heterogeneous system

Source : Intel, NVIDIA, Khronos AMD, References

The OpenCL Specification – Models

- The OpenCL specification is defined in four parts, called models, that can be summarized as follows.
  - Memory Models
    - The Collection of memory regions within OpenCL and how they interact during at OpenCL computation
  - Programming Model
    - The high-level abstractions a programmer uses when designing algorithms to implement an application

Source : Intel, VIDIA, Khronos AMD, References

The OpenCL Specification

❖ Platform model :

- Specifies that there is one processor coordinating the execution (***the host***) and one or more processors capable of executing OpenCL C Code (***the devices***).
- It defines an abstract hardware model that is used by programmers when writing OpenCL functions (Called ***Kernels***) that execute on the devices.
- The platform model defines the relation between the host an device.
  - i.e., OpenCL implementation executing on a host x86 GPU, which is using a GPU device as an accelerator

Source : Intel, NVIDIA, Khronos AMD, References

The OpenCL Specification

❖ Platform model :

- Platforms can be thought of a vendor – specific implementations of the OpenCL API.
- The platform model also presents an abstract device architecture that programmers target writing OpenCL C code.
- Vendors map this abstraction architecture to the physical hardware.

Source : Intel, NVIDIA, Khronos AMD, References

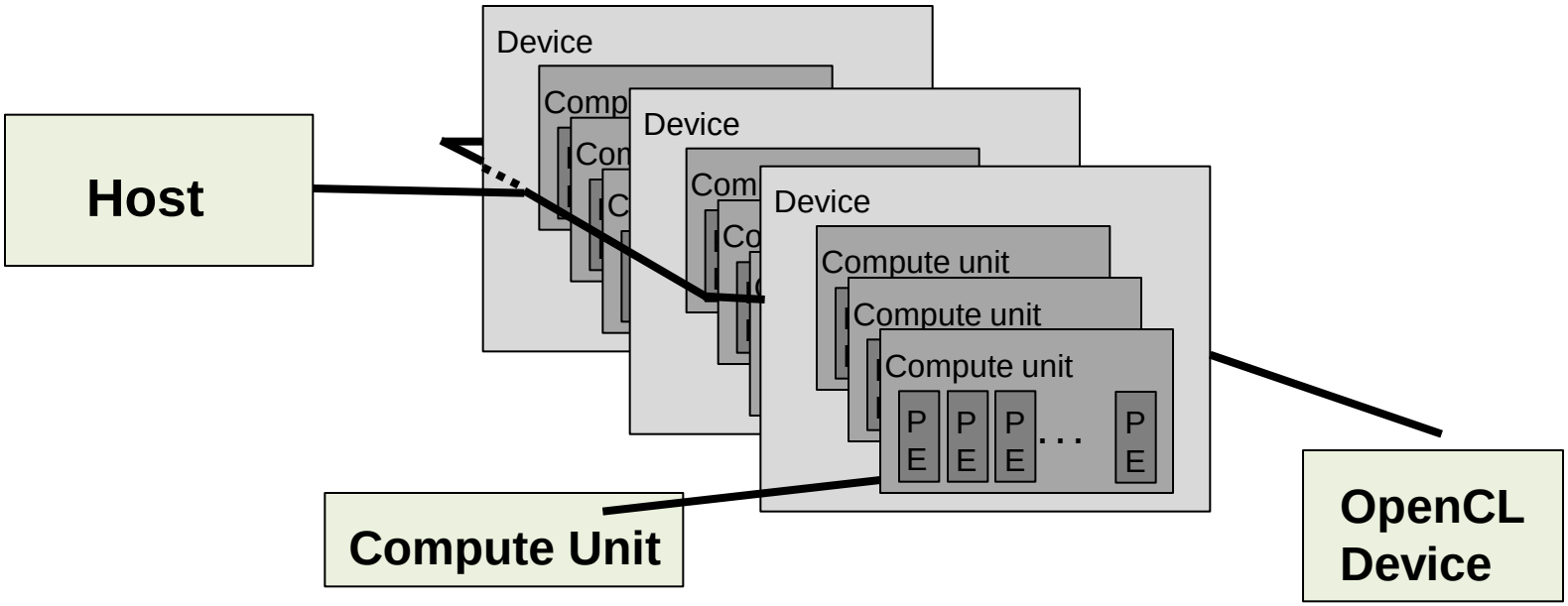
OpenCL PLATFROM AND DEVICES

Host-Device Interaction

- Platform Model
  - Provides an abstract hardware model for devices
  - Present an abstract device architecture that programmers target when writing OpenCL C code.
  - Vendor-specific implementation of the OpenCL API.
- Platform Model
  - Defines a device as an array of compute units
    - Compute units are further divided into processing elements
    - OpenCL device schedule execution of instructions.

Source : Intel, NVIDIA, Khronos AMD, References

OpenCL Platform Model

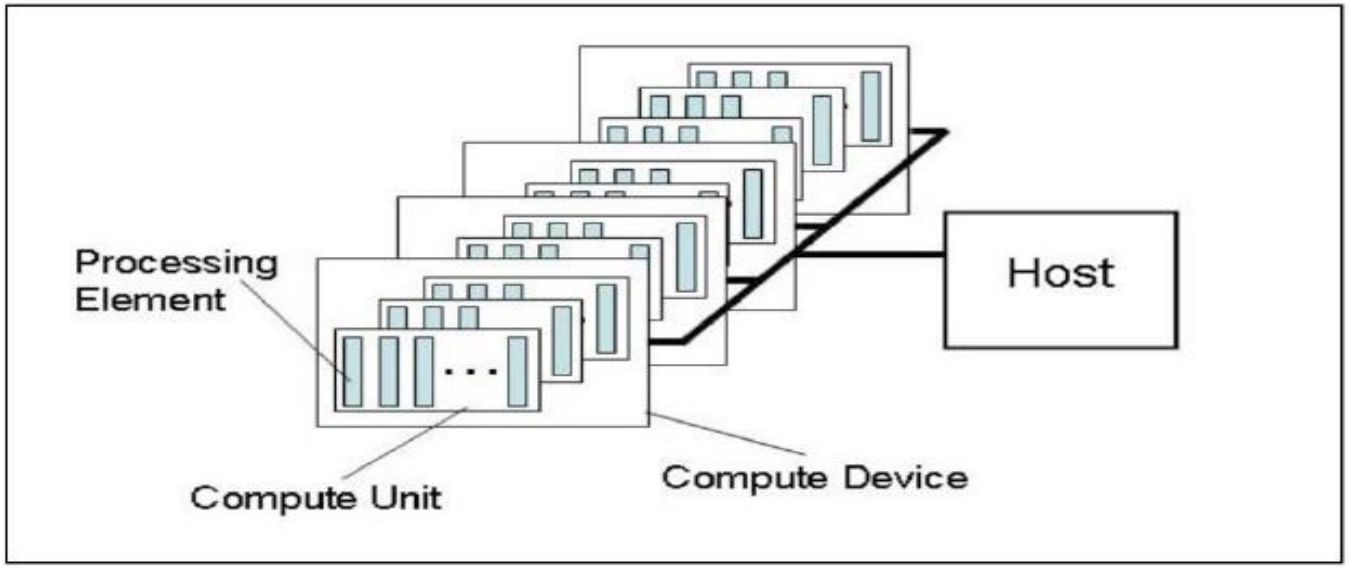


The platform model defines an abstract architecture for devices.

- The host is connected to one or more devices
- Device is where the stream of instructions (or kernels) execute (an OpenCL device is often referred to as a **compute device**
- A device can be a CPU, GPU, DSP, or any other processor provided by Hardware and supported by the OpenCL Vendor

Source : Intel, NVIDIA, Khronos AMD, References

OpenCL Platform Model



- One Host + one or more compute Devices
  - Each compute Device is connected to one or more Compute Units.
    - Each compute Unit is further divided into one or more Processing Elements

Source : Intel, NVIDIA, Khronos AMD, References

OpenCL PLATFROM Model

How to discover available platforms for a given system ?

```
cl_int
ClGetPlatformIds(cl_unit num_entries,
                 cl_platform_Id *platforms,
                 cl_unit *num_platforms)
```

❖ Platform Model

- Defines a device as an array of compute units
  - Compute units are further divided into processing elements
  - OpenCL device schedule execution of instructions.

Source : NVIDIA, Khronos AMD, References

OpenCL PLATFORM Model

How to discover available platforms for a given system.

- Application calls **`ClGetPlatformIds() twice`**
  - The **first** call passes an **unsigned int** pointer as the **`num_platforms`** argument and NULL is passes as the **platform** argument.
    - The programmer can then allocate space to hold the platform information.
  - The **second** call, a **`cl_platform_id`** pointer is passed to the implementation with enough space allocated for **`num_entries`** platforms.

Source : NVIDIA, Khronos AMD, References

OpenCL PLATFROM AND DEVICES

After platforms have been discovered, How to determine which implementation (vendor) the platform was defined by ?

The **`ClGetPlatformInfo()`** call determines implementation

The **`clGetDeviceIDs()`** call works very similar to **`ClGetPlatformId()`**

How to use `device_type` argument ?

GPUs : **`cl_DEVICE_TYPE_GPU`**

CPUs : **`cl_DEVICE_TYPE_CPU`**

All devices : **`cl_DEVICE_TYPE_ALL`** & other options

`Cl_GetDeviceinfo()` is called to retrieve information such as name, type, and vendor from each device.

Source : Inttel, NVIDIA, Khronos AMD, References

OpenCL PLATFROM Model

After platforms have been discovered, How to determine which implementation (vendor) the platform was defined by ?

The `clGetDeviceIDs()`

```
cl_int
clGetDeviceIDs(cl_platform_id platform,
               cl_DEVICE_TYPE_GPU device_type,
               cl_unit num_entries,
               cl_device_id *devices,
               cl_uint *num_devices)
```

Source : Intel, NVIDIA, Khronos AMD, References

OpenCL PLATFORM Model

How to get printed information about the OpenCL, supported platforms and devices in a system ?

`CLinfo` prorgam in the AMD APP SDK

Uses **`clGetplatforminfo()`** and **`clGetDeviceInfo()`**

Hardware details such as memory size and bas widths are available using the commands

$ **`./CLinfo`** program gives complete information

Source : Intel, NVIDIA, Khronos AMD, References

OpenCL PLATFROM AND DEVICES

```
$ ./CLinfo
```

Number of platforms :	1
Platform Profiles :	FULL_PROFILE
Platform Version :	OpenCL 1.1 AMD SDK –v2.4
Platform Name :	AMD Accelerated Parallel Processing
Platform Vendor :	Advanced Micro Devices, Inc.
Number of Devices :	2
Device Type :	CL_DEVICE_TYPE_GPU
Name :	Cypress
Max Compute Units :	20
Address bits	32

OpenCL PLATFROM AND DEVICES

```
$ ./CLinfo
```

Max Memory Allocation:	268435456
Global Memory size :	1073741824
Constant buffer size :	65536
Local Memory type :	Scratchpad
Local Memory size :	32768
Device endianess :	little
Device Type :	CL_DEVICE_TYPE_CPU
Max Compute units :	16
Name :	AMD Phenom™ 11 X4 945 Processor

Source : NVIDIA, Khronos AMD, References

The OpenCL Specification

❖ **Execution model :**

- Defines
  - How the OpenCL environment is configured on the host
  - How kernels are executed on device
- This includes
  - Setting up an OpenCL context on the host,
  - Providing mechanism for host-device interaction, &
  - defining a concurrency model used for kernel execution on device
  - The host sets up a kernel for the GPU to run and instantiates it with some special degree of parallelism.

Source : NVIDIA, Khronos AMD, References

The OpenCL Execution Model

- **Execution Model**
  - Application consists of **two** distinct parts
  - **The host program**
    - Runs on the host
    - OpenCL does not define the details of how the host progrma works, only how it interacts with objects defined in OpenCL
  - **A Collection of Kernels**
    - The Kernel execute on the OpenCL device

Source : NVIDIA, Khronos AMD, References

OpenCL Implementation Steps

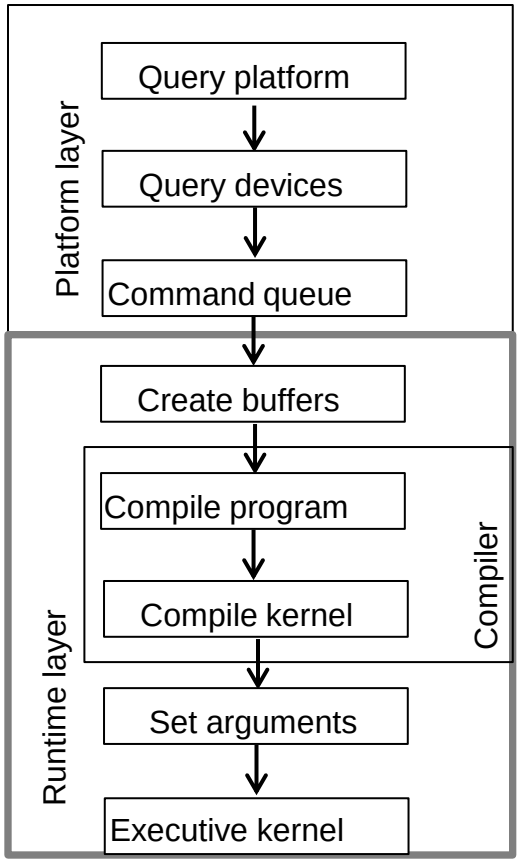


Figure 2 Programming steps to writing a complete OpenCL applications

OpenCL Important Steps – Implementation

Step 1 : Discover and initialize the platforms
Step 2 : Discover and initialize the devices
Step 3 : Create context
Step 4 : Create a command queue
Step 5 : Create device buffers
Step 6 : Write host data device buffers
Step 7 : Create and compile the program
Step 8 : Create the kernel
Step 9 : Set the kernel arguments
Step 10 : Configure the work -items structure
Step 11 : Enqueue the kernel for execution
Step 12 : Read the output buffer back to the host
Step 13 : Release OpenCL resources

OpenCL Important Steps – Implementation

Step 1 : *Discover and initialize the platforms*

Step 2 : *Discover and initialize the devices*

Step 3 : *Create context*

Step 4 : *Create a command queue*

Step 5 : *Create device buffers*

Step 6 : *Write host data device buffers*

The OpenCL specification in four parts, called models.

- **Platform Model**
- **Execution Model**
- **Memory Model**
- **Programming Model**

OpenCL Important Steps – Implementation

Step 7 : *Create and compile the program*

Step 8 : *Create the kernel*

Step 9 : *Set the kernel arguments*

Step 10 : *Configure the work -items structure*

Step 11 : *Enqueue the kernel for execution*

Step 12 : *Read the output buffer back to the host*

Step 13 : *Release OpenCL resources*

The OpenCL specification in four parts, called models.

- **Platform Model**
- **Execution Model**
- **Memory Model**
- **Programming Model**

OpenCL Important Steps – Implementation

- Create an OpenCL context on the first available device
- Create a command –queue on the first available device
- Load a kernel file (hello-world.cl) and build it into a program object
- Create a kernel object for the kernel function hello_world()
- Query the kernel for execution
- Read the results of the kernel back into the result buffer

OpenCL Important Steps – Implementation

```
_kernel void hello_kernel(_global *, *, )
{
     int gid = get_global_id(0);
    ……
 }

int main (int argc,  char** argv)
{
// Create an OpencL context on first available platform

// Create an command-queue on the first device
//  available on the created context
```

The OpenCL Execution Model

- **Execution Model - Kernels**
  - **A Collection of Kernels**
    - Execute on the OpenCL device
    - Do the real work of an OpenCL application
    - Simple functions transform **input** memory objects into **output** memory objects

Execution Model - Kernels

  - **OpenCL defines two types of Kernels**
    - **OpenCL** Kernels & **Native** Kernels

Source : Khronous, & References

The OpenCL Execution Model

❖ **Execution Model : Defines how the kernels execute**

➢ **Several Steps Exist.**

- **FIRST :** How an individual kernel runs on an OpenCL device ?
- **Second:** How the host defines the **context** for kenrel execution
- **THIRD:** How the kernels are enqueued for execution

Source : Intel, NVIDIA, Khronos AMD, References

The OpenCL Execution Model

❖ Execution Model - Kernels

➢ OpenCL Kernels

- Written in OpenCL C programming language and compiled with the OpenCL Compiler
- All OpenCL implementations must support OpenCL Kernels

➢ Native Kernels

- Functions created outside of **OpenCL** and accessed within **OpenCL** through a function pointer. (An Optional functionality within in **OpenCL** exist)

Source : NVIDIA, Khronos AMD, References

The OpenCL Execution Model

- The OpenCL Execution Environment defines the following how the kernel execute
  - Contexts
  - Command Queues
  - Events
  - Memory Objects (Buffers -large array /images
    - Buffers (allocate buffer & return memory object)
    - Image (2D & 3D)
  - Flush & Finish

Source : Intel NVIDIA, Khronos AMD, References

The OpenCL on Intel Xeon Phi

Mapping :OpenCL constructs to Intel Xeon Phi coprocessor

- Conceptually, at initialization time, the OpenCL driver creates 240 SW threads and pins them to the HW threads (for a 60-core configuration).
- Then, following a **`clEnqueueNDRange()`** call, the driver schedules the work groups (WG) of the current NDRange on the 240 threads.
- A WG is the smallest task being scheduled on the threads. So calling **`clEnqueueNDRange()`** with less than 240 WGs, leaves the coprocessor underutilized

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

Mapping :OpenCL constructs to Intel Xeon Phi coprocessor

- The OpenCL compiler implicitly vectorizes the WG routine based on dimension zero loop, i.e., the dimension zero loop is unrolled by the vector size.

```
__Kernel ABC(…)
for(int i = 0; i < get_local_size(2); i++)
for(int j = 0; j < get_local_size(1); j++)
for(int k = 0; k < get_local_size(0); k += VECTOR_SIZE)
   Vector_Kernel_Body;
```

The vector size of Intel Xeon Phi coprocessor is 16,

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

The OpenCL on Intel Xeon Phi

Mapping :OpenCL constructs to Intel Xeon Phi coprocessor

- While the OpenCL specification provides various ways to express parallelism and concurrency, some of them will not map well to Intel Xeon Phi coprocessor. Most importantly, design your application to exploit its parallelism

Multi-threading

- To get good utilization of the 240 HW threads, it's best to have more than 1000 WGs per NDRange. Having 180–240 WGs per NDRange will provide basic threads utilization; however, the execution may suffer from poor load-balancing and high invocation overhead.

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

The OpenCL on Intel Xeon Phi

Mapping : OpenCL constructs to Intel Xeon Phi coprocessor Multi-threading

- **Recommendation**: Have at least 1000 WGs per NDRange to optimally utilize the Intel Xeon Phi coprocessor HW threads. Applications with NDRange of 100 WGs or less will suffer from serious under-utilization of threads

- Single WG execution duration also impacts the threading efficiency. Lightweight WGs are also not recommended, as these may suffer from relatively high overheads.

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

The OpenCL on Intel Xeon Phi

Mapping : OpenCL constructs to Intel Xeon Phi coprocessor

Vectorization :

- OpenCL on Intel Xeon Phi coprocessor includes an implicit vectorization module. The OpenCL compiler automatically vectorizes the implicit WG loop over the work items in dimension zero (see example above).
- The vectorization width is currently 16, regardless of the data type used in the kernel. In future implementations, we may vectorize even 32 elements. As OpenCL work items are guaranteed to be independent, the OpenCL vectorizer needs no feasibility analysis to apply vectorization.
- Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

The OpenCL on Intel Xeon Phi

Mapping : OpenCL constructs to Intel Xeon Phi coprocessor

Vectorization :

- Note that the vectorized kernel is only used if the local size of dimension zero is greater than or equal to 16. Otherwise, the OpenCL runtime runs scalar kernel for each of the work items. If the WG size at dimension zero is not divisible by 16, then the end of the WG needs to be executed by scalar code. This isn't an issue for large WGs, e.g., 1024 items at dimension zero, but is for WGs of size 31 on dimension zero.

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

The OpenCL on Intel Xeon Phi

Mapping : OpenCL constructs to Intel Xeon Phi coprocessor

Vectorization :

- **Recommendation 1:** Don't manually vectorize kernels, as the OpenCL compiler is going to scalarize your code to prepare it for implicit vectorization.

- **Recommendation 2:** Avoid using a WG size that is not divisible by 32 (16 will work for now).

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

The OpenCL on Intel Xeon Phi

Mapping : OpenCL constructs to Intel Xeon Phi coprocessor

Vectorization :

- **Work-Item-ID non-uniform control flow**
  Understand the difference between uniform and nonuniform control flow in the context of vectorization

- **Data Alignment :** For various reasons, memory access that is vector-size-aligned is faster than unaligned memory access. In the Intel Xeon Phi coprocessor, OpenCL buffers are guaranteed to start on a vector-size-aligned address

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

Mapping : OpenCL constructs to Intel Xeon Phi coprocessor

Vectorization :

- **Recommendation 1:** Don't use NDrange offset. If you have to use an offset, then make it a multiple of 32, or at least a multiple of 16.

- **Recommendation 2:** Use local size that is a multiple of 32, or at least of 16..

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

The OpenCL on Intel Xeon Phi

Mapping : OpenCL constructs to Intel Xeon Phi coprocessor

Algorithm Design :

- **Intra WG data reuse**

  Designing your application to maximize the amount of data reuse from the caches is the first memory optimization to apply.

- **Data Data access pattern :** Consecutive data access usually allows the best memory system performance

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

Mapping : OpenCL constructs to Intel Xeon Phi coprocessor

Algorithm Design :

- **Data layout : Pure SOA (Structure-of-Arrays)** data layout results in simple and efficient vector loads and stores

- With **AOS (Array-of-Structures)** data layout, the generated vectorized kernel needs to load and store data via gather and scatter instructions, which are less efficient than simple vector load and store.

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

Mapping : OpenCL constructs to Intel Xeon Phi coprocessor

Algorithm Design :

- **Data Prefetching :** With the Intel Xeon Phi coprocessor being an in-order machine, data prefetching is an essential way to bring data closer to the cores, in parallel with other computations. Loads and stores are executed serially, with parallelism.
- Manual prefetching can be inserted by the programmer into the OpenCL kernel, via the prefetch built-in.
- Automatic SW prefetches to the L1 and L2 are inserted by the OpenCL compiler for data accessed in future iterations,

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

The OpenCL on Intel Xeon Phi

Mapping : OpenCL constructs to Intel Xeon Phi coprocessor

Summary : While designing your OpenCL application for Intel Xeon Phi coprocessor, you should pay careful attention to the following aspects:

- Include enough work groups within each NDRange—a minimum of 1000 is recommended.
- Avoid lightweight work groups. Don't hesitate using the maximum local size allowed (currently 1024). Keep the WG size a multiple of 32.
- Avoid ID(0) dependent control flow. This allows efficient implicit vectorization.

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

The OpenCL on Intel Xeon Phi

Mapping : OpenCL constructs to Intel Xeon Phi coprocessor

Summary : While designing your OpenCL application for Intel Xeon Phi coprocessor, you should pay careful attention to the following aspects:

- Prefer consecutive data access.
- Data layout preferences: AOS for sparse random access; pure SOA or AOSOA(32) otherwise.
- Exploit data reuse through the caches within the WG—tiling/blocking.
- If auto-prefetching didn't kick in, use the PREFETCH built-in to bring the global data to the cache 500–1000 cycles before use.
- Don't use local memory. Avoid using barriers.

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

References

1. Randi J. Rost, OpenGL – shading Language, Second Edition, Addison Wesley 2006
2. GPGPU Reference http://www.gpgpu.org
3. NVIDIA http://www.nvidia.com
4. NVIDIA tesla http://www.nvidia.com/object/tesla_computing_solutions.html
5. RAPIDMIND http://www.rapidmind.net
6. Peak Stream - Parallel Processing (Acquired by Google in 2007) http:/www.google.com
7. guru3d.com http://www.guru3d.com/news/sandra-2009-gets-gpgpu-support/
   ATI & AMD http://ati.amd.com/products/radeon9600/radeon9600pro/index.html
8. AMD http:www.amd.com
9. AMD Stream Processors http://ati.amd.com/products/streamprocessor/specs.html
10. RAPIDMIND & AMD http://www.rapidmind.net/News-Aug4-08-SIGGRAPH.php
11. General-purpose computing on graphics processing units (GPGPU)
    http://en.wikipedia.org/wiki/GPGPU
12. Khronous Group, OpenGL 3, December 2008 URL : http://www.khronos.org/opencl
13. *OpenCL - The open standard for parallel programming of heterogeneous systems* URL :
    http://www.khronos.org/opencl
14. Programming the GPU and a brief intro to the OPENGL shading language – Marcel Cohan & VVR Talk
15. David B Kirk, Wen-mei W. Hwu nvidia corporation, 2010, Elsevier, Morgan Kaufmann Publishers, 2011
16. Benedict R Gaster, Lee Howes, David R Kaeli, Perhadd Mistry Dana Schaa, Heterogeneous Computing with OpenCL, Elsevier, Moran Kaufmann Publishers, 2011
17. The OpenCL 1.2 Specification (Document Revision 15) Last Released November 15, 2011
    Editor : Aaftab Munshi Khronos OpenCL Working Group
18. The OpenCL 1.1 Quick Reference card

References

19. http://developer.amd.com/sdks/AMDAPPSDK/downloads/Pages/default.aspx AMD APP SDK with OpenCL 1.2 Support
20. http://developer.amd.com/sdks/AMDAPPSDK/downloads/Pages/default.aspx#oneAMD-APP-SDKv2.7 (Linux) with OpenCL 1.2 Support
21. http://icl.cs.utk.edu/magma/software/ MAGMA OpenCL
22. http://developer.amd.com/zones/OpenCLZone/pages/GettingStarted.aspx Getting Started with OpenCL
23. http://developer.amd.com/openclforum AMD Developer OpenCL FORUM
24. http://developer.amd.com/zones/OpenCLZone/programming/pages/benchmarkingopenclperformance.aspx AMD Developer Central - Programming in OpenCL - Benchmarks performance
25. http://developer.amd.com/sdks/AMDAPPSDK/assets/opencl-1.2.pdf OpenCL 1.2 (pdf file)
26. http://developer.amd.com/zones/opensource/pages/ocl-emu.aspx AMD OpenCL Emulator-Debugger
27. http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf The OpenCL 1.2 Specification (Document Revision 15) Last Released November 15, 201 Editor : Aaftab Munshi <I> Khronos OpenCL Working Group
28. http://www.khronos.org/registry/cl/sdk/1.1/docs/man/xhtml/ OpenCL1.1 Reference Pages
29. The Intel SDK for OpenCL Applications XE – Optimization Guide includes many more details.

Source : Intel, NVIDIA, Khronos AMD, References

Intel Xeon Phi - Coprocessors :

Profiling & Timing

Part-4

Profiling & Timing

Intel Xeon Phi : Profiling & Timings

- The Intel Composer XE – Development tool and SDK suite available for developing **Intel Xeon Phi**

  - Intel supports event-monitoring registers. On the coprocessor these are similar to some counters on a processor, but with additional abilities for the higher core count, higher threads per core, and wider vectors.

  - Using counters instead of more intrusive techniques (like profiling compiler time option `-pg`) is critical when dealing with high performance programs.

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

Intel Xeon Phi : Profiling & Timings

- The Intel Composer XE – Development tool and SDK suite available for developing **Intel Xeon Phi**
  - Intel Vtune Amplifier XE product
  - Open source community has Performance Application Programming Interface (PAPI).
  - MPI – Intel Trace Analyzer and Collector (ITAC).

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

Intel Xeon Phi : The Intel Composer XE 2013

- The Intel Composer XE – Development tool and SDK suite available for developing Intel Xeon Phi
  - It includes C/C++ Fortran Complier
  - It includes runtime libraries like OpenMP, thread etc. Debuging tool and math kernel library (MKL)
  - Supports various parallel programming models fro Intel Xeon Phi such as Intel Cilk Plus, Intel Threading Building blocks (TBB), OpenMP and Pthread
  - It includes Intel MKL

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

Intel Trace Analyzer and Collector (ITAC)

❖ **Intel MPI, Intel Trace Analyzer and Collector(ITAC) on MIC**

- Intel Trace Collector gathers information from running programs into a trace file, and the Intel Trace Analyzer allows the collected data to be viewed and analyzed after a run.
- The Intel Trace Analyzer and Collector support processors and coprocessors.
- The Trace Collector can integrate information from multiple sources including an instrumented Intel MPI Library and PAPI.
- Trace file from an application running on the **host system** and **coprocessor** simultaneously can be generated
- Generate trace file only **on Coprocessor** system

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

Intel Vtune Amplier XE 2013

- Rich set of performance data for hotspots, threading, locks & waits, bandwidth
- Ability to both collect and view sampled data from an Intel Xeon Phi coprocessor.
  - Hotspot Analysis—Quickly locate code that is taking a lot of time. See the calling sequences.
  - Lightweight Hotspot Analysis—Low overhead, high resolution using on-chip hardware.
  - Locks & Waits—Tune threading. Find synchronization objects impeding performance scaling.
  - System Wide Analysis—Tune drivers, kernel modules and multi-process apps.
  - Call Count Analysis—Find code that will benefit from inlining.
  - Bandwidth, Memory, Branch analysis, more—Advanced analysis
  - MPI applications— Analyze hybrid applications using MPI and OpenMP.

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

Intel Vtune Performance Analyzer 2013

- The VTune™ Performance Analyzer provides information on the performance of your code.
- The VTune analyzer shows you the performance issues, enabling you to focus your tuning effort and get the best performance boost in the least amount of time

Open Source Software Tool on Intel Xeon Phi

Performance Application Programming Interface (PAPI) – Open Source Tool

- PAPI provides a consistent interface and methodology for use of the performance counter

- hardware found in most major microprocessors including the Intel Xeon Phi coprocessor. PAPI is used by quite a number of open source tools (a list is available on the PAPI Web site

Timing on Intel Xeon Phi

`Clocksources` on the coprocessor

- There are two clock generators that can generate clock signals.
  - At the system level is the PCIe clock generator;
  - The second is the ICC PLL.
- From the programmers point of view there are two clock sources accessible on the coprocessor: MIC Elapsed Time Counter (**`micetc`**) and the Time Stamp Counter (**`tsc`**).
- The default clock source on the coprocessor has been **`micetc`**. The **`micetc clocksource`** is also compensated for power management events delivering a very stable **`clocksource.`**

Timing on Intel Xeon Phi

Measuring timing and data in offload regions

- You can measure both the amount of time it takes to execute an offload region of code, as well as the amount of data transferred during the execution of the offload region.

- From the programmers point of view there are two clock sources accessible on the coprocessor: MIC Elapsed Time Counter (**`micetc`**) and the Time Stamp Counter (**`tsc`**).

- The default clock source on the coprocessor has been **`micetc`**. The **`micetc clocksource`** is also compensated for power management events delivering a very stable **`clocksource.`**

Intel Xeon Phi - Coprocessors : system Performance Results

Part-5

Benchmark Results

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)*

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

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
 …….
```

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

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
 ……..
 ……..
 ……..
```

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

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.X
```

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

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

Intel Xeon Phi - Coprocessors : system Performance Results

Part-6

Low Level Benchmark Results

Intel Xeon –Phi Programming Paradigms

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

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

Intel Xeon Phi - Coprocessors : Tuning and Performance Issues

Part-7

Tips for Tuning and Performance

An Overview of Intel Xeon Phi – Tuning & Perf.

Lecture Outline

Following topics will be discussed

- Understanding of Intel Multi-Core Systems with Intel Xeon Phi Programming from Performance Point of View

Xeon Phi : Programming Environment

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

  OpenMP, Intel TBB, Cilk Plus, Pthreads

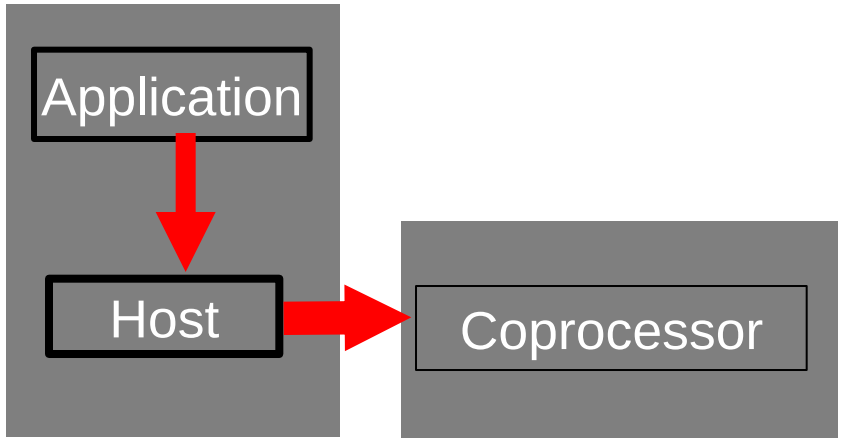
- **Message Passing Programming**

  (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/

Intel Xeon-Phi : Shared Address Space 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 utilizing 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 : Shared Address 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 Intel Xeon coprocessor.

- Resource availability and Memory access is an important for threading on all 60 Cores.

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

Intel Xeon-Phi : Programming Env.

Keys to Productive Performance

- 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 ® Cilk TM Plus, OpenMP*, Intel ® Threading Building Blocks, Pthreads, etc.)
- Go asynchronous to overlap computation and communication

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

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.

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.

Summary: Tricks for Performance

Performance on Xeon Phi using different prog.

- 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
- Loop Optimizations may benefit performance

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

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 Coprocessor : Prog. Env & Tips for obtaining Performance (Part-II)

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/

A Family of Parallel Programming Models Developer Choice

Step 1 :

Intel® Cilk™ Plus	Intel® Threading Building Blocks	Domain-Specific Libraries	Established Standards	Research and Development
C/C++ language extensions to simplify parallelism	Widely used C++ template library for parallelism	Intel® Integrated Performance Primitives	Message Passing Interface (MPI)	Intel® Concurrent Collections
			OpenMP* (offload TR coming "real soon")	Intel® Offload Extensions
		Intel® Math Kernel Library		Intel® SPMD Parallel Compiler (ispc)
			Coarray Fortran	
Open sourced	Open sourced		OpenCL*	
Also an Intel product	Also an Intel product			

Applicable to Multicore and Manycore Programming

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

Objective of Scalar and Serial Optimization

Step 2 :

- Obtain the most efficient implementation for the problem at hand
- Identify the opportunity for vectorization and parallelization
- Create Base to account for vectorization and parallelization gains
  - Avoid situation when vectorized, slower code was parallelized and create a false impression of performance gain

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

Algorithmic Optimizations

- Elevate constants out of the core loops
  - Compiler can do it, but it need your cooperation
  - Group constants together
- Avoid and replace expensive operations
  - divide a constant can usually be replace by multiplying its reciprocal
- Strength reduction in hot loop
  - People like inductive method, because it's clean
  - Iterative can strength reduce the operation involved
  - In this example, exp() is replace by a simple multiplication

```
const double      dt = T / (double)TIMESTEPS;
    const double     vDt = V * sqrt(dt);
    for(int i = 0; i <= TIMESTEPS; i++){
     double price = S * exp(vDt * (2.0 * i -
                TIMESTEPS));
          cell[i] = max(price - X, 0);
                    }
```

```
const double factor = exp(vDt * 2);
  double        price = S * exp(-
        vDt(2+TIMESTEPS));
for(int i = 0; i <= TIMESTEPS; i++){
        price = factor * price;
     cell[i] = max(price - X, 0);
                }
```

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

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/

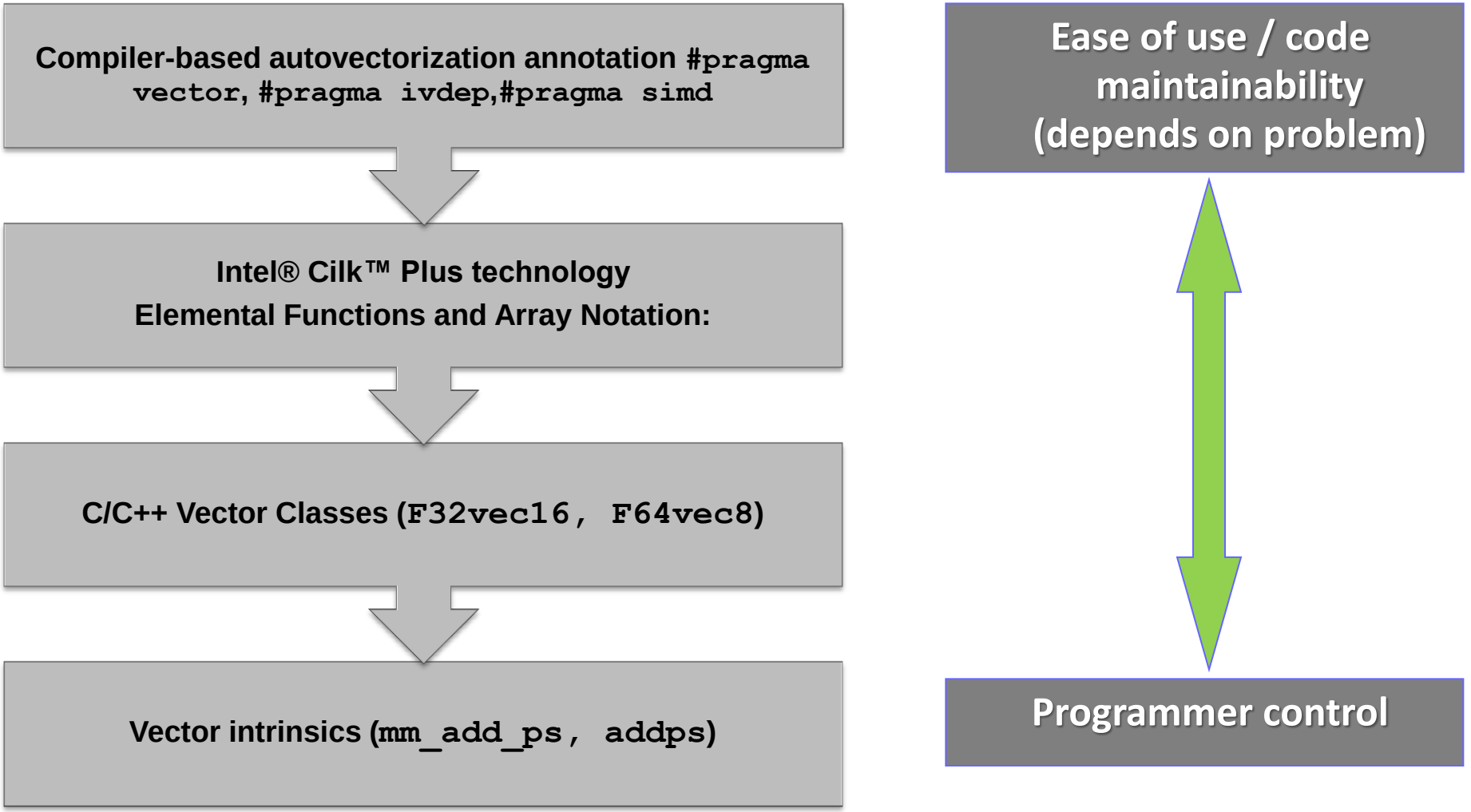
Vectorization and SIMD Execution

Step 3 :

- **SIMD**
  - Flynn's Taxonomy: Single Instruction, Multiple Data
  - CPU perform the same operation on multiple data elements
- **SISD**
  - Single Instruction, Single Data
- **Vectorization**
  - In the context of Intel® Architecture Processors, the process of transforming a scalar operation (SISD), that acts on a single data element to the vector operation that that act on multiple data elements at once(SIMD).
  - Assuming that setup code does not tip the balance, this can result in more compact and efficient generated code
  - For loops in "normal" or "unvectorized" code, each assembly instruction deals with the data from only a single loop iteration

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

SIMD Abstraction – Options Compared



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

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/

Options for Parallelism on Intel® Architecture

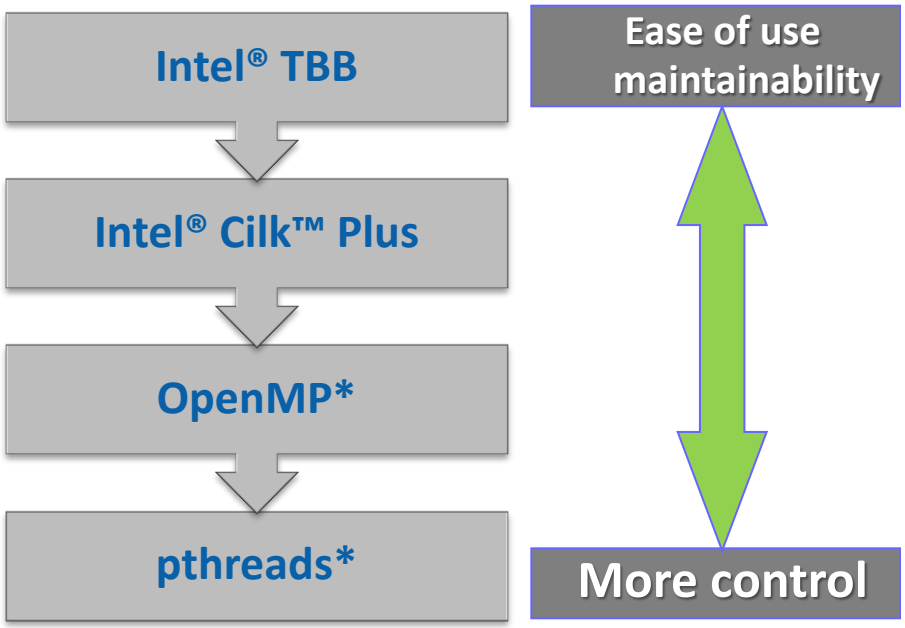
Step 4 :

- C++ template Library of parallel algorithms, containers
- Load balancing via work stealing

- Keyword extension of C/C++, Serial equivalence via compiler
- Load balancing via work stealing

- Well known industry standard
- Best suited when resource utilization is known at design time

- Time-tested industry standard for Unix-like
- Common denominator or other high level threading libraries



- What's available on Intel® host processor are also available on Intel® target coprocessor
- Many others (boost) are ported to the coprocessor
- Choose the best threading model your problem dictates

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

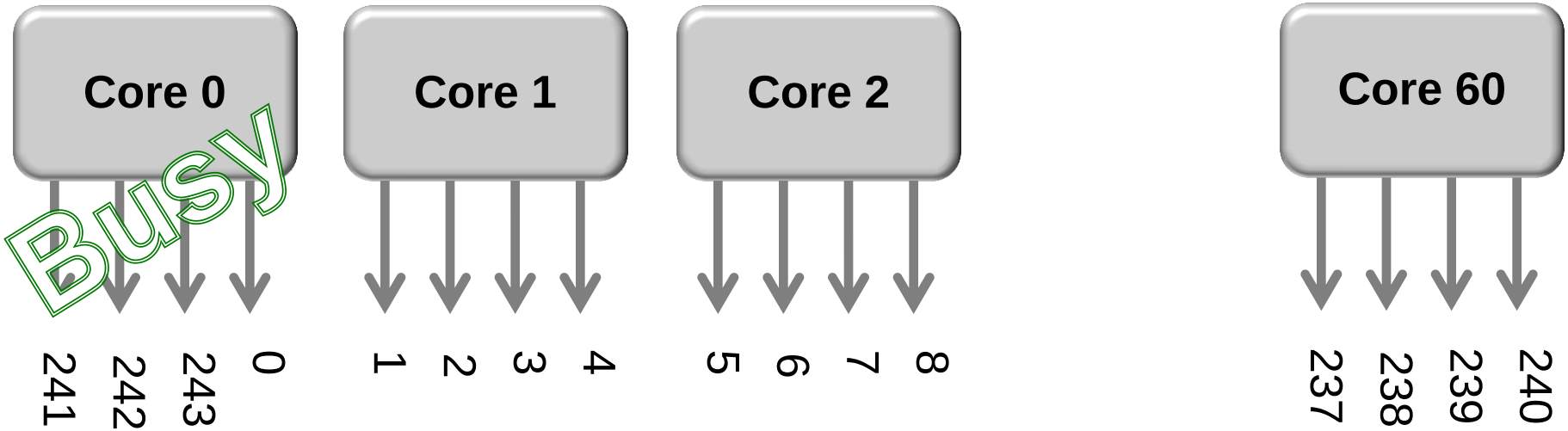
Options for Parallelism – pthreads*

- POSIX* Standard for thread API with 20 years history
- Foundation for other high level threading libraries
- Independently exist on the host and Intel® MIC
- No extension to go from the host to Intel® MIC
- Advantage: Programmer has explicit control
  - From workload partition to thread creation, synchronization, load balance, affinity settings, etc.
- Disadvantage: Programmer has too much control
  - Code longevity
  - Maintainability
  - Scalability

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

Thread Affinity using pthreads*

- Partition the workload to avoid load imbalance
  - Understand static vs. dynamic workload partition
- Use pthread API, define, initialize, set, destroy
  - Set CPU affinity with `pthead_setaffinity_np()`
  - Know the thread enumeration and avoid core 0
  - Core 0 boots the coprocessor, job scheduler, service interrupts

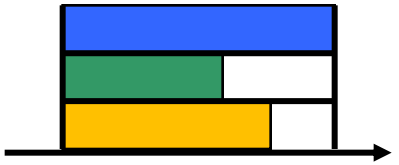


Source : References & Intel Xeon-Phi; http://www.intel.com/

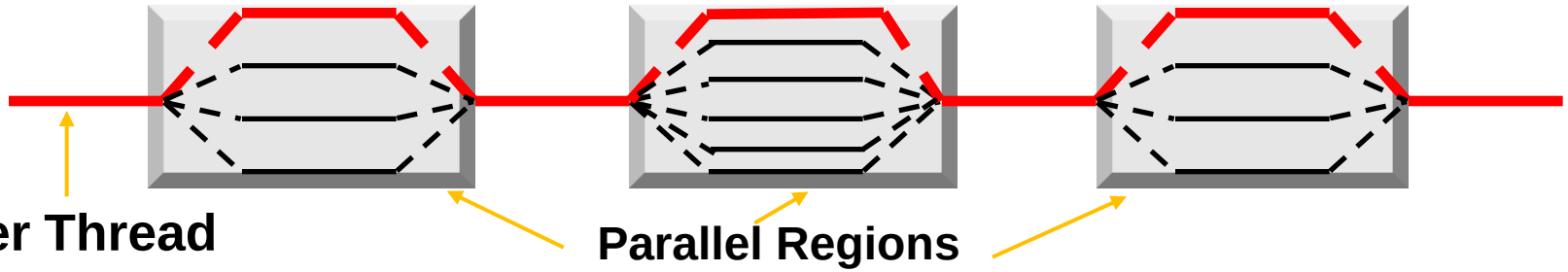
Options for Parallelism – OpenMP*

- 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/

413

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
```

Scale from Multicore to Manycore

Step 5 :

A Tale of Two Architectures

	Intel® Xeon® processor	Intel® Xeon Phi™ Coprocessor
Sockets	2	1
Clock Speed	2.6 GHz	1.1 GHz
Execution Style	Out-of-order	In-order
Cores/socket	8	Up to 61
HW Threads/Core	2	4
Thread switching	HyperThreading	Round Robin
SIMD widths	8SP, 4DP	16SP, 8DP
Peak Gflops	692SP, 346DP	2020SP, 1010DP
Memory Bandwidth	102GB/s	320GB/s
L1 DCache/Core	32kB	32kB
L2 Cache/Core	256kB	512kB
L3 Cache/Socket	30MB	none

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

Assessing potential

- **Threads**
  - Code analysis – loop nesting, iteration counts, determinism
  - Intel Vtune™ Amplifier timeline analysis – existence of applciation serialization
  - Performance vs. threads – knee of the curve
- **Vectorization**
  - VTune Amplifier hot spots and compiler VEC reports
  - HW PerfMon-based evaluation
  - Performance vs. vectorization on/off
- **Bandwidth**
  - HW PerfMon-based evaluation

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

More on Thread Affinity

- Bind the worker threads to certain processor core/threads
- Minimizes the thread migration and context switch
- Improves data locality; reduce coherency traffic
- Two components to the problem:
  - How many worker threads to create?
  - How to bind worker threads to core/threads?
- Two ways to specify thread affinity
  - Environment variables OMP_NUM_THREADS, KMP_AFFINITY
  - C/C++ API: `kmp_set_defaults("KMP_AFFINITY=compact")`
    `omp_set_num_threads(244);`
    - Add to your source file#include <omp.h>
    - Compiler with –openmp
    - Use libiomp5.so

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

The Optimal Thread Number

- Intel MIC maintains 4 hardware contexts per core
  - Round-robin execution policy,
  - Require 2 threads for decent performance
  - Financial algorithms takes all 4 threads to peak
- Intel Xeon processor optionally use HyperThreading
  - Execute-until-stall execution policy
  - Truly compute intensive ones peak with 1 thread per core
  - Finance algorithms likes HyperThreading, 2 threads per core
- Use OpenMP application with NCORE number of cores
  - **Host only:** 2 x ncore (or 1x if HyperThreading disabled)
  - **MIC Native:** 4 x ncore
  - **Offload:** 4 x (ncore-1) OpenMP runtime avoids the core OS runs

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

Intel Xeon Phi Coprocessor : Prog. Env & Tips for obtaining Performance (Part-III)

Intel Xeon Phi Coprocessor : Prog. Env &

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/

Prog.API - Multi-Core Systems with Devices

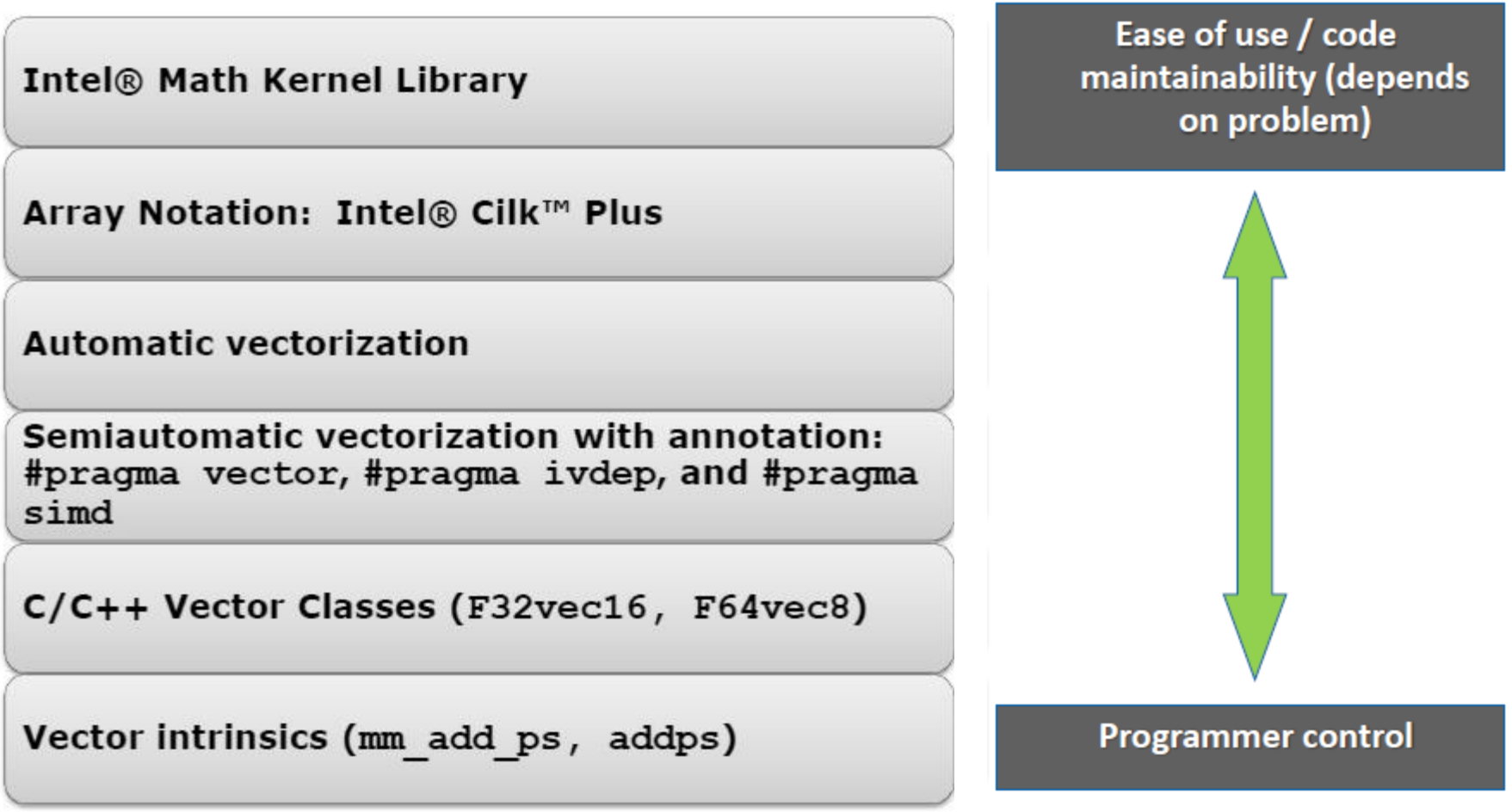
Performance: Intel Xeon-Phi Coprocessor

- 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/

Intel Xeon-Phi : Prog. Env. Perf Issuses

Options for Vectorization : Use Tools



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

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

Optimised Offloaded Code

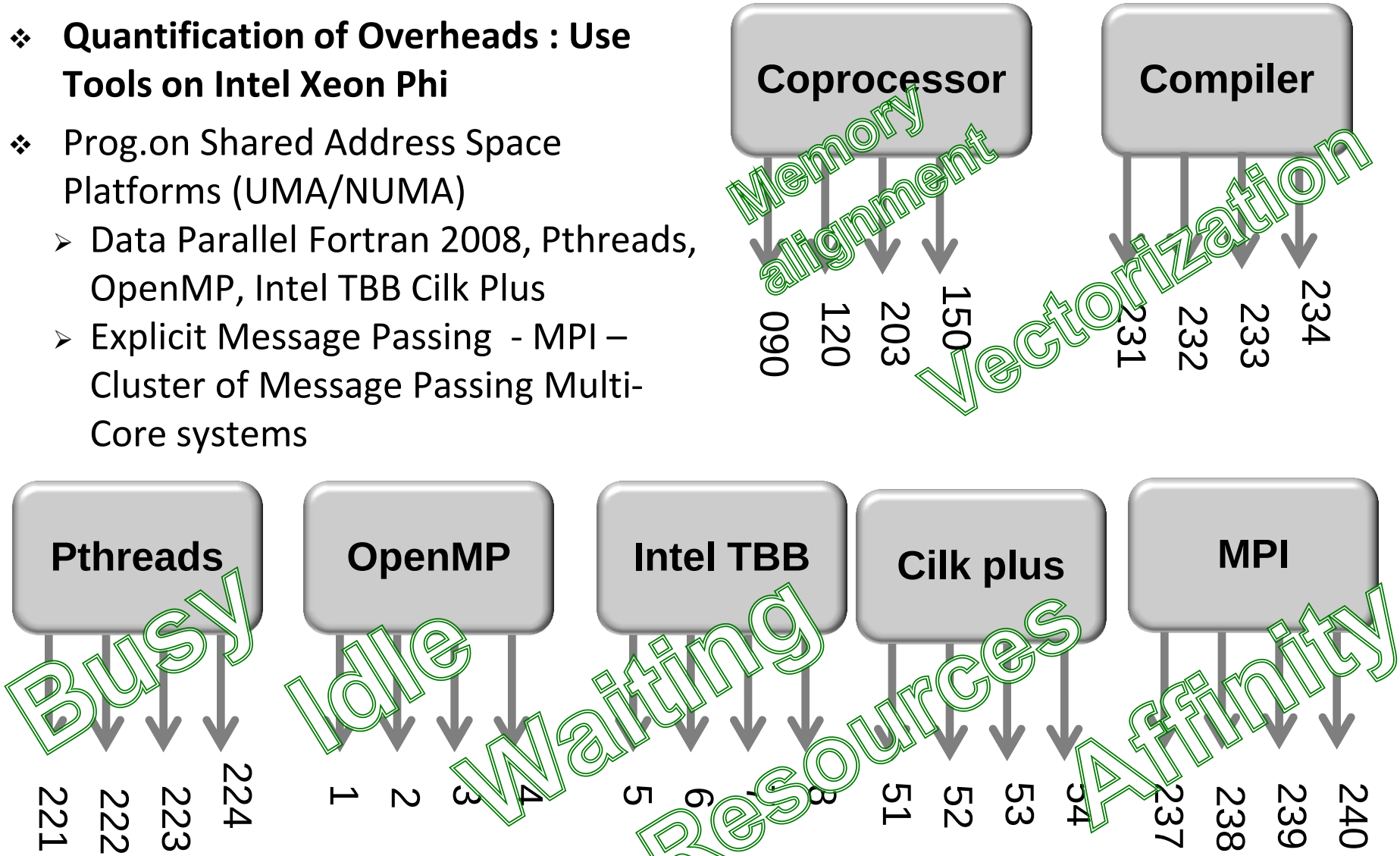
Tuning & Performance :

- 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 Prog. : Tools to Measure Overheads

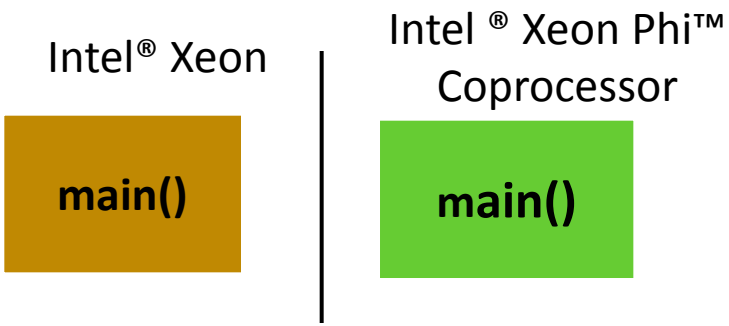
- **Quantification of Overheads : Use Tools on Intel Xeon Phi**
- Prog.on Shared Address Space Platforms (UMA/NUMA)
  - Data Parallel Fortran 2008, Pthreads, OpenMP, Intel TBB Cilk Plus
  - Explicit Message Passing - MPI – Cluster of Message Passing Multi-Core systems



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

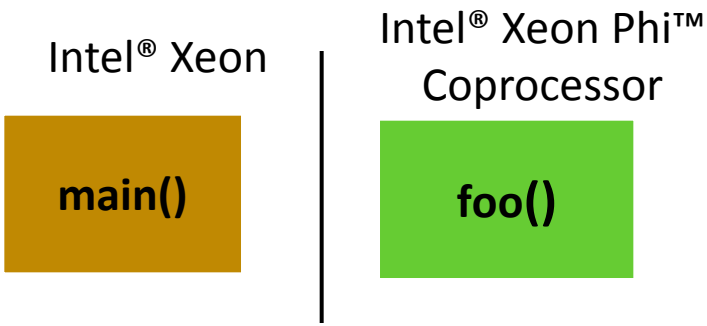
Intel Xeon & Xeon Phi : Execution Modes

- **Quantification of Overheads – Explicit / Implicit Data Transfer – Using Offload**



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 (…)

Intel Xeon-Phi : Programming Env.

Pros:

- Compilation with an additional Intel compiler flag (`-mmic`);
- Scalability tests: fast and smooth;
- Quick analysis with Intel tools (VtuneT, Itac Intel Trace Analyzer and Collector;
- Porting time: one day with validation of the numerical result;
- expert developer of FARM, with good knowledge of the Intel Compiler, But with only a basic knowledge of MIC.
- Best scalability with OpenMP and Hybrid.

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

Xeon Phi : Programming Environment

Porting on MIC : Issues to be addressed

- **MPI_Init** routine problem: increasing CPU time for increasing number of processes; Same problem when using two MICs together;
- Detailed analysis of OpenMP threads & thread affinity and Memory available per thread
- Execution time depends strongly from code vectorization, so compiler vectorization for data parallel and task parallel constructs
- code re-structure and memory access pattern are a key point to have a vectorizable satisfactory overall Performances.

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

Intel Xeon Phi : Performance Issues

Factors to work around

- Limited problem size or limited exposure
  - Inherent lack of available parallelism
  - Parallelism not adequately exposed by programmer
- Excessive synchronization
  - Inhibits harvesting thread parallelism
- ISA-specific issues
  - Data structures excessively rely on scatter/gather
  - Use of 64b integer indices and 64 INT←→ FP conversion
- Offload overhead
  - Excessive communication/computation ratio, unhidden communication
- Memory footprint and working set size
  - Limited to 8GB, unless you “overlay,” e.g. with offload

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

Intel Xeon Phi : Performance Issues

Prefetch on Intel Multicore and Many-core

- **Objective:** Move data from memory to L1 or L2 Cache in anticipation of CPU Load/Store
- More import on in-order Intel Xeon Phi Coprocessor
- Less important on out of order Intel Xeon Processor
- Compiler prefetching is on by default for Intel® Xeon Phi™ coprocessors at –O2 and above
- Compiler prefetch is not enabled by default on Intel® Xeon® Processors
  - Use external options `-opt-prefetch[=n] n = 1.. 4`
- Use the compiler reporting options to see detailed diagnostics of prefetching per loop
  - Use `-opt-report-phase hlo -opt-report 3`

Intel Xeon Phi : Performance Issues

Automatic Prefetches

Loop Prefetch

- Compiler generated prefetches target memory access in a future iteration of the loop
- Target regular, predictable array and pointer access

Interactions with Hardware prefetcher

- Intel® Xeon Phi™ Comprocessor has a hardware L2 prefetcher
- If Software prefetches are doing a good job, Hardware prefetching does not kick in
- References not prefetched by compiler may get prefetched by hardware prefetcher

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

Intel Xeon Phi : Performance Issues

Explicit Prefetch

- **Use Intrinsics**
  - `_mm_prefetch((char *) &a[i], hint);`
    See xmmintrin.h for possible hints (for L1, L2, non-temporal, …)
  - But you have to specify the prefetch distance
  - Also gather/scatter prefetch intrinsics, see zmmintrin.h and compiler user guide, e.g. _mm512_prefetch_i32gather_ps
- **Use a pragma / directive (easier):**
  - **#pragma prefetch a [:hint[:distance]]**
  - You specify what to prefetch, but can choose to let compiler figure out how far ahead to do it.
- **Use Compiler switches:**
  - `-opt-prefetch-distance=n1[,n2]`
  - specify the prefetch distance (how many iterations ahead, use n1 and prefetches inside loops. n1 indicates distance from memory to L2.

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

Intel Xeon Phi : Performance Issues

Streaming Store

- Avoid read for ownership for certain memory write operation
- Bypass prefetch related to the memory read
- Use `#pragma vector nontemporal(v1,…)` to drop a hint to compiler
- Without Streaming Stores 448 B[illegible]

  - With Streaming Stores, 320 Bytes read/write per iteration
  - Relief Bandwidth pressure; improve cache utilization
  - –vec-report6 displays the compiler action

```
for (int chunkBase = 0; chunkBase < OptPerThread; chunkBase += CHUNKSIZE)
{
#pragma simd vectorlength(CHUNKSIZE)
#pragma simd
#pragma vector aligned
#pragma vector nontemporal (CallResult, PutResult)
      for(int opt = chunkBase; opt < (chunkBase+CHUNKSIZE); opt++)
      {
         float CNDD1;
         float CNDD2;
         float CallVal =0.0f, PutVal  = 0.0f;
         float T = OptionYears[opt];
         float X = OptionStrike[opt];
         float S = StockPrice[opt];
                        ……

         CallVal  = S * CNDD1 - XexpRT * CNDD2;
         PutVal  = CallVal  +  XexpRT - S;
         CallResult[opt] = CallVal ;
         PutResult[opt] = PutVal ;
      }
}
```

```
bs_test_sp.c(215): (col. 4) remark: vectorization support: streaming store was generated for CallResult.
bs_test_sp.c(216): (col. 4) remark: vectorization support: streaming store was generated for PutResult.
```

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

Intel Xeon Phi : Performance Issues

Data Blocking

- Partition data to small blocks that fits in L2 Cache
  - Exploit data reuse in the application.
  - Ensure the data remains in the cache across multiple uses
  - Using the data in cache remove the need to go to memory
  - Bandwidth limited program may execute at FLOPS limit
- Simple case of 1D
  - Data size DATA_N is used WORK_N times from 100s of threads
  - Each handles a piece of work and have to traverse all data

Without Blocking

- 100s of thread pound on different area of DATA_N
- Memory interconnet limit the performance

```
#pragma omp parallel for
for(int wrk = 0; wrk < WORK_N; wrk++)
{
     initialize_the_work(wrk);
     for(int ind = 0; ind < DATA_N; ind++)
     {
        dataptr  datavalue = read_data(dataind);
        result = compute(datavalue);
        aggregate = combine(aggregate, result);
     }
     postprocess_work(aggregate);
}
```

With Blocking

- Cacheable BSIZE of data is processed by all 100s threads a time
- Each data is read once kept reusing until all threads are done with it

```
for(int BBase = 0; BBase < DATA_N; BBase += BSIZE)
{
#pragma omp parallel for
    for(int wrk = 0; wrk < WORK_N; wrk++)
    {
         initialize_the_work(wrk);
         for(int ind = BBase; ind < BBase+BSIZE; ind++)
         {
            dataptr  datavalue = read_data(ind);
            result = compute(datavalue);
            aggregate[wrk] = combine(aggregate[wrk], result);
          }
          postprocess_work(aggregate[wrk]);
    }
}
```

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

Intel Xeon Phi : Performance Issues

Memory Alignment

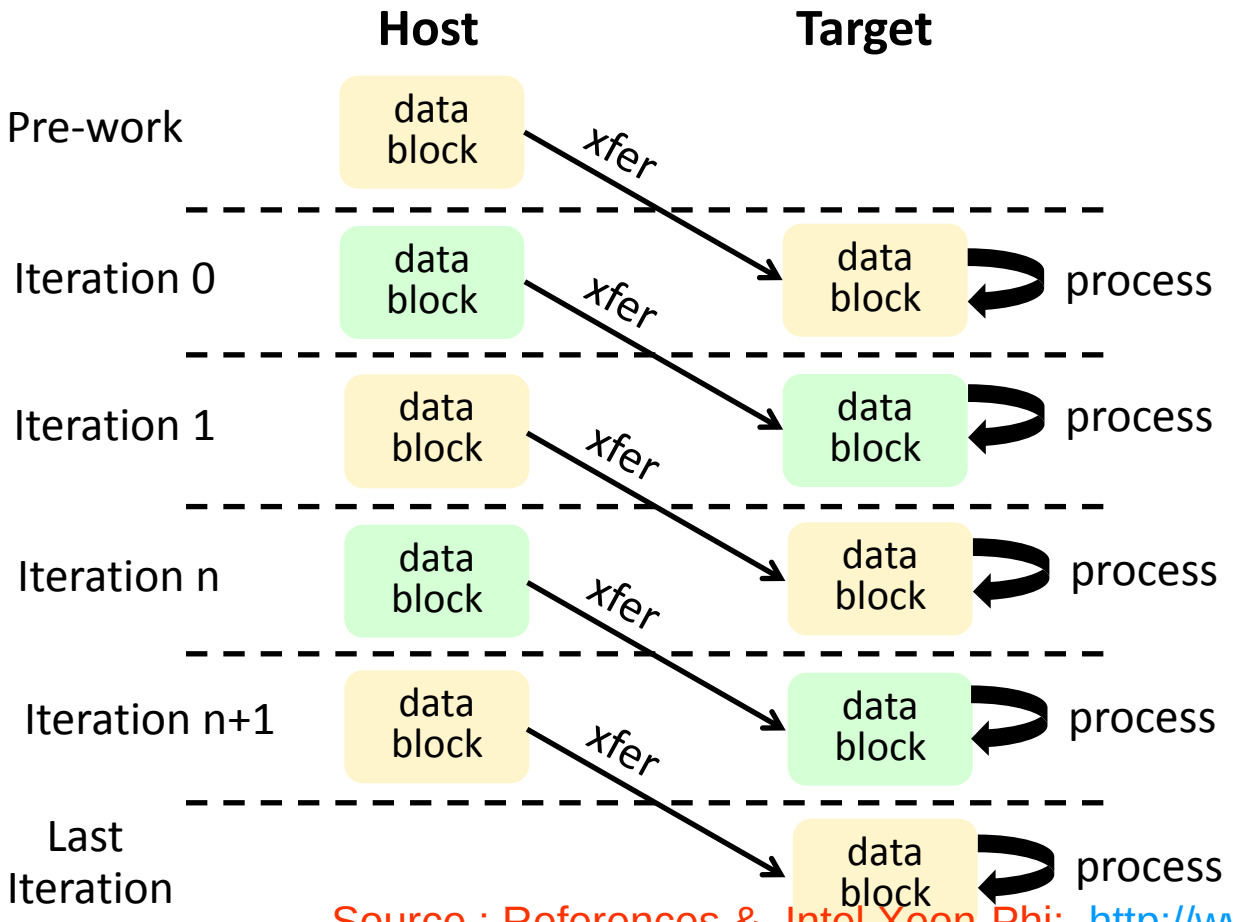
- Allocated memory on heap
  - `_mm_malloc(int size, int aligned)`
  - `scalable_aligned_malloc(int size, int aligned)`
- Declarations memory:
  - `__attribute__((aligned(n))) float v1[];`
  - `__declspec(align(n)) float v2[];`
- Use this to notify compiler
  - `__assume_aligned(array, n);`
- Natural boundary
  - Unaligned access can fault the processor
- Cacheline Boundary
  - Frequently accessed data should be in 64
- 4K boundary
  - Sequentially accessed large data should be in 4K boundary

Instruction	Length	Alignment
SSE	128 Bits	16 Bytes
AVX	256 Bits	32 Bytes
IMCI	512 Bits	64 Bytes

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

Double Buffering Example

- Transfer and work on a dataset in small pieces
- While part is being transferred, work on another part!

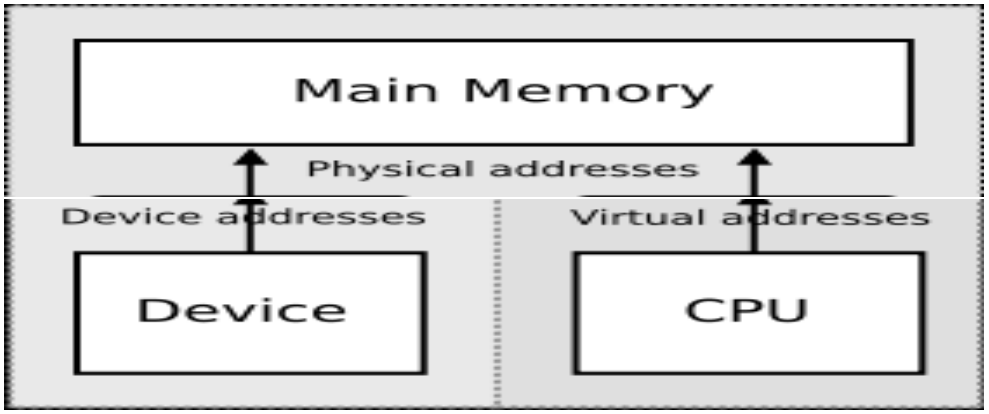


Source : References & Intel Xeon-Phi; http://www.intel.com/

Memory Mapping

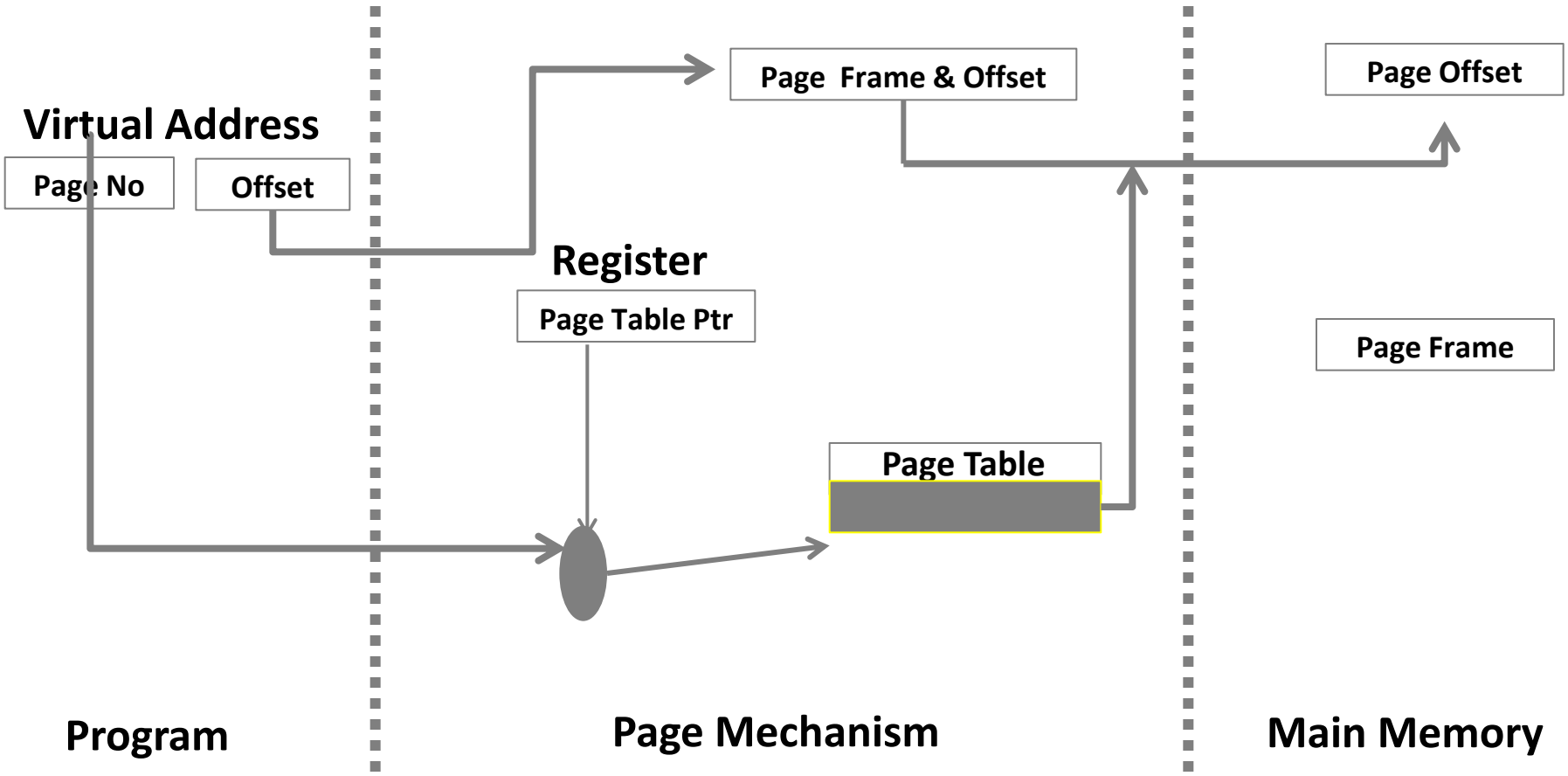
Implementation: Matrix into Matrix Multiplication using mmap (Assume that Matrix Size A = 1,00,000 Real float and Matrix Size B = 1,00,000 Real float)

- Translation of address issued by some device (e.g., CPU or I/O device) to address sent out on memory bus (physical address)
- Mapping is performed by memory management units



Computing – Enabling Huge Memory – Implementation using Memory Mapping (mmap)

Address Mapping Function (Review)



Memory – Huge Pages and Pre-faulting

- IA processors support multiple page sizes; commonly 4K and 2MB
- *Some* applications will benefit from using huge pages
  - Applications with sequential access patterns will improve due to larger TLB "reach"
- TLB miss vs. Cache miss
  - TLB miss means walking the 4 level page table hierarchy
    - Each page walk could result in additional cache misses
  - TLB is a scarce resource and you need to "manage" them well
- On Intel® Xeon Phi™ Coprocessor
  - 64 entries for 4K, 8 entries for 2MB
  - Additionally, 64 entries for second level DTLB.
    - Page cache for 4K, L2 TLB for 2MB pages
- Linux supports huge pages – CONFIG_HUGETLBFS
  - 2.6.38 also has support for Transparent Huge Pages (THP)
- Pre-faulting via MAP_POPULATE flag to mmap()

Intel Xeon Phi : The Intel Composer XE 2013

- The Intel Composer XE – Development tool and SDK suite available for developing Intel Xeon Phi
  - It includes C/C++ Fortran Complier
  - It includes runtime libraries like OpenMP, thread etc. Debuging tool and math kernel library (MKL)
  - Supports various parallel programming models fro Intel Xeon Phi such as Intel Cilk Plus, Intel Threading Building blocks (TBB), OpenMP and Pthread
  - It includes Intel MKL

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

Intel Trace Analyzer and Collector (ITAC)

❖ **Intel MPI, Intel Trace Analyzer and Collector(ITAC) on MIC**

- Intel Trace Collector gathers information from running programs into a trace file, and the Intel Trace Analyzer allows the collected data to be viewed and analyzed after a run.
- The Intel Trace Analyzer and Collector support processors and coprocessors.
- The Trace Collector can integrate information from multiple sources including an instrumented Intel MPI Library and PAPI.
- Trace file from an application running on the **host system** and **coprocessor** simultaneously can be generated
- Generate trace file only **on Coprocessor** system

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 & Acknowledgements

References

57. Andrews, Grogory R. (2000), Foundations of Multithreaded, Parallel, and Distributed Programming, Boston, MA : Addison-Wesley
58. Butenhof, David R (1997), Programming with POSIX Threads , Boston, MA : Addison Wesley Professional
59. Culler, David E., Jaswinder Pal Singh (1999), Parallel Computer Architecture - A Hardware/Software Approach , San Francsico, CA : Morgan Kaufmann
60. Grama Ananth, Anshul Gupts, George Karypis and Vipin Kumar (2003), Introduction to Parallel computing, Boston, MA : Addison-Wesley
61. Intel Corporation, (2003), Intel Hyper-Threading Technology, Technical User's Guide, Santa Clara CA : Intel Corporation Available at : http://www.intel.com
62. Shameem Akhter, Jason Roberts (April 2006), Multi-Core Programming - Increasing Performance through Software Multi-threading , Intel PRESS, Intel Corporation,
63. Bradford Nichols, Dick Buttlar and Jacqueline Proulx Farrell (1996), Pthread Programming O'Reilly and Associates, Newton, MA 02164,
64. James Reinders, Intel Threading Building Blocks – (2007) , O'REILLY series
65. 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
66. Intel Threading Methodology ; Principles and Practices Version 2.0 copy right (March 2003), Intel Corporation
67. William Gropp, Ewing Lusk, Rajeev Thakur **(1999),** Using MPI-2, Advanced Features of the Message-Passing Interface, The MIT Press..
68. Pacheco S. Peter, **(1992),** Parallel Programming with MPI, , University of Sanfrancisco, Morgan Kaufman Publishers, Inc., Sanfrancisco, California
69. Kai Hwang, Zhiwei Xu, (**1998**), Scalable Parallel Computing (Technology Architecture Programming), McGraw Hill New York.
70. Michael J. Quinn (**2004**), Parallel Programming in C with MPI and OpenMP McGraw-Hill International Editions, Computer Science Series, McGraw-Hill, Inc. Newyork
71. Andrews, Grogory R. **(2000),** Foundations of Multithreaded, Parallel, and Distributed Progrmaming, Boston, MA : Addison-Wesley

References & Acknowledgements

References

72. SunSoft. Solaris multithreaded programming guide. SunSoft Press, Mountainview, CA, **(1996),** Zomaya, editor. Parallel and Distributed Computing Handbook. McGraw-Hill,
73. Chandra, Rohit, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, and Ramesh Menon, **(2001)**,Parallel Programming in OpenMP San Fracncisco Moraan Kaufmann
74. S.Kieriman, D.Shah, and B.Smaalders **(1995),** Programming with Threads, SunSoft Press, Mountainview, CA. 1995
75. Mattson Tim, **(2002),** Nuts and Bolts of multi-threaded Programming Santa Clara, CA : Intel Corporation, Available at : http://www.intel.com
76. I. Foster **(1995,** Designing and Building Parallel Programs ; Concepts and tools for Parallel Software Engineering, Addison-Wesley (1995
77. J.Dongarra, I.S. Duff, D. Sorensen, and H.V.Vorst **(1999),** Numerical Linear Algebra for High Performance Computers (Software, Environments, Tools) SIAM, 1999
78. OpenMP C and C++ Application Program Interface, Version 1.0". **(1998),** OpenMP Architecture Review Board. October 1998
79. D. A. Lewine. *Posix Programmer's Guide:* **(1991),** Writing Portable Unix Programs with the Posix. 1 Standard. O'Reilly & Associates, 1991
80. 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/
81. 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].
82. 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*
83. A. Zomaya, editor. Parallel and Distributed Computing Handbook. McGraw-Hill, **(1996)**
84. OpenMP C and C++ Application Program Interface, Version 2.5 (**May 2005**)", From the OpenMP web site, URL **: http://www.openmp.org/**
85. Stokes, Jon 2002 Introduction to Multithreading, Super-threading and Hyper threading *Ars Technica*, October **(2002)**

References & Acknowledgements

References

86. Andrews Gregory R. 2000, Foundations of Multi-threaded, Parallel and Distributed Programming, Boston MA : Addison – Wesley (**2000)**
87. 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)**
88. http://www.erc.msstate.edu/mpi
89. http://www.arc.unm.edu/workshop/mpi/mpi.html
90. http://www.mcs.anl.gov/mpi/mpich
91. The MPI home page, with links to specifications for MPI-1 and MPI-2 standards : http://www.mpi–forum.org
92. Hybrid Programming Working Group Proposals, Argonne National Laboratory, Chiacago (2007-2008)
93. TRAC Link : https://svn.mpi-forum.org/trac/mpi-form-web/wiki/MPI3Hybrid
94. Threads and MPI Software, Intel Software Products and Services 2008 - 2009
95. Sun MPI 3.0 Guide November 2007
96. Treating threads as MPI processes thru Registration/deregistration –Intel Software Products and Services 2008 – 2009
97. Intel MPI library 3.2 - http://www.hearne.com.au/products/Intelcluster/edition/mpi/663/
98. http://www.cdac.in/opecg2009/
99. PGI Compilers http://www.pgi.com

References & Acknowledgements

References

100. Andrews Gregory R. 2000, Foundations of Multi-threaded, Parallel and Distributed Programming, Boston MA : Addison – Wesley (**2000)**
101. 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)**
102. http://www.nvidia.com/object/nvidia-kepler.html NVIDIA Kepler Architecture 2012
103. http://developer.nvidia.com/cuda-toolkit NVIDIA CUDA toolkit 5.0 Preview Release April 2012
104. http://developer.nvidia.com/category/zone/cuda-zone NVIDIA Developer Zone
105. http://developer.nvidia.com/gpudirect RDMA for NVIDIA GPUDirect coming in CUDA 5.0 Preview Release, April 2012
106. http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf NVIDIA CUDA C Programming Guide Version 4.2 dated 4/16/2012 (April 2012)
107. http://developer.download.nvidia.com/assets/cuda/files/CUDADownloads/TechBrief_Dynamic_Parallelism_in_CUDA.pdf Dynamic Parallelism in CUDA Tesla K20 Kepler GPUs - Prelease of NVIDIA CUDA 5.0
108. http://developer.nvidia.com/cuda-downloads NVIDIA Developer ZONE - CUDA Downloads CUDA TOOLKIT 4.2
109. http://developer.nvidia.com/gpudirect NVIDIA Developer ZONE – GPUDirect
110. http://developer.nvidia.com/openacct OpenACC – NVIDIA
111. http://developer.nvidia.com/cuda-toolkit Nsight, Eclipse Edition Pre-release of CUDA 5.0, April 2012
112. The OpenCL Specification, Version 1.1,</B> Published by Khronos OpenCL Working Group, Aaftab Munshi (ed.), 2010.
113. NVIDA CUDA C Programming Guide Version V4.0, May 2012 (5/6/2012) http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf
114. http://www.khronos.org/files/opencl-1-1-quick-reference-card.pdf The OpenCL 1.1 Quick Reference card.
115. NVIDIA Developer Zone http://developer.nvidia.com/category/zone/cuda-zone
116. NVIDIA CUDA Toolkit 4.0 (May 2012) http://developer.nvidia.com/cuda-toolkit-4.0

References & Acknowledgements

References

117. Randi J. Rost, OpenGL – shading Language, Second Edition, Addison Wesley 2006
118. GPGPU Reference http://www.gpgpu.org
119. NVIDIA http://www.nvidia.com
120. NVIDIA tesla; http://www.nvidia.com/object/tesla_computing_solutions.html
121. NVIDIA CUDA Reference; http://www.nvidia.com/object/cuda_home.html
122. CUDA sample source code: http://www.nvidia.com/object/cuda_get_samples.html
123. href://www.nvidia.com/object/cuda_learn_products.html List of NVIDIA GPUs compatible with CUDA:
124. Download the CUDA SDK: www.nvidia.com/object/cuda_get.html
125. Specifications of nVIDIA GeForce 8800 GPUs:
126. RAPIDMIND http://www.rapidmind.net
127. Peak Stream - Parallel Processing (Acquired by Google in 2007) http:/www.google.co
128. http://www.guru3d.com/news/sandra-2009-gets-gpgpu-support/
129. AMD http:www.amd.com
130. AMD Stream Processors http://ati.amd.com/products/streamprocessor/specs.html
131. RAPIDMIND & AMD http://www.rapidmind.net/News-Aug4-08-SIGGRAPH.php
132. http://www-graphics.stanford.edu/projects/brookgpu/ Merrimac - Stream Arch. Standford Brook for GPUs
133. Standford : Merrimac - Stream Architecture http://merrimac.stanford.edu/
134. ATI RADEON - AMD http://www.canadacomputers.com/amd/radeon/
135. ATI & AMD - Technology Products http://ati.amd.com/products/index.html
136. Sparse Matrix Solvers on the GPU ; conjugate Gradients and Multigrid by Jeff Bolts, Ian Farmer, Eitan Grinspum, Peter Schroder , Caltech Report (2003); Supported in part by NSF, nVIDIA, etc....
137. Scan Primitives for GPU Computing by Shubhabrata Sengupta, Mark Harris*, Yao Zhang and John D Owens University of California Davis & *nVIDIA Corporation Graphic Hardware (2007).
138. Horm D; Stream reduction operations for GPGPU applciations in GPU Genes 2 Phar M., (Ed.) Addison Weseley, March 2005; Chapter 36, pp. 573-589 Graphic Hardware (2007).
139. Bollz J., Farmer I., Grinspun F., Schroder F : Sparse Matris Solvers on the GPU ; Conjugate Gradients and multigrid ACM Transactions on Graphics (Proceedings of ACM SIGRAPH 2003) 22, 2 (Jul y2003) pp 917-924 Graphic Hardware (2007).
140. NVIDIA CUDA Compute Unified Device Architecture – Prog. Guide - Ver1.1 November 2007

References & Acknowledgements

References

141. Tom R. Halfhill, *Number crunching with GPUs PeakStream Math API Exploits Parallelism in Graphics Processors, Ocotober 2006; Microprocessor* http://www.mdronline.com
142. Tom R. Halfhill, *Parallel Processing with CUDA Nvidia's High-Performance Computing Platform Uses Massive Multithreading ;* Microprocessors, Volume 22, Archive 1, January 2008 http://www.mdronline.com
143. J. Tolke, M.Krafczyk *Towards Three-dimensional teraflop CFD Computing on a desktop PC using graphics hardware Institute for Computational Modeling in Civil Engineering,* TU Braunschweig (2008)
144. I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Hoston, P.Hanrahan, Brook for GPUs ; *Stream Computing on GRaphics Hadrware, ACM Tran. GRaph (SIGGRAPH) 2008*
145. Z. Fan, F. Qin, A.E. Kaufamm, S. Yoakum-Stover*, GPU cluster for Hgh Performance Computing in : Proceedings of ACM/IEEE Superocmputing Conference 2004 pp. 47-59.*
146. J. Kriiger, R. Wetermann, *Linear Algeria operators for GPU implementation of Numerical Algorithms ACm Tran, Graph (SIGGRAPH) 22 (3) pp. 908-916. (2003)*
147. Tutorial SC 2007 SC05 : *High Performance Computing with CUDA*
148. *FASTRA* http://www.fastra.ua.ac.bc/en/faq.html
149. *AMD Stream Computing software Stack ;* http://www.amd.com
150. *BrookGPU :* http://graphics standafrod.edu/projects/brookgpu/index.html
151. *FFT – Fast Fourier Transform* www.fftw.org
152. BLAS – *Basic Linear Algebra Suborutines –* www.netlibr.org/blas
153. LAPACK : *Linear Algebra Package –* www.netlib.org/lapack
154. Dr. Larry Seller, Senipr Principal Engineer; Larrabee : A Many-core Intel Architecture for Visual computing, Intel Deverloper FORUM 2008
155. *Tom R Halfhill,* Intel's Larrabee Redefines GPUs – Fully Programmable Many core Processor Reaches Beyond Graphics, Microprocessor Report September 29, 2008
156. Tom R Halfhill AMD's Stream Becomes a River – Parallel Processing Platform for ATI GPUs Reaches More Systems, Microprocessor Report December 2008
157. AMD's ATI Stream Platform http://www.amd.com/stream
158. General-purpose computing on graphics processing units (GPGPU) http://en.wikipedia.org/wiki/GPGPU
159. Khronous Group, OpenGL 3, December 2008 URL : http://www.khronos.org/opencl

References & Acknowledgements

References

160. NVIDA CUDA C Programming Guide Version V4.0, May 2012 (5/6/2012) http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf
161. NVIDIA Developer Zone http://developer.nvidia.com/category/zone/cuda-zone
162. NVIDIA CUDA Toolkit 4.0 (May 2012) http://developer.nvidia.com/cuda-toolkit-4.0
163. NVIDIA CUDA Toolkit 4.0 Downloads http://developer.nvidia.com/cuda-toolkit
164. NVIDIA Developer ZONE – GPUDirect http://developer.nvidia.com/gpudirect
165. NVIDIA OpenCL Programming Guide for the CUDA Architecture version 4.0 Feb, 2012 (2/14,2012) http://developer.download.nvidia.com/compute/DevZone/docs/html/OpenCL/doc/OpenCL_Programming_Guide.pdf
166. Optimization : NVIDIA OpenCL Best Practices Guide Version 1.0 Feb 2012 http://developer.download.nvidia.com/compute/DevZone/docs/html/OpenCL/doc/OpenCL_Best_Practices_Guide.pdf
167. NVIDIA OpenCL JumpStart Guide - Technical Brief http://developer.download.nvidia.com/OpenCL/NVIDIA_OpenCL_JumpStart_Guide.pdf
168. NVIDA CUDA C BEST PRACTICES GUIDE (Design Guide) V4.0, May 2012
169. http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Best_Practices_Guide.pdf
170. NVIDA CUDA C Programming Guide Version V5.0, May 2012 (5/6/2012)
171. http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf
172. Programming Massively Parallel Processors - A Hands-on Approach, David B Kirk, Wen-mei W. Hwu, Nvidia corporation, 2010, Elsevier, Morgan Kaufmann Publishers, 2011
173. Aftab Munshi Benedict R Gaster, timothy F Mattson, James Fung, Dan Cinsburg, Addison Wesley, OpenCL Progrmamin Guide, Pearson Education, 2012
174. The OpenCL 1.2 Specification Khronos OpenCL Working Group
175. http://www.khronos.org/files/opencl-1-2-quick-reference-card.pdf" The OpenCL 1.2 Quick-reference-card ; Khronos OpenCL Working Group
176. http:/www.openmp.org OpenMP 4.0

References

177. Randi J. Rost, OpenGL – shading Language, Second Edition, Addison Wesley 2006
178. GPGPU Reference http://www.gpgpu.org
179. NVIDIA http://www.nvidia.com
180. NVIDIA tesla http://www.nvidia.com/object/tesla_computing_solutions.html
181. RAPIDMIND http://www.rapidmind.net
182. Peak Stream - Parallel Processing (Acquired by Google in 2007) http:/www.google.com
183. guru3d.com http://www.guru3d.com/news/sandra-2009-gets-gpgpu-support/
    ATI & AMD http://ati.amd.com/products/radeon9600/radeon9600pro/index.html
184. AMD http:www.amd.com
185. AMD Stream Processors http://ati.amd.com/products/streamprocessor/specs.html
186. RAPIDMIND & AMD http://www.rapidmind.net/News-Aug4-08-SIGGRAPH.php
187. General-purpose computing on graphics processing units (GPGPU)
    http://en.wikipedia.org/wiki/GPGPU
188. Khronous Group, OpenGL 3, December 2008 URL : http://www.khronos.org/opencl
189. *OpenCL - The open standard for parallel programming of heterogeneous systems* URL :
    http://www.khronos.org/opencl
190. Programming the GPU and a brief intro to the OPENGL shading language – Marcel Cohan & VVR Talk
191. David B Kirk, Wen-mei W. Hwu nvidia corporation, 2010, Elsevier, Morgan Kaufmann Publishers, 2011
192. Benedict R Gaster, Lee Howes, David R Kaeli, Perhadd Mistry Dana Schaa, Heterogeneous Computing with OpenCL, Elsevier, Moran Kaufmann Publishers, 2011
193. The OpenCL 1.2 Specification (Document Revision 15) Last Released November 15, 2011
    Editor : Aaftab Munshi Khronos OpenCL Working Group
194. The OpenCL 1.1 Quick Reference card

References

195. http://developer.amd.com/sdks/AMDAPPSDK/downloads/Pages/default.aspx AMD APP SDK with OpenCL 1.2 Support
196. http://developer.amd.com/sdks/AMDAPPSDK/downloads/Pages/default.aspx#oneAMD-APP-SDKv2.7 (Linux) with OpenCL 1.2 Support
197. http://icl.cs.utk.edu/magma/software/ MAGMA OpenCL
198. http://developer.amd.com/zones/OpenCLZone/pages/GettingStarted.aspx Getting Started with OpenCL
199. http://developer.amd.com/openclforum AMD Developer OpenCL FORUM
200. http://developer.amd.com/zones/OpenCLZone/programming/pages/benchmarkingopenclperformance.aspx AMD Developer Central - Programming in OpenCL - Benchmarks performance
201. http://developer.amd.com/sdks/AMDAPPSDK/assets/opencl-1.2.pdf OpenCL 1.2 (pdf file)
202. http://developer.amd.com/zones/opensource/pages/ocl-emu.aspx AMD OpenCL Emulator-Debugger
203. http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf The OpenCL 1.2 Specification (Document Revision 15) Last Released November 15, 201 Editor : Aaftab Munshi <I> Khronos OpenCL Working Group
204. http://www.khronos.org/registry/cl/sdk/1.1/docs/man/xhtml/ OpenCL1.1 Reference Pages
205. The Intel SDK for OpenCL Applications XE – Optimization Guide includes many more details.

Source : Intel, NVIDIA, Khronos AMD, References

An Overview of Prog. Env on Intel Xeon-Phi

Summary

- An Overview of Intel Xeon-Phi Coprocessor Architecture & Software Environment is discussed
- Programming paradigms on Intel Xeon-Phi Coprocessor are discussed
- Tips for Tuning & Performance Issues on Intel Xeon-Phi Coprocessor are discussed

Thank You

Any questions ?