Intel® Xeon Phi™ Coprocessor Architecture for Software Developers

ABSTRACT: This article helps the reader develop an understanding of the design decisions behind the Intel Xeon Phi™ coprocessor micro-architecture and how it complements the Intel® Xeon product line, provides a brief refresher of modern computer architecture, and describes various aspects of the Intel Xeon Phi™ architecture at a high level. The reader will develop an understanding of Intel® Manycore architecture and how it addresses the massively parallel one chip computational challenge. It will introduce in summary the capabilities and limitations as well as key impact points for the software and hardware evaluators considering this platforms for technical computing in order to set the stage for the deeper discussions in following chapters.

==========

Technical computing can be defined as application of mathematical and computational principles to solve engineering and scientific problems. It has become an integral part of research and development of new technologies in modern civilization. It is used in industry and academia equally to produce new products, produce weather forecasting, enhance geosciences exploration, financial modeling, car crash simulation, electromagnetic field propagation from mobile phones, and so on.

Computer technology has made substantial progress over the last couple of decades by introducing superscalar processors with pipelined vector architecture. We have also seen the rise of parallel processing in the lowest computational segment, such as handheld devices. Today one can buy as much computational power as earlier supercomputers for less than a thousand dollars.

However, current computational power still is not enough for the type of research needed to push the edge of understanding of the physical processes addressed by technical computing applications. Massively parallel processors like the Intel® Xeon Phi™ product family have been developed to increase the computational power to remove these research barriers. However, to exploit Intel® Many Integrated Core architecture (Intel® MIC) coprocessors capable of providing more than teraflops of double-precision floating-point performance in technical computing applications will require careful design of algorithms and data structures, as well as an understanding to match the hardware capabilities and idiosyncrasies of these processor architectures. This article gives an in-depth look at the Intel Xeon Phi coprocessor architecture and corresponding parallel data structure and algorithm used by various technical computing applications that will be suitable for such a coprocessor. We shall also look at the various sources code–level optimizations that can be done to exploit some of the processor features.

Processor micro-architecture describes the arrangements and relationship between different components to perform the computation. In recent years with the advent of semiconductor technologies, hardware companies were able to put many processing cores on a die and interconnect them intelligently to allow massive computing power in the range of teraflops (trillions of mathematical operations per second) of double-precision arithmetic, which was achieved first by the supercomputer ASCI Red in the not-so-distant past in 1996.

History of Intel® Xeon Phi™ Development

Intel Xeon Phi started its life while looking for a solution to reduce power consumption of Intel Xeon family of processors developed around 2001. It was determined that simple low frequency Intel Manycore architecture with appropriate software support would be able to produce better performance/watt efficiency. This required a new micro architectural design. The question was, could we use the x86 cores for it? The answer was yes, since the instruction set architecture (ISA) needed for x86 compatibility dictates a small percentage (less than 10 percent) of power consumption, whereas the hardware implementation and circuit complexity dictates most of the power dissipation in a general-purpose processor.

The architecture team played on a simulator with various architecture features, like removing out-of-order execution, hardware multithreading, long vectors, and so on, to develop a new architecture that could be applied to throughput-oriented workloads. A graphics workload fits nicely as throughput-oriented work, where many threads can work in parallel to compute the final solution.

The design team focused on in-order core, x86 ISA, smaller pipeline, wider SIMD and SMT. So they started with Pentium® 5 cores connected through ring interface and added fixed-function units such as a texture

sampler to help with graphics. The design goal was to create architecture with the proper balance between chip-level multiprocessing with thread and data-level parallelism. The simulator was used to understand various performance issues and tune the core and un-core design.

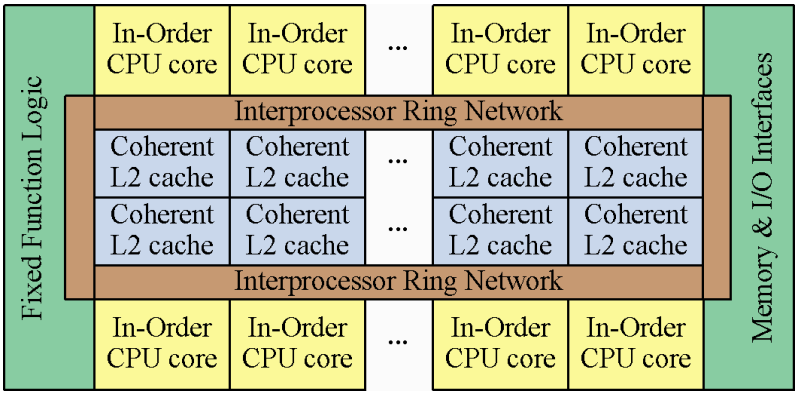

Figure 1 Early Intel® MIC Architecture based Silicon (Codenamed Larrabee) Block Diagram (source: http://software.intel.com/sites/default/files/m/9/4/9/larrabee_manycore.pdf)

In addition to understanding the use of such technology in graphics, Intel also recognized that there are various scientific and engineering applications that may benefit from many core architectures. These applications were highly compute intensive, thread and process scalable, and can benefit from the manycore architecture. During this time period the HPC industry also started playing around with using graphics cards for general-purpose computation. It was obvious that there was promise to such technology.

At this time a handful of software engineers from Intel were able to demonstrate that one could make computational kernels to be speed up quite a bit with such low frequency, highly parallel architecture that overall application performance would improve even in a coprocessor model, and they requested funding for such a project. During the same time period, with the industry and academia experimenting with the coprocessor model for technical computing and scientific applications, the project was approved and the first proof of concept work was started on "Larrabee 1,"shown in Figure 1, which was named by Intel marketing team as Knights project and work on project codenamed Knights Ferry solution as a proof of concept started. The visual computing product team within Intel started developing software targeted towards technical computing applications. Although the hardware did not change, their early drivers were based on graphics software needs and catered to graphics API needs, which were mainly Windows† based at that point.

The first thing we recognized was that a big part of technical and scientific computing was done on the Linux† platform. So the first step was to create software support for Linux. We also needed to develop a programming language that could leverage the existing skills of the software developers to create multithreaded applications using MPI and OpenMP with the C, C++, and Fortran languages. The Intel compiler team went to the drawing board to define language extensions that would allow users to write applications that could run on coprocessors and host at the same time to leverage the compute power of both. Other Intel Tools teams like cluster tools (MPI), Debugger, Amplifier XE, Math Kernel Library, and the Numeric team all went back to the design board to make their tools and libraries to support the new coprocessor architecture.

As the hardware were x86 cores, the driver team ported a modular microkernel that was based on standard Linux kernel source. The goal of the first phase of development was to prove and hash out the usability of the tools and language extensions that Intel was making. The goal was to come out with a hardware and software solution that could fill the need of technical computing applications. The hardware roadmap included a new hardware architecture code-named KNC that could provide 1 teraFLOP of double-precision performance with reliability and power management features required by such computations. This hardware was later marketed as Intel Xeon Phi and is what is covered in this article.

Evolution From Von Neumann Architecture to Intel® Xeon Phi™ Architecture

There are various functional units in modern day computer architecture that need to be carefully designed and developed to achieve target power/performance. The center of these functional units is a generic programmable processor that works in combination with other components like memory, peripherals and other co-processors to perform its tasks. It is important to understand the basic computer architecture to get the grasp of Intel® Xeon Phi™ architecture since in essence it is a specialized architecture with many of the components used in designing a modern parallel computer.

Commonly used computer architecture is known as Von Neumann architecture shown in Figure 2.

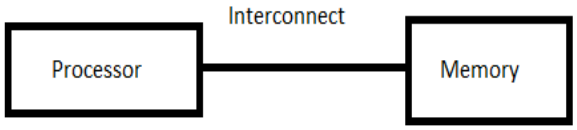

Figure 2 - Von Neumann Architecture

In this basic but fundamental design the processor is responsible for arithmetic and logic operations and gets its data and instructions from the memory. It fetches instructions from memory pointed to by instruction pointer and executes the instruction. If the instruction needs a data, it also collects the data from the memory location pointed to by instruction and executes on it.

Over the last few decades computer architecture has evolved from this basic Von Neumann architecture to accommodate for some physical necessities like need for faster data access to implement cache subsystem. It is also seen that depending on the computational task at hand demand is made for different part of the computer architecture. This article is concerned about scientific computing and hence our focus will remain on Xeon Phi™ architecture in the context of scientific computing.

For scientific computing, it is often found that many modern day computations depend on fast access to the data it needs. The processors at high-level are now designed with two distinct but important components known as the core and uncore. Couple of decades ago, the core was assumed to be the most important component of computer architecture and was subject to a lot of research and development. The uncore components of modern day computers play more fundamental role in scientific application performance and often consumes more power and silicon chip area than the core. The core components consist of engines that do the computations. These include vector units in many of the modern processors. The uncore components includes cache, memory and peripheral components.

The Cache Subsystem

General computer architecture with cache subsystem is designed to reduce the memory bandwidth/latency bottleneck discovered in the Von Neumann architecture. A cache memory is a high speed memory with low latency and bandwidth connection to the core to supply data to instructions executing in the core. A subset of data currently being worked on by a computer program is saved in the cache to speed up instruction execution based on generally observed temporal and spatial locality of computer programs. The general architecture of such a computer is shown in Figure 3. It can be seen that a cache is added to the processor core and connect through a memory controller (MC) to communicate with the main memory. The memory controller on modern chips is often fabricated on die for reducing the memory access latency.

One common cache architecture design progression is to introduce and vary multiple levels of caches between the core and the main memory to reduce the access latency and interconnect bandwidth. The cache design is still developing as memory bottlenecks are becoming more severe as the processor technology evolves. New memory technologies and semiconductor processes are allowing processor designers to play with various cache configurations as the architecture evolves.

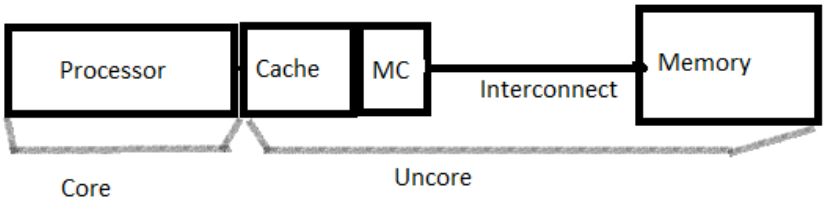

Figure 3 - Computer Architecture with cache memory. The memory controller is responsible to manage data movement to / from the processor.

We shall see in subsequent chapters that cache subsystem plays an extremely important role in application performance on a given computer architecture. In addition, the introduction of cache to speedup applications introduces a cache coherency problem in a manycore system. These results from the fact that the data updated in the cache may not reflect the data in the memory for the same variable. This gets even more complex when the processor implements multi-level cache.

There are various protocols designed to make the data in the cache of each cores of a multicore processor to remain consistent when they are modified to maintain applications correctness. We shall cover one such protocol implemented in Intel® Xeon Phi™ in chapter [update].

During these developments, the computer architecture was inherently single threaded from hardware perspectives although clever time sharing process developed and supported in computer operating systems gave the users the illusion of multiple processes being run by the computer simultaneously. We shall see in subsequent sections that each of these core component of the basic computer architecture shown in Figure 3 has evolved over time and functionalities added to achieve the current version of Xeon Phi coprocessor architecture.

Core Improvements

Instruction Level Parallelism

With the development of better semiconductor process technologies, computer architects were able to execute more and more instructions in parallel and pipelined fashion and implemented what is called instruction level parallelism.

The instructions those are executed in a processors goes through several stages as they flow through logic circuits at the rhythm of heartbeat known as core clocks. At each clock a part of the instruction is executed. However, it is possible to stagger multiple instructions together so that the various stages of multiple instructions can be executed in the same cycle. This is the principle behind the pipelined executions.

All computer instructions based on Von Neumann architecture goes through certain high level basic stages. These are instructions fetch (IF) where the next instruction to be executed by the core is accessed. The

instructions usually reside in the instruction cache or fetched from main memory and cache hierarchy at this stage. Note that each stage will take minimum of one cycle but may extend further if it gets blocked on some resource issue. For example, if the instructions to be executed are not in the cache, it has to be fetched from memory and in worst case from a nonvolatile storage area like hard disk, solid state disk or even flash memory.
Once the instructions are fetched, it has to be decoded to understand how to execute the instruction. Now the instruction usually works on some sort of data which could be in the register (fastest memory nearest to the core) or a memory. The semantics of the instructions is defined by some rules and behavioral model known as instruction set architecture or ISA.

The decoded instruction than moves on to the next the execution stage [E] where all necessary memory or cache access happens and execution completes when all the necessary data is available. Otherwise a pipeline stall may happen to wait for data to come to memory. Once the E stage completes, the data is written back [WB stage] to memory/register or flags updates to change the processor state.

We shall see as we follow the evolution of the Von Neumann architecture, that each of the execution stages themselves are modified to satisfy the constant increase in computing demand from computer users.

The first thing the engineers and scientists wanted to implement is how to increase the instruction level parallelism so that more than one instructions can be executed at the same time and also increase the rate of computation by increasing the clock rate at which these instructions executes.

Instruction Pipelining

Execution stage itself may take multiple cycles to account for complexity of the semantics of that instruction. The fundamental pipelining described above is shown in Figure 4. Note that this is a very simplified step of a complex instruction execution phase and may not resemble the execution stages at the detailed level. However we will see that even if today's processor executes instructions in much higher complexity for each of these stages, in essence we still follow the high level classical instruction stages described here.

Figure 4 Pipeline Stages for an instruction execution

Figure 5 shows how the pipelining process helps various stages of two different instructions to overlap thus providing instruction level parallelism. In this figure the first instruction inst1 after being fetched from memory enters the instruction decode stage. Since these stages are executed in different hardware components, the second instruction fetch can happen when the first instruction is in the decode stage. So in clock tick 2, the first instruction is decoded and the second instruction is fetched thus overlapping execution of two instructions.

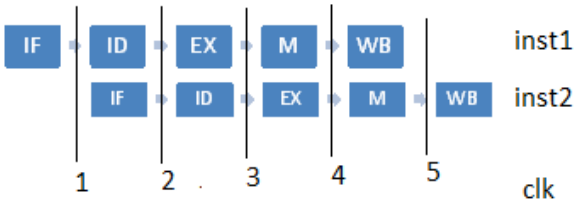

Figure 5 : Instruction pipeline showing two instructions executing at the same clock cycle but at different stages

However, processor engineers were looking for more parallelism to satisfy the demand of computer users wanting to execute faster and complex applications on these hardware. The next development happened where more than one pipeline was created to execute more than one instruction to be executed in parallel in the same cycle. In this case, some of the hardware functions were replicated so that more in addition to pipelining described in Figure 5, two independent instructions can be executed in two different pipelines. Thus they can be both in execution stage at the same clock cycle. This architecture is generally known as super scalar architecture. One such architecture which was in wide use in early nineties was Intel® P5 architecture. The Intel® Xeon Phi™ core is based on such architecture and contains two independent pipelines known as U and V pipelined. The names 'U' and 'V' does not have any significance to it and often an outcome of some Intel engineers random choice of letters to name such units. We shall later see in details how the instructions can be dispatched to two pipelines and the limitation on which instructions can be dispatched and how in these super scalar architecture.

At this stage, engineers figured out that they can speed up computation by increasing the core clock speed and thus stared the march towards increasing clock rate. However, increased clock rate required that each of the stages described above be broken into several sub stages to be able to execute with each clock tick. At some point the number of stages shown above had increased over 30 from 5 basic stages shown above to accommodate faster processor clock rate. This resulted in faster and faster processor that can execute a single thread of instructions at a speed that was improving with clock rate improvement in each subsequent processor generation.

However, this progress hit a brick wall commonly known as 'power wall' since increased clock rate was resulting in too much wasted energy in the form of heat. So engineers went back to design board and in case of Intel® Xeon Phi® the instructions goes through fewer number of stages than the Pentium 4 families of processors.

Another way to improve instruction level parallelism was through introduction of out of order instruction processing. Where the hardware had components to detect independent instructions and by increasing resources in the processor units, was able to execute them out of order and in many case in speculative fashion from the program enforced ordering dictated by a sequential code. However, to maintain the consistency semantics of program execution which requires that processor state should be changed as the programmer desired in the original code, the write back stage (WB) was maintained in the program order.

Multithreading

As the processor frequency was coming down to reduce power dissipation resulting from high speed switching, the engineers turned to hardware multithreading to increase parallelism. In this case, many of the processors resources are replicated in hardware so that applications can indicate to the operating system that it can execute multiple instruction streams in parallel through high level parallelism constructs like OpenMP*, thread*. MPI* etc. To the operating system this looked like a multiple processors working together to achieve the performance it wants.

In Intel® Xeon Phi™ Processor, there are four hardware threads sharing the same core and appearing as if the there are four processors connected to a shared cache subsystem as shown in Figure 6. In this case the multithreading support in the core is displayed as logical processors as they still share some resources among themselves.

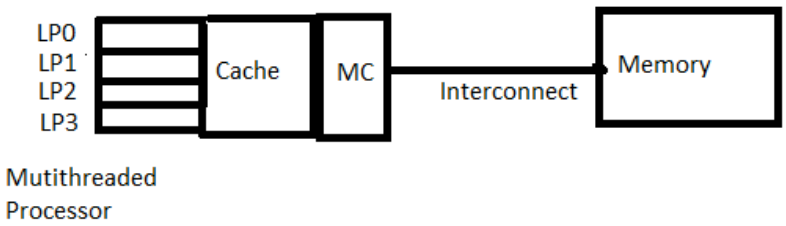

Figure 6: Multithreaded processor cores with Super Scalar Execution Units. The LP0-3 in the diagram indicates logical processors. MC indicates the memory controller controlling data flow to/from the logical processors.

Manycore Architecture

Next evolution in the processor architecture was a logical evolution from multithreading where rather than sharing some of the resources needed for instruction execution, the designer cloned the whole core multiple times to allow multiple threads of execution to happen in parallel. This type of manycore architecture where all cores are similar is known as homogeneous manycore architecture. There are other possibilities like heterogeneous manycore architecture where all the cores in the processor may not be identical.

The evolution to manycore architecture allowed applications to improve performance without increasing the clock frequencies. But this moved the burden of achieving application performance improvement more from hardware engineers towards software engineers. Software engineers and computer scientists leveraged years of experience in developing parallel applications used in technical computing and HPC application to start exploiting the manycore architecture. Although the parallel constructs to exploit such machines are still in infancy, there are sufficient tools to start developing for these machines at the time of this writing. Figure 7 shows the initial thinking of architectures where more than one core was made part of a processor. In this case the cores (indicated by P0-Pn) were connected to a common interconnect known as bus through cache subsystem (indicated by 'C' in the diagram) and shared the bus bandwidth with each other.

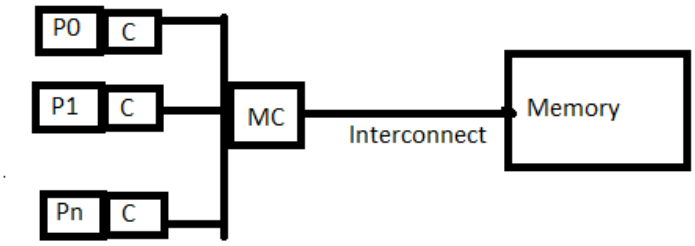

Figure 7: Evolving architecture towards a manycore processor-based computer. Here 'C' indicates cache, MC indicates memory controller and Px indicates processor cores.

Interconnect improvements

While using multiple cores to create a processor it was soon discovered that single shared interconnect architecture like bus used in some early processor design is a bottleneck towards extracting parallel application performance. On the other hand, computer designers have worked with interconnection technologies for super computers for a while when the design of Intel® Xeon Phi™ processor was undertaken. So the architects designing the interconnect had a knowledge base to refer to for developing these architectures.

The interconnect topology selected for a manycore processor is determined by the latency, bandwidth and cost of implementing such technology. The interconnect technology chosen for Intel® Xeon Phi™ is a bidirectional ring topology. Here all the cores talk to each other including memory through memory controller through a bidirectional interconnect. We shall look at the interconnect technology implemented in Xeon Phi in subsequent chapter. Note that as new designs comes out, the interconnect technology will also evolve to provide low latency/high bandwidth network.

Figure 8 shows the evolution of manycore processor depicted in Figure 7 where the cores are connected in a ring network. Note that the ring network allows memory to be connected to the network through a memory controller responsible for getting data to the cores as requested. There may be one or more memory controller to improve the memory BW.

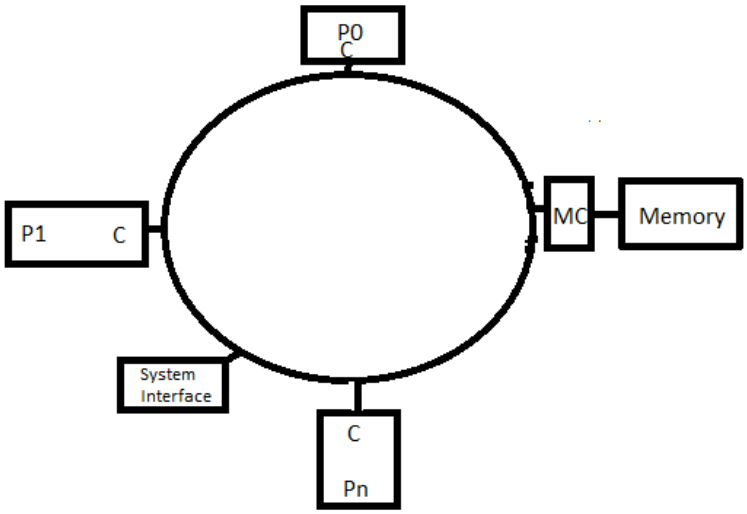

Figure 8: Manycore processor architecture with cores connected through a ring bus. In this figure P0-Pn indicates the cores, 'C' indicates the cache and MC indicates the memory controller.

System Interconnect

In addition to talking to memory through memory interconnect, coprocessors like Intel® Xeon Phi™ also are often placed on PCIe slots to work with the host processors like Intel® Xeon Processors. This is done by incorporating system interface logic that can support standard I/O protocol like PCI express to communicate with the host. In Figure 8, the system interface controller is shown as another box connected to ring described as system interface controller.

Figure 9 shows a system level view of Xeon Phi coprocessor working with a host processor over PCI express interface. Note that the data movement between the host memory and Xeon Phi memory can happen through DMA without host processor intervention in certain cases which will be covered in the detailed section. It is possible to connect multiple Intel® Xeon Phi™ cards to the host system to increase computational power.

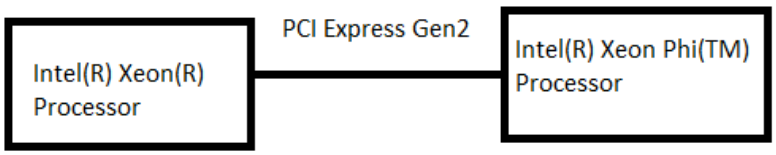

Figure 9: System with Intel® Xeon Phi™ Coprocessor

Figure 10 is that of a Intel® Xeon Phi™ Coporcessor packaged as a PCIe gen2 based card. This card can be placed on a validated server or workstation platforms in various configurations to complement parallel processing power of host processors line Intel® Xeon® processor.

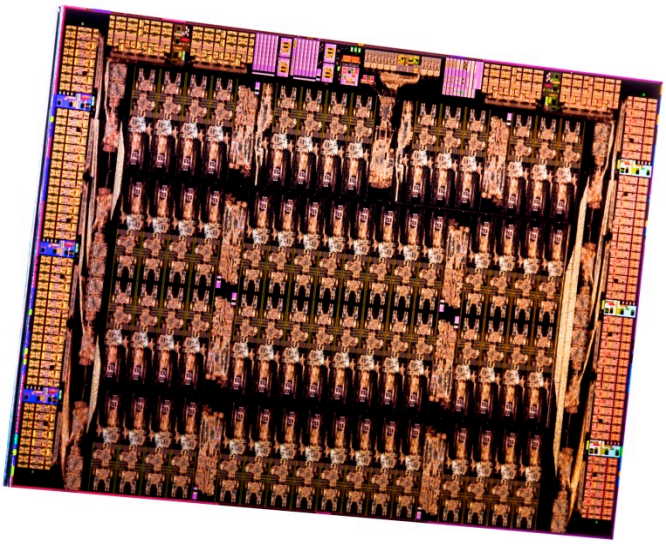

Figure 9 Picture of Intel® Xeon Phi™ Coprocessor and Wafer Photo.
Source:http://newsroom.intel.com/docs/DOC-3126#multimedia

Intel® Xeon Phi™ Coprocessor Chip Architecture

This chapter looks at various functional units of the Intel Xeon Phi coprocessor and why they are designed the way they are.

Figure 10 shows a simple diagram of the logical layout of critical chip components of the Intel Xeon Phi coprocessor architecture. This coprocessor consists of the following components, some of which are shown in Figure 10:

- Coprocessor cores: These are based on P54c (Intel® Pentium® from 1995) cores with major modifications including Intel® 64 ISA, 4-way SMT, new vector instructions, and increased cache sizes.
- VPU: Vector Processing Units capable of performing 512-bit vector operations on 16 single-precision or 8 double-precision floating-point arithmetic operations as well as integer operations.
- L2 Cache: This is the L2 cache and uncore interface.
- Ring Interconnect: Interconnect between cores and rest of the coprocessor's components like memory controllers, PCI interface chip, and so on.
- Memory Controller: Interface between the ring and the GDDR memory.
- PCIe interface: To connect with PCIe bus.

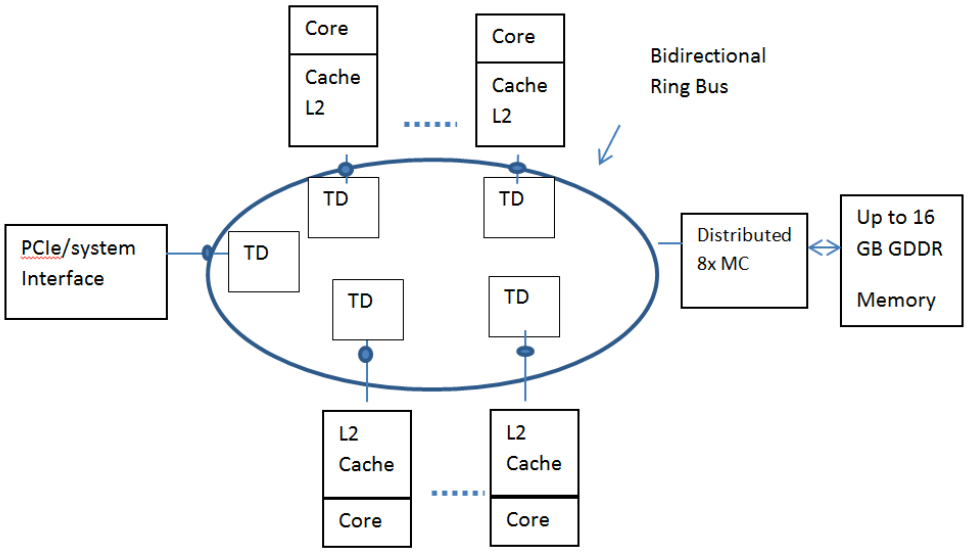

Figure 10 Logical Layouts of Functional Components

Intel Xeon Phi consists of up to 62 Intel architecture cores. However, for these many cores and functional units to access and communicate with each other, a carefully designed interconnect(s) is needed to hold the memory/data/control traffic between the cores and various parts of the chip. Figure1.12 shows the logical layout of the Intel Xeon Phi coprocessor. Actual physical layout of the individual functional units may be vastly different from what is depicted here. For example, the eight memory controllers shown in the diagram are physically distributed on the ring for optimal memory access latency. The L2 caches are fully coherent with each other. Coherency is maintained by GOALS coherency protocols. The functional units communicate with each other by on-die bidirectional interconnect.

Each of the eight memory controllers has two channels giving a total of 16 channels that can run up to 5.5 GT/s. This can provide an aggregate bandwidth (BW) of BW=Number of Channels × 4 bytes/channel × 5.5 GT/s = 16 × 5.5 × 4 =352 GB/s theoretical peak.

The system interface of the chip supports PCIe2 x 16 protocols with 256-byte packets.

The chip also provides reliability features useful in a technical computing environment. These include parity support in L1 cache, ECC on L2 and memory transactions, CRC and command-address parity on memory I/O.

==========

This article is based on material found in the book *Intel® Xeon Phi™ Coprocesser Micro Architecture and Tools.* Visit the Intel Press web site to learn more about this book:

http://noggin.intel.com/intelpress/categories/books/intel%C2%AE-xeon-phi%E2%84%A2-coprocesser-micro-architecture-and-tools

Also see our Recommended Reading List for related topics: www.intel.com/technology/rr

About the Author

Reza Rahman is a Sr. Staff Engineer at Intel Software and Services Group. Reza lead the worldwide technical enabling team for Intel Xeon Phi™ product through Intel software engineering team. He played a key role during the inception of Intel MIC product line by presenting value of such architecture for technical computing and leading a team of software engineers to work with 100s of customers outside Intel Corporation to optimize code on Intel® Xeon Phi™. He worked internally with hardware architects and Intel compiler and tools team to optimize and add features to improve performance of Intel MIC software and hardware components to meet the need of technical computing customers. He has been with Intel for 19 years. During his first seven years he was at Intel Labs working on developing audio/video technologies and device drivers. Rest of the time at Intel he has been involved in optimizing technical computing applications as part of software enabling team. He is also believes in standardization process allowing software and hardware solutions to interoperate and has been involved in various industry standardization group like World Wide Web consortium (W3C).

Reza holds a Masters in computer Science from Texas A&M university and Bachelors in Electrical Engineering from Bangladesh University of engineering and Technology.

==========

