ISSN: 2350-0328
International Journal of Advanced Research in Science, Engineering and Technology
Vol. 4, Issue 9 , September 2017

# Architecture, Programming and Performance of MIC Phi Coprocessor

**JanuszKowalik, Piotr Arłukowicz**

Professor (ret), The Boeing Company, Washington, USA
Assistant professor, Faculty of Mathematics, Physics and Informatics, University of Gdańsk, Poland

**ABSTRACT**: The article is showing the advantage of using Xeon PHI processors along with well-established technologies like MPI and OpenMP over newer but more troublesome OpenCL and GPGPU approach. The state-of-the art Xeon PHI technology is presented and brief summary about the architecture and performance is discussed.

**KEYWORDS**: Xeon Phi, Performance, MPI, Saxpy, Vectorization

## I. INTRODUCTION

Typical solution algorithm for scientific and technical problem is a mix of highly parallel, lowly parallel and sequential components. This led computer designers to the idea of heterogeneous computing technologies where architecturally different processors execute parts of algorithms. Two distinct heterogeneous architectures appeared between 1990 and 2012. Table 1 summarizes key features of the General Purpose Graphics Processing Units (GPGPU) and the Many Integrated Core (MIC).

Table. 1. Summary of the GPGPU and the MIC technologies.

| Two heterogeneous computer technologies GPGPU and MIC | |
|---|---|
| 1990s | 2010+ |
| GPGPU | Intel MIC technology |
| General Purpose Graphics Processing | Many Integrated Core |
| Unit highly parallel | Xeon PHI |
| ACCELERATOR | COPROCESSOR |
| Devices slaves to CPU | Peer to host CPU |
| Require new programming tools: CUDA, OpenCL | Uses well established software development languages and tools: C, C++, Fortran, OpenMP, MPI |

Both technologies objective is speeding up computation but their architecture, software and program execution modes are significantly different. GPGPU highly parallel devices are accelerators whose relationship to CPU can be called the master-slave relation. On the other hand (MIC) Phi coprocessor is a peer partner of CPU that can either collaborate with CPU or execute algorithms independently alone. Intel Phi architecture is a part of the announced in 2016 new HPC initiative called the Scalable System Framework. Intel expects this framework to be a steptowards exascale computing.

## II. ARCHITECTURE

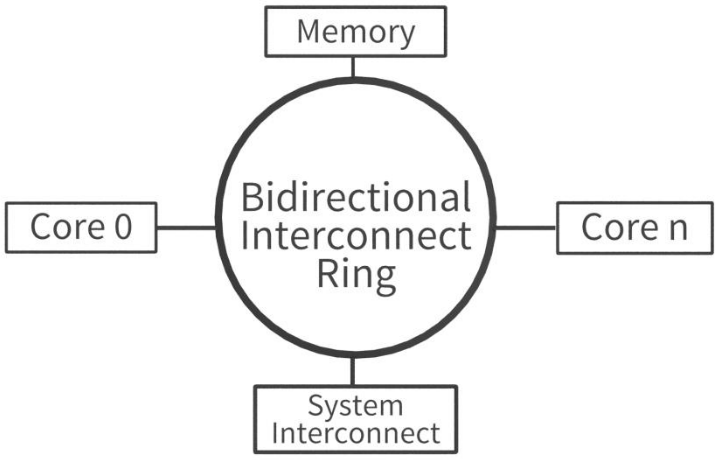

**Fig. 2. Phi coprocessor architecture**

The simplest architectural description of Phi coprocessor is the Symmetric Multiprocessor (SMP) on a chip [1]. A slightly more precise definition is the Cache Coherent Uniform MemoryAccess (ccUMA) processor. The cores are connected by a bidirectional bus shown in Fig.2. Phi clock frequency is lower than the state of the art Intel CPU and the memory size is smaller. The impressive performance of Phi comes not from clock frequency but its many core parallelism. Each core has two processing units: the scalar processing Unit (SPU) and the vector processing unit (VPU). Instruction is fetched, decoded and then executed either by SPU or VPU.

Both SPU and VPU have access to scalar and vector registers, data and instruction L1 cache memories and L2 cache. The simplest heterogeneous platform with Phi contains one or two CPU processors and one Phi coprocessor. It is important to note that these processors do not share memory. They communicate via Peripheral Communication Interface express bus (PCI). This bus is relatively slow and may become a bottleneck if improperly used.

Phi coprocessor is one of the most successful building blocks of large supercomputers. In November 2015 TOP-500 list contained 35 supercomputers using Phi coprocessors. Among them was the champion Tianhe-2 supercomputer in China and Stampede in the Texas Advanced Application Computing Center.

Phi can be regarded as a Linux’s cluster network node. This and the Phi peerage relative to host CPU justify the name coprocessor rather than accelerator although PHI can accelerate computation. Further arguments supporting the Phi status as coprocessor will be explained in the next section on Software and Programming.

## III. SOFTWARE AND PROGRAMMING

The key software observation is that software stack architecture of CPU and of Phi are almost identical. This allows user to choose from several options for executing program. We discuss these options later in this section. One of the very pregnant in consequences decision made by Intel was not designing new parallel languages for Phi. Instead Intel extended existing familiar languages and accepted by the HPC community standard programming tools. The languages include: C, C++, Fortran and OpenCL [5].

Software development tools include: OpenMP [2], MPI [3], Cilk Plus [4], Thread Building Blocks (TBB) and Intel Mathematical Kernel Library (MKL). This decision positively influenced porting existing software to platforms withPHI coprocessors.Fig.3 illustrates porting advantage for coprocessor PHI.

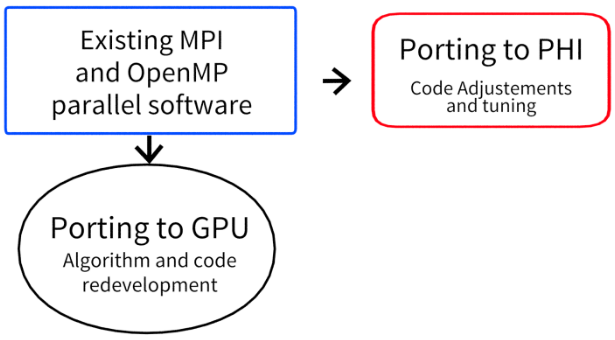
Existing MPI and OpenMP parallel software
Porting to PHI
Code Adjustements and tuning
Porting to GPU
Algorithm and code redevelopment

Fig.3. Phi porting advantage.

For the sake of fairness it should be added that GPUs have proved to be successful in application areas dealing with graphics and visualization and may be the preferred technology for corresponding users. On the other hand it might be hard for GPU to compete with the MIC technology in the application areas where OpenMP and MPI already have been used as standard tools for parallel software. Initial feedback from users who ported their OpenMP and MPI programs to computers with Phi indicates that porting has been easy and economically attractive.

MIC hardware and software architectures allow users of platforms with Phi three option of using Phi coprocessor shown in Fig.4.

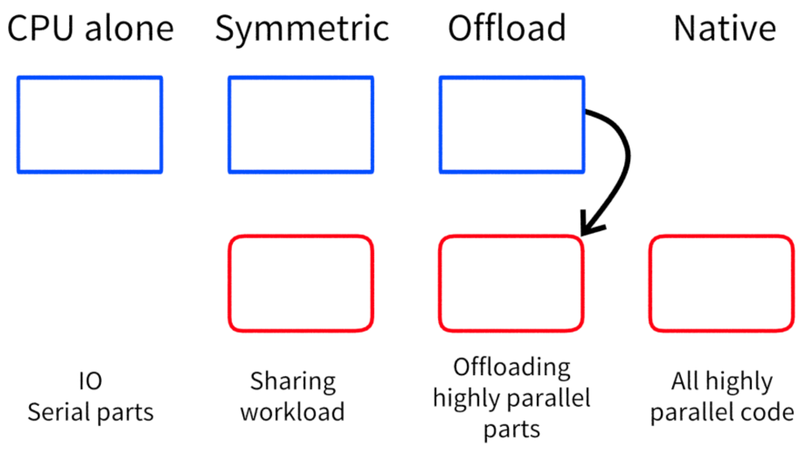

Fig.4 Three options for executing programs.

**Symmetric.**

Here workload is divided and one part is executed on CPU and the second part on Phi. This option uses the entire platform power. An example could be an MPI application where some processes are running on CPU and other on Phi. In this case the programmer should ensure well balanced workloads keeping in mind the CPU and Phi performance differences.

**Offload.**

In This option the code is compiled and executed on CPU but some highly parallel parts of the algorithm are offloaded to the Phi coprocessor. Offload is somewhat similar to the GPU style of computing.

ISSN: 2350-0328
International Journal of Advanced Research in Science, Engineering and Technology
Vol. 4, Issue 9 , September 2017

**Native.**

Phi coprocessor working alone. The code is compiled and executed on Phi coprocessor. This option is very fast for codes that are uniformly highly parallel. The offload option has two cases: firstly non-shared memory case for applications with bitwise capable data such as arrays and scalars and secondly virtual shared memory case for problems with complex data structures using pointers such as trees and linked lists. Two cases are shown in Fig. 5.

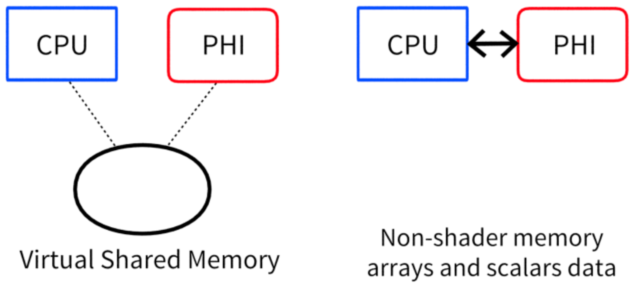

Fig. 5. Two offload cases.

Most common data types in scientific and technical application are scalars and arrays so we can use the non-shared memory case. In non-shared memory case the data are transferred between CPU and Phi via PCI. To minimize the communication penalty we minimize the ratio of transferred data size to the amount of computation. An example of favorable ratio is, if transferring n data items to Phi is followed by executing n-square operations. Consider for instance computation of matrix/vector product. If matrix A is stored in Phi then transferring vector v from CPU to Phi and computing the Av product would satisfy the desired property of minimizing ratio of transfer to computation.

## VI. PERFORMANCE

A helpful hint how we should use a heterogeneous platform. With Phi is provided by Fig.6 showing performance of CPU and Phi as functions of used threads.

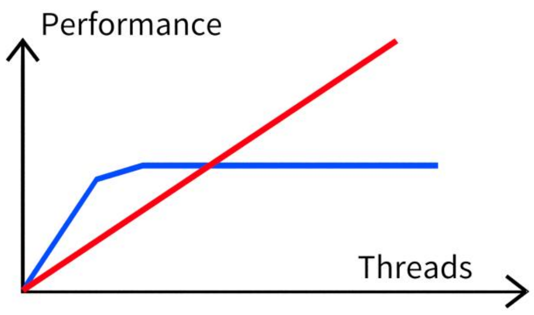

Fig.6. CPU (blue) and Phi (red) performance

The central idea of using heterogeneous computer is assigning parts of code to processors that execute them best. In this spirit I/O, sequential and lowly parallel tasks should be assigned to CPU. Phi coprocessor is best for highly parallel tasks. Obvious assignment cases include:

a) run uniformly highly parallel code on Phi coprocessor alone.
b) small problems that do not scale up to using about 50-60 cores should use CPU
c) for problems that have well defined highly parallel components consider offload.

For some algorithms choosing offload versus native execution option requires a trade-off consideration. Offload needs data transfer reducing performance but each device is working optimally. On the other hand the native option eliminates data transfer but some scalar or lowly parallel tasks have to be executed by Phi coprocessor that performs best for highly parallel tasks.

In general it is recommended to involve all cores with each core running at least two threads. This advice boils down to using at least 100 threads total. Published results of benchmark testing and real applications fully support this recommendation. This rule has been termed the scaling rule by Intel researchers [1]. They demonstrated that scaling is not sufficient for achieving optimal Phi performance of 1 TeraFlops for double precision. The second required key performance factor is vectorization.

## V. VECTORIZATION

Vectorization is a group of data of the same type than is processed by a single operation. Vectorization is also called the Single Instruction Multiple Data (SIMD). In Phi coprocessor vectorization is using the data parallel engines VPU that perform SIMD operations. The Intel Xeon Phi coprocessors vector engine VPU supports 512-bit vector width.

The vector instructions mean performing 16 single precision or 8 double precision arithmetic operations simultaneously. The performance boost that vectorization offers is a key to the Intel Xeon Phi coprocessor speed. Intel research documents suggest that if the application is not bandwidth limited, the best use of Intel Xeon Phi is when most operations are vector instructions. Very fortunately many algorithms for solving scientific and engineering problems use extensively linear algebra containing vector operations.

Several examples of linear algebra SIMD operations are listed below

1. Sum of two vectors `z[i]=x[i]+y[i]`

2. Vector triad `d[i]=a[i]+b[i]*c[i]`

3. Saxpy `y[i]=a*x[i]+y[i]`

4. Dot product is SIMD followed by reduction `a[1]*b[1] +a[2]*b[2]+...`

5. Euclidean norm $V_1 \times V_2 = \sqrt{\sum_i v^2}$

6. Matrix/vector product has two loops. The external loop can be parallelized. The internal loop is vectorizable.

7. Matrix/matrix product.

8. Matrix LU decomposition for solving general linear equations.

9. The Conjugate Gradient Method (CGM)

The Conjugate Gradient Method for solving linear equations with positive definite matrices is fully vectorizable. Every iteration of CGM contains one matrix/vector product, two vector dot products and three Saxpy operations.

All these operations are vectorizable using SIMD instructions.There are several requirements for vectorizing a loop:

1. The number of loop iterations is countable
2. The loop consists of a single block without jumps or branches
3. The Block has single entry and exit
4. There are no function calls inside the block except the vectorized common mathematical functions such as `sin`, `cos`, `sqrt` etc.

A block satisfying these requirements is regarded as a single instruction and it is executed in parallel as SIMD. Below is an example of SIMD block.

```
void example(double*a, double*b, double*c, double*d, int n){
int i;
  #pragmasimd
  for (i = 0; i < n; i++) {
    a[i] = c[i] * d[i];
b[i] = c[i] - d[i];
  }
}
```

Obstacles to vectorization include: noncontiguous data access in memory, indirect addressing, and data dependency.

## REFERENCES.

[1]J.Jeffers and J.Reinders"Intel Xeon Phi Coprocessor High Performance Programming" Elsevier and Morgan Kaufmann 2013.
[2]B.Chapman, G. Jost and R. van der Pas, "Using OpenMP", The MIT Press 2008.
[3]W.Gropp, E.Lusk and A.Skjellum, "Using MPI", The MIT Press 1994.
[4]A.D.Robinson "Parallel Programming with CilkPlus" Intel document 2012.
[5]J.Kowalik and T.Puzniakowski "Using OpenCL" IOS Press, 2012.