

# hyPACK-2013 Four-Days Technology Workshop on Hybrid Computing - Coprocessors & Accelerators -Power-Aware Computing & Performance of Application Kernels Jointly Organized by Centre for Development of Advanced Computing (C-DAC), Pune Centre for Modelling & Simulation (CMSD), HPC Facility, University of Hyderabad, Venue : CMSD, University of Hyderabad, Hyderabad Dates : October 15 (Tuesday) – October 18 (Friday)

# hyPACK-2013 : A Suite of programs on of Intel Xeon-Phi Coprocessors

- A set of programs and libraries, used for NLA problems and application kernels are developed on Intel Xeon-Phi Coprocessors. The aim of the programs is to understand tuning & performance issues on Intel Xeon-Phi Compiler Vectorization features, x86 SMP features, Intel MKL Lib. and obtain the best achievable performance for NLA kernels. The Xeon-Phi Compiler assisted Vectorization pragmas and Intel MKL libraries are used. Some codes compare the best performance of selective NLA codes, application kernels and application & system benchmarks and report the results using with and without various capabilities of Intel's compiler optimization features of Xeon-Phi Coprocessors using shared Memory Programing OpenMP /Pthreads/ Intel TBB and Explicit Message Passing Interface (MPI).
- The list of programs are developed with the help of Intel Xeon-Phi Coprocessors documents which are publicly available as Intel Books, Intel documents, Conferences, Workshops, Notes Material and especially recent webinars have been partially incorporated. List of references as given in annexure-I are used for this work.
  - 1.1. Auto-Parallelization Compiler features; Automatic offload & Compiler-Assisted Offload

#### **1.1.1. Back-Ground & Skill-Set Required:**

- Compiler automatically translates portions of serial code into equivalent multithreaded code with using these options: -parallel /Qparallel
- The auto-parallelizer analyzes the dataflow of loops and generates multithreaded code for those loops which can safely and efficiently be executed in parallel. The autoparallelizer report can provide information about program sections that could be parallelized by the compiler.
- **1.1.2.** Write your own program for NLA kernel codes using auto-parallelisation features on Xeon-Phi Coprocessors & analyze the compiler generated optimization reports for various problem sizes for typical matrix-matrix multiplication algorithms and obtain maximum achievable performance.

**1.2.** Using Intel Compiler (loop optimization pragmas/directives) Automatic offload & Compiler-Assisted Offload

## 1.2.1. Back-Ground & Skill Set Required:

Get exposure to various vectorization flags as given below.

- IVDEP : ignore vector dependency \* LOOP COUNT : advise typical iteration count(s) \* UNROLL : suggest loop unroll factor
- **DISTRIBUTE POINT**: advise where to split loop
- **PREFETCH** : hint to **prefetch** data
- VECTOR : vectorization hints : \* Aligned assume data is aligned; \* Always override cost model; \* Non-temporal advise use of streaming stores
- **NOVECTOR** : do not vectorize
- **PARALLEL** : override efficiency heuristics for auto-parallelization
- **1.2.2.** Write your own program for NLA kernel codes *with* or *without* use of Intel MKL libraries, and Application kernels i.e., Solution of Partial differential Eqs. (PDEs) using various loop optimization and compiler techniques.
- **1.3.** Using Intel Compiler (loop optimization pragmas/directives) GAP Guided Automatic Parallelization; Automatic offload & Compiler-Assisted Offload

## 1.3.1. Back-Ground & Skill Set Required:

- Use compiler to help detect what is blocking optimizations in particular vectorization, parallelization and data transformations gives advice on how to change code, add directives, add compiler options
- Extend diagnostic message for failed vectorization and parallelization by specific hints to fix problem
- Not possible to do Automatic vectorizer or parallelizer and it is restricted to changes applied to the program to be compiled
- **1.3.2.** Write your own software modules for NLA Kernels using compiler auto-parallelization features of Intel Xeon-Phi and analyze the GAP generated optimization reports. Summarize the performance and scalability issues for various problems size of your code.

# **1.4.** Vectorization Programming with SIMD update

# 1.4.1. Back-Ground & Skill Set Required:

- (Vectorization: Pragma/Directive SIMD) : Positioning of SIMD Vectorization (Use compiler based fully automatic vectorization) ; Use Auto-Vectorization hints (**#pragma** ivdep); User Mandated Vectorization (SIMD Directive); Use SIMD Intrinsic Class (F32Vec4 add); Vector intrinsic (mm\_add\_ps());
- **1.4.2.** Write your own software modules for NLA kernels using various clauses of SIMD Directives. Analyze the Vectorization reports and summarize performance issues for different problems size.

## **1.5.** Vectorization Ailgn The Data

## 1.5.1. Back-Ground & Skill Set Required :

- **Vectorization** : Align Data Role of compiler :
- declspec(align(n, [offset])) : Instructs the compiler to create the variable so that it is aligned on an "n"-byte boundary, with an "offset" (Default=0) in bytes from that boundary

- void\* \_mm\_malloc (int size, int n) Instructs the compiler to create a pointer to memory such that the pointer is aligned on an n-byte boundary and tell the compiler...
- #pragma vector aligned | unaligned
- **Vectorize** using aligned or unaligned loads and stores for vector accesses, overriding compiler's cost model
- **\_\_assume\_aligned(a,n)** Instructs the compiler to assume that array a is aligned on an n-byte boundary n=16 for SSE, n=32 for AVX
- **1.5.2.** Write your own suite of programs for NLA Kernels (Vector-Vector Addition, Matrix-Matrix Addition), Computational Mathematics kernels & PDE solvers using vector aligned data features of Intel Xeon-Phi using declspec(align(\*)). Analyze Vectorization reports & summarize the performance issues for different problems size of your code. (SIMD Directives - & IVDEP directives) DIRECTIVES /PRAGMAS to assist OR REQUIRE VECTORIZATION
- 1.6. Obtain the performance for Vector into Vector Multiplication and Matrix into Matrix Multiplication using Intel MKL Libraries on Intel XeonPhi Coprocessors & Automatic offload & Compiler-Assisted Offload
- 1.6.1. Back-Ground & Skill Set Required: Use Intel MKL libraries for NLA Kernels and write programs using libraries such as Level1, Level2, Level3 BLAS, SGEMM, DGEMM, Managing Multi-Core Performance: Use OpenMP & KMP\_AFFINITY to obtain best performance
- **1.6.2.** Write your own software modules for NLA kernels using Intel MKL with (**a**) compiler assisted offload and (**b**), Vectorization features
- 1.7. Obtain the performance using Intel MKL Libraries on Intel XeonPhi
- 1.7.1. Back-Ground & Skill Set Required:

Use Intel MKL libraries for NLA Kernels and write programs using libraries such as Level1, Level2, Level3 BLAS, SGEMM, DGEMM, DDOT, DETRF, and Direct Sparse Solver

- Managing Multi-Core Performance: Use OpenMP & KMP\_AFFINITY to obtain best performance (Usesystem function sched\_setaffinity to bind the threads to the cores on different sockets. Intel MKL accesses the memory functions by pointers i\_malloc, i\_free, etc., which are visible at the application level.
- Aligning Addresses on 128-byte Boundaries: Intel MKL or for reproducible results from run to run of Intel MKL functions require alignment of data arrays. To align an array on 128-byte boundaries, use mkl\_malloc() in place of system provided memory allocators,
- To Improve Performance Call Intel MKL memory allocation routines, such as mkl\_malloc; & Use the mmap system call with the MAP\_HUGETLB flag (Refer compiler assisted off-load- guide)

- **1.7.2.** Write your own software modules for NLA kernels using Intel MKL with (**a**) compiler assisted offload (**b**) Reusing data that already exists in the memory of the coprocessor helps to reduce transferring data for an example which illustrates how to perform multiple operations on a single set of input matrices.
- **1.8.** Obtain the performance of NLA codes using different types of array operations and analyze the performance on Intel Xeon-Phi.
- **1.8.1. Back-Ground & Skill Set Required:** Array Operations (SoA or AoS); Vectorization matching in Array Operations
- **1.8.2.** Write your own program for NLA kernels *with* and *without* array operations using vectorization features
- **1.9.** Obtain the performance of matrix into matrix multiplication code based on block partitioning of matrices on Intel Xeon-Phi.
- **1.9.1.** Back-Ground & Skill Set Required: Array Operations, Xeon-Phi Persistent Storage APIs, Dynamic Memory Declarations, Using Pointers, Asynchronous data transfers, and Double buffering.
- **1.9.2.** Write your own program for Matrix-Matrix Multiplication based on Block-partitioning of input matrices and use the Xeon-Phi Programming Environment features such as (a). Allocated Persistent Storage on Co-Processor (b). Asynchronous data transfer from the coprocessor to the processor (c). Double buffers inputs to an offload
- 1.10. Obtain the performance of NLA codes that use read /write files on Coprocessor
- **1.10.1. Back-Ground & Skill Set Required:** Array Operations (SoA or AoS); Vectorization matching in Array Operations Understand role of **MIC\_PROXY\_IO**
- **1.10.2.** Write your own program to perform large scale I/O operations and quantify the overheads.
- **1.11.** Write your own code for matrix-matrix multiplication using different access patterns of matrices using SoS / AoS and analyze the performance.
- 1.11.1. Back-Ground & Skill Set Required: Array Operations (SoA or AoS); Vectorization matching in Array Operations Understand role of MIC\_PROXY\_IO
- **1.11.2.** Write your own program to perform large scale I/O operations and quantify the overheads.
- **1.12.** Xeon Phi Coprocessor Arch: Measure copy Memory Bandwidth with varying number of thread count
- **1.12.1. Background & Skill Set Required :** Understand the processor core, Vector Processing Unit (VPU), Core Ring Interface (CRI), RING, SBOC, GBOX, PMU; Understand access time & memory bandwidth limitations
- **1.12.2.** Write your own program to measure *copy-memory bandwidth* using openMP or Pthreads, using 8/16/32 cores of Intel Xeon-Phi with different work-loads, and analyze the performance
- **1.12.3.** Obtain Performance of Stream OpenMP benchmark on Intel Xeon-Phi and compare the performance with the output of previous example using different programming paradigms.
- **1.13.** Xeon Phi Coprocessor Arch: Measure MPI Latency & Bandwidth in NATIVE mode using various MIC coprocessors on a node as well as in cluster.
- **1.13.1. Background & Skill Set Required :** MPI Benchmarks, Understand Software and hardware overheads- Interconnection networks of Cluster, tune MPI on cluster, MPI point-to-point & Collective Communications /Computations

- **1.13.2.** Write your own program to measure latency, bandwidth and quantify overheads using MPI point-to-point and Collective communications on Intel Xeon-Phi Coprocessors in a Message Passing Cluster with different message sizes & analyze the performance
- 1.14. Xeon-Phi Coprocessor Arch Measure Performance using OpenMP, Pthreads & Intel TBB for Vector-Vector & Matrix-Matrix Multiplication

#### 1.14.1. Background & Skill Set Required:

- Important features of Cache hierarchy are: The L1 cache has a 32 KB L1 instruction cache and 32 KB L1 data cache. Associativity is 8-way, with a cache line-size of 64 byte. It has a load-to-use latency of 1 cycle
- The L2 cache is a unified cache which is inclusive of the L1 data and instruction caches. Each core contributes 512 KB of L2 to the total global shared L2 cache storage. If no cores share any data or code, then the effective total L2 size of the chip is up to 31 MB
- 1.14.2. Write your own software modules for NLA or based on SGEMM/ GEMM) kernels using openMP or Pthreads, using 8/16/32 cores of Intel Xeon-Phi with different work-loads, and analyze the performance (Get System Info & Device Coprocessor using -ifconfig & micinfo) in terms of Gflops.
- 1.15. Xeon Phi-: Native Compilation/Compiler's offload pragmas

## 1.15.1. Background & Skill Set Required:

- Align data to 64 Bytes (512 Bits) for Xeon-phi and perform vectorization (due to the large SIMD width of 64 bytes & new set of instructions on Xeon-Phi may allow more loops to be parallelized on the Coprocessor, subject to requirements for vectorization loops is satisfied.
- Use Compiler Pragmas and use compiler options such as -vecreport2 to see if loops were vectorized for Xeon-Phi (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.
- 1.15.2. Write your own software modules for NLA (SGEMM/ DGEMM) kernels code using openMP allocated memory on the heap aligned to 64 byte boundary & analyze the performance issues & scalability issues (Use **#pragma vector** aligned "**#pragma** ivdep" & "posix memalign" for dynamic memory alignment)
- **1.16.** Xeon Phi-: Information about Compiler's offload (Automatic Off-load)

#### 1.16.1. Background & Skill-Set Required:

- Use the compiler option -vec-report2 to know about which loops have been vectorized on the host and the Xeon-Phi coprocessor:
- To obtain information about performance and data transfers at runtime, set-up the environment variable **OFFLOAD REPORT**
- CDAC@john:~ export OFFLOAD\_REPORT=2 CDAC@john:~ use ./run
- A subroutine with all computations to be performed on Xeon-Phi can be called within an offloaded block region. To do this, the function has to be declared with **attribute** ((target(mic))).

```
{
  MatMatMult (n,a,b,c);
}
```

}

- **1.16.2.** Write your own program to analyze the CPU time, Xeon-Phi time, CPU-to-Xeon-Phi Data transfer time and Xeon-Phi-CPU data transfer time and quantify the time taken for different problem sizes with respect to the number of OpenMP threads used and understand data transfers over the PCIe bus from the host to the accelerator and vice versa.
- **1.16.3.** Write your own code for matrix-matrix multiplication in which matrix- multiplication code of the previous example is called as a subroutine call and analyze the performance & scalability analysis
- 1.17. Xeon Phi: Explicit Sharing workload between the Coprocessor and host using OpenMP or Pthreads

## 1.17.1. Background & Skill-Set Required :

To distribute work between the host and Coprocessor for typical NLA kernel using OpenMP, Pthreads and MPI

```
#pragma omp parallel
     {
#pragma omp sections
         Ł
#pragma omp section
//section running on the coprocessor
              offload
#pragma
                            target(mic)
                                               in(Mata,Matb:length(n*n))
 inout(Matc:length(n*n))
   ł
        MatMatMulit(n,a,b,c);
  }
            }
#pragma omp section
       ł
        //section running on the host
          MatMatMut(n, MatP, MatQ, MatR);
       }
   }
}
```

- **1.17.2.** Write your own code to implement Matrix into Vector Multiplication and *n*-body simulation algorithm using OpenMP work-sharing constructs in which computations are performed on host and Coprocessor for different problem sizes.
- **1.18.** Performance: Persistent data on the Coprocessors Address data transfers over the slow PCIe bus from the host

#### 1.18.1. Background & Skill-Set Required:

- The data transfers over the PCIe bus from the host to the accelerator (Coprocessor) and vice versa is the main bottleneck for obtaining performance.. To increase the performance of code, it is necessary to keep the data on Coprocessor between computations using the same data in-order to minimize the data transfer and thereby increase the performance
- #define ALLOC alloc\_if(1)
- #define FREE free\_if(1)
- #define RETAIN free\_if(0)
- #define REUSE alloc\_if(0)

one can simply use the following notation:

- to allocate data and keep it for the next offload
- 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(1) REUSE FREE)

- **1.18.2.** Write your own code for matrix system of linear equations by Conjugate Gradient Method in which some vector values persistent on Coprocessor.
- **1.18.3.** Write your own code to implement solution of heat transfer PDE application in which assembly matrices of Finite Element Computations is performed on Intel Xeon-Phi
- **1.18.4.** Write your own program to perform and Matrix-Vector Multiplication having appropriate data of matrices persistent on Coprocessor.
- 1.19. Performance: OpenMP and Loop un-rolling with nested loops & Vectorization

## 1.19.1. Background & Skill Set Required:

- Loop Un-rolling & Vectorization using OpenMP implementation
- 1.19.2. Write your own codes for NLA kernels & PDE Solver using MPI-OpenMP (with Collapse and without Collapse) and Loop un-rolling (nested loops) with Vectorization (ivdep and vector aligned) (use OpenMP supported four different kinds of loop scheduling
- **1.20.** Performance: Using MKL (SGEMM/DGEMM) for offloading and BLAS routines Open MP implementation

## **1.20.1. Background & Skill Set Required:**

- Understand performance of SGEMM/DGEMM on number of Cores
- MKL memory Allocation Aps

- **1.20.2.** Write your own codes for NLA kernels \using Intel MKL SGEMM & DGEMM library with optimal use of Intel Xeon-Phi Cores and obtain the best performance of code using MIC Vectorization features.
- **1.21.** Simultaneous computation on host and accelerator OpenMP

## 1.21.1. Background & Skill-Set Required:

 To perform computations on host and the Coprocessor, function is generated for both Xeon-Phi and CPU; One thread executes the offload on Xeon-Phi. The other thread executes the same function on the host.

- **1.21.2.** Write your own program for implementation of PDE solver using Finite Difference Method (FDM) using OpenMP and MPI. The computations are performed on host and the Coprocessors
- **1.21.3.** Write your own program Monte Carlo Simulation using Intel Xeon-Phi Coprocessor features
- **1.22.** Overlap Computation and Communication Asynchronous Transfer Using Signals OpenMP & Pthreads

## 1.22.1. Background & Skill-Set Required:

- To overlap computations & communication, a function is generated on CPU which generalizes data decomposition on host processor. One thread executes the offload code on MIC, while the other thread transfer the data from host to MIC.
- Start an asynchronous transfer, tracking **signal in1** and Start once the completion of the transfer of in1 in **signaled** (wait)

```
#pragma offload_transfer target(mic:0) \
    in(in1:length(cnt)alloc_if(0) free_if(0)) signal(in1)
#pragma offload target(mic:0) nocopy(in1) wait(in1) \
    out(res1:length(cnt) alloc_if(0) free_if(0))
#pragma offload_transfer target(mic:0) \
    nocopy(in1:length(cnt) alloc_if(0) free_if(1))
```

**1.22.2.** Write your own program for implementation of NLA kernels & PDE Solver in twodimensional regions using MPI-OpenMP in which computations are performed on host and the Coprocessor.

# **1.23.** Overlap Computation and communication - Asynchronous Transfer & Double Buffering-OpenMP & Pthreads

#### **1.23.1.** Background & Skill Set Required:

- To overlap computations and communication, function is generated on CPU which generalizes data decomposition on host. While one thread executes the offload code on MIC while the other thread transfer the data from host to MIC (Send the buffer to in1 in (in1 : length(cnt) and send buffer in2 in (in2 : length(cnt) while in1 is in process. Send buffer in1 while in2 is in process.
- **1.23.2.** Write your own program for implementation of PDE solver using Finite Element Method (FEM) in two-dimensional regions using MPI OpenMP in which the computations are performed on host and the Coprocessor.

#### 1.24. Performance of Tuning OpenMP codes on Xeon-Phi Modifying Stack Size

#### 1.24.1. Background & Skill-Set Required:

- Export KMP\_STACKSIZE=4M; If the units are unspecified for KMP\_STACKSIZE, the number is assumed to be in bytes. KMP\_STACKSIZE overrides any OMP\_STACKSIZE setting
- If offloading computation from the host and MIC\_ENV\_PREFIX is not defined, the stacksize environment variables are copied from host to target environment when the target process is spawned (along with the rest of the environment). With MIC\_ENV\_PREFIX defined, users can define separate settings in the host environment for both host and target values:

#### export MIC\_ENV\_PREFIX=MIC export OMP\_STACKSIZE=8M export MIC OMP\_STACKSIZE=2

- The Intel Xeon Phi processors have more than 60 cores, each with four threads, or over 240 hardware threads. The default OMP stack size is **4MB**
- The host environment setting of MIC\_STACKSIZE should be done. This controls the size of the stack in the target process where in offloaded code is run and so only applies to offloaded code. The default size of this stack is 12 MB. Also, native applications run in a process whose default stack size is 8 MB
- 1.24.2. Write your own program for NLA Kernels and an implementation of PDE solver by FDM
  in 2D regions using MPI OpenMP in which the computations are performed using
  MIC\_KMP\_AFFINITY=verbose, granularity = fine, scatter, compact, and
  gather
- **1.25.** Number of threads running in parallel section on Xeon-Phi
- **1.25.1.** Write your own program to report the number of threads running in a parallel section and analyze the performance for different problems size of NLA kernels
- **1.26.** Get Number of Xeon-Phi Cards using for an application

#### **1.26.1.** Background & Skill-Set Required:

- The status of each card can be found out using **micinfo** - provides information about host & Coprocessor and system configuration

```
Codes run on Card # 1 or #2
#pragma offload target(mic [ :<expr> ] ) .
card # = <expr> % number_of_devices
/* Code must run on card #, aborts if not available
  (counts from 0)
  If -1, runtime chooses card, aborts if not available
```

If not present, runtime chooses card or runs on host if
none available
\*/
APIs:
 #include offload.h(C/C++); USE MIC\_LIB (Fortran)
 int\_Offload\_number\_of\_devices() /\* (C/C++) \*/
 result = OFFLOAD\_NUMBER\_OF\_DEVICES() /\* (Fortran) \*/
 // Returns # of Intel MIC devices installed, or 0 if none
 int\_Offload\_get\_device\_number() /\* (C/C++) \*/
 result = OFFLOAD\_GET\_DEVICE\_NUMBER() /\* (Fortran) \*/
 // Returns card number where executed, counting from 0,
 (or -1 for CPU)
 // Can use to share work explicitly by card number

- **1.26.2.** Write your own program to compute the matrix vector multiplication and sparse matrix into vector multiplication mapping part of computations onto each MIC of the host system.
- 1.27. The use of the larger 2MB pages -Enabling Huge Paging on MIC with libhugetlbfs library

## 1.27.1. Background & Skill Set Required :

- For typical matrix computation algorithms, if the array is sufficiently large, then each memory access can cause a TLB miss and corresponding performance drop. Native applications should use mmap or hugetlbfs to allocate memory with large pages.
- Setting PHI\_USE\_2MB\_BUFFERS (when MIC\_PREFIX=PHI) tells the runtime to allocate heap variables whose size is greater than the value specified by PHI USE 2MB BUFFERS in 2MB pages.
- **1.27.2.** Write your own program for implementation of PDE solver using Finite Finite Difference Method (FDM) using MPI & OpenMP, combination of MPI –OpenMP. The software module should use **larger 2MB** pages. The importance of larger pages for floating-point dominated FDM application is required as it performs array operation the computations on host and the Coprocessor.
- 1.28. The use of the larger 2MB pages -Enabling Huge Paging on MIC use mmap () library (hugetlbpage support)

#### **1.28.1.** Background & Skill Set Required :

Exposure to hugetlbpage support in the Linux kernel which is built on top of multiple page size support that is provided by most modern architectures. Users can use the huge page support in Linux kernel by either using the mmap system or huge page support should show the number of configured huge pages in the system by running the "cat /proc/meminfo" command.

- **1.28.2.** Write your own program for implementation of PDE solver using Finite Finite Difference Method (FDM) using MPI & OpenMP, combination of MPI OpenMP using **mmap**
- **1.29.** Measure Performance per Watt: Port for selective application and Macro /Micro benchmarks on Intel Xeon-Phi Coprocessor
- **1.29.1.** Measure performance per watt using external power-off meters and appropriate APIs

# Annexure-I: References

#### **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. <u>http://www.tacc.utexas.edu/</u>
- 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<sup>™</sup> (MIC) Parallelization & Vectorization, Intel Many Integrated Core Architecture, Software & Services Group, Developers Relations Division
- 13. Intel® Xeon Phi<sup>™</sup> (MIC) Programming, Rama Malladi, Senior Application Engineer, Intel Corporation, Bengaluru India April 2013
- 14. Intel® Xeon Phi<sup>™</sup> (MIC) Performance Tuning, Rama Malladi, Senior Application Engineer, Intel Corporation, Bengaluru India April 2013
- 15. Intel® Xeon Phi<sup>™</sup> 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<sup>™</sup> 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<sup>™</sup> 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<sup>™</sup> 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
- 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, <u>http://software.intel.com/en-us/mic-developer</u>
- 28. Intel Many Integrated Core Architecture User Forum, http://software.intel.com/en-us/forums/intel-many-integrated-core
- 29. Intel Developer Zone: Intel Math Kernel Library, <u>http://software.intel.com/en-us</u>
- Intel Xeon Processors & Intel Xeon Phi Coprocessors Introduction to High Performance Applications Development for Multicore and Manycore – Live Webinar, 26.-27, February .2013, recorded http://software.intel.com/en-us/articles/intel-xeon-phi-training-m-core
- 31. Intel Cilk Plus Home Page, http://cilkplus.org/
- 32. James Reinders, Intel Threading Building Blocks (Intel TBB), O'REILLY, 2007
- 33. Intel Xeon Phi Coprocessor Developer's Quick Start Guide, <u>http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-developers-quick-start-guide</u>
- 34. Using the Intel MPI Library on Intel Xeon Phi Coprocessor Systems, http://software.intel.com/en-us/articles/using-the-intel-mpi-library-on-intel-xeon-phi-coprocessor-systems
- 35. An Overview of Programming for Intel Xeon processors and Intel Xeon Phi coprocessors, <u>http://software.intel.com/sites/default/files/article/330164/an-overview-of-programming-for-intel-xeon-processors\_and-intel-xeon-phi-coprocessors\_1.pdf</u>
- 36. Programming and Compiling for Intel Many Integrated Core Architecture, <u>http://software.intel.com/en-us/articles/programming-and-compiling-for-intel-many-integrated-core-architecture</u>
- 37. Building a Native Application for Intel Xeon Phi Coprocessors, http://software.intel.com/en-us/articles/
- 38. Advanced Optimizations for Intel MIC Architecture, <u>http://software.intel.com/en-us/articles/advanced-optimizations-for-intel-mic-architecture</u>
- 39. Optimization and Performance Tuning for Intel Xeon Phi Coprocessors Part 1: Optimization Essentials, <u>http://software.intel.com/en-us/articles/optimization-and-performance-tuning-for-intel-xeonphi-</u> <u>coprocessors-part-1-optimization</u>
- 40. Optimization and Performance Tuning for Intel Xeon Phi Coprocessors, Part 2: Understanding and Using Hardware Events, <u>http://software.intel.com/en-us/articles/optimization-and-performance-tuning-for-intel-xeon-phi-coprocessors-part-2-understanding</u>
- 41. Requirements for Vectorizable Loops, http://software.intel.com/en-us/articles/requirements-for-vectorizable-
- 42. 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
- 43. <u>http://software.intel.com/mic-developer</u>
- 44. Loc Q Nguyen, Intel Corporation's Software and Services Group, Using the Intel® MPI Library on Intel® Xeon Phi<sup>™</sup> Coprocessor System,
- 45. Frances Roth, System Administration for the Intel® Xeon Phi<sup>™</sup> Coprocessor, Intel white Paper
- 46. Intel® Xeon Phi<sup>™</sup> Coprocessor, James Reinders, Supercomputing 2012 Presentation
- 47. Intel<sup>®</sup> Xeon Phi<sup>™</sup> Coprocessor Offload Compilation, Intel software