C-DAC Four Days Technology Workshop

ON

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

hyPACK-2013
Mode 3 : Intel Xeon Phi Coprocessors

Lecture Topic :
Intel Xeon-Phi Coprocessor : Cilk Plus

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

An Overview of Xeon Phi – Prog. Shared Address Space Prog,. – Cilk Plus

Lecture Outline

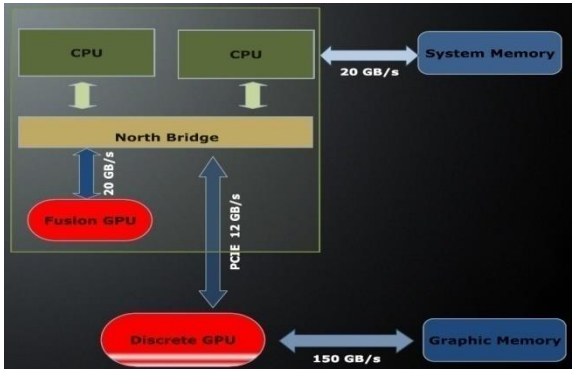
Following topics will be discussed

- Understanding of Xeon –Phi Architectures
- Programming on Xeon-Phi – Prog. -Cilk Plus
- Tuning & Performance – Software Threading

Intel Xeon-Phi
Shared Address Space Programming
Cilk Plus

Systems with Accelerators

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



CPU — ACC

Single thread perf. **throughput**

CPU ACC.

Physical integration

CPU & ACC

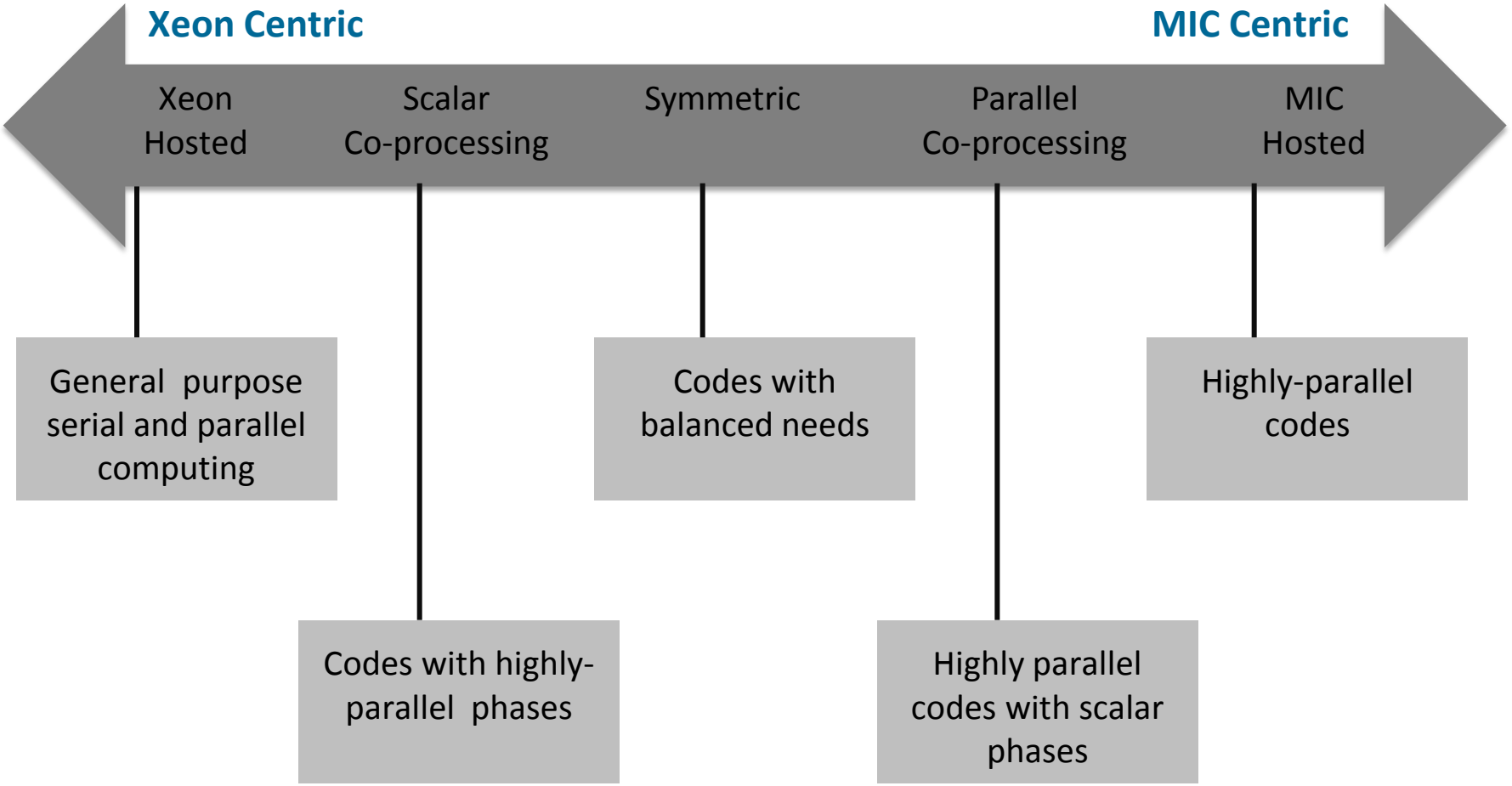
Architectural integration

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

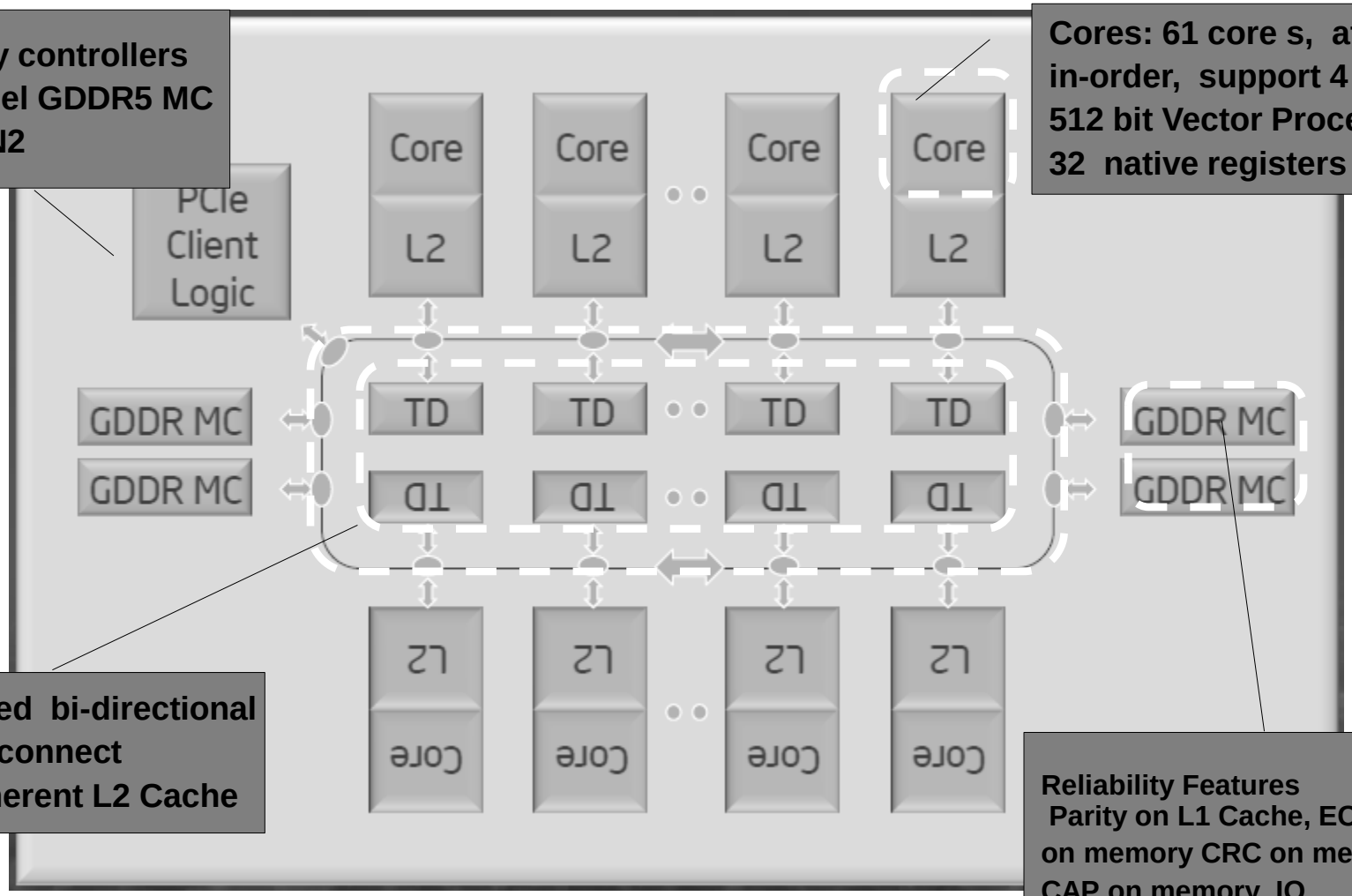
MIC Architecture, System Overview

Compute modes vision



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

Intel® Xeon Phi™ Architecture Overview



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

Intel Xeon-Phi
Shared Address Space Programming
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

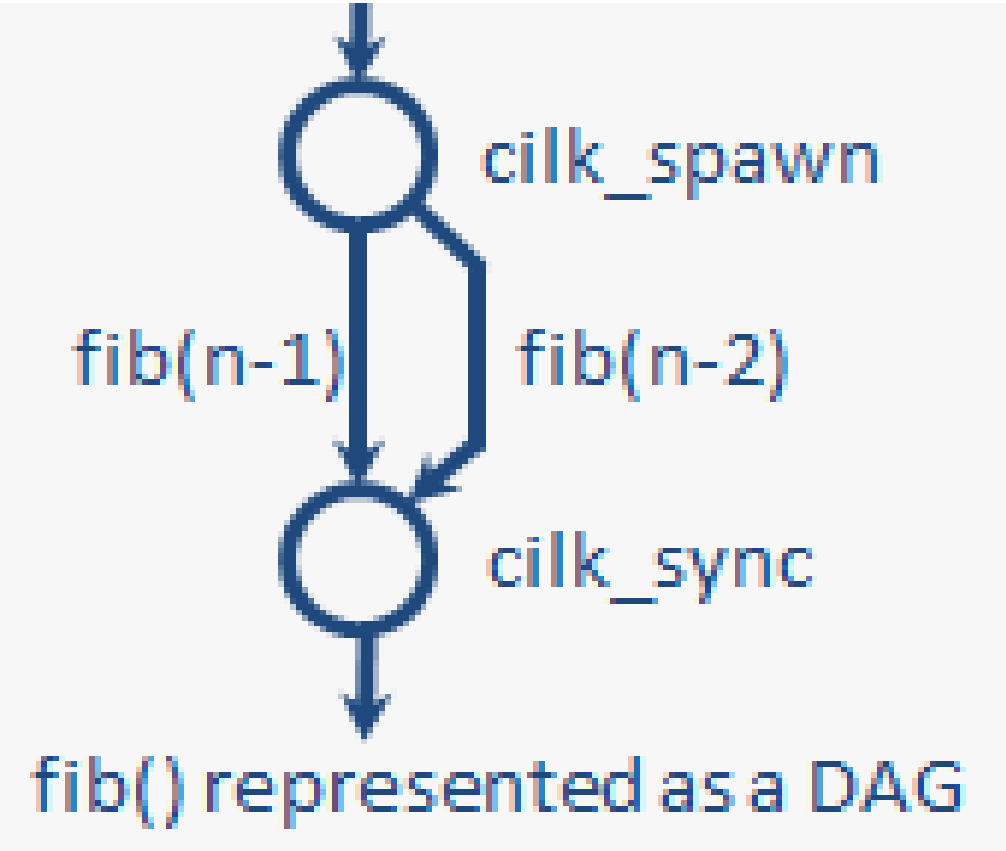
```
class_Cilk_Shared common {
    int data1;
    int *data2;
    class common *next;
    void process();
}
_Cilk_Shared class common obj1, obj2;
_Cilk_spawn _offload obj1.process();
_Cilk_spawn  _offload  obj2.process();
```

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)

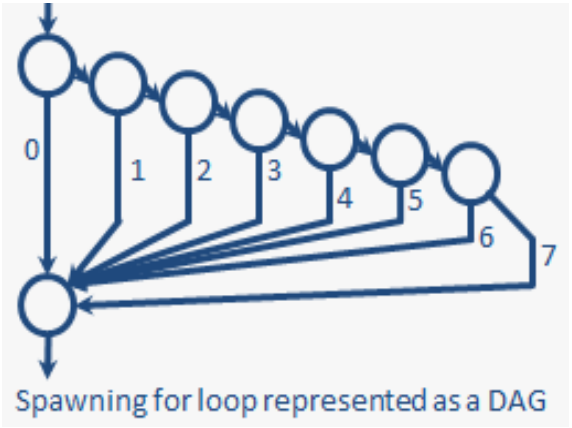
```
int fib(int n)
{
  if (n < 2)
      return n;
      int x = cilk_spwan fib(n-1);
      int y = fib(n-2);
      cilk_sync;
      return x + y;
 }
```

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

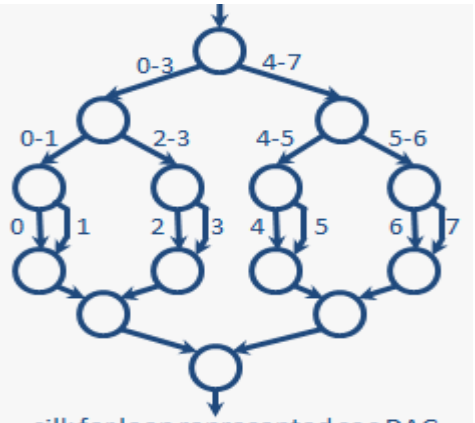
Intel Xeon-Phi : Prog. Env. Cilk Plus

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

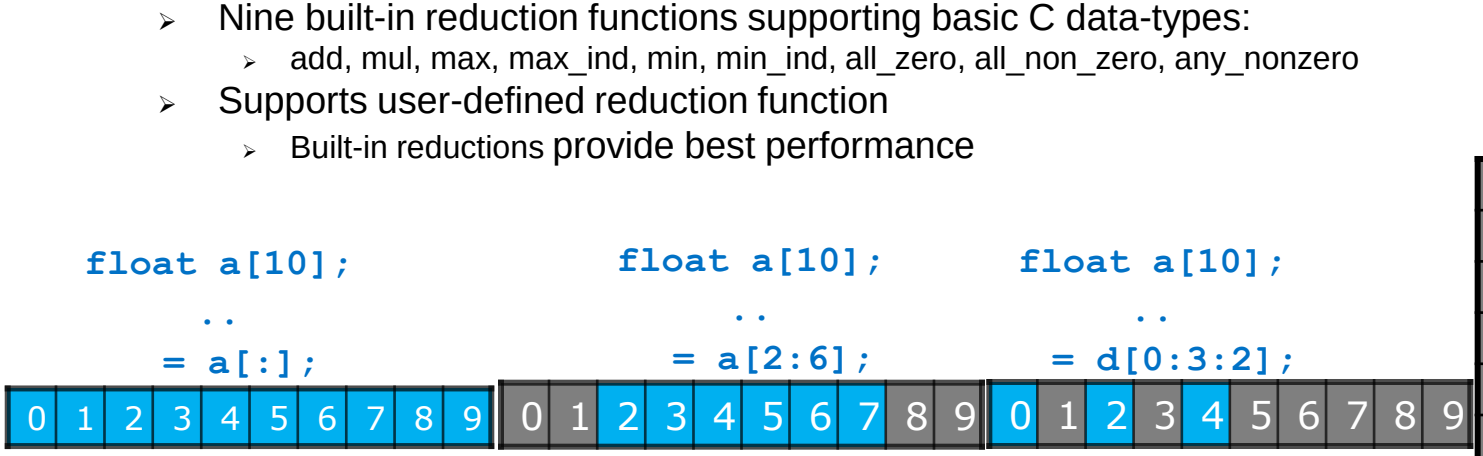
Source : References & Intel Xeon-Phi; http://www.intel.com/

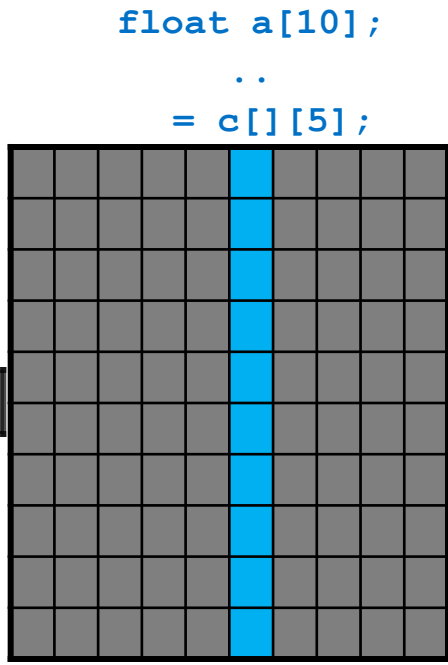
Intel® Cilk™ Plus Array Notation

- 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/

References & Acknowledgements

References :

1. Theron Voran, Jose Garcia, Henry Tufo, University Of Colorado at Boulder National Center or Atmospheric Research, TACC-Intel Hihgly Parallel Computing Symposium, Austin TX, April 2012
2. Robert Harkness, Experiences with ENZO on the Intel R Many Integrated Core (Intel MIC) Architecture, National Institute for Computational Sciences, Oak Ridge National Laboratory
3. Ryan C Hulguin, National Institute for Computational Sciences, Early Experiences Developing CFD Solvers for the Intel Many Integrated Core (Intel MIC) Architecture, TACC-Intel Highly Parallel Computing Symposium April, 2012
4. Scott McMillan, Intel Programming Models for Intel Xeon Processors and Intel Many Integrated Core (Intel MIC) Architecture, TACC-Highly Parallel Comp. Symposium April 2012
5. Sreeram Potluri, Karen Tomko, Devendar Bureddy , Dhabaleswar K. Panda, Intra-MIC MPI Communication using MVAPICH2: Early Experience, Network-Based Computing Laboratory, Department of Computer Science and Engineering The Ohio State University, Ohio Supercomputer Center, TACC-Highly Parallel Computing Symposium April 2012
6. Karl W. Schulz, Rhys Ulerich, Nicholas Malaya ,Paul T. Bauman, Roy Stogner, Chris Simmons, Early Experiences Porting Scientific Applications to the Many Integrated Core (MIC) Platform ,Texas Advanced Computing Center (TACC) and Predictive Engineering and Computational Sciences (PECOS) Institute for Computational Engineering and Sciences (ICES), The University of Texas at Austin ,Highly Parallel Computing Symposium ,Austin, Texas, April 2012
7. Kevin Stock, Louis-Noel, Pouchet, P. Sadayappan ,Automatic Transformations for Effective Parallel Execution on Intel Many Integrated, The Ohio State University, April 2012
8. http://www.tacc.utexas.edu/
9. Intel MIC Workshop at C-DAC, Pune April 2013
10. First Intel Xeon Phi Coprocessor Technology Conference iXPTC 2013 New York, March 2013
11. Shuo Li, Vectorization, Financial Services Engineering, software and Services Group, Intel ctel Corporation;
12. Intel® Xeon Phi™ (MIC) Parallelization & Vectorization, Intel Many Integrated Core Architecture, Software & Services Group, Developers Relations Division

References & Acknowledgements

References :

13. Intel® Xeon Phi™ (MIC) Programming, Rama Malladi, Senior Application Engineer, Intel Corporation, Bengaluru India April 2013
14. Intel® Xeon Phi™ (MIC) Performance Tuning, Rama Malladi, Senior Application Engineer, Intel Corporation, Bengaluru India April 2013
15. Intel® Xeon Phi™ Coprocessor Architecture Overview, Dhiraj Kalamkar, Parallel Computing Lab, Intel Labs, Bangalore
16. Changkyu Kim,Nadathur Satish ,Jatin Chhugani ,Hideki Saito,Rakesh Krishnaiyer ,Mikhail Smelyanskiy ,Milind Girkar, Pradeep Dubey, Closing the Ninja Performance Gap through Traditional Programming and Compiler Technology , Technical Report Intel Labs , Parallel Computing Laboratory , Intel Compiler Lab, 2010
17. Colfax International Announces Developer Training for Intel® Xeon Phi™ Coprocessor, Industry First Training Program Developed in Consultation with Intel SUNNYVALE, CA, Nov, 2012
18. Andrey Vladimirov Stanford University and Vadim Karpusenko , Test-driving Intel® Xeon Phi™ coprocessors with a basic N-body simulation Colfax International January 7, 2013 Colfax International, 2013 http://research.colfaxinternational.com/
19. Jim Jeffers and James Reinders,Intel® Xeon Phi™ Coprocessor High-Performance Programming by Morgann Kauffman Publishers Inc, Elsevier, USA. 2013
20. Michael McCool, Arch Robison, James Reinders, Structured Parallel Programming: Patterns for Efficient Computation, Morgan Kaufman Publishers Inc, 2013.
21. Dan Stanzione, Lars Koesterke, Bill Barth, Kent Milfeld by Preparing for Stampede: Programming Heterogeneous Many-Core Supercomputers. TACC, XSEDE 12 July 2012
22. John Michalakes, Computational Sciences Center, NREL, & Andrew Porter, Opportunities for WRF Model Acceleration, WRF Users workshop, June 2012
23. Jim Rosinski , Experiences Porting NOAA Weather Model FIM to Intel MIC, ECMWF workshop On High Performance Computing in Meteorology, October 2012
24. Michaela Barth, KTH Sweden , Mikko Byckling, CSC Finland, Nevena Ilieva, NCSA Bulgaria, Sami Saarinen, CSC Finland, Michael Schliephake, KTH Sweden, Best Practice Guide Intel Xeon Phi v0.1, Volker Weinberg (Editor), LRZ Germany March 31 ,2013

References & Acknowledgements

References :

25. Barbara Chapman, Gabriele Jost and Ruud van der Pas, Using OpenMP, MIT Press Cambridge, 2008
26. Peter S Pacheco, An Introduction Parallel Programming, Morgann Kauffman Publishers Inc, Elsevier, USA. 2011
27. Intel Developer Zone: Intel Xeon Phi Coprocessor,
28. http://software.intel.com/en-us/mic-developer
29. Intel Many Integrated Core Architecture User Forum,
30. http://software.intel.com/en-us/forums/intel-many-integrated-core
31. Intel Developer Zone: Intel Math Kernel Library, http://software.intel.com/en-us
32. Intel Xeon Processors & Intel Xeon Phi Coprocessors – Introduction to High Performance Applications Development for Multicore and Manycore – Live Webinar, 26.-27, February .2013,
33. recorded http://software.intel.com/en-us/articles/intel-xeon-phi-training-m-core
34. Intel Cilk Plus Home Page, http://cilkplus.org/
35. James Reinders, Intel Threading Building Blocks (Intel TBB), O'REILLY, 2007
36. Intel Xeon Phi Coprocessor Developer's Quick Start Guide,
37. http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-developers-quick-start-guide
38. Using the Intel MPI Library on Intel Xeon Phi Coprocessor Systems,
39. http://software.intel.com/en-us/articles/using-the-intel-mpi-library-on-intel-xeon-phi-coprocessor-systems
40. An Overview of Programming for Intel Xeon processors and Intel Xeon Phi coprocessors,
41. http://software.intel.com/sites/default/files/article/330164/an-overview-of-programming-for-intel-xeon-processors-and-intel-xeon-phi-coprocessors_1.pdf
42. Programming and Compiling for Intel Many Integrated Core Architecture,
43. http://software.intel.com/en-us/articles/programming-and-compiling-for-intel-many-integrated-core-architecture
44. Building a Native Application for Intel Xeon Phi Coprocessors,
45. http://software.intel.com/en-us/articles/

References & Acknowledgements

References :

46. Advanced Optimizations for Intel MIC Architecture, http://software.intel.com/en-us/articles/advanced-optimizations-for-intel-mic-architecture
47. Optimization and Performance Tuning for Intel Xeon Phi Coprocessors - Part 1: Optimization Essentials, http://software.intel.com/en-us/articles/optimization-and-performance-tuning-for-intel-xeonphi-coprocessors-part-1-optimization
48. Optimization and Performance Tuning for Intel Xeon Phi Coprocessors, Part 2: Understanding and Using Hardware Events, http://software.intel.com/en-us/articles/optimization-and-performance-tuning-for-intel-xeon-phi-coprocessors-part-2-understanding
49. Requirements for Vectorizable Loops,
50. http://software.intel.com/en-us/articles/requirements-for-vectorizable-
51. R. Glenn Brook, Bilel Hadri, Vincent C. Betro, Ryan C. Hulguin, Ryan Braby. Early Application Experiences with the Intel MIC Architecture in a Cray CX1, National Institute for Computational Sciences. University of Tennessee. Oak Ridge National Laboratory. Oak Ridge, TN USA
52. http://software.intel.com/mic-developer
53. Loc Q Nguyen , Intel Corporation's Software and Services Group , Using the Intel® MPI Library on Intel® Xeon Phi™ Coprocessor System,
54. Frances Roth, System Administration for the Intel® Xeon Phi™ Coprocessor, Intel white Paper
55. Intel® Xeon Phi™ Coprocessor, James Reinders, Supercomputing 2012 Presentation
56. Intel® Xeon Phi™ Coprocessor Offload Compilation, Intel software

References

1. Andrews, Grogory R. (2000), Foundations of Multithreaded, Parallel, and Distributed Programming, Boston, MA : Addison-Wesley
2. Butenhof, David R (1997), Programming with POSIX Threads , Boston, MA : Addison Wesley Professional
3. Culler, David E., Jaswinder Pal Singh (1999), Parallel Computer Architecture - A Hardware/Software Approach , San Francsico, CA : Morgan Kaufmann
4. Grama Ananth, Anshul Gupts, George Karypis and Vipin Kumar (2003), Introduction to Parallel computing, Boston, MA : Addison-Wesley
5. Intel Corporation, (2003), Intel Hyper-Threading Technology, Technical User's Guide, Santa Clara CA : Intel Corporation Available at : http://www.intel.com
6. Shameem Akhter, Jason Roberts (April 2006), Multi-Core Programming - Increasing Performance through Software Multi-threading , Intel PRESS, Intel Corporation,
7. Bradford Nichols, Dick Buttlar and Jacqueline Proulx Farrell (1996), Pthread Programming O'Reilly and Associates, Newton, MA 02164,
8. James Reinders, Intel Threading Building Blocks – (2007) , O'REILLY series
9. Laurence T Yang & Minyi Guo (Editors), (2006) High Performance Computing - Paradigm and Infrastructure Wiley Series on Parallel and Distributed computing, Albert Y. Zomaya, Series Editor
10. Intel Threading Methodology ; Principles and Practices Version 2.0 copy right (March 2003), Intel Corporation
11. William Gropp, Ewing Lusk, Rajeev Thakur **(1999),** Using MPI-2, Advanced Features of the Message-Passing Interface, The MIT Press..
12. Pacheco S. Peter, **(1992),** Parallel Programming with MPI, , University of Sanfrancisco, Morgan Kaufman Publishers, Inc., Sanfrancisco, California
13. Kai Hwang, Zhiwei Xu, (**1998**), Scalable Parallel Computing (Technology Architecture Programming), McGraw Hill New York.
14. Michael J. Quinn (**2004**), Parallel Programming in C with MPI and OpenMP McGraw-Hill International Editions, Computer Science Series, McGraw-Hill, Inc. Newyork
15. Andrews, Grogory R. **(2000),** Foundations of Multithreaded, Parallel, and Distributed Progrmaming, Boston, MA : Addison-Wesley

References

16. SunSoft. Solaris multithreaded programming guide. SunSoft Press, Mountainview, CA, **(1996),** Zomaya, editor. Parallel and Distributed Computing Handbook. McGraw-Hill,
17. Chandra, Rohit, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, and Ramesh Menon, **(2001)**,Parallel Programming in OpenMP San Fracncisco Moraan Kaufmann
18. S.Kieriman, D.Shah, and B.Smaalders **(1995),** Programming with Threads, SunSoft Press, Mountainview, CA. 1995
19. Mattson Tim, **(2002),** Nuts and Bolts of multi-threaded Programming Santa Clara, CA : Intel Corporation, Available at : http://www.intel.com
20. I. Foster **(1995,** Designing and Building Parallel Programs ; Concepts and tools for Parallel Software Engineering, Addison-Wesley (1995
21. J.Dongarra, I.S. Duff, D. Sorensen, and H.V.Vorst **(1999),** Numerical Linear Algebra for High Performance Computers (Software, Environments, Tools) SIAM, 1999
22. OpenMP C and C++ Application Program Interface, Version 1.0". **(1998),** OpenMP Architecture Review Board. October 1998
23. D. A. Lewine. *Posix Programmer's Guide:* **(1991),** Writing Portable Unix Programs with the Posix. 1 Standard. O'Reilly & Associates, 1991
24. Emery D. Berger, Kathryn S McKinley, Robert D Blumofe, Paul R.Wilson, *Hoard : A Scalable Memory Allocator for Multi-threaded Applications* ; The Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX). Cambridge, MA, November (**2000**). Web site URL : http://www.hoard.org/
25. Marc Snir, Steve Otto, Steyen Huss-Lederman, David Walker and Jack Dongarra, (**1998**) *MPI-The Complete Reference: Volume 1, The MPI Core, second edition* [MCMPI-07].
26. William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir (**1998**) *MPI-The Complete Reference: Volume 2, The MPI-2 Extensions*
27. A. Zomaya, editor. Parallel and Distributed Computing Handbook. McGraw-Hill, **(1996)**
28. OpenMP C and C++ Application Program Interface, Version 2.5 (**May 2005**)", From the OpenMP web site, URL **: http://www.openmp.org/**
29. Stokes, Jon 2002 Introduction to Multithreading, Super-threading and Hyper threading *Ars Technica*, October **(2002)**

References

30. Andrews Gregory R. 2000, Foundations of Multi-threaded, Parallel and Distributed Programming, Boston MA : Addison – Wesley (**2000)**
31. Deborah T. Marr , Frank Binns, David L. Hill, Glenn Hinton, David A Koufaty, J . Alan Miller, Michael Upton, "Hyperthreading, Technology Architecture and Microarchitecture", Intel (**2000-01)**
32. http://www.erc.msstate.edu/mpi
33. http://www.arc.unm.edu/workshop/mpi/mpi.html
34. http://www.mcs.anl.gov/mpi/mpich
35. The MPI home page, with links to specifications for MPI-1 and MPI-2 standards : http://www.mpi–forum.org
36. Hybrid Programming Working Group Proposals, Argonne National Laboratory, Chiacago (2007-2008)
37. TRAC Link : https://svn.mpi-forum.org/trac/mpi-form-web/wiki/MPI3Hybrid
38. Threads and MPI Software, Intel Software Products and Services 2008 - 2009
39. Sun MPI 3.0 Guide November 2007
40. Treating threads as MPI processes thru Registration/deregistration –Intel Software Products and Services 2008 – 2009
41. Intel MPI library 3.2 - http://www.hearne.com.au/products/Intelcluster/edition/mpi/663/
42. http://www.cdac.in/opecg2009/
43. PGI Compilers http://www.pgi.com

An Overview of Multi-Core Processors

Conclusions

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

Thank You

Any questions ?