C-DAC Four Days Technology Workshop

ON

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

hyPACK-2013
(Mode-1:Multi-Core)

Lecture Topic:

Multi-Core Processors :

Tuning & Performance /Compilers Part-I

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

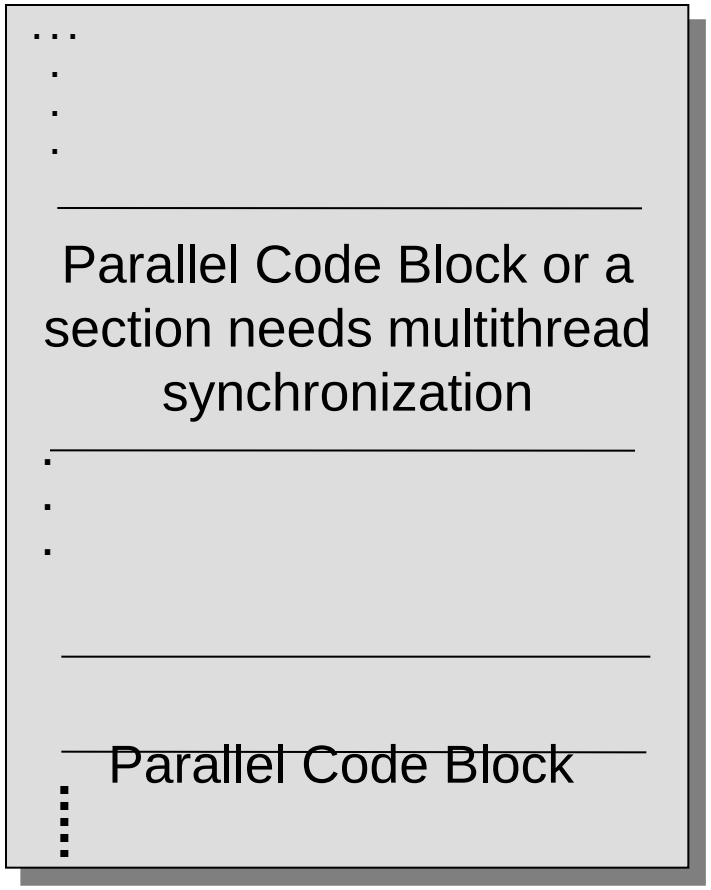
Lecture Outline

Following Topics will be discussed

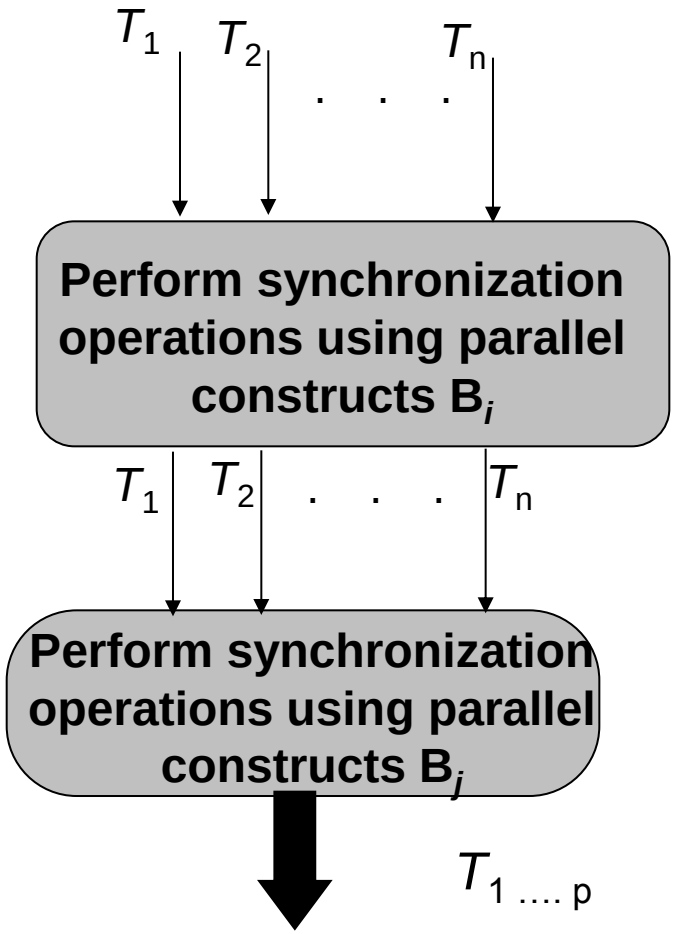
- An overview Tuning & Performance on Multi Cores
- Understanding Single Core /Multi Core – Compiler Switches
- Automatic Parallelization & Compiler Optimization
- Performance issues - Examples

Operational Flow of Threads for an Application

Implementation Source Code



Operational Flow of Threads



Source : Reference : [6]

Programming Multicore Processors

- Explicit Parallel Programming
  - Thread-based Programming Models.
  - Data Parallel Programming Models
  - Stream Programming Models
- Automatic Parallelization
  - Features of Most compliers for SMP systems, but currently see very little practical use
  - Polyhedral framework for dependencies and loop transformations – enabling composition of complex transformations over multiple statements.

Source : Reference : [4], [6]

Using Your Compiler Effectively : Basic Compiler Tech.

Questions to be addressed :

- How compiler optimizations can help user to get good performance?
- What you can do in your sequential program to get uniprocessor performance?
- What can you do in your parallel program to get good performance on given parallel machine?

Remarks:

- It is important to know whether the compiler is compiling the code optimally so that you can adjust the code, compiler options or something else.

Improving Single Core Performance

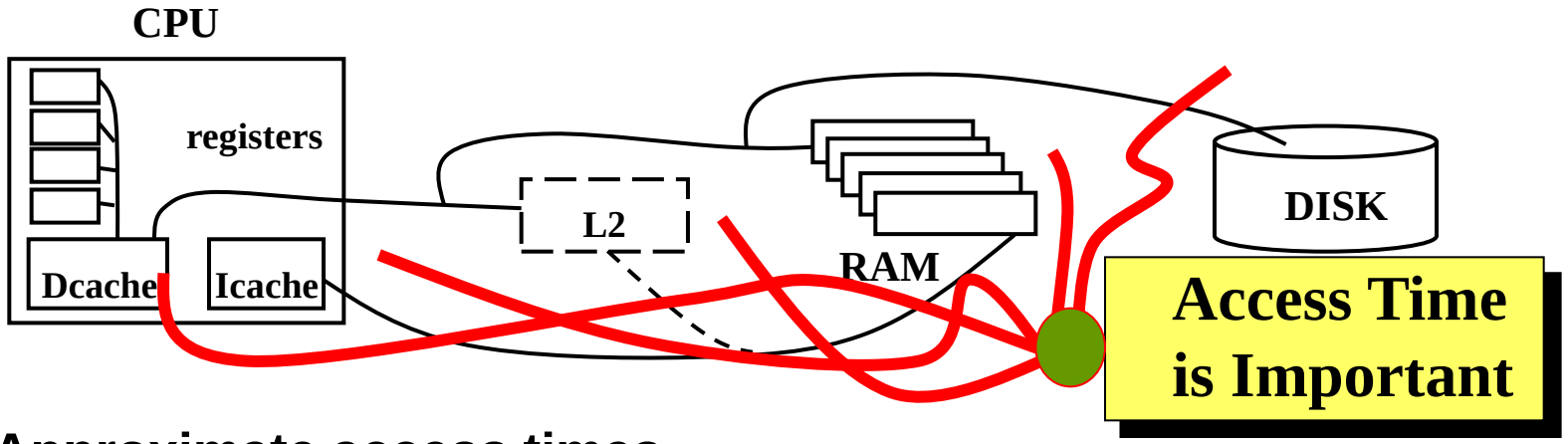
- How much sustained performance one can achieve for given program on a machine ?
- It is programmer's job to take advantage as much as possible of the CPU's hardware /software characteristics to boost the performance of the program !
- Quite often, just a few simple changes to one's code improves performance by a factor of 2, 3 or better !
- Also, simply compiling with some of the optimization flags (-O3, - fast, ….) can improve the performance dramatically !

Single Core Performance: Compiler Optimization

- Performance has to do with the following :
  - Problem size and precision (Role of Compiler)
  - Execution time - Computational Issues (Role of Compilers)
  - Ease of Programming (Role of Compilers)
- Questions to be addressed
  - How big a problem can I solve ?
  - How precise is the solution of my problem ?
  - How long will it take for the computer to run the program to completion ?

The Memory sub-system : Access time

- A lot of time is spent accessing/storing data from/to memory. It is important to keep in mind the relative times for each memory types:

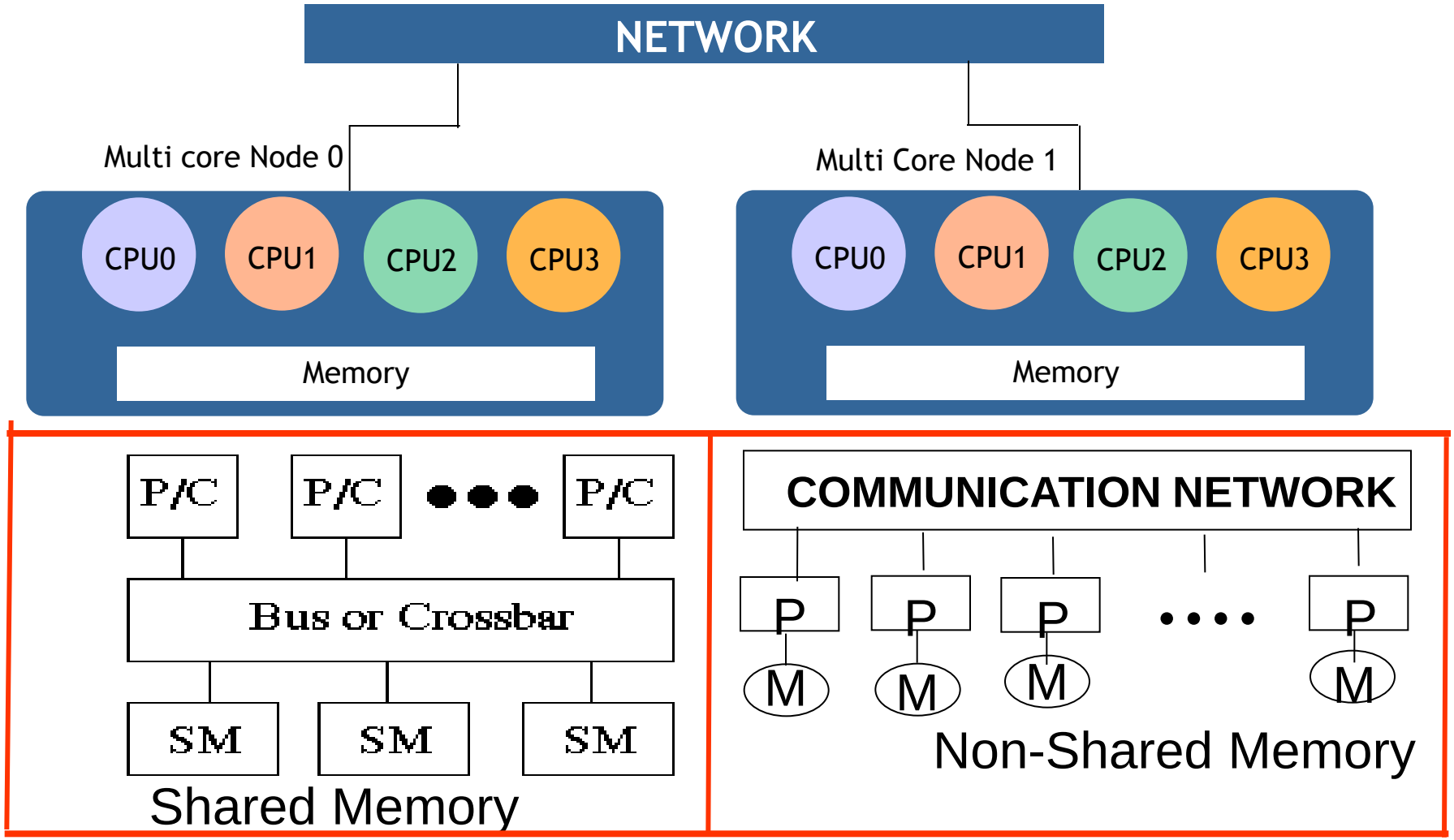


- **Approximate access times**
  - CPU-registers: 0 cycles (that's where the work is done!)
  - L_1 Cache: 1 cycle (Data and Instruction cache). Repeated access to a cache takes only 1 cycle
  - L_2 Cache (static RAM): 3-5 cycles?
  - Memory (DRAM): 10 cycles (Cache miss);
  - 30-60 cycles for Translation Lookaside Buffer (TLB) update
  - Disk: about 100,000 cycles!
  - connecting to other nodes - depending on network latency

Memory Management

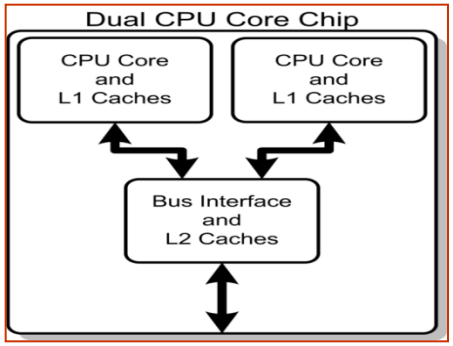
- Memory Reference Optimization and Managing Memory Overheads play an key role for performance
  - Hierarchical Memory features of Memory Sub-System
  - Getting memory references right is one of the most important challenges of application performance
  - Memory access patterns for performance
  - Cache Performance and Cache Miss
  - Cache Memories for Reducing Memory Overheads
  - Role of Data Reuse on Memory system performance
  - Techniques for Hiding Memory Latency (Multi-threading)

General-Purpose Clusters /Multi Cores

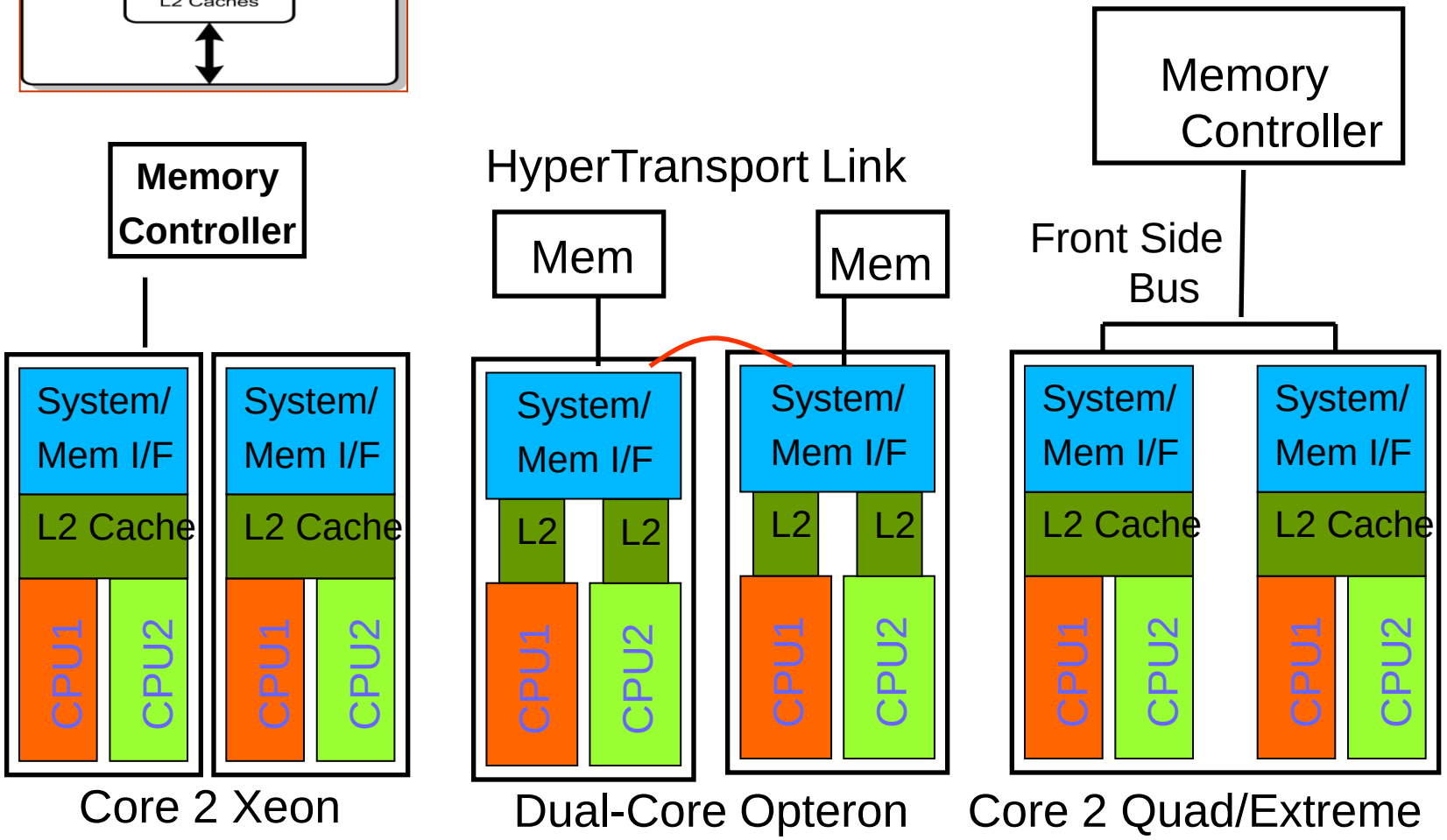


Source : http://www.intel.com; http://www.amd.com; Reference [4], [6]

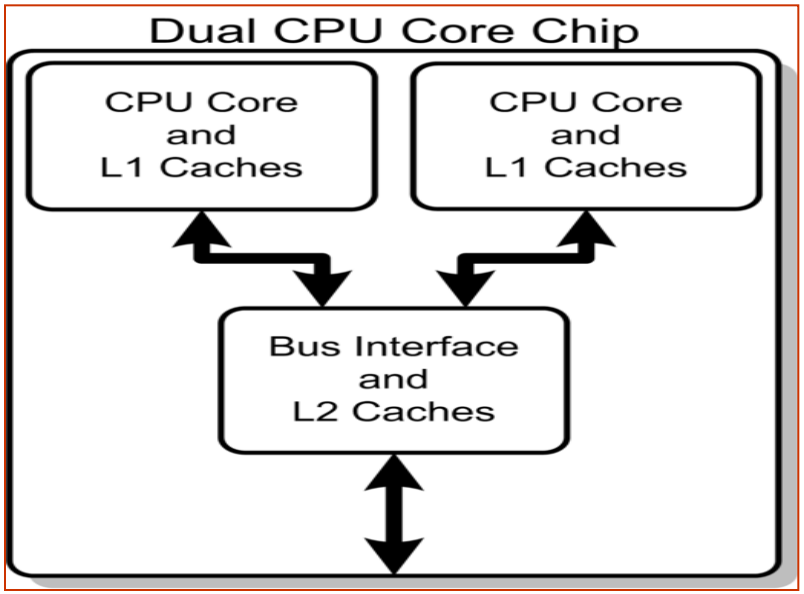
Multi Cores Processors



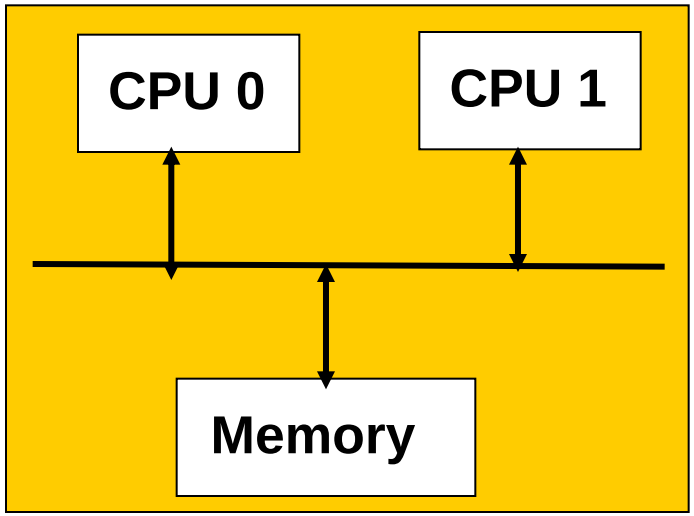
Source : http://www.intel.com; http://www.amd.com



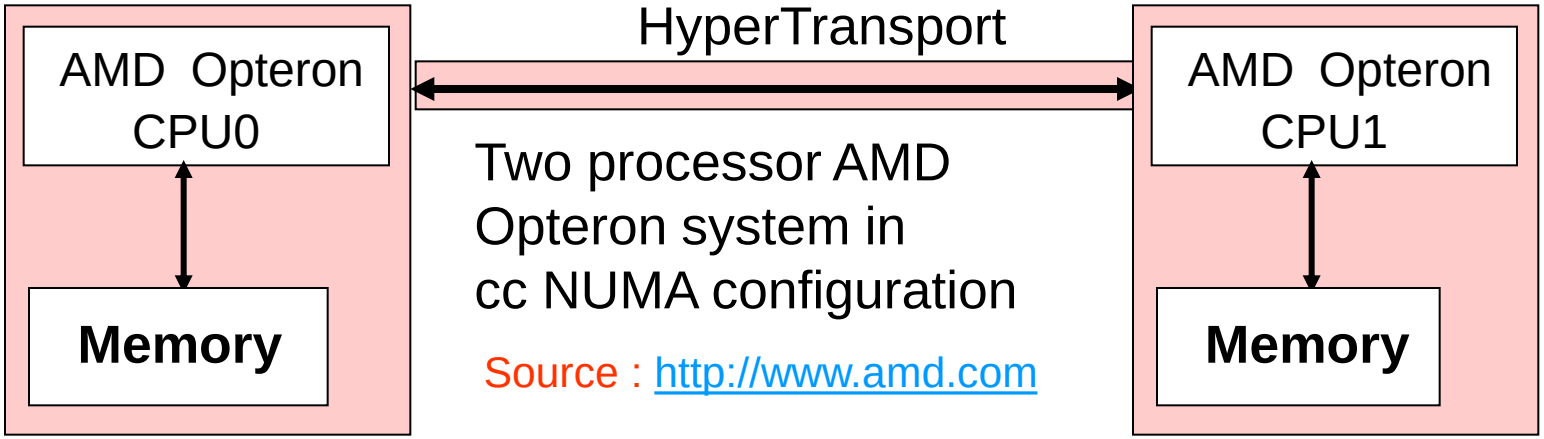
Multi Cores Processors



Two processor Dual Core



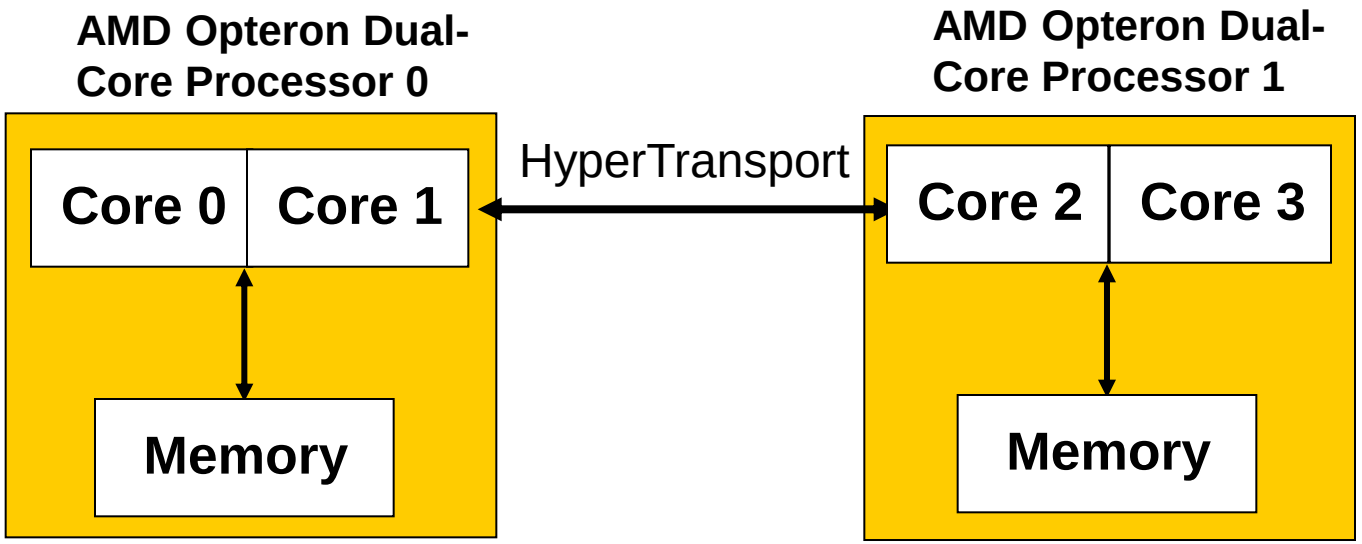
Simple SMP Block Diagram for a two processors



Two processor AMD Opteron system in cc NUMA configuration

Source : http://www.amd.com

Multi Cores Processors



Dual-Core AMD Opteron Processor configuration

- AMD : Cache-Coherent nonuniform memory access (ccNUMA)
  - Two or more processors are connected together on the same motherboard
  - In ccNUMA design, each processor has its own memory system.
  - The phrase 'Non Uniform Memory access' refers to the potential difference in latency

Source : http://www.amd.com

Multi Cores Today

Dual-Core AMD Opteron Processor configuration

- AMD : Cache-Coherent nonuniform memory access (ccNUMA)
  - Two or more processors are connected together on the same motherboard
  - Dual Core AMD Opteron processors share the on-chip integrated memory controller and memory.
  - Because of the difference in latencies in ccNUMA systems, the OS must make determinations that enable best performance.
  - The optimization applies to 32-bit & 64 bit software

General 64-Bit Optimisations

Dual-Core AMD Opteron Processor configuration

- Current Systems (Intel /AMD 64 architecture) allows a 64-bit operating system to run existing 16-bit & 32-bit applications
- 64-bit mode , which provides 64-bit addressing and expanded register resources to support higher performance for re-complied 64-bit programs.
  - Use 64-bit registers for 64-bit integer arithmetic.
  - Load-Execute Instructions.

General 64-Bit Optimizations

- Current Systems (Intel /AMD 64 architecture) allows a 64-bit operating system to run existing 16-bit & 32-bit applications

- 64-bit mode, which provides 64-bit addressing and expanded register resources to support higher performance for re-complied 64-bit programs.
  - Use 64-bit registers for 64-bit integer arithmetic.

Instruction-Decoding Optimizations

- Load-Execute Instructions

Cache & Memory Optimisations

- Memory-Size Mismatches
- Cc-NUMA
- Prefetch instructions (Increase effective Bandwidth)
- Memory Copy
- Stack Considerations
- Cache Issues when Writing Instruction Bytes to Memory

Programming Models

- Data Parallel models
  - Microsoft Research Accelerator
- Multi-threaded Models
  - OpenMP, MPI
  - Cilk
  - CUDA (NviDIA)
- Streaming Models
  - Streamit
  - Cilk
  - Peakstream (Brook)

- Fortran 95, C, and C++ compilers for Linux on AMD Opteron and Intel 64-bit and 32-bit x86 CPUs

Compiler Comparisons Table

Critical Features Supported by x86 Compilers

	Vector SIMD Support	Peels Vector Loops	Global IPA	OpenMP	Links ACML Lib	Profile Guided Feedback	Aligns Vector Loops	Parallel Debuggers	Large Array Support	Medium Memory Model
PGI	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
GNU	✗	✗	✗	✗	✓	✓	✗	✗		✓
Intel	✓	✓	✓	✓		✓	✓	✓	✓	✗
Pathscale	✓		✓	✓	✓	✓	✓	✓	✓	✓
SUN	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Intel Compiler use Intel MKL libraries

Gains from tuning categories

Tuning Category	***Typical Range of Gain***
Source range	***25-100%***
Compiler Flags	***5-20%***
Use of libraries	***25-200%***
Assembly coding / tweaking	***5-20%***
Manual prefetching	***5-30%***
TLB thrashing/cache	***20-100%***
Using vis.inlines/micro-vectorization	***100-200%***

Compiler Techniques : Background

❖ Compilers (1)

- Compilers : translate the abstract operational semantics of a program into a form that makes effective use of a highly complex machine architecture
- Different architectural features exist and sometimes interact in complex ways.
- There is often trade-off between exploiting parallelism and exploiting locality to reduce yet another widening gap the memory wall.
- For the compiler : This means combining multiple program transformations (polyhedral models are useful here)
- Access latency and bandwidth of the memory subsystems have always been a bottleneck. Get worse with Mutli-core..

Compiler Techniques : Background

❖ Compilers (2)

- Program optimization is over huge and unstructured search spaces: this combinational task is poorly achieved in general, resulting in weak scalability and disappointing sustained performance.
- Even when programming models are explicitly parallel (data parallelism, threads, etc.,) advanced compiler technology is needed.
  - To relieve the programmer from scheduling and mapping the application to computational cores
  - For understanding the memory model and communication details

Compiler Techniques : Background

❖ Compilers (3)

- Even with annotations (e.g., OpenMP directives) or sufficient static information, compilers have a hard time exploring the huge and unstructured search space associated with lower level mapping and optimization challenges.

- The compiler and run-time system are responsible for most of the code generation decisions to map the simplified and ideal operational semantics of the source program to the highly complex machine architecture.

Complier Optimizations flags

Platform	Compiler Command	Description
IBM AIX	`xlc_r / cc_r`	C (ANSI / non-ANSI)
	`xlc_r`	C++
	`xlf_r -qnosave` `xlf90_r -qnosave`	Fortran – using IBM's Pthreads API (non-portable)
INTEL LINUX	`icc -pthread`	C
	`icpc -pthread`	C++
COMPAQ Tru64	`cc -pthread`	C
	`cxx -pthread`	C++
All Above Platforms	`gcc -pthread`	GNU C
	`g++ -pthread`	GNU C++
	`guidec -pthread`	KAIC (if installed)
	`kcc -pthread`	KAIC++ (if installed)

Compiler Options for Performance

Compiler optimization options:

- -xO1 thru -xO5 (default is “none”, -O implies -xO3)
- -fast:easy to use, best performance on most code,but it assumes compile platform = run platform and makes Floating point arithmetic simplifications.
- Understand program behavior and assert to optimizer:
  - -xrestrict, if only restricted pointers are passed to functions
  - -xalias_level, if pointers behave in certain ways
  - -fsimple if FP arithmetic can be simplified
- Target machine-related:
  - -xprefetch, -xprefetch_level
  - -xtarget=, -xarch=, -xcache=, -xchip=
  - -xvector to convert DO loops into vector

Single Core Performance: Compiler Optimization

Compiler Optimization Switches

- Fortran and C compilers have different levels of optimization that can do a fairly good job at improving a program's performance. The level is specified at compilation time with –O switch.
  - A same level of optimization on different machines will not always produce the same improvements (don't be surprised!)
  - –O is either default level of optimization. Safe level of optimization.
  - – O2 (same as –O on some machines) simple inline optimizations
  - – O3 (and –O4 on some machines) more complex optimizations designed to pipeline code, but may alter semantics of program
  - – fast Selects the optimum combination of compilation options for speed.
  - – parallel Parallelizes loops.
- Quite often, just a few simple changes to one's code improves performance by a factor of 2,3, or better!

Basic Compiler Techniques : Local variables on the Stack

- `-stackvar`
  - Tells the compiler to put most variables on the stack rather than statically allocate them.
  - `-stackvar` is almost always a good idea, and it is crucial when parallelization.
  - Concurrently running two copies of a subroutine that uses static allocation almost never works correctly.
  - You can control stack versus static allocation for each variable.
  - Variables that appear in DATA, COMMON, SAVE, or EQUIVALENCE statements will be static regardless of whether you specify `-stackvar`.

Basic Compiler Techniques

(Contd..)

- `fast`
  - Run program with a reasonable level of optimization may change its meaning on different machines.
  - It strikes balance between speed, portability, and safety.
  - `-fast` is often a good way to et a first-cut approximation of how fast your program can run with a reasonable level of optimization
  - `-fast` should not be used to build the production code.
  - The meaning of `-fast` will often change from one release to another
  - As with `-native`, `-fast` may change its meaning on different machines

Single Core : Compiler Features

- `O` : Set optimization level
- `fast` : Select a set of flags likely to improve speed
- `stackvar` : put local variables on stack
- `xlibmopt` : link optimized libraries
- `xarch` : Specify instruction set architecture
- `xchip` : Specifies the target processor for use by the optimizer.
- `native` : Compile for best performance on localhost.
- `xprofile` : Collects data for a profile or uses a profile to optimize.
- `fns` : Turns on the SPARC nonstandard floating-point mode.
- `xunroll n` : Unroll loops n times.

Source Code Optimizations

- Improve usage of data cache, TLB
- Use VIS instructions (templates) directly, via –xvis option
- Optimize data alignment (also: #pragma align,dalign)
- Prevent Register Window overflow
- Creating inline assembly templates for performance critical routines
- Loop Optimizations that compilers may miss:
  - Restructuring for pipelining and prefetching
  - Loop splitting/fission
  - Loop Peeling
  - Loop interchange
  - Loop unrolling and tiling
  - Pragma directed

Parallel programming-Compilation switches

Automatic and directives based parallelization

Allow compiler to do automatic and directive – based parallelization

- `-x autopar`, `-x explicitpar`, `-x parallel`, -tell the compiler to parallelize your program.
  - `xautopar`: tells the compiler to do only those parallelization that it can do automatically
  - `xexplicitpar`: tells the compiler to do only those parallelization that you have directed it to do with programs in the source
  - `xparallel`: tells the compiler to parallelize both automatically and under pragma control
  - `xreduction`: tells the compiler that it may parallelize reduction loops. A reduction loop is a loop that produces output with smaller dimension than the input.

Path Scale Compiler Benchmarks

- **Pathscale Compilers publish SPEC results on SPEC web-site**
  - SPEC@CPU2000
  - SPEC@Int2000
  - SPEC@fp2000
  - SPEC ompM2001 suite of OpenMP Benchmarks
- AMD SPEC Results for Dual Core Opteron using PathScale Compilers
- IBM, HP, Fujitsu-Siemens, Sun and AMD use PathScale Compliers to get performance on AMD64-based Linux Systems.

http://www.**pathscale**.com/

Maintaining Stability while Optimizing

- **STEP 0:** Build application using the following procedure:
  - compile all files with the most aggressive optimization flags below:
    - `-tp k8-64 –fastsse`
  - if compilation fails or the application doesn't run properly, turn off vectorization:
    - `-tp k8-64 –fast –Mscalarsse`
  - if problems persist compile at Optimization level 1:
    - `-tp k8-64 –O0`
- **STEP 1:** Profile binary and determine performance critical routines
- **STEP 2:** Repeat STEP 0 on performance critical functions, one at a time, and run binary after each step to check stability

PGI Compiler Flags – Optimization Flags

- Below are 3 different sets of recommended PGI compiler flags for flag mining application source bases:
  - **Most aggressive: -tp k8-64 –fastsse –Mipa=fast**
    - enables instruction level tuning for Opteron, O2 level optimizations, sse scalar and vector code generation, inter-procedural analysis, LRE optimizations and unrolling
    - strongly recommended for any single precision source code
  - **Middle of the ground: -tp k8-64 –fast –Mscalarsse**
    - enables all of the most aggressive except vector code generation, which can reorder loops and generate slightly different results
    - in double precision source bases a good substitute since Opteron has the same throughput on both scalar and vector code
  - **Least aggressive: -tp k8-64 –O0 (or –O1)**

PGI is an independent supplier of high performance scalar and parallel compilers and tools for workstations, servers, and high-performance computing. http://www.pgroup.com/

PGI Compiler Flags – Functionality Flags

- **-mcmodel=medium**
  - use if your application statically allocates a net sum of data structures greater than 2GB
- **-Mlarge_arrays**
  - use if any array in your application is greater than 2GB
- **-KPIC**
  - use when linking to shared object (dynamically linked) libraries
- **-mp**
  - process OpenMP/SGI directives/pragmas (build multi-threaded code)
- **-Mconcur**
  - attempt auto-parallelization of your code on SMP system with OpenMP

PGI Compiler Flags – Optimization Flags

Below are 3 different sets of recommended PGI compiler flags for flag mining application source bases:

- **Most aggressive: -O3**
  - loop transformations, instruction preference tuning, cache tiling, & SIMD code generation (CG). Generally provides the best performance but may cause compilation failure or slow performance in some cases
  - strongly recommended for any single precision source code
- **Middle of the ground: -O2**
  - enables most options by –O3, including SIMD CG, instruction preferences, common sub-expression elimination, & pipelining and unrolling.
  - in double precision source bases a good substitute since Opteron has the same throughput on both scalar and vector code
- **Least aggressive: -O1**

Absoft Compiler Flags – Functionality Flags

- **-mcmodel=medium**
  - use if your application statically allocates a net sum of data structures greater than 2GB
- **-g77**
  - enables full compatibility with g77 produced objects and libraries
    - **(must use this option to link to GNU ACML libraries)**
- **-fpic**
  - use when linking to shared object (dynamically linked) libraries
- **-safefp**
  - performs certain floating point operations in a slower manner that avoids overflow, underflow and assures proper handling of NaNs

Pathscale Compiler Flags – Optimization Flags

PathScale Compiler Suite has been optimized for both the AMD64 and EM64T architectures. The PathScale™ Compiler Suite is consistently proving to be the highest performing 64-bit compilers for AMD-based Opteron.

- **Most aggressive: -Ofast**
  - Equivalent to –O3 –ipa –OPT:Ofast –fno-math-errno
- **Aggressive : -O3**
  - optimizations for highest quality code enabled at cost of compile time
  - Some generally beneficial optimization included may hurt performance
- **Reasonable: -O2**
  - Extensive conservative optimizations
  - Optimizations almost always beneficial
  - Faster compile time
  - Avoids changes which affect floating point accuracy

http://www.**pathscale**.com/

Pathscale Compiler Flags – Functionality Flags

- **- mcmodel=medium**
  - use if static data structures are greater than 2GB
- **- ffortran-bounds-check**
  - (fortran) check array bounds
- **- shared**
  - generate position independent code for calling shared object libraries
- **Feedback Directed Optimization**
  - **STEP 0**: Compile binary with -fb_create_fbdata
  - **STEP 1**: Run code collect data
  - **STEP 2**: Recompile binary with -fb_opt fbdat
- **- march = (opteron|athlon64|athlon64fx)**
  - Optimize code for selected platform (Opteron is default)

http://www.**pathscale**.com/

Pathscale Compiler Flags – Functionality Flags

PathScale 2.1 64-bit optimization flags:

F77: -O3 -LNO:fu=9OPT:div_split:fast_math:fast_sqrt -IPA:plimit=3500

F90: -Ofast -OPT:fast_math=on -WOPT:if_conv=off -LNO:fu=9:full_unroll_size=7000

http://www.**pathscale**.com/

Intel Caneland (Quad Core) System Configuration

Comp System Conf.	Intel Caneland (Quad Socket Quad Core)
CPU	Quad-Core Genuine Intel(R) CPU - Tigerton
No of Sockets /Cores	4 Sockets (Total : 16 Cores)
Clock-Speed	2.4 GHz per Core
Peak(Perf.)	153.6 Gflops
Memory/Core	4 GB per Core
Memory type	FBDIMM
Total Memory	64 GB
Cache	L1 = 128 KB; L2 = 8 MB Per socket shared
OS	Red Hat Enterprise Linux Server release 5 (Tikanga) x86_64 (64 bit)
Compilers	Intel 10.0(icc; fce; OpenMP)
MPI	Intel (/opt/intel/ict/3.0.1/mpi/3.0/bin64)
Math Libraries	Math Kernel Library 9.1

Linear Algebra (LA)

- **B**asic **L**inear **A**lgebra **S**ubroutines (**BLAS**)
  - Level 1 (vector-vector operations)
  - Level 2 (matrix-vector operations)
  - Level 3 (matrix-matrix operations)
  - Routines involving sparse vectors
- Linear **A**lgebra **PACK**age **(LAPACK)**
  - leverage BLAS to perform complex operations
  - 28 Threaded LAPACK routines
  - **Fast Fourier Transforms** (**FFTs**)
- 1D, 2D, single, double, r-r, r-c, c-r, c-c support

- **C and Fortran interfaces**

Multi Core Processors Performance: Use of MATH LIBRARIES

- BLAS, IMSL, NAG, LINPACK, ScaLAPACK LAPACK, etc.
  - Calls to these math libraries can often simplify coding.
  - They are portable across different platforms
  - They are usually fine-tuned to the specific hardware as well as to the sizes of the array variables that are sent to them
  - Example : Intel MKL & AMD Opteron ACML
- User can often parallelize at a higher level by running the performance subroutines serially.
  - It also has more favorable cache behavior
  - Synchronization points may be less
  - Performance gain is expected but depends on the problem size.

Mathematical Libraries (Benchmarks)

Features

- BLAS, LAPACK, FFT Performance
- Open MP Performance
- **AMD :** ACML 2.5 /2.X or 3.X
- **Intel :** MKL
- **IBM** : Power 5/6 - ESSL

- **How good is Benchmark performance?**

Top500 : Benchmark on Multi Core Systems

Multi Core (CPUs)	HPL Matrix Size/ Block size/ (P,Q)	Peak Perf (Gflops)	Sust. Perf (Gflops)	Utilization (%)
4	40960/120(2,2)	38.4	32.54	84.73
8	42240/120(4,2)	76.8	60.72	79.06
16	40960/200(4,4)	153.6	97.09	63.20
	83456/200(4,4) Used 56 GB	153.6$	116.2	76.0
	88000/200(4,4) * 64 GB can be used	153.6*	122.3	79.4

Used Env : Intel 10.0(icc, MPI); Compiler Flag : -O3, -funroll-loops,-fomit-frame-pointer.

For Top-500, algorithm parameters, tuning & performance of Compiler optimisations are not tried to extract the sustained Performance.

Conclusions

- An Overview of Compilation Features of Multi Core Systems
- Performance of Top-500 Benchmark using Compiler Flags.
- An Overview of Compiler Suite performing 64-bit compilers for AMD-based Opteron

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

References

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

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

References

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)**

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)**

Thank You

Any questions ?