

C-DAC Four Days Technology Workshop

ON

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

hyPACK-2013

Mode-1 : Multi-Core & Mode-2 : ARM Processors

**Topic : An Overview Multi-Core Processors -
Architecture, Prog. Env - Software Threading &
Performance Issues & An Overview of ARM
Multi-Core Processors**

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

An Overview of Multi-Core & ARM Processors

Lecture Outline

Following topics will be discussed

- ❖ Understanding of Multi-Core Architectures
- ❖ Programming on Multi-Core Processors
- ❖ Tuning & Performance – Software Threading
- ❖ An Overview of ARM Processors – Prog. Env

Part-I :

An Overview of Multi-Core Processors

History

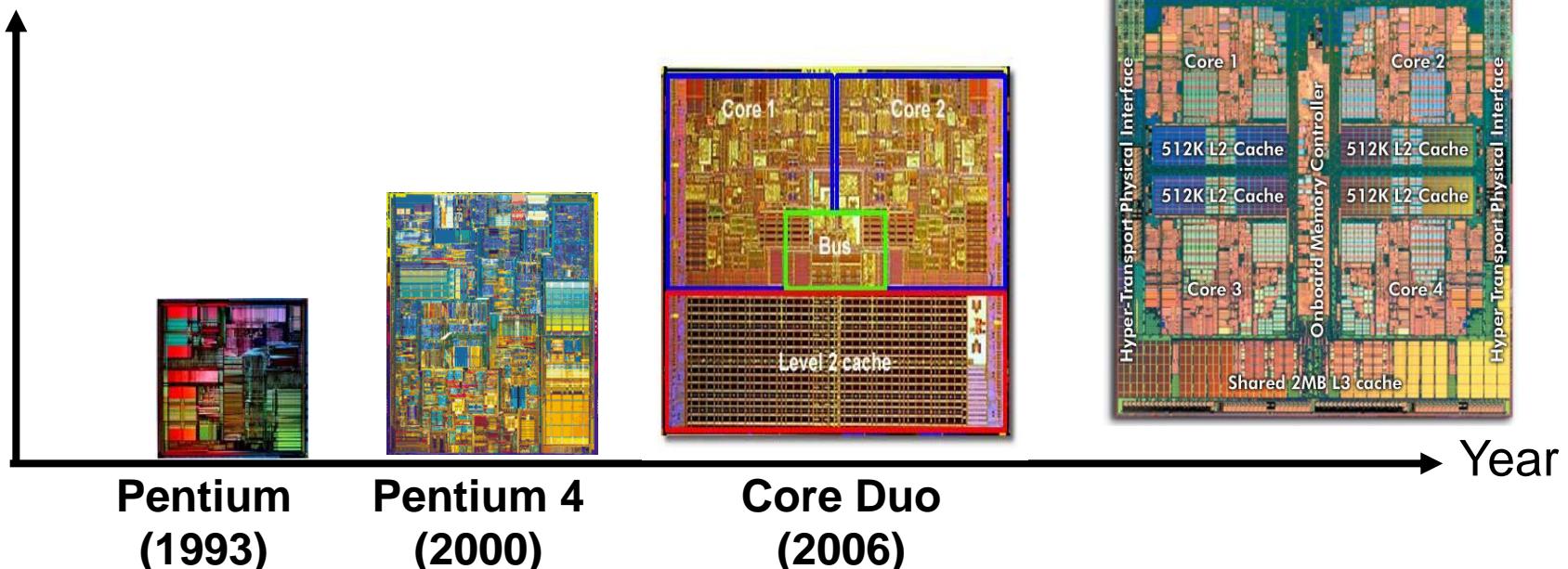
Multi-Core History

- ❖ Parallel computing has been used for HPCs and networking.
 - PRAM & other shared memory models - Limitations.
 - BSP & LogP (message passing models) were used.
 - Only for HPC specialists.
 - Demand complicated system analyze per application.
 - Clusters are becoming popular (NUMA, CC NUMA)
- ❖ HW (Scalability) Constraints force Multicore architectures.
- ❖ Era of frequency race

Multi-Core History

- ❖ No more frequency race
 - The era of multi cores
- ❖ Parallel programming is not easy
 - Split a sequential task into multiple sub tasks
 - Data Parallelism – Task Parallelism

Performance



Multi-Core History

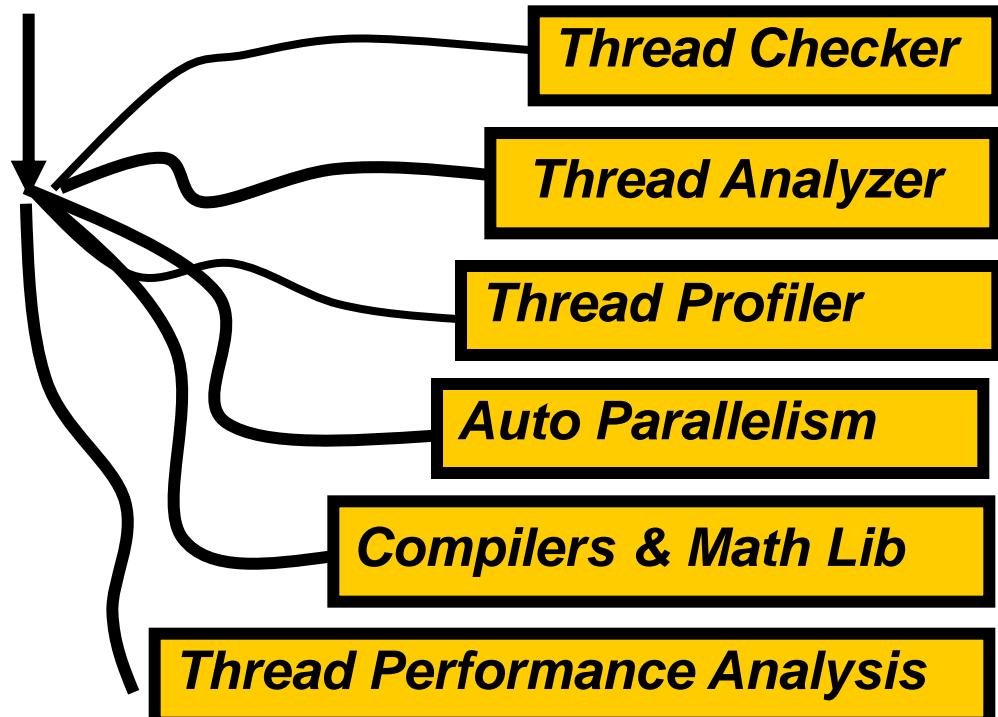
- ❖ Moving from Multiple processor on single box (SMP) Multiple Core on Single Chip.
- ❖ Four, even six/eight/twelve, processor cores on the same die are fast becoming commonplace.
- ❖ Moving to a **multi-core world** means applications will have to be written in a different manner.

Source : <http://www.intel.com> ; Reference : [6]

Multi-Core History

- ❖ Multi-core architectures involve multi-processing, and to take advantage of that, parallel programming is almost compulsory.
- ❖ The lack of parallel-programming tools and expertise is threatening the progress of multi-core architectures.

Source : <http://www.intel.com> ; Reference : [6]



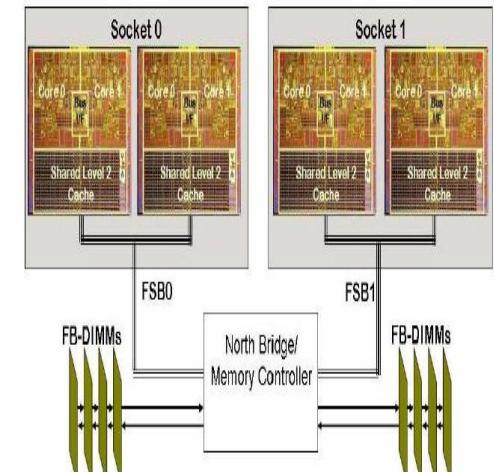
Multi-Cores - Parallel Programming -Difficulties

Programmers Challenge :

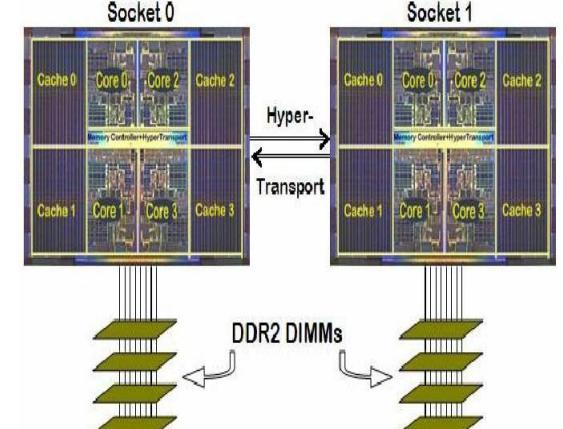
- ❖ Identify independent pieces of a task that can be executed in parallel
 - Coordinate their Execution
 - Managing Communications
 - Managing Synchronization
- ❖ A program with a communication or synchronization bottleneck will be unable to take full advantage of the available Cores
- ❖ Scalable Programs that avoid such bottlenecks are surprisingly difficult to construct

Reference : [6], [29], [31]

Intel Quad Core (Clovertown)



AMD Quad Core



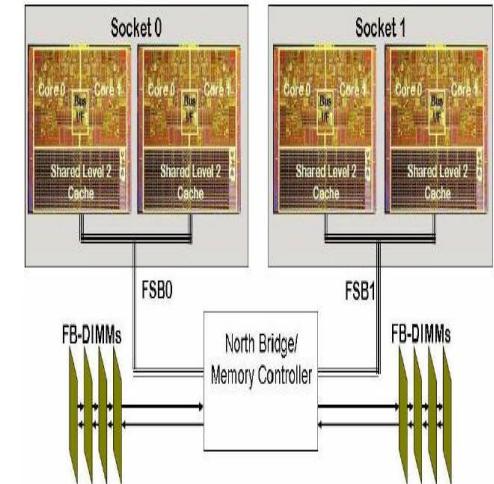
Multi-Cores - Parallel Programming -Difficulties

Challenge: taking advantage of Multi-Core

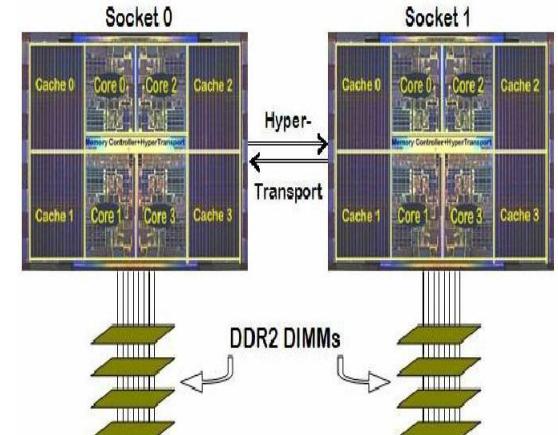
- ❖ Parallel Prog. is difficult with locks:
 - Deadlock, convoys, priority inversion
 - Conservative, poor composability
 - Lock ordering complicated
 - Performance-complexity tradeoff

- ❖ Transactional Memory in the OS
 - Benefits user programs
 - Simplifies programming

Intel Quad Core (Clovertown)



AMD Quad Core

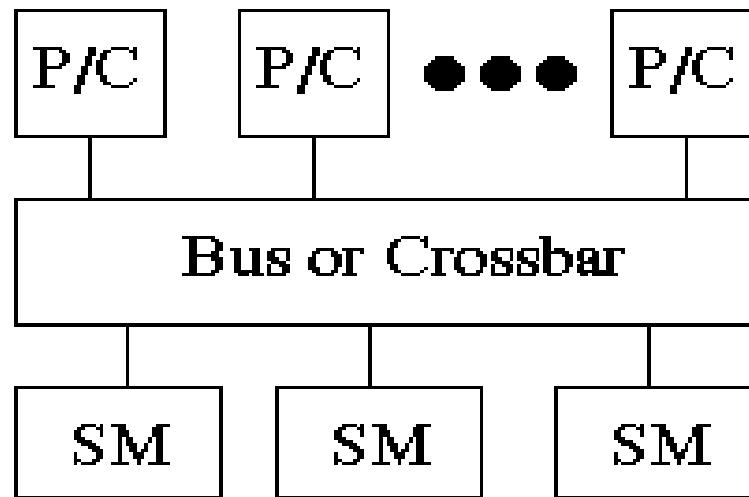


Multi-Cores - Parallel Programming Difficulties

- ❖ Multicore architectures force us to rethink how we do synchronization
- ❖ Parallel programming has traditionally been considered using locks to synchronize concurrent access to shared data.
- ❖ Standard locking model won't work
- ❖ Lock-based synchronization, however, has known pitfalls: using locks for fine-grain synchronization and composing code that already uses locks are both difficult and prone to deadlock.
- ❖ **Transactional model might**
 - Software
 - Hardware
 - Programming Issues

Symmetric Multiprocessors (SMPs) : Issues

(Contd...)

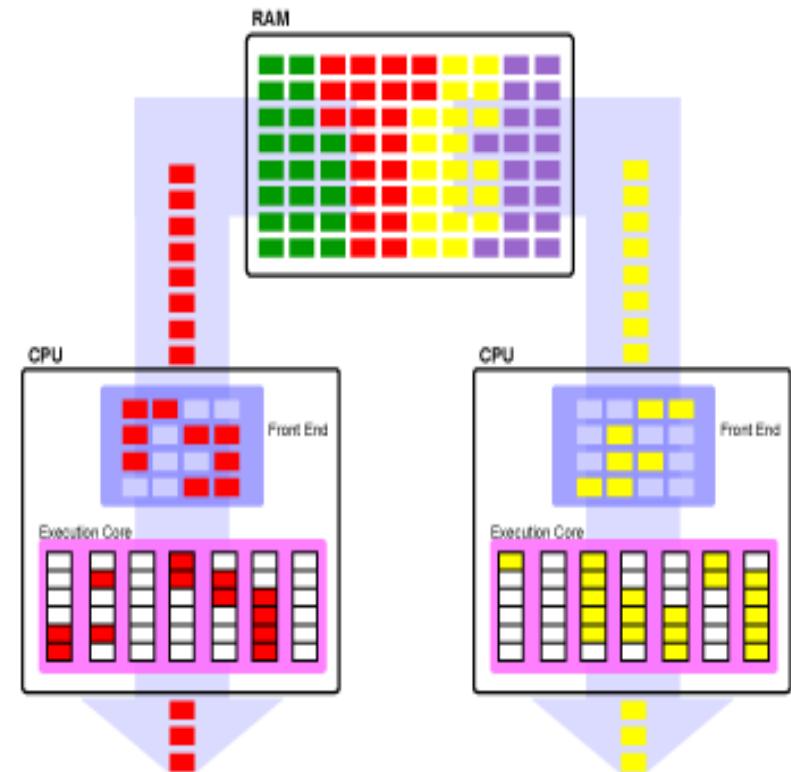


P/C : Microprocessor and cache; SM : Shared memory

- ❖ Uses commodity microprocessors with on-chip and off-chip caches.
- ❖ Processors are connected to a shared memory through a high-speed snoopy bus

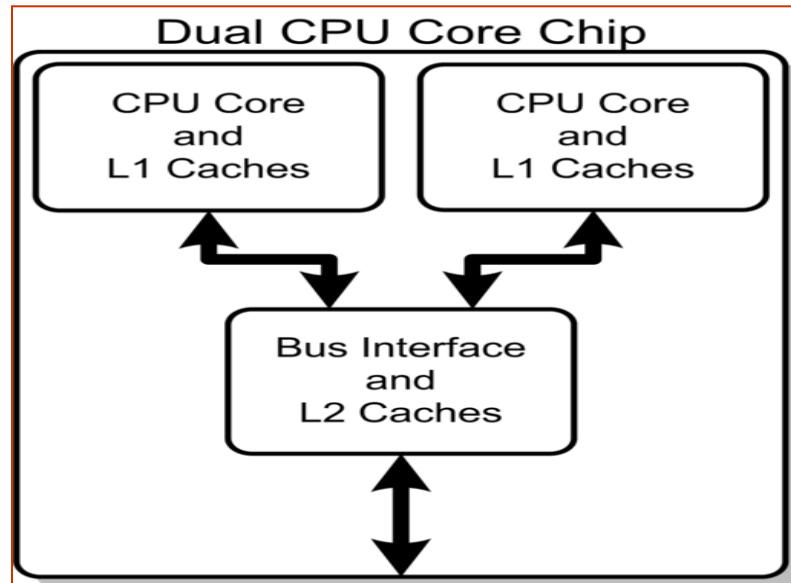
Symmetric Multiprocessors (SMPs) : Issues

- Two processors is involved.
- OS schedule two processes for execution by either CPU
- Not allowed to monopolize
- Less waiting time
- Number of empty execution slots doubled
- Efficiency - No improvement in CPU utilization
- Time Slicing Issues

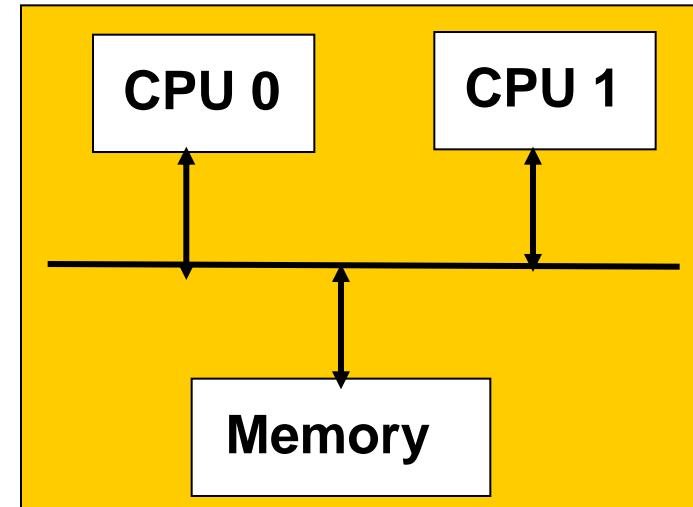


Reference : [6], [29], [31]

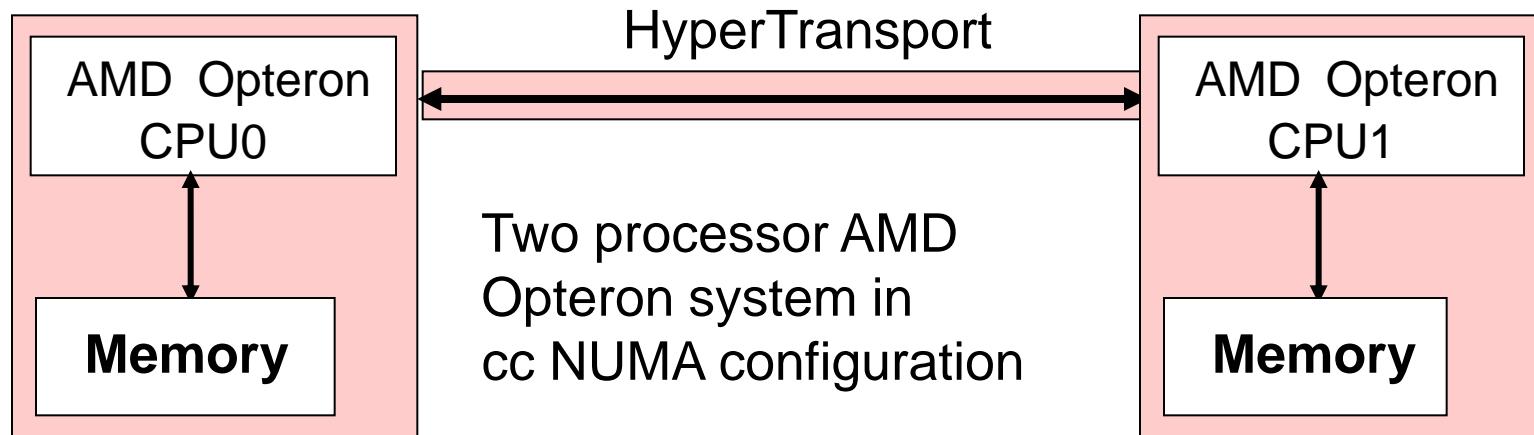
Multi Cores Today



Two processor Dual Core



Simple SMP Block Diagram
for a two processors

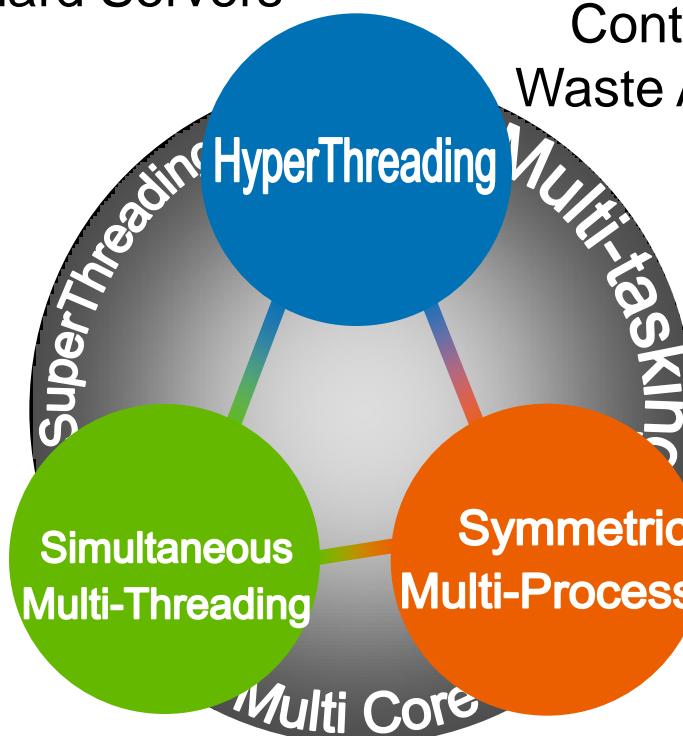
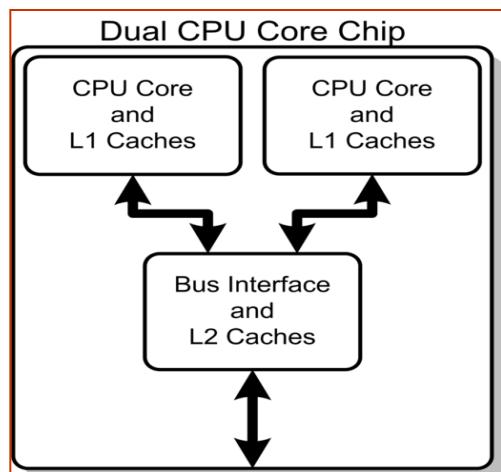


Multi-Cores - An Overview of threading

SMP and Cluster Platforms based on



Industry Standard Servers



Single Threaded CPU
Preemptive **vs.** co-operative
Multitasking
Context, process and Thread
Waste Associated with Threads
Time Slicing
I/O Threads

Implementing Hyper-threading

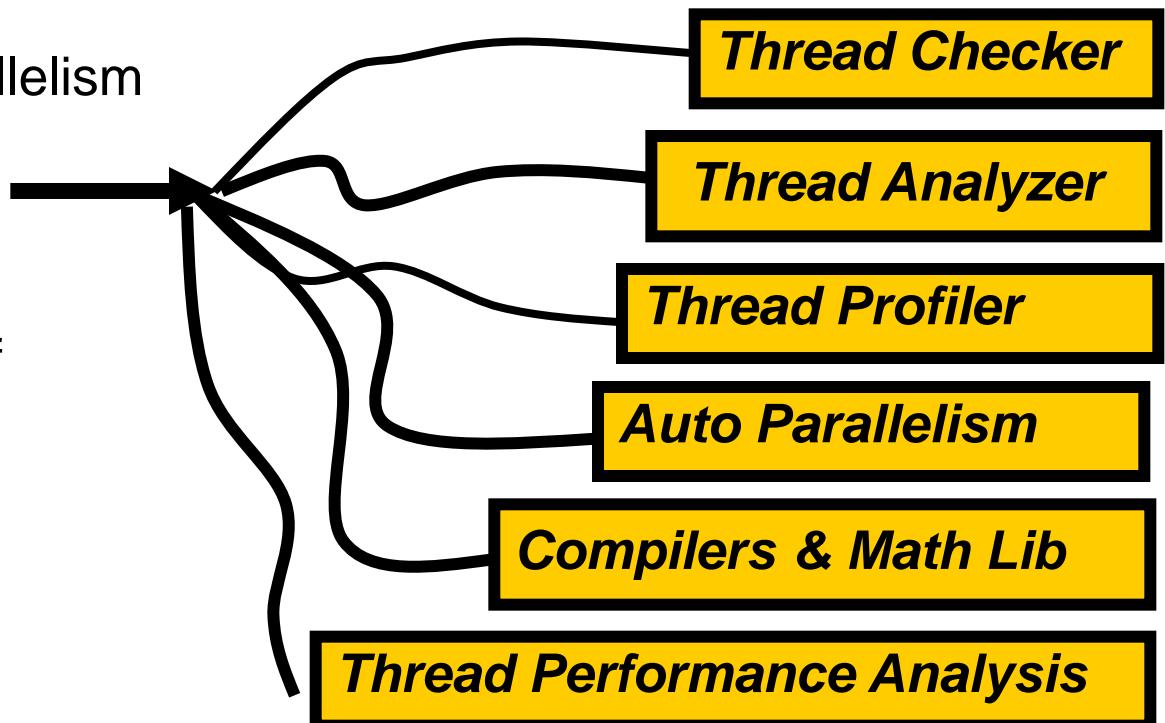
- Replicated
- Partitioned
- Shared
- Caching &
- SMT

Source : <http://www.intel.com> ; <http://www.amd.com>; Reference : [6], [29], [31]

MultiCores - An Overview of threading -tools

- ❖ Performance
- ❖ Tools to Discover Parallelism
- ❖ Use of Math Libraries
- ❖ Measure Overheads of Threads
- ❖ Hardware Counters

Source : <http://www.intel.com> ; Reference : [6]



Multi-Core Processors - Programming Issues

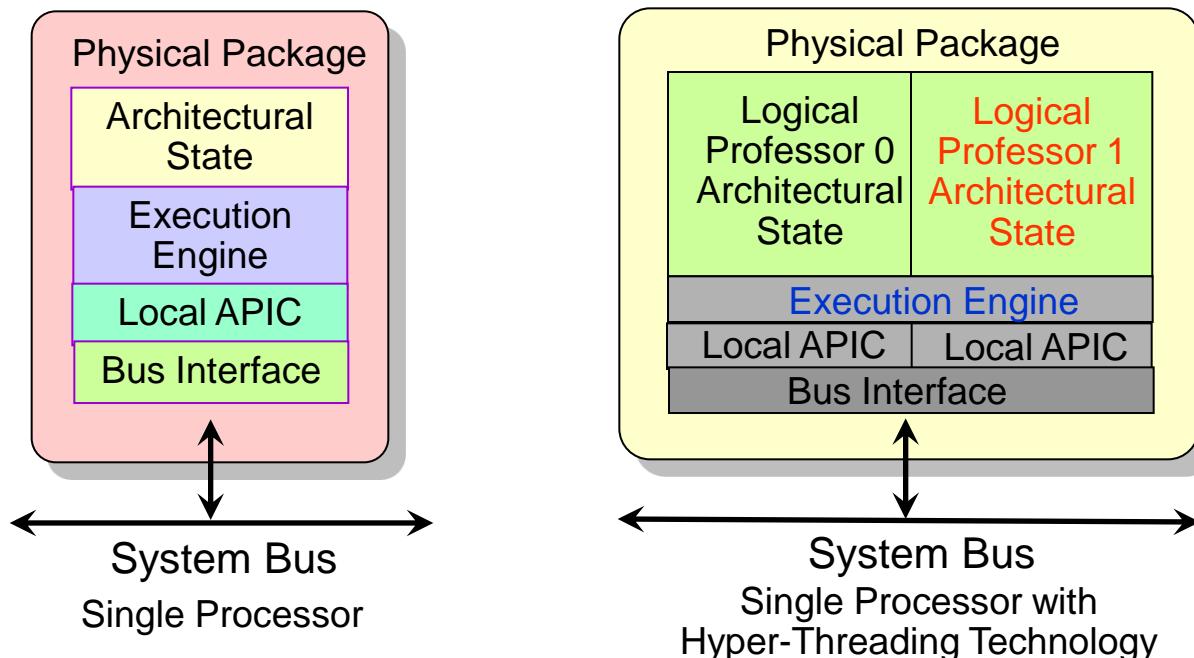
- ❖ Out of Order Execution
- ❖ Preemptive and Co-operative Multitasking
- ❖ SMP to the rescue
- ❖ Super threading with Multi threaded Processor
- ❖ Hyper threading the next step (Implementation)
- ❖ Multitasking
- ❖ Caching and SMT

Multi-Core Processors - Programming Issues

- ❖ **Issues & Challenges**
 - Process, context and thread
 - Multi-tasking: Each program has a mind of its own
 - Pipeline bubbles
 - Number of Instructions per clock cycle
 - Time slicing – up-restore- context switches
 - The concept of “State” – Halting /Restoring
 - Wasting number of number of CPU cycles...
 - Execution efficiency improves ?

Hyper-threading : Partitioned Resources

- ❖ Hyper-threading (HT) technology is a hardware mechanism where multiple independent hardware threads get to execute in a single cycle on a single super-scalar processor core.

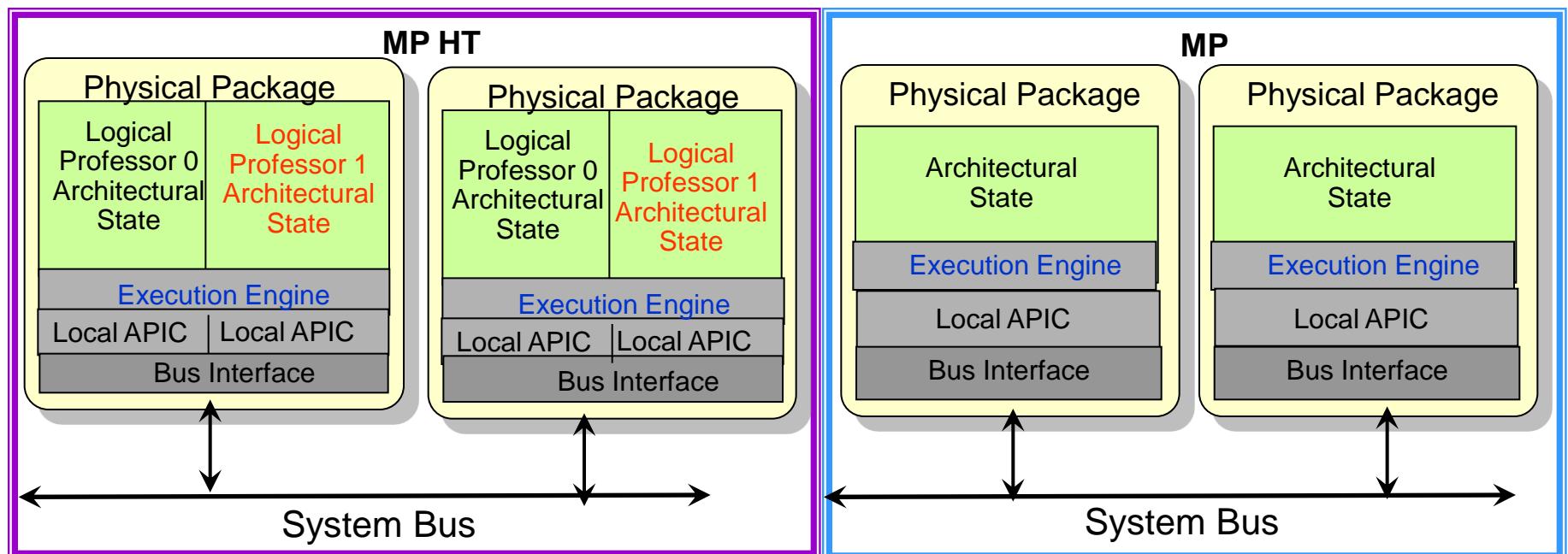


Single Processor System without Hyper-Threading Technology and Single Processor System with Hyper-Threading Technology

Source : <http://www.intel.com> ; Reference : [6], [10], [19], [23], [24],[29], [31]

Hyper-threading Technology

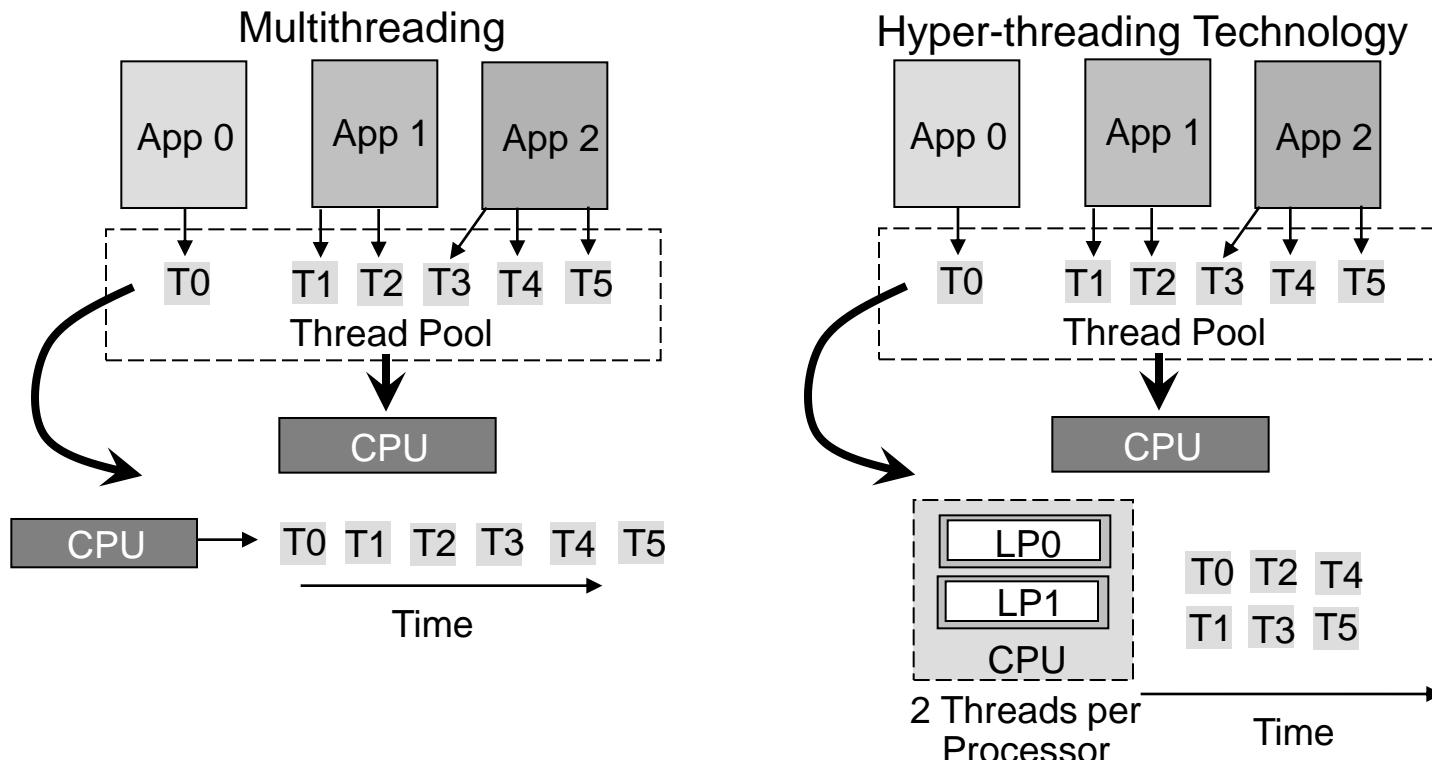
- ❖ Multi-processor with and without Hyper-threading (HT) technology



Source : <http://www.intel.com> ; Reference : [6], [29]. [31]

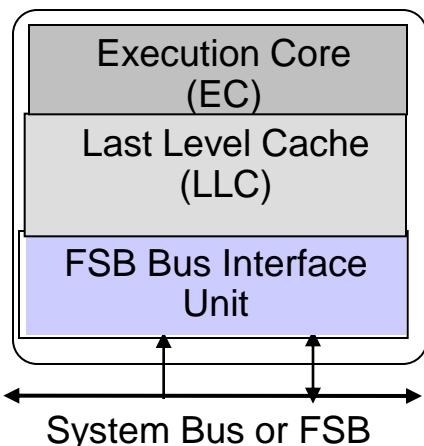
Multi-threaded Processing using Hyper-Threading Technology

- ❖ Time taken to process n threads on a single processor is significantly more than a single processor system with HT technology enabled.

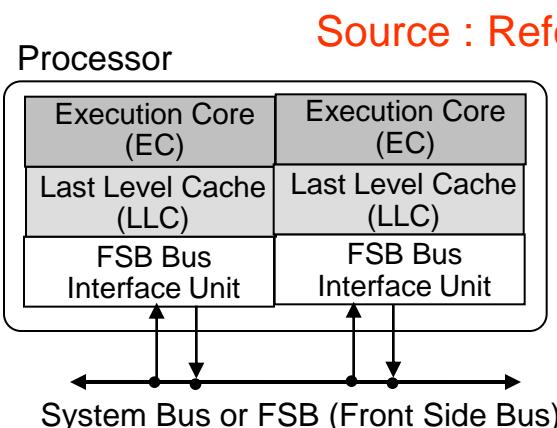


Source : <http://www.intel.com> ; Reference : [6], [29], [31]

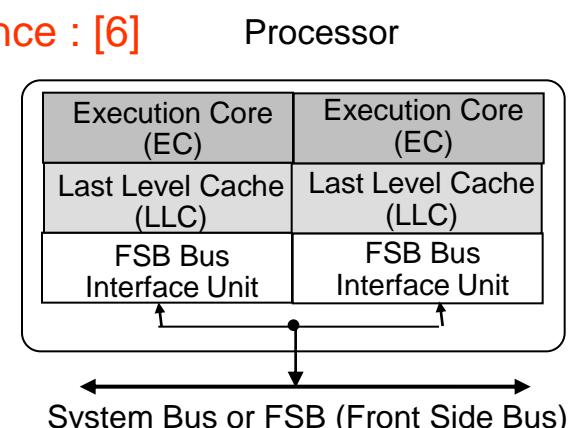
Multi-Core Processor Configurations



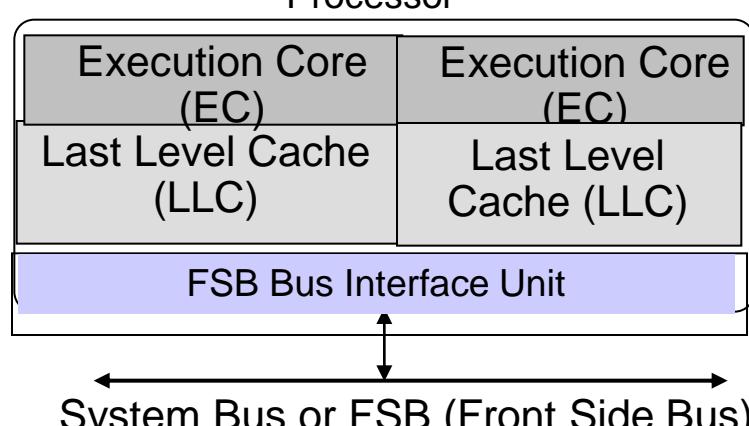
(a) Single Core Processor



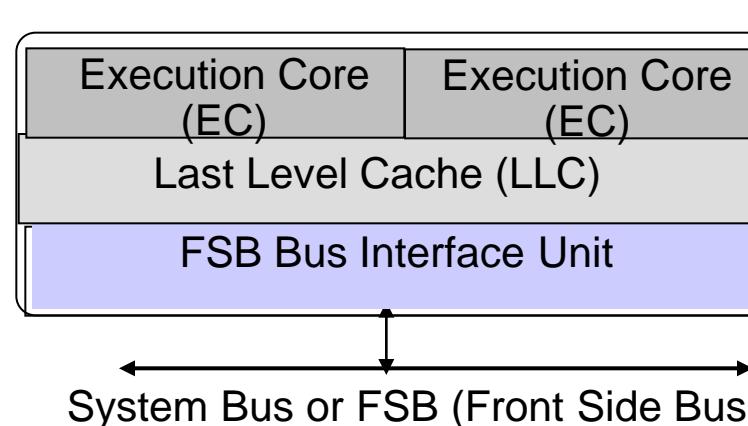
(b) Multi-Core Processor with Two Cores and Individual FSB



(c) Actual representation of show shared FSB

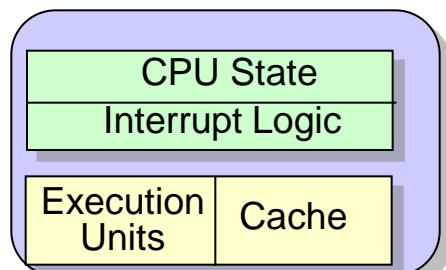


(d) Multi-Core Processor with Two Cores and Shared FSB

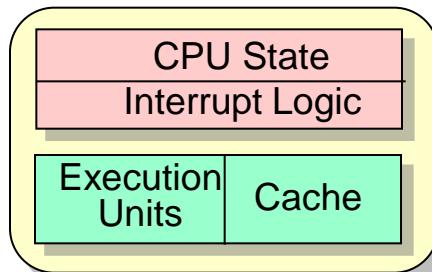


(e) Multi-Core Processor with Two Cores with Shared LLC and FSB

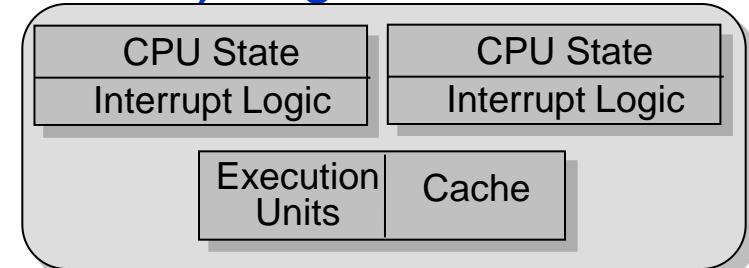
Simple Comparison of Single-core, Multi-processor, and multi-Core Architectures



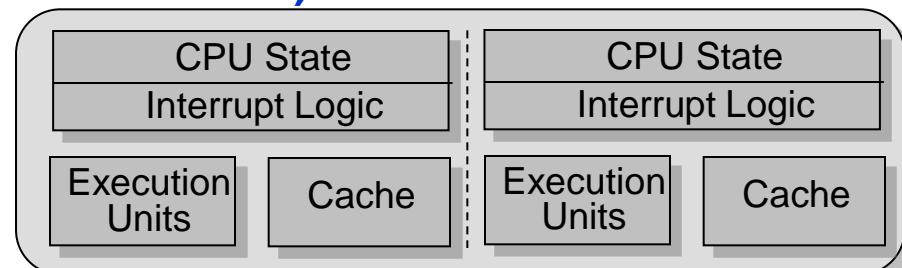
A) Single Core



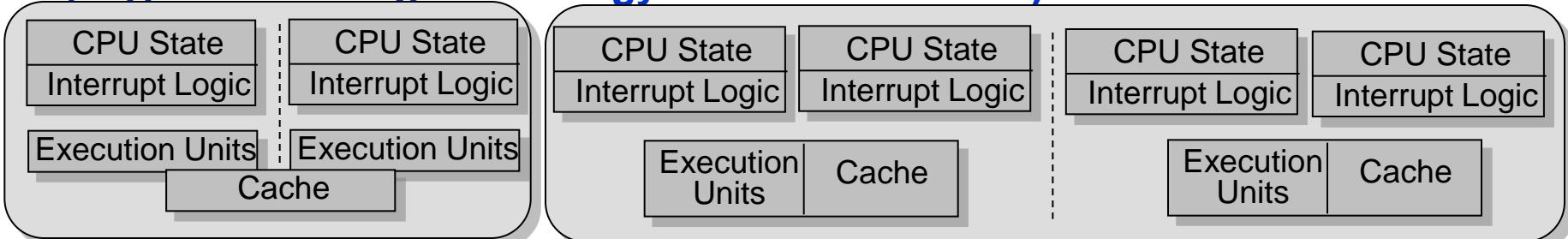
B) Multi Processor



C) Hyper-Threading Technology



D) Multi-core



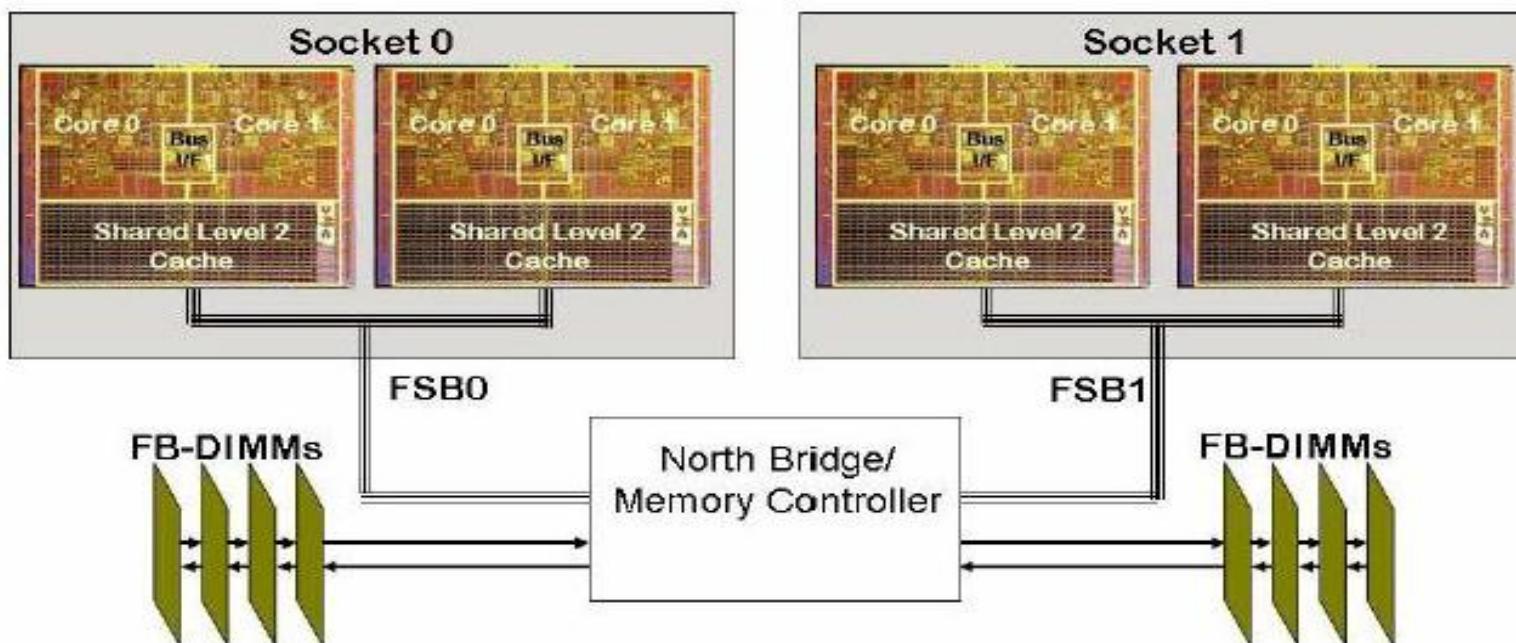
E) Multi-core with Shared Cache

F) Multi-core with Hyper-threading Technology

Source : <http://www.intel.com> ; Reference : [4],[6], [29], [31]

Multi-Core Processors

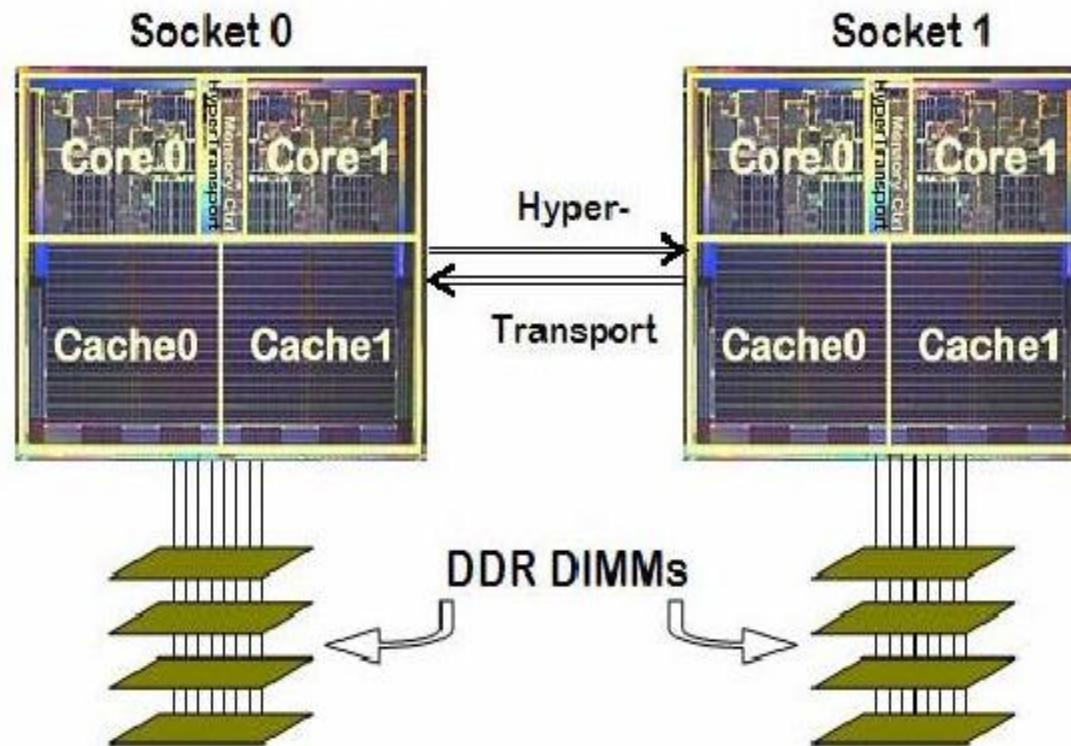
Intel Quad Core (Clovertown) Server Processor with *Blackford Chipset* : Clovertown will consist of two Woodcrest dice crammed into a single package.



source : <http://www.intel.com>

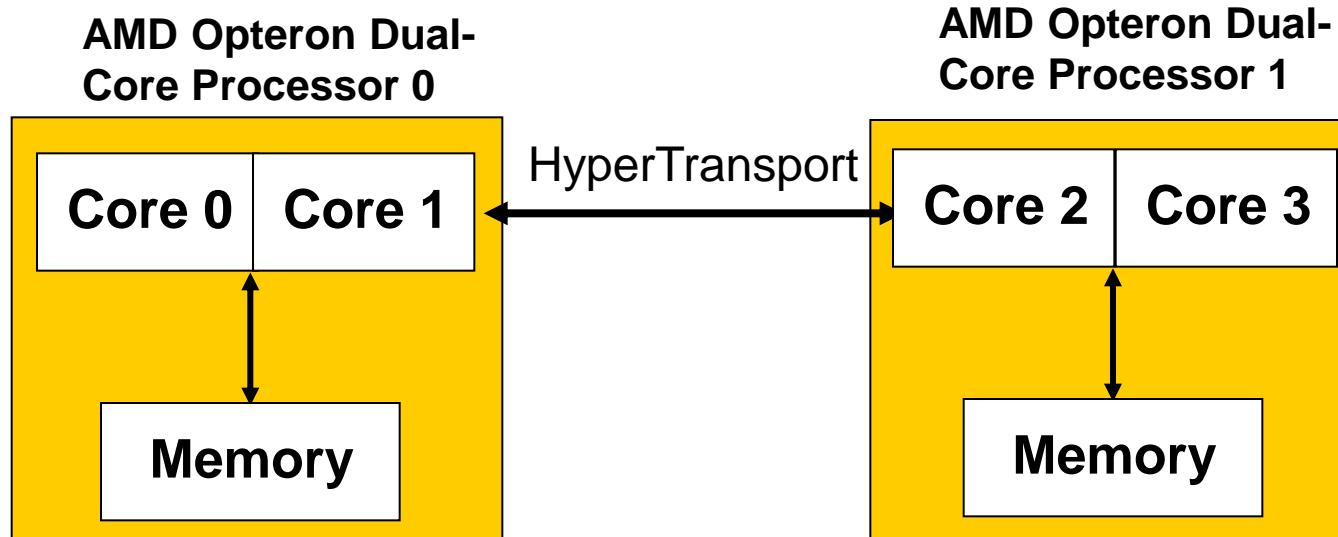
AMD Opteron Dual Core Processor

- ❖ AMD Dual-Core Opteron, Circa 2005
Socket F used in the motherboard OEMs



source : <http://www.amd.com>

AMD Multi Cores



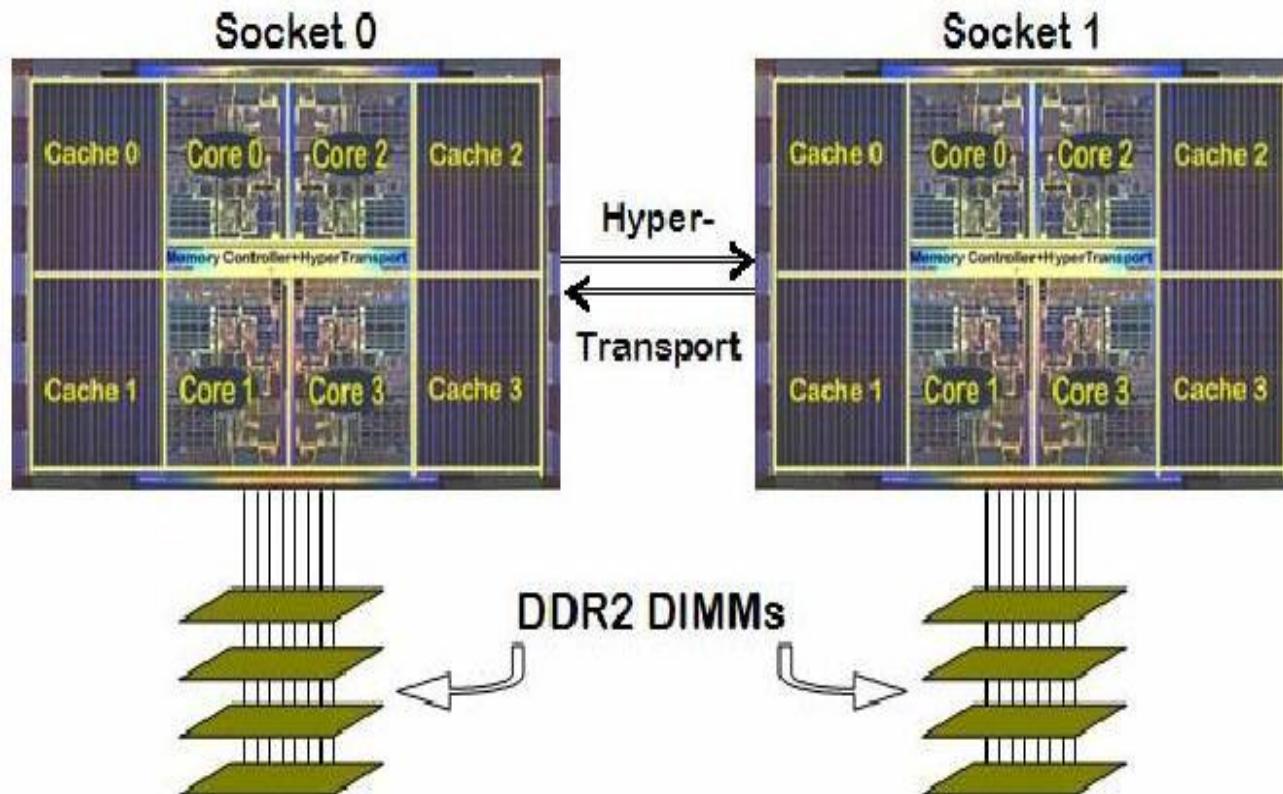
Dual-Core AMD Opteron Processor configuration

- ❖ 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>

AMD Opteron Quad Core Processor

- ❖ AMD Quad-Core Opteron, Circa 2007



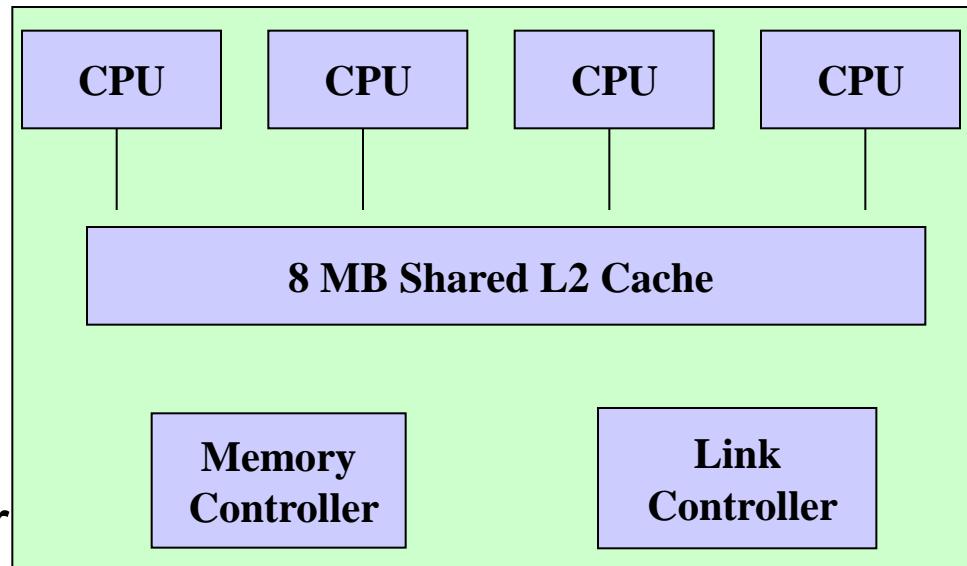
source : <http://www.amd.com>

Intel Dunnington (6 Core) System Configuration

Comp System Conf.	Intel Dunnington (six Core)
CPU – Frequency	2.67 GHz
No of Sockets /Cores	4 Sockets (Total : 24 Cores)
Chipset	Clarksboro
FSB Frequency	1067 MHz
Memory Technology	DDR2- 667 FB
Core Design/Technology	Penryn (45 nm)
L1 Dcache & L1 Cache	L1 Dcache=32 KB; L1 Cache =32 KB
L2 Cache	3 MB per 2 Core
L3 Cache	Shared 16 MB (40 ns latency)
Peak(Perf.)	256 Gflops (Approximately)
Memory/Core	2-4 GB per Core
Total Memory	48-96 GB
OS	Red Hat Enterprise Linux Server x86_64 (64 bit)
Prog. Env	Intel 10.0(icc; fce; OpenMP, MPI)
Math Libraries	Math Kernel Library 9.1

Intel Nehalem-EP 2 S (Quad Core) System

- ❖ 4 Cores
- ❖ 8M on-chip Shared Cache
- ❖ Intel QuickPath Interconnect
- ❖ Integrated Memory Controller (DD3)
- ❖ Power : 60W – 95 W
- ❖ Simultaneous Multi-Threading (**SMT**)
 - Single Core to execute 2 separate threads (Efficient Resource Utilisation & Greater Performance)
 - Using “*-multi n*” option to use multiple cores available on the Nehalem system



Intel Nehalem-EP 2 S (Quad Core) System

Features – to improve performance

- ❖ SMP : Multiple threads share resource such as L0, L1 Cache (*A small LO cache on top of a traditional L1 cache has the advantages of shorter access time and lower power consumption. The performance loss in case of cache misses is possible with LO & Cache Management schemes required.*)
- ❖ Streaming SIMD Extensions (SSE)
 - support SIMD operation & Vectorization
 - Increase processor throughput by performing multiple computations in a single instruction
 - Useful for Matrix Computations Data Parallel Operations
 - Useful for Complex Arithmetic & Video Codec Algorithms

Part-II :

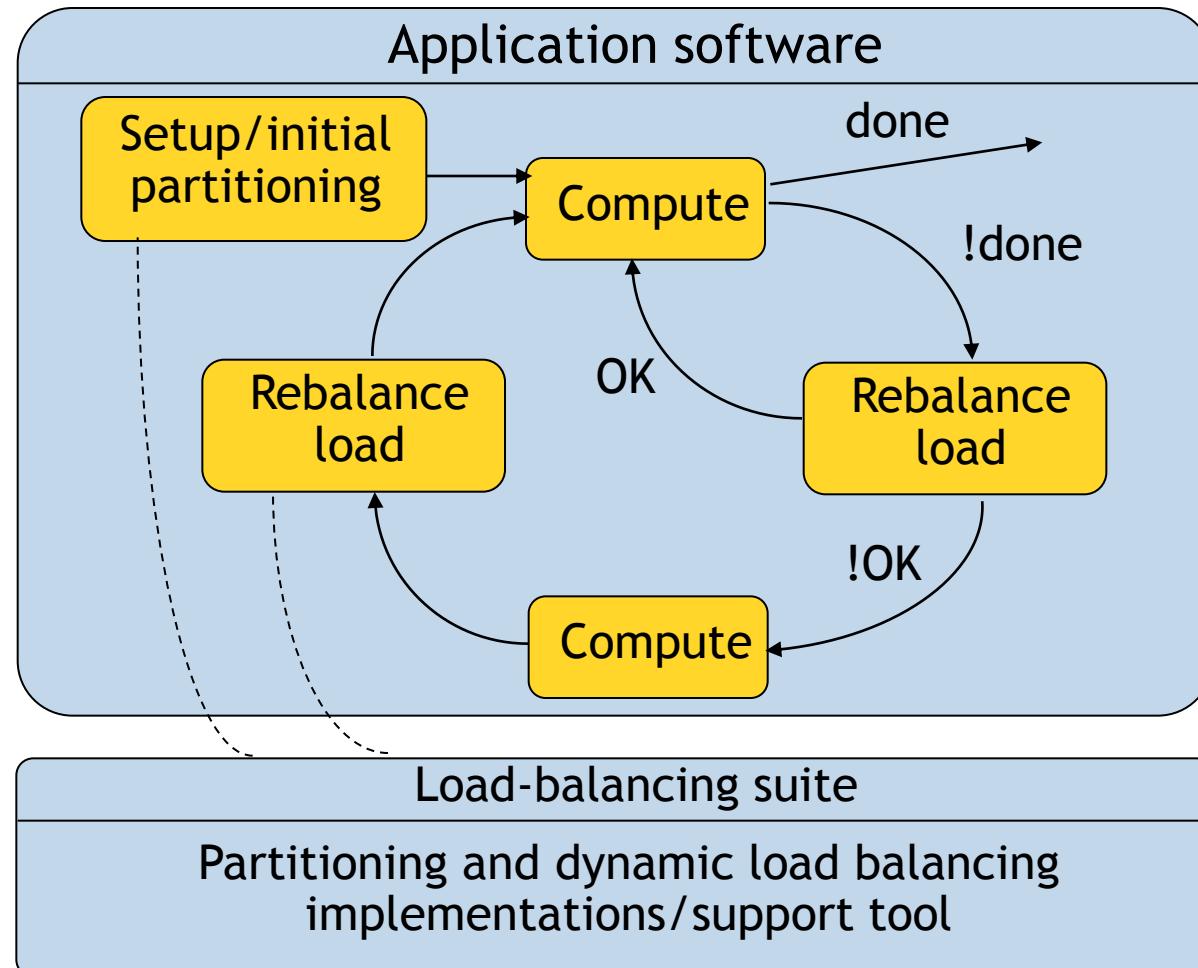
**An Overview of Multi-Core Processors
Application Perspective**

Application Perspective : Multi Cores

- ❖ Threads of Computation : Work is partitioned amongst the threads – Data Handling & Synchronization Issues

Computational requirements dynamically changes –

- Cache Friendly applications
- I/O Intensive applications

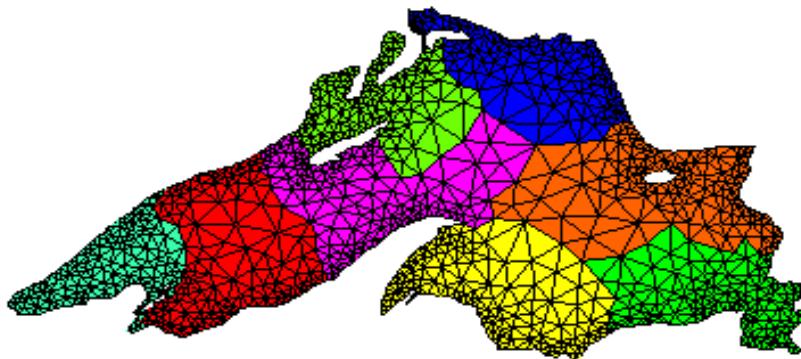


Source : Reference [4],[6]

Types of Parallelism : Task Parallelism

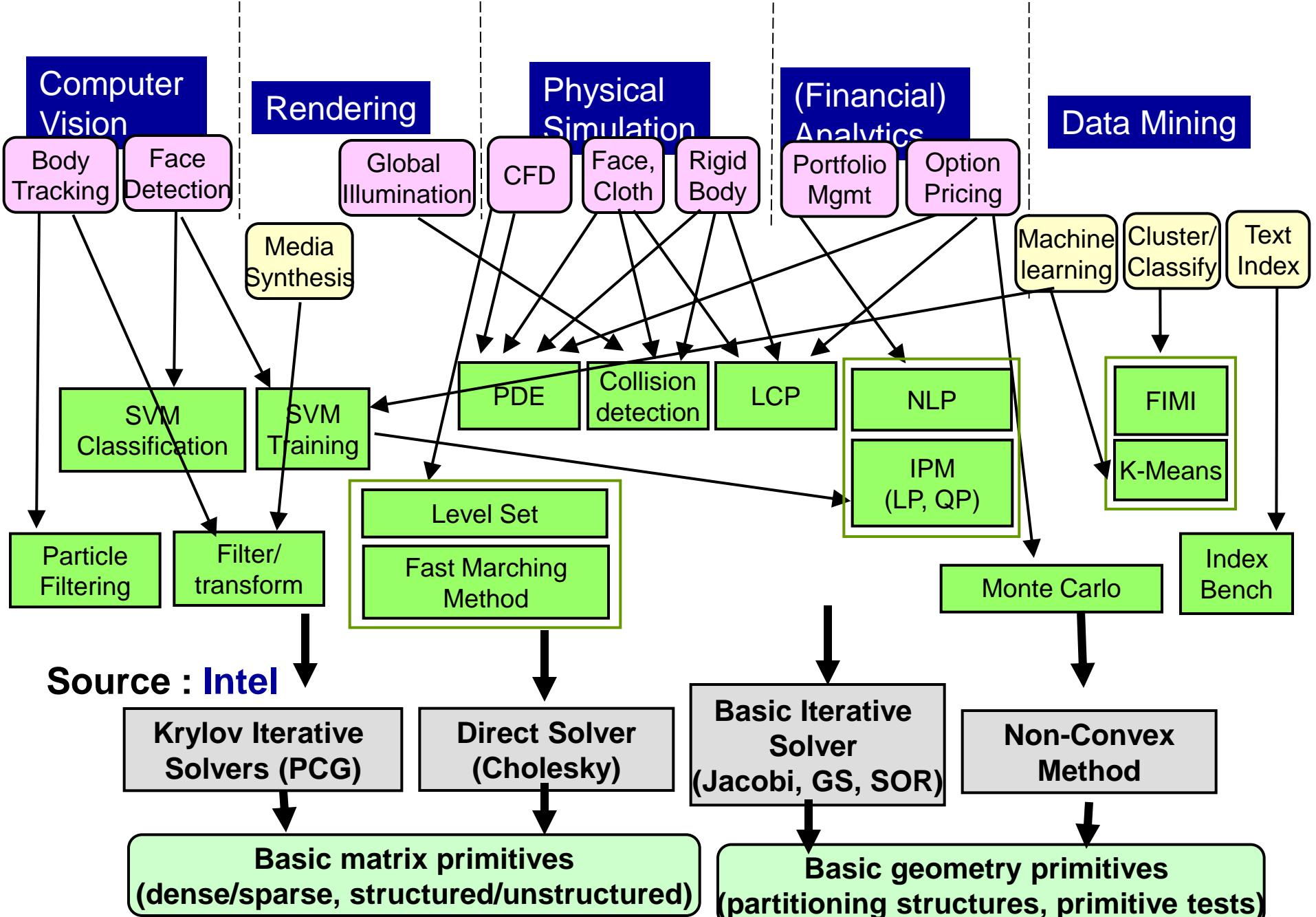
(Contd...)

- ❖ Parallel Unstructured Adaptive Mesh/Mesh Repartitioning methods



Finite Element Unstructured Mesh for Lake Superior Region

- ❖ HPF / Automatic compiler techniques may not yield good performance for unstructured mesh computations.
- ❖ Choosing right algorithm and message passing is a right candidate for partition (decomposition) of unstructured mesh (graph) onto processors. Task parallelism is right to obtain concurrency.



Multi-Cores - Parallel Programming Difficulties

- ❖ Multicore architectures force us to rethink how we do synchronization
- ❖ Parallel programming has traditionally been considered using locks to synchronize concurrent access to shared data.
- ❖ Standard locking model won't work
- ❖ Lock-based synchronization, however, has known pitfalls: using locks for fine-grain synchronization and composing code that already uses locks are both difficult and prone to deadlock.
- ❖ Transactional model might
 - Software
 - Hardware
 - Programming Issues

Load Balancing Techniques

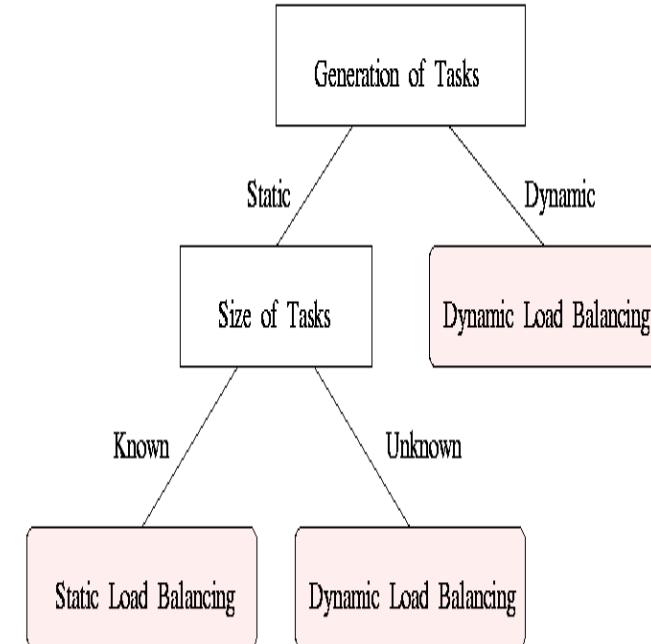
(Contd...)

❖ Static load-balancing

- Distribute the work among processors prior to the execution of the algorithm
- Matrix-Matrix Computation
- Easy to design and implement

❖ Dynamic load-balancing

- Distribute the work among processors during the execution of the algorithm
- Algorithms that require dynamic load-balancing are somewhat more complicated (Parallel Graph Partitioning and Adaptive Finite Element Computations)



Parallel Algorithmic Design

- ❖ Data parallelism; Task parallelism; Combination of Data and Task parallelism
- ❖ Decomposition Techniques
- ❖ Static and Load Balancing
 - Mapping for load balancing
 - Minimizing Interaction
 - Overheads in parallel algorithms design
- ❖ Data Sharing Overheads

Source : Reference :[1], [4]

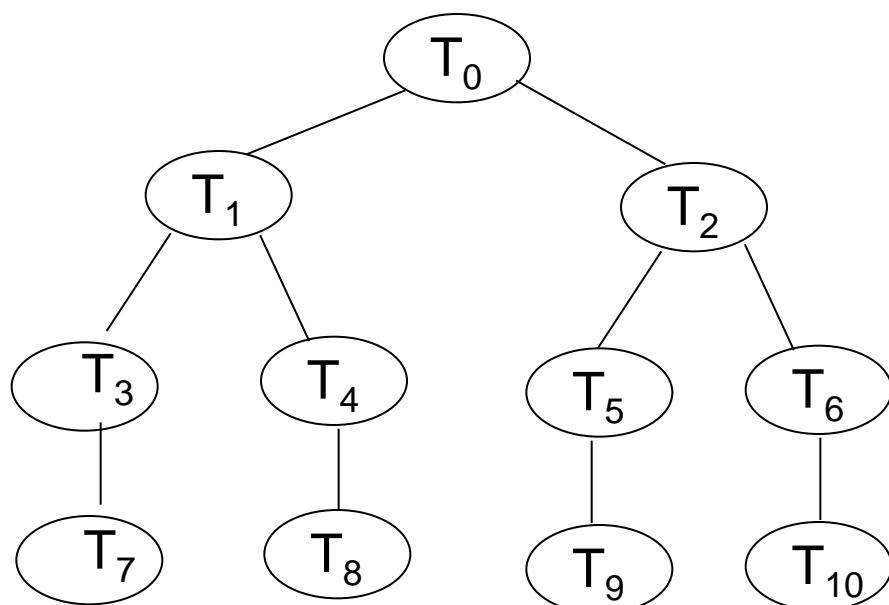
Parallel Algorithms and Design

Questions to be answered

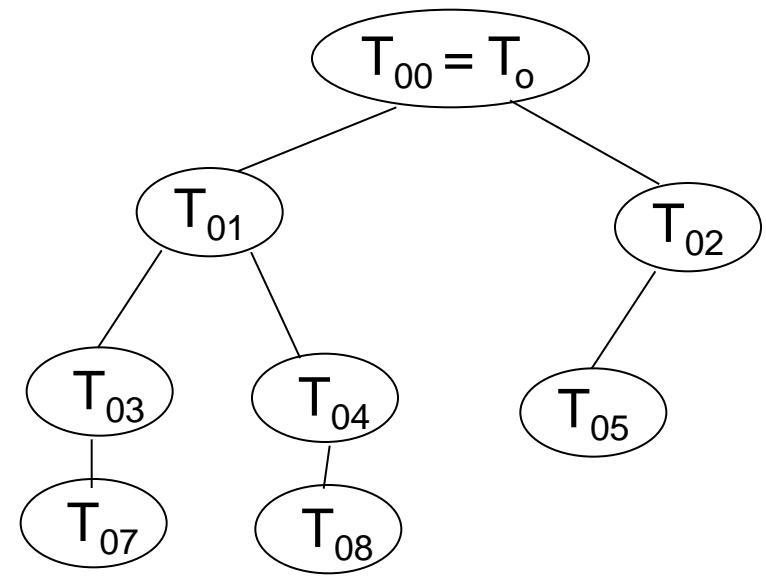
- ❖ How to partition the data?
 - ❖ Which data is going to be partitioned?
 - ❖ How many types of concurrency?
 - ❖ What are the key principles of designing parallel algorithms?
 - ❖ What are the overheads in the algorithm design?
 - ❖ How the mapping for balancing the load is done effectively?
- Decomposition techniques**
- ❖ Recursive decomposition
 - ❖ Data decomposition
 - ❖ Exploratory decomposition
 - ❖ Hybrid decomposition

Types of Parallelism : Task Parallelism

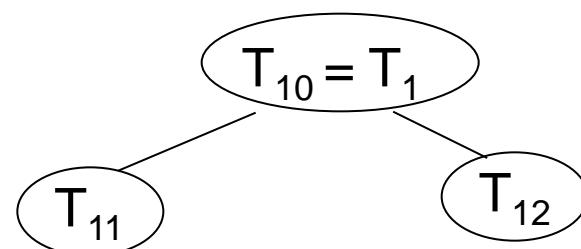
Task Graph



Sub Task Graph for T_o



Sub Task Graph for T_1



Programming Aspects –Example

Implementation of Streaming Media Player on Multi-Core

- ❖ One decomposition of work using Multi-threads
- ❖ It consists of
 - A thread Monitoring a network port for arriving data,
 - A decompressor thread for decompressing packets
 - Generating frames in a video sequence
 - A rendering thread that displays frame at programmed intervals

Programming Aspects -Example

Implementation of Streaming Media Player on Multi-Core

- ❖ The thread must communicate via shared buffers –
 - an **in-buffer** between the network and **decompressor**,
 - an **out-buffer** between the **decompressor** and **renderer**
- ❖ It consists of
 - Listen to portGather data from the network
 - Thread generates frames with random bytes (Random string of specific bytes)
 - Render threads pick-up frames & from the out-buffer and calls the display function
 - Implement using the Thread Condition Variables

Refer HeMPA-2011 web-page POSIX Threads

PetSc Library : Scientific Computations

Portable Extensible Toolkit for Scientific computations

- ❖ PETSc, pronounced PET-see (the S is silent), is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It supports MPI, [shared memory pthreads](#), and [NVIDIA GPUs](#), as well as hybrid MPI-shared memory pthreads or MPI-GPU parallelism.
- ❖ It consists of
 - Finite Element Solver : Unstructured Adaptive Finite Element Lib.
 - Finite volume Solver
 - General purpose CFD Solver
 - C++ Finite element Library

Refer HeGaPa-2012 web-page CUDA /OpenCL

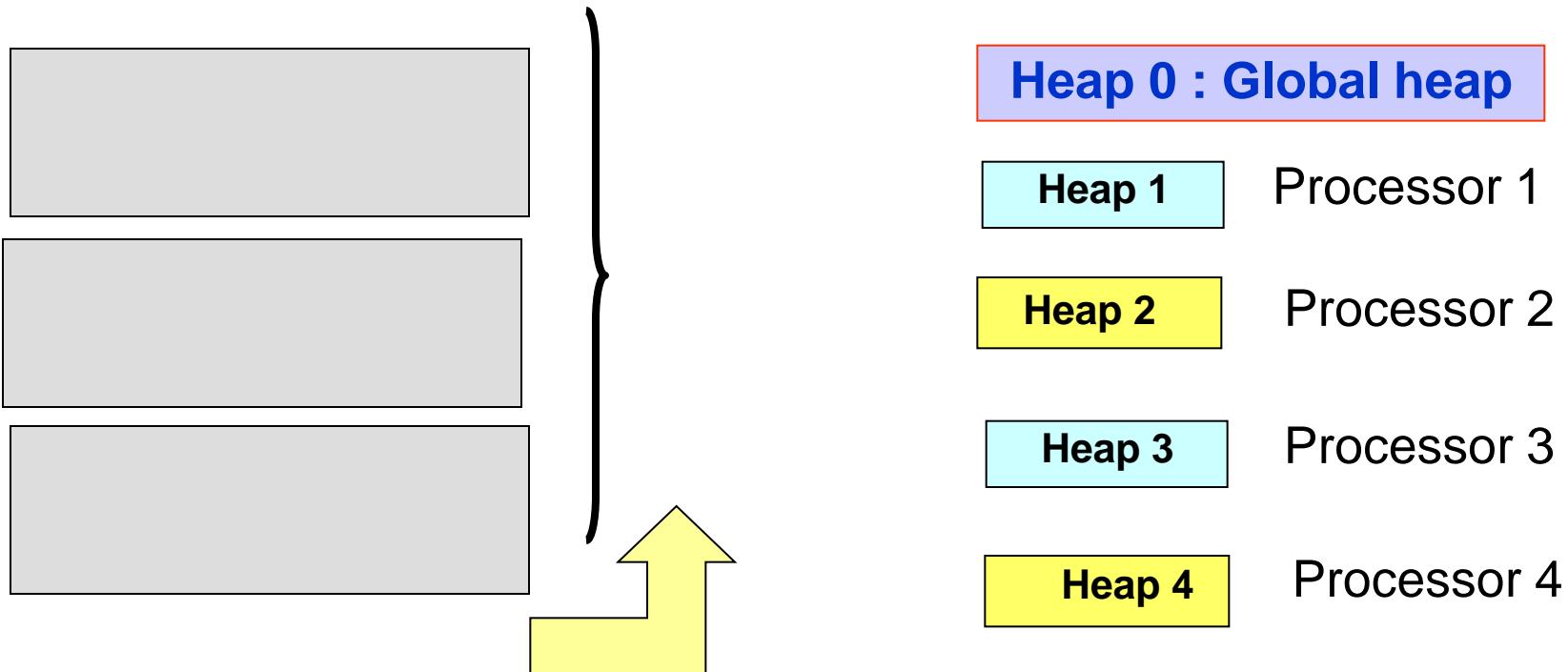
Part-III :

**An Overview of Multi-Core Processors
Memory Allocators**

An Overview of Memory Allocator for Multithreaded Application

- ❖ **Memory Allocation is often a bottleneck that severely limits program scalability on multiprocessor systems**
 - Existing Serial memory allocations do not scale well for multithreaded applications.
 - Concurrent memory allocators do not provide one or more following features....
 - Speed (fast malloc & free)
 - Scalability
 - False Sharing avoidance (Cache line)
 - Low fragmentation (Poor Data Locality, Paging)
 - Still some execution block is utilized
 - Blowup

Hoard : A Memory allocator



Superblocks

Each superblock has some blocks
(Empty/Partially filled /Fully Filled)

**thread ‘k’ maps to
heap ‘k’**

Source : <http://www.hoard.org/>

Hoard : A Memory allocator

❖ Example :

- Threads in **Producer–consumer** relationship
 - Blow-up mechanism exists
 - Memory Consumption grows linearly
- **Producer** thread repeatedly allocates a block of memory and it gives it to a **consumer** thread which frees it.
- If the memory freed by the **consumer** is unavailable to the **producer**, the program consumes more and more memory as it runs...
- Memory Consumption grows without bound while the memory required....

For more details , refer HeGaPa-2012web-page

Hoard : A Memory allocator

- ❖ Allocation and freeing in Hoard Memory Allocator
- ❖ Hoard maintains usage statistics for each heap
 - The amount of memory allocated by Hoard from the operating system held in heap i.
 - The amount of memory in use (“**Live**”) in heap “I”
- ❖ Hoard allocates memory from the system in chunks as well as **superblocks**
- ❖ Each superblock is an array of some number of blocks (objects) and contains a free list of its available blocks maintained in LIFO order to improve locality.
 - All the superblocks are of same size (S), a multiple of system page size.

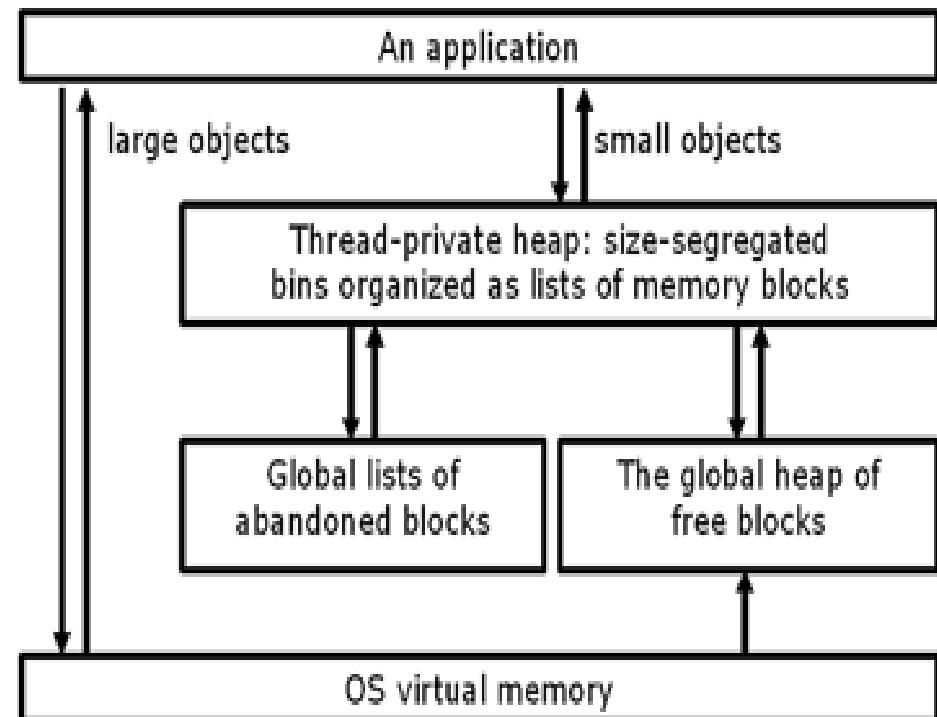
Intel Scalable Memory Allocator

- ❖ Standard malloc implementations still do not provide proper scalability for multi-threaded applications.
- ❖ The TBB allocator uses *thread-private heaps*. Such a design has proven to cut down on the amount of code that requires and reduce false sharing, thus providing better scalability.
- ❖ Each thread allocates its own copy of heap structures and accesses it via thread-specific data (TSD) using corresponding system APIs.

Source : <http://www.intel.com/technology/>

Intel Scalable Memory Allocator

- ❖ The allocator requests memory from the OS in 1MB chunks and divides each chunk into 16K-byte aligned blocks. These blocks are initially placed in the global heap of free blocks.



■ Source : <http://www.intel.com/technology>

Google Perftools

- ❖ Multi-threaded applications in C++ with templates. Includes TCMalloc, heap-checker, heap-profiler and cpu-profiler
- ❖ Works with STL
- ❖ Perf Tools is a collection of a high-performance multi-threaded malloc() implementation, and performance analysis tools. <http://code.google.com/p/google-perftools>
- ❖ PerfTools help one to identify spots in a program that are responsible for CPU consumption.
<http://minos.phy.bnl.gov/~bviren/minos/software/prof/PerfTools/doc/>

Top Linux Tool

- ❖ Monitoring resource usage, optimization our system, identifying memory leak. Top provide almost everything we need to monitor our system's resource usage within single shot.

top – b

top - 15:22:45 up 4:19, 5 users, load average: 0.00, 0.03, 0.00

Tasks: 60 total, 1 running, 59 sleeping, 0 stopped, 0 zombie

Cpu(s): 3.8%us, 2.9% sy, 0.0% ni, 89.6% id, 3.3% wa, 0.4% hi, 0.0% si

Mem: 515896k total, 495572k used, 20324k free, 13936k buffers

Swap: 909676k total, 4k used, 909672k free, 377608k cached

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND	1
root	16	0	1544	476	404	S	0.0	0.1	0.1	0:01.35	init	
2	root	34	19	0	0	0	S	0.0	0.0	0:00.02	ksoftirqd/0	
3	root	10	-5	0	0	0	S	0.0	0.0	0:00.11	events/0	

time top -b -n 1;

top -p 4360 4358

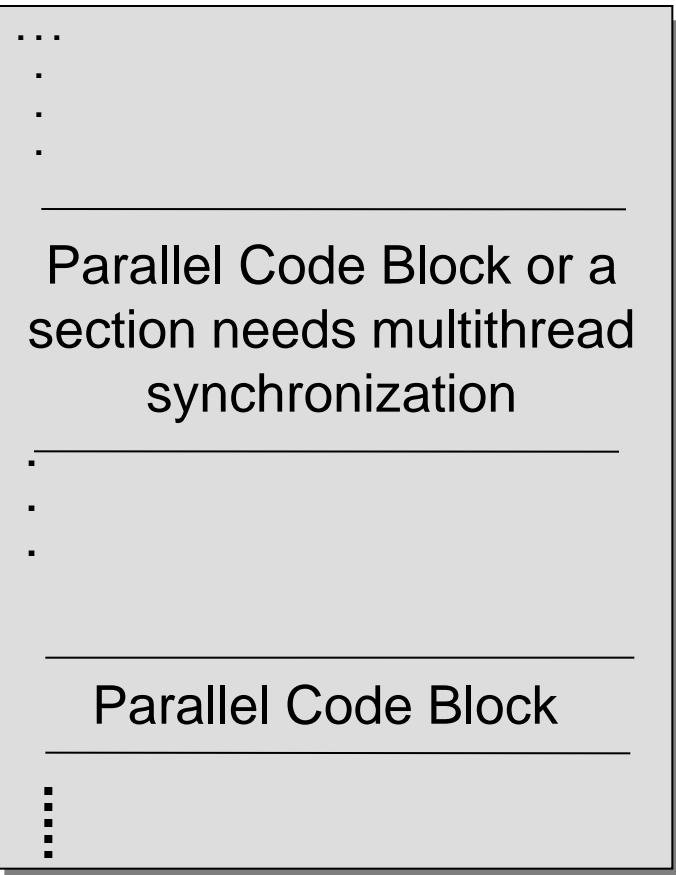
Part-IV :

An Overview of Multi-Core Processors

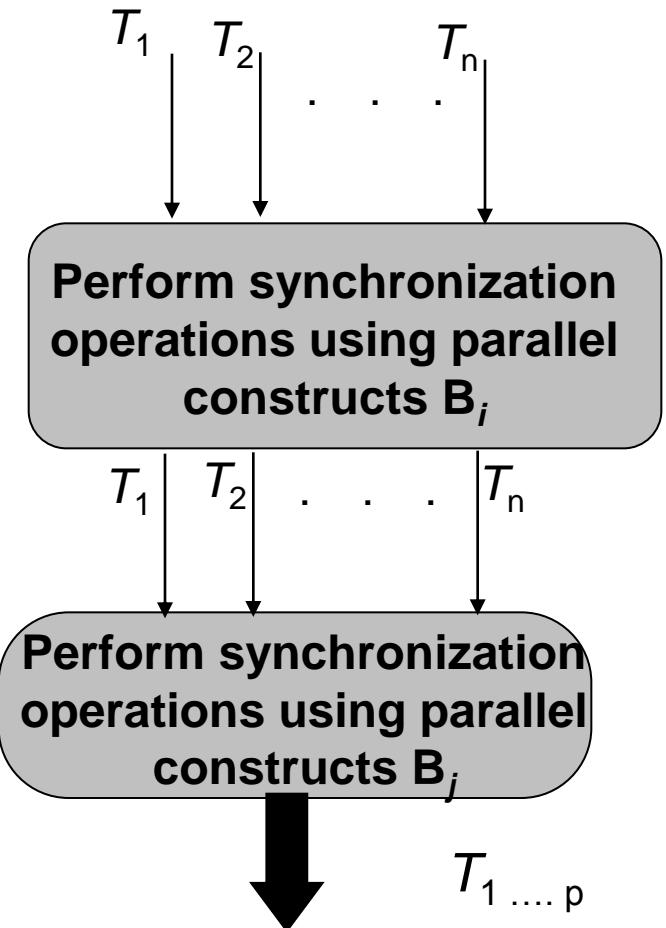
System Overview of Threading

Operational Flow of Threads for an Application

Implementation Source Code



Operational Flow of Threads



Source : Reference [4],[6], [7]

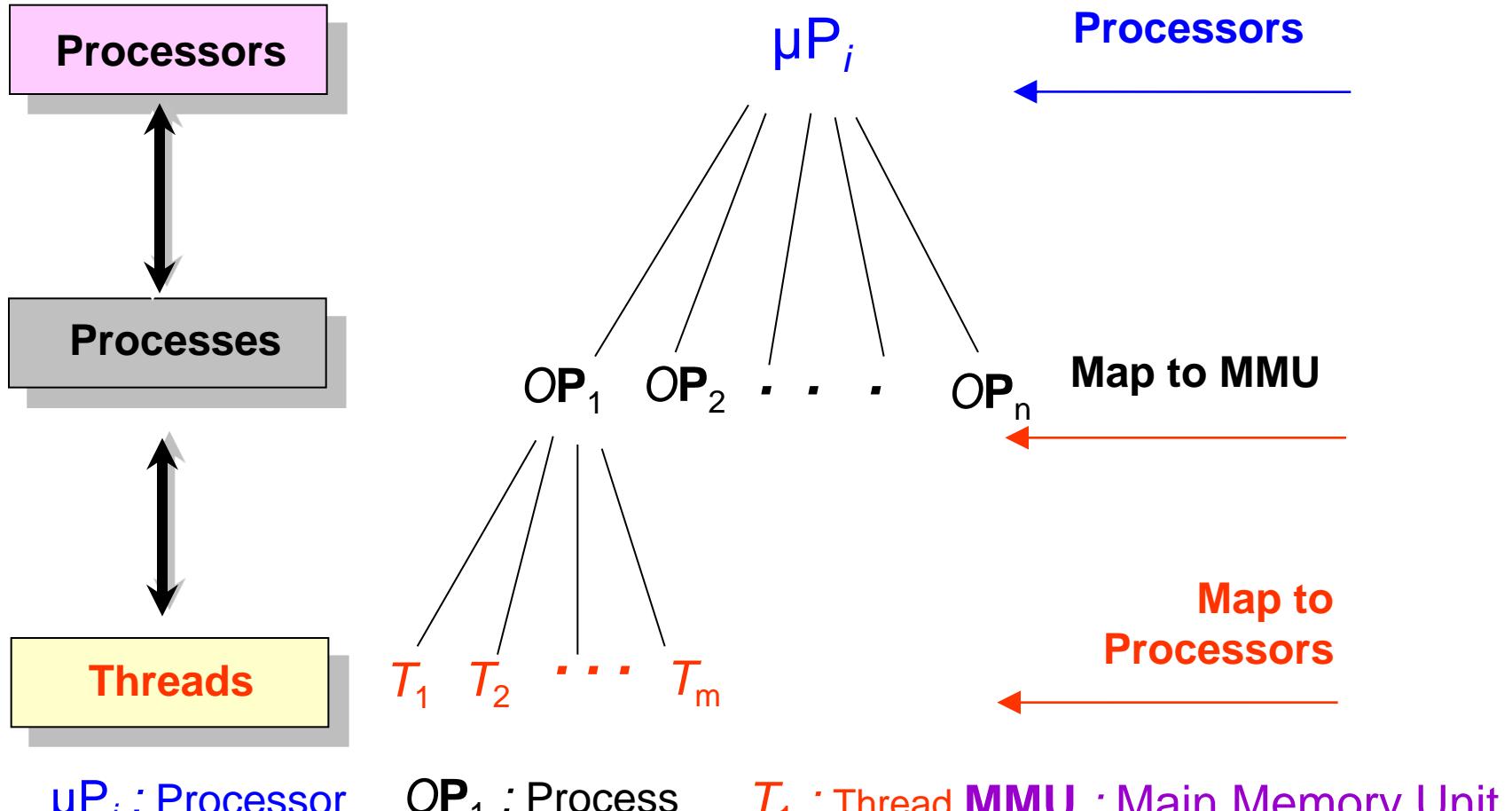
Defining Threads : What are threads ?

- ❖ A **thread** is defined as an independent stream of instructions that can be scheduled to run as such by the operating system.
- ❖ A **thread** is a discrete sequence of related instructions that is executed independently of other instructions sequences
- ❖ A **process** can have several threads, each with its own independent flow of control.
- ❖ **Threads share** the resources of the process that created it.

Implementation specific issues of Pthreads :

- Synchronization
- Sharing Process Resource
- Communication
- Scheduling

Relationship among Processors, Processes, & Threads



Source : Reference [4],[6], [7]

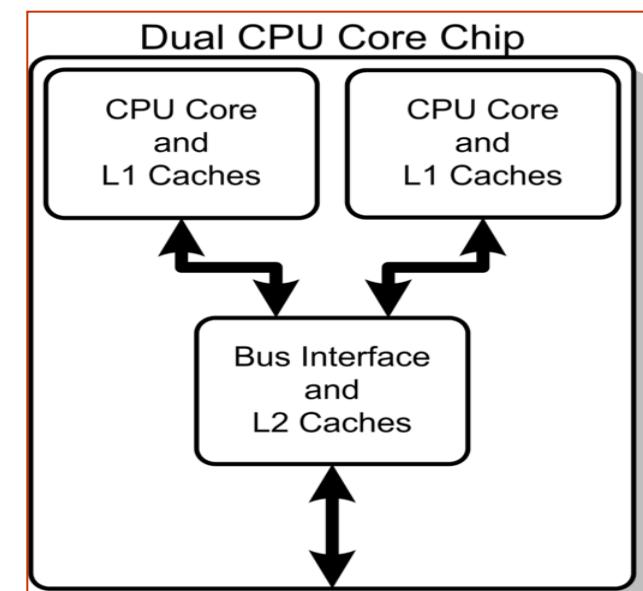
Threads Parallel Programming

(Contd...)

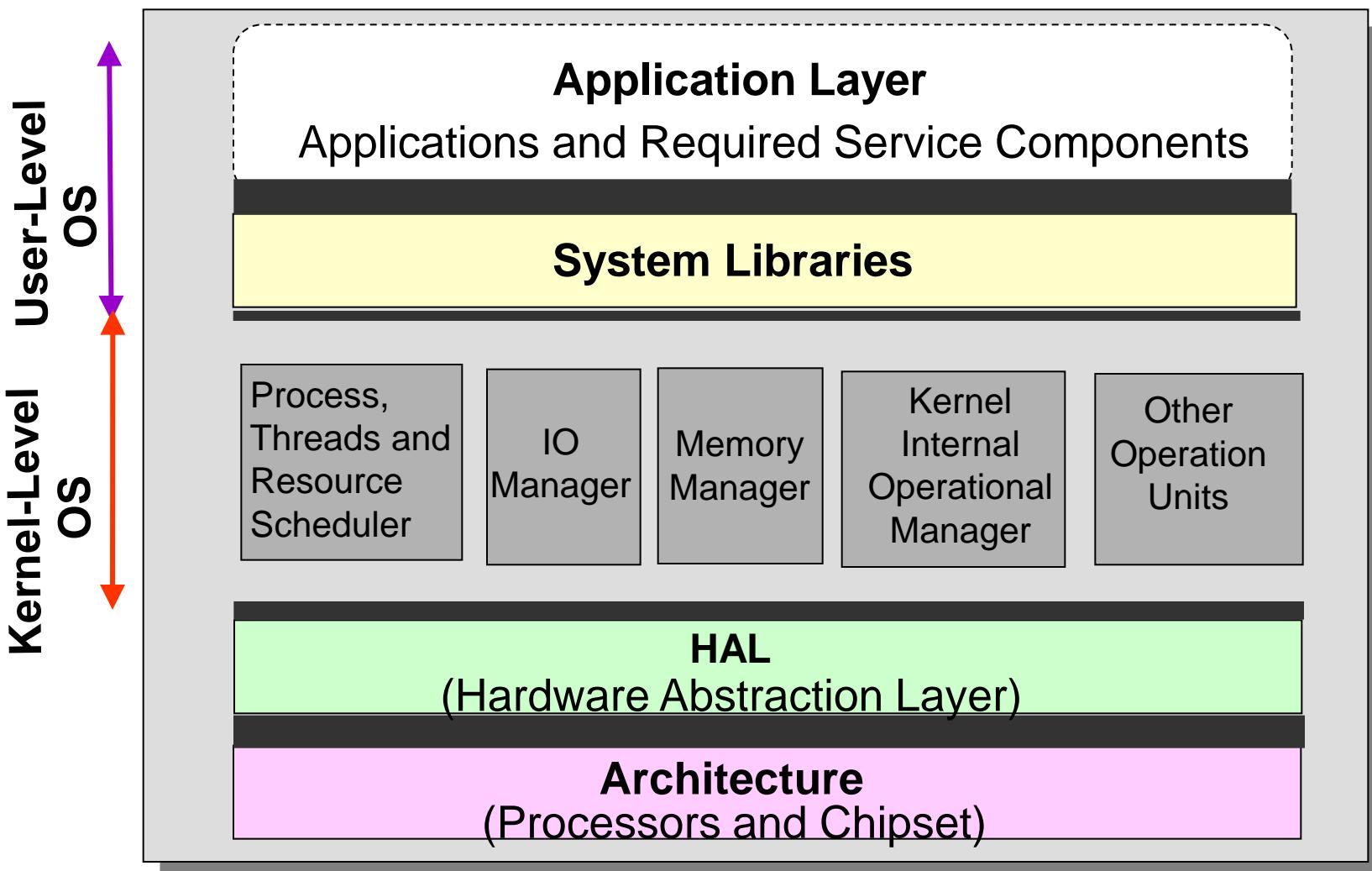
- ❖ A thread is a user-level concept that is invisible to the kernel
- ❖ Because threads are user-level object, thread operations such as switching from one thread to another are fast because they do not incur a context switch
 - Threads are not visible to the kernel
 - Threads are not scheduled for CPUs
 - Threads have un-describable blocking behavior
 - Threaded programs have more overhead than a non-threaded one.

System Overview of Threads

- ❖ Each Thread maintains its current machine state
- ❖ At the hardware level, a thread is an execution path that remains independent of other hardware thread execution paths.
- ❖ The operating system maps software threads to hardware execution resources
- ❖ Too much threading can hurt Performance



Different Layers of the Operating System /Threads



Source : Reference [4],[6], [7]

System Overview of Threads

- ❖ Three levels of threading is commonly used
- ❖ Each program thread frequently involves all three levels

User-Level Threads

Used by executable application and handled by user-level OS

Kernel-Level Threads

Used by operating system kernel and handled by kernel-level OS

Hardware Threads

Used by each Processor

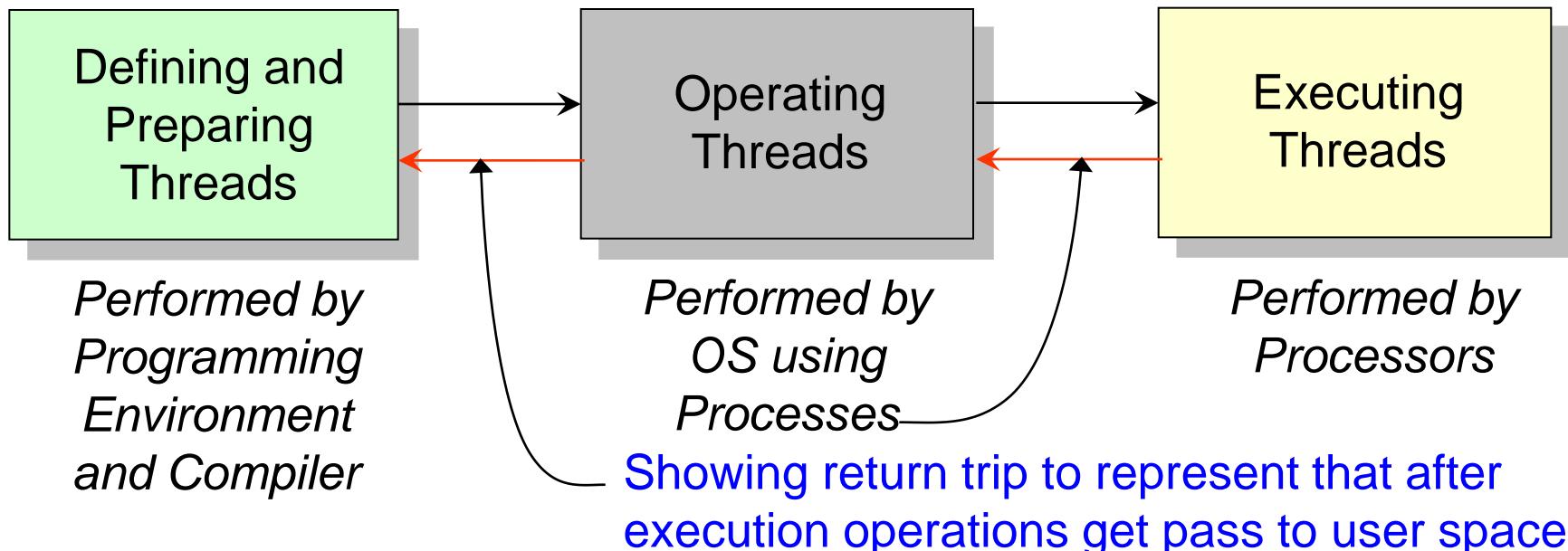
Computation Model of Threading

Source : Reference [4],[6], [7]

System View of Threads

Threads Above the Operating System

- ❖ Understand the problems - Face using the threads – Runtime Environment



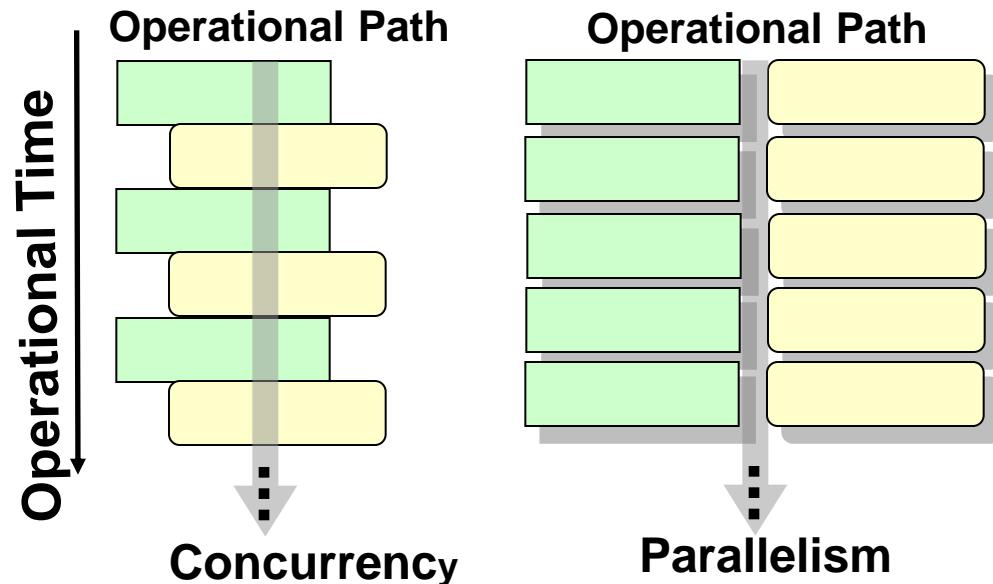
Flow of Threads in an Execution Environment

Source : Reference [4],[6], [7]

System Overview of Threads

Threads Inside the Hardware

- ❖ Concurrency versus Parallelism
- ❖ Thread stack allocation
- ❖ Sharing hardware resources among executing threads – Concurrency
- ❖ Hyper-threading Technology
- ❖ Chip Multi-threading (CMT)
- ❖ Simultaneous Multi-threading (SMT)



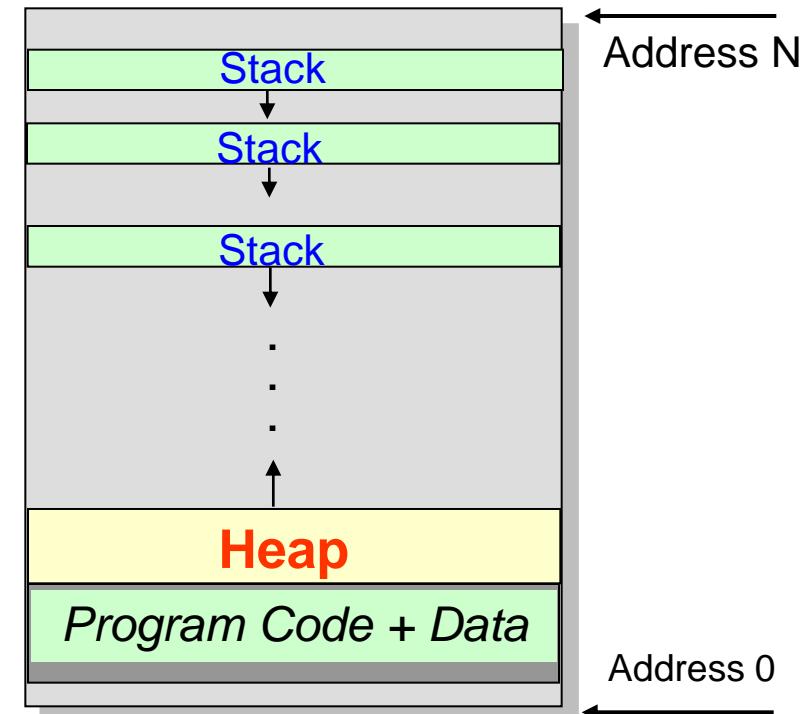
Source : Reference [4],[6], [7]

System Overview of Threads

Stack Layout in a Multi-threaded Process

After thread creation, each thread needs to have its own stack space.

- ❖ Thread stack size
- ❖ Thread stack allocation *Region for Thread 1* {
 Region for Thread 2 {
- ❖ Know Operating System Limitations
- ❖ Default Stack Size may vary from system to system
- ❖ Performance may vary from system to system
- ❖ Bypass the default stack manager and manage stacks on your own as per application demands

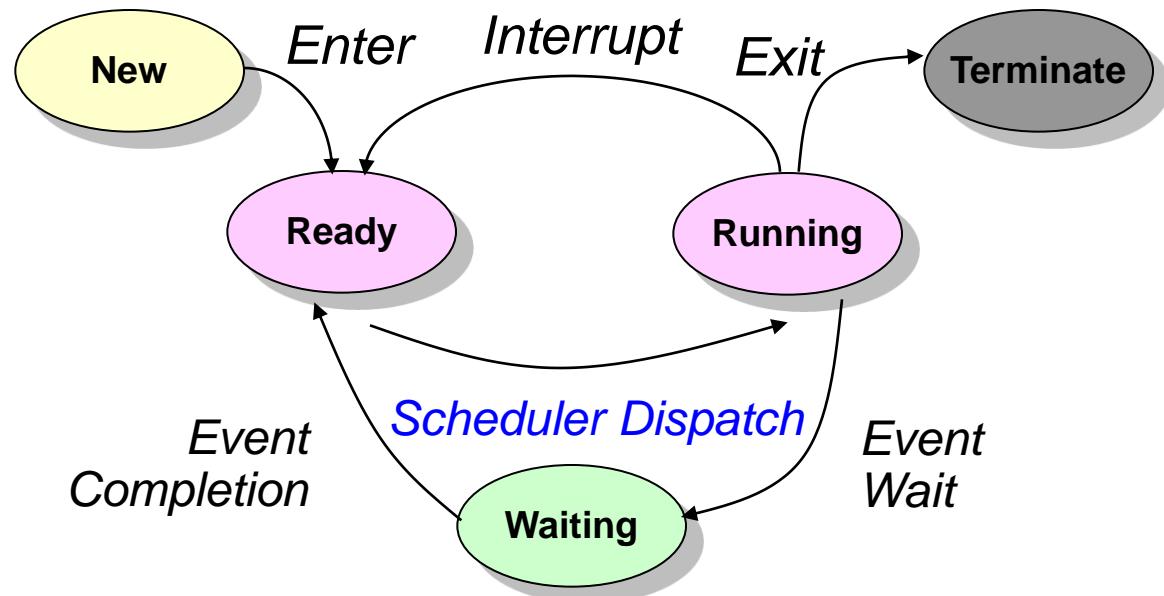


Source : Reference [4],[6], [7]

System Overview of Threads

State Diagram for a Thread

- ❖ Threads Creation
- ❖ Four Stages of Thread Life Cycle
 - Ready,
 - Running,
 - Waiting (blocked),
 - Termination
- ❖ Finer Stages in debugging or analyzing a threaded application



Source : Reference [4],[6], [7]

Part-V :

An Overview of Multi-Core Processors

POSIX-Threads

Commonly Encountered Questions While Threading Application ?

- ❖ What should the expected speedup be?
- ❖ Will the performance meet expectations?
- ❖ Will it scale if the more number of processor added?
- ❖ Which threading model is it?

Commonly Encountered Questions While Threading Application ?

Analysis

- ❖ Where to thread ?
thread the more time consuming section of code like loops
- ❖ How long would it take to thread?
Very minimum time just need to use some directives/lib. routine
- ❖ How much re-design / efforts is required?
Very less
- ❖ Is it worth threading the selected region ?
Appears to have minimal dependencies
Consuming over 90% of run time

What are Pthreads?

- ❖ POSIX threads or Pthreads is a portable threading library which provides consistent programming interface across multiple operating systems.
- ❖ It is set of C language programming types and procedure calls, implemented with pthread.h file and a thread library.
- ❖ Set of threading interfaces developed by IEEE committee in charge of specifying a portable OS Interface.
- ❖ Library that has standardized functions for using threads across different platform.

Pthread APIs

- ❖ Pthread APIs can be grouped into three major classes:
 - Thread Management - thread creation, joining, setting attributes etc.
`pthread_create(thread,attr,start_routine,arg)`
 - Thread Synchronization - functions that deal with Mutex
`pthread_mutex_init(mutex , attr)`
`pthread_mutex_destroy(mutex)`
`pthread_mutex_lock(mutex)`
`pthread_mutex_trylock(mutex)`
`pthread_mutex_unlock(mutex)`
 - Condition Variables - functions that deal with condition variables
`pthread_cond_init(condition,attr)`
`pthread_cond_destroy(condition,attr)`

Pthread APIs

- ❖ All identifiers in the thread library begins with **pthread_**

Routine Prefix	Functional Group
pthread_	Threads themselves and misc subroutines
pthread_attr_	Thread Attribute objects
pthread_mutex_	Mutex related routines
pthread_cond_	Condition variable related

`pthread_detach(pthread_t thread_to_detach)`

`pthread_exit(void *value_ptr)`

`pthread_join(threadId , status)`

Pthread APIs

- ❖ **Semaphores** : A semaphore is a counter that can have any nonnegative value. Threads wait on a semaphore.
- ❖ When the semaphore's value is 0, all threads are forced to wait. When the value is non-zero, a waiting thread is released to work.
- ❖ Pthreads does not implement semaphores, they are part of a different POSIX specification.
- ❖ Semaphores are used to conjunction with Pthreads' thread-management functionality

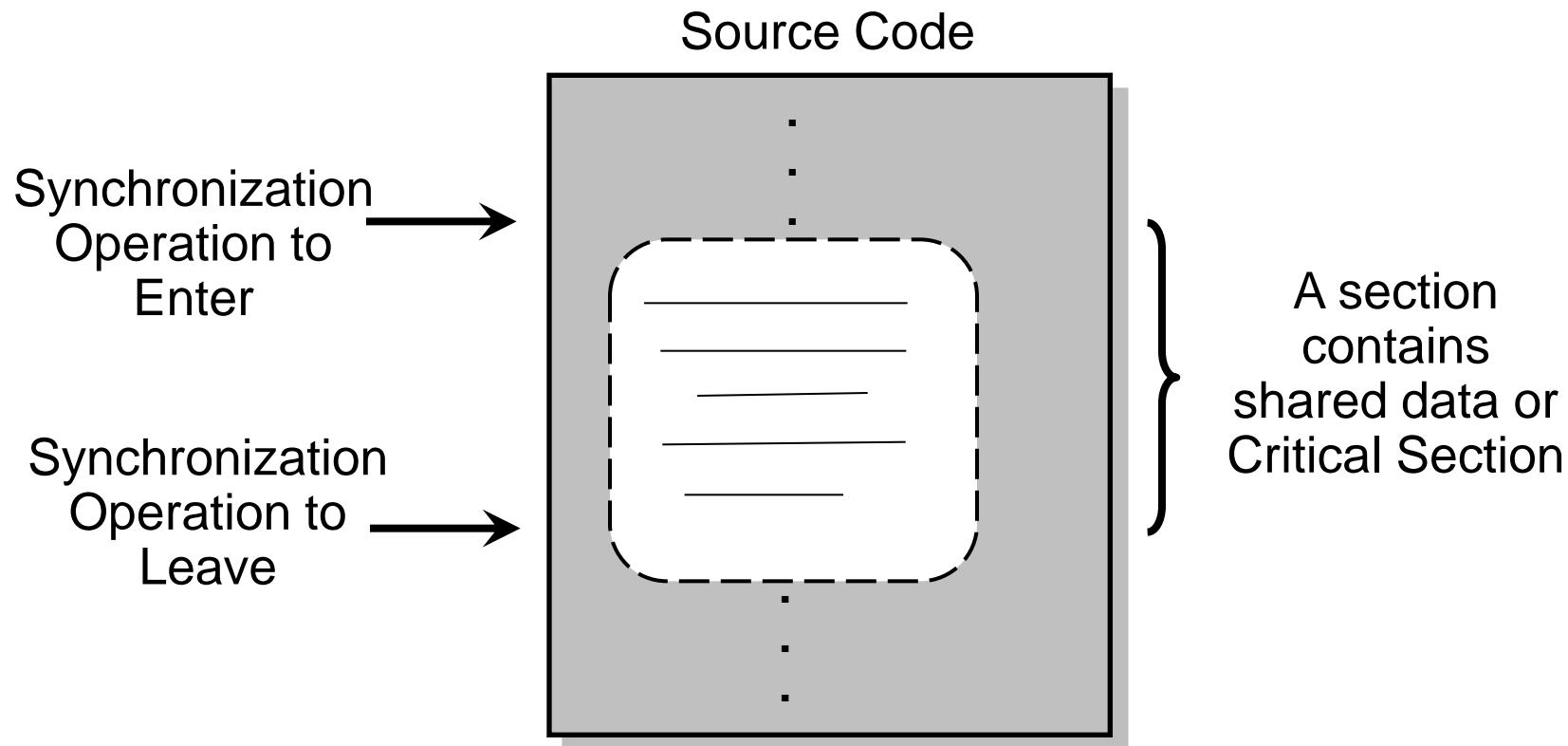
Usage : Include <semaphore.h>

- `sem_init(*, *, ...*)`
- `sem_post(*, *, ...*)`
- `sem_wait(*, *, ...*)`

Pthread APIs – Key Points

- ❖ Threads can communicate with one another using events
- ❖ A care is needed to terminate the Thread while using the C runtime library.
- ❖ Thread synchronizations can be accomplished through the use of Mutexes, Semaphores, Critical Sections, and Interlocked functions
- ❖ Windows support multiple thread-priority levels
- ❖ Processor affinity is a mechanism that allows the programmer to specify which processor a thread should try to run on. – OS play an important role on Multi Core processor
- ❖ POSIX Threads (Pthreads) is a portable threading APIs that is supported on a number of platforms.

Generic Representation of Synchronization Block inside Source Code



Source : Reference [4],[6], [7]

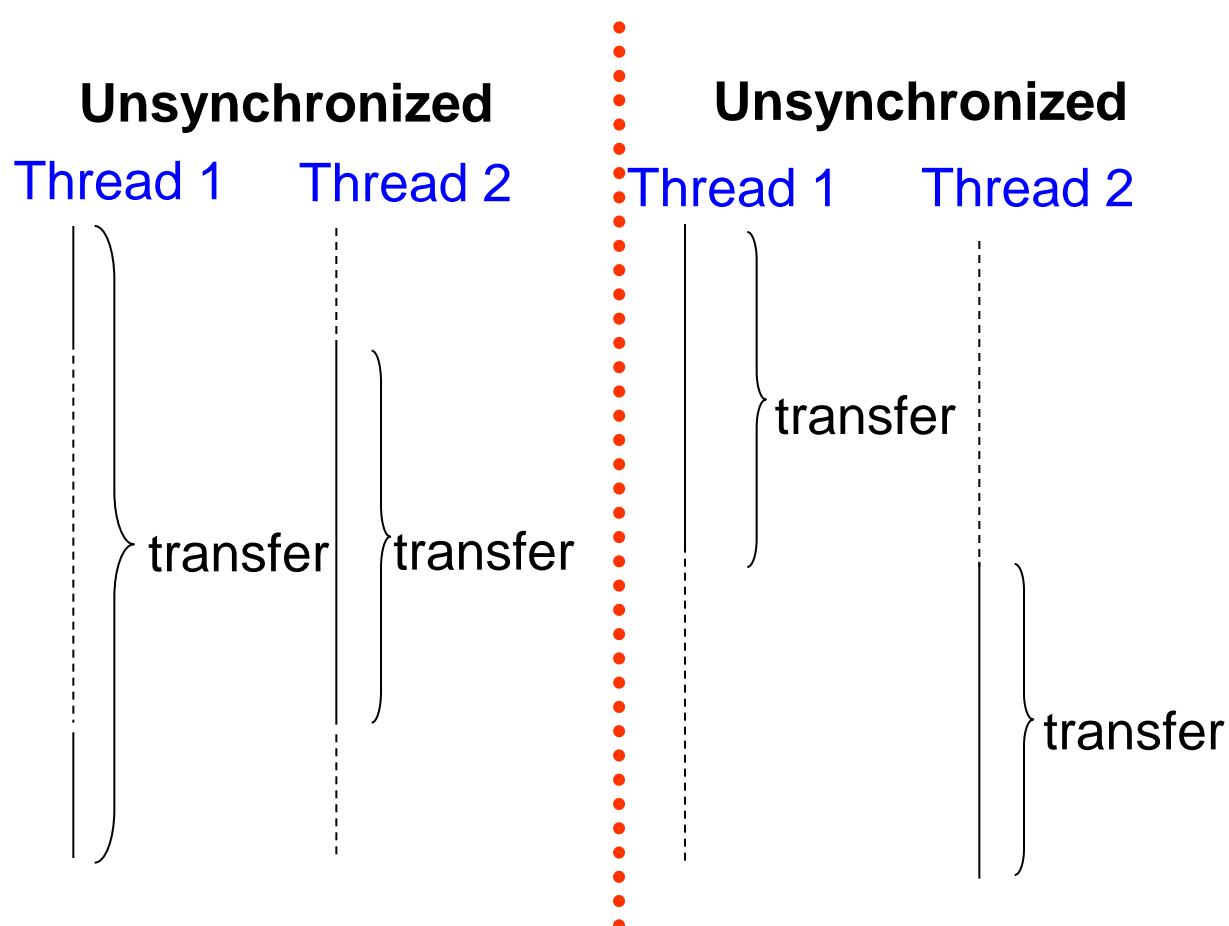
- ❖ Two types of Synchronization operations are widely used : Mutual exclusion and Condition synchronization

Comparison of Unsynchro nized / Synchronized threads

- ❖ Too little / too much synchronization

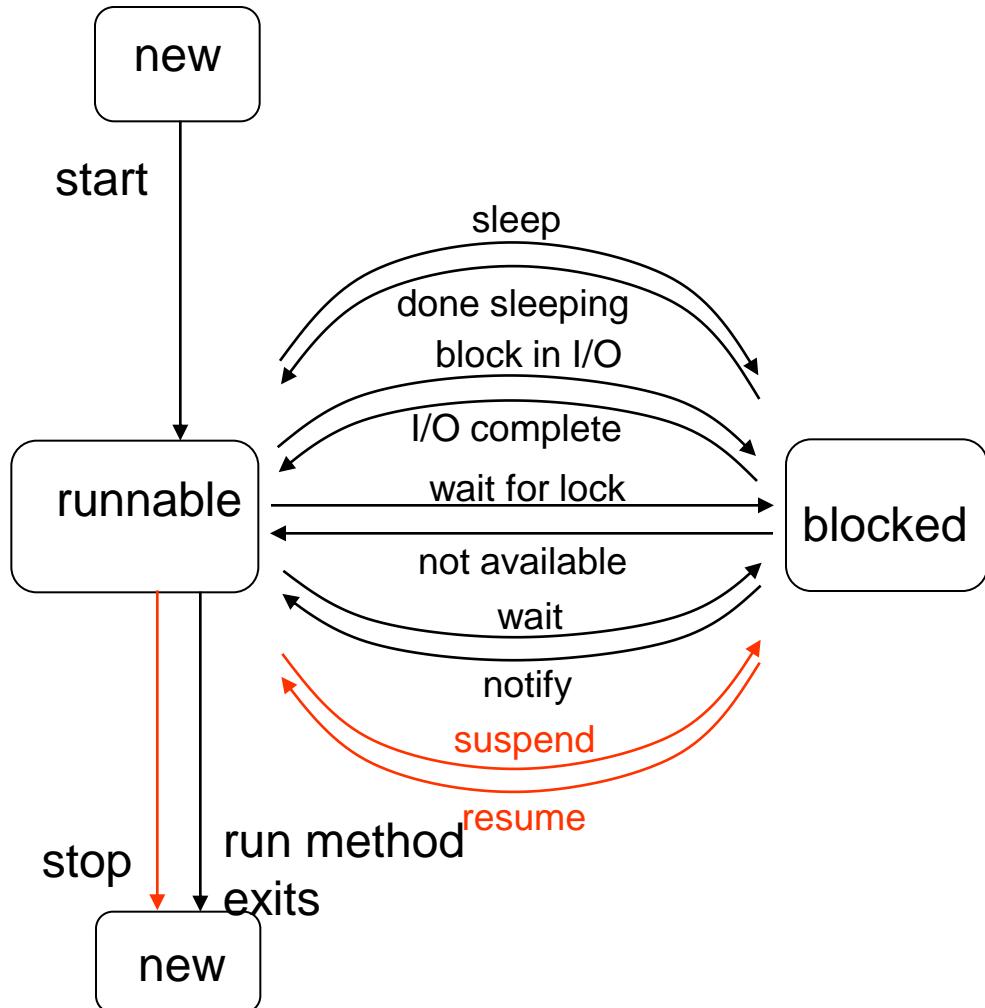
- In-Correct Results

- Performance – Slow done the results

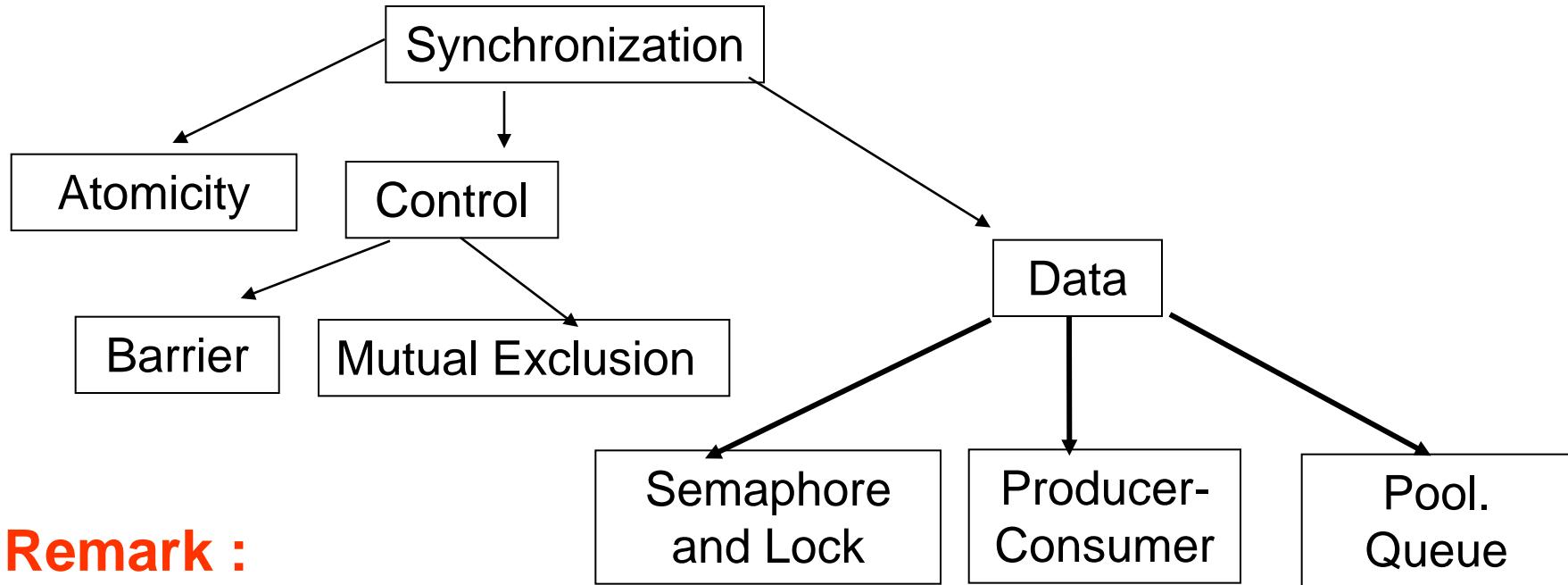


Pthreads:Synchronization & Thread States

- ❖ I/O Requests
- ❖ Read-Write Locks
- ❖ Available CPU
- ❖ Release Locks
- ❖ Critical Sections



Pthreads : Various types of synchronization



Remark :

- ❖ Use of Scheduling techniques as means of Synchronization is not encouraged. – Thread Scheduling Policy ,High Priority & Low Priority Threads
- ❖ Atomic operations are a fast and relatively easy alternative to mutexes. They do not suffer from the deadlock.

Synchronizing Primitives in Pthreads

❖ Common Synchronization Mechanism

- *Read/Write exclusion*
- *Thread safe data structures*
- *Condition variable functions*
- *Semaphores*

❖ Mutex Variables

- To protect a shared resource from a race condition, we use a type of synchronization called *mutex* exclusion, or *mutex lock* for short
- Critical section : Provide access to the code paths or routines that access data -
- How large does a critical section have to be to require protection through a *mutex* ?
- *Pthread* library operations such as *mutex locks* and *unlocks* work properly regardless of the platform you are using and the number of CPUs in the system.

Synchronization Primitives in Pthreads

- ❖ Mutual Exclusion for Shared Variables
- ❖ Implementation of critical sections and atomic operations using **mutex-locks** (mutual exclusion locks)
- ❖ Mutex locks have two states (locked and unlocked) Use functions **pthread_mutex_lock** & **pthread_mutex_unlock** function)
- ❖ A function to initialize a mutex-lock to its unlocked state - **pthread_mutex_init** function)

Synchronization Primitives in Pthreads

- ❖ Controlling Thread Attributes and Synchronization
 - Attribute Objects for Threads
 - Attribute Objects for Mutexes
- ❖ Thread Cancellation
 - Clean-up functions are invoked for reclaiming the thread data structures
- ❖ Composite synchronization Primitives
 - Read-Write Locks (Data Structure is read frequently but written infrequently.)
 - Issues of Multiple reads /Serial writes
 - Issues of Read Locks; read-write locks etc...

Synchronization Primitives in Pthreads

❖ Barriers

- A barrier call is used to hold a thread until all other threads participating in the barrier have reached the barrier
- Barriers can be implemented using a **counter**, a **mutex**, and a **condition** variable.
- A single integer is used to keep track of the number of threads that have reached the Barrier

❖ Remark :

- Barrier implementation using mutexes may suffer from the overhead of busy-wait.

Synchronization Primitives in Pthreads

- ❖ Mutual Exclusion for Shared Variables
 - Thread APIs provide support for implementing critical sections and atomic operations using mutex-locks (mutual exclusion locks)
- ❖ Condition Variables for Synchronization
 - When thread performs a condition wait, it takes itself off the runnable list – Does not use any CPU cycle

Remark :

- Mutex Lock consumes CPU cycles as it polls for the lock
- Condition wait consumes CPU cycles when it is woken up

Composite Synchronization Constructs

- ❖ **Barrier** : A barrier call is used to hold a thread until all other threads participating in the barrier have reached the barrier.
 - Barrier can be implemented using a **counter**, a **mutex**, and a **condition** variable.
 - Overheads will vary for large number of **threads**.
 - Performance of programs depends upon the application characteristics such as the number of threads & the number of **condition** variable **mutexes** pairs for implementation of a barrier for **n threads**.

Composite Synchronization Constructs

- ❖ The higher level synchronization constructs can be built using basic constructs.
- ❖ **Read-Write Constructs**
 - A data structure is read frequently but written infrequently.
 - Multiple reads can proceed without any coherence problems. Write must be serialized.

❖ A structure can be defined as **read-write** lock

Example 1 : Using read-write locks for computing the minimum of a list of integers

Example 2 : Using read-write locks for implementing hash tables.

Source : Reference : [4]

Composite Synchronization Constructs

❖ Read-Write Struct

```
➤ typedef struct {
    int readers;
    int writer;
    pthread_conf_t readers_proceed;
    pthread_cond_t writer_proceed;
    int pending_writers;
    pthread_muex_t read_write_lock;
} mylib_rwlock_t;
```

Source : Reference : [4]

For more details on programs refer HeGaPa-2012 web-page

Composite Synchronization Constructs

❖ Read-Write Constructs

- Offer advantages over normal locks
- For frequent reads /Writes, overhead is less
- Using normal **mutexes** for writes is advantages when there are a significant number of **read** operations

- ❖ For performance of database applications (hash tables) on Multi Cores, the **mutex** lock version of the program hashes key into the table requires suitable modification.

Source : Reference : [4]

Pthreads:Performance issues-Synchronization Overhead

- ❖ Performance depends on input workload :
 - Increasing clients and contention
 - Number of clients vs Ratio of Time to Completion
 - Performance depends on a good locking strategy
 - No locks at all;One lock for the entire data base;
One lock for each account in the data base
 - Performance depends on the type of work threads do
 - Percentage of Thread I/O vs CPU and Ratio of
Time to Completion

Pthreads:Performance issues-Synchronization Overhead

❖ How do your threads spend their time ?

- Profiling a program is a good step toward identifying its performance bottlenecks (CPU Utilization, waiting for locks and I/O completion)
- Do the threads spend most of their time blocked, waiting for their threads to release locks ?
- Are they *Runnable* for most of their time but not actually running because other threads are monopolizing the available CPUs ?
- Are they spending most of their time waiting on the completion of I/O requests ?

Spawning Threads

- ❖ Initialize Attributes (pthread_attr_init)
 - Default attributes OK
- ❖ Put thread in system-wide scheduling contention
 - `pthread_attr_setscope(&attrs, PTHREAD_SCOPE_SYSTEM);`
- ❖ Spawn thread (pthread_create)
 - Creates a thread identifier
 - Need attribute structure for thread
 - Needs a function where thread starts
 - One 32-bit parameter can be passed (`void *`)

Source : Reference : [4],[6], [29]

Thread Spawning Issues

- ❖ How does a thread know which thread it is? Does it matter?
 - Yes, it matters if threads are to work together
 - Could pass some identifier in through parameter
 - Could contend for a shared counter in a critical section
 - `pthread_self()` returns the thread ID, but doesn't help.
- ❖ How big is a thread's stack?
 - By default, not very big. (What are the ramifications?)
 - `pthread_attr_setstacksize()` changes stack size

Join Issues

- ❖ Main thread must join with child threads (pthread_join)
 - Why?
 - Ans: So it knows when they are done.
- ❖ pthread_join can pass back a 32-bit value
 - Can be used as a pointer to pass back a result
 - What kind of variable can be passed back that way? Local?
Static? Global? Heap?

Thread Pitfalls

❖ Shared data

- 2 threads perform
 $A = A + 1$

Thread 1:

- 1) Load A into R1
- 2) Add 1 to R1
- 3) Store R1 to A

Thread 1:

- 1) Load A into R1
- 2) Add 1 to R1
- 3) Store R1 to A

- Mutual exclusion preserves correctness
 - Locks/mutexes
 - Semaphores
 - Monitors
 - Java “synchronized”

❖ False sharing

- Non-shared data packed into same cache line

```
int thread1data;  
int thread1data;
```

- Cache line ping-pongs between CPUs when threads access their data

❖ Locks for heap access

- malloc() is expensive because of mutual exclusion
- Use private

What is a Data Race?

A data-race occurs under the following conditions:

- ❖ Two or more threads concurrently accessing the same memory location.
- ❖ At least one of the threads is accessing the memory
- ❖ Location for writing
- ❖ The threads are not using any exclusive locks to control their accesses to that memory.

For more details refer HeGaPa-2012 web-page

Data Races, Deadlocks & Live Locks

- ❖ Un-synchronized access to shared memory can introduce Race conditions
 - Results depends on relative timings of two or more threads
 - Solaris, Posix Multi-threaded Programming
- ❖ Example :
 - Two threads trying to add to a shared variable x, which have an initial value of 0.
 - Depending on upon the relative speeds of the threads, the final value of x can be 1, 2, or 3.
- Source : Reference [6]
- ❖ Parallel Programming would be lot of easier
- ❖ Multi-threaded Compiler & Tools may give clue to programmer

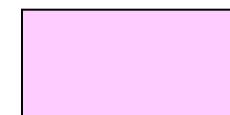
Data Races, Deadlocks & Live Locks

- ❖ The interactions of Memory, Cache, and Pipeline should be examined carefully.
 - Thread Private
 - Thread shared read only
 - Exclusive Access
 - Read and Write by Unsynchronized threads

Data Races, Deadlocks & Live Locks

❖ Deadlock conditions

1. A thread is allowed to whole one resource while requesting another
2. No thread is willing to relinquish a resource that is has acquired
3. Access to each resource is exclusive

Original Code	Thread 1	Thread 2
	$T = x$	$u = x$ $x = u + 2$
Interleaving #1	$(x \text{ is } 0)$ $t = x$  $x = t + 1$ $(x \text{ is } 1)$	 $u = x$ $x = u + 2$ 

Deadlock is caused by Cycle of operations

Data Races, Deadlocks & Live Locks

❖ Deadlock Conditions

4. There is a cycle of threads trying to acquire resources, where each resource is held by one thread and requested by another

Deadlock Conditions can be avoided by breaking any one of the conditions

	Thread 1	Thread 2
Interleaving #2	$t = x$ $x = t + 1$ $(x \text{ is } 1)$	$u = x$ $x = u + 2$ $(x \text{ is } 2)$
Interleaving #3	$(x \text{ is } 0)$ $t = x$ $x = t + 1$ $(x \text{ is } 1)$	$u = x$ $x = u + 2$ $(x \text{ is } 3)$

Data Races, Deadlocks & Live Locks

- ❖ **Locks** : Locks are similar to semaphores except that a single thread handles lock at one instance. Two basic **atomic** operations get performed on a lock are **acquire()** & **release ()**.
- ❖ **Mutexes** : The **mutex** is the simplest lock an implementation can use.
- ❖ **Recursive Locks** : Recursive locks are locks that may be repeatedly acquired by the thread that currently owns the lock without causing the thread to deadlock.
- ❖ **Read-Write Locks** : Read-Write locks are also called shared-exclusive or multiple-read/single-write locks or non-mutual exclusion semaphores. Read-Write locks allow simultaneous read access to multiple threads but limit the write access to only one thread.
- ❖ **Spin Locks** : Spin locks are non-blocking locks owned by a thread. Spin locks are used mostly on multiprocessors .

Source : Reference [4],[6]

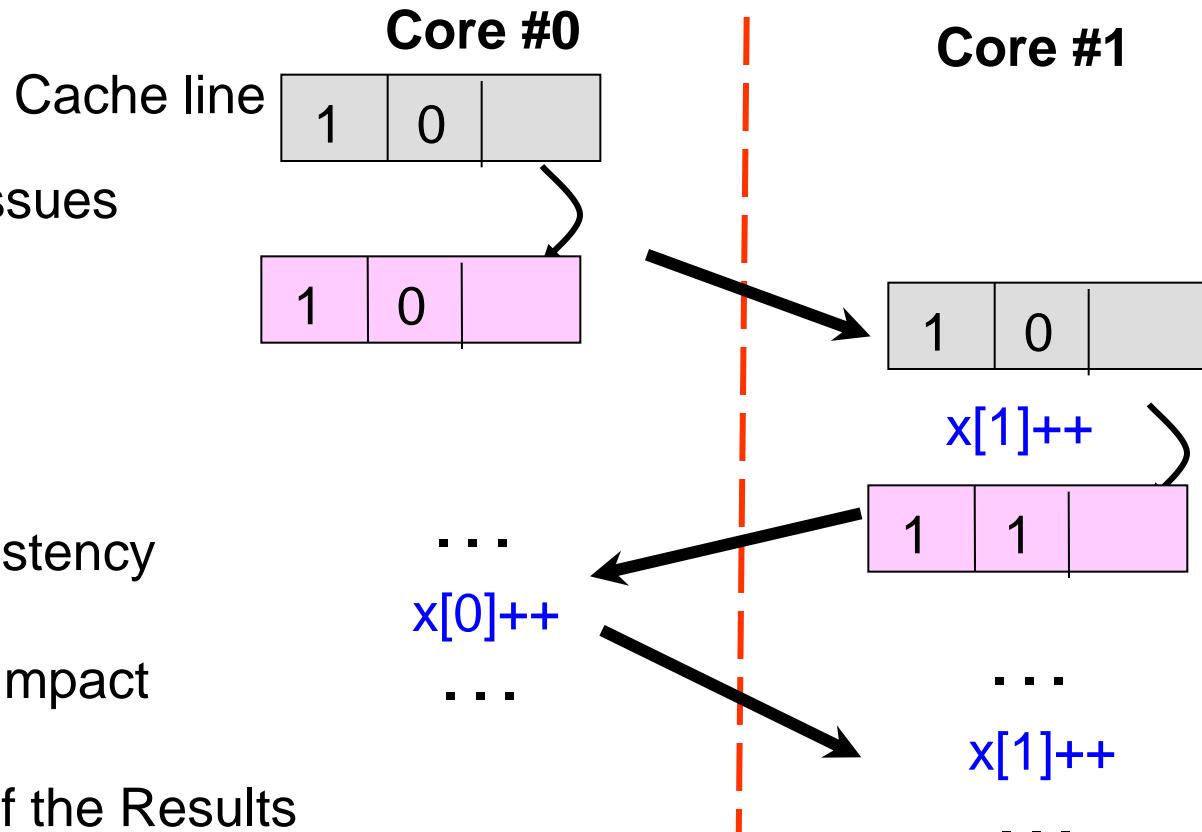
Cache Line Ping-Pong Caused by False Sharing

❖ Cache Related Issues

- Cache Line (Estimate the cache line size of the Multi core Systems
(Remark : Dual Core Processors share L1 Cache))
- False Sharing (The data can be pushed into different cache lines, thereby pushing reduce the false sharing overhead.)
- Performance Impact may vary from problem to problem. (Cache friendly programs such as Dense Matrix Computations & Producer –Consumer using condition variables, mutexes – have different flow of computation and synchronization.)
- Use of Scheduling techniques as a means of synchronization may give rise to Memory in-consistency when two threads share the data

Cache Line Ping-Pong Caused by False Sharing

- ❖ Cache Related Issues
 - Cache Line
 - False Sharing
 - Memory consistency
 - Performance Impact
 - Correctness of the Results



Source : Reference [4],[6]

Cache Line Ping-Pong Caused by False Sharing

- **Remark :** Multiple threads manipulates a single piece of data
- Multiple threads manipulate different parts of large data structure, the programmer should explore ways of breaking it into smaller data structures and making them private to the thread manipulating them
- Making **memory consistency** across the threads is an important and it is for hardware efficiency.

Microsoft Windows using C /C++ languages

- ❖ Win 32 / Microsoft Foundation Class Library (**MFC**) wrapped Windows **API** functionality in C++ classes
 - Provides Developers with C/C++ interface for developing windows applications
- ❖ Performance Issues – Concept of Virtual Machine op-codes & Overhead Minimization
- ❖ Performance Issues – run in a Managed runtime environment
- ❖ Legacy Application Support

Source : Reference [4],[6], [7]

Microsoft Windows using C /C++ languages

- ❖ Creating Threads
 - `CreateThread();`
- ❖ Terminate the Thread
 - `ExitThread();`
- ❖ Managing Threads
- ❖ Thread Communication using Windows events
- ❖ Thread Synchronization
- ❖ Thread Atomic Operations
- ❖ Thread Pools; Thread Priority & Thread Affinity

Source : Reference [4],[6], [7]

Threading APIs for Microsoft .NET Framework

- ❖ Provide common execution environment for all the major languages : C++ & Visual Basic; C#
 - **ThreadStart()** – Constructs a new thread
- ❖ Microsoft .NET framework Class Library – provides examples of the APIs
- ❖ Managing Threads
- ❖ Thread Synchronization
- ❖ Thread Atomic Operations
- ❖ Thread Pools; Thread Affinity
- ❖ Thread Priority - .Net framework supports five levels thread priority

Source : Reference [4],[6], [7]

Part-V :

**An Overview of Multi-Core Processors
POSIX-Threads (Case Study)**

Multi-threaded Processing : Pthreads Prog.

Example: To perform Vector-Vector Multiplication.

Sequential Code:

In main().... :

```
// declarations and do memory allocations for vectors  
// Fill both vectors  
vec_vec_mult(vecA,vecB); // call vector multiplication function
```

Function Definition :

```
void vec_vec_mult(int *vecA,int *vecB)  
{  
    int sum=0,i;  
    for(i=0;i < VecSize; i++)  
        sum += va[i] * vb[i];  
  
    printf("\n Result of Vector Multiplication = %d",sum);  
}
```

Multi-threaded Processing : Pthreads Prog.

Pthread Code:

In main().... :

```
// declarations and do memory allocations  
// Fill both vectors  
dist = VecSize / NumThreads;                                // Divide vectors here  
for (counter = 0; counter < NumThreads; counter++) // Call thread function  
    pthread_create(&threads[counter], &pta, (void *(*) (void *)) doMyWork, (void *)  
(counter + 1));
```

Thread Function Definition :

```
void * doMyWork(int myId)  
{  
int counter, mySum = 0;  
/*calculating local sum by each thread */  
for (counter = ((myId - 1) * dist); counter <= ((myId * dist) - 1); counter++)  
    mySum += VecA[counter] * VecB[counter];  
  
/*updating global sum using mutex lock */  
pthread_mutex_lock(&mutex_sum);  
    finalsum += mySum;  
pthread_mutex_unlock(&mutex_sum);  
}
```

Multi-threaded Processing : Pthreads Prog.

Example: Finding the Minimum Value in the Integer List.

Sequential Code:

In main().... :

```
list = (int *) malloc (sizeof(int) *numElements); // memory allocation  
// Here fill list with random numbers  
min = findmin(list,numElements); // call function to find min val
```

Function Definition :

```
int findmin(int *list,int numElements)  
{  
    minval = list[0];  
    for(counter = 0 ; counter < numElements ; counter++)  
    {  
        if(list[counter]<minval) {  
            minval = list[counter]; }  
    } return minval;  
}
```

Multi-threaded Processing : Pthreads Prog.

Example: Finding the Minimum Value in the Integer List.

Sequential Code:

In main().... :

```
list = (int *) malloc (sizeof(int) *numElements); // memory allocation  
// Here, fill list with random numbers  
min = findmin(list,numElements); // call function to find min val
```

Function Definition :

```
int findmin(int *list,int numElements)  
{  
    minval = list[0];  
    for(counter = 0 ; counter < numElements ; counter++)  
    {  
        if(list[counter]<minval) {  
            minval = list[counter]; }  
    } return minval;  
}
```

Multi-threaded Processing : Pthreads Prog.

Example: Finding the Minimum Value in the Integer List.

Pthread Code :

In main().... :

```
partial_list_size = NumElements / NumThreads;      // Here divide list
list = (long int *)malloc(sizeof(long int) * NumElements); // memory allocation
minimum_value = list[0];
for(counter = 0 ; counter < NumThreads ; counter++) // Call thread function
{
    pthread_create(&threads[counter],&pta,(void *(*) (void *)) find_min,(void *) (counter+1));
}
```

Thread Function Definition :

```
void *find_min(void * myid )
{
    int myId = (int)myid;
    my_min = list[(myId-1)*partial_list_size];
for (counter = ((myId - 1) * partial_list_size); counter <=((myId * partial_list_size) -1);counter++)
    if (list[counter] < my_min) my_min = list[counter];
}
pthread_mutex_lock(&minimum_value_lock) ;
if (my_min < minimum_value)
    minimum_value = my_min;
pthread_mutex_unlock(&minimum_value_lock) ; }
```

Multi-threaded Processing : Pthreads Prog.

Example: Finding the Minimum Value in the Integer List.

Explanation :

Here we divide list depending upon number of threads

$$\text{partial_list_size} = \text{NumElements} / \text{NumThreads};$$

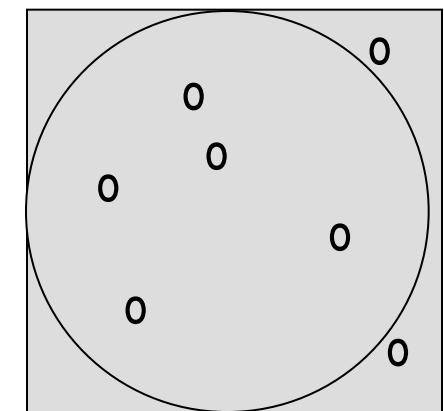
Each thread will find minimum value in its own part of list (say from 0-7 or 8-15) and using mutex lock assigns its calculated value to final minimum value.(see last if loop carefully)

Pthreads Prog. : Example : π Value

1. Assign fixed number of points to each thread.
2. Each thread generates random points and keeps track of the number of points that land in circle locality.
3. After all threads finish execution, their counts are combined to compute the value of π

Performance Issues

- ❖ False Sharing of data items (Two adjoining data items (which likely reside on the same cache line) are being continually written to by threads that might be scheduled on different cores).
- ❖ Estimate the cache line size of the cores and use higher dimensional arrays that are proportional to number of cores which share the cache line.



Synchronization Primitives in Pthreads

Example: Two threads on 2 cores are both trying to increment a variable x at the same time (Assume x is initially 0)

THREAD 1 :	THREAD 1 :
Increment (x)	Increment (x)
{	{
x= x+1	x= x+1
}	}
THREAD 1:	THREAD 1:
10 LOAD A, (x address)	10 LOAD A, (x address)
20 ADD A, 1	20 ADD A, 1
30 STORE A, x address	30 STORE A, x address

Use Threaded APIs mutex-locks (Mutual exclusion locks) to avoid Race Conditions

Source : Reference [4],[6], [7]

Synchronization Primitives in Pthreads

- ❖ **Example :** Computing the minimum entry in a list of integers
 - The list is partitioned equally among the threads
 - The size of each thread's partition is stored in the variable
- ❖ Performance for large number of threads is not scalable
(At any point of time, only one thread can hold a lock, only one thread can test updates the variable.)

For more details on programs refer HeGaPa-2012 web-page

Synchronization Primitives in Pthreads

:Alleviating Locking Overheads

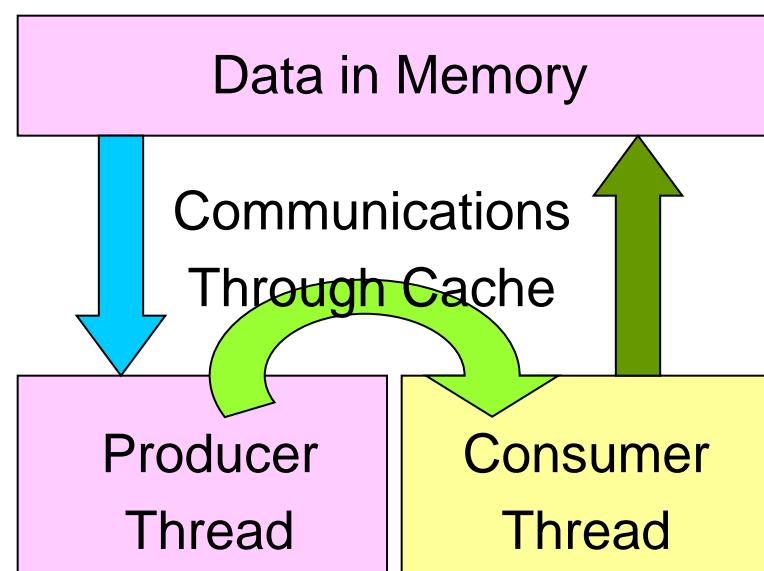
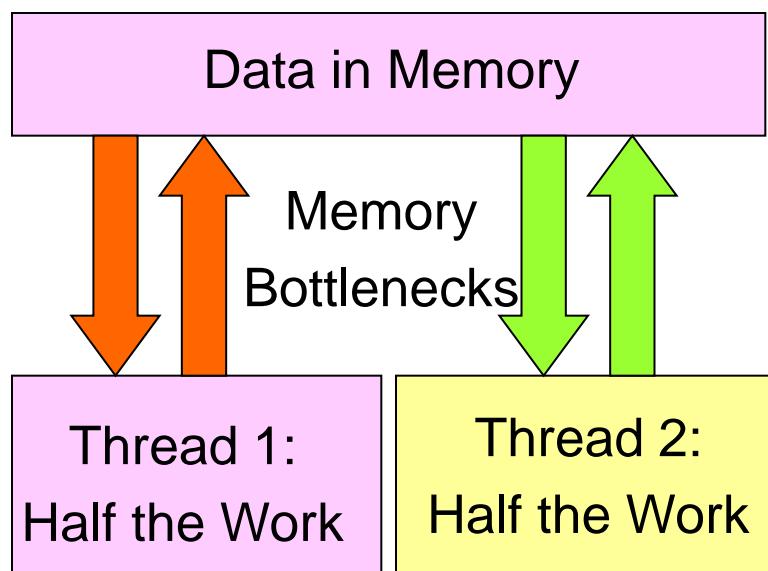
❖ Example : Finding *k-matches* in a list

- Finding **k** matches to a query item in a given list.
(The list is partitioned equally among the threads.
Assume that the list has **n** entries, each of **p**
threads is responsible for searching **n/p** entries of
the list.)
- Implement using **pthread_mutex_lock**.
- Reduce the idling overhead associated with locks
using **pthread_mutex_trylock**. (Reduce the
Locking overhead can be alleviated)

Source : Reference [4]

Producer/Consumer Problem : Synchronizing Issues

- ❖ Producer thread generates tasks and inserts it into a work-queue.
- ❖ The consumer thread extracts tasks from the task-queue and executes them one at a time.



Source : Reference [4],[6], [7]

Producer/Consumer Problem : Psuedo code

```
Semaphore s
void producer( ) {
    while (1) {
        <produce the next data>
        s->release( )
    }
}
void consumer ( ) {
    while (1) {
        s->wait( )
        <Consume the next data>
    }
}
```

Remarks : Neither producer nor consumer maintains an order. Synchronization problem exists. Buffer Size needs to be within a boundary to handle.

Producer & Consumer : (1). Using Semaphores; (2) Critical Directives (Mutexes – Locks); (3). Condition Variables

Source : Reference [4],[6], [7]

Producer/Consumer Problem : Dual Semaphores Solution

```
Semaphore sEmpty, sFull
void producer( ) {
    while (1) {
        sEmpty->wait ( )
        <produce the nest data >
        sFull->release( )
    }
}
void consumer ( ) {
    while (1) {
        sFull->release ( )
        <Consume the next data>
        sEmpty->wait ( )
    }
}
```

Remarks : Two independent Semaphores are used to maintain the boundary of buffer. **sEmpty**, and **sFull** retain the constraints of buffer capacity for operating threads.

Source : Reference [4],[6], [7]

Producer & Consumer : Critical Directive

- ❖ Producer thread generates tasks and inserts it into a work-queue.
- ❖ The consumer thread extracts tasks from the task-queue and executes them one at a time.
 - There is concurrent access to the task-queue, these accesses must be serialized using critical blocks.
 - The tasks of inserting and extracting from the task-queue must be serialized.
 - Define your own “`insert_into_queue`” and “`extract_from_queue`” from queue (Note that `queue full` & `queue empty` conditions must be explicitly handled)

Producer & Consumer : Critical Directive

- ❖ **Critical** Section directive is a direct application of the corresponding **mutex** function in **Pthreads**
- ❖ Reduce the size of the critical section in **Pthreads/OpenMP** to get better performance (Remember that **critical** section represents **serialization** points in the program)
- ❖ **Critical section** consists simply of an update to a single memory location.
- ❖ **Safeguard** : Define Structured Block I.e. no jumps are permitted into or out of the block. This leads to the threads wait indefinitely.

Producer & Consumer : Critical Directive

❖ Possibilities & Implementation Issues on Multi cores

- The producer thread must not overwrite the shared buffer when the previous task has not been picked up by a consumer thread
 - The consumer threads must not pick-up tasks until there is something present in the shared data structure.
 - Individual consumer threads should pick-up tasks one at a time.
- ❖ Implementation can be done using variable called **task_variable** which handles the wait condition of consumer & producer.

Producer & Consumer : Critical Directive

❖ Implementation & Performance Issues on Multi cores

- If **task_variable** = 0
 - *Consumer* threads wait but the *producer* thread can insert tasks into the shared data structure.
- If **task_variable** = 1
 - *Producer* threads wait to insert the task into the shared data structure but one of the *Consumer* threads can pick up the task available.
- All these operations on the variable **task_variable** should be protected by mutex-locks to ensure that only one thread is executing test-update on it.

Source : Reference [4],[6], [7]

Producer & Consumer : Critical Directive

❖ Performance Issues on Multi cores

- Consumer thread waits for a task to become available and executes when it is available.
- Locks represent sterilization points since critical sections must be executed by one after the other.
- Handle Shared Data Structures and Critical sections to reduce the idling overhead.

❖ Alleviating Locking Overheads

- To reduce the idling overhead associated with locks using **pthread_mutex_trylock**.

For more details on programs refer HeGaPa-2012 web-page

Producer & Consumer : Condition Variable for Synchronization

- ❖ A Condition Variable is a data object used for synchronization threads. This variable allows a thread to block itself until specified data reaches a predefined state.
- ❖ A condition variable always has a *mutex* associated with it.
- ❖ Use functions `pthread_cond_init` for initializing and `pthread_cond_destroy` for destroying condition variables.
- ❖ The concept of polling for lock as it consumes CPU cycles can be reduced. Use of condition variables may not use any CPU cycles until it is woken up.

Programming Aspects Examples

Implementation of Streaming Media Player on Multi-Core

- ❖ One decomposition of work using Multi-threads
- ❖ It consists of
 - A thread Monitoring a network port for arriving data,
 - A decompressor thread for decompressing packets
 - Generating frames in a video sequence
 - A rendering thread that displays frame at programmed intervals

Source : Reference : [4]

Programming Aspects Examples

Implementation of Streaming Media Player on Multi-Core

- ❖ The thread must communicate via shared buffers –
 - an **in-buffer** between the network and **decompressor**,
 - an **out-buffer** between the **decompressor** and **renderer**
- ❖ It consists of
 - Listen to portGather data from the network
 - Thread generates frames with random bytes (Random string of specific bytes)
 - Render threads pick-up frames & from the out-buffer and calls the display function
 - Implement using the Thread Condition Variables

Programming Examples

- ❖ Pthreads programs to illustrate read write API library calls :
Programs that illustrate the use of Read-Write Lock using different read-write lock APIs are described. Sample demo code that gives basic idea of how to use Read-Write Lock and one sample application using both mutex and Read-Write Lock is described so that one can get better idea of what is exact difference between these synchronization constructs and how to use them.

Programming Aspects Examples

- ❖ Pthreads programs to illustrate producer consumer program for large no. of threads

Programs that illustrate the application of Pthreads to producer / consumer problem with large number of producers and consumers. It illustrates the usage of Pthreads for large no. of threads reading and writing to vectors implemented in 'indexed-access' (or array implementation) and 'sequential-access' (or linked list implementation). It also shows how the problem can be solved using the Mutex objects and condition-variable objects of Pthreads. It also illustrates 'thread-affinity' setting, to bind threads to particular number of cores. For different thread-affinity masks, the performance can be observed.

Key Points

- Set up all the requirements for a thread before actually creating the thread. This includes initializing the data, setting thread attributes, thread priorities, mutex, attributes, etc...
- Buffer management is required in applications such as producer and consumer problems.
- Define synchronizations and data replication wherever it is possible and address stack variables,
- Avoid Race Conditions in designing algorithms and implementation
- Extreme caution is required to avoid parallel overheads associated with synchronization
- Design of asynchronous Programs and use of scheduling techniques require attention.

Key Points

- Match the number of runnable software threads to the available hardware threads
- Synchronization : In correct Answers ; Performance Issues
- Keeps Locks private
- Avoid dead-locks by acquiring locks in a consistent order
- Memory Bandwidth & contention Issues
- Lock contention (Using Multiple distributed locks)
- Design Lockless Algorithms – Advantages & dis-advantages
- Cache lines are – Hardware threads
- Writing synchronized code – Memory Consistency

Part-V :
Java Threading

Java Threads

- ❖ Threading and synchronization built in
- ❖ An object can have associated thread
 - Subclass Thread or Implement Runnable
 - “run”method is thread body
 - “synchronized”methods provide mutual exclusion
- ❖ Main program
 - Calls “start”method of Thread objects to spawn
 - Calls “join”to wait for completion

Java concurrency Packages

- ❖ The Concurrency Utilities packages provide a powerful, extensible framework of high-performance threading utilities such as thread pools and blocking queues.
- ❖ Concurrency packages provide low-level primitives for advanced concurrent programming.
- ❖ Concurrency packages provide utilities for concurrent programming, like Collections Framework for data structures.

Java concurrency package & subpackages

- ❖ **java.util.concurrent:** This package includes a few small standardized extensible frameworks, as well as some classes that provide useful functionality.
- ❖ **java.util.concurrent.locks:** Interfaces and classes providing a framework for locking and waiting for conditions that is distinct from built-in synchronization and monitors. The framework permits much greater flexibility in the use of locks and conditions.
- ❖ **java.util.concurrent.atomic:** A small toolkit of classes that support lock-free thread-safe programming on single variables.

java.util.concurrent package

java.util.concurrent package provides:

❖ Executors:

Executors is a simple standardized interface for defining custom thread-like subsystems, including thread pools, asynchronous IO, and lightweight task frameworks. It manages queuing and scheduling of tasks, and allows controlled subsystem shutdown.

❖ Queues:

The java.util.concurrent classes supplies an efficient scalable thread-safe non-blocking FIFO queue. Five implementations in java.util.concurrent blocking versions of put and take operation.

java.util.concurrent package

java.util.concurrent package provides:

- ❖ Synchronizers

Semaphore is a classic concurrency tool. **CountDownLatch** is a very simple yet very common utility for blocking until a given number of signals, events, or conditions hold. A **CyclicBarrier** is a resettable multiway synchronization point useful in some styles of parallel programming.

An **Exchanger** allows two threads to exchange objects at a rendezvous point, and is useful in several pipeline designs.

java.util.concurrent package

java.util.concurrent package provides:

- ❖ **Concurrent Collections**

For many threads are expected to access a given collection, A ***ConcurrentHashMap*** , ***ConcurrentSkipListMap*** and ***CopyOnWriteArrayList*** is normally preferable to a synchronized ***HashMap***, synchronized ***TreeMap*** and synchronized ***ArrayList*** when the expected number of reads and traversals greatly outnumber the number of updates to a list.

java.util.concurrent.locks package

java.util.concurrent.locks package provides:

- ❖ A **ReentrantLock** reentrant mutual exclusion Lock with the same basic behavior and semantics as the implicit monitor lock accessed using synchronized methods and statements, but with extended capabilities.
- ❖ The **ReadWriteLock** interface similarly defines locks that may be shared among readers but are exclusive to writers.
- ❖ The Condition interface describes condition variables that may be associated with Locks. These are similar in usage to the implicit monitors accessed using Object.wait, but offer extended capabilities.

java.util.concurrent.atomic package

java.util.concurrent.atomic package provides:

- ❖ Atomic classes are designed primarily as building blocks for implementing non-blocking data structures and related infrastructure classes.
- ❖ **AtomicBoolean**, **AtomicInteger**, **AtomicLong**, and **AtomicReference** classes provide access and updates to a single variable of the corresponding type.

Part-VI:

An Overview of Multi-Core Processors

OpenMP

OpenMP: Programming Model

❖ Threads based parallelization

- Open MP is based on the existence of multiple threads in the shared memory programming paradigm

❖ Explicit parallelization

- It is an explicit programming model, and offers full control over parallelization to the programmer

❖ Compiler directive based

- All of OpenMP parallelization is supported through the use of compiler directives

❖ Nested parallelism support

- The API support placement of parallel construct inside other parallel construct

Source : Reference : [4], [6], [14],[17], [22], [28]

OpenMP : Work-sharing construct 'for'

❖ OpenMP is usually used to parallelize loops:

- Find your most time consuming loops.
- Split them up between threads

Split-up this loop between multiple threads

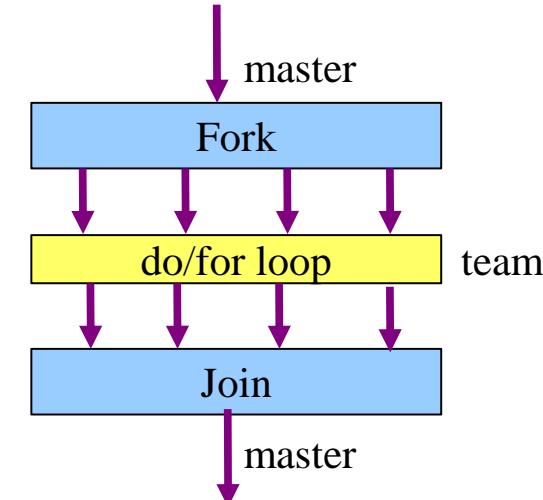
```
void main()
{
    double Res[1000];

    for(int i=0;i<1000;i++) {
        do_huge_comp(Res[i]);
    }
}
```

Sequential Program

```
#include "omp.h"
void main()
{
    double Res[1000];
    #pragma omp parallel for
    for(int i=0;i<1000;i++) {
        do_huge_comp(Res[i]);
    }
}
```

Parallel Program



OpenMP : How is OpenMP typically used? (C/C++)

- ❖ OpenMP is usually used to parallelize loops:

- Find your most time consuming loops.
- Split them up between threads.

Split-up this loop between multiple threads

```
void main()
{
    double Res[1000];

    for(int i=0;i<1000;i++) {
        do_huge_comp(Res[i]);
    }
}
```

Sequential Program

```
#include "omp.h"
void main()
{
    double Res[1000];
    #pragma omp parallel for
    for(int i=0;i<1000;i++) {
        do_huge_comp(Res[i]);
    }
}
```

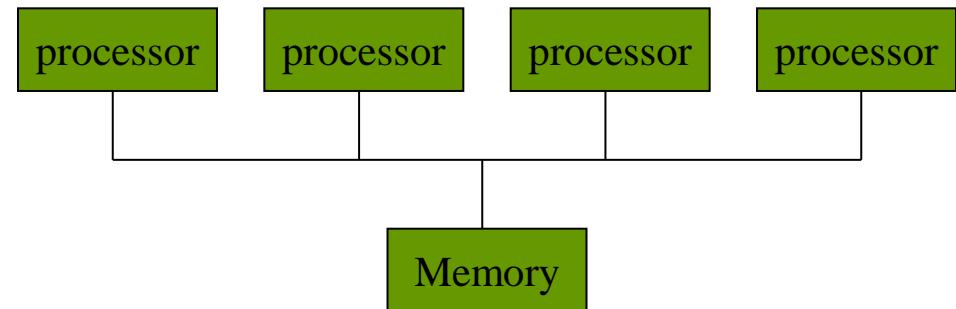
Parallel Program

For more details on programs refer HeGaPa-2012 web-page

OpenMP: Programming Model

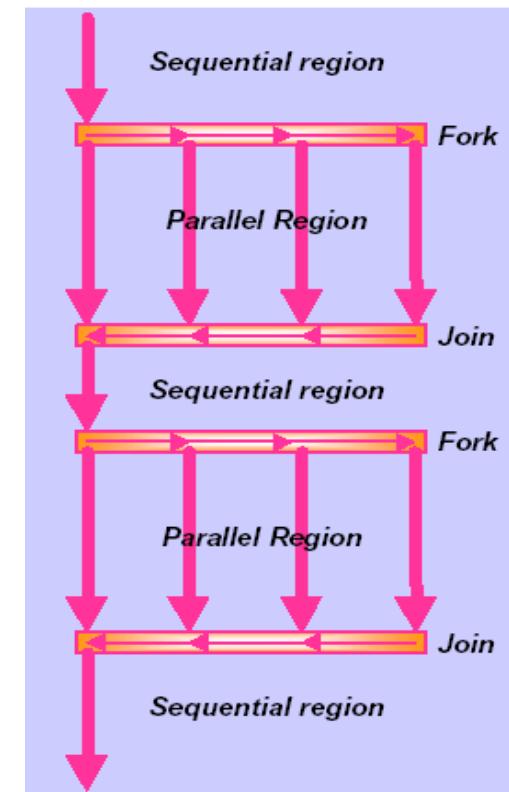
❖ Shared Memory Model

- Processes synchronize and communicate with each other through shared variables
- Supports *incremental parallelization*.



❖ Fork - Join parallelism

- OpenMP uses fork and join model for parallel execution
- OpenMP programs begin with single process: **master thread**.
- FORK : Master thread creates a team of parallel threads
- JOIN: When the team threads complete the statements in parallel region, they synchronize and terminate leaving master thread.
- Parallelism is added incrementally



OpenMP: Programming Model

❖ Dynamic threads

- The API provides dynamic altering of number of threads (Depends on the implementation)

How do threads interact?

❖ OpenMP is shared memory model.

- Threads communicate by sharing variables

❖ Unintended sharing of data can lead to race conditions:

- Race condition : when the program's outcome changes as the threads are scheduled differently
- To control race conditions: Use synchronization to protect data conflicts

OpenMP : Work-sharing Construct

for directive

for directive identifies the iterative work-sharing construct.

#pragma omp for [clause[,]clause]... new-line

for-loop

Clause is one of the following:

private(variable list)

firstprivate (variable list)

lastprivate (variable list)

reduction (variable list)

ordered , nowait

For more details on refer OpenMP on HeGaPa-2012 web-page

Synchronization construct: Barrier

Suppose we run each of these two loops in parallel over i:

```
for (i=0; i < N; i++)
    a[i] = b[i] + c[i];
```

```
for (i=0; i < N; i++)
    d[i] = a[i] + b[i];
```

This may give us a wrong answer ?

For more details on programs refer HeGaPa-2012 web-page

Synchronization construct : Barrier

We need to have updated all of a[] first, before using a[]

```
for (i=0; i < N; i++)  
    a[i] = b[i] + c[i];
```

wait !

```
for (i=0; i < N; i++)  
    d[i] = a[i] + b[i];
```

barrier

All threads wait at the barrier point and only continue
when all threads have reached the barrier point

For more details on programs refer HeGaPa-2012 web-page

OpenMP 3.0 Features

- ❖ Runtime Library Routines
- ❖ Lock Routines
- ❖ Timing Routines
- ❖ Data Environment
- ❖ Data-Sharing Attribute Clauses
- ❖ Data Copying Clauses
- ❖ Conditional compilation

For more details on programs, refer HeGaPa-2012 web-page

OpenMP 3.0 Features

- ❖ The **internal control variables (ICVs)** control the behavior of an OpenMP program. These ICVs store information such as the number of threads to use for future **parallel** regions, the schedule to use for work sharing loops and whether nested parallelism is enabled or not. Programs on how ICVs affect the operation of **parallel** regions are illustrated in OpenMP 3.0

OpenMP 3.0 Features

- ❖ OpenMP 3.X important features on **Task Scheduling**, **parallel construct**, **work-sharing construct**, **combined parallel work-sharing Constructs**, and **Synchronization constructs** are discussed. Example programs on number of threads for a **parallel** region, schedule of a **work-sharing** loop are provided.

OpenMP 3.0 Features

- ❖ An overview of several clauses for controlling the data environment during the execution of **parallel** clause, **task work-sharing** regions is discussed.
- ❖ Programs based on OpenMPI API runtime library routines **runtime library definitions**, **Execution environment routines**, **Lock routines** and **portable timer routine** are supported in the Hands-on Session.

OpenMP 3.0 Features

- ❖ Internal Control Variables (ICVs)
- ❖ Parallel Constructs
- ❖ Worksharing Constructs
- ❖ Combined Parallel Work sharing Constructs
- ❖ Task Construct
- ❖ Task Scheduling
- ❖ Master and Synchronization Constructs

For more details on programs refer HeGaPa-2012 web-page

Explicit Threads *versus* OpenMP Based Prog.

- ❖ OpenMP provides a layer on top of naïve threads to facilities a variety of thread-related tasks.
- ❖ Using Directives provided by OpenMP, a programmer is get rid of the task of initializing attribute objects, setting up arguments to threads, partitioning iteration spaces etc.... (This may be useful when the underlying problem has a static and /or regular task graph.)
- ❖ The overheads associated with automated generation of threaded code from directives have been shown to be minimal in the context of a variety of applications.

Explicit Threads *versus* OpenMP Based Prog.

- ❖ An Artifact of Explicit threading is that data exchange is more apparent. This helps in alleviating some of the overheads from data movement, false sharing, and contention.
- ❖ Explicit threading also provides a richer API in the form of condition waits.
- ❖ Locks of different types, and increased flexibility for building composite synchronization operations

Explicit Threads *versus* OpenMP Based Prog.

- ❖ Compiler support on Multi-Cores play an important role
- ❖ Issues related to OpenMP performance on Multi cores need to be addressed.
- ❖ Inter-operability of OpenMP/Pthreads on Multi-Cores require attention -from performance point of view
- ❖ Performance evaluation and use of tools and Mathematical libraries play an important role.

Source : Reference [4]

Explicit Threads *versus* OpenMP Based Prog.

- ❖ An Artifact of Explicit threading is that data exchange is more apparent. This helps in alleviating some of the overheads from data movement, false sharing, and contention.
- ❖ Explicit threading also provides a richer API in the form of condition waits.
- ❖ Locks of different types, and increased flexibility for building composite synchronization operations

Part-VII:

An Overview of Multi-Core Processors

Intel Threading Building Blocks (TBB)

Threading Building Blocks – Overview

Overview

- ❖ TBB enables developer to specify tasks instead of threads
- ❖ TBB targets threading for performance on Multi-cores
- ❖ TBB Compatible with other threading packages
- ❖ TBB emphasizes scalable, data-parallel programming
- ❖ Link libraries such as Intel's Math Kernel Library (MKL) and Integrated Performance Primitives (IPP) library are implemented internally using OpenMP.
 - You can freely link a program using Threading Building Blocks with the Intel MKL or Intel IPP library

Threading Building Blocks – Overview

Features

- ❖ Provides rich set of templates
 - Templates and C++ concept of generic programming (C++ Standard Template Library (STL))
- ❖ Does not require special language or compilers
- ❖ Ability to use Threading Building Blocks – any processor with any C++ Complier
- ❖ Promote Scalable Data Parallelism

TBB templates

- ❖ Initializing and Terminating the Library
- ❖ Loop Parallelization
 - **Parallel_for, Parallel_reduce, Parallel_scan** (Grain size, Interval, Workload for iteration, Time Taken)
 - Automatic Grain Size (Not easy) – Performance Issues
 - Recursive Range Specifications (block_range), Partitioning

- ❖ **parallel_while**

- Use of an unstructured stream or pile of work.
Offers the ability to add additional work to the file running

- ❖ pipeline (Throughput of pipeline)

- ❖ **Parallel_sort**

- Algorithm complexity
- ❖ Parallel algorithms for Streams

Application Perspective - Parallel Algorithms Design -TBB

How TBB can hide several task Scheduling events ?

- ❖ Static Load Balancing
 - Mapping for load balancing
 - Minimizing Interaction
- ❖ Data Sharing Overheads
- ❖ Dynamic Load Balancing
 - Overheads in parallel algorithms design

TBB templates- Advanced Algorithms

❖ Containers

- TBB provides highly concurrent containers that permit multiple threads to invoke a method simultaneously on the same container.
- **Concurrent queue, vector, and hash map** are provided.
- These can be used with the library, OpenMP, or raw threads

❖ Remark : Highly concurrent containers are very important because STL containers generally are not concurrent friendly.

- ❖ TBB provides Fine-grain locking and Lock free algorithm
- Algorithm complexity
- ❖ Parallel algorithms for Streams

TBB Scalable Memory Allocation

- ❖ Problems in Memory Allocation
 - Each Competes for global lock for each allocation and deallocation from a single global heap
 - False Sharing
- ❖ TBB offers two memory allocators
 - Scalable_allocator
 - Cache_aligned_allocator
- ❖ Memory Consistency and Fence
- ❖ TBB provides atomic templates

TBB Mutual Exclusion / Time

- ❖ TBB - Mutex
 - Mutual exclusion will be in terms of tasks. Mutual exclusion of tasks will lead to mutual exclusion of the corresponding threads upon which TBB maps defined tasks.
 - When to Use Mutual Exclusion (To prevent race conditions and other non-deterministic and undesirable behavior of tasks)
 - Mutex behavior
- ❖ TBB provides a thread-safe and portable method to compute elapsed time
 - tick_count Class

TBB Task Scheduler

- ❖ TBB - Task-based programming can improve the performance
 - Task Scheduler manages a thread pool and hides complexity which is much better than Raw Native Threads
 - OverSubscription – Getting the number of threads right is difficult
 - Fair Scheduling – OS uses; round-robin fashion
- ❖ Load imbalance can be handled easily comparison to thread-based programming
- ❖ Portability – TBB interfaces

Application Perspective - Parallel Algorithms Design -TBB

How TBB can hide several task Scheduling events ?

- ❖ Maximize data locality; Minimize volume of data
- ❖ Minimize frequency of Interactions; Overlapping computations with interactions.
 - Data replication; Minimize construction and Hot spots.
 - Use highly optimized collective interaction operations.
 - Collective data transfers and computations
 - Maximize Concurrency.
- ❖ **Types of Parallelism :** Data parallelism and Task parallelism
Combination of Data & Task parallelism; Stream parallelism

Threading Building Blocks – Overview

Benefits

- ❖ Scalability
 - Data Parallel Programming – Applications
 - Take advantage of all cores on Multi core Processor
- ❖ Specify Tasks instead of Threads
 - Runtime library
 - Automatically schedules tasks onto threads
 - Makes use of efficient processor resources
 - Load balancing many tasks

Threading Building Blocks - Overview

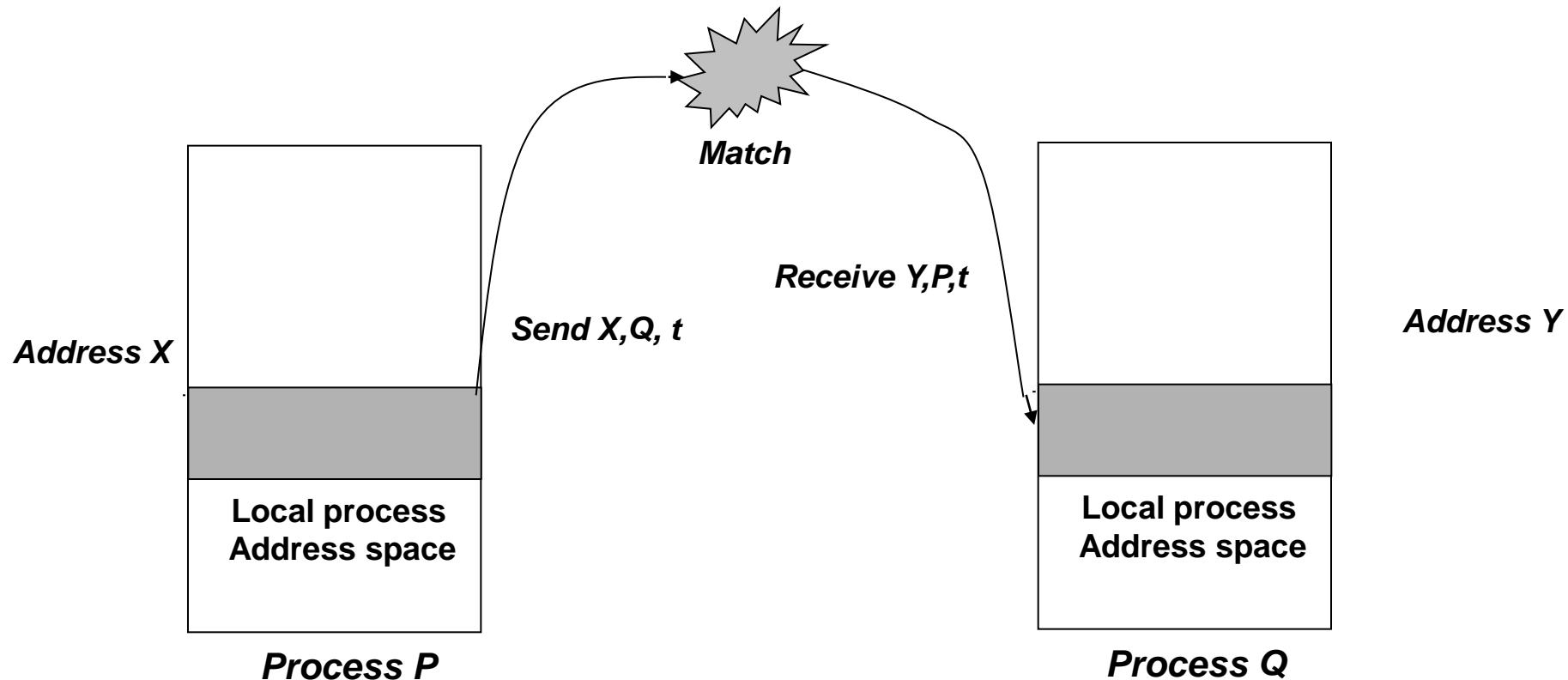
Benefits

- ❖ Task Scheduling
 - To use TBB library, you specify tasks, not threads
- ❖ Library maps tasks onto threads in an efficient manner
 - Writing Parallel_for loop – tedious using threading packages
 - Scalable program – harder ; No benefits in Performance
- ❖ Templates can give a creditable idea of performance with respect to problem size as the number of core increases

For more details on programs refer HeGaPa-2012 web-page

Part-VIII:
An Overview of Multi-Core Processors
MPI-1.x

The Message Passing Abstraction

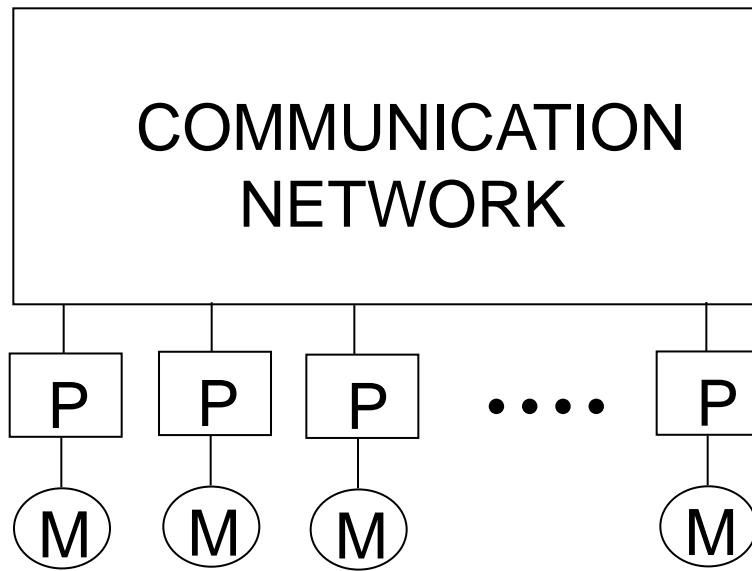


User-Level Send/receive message-passing abstraction : A data transfer from one local address space to another occurs when a *send* to particular processes matches with a *receive* posted by that process

For more details on programs refer HeGaPa-2012 web-page

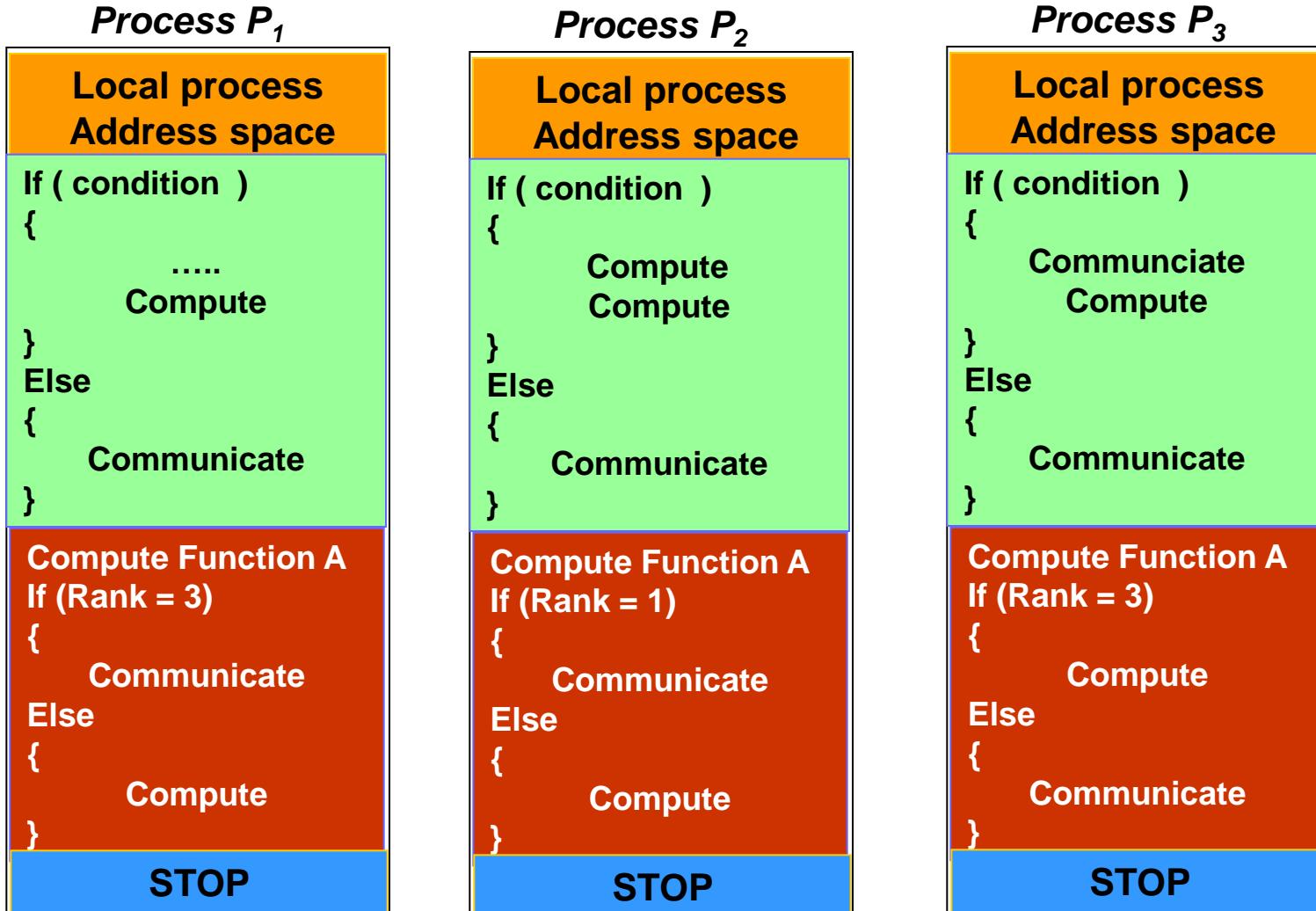
Message Passing Architecture Model

Message-Passing Programming Paradigm : Processors are connected using a message passing interconnection network.



- ❖ On most Parallel Systems, the processes involved in the execution of a parallel program are identified by a sequence of non-negative integers. If there are p processes executing a program, they will have ranks 0, 1, 2, ..., $p-1$.

The Message Passing Abstraction



SPMD /MPMD Program

Message Passing with SPMD :C program

```
main (int args, char **argv)
{
    if (process is to become a controller process)
    {
        Controller /* Arguments */;
    }
    else
    {
        Worker /* Arguments */;
    }
}
```

What is MPMD (Non-SPMD)?

- ❖ Different programs run on different nodes.
- ❖ If one program controls the others then the controlling program is called the *Master* and the others are called the *slaves*.

Basic steps in an MPI program

- ❖ Initialize for communications
- ❖ Communicate between processors
- ❖ Exit in a “clean” fashion from the message-passing system when done communicating.

Source : Reference : [11], [12], [25], [26]

Format of MPI Calls

C Language Bindings

```
Return_integer = MPI_Xxxxx(parameter, ...);
```

- ❖ Return_integer is a return code and is type integer. Upon success, it is set to MPI_SUCCESS.
- ❖ Note that case is important
- ❖ MPI must be capitalized as must be the first character after the underscore. Everything after that must be lower case.
- ❖ C programs should include the file mpi.h which contains definitions for MPI constants and functions

Format of MPI Calls

Fortran Language Buildings

Call MPI_XXXXX(parameter,..., ierror)

or

call mpi_xxxxx(parameter,..., ierror)

- ❖ Instead of the function returning with an error code, as in C, the Fortran versions of MPI routines usually have one additional parameter in the calling list, `ierror`, which is the return code. Upon success, `ierror` is set to `MPI_SUCCESS`.
- ❖ Note that case is not important
- ❖ Fortran programs should include the file `mpif.h` which contains definitions for MPI constants and functions

Exceptions to the MPI call formats are timing routines

- ❖ Timing routines
 - MPI_WTIME and MPI_WTICK are functions for both C and Fortran
- ❖ Return double-precision real values.
- ❖ These are not subroutine calls

Fortran

Double precision MPI_WTIME()

C

Double precision MPI_Wtime(void);

MPI Messages

- ❖ **Message** : data (3 parameters) + envelope (3 parameters)
 - **Data** : startbuf, count, datatype
 - **Startbuf**: address where the data starts
 - **Count**: number of elements (items) of data in the message
 - **Envelope** : dest, tag, comm
 - **Destination or Source**: Sending or Receiving processes
 - **Tag**: Integer to distinguish messages

Communicator

- ❖ The communicator is communication “universe.”
- ❖ Messages are sent or received within a given “universe.”
- ❖ The default communicator is MPI_COMM_WORLD

Initializing MPI

- ❖ Must be first routine called.

- ❖ C

```
int MPI_Init(int *argc,char ***argv);
```

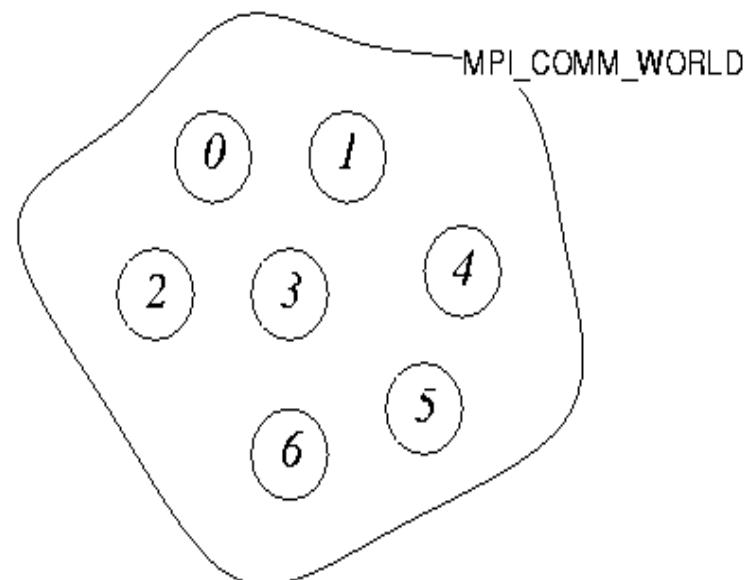
- ❖ Fortran

```
MPI_INIT(IERROR)
```

```
integer IERROR
```

MPI_COMM_WORLD communicator

A communicator is MPI's mechanism for establishing individual communication "universe."



- ❖ What is my process id number ?

MPI_COMM_RANK (Rank starts from the integer value 0 to)

Fortran : call MPI_COMM_RANK (comm, rank, ierror)
 integer comm, rank, ierror

C : int MPI_Comm_rank (MPI_Comm comm, int *rank)

Questions :

- ❖ How many processes are contained within a communicator?
- ❖ How many processes am I using?

MPI_COMM_SIZE

Fortran : call MPI_COMM_SIZE (comm, size, ierror)

C : int MPI_Comm_size (MPI_Comm comm, int *size)

- ❖ What is my process id number ?

MPI_COMM_RANK (Rank starts from the integer value 0 to)

Fortran : call MPI_COMM_RANK (comm, rank, ierror)
integer comm, rank, ierror

C : int MPI_Comm_rank (MPI_Comm comm, int *rank)

Exiting MPI

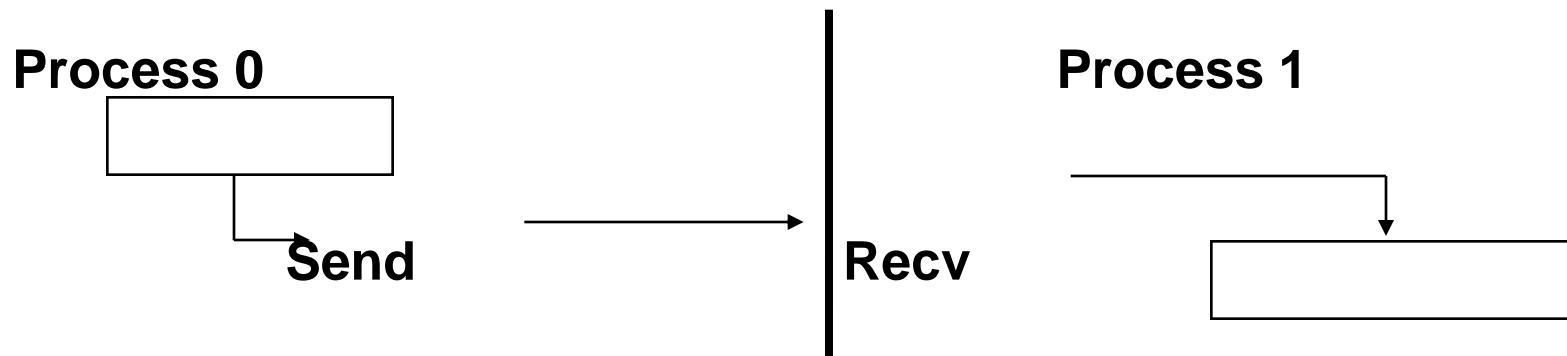
- ❖ C : int MPI_Finalize()
- ❖ Fortran
MPI_FINALIZE(IERROR)
INTEGER IERROR

Note : Must be called last by all processes.

What makes an MPI Program ?

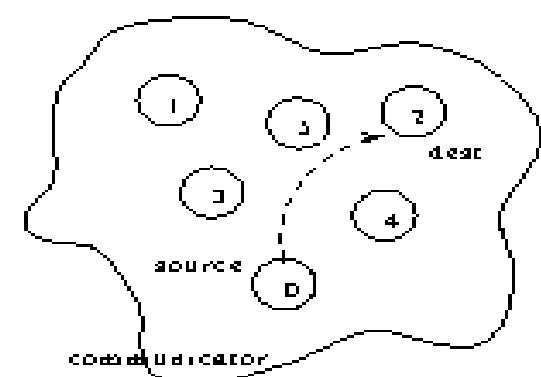
- ❖ Include files
mpi.h (C)
mpif.h (Fortran)
- ❖ Initiation of MPI
MPI_INIT
- ❖ Completion of MPI
MPI_FINALIZE

MPI Send and MPI Receive Library Calls



Fundamental questions answered

- ❖ To whom is data sent? What is sent?
- ❖ How does the receiver identify it?
- ❖ Communication between two processes
- ❖ Source process sends message to *destination* process
- ❖ Communication takes place within a *communicator*
- ❖ Destination process is identified by its *rank* in the communicator



Is MPI Large or Small?

(Contd...)

The MPI Message Passing Interface Small or Large

MPI can be small. : One can begin prog. with 6 MPI function calls

MPI_INIT	<i>Initializes MPI</i>
MPI_COMM_SIZE	<i>Determines number of processors</i>
MPI_COMM_RANK	<i>Determines the label of the calling process</i>
MPI_SEND	<i>Sends a message</i>
MPI_RECV	<i>Receives a message</i>
MPI_FINALIZE	<i>Terminates MPI</i>

MPI can be large

One can utilize any of 125 functions in MPI.

Is MPI Large or Small?

(Contd...)

MPI can be small. : One can begin prog. with 6 MPI function calls

MPI_Init (&argc, &argv); *Initializes MPI*

MPI_Comm_size (MPI_COMM_WORLD, &Numprocs);

Determines number of processors

MPI_Comm_rank (MPI_COMM_WORLD, &MyRank);

Determines the label of the calling process

MPI_Send (void* buf, int count, MPI_Datatype datatype, int dest,

int tag MPI_Comm comm) ;

Sends a message

MPI_Recv(void*buf, int count, MPI_Datatype datatype, int source,

int tag MPI_Comm comm,

MPI_Status *status);

Receives a message

MPI_Finalize(); *Terminates MPI*

MPI Point-to-Point Communication

MPI Routines used in Hello_World Program : MPI_Send/MPI_Recv

Synopsis : C

```
int MPI_Send (void* buf, int count, MPI_Datatype datatype, int dest,  
              int tag MPI_Comm comm) ;
```

```
int MPI_Recv(void*buf, int count, MPI_Datatype datatype, int source,  
             int tag MPI_Comm comm, MPI_Status *status);
```

Synopsis :Fortran

```
MPI_SEND (buf, count, datatype, dest, tag, comm, ierror)
```

```
MPI_RECV (buf, count, datatype, source, tag, comm, ierror)
```

```
<type> buffer(*),
```

```
integer count, datatype, dest, source, tag, comm, ierror
```

Message Envelope in MPI : Status

Status is a pointer to a structure which holds various information about the message received.

MPI_Status Status

Source process rank and the actual message tag can be found in the two fields

Status. MPI_SOURCE

Status. MPI_TAG

Routine **MPI_Get_count(&Status, MPI_INT, &C)** uses information in Status to determine the actual number of data items of a certain datatype(i.e MPI_INT) and puts the number in C.

MPI Point-to-Point Communication: Communication Modes

Synchronous: The send cannot return until the corresponding receive has started. An application buffer is available in the receiver side to hold the arriving message.

Buffered : Buffered send assumes the availability of buffer space which is specified by the MPI_Buffer_attach(buffer,size) which allocates user buffer of size bytes.

Standard : The send can be either synchronous or buffered, depending on the implementation.

Ready: The send is certain that the corresponding receive has already started. It does not have to wait as in the synchronous mode.

Source : Reference : [11], [12], [25], [26]

MPI Point-to-Point Communication: Communication Modes

MPI Primitive	Blocking	Nonblocking
Standard Send	<code>MPI_Send</code>	<code>MPI_Isend</code>
Synchronous Send	<code>MPI_Ssend</code>	<code>MPI_Issend</code>
Buffered Send	<code>MPI_Bsend</code>	<code>MPI_Ibsend</code>
Ready Send	<code>MPI_Rsend</code>	<code>MPI_Irsend</code>
Receive	<code>MPI_Recv</code>	<code>MPI_Irecv</code>
Completion Check	<code>MPI_Wait</code>	<code>MPI_Test</code>

Different Send/Receive operations in MPI

MPI Collective Communications

Collective Communications

- ❖ The sending and/or receiving of messages to/from groups of processes.
- ❖ A collective communication implies that all processes need participate in a global communication operation.
- ❖ Involves coordinated communication within a group of processes
- ❖ No message tags used
- ❖ All collective routines block until they are locally complete

Source : Reference : [11], [12], [25], [26]

MPI Collective Communications and Computations

- ❖ Communications involving a group of processes.
- ❖ Called by all processes in a communicator.
- ❖ Examples:
 - Barrier synchronization.
 - Broadcast, scatter, gather.
 - Global sum, global maximum, etc.
- ❖ Two broad classes :
 - Data movement routines
 - Global computation routines

MPI Collective Communications and Computations

Allgather	Allgatherv	Allreduce
Alltoall	Alltoallv	Bcast
Gather	Gatherv	Reduce
Reduce Scatter	Scan	Scatter
Scatterv		

- ❖ All versions deliver results to all participating processes
- ❖ V-version allow the chunks to have different non-uniform data sizes (Scatterv, Allgatherv, Gatherv)
- ❖ All reduce, Reduce , ReduceScatter, and Scan take both built-in and user-defined combination functions

Source : Reference : [11], [12], [25], [26]

MPI Collective Communications

Type	Routine	Functionality
Data Movement	MPI_Bcast	One-to-all, Identical Message
	MPI_Gather	All-to-One, Personalized messages
	MPI_Gatherv	A generalization of MPI_Gather
	MPI_Allgather	A generalization of MPI_Gather
	MPI_Allgatherv	A generalization of MPI_Allgather
	MPI_Scatter	One-to-all Personalized messages
	MPI_Scatterv	A generalization of MPI_Scatter
	MPI_Alltoall	All-to-All, personalized message
	MPI_Scatterv	A generalization of MPI_Alltoall

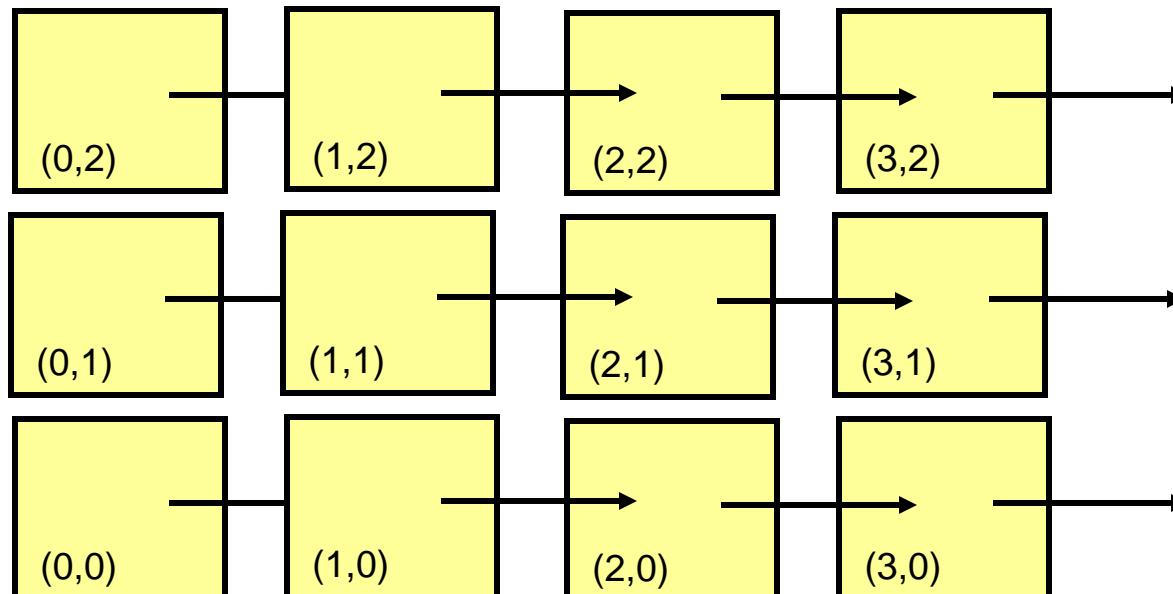
MPI Derived Datatypes

Message type

- ❖ A message contains a number of elements of some particular datatype
- ❖ MPI datatypes:
 - Basic types
 - Derived Data types (Vectors; Structs; Others)
- ❖ Derived types can be built up from basic types
- ❖ C types are different from Fortran types

MPI Process Topologies - Cartesian topology

- ❖ A two dimensional Cartesian Decomposition
- ❖ MPI provides a collection of routines for defining, examining, and manipulating Cartesian topologies



- ❖ For example, the **second** process from the **left** and the **third** from the **bottom** is labeled as (1,2)

MPI Communicators

- ❖ Two types of Communicators

Intra-communicators:Collection of processes that can send messages to each other in engage in collective communication operations

Inter-communicators :Used for sending messages between processes belonging to disjoint **intra-communicators**

- ❖ Why should we bother about **inter-communicators** ?
- ❖ A minimal (**intra**) communicator is composed of
 - **a group** (is an ordered collection of processes. If a group consists of p process, each process in the group is assigned a unique **rank**)
 - **a context** (is a system defined object that uniquely identifies a communicator)

Source : Reference : [11], [12], [25], [26]

Part-VIII:

An Overview of Multi-Core Processors

MPI-2.x

Introduction to Parallel I/O in MPI-2 - Outline

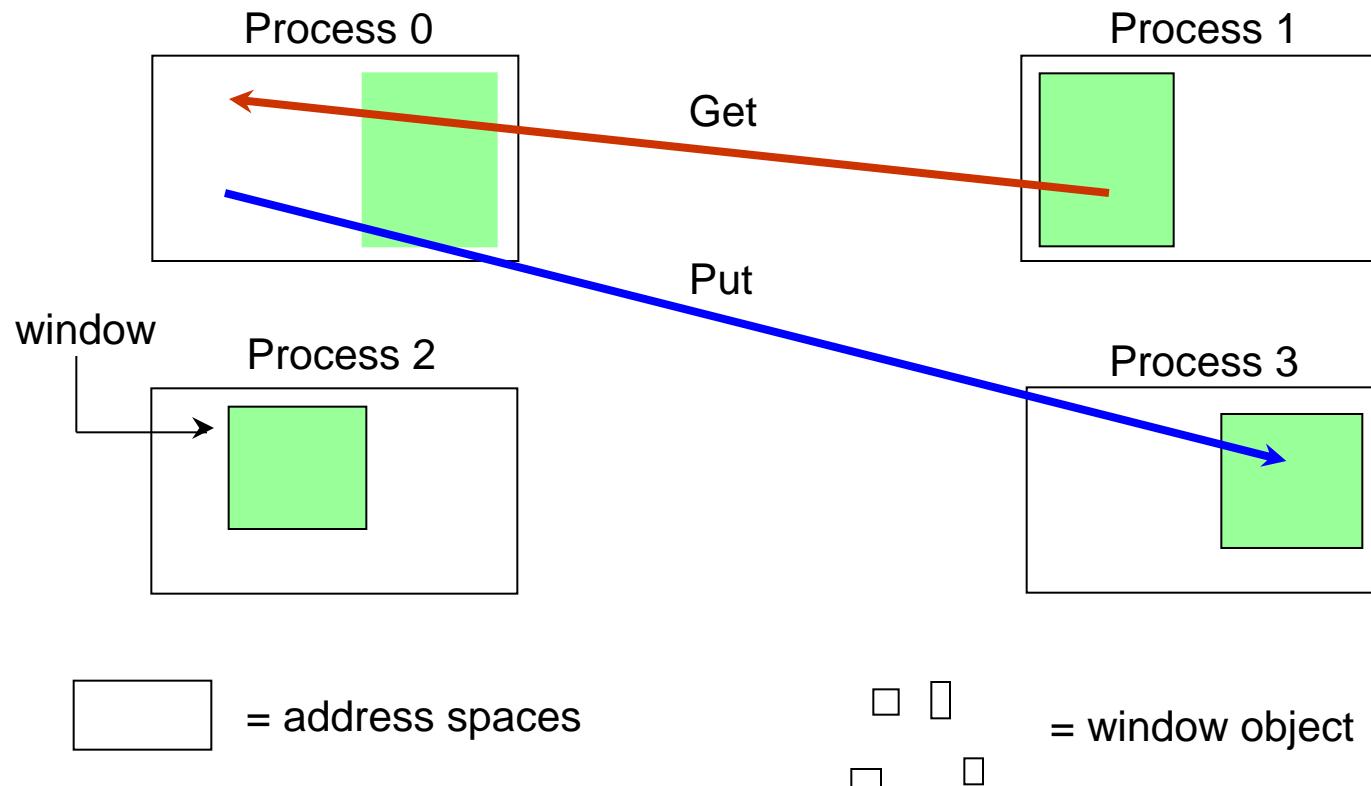
- ❖ Why do I/O in MPI?
- ❖ Non-parallel I/O from an MPI program
- ❖ Non-MPI parallel I/O to shared file with MPI I/O
- ❖ parallel I/O to shared file with MPI I/O
- ❖ Fortran-90 version
- ❖ Reading a file with a different number of processes
- ❖ C++ version
- ❖ Survey of advanced features in MPI I/O

Source : Reference : [4], [6], [11],[12],[25], [26]

MPI-2: Introduction to Remote Memory Access

- ❖ Features in MPI I/O
- ❖ Remote Memory Access - Windows
- ❖ One-sided operations
- ❖ Synchronization
- ❖ Fortran 90 and MPI C++
- ❖ Balancing efficiency and portability across a wide class of architectures

Remote Memory Access Windows and Window Objects



MPI-2 Advantages of Remote Memory Access Operations

- ❖ Multiple data transfer with a single synchronization operation
 - like BSP model
- ❖ Bypass tag matching
 - effectively precomputed as part of remote offset
- ❖ Significantly faster than send/receive on SGI Origin, for example, provided special memory is used.

Source : Reference : [4], [6], [11], [12],[24],[25], [26]

MPI - Shared Memory Allocation

- ❖ Supports communication via shared memory between MPI Processes
 - Based on Message size (Short or Long Messages)
 - Compromise between Performance and Memory Use
 - Implementation varies as per Vendor Specification
 - Works for SMPs and NUMA based shared Memory Computer.

Source : Reference : [4], [6], [11],[12],[24],[25], [26], MPI-2 or SunMPI 3.0

MPI – Multithreaded Programming

❖ MPI-2 Specification - MPI & threads

- Use thread-safe library (For ex : `libmpi_mt.so` in SUN MPI 3.0)
- When two concurrently running threads make MPI calls, the outcome will be as if the calls executed in some order.
- Blocking MPI calls will block the calling thread only. A blocked calling thread will not prevent progress of other runnable threads on the same process, nor will it prevent them from executing MPI calls.
- Multiple sends and receives are concurrent

Reference: Sun MPI 3.0

Source : Reference : [36], [37], [38], [39], [40], [41]

Threads and Processes in MPI-2

- ❖ Thread systems where the operating system (the kernel) is not involved in managing the individual threads are called *user threads*.
 - *User threads* tend to be *faster* than kernel threads (User threads takes smaller time to switch between the threads within the same process)
 - Restriction : System calls will block all threads in the process containing the thread, made the system call.) Not just the calling thread)
- ❖ Difficult to write truly portable multithreaded programs (Application can not assume the the entire process will not be blocked when a thread calls a library routine)
- ❖ The POSIX thread (Pthreads) specification does not specify whether the threads are user or kernel; it it is upto threads implementation

Source : Reference : [4], [6], [11],[12],[24],[25], [26]

MPI-2.0 : Thread Safe Library

- ❖ Mixed-Model Programming - MPI for SMP Clusters
 - MPI for Multi core Processors
- Thread safe - **libmpi_mt.so**
- Non-thread-safe (Default) - **libmpi.so**

For programs that are not multi-threaded, the user should use **libmpi.so** whenever possible for maximum performance.

Reference: Sun MPI 3.0

Source : Reference : [4], [6], [11],[12],[24],[25], [26], [36], [37], [38], [39], [40], [41]

MPI-2 : Introduction to Thread Safety

❖ Thread Safety & Message Passing Library

- Thread safety means that multiple threads can be executing Message Passing library calls without interfacing with one another
- Thread unsafety occurs when the message passing system is expected to hold certain parts of the process state.
- It is impossible to hold certain parts of the process state and it is impossible to hold that process state for more than one thread at time.

Source : Reference : [4], [6], [11],[12],[24],[25], [26], [36], [37], [38], [39], [40], [41]

MPI-2 : Introduction to Thread Safety

❖ Thread Safety & Message Passing Library

- Example : The concept of “the most recently received message” to avoid passing a status stored on the process’s stack
 - That is user code will look something like

```
recv(msg, type);  
src = get_src();  
len = get_len();
```
 - Single threaded case - Works well
 - Multi-threaded case - several *receives* may be in progress simultaneously
 - When **get_src** is called, it may not be clear for which message the source is supposed to be returned.
- MPI provides thread safe implementations so that MPI can work hand to hand with thread libraries

MPI – Multithreaded Programming (Thread Safe)

- ❖ Each thread within an MPI process may issue MPI calls; however, threads are not separately addressable.
 - That is, the rank of a send or receive call identifies a process, not a thread, meaning that no order is defined for the case where two threads call
 - **MPI_Recv** with the same tag and communicator. Such threads are said to be in conflict.
- ❖ **Note** : Overheads in MPI implementation –
 - How to handle conflicts and Data Races ?
 - How to write thread Safe programs ?

MPI – Multithreaded Programming (Thread Safe)

- ❖ Each thread within an MPI process may issue MPI calls; however, threads are not separately addressable.
 - If threads within the same application post conflicting communication calls, data races will result.
 - You can prevent such data races by using distinct communicators or tags for each thread.
 - Prevention of data races and conflict - User can write thread safe programs

Source : Reference : [36], [37], [38], [39], [40], [41]

MPI – Multithreaded Programming (Thread Safe)

- ❖ Adhere to these guidelines:

- You must not have an operation posted in one thread and then completed in another.
- you must not have a request serviced by more than one thread.
- A data type or communicator must not be freed by one thread while it is in use by another thread.
- Once MPI_Finalize has been called, subsequent calls in any thread will fail.

MPI – Multithreaded Programming (Thread Safe)

❖ **Adhere to these guidelines:**

- You must ensure that a sufficient number of lightweight processes (LWPs) are available for your multithreaded program. Failure to do so may degrade performance or even result in deadlock.
- You cannot stub the thread calls in your multithreaded program by omitting the threads libraries in the link line.
- The libmpi.so library automatically calls in the threads libraries, which effectively overrides any stubs.

Reference: Sun MPI 3.0

MPI – Multithreaded Programming (Thread Safe)

MPI Library Calls - Guidelines

- Provides specific guidelines that apply for specific some routines - Collective calls and Communicator operations
- **MPI_Wait, MPI_Waitall, MPI_Waitany, MPI_Waitsome**

In a program where two or more threads call one of these routines, you must ensure that they are not waiting for the same request. Similarly, the same request cannot appear in the array of requests of multiple concurrent wait calls.

Source : Reference : [25], [26], [36], [37], [38], [39], [40], [41]

MPI – Multithreaded Programming (Thread Safe)

MPI Library Calls - Guidelines

❖ MPI_Cancel

One thread must not cancel a request while that request is being serviced by another thread.

❖ MPI_Probe, MPI_Iprobe

A call to MPI_Probe or MPI_Iprobe from one thread on a given communicator should not have a source rank and tags that match those of any other probes or receives on the same communicator. Otherwise, correct matching of message to probe call may not occur.

Source : Reference : [25], [26], [36], [37], [38], [39], [40], [41]

MPI – Multithreaded Programming (Thread Safe)

MPI Library Calls - Guidelines

❖ Collective Calls

- Collective calls are matched on a communicator according to the order in which the calls are issued at each processor.
- All the processes on a given communicator must make the same collective call.
- You can avoid the effects of this restriction on the threads on a given processor by using a different communicator for each thread.

Source : Reference : [4], [6], [11],[12],[24],[25], [26], MPI-2 or SunMPI 3.0

MPI Library Calls - Guidelines

❖ Communicator Operations

- Use the same or different communicators.
- threads in different processes participating in the same communicator operation require grouping
- Do not free a communicator in one thread if it is still being used by another thread.

Source : Reference : [4], [6], [11],[12],[24],[25], [26], [36], [37], [38], [39], [40], [41]

MPI – Multithreaded Programming (Thread Safe)

➤ Communicator Operations

Each of the communicator functions operates simultaneously with each of the noncommunicator functions, regardless of what the parameters are and of whether the functions are on the same or different communicators. However, if you are using multiple instances of the same communicator function on the same communicator, where all parameters are the same, it cannot be determined which threads belong to which resultant communicator. Therefore, when concurrent threads issue such calls, you must assure that the calls are synchronized in such a way that threads in different processes participating in the same communicator operation are grouped. Do this either by using a different base communicator for each call or by making the calls in single-thread mode before actually using them within the separate threads. Do not free a communicator in one thread if it is still being used by another thread.

Reference: Sun MPI 3.0

Source : Reference : [4], [6], [11],[12],[24],[25], [26], [36], [37], [38], [39], [40], [41]

Threads and MPI in MPI-2

Thread Safety & MPI – Issues to be addressed

- ❖ Performance tradeoffs between multi-threaded and single-threaded code.
 - I/O operations
 - Against Inconsistent updates to the same memory location from different threads
 - Software locks and System locks are quite expensive
- ❖ Vendors sometimes provide single threaded /Multi-threaded libraries
 - Have I been linked with the right library ?
 - DO I suffer with occasional and mysterious errors

Source : Reference : [4], [6], [11],[12],[24],[25], [26]

Threads and MPI in MPI-2

MPI Library Calls - Guidelines

- ❖ MPI-2 function to initialize

- **int MPI_Init_thread
int *argc,char ***argv,int required, int *provided)**
(C-Binding)
- **MPI_INIT_THREAD(required, provided, ierror)**
Fortran binding

Regardless of whether **MPI_Init** or **MPI_Init_thread** is called, the MPI program must end with a call to **MPI_Finalize**

Source : Reference : [4], [6], [11],[12],[24],[25], [26], [36], [37], [38], [39], [40], [41]

Threads and MPI in MPI-2

MPI Library Calls - Guidelines

- ❖ MPI-2 specifies four levels of thread safety
 - MPI_THREAD_SINGLE: only one thread
 - MPI_THREAD_FUNNELED: only one thread that makes MPI calls
 - MPI_THREAD_SERIALIZED: only one thread at a time makes MPI calls
 - MPI_THREAD_MULTIPLE: any thread can make MPI calls at any time
- ❖ `MPI_Init_thread(..., required, &provided)` can be used instead of `MPI_Init`

Source : Reference : [4], [6], [11],[12],[24],[25], [26], [36], [37], [38], [39], [40], [41]

Part-VIII:

An Overview of Multi-Core Processors

MPI-3.x

MPI 3.0 Efforts

- ❖ Increasing prevalence of multi- and many-core processors calls for extended MPI facilities for dealing with threads as first class MPI entities.
- ❖ This leads to issues like the Probe/Recv consistency issue in MPI
- ❖ Efforts seeks to introduce a powerful and convenient way of direct addressing of the threads as MPI processes.

MPI 3.0 Efforts

- ❖ Treating Threads as MPI Processes
- ❖ Dynamic Thread levels
- ❖ I/O threads
- ❖ Address Thread Locks & MPI

Source : Reference : MPI-3 or SunMPI 3.0

MPI 3.0 Thread Init/Finalize Routines

- ❖ **Problem :**
 - MPI currently does not explicitly know threads
 - Process can be mapped to different cores/SMTs
 - Thread scheduling is left to the OS
- ❖ **Relevant Issues:**
 - Explicit Thread Init/Finalize Routines
 - Allow the process manager to perform intelligent mapping
 - Optional calls - application does not necessarily have to call

MPI 3.0 - Dynamic Threads Levels

- ❖ Problem: MPI specifies thread-level support at Init time
 - Even if a small fraction of the code uses THREAD-MULTIPLE, the entire code is forced to go through locks
- ❖ Performance Impact (messaging rate)
- ❖ Efforts
 - Add calls for **`MPI_Set_thread_level()`** to dynamically change thread-level within the application

Source : Reference : Intel MPI, MPI-3, SunMPI 3.0, 36,37,38,39,40,41

MPI 3.0 - Dynamic Threads Levels

- ❖ MPI_Set_thread_level(int required, int* provided)
 - Hinting mechanism only
- ❖ Relevant Issues:
 - If an implementation allows the thread-level reduction, but not increase, the application might not be able to deal with it
 - Asynchronous Progress Threads
- ❖ Requires synchronization with the progress thread to change level
 - Collective Operations: Some MPI implementations use different collective operations based on the thread-level

Source : Reference : Intel MPI, MPI-3, SunMPI 3.0, 36,37,38,39,40,41

MPI treating Threads as MPI process

- ❖ A new collective routine `MPI_Comm_thread_register()` is introduced to create a communicator in which existing threads become MPI processes with unique ranks.
- ❖ The existing routine `MPI_Comm_free()` is extended to operate on the resulting communicators.
- ❖ Prerequisite: `MPI_THREAD_MULTIPLE` thread support level

Reference: Intel MPI

Source : Reference : Intel MPI, MPI-3, SunMPI 3.0, 36,37,38,39,40,41

MPI treating Threads as MPI process

MPI_comm_thread_register (basic language binding)

`MPI_Comm_thread_register(comm, local_thread_index, local_num_threads, newcomm)`

IN comm

original communicator

IN local_thread_index

index of the calling thread (0 to local_num_threads – 1) on the current MPI process in comm

IN local_num_threads

total number of threads issuing this call on the current MPI process in comm

comm

OUT newcomm

new communicator based on threads

Reference: Intel MPI

Source : Reference : Intel MPI, MPI-3, SunMPI 3.0, 36,37,38,39,40,41

MPI treating Threads as MPI process

MPI_comm_thread_register (basic language binding)

C:

```
int      MPI_Comm_thread_register(MPI_Comm      comm,      int
local_thread_
Index, int_local_num_threads, MPI_Comm *newcomm)
```

Fortran:

```
MPI_COMM_THREAD_REGISTER(INTEGER   COMM,   INTEGER
LOCAL_THREAD_INDEX,   INTEGER   LOCAL_NUM_THREADS,
INTEGER NEWSOMM, INTEGER IERROR)
```

Reference: Intel MPI

Source : Reference : Intel MPI, MPI-3, 36,37,38, 40,41

MPI treating Threads as MPI process

Example: OpenMP parallel section

```
!$OMP parallel num_threads(4)
call MPI_COMM_THREAD_REGISTER(      &
MPI_COMM_WORLD, OMP_GET_THREAD_NUM(),
& OMP_GET_NUM_THREADS(), NEWCOMM)
!
!      Whatever MPI operations on and in NEWCOMM
!
call      MPI_COMM_FREE(NEWCMM)
!$OMP parallel end
```

Source : Reference : Intel MPI, MPI-3, 36,37,38, 40,41

Intel MPI Library : MPI-2 Features

- ❖ MPI-2 Specification conformance
 - Standardized job startup mechanism (**mpiexec**)
 - Process Spawning /attachment (**Socket device only**)
 - One-sided communication
 - Extended collective operations
 - File I/O
 - C, C++, Fortran-99 & Fortran-90 language

Download v2.0

- ❖ Intel MPI : Part of the cluster Tools (free eval.)

www.intel.com/software/products/cluster

Intel MPI Library : MPI-2 Features

Device

- ❖ Selected at runtime
 - Shm (**shared memory only**)
 - sock (**Sockets only**)
 - ssm (Shared memory + sockets)
 - rdma (DAPL only) DAPL = Direct Access Programming Library (DAPL*)
 - rdssm (Shared memory + DAPL + sockets)
- Static socket device as fallback

Intel MPI Library : MPI-2 Features

- ❖ Environment variables for runtime control over
 - Process pinning
 - Optimized collective operations
 - Device-specific protocol thresholds
 - Collective algorithm thresholds
 - Enhanced memory registration Cache
 - Platform-specific fine grain timer
 - mpixec process management –
 - mpirun script that automates MPD startup and cleanup

Intel MPI Library : MPI-2

Download v2.0

❖ Intel MPI : Part of the Intel cluster Tools (free eval.)

www.intel.co/software/products/cluster

❖ Intel Cluster Toolkit 2.0 :

- Etnus Total View Debugger <http://www.totalviewtech.com>
(Tool for Multi-threaded applications – Dynamic Memory Management tool, MPI debugging, workbench, Memory Scape)
- Allnea DDT debugger (Distributed Debugging Tool is a comprehensive graphical debugger for scalar, multi-threaded and large-scale parallel applications that are written in C, C++ and Fortran <http://www.allinea.com>

Open MPI : Open Source High Performance Computing

The Open MPI Project is an open source [MPI-2](#) implementation that is developed and maintained by a consortium of academic, research, and industry partners. Open MPI is therefore able to combine the expertise, technologies, and resources from all across the High Performance Computing community in order to build the best MPI library available. Open MPI offers advantages for system and software vendors, application developers and computer science researchers

<http://icl.cs.utk.edu/open-mpi/>

<http://www.mpi-forum.org/>

Part-IX:

An Overview of Multi-Core Processors

Prog.Env. – Compilers

Tuning & Performance

(Part-I)

Gains from tuning categories

<u>Tuning Category</u>	<u>Typical Range of Gain</u>
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%

Source : Reference [4],[6], [7], [8], [32], 34]

Loop Optimization Techniques

- ❖ Dependence Analysis
- ❖ Transformation Techniques
- ❖ Loop distribution
- ❖ Loop Alignment
- ❖ Node Splitting
- ❖ Strip Mining
- ❖ Loop Collapsing
- ❖ Loop Fission Loop Fusion
- ❖ Wave front method
- ❖ Loop Optimizations

More about Loop Optimizations

- ❖ Basic Loop Unrolling & Qualifying Candidates for Loop Unrolling
- ❖ Negatives of Loop Unrolling
- ❖ Outer and Inner Loop Unrolling
- ❖ Associative Transformations
- ❖ Loop Interchange

Source : Reference [4],[6], [7], [8], [32], 34]

Loop Optimizations : Basic Loop Unrolling

- ❖ Loop optimizations accomplish three things :
 - Reduce loop overhead
 - Increase Parallelism
 - Improve memory performance patterns
- ❖ Understanding your tools and how they work is critical for using them with peak effectiveness. For performance, a compiler is your best friend.
- ❖ Loop unrolling is performing multiple loop iterations per pass.
- ❖ Loop unrolling is one of the most important optimizations that can be done on a pipelined machine.
- ❖ Loop unrolling helps performance because it fattens up a loop with calculations that can be done in parallel
- ❖ **Remark** : Never unroll an inner loop.

Qualifying Candidates for Loop Unrolling

- ❖ The previous example is an ideal candidate for loop unrolling.
- ❖ Study categories of loops that are generally not prime candidates for unrolling.
 - Loops with low trip counts
 - Fat loops
 - Loops containing branches
 - Recursive loops
 - Vector reductions
- ❖ To be effective, loop unrolling requires that there be a fairly large number of iterations in the original loop.
- ❖ When a trip count in loop is low, the preconditioning loop is doing proportionally large amount of work.

Qualifying candidates for Loop Unrolling

(Contd..)

❖ Loop containing procedure calls

Loop containing subroutine or function calls generally are not good candidates for unrolling.

- **First** : They often contain a fair number of instructions already. The function call can cancel many more instructions.
- **Second** : When the calling routine and the subroutine are compiled separately, it is impossible for the compiler to intermix instructions.
- **Last** : Function call overhead is expensive. Registers have to be saved, argument lists have to be prepared. The time spent calling and returning from a subroutine can be much greater than that of the loop overhead.

Source : Reference [4],[6], [7], [8], [32], 34]

Loop Unrolling Issues on Multi-Cores

- ❖ Loop unrolling always adds some run time to the program.
- ❖ If you unroll a loop and see the performance dip little, you can assume that either:
 - The loop wasn't a good candidate for unrolling in the first place or
 - A secondary effort absorbed your performance increase.
- ❖ Other possible reasons
 - Unrolling by the wrong factor –Data Race Conditions
 - Register spitting
 - Instruction cache miss – False Sharing of Data
 - Other hardware delays -
 - Outer loop unrolling - Data Re-Use in the Caches of Multi Cores

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

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 Multi-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 Options for Performance

❖ 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.

Compiler Options for Performance

- ❖ Improve usage of data cache, TLB
- ❖ Use VIS instructions (templates) directly, via –xvis option
- ❖ Optimize data alignment 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

Source : Reference [4],[6], [7], [8], [32], 34]

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

Compiler Options for Performance

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 get 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

Multi-Core Complier Optimizations flags

- 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.

Multi-Core 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++
All Above Platforms	gcc -pthread	GNU C
	g++ -pthread	GNU C++
	guidec -pthread	KAIC (if installed)
	kcc -pthread	KAIC++ (if installed)

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.

Tuning & Performance with Compilers

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**

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_fbdta
 - **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

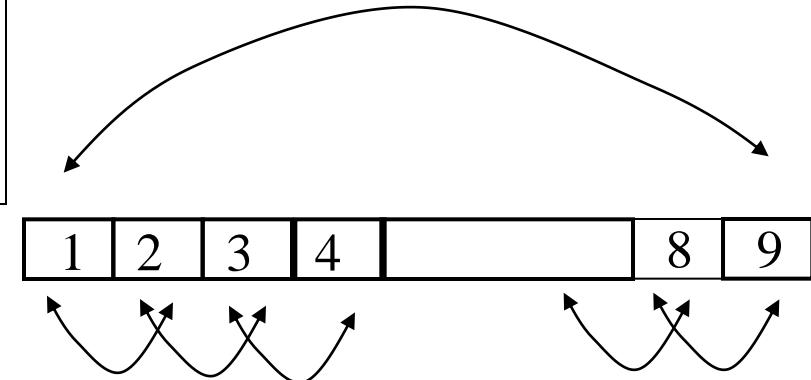
F90: -Ofast -OPT:fast_math=on -WOPT;if_conv=off -LNO:fu=9:full_unroll_size=7000

<http://www.pathscale.com/>

Loop Optimization: Neighbor Data Dependency

Example: data wrap around, untuned version

```
jwrap = ARRAY_SIZE -1;  
for(i=0; i<ARRAY_SIZE; i++)  
    b[i] =(a[i]+a[jwrap])*0.5;  
    jwrap = i; }
```



- ❖ Compiler optimizations will not be able to determine that `a[jwrap]` is a neighbor value

Example: data wrap around, tuned version:

```
b[0] = (a[0] + a[ARRAY_SIZE -1]) * 0.5;  
for(i=1 ; i < ARRAY_SIZE ; i++)  
    b[i] = (a[i]+a[i-1]) * 0.5;
```

Remark : Once the program is debugged, declare arrays to exact sizes whenever possible. This reduces memory use and also optimizes pipelining and cache utilization.

Part-IX:

An Overview of Multi-Core Processors

Prog.Env. – Compilers

Tuning & Performance

(Part-II)

Tuning & Performance on Multi-Core Processors

Algorithm Overheads

- ❖ Data parallelism;
 - ❖ Task parallelism;
 - ❖ Combination of Data and Task parallelism
 - ❖ Static and Load Balancing
 - Mapping for load balancing
 - Minimizing Interaction
 - Overheads in parallel algorithms design
 - ❖ Data Sharing Overheads
- Decomposition techniques**
- ❖ Recursive decomposition
 - ❖ Data decomposition
 - ❖ Exploratory decomposition
 - ❖ Hybrid decomposition

Source : Reference :[1], [4]

Tuning & Performance on Multi-Core Processors

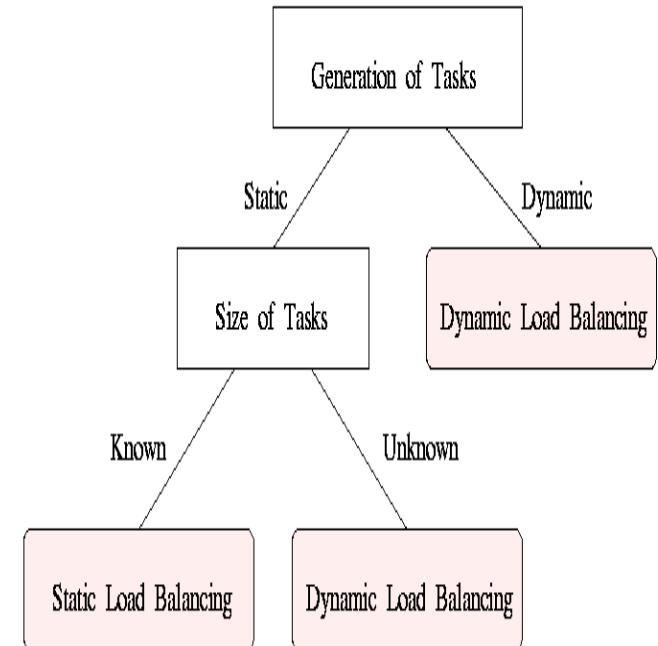
Load Balancing Techniques

❖ Static load-balancing

- Distribute the work among processors prior to the execution of the algorithm
- Matrix-Matrix Computation
- Easy to design and implement

❖ Dynamic load-balancing

- Distribute the work among processors during the execution of the algorithm
- Algorithms that require dynamic load-balancing are somewhat more complicated (Parallel Graph Partitioning)



System Configuration

- ❖ System configuration (OS, Compilers, Profilers)
- ❖ Interconnection Net-work – Point-point, One-all & All-to-All communications (Buffer Mechanism)
- ❖ Disk I/O Computations (Tuning II/O Controller)
 - Hardware; Processor Choice; Interconnect Choice
 - Network Card Choice (in some cases); IO Subsystem
 - Software; Compiler Choice; Compiler Options
 - MPI Choice; MPI Tuning; NIC Driver Tuning
 - Switch Tuning; OS Choice
- ❖ Sequential code Optimization (Compiler – Code Restructuring, Loop Optimization, Memory Allocators)

Tuning & Performance on Multi-Core Processors

Simple Approach

- ❖ Use Compiler Switches & explore performance – Localize the Data – Cache Utilization
- ❖ Use Profiler to understand behavior of programme
- ❖ Use Linux tool “top” to know about the CPU & Memory Utilization as well as Scalability with respect to varying size
- ❖ Threading APIs used; Locks & Heap contention
- ❖ Thread affinity – Explore the performance
- ❖ Sequential code Optimization – Use tuned libraries
- ❖ Check for Swapping (is the code Swapping ?) – Use “top” tools

Tuning & Performance on Multi-Core Processors

Comparing compiler at various levels of Optimization

- ❖ Compiler vendors, sometimes include aggressive optimization at a lower level (For example -O2 may include some optimizations that other compiler vendors put in at -O3)
- ❖ Difficult to compare the same optimization levels among Compilers

CPU /Compiler	-O2	-O3	Aggressive	Tuning
AMD/PGI				
AMD PathScale				
Intel /PGI				
Intel Software				

- ❖ AbSoft
- ❖ Gcc

- ❖ Increasing the level of optimization doesn't improve the performance of the code.

Source : Reference : PGI

Tuning & Performance on Multi-Core Processors

Multi-Core - Types of Data Provided O/S tools

- ❖ CPU Utilization : - by privilege level, on each logical processors and the total
- ❖ Memory Usage : Physical, virtual and page file
- ❖ Network traffic: bytes in and out, errors, broadcasts, etc., OS socket usage
- ❖ Disk Traffic : reads/writes, bytes read/written, merged IO requested, average wait time, queue depth
- ❖ Process Information – started processes, context switches /sec, scheduler queue depth
- ❖ And more !

Source : Reference : PGI

Tuning & Performance on Multi-Core Processors

Multi-Core Sub-system – Details

Intel Compiler Optimization Switches

- ❖ Intel High-Level Optimizations (HLO)
- ❖ Intel Multiphase Optimizations
 - (IPO – Interprocedural Optimizations)
 - PGO –(Profile Guided Optimization) Switches
- ❖ Intel Math Kernel Library (MKL)
- ❖ Intel Integrated Performance Primitives

Source : Reference : PGI

Tuning & Performance on Multi-Core Processors

General Optimizations (Refer PGI Compiler suite)

Linux	Windows	-
-O0	/Od	Disables optimization
-g	/Z1	Creates symbols
-O1	/O1	Optimize binary Code for Server Size
-O2	/O2	Optimize for Speed (default)
-O3	/O3	Optimize for Data Cache : Loop floating point code
-axP	QaxP	Optimize for Intel Processors with SSE 3 Capabilities.
-parallel	-Qparallel	Auto-parallelization

Source : Reference : PGI

Tuning & Performance on Multi-Core Processor Clusters

Multi-Core Computing Systems

- ❖ Intel
- ❖ Cray
- ❖ IBM - Cell
- ❖ AMD
- ❖ SGI
- ❖ SUN
- ❖ HP

Tuning & Performance on Multi-Core Processors

Multi-Core Sub-system – Details

- ❖ CPU Utilization; Memory Usage; Network traffic, Disk Usage, Process Information
- ❖ Intel Compiler Optimization Switches
- ❖ Intel High-Level Optimizations (HLO)
- ❖ Intel Multiphase Optimizations (IPC, PGO)
- ❖ Intel Multiphase Optimizations (PGO – Profile Guided Optimization Switches)
- ❖ Intel Math Kernel Library (MKL)
- ❖ Intel Integrated Performance Primitives

Source : Reference : Intel PGI

Tuning & Performance on Multi-Core Processors

Multi-Core Intel Compiler – Starting Steps

Explore using Intel Compiler

- ❖ Optimization switches – provides a way to use Intel's new instructions/technologies
 - Vectorization – compiler switches + compiler directive allow certain loops to be parallelized via instruction parallelism (SIMD instructions)
- ❖ Multipass Optimizations (IPO, PGO) – provides a way to tune across function/files and use actual execution feedback to guide compiler optimizations.
- ❖ Explore using libraries that have already tuned common operations /functions
 - Intel's Math Kernel Library (MKL) – math functions
 - Intel's Integrated Performance Primitives (IPP), graphic media functions

Tuning & Performance on Multi-Core Processors

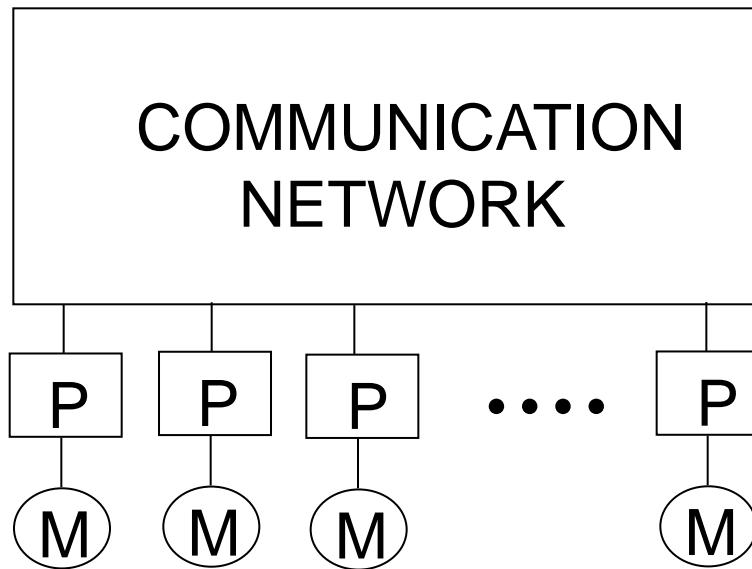
Multi-Core Intel Compiler – Starting Steps

Explore using Intel Compiler

- ❖ Explore tuning how memory is used if critical to application
- ❖ Understand object creation/destruction – objects be reused ?
- ❖ Loops : Understand how memory is accessed (patterns and alignment)
 - Many different memory tuing techniques can be applies depending on type of performance issue
 - Discussed in Addressing Common Performance Section
- ❖ Data organization/structures : Is data being handled optimally
 - Remove loop invariant code from hotspots
- ❖ Do only what is absolutely necessary in hotspots
 - Use Intel's Math Kernel Library (MKL) – math functions

Message Passing Architecture Model

Message-Passing Programming Paradigm : Processors are connected using a message passing interconnection network.



- ❖ On most Parallel Systems, the processes involved in the execution of a parallel program are identified by a sequence of non-negative integers. If there are p processes executing a program, they will have ranks 0, 1, 2, ..., $p-1$.

Tuning & Performance on Multi-Core Processor Clusters

Interconnection Networks – Latency Bandwidth

- ❖ **TCP**
 - GiGE,
 - GigE with Jumbo Frames,
 - GAMMA, Level 5 GigE
 - 10 Gigabit Ethernet
 - 100 gigabit Ethernet
- ❖ **Myrinet**
- ❖ **Infiniband**
- ❖ **PathScale (Infinipath)**
- ❖ **Quadrics**
- ❖ **Dolphin**
- I/O time for Codes (CFD /Seismic Codes)

Tuning & Performance on Multi-Core Processor Clusters

MPI Libraries (Open Source)

- ❖ Open-source
- ❖ **MPICH1 (Standard)**
- ❖ **MPICH2 (Standard)**
- ❖ LAM
- ❖ Open-MPI (*)
- ❖ GAMMA-MPI
- ❖ FT-MPI
- ❖ LA-MPI
- ❖ LA-MPI
- ❖ PACX-MPI
- ❖ MVAPICH
- ❖ OOMPI
- ❖ MPICH-GM
- ❖ MVICH
- ❖ MP_Lite

(*) OpenMPI combines the best features of LAM, FT-MPI, LA-

MPI, and PACX-MPI. It supports TCP, Myrinet, Infiniband networks. <http://icl.cs.utk.edu/open-mpi/>

- ❖ OpenMPI has addition of the fault tolerance capability of FT-MPI.
- ❖ This will allow an MPI code to lose a node and then add a new node to finish the computations without lose of data.

Tuning & Performance on Multi-Core Processor Clusters

Test the following MPI Libraries (Open Source)

- ❖ Open-source
 - ❖ **MPICH1 (Standard)**
 - ❖ **MPICH2 (Standard)**
 - ❖ LAM
 - ❖ MPICH-GM (for Myrinet)
 - ❖ MVAPICH (for Infiniband)
 - ❖ Atleast one commercial MPI
-
- ❖ Choice of Compilers - Computing systems – Price / Performance
 - ❖ Choice of MPI libraries Computing Systems – Price / Performance

Tuning & Performance on Multi-Core Processor Clusters

Interconnection Networks – Latency Bandwidth

- ❖ **TCP**
 - GiGE,
 - GigE with Jumbo Frames,
 - GAMMA, Level 5 GigE
 - 10 Gigabit Ethernet
 - 100 gigabit Ethernet
- ❖ **Myrinet**
- ❖ **Infiniband**
- ❖ **PathScale (Infinipath)**
- ❖ **Quadrics**
- ❖ **Dolphin**
- I/O time for Codes (CFD /Seismic Codes)

Tuning & Performance on Multi-Core Processor Clusters

❖ MPI Library calls & MPI algorithms implementation

- MPI Point-to-Point communication Calls;
- MPI collective Communications,
- MPI Communication & Computation Library Calls
- MPI-2 library Calls
- MPI I/O library Calls

❖ Communication overheads for all MPI library calls

❖ MPI 3.0 Thread enabled MPI

MPI Point-to-Point Communication: Communication Modes

MPI Primitive	Blocking	Nonblocking
Standard Send	<code>MPI_Send</code>	<code>MPI_Isend</code>
Synchronous Send	<code>MPI_Ssend</code>	<code>MPI_Issend</code>
Buffered Send	<code>MPI_Bsend</code>	<code>MPI_Ibsend</code>
Ready Send	<code>MPI_Rsend</code>	<code>MPI_Irsend</code>
Receive	<code>MPI_Recv</code>	<code>MPI_Irecv</code>
Completion Check	<code>MPI_Wait</code>	<code>MPI_Test</code>

Different Send/Receive operations in MPI

MPI Collective Communications

Type	Routine	Functionality
Data Movement	MPI_Bcast	One-to-all, Identical Message
	MPI_Gather	All-to-One, Personalized messages
	MPI_Gatherv	A generalization of MPI_Gather
	MPI_Allgather	A generalization of MPI_Gather
	MPI_Allgatherv	A generalization of MPI_Allgather
	MPI_Scatter	One-to-all Personalized messages
	MPI_Scatterv	A generalization of MPI_Scatter
	MPI_Alltoall	All-to-All, personalized message
	MPI_Scatterv	A generalization of MPI_Alltoall

Tuning & Performance on Multi-Core Processor Clusters

- ❖ Commercial Code tuning
 - File System
 - Network (TCP Buffers NIC, Switches)
 - L1 Cache /L2 Cache on Multi-Core Processors
 - Compiler Switches
 - MPI Libraries
 - OS & Memory Usage (80 %)
- ❖ Scalability of computing Systems & Performance Issues are Challenging

Tuning & Performance on Multi-Core Processor Clusters

Thread Safety & MPI – Issues to be addressed

- ❖ Performance tradeoffs between multi-threaded and single-threaded code.
 - I/O operations
 - Against Inconsistent updates to the same memory location from different threads
 - Software locks and System locks are quite expensive
- ❖ Vendors sometimes provide single threaded /Multi-threaded libraries
 - Have I been linked with the right library ?
 - DO I suffer with occasional and mysterious errors

Source : Reference : [4], [6], [11],[12],[24],[25], [26]

Part-X:

An Overview of Multi-Core Processors

Benchmarks

Performance Characteristics

Approaches to measure performance

- ❖ Several Approaches exist to measure performance of a Multicore System
 - Summarize several key architecture of a given computer system and relate them in order to get measure of its performance
 - Most of the measures are based on some engineering or design considerations rather theoretical calculations
 - Define set of programs and observe the system's run times on those programs

Performance Characteristics: Peak Performance

Peak Performance

- ❖ Defined as the MFLOPS rate which the manufacturer guarantees the computer will never exceed
 - It is obtained by taking the clock rate of the given system and dividing it by the number of clock cycles a floating point instruction requires
- ❖ Peak Performance calculations assume the maximum number of operations that the hardware can execute in parallel or concurrently
- ❖ Peak Performance is a rough hardware measure; it essentially reflects the cost of the system
- ❖ There are some (rare) instances where peak performance can give a creditable idea of performance

Performance Characteristics: Sustained Performance

Sustained Performance

- ❖ It may be defined as the highest MFLOPS rate that can actual program achieved doing something recognizably useful for certain length of time
- ❖ It essentially provide an upper bound on what a programmer may be able to achieve

Efficiency rate = The achieved (sustained) performance divided by the peak performance

Note : The advantage of this number is its independent of any absolute speed.

Performance: Benchmarks Classification

Benchmark Classification

- ❖ Benchmarks can be classified according to applications
 - Scientific Computing
 - Commercial applications
 - Network services
 - Multi media applications
 - Signal processing
- ❖ Benchmark can also be classified as
 - Macro benchmarks and Micro benchmarks

Macro Benchmarks

- ❖ Measures the performance of computer system as a whole.
- ❖ Compares different systems with respect to an application class, and is useful for the system BUYER. However, macro benchmarks do not reveal why a system performs well or badly.

Name	Area
NAS	Parallel Computing (CFD)
PARKBENCH	Parallel Computing
SPEC	A mixed benchmark family
Splash	Parallel Computing
STAP	Signal Processing
TPC	Commercial Applications

Performance: Micro Benchmarks

(Contd...)

Micro Benchmarks : Synthetic kernels & measure a specific aspect of computer system (CPU speed, Memory speed, I/O speed, Operating system performance, Networking)

Name	Area
LAPACK; ScaLAPACK; LINPACK; BLASBench; HPCC suite Benchmarks	Numerical Computing (Linear Algebra)
LMBENCH	System Calls and data movement operations in UNIX
STREAM	Memory Bandwidth

Source : <http://www.netlib.org>

Benchmarks on Multi Core Systems

(Contd...)

Micro/Macro Benchmarks for Multi Core Processors

Name	Area
SuperLU	Numerical Computing (Linear Algebra)
HPCC Suite (Top-500)	
LMBENCH	System Calls & Data movement in Unix/Linux Environment
LLCBench	Low Level Cache Benchmarks
STREAM	Memory Bandwidth
I/O -Bench	I/O Benchmarks
TIO-Bench	Thread I/O Benchmarks
NAMD	Nanoscale Molecular Dynamics
NAS	Computational Fluid Dynamics

Benchmarks on Multi Core Systems

LLCBench

- **BLASBench** : equivalent to HPCC-DGEMM. DGEMM (double precision Matrix into Matrix Multiplication) achieved performance of 4.7 Gflops, which is equivalent to 90 % of the peak performance (5.2 Gflops)
- **MpBench** – MPI benchmarks executed (Equivalent version of HPCC-HLRS Suites available)
- **CacheBench** – Low level Cache Benchmarks executed

Micro Benchmarks: STREAM

- ❖ The Stream is a simple synthetic benchmark maintained by John McCalpin of SGI.
 - It measures sustainable memory bandwidth (in MBs) and the corresponding computation rate.
 - The motivation for developing the STREAM benchmark is that processors are getting faster more quickly than memory, and more programs will be limited in performance by the memory bandwidth, rather than by the processor speed.
 - The benchmark is designed to work with data sets much larger than the available cache
 - The STREAM Benchmark performs four operations for number of iterations with unit stride access.

EEMC Benchmarks

- ❖ **EEMBC**, the Embedded Microprocessor Benchmark Consortium,
 - It is a non-profit corporation formed to standardize on real-world, embedded benchmark software to help designers select the right embedded processors for their systems.
- ❖ **EEMBC** is a collection of "algorithms" and "applications" organized into benchmark suites targeting telecommunications, networking, digital media, Java, automotive/industrial, consumer, and office equipment products.

Source : <http://www.eembc.org/>

EEMC Benchmarks

- ❖ **EEMBC**, the Embedded Microprocessor Benchmark Consortium,
- ❖ An additional suite of benchmarks, called MultiBench, specifically targets the capabilities of multicore processors based on an SMP architecture.
- ❖ These benchmarks may be obtained by joining EEMBC's open membership or through a corporate or university licensing program.
- ❖ The EEMBC Technology Center manages development of new benchmark software and certifies benchmark test results.

Source : <http://www.eembc.org/>

EEMC Benchmarks

❖ Benchmark Scores

- Automotive Consumer Digital Entertainment Java/CLDC

❖ Software Licensing and Membership

- AutoBench, ConsumerBench, DENBench, GrinderBench (Java)

❖ Hypervisors

- MultiBench, Networking, OABench, TeleBench, HyperBench

❖ Power/Energy

- EnergyBench

Source : <http://www.eembc.org/>

EEMC Benchmarks

❖ MultiBench™ 1.0 Multicore Benchmark Software

- Extends EEMBC benchmark scope to analyze multicore architectures, memory bottlenecks, OS scheduling support, efficiency of synchronization, and other related system functions.
- Measures the impact of parallelization and scalability across both data processing and computationally intensive tasks
- Provides an analytical tool for optimizing programs for a specific processor
- Leverages EEMBC's industry-standard, application-focused benchmarks in hundreds of workload combinations
- First generation targets the evaluation and future development of scalable SMP architectures
- MultiBench™ is a suite of embedded benchmarks that allows processor and system designers to analyze, test, and improve multicore architectures and platforms. MultiBench uses standardized workloads and a test harness that provides compatibility with a wide variety of multicore embedded processors and operating systems.

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, PGI Compilers to get performance on AMD64-based Linux Systems.

Source : Reference : PGI

<http://www.pathscale.com/>

Performance of Math Kernel Libraries

- ❖ Compiler Optimization
 - ❖ CPU Usage : Floating Point Optimization & Register Use
 - ❖ Cache : Data Locality – Cache Interleaving
 - ❖ TLB : Maximum use the data on page /page swapping
 - ❖ Memory bandwidth – minimize the access the higher levels of memory
 - ❖ Core : Use all the cores available using threads & address the thread affinity issues.
-
- Atlas
 - Intel MKL www.intel.com/software/products/mkl

Intel Performance Libraries - Intel Math Kernel Library (Intel MKL)

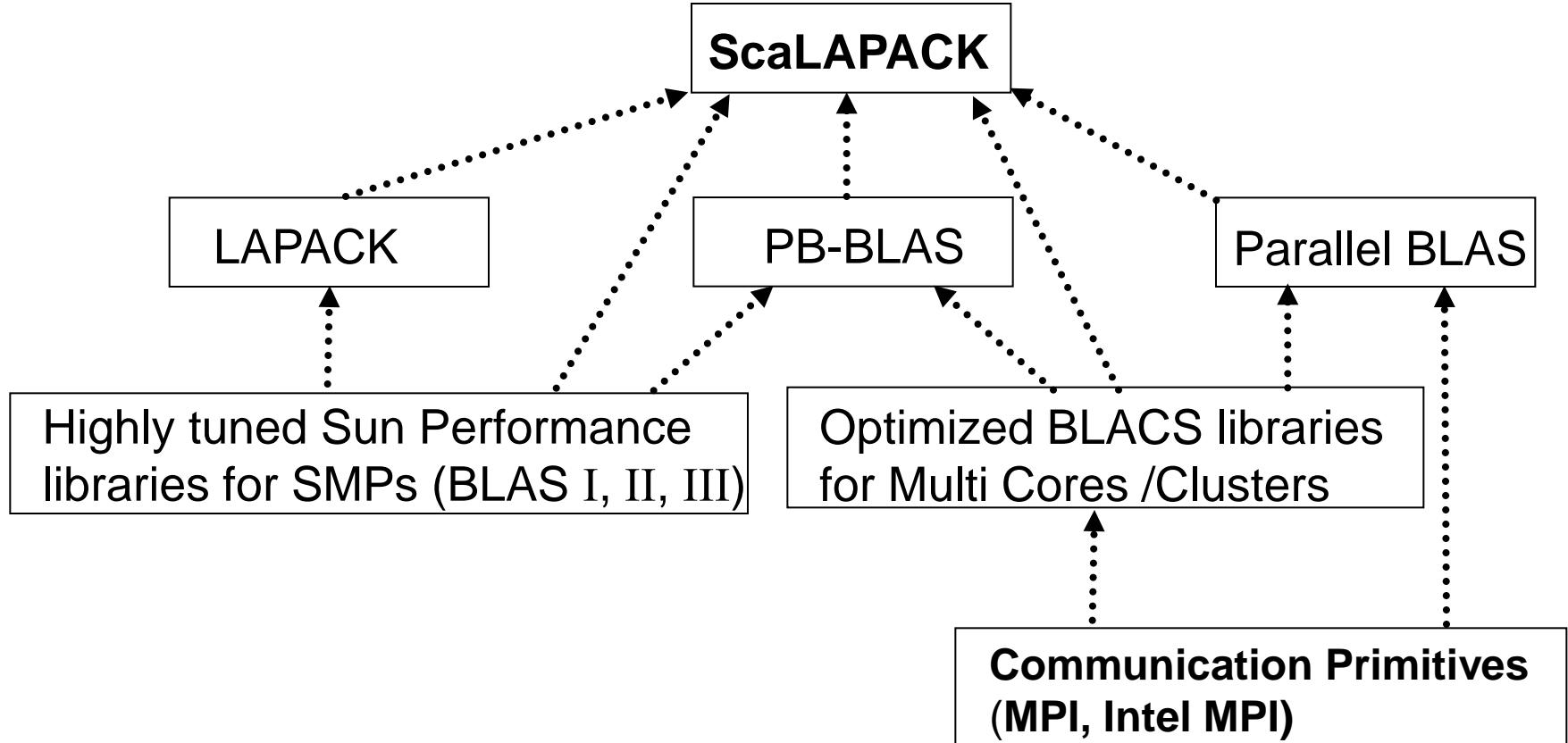
- ❖ Intel 's Engineering, Scientific and financial mathematical Library Intel MKL www.intel.com/software/products/mkl
 - Solvers (BLAS, LINPACK)
 - Eigenvector/Eigenvalue solver (BLAS, LINPACK)
 - Some Quantum Chemistry – (dgemm Matrix library)
 - PDEs, Signal processing, Seismic, Solid State Physics (FFTs)
 - General scientific, financial [vector transcendental functions (VML), and vector random number generators (VSL)]
 - Sparse Solvers (PARDISO, DSS * ISS)

Intel Performance Libraries - Intel Math Kernel Library (Intel MKL)

- ❖ Intel 's Engineering, Scientific and financial mathematical Library
- ❖ **Address :**
 - Solvers (BLAS, LINPACK)
 - Eigenvector/Eigenvalue solver (BLAS, LINPACK)
 - some Quantum Chemistry – (dgemm Matrix library)
 - PDEs, Signal processing, Seismic, solid State Physics (FFTs)
 - General scientific, financial [vector transcendental functions (VML), and vector random number generators (VSL)]

Micro Benchmarks: ScaLAPACK

(Contd...)



Source : <http://www.netlib.org>

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.

Linear Algebra (LA)

- ❖ Basic Linear Algebra Subroutines (BLAS)
 - Level 1 (vector-vector operations)
 - Level 2 (matrix-vector operations)
 - Level 3 (matrix-matrix operations)
 - Routines involving sparse vectors
- ❖ Linear Algebra PACKage (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

Features

- ❖ BLAS, LAPACK, FFT Performance
- ❖ Open MP Performance
- ❖ **AMD** : ACML 2.5 /2.X or 3.X
- ❖ **Intel** : MKL
- ❖ **IBM** : Power –5 / Power -6 /Power -7 ESSL
- ❖ **How good is Benchmark performance?**

Mathematical Libraries (Benchmarks)

- ❖ ATLAS (Automatically Tuned Linear Algebra Software)
<http://www.netlib.org/atlas>
- ❖ GoToBLAS :The GOTO library is an optimised implementation of BLAS routines, developed by Kazushige Goto at the University of Texas at Austin, providing an alternative to MKL or SCSL
- ❖ SCSL : Scientific Computing Software Library (is a comprehensive collection of scientific and mathematical functions)
- ❖ PLASMA : PLASMA is to address the performance shortcomings of the **LAPACK** and **ScaLAPACK** libraries on multicore processors and multi-socket systems of multicore processors
- ❖ PLASMA : Parallel Linear Algebra Software for Multi-Core Architecture
Ref: <http://icl.cs.utk.edu/plasma>

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

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.

Dual Core System : Configuration & Prog. Env

System Details	Multi Core : IWILL H205	SunFire 4600
CPU	Dual-Core AMD Opteron (tm) Processor 8218	Dual-Core AMD Opteron(tm) Processor 885
No Sockets/Cores	4 Sockets (Total : 8 Cores)	8 Sockets (Total : 16 Cores)
Clock-Speed	2.6 GHz per core	2.6 GHz per core
Peak(Perf.)	41.6 Gflops	83.2 Gflops
Memory/Core	1 GB per core	4 GB per core
Memory type	DDR2	DDR2
Total Memory	8 GB	64 GB
Cache	L1 = 128 KB; L2 = 1 MB	L1 = 128 KB; L2 = 1 MB
OS	Cent OS 4.4 x86_64 (64 bit)	CentOS 4.4 (Final) x86_64 (64 bit)
Compilers	Intel 9.1(icc; fce; OpenMP)	Intel 9.1(icc; fce; OpenMP)
MPI	mpicc:Intel MPI 2 .0 7 gcc/gfortran mpiicc : Intel MPI 2.0 /icc, ifort	mpicc:Intel MPI 2 .0 7 gcc/gfortran mpiicc : Intel MPI 2.0 /icc, ifort
Math Libraries	ACML 3.5.0	ACML 3.5.0

Dual Core Processors : Configuration & Prog. Env

System Details	Multi Core : HP DL485	DELL PowerEdge 6950
CPU	Dual-Core AMD Opteron (tm) Processor 8200 SE	Dual-Core AMD Opteron™ Processor 8218
No of Sockets/Cores	4 Sockets (total 8 Cores)	4 Sockets (total 8 Cores)
Clock-Speed	2.8 GHz per core	2.6 GHz per core
Peak Performance	44.8 Gflops	41.2 Gflops
Memory/Core	2 GB per core	2 GB per core
Memory type	DDR2 667 MHz	DDR2 667 MHz
Total Memory	16 GB	16 GB
Cache	L1 = 64 KB; L2 = 1 MB	L1 =128 KB; L2 = 2 MB
OS	Cent OS 4.4 x86_64 (64 bit)	Cent OS 4.4 x86_64 (64 bit)
Compilers	Intel 9.1(icc; fce; OpenMP)	Intel 9.1(icc; fce; OpenMP)
MPI	mpicc:Intel MPI 2 .0 7 gcc/gfortran mpiicc : Intel MPI 2.0 /icc, ifort	mpicc:Intel MPI 2 .0 7 gcc/gfortran mpiicc : Intel MPI 2.0 /icc, ifort
Math Libraries	ACML 3.5.0	ACML 3.5.0

Dual Core Processors - Prog. Env & Benchmark Rules

Rule 1 :The same Prog. environment is used for execution of Benchmark

Rule 2 :Process (Job) binding to particular CPU(s) is not considered

Rule 3 :Algorithmic parameters are not FULLY optimised

Computing System	Programming Environment Details	
IWILL SunFire HP DL485 DELL	Operating System	Cent OS 4.4 x86_64 (64 bit), Kernel 2.6.9
	Compilers	Intel 9.1.(icc; fce; OpenMP)
	Flags	- O3, -ip -funroll-loops,
	MPI	Intel MPI 2.0
	Libraries	ACML 3.5.0

Remarks 1 :For selective Benchmark suites (HPCC), the input problem parameters are same on the target systems. Tuning & Performance Optimisation is not carried out

Dual Cores: HPCC – (Top-500) : Results & Performance

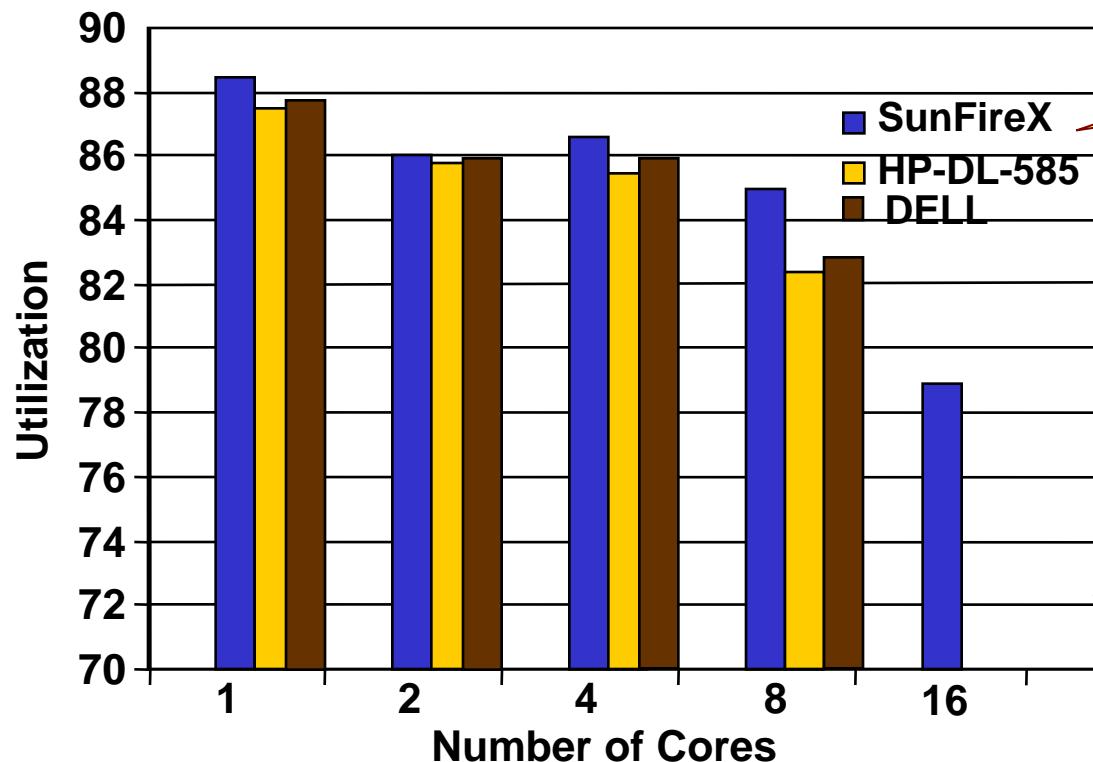
Comp. System	Multi Core (CPUs)	Matrix Size/ Block size/ (P,Q)	Peak Perf (Gflops)	Sust. Perf (Gflops)	Utilization (%)
--------------	-------------------	--------------------------------	--------------------	---------------------	-----------------

SunFireX	1	46080/128(1,1)	5.2	4.60	88.46
	2	46080/128(2,1)	10.4	8.94	86
	4	52000/160(2,2)	20.8	18.01	86.58
	8	56320/128(4,2)	41.6	35.37	85.02
	16	72000/192(8,2)	83.2	65.66	78.91

HP-DL585	1	24000/160(1,1)	5.6	4.95	87.5
	2	38400/160(2,1)	11.2	9.6	85.8
	4	38400/160(2,2)	22.4	19.2	85.5
	8	40448/160(4,2)	44.8	36.9	82.4

Algorithm parameters, tuning & performance Compiler optimisations are not tried to extract the sustained performance.

Dual Cores: HPCC – (Top-500) : Results & Performance



Algorithm parameters, tuning & performance of Compiler optimisations are not tried to extract the sustained performance.

Minimum and Maximum Performance differ by 2-4 %

Choice of HPL Problem Size parameters play an important role

Issues to be addressed for Performance Enhancements

- Memory per Core
- L1 Cache
- Tuned Compilers
- Mathematical Kernels
- Operative System
- Process Affinity

Dual Cores: HPCC –Top 500 : Performance

Computing System	Multi Core (CPUs)	Matrix Size/ Block size/ (P,Q)	Peak Perf (Gflops)	Sust. Perf (Gflops)	Utilization (%)
IWILL	1	25600/128(1,1)	5.2	4.498	86.5
	2	25600/128/(2,1)	10.4	8.76	84.2
	4	25600 /120(4,1)	20.8	17.1	82.1
	8	30208/128(8,1)	41.6	31.7	76.7
SunFireX	1	25600/128(1,1)	5.2	4.60	88.5
	2	25600/128/ (2,1)	10.4	8.799	84.6
	4	25600 /120(4,1)	20.8	17.3	83.2
	8	30208/128(8,1)	41.6	32.0	76.9
HP-DL585	1	25600/128(1,1)	5.6	4.86	86.87
	2	25600/128 (2,1)	11.2	9.286	82.91
	4	25600 /120(4,1)	22.4	18.43	82.27
	8	30208/128(8,1)	44.8	33.84	75.54
DELL PowerEdge 6950	1	25600/128(1,1)	5.2	4.57	87.88
	2	25600/128 (2,1)	10.4	8.84	85.00
	4	25600 /120(4,1)	20.8	17.15	82.45
	8	30208/128(8,1)	41.6	29.6	71.15

For Top-500, algorithm parameters, tuning & performance of Compiler optimisations are not tried to extract the sustained Performance. Input Parameters on IWILL /SUNFIRE /HP DL585 /DELL are precisely the same.

Part-XI:
An Overview of Multi-Core Processors
Prog.Env. - Software tools Overview

AMD Software Products on Multi-core Processors

Visit <http://developer.amd.com/cpu/Pages/default.aspx>

- ❖ AMD Code Analyst Performance Analyzer for Linux
(Profiling & pipeline simulation;
timer-based, event-based, & thread profiling)
- ❖ AMD PMU Extension Driver
- ❖ AMD Performance Libraries
 - AMD Core Math Library (ACML)
 - A set of C/C++ and Fortran algorithms, focusing on Basic Linear Algebra (BLAS), Linear Algebra (LAPACK), Fast Fourier Transforms (FFTs), and other math functions tuned for 32-bit and 64-bit performance on Opteron processors.
 - ACML – GPU
 - AMD String Library; SSEPlus Project; AMD LibM
- ❖ AMD SmNow ; APML Tools; DMTF DASH

AMD Software Products on Multi-core Processors

Visit <http://developer.amd.com/cpu/Pages/default.aspx>

- ❖ GCC & GNU Tools on AMD
- ❖ x86 Open64 compiler Suite
- ❖ GCC & GNU
 - GCC 4.1.2 Downloads
 - GCC 4.2 Downloads
 - ACML – GPU
- ❖ x86 open64 compiler suite
- ❖ AMD Lightweight Profiling Specification

HP Software Products on Multi-core Processors

Visit <http://h20338.www2.hp.com/hpc/us/en/linux-value-pack.html>
<http://h20338.www2.hp.com/HPC/cache/277914-0-0-0-121.html>

- ❖ HP-MPI
- ❖ HP & Platform LSF
- ❖ HP UPC : Universal Parallel compiler
- ❖ HPC Library - Multi-Core Optimization

IBM Prog. Env. on Multi-core Processors

Visit <http://www.ibm.com/developerworks/aix/>

<http://www-03.ibm.com/systems/software/parallel/index.html>

Parallel Environment is a high-function development and execution environment for parallel applications (distributed-memory, message-passing applications running across multiple nodes).

It is designed to help organizations develop, test, debug, tune and run high-performance parallel applications written in C, C++ and Fortran on Power Systems clusters. Parallel Environment runs on AIX® or Linux®.

IBM Prog. Env. on Multi-core Processors

Visit <http://www.ibm.com/developerworks/aix/>
<http://www-03.ibm.com/systems/software/parallel/index.html>

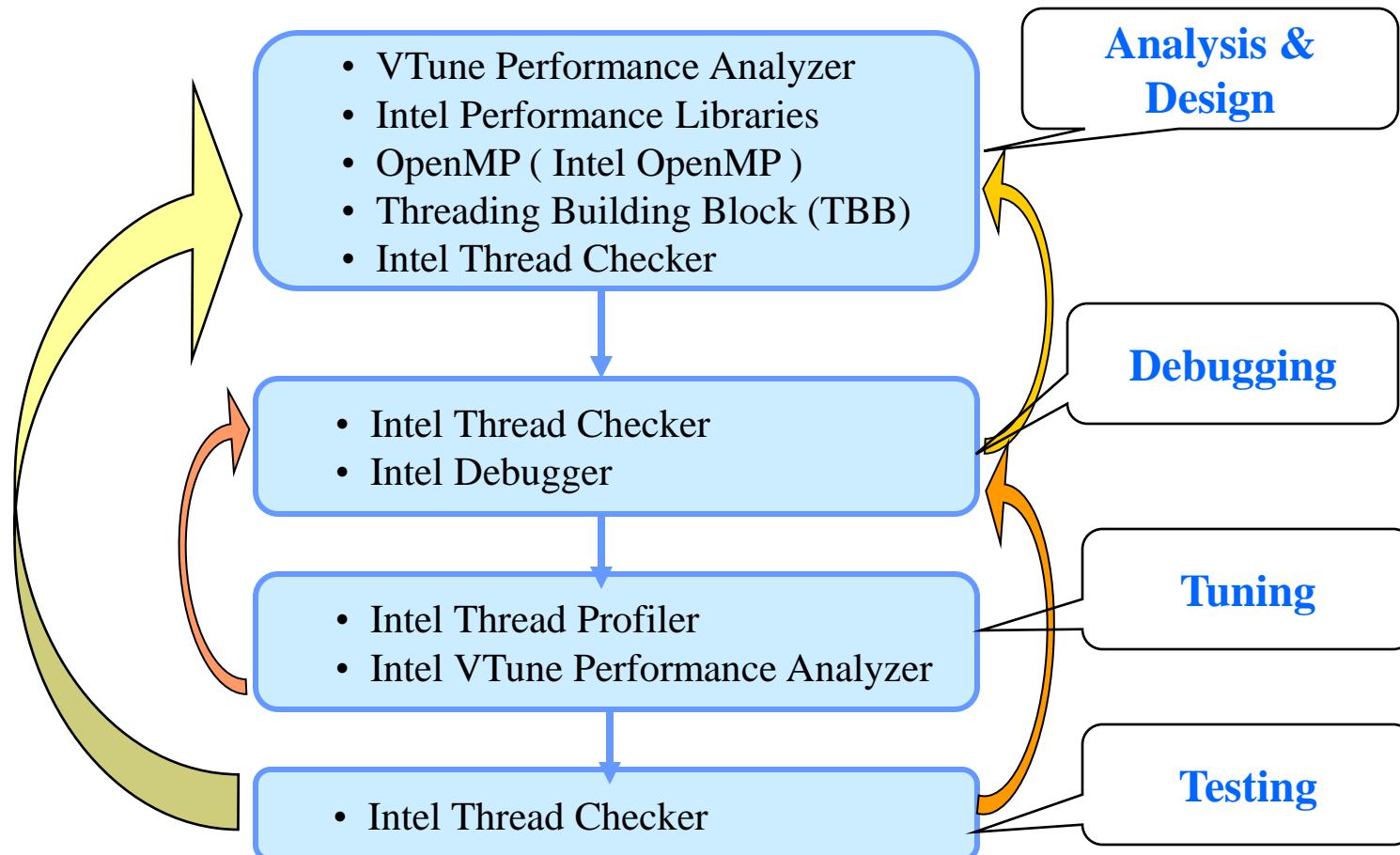
- ❖ Parallel Environment includes the following components:
 - The Parallel Operating Environment (POE) for submitting and managing jobs.
 - IBM's MPI and LAPI libraries for communication between parallel tasks.
 - A parallel debugger (pdb) for debugging parallel programs.
 - IBM High Performance Computing Toolkit for analyzing performance of parallel and serial applications.

Intel Software Products on Multi-core Processors

Visit <http://www.intel.com/software/products>

- ❖ Intel® C++ and FORTRAN Compilers: Generate highly optimized executable code for Intel® 64 and IA-32 processors.
- ❖ Intel® VTune™ Performance Analyzer: Collects & displays Intel architecture specific performance data from system-wide to specific source lines.
- ❖ Intel® Performance Libraries: Consists of set of software libraries optimized for Intel processors. These include the Intel® Math Kernel Library (Intel® MKL) and the Intel Integrated Performance Primitives (Intel® IPP).
- ❖ Intel® Threading Tools: These tools help debug and optimize threaded code performance. They are the Intel® Thread Checker and Thread Profiler.
- ❖ Intel Performance Tuning Utility & several other tools posted on www.whatif.intel.com.

Intel : Performance Improvement Cycle



Sour : <http://www.intel.com/software/products>

Intel Thread Checker : Features

Intel® Thread Checker detects data races, deadlocks, stalls, and other threading issues. It can detect the potential for these errors even if the error does not occur during an analysis session.

- ❖ Detect the potential errors.
- ❖ Filter out specific types of Diagnostics
- ❖ Identify critical source locations
- ❖ Get tips to improve the robustness

Intel Thread Checker : Benefits

Intel® Thread Checker detects data races, deadlocks, stalls, and other threading issues. It can detect the potential for these errors even if the error does not occur during an analysis session.

- ❖ Pinpoint the function, context, line, variable, and call stack in the source code to aid analysis and repair of bugs
- ❖ Identify nearly impossible-to-find data races and deadlocks using an advanced error detection engine. Helps to reduce untraceable errors.
- ❖ Instrumental for effective design of threaded applications
- ❖ Errors do not need to actually occur to be detected. Make the code as more robust

Intel Thread Profiler

Intel® Thread Profiler helps you to improve the performance of applications threaded with Windows API, OpenMP, or POSIX threads (Pthreads).

- ❖ Identify bottlenecks that limit the parallel performance of your multi threaded application.
- ❖ Locate synchronization delays, stalled threads, excessive blocking time, and ineffective utilization of processors.
- ❖ Find the best sections of code to optimize for sequential performance and for threaded performance.
- ❖ Compare scalability across different numbers of processors or using different threading methods.

Intel Vtune Performance Analyzer

The VTune™ Performance Analyzer provides information on the performance characteristics of given code . The VTune analyzer shows you the performance issues, enabling you to focus your tuning effort and get the best performance boost in the least amount of time.

- ❖ Sampling
- ❖ Call graph
- ❖ Counter Monitor
- ❖ Monitor Performance

Intel Vtune Performance Analyzer

- ❖ VTune Performance Analyzer related to threading performance:
 - Improving the efficiency of computation
 - Improving your threading Model
- ❖ Determine what parts of your application (if any) that, when threaded, will speed up your app
 - CPU-bound apps can potentially run twice as fast on dual-core processors
 - Memory bound apps may potentially run 50% faster
 - I/O bound applications may not run any faster threading model

Intel Debugger 11.0 for Linux

- ❖ The Intel® C++ Application Debugger (Intel ® IDB 11. 0) is part of Intel Compiler packages
 - ❖ Command line and GUI Interface is supported
 - ❖ Features
 - Enhanced Thread Awareness
 - Thread Specific Breakpoint Handling
 - SIMD SSE Window
 - OpenMP Windows
 - Thread Control & Thread Synchronization Breakpoint
 - Thread Groupware breakpoint

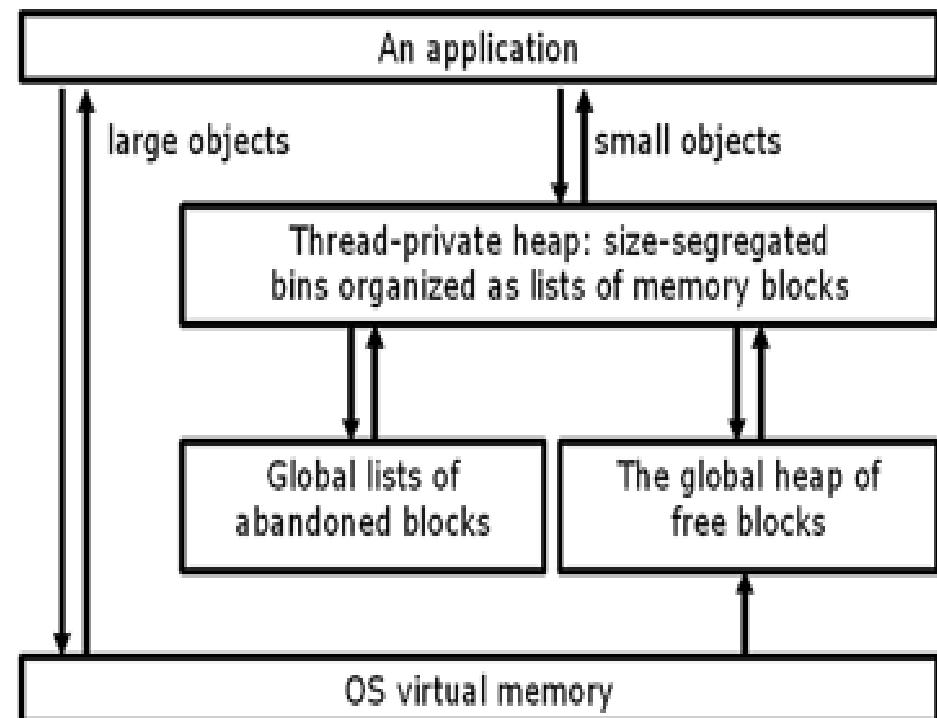
Intel Vtune Performance Analyzer

Dedicated OpenMP Info Windows.

- ❖ Threads
- ❖ Teams
- ❖ Tasks
- ❖ Tasks spawn trees
- ❖ Barriers*
- ❖ Taskwaits
- ❖ Locks

Intel Scalable Memory Allocator

- ❖ The allocator requests memory from the OS in 1MB chunks and divides each chunk into 16K-byte aligned blocks. These blocks are initially placed in the global heap of free blocks.



Debugger tools on Multi-Core Processors

Total View

- ❖ Etnus Total View Debugger <http://www.totalviewtech.com>
(Tool for Multi-threaded applications – Dynamic Memory Management tool, MPI debugging, workbench, Memory Scape)

Allinea DDT Debugger

- ❖ Allinea DDT debugger (Distributed Debugging Tool is a comprehensive graphical debugger for scalar, multi-threaded and large-scale parallel applications that are written in C, C++ and Fortran <http://www.allinea.com>

Performance Profiler tools

- ❖ **Google PerfTool**
- ❖ Multi-threaded applications in C++ with templates. Includes TCMalloc, heap-checker, heap-profiler and cpu-profiler
- ❖ Works with STL
- ❖ Perf Tools is a collection of a high-performance multi-threaded malloc() implementation, and performance analysis tools. <http://code.google.com/p/google-perf-tools>

PerTools :

- ❖ PerTools help one to identify spots in a program that are responsible for CPU consumption.

<http://minos.phy.bnl.gov/~bviren/minos/software/prof/PerfTools/doc/>

HPC Tool Kit

- ❖ Binary-level measurement and analysis
- ❖ Support top down performance analysis
- ❖ Scalability bottlenecks on large scale parallel systems (MPI)
- ❖ Scaling on multi-core processors – Performance analysis of multi-threaded code
- ❖ Combine multiple profiles – multiple threads, multiple processes, multiple executions
- ❖ Visualization – explore performance data from multiple perspectives
- ❖ Path Profiling – hardware counter overflows (Use PAPI)
- ❖ Unwinding Optimized code
 - Use binary analysis
- ❖ Low Overhead of tool
- ❖ Tool Performance for codes having weak/ strong scaling

<http://hpctoolkit.org/documentation.html>

MPI Profiling Library

- ❖ **mpiP** is a lightweight profiling library for MPI applications. Because it only collects statistical information about MPI functions,
- ❖ All the information captured by **mpiP** is task-local. It only uses communication during report generation, typically at the end of the experiment, to merge results from all of the tasks into one output file.
- ❖ Gathers MPI information through the MPI profiling layer, mpiP is a link-time library.
- ❖ Low Overhead of tool
- ❖ Tool Performance for codes having weak/ strong scaling
<http://sourceforge.net/projects/mpip>

MPI Jumpshot

- ❖ Performance Visualization for Parallel Programs. Jumpshot is a Java-based visualization tool for doing postmortem performance analysis.
- ❖ <http://www.mcs.anl.gov/research/projects/perfvis/software/viewers/index.htm>

Performance Analysis tool : KOJAK

- ❖ KOJAK is a performance-analysis tool for parallel applications supporting the programming models MPI, OpenMP, SHMEM, and combinations thereof.
- ❖ Its functionality addresses the entire analysis process including instrumentation, post-processing of performance data, and result presentation.
- ❖ It is based on the idea of automatically searching event traces of parallel applications for execution patterns indicating inefficient behavior.
- ❖ The results are made available to the user in a flexible graphical user interface, where they can be investigated on varying levels of granularity.
- ❖ <http://icl.cs.utk.edu/kojak/>

Scalable Performance Analysis tool : SCALASCA

- ❖ SCALASCA project (SCalable performance Analysis of LArge SCale Applications), we have developed a scalable trace analysis tool based on the KOJAK approach that exploits both distributed memory and parallel processing capabilities available on modern large-scale systems.
- ❖ SCALASCA analyzes separate local trace files in parallel by replaying the original communication on as many CPUs as have been used to execute the target application itself.
- ❖ <http://icl.cs.utk.edu/scalasca/>

Abssoft Compilers for HPC

- ❖ Abssoft Fortran represents the highest performing 64-bit compiler available for HPC, Linux, Windows and MacOS environments. Utilizing Cray/SGI technology and advanced optimizations tuned for the individual high performance features of the AMD64 & Intel Xeon EM64T single and multi-core processors
- ❖ Abssoft Supports OpenMP, auto-parallelization, auto-vectorization
- ❖ Abssoft compilers support 64-bit or 32-bit code generation, are fully compatible with gnu tool chain or system tools (MPI, C++, Debuggers)

<http://www.absoft.com/>

Top Linux Tool

- ❖ Monitoring resource usage, optimization our system, identifying memory leak. Top provide almost everything we need to monitor our system's resource usage within single shot.

top – b

top - 15:22:45 up 4:19, 5 users, load average: 0.00, 0.03, 0.00

Tasks: 60 total, 1 running, 59 sleeping, 0 stopped, 0 zombie

Cpu(s): 3.8% us, 2.9% sy, 0.0% ni, 89.6% id, 3.3% wa, 0.4% hi, 0.0% si

Mem: 515896k total, 495572k used, 20324k free, 13936k buffers

Swap: 909676k total, 4k used, 909672k free, 377608k cached

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND	1
root	16	0	1544	476	404	S	0.0	0.1	0.1	0:01.35	init	
2	root	34	19	0	0	0	S	0.0	0.0	0:00.02	ksoftirqd/0	
3	root	10	-5	0	0	0	S	0.0	0.0	0:00.11	events/0	

time top -b -n 1;

top -p 4360 4358

What is PAPI ?

- ❖ PAPI is an acronym for Performance Application Programming interface.
- ❖ PAPI is a specification of a cross-platform interface to hardware performance counters on modern microprocessors. These counters exist as a small set of registers that count events, which are occurrences of specific signals related to a processor's function.

Source : <http://icl.cs.utk.edu/papi/index.html>

Why PAPI ?

- ❖ The purpose of the PAPI is to design, standardize and implement a portable API to access the hardware performance monitor counters found on most modern microprocessors.
- ❖ PAPI can
 - Provide a solid foundation for cross platform performance analysis tools
 - Characterize application and system workload on the CPU
 - simulate the performance tool development
 - simulate research on more sophisticated feedback driven compilation techniques

Source : <http://icl.cs.utk.edu/papi/index.html>

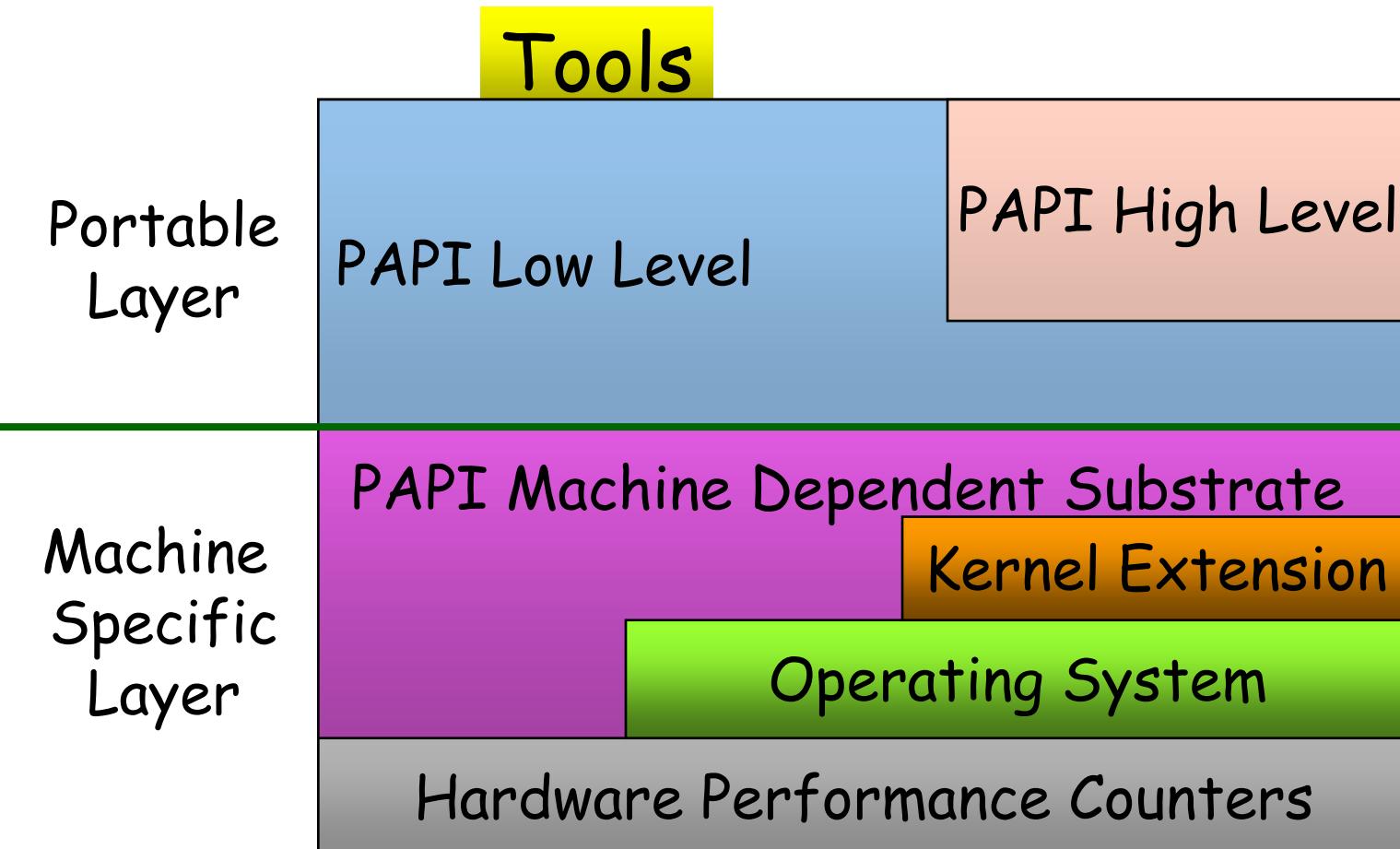
Hardware Performance Counters ?

- ❖ Hardware performance counters, or Hardware counters are a set of special-purpose registers built in modern microprocessors to store the counts of hardware-related activities within computer systems.
- ❖ Compared to software profilers, hardware counters provide low-overhead access to a wealth of detailed performance information related to CPU's function units, caches and main memory etc.

Following table shows some examples of hardware counters

Processor	# HC
Intel Pentium	18
IA-64	4
Power 4	8
AMD-Athlon	4

PAPI Architecture



Using PAPI

- ❖ Installation of PAPI on Linux-x86 the kernel be patched and recompiled with the **PerfCtr** patch
- ❖ Include the header file “**papi.h**” for C programs and “**fpapi.h**” for Fortran programs
- ❖ Compiling with PAPI : Use **-L<PAPI PATH>/lib-lpapi** with the compilation process of the application.
- ❖ Using PAPI in an application requires a few steps :
 - Including the event definition
 - Initializing the PAPI lib
 - Setting up the performance counters
 - Linking with PAPI lib

Using PAPI

- ❖ Relevant hardware counter data:

- Total cycles
- Total instructions
- Floating point operations
- Load/store instructions
- Cycles stalled
 - ✓ waiting for memory access
 - ✓ waiting for resource
- Conditional branch instructions
 - ✓ executed
 - ✓ mispredicted

Using PAPI : How do I optimize my application ?

1. Optimize compiler switches
2. Integrate libraries
3. Profile
4. Optimize blocks of code that dominate execution time by using hardware counter data to determine why the bottlenecks exist
5. Always examine correctness at every stage!
6. Go To 3...

Using PAPI Utilities

Commands available in bin dir of PAPI Installation:

❖ papi_avail

- It is a utility program that provides availability and detail information for PAPI preset events.
- Available options are **-a**, **-d**, **-t**,
-e <event_name>

❖ papi_cost

- Computes execution time cost for basic PAPI operations
- Computes min, max, mean std. Deviation of execution times for PAPI start/stop pairs and for PAPI reads.

❖ papi_mem_info

- Utility program provides information on the memory architecture

Using PAPI : Events

- ❖ Events are occurrences of specific signals related to a processor's function.
Ex: cache misses, number of floating point operations
- ❖ Preset events are mappings from symbolic names to machine specific definitions for a particular hardware resource.
 - ❖ Ex: **PAPI_TOT_CYC** (I.e Total Cycles), **PAPI_FLOPS**
- ❖ Native events comprise the set of all events that are countable by the CPU.

Using PAPI : PAPI library Interface

- ❖ PAPI provides two APIs to access the underlying counter hardware:
 - The low level interface manages hardware events in user defined groups called EventSets. (PAPI low level)
 - The high level interface simply provides the ability to start, stop and read the counters for a specified list of events. (PAPI high level)

Advanced PAPI features : PAPI with Threads

- ❖ PAPI must be able to support both explicit (library calls) and implicit (compiler directives) threading models.
- ❖ PAPI only supports thread level measurements only if the threads have a scheduling entity known and handled by the operating system's kernel.
- ❖ Thread support in the PAPI library can be initialized by calling the function

PAPI_thread_init(handle)

handle -- Pointer to a routine that returns the current thread ID.

Advanced PAPI features : PAPI with Threads

API's for Threads

- ❖ **PAPI_thread_init(handle)**
 - Thread support in PAPI is initialised
- ❖ **PAPI_thread_id()**
 - get the thread identifier of the current thread
- ❖ **PAPI_get_thr_specific(tag, ptr)**
 - retrieve the pointer from the array with index tag
- ❖ **PAPI_set_thr_specific(tag, ptr)**
 - save ptr into an array indexed by tag

Part-XII:
An Overview of Xeon Mult-Core Systems

Intel Xeon-Host : system configuration

System 1 : Intel Sandy Bridge Server

- ❖ Intel Software Development Platform (Intel SDP) MAK F1 Family.
- ❖ **Platform** : Intel (r) Many Integrated Core Architecture
- ❖ **Platform Code Name** : Knights Ferry
- ❖ **CPU Chipset Codename** : Westmere EP/ Tylersburg UP
- ❖ **Board Codename** : Sandy Core.
- ❖ **CPU** : Intel Xeon X5680 Westmere 3.33GHz 12MB L3 Cache
LGA 1366 130W Six-Core Server Processor BX80614X5680

Source : www.cdac.in/ Intel

Intel Xeon-Host : system configuration

System 2 : Super Micro SYS-7047GR-TPRF Server

- ❖ **Chipset** : Intel C602 Chipset,
- ❖ **Mother board** : Super X9DRG-QF,
- ❖ **CPU** : Intel Xeon processor E5-2643 (quad core) (up to 150W TDP),
Support for Xeon Phi - 5110P.
- ❖ **Memory** : 32 GB DDR3 ECC Registered memory(1600 MHz ECC supported DDR3 SDRAM 72-bit, 240-pin gold-plated DIMMs),
- ❖ **Expansion slot** : with 4x PCI-E 3.0 x16 (double-width), 2x PCI-E x8),
- ❖ **IPMI** : Support for IPMI (Support for Intelligent Platform Management Interface v.2.0, IPMI 2.0 with virtual media over LAN and KVM-over-LAN support),
- ❖ **Power** : 1620W high-efficiency redundant power supply w/PMBus.
- ❖ **Storage** : SATA 3.0 6Gbps with RAID 0,1 support ,1 TB SATA Hard Disk,
- ❖ **Network** : Intel i350 Dual Port Gigabit Ethernet with support of 10BASE-T, 100BASE-TX, and 1000BASE-T, RJ45 output and 1x Realtek RTL8201N\PHY (dedicated IPMI port)

Intel Xeon-Host : Benchmarks Performance

Systems 3 : Host : Xeon (Memory Bandwidth (BW) - Xeon: 8 bytes/channel * 4 channels * 2 sockets * 1.6 GHz = 102.4 GB/s)

PARAM YUVA-II Intel Xeon- Node

- Node : Intel-R2208GZ; Intel Xeon E52670;
- Core Frequency : 2.6GHz;
- Cores per Node : 16 ;
- Peak Performance /Node : 2.35 TF;
- Memory : 64 GB;

Source : www.cdac.in/ Intel

PARAM YUVA-II Intel Xeon- Node Benchmarks(*)

Xeon Node Memory Bandwidth :

8 bytes/channel * 4 channels * 2 sockets * 1.6 GHz = 102.4 GB/s)

Experiment Results : Achieved Bandwidth : 70 % ~75 % Effective bandwidth can be improved in the range of 10% to 15% with some optimizations

PARAM YUVA Node : Intel-R2208GZ; Intel Xeon E52670; Core Frequency : 2.6GHz; Cores per Node : 16 ; Peak Performance /Node : 2.35 TF; Memory : 64 GB;

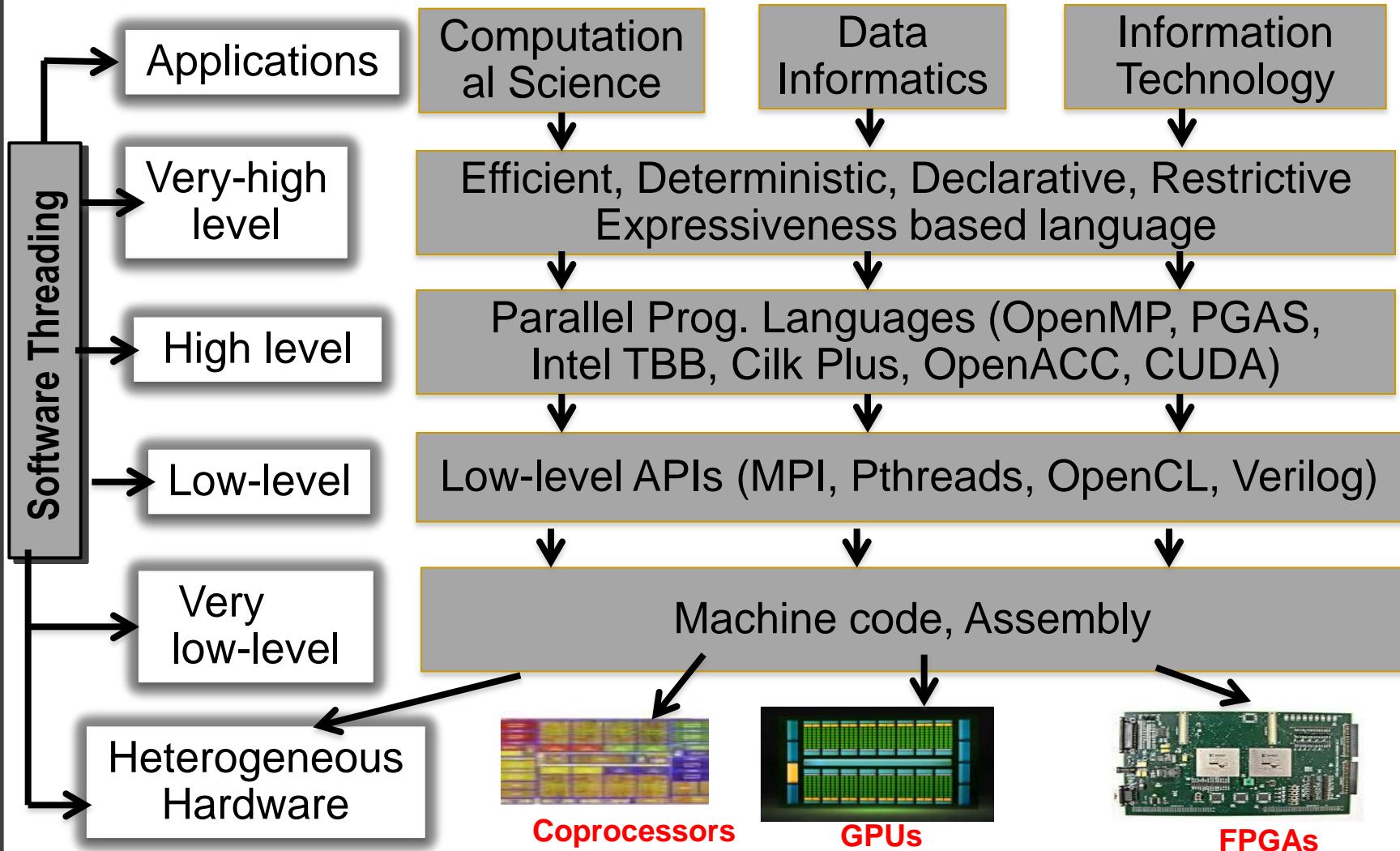
Data Size (MegaBytes)	No. of Cores (OpenMP)	Sustained Bandwidth (GB/sec)
1024	16	72.64

(*) = Bandwidth results were gathered using untuned and unoptimized versions of benchmark (In-house developed) and Intel Prog. Env

Source : <http://www.intel.com>; Intel Xeon-Phi books, conferences, Web sites, Xeon-Phi Technical Reports <http://www.cdac.in/>

<http://www.intel.in/content/dam/www/public/us/en/documents/performance-briefs/xeon-phi-product-family-performance-brief.pdf>

Prog.API - Multi-Core Systems with Devices



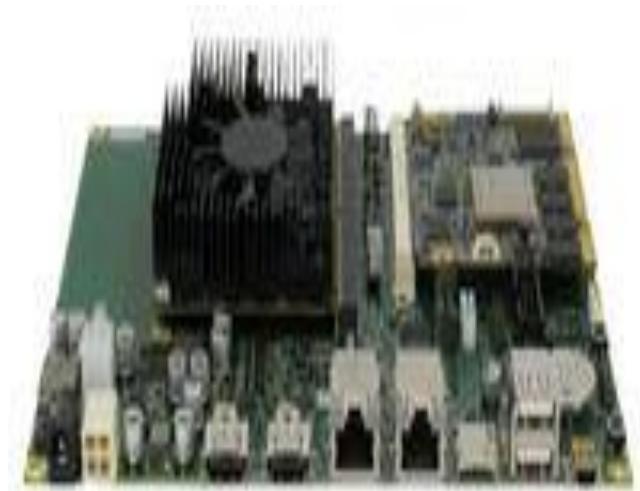
Source : NVIDIA, AMD, SGI, Intel, IBM Alter, Xilinx & References

Part-XII:

An Overview of Arm Multi-Core System

NVIDIA ARM With Carma DevKit

Carma , the board includes the company's Tegra 3 quad-core ARM A9 processor, a Quadro 1000M GPU with 96 cores (good for 270 single-precision GFlops), as well as a PCIe X4 link, one Gigabit Ethernet interface, one SATA connector, three USB 2.0 interfaces as well as a Display port and HDMI. 2GB GPU Memory

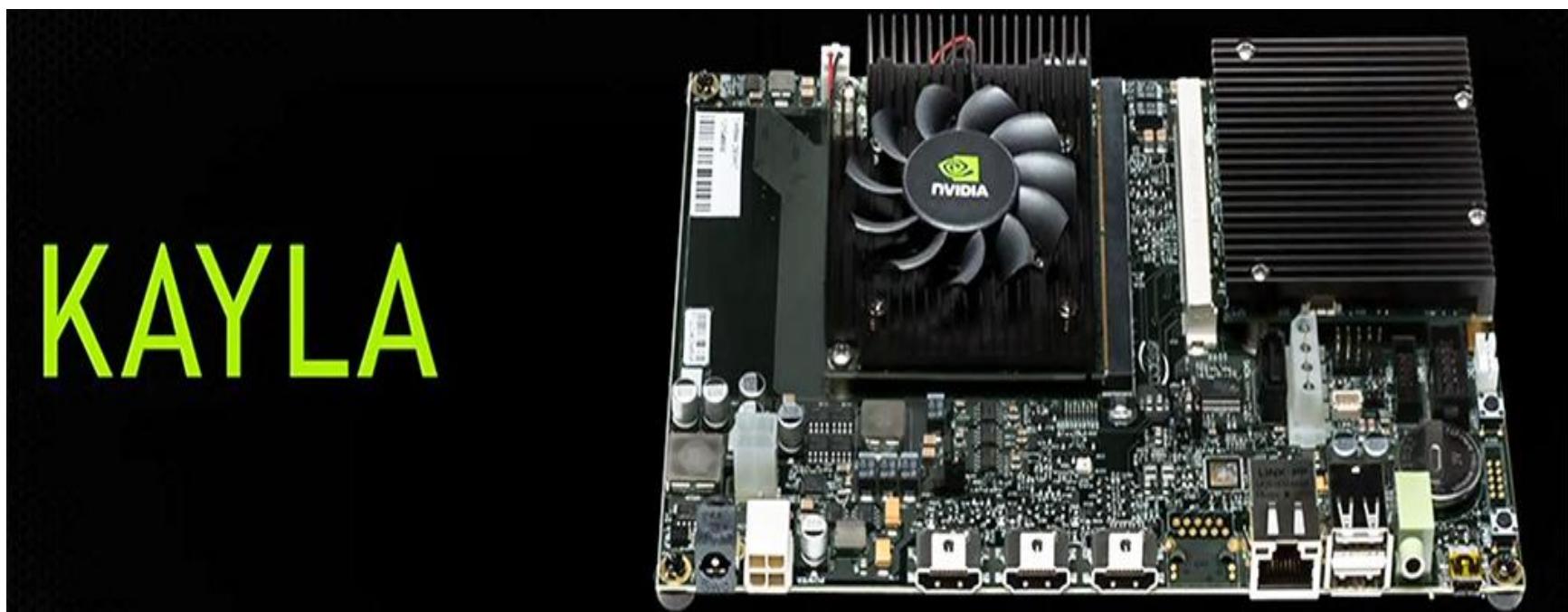


- ❖ It uses the Tegra 3 chip as the basis and, thus, has four ARM cores and an NVIDIA GPU.
- ❖ In addition, the platform has 2 GB of DDR3 RAM (random access memory) as well.
- ❖ CUDA toolkit and a Ubuntu Linux-based OS

Source : www.nvidia.com

NVIDIA ARM With KAYLA DevKit

- ❖ Mini-ITX motherboard designed for developers
- ❖ Features a Tegra 3 SoC, 2GB RAM, low power Kepler-based GPU (1GB RAM & 2 SMX or 384 CUDA cores, MXM/PCIe FF), 10W
- ❖ Supports CUDA 5.0



Source : www.nvidia.com

NVIDIA ARM With KAYLA DevKit

- ❖ Introducing the Kayla DevKit for computing on the ARM architecture – where supercomputing meets mobile computing.
- ❖ The Kayla DevKit hardware is composed of mini-ITX carrier board and NVIDIA® GeForce® GT640/GDDR5 PCI-e card.
- ❖ The mini-ITX carrier board is powered by NVIDIA Tegra 3 Quad-core ARM processor while GT640/GDDR5 enables Kepler GK208 for the next generation of CUDA and OpenGL application. Pre-installed with CUDA 5 and supporting OpenGL 4.3.
- ❖ Kayla provides ARM application development across the widest range of application types.
- ❖ Kayla brings all modern visual benefits to mobile processor, and accelerate application development to next generation Logan SOC.

NVIDIA ARM With KAYLA DevKit

Form Factor

Kayla mITX

[Buy Now](#)

CPU

[NVIDIA® Tegra® 3 ARM Cortex A9 Quad-Core](#) with NEON

GPU

NVIDIA® GeForce®
GT640/GDDR5 (TO BE
PURCHASED SEPARATELY) [Buy Now](#)

Memory

2GB DRAM

CPU - GPU Interface

PCI Express x16 / x4

Network

1x Gigabit Ethernet

Storage

1x SATA 2.0 Connector

USB

2x USB 2.0

Software

Linux Ubuntu Derivative OS
CUDA 5 Toolkit

An Overview of Multi-Core Processors

Conclusions

- ❖ An Overview of Multi-Core Architectures, Programming on Multi-Core Processors , Tuning & Performance of Software threading, & Multi-Core Software tools, Xeon (Sandy Bridge) multi-Core System & ARM Multi-core Systems are discussed.

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**)

References

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, Chicago (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>

Thank You
Any questions ?