Parallel Processing with Distribution, Grids & Clouds

Denis Caromel, Arnaud Contes

Univ. Nice, ActiveEon

Traditional Parallel Computing & HPC Solutions

- Parallel Computing
  - ❑ Principles
  - ❑ Parallel Computer Architectures
  - ❑ Parallel Programming Models
  - ❑ Parallel Programming Languages
- Grid Computing
  - ❑ Multiple Infrastructures
  - ❑ Using Grids
  - ❑ P2P
  - ❑ Clouds
- Conclusion

Parallel (Computing)

- **Execution of several activities at the same time.**
  - **2 multiplications at the same time on 2 different processes,**
  - **Printing a file on two printers at the same time.**

Sequential programming	Parallel programming
• Single execution flow • Single instruction at one time • Single processor • Unique model (von Neumann)	• Several execution flows • Several instructions executed simultaneously • Several processors • No unique model

Why Parallel Computing ?

- Save time - wall clock time
- Solve larger problems
- Parallel nature of the problem, so parallel models fit it best
- Provide concurrency (do multiple things at the same time)
- Taking advantage of non-local resources
- Cost savings
- Overcoming memory constraints
- Can be made highly fault-tolerant (replication)

What application ?

Traditional HPC

- Nuclear physics
- Fluid dynamics
- Weather forecast
- Image processing, Image synthesis, Virtual reality
- Petroleum
- Virtual prototyping
- Biology and genomics

Enterprise App.

- J2EE and Web servers
- Business Intelligence
- Banking, Finance, Insurance, Risk Analysis
- Regression tests for large software
- Storage and Access to large logs
- Security: Finger Print matching, Image behavior recognition

How to parallelize ?

- 3 steps :

1. Breaking up the task into smaller tasks
2. Assigning the smaller tasks to multiple workers to work on simultaneously
3. Coordinating the workers

- Seems simple, isn't it ?

Additional definitions

Concurrency	Simultaneous access to a resource, physical or logical Concurrent access to variables, resources, remote data
Distribution	Several address spaces
Locality	Data located close to CPU (If several CPU and hard disks.)

Parallelism vs Distribution vs Concurrency

- Parallelism sometimes proceeds from distribution:
  - Problem domain parallelism
  - E.g: Collaborative Computing
- Distribution sometimes proceeds from parallelism:
  - Solution domain parallelism
  - E.G.: Parallel Computing on Clusters
- Parallelism leads naturally to Concurrency:
  - Several processes trying to print a file on a single printer

Levels of Parallelism HardWare

- Bit-level parallelism
  - Hardware solution
  - based on increasing processor word size
    - 4 bits in the ‘70s, 64 bits nowadays

Focus on hardware capabilities for structuring

- Instruction-level parallelism
  - A goal of compiler and processor designers
  - Micro-architectural techniques
    - Instruction pipelining, Superscalar, out-of-order execution, register renamming

Focus on program instructions for structuring

Levels of Parallelism SoftWare

- Data parallelism (loop-level)
  - ❑ Distribution of data (Lines, Records, Data-structures, …) on several computing entities
  - ❑ Working on local structure or architecture to work in parallel on the original

Focus on the data for structuring

- Task Parallelism
  - ❑ Task decomposition into sub-tasks
  - ❑ Shared memory between tasks or
  - ❑ Communication between tasks through messages

Focus on tasks (activities, threads) for structuring

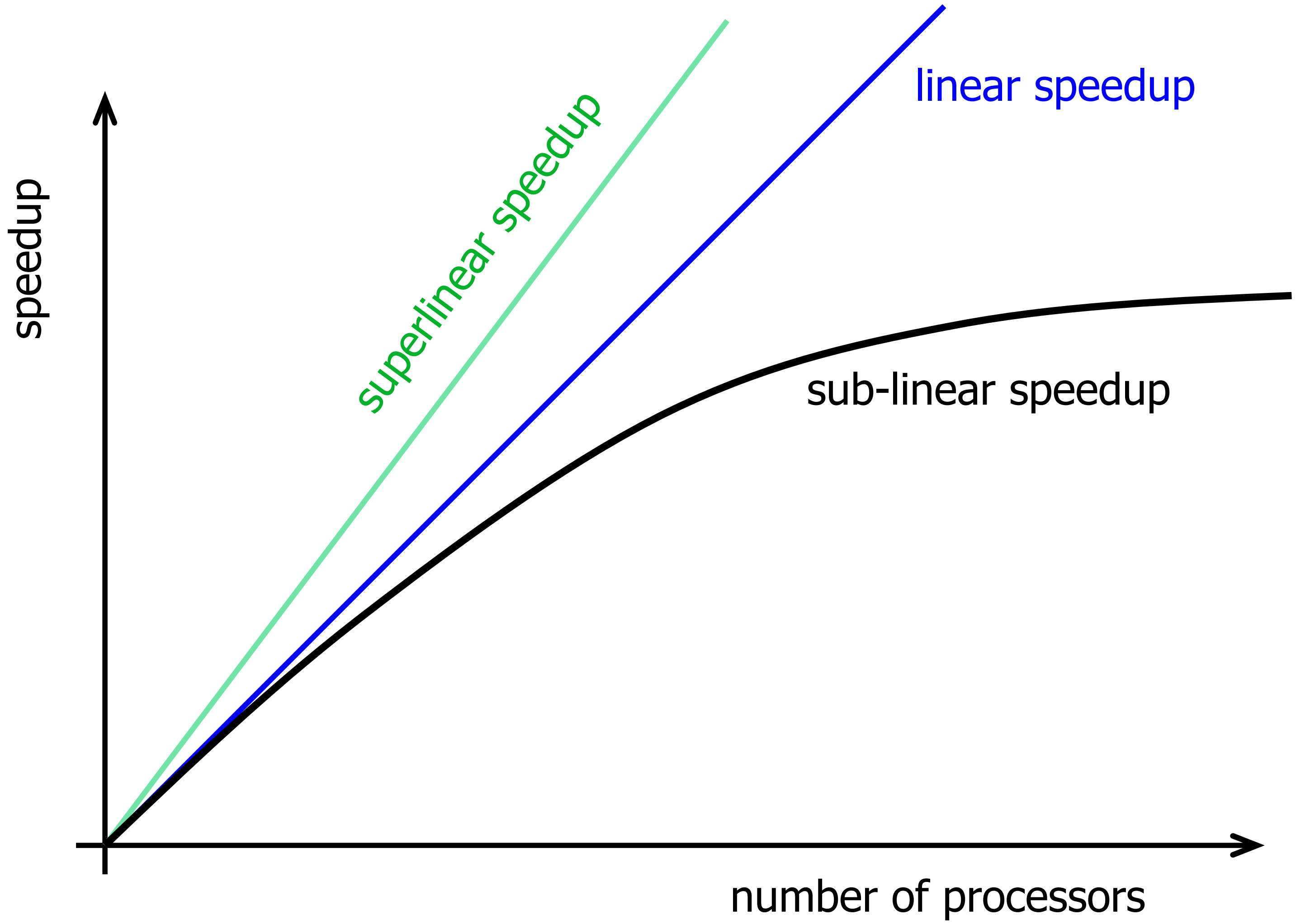
Performance ?

- Performance as Time
  - Time spent between the start and the end of a computation
- Performance as rate
  - MIPS (Millions of Instructions / sec)
    - Not equivalent on all architectures
- Peak Performance
  - Maximal Performance of a Resource (theoretical)
  - Real code achieves only a fraction of the peak performance

Code Performance

- how to make code go fast : “High Performance”
- Performance conflicts with
  - Correctness
    - By trying to write fast code, one can breaks it
  - Readability
    - Multiplication/division by 2 versus bit shifting
    - Fast code requires more lines
    - Modularity can hurt performance
      - Abstract design
  - Portability
    - Code that is fast on machine A can be slow on machine B
    - At the extreme, highly optimized code is not portable at all, and in fact is done in hardware.

Speedup


superlinear speedup
linear speedup
speedup
sub-linear speedup
number of processors

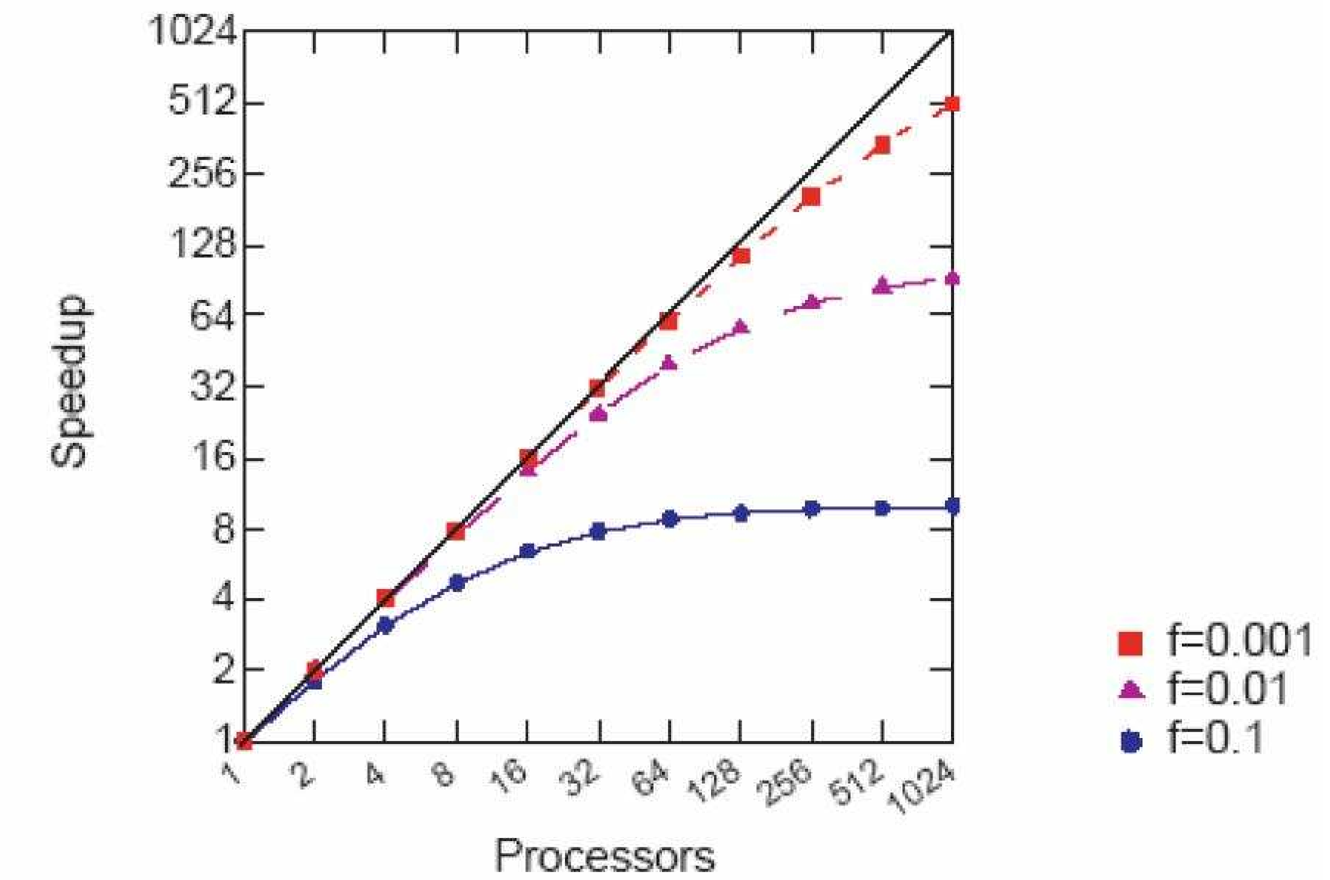
Super Linear Speedup

- Rare
- Some reasons for speedup > p (efficiency > 1)
  - Parallel computer has p times as much RAM so higher fraction of program memory in RAM instead of disk
    - An important reason for using parallel computers
  - Parallel computer is solving slightly different, easier problem, or providing slightly different answer
  - In developing parallel program a better algorithm was discovered, older serial algorithm was not best possible

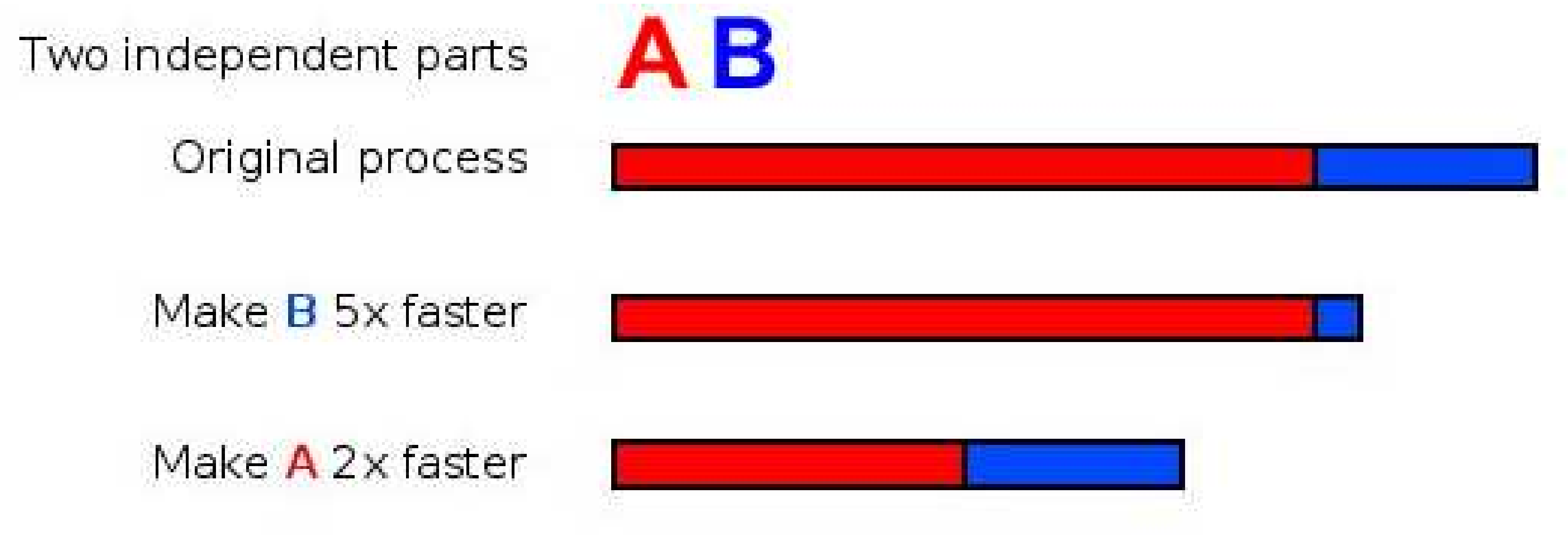
Amdahl's Law

- Amdahl [1967] noted: given a program,
  - ❑ let f be fraction of time spent on operations that must be performed serially.
- Then for p processors,
  - ❑ *Speedup(p) ≤ 1/(f + (1 − f)/p)*
- Thus no matter how many processors are used
  - ❑ *Speedup ≤ 1/f*
- Unfortunately, typically f was 10 –20%
- *Useful rule of thumb :*
  - ❑ *If maximal possible speedup is S, then S processors run at about 50% efficiency.*

Maximal Possible Speedup



Another View of Amdahl's Law



- If a significant fraction of the code (in terms of time spent in it) is not parallelizable, then parallelization is not going to be good

Scalability

- Measure of the “effort” needed to maintain efficiency while adding processors
- For a given problem size, plot Efd(p) for increasing values of p
  - It should stay close to a flat line
- Isoefficiency: At which rate does the problem size need to be increased to maintain efficiency
  - By making a problem ridiculously large, one can typically achieve good efficiency
  - Problem: is it how the machine/code will be used?

Traditional Parallel Computing & HPC Solutions

- Parallel Computing
  - Principles
  - Parallel Computer Architectures
  - Parallel Programming Models
  - Parallel Programming Languages
- Grid Computing
  - Multiple Infrastructures
  - Using Grids
  - P2P
  - Clouds
- Conclusion

Michael Flynn’s Taxonomy

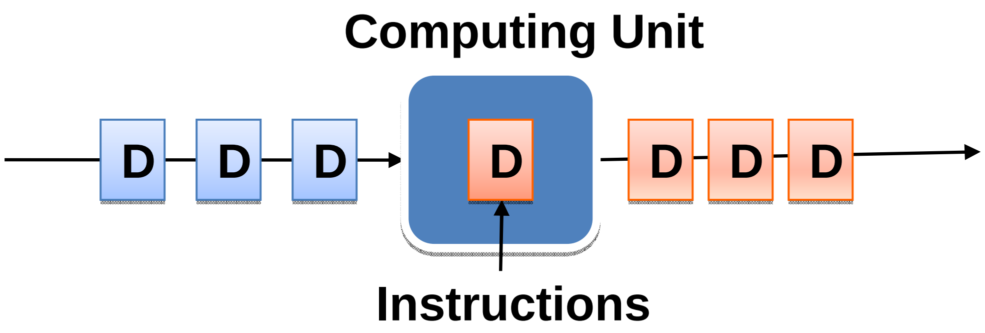
classification of computer architectures

Data & Operand (instructions)

Instructions Data streams	Single Instruction (SI)	Multiple Instruction (MD)
Single Data (SD)	SISD Single-threaded process	MISD Pipeline architecture
Multiple Data (MD)	SIMD Vector processing	MIMD Multi-threaded programming

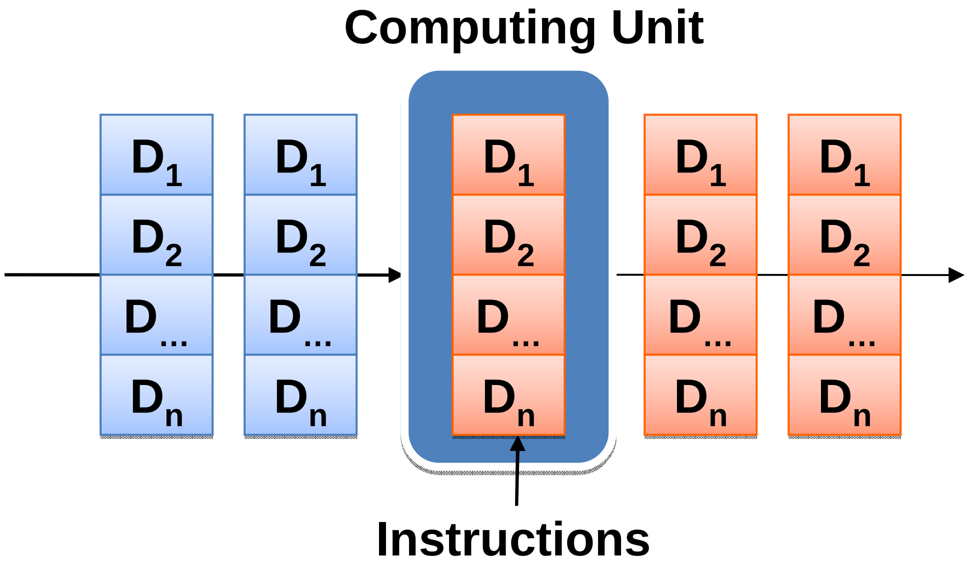
Single Instruction Single Data Stream

- A single processor executes a single instruction stream
- Data stored in a single memory
- Corresponds to the Von Neumann architecture



Single Instruction Multiple Data Streams

- Vector processors
  - ❑ Instructions executed over vectors of data
- Parallel SIMD
  - ❑ Synchronous execution of the same instruction



Cray 1 Vector machine, 70s,

CPU 64bits, 8Mo RAM, 166 MFlops weighed 5.5 tons

Cray X1E - 2005

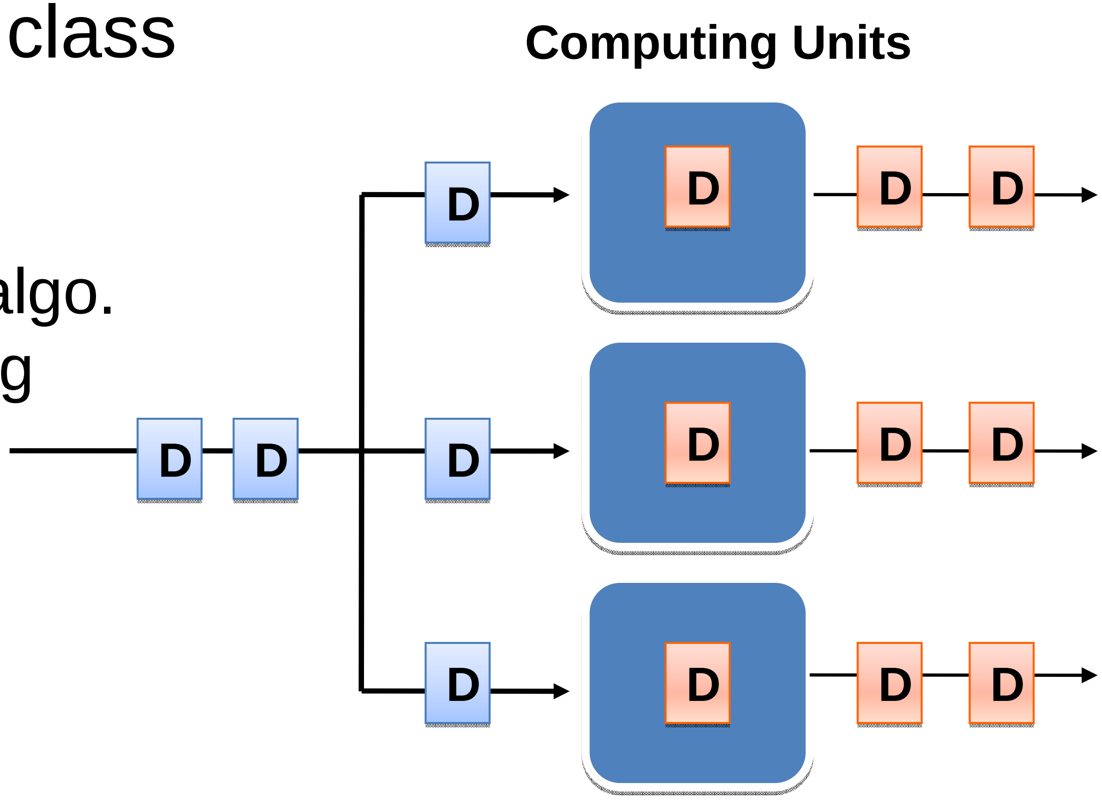
CPUs 1020* 1GHz, 4080 Go RAM,18 Tflops,

rank 72

Multiple Instructions Single Data Streams

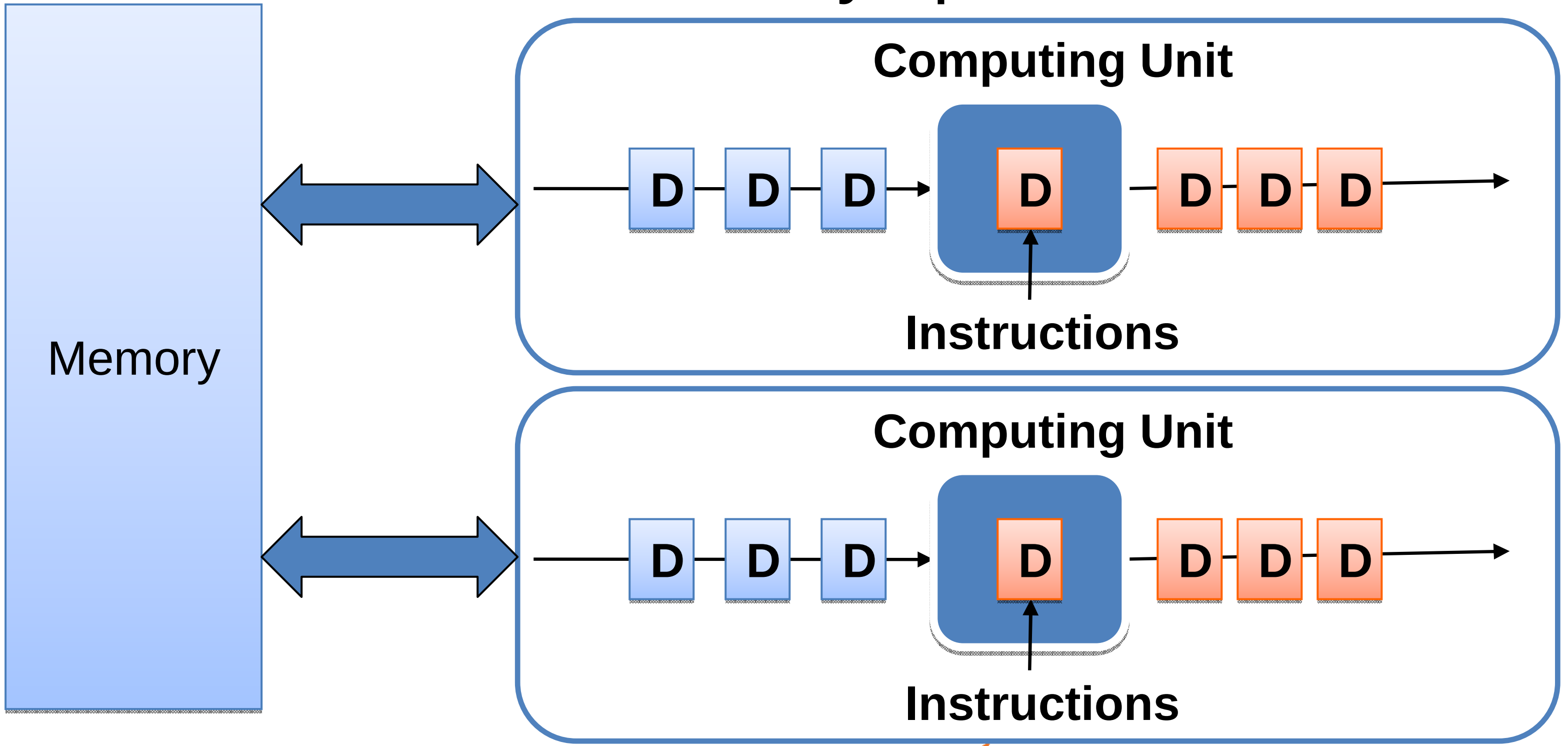
Few examples of this architecture in this class (systolic arrays)

- ❑ Cryptography algo.
- ❑ 3D – Raytracing engines



Multiple Instructions Multiple Data Streams

- Distributed systems are MIMD architectures
- Either exploiting a single shared memory space or a distributed memory space.



Sharing Memory or not

- **Shared memory systems**:

  Shared memory systems have multiple CPUs all of which share the same address space (SMP)

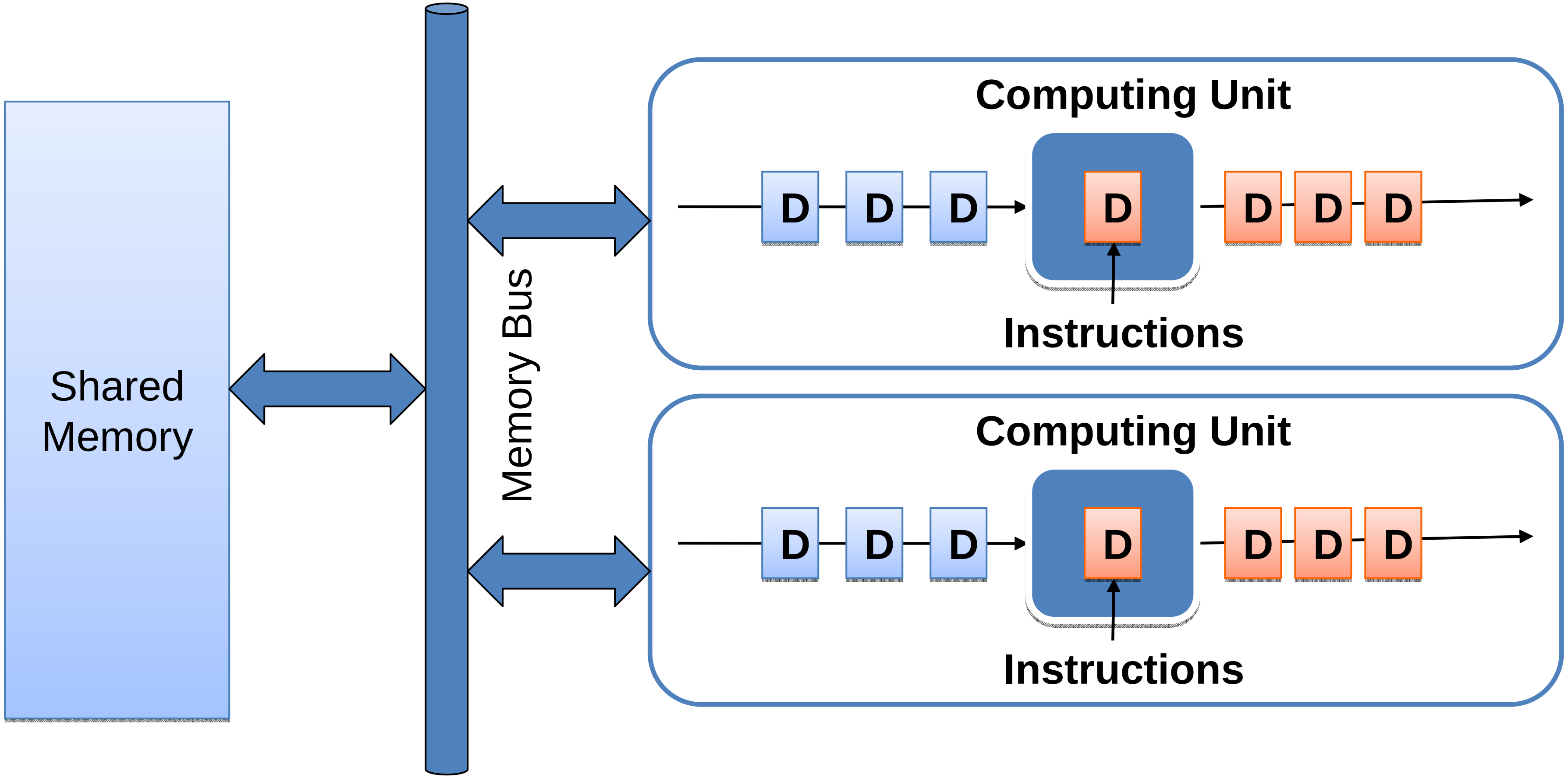
  - ❑ Uniform Memory Access
  - ❑ Non Uniform Memory Access

- **Distributed memory systems**:

  In this case each CPU has its own associated memory, interconnected computers

Multiple Instructions Multiple Data Streams

Shared-memory	**Multiple CPUs with a shared Memory**

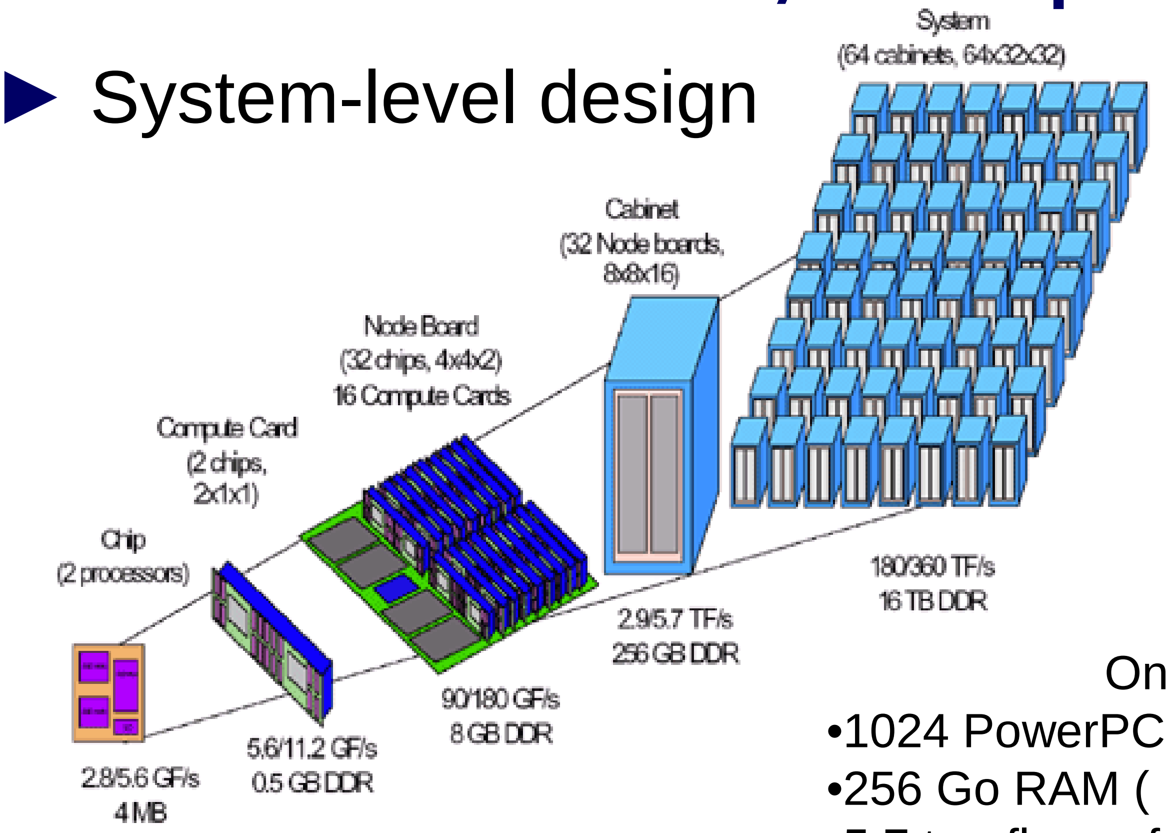


Symmetric Multi Processing System

- SMP machine
  - ❑ Multiple CPUs
  - ❑ A single memory control
  - ❑ Uniform Memory Access
- Usually Entry-Level Servers
  - ❑ Easy and cheap with few processors
  - ❑ Hard & very expensive to design with 8+ CPUs
- Multicores CPUs are SMP
  - ❑ Your laptop is probably an SMP machine (dual core), mine is …

IBM Blue Gene/L SuperComputer

► System-level design



One Cabinet

•1024 PowerPC 700Mhz

•256 Go RAM (up to 2Go) /

•5.7 teraflops of processing power

•IBM version of a Linux Kernel on processing nodes

•Novell Linux on Management Nodes

IBM Blue Gene/L SuperComputer

- Maximum size of 65,536 compute nodes
  - 2007 : up to 1000 Tflops/s

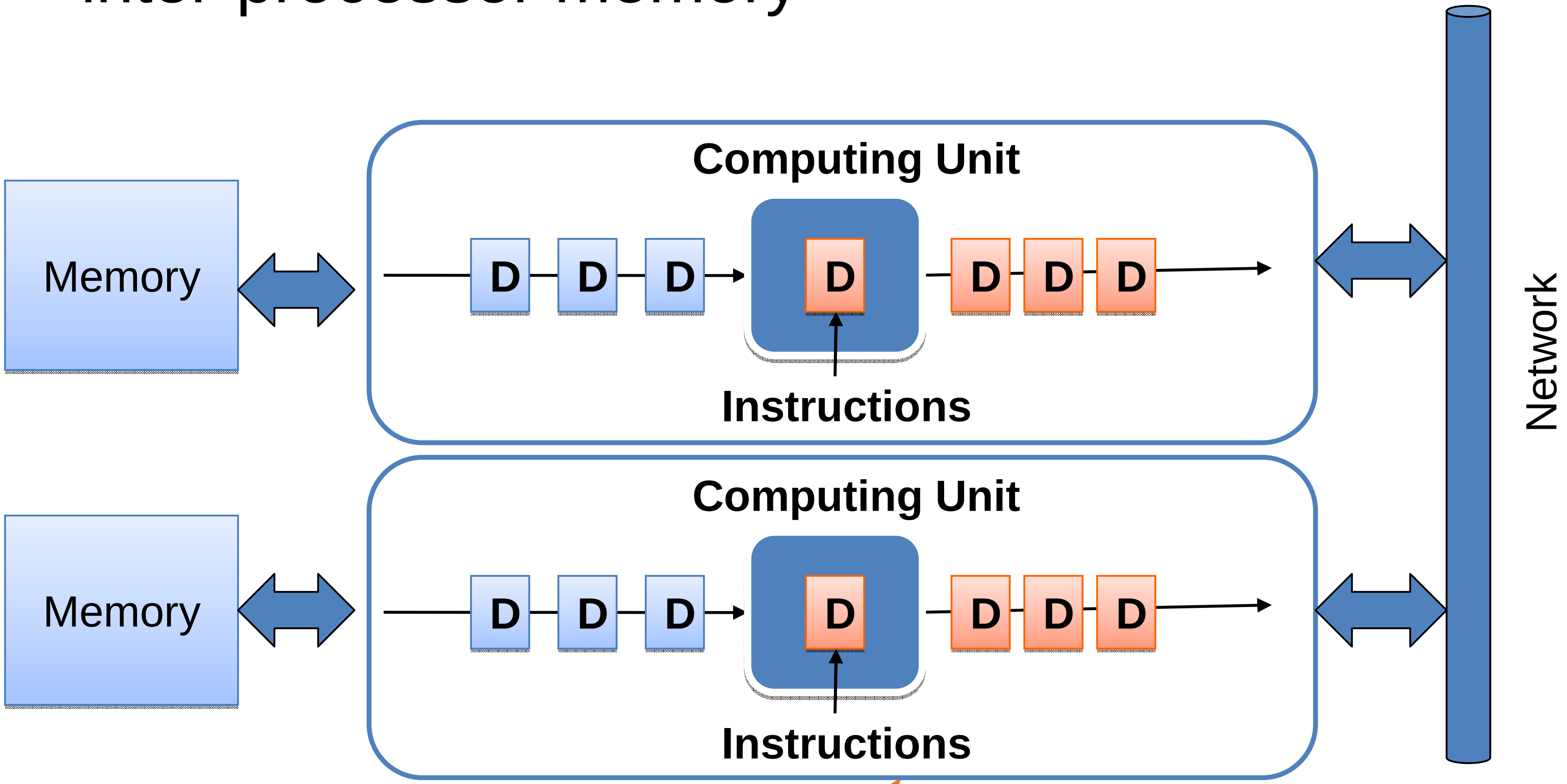
- 1000 Tflops/s cost (only) 200 M$
- 5 MW of power for 1000 Tflop/s
- ~300 tons of cooling
- 4,000 sq ft of floor space

Shared Memory, Conclusion

- Advantages
  - Memory scalable to number of processors. Increase number of processors, size of memory and bandwidth increases.
  - Each processor can rapidly access its own memory without interference
- Disadvantages
  - Difficult to map existing data structures to this memory organization
  - User responsible for sending and receiving data among processors
  - To minimize overhead and latency, data should be blocked up in large chunks and shipped before receiving node needs it

MIMD, Distributed Memory

- Require a communication network to connect inter-processor memory



Distributed Memory, Conclusion

- Advantages:
  - Memory is scalable with number of processors. Increase the number of processors and the size of memory increases proportionately.
  - Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency.
  - Cost effectiveness: can use commodity, off-the-shelf processors and networking.
- Disadvantages:
  - The programmer is responsible for many of the details associated with data communication between processors.
  - It may be difficult to map existing data structures, based on global memo

Traditional Parallel Computing & HPC Solutions

- ► Parallel Computing
  - ❑ Principles
  - ❑ Parallel Computer Architectures
  - ❑ Parallel Programming Models
  - ❑ Parallel Programming Languages
- ► Grid Computing
  - ❑ Multiple Infrastructures
  - ❑ Grids
  - ❑ P2P
  - ❑ Clouds
- ► Conclusion

Parallel Programming Models

- several parallel programming models in common use:
  - ❑ Threads (Posix)
  - ❑ Shared Memory (OpenMP)
  - ❑ Message Passing (MPI)
  - ❑ Data Parallel (Fortan)
  - ❑ Hybrid (MPI + Posix)

Issues When Parallelizing

- Common issue: Partitioning
  - Data decomposition
  - Functional decomposition

- 2 possible outputs
  - Embarrassingly Parallel
    - Solving many similar, but independent, tasks : parameter sweeps.
  - Communicating Parallel Computing
    - Solving a task by simultaneous use of multiple processors, all elements (intensively) communicating

Communicating Tasks

- Cost of communications
- Latency vs. Bandwidth
- Visibility of communications
- Synchronous vs. asynchronous communications
- Scope of communications
  - ❑ *Point-to-point*
  - ❑ *Collective*
- Efficiency of communications

Data Dependencies

- A ***dependence*** exists between program statements when the order of statement execution affects the results of the program.
- A ***data dependence*** results from multiple uses of the same location(s) in storage by different tasks.
- Dependencies are one of the primary inhibitors to parallelism.
- Handle Data Dependencies:
  - Distributed memory - communicate required data at synchronization points.
  - Shared memory -synchronize read/write operations between tasks.

The Memory Bottleneck

- The memory is a very common bottleneck that programmers often don't think about
  - When you look at code, you often pay more attention to computation
  - a[i] = b[j] + c[k]
    - The access to the 3 arrays take more time than doing an addition
    - For the code above, the memory is the bottleneck for most machines!

- In the 70's, everything was balanced
  - The memory kept pace with the CPU
    - n cycles to execute an instruction, n cycles to bring in a word from memory
- No longer true
  - Memories have gotten 100x larger
  - CPUs have gotten 1,000x faster
  - Data have gotten 1,000,000x larger

Flops are free and bandwidth is expensive and processors are STARVED for data

Memory and parallel programs

- Principle of locality: make sure that concurrent processes spend most of their time working on their own data in their own memory
  - Place data near computation
  - Avoid modifying shared data
  - Access data in order and reuse
  - Avoid indirection and linked data-structures
  - Partition program into independent, balanced computations
  - Avoid adaptive and dynamic computations
  - Avoid synchronization and minimize inter-process communications
- Locality is what makes efficient parallel programming painful
  - As a programmer you must constantly have a mental picture of where all the data is with respect to where the computation is taking place

Duality: Copying vs. Sharing

Sharing

- Advantages:
  - Immediate Update
  - Consistency
- Drawbacks:
  - Concurrent accesses have to be coordinated

Copying

- Advantages :
  - Unique access
  - Fast
- Drawbacks :
  - Update cost
  - No coherency

► Shared memory does not allow scalability

► (Raw) Message Passing is too Complex

Classification Extension

- Single Program, Multiple Data streams (SPMD)
  - Multiple autonomous processors simultaneously executing the same program on different data , but at independent points, rather than in the lockstep that SIMD imposes
    - Typical MPI like weather forecast

- Multiple Program Multiple Data Streams (MPMD)
  - Multiple autonomous processors simultaneously operating at least 2 independent programs.
    - Master Workers,
    - SPMD Numerical + Parallel Visualization

Architecture to Languages

SMP:

- ► Shared-Memory Processing
- ► Symmetric Multi Processing

MPP:

- ► Message Passing Processing
- ► Massively Parallel Processing

Parallel Programming Models

- Implicit
  - ❑ Sequential Model and automatic parallelization:
    - Analysis of data dependencies by a parallelizing compiler
    - Coming back with Multicores, but … has been hard, still will be
  - ❑ Parallel Runtime (hidden language)
  - ❑ No user specification nor control over the scheduling of calculation or the placement of data

Parallel Programming Models

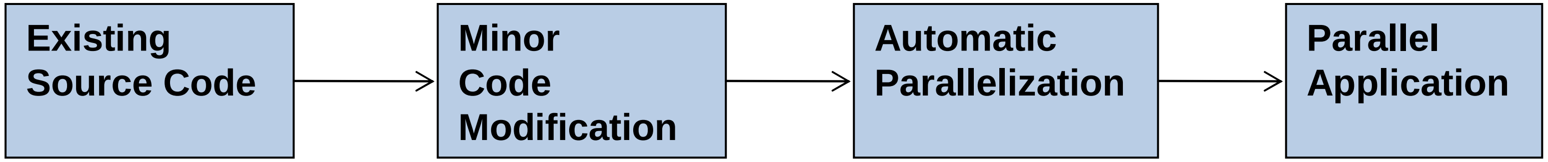
- Explicit
  - Programmer is responsible for the parallelization work:
    - Task and Data decomposition
    - Mapping Tasks and Data to resources
    - Communication or synchronization management

- Several classes:
  - Control (loops and parallel regions) directives (Fortran-S, KSR-Fortran, OpenMP)
  - Data distribution: HPF (historical)
  - Distributed models: PVM , MPI, ProActive

Parallel Programming Models

Strategy 1: Automatic parallelization

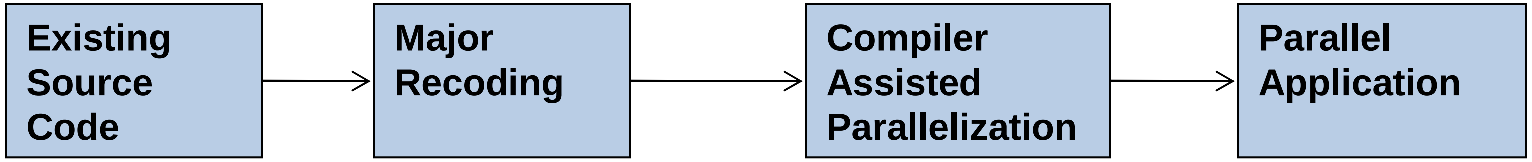
- Parallelization made by the compiler
- No control by the programmer
- Difficult to achieve



Parallel Programming Models

Strategy 2: Major Recoding

- Writing of the parallel application from scratch
- Low code reusability
- Usually limited to data distribution and placement

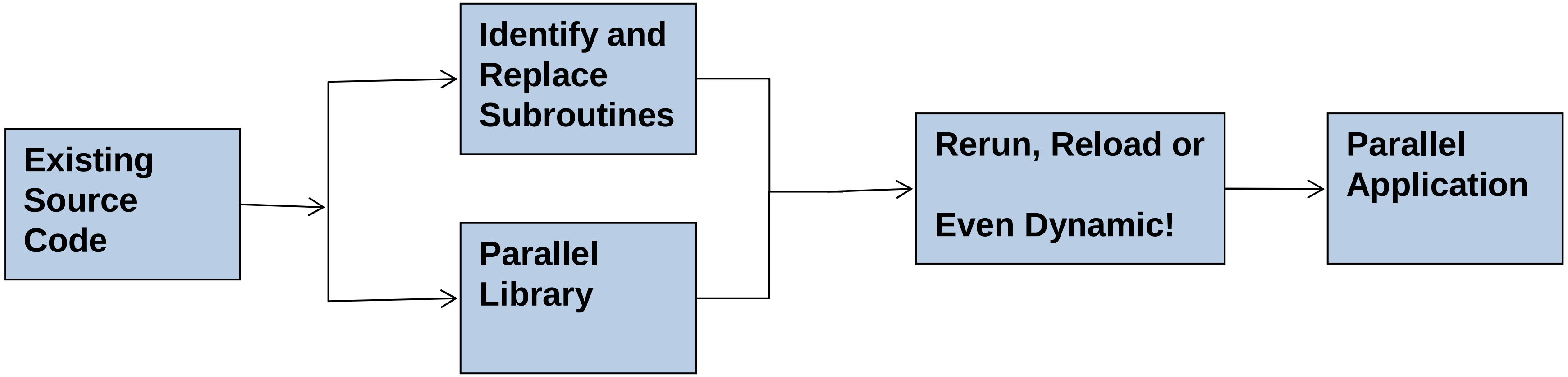


MPI, OpenMP, PVM

Parallel Programming Models

Strategy 3: Parallel Libraries (ProActive)

- Efficient implementation with libraries of code which help managing the parallelization


Existing Source Code
Identify and Replace Subroutines
Parallel Library
Rerun, Reload or Even Dynamic!
Parallel Application

ProActive, GridGain, Hadoop

Traditional Parallel Computing & HPC Solutions

- ► Parallel Computing
  - ❑ Principles
  - ❑ Parallel Computer Architectures
  - ❑ Parallel Programming Models
  - ❑ Parallel Programming Languages
- ► Grid Computing
  - ❑ Multiple Infrastructures
  - ❑ Using Grids
  - ❑ Using Clouds
- ► Conclusion

Parallel Programming Languages

Goals

- System architecture transparency
- Network communication transparency
- Easy-to-use
- Fault –tolerance
- Support of heterogeneous systems
- Portability
- High level programming language
- Good scalability
- Some parallelism transparency

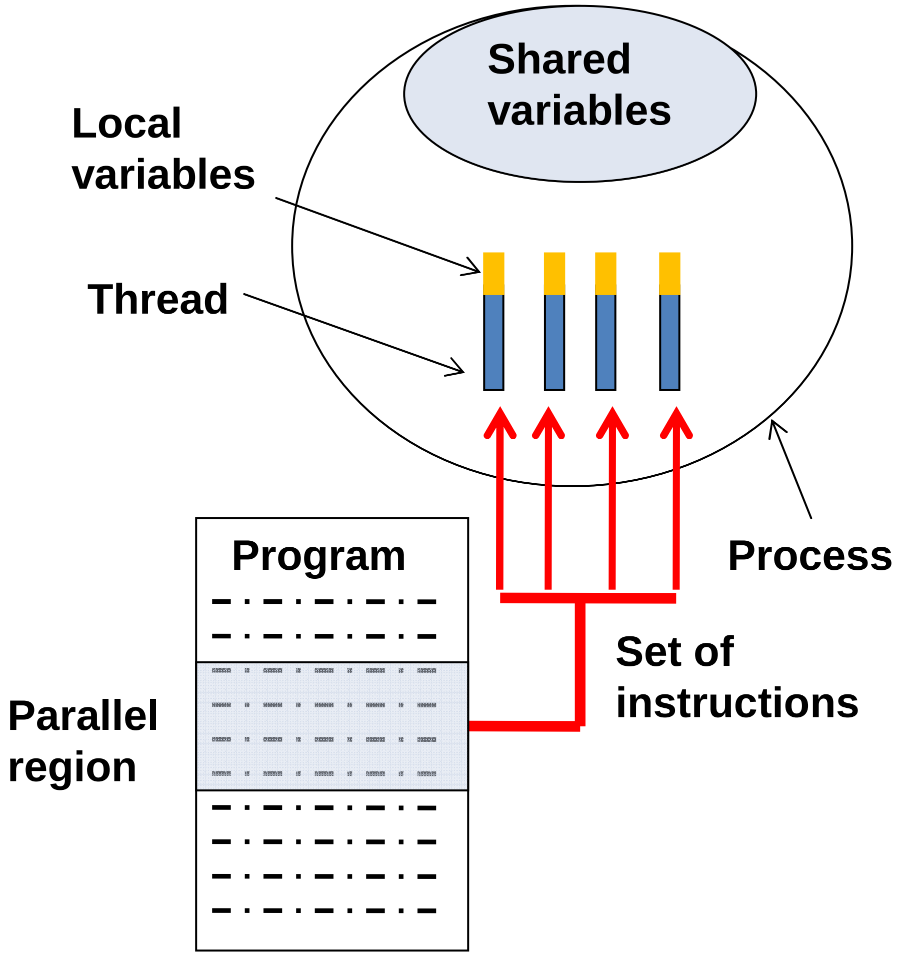
OpenMP: Shared Memory

Application Programming Interface

- Multiplatform shared memory multi-threads programming
- Compiler directives, library routines, and environnement variables
- For C++ and Fortran

OpenMP: General Concepts

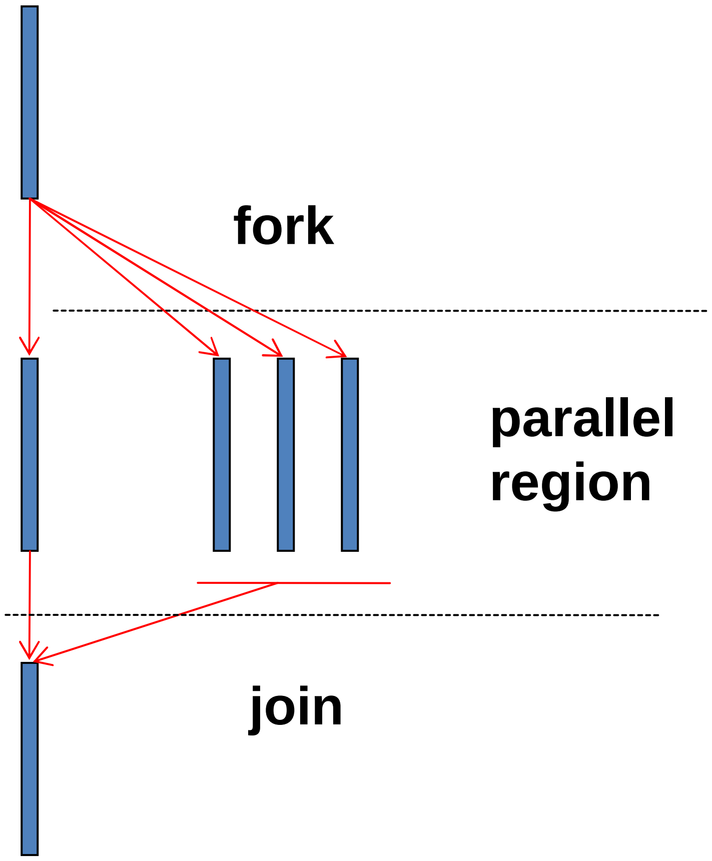
- An OpenMP program is executed by a unique process
- This process activates threads when entering a parallel region
- Each thread executes a task composed by several instructions
- Two kinds of variables:
  - ❑ Private
  - ❑ Shared



OpenMP

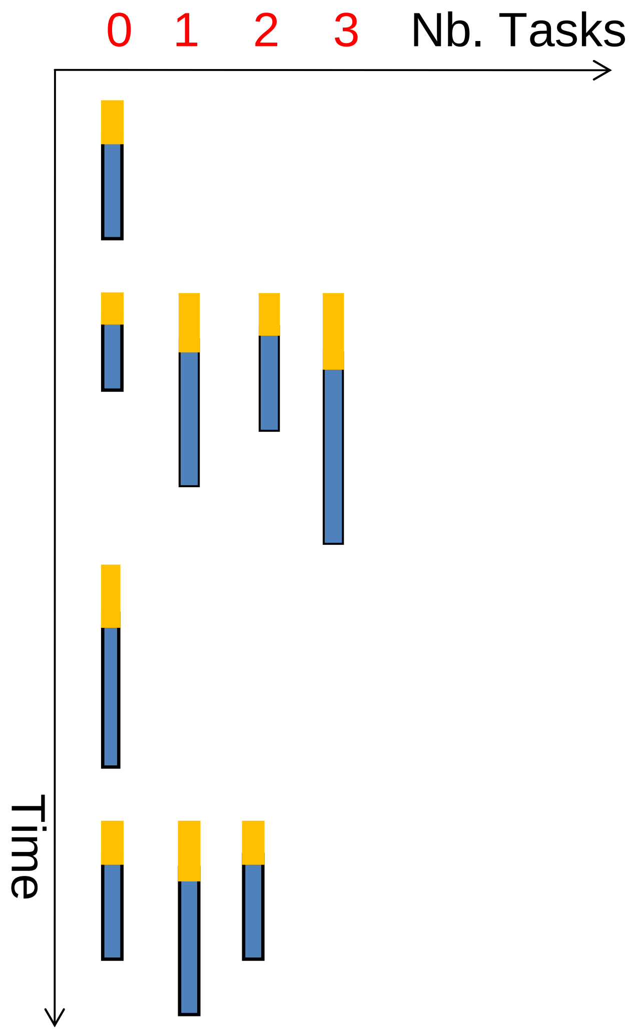
- The programmer has to introduce OpenMP directives within his code

- When program is executed, a parallel region will be created on the “fork and join” model

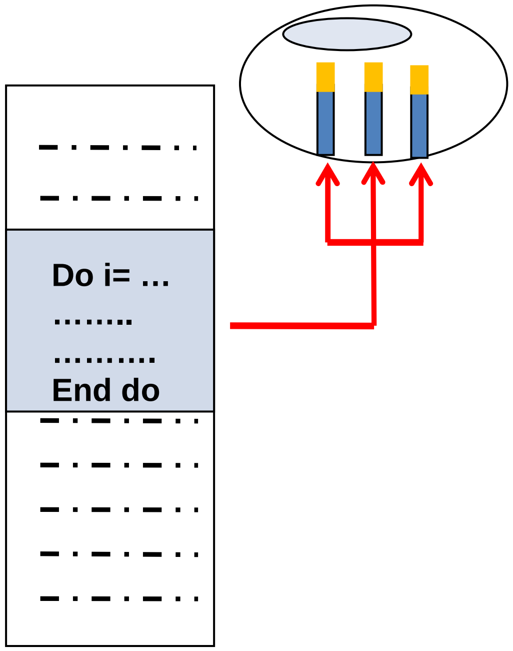


OpenMP: General Concepts

- An OpenMP program is an alternation of sequential and parallel regions
- A sequence region is always executed by the master task
- A parallel region can be executed by several tasks at the same time

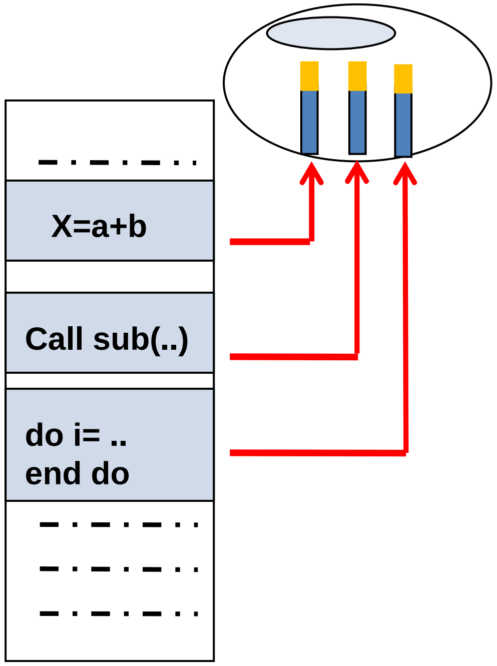


OpenMP: General Concepts



Do i= …
……..
……….
End do

Looplevel parallelism

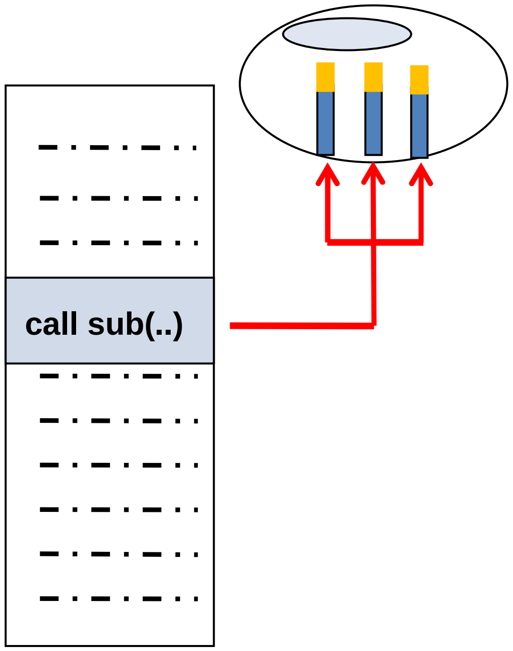


X=a+b

Call sub(..)

do i= ..
end do

Parallel sections

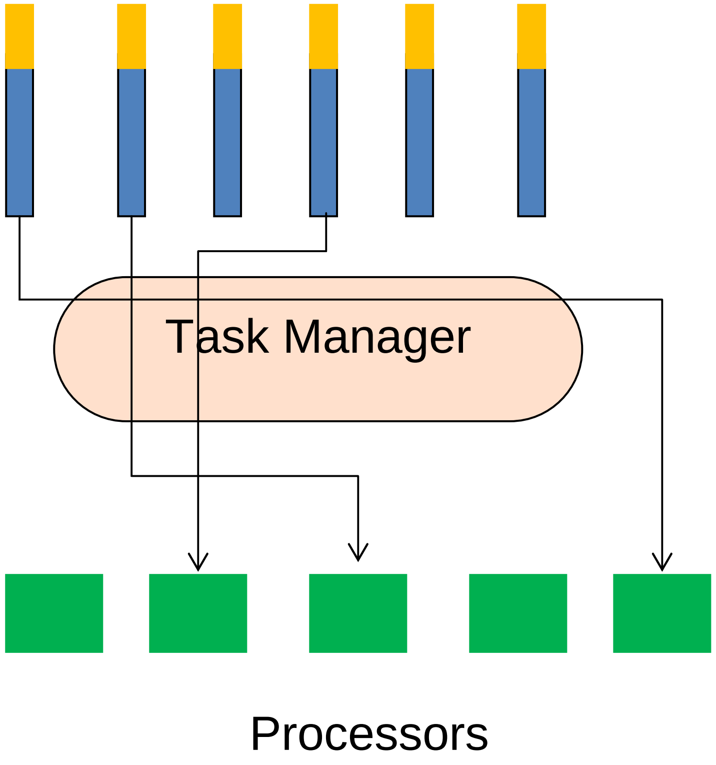


call sub(..)

Parallel procedure (orphaning)

OpenMP: General Concepts

- A task is affected to a processor by the Operating System



OpenMP Basics: Parallel region

- inside a parallel region:
  - ❑ by default, variables are shared
  - ❑ all concurrent task execute the same code
- there is a default synchronization barrier at the end of a parallel region

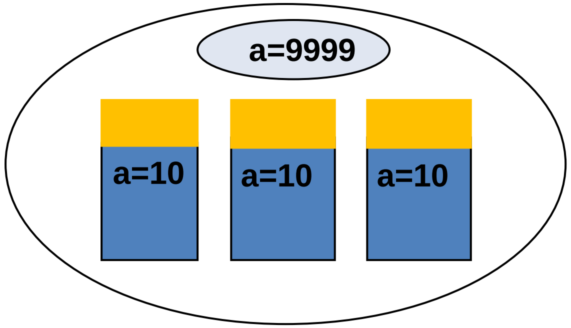
```
Program parallel
            use OMP_LIB
            implicit none
            real ::a
            logical ::p

            a=9999.  ; p= false.
            !$OMP PARALLEL
                 !$ p = OMP_IN_PARALLEL()
                 print *, “A value is :”,a &
                                       “; p value is:
                 ”,p
            !$OMP END PARALLEL
end program parallel
```

```
> export OMP_NUM_THREADS=3; a. out;
> A value is 9999.  ; p value is: T
> A value is 9999.  ; p value is: T
> A value is 9999.  ; p value is: T
```

OpenMP Basics: Parallel region

- By using the DEFAULT clause one can change the default status of a variable within a parallel region
- If a variable has a private status (PRIVATE) an instance of it (with an undefined value) will exist in the stack of each task.

```
Program parallel
          use OMP_LIB
          implicit none
          real ::a
          a=9999.
          !$OMP PARALLEL DEFAULT(PRIVATE)
              a=a+10.
              print *, "A value is : ",a
          !$OMP END PARALLEL
end program parallel
```



```
> export OMP_NUM_THREADS=3; a. out;
> A value is :  10
> A value is :  10
> A value is :  10
```

OpenMP Basics: Synchronizations

- The **BARRIER** directive synchronizes all threads within a parallel region
- Each task waits that all tasks have reached this synchronization point before continuing its execution

```
program parallel
implicit none
real,allocatable,dimension(:) :: a, b
integer :: n, i
 n = 5
!$OMP PARALLEL
      !$OMP SINGLE
                allocate(a(n),b(n))
      !$OMP END SINGLE
      !$OMP MASTER
                read(9) a(1:n)
      !$OMP END MASTER
      !$OMP BARRIER
      ! $OMP DO SCHEDULE(STATIC)
                do i = 1, n
                b(i) = 2.*a(i)
                end do
      !$OMP SINGLE
                deallocate(a)
      !$OMP END SINGLE NOWAIT
!$OMP END PARALLEL
print *, "B vaut : ", b(1:n)
end program parallel
```

OpenMP, Conclusion

- Explicit Parallelism and Synchronisation

OpenMP Is Not:

- Meant for distributed memory parallel
- Necessarily implemented identically by all vendors
- Guaranteed to make the most efficient use of shared memory

MPI, Message Passing Interface

- Library specification for message-passing
- Proposed as a standard
- High performance on both massively parallel machines and on workstation clusters
- Supplies many communication variations and optimized functions for a wide range of needs
- Helps the production of portable code, for
  - ❑ distributed-memory multiprocessor machine
  - ❑ a shared-memory multiprocessor machine
  - ❑ a cluster of workstations

MPI, Message Passing Interface

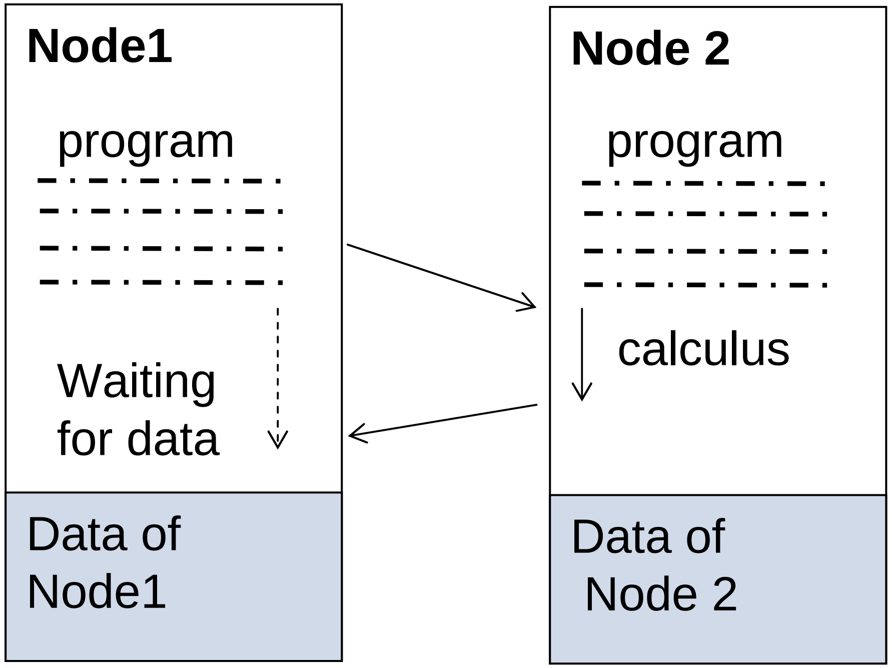
- MPI is a specification, not an implementation
  - MPI has Language Independent Specifications (LIS) for the function calls and language bindings
- Implementations for
  - C, C++, Fortran
  - Python
  - Java

MPI, Message Passing Interface

- MPI is a collection of functions, handling:
  - Communication contexts
  - Point to Point communications
    - Blocking
    - Non blocking
    - Synchronous or Asynchronous.
  - Collectives Communications
  - Data Templates (MPI Datatype)
  - Virtual Topologies
  - Parallel I/O
  - Dynamic management of processes (spawn, semaphores, critical sections…)
  - Remote Direct Memory Access (high troughput, low latency)

MPI Basics

- The overwhelmingly most frequently used MPI commands are variants of
  - □ *MPI_SEND() to send data*
  - □ *MPI_RECV() to receive it.*



- There are several blocking, synchronous, and non-blocking varieties.

MPI Principles Behind

- Design to please all vendors in the consortium:
  - ❑ As many primitives as vendors optimizations

- Design to optimize:
  - ❑ Copy, or Latency
  - ❑ Zero Copy Attempt, Buffer Management
  - ❑ Programmer's Choice vs. Dynamic, Adaptive

Message Passing Interface

- Difficulties:
  - Application is now viewed as a graph of communicating processes. And each process is :
    - Written in a standard sequential language (Fortran, C, C++)
    - All variables are private (no shared memory) and local to each process
    - Data exchanges (communications) between processes are explicit : call of functions or subroutines
  - Mapping of the processes graph onto the real machine (one or several processes on one processor)
  - Too low level

MPI – low level

- The user has to manage:
  - the communication and the synchronization between processes
  - data partitioning and distribution
  - mapping of processes onto processors
  - input/output of data structures
- It becomes difficult to widely exploit parallel computing
- The “easy to use” goal is not accomplished

Main MPI problems for Modern Parallel Computiing

- Too Primitive (no Vertical Frameworks)
- Too static in design
- Too complex interface (API)
  - More than 200 primitives and 80 constants
- Too many specific primitives to be adaptive
  - `Send`, `Bsend`, `Rsend`, `Ssend`, `Ibsend`, etc.
- Typeless (Message Passing rather than RMI)
- Manual management of complex data structures

Languages, Conclusion

- Program remains too static in design
  - Do not offer a way to use new resources that appears at runtime
- Bound to a given distributed system (cluster)
  - Hard to cross system boundaries

Traditional Parallel Computing & HPC Solutions

- Parallel Computing
  - ❑ Principles
  - ❑ Parallel Computer Architectures
  - ❑ Parallel Programming Models
  - ❑ Parallel Programming Languages
- Grid Computing
  - ❑ Multiple Infrastructures
  - ❑ Using Grids
  - ❑ P2P
  - ❑ Clouds
- Conclusion

The Grid Concept

Rational

Computer Power

is like

Electricity

Can hardly be stored if not used

Solution

One vast computational resource

1. Global management,
2. Mutual sharing of the resource

- ► The Grid is a service for sharing computer power and data storage capacity over the Internet.

The Grid Concept

Rational
Computer Power is like Electricity Can hardly be stored if not used

Solution
One vast computational resource 1. Global management, 2. Mutual sharing of the resource

- However CPU cycles are harder to share than electricity
  - ❑ Production cannot be adjusted
  - ❑ Cannot really be delivered where needed
  - ❑ Not fully interoperable:
    - Incompatible Hardware
    - Multiple Administrative Domains

Original Grid Computing

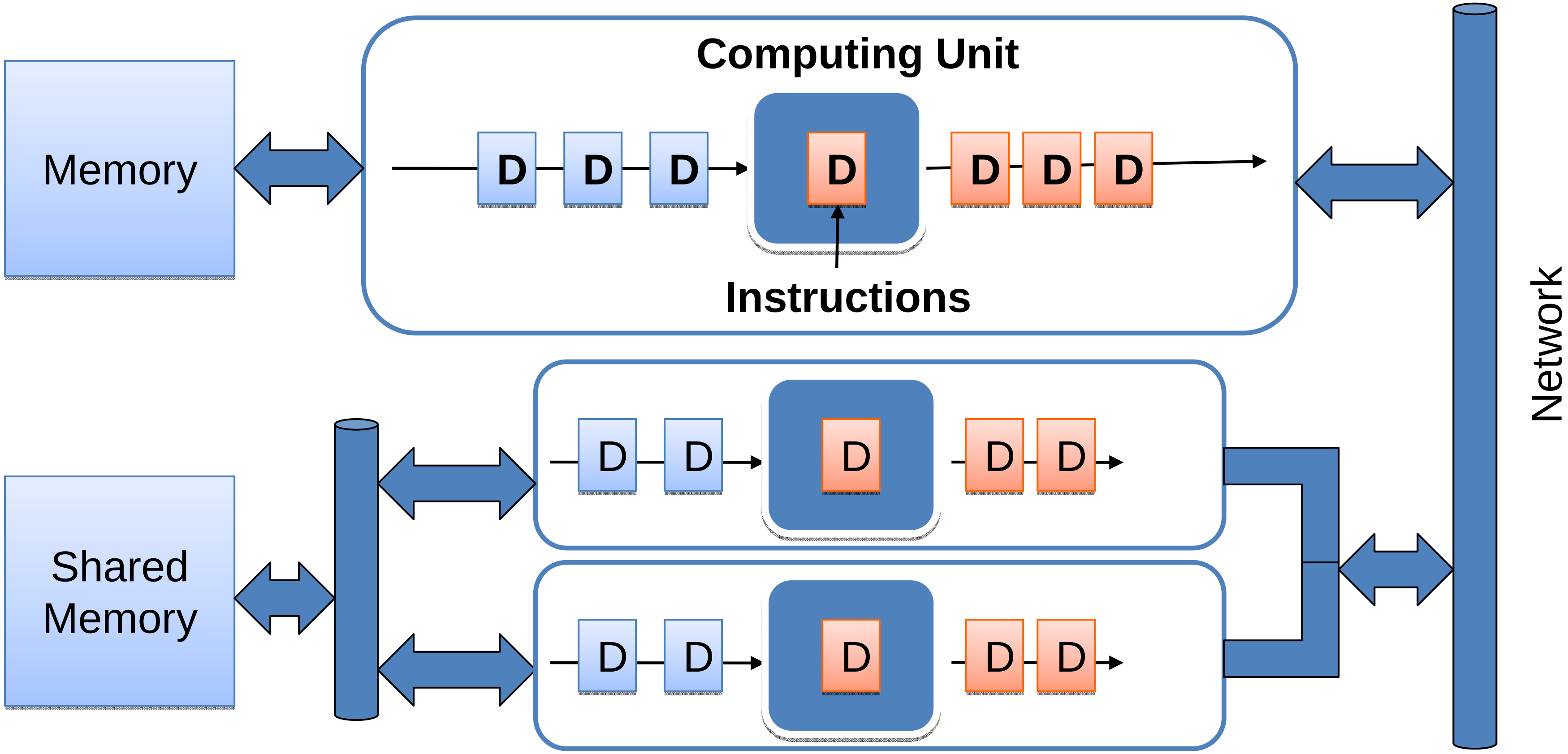
A Grid is a Distributed System

- Coordinate resources (no centralized scheduling) belonging to different organizations and domains.
- Provide security and respect all local policies
- Use standard interfaces and protocols, promote open standard such as TCP/IP, X509, Ldap, Ssl, ...
- Insure complex quality of services : resources co-allocation, handles replicates, reliability, ...
- Communications between different grids must be possible via one standard “inter-grid” protocol

Grid

Multiple Instructions Multiple Data Streams

Multiple heterogeneous computers with their own memory connected through a network



Grid Computing: Fundamentals

Why using Grid Computing?

- ► Optimizing the use of resources
  - ❑ Running a job on a remote resource (machine)
    - Application should be able to execute remotely
    - The remote machine should meet all the hardware and software requirements imposed by the application
  - ❑ Use of Idle CPUs of desktops and servers machines in the enterprise
  - ❑ Use of available disk storage on the machines of the enterprise

Reduced Cost, Increased Capability & Flexibility

Grid Computing: Fundamentals

How to use Grid Computing?

- Global (enterprise-wide Job Scheduling):
  - ❑ Define job priorities
  - ❑ Schedule jobs on machines with low utilization
  - ❑ Split jobs in sub-jobs and schedule on the grid

- Manage SLA, QoS with scheduling strategies to meet deadlines, contracted SLA

Grid Computing: Fundamentals

How to use Grid Computing?

- Parallel computation
  - Application divided in several jobs executed in parallel
  - Communication issues
  - Scalability issues
    - The scalability of a system decreases when the amount of communication increases

Grid Computing: Fundamentals

How to use Grid Computing?

- Virtualizing resources and organizations for collaboration
  - ❑ Putting together heterogeneous systems to create a large virtual computing system
  - ❑ Sharing of resources like software, data, services, equipments, licenses, etc.
  - ❑ Implementing priority and security policies

Grid Computing

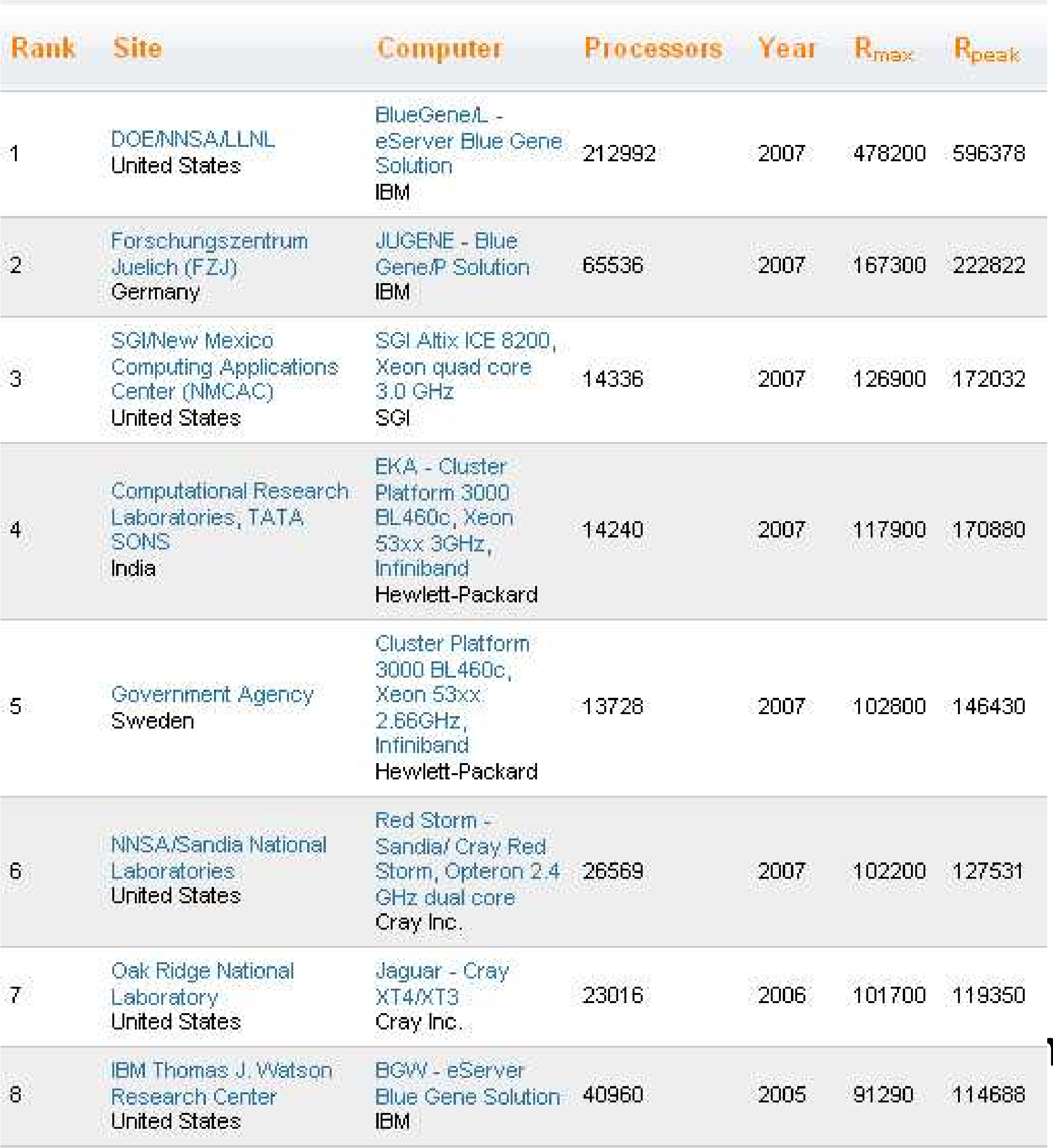
Different kinds of Grids

- Computing Grid:
  - Aggregate computing power
- Information Grid:
  - Knowledge sharing
  - Remote access to Data owned by others
- Storage Grid:
  - Large scale storage
  - Can be internal to a company

The multiple GRIDs

- Scientific Grids :
  - Parallel machines, Clusters
  - Large equipments: Telescopes, Particle accelerators, etc.
- Enterprise Grids :
  - Data, Integration: Web Services
  - Remote connection, Security
- Internet Grids (miscalled P2P grid):
  - Home PC: Internet Grid (e.g. SETI@HOME)
- Intranet Desktop Grids
  - Desktop office PCs: Desktop Intranet Grid

Top 500



Rank	Site	Computer	Processors	Year	R_{max}	R_{peak}
1	DOE/NNSA/LLNL United States	BlueGene/L - eServer Blue Gene Solution IBM	212992	2007	478200	596378
2	Forschungszentrum Juelich (FZJ) Germany	JUGENE - Blue Gene/P Solution IBM	65536	2007	167300	222822
3	SGI/New Mexico Computing Applications Center (NMCAC) United States	SGI Altix ICE 8200, Xeon quad core 3.0 GHz SGI	14336	2007	126900	172032
4	Computational Research Laboratories, TATA SONS India	EKA - Cluster Platform 3000 BL460c, Xeon 53xx 3GHz, Infiniband Hewlett-Packard	14240	2007	117900	170880
5	Government Agency Sweden	Cluster Platform 3000 BL460c, Xeon 53xx 2.66GHz, Infiniband Hewlett-Packard	13728	2007	102800	146430
6	NNSA/Sandia National Laboratories United States	Red Storm - Sandia/ Cray Red Storm, Opteron 2.4 GHz dual core Cray Inc.	26569	2007	102200	127531
7	Oak Ridge National Laboratory United States	Jaguar - Cray XT4/XT3 Cray Inc.	23016	2006	101700	119350
8	IBM Thomas J. Watson Research Center United States	BGW - eServer Blue Gene Solution IBM	40960	2005	91290	114688

http://www.top500.org

max and **Rpeak** values are in GFlops

Notes on Top 500 Benchmarks

- The HPL (High-Performance Linpack) used as a benchmark: problem size yielding the highest performance, often the largest problem size that will fit in memory.
- The HPL benchmark provides the following information:
- Rpeak: The theoretical maximum FLOPS for the system determined by multiplying the floating-point operations per clock cycle, the CPU clock, and the number of processors.
- Rmax: The maximum number of FLOPS achieved for that problem size.
- Nmax: The matrix size
- N1/2: The problem size achieving 50% of Rmax. A low N1/2 shows a robust system delivering strong performance on a broad range of problem sizes.

Typology of Big Machines

MPP: Message Passing Parallelism

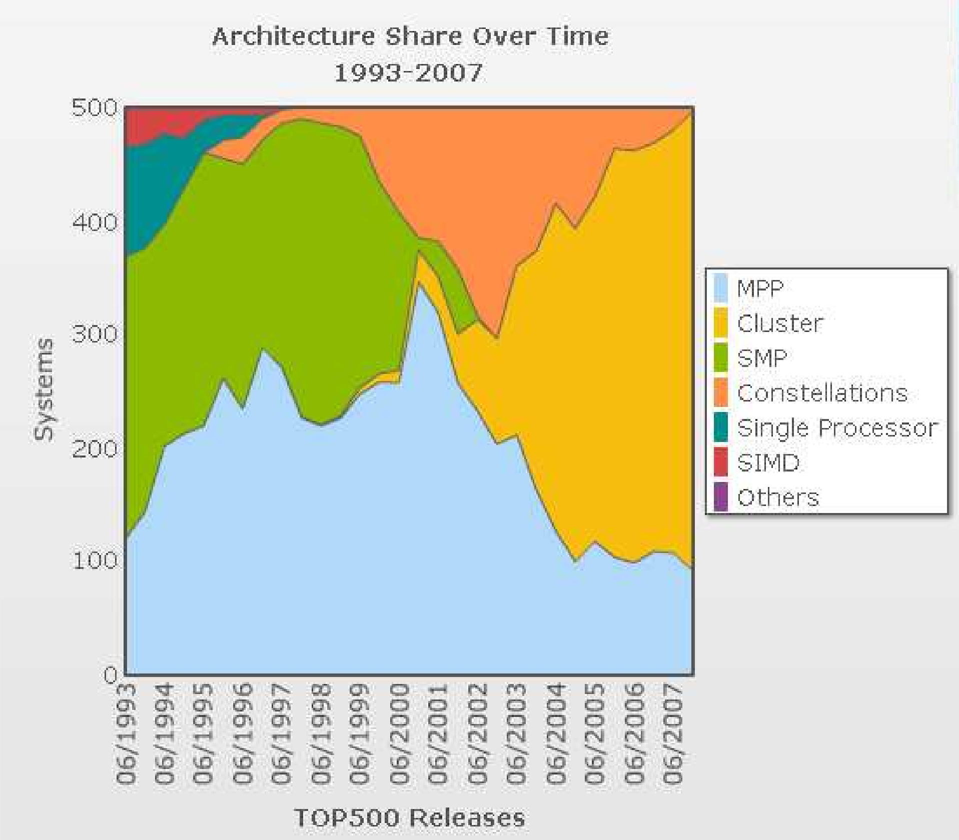
Cluster: MPP Commodity Procs + Interconnects

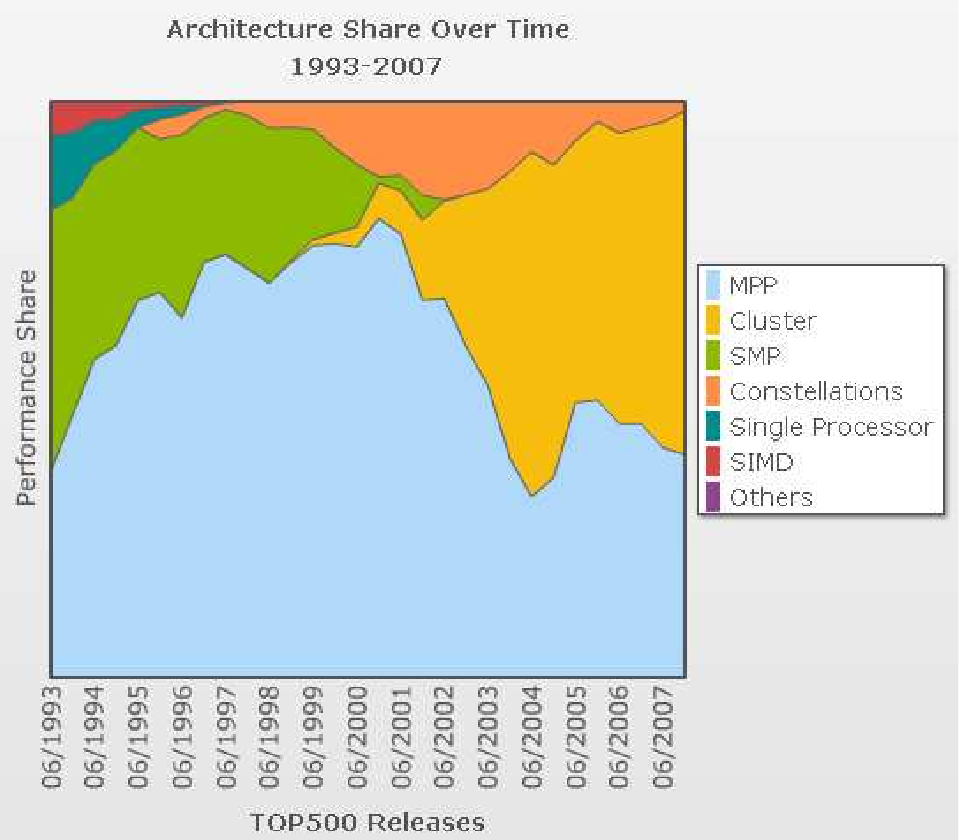
SMP: Shared Memory Machines

Constellation: A combination of SMPs or MPPs

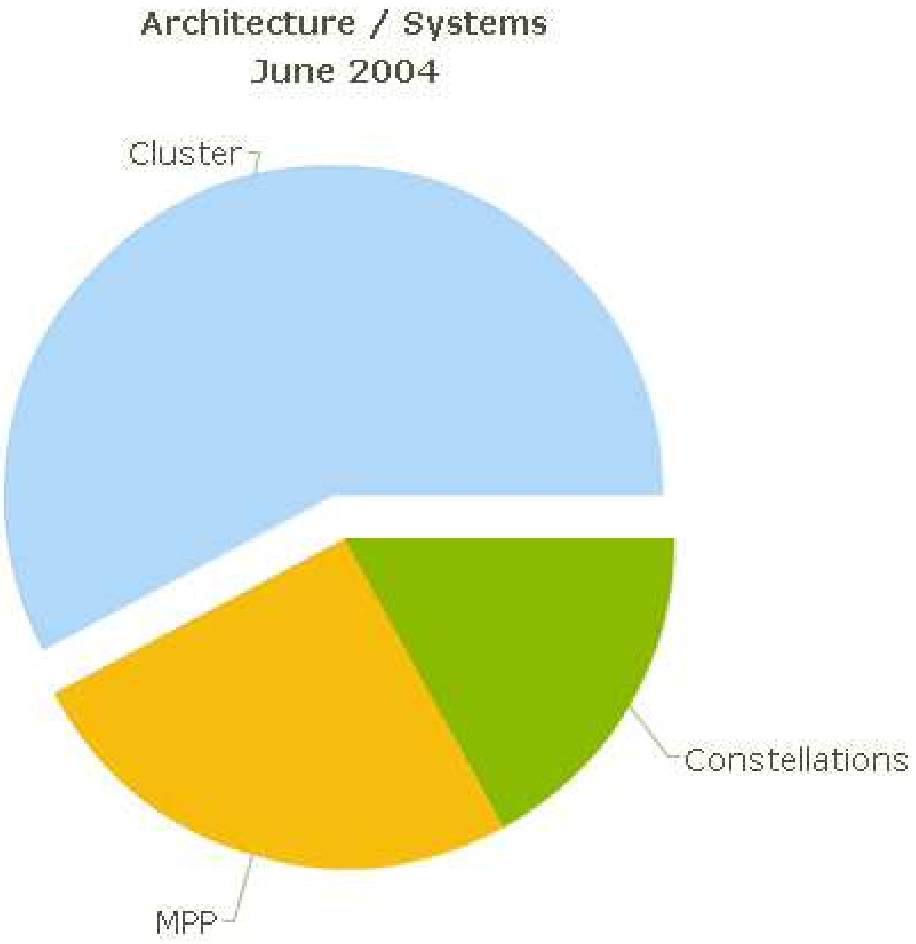
SIMD: Vector Machine

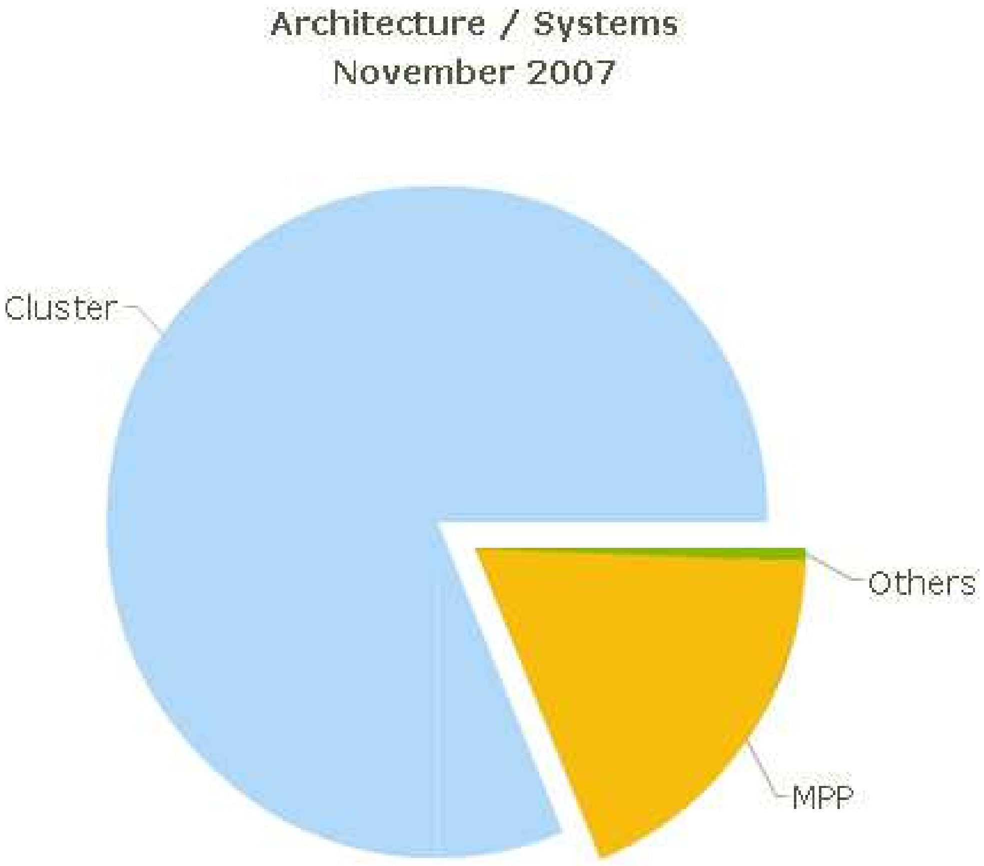
Top 500: Architectures



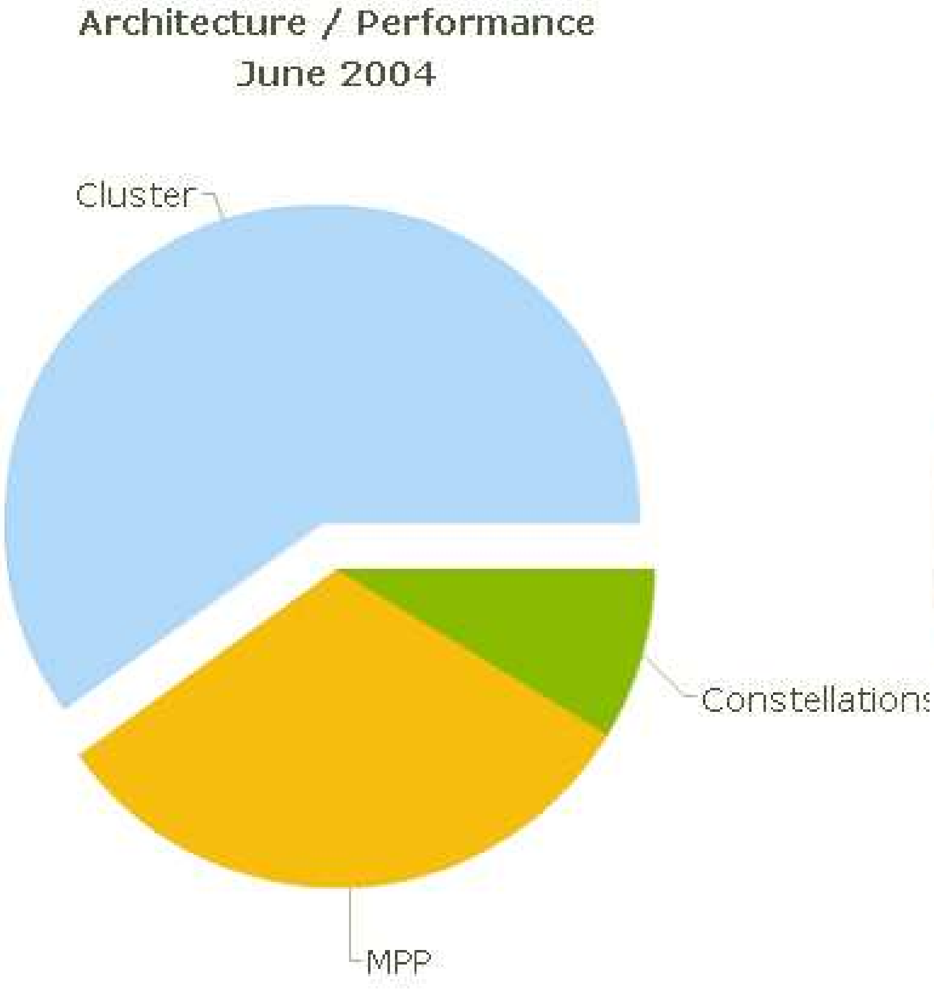


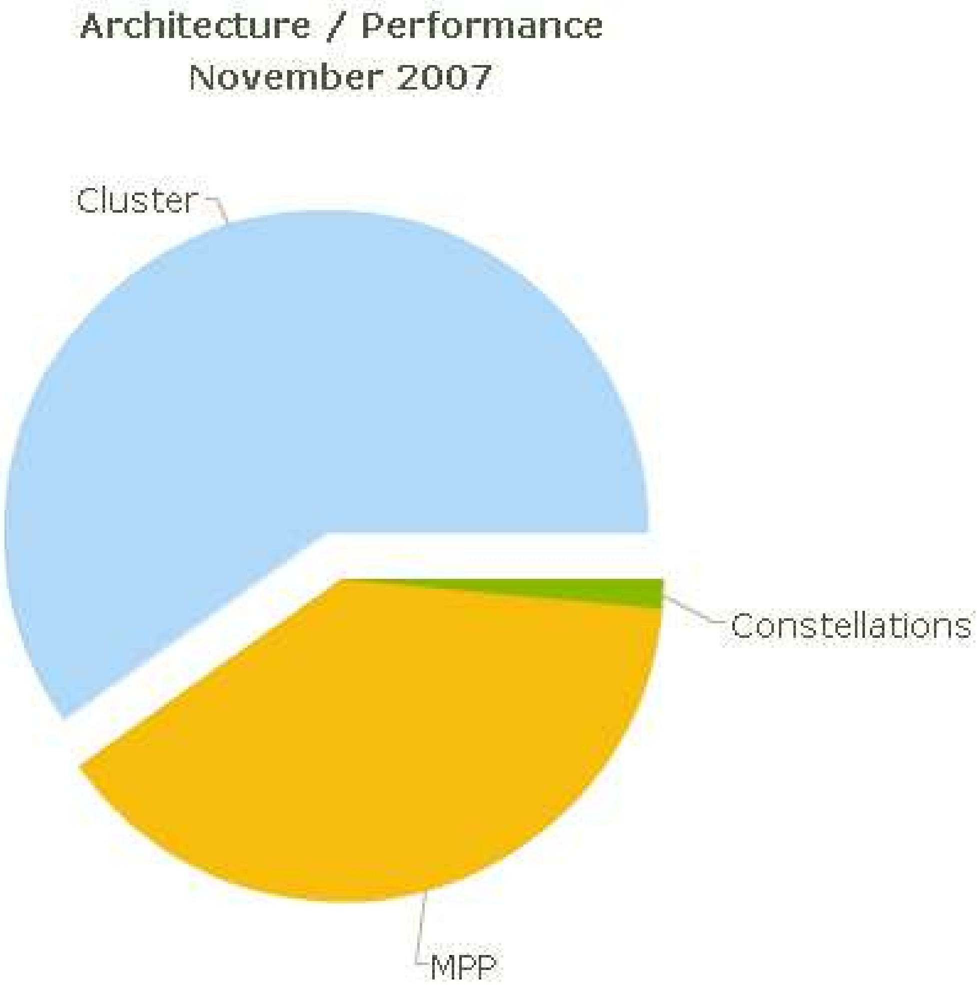
Top 500: Architectures



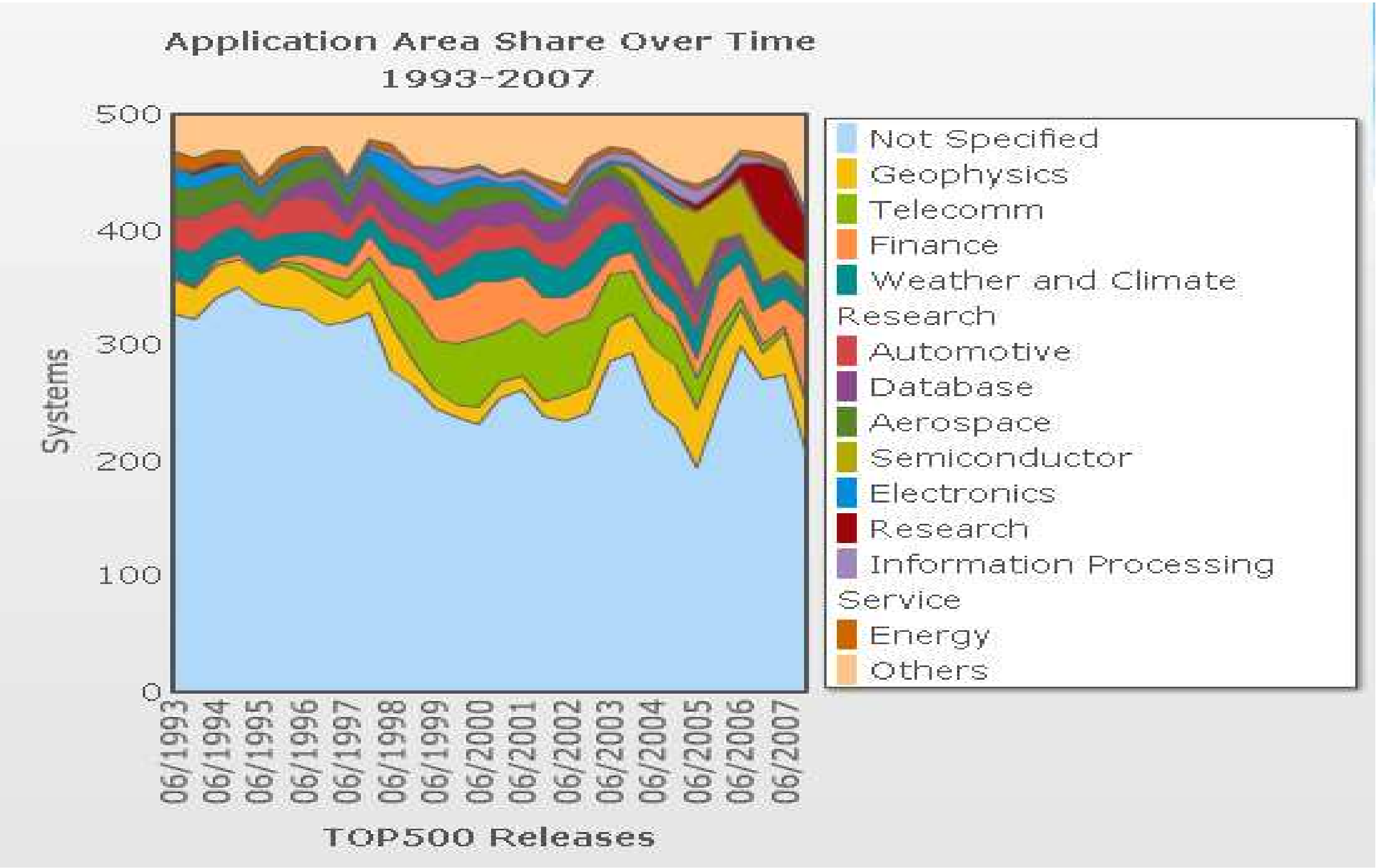


Top 500: Architectures

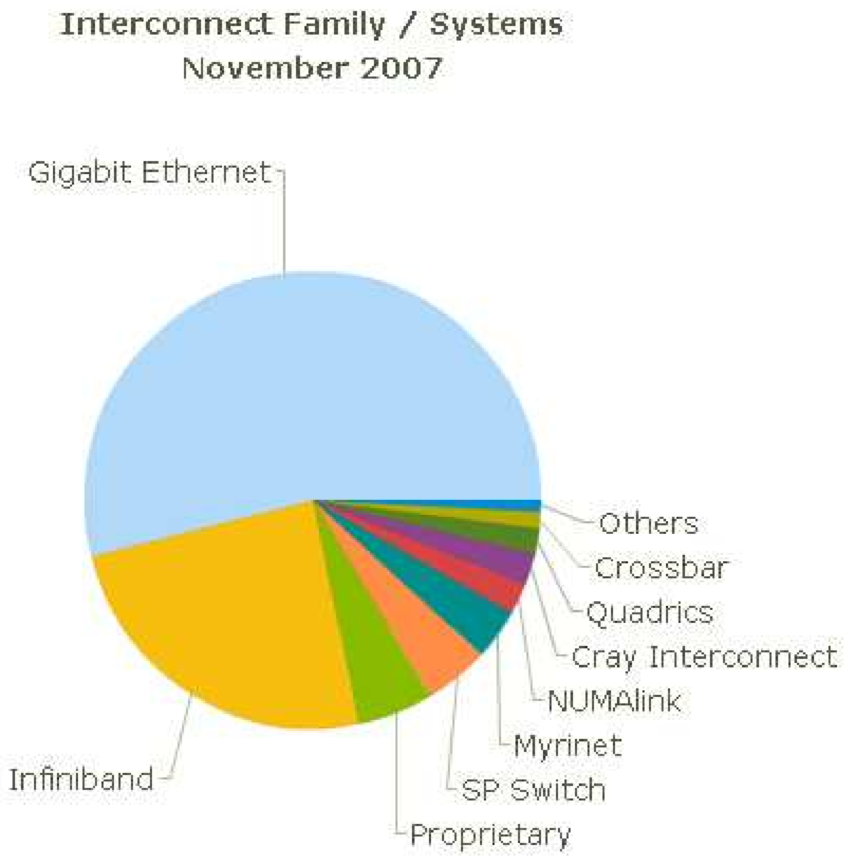


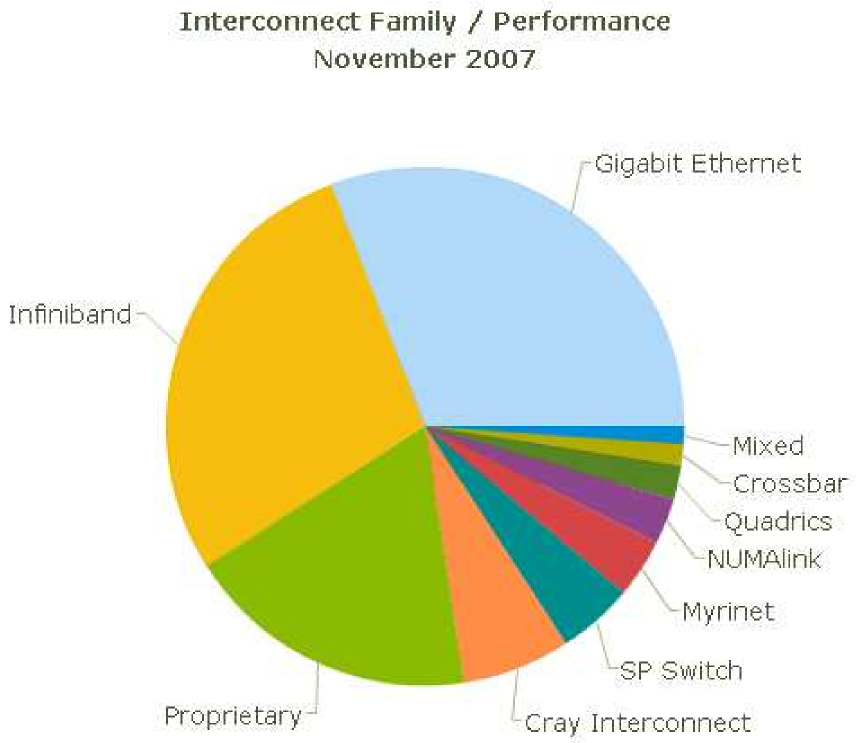


Top 500: Applications

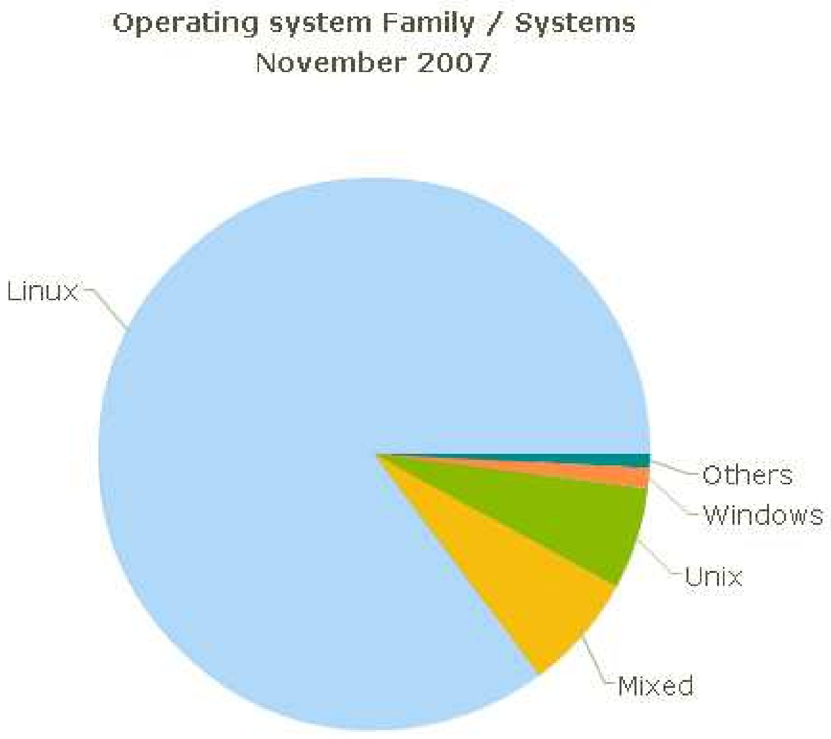


Top 500: Interconnect Trend





Top 500: Operating Systems



Grid, Conclusion

- The goal to present a vast computational resource is not completely reached.
  - Still a system with boundaries and limitations
- Only a grid instance can be seen as a computational resource
- Using different grid instances is not transparent
  - Need for virtualization at middleware level
- Too static design from application POV
  - A Grid is not meant to adapt itself to an application

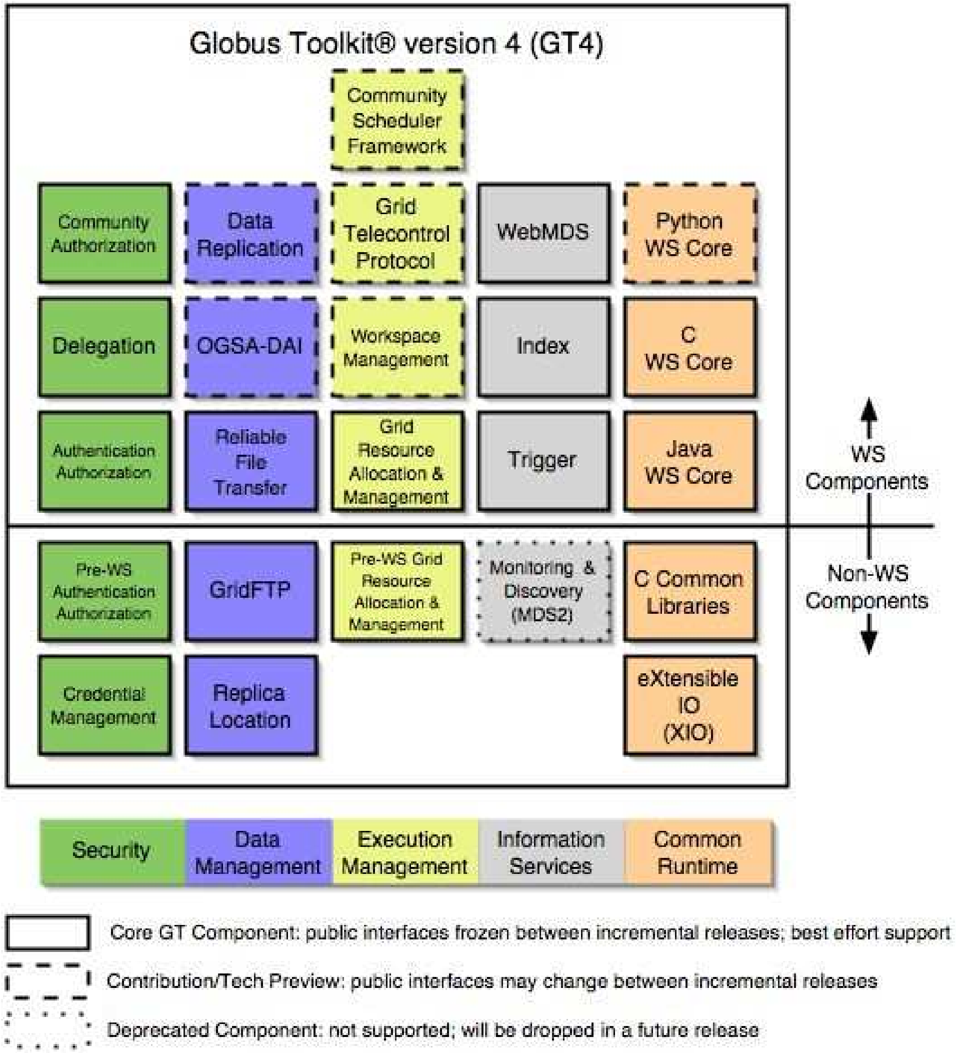
Traditional Parallel Computing & HPC Solutions

- ► Parallel Computing
  - ❑ Principles
  - ❑ Parallel Computer Architectures
  - ❑ Parallel Programming Models
  - ❑ Parallel Programming Languages
- ► Grid Computing
  - ❑ Multiple Infrastructures
  - ❑ Using Grids
  - ❑ P2P
  - ❑ Using Clouds
- ► Conclusion

The Globus Toolkit

- **A Grid development environment**
  - ❑ Develop new OGSA-compliant Web Services
  - ❑ Develop applications using Java or C/C++ Grid APIs
  - ❑ Secure applications using basic security mechanisms
- **A set of basic Grid services**
  - ❑ Job submission/management
  - ❑ File transfer (individual, queued)
  - ❑ Database access
  - ❑ Data management (replication, metadata)
  - ❑ Monitoring/Indexing system information
- **Tools and Examples**
- **The prerequisites for many Grid community tools**

The Globus Toolkit



Globus Toolkit® version 4 (GT4)

Community Scheduler Framework

Community Authorization | Data Replication | Grid Telecontrol Protocol | WebMDS | Python WS Core

Delegation | OGSA-DAI | Workspace Management | Index | C WS Core

Authentication Authorization | Reliable File Transfer | Grid Resource Allocation & Management | Trigger | Java WS Core

WS Components

Non-WS Components

Pre-WS Authentication Authorization | GridFTP | Pre-WS Grid Resource Allocation & Management | Monitoring & Discovery (MDS2) | C Common Libraries

Credential Management | Replica Location | eXtensible IO (XIO)

Security | Data Management | Execution Management | Information Services | Common Runtime

Core GT Component: public interfaces frozen between incremental releases; best effort support

Contribution/Tech Preview: public interfaces may change between incremental releases

Deprecated Component: not supported; will be dropped in a future release

The Globus Toolkit

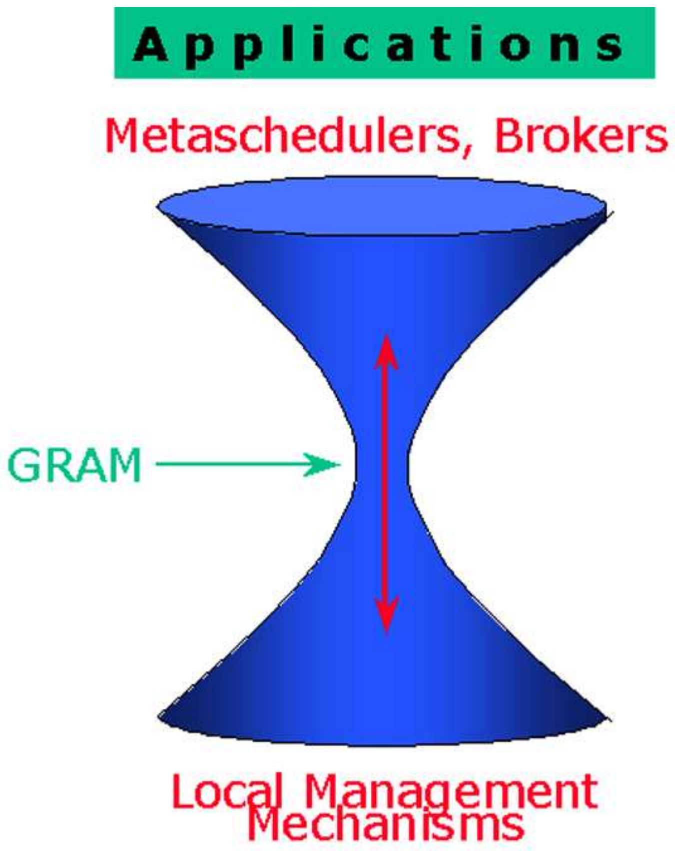
- **Areas of Competence**
  - □ “Connectivity Layer” Solutions
  - □ “Resource Layer” Solutions
  - □ “Collective Layer” Solutions

The Globus Toolkit

GRAM - Basic Job Submission and Control Service

- A uniform service interface for remote job submission and control

- GRAM is *not* a scheduler
  - it can be used as *either* an interface to a scheduler *or* the interface that a scheduler uses to submit a job to a resource.



How To Use the Globus Toolkit

- **By itself, the Toolkit has surprisingly limited end-user value.**
  - There’s very little user interface material there.
  - You can’t just give it to end users (scientists, engineers, marketing specialists) and tell them to do something useful!
- **The Globus Toolkit is useful to system integrators.**
  - You’ll need to have a specific application or system in mind.
  - You’ll need to have the right expertise.
  - You’ll need to set up prerequisite hardware/software.
  - You’ll need to have a plan…

Traditional Parallel Computing & HPC Solutions

- ► Parallel Computing
  - ❑ Principles
  - ❑ Parallel Computer Architectures
  - ❑ Parallel Programming Models
  - ❑ Parallel Programming Languages
- ► Grid Computing
  - ❑ Multiple Infrastructures
  - ❑ Using Grids
  - ❑ P2P
  - ❑ Clouds
- ► Conclusion

Peer to Peer

- What is a P2P system?
  - A system where all participants are equals
  - A system which uses the resources of the enterprise, of the Internet
- Structured
  - Peers are associated using an algorithm (Distributed Hash Table) and the placement of resources is controlled
- Unstructured
  - Peers are “randomly” associated and the resources randomly distributed
- P2P deals with 2 resources
  - Files/Data : P2P File Sharing
  - CPUs : Edge Computing or Global Computing

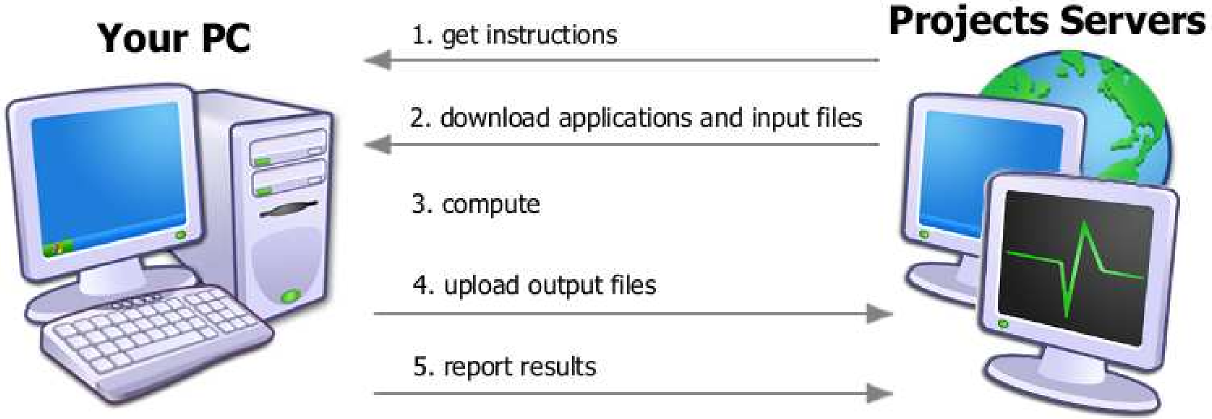
P2P Architectures and techniques

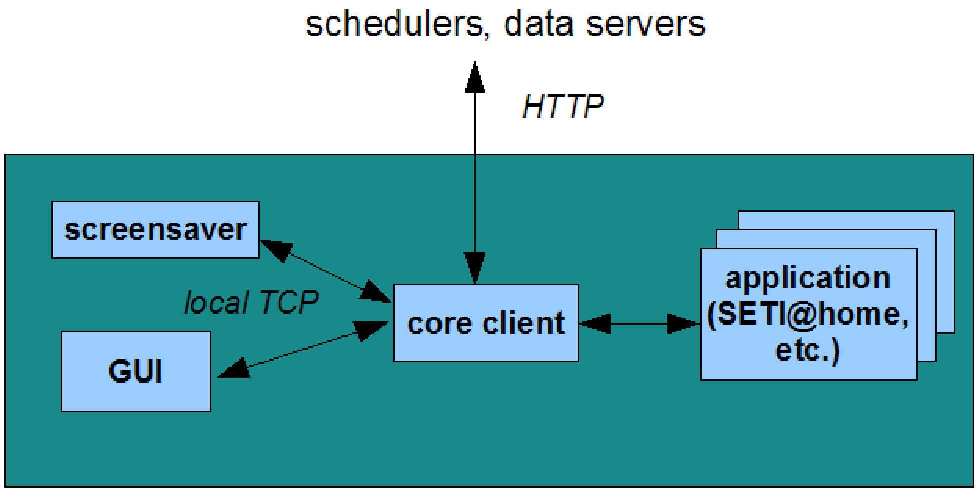
Boinc (*@home)

- "An open source platform for volunteer computing"
- Internet Computing
- Master-Slave applications
  - Servers have tasks to be performed
  - Clients connect to servers to get work
  - No client-to-client communication

P2P Architectures and techniques

Boinc (*@home)





http://boinc.berkeley.edu

P2P Architectures and techniques

Condor

- Workload management system for compute-intensive jobs
- Provides
  - Job queuing mechanism
  - Scheduling policy
  - Resource monitoring and management
- Matchmaking
  - A file indicates resources available
  - When new resources are needed:
    Condor dynamically provides the corresponding resources

http://www.cs.wisc.edu/condor/

JXTA (Juxtapose)

- Open source p2p protocol specification
- Started by Sun Microsystems in 2001

- Set of open protocols to allow any devices to communicate in a P2P manner
- Handles NAT, firewalls…
- It is a low level specification
  - ❑ Only provides the infrastructure
  - ❑ No specific mechanism for programming the application

P2P, Conclusion

- Resources' pool size is dynamic
  - Can adapt to application needs
  - Best effort most of the time, QoS needed
- Resources are volatile
  - Need for fault-tolerant applications
- No real industrial vendors

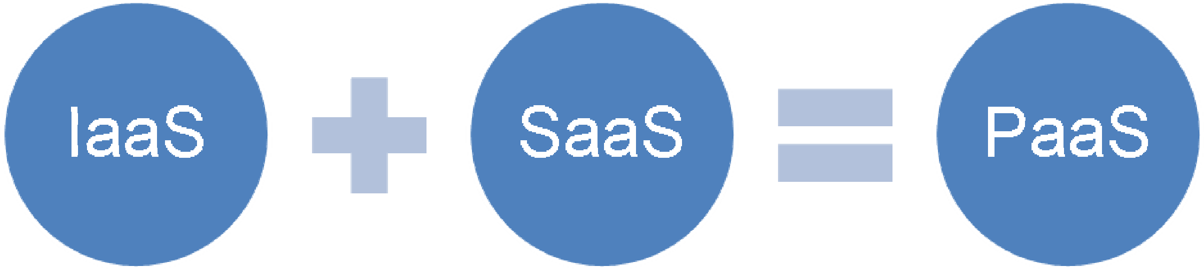
Traditional Parallel Computing & HPC Solutions

- ► Parallel Computing
  - ❑ Principles
  - ❑ Parallel Computer Architectures
  - ❑ Parallel Programming Models
  - ❑ Parallel Programming Languages
- ► Grid Computing
  - ❑ Multiple Infrastructures
  - ❑ Using Grids
  - ❑ P2P
  - ❑ Clouds
- ► Conclusion

Cloud Computing

Cloud computing is a label for the subset of grid computing that includes utility computing and other approaches to the use of shared computing resources **(Wikipedia)**

IaaS	SaaS	PaaS
Infrastructure As A Service	Software as A Service	Platform as A Service



IaaS + SaaS = PaaS

Some Clouds

- Peer to Peer File sharing: Bit torrent
- Web based Applications:
  - Google Apps
  - Facebook
- New Microsoft OS with cloud computing applications
- Web Based Operating Systems
  - http://icloud.com/
- Compute and Store Infrastructure: IaaS
  - Amazon EC2 S3

Cloud Computing

- Perceived benefits
  - Easy to deploy
  - Cheap (if not used peak 100 %)
    - Pay per use model
    - Outsourcing, reduce in-house costs
  - Infinite capacities (storage, computation, …)

- Challenges (same as grid)
  - Security
  - Performance
  - Availability
  - Integrate with existing softwares
  - Customization

Cloud Computing

- **What customers want from cloud computing:**
  - ❑ Competitive pricing
  - ❑ Performance assurances (SLA)
  - ❑ Understand my business & industry
  - ❑ Ability to move cloud offerings back on-premise

Hype vs Reality

Hype	Reality
All of corporate computing will move to the cloud.	Low-priority business tasks will constitute the bulk of migration out of internal data centers.
The economics are vastly superior.	Cloud computing is not yet more efficient than the best enterprise IT departments.
Mainstream enterprises are using it.	Most current users are Web 2.0-type companies (early adopters)
It will drive IT capital expenditures to zero.	It can reduce start-up costs (particularly hardware) for new companies and projects.
It will result in an IT infrastructure that a business unit can provision with a credit card.	It still requires a savvy IT administrator, developer, or both.

Source: CFO magazine

Cloud, conclusion

- Another step towards integration of grid computing within application
- Possible to adapt resource to applications
- Several vendors exist, market exists
  - Amazon Ec2, Flexiscale, GoGrid, Joyent,

Traditional Parallel Computing & HPC Solutions

- ► Parallel Computing
  - ❑ Principles
  - ❑ Parallel Computer Architectures
  - ❑ Parallel Programming Models
  - ❑ Parallel Programming Languages
- ► Grid Computing
  - ❑ Multiple Infrastructures
  - ❑ Using Grids
  - ❑ P2P
  - ❑ Clouds
- ► Conclusion

The Weaknesses and Strengths of Distributed Computing

- In any form of computing, there is always a tradeoff in advantages and disadvantages

- Some of the reasons for the popularity of distributed computing :
  - ❑ **The affordability of computers and availability of network access**
  - ❑ **Resource sharing**
  - ❑ **Scalability**
  - ❑ **Fault Tolerance**

The Weaknesses and Strengths of Distributed Computing

- Disadvantages of distributed computing:
  - **Multiple Points of Failures:** the failure of one or more participating computers, or one or more network links, can spell trouble.
  - **Security Concerns**: In a distributed system, there are more opportunities for unauthorized access.
  - **Malicious** worms, viruses, etc.
  - Personal **Identity theft** – social, medical, …
  - Lack of interoperability between Grid Systems

Solutions for Parallel and Distributed Processing (a few of them…)

EXPLICIT DISTRIBUTION AND PARALLELISM		IMPLICIT DISTRIBUTION AND PARALLELISM	
SHARED MEMORY	COMMUNICATING PROCESSES	PROGRAMMING BASED	GUI BASED
ORACLE Coherence hp invent Efficient Java-centric grid ibis	GridGain GRID COMPUTING digipede GIGASPACES	SCALI higher performance computing hadoop	Data Synapse GRIDSYSTEMS

Software Shared Memory (Distributed Shared Memory)

- Emulate a distributed shared memory at *software level*
  - □ `write(key, value)`, `read(key)` **and** `take(key)`

- Many java-based solutions
  - □ Commercial products

ORACLE Coherence — JSR 107 implementation (JCache)

GIGASPACES — Extended implementation of JavaSpaces (Jini)

  - □ Open source solutions
    - SlackSpaces, SemiSpace, PyLinda (Python)…

Communicating processes

- Enable communication between remote processes
  - Message passing : `send(…),receive(…),…`
  - RPC : `func(…),object.foo(…)`
- Many MPIs (open-source and commercial)

SCALI higher performance computing Optimized implementations

hp invent Hardware specific implementations

- In Java : RMI (synchronous only)

Efficient Java-centric grid ibis Fast-RMI implementation (open-source)

Implicit Programming-based

- Parallelism is *predefined* in the solution

- The user writes tasks and applies predefined parallelism patterns (skeletons)
  - Farm, pipeline, map, divide-and-conquer, branch&bound

- Can rely on user-defined methods
  - Split/merge,map/reduce,…

- Commercial Open-source Java solution

GridGain GRID COMPUTING — Annotation-based with split/merge methods

hadoop — Map/Reduce with Distributed File System (~ Google)

Implicit GUI-based

- Tasks are third party applications
  - Parallelism can be deduced from…
    - Parameters (parameters sweeping)
    - Tasks flow

- Create applications from the GUI
    - Visual composition of tasks
    - Parameter sweeping wizards

- Solutions with similar capabilities

digipede Commercial .Net based solution (Windows only)

Data Synapse Commercial multi-platform solution

GRIDSYSTEMS Commercial Open-source multi-platform solution

Conclusion

The need to unify Distribution and Multi-Core

- ► Uniform Framework for
  - ❑ Multi-Threading (for Multicores)
  - ❑ Multi-Processing (for Distribution)

Need for resources virtualization to knit together all available resources

- ► Basic programming model
  - ❑ Asynchronous
  - ❑ Insensibility to Sharing or Not (even if used at implementation)
  - ❑ Taking advantage of multicores, However resisting to Distribution

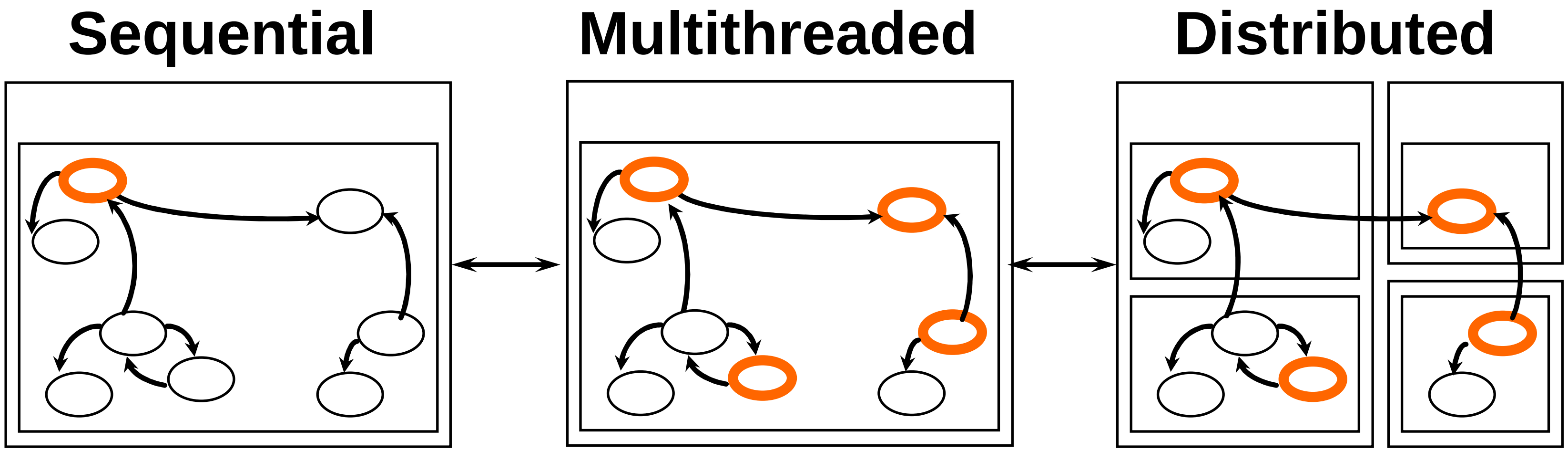
Need for new programming paradigms

Conclusion

The need to unify Distribution and Multi-Core

ProActive
Programming, Composing, Deploying on the Grid

Seamless



Java Middleware for Parallel and Distributed computing

General Tradeoff:
ProActive Design Decision

- Implicit vs. Explicit:
  - Explicit: User definition of Activities
- Copied vs. Shared:
  - Copy (with shared optimization with no behavior consequences)
- Abstract away infrastructure details

Conclusion

Abstracting Away Architecture

- **Basic Programming model**
  - ❑ Independent on the physical architecture
  - ❑ Hardware resources handled by the middleware
  - ❑ Virtualisation of multi-core, multi-processors, servers, clusters
  - ❑ Interoperability with different Grid middleware
  - ❑ Managed Security

- **Vertical Programming Model, Specialized (Master/Workers, Branch&Bound, Skeleton, …)**

Parallelism: Problem / Solution

Embarrassingly Parallel Applications	• Independent Tasks → Master Slave package *Monte Carlo Simulation (in Financial Math, Non Linear Physic, ...)* • Dynamic Generation of Tasks → High-Level Patterns (Skeleton Framework) *Post-Production*
Slightly Communicating Applications	• Branch & Bound package Flow-Shop
Highly Communicating Applications	• Dynamic,Numerical → OO SPMD *Electro-Magnetism, Vibro-Acoustic, Fluid Dynamics* • Unstructured → Basic API with Active Objects and Groups *EDA (Electronic Design Automation for ICs), N-Body*
Non-Java code	• Numerical MPI → Legacy Code Wrapping • Script code → Script Legacy Code Wrapping

Conclusion

Various Applications with Different Needs

- A set of parallel programming frameworks in Java
  - Active Objects (Actors)
  - Master/Worker
  - Branch & Bound
  - SPMD
  - Skeletons
  - Event Programming
  - Mathlab, Scilab
  - A component framework as a reference implementation of the GCM
  - Legacy Code Wrapping, Grid Enabling

ProActive
Programming, Composing, Deploying on the Grid

- To **simplify** the programming of Parallel Applications

Conclusion

Local Machine, Enterprise Servers, Enterprise Grids, SaaS-Clouds

- **Resource Management**
  - ❑ Still the need for In-Enterprise sharing (vs. SaaS, Cloud)
    - **Meta-Scheduler/RM for**
    - **Dynamic scheduling and resource sharing**
    - **Various Data Spaces, File Transfer**

- **Deployment & Virtualization**
  - ❑ A Resource acquisition, virtualization and deployment framework
    Multi-core processors, distributed on Local Area Network (LAN), on clusters and data centers, on intranet and Internet Grids

Conclusion

Needs for Tools for Parallel Programming

► **Parallel Programming needs Tools:**

- **Understand**
- **Analyze**
- **Debug**
- **Optimize**

ProActive
Programming, Composing, Deploying on the Grid

features Graphical User Interface : IC2D

A RCP application composed of plugins for

❑ Visualizing
❑ Analyzing
❑ Debugging
❑ Optimizing ProActive Applications

Solutions for Parallel and Distributed Processing (a few of them…)

EXPLICIT DISTRIBUTION AND PARALLELISM		IMPLICIT DISTRIBUTION AND PARALLELISM	
SHARED MEMORY	COMMUNICATING PROCESSES	PROGRAMMING BASED	GUI BASED
ORACLE Coherence GIGASPACES	SCALI higher performance computing hp invent Efficient Java-centric grid ibis ProActive Programming, Composing, Deploying on the Grid CORE	GridGain GRID COMPUTING hadoop ProActive Programming, Composing, Deploying on the Grid HIGH-LEVEL API	Data Synapse digipede GRIDSYSTEMS ProActive Programming, Composing, Deploying on the Grid SCHEDULER

Backed up by

ActiveEon

SCALE BEYOND LIMITS

- Technical Support
- Training
- Certification
- IT Consulting
- Added Value Software