High Performance Computing - Session 1

An Overview of Parallel Computation

Jayanti Prasad
http://www.iucaa.ernet.in/ jayanti/

Inter-University Centre for Astronomy & Astrophysics
Pune, India (411007)

`November 11, 2011`

Plan of the Talk

- Introduction
    - Why Parallel Computation ?
    - Performance
    - Parallel Problems
- Platforms
    - A Model Computer
    - Parallel Platforms
- Models of Parallel Programming
    - Shared Memory Programming
    - Distributed Memory Programming
    - GPU Programming
- Summary and Conclusions

Why Parallel Computation ?

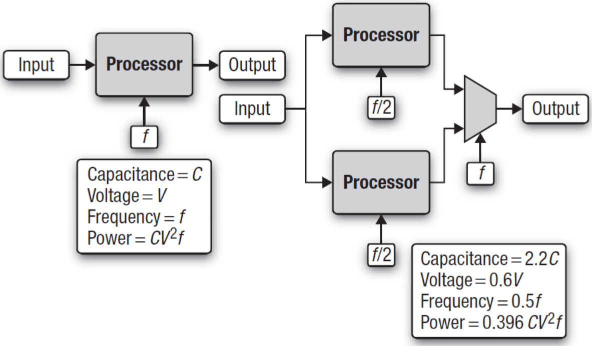
- Solving problems fast (saving time and money !).
- Solving more problems (concurrency !).
- Solving large problems, or problems which are not possible to solve on a single computer (discoveries !).

Examples:

- Pattern matching or search
- Processing large data volume.
- Simulations

Performance

- Processor:
  - Higher clock rate (at present around 3 Ghz).
  - More instructions per clock cycle.
  - More number of transistors (at present around 1 billion).
- `Performance (and complexity) of processors doubles in every 18 months (Moore's law) and making faster processors is difficult due to heating and speed of light problem.`
- Memory:
  - Latency
  - Bandwidth
  - Hierarchy (caches)
- `So far the performance growth due to increase in the clock rate has been 55% and that due to the number of transistors has been 75%.`



The rate at which instructions are retired is the same in these two cases, but the power is much less with two cores running at half the frequency of a single core.

`More cores with slow clock are preferred than one core with fast clock.`

Parallel Problems

- Not all problems are parallel problems.
- In general, there are always some sections of a large problem which can be solved in parallel.
- `The gain in the computational time or speed up`[1] `due to parallel computation for a problem depends on the fraction of the total time the problem spends in the parallel sections (Amdahl's law).`
- In most cases the speed up never varies linearly with the problem size, and the number of processing units.
- If the time taken in communication or Input-Output (IO) is more than the computation time, chances of performance gain due to parallel computation are less.

---

[1] time taken by a single processing unit/time taken by N processing units

Examples

- Scalar Product:

$$S = \sum_{i=1}^{N} A_i B_i$$

- Linear-Algebra: Matrix multiplication

$$C_{ij} = \sum_{k=1}^{M} A_{ik} B_{kj}$$

- Integration:

$$y = 4 \int_0^1 \frac{dx}{1+x^2}$$

- Dynamical Simulations:

$$f_i = \sum_{j=1}^{N} \frac{m_j(\vec{x}_j - \vec{x}_i)}{(|\vec{x}_j - \vec{x}_i|)^{3/2}}$$

How to do parallel computation ?

1. Identify the sections of your problem which are independent (asynchronous) and so can be solved in parallel (concurrently).
2. Map the parallel sections following some efficient scheme (decomposition), on the hardware resources you have, using some software tools.

In general, there is a many to one mapping between the multi-dimensional space of the parallel sections and the computing units.

$$f : I^n - -- > I^p \tag{1}$$

where n is the dimensionality of the "problem space" and p is the dimensionality of the "processing space".

Modern Computer

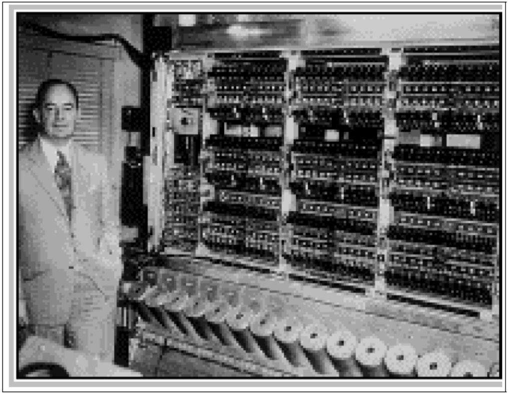
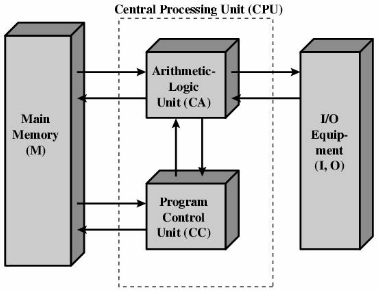


Figure 1: John von Neumann in front of the computer he built at the Institute for Advanced Study in Princeton (Courtesy of the Archives of the Institute for Advanced Study).

Von Neumann Model (Architecture)

- A memory containing both data and Instructions.
- A calculating unit capable of performing both arithmetic and logical operations on the data.
- A control unit which could interpret an instruction retrieved from the memory and select alternative courses of action based on the results of the previous operations.



A typical Blade of Cray CX1

Model Computer

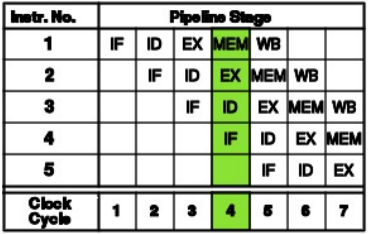
Flynn's taxonomy

- `Single-Instruction, Single-Data (SISD)` - *von Neumann model*.
- `Multiple-Instruction, Single-Data (MISD)`.
- `Single-Instruction, Multiple-Data (SIMD)`.
- `Multiple-Instruction, Multiple-Data (MIMD)`.

- A sequential computer consists of a memory connected to a processor via a datapath and all three components present bottlenecks to the overall processing rate of a computer system.
- New innovations leading to multiplicities in processing units, datapaths, and memory units have been used to addresses these bottlenecks.

Parallel Platforms : Pipeline

- A design technique to increase the instruction throughput (the number of instructions that can be executed in a unit of time).
- Split the processing of a computer instruction into a series of small independent steps, which allows execution of multiple instructions.



Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

Basic five-stage pipeline in a RISC machine (IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back). In the fourth clock cycle (the green column), the earliest instruction is in MEM stage, and the latest instruction has not yet entered the pipeline.

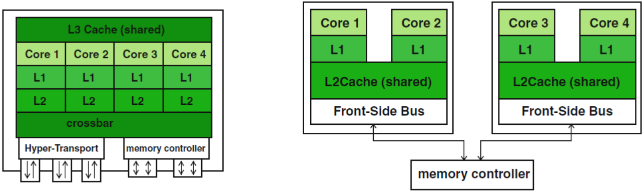
Parallel Platforms : Vector Processors

- A processor that performs one instruction on several data sets is called a vector processor.
- The most common form of parallel computation is in the form of Single Instruction Multiple Data (SIMD) i.e., same computational steps are applied on different data sets.
- Problems which can be broken into small problems for parallelization are called embarrassingly parallel problems e.g., SIMD.

Parallel Platforms : Multi-core Processors

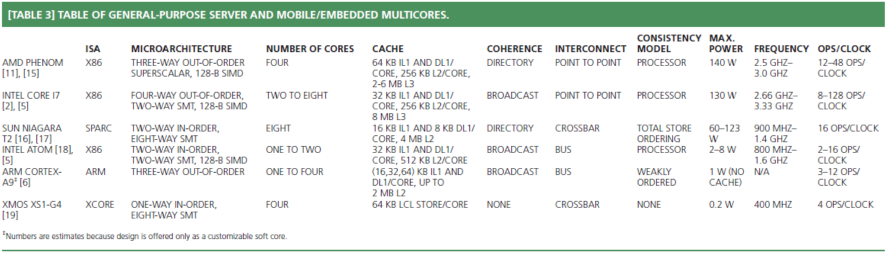
- A single processor can have more than one computation units, called cores, having their own resources for executing instructions independently.
- A multiprocessor system have many processors and each one of them can have more than one cores. Note that just by looking on the motherboard you can count the processors but not the cores.
- A single core can support more than one threads.
- A multi-core processor presents multiple virtual CPUs to the user and operating system.
- Note that in general all the cores of a processor share the main memory and some cache memory.

Parallel Platforms : Multi-core processors



Quad-Core AMD Opteron (*left*) vs. Intel Quad-Core Xeon architecture (*right*) as examples for a hierarchical design

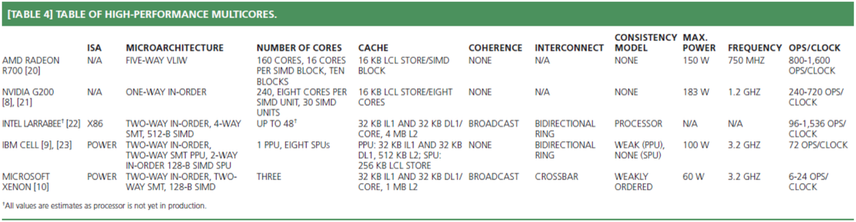
Parallel Platforms: Multi-core processors



[TABLE 3] TABLE OF GENERAL-PURPOSE SERVER AND MOBILE/EMBEDDED MULTICORES.

	ISA	MICROARCHITECTURE	NUMBER OF CORES	CACHE	COHERENCE	INTERCONNECT	CONSISTENCY MODEL	MAX. POWER	FREQUENCY	OPS/CLOCK
AMD PHENOM [11], [15]	X86	THREE-WAY OUT-OF-ORDER SUPERSCALAR, 128-B SIMD	FOUR	64 KB IL1 AND DL1/ CORE, 256 KB L2/CORE, 2-6 MB L3	DIRECTORY	POINT TO POINT	PROCESSOR	140 W	2.5 GHZ–3.0 GHZ	12–48 OPS/ CLOCK
INTEL CORE I7 [2], [5]	X86	FOUR-WAY OUT-OF-ORDER, TWO-WAY SMT, 128-B SIMD	TWO TO EIGHT	32 KB IL1 AND DL1/ CORE, 256 KB L2/CORE, 8 MB L3	BROADCAST	POINT TO POINT	PROCESSOR	130 W	2.66 GHZ–3.33 GHZ	8–128 OPS/ CLOCK
SUN NIAGARA T2 [16], [17]	SPARC	TWO-WAY IN-ORDER, EIGHT-WAY SMT	EIGHT	16 KB IL1 AND 8 KB DL1/ CORE, 4 MB L2	DIRECTORY	CROSSBAR	TOTAL STORE ORDERING	60–123 W	900 MHZ–1.4 GHZ	16 OPS/CLOCK
INTEL ATOM [18], [5]	X86	TWO-WAY IN-ORDER, TWO-WAY SMT, 128-B SIMD	ONE TO TWO	32 KB IL1 AND DL1/ CORE, 512 KB L2/CORE	BROADCAST	BUS	PROCESSOR	2–8 W	800 MHZ–1.6 GHZ	2–16 OPS/ CLOCK
ARM CORTEX-A9‡ [6]	ARM	THREE-WAY OUT-OF-ORDER	ONE TO FOUR	(16,32,64) KB IL1 AND DL1/CORE, UP TO 2 MB L2	BROADCAST	BUS	WEAKLY ORDERED	1 W (NO CACHE)	N/A	3–12 OPS/ CLOCK
XMOS XS1-G4 [19]	XCORE	ONE-WAY IN-ORDER, EIGHT-WAY SMT	FOUR	64 KB LCL STORE/CORE	NONE	CROSSBAR	NONE	0.2 W	400 MHZ	4 OPS/CLOCK

‡Numbers are estimates because design is offered only as a customizable soft core.

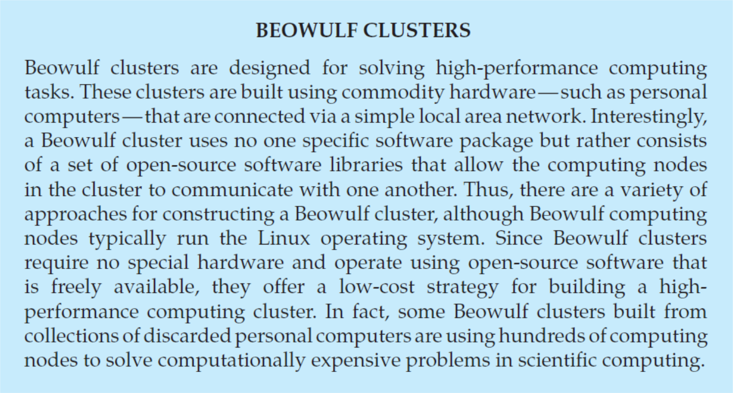


[TABLE 4] TABLE OF HIGH-PERFORMANCE MULTICORES.

	ISA	MICROARCHITECTURE	NUMBER OF CORES	CACHE	COHERENCE	INTERCONNECT	CONSISTENCY MODEL	MAX. POWER	FREQUENCY	OPS/CLOCK
AMD RADEON R700 [20]	N/A	FIVE-WAY VLIW	160 CORES, 16 CORES PER SIMD BLOCK, TEN BLOCKS	16 KB LCL STORE/SIMD BLOCK	NONE	N/A	NONE	150 W	750 MHZ	800-1,600 OPS/CLOCK
NVIDIA G200 [8], [21]	N/A	ONE-WAY IN-ORDER	240, EIGHT CORES PER SIMD UNIT, 30 SIMD UNITS	16 KB LCL STORE/EIGHT CORES	NONE	N/A	NONE	183 W	1.2 GHZ	240-720 OPS/ CLOCK
INTEL LARRABEE† [22]	X86	TWO-WAY IN-ORDER, 4-WAY SMT, 512-B SIMD	UP TO 48†	32 KB IL1 AND 32 KB DL1/ CORE, 4 MB L2	BROADCAST	BIDIRECTIONAL RING	PROCESSOR	N/A	N/A	96-1,536 OPS/ CLOCK
IBM CELL [9], [23]	POWER	TWO-WAY IN-ORDER, TWO-WAY SMT PPU, 2-WAY IN-ORDER 128-B SIMD SPU	1 PPU, EIGHT SPUs	PPU: 32 KB IL1 AND 32 KB DL1, 512 KB L2; SPU: 256 KB LCL STORE	NONE	BIDIRECTIONAL RING	WEAK (PPU), NONE (SPU)	100 W	3.2 GHZ	72 OPS/CLOCK
MICROSOFT XENON [10]	POWER	TWO-WAY IN-ORDER, TWO-WAY SMT, 128-B SIMD	THREE	32 KB IL1 AND 32 KB DL1/ CORE, 1 MB L2	BROADCAST	CROSSBAR	WEAKLY ORDERED	60 W	3.2 GHZ	6-24 OPS/ CLOCK

†All values are estimates as processor is not yet in production.

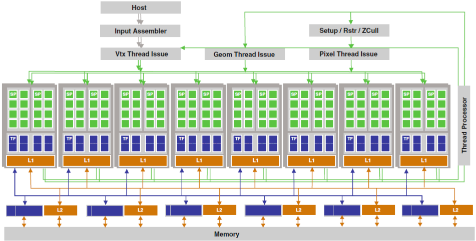
Parallel Platforms : Clusters



BEOWULF CLUSTERS

Beowulf clusters are designed for solving high-performance computing tasks. These clusters are built using commodity hardware—such as personal computers—that are connected via a simple local area network. Interestingly, a Beowulf cluster uses no one specific software package but rather consists of a set of open-source software libraries that allow the computing nodes in the cluster to communicate with one another. Thus, there are a variety of approaches for constructing a Beowulf cluster, although Beowulf computing nodes typically run the Linux operating system. Since Beowulf clusters require no special hardware and operate using open-source software that is freely available, they offer a low-cost strategy for building a high-performance computing cluster. In fact, some Beowulf clusters built from collections of discarded personal computers are using hundreds of computing nodes to solve computationally expensive problems in scientific computing.

Parallel Platforms : Graphical Processing Units (GPUs)

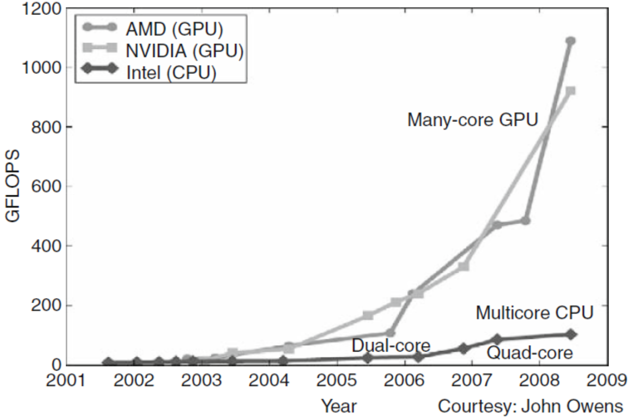


Architectural overview of Nvidia GeForce 8800.

So, how can NVIDIA offer hundreds of thread processors while the rest of the industry can deliver only dual- and quad-core processors?

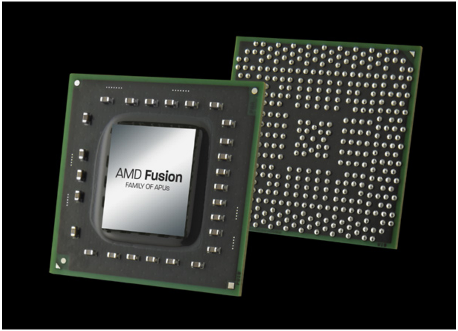
`NVIDIA designers use a common architectural building block, called a multiprocessor, that can be replicated as many times as required to provide a large number of processing cores (or thread processors) on a GPU board for a given price point.`

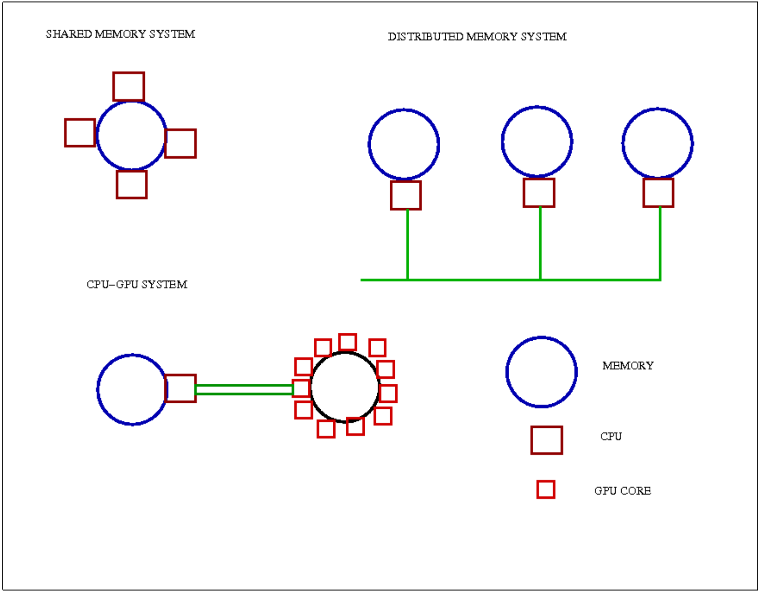
Parallel Platforms :Graphical Processing Units (GPUs



Parallel Platforms : Accelerated Processing Units (APUs)

Using its Fusion technology, AMD incorporates multi-core CPU (x86) technology with a powerful DirectX capable discrete-level graphics and parallel processing engine onto a single die to create the first Accelerated Processing Unit (APU).




SHARED MEMORY SYSTEM
DISTRIBUTED MEMORY SYSTEM
CPU–GPU SYSTEM
MEMORY
CPU
GPU CORE

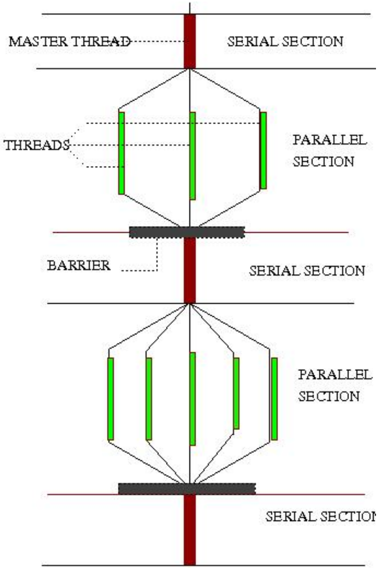
Shared Memory Programming

- Shared memory programming can be done on a system which has more than one computing units (core) sharing the same physical memory.
- The data between different computing units is shared in the form of shared variables.
- There are many tools (Application Programming Interfaces or API) like OpenMp, pthreads and Intel Threading Blocks (ITBB) available for shared memory programming.
- Note that shared address space model is different from shared memory model.

Processes and Threads

- The building blocks of a Linux system are *processes*.
- Each process has it own data, instructions and memory space.
- Threads are sub-units of processes and easy to create (because no data protection is needed) and share memory space.
- The ability of threads to run simultaneously can be used to do many independent tasks concurrently.
- Threading API provide tools to assign id to threads, share data and instruction and for synchronization.
- In general, multi-threading problems follow the fork-join model.

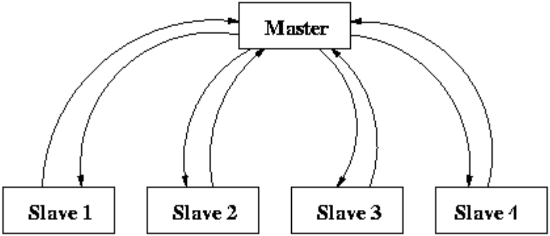
Fork-Join


MASTER THREAD
SERIAL SECTION
THREADS
PARALLEL SECTION
BARRIER
SERIAL SECTION
PARALLEL SECTION
SERIAL SECTIO

Distributed Memory Programming

- Data and instructions between a set of homogeneous computing units are shared by explicit communications using tools (API) like MPI.
- Each computing unit is assigned a unique identification number (id) which is used to establish communication and share the data and instructions.
- Communication between computing units may be one to one (send, receive type) or it can be collective (broadcast, scatter, gather etc.).
- There is no upper limit on the number of the computing units the system can have, however, communication complexity and overhead makes it difficult to make a very large system.
- In general, the computation follows the master-slave paradigm.

Master-Slave Model



GPU Programming

- On a General Purpose Graphical Processing Unit (GP-GPU) a large number of processing units (cores) are available which can work simultaneously.
- The sections of a program which take a lot of time, and can be easily split into tasks which can be run in parallel, can be transferred to the GPU.
- GPUs are very good for SIMD system.
- The GPU and CPU do not share the memory space so the data has to be explicitly copied from the CPU to the GPU and back.
- For Nvidia GPUs a C like programming environment (CUDA) is available.
- OpenCL can be used to program any GPGPU.

Summary and conclusions

- It is not easy to make a super-fast single processor so multi-processor computing is the only way to get more computing power.
- When more than one processors (cores) share the same memory, shared memory programming is used e.g., pthreads,OpenMp, ITBB etc.
- Shared memory programming is fast and it easy to get linear scaling since communication is not an issue.
- When processors do not share memory, explicit communication is used as in MPI and PVM.
- Distributed memory programming is the main way to solve large problems (when thousands of processors are needed).
- General Purpose Graphical Processing Units (GP-GPU) can provide very high performance at very low cost, however, programming is somewhat complicated and parallelism is limited to only SIMD.

Thank You !

http://www.iucaa.ernet.in/ jayanti/