INTRODUCTION TO PARALLEL COMPUTING

Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar

To accompany the text ``Introduction to Parallel Computing'',
Addison Wesley, 2003.

TOPIC OVERVIEW

Motivating Parallelism

Scope of Parallel Computing Applications

Organization and Contents of the Course

MOTIVATING PARALLELISM

The role of parallelism in accelerating computing speeds has been recognized for several decades.

Its role in providing multiplicity of datapaths and increased access to storage elements has been significant in commercial applications.

The scalable performance and lower cost of parallel platforms is reflected in the wide variety of applications.

THE COMPUTATIONAL POWER ARGUMENT

Moore's law states [1965]:

"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000."

THE COMPUTATIONAL POWER ARGUMENT

Moore attributed this doubling rate to exponential behavior of die sizes, finer minimum dimensions, and ``circuit and device cleverness''.

In 1975, he revised this law as follows:

"There is no room left to squeeze anything out by being clever. Going forward from here we have to depend on the two size factors - bigger dies and finer dimensions."

He revised his rate of *circuit complexity* doubling to 18 months and projected from 1975 onwards at this reduced rate.

THE COMPUTATIONAL POWER ARGUMENT

If one is to buy into Moore's law, the question still remains - how does one translate transistors into useful OPS (operations per second)?

The logical recourse is to rely on parallelism, both implicit and explicit.

Most serial (or seemingly serial) processors rely extensively on implicit parallelism.

We focus in this class, for the most part, on explicit parallelism.

THE MEMORY/DISK SPEED ARGUMENT

While clock rates of high-end processors have increased at roughly 40% per year over the past decade, DRAM access times have only improved at the rate of roughly 10% per year over this interval.

This mismatch in speeds causes significant performance bottlenecks.

Parallel platforms provide increased bandwidth to the memory system.

Parallel platforms also provide higher aggregate caches.

Principles of locality of data reference and bulk access, which guide parallel algorithm design also apply to memory optimization.

Some of the fastest growing applications of parallel computing utilize not their raw computational speed, rather their ability to pump data to memory and disk faster.

THE DATA COMMUNICATION ARGUMENT

As the network evolves, the vision of the Internet as one large computing platform has emerged.

This view is exploited by applications such as SETI@home and Folding@home.

In many other applications (typically databases and data mining) the volume of data is such that they cannot be moved.

Any analyses on this data must be performed over the network using parallel techniques.

PARALLELISM VS. CONCURRENCY

Parallelism and concurrency are two closely related terms.

Concurrency deals with the specification of tasks that may potentially execute in parallel. It is a programming language concept that deals primarily with semantics and correctness. Serial execution of concurrent programs helps tolerate high latency of tasks (e.g. network transfers), provide interactive response (e.g. user interfaces), and fault tolerance.

Parallel execution of concurrent tasks involves the hardware platform, the runtime environment, and associated execution overheads. For this reason, parallelism primarily deals with considerations of performance.

PURDUE NExT

OVERVIEW OF PARALLEL COMPUTING PLATFORMS

Dichotomy of Parallel Computing Platforms

Communication Model of Parallel Platforms

Physical Organization of Parallel Platforms

Communication Costs in Parallel Machines

Messaging Cost Models and Routing Mechanisms

Mapping Techniques

Heterogeneous Parallel Platforms

DICHOTOMY OF PARALLEL COMPUTING PLATFORMS

An explicitly parallel program must specify concurrency and interaction between concurrent subtasks.

The former is sometimes also referred to as the control structure and the latter as the communication model.

CONTROL STRUCTURE OF PARALLEL PROGRAMS

Parallelism can be expressed at various levels of granularity - from instruction level to processes.

Between these extremes exist a range of models, along with corresponding architectural support.

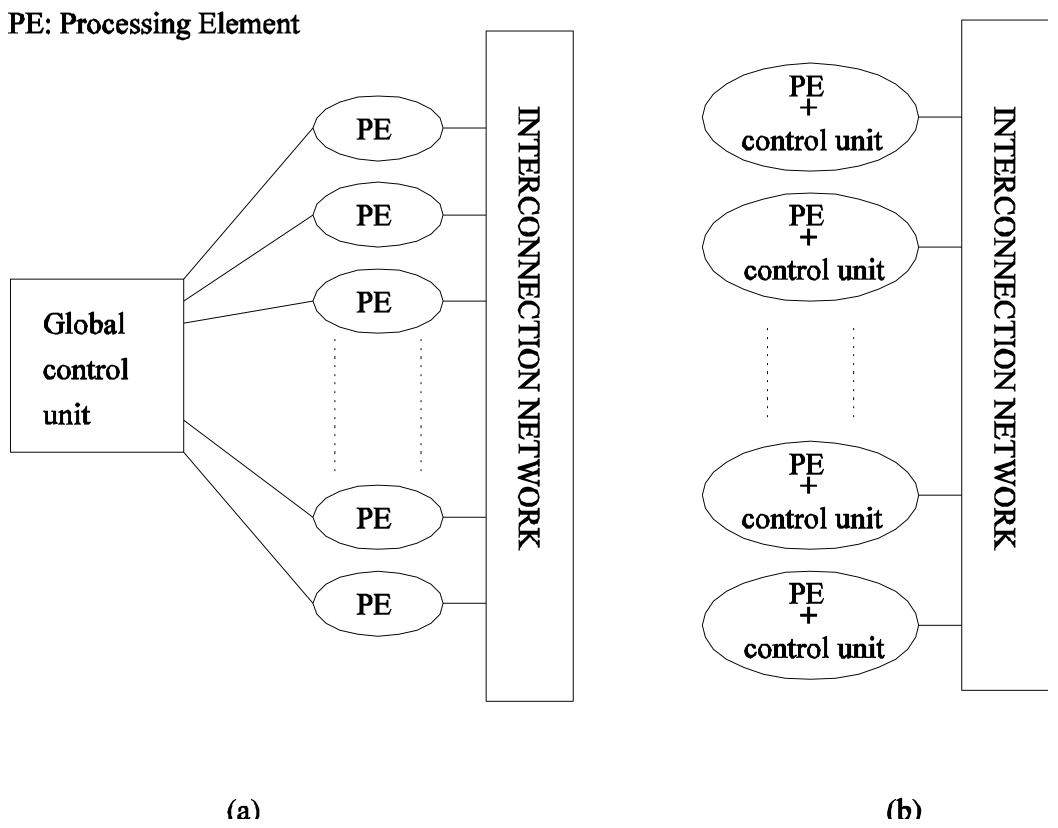
CONTROL STRUCTURE OF PARALLEL PROGRAMS

Processing units in parallel computers either operate under the centralized control of a single control unit or work independently.

If there is a single control unit that dispatches the same instruction to various processors (that work on different data), the model is referred to as single instruction stream, multiple data stream (SIMD).

If each processor has its own control unit, each processor can execute different instructions on different data items. This model is called multiple instruction stream, multiple data stream (MIMD).

SIMD AND MIMD PROCESSORS



A typical SIMD architecture (a) and a typical MIMD architecture (b).

SIMD PROCESSORS

Some of the earliest parallel computers such as the Illiac IV, MPP, DAP, CM-2, and MasPar MP-1 belonged to this class of machines.

Variants of this concept have found use in co-processing units such as the MMX units in Intel processors and DSP chips such as the Sharc.

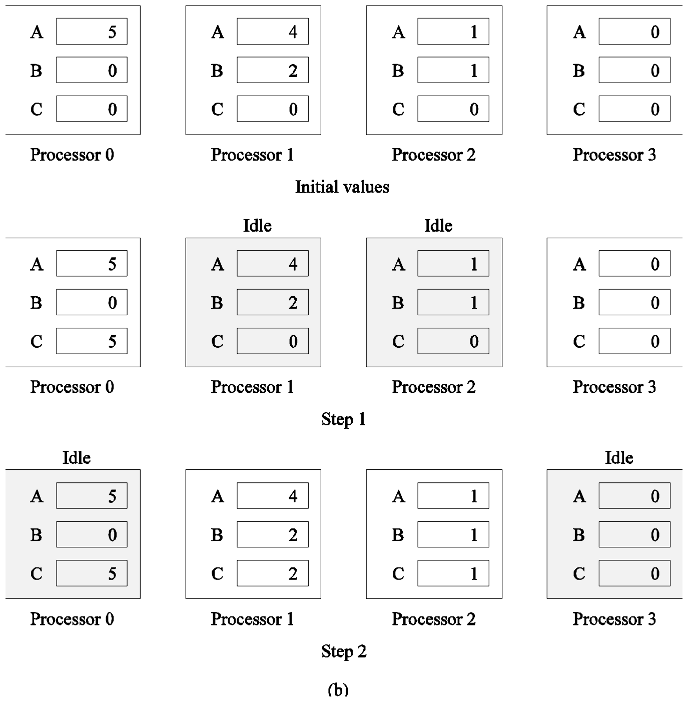
SIMD relies on the regular structure of computations (such as those in image processing).

It is often necessary to selectively turn off operations on certain data items. For this reason, most SIMD programming paradigms allow for an ``activity mask'', which determines if a processor should participate in a computation or not.

CONDITIONAL EXECUTION IN SIMD PROCESSORS

```
if (B == 0)
        C = A;
else
        C = A/B;
```

(a)



Executing a conditional statement on an SIMD computer with four processors: (a) the conditional statement; (b) the execution of the statement in two steps.

MIMD PROCESSORS

In contrast to SIMD processors, MIMD processors can execute different programs on different processors.

A variant of this, called single program multiple data streams (SPMD) executes the same program on different processors.

It is easy to see that SPMD and MIMD are closely related in terms of programming flexibility and underlying architectural support.

Examples of such platforms include current generation Sun Ultra Servers, SGI Origin Servers, multiprocessor PCs, workstation clusters, and the IBM SP.

COMMUNICATION MODEL OF PARALLEL PLATFORMS

There are two primary forms of data exchange between parallel tasks - accessing a shared data space and exchanging messages.

Platforms that provide a shared data space are called shared-address-space machines or multiprocessors.

Platforms that support messaging are also called message passing platforms or multi-computers.

PURDUE NExT

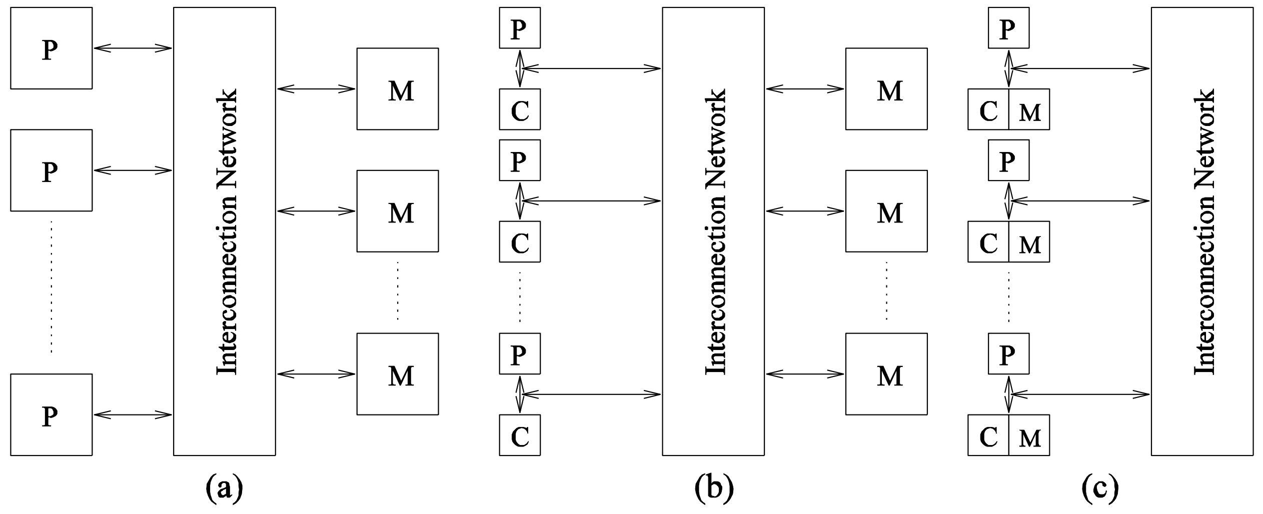
SHARED-ADDRESS-SPACE PLATFORMS

Part (or all) of the memory is accessible to all processors.

Processors interact by modifying data objects stored in this shared-address-space.

If the time taken by a processor to access any memory word in the system global or local is identical, the platform is classified as a uniform memory access (UMA), else, a non-uniform memory access (NUMA) machine.

NUMA AND UMA SHARED-ADDRESS-SPACE PLATFORMS



Typical shared-address-space architectures: (a) Uniform-memory access shared-address-space computer; (b) Uniform-memory-access shared-address-space computer with caches and memories; (c) Non-uniform-memory-access shared-address-space computer with local memory only.

NUMA AND UMA SHARED-ADDRESS-SPACE PLATFORMS

The distinction between NUMA and UMA platforms is important from the point of view of algorithm design. NUMA machines require locality from underlying algorithms for performance.

Programming these platforms is easier since reads and writes are implicitly visible to other processors.

However, read-write data to shared data must be coordinated (this will be discussed in greater detail when we talk about threads programming).

Caches in such machines require coordinated access to multiple copies. This leads to the cache coherence problem.

A weaker model of these machines provides an address map, but not coordinated access. These models are called non cache coherent shared address space machines.

SHARED-ADDRESS-SPACE VS. SHARED MEMORY MACHINES

It is important to note the difference between the terms shared address space and shared memory.

We refer to the former as a programming abstraction and to the latter as a physical machine attribute.

It is possible to provide a shared address space using a physically distributed memory.

MESSAGE-PASSING PLATFORMS

These platforms comprise of a set of processors and their own (exclusive) memory.

Instances of such a view come naturally from clustered workstations and non-shared-address-space multi-computers.

These platforms are programmed using (variants of) send and receive primitives.

Libraries such as MPI and PVM provide such primitives.

PURDUE
NExT

MESSAGE PASSING
VS.
SHARED ADDRESS SPACE PLATFORMS

Message passing requires little hardware support, other than a network.

Shared address space platforms can easily emulate message passing. The reverse is more difficult to do (in an efficient manner).