CS 689–001
Parallel Numerical Computation

Lecture 1*

Parallel Computing Overview

Professor Jun Zhang

Department of Computer Science

University of Kentucky

Lexington, KY 40506–0046

January 9, 1999

*Some of the materials were modified from the class notes of *Parallel Numerical Algorithms* written by Professor Michael T. Heath at UIUC.

Computer Generations in 50 Years

Generation Period	Technology and Architecture	Software and Operating System	Representative Systems
First 1946 - 1956	Vacuum tubes and relay memory, single-bit CPU With accumulator-based instruction set	Machine/assembly languages, programs without subroutines	ENIAN, IBM 701, Princeton IAS
Second 1956 1967	Discrete transistors, core memory, floating-point accelerator, I/O channels	Algol and Fortran with compilers, batch processing OS	IBM 7030, CDC 1604, Univac LARC
Third 1967 - 1978	Integrated circuits, pipelined CPU, microprogrammed control unit	C language multiprogramming timesharing OS	PDP-11, IBM 360/370, CDC 6600
Fourth 1978 - 1989	VLSI microprocessors solid-state memory multiprocessors, vector supercomputers	Symmetric multiprocessing, parallelizing compilers message-passing libraries	IBM PC, VAX 9000, Cary X/MP
Fifth 1990 - present	ULSI circuits, scalable parallel computers, workstation clusters, Intranet, Internet	Java, microkernels, multithreading, distributed OS, World Wide Web	IBM SP2, SGI Origin 2000, Digital TruCluster

Need for Parallel Computation

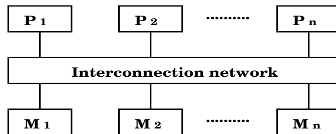
- Fundamental limits on single processor speed
  - Speed of light
- Cost effective throughput (the number of jobs processed in a unit time)
  - Larger scale modeling
  - Time sensitive computations
    - Weather modeling & forecasting
    - Defense reaction
    - Large database access
- Leveraging of existing resources
  - Many workstations are idle

Not to Be Excited

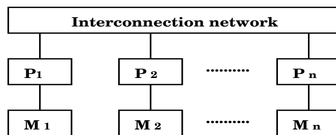
- Little software available
  - No universally acceptable standards
  - Lack of portable parallel libraries
- Immature computing environment
  - Hard to program parallel computers
  - To be improved by MPI, PVM, etc.
- Unstable commercial market
  - Many early supercomputer makers have gone out of business (Cray, Thinking Machine)
  - Some are still alive (SGI, IBM, Meiko)

Model Parallel Architectures

Shared Memory Multiprocessor



Distributed Memory Multiprocessor



Characteristics of Parallel Architectures

- Control mechanism: SIMD vs MIMD
- Operation: synchronous vs asynchronous
- Memory organization: private vs shared
- Address space: local vs global
- Memory access: uniform vs nonuniform
- Granularity: power of individual processors (coarse and fine grain systems)
- Interconnection network topology:
  - dynamic interconnection
  - static interconnection

Architectural Design Tradeoffs

	Memory Organization	
	shared	distributed
Programmability	easy	harder
Scalability	harder	easier

- Shared memory machines (1997)
  - HP Exemplar – 64 nodes
  - SGI Origin 2000 – 128 Nodes
- Distributed memory machines (1996)
  - IBM SP2 – 521 nodes
  - Cray T3E (9000 model) – 2048 nodes

Parallel Machine Categories

- Vector or array processor
- SMP: symmetric multiprocessor
- MPP: massively parallel processor
- DSM: distributed shared memory
- NOW: network of workstations
- (COW: cluster of workstations)
- Hybrids and combinations:
  - SMP or MPP with vector processors
  - networked cluster of SMPs, etc.

Dead and Live Parallel Computers

- Vector or array processor (dead)
  - Vector supercomputers: Cary, CDC, ETA
  - Minisupers: Convex, Alliant
  - Array processors: FPS
- SMP:
  - Early model: Sequent, Encore
  - Current models: DEC, HP, IBM, SGI, Sun, PCs

Dead and Live Parallel Computers

- MPP:
  - Early models: Ncube, Intel IPSC
  - SIMD: TMC CM-1, CM-5, MasPar
  - Late models: Intel Paragon, IBM SP, Cray T3D/E
- MPP vector:
  - Intel IPSC/2, FPS T-Series, TMC CM-5
- DSM:
  - Convex Exemplar, SGI Origin

Memory Hierarchy

High performance computer architectures are characterized by memory hierarchy:

- registers
- on-chip cache(s)
- off-chip cache(s)
- random access memory (RAM)
- remote memory (off-processor)
- virtual memory (paging)
- secondary storage (disks)
- tertiary storage (tapes)

Data locality and reuse are critical for good performance (cost performance tradeoff)

Parallel Programming Models

- Data parallel (array operations)
  - F90, HPF
- Message passing (send, recv, etc.)
  - MPI, PVM
- Pool of tasks (threads)
  - pthreads, Open MP
- Loop based (directives or automatic parallelization compiler)
- Data flow (functional languages)
- Linear algebra: BLAS, PBLAS, BLACS
- Process mode: MPMD, SPMD

Parallel Algorithm Design

- Decompose problem into fine grained tasks to maximize potential parallelism.
- Determine communication pattern among tasks
- Combine several fine grained tasks into coarse grained tasks, if necessary, to reduce communication requirements or other overhead costs.
- Assign tasks to processors, subject to tradeoff between communication and concurrency.

Parallel Algorithmic Paradigms

- Domain decomposition (based on data)
- Functional decomposition (based on computation)
- Embarrassingly parallel (independent tasks)
- Data parallel (array operations)
- Divide and conquer (tree like decomposition)
- Pipelining (overlapping stages)

Communication Patterns

- Latency and bandwidth
- Routing (store-and-forward, cut-through)
- Send and receive (one-to-one)
- Broadcast (one-to-all, assigning parameters)
- Reduction (all-to-one, computing inner product)
- All-to-all
- Contention, aggregate bandwidth
- Livelock and deadlock

Assigning Work/Data to Processors

- Partitioning
- Granularity
- Mapping
- Scheduling
- Load balancing

In certain applications, load balancing is hard to achieve with static scheduling. Dynamic scheduling may be employed with more overhead. Almost any parallel implementation incurs overhead, the mapping scheme needs to take relative gains and losses into consideration when attempting to use more processors

Factors Affecting Performance

- Load balance: work divided evenly
- Concurrency: work done simultaneously
- Overhead: work not present in serial computation
  - communication
  - synchronization
  - redundant computation
  - speculative computation

Measures of Performance

$T_1 =$ serial execution time on 1 processor

$T_p =$ parallel execution time on p processor

Speedup: $S_p = T_1/T_p$

Efficiency: $E_p = T_1/(pT_p)$

Thus, $E_p = S_p/p$ and $S_p = pE_p$

Pseudotheorem: $S_p \leq p$ and $E_p \leq 1$

But "speedup anomalies", such as "superlinear speedup", can occur in practice, for example, due to increased resources as p increase. In certain cases, the use of cache may be the reason. All these effects have nothing to do with parallelism

Amdahl's Law

Assumption: serial fraction $= s, 0 \leq s \leq 1$,

p-fold parallel fraction $= 1 - s$

Then

$$\begin{aligned} T_p &= sT_1 + (1-s)T_1/p \\ S_p &= p/(sp + (1-s)) \\ E_p &= 1/(sp + (1-s)) \end{aligned}$$

Hence: $S_p \to 1/s$ and $E_p \to 0$ as $p \to \infty$

For example, if $s = 0.1$, then maximum possible speedup is 10, regardless of number of processors used

This result induced early pessimism (1967) on potential of parallel computing

Problem Scaling

Amdahl's law is relevant only is serial fraction is independent of problem size (rarely true)

Larger computers solve larger problems, serial fraction usually decreases with problem size

Rate of problem growth can be characterized by keeping some quantity constant as number of processors varies. Some plausible invariants include

overall problem size	work per processor
overall execution time	memory / processor
efficiency	computational error

Scalability

Scalability refers to the effectiveness with which a parallel algorithm can utilize additional processors

An algorithm is scalable as number of processors grows if its efficiency can be maintained at a constant value (or at least bounded away from zero) by increasing problem size

Note that an algorithm that is scalable in this sense may still not be practical if growth rate required for problem size is so great that total run time becomes unacceptable

An algorithm is scalable with respect to a parallel architecture, *scalable parallel system*

Sparse Matrix Example

Computational load for series of grid problems

millions of flops

number	grid	computational load		
procs.	size	averag.	maxim.	ratio
1	132	37	37	1.00
2	164	37	37	1.01
4	205	37	37	1.01
8	256	37	43	1.15
16	320	37	46	1.24
32	400	37	47	1.27
64	500	37	49	1.32
128	624	37	47	1.26