2.2 Using Workstation Clusters

SOFTWARE TOOLS

$ **several workstation packages for workstation cluster parallel programming: (multiprocessor versions are also available)**

- **PVM (Parallel Virtual Machine) - Oak Ridge National Lab**
  - **homogeneous & heterogeneous workstations**
  - **C and FORTRAN support**
  - **public-domain (`http://www.netlib.org/pvm3/`)**
- **MPI (Message Passing Interface) - MPI Forum**
  - **provides a message-passing standard**
  - **Several public-domain implementations (`http://www.osc.edu/mpi/`)**
    - **LAM (Ohio Supercomputing Center)**
    - **MPICH (Argonne National Lab and Mississippi State University)**
    - **CHIMP (Edinburgh Parallel Computing Center)**
    - **UNIFY (Mississippi State University)**
- **Vendor-Specific proprietary message-passing packages:**
  - **MPL (IBM) for SP-2, etc.**

PVM (Parallel Virtual Machine)

Programmer is responsible for

- decomposing the problem into separate (C or FORTRAN) programs.
  - compile each program for the specific ws type.
    - homogeneous or heterogeneous workstations
    - common file system support or not
- defining the set of computers used on a problem
  - use a **hostfile** containing the names of computers
  - start one machine and add others from the PVM console

What if (# programs) > (# processes) & why do we prefer this?

Message routing in PVM:

- routing is done by using daemon processes installed by PVM on the computers that form the virtual machine. Each PVM daemon keeps sufficient information that helps to select the routing path.

Process Creation and Execution

- Processes can be started dynamically.
- PVM processes usually organized in a <u>master-slave</u> arrangement whereby a single master spawns other processes.

  pvm_spawn( )
  ↑ Name of the process, which computer to use, etc.
  ← returns an array of task IDs of the processes started.
- A process must enroll in PVM using pvm_mytid( ) and can leave the system with pvm_exit( ).

Basic Message-Passing Routines

- Programs communicate by message-passing using:
  - pvm_send( ) and pvm_recv( ).
- All PVM send routines are non-blocking (or asynchronous) pvm_send()
- PVM receive routines can be either blocking (synchronous) or nonblocking: pvm_recv( ) and pvm_nrecv( )
  - **(Sending/receiving data are done through message buffers)**
- A message tag is attached to each message, to differentiate between types of messages being sent.
  - **Both message tag and source wildcards (i.e.,-1) are available.**
- If an array of items is going to be sent:
  - pvm_psend( ) and pvm_precv( ) **are used.**
- If various types of data are to be sent at once, data has to be packed into a PVM buffer before being sent and unpacked when received:
  - pvm_pkint( ), pvm_pkstr( ), pvm_pkfloat
  - pvm_upkint( ), pvm_upkstr( ), pvm_upkfloat.

Broadcast, Multicast, Scatter, Gather, Reduce Routines

- pvm_bcast( )
- pvm_scatter( )
- pvm_gather( )
- pvm_reduce( )

} used with a group of processes after the group is formed.

pvm_joingroup( ) - process joins a named group.

pvm_mcast( ) - is not a group operation, it sends the contents of send buffer to each process that is listed in a task ID array.

<u>EXAMPLE:</u> A PVM program that adds a group of 1000 numbers together.

2.3 Evaluating Parallel Programs

<u>PARALLEL EXECUTION TIME</u>

For a parallel algorithm: $t_P = t_{comp} + t_{comm}$

Calculation of t_{comp}:

computation steps of the most complex process.

`Assumption: all processors are the same & operating at the same speed.` **What if not?** Load balancing required.

Calculation of t_{comm}:

depends upon size of message, underlying icnw, and transfer mode.

In a ws cluster, communication time depends upon many factors, including network structure and network contention.

$t_{comm} = t_{startup} + n\, t_{data}$ (bits/sec)

↑ startup time (message latency) ↑ xmission time to send one word

Message Latency is the time to send a message with no data (including pack & unpack times.)

We have ignored the following factors:

- contention of the communication network
- not having source and destination directly linked

And assumed that:

- the overhead incurred by including information other than data in the packet is a constant and can be part of $t_{startup}$.

Important Note on Interpretation of Equations

We will make many simplifying assumptions in subsequent chapters:

- t_{comp} & t_{comm} is measured in units of an arithmetic operation (system dependent)
- system is homogeneous (identical processors with equal speed)
- all arithmetic operations require the same time
- we count the number of computation steps
  - on one processor when all processes perform the same operation
  - in longest process in other situations.

  $t_{comp} = m$
- t_{comm} for sending an integer or a real takes the same time. Sending q messages of *n* data items takes

  $t_{comm} = q\ (t_{startup} + n\ t_{data})$

  $t_p = t_{comp} + t_{comm}$

In Cray T3D, PVM's

- Startup time: 3 µsec
- Xmission time: 63 ns/double
- Arithmetic op. time: 11 ns

In IBM SP-2, MPI's

- Startup time: 35 µsec
- Xmission time: 230 ns/double
- Arithmetic op. time: 4.2 ns

$t_{startup}$:

- is one/two orders of magnitude greater than t_{data}.
- will dominate the communication time in many cases, unless *n* is quite large.

Example

- Suppose that a computer can operate at 200 MFLOPs (200 million floating-point operations per second) and the startup time is 1 µs.
- Then the computer could execute 200 f-p operations in the time taken in the message startup.

Latency Hiding

The deleterious effect on the execution time as shown in previous example is known as the Achilles' heel of message-passing computers.

- **Latency Hiding:** One way to ameliorate the situation is to overlap the communication with subsequent computations (keeping processor busy with useful work while waiting for the communication to be completed)
  - nonblocking send routines & (locally) blocking send routines.
  - Using parallel slackness
    - mapping multiple processes on a processor (virtual processors) and using time-sharing facility.
    - an m-process(or) algorithm implemented on an n-processor machine is said to have a parallel slackness of m/n for that machine, where n<m.
  - Using threads

Time Complexity of a Parallel Algorithm

If we use time complexity analysis, which hides lower terms, t_{comm} will have a time complexity of O(*n*).

Complexity of t_p will be the sum of the computation and communication.

Example Problem:

Suppose that we were to add n numbers on two computers, where each computer adds n/2 numbers together.

- The numbers are initially held by the first computer.
- The second computer submits its result to the first computer for adding the two partial sums together.

Computation/Communication Ratio

Communication is very costly.

If both t_{comp} and t_{comm} has the same complexity ⇒ ?

Ideally t_{comp} >> t_{comm} and ↑n will improve performance.

Example: N-body problem

t_{comm} is O(N) and t_{comp} is $O(N^2)$

we can find an N where t_{comm} will dominate the t_{comp}

Cost-Optimal Algorithms

The cost to solve a problem is proportional to the execution time on a single processor system (using the fastest known sequential algorithm.)

Cost = $t_p \times n = k \times t_s$ where k is a constant.

A parallel algorithm is cost-optimal if

(Parallel time complexity) x (# processors) = sequential time complexity

Example:

Suppose the best known sequential algorithm for a problem has time complexity of $O(n \log n)$. Are the following cost optimal?

- A parallel algorithm for the same problem that uses *n* processes and has a time complexity of O(log *n*).
- A parallel algorithm that uses n^2 processors and has time complexity of O(1).

Comments on Asymptotic Analysis

Time complexity is

- often used for
  - sequential program analysis
  - theoretical analysis of parallel programs
- much less useful for evaluating the potential performance of parallel programs:
  - Big-Oh and other complexity notations use <u>asymptotic methods</u> (the variable under consideration to tend to infinity) however
    - often the # processors are constrained and
    - data sizes are finite and manageable.
  - Analysis often ignores lower terms that could be important (e.g., $\mathbf{t}_{startup}$ dominates overall communication time when n is large)
  - Analysis also ignores other factors that appear in real computers, such as communication contention.