Work loads

- Let's assume that we will be using only MIMD systems (SMP, cluster, NOW, Vector)
- Let's also assume that we have multiple processors, memory, IO subsystems available and that there is some kind of OS available

High throughput

- Typically many serial programs run at the same time
  - Example: rerun the test many times with different input parameters
  - Important advantage: jobs are serial -> NO software rewriting

Classes of use

- Batch
- Interactive
- Multi-job parallel

Batch systems allow you to schedule multiple jobs on multiple systems at the same time and preserve some control on what runs where

Implies: Static load balancing

Target systems may or may not run more than one job at any time

Threading

Important issues:
- Can nested jobs be run? (multi-job parallel is...)
- Does the system take into account enough parameters to make an accurate schedule?
  - Such as memory, IO, cache sizes
- Does the system die from overloaded IO
High Throughput Jobs

Nodes

Throughput = (3 + 2) / 2 = 2.5 units/time

Wasted capacity

Problems if the Jobs are of unequal times And nodes are not all the same
High Throughput

Problems if the Jobs are of unequal times
And nodes are not all the same

High throughput

Interactive systems
Logging into a database front end and submitting queries directly
Logging into a system and starting a job
But the system chooses a spare node for you to log into or to run the command/application on
Issues involve the collection of IO
Used to reduce the response time you the user sees

High Performance = Parallel Computing

As a single processor can only do so much work,
To do more we need more processors

Parallel Computing is a means to use more processors
Parallel Computing is using more than one task to simultaneously compute some result
A task is not always a Unix Process, thread etc
The task does not have to exist for the whole calculation
The task might not have an affinity to a particular CPU etc
It might have to share, migrate...

What is our goal

Read data
Compute
Check result
Post process
Write results

Pre-process
Off line?
What is our goal

Read data
Compute
Check result
Post process
Write results

Pre-process
Off line?

Loop

Tasks/procs

Parallel overheads

Leaving serial code would make Amdahl happy

Post processing and IO can also be speeded up (may reduce other overheads)
Parallel Algorithm is a list of instructions on how to execute a problem across multiple processors/tasks. Depending on the target system it may consist of:
- Identifying parts of the problem that are possible to run concurrently
- Mapping the work onto a number of tasks/processors
- Distributing the input, intermediate and output data
- Managing any sharing of data
- Coordinating (synchronizing) tasks/processors

Decomposition
- Dividing up the work into smaller parts
- Parallel decomposition is where these smaller parts can be executed concurrently (i.e., in parallel)
- Some parts of work might depend on the results of others
- Task graph can then be used to solve the dependencies

Calculation of each $y[i]$ value could be a task

Degree of Concurrency = $N$
- Inverse of this indicates the granularity: fine
Principles of Parallel Algorithms

Where it is trivial to make parallel due to
No dependencies -> embarrassingly parallel

Unless we count the cost collecting \( y \) into one location?

Degree of Concurrency = \( N \)

Calculation of each \( y(N/2) \) values is a task

Task 1

Task 2

Degree of Concurrency = 2

Inverse of this indicates the
Granularity : course

Two different task solving
the same problem
(binary verses pseudo binomial)

Both task mapping have the same
Highest degree of concurrency

I C L
Mapping of tasks to processors

No transfer of data

Transfer or sharing of data

2/7/2006 9:26 PM G.Fagg Cluster Computing 43

Task 1: C1,1 = A1,1 \cdot B1,1
Task 2: C1,1 = C1,1 + A1,2 \cdot B2,1
Task 3: C1,2 = A1,1 \cdot B1,2
Task 4: C1,2 = C1,2 + A1,2 \cdot B2,2
Task 5: C2,1 = A2,1 \cdot B1,1
Task 6: C2,1 = C2,1 + A2,2 \cdot B2,1
Task 7: C2,2 = A2,1 \cdot B1,2
Task 8: C2,2 = C2,2 + A2,2 \cdot B2,2
Task 1: \( C_{1,1} = A_{1,1} \cdot B_{1,1} \)

Task 2: \( C_{1,1} = C_{1,1} + A_{1,2} \cdot B_{2,1} \)

Multiply add could be fast if \( C_{1,1} \) owned by Task 2.

No explicit storage for intermediate values of \( C_{1,1} \).
Concurrency increase from 4 to 6 but also increase memory requirements.

Communication
Takes time & Effects synchronization
To reduce these effects limit the number of communication partners.

Understanding Parallel Performance
If a serial computation takes $T_s$ time
Then running on 2 processors should take $T_s/2$ time?
On $N$ processors $T_s/N$ time?
This is linear speedup

Understanding Parallel Performance
Proc 1: computing, recv, computing
Proc 2: computing, idle, computing, idle

Tp2
Understanding Parallel Performance

To (overhead time) = N*To - Ts
S (speed-up) = Ts / Tp -> N if perfect
E (efficiency) = S / N = Ts / N*To
If To > 0 then the efficiency less 1 (100%)

Overhead from being parallel

E = 1 / (1 + (To/Ts))
But To is proportional to N
I.e. if there is any overhead, then as N increases efficiency must go down!
As N increases we might have to distribute data across more tasks... Or share data more...

Fixed problem size

Fixed # procs

100%
Understanding Parallel Performance

Slower

Faster

Slower

Faster
Understanding Parallel Performance

Slower

Faster

Might not be as bad as it looks
If each task performs IO
Then OS overlaps IO with computation

» Scalable system (hardware and algorithm) can be described as one where
the efficiency remains constant as the number of nodes are increased if the
problem size is also increased accordingly

Parallel Communication

The distribution of tasks makes the use of initial values a potential bottleneck.
If a non-shared address space is used, this is usually performed by
explicit message passing.
If multiple parties need these values then depending on the number of
tasks and data size this can be a considerable overhead.

Parallel Communication

Calculation of each
y[i] value could be
a task

But only if each task has a complete copy of 'b'
Parallel Communication

Flat tree (or sequential broadcast)

Completes in N-1 time steps...

Binary tree (fan out of 2)
Parallel Communication

Time to complete 3.. (log based not linear $O(n)$)

If $N$ was 10000...
(and remember the numbers from the top500)

Number of simultaneous communications
Approximately a power of 2
**Parallel Communication**

- Can be made very very efficient.
- Correct algorithms should be used to distribute and collect (partial) results.
- Same patterns can be used to implement Parallel IO.
- If combined with replicated computation can lead to reduction of serial components and parallel overheads.

**Parallel IO**

- IO can be a big problem /bottleneck

**High Performance Workloads**

- High Performance = Parallel
- Almost every part of your application can be made parallel
- BUT
- If the method used shares data, this is either via IO to disk or communication to another node’s memory then you need enough networking
- The problem drives the needs of the target system

**Perfect Parallel**

- Tasks/procs
- Initial distribution (PIO / Parallel communication)
- Parallel execution
- Parallel convergence/termination detection
- Parallel post processing
- Parallel result collection (PIO / Parallel comms)

---

**Title goes here**