Multi-Machine Parallelism

Peter Chapin

CIS-4230, Parallel Programming

Vermont State University

Shared Memory

- PROs
  - High-speed access to shared data.
  - Fast synchronization.
  - Widely used and understood programming model (threads).
  - Support in many languages.
    - C/C++ 2011
    - Java
  - Support from many standards.
    - POSIX threads
    - OpenMP
- CONs
  - Doesn’t scale.

Message Passing

- Processing elements share data by passing messages to each other.
- More general!
  - It’s easy to do message passing between processes/threads on one machine.
  - It’s hard to simulate shared memory across multiple machines.
  - *Programs written to pass messages can be used in more contexts.*
- Some languages/libraries focus on message passing.
  - Erlang
  - Scala/Akka actors
  - Ada provides both message passing (“rendezvous”) and shared memory (“protected object”) primitives.

Overhead

- Message passing entails more overhead than shared memory.
  - … especially for messages sent over the network!
- It is essential to design a program to account for this.
  - The on-node computation must swamp message passing overhead.
  - Message passing must be asymptotically faster than on-node computation.
- Example: $O(n)$ time for messages; $O(n^2)$ time to compute.
  - As *n* grows, the message passing overhead becomes insignificant.
  - Solarium: At each iteration, send the dynamics of every object to all nodes.
  - Solarium: Nodes compute new dynamics of their fractions (n/m) of objects.
  - Solarium: New dynamics gathered and re-broadcast for the next pass.

Granularity

- Very fine grain…
  - For example, different sub-expressions execute in parallel.
  - Sub-expressions should be side-effect-free (purely functional).
  - *Shared memory*

```
X := (A + B) * (C + D);
```

Granularity

- Fine grain…
  - For example, different iterations of a loop run in parallel.
  - Loop iterations must be independent.
  - *Shared memory*

```
for (int i = 0; i < COUNT; ++i) {
    array[i] = f(i);
}
```

Granularity

- Explicit threads…
  - Multiple functions run in parallel, one in each thread.
  - Access to shared data must be carefully synchronized.
  - *Shared memory*

```
void *thread_1(void *arg)
{
    // …
}

void *thread_2(void *arg)
{
    // …
}
```

Granularity

- Explicit processes…
  - Multiple processes run in parallel.
  - Sharing data requires operating system assistance.
  - *Shared memory or message passing*

```
$ process_1 &
$ process_2 &
$ process_3 | process_4
```

Granularity

- Clusters
  - Multiple machines that are adjacent geographically and administratively.
  - Dedicated network communication.
  - *Message passing*

ASC Q cluster at Los Alamos National Laboratory

http://www.ctwatch.org/quarterly/print.php%3Fp=89.html

Granularity

- Wide area distributed computing
  - Many machines are spread over a broad geographic and administrative space.
  - No communication between worker nodes.
  - *Message passing*

SETI@HOME

BOINC

Rosetta@home
Protein Folding, Design, and Docking

LHC@home
SixTrack

http://boinc.berkeley.edu/

Programming Clusters

- Low level…
  - Write separate programs for nodes.
  - Communication via explicit network programming.
- Very flexible… lots of work

Programming Clusters

- High level…
  - Write a single program in a special programming language.
  - Let the compiler distribute to cluster nodes and worry about communication.
- *Open research problem*

Programming Clusters

- Real-life approach…
  - Write a program using a special message-passing library for communication.
  - Library optimizes messages.
- Prime example:
  - MPI ("Message Passing Interface")
    - http://www.mcs.anl.gov/research/projects/mpi/
  - This is the approach we will study.

Hybrid Programming

- Write for a cluster and run on a single multi-core node.
  1. Create a single-threaded MPI-based program.
  2. Launch several copies of it on one machine to use its cores.
  3. MPI library passes messages efficiently using the OS IPC mechanism.
- Write for a cluster and make the node programs multi-threaded.
  1. Use MPI for inter-node communication.
  2. Use thread management (POSIX threads? OpenMP?) on each node.
  3. Take advantage of threading's low overhead and multi-machine scalability.
- Write for a cluster and launch multiple copies per node.