MPI: Message-Passing Interface

Ioan Raicu
Center for Ultra-scale Computing and Information Security
Department of Electrical Engineering & Computer Science
Northwestern University

EECS 395 / EECS 495
Hot Topics in Distributed Systems: Data-Intensive Computing
March 9th, 2010
Most widely used for programming parallel computers (clusters of workstations)

Key attributes:
- Partitioned address space
- Explicit parallelization

Process interactions
- Send and receive data
**Communications**
- Sending and receiving messages
- Primitives
  - send(buff, size, destination)
  - receive(buff, size, source)
  - Blocking vs non-blocking
  - Buffered vs non-buffered
- Message Passing Interface (MPI)
  - Popular message passing library
  - ~125 functions
Time required to compute the $NxN$ matrix product $C=A*B$

Assuming you can address 64GB from one task, can you wait a month?

How to balance computational goal vs. compute resources?

Choose the right scale!
Let’s jump to an example

• Sharks and Fish II: \( N^2 \) parallel force evaluation
• e.g. 4 CPUs evaluate force for 125 fish

| 31 | 31 | 31 | 32 |

• Domain decomposition: Each CPU is “in charge” of ~31 fish, but keeps a fairly recent copy of all the fishes positions (replicated data)
• Is it not possible to uniformly decompose problems in general, especially in many dimensions
• This toy problem is simple, has fine granularity and is 2D
• Let’s see how it scales
Data:
n_fish → global
my_fish → local
fish_i = {x, y, …}

Dynamics:
F = ma
...
V = Σ 1/r_{ij}

dq/dt = m * p
dp/dt = -dV/dq

MPI_Allgatherv(myfish_buf, len[rank], MPI_FishType…)

for (i = 0; i < my_fish; ++i) {
    for (j = 0; j < n_fish; ++j) {  //  i!=j
        a_i += g * mass_j * (fish_i − fish_j) / r_{ij}
    }
}

Move fish
Sharks and Fish II: How fast?

• 100 fish can move 1000 steps in
  1 task → 5.459s
  32 tasks → 2.756s

\[ \text{x 1.98 speedup} \]

• 1000 fish can move 1000 steps in
  1 task → 511.14s
  32 tasks → 20.815s

\[ \text{x 24.6 speedup} \]

• So what’s the “best” way to run?
  – How many fish do we really have?
  – How large a computer (time) do we have?
  – How quickly do we need the answer?
Running `fish_sim` for 100-1000 fish on 1-32 CPUs we see

\[
\text{time} \sim \text{fish}^2
\]

Scaling: Good 1st Step: Do runtimes make sense?
Scaling studies involve changing the degree of parallelism. Will we be changing the problem also?

– Strong scaling \(\rightarrow\) Fixed problem size

– Weak scaling \(\rightarrow\) Problem size grows with additional compute resources

How do we measure success in parallel scaling?

– Speed up = \(T_s/T_p(n)\)

– Efficiency = \(T_s/(n*T_p(n))\)

Multiple definitions exist!
Why does efficiency drop?

- Serial code sections → Amdahl’s law
- Surface to Volume → Communication bound
- Algorithm complexity or switching
- Communication protocol switching
- Scalability of computer and interconnect
In general, changing problem size and concurrency expose or remove compute resources. Bottlenecks shift.

In general, first bottleneck wins.

Scaling brings additional resources too.
  – More CPUs (of course)
  – More cache(s)
  – More memory BW in some cases
Scaling: Superlinear Speedup

speed up (tn/t1)

OMP dot product (x*x)

log size of vector x in MB
(arrows show aggregate L2 cache size)

# CPUs (OMP)
MPI_Allreduce buffer size is 32 bytes.

Q: What resource is being depleted here?
A: Small message latency

1) Compute per task is decreasing
2) Synchronization rate is increasing
3) Surface:Volume ratio is increasing
Load Balance: Application Cartoon

Unbalanced:

Task 1
Task 2
Task 3
Task 4

Balanced:

Task 1
Task 2
Task 3
Task 4

Universal App

Sync
Flop
I/O

Time saved by load balance

Will define synchronization later
Load Balance: performance data

Communication Time: 64 tasks show 200s, 960 tasks show 230s

MPI ranks sorted by total communication time
while(1) {
    do_flops(N_i);
    MPI_Alltoall();
    MPI_Allreduce();
}
Load Balancing
Load Balancing

MPI Rank ➔

Time ➔

Flops
Exchange
Sync
Load Balance: analysis

- The 64 slow tasks (with more compute work) cause 30 seconds more “communication” in 960 tasks.
- This leads to 28800 CPU*seconds (8 CPU*hours) of unproductive computing.
- All load imbalance requires is one slow task and a synchronizing collective!
- Pair well problem size and concurrency.
- Parallel computers allow you to waste time faster!
Scaling of MPI_Barrier()
It’s hard to discuss synchronization outside of the context a particular parallel computer.

 MPI timings depend on HW, SW, and environment:
  - How much of MPI is handled by the switch adapter?
  - How big are messaging buffers?
  - How many thread locks per function?
  - How noisy is the machine (today)?

This is hard to model, so take an empirical approach based on an IBM SP which is largely applicable to other clusters…
• 6080 dedicated CPUs, 96 shared login CPUs
• Hierarchy of caching, speeds not balanced
• Bottleneck determined by first depleted resource
MPI Performance is often Hierarchical

message size and task placement are key to performance
The set of all possibly latencies describes the interconnect geometry from the application perspective.
Synchronization: Summary

• As a programmer you can control
  – Which MPI calls you use (it’s not required to use them all).
  – Message sizes, Problem size (maybe)
  – The temporal granularity of synchronization, i.e., where do synchronization occur.

• Language writers and system architects control
  – How hard is it to do the above
  – The intrinsic amount of noise in the machine
### Parallel File I/O: Strategies

<table>
<thead>
<tr>
<th>1) Serial</th>
<th>2) Multiple File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task1 ← File</td>
<td>Task1 ← File1</td>
</tr>
<tr>
<td>Task2</td>
<td>Task2 ← File2</td>
</tr>
<tr>
<td>Task3</td>
<td>Task3 ← File3</td>
</tr>
<tr>
<td>Task4</td>
<td>Task4 ← File4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3) POSIX I/O</th>
<th>4) MPI I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task1 ← File</td>
<td>Task1</td>
</tr>
<tr>
<td>Task2</td>
<td>Task2 ←</td>
</tr>
<tr>
<td>Task3</td>
<td>Task3 ← File</td>
</tr>
<tr>
<td>Task4</td>
<td>Task4</td>
</tr>
</tbody>
</table>

Some strategies fall down at scale.
Parallel File I/O: Metadata

- A parallel file system is great, but it is also another place to create contention.
- Avoid unneeded disk I/O, know your file system
- Often avoid file per task I/O strategies when running at scale