Using Happens-Before Relationship to debug MPI non-determinism

Anh Vo and Alan Humphrey
{avo,ahumphre}@cs.utah.edu
Distributed event ordering is crucial

- Bob receives two undated letters from his dad
  - One says “Mom is sick”
  - One says “Mom is well”
- Is Mom sick or well?
Logical clocks

• Can’t trust physical clocks because processes clocks are not synchronized
• Need to use logical clocks
• Lamport clocks and vector clocks are two examples of logical clocks
• The Clock Condition: if a \rightarrow b then C(a) < C(b)
Lamport’s Happens-before

• If a and b are events $a^{th}$ and $b^{th}$ in process P and $a < b$, $a \rightarrow b$
• If a is the sending event of message m and b is the receiving event of message m, then $a \rightarrow b$
• If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$
Lamport’s algorithm

• Assumption:
  – Send’s and Receive’s are considered events
  – Local events are ordered
• Each process has a clock C
• Increase the clock before each event e and associate the clock to the event: C(e).
• Attach the clock to each outgoing message (piggybacking)
• Upon receiving a message, update the local clock so that it’s higher than the piggybacked clock
Example illustrating Lamport clocks

We have:
- a→c: local ordering
- a→b: send/recv
- c→d: send/recv
- a→c: transitive
- a→d: transitive
- b→d: transitive
Lamport clocks do not capture partial ordering

• Does the converse of the clock condition hold?
• If $C(a) < C(b)$ then $a \rightarrow b$?

Does $e \rightarrow c$, $e \rightarrow d$ ?
Vector clocks can capture partial ordering

- Each process $i$ keeps a vector of clocks $V_i[0..n]$
- $V_i[j]$ represents $i$’s knowledge about $j$
- Updating rules are similar:
  - Per event, process $I$ increments $V_i[i]$ instead
- Vector Clock maintains: $a \rightarrow b$ iff $V(a) < V(b)$
- $V(a) < V(b)$ iff
  \[
  \forall i \ V(a)[i] \leq V(b)[i] \ AND \ \exists j \ V(a)[i] < V(b)[j]
  \]
- Two events are concurrent if neither $a \rightarrow b$ nor $b \rightarrow a$ holds
Vector clocks example

V(e) and V(d) are incomparable, which means they’re concurrent.
Happens-Before in MPI

- MPI is both asynchronous and synchronous.
- Local events are not necessarily happens-before ordered.

Both Isend can match Irecv because there’s no HB ordering between Irecv and Barrier in P2, and no HB ordering between Isend and Barrier in P0.
Match-Before is more useful than Happen-Before

- It’s more useful to capture Match-Before relationship in MPI
- It’s possible to “happen” before, but “match” after

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Isend (2,3)</td>
<td>MPI_Isend(2,3)</td>
<td>MPI_Irecv (*,3,s1)</td>
</tr>
<tr>
<td>MPI_Isend(2,5)</td>
<td>MPI_Isend(2,5)</td>
<td>MPI_Recv(*,5,s2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPI_Send(s2.SRC)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPI_Wait()</td>
</tr>
</tbody>
</table>
MPI Match-Before

• Blocking calls (Ssend, Recv, Barrier, Wait) match-before any calls issued after it
• Nonblocking calls match-before their corresponding waits
• Sends and receives to/from the same process follow MPI non-overtaking rule
  – i.e., if two sends can match a receive, the send issued earlier would match first. Likewise for receives
Why do we care about MPI match-before

• We use MPI match-before to help detect potential matches of wildcard receives
  – Bugs due to wildcard receives are very hard to debug due to their intermittent appearances

• The idea is to find all sends that do not have match-before relationship with a recv

```
P0
---
MPI_Isend (data=22,P2)
MPI_Barrier

P1
---
MPI_Barrier

P2
---
MPI_Irecv (*)
MPI_Barrier
If (data=33)
ERROR;
```
ISP uses Match-Before to detect alternative matches

• Find all possible concurrent Sends for a wildcard Receives
• Force the match of each of those Sends by dynamically rewriting ANY_SOURCE into a specific target
• Detect deadlocks, resource leaks, local assertion violations
ISP Framework

- Hijack MPI Calls
- Scheduler decides how they are sent to the MPI runtime
- Scheduler plays out only the RELEVANT interleavings
Barrier

Isend(1, req)

Wait(req)

scheduler

Irecv(*, req)

Barrier

recv(2)

Wait(req)

sendNext

scheduler

Isend(1)

Barrier

Irecv(*)

Barrier

MPI Runtime
ISP Scheduler Actions (animation)

- **Isend(1, req)**
- **Irecv(\ast, req)**
- **Barrier**
- **Wait(req)**
- **Recv(2)**
- **Wait(req)**
- **Barrier**
- **Isend(1)**
- **Irecv(\ast)**
- **Barrier**
- **Barrier**
ISP Scheduler Actions (animation)

MPI Runtime

Scheduler

Isend(1)
Barrier
Wait (req)
Irecv(*)
Barrier
Recv(2)
SendNext
Isend(1)
Barrier
Wait (req)
Irecv(*)
Barrier
Recv(2)

No Match-Set

Deadlock!
USING ISP
Using ISP

• **ISP is a powerful tool**
  – Instruments code automatically at compile-time
  – Gives coverage / deadlock detection guarantees

• **Difficult to use for novices**
  – Hard to interpret results of a verification run
  – Needs an intuitive front-end to hide gory details

• **GEM (Graphical Explorer of MPI Programs)**
  – Eclipse plug-in addresses these difficulties
  – Helps visualize the “Happens Before” relationship
A “Small" ISP Log File
(command line ISP)

1 0 0 1 4 Isend 1 0 0 0:1:2: 2 } { [ 1 2 ] [ 1 3 ] [ 0 0 ] } Match: 1 0 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 421 0 1 4 1 Barrier 0 0:1:2: 2 } { [ 1 1 ] [ 2 0 ] [ 1 2 ] [ 0 1 ] [ 2 1 ] [ 2 2 ] } Match: -1 1 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 431 0 2 6 6 Wait { 3 } {} Match: -1 -1 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 441 0 3 10 11 Finalize { } { [ 1 4 ] [ 2 3 ] [ 0 3 ] } Match: -1 -1 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 571 1 0 2 5 Irecv -1 0 0 0:1:2: 2 3 } { [ 0 0 ] [ 0 2 ] } Match: 0 0 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 491 1 3 11 9 Wait { 4 } { [ 2 2 ] [ 2 3 ] [ 1 3 ] } Match: -1 -1 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 501 1 4 12 12 Finalize { } { [ 0 3 ] [ 2 3 ] [ 1 4 ] } Match: -1 -1 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 571 2 0 3 3 Barrier 0 0:1:2: 1 2 } { [ 0 1 ] [ 1 1 ] [ 0 2 ] [ 2 0 ] [ 1 2 ] } Match: -1 -1 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 521 2 1 8 7 Isend 1 0 0 0:1:2: 2 } { [ 1 2 ] [ 1 3 ] [ 2 1 ] } Match: 1 2 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 541 2 2 9 10 Wait { 3 } { [ 1 3 ] [ 1 4 ] [ 2 2 ] } Match: -1 -1 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 551 2 3 13 13 Finalize { } { [ 0 3 ] [ 1 4 ] [ 2 3 ] } Match: -1 -1 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 572 0 0 1 -1 Isend 1 0 0 0:1:2: 2 } {} Match: -1 -1 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 422 0 1 4 14 Barrier 0 0:1:2: 2 } { [ 1 2 ] [ 2 1 ] [ 2 2 ] } Match: -1 -1 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 432 0 2 6 -1 Wait { } {} Match: -1 -1 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 442 1 0 2 18 Irecv -1 0 0 0:1:2: 2 } { [ 2 2 ] } Match: 2 1 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 472 1 1 5 15 Barrier 0 0:1:2: 2 } { [ 0 2 ] [ 2 1 ] [ 2 2 ] } Match: -1 -1 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 482 1 2 7 -1 Recv 2 0 0 0:1:2: 2 } {} Match: -1 -1 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 492 2 0 3 16 Barrier 0 0:1:2: 1 2 } { [ 0 2 ] [ 1 2 ] } Match: -1 -1 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 522 2 1 8 17 Isend 1 0 0 0:1:2: 2 } { [ 1 2 ] } Match: 1 0 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 542 2 2 9 19 Wait { 3 } {} Match: -1 -1 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 552 2 3 23 -1 Finalize { } {} Match: -1 -1 File: 68 /home/alan/workspace-gem/MPI_POE_Illustration/src/POE-Illustration.c 572

DEADLOCK
Log File Interpreted by GEM

Runtime Information:
No errors found. Check Browser for warnings.

Transition: 14/133350
Interleaving: 1/1
Step Order for MPI Calls
- Internal Issue
- Program

Code Windows:
Rank: 0
File: io.c Line: 172
Originating Call
{}
MAKECSR(i, your_nvtxs, your_xad);  
maxnedges = (maxnedges < your_xad[your_nvtxs]) ?  
your_xad[your_nvtxs]: maxnedges;
if (pe < npes-1) {
    MPI_Send(void *your_xad, your_nvtxs+1, IDX_DATATYPE, pe, 0,  
    MPI_Send(void *your_vwgts, your_nvtxs*ncon, IDX_DATATYPE, pe, 0,  
    else {
        for (i=0; i<your_nvtxs+1; i++)  
        xad[i] = your_xad[i];
    }

vwgts[i] = your_vwgts[i];
}
}
fclose(pw);
GKfree(&your_xad, &your_vwgts, LTERM);
}
else {
    MPI_Recv(void *xad, nvtxs+1, IDX_DATATYPE, npes, 0, comm);  
    MPI_Recv(void *vwgt, nvtxs*ncon, IDX_DATATYPE, npes-1, 0, comm,
    MPI_Recv(void *vwgt, nvtxs*ncon, IDX_DATATYPE, npes-1, 1, comm,
    graph->nedges = xad[nvtxs];
    adjncy = graph->adjncy = idxmalloc(xad[nvtxs], ParallelReadGraph,
    Paradean graph->graph = idxmalloc(xad[nvtxs], ParallelReadGraph,
}
GEM – ISP’s Eclipse Integration