MPI State Profiling

CScADS Petascale Performance Workshop
Snowbird, UT, July 21, 2008

Marty Itzkowitz
Sun Studio Performance Analyzer Team
Sun Microsystems Inc.
marty.itzkowitz@sun.com
Outline

• MPI Model, and Sun Studio collect
  MPI version: Sun's CT 8, based on Open MPI
• MPI State Profiling
• Experiments and Data
  • Functions
  • Callers and Callees
  • Annotated Source
• Correlating the Data
• Pros and Cons

Work by Oleg Mazurov, Eugene Loh, Yukon Maruyama, Terry Dontje, and Marty Itzkowitz
MPI User-model

• Multiple processes run simultaneously
  • Processes communicate by sending messages
  • May be hybrid: OpenMP and Open MPI
  • May be SIMD or MIMD

• Implementation
  • Processes launched with `mpirun`
    • Either on same node, or across cluster of nodes
  • MPI Data collected with `collect -M` on `mpirun` ...
    • Insert `--` between `mpirun` and its target(s)
  • So-called “founder” experiment has message trace
    • Each process has a “subexperiment” inside founder experiment
MPI State Profiling

• Three states:
  • Not-in-MPI, MPI-Work, MPI-Wait
  • More states could be added (e.g., wait for specific events)
• API to read state
  • Thread-specific, asynchronous-signal-safe
• Called whenever clock profile signal is received
  • Solaris – no matter what thread is doing
  • Linux – only when thread is on CPU
• Callstack recorded with data
MPI State Transitions

- Entry to library: Not-in-MPI ==> MPI-Work
- Start of waiting: MPI-Work ==> MPI-Wait
  - Whether busy waiting, or giving up the CPU
- End of waiting: MPI-Wait ==> MPI-Work
- Return from library: MPI-Work ==> Not-in-MPI
MPI State Metrics

• If User-CPU and MPI-Work:
  • Increment MPI Work Time

• If not User-CPU, or User-CPU and MPI-Wait:
  • Increment MPI Wait Time

• Metrics attributed to frames in the callstack
  • Map to functions
  • Map to source lines
MPI Tracing

• Trace calls to MPI library APIs
  • Open Source VampirTrace
  • Metrics
    • MPI-Sends, MPI-Recvs, MPI-Other
    • MPI-bytes-sent, MPI-bytes-recvd
    • MPI-time
  • Callstack recorded with each event
  • Also MPI Timeline, MPI Charts
    • Sends and receives matched in MPI Timeline
MPI Example: **bt.A.25**

- NAS Parallel Benchmark BT
  - Run 25 MPI processes
- `collect -M on mpirun -np 25 ...`
  - Collect MPI API traces from all 25 processes
    - Open Source VampirTrace
  - Collect MPI State data with clock profiles
# Function List

Sorted by MPI Wait time

<table>
<thead>
<tr>
<th>User CPU (sec)</th>
<th>MPI CPU (sec)</th>
<th>MPI Wait ∨ (sec)</th>
<th>MPI Work ∨ (sec)</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1765.925</td>
<td>1834.497</td>
<td>1719.353</td>
<td>18.723 &lt;Total&gt;</td>
<td></td>
</tr>
<tr>
<td>1765.835</td>
<td>1834.497</td>
<td>1718.313</td>
<td>18.673 _start</td>
<td></td>
</tr>
<tr>
<td>1754.257</td>
<td>1834.497</td>
<td>1660.231</td>
<td>18.673 main</td>
<td></td>
</tr>
<tr>
<td>1754.197</td>
<td>1834.497</td>
<td>1666.271</td>
<td>10.673 MAIN</td>
<td></td>
</tr>
<tr>
<td>1717.001</td>
<td>1191.516</td>
<td>1168.548</td>
<td>17.502 <em>edi</em></td>
<td></td>
</tr>
<tr>
<td>930.291</td>
<td>1010.658</td>
<td>1002.241</td>
<td>10.027 MPI_WAIT</td>
<td></td>
</tr>
<tr>
<td>559.431</td>
<td>413.011</td>
<td>408.755</td>
<td>3.873 y_solve_</td>
<td></td>
</tr>
<tr>
<td>500.450</td>
<td>326.303</td>
<td>332.333</td>
<td>4.203 z_solve_</td>
<td></td>
</tr>
<tr>
<td>9.937</td>
<td>406.437</td>
<td>326.370</td>
<td>1.111 setup_mpi_</td>
<td></td>
</tr>
<tr>
<td>9.917</td>
<td>406.410</td>
<td>328.350</td>
<td>1.111 MPI_INIT</td>
<td></td>
</tr>
<tr>
<td>423.336</td>
<td>265.426</td>
<td>261.093</td>
<td>3.940 x_solve_</td>
<td></td>
</tr>
<tr>
<td>357.950</td>
<td>177.394</td>
<td>168.698</td>
<td>4.963 MPI_WAITAL</td>
<td></td>
</tr>
</tbody>
</table>

Sorted by MPI Wait time:

| User CPU | 15.001 (0.91%) | 176.925 |
| Wall | 17.272 (0.66%) | 190.759 |
| Total Time | 17.272 (0.66%) | 190.759 |
| System CPU | 0.020 (0.11%) | 0.020 |
| User Lock | 0 (0%) | 0 |
| Text Page Fault | 0 (0%) | 0 |
| Data Page Fault | 0 (0%) | 0 |
| Other Wait | 1.171 (0.14%) | 17.272 |
| MPI | 0.000 (0.0%) | 0.000 |
# Caller-Callees

Navigate with MPI Wait time to find delays
Source Display

Find lines where MPI Wait time is high
MPI Example: *is.B.16*

- NAS Parallel Benchmark IS
  - Run 16 MPI processes
  - `collect -M on mpirun -np 16 ...`
  - Collect MPI API traces from all sixteen processes
    - Open Source Vampir Trace
  - Collect MPI State data with clock profiles
Function List

Sorted by MPI Wait time – significant wait time in

MPI_Alltoallv, MPI_Alltoall, MPI_Allreduce

Also, in MPI_Init and MPI_Finalize
Caller-Callees

Navigate with MPI Wait time to find delays
Find lines where MPI Wait time is high:
I. MPI_Allreduce()
Source Display, II

Find lines where MPI Wait time is high:

2. MPI_Alltoallv()
Correlating the Data

• All trace and profile records have callstacks
  • Suppress frames below API-layer of MPI

• Aggregate Messages into Transactions
  • Based on common send/recv callstacks

• Profile records with non-zero MPI-Wait / MPI Work time
  • Must be inside MPI
  • Will match a send or recv trace record callstack

• Attribute MPI-Wait / MPI-Work time to Transactions
  • Both sender and receiver times
  • Select expensive transactions to display
Advantages and Disadvantages

• Advantages:
  • Easy implementation in Open MPI libraries
  • Scalability: profiling allows easy control of data volume
  • Direct navigation to source: no hunting through messages

• Disadvantages
  • Data is statistical, may miss anomalies
  • Dilation (in current prototype)
    • ~ 0.05 – 0.15 μsecs. in latency for small (0-, 8-byte) messages

• We will offer source to Open MPI Community
MPI State Profiling

Supplemental Information
BT -- MPI-Timeline, I

Only 2% of all messages shown
BT -- MPI-Timeline, II

All messages shown
BT -- MPI-Timeline, III

Zoom in, to show individual messages
BT -- MPI Charts, I

Shows time in functions, aggregated over all ranks
BT -- MPI Charts, II

Shows count of messages between pairs of processes
BT -- MPI Charts, III

Shows byte-count of messages between pairs of processes
BT – Traditional Timeline, 1

Shows both profile and MPI Trace Records
BT – Traditional Timeline, II

Profile records

MPI Trace records

Matching callstacks when inside MPI call
IS -- MPI-Timeline, I

Full scale, tiny run, dominated by MPI_Init and MPI_Finalize
IS -- MPI-Timeline, II

Zoomed in, to show pattern during computation
Almost all MPI operations are collectives
MPI State Profiling

Marty Itzkowitz
Sun Studio Performance Analyzer Team
Sun Microsystems Inc.
marty.itzkowitz@sun.com