Agenda

- MPI and MPInside overview
- Profiling capabilities
  - Basic features
  - Collective wait time evaluation
  - Send late time
- Modeling capabilities
- Case study
- Availability
- Conclusion
How important is MPI for SGI

- The Message Passing Interface (MPI) standard
  MPI is a library specification for message-passing, proposed as a standard by a broadly based committee of vendors, implementers, and users.
- SGI is a major provider in the High Performance Processing (HPC) world with 20 machines in the top 500 faster computer in the world

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>Manufacturer</th>
<th>Computer</th>
<th>Country</th>
<th>Year</th>
<th>Cores</th>
<th>RMax</th>
<th>RPeak</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NASA/Ames Research Center/IAS</td>
<td>SGI</td>
<td>Altix ICE 8200 EX, Xeon 64, United States</td>
<td>2009</td>
<td>51200</td>
<td>40700</td>
<td>603829</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>New Mexico Computing Applications Center (NMCAC)</td>
<td>SGI</td>
<td>Altix ICE 9200, Xeon X5690, United States</td>
<td>2007</td>
<td>14000</td>
<td>13200</td>
<td>172032</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Grand Equipment National de Calcul Intensif (GECI)</td>
<td>SGI</td>
<td>Altix ICE 8200 EX, Xeon X5650, France</td>
<td>2009</td>
<td>12288</td>
<td>128400</td>
<td>146736</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Total Exploration Production (TEP)</td>
<td>SGI</td>
<td>Altix ICE 8200 EX, Xeon X5650, France</td>
<td>2009</td>
<td>12288</td>
<td>128400</td>
<td>146736</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>HLRI at Universität Hannover / PRZIN</td>
<td>SGI</td>
<td>Altix ICE 8200 EX, Xeon X6, Germany</td>
<td>2009</td>
<td>7600</td>
<td>82570</td>
<td>99099.6</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>HLRI at ZIB / Konrad Zuse-Zentrum für Informationsverarbeitung (ZIB)</td>
<td>SGI</td>
<td>Altix ICE 8200 EX, Xeon X6, Germany</td>
<td>2009</td>
<td>7600</td>
<td>82570</td>
<td>99099.6</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>US Army Research Laboratory (ARL)</td>
<td>SGI</td>
<td>Altix ICE 8200 Enhanced X, United States</td>
<td>2009</td>
<td>6666</td>
<td>70000</td>
<td>74547.2</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>NASA/Ames Research Center/IAS</td>
<td>SGI</td>
<td>Altix 4700.15 GHz, United States</td>
<td>2009</td>
<td>3728</td>
<td>56520</td>
<td>62395.2</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>Wright-Patterson Air Force Base/DoD ASC</td>
<td>SGI</td>
<td>Altix 4700.15 GHz, United States</td>
<td>2009</td>
<td>3728</td>
<td>56520</td>
<td>62395.2</td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>National Institute for Materials Science</td>
<td>SGI</td>
<td>Altix 4700.15 GHz, Xeon X6, Japan</td>
<td>2009</td>
<td>4096</td>
<td>42600</td>
<td>45876.2</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>HLRI at Universität Halle</td>
<td>SGI</td>
<td>Altix ICE 8200 EX, Xeon X6, Germany</td>
<td>2009</td>
<td>2560</td>
<td>26650</td>
<td>30720.2</td>
<td></td>
</tr>
<tr>
<td>152</td>
<td>HLRI at ZIB / Konrad Zuse-Zentrum für Informationsverarbeitung (ZIB)</td>
<td>SGI</td>
<td>Altix ICE 8200 EX, Xeon X6, France</td>
<td>2009</td>
<td>2560</td>
<td>26650</td>
<td>30720.2</td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>ROCHEP/Compute Canada</td>
<td>SGI</td>
<td>Altix XE320 Cluster, Xeon 2 BG Canada</td>
<td>2009</td>
<td>2464</td>
<td>21600</td>
<td>27688.6</td>
<td></td>
</tr>
<tr>
<td>301</td>
<td>TIFR/IMR</td>
<td>SGI</td>
<td>Altix ICE 8200 EX, Xeon X6, France</td>
<td>2009</td>
<td>2048</td>
<td>21360</td>
<td>22937.6</td>
<td></td>
</tr>
<tr>
<td>302</td>
<td>Wright-Patterson Air Force Base/DoD ASC</td>
<td>SGI</td>
<td>Altix ICE 8200 Enhanced X, United States</td>
<td>2009</td>
<td>2048</td>
<td>21160</td>
<td>22937.6</td>
<td></td>
</tr>
<tr>
<td>356</td>
<td>SGI</td>
<td>SGI</td>
<td>Altix ICE 8200, 3.0 GHz, United States</td>
<td>2007</td>
<td>2048</td>
<td>19360</td>
<td>24576.6</td>
<td></td>
</tr>
<tr>
<td>470</td>
<td>Maine National Laboratory</td>
<td>SGI</td>
<td>Altix ICE 8200, 3.0 GHz, United States</td>
<td>2007</td>
<td>2048</td>
<td>17760</td>
<td>21781.6</td>
<td></td>
</tr>
<tr>
<td>480</td>
<td>University of Minnesota</td>
<td>SGI</td>
<td>Altix ICE 8200 Cluster, United States</td>
<td>2007</td>
<td>2048</td>
<td>17310</td>
<td>21781.6</td>
<td></td>
</tr>
</tbody>
</table>

A large majority of the flops executed with these machines are executed by MPI applications.
A general performance tool tradeoff

- What do we want to know with the tool
  - Data about elements that are important for the MPI library developer may be of little interest for the user that have no way to interact with such elements

- How to use the tool
  - Use the binary the way it is or change it (recompile, relink, insert calls in the source)
  - How to fire up the tool

- What to gather
  - Cumulative measurement stats
  - Modeled results
  - Traces

- What to report
  - Raw data
    - Need for powerful post-processing?
  - Pre-interpreted data

- Amount of resources to make it
To gain a better understanding of the interaction between application and MPI library/interconnect network by diving “inside” the internals of each

For the application developers to understand the consequence of their choices for exchanging data

For our performance engineering group that needs to commit on application performance with future hardware
MPInside Design goals

- To require no re-compilation or re-linking.
- To use a simple command line interface.

- To be useable with thousands of ranks without overhead.
- To work without traces and without post-processing.
  - This is a strong constraint that needs to use innovative and courageous solutions
- To handle various communication models, in particular the perfect interconnect (zero latency, infinite bandwidth).

- To produce simple text, easy to parse, raw output to be processed with common scripting tools (awk,..) and a spreadsheet

- To be portable to any MPI library for its basic features.
- To support the full MPI 1.2 specifications and the MPI-2 one-sided communications.
### MPInside Basic Statistics

#### CPU Times and Overhead

<table>
<thead>
<tr>
<th>Elapsed Times (s)</th>
<th>CPU Utilization</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0486</td>
<td>2.1606</td>
<td></td>
</tr>
<tr>
<td>0.0000</td>
<td>0.0034</td>
<td></td>
</tr>
<tr>
<td>0.0146</td>
<td>0.0033</td>
<td></td>
</tr>
<tr>
<td>0.0471</td>
<td>0.2337</td>
<td></td>
</tr>
<tr>
<td>0.0000</td>
<td>0.0042</td>
<td></td>
</tr>
<tr>
<td>0.0150</td>
<td>0.0020</td>
<td></td>
</tr>
<tr>
<td>0.0458</td>
<td>0.2484</td>
<td></td>
</tr>
<tr>
<td>0.0000</td>
<td>0.0042</td>
<td></td>
</tr>
<tr>
<td>0.1223</td>
<td>0.0023</td>
<td></td>
</tr>
<tr>
<td>0.0450</td>
<td>0.2633</td>
<td></td>
</tr>
<tr>
<td>0.0000</td>
<td>0.0042</td>
<td></td>
</tr>
<tr>
<td>0.1361</td>
<td>0.0016</td>
<td></td>
</tr>
</tbody>
</table>

#### Kbytes with Attribute

<table>
<thead>
<tr>
<th>Kbytes</th>
<th>Send</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>400000</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>400000</td>
</tr>
</tbody>
</table>

#### Number of Request with Attribute

<table>
<thead>
<tr>
<th>Number of Requests</th>
<th>Send</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1000</td>
</tr>
</tbody>
</table>

#### Kbytes with Attribute

<table>
<thead>
<tr>
<th>Kbytes</th>
<th>Recv</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>400000</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>400000</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Number of Request with Attribute

<table>
<thead>
<tr>
<th>Number of Requests</th>
<th>Recv</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1000</td>
</tr>
<tr>
<td>2</td>
<td>1000</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1000</td>
</tr>
</tbody>
</table>

---

**mpi** run –np NNN **MPInside** <cmd>
A simple MPI_Barrier is inserted before the collective function assuming:
<Time collective> = <time to synchronize> + <time collective with fully synchronized arrivals>
Late Senders

setenv MPINSIDE_EVAL_SLT

Calculates per-rank times when sends were late for blocking MPI functions (MPI_Recv, MPI_Wait,..)
Late Senders Time: How to capture it?

- Have a synchronized clock (SGI Altix, UV, future ICE):
  - A mechanism to send/recv a supplemental piece of information (at least the send posting time) with the user buffer needs to be implemented
- Clocks are not synchronized and deviates (the casual situation on clusters)
  - Use a stuttering method:
    - Send/recv first a zero message size then the data
      - Time \( \text{recv} \) = Time Waiting send is posted + time transfer
      - Approximated as
        - Time to get zero message size + time to recv when send is surely posted
      - Only applicable with Bandwidth sensitive applications
- More on next slide
Stuttering method: Just look at my local clock

**Rank 0**
1: MPI_Irecv(from 1, app tag)
2: MPI_send(to 1)
3: MPI_Wait (recv)

**Stuttering method:**
1a: MPI_Irecv(from 1, app tag)
2a: MPI_Isend(to 1, app tag)
2b: MPI_Isend(to 1, data, token tag)
2c: MPI_Wait (request 2a)
2d: MPI_Wait (request 2b)
3a: MPI_Wait (request 1a)
3b: MPI_Recv(from 1, token tag)

**Rank 1**
MPI_Irecv(from 0, app tag)
MPI_send(to 0)
MPI_Wait (recv)

MPI_Irecv(app tag from 0)
MPI_Isend(to 0, app tag)
MPI_Isend(to 0, data, token tag)
MPI_Wait (request 2a)
MPI_Wait (request 2b)
MPI_Wait (request 1a)
MPI_Recv(from 0, token tag)

- The 3a “zero message” Time is the “Send Late Time”
- The 3b time is the time of a transfer with ready send
PARATEC Example

PARATEC 256 ranks, SGI Altix ICE

Send Late time and Collective wait time evaluation

Basic profiling

Elapsed time (s)

MPI ranks

allred
b_allred
wait
w_wait
Other MPI
Compute
Derived Statistics

Linpack 256 CPU on ICE

- **actual Mb/s**
- **Mb/s seen by the application**

![Graph showing the performance of Linpack 256 CPU on ICE with MPI ranks on the x-axis and Mbytes/s on the y-axis. The graph illustrates the actual and perceived bandwidth differences across different MPI ranks.](image-url)
MPInside Modeling

- Uses virtual clocks to perform on-the-fly “what if” experiments. Such virtual clocks are incremented by the measured computational times and by an evaluation of the communication times.
- Communication model:
  - \( T(size) = \text{latency} + \frac{\text{size}}{\text{bandwidth}(\text{size, network load})} \)
- “Perfect” interconnect:
  - \( \text{latency} = 0, \text{bandwidth} = \infty \)
MPInside Modeling continue

- As there is no standard mechanism in MPI for the library to notify tools for internal event a deep knowledge of the MPI library internals is necessary to handle collective function properly. This is why modeling is restricted to the SGI MPI library.

- Perfect interconnect is an exception:
  - For each collective operation all the virtual clocks are exchanged between processors. The latter arrival imposes its clock. Then the collective operation is perfect.
Perfect Interconnect Example

setenv MPINSIDE_MODEL PERFECT+1.0
Case Study

SPEC MPI candidate code, 256 ranks, SGI Altix ICE

- **Computation (Comput)**
- **w_sdrv_R**
- **sdrv_R**
- **b_bcast**
- **bcast**
- **barrier**
- **Other MPI**

**SLT + collective wait time**

**Perfect interconnect**
Communication “Stiffness”

Such information is carried in the supplemental piece of information necessary for the Send Late Time evaluation.
Lowering the Stiffness

EM code performance on SGI Altix ICE

Scaling relative to 64 ranks

- Original, SendRecv, stiffness = 38
- New, Isend / Irecv, stiffness = 2
MPInside: MPI function “branches” with “partner” cross references

Run:
setenv MPINSIDE_CALL_STACK_DEPTH 5
Setenv MPINSIDE_CROSS_REFERENCE
mpirun -np xxx MPInside your_apps your_args

MPInside report rank 0

<table>
<thead>
<tr>
<th>Function</th>
<th>Brid Time(s)</th>
<th>Self%</th>
<th>Tot%</th>
<th>#reqs_S</th>
<th>#reqs_R</th>
<th>avr_szS</th>
<th>avr_szR</th>
<th>W_miss%</th>
<th>Rcv_W(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Recv</td>
<td>#265</td>
<td>1.552</td>
<td>38.06</td>
<td>38.1</td>
<td>0</td>
<td>105</td>
<td>0</td>
<td>182972</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Ancestors: HPL_spreadT HPL_pdlaswp01T HPL_pdupdateTT HPL_pdgesv0 HPL_pdgesv HPL_pdtest

...........

<table>
<thead>
<tr>
<th>Function</th>
<th>Brid Time(s)</th>
<th>Self%</th>
<th>Tot%</th>
<th>#reqs_S</th>
<th>#reqs_R</th>
<th>avr_szS</th>
<th>avr_szR</th>
<th>W_miss%</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Send</td>
<td>#19</td>
<td>0.125</td>
<td>3.06</td>
<td>90.0</td>
<td>201</td>
<td>0</td>
<td>370801</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Ancestors: HPL_bcast_1rinM HPL_bcast HPL_pdgesv0 HPL_pdgesv HPL_pdtest main

...........

<table>
<thead>
<tr>
<th>Function</th>
<th>Brid Time(s)</th>
<th>Self%</th>
<th>Tot%</th>
<th>#reqs_S</th>
<th>#reqs_R</th>
<th>avr_szS</th>
<th>avr_szR</th>
<th>W_miss%</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Wait</td>
<td>#524</td>
<td>0.983</td>
<td>24.10</td>
<td>62.2</td>
<td>0</td>
<td>1200</td>
<td>0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Ancestors: HPL_rollT HPL_pdlaswp01T HPL_pdupdateTT HPL_pdgesv0 HPL_pdgesv HPL_pdtest
Partners_l_0: 0:#273:100.00

...........

<table>
<thead>
<tr>
<th>Function</th>
<th>Brid Time(s)</th>
<th>Self%</th>
<th>Tot%</th>
<th>#reqs_S</th>
<th>#reqs_R</th>
<th>avr_szS</th>
<th>avr_szR</th>
<th>W_miss%</th>
<th>Rcv_W(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Irecv</td>
<td>#273</td>
<td>0.026</td>
<td>0.63</td>
<td>96.9</td>
<td>0</td>
<td>1200</td>
<td>0</td>
<td>70269</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Ancestors: HPL_rollT HPL_pdlaswp01T HPL_pdupdateTT HPL_pdgesv0 HPL_pdgesv HPL_pdtest

...........

• More about Received partners on next slide.
MPInside:
MPI Receive function branch partners

- Partner list format: **CPU:#Branch:Wait:Send_late**

**CPU:** Rank number that did an MPI Send/Isend for this branch:

**#Branch:** MPI_Send/Isend Branch ident(brid):

**Wait:** percent of this MPI_Recv that involved this “A” rank “#B” MPI Send/Isend branch

**Send_late:** percent of this MPI_Recv where the corresponding Send was arriving late

For example: “240:#19:14.81:97.18” means:
This MPI_Recv branch was “partner” with the MPI_Send branch id 19 of CPU 240 and this partnership is accountable for 14.81% of this MPI_Recv branch communication time and 97.18% of this 14.81% was just wait because the sends were arriving late
Basic profiling functionality is supported for SGI MPT, Intel MPI, HP MPI and ScaliMPI Platform MPI
  – Open MPI support to be added soon.

General modeling capabilities require detailed knowledge of the inner workings of the library
  – Current support is for SGI MPT only.

Perfect interconnect modeling is currently supported on all MPI supported. MPInside is available via SupportFolio
Tools need MPI standardizations

- Performance tools are of great importance for parallel applications in particular for MPI applications.
- MPI Standard only provides the “PMPI” mechanism allowing easy wrapping of MPI functions
  - Better than nothing but this is not much as wrapping is easy
- Advance profiling interfaces should be part of the standard:
  - For notification to the tools about MPI internal library events
    - P2P collective transfers
    - Operation delayed because of lack of buffers
  - ..
  - A mechanism to carry supplemental information that the tools may wish to associate with user messages
- Standard don’t want to impose a particular implementation but all the MPI work more or less the same. Based on this experience more attention should be given to MPI tools