Outline

- Practical Performance Analysis and Tuning
  - Debugging
    - TotalView
    - DDT
  - User Code Profiling
    - gprof
  - MPI Profiling
    - mpiP
  - MPI Analysis and Checking
    - MARMOT
Practical Performance Analysis and Tuning

- Successful tuning is combination of
  - Right algorithm and libraries
  - Compiler flags and pragmas / directives
    (Learn and use them)
  - THINKING

- Measurement is better than reasoning / intuition (= guessing)
  - To determine performance problems
  - To validate tuning decisions / optimizations
    (after each step!)

- It is easier to optimize a slow correct program than to debug a fast incorrect one
  - Debugging before Tuning
  - Nobody really cares how fast you can compute the wrong answer

- The 80/20 rule
  - Program spends 80% time in 20% of code
  - Programmer spends 20% effort to get 80% of the total speedup possible in the code
  - Know when to stop!

- Don’t optimize what doesn’t matter
  - Make the common case fast
Typical Performance Analysis Procedure

1. Do I have a performance problem at all?
   - Time / hardware counter measurements
   - Speedup and scalability measurements
2. What is the main bottleneck (calculation/communication...)?
   - Flat profiling (sampling / prof)
3. Where is the main bottleneck?
   - Call graph profiling (gprof)
   - Detailed (basic block) profiling
4. Where is the bottleneck exactly and why is it there?
   - Trace selected parts to keep trace files manageable

- Does my code have scalability problems?
  - Profile code for typical small and large processor count
  - Compare profiles function-by-function

Limiting Trace File Size

- Use smallest number of processors possible
- Use smallest input data set (e.g. number of time steps)
- Limit trace file size
  - By environment variables / API functions / config files?
- Trace only selected parts by using control functions to switch tracing on/off
  - Select important parts only
  - For iterative programs: trace some iterations only
- Trace only important events
  - No MPI administrative funcs and/or MPI collective funcs
  - Only user function in call tree level 1 or 2
  - Never functions called in (inner) loops

NOTE: Make sure to collect complete message traffic!
- For non-blocking (MPI_I*) include MPI_Wait*+MPI_Test*
Outline

- Practical Performance Analysis and Tuning
- Debugging
  - TotalView
  - DDT
- User Code Profiling
  - gprof
- MPI Profiling
  - mpiP
- MPI Analysis and Checking
  - MARMOT

Outline

- Practical Performance Analysis and Tuning
- Debugging
  - TotalView
  - DDT
- User Code Profiling
  - gprof
- MPI Profiling
  - mpiP
- MPI Analysis and Checking
  - MARMOT

Parallel Debugger

- UNIX Symbolic Debugger
  for C, C++, f77, f90, PGI HPF, assembler programs
- „Standard“ debugger
- Special, non-traditional features
  - Multi-process and multi-threaded
  - C++ support (templates, inheritance, inline functions)
  - F90 support (user types, pointers, modules)
  - 1D + 2D Array Data visualization
  - Support for parallel debugging
    - MPI (Vendor+MPICH: automatic attach, message queues)
    - OpenMP (Vendor, PGI, KAI)
    - pthreads (AIX, IRIX)

- http://www.etnus.com
TotalView Process Window

- Stack trace
- Breakpoints
- Active threads
- Toolbar for common options
- Local variables for selected Stack frame
- Source code window

Totalview: Non-standard Features

- Call graph
- Data visualization
- Message queue graph
**DDT Parallel Debugger**

- UNIX Graphical Debugger for C, C++, f77, f90 programs
- Modern, easy-to-use debugger
- Special, non-traditional features
  - Multi-process and multi-threaded
  - 1D + 2D Array Data visualization
  - Support for MPI parallel debugging
    - (Vendor+MPICH: automatic attach, message queues)
  - Job submission from within debugger
- [http://www.allinea.com](http://www.allinea.com)

---

**DDT Main Window**

1. Menu bar
2. Process controls
3. Process group window
4. File window
5. Code window
6. Variable window
7. Evaluate window
8. Output window
9. Status bar
**DDT: Non-standard Features**

- **Message queue graph**
- **Memory Usage**
- **Multi-Dimensional Array Viewer**

---

**Outline**

- Practical Performance Analysis and Tuning
  - Debugging
    - TotalView
    - DDT
  - User Code Profiling
    - gprof
  - MPI Profiling
    - mpiP
  - MPI Analysis and Checking
    - MARMOT
User Code Profiling: gprof

- gprof shows where the program spent its time and the dynamic call tree
  - Indicates which functions are candidates for tuning
  - Identifies which functions are being called

- Profiler information collected during execution
  - Can be used on large and complex programs
- Compile and link with –pg

- Program execution generates gmon.out file (recent versions support GMON_OUT_PREFIX environment variable to control the name)

```
gprof [options] [executable [profile-files...]]
gprof
```

Gprof Flat Profile

- The flat profile shows the total amount of time the program spent executing each function
  - % Time (add to 100%)
  - Cumulative seconds (inclusive time)
  - Self seconds (exclusive time)
  - Calls
  - Self ms/call (average exclusive time per call)
  - Total ms/call (average inclusive time per call)
  - Name
- Time interval is 0.01 seconds
- Functions are sorted by decreasing run-time
**Gprof Flat Profile Example**

- Shows percentage of time spent in each function (inclusive/exclusive), the number of calls, and time per call. Sorted by total.

<table>
<thead>
<tr>
<th>time % cumulative</th>
<th>self seconds</th>
<th>self seconds</th>
<th>calls</th>
<th>ms/call</th>
<th>ms/call</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>82.90</td>
<td>31.55</td>
<td>31.55</td>
<td>27648</td>
<td>1.14</td>
<td>1.14</td>
<td>pc_jac2d_blk3_</td>
</tr>
<tr>
<td>7.54</td>
<td>34.42</td>
<td>2.87</td>
<td>1</td>
<td>2870.00</td>
<td>37910.00</td>
<td>cg3_blk_</td>
</tr>
<tr>
<td>5.96</td>
<td>36.69</td>
<td>2.27</td>
<td>28672</td>
<td>0.08</td>
<td>0.08</td>
<td>matvec2d_</td>
</tr>
<tr>
<td>2.13</td>
<td>37.50</td>
<td>0.81</td>
<td>55296</td>
<td>0.01</td>
<td>0.01</td>
<td>dot_prod2d_</td>
</tr>
<tr>
<td>0.87</td>
<td>37.83</td>
<td>0.33</td>
<td>54</td>
<td>6.11</td>
<td>6.11</td>
<td>add_exchange2d_</td>
</tr>
<tr>
<td>0.34</td>
<td>37.96</td>
<td>0.13</td>
<td></td>
<td></td>
<td></td>
<td>main</td>
</tr>
<tr>
<td>0.21</td>
<td>38.04</td>
<td>0.08</td>
<td>27</td>
<td>2.96</td>
<td>2.96</td>
<td>cs_jac2d_blk3_</td>
</tr>
<tr>
<td>0.03</td>
<td>38.05</td>
<td>0.01</td>
<td></td>
<td></td>
<td></td>
<td>OpenUnit</td>
</tr>
<tr>
<td>0.03</td>
<td>38.06</td>
<td>0.01</td>
<td></td>
<td></td>
<td></td>
<td>dtime_</td>
</tr>
</tbody>
</table>

**Gprof Call Graph Profile**

- Shows time spent in each function and its children.

```
<table>
<thead>
<tr>
<th>index</th>
<th>%time</th>
<th>self descents</th>
<th>called</th>
<th>called+</th>
<th>name</th>
<th>parents</th>
<th>index</th>
<th>children</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.17</td>
<td>2.83</td>
<td></td>
<td>1</td>
<td>1/1</td>
<td>__start [2]</td>
<td>main [1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>relax [3]</td>
<td>saveOutput</td>
<td>[27]</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>malloc [5]</td>
<td>readInput</td>
<td>[40]</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>update [68]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
6.6s              | <spontaneous>|
| [2]   | 99.3  | 0.00          | 3.00  | 1/1     | __start [2]| main [1]  |       |          |
| 0.17  | 2.83  |               | 1     | 1/1     | main [1]   | exit [94] |       |          |
| 0.00  | 0.00  |               |       |         | exit [94]  |          |       |          |
```

```
Outline

- Practical Performance Analysis and Tuning
- Debugging
  - TotalView
  - DDT
- User Code Profiling
  - gprof
- MPI Profiling
  - mpiP
- MPI Analysis and Checking
  - MARMOT

MPI Profiling: mpiP

- Scalable, light-weight MPI profiling library
- Generates detailed text summary of MPI behavior
  - Time spent at each MPI function callsite
  - Bytes sent by each MPI function callsite (where applicable)
  - Configurable traceback depth for function callsites
- Controllable from program using MPI_Pcontrol
  - Allows you to profile just one code module or cycle
  - Allows mpiP profile dumps mid-run (in version 2.7)
- Uses PMPI interface ⇒ only re-link of application necessary

http://www.llnl.gov/CASC/mpip/
### mpiP Text Output Example

####MPI Time (seconds)

<table>
<thead>
<tr>
<th>Task</th>
<th>AppTime</th>
<th>MPITime</th>
<th>MPI%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>0.000243</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>99.92</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td>99.92</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>10</td>
<td>99.92</td>
</tr>
<tr>
<td>*</td>
<td>40</td>
<td>30</td>
<td>74.94</td>
</tr>
</tbody>
</table>

####Aggregate Time (top twenty, descending, milliseconds)

<table>
<thead>
<tr>
<th>Call</th>
<th>Site</th>
<th>Time</th>
<th>App%</th>
<th>MPI%</th>
<th>COV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barrier</td>
<td>2</td>
<td>3e+04</td>
<td>75.00</td>
<td>100.00</td>
<td>0.67</td>
</tr>
<tr>
<td>Barrier</td>
<td>1</td>
<td>0.405</td>
<td>0.00</td>
<td>0.00</td>
<td>0.59</td>
</tr>
</tbody>
</table>

####Aggregate Sent Message Size (top twenty, descending, bytes)

<table>
<thead>
<tr>
<th>Call</th>
<th>Site</th>
<th>Count</th>
<th>Total</th>
<th>Avrg</th>
<th>MPI%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send</td>
<td>7</td>
<td>320</td>
<td>1.92e+06</td>
<td>6e+03</td>
<td>99.96</td>
</tr>
<tr>
<td>Bcast</td>
<td>1</td>
<td>12</td>
<td>336</td>
<td>28</td>
<td>0.02</td>
</tr>
</tbody>
</table>

####Callsites: 2

<table>
<thead>
<tr>
<th>ID</th>
<th>Lev</th>
<th>File/Address</th>
<th>Line</th>
<th>Parent_Funct</th>
<th>MPI_Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>9-test-mpip-time.c</td>
<td>52</td>
<td>main</td>
<td>Barrier</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>9-test-mpip-time.c</td>
<td>61</td>
<td>main</td>
<td>Barrier</td>
</tr>
</tbody>
</table>

####Callsite Time statistics (all, milliseconds): 8

<table>
<thead>
<tr>
<th>Name</th>
<th>Site</th>
<th>Rank</th>
<th>Count</th>
<th>Max</th>
<th>Mean</th>
<th>Min</th>
<th>App%</th>
<th>MPI%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barrier</td>
<td>1</td>
<td>*</td>
<td>4</td>
<td>0.174</td>
<td>0.137</td>
<td>0.107</td>
<td>0.00</td>
<td>44.03</td>
</tr>
<tr>
<td>Barrier</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0.136</td>
<td>0.136</td>
<td>0.136</td>
<td>0.00</td>
<td>55.97</td>
</tr>
<tr>
<td>Barrier</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1e+04</td>
<td>1e+04</td>
<td>1e+04</td>
<td>1e+04</td>
<td>99.92</td>
</tr>
<tr>
<td>Barrier</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1e+04</td>
<td>1e+04</td>
<td>1e+04</td>
<td>1e+04</td>
<td>99.92</td>
</tr>
<tr>
<td>Barrier</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1e+04</td>
<td>1e+04</td>
<td>1e+04</td>
<td>1e+04</td>
<td>99.92</td>
</tr>
<tr>
<td>Barrier</td>
<td>2</td>
<td>*</td>
<td>4</td>
<td>1e+04</td>
<td>7.5e+03</td>
<td>0.136</td>
<td>74.94</td>
<td>100.00</td>
</tr>
</tbody>
</table>
mpiPview: An MpiP Output Viewer

- Organizes and condenses mpiP output
  - Allow users to find key mpiP data quickly
  - Hides complexity of large scale runs until needed
  - Shows source code for the MPI callsites reported on
- Open source, portable, part of Tool Gear
  - Download from http://www.llnl.gov/CASC/tool_gear/

Outline

- Practical Performance Analysis and Tuning
- Debugging
  - TotalView
  - DDT
- User Code Profiling
  - gprof
- MPI Profiling
  - mpiP
- MPI Analysis and Checking
  - MARMOT
**MPI Analysis and Checking: MARMOT**

- Tool for automatic runtime analysis of MPI applications
  - Detect incorrect use of MPI
  - Detect non-portable constructs
  - Detect possible race conditions and deadlock
- Reported as human-readable log file
- Classified into errors, warnings, and remarks
- Uses PMPI interface \(\Leftrightarrow\) only re-link of application necessary

Collaborative project between
- HLRS, Stuttgart
- ZIH, Dresden


---

**MARMOT Architecture**

User Application

PMPI Interface

MARMOT Core Module

MPI Library

Rank 0

\[\cdots\]

Rank N

\[\cdots\]

Rank N+1

Debug Server (additional Process)
Examples of MARMOT Checks

- **Local checks**
  - Verification of proper construction and usage of MPI resources like communicators, groups, data types, requests, ...
  - Verification of all other MPI call arguments like ranks, tags, ...

- **Global checks (work in progress)**
  - Check matching send/receive pairs for consistency
  - Check collective calls for consistency
  - Signal conditions, e.g. deadlocks

- **Three different log file types (selected by an environment variable)**
  - ASCII
  - HTML
  - Cube (using KOJAK’s presentation tool CUBE)

---

Example MARMOT ASCII Log File

% export MARMOT_LOGFILE_TYPE=0

8: Warning from rank 0 with Text: WARNING: Tag = 39000 greater than 32767 !
MPI only guarantees tags up to this. THIS implementation allows tags up to 136479576.

On call: MPI_Send
for MPI-Standard see:/usr/local/share/doc/marmot/MPI-Standard/...

9: Note from rank 0 with Text: performing

On call: MPI_Recv

9: Error from rank -1 with Text: WARNING: all clients are pending!

Last calls (max. 10) on node 0:

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Call</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MPI_Init(argc, **argv)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>MPI_Comm_rank(comm = MPI_COMM_WORLD, *rank)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>MPI_Comm_size(comm = MPI_COMM_WORLD, *size)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>MPI_Send(*buf, count = 1, datatype = MPI_INT, dest = 1, tag ...</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>MPI_Recv(*buf, count = 1, datatype = MPI_INT, source = 1, ...</td>
<td></td>
</tr>
</tbody>
</table>

Last calls (max. 10) on node 1:

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Call</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>MPI_Init(argc, **argv)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>MPI_Comm_rank(comm = MPI_COMM_WORLD, *rank)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>MPI_Comm_size(comm = MPI_COMM_WORLD, *size)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>MPI_Recv(*buf, count = 1, datatype = MPI_INT, source = 0, ...</td>
<td></td>
</tr>
</tbody>
</table>
Example MARMOT HTML Log File

% export MARMOT_LOGFILE_TYPE=1

MARMOT CUBE Log File

% export MARMOT_LOGFILE_TYPE=2