High performance tools to debug, profile, and analyze your applications

High-productivity development tools for science

Beau Paisley
bpaisley@allinea.com
Industry Standard Tools
Today’s Challenge

Q: What is the impact of current trends in HPC on your application?

Q: How can you make your science run well on the available system?

A: Development.

Development implies both fixing problems and optimizing the computation.
Debugging in Practice, ...

- Run
- Compile
- Crash
- Insert print statements
- Hypothesis
Optimization in Practice, ...

- Insert timers
- Change code
- Run code
- Analyse result
- Change code
- Analyse result
- Run code
- Insert timers
HPC means being able to work productively on remote machines

- Linux
- OS/X
- Windows
- Multiple hop SSH
- RSA + Cryptocard
- Uses server license
Submit to job queues or run interactively on any system
Simplified Code Optimization

- Small data files
- <5% slowdown
- No instrumentation
- No recompilation
While still connected to the server we switch to the debugger.
It's already configured to reproduce the profiling run.
Our tools understand your version control system
Most new bugs are in or around recently changed code
We can visualize multidimensional data across all processes.
And generate statistical summaries of their contents.
Variables are compared across all threads and processes automatically
These arrays are all pointing to the same area of memory!
Verify our fix before committing it
A tracepoint shows the arrays pointers are swapping correctly now
Debug with the Scientific Method
Debugging While you Sleep

![DDT Debugging Interface]

- **Messages**:
  - # 2: Type: i, Time: 0:01.649, Processes: 0-3, Message: Startup complete.
  - # 3: Type: n/a, Time: 0:01.655, Message: Select process group All
  - # 4: Type: i, Time: 0:01.656, Processes: 0-3, Message: Add tracepoint for cstartmpi.c:109 Vars: x, y
  - # 5: Type: n/a, Time: 0:01.658, Message: Add Expression to Evaluate: my_rank
  - # 6: Type: n/a, Time: 0:01.658, Message: Add Expression to Evaluate: tables
  - # 7: Type: Additional Information

- **Stacks**

- **Current Stack**

- **Locals**
  - Name: argc, Value: 1
  - Name: argv, Value: 0xfffffffff548
  - Name: beingMatched, Value: 32767
  - Name: bigArray, Value: 0
A Productive HPC Development Workflow
Analyze and tune application performance

A single-page report on application performance for users and administrators

Identify configuration problems and resource bottlenecks immediately

Track mission-critical performance over time and after system upgrades

Ensure key applications run at full speed on a new cluster or architecture

Summary: clover_leaf is CPU-bound in this configuration

CPU

<table>
<thead>
<tr>
<th>Description</th>
<th>Time (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time spent running application code</td>
<td>10.0%</td>
</tr>
<tr>
<td>Time spent in system calls</td>
<td>90.0%</td>
</tr>
</tbody>
</table>

This application run was CPU-bound. A breakdown of this time and advice for investigating further is in the CPU section below.

As little time is spent in system calls, this code may also benefit from running at larger scales.

MPI

<table>
<thead>
<tr>
<th>Description</th>
<th>Time (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time in point-to-point calls</td>
<td>42.7%</td>
</tr>
<tr>
<td>Effective process rate</td>
<td>1.00 MHz</td>
</tr>
</tbody>
</table>

Most of the time is spent in point-to-point calls with a low transfer rate. This can be caused by inefficient message sizes, such as many small messages, or by inefficient blocks causing processes to wait.
The collective transfer rate is very low. This suggests load imbalance or causing synchronization overhead; use an MPI profiler to investigate further.

I/O

<table>
<thead>
<tr>
<th>Description</th>
<th>Time (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Effective process read rate</td>
<td>0.00 bytes</td>
</tr>
</tbody>
</table>

The per-core performance is memory-bound. Use a profiler to identify I/O containing latency delays that impact overall performance.

Little time is spent in I/O-bound execution. Check the compiler optimization advice to see if any topups could be improved.

OpenMP

<table>
<thead>
<tr>
<th>Description</th>
<th>Time (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compilation</td>
<td>100.0%</td>
</tr>
<tr>
<td>Synchronization</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

This application run was CPU-bound. A breakdown of this time and advice for investigating further is in the CPU section below.

As little time is spent on OpenMP tasks, this code may also benefit from running at larger scales.
A single-page report for users and administrators

Summary: clover_leaf is **CPU-bound** in this configuration

- **CPU**: 80.6%
  - Time spent running application code. High values are usual.
  - This is high; check the CPU performance section for optimi
- **MPI**: 19.4%
  - Time spent in MPI calls. High values are usually bad.
  - This is low; this code may benefit from increasing the proc
- **I/O**: 0.1%
  - Time spent in filesystem I/O. High values are usually bad.
  - This is very low; however single-process I/O often causes i

This application run was CPU-bound. A breakdown of this time and advice for investigating further is if
As little time is spent in MPI calls, this code may also benefit from running at larger scales.

**CPU**
- A breakdown of the 80.6% CPU time:
  - Single-core code: 0.4%
  - OpenMP regions: 99.6%
  - Scalar numeric ops: 42.4%
  - Vector numeric ops: 4.9%
  - Memory accesses: 53.6%

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.
Little time is spent in vectorized instructions. Check the compiler’s vectorization advice to see why key loops could not be vectorized.

**MPI**
- A breakdown of the 19.4% MPI:
  - Time in collective calls
  - Time in point-to-point calls
  - Effective process collective rate
  - Effective process point-to-point rate

Most of the time is spent in point-to-point. This can be caused by inefficient message passing, or by imbalanced workload.
The collective transfer rate is very low causing synchronization overhead; u

**I/O**
- A breakdown of the 0.1% I/O time:
  - Time in reads: 0.0%
  - Time in writes: 100.0%
  - Effective process read rate: 0.00 bytes/s

**OpenMP**
- A breakdown of the 99.6% time:
  - Computation
  - Synchronization
  - Physical core utilization

---

**No recompilation or instrumentation necessary**

**Less than 5% application slowdown on most systems**

**Summarizes performance of individual application runs**

**Save data in HTML, TXT or CSV formats for analysis**
High performance tools to debug, profile, and analyze your applications