ARCHER Performance and Debugging Tools

Slides contributed by Cray and EPCC
The Porting/Optimisation Cycle

Modify

Optimise

Debug

Cray Performance Analysis Toolkit (CrayPAT)

ATP, STAT, FTD, DDT
Debug

ATP, STAT, FTD, Totalview
Abnormal Termination Processing (ATP)

For when things break unexpectedly…
(Collecting back-trace information)
Debugging in production and scale

- Even with the most rigorous testing, bugs may occur during development or production runs.
  - It can be very difficult to recreate a crash without additional information
  - Even worse, for production codes need to be efficient so usually have debugging disabled
- The failing application may have been using tens of or hundreds of thousands of processes
  - If a crash occurs one, many, or all of the processes might issue a signal.
  - We don’t want the core files from every crashed process, they’re slow and too big!
  - We don’t want a backtrace from every processes, they’re difficult to comprehend and analyze.
ATP Description

• Abnormal Termination Processing is a lightweight monitoring framework that detects crashes and provides more analysis
  • Designed to be so light weight it can be used all the time with almost no impact on performance.
  • Almost completely transparent to the user
    • Requires atp module loaded during compilation (usually included by default)
    • Output controlled by the ATP_ENABLED environment variable (set by system).
  • Tested at scale (tens of thousands of processors)

• ATP rationalizes parallel debug information into three easier to user forms:
  1. A single stack trace of the first failing process to stderr
  2. A visualization of every processes stack trace when it crashed
  3. A selection of representative core files for analysis
Usage

Compilation – environment must have module loaded

module load atp

Execution (scripts must explicitly set these if not included by default)

export ATP_ENABLED=1
ulimit -c unlimited

More information (while atp module loaded)

man atp

ATP respects ulimits on corefiles. So to see corefiles the ulimit must change. On crash ATP will produce a selection of relevant cores files with unique, informative names.
Stack Trace Analysis Tool (STAT)

For when nothing appears to be happening…
STAT

- Stack Trace Analysis Tool (STAT) is a cross-platform tool from the University of Wisconsin-Madison.
- ATP is based on the same technology as STAT. Both gather and merge stack traces from a running application’s parallel processes.
- It is very useful when application seems to be stuck/hung
- Full information including use cases is available at http://www.paradyn.org/STAT/STAT.html
- Scales to many thousands of concurrent process, only limited by number file descriptors
- STAT 1.2.1.3 is the default version on Sisu.
2D-Trace/Space Analysis

Appl

Appl

Appl

Appl

Appl

main

10798:[0,3-10799] 1:[1]

PMPI_BARRIER

do_SendOrStall

MPI_Waitall

10798:[0,3-10799]

MPIDR_CRAY_SMPClus_Barrier

10798:[0,3-10799]

MPIR_BARRIER

10798:[0,3-10799]

1:[2]

MPIC_Sendrecv

MPIDR_CRAY_Progress_wait

1:[2]

MPIDR_CRAY_progress

1:[2]

MPIDR_CRAY_ptldev_progress

archer

epcc
Using STAT

Start an interactive job...

module load stat

<launch job script> &

# Wait until application hangs:

STAT <pid of aprun>

# Kill job

statview STAT_results/<exe>/<exe>.0000.dot
LGDB

Diving in through the command line…
LGDB is a line mode parallel debugger for Cray systems
- Available through cray-lgdb module
- Binaries should be compiled with debugging enabled, e.g. –g. (Or Fast-Track Debugging see later).
- The recent 2.0 update has introduced new features. All previous syntax is deprecated

It has many of the features of the standard GDB debugger, but includes extensions for handling parallel processes.

It can launch jobs, or attach to existing jobs

1. To launch a new version of <exe>
   1. Launch an interactive session
   2. Run lgdb
   3. Run launch $pset{nprocs} <exe>

2. To attach to an existing job
   1. find the <apid> using apstat.
   2. launch lgdb
   3. run attach $<pset> <apid> from the lgdb shell.
DDT Debugging

Graphical debugging on ARCHER
Debugging MPI programs: DDT

- Allinea DDT installed on ARCHER
  - TotalView no longer available

- The recommended way to use DDT on ARCHER is to install the free DDT remote client on your workstation or laptop and use this to run DDT on ARCHER.

- The version of the DDT remote client must match the version of DDT installed on ARCHER
  - currently version 4.1
Compiling for debugging

- install the source code on the /work filesystem

- compile the executable into a location on /work to ensure that the running job can access all of the required files.

- Turn off compiler optimisation and turn on debugging
  - -O0 -g
Remote client

• Install the remote client and run it:

• Configure Remote Launch
  • Hostname: username@login.archer.ac.uk
  • Installation Directory: /opt/cray/ddt/4.0.1.0_32296

• Configure job submission
  • Click “Options”
  • Choose “Job Submission”
  • Change submission template to:
    • /home/y07/y07/cse/allinea/templates/archer_phase1.qtf
  • Including “Edit Queue Submission Parameters… (can also be done at run time)
  • Change time limit if required
  • Add budget code
Job Submitted

Your job has been submitted to the queue. Allinea DDT will continue automatically once the job has been started.

Waiting for job to start...
```c
return(str);

int main(int argc, char *argv[])
{
    int rank, thread;
    cpu_set_t coremask;
    char clBuf[7 * CPU_SETSIZE], hbuf[64];
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    memset(clBuf, 0, sizeof(clBuf));
    memset(hbuf, 0, sizeof(hbuf));
    (void) gethostname(hbuf, sizeof(hbuf));
    #pragma omp parallel private(thread, coremask, clBuf)
    {
        thread = omp_get_thread_num();
        (void) sched_getaffinity(0, sizeof(coremask), &coremask);
        cpuset_to_cstr(&coremask, clBuf);
        #pragma omp barrier
    }
    thread = omp_get_thread_num();
    (void) sched_getaffinity(0, sizeof(coremask), &coremask);
    cpuset_to_cstr(&coremask, clBuf);
    #pragma omp barrier
```
DDT options

• Play: run processes in current group until they are stopped.
• Pause: pause processes in current group for examination.
• Add Breakpoint: adds a breakpoint at a line of code, or a function, causing processes to pause when they reach it.
• Step Into: step the current process group by a single line or, if the line involves a function call, into the function instead.
• Step Over: steps the current process group by a single line.
• Step Out: will run the current process group to the end of their current function, and return to the calling location.
Optimise

Cray Performance Analysis Toolkit (CrayPAT)
### Sampling

**Advantages**
- Only need to instrument main routine
- Low Overhead – depends only on sampling frequency
- Smaller volumes of data produced

**Disadvantages**
- Only statistical averages available
- Limited information from performance counters

### Event Tracing

**Advantages**
- More accurate and more detailed information
- Data collected from every traced function call not statistical averages

**Disadvantages**
- Increased overheads as number of function calls increases
- Huge volumes of data generated

The best approach is *guided tracing*.

*e.g. Only tracing functions that are not small (i.e. very few lines of code) and contribute a lot to application’s run time.*

APA is an automated way to do this.
Automatic Profile Analysis

A two step process to create a guided event trace binary.
Program Instrumentation - Automatic Profiling Analysis

- **Automatic profiling analysis** (APA)

- Provides simple procedure to instrument and collect performance data as a first step for novice and expert users

- Identifies top time consuming routines

- Automatically creates instrumentation template customized to application for future in-depth measurement and analysis
Steps to Collecting Performance Data

- Access performance tools software
  ```bash
  % module load perftools
  ```
- Build application keeping .o files (CCE: -h keepfiles)
  ```bash
  % make clean
  % make
  ```
- Instrument application for automatic profiling analysis
  - You should get an instrumented program a.out+pat
    ```bash
    % pat_build -O apa a.out
    ```
- Run application to get top time consuming routines
  - You should get a performance file (“<sdatafile>.xf”) or multiple files in a directory <sdatadir>
    ```bash
    % aprun ... a.out+pat (or qsub <pat script>)
    ```

We are telling pat_build that the output of this sample run will be used in an APA run
Steps to Collecting Performance Data (2)

• Generate text report and an .apa instrumentation file
  
  % pat_report -o my_sampling_report [<sdatafile>.xf | <sdatadir>]

• Inspect .apa file and sampling report

• Verify if additional instrumentation is needed
Generating Event Traced Profile from APA

- Instrument application for further analysis (a.out+apa)
  
  \% pat_build -O <apafilename>.apa

- Run application

  \% aprun ... a.out+apa (or qsub <apa script>)

- Generate text report and visualization file (.ap2)

  \% pat_report -o my_text_report.txt [<datafile>.xf | <datadir>]

- View report in text and/or with Cray Apprentice²

  \% app2 <datafilename>.ap2
Analysing Data with \texttt{pat\_report}
Using pat_report

- Always need to run pat_report at least once to perform data conversion
  - Combines information from xf output (optimized for writing to disk) and binary with raw performance data to produce ap2 file (optimized for visualization analysis)
  - Instrumented binary must still exist when data is converted!
  - Resulting ap2 file is the input for subsequent pat_report calls and Apprentice²
  - xf and instrumented binary files can be removed once ap2 file is generated.

- Generates a text report of performance results
  - Data laid out in tables
  - Many options for sorting, slicing or dicing data in the tables.
    - `pat_report -O <table option> *.ap2`
    - `pat_report -O help` (list of available profiles)
  - Volume and type of information depends upon sampling vs tracing.
Job Execution Information

CrayPat/X:  Version 6.1.2 Revision 11877 (xf 11595)  09/27/13 12:00:25

Number of PEs (MPI ranks):  32

Numbers of PEs per Node:  16  PEs on each of  2  Nodes

Numbers of Threads per PE:  1

Number of Cores per Socket:  12

Execution start time:  Wed Nov 20 15:39:32 2013

System name and speed:  mom2 2701 MHz
## Sampling Output (Table 2)

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Function</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Source</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Line</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PE=HIDE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>100.0%</th>
<th>7607.1</th>
<th>--</th>
<th>--</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>67.6%</td>
<td>5139.8</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td>67.5%</td>
<td>5136.8</td>
<td>--</td>
<td>--</td>
<td>cfd_</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>training/201312-CSE-EPCC/reggrid/cfd.f</td>
</tr>
</tbody>
</table>

| 4       | 1.1%   | 85.7| 31.3| 27.6% | line.202 |
| 4       | 25.0%  | 1905.1| 319.9| 14.8% | line.204 |
| 4       | 12.4%  | 943.9| 329.1| 26.7% | line.206 |
| 4       | 23.5%  | 1785.5| 402.5| 19.0% | line.216 |
| 4       | 4.3%   | 324.9| 134.1| 30.2% | line.218 |

<table>
<thead>
<tr>
<th>31.8%</th>
<th>2421.7</th>
<th>--</th>
<th>--</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.7%</td>
<td>1038.5</td>
<td>315.5</td>
<td>24.1%</td>
<td>MPI_SSEND</td>
</tr>
<tr>
<td>7.2%</td>
<td>547.1</td>
<td>3554.9</td>
<td>89.5%</td>
<td>mpi_recv</td>
</tr>
<tr>
<td>7.1%</td>
<td>540.4</td>
<td>3559.6</td>
<td>89.6%</td>
<td>MPI_WAIT</td>
</tr>
<tr>
<td>3.8%</td>
<td>290.8</td>
<td>319.2</td>
<td>54.0%</td>
<td>mpi_finalize</td>
</tr>
</tbody>
</table>

---

[Sample output from Archer supercomputer at EPCC]
## pat_report: Flat Profile

### Table 1: Profile by Function

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb. Samp</th>
<th>Imb. Samp%</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>7607.1</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td>67.6%</td>
<td>5139.8</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td>67.5%</td>
<td>5136.8</td>
<td>1076.2</td>
<td>17.9%</td>
<td>cfd_</td>
</tr>
</tbody>
</table>

### Observations and suggestions

**MPI Grid Detection:**

A linear pattern was detected in MPI sent message traffic.
For table of sent message counts, use -O mpi_dest_counts.
For table of sent message bytes, use -O mpi_dest_bytes.
<table>
<thead>
<tr>
<th>Counter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perf. Count HW Cache L1D: Access</td>
<td>99236829284</td>
</tr>
<tr>
<td>Perf. Count HW Cache L1D: Prefetch</td>
<td>1395603690</td>
</tr>
<tr>
<td>Perf. Count HW Cache L1D: Miss</td>
<td>5235958322</td>
</tr>
<tr>
<td>CPU_CLK_Unhalted:Thread_P</td>
<td>229602167200</td>
</tr>
<tr>
<td>CPU_CLK_Unhalted:Ref_P</td>
<td>7533538184</td>
</tr>
<tr>
<td>DTLB_Load_Misses:Miss_Causes_A_Walk</td>
<td>29102852</td>
</tr>
<tr>
<td>DTLB_Store_Misses:Miss_Causes_A_Walk</td>
<td>6702254</td>
</tr>
<tr>
<td>L2_Rqsts:All_Demand_Data_RD</td>
<td>3448321934</td>
</tr>
<tr>
<td>L2_Rqsts:Demand_Data_RD_Hit</td>
<td>3019403605</td>
</tr>
<tr>
<td>User time (approx)</td>
<td>76.128 secs</td>
</tr>
<tr>
<td>User cycles</td>
<td>205620987829</td>
</tr>
<tr>
<td>CPU_CLK</td>
<td>3.048GHz</td>
</tr>
<tr>
<td>TLB utilization</td>
<td>2956.80 refs/miss</td>
</tr>
<tr>
<td>D1 cache hit, miss ratios</td>
<td>95.1% hits, 4.9% misses</td>
</tr>
<tr>
<td>D1 cache utilization (misses)</td>
<td>20.22 refs/miss, 2.527 avg hits</td>
</tr>
<tr>
<td>D2 cache hit, miss ratio</td>
<td>91.8% hits, 8.2% misses</td>
</tr>
<tr>
<td>D1+D2 cache hit, miss ratio</td>
<td>99.6% hits, 0.4% misses</td>
</tr>
<tr>
<td>D1+D2 cache utilization</td>
<td>246.83 refs/miss, 30.853 avg hits</td>
</tr>
<tr>
<td>D2 to D1 bandwidth</td>
<td>2764.681MB/sec</td>
</tr>
<tr>
<td>D2 to D1 bandwidth</td>
<td>220692603786 bytes</td>
</tr>
</tbody>
</table>
Some important options to pat_report -O

callers	Profile by Function and Callers
callers+hwpc	Profile by Function and Callers
callers+src	Profile by Function and Callers, with Line Numbers
callers+src+hwpc	Profile by Function and Callers, with Line Numbers
calltree	Function Calltree View
heap_hiwater	Heap Stats during Main Program
hwpc	Program HW Performance Counter Data
load_balance_program+hwpc	Load Balance across PEs
load_balance_sm	Load Balance with MPI Sent Message Stats
loop_times	Loop Stats by Function (from -hprofile_generate)
loops	Loop Stats by Inclusive Time (from -hprofile_generate)
mpi_callers	MPI Message Stats by Caller
profile	Profile by Function Group and Function
profile+src+hwpc	Profile by Group, Function, and Line
samp_profile	Profile by Function
samp_profile+hwpc	Profile by Function
samp_profile+src	Profile by Group, Function, and Line

For a full list see pat_report -O help
Debugging practical

- Try out DDT and/or STAT