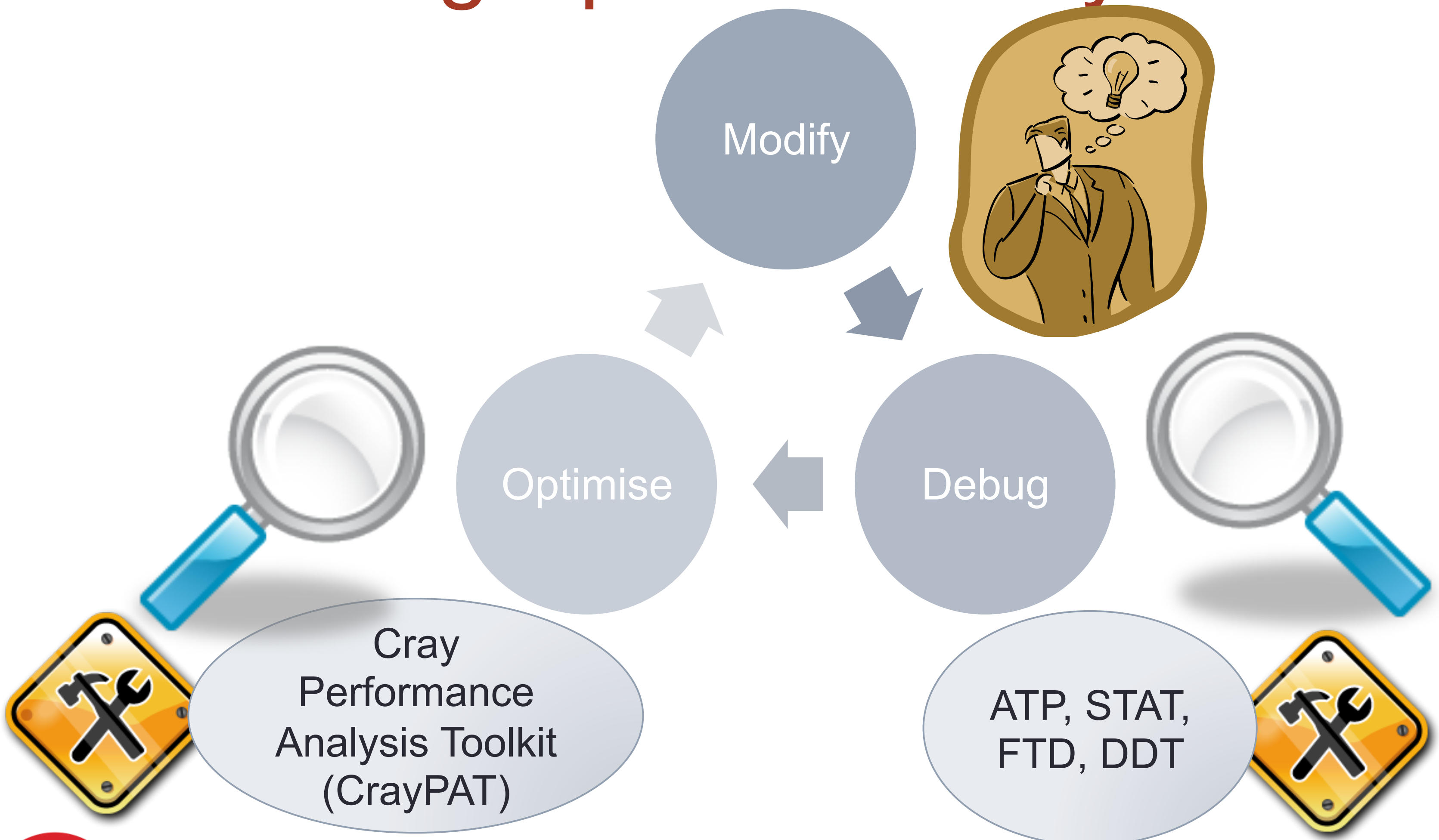
ARCHER Performance and Debugging Tools

Slides contributed by Cray and EPCC

The Porting/Optimisation Cycle



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
```

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.

More information (while atp module loaded)

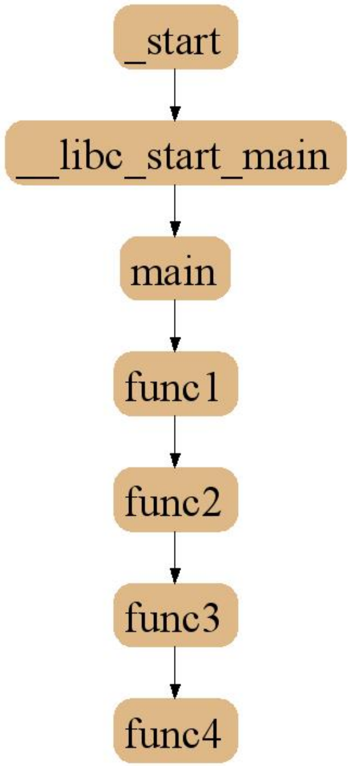
```
man atp
```

Stack Trace Analysis Tool (STAT)

For when nothing appears to be happening…

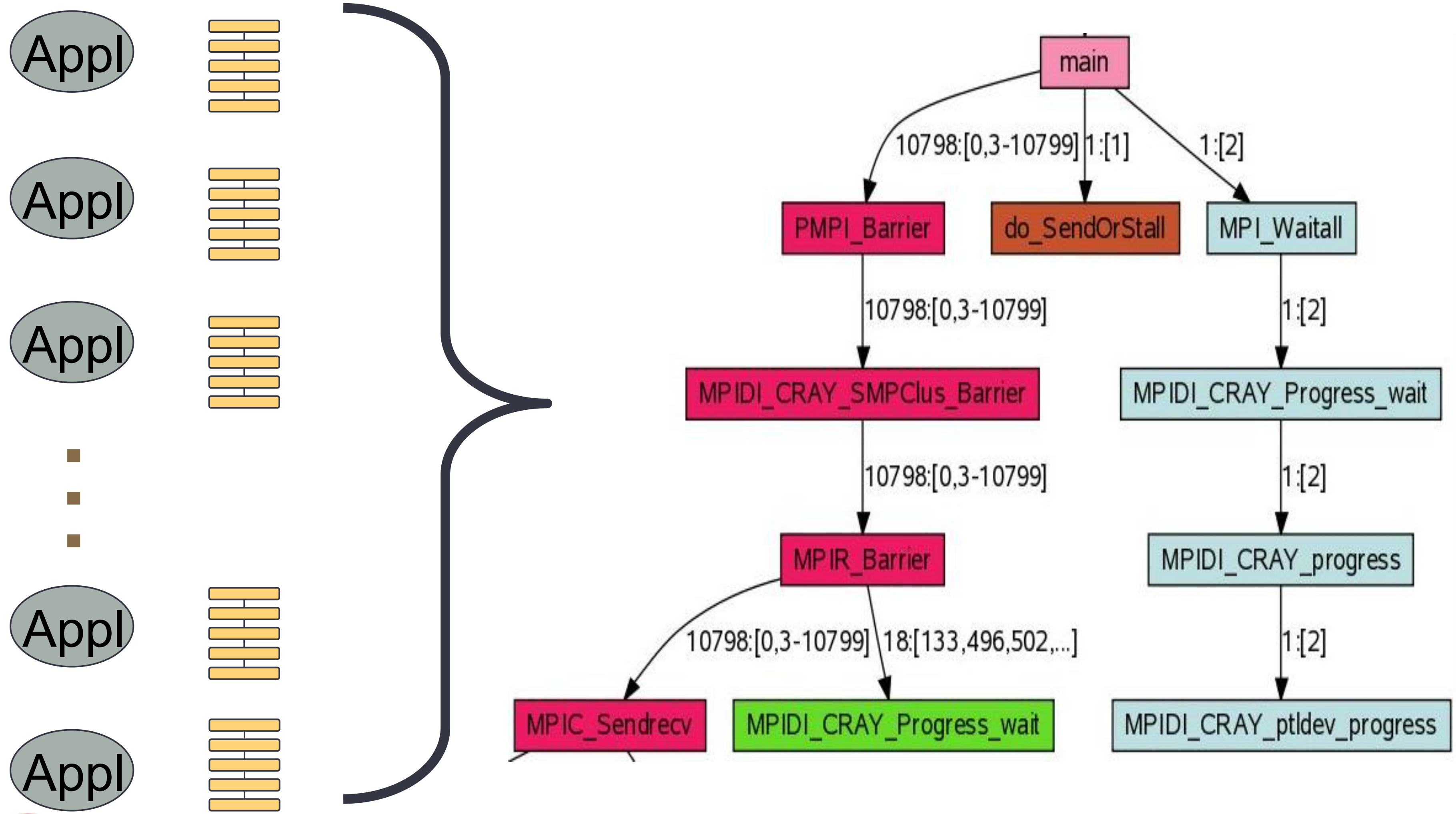
STAT

- 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



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 - Command line debugging

- 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

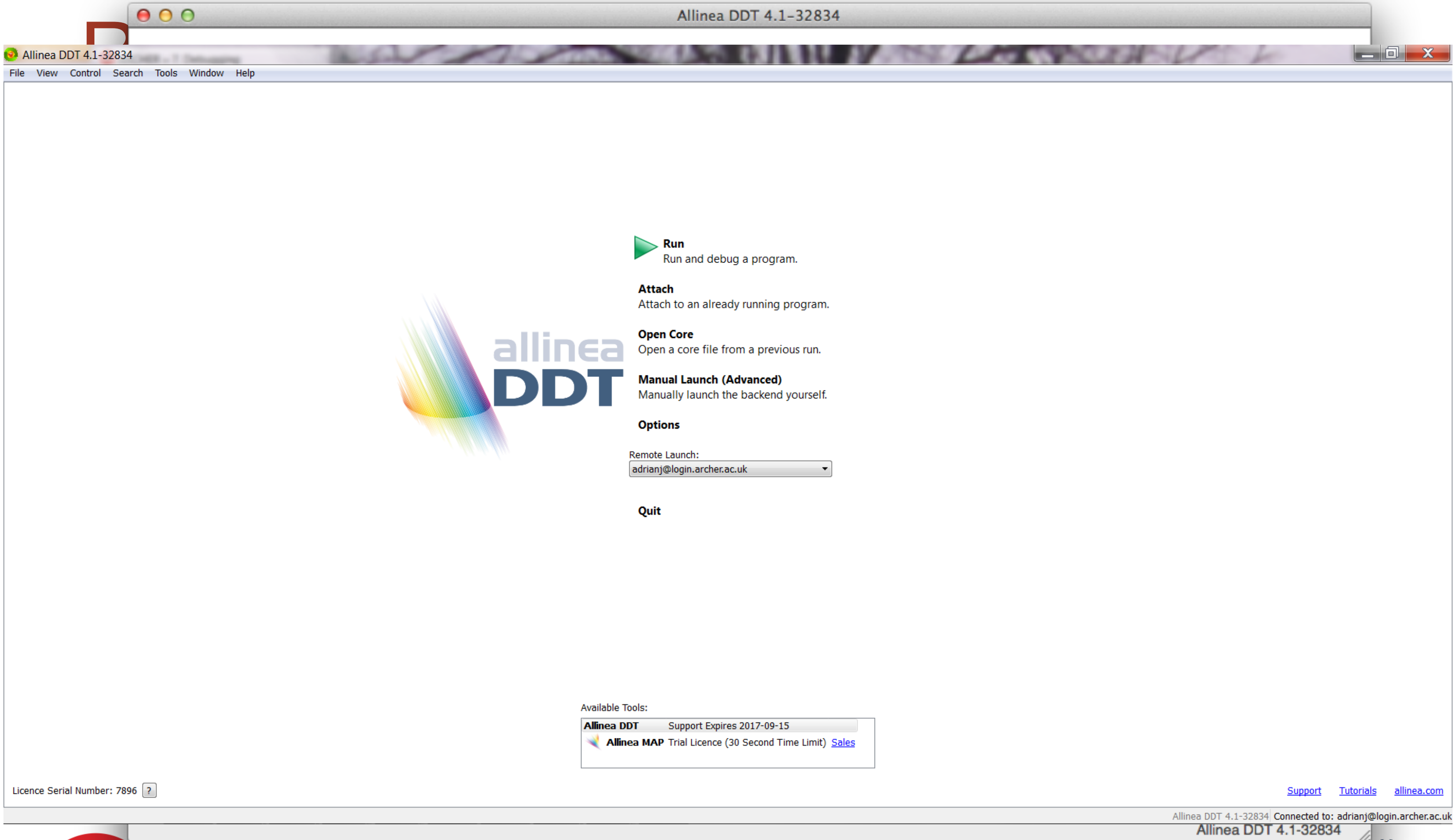
- Allinea DDT installed on ARCHER
  - TotalView no longer available

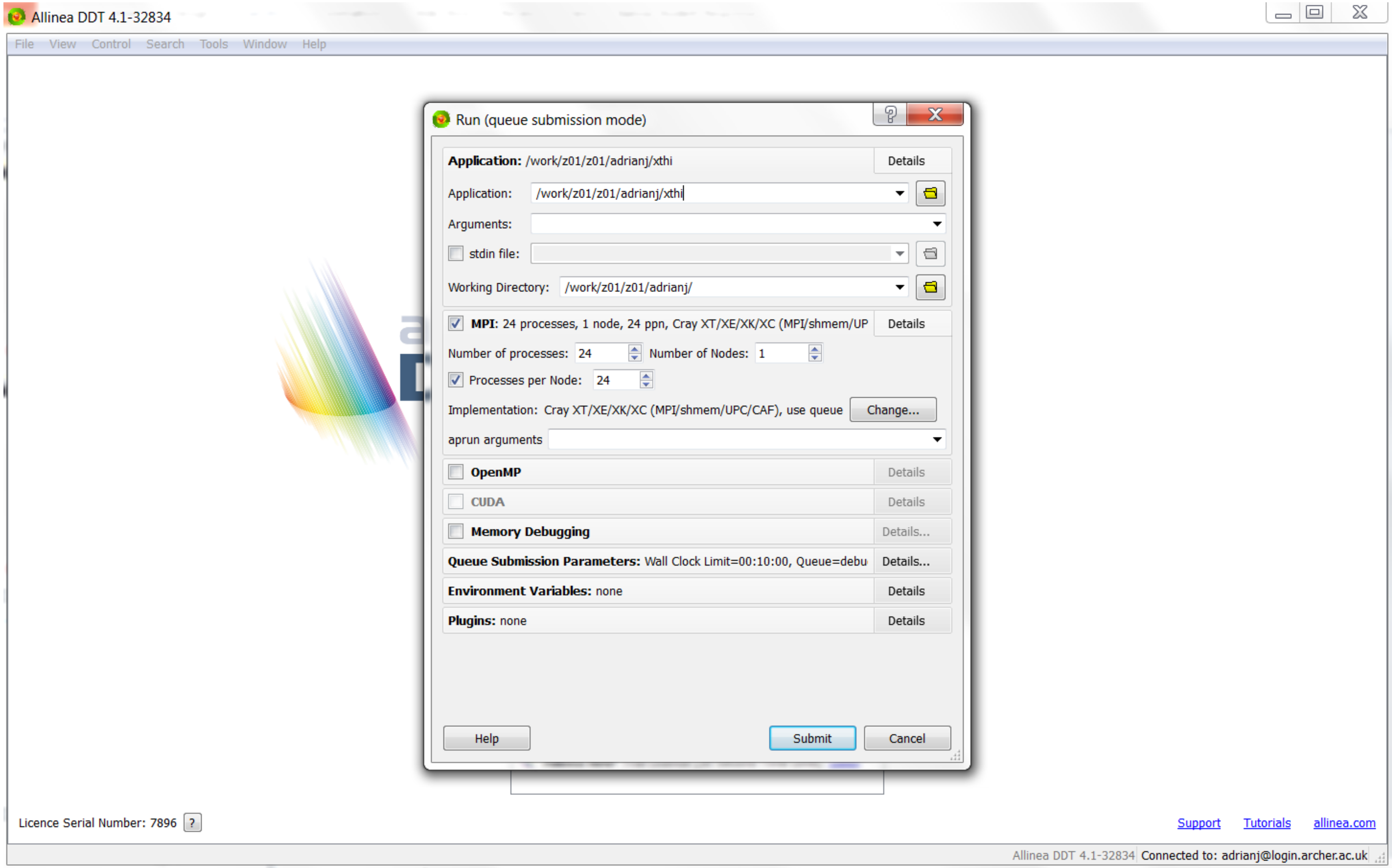
- The recommended way to use DDT on ARCHER is to install the free DDT remote client on your workstation or laptop and use this to run DDT on ARCHER.

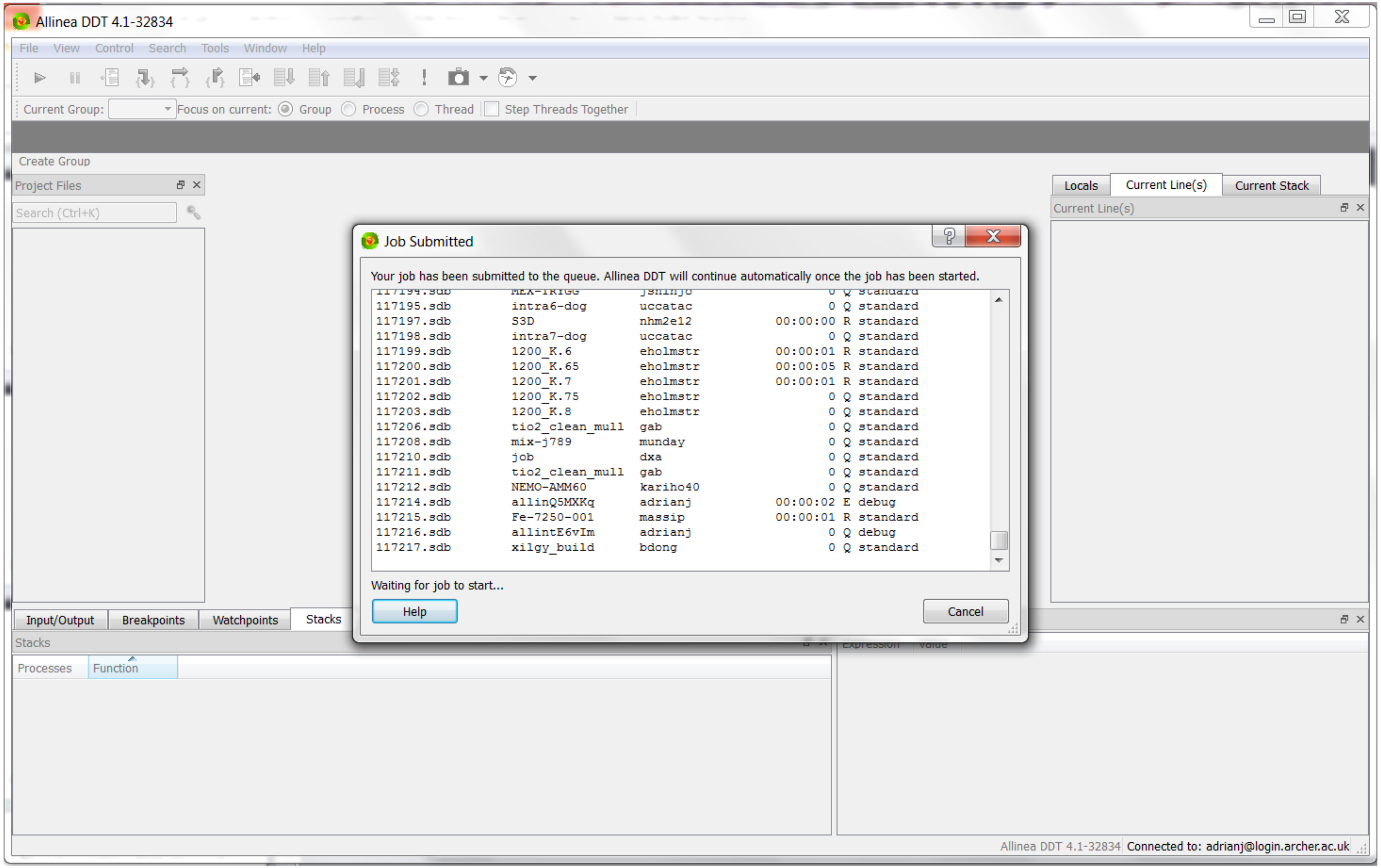
- The version of the DDT remote client must match the version of DDT installed on ARCHER
  - **currently version 4.1**
  - http://www.allinea.com/products/downloads/clients

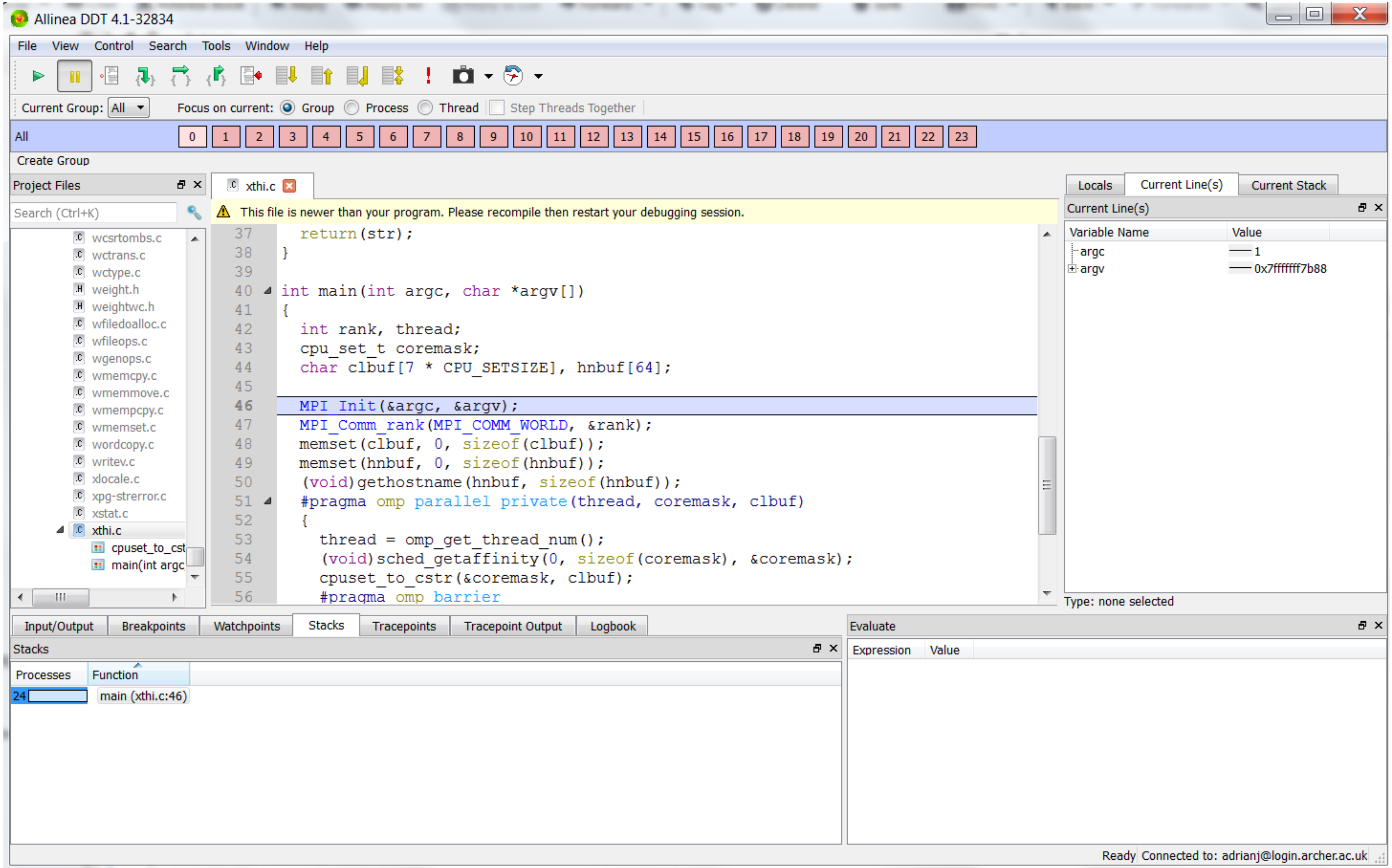
Compiling for debugging

- 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


Allinea DDT 4.1-32834
File View Control Search Tools Window Help
Run
Run and debug a program.
Attach
Attach to an already running program.
Open Core
Open a core file from a previous run.
Manual Launch (Advanced)
Manually launch the backend yourself.
Options
Remote Launch:
adrianj@login.archer.ac.uk
Quit
allinea
DDT
Available Tools:
Allinea DDT Support Expires 2017-09-15
Allinea MAP Trial Licence (30 Second Time Limit) Sales
Licence Serial Number: 7896
Support Tutorials allinea.com
Allinea DDT 4.1-32834 Connected to: adrianj@login.archer.ac.uk


Allinea DDT 4.1-32834
File View Control Search Tools Window Help
Run (queue submission mode)
Application: /work/z01/z01/adrianj/xthi
Details
Application: /work/z01/z01/adrianj/xthi
Arguments:
stdin file:
Working Directory: /work/z01/z01/adrianj/
MPI: 24 processes, 1 node, 24 ppn, Cray XT/XE/XK/XC (MPI/shmem/UP
Details
Number of processes: 24
Number of Nodes: 1
Processes per Node: 24
Implementation: Cray XT/XE/XK/XC (MPI/shmem/UPC/CAF), use queue
Change...
aprun arguments
OpenMP
Details
CUDA
Details
Memory Debugging
Details...
Queue Submission Parameters: Wall Clock Limit=00:10:00, Queue=debu
Details...
Environment Variables: none
Details
Plugins: none
Details
Help
Submit
Cancel
Licence Serial Number: 7896
Support
Tutorials
allinea.com
Allinea DDT 4.1-32834
Connected to: adrianj@login.archer.ac.uk


Allinea DDT 4.1-32834
File View Control Search Tools Window Help
Current Group: Focus on current: Group Process Thread Step Threads Together
Create Group
Project Files
Search (Ctrl+K)
Locals Current Line(s) Current Stack
Current Line(s)
Job Submitted
Your job has been submitted to the queue. Allinea DDT will continue automatically once the job has been started.
117195.sdb intra6-dog uccatac 0 Q standard
117197.sdb S3D nhm2e12 00:00:00 R standard
117198.sdb intra7-dog uccatac 0 Q standard
117199.sdb 1200_K.6 eholmstr 00:00:01 R standard
117200.sdb 1200_K.65 eholmstr 00:00:05 R standard
117201.sdb 1200_K.7 eholmstr 00:00:01 R standard
117202.sdb 1200_K.75 eholmstr 0 Q standard
117203.sdb 1200_K.8 eholmstr 0 Q standard
117206.sdb tio2_clean_mull gab 0 Q standard
117208.sdb mix-j789 munday 0 Q standard
117210.sdb job dxa 0 Q standard
117211.sdb tio2_clean_mull gab 0 Q standard
117212.sdb NEMO-AMM60 kariho40 0 Q standard
117214.sdb allinQ5MXKq adrianj 00:00:02 E debug
117215.sdb Fe-7250-001 massip 00:00:01 R standard
117216.sdb allintE6vIm adrianj 0 Q debug
117217.sdb xilgy_build bdong 0 Q standard
Waiting for job to start...
Help
Cancel
Input/Output Breakpoints Watchpoints Stacks
Stacks
Processes Function
Allinea DDT 4.1-32834 Connected to: adrianj@login.archer.ac.uk



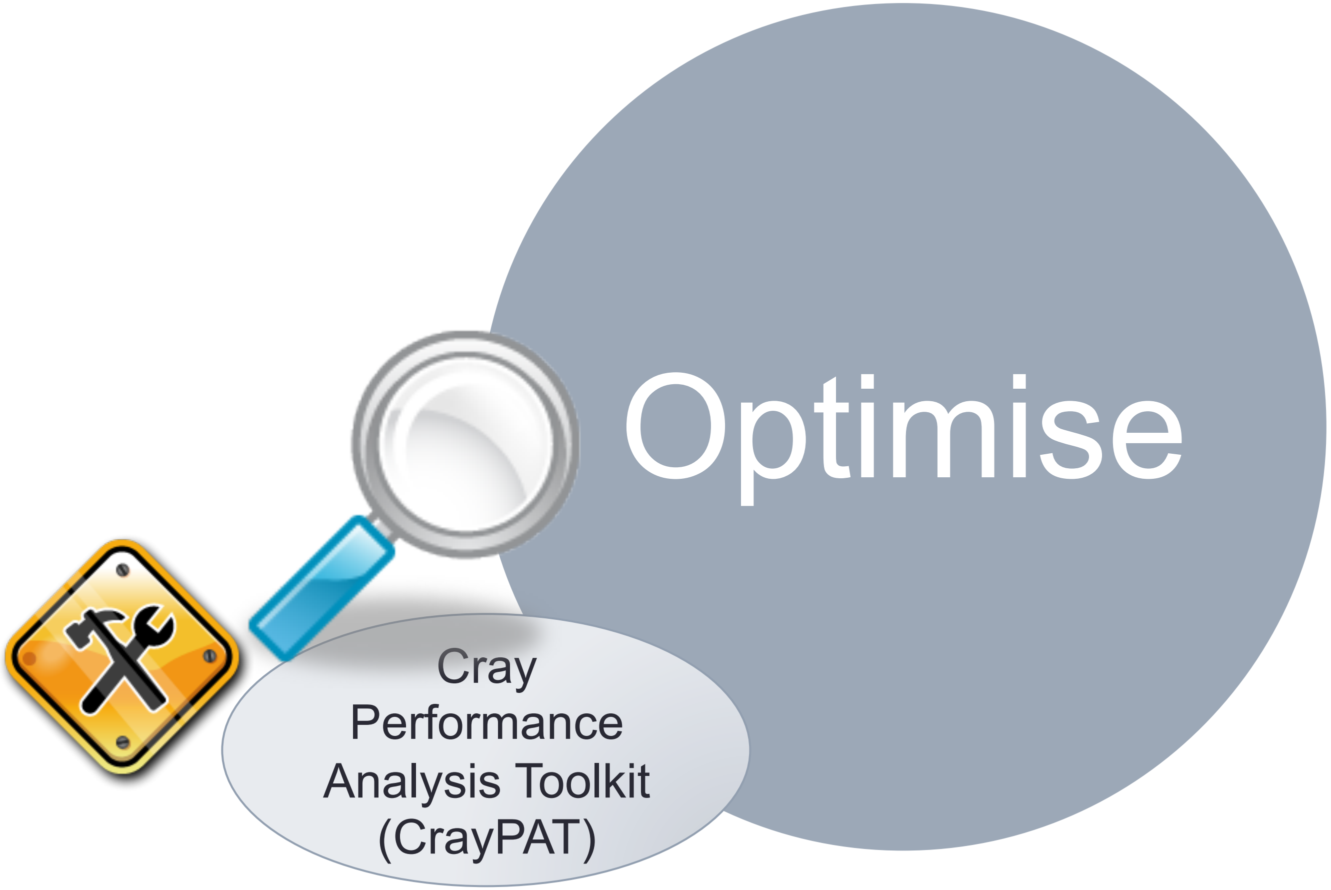
archer

epcc

THE UNIVERSITY OF EDINBURGH

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

```
% module load perftools
```

- Build application keeping .o files (CCE: `-h keepfiles`)

```
% make clean
% make
```

- Instrument application for automatic profiling analysis
  - You should get an instrumented program `a.out+pat`

```
% pat_build -O apa a.out
```

We are telling `pat_build` that the output of this sample run will be used in an APA run

- Run application to get top time consuming routines
  - You should get a performance file (“`<sdatafile>.xf`”) or multiple files in a directory `<sdatadir>`

```
% aprun … a.out+pat  (or qsub <pat script>)
```

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 <apafile>.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 Apprentice^2

```
% app2 <datafile>.ap2
```

Analysing Data with `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 Apprentice[2]
  - `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)

```
 Samp%  |  Samp  |  Imb.  |  Imb.  |Group
        |        |  Samp  | Samp%  | Function
        |        |        |        |  Source
        |        |        |        |   Line
        |        |        |        |    PE=HIDE

 100.0% | 7607.1 |     -- |     -- |Total
|-----------------------------------------------------------------------
|  67.6% | 5139.8 |     -- |     -- |USER
||----------------------------------------------------------------------
|  67.5% | 5136.8 |     -- |     -- | cfd_
3        |        |        |        |  training/201312-CSE-EPCC/reggrid/cfd.f
||||--------------------------------------------------------------------
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
||||====================================================================
||======================================================================
|  31.8% | 2421.7 |     -- |     -- |MPI
||----------------------------------------------------------------------
||  13.7% | 1038.5 |  315.5 |  24.1% |MPI_SSEND
||   7.2% |  547.1 | 3554.9 |  89.5% |mpi_recv
||   7.1% |  540.4 | 3559.6 |  89.6% |MPI_WAIT
||   3.8% |  290.8 |  319.2 |  54.0% |mpi_finalize
|=======================================================================
```

archer

epcc

THE UNIVERSITY OF EDINBURGH

pat_report: Flat Profile

```
Table 1:  Profile by Function

 Samp%   |   Samp   |  Imb.   |  Imb.   |Group
         |          |  Samp   | Samp%   | Function
         |          |         |         |  PE=HIDE

 100.0% | 7607.1 |     -- |     -- |Total
|-----------------------------------------------
|  67.6% | 5139.8 |     -- |     -- |USER
||----------------------------------------------
|  67.5% | 5136.8 | 1076.2 |  17.9% | cfd_
||==============================================
|  31.8% | 2421.7 |     -- |     -- |MPI
||----------------------------------------------
||   13.7% | 1038.5 |  315.5 |  24.1% |MPI_SSEND
||    7.2% |  547.1 | 3554.9 |  89.5% |mpi_recv
||    7.1% |  540.4 | 3559.6 |  89.6% |MPI_WAIT
||    3.8% |  290.8 |  319.2 |  54.0% |mpi_finalize
|===============================================
================  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.
```

pat_report: Hardware Performance Counters

```
================================================================
  Total
----------------------------------------------------------------
  PERF_COUNT_HW_CACHE_L1D:ACCESS                  99236829284
  PERF_COUNT_HW_CACHE_L1D:PREFETCH                 1395603690
  PERF_COUNT_HW_CACHE_L1D:MISS                     5235958322
  CPU_CLK_UNHALTED:THREAD_P                      229602167200
  CPU_CLK_UNHALTED:REF_P                           7533538184
  DTLB_LOAD_MISSES:MISS_CAUSES_A_WALK                29102852
  DTLB_STORE_MISSES:MISS_CAUSES_A_WALK                6702254
  L2_RQSTS:ALL_DEMAND_DATA_RD                      3448321934
  L2_RQSTS:DEMAND_DATA_RD_HIT                      3019403605
  User time (approx)             76.128 secs   205620987829 cycles
  CPU_CLK                         3.048GHz
  TLB utilization              2956.80 refs/miss     5.775 avg uses
  D1 cache hit,miss ratios        95.1% hits          4.9% misses
  D1 cache utilization (misses)   20.22 refs/miss     2.527 avg hits
  D2 cache hit,miss ratio         91.8% hits          8.2% misses
  D1+D2 cache hit,miss ratio      99.6% hits          0.4% misses
  D1+D2 cache utilization        246.83 refs/miss    30.853 avg hits
  D2 to D1 bandwidth           2764.681MB/sec  220692603786 bytes
```

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
- http://tinyurl.com/archer100214/DebuggingHandout.pdf