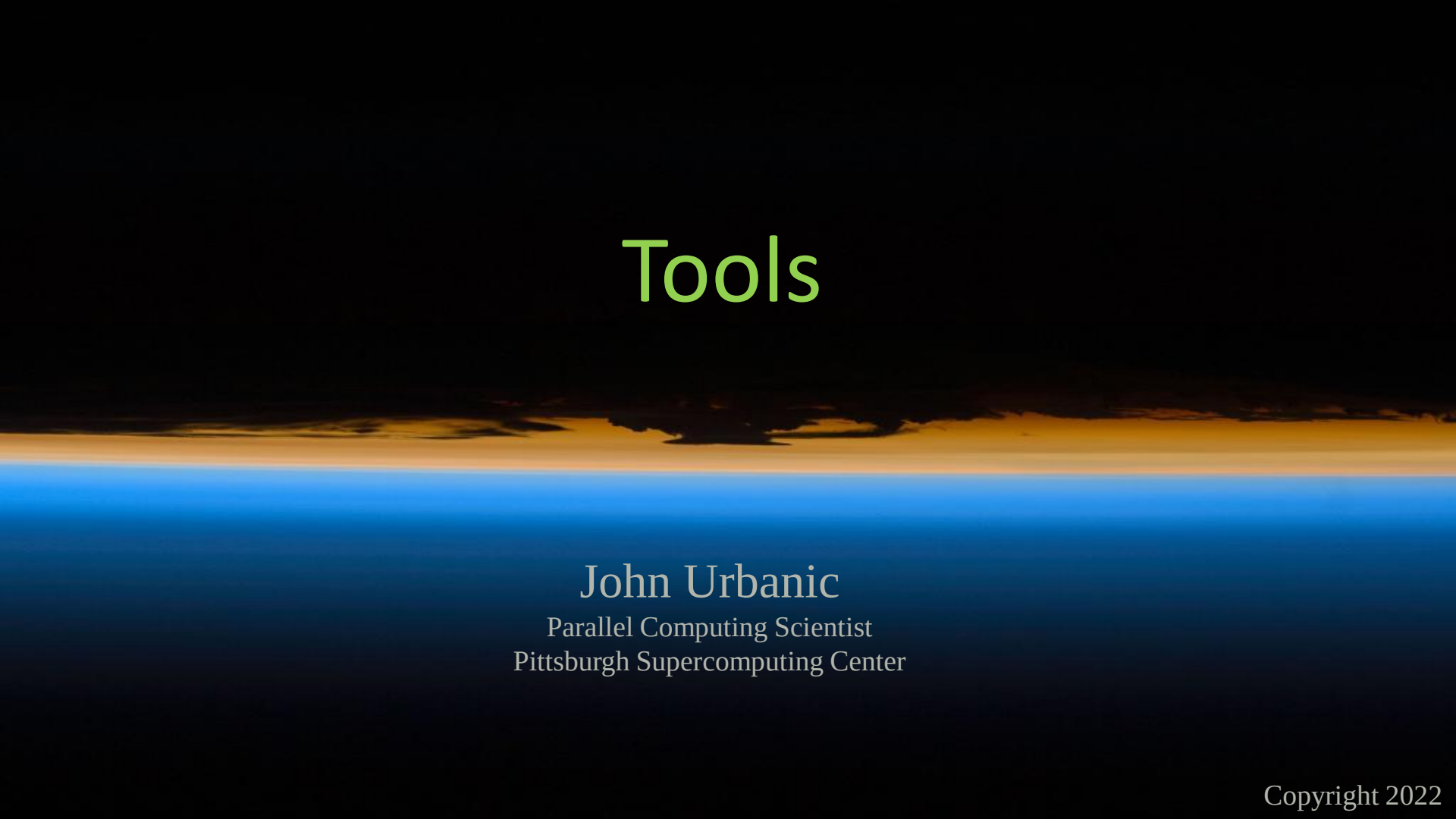


Tools

John Urbanic
Parallel Computing Scientist
Pittsburgh Supercomputing Center

Tools In Parallel Programming

Given that the philosophy of these workshops is to teach portable programming techniques and avoid platform or version specific information, I have always reluctantly avoided talking about tools. The best ones are licensed, and in parallel programming there are no open source debuggers worth speaking of.

However, we always get feedback requesting this, and there are two types of tools that are wildly useful in this field:

- Debuggers
- Profilers

So we will carve out a few minutes to expose you to two of the best examples.

Debuggers

A good debugger is an invaluable time saver. It is shameful how under-exploited they are in general serial programming. In parallel programming this is no less true. However there is one excuse: there are only two decent parallel debuggers (Totalview and DDT) and they are both proprietary.

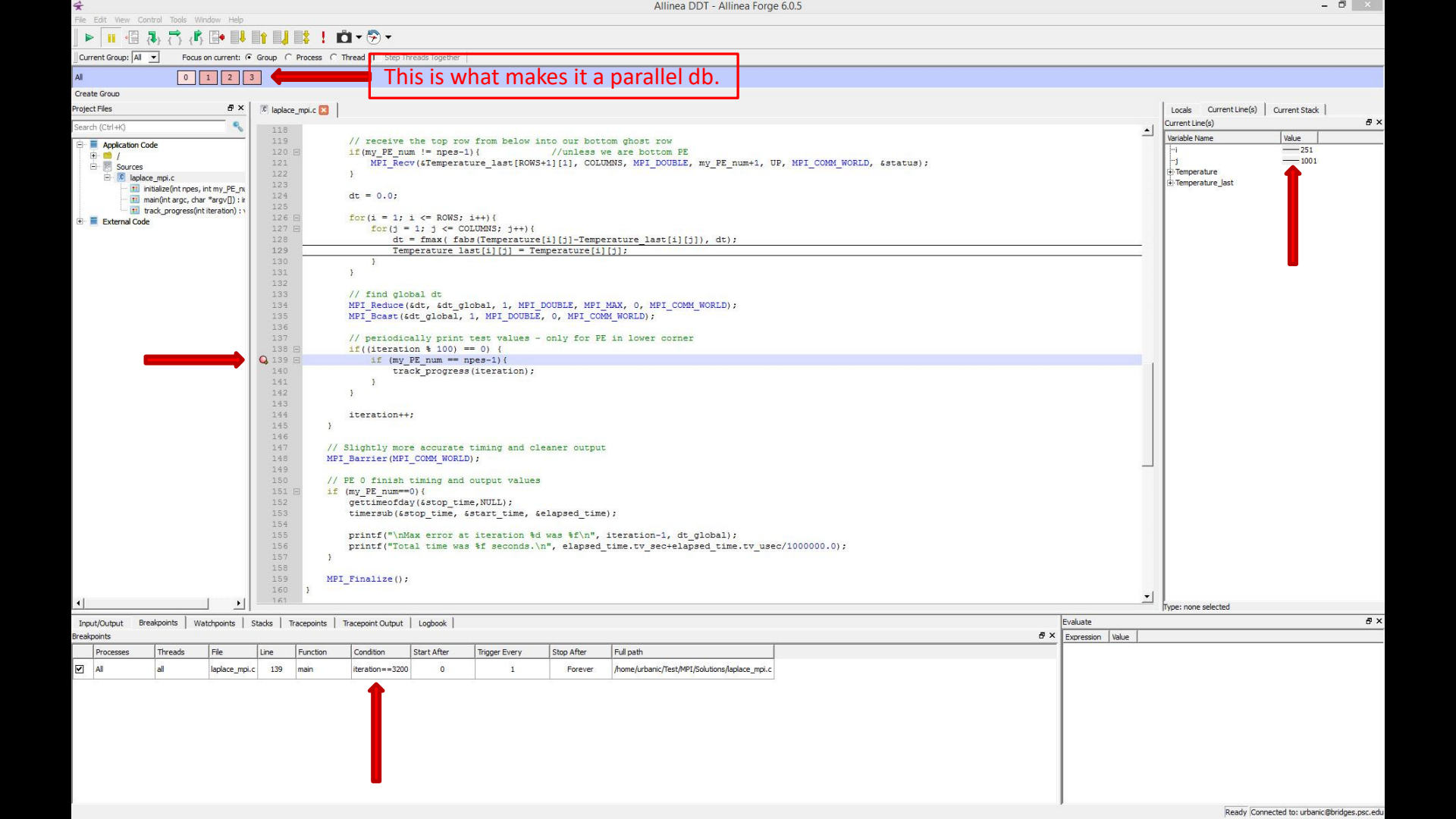
They are very easy to use

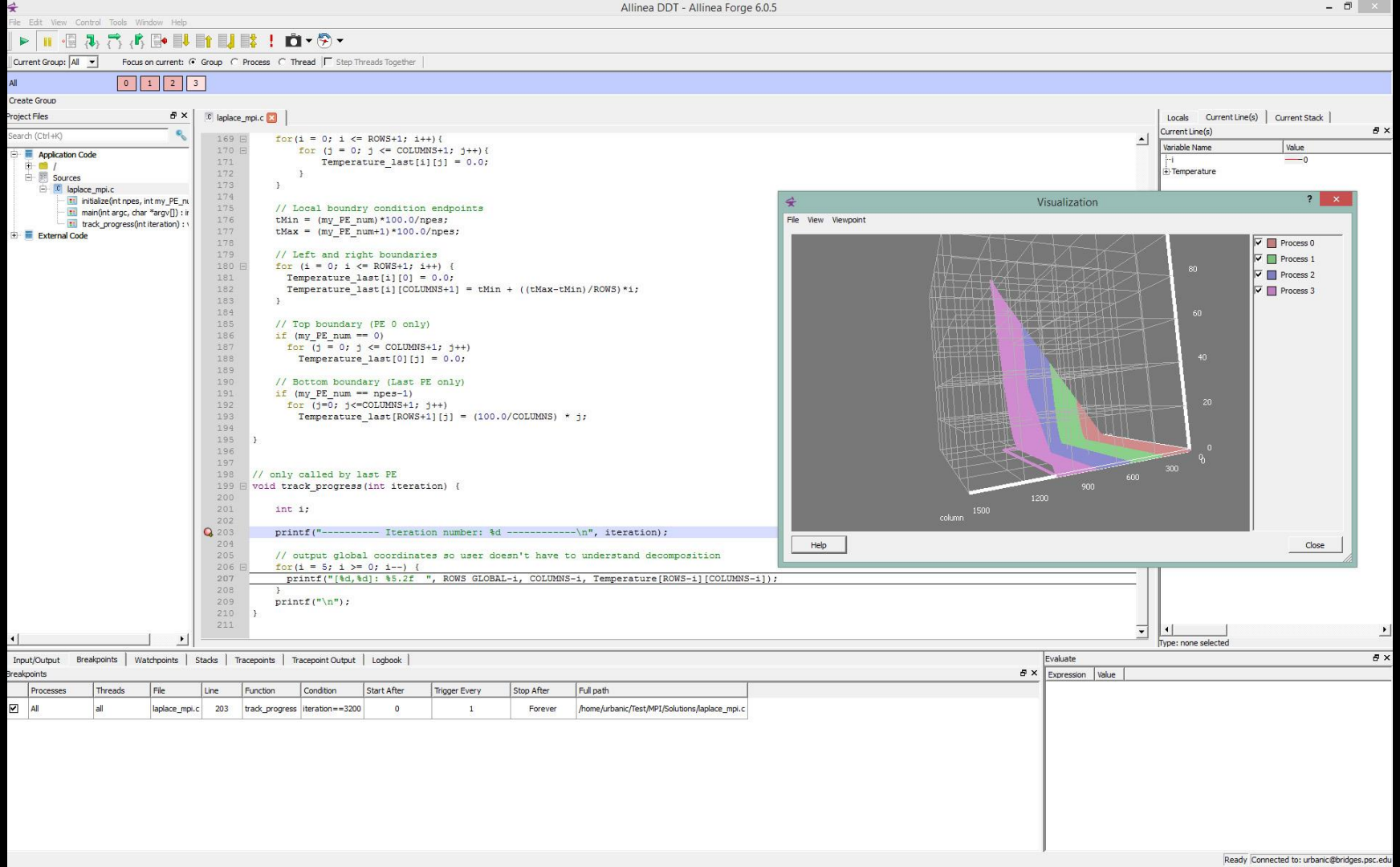
Just compile with the –g option and your executable will include everything needed to interact with the debugger at runtime. Note that this does impact performance substantially, so you wouldn't want to do this all the time.

DDT allows us to skip the X Windows complication

Most unix-land tools with nice GUIs require an X server driven display. DDT now has a desktop client that avoids what have been tricky configuration and maddening performance issues.

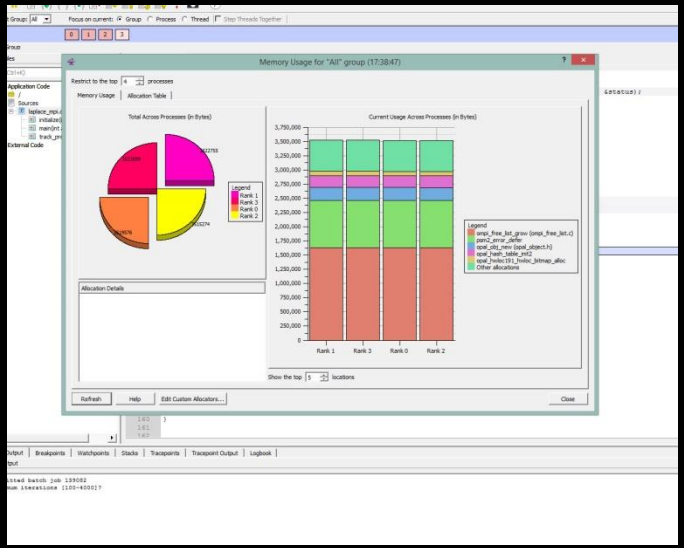
Note that remote usage, especially with a queuing system such as Slurm, requires the debugger to negotiate details with the remote platform (Bridges). It will do all of this automatically for you *once you configure the system the first time*. Our web pages will explicitly step you through this process.


Allinea DDT - Allinea Forge 6.0.5
This is what makes it a parallel db.
laplace_mpi.c
Project Files
Application Code
Sources
External Code
Locals
Current Line(s)
Current Stack
Variable Name
Value
i
251
j
1001
Temperature
Temperature_last
Breakpoints
Processes
Threads
File
Line
Function
Condition
Start After
Trigger Every
Stop After
Full path
All
all
laplace_mpi.c
139
main
iteration==3200
0
1
Forever
/home/urbanic/Test/MPI/Solutions/laplace_mpi.c
Evaluate
Expression
Value


Allinea DDT - Allinea Forge 6.0.5
File Edit View Control Tools Window Help
Current Group: All
Focus on current: Group Process Thread Step Threads Together
All 0 1 2 3
Create Group
Project Files
Search (Ctrl+K)
Application Code
/
Sources
laplace_mpi.c
initialize(int npes, int my_PE_n
main(int argc, char *argv[]) : i
track_progress(int iteration) :
External Code
laplace_mpi.c
169 for(i = 0; i <= ROWS+1; i++){
170 for (j = 0; j <= COLUMNS+1; j++){
171 Temperature_last[i][j] = 0.0;
172 }
173 }
174
175 // Local boundry condition endpoints
176 tMin = (my_PE_num)*100.0/npes;
177 tMax = (my_PE_num+1)*100.0/npes;
178
179 // Left and right boundaries
180 for (i = 0; i <= ROWS+1; i++) {
181 Temperature_last[i][0] = 0.0;
182 Temperature_last[i][COLUMNS+1] = tMin + ((tMax-tMin)/ROWS)*i;
183 }
184
185 // Top boundary (PE 0 only)
186 if (my_PE_num == 0)
187 for (j = 0; j <= COLUMNS+1; j++)
188 Temperature_last[0][j] = 0.0;
189
190 // Bottom boundary (Last PE only)
191 if (my_PE_num == npes-1)
192 for (j=0; j<=COLUMNS+1; j++)
193 Temperature_last[ROWS+1][j] = (100.0/COLUMNS) * j;
194
195 }
196
197
198 // only called by last PE
199 void track_progress(int iteration) {
200
201 int i;
202
203 printf("---------- Iteration number: %d ----------\n", iteration);
204
205 // output global coordinates so user doesn't have to understand decomposition
206 for(i = 5; i >= 0; i--) {
207 printf("[%d,%d]: %5.2f ", ROWS_GLOBAL-i, COLUMNS-i, Temperature[ROWS-i][COLUMNS-i]);
208 }
209 printf("\n");
210 }
211
Locals Current Line(s) Current Stack
Current Line(s)
Variable Name Value
i 0
Temperature
Visualization
File View Viewpoint
Process 0
Process 1
Process 2
Process 3
80
60
40
20
0
0
300
600
900
1200
1500
column
Help
Close
Type: none selected
Input/Output Breakpoints Watchpoints Stacks Tracepoints Tracepoint Output Logbook
Breakpoints
Processes Threads File Line Function Condition Start After Trigger Every Stop After Full path
All all laplace_mpi.c 203 track_progress iteration==3200 0 1 Forever /home/urbanic/Test/MPI/Solutions/laplace_mpi.c
Evaluate
Expression Value
Ready Connected to: urbanic@bridges.psc.edu

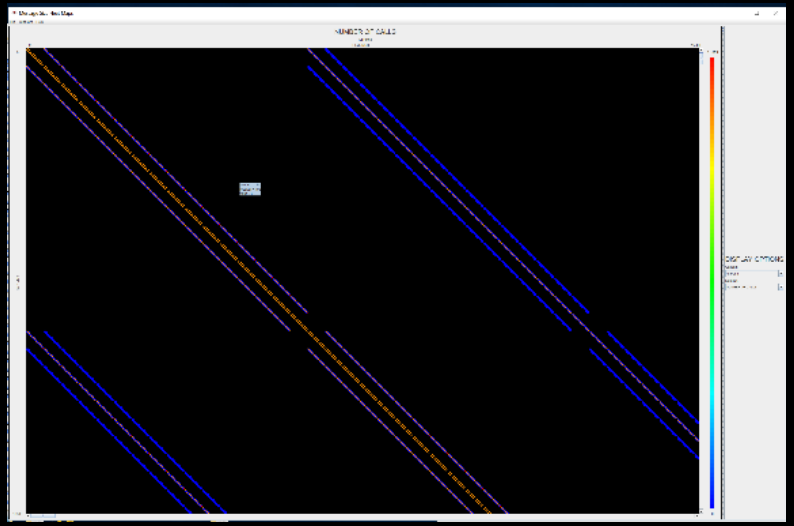
Other Nifty Features

- Can deal with very large numbers of PEs
  - Has ways of coalescing PEs into groups, etc.
  - Has ways of comparing and contrasting across PEs to find anomalies
- Can connect to running jobs
- Also great for OpenMP and CUDA/OpenACC
- Also good for memory debugging
- and MPI tracing

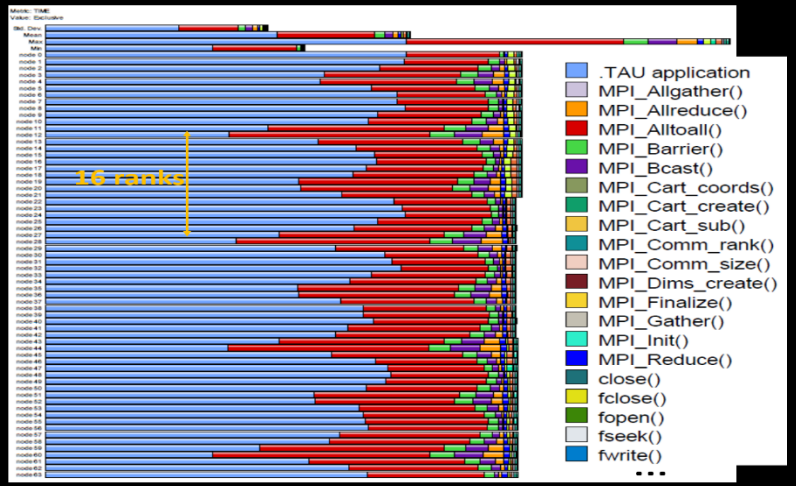


Profilers

Great Open Source Option: TAU

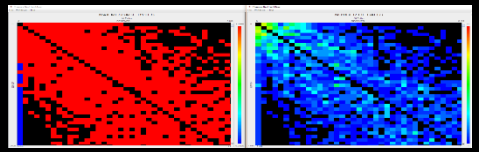


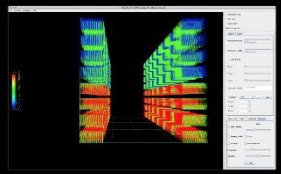
Nearest neighbor communications in 3D.



Comp/comm ratio and other hotspots.

Lots more...



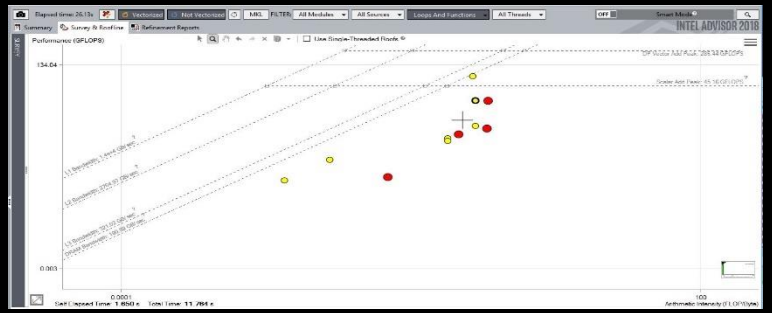


Profilers

Intel: Vtune, Advisor & Trace Analyzer

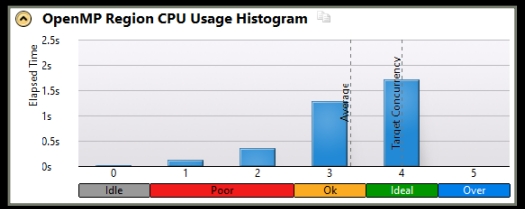
Message tracking a tricky All-To-All based code.

- Hotspot (Statistical call tree), Call counts
- Thread Profiling – Concurrency, Lock & Waits Analysis
- Cache miss, Bandwidth analysis…[1]
- GPU Offload and OpenCL™ Kernel Tracing
- View Results on the Source / Assembly
- OpenMP Scalability Analysis
- Visualize Thread & Task Activity on the Timeline

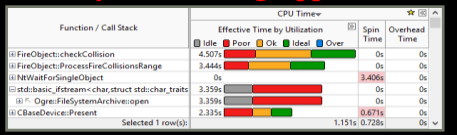


Roofline Analysis.

Tune OpenMP Scalability



Quickly Find Tuning Opportunities



Visualize & Filter Data