



# MPI Programming Tools

John Urbanic

Parallel Computing Scientist  
Pittsburgh Supercomputing Center

Distinguished Service Professor  
Carnegie Mellon University

# 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.



Current Group: All Focus on current: Group Process Thread Step threads together

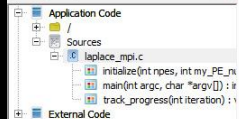
All 0 1 2 3

This is what makes it a parallel db.

Create Group

Project Files

Search (Ctrl+K)



laplace\_mpi.c

```
118
119 // receive the top row from below into our bottom ghost row
120 if(my_PE_num != npes-1){ //unless we are bottom PE
121     MPI_Recv(&Temperature_last[ROWS+1][1], COLUMNS, MPI_DOUBLE, my_PE_num+1, 0, MPI_COMM_WORLD, &status);
122 }
123
124 dt = 0.0;
125
126 for(i = 1; i <= ROWS; i++){
127     for(j = 1; j <= COLUMNS; j++){
128         dt = fmax(fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
129         Temperature_last[i][j] = Temperature[i][j];
130     }
131 }
132
133 // find global dt
134 MPI_Reduce(&dt, &dt_global, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
135 MPI_Bcast(&dt_global, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
136
137 // periodically print test values - only for PE in lower corner
138 if((iteration % 100) == 0){
139     if(my_PE_num == npes-1){
140         track_progress(iteration);
141     }
142 }
143
144 iteration++;
145
146
147 // Slightly more accurate timing and cleaner output
148 MPI_Barrier(MPI_COMM_WORLD);
149
150 // PE 0 finish timing and output values
151 if(my_PE_num==0){
152     gettimeofday(&stop_time,NULL);
153     timersub(&stop_time, &start_time, &elapsed_time);
154
155     printf("\nMax error at iteration %d was %f\n", iteration-1, dt_global);
156     printf("Total time was %f seconds.\n", elapsed_time.tv_sec+elapsed_time.tv_usec/1000000.0);
157 }
158
159 MPI_Finalize();
160
161 }
```

Locals Current Line(s) Current Stack

Current Line(s)

Variable Name	Value
i	251
j	1001
Temperature	
Temperature_last	

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	139	main	iteration==3200	0	1	Forever	/home/urbanic/Test/MPI/Solutions/laplace_mpi.c

Evaluate

Expression Value



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
- External Code

laplace\_mpi.c

```

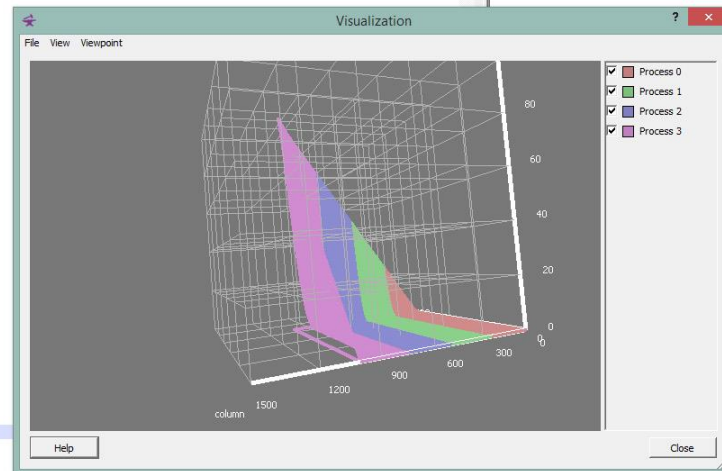
169 for (i = 0; i <= ROWS+1; i++){
170     for (j = 0; j <= COLUMNS+1; j++){
171         Temperature_last[i][j] = 0.0;
172     }
173 }
174
175 // Local boundary condition endpoints
176 tMin = (my_PE_num)*100.0/npes;
177 tMax = (my_PE_num+1)*100.0/npes;
178
179 // Left and right boundaries
180 for (i = 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 // only called by last PE
198 void track_progress(int iteration) {
199     int i;
200     printf("----- Iteration number: %d -----\\n", iteration);
201
202     // output global coordinates so user doesn't have to understand decomposition
203     for (i = 5; i >= 0; i--) {
204         printf("[%d,%d]: %5.2f ", ROWS GLOBAL-i, COLUMNS-i, Temperature[ROWS-i][COLUMNS-i]);
205     }
206     printf("\\n\\n");
207 }
208
209 }
210
211

```

Locals Current Line(s) Current Stack

Current Line(s)

Variable Name	Value
Temperature	0



Input/Output Breakpoints Watchpoints Stacks Tracepoints Tracepoint Output Logbook

Breakpoints

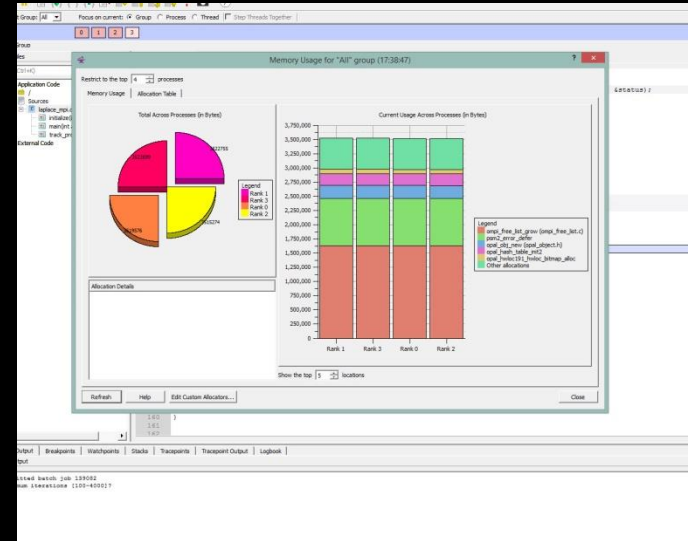
	Processes	Threads	File	Line	Function	Condition	Start After	Trigger Every	Stop After	Full path
<input checked="" type="checkbox"/>	All	all	laplace_mpi.c	203	track_progress	iteration==3200	0	1	Forever	/home/urbanic/Test/MPI/Solutions/laplace_mpi.c

Evaluate

Expression Value

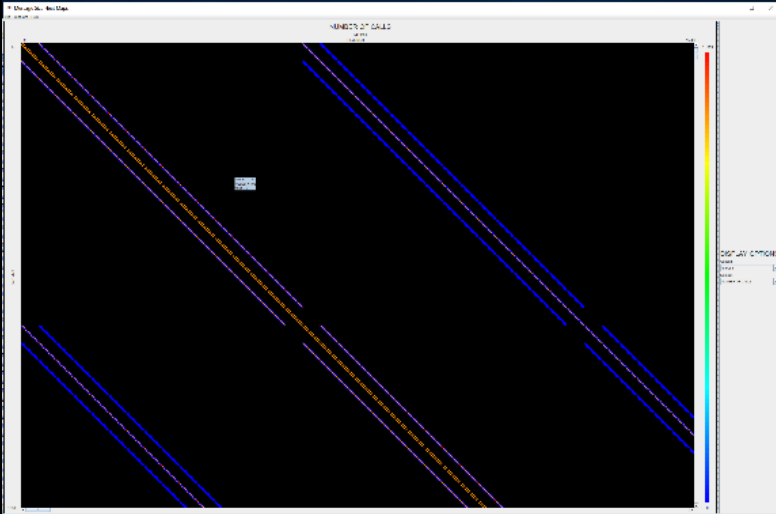
# Other Nifty Features

- 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

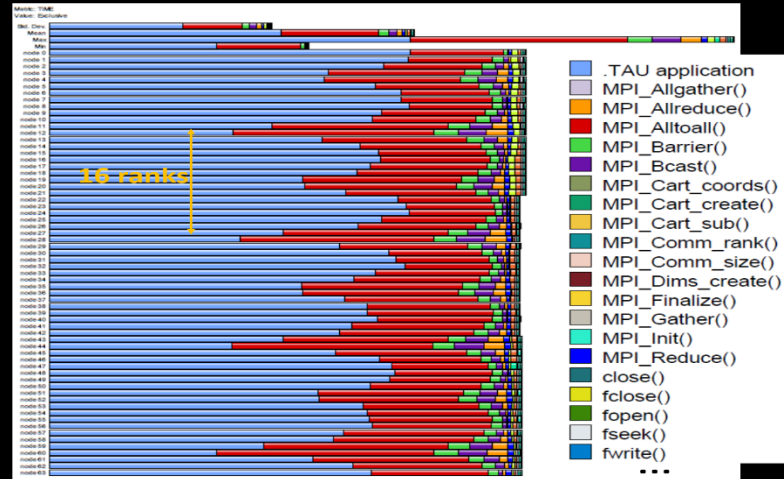


# Profilers

## Great Open Source Option: TAU

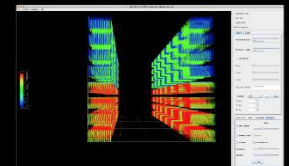
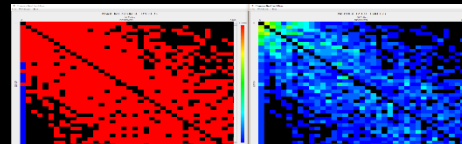


Nearest neighbor communications in 3D.



Comp/comm ratio and other hotspots.

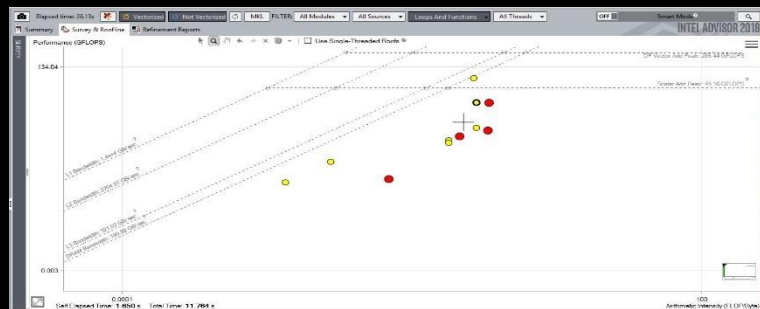
Lots more...



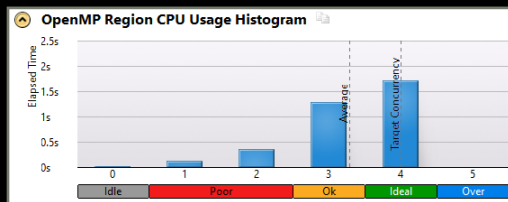
# Profilers

## Intel: Vtune, Advisor & Trace Analyzer

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



### Tune OpenMP Scalability



### Quickly Find Tuning Opportunities

Function / Call Stack	CPU Time			Spin Overhead Time	Spin Overhead Time
	Effective Time by Utilization	Idle	Poor	Ok	Ideal
0 FireObject::checkCollision	4.507s	Poor	Ok	Over	0s
0 FireObject::ProcessFireCollisionsRange	3.444s	Poor	Ok	Over	0s
0 NVWaitForSingleObject	0s	Ok	Ok	Ok	3.406s
0 std::basic_filestream::char_struct std::richer_traits	3.359s	Poor	Ok	Over	0s
0 CBaseDevice::Present	3.359s	Poor	Ok	Over	0s
0 CBaseDevice::Present	2.335s	Poor	Ok	Over	0.671s
Selected 1 row(s)					1.151s 0.728s

### Visualize & Filter Data

