Lab 5 TDDC78 – Tools & Debugging

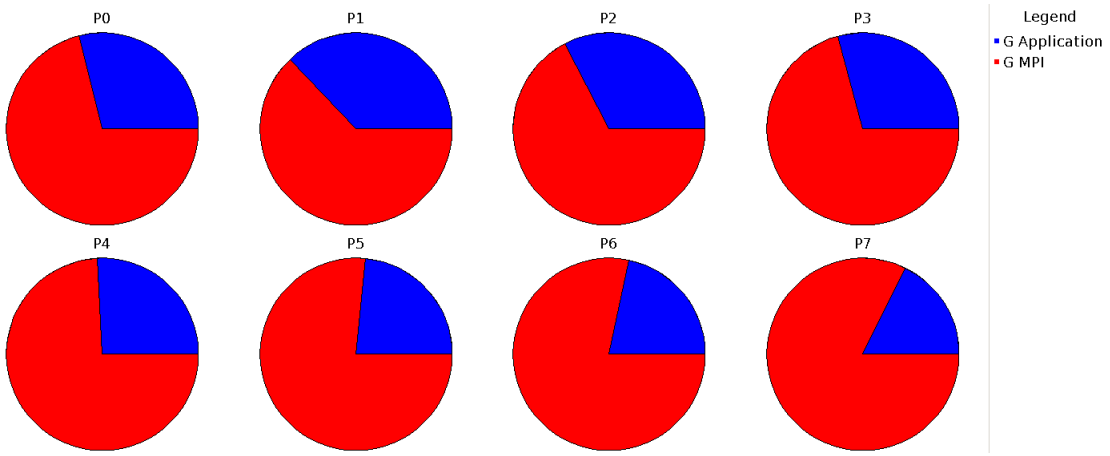
Jonathan Bosson, jonbo665

1 TotalView

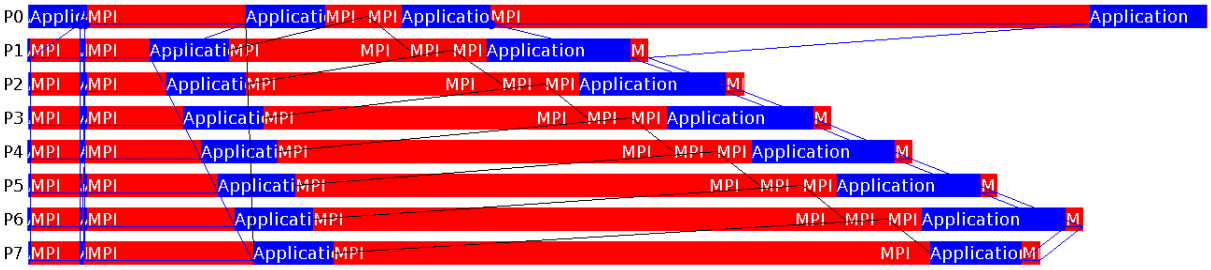
The tool TotalView was very complicated to work with. In terms of starting it it was quite straight forward but the learning curve seemed very steep so the interface was initially quite confusing. Once I got it working and had understood the UI it was pretty straight forward to use breakpoints and step forward in the code by single step or continue to run it. Previously I've worked with Visual Studios debugger as well as GDB. Compared to GDB, the graphical interface was a huge benefit. It's easier to understand what is happening through an interface than through a text based terminal debugger. TotalView seemed quite adept at visualizing multithreading in a program, something that VS can't do well. In terms of how useful, I don't feel TotalView would benefit me personally whilst developing parallel programs with the reason that I can't manage the program well. The program is really vast and powerful, and if more time was put in to learn all of its uses it would be of great help.

2 Tracing with ITAC

Compared to TotalView was ITAC really easy and smooth to get into. I didn't run into any problems and the user interface was much easier to understand. A small problem was that I seemed to be unable to close charts once opened, which in turn clogged up the screen quite quickly. My solution was to reopen the '.stf' to get a clean slate. The interface had a global timeline in which you could easily filter which specific part of the execution process was interesting to analyze. Below there is a couple of figures all viewed from a total timeline perspective using 8 processes running the MPI blurfilter program from lab 1.



The figure above displays pie charts of what each process' did during its usage. It can be seen that the major part of what most processes did was related to the MPI interface, this suggests that we are reaching a saturated number of processes working with the program. If we refer back to the graph displaying execution time in lab 1 we can see that adding more cores increased the total run time.



Second figure is an event chart that displays the communication between the different processes. It's quite clear to see at what times the processes had to wait for each other before they could continue with **P0** being the root node.

	P0	P1	P2	P3	P4	P5	P6	P7	
MPI_Bcast	10e-6	37.562e-3	37.563e-3	37.563e-3	37.561e-3	37.567e-3	37.564e-3	37.566e-3	26
MPI_Gatherv	451.206e-3	12.792e-3	12.836e-3	12.357e-3	12.447e-3	12.445e-3	12.485e-3	12.842e-3	53
MPI_Scatter	118.521e-3	46.424e-3	59.425e-3	72.615e-3	85.197e-3	98.064e-3	111.208e-3	124.019e-3	71
MPI_Allgatherv	472e-6	1.296e-3	971e-6	872e-6	448e-6	1.632e-3	1.063e-3	671e-6	7
Sum	570.209e-3	98.074e-3	...	...	...	...	162.32e-3	175.098e-3	1
Mean	142.552e-3	...	...	...	...	37.427e-3	40.58e-3	43.7745e-3	
StdDev	184.628e-3	...	...	27.523e-3	...	...	42.8612e-3	48.1987e-3	

480e-3
432e-3
384e-3
336e-3
288e-3
240e-3
192e-3
144e-3
96e-3
48e-6

Figure 3 shows how much work the different function calls took with blue being little and red being much. Preferably would we want to look into that function and try to divide it more evenly amongst the processes. We can see that the **MPI_Gatherv** required a lot from the root node, due to it collecting the data from each processor.