TDDC78 - Programming of Parallel Computers

Lab5 - Tools

Authors:
David Tran - Davtr766
Jakob Bertlin - Jakbe457

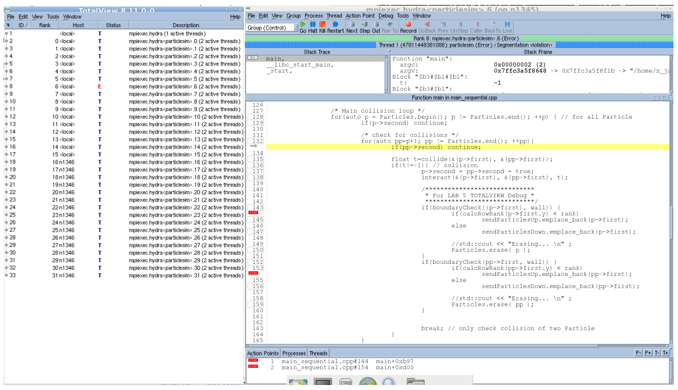
I. Introduction

Parallel algorithm can be very tricky to create and implement. To efficiently parallelize a sequential algorithm often requires good knowledge about parallelization techniques as well as good knowledge about the library that is being used to parallelize. And sometimes even the most experienced developer falls short.

To help with these difficult tasks there are a number of powerful tools available for the developers use, and in this report we will be covering ITAC and TotalView.

II. TotalView

TotalView is debugger made especially to debug multiprocess environments and gives the user a good overview over every process running. For example, in this case a program with a segmentation fault from a previous course lab is running on the Totalview debugger. In figure 1 we can see that


TotalView 8.11.0-0
File Edit View Tools Window Help
mpiexec.hydra<particlesim>.6 (on n1345)
File Edit View Group Process Thread Action Point Debug Tools Window Help
Group (Control)
Go Halt Kill Restart Next Step Out Run To Record GoBack Prev UnStep Caller BackTo Live
Stack Trace
Stack Frame
Action Points Processes Threads
P- P+ T- T+

ID	Rank	Host	Status	Description
1		<local>	T	mpiexec.hydra (1 active threads)
2	0	<local>	T	mpiexec.hydra<particlesim>.0 (2 active threads)
3	1	<local>	T	mpiexec.hydra<particlesim>.1 (2 active threads)
4	2	<local>	T	mpiexec.hydra<particlesim>.2 (2 active threads)
5	3	<local>	T	mpiexec.hydra<particlesim>.3 (2 active threads)
6	4	<local>	T	mpiexec.hydra<particlesim>.4 (2 active threads)
7	5	<local>	T	mpiexec.hydra<particlesim>.5 (2 active threads)
8	6	<local>	E	mpiexec.hydra<particlesim>.6 (2 active threads)
9	7	<local>	T	mpiexec.hydra<particlesim>.7 (2 active threads)
10	8	<local>	T	mpiexec.hydra<particlesim>.8 (2 active threads)
11	9	<local>	T	mpiexec.hydra<particlesim>.9 (2 active threads)
12	10	<local>	T	mpiexec.hydra<particlesim>.10 (2 active threads)
13	11	<local>	T	mpiexec.hydra<particlesim>.11 (2 active threads)
14	12	<local>	T	mpiexec.hydra<particlesim>.12 (2 active threads)
15	13	<local>	T	mpiexec.hydra<particlesim>.13 (2 active threads)
16	14	<local>	T	mpiexec.hydra<particlesim>.14 (2 active threads)
17	15	<local>	T	mpiexec.hydra<particlesim>.15 (2 active threads)
18	16	n1346	T	mpiexec.hydra<particlesim>.16 (2 active threads)
19	17	n1346	T	mpiexec.hydra<particlesim>.17 (2 active threads)
20	18	n1346	T	mpiexec.hydra<particlesim>.18 (2 active threads)
21	19	n1346	T	mpiexec.hydra<particlesim>.19 (2 active threads)
22	20	n1346	T	mpiexec.hydra<particlesim>.20 (2 active threads)
23	21	n1346	T	mpiexec.hydra<particlesim>.21 (2 active threads)
24	22	n1346	T	mpiexec.hydra<particlesim>.22 (2 active threads)
25	23	n1346	T	mpiexec.hydra<particlesim>.23 (2 active threads)
26	24	n1346	T	mpiexec.hydra<particlesim>.24 (2 active threads)
27	25	n1346	T	mpiexec.hydra<particlesim>.25 (2 active threads)
28	26	n1346	T	mpiexec.hydra<particlesim>.26 (2 active threads)
29	27	n1346	T	mpiexec.hydra<particlesim>.27 (2 active threads)
30	28	n1346	T	mpiexec.hydra<particlesim>.28 (2 active threads)
31	29	n1346	T	mpiexec.hydra<particlesim>.29 (2 active threads)
32	30	n1346	T	mpiexec.hydra<particlesim>.30 (2 active threads)
33	31	n1346	T	mpiexec.hydra<particlesim>.31 (2 active threads)

Rank 6: mpiexec.hydra<particlesim>.6 (Error)
Thread 1 (47811448381088): particlesim (Error) <Segmentation violation>

```
main,
__libc_start_main,
_start,
```

```
Function "main":
  argc:                    0x00000002 (2)
  argv:                    0x7ffc3a5f8648 -> 0x7ffc3a5f8f1b -> "/home/x_ja
Block "$b3#$b1#$b1":
  t:                       -1
Block "$b3#$b1":
```

Function main in main_sequential.cpp

```

        /* Main collision loop */
        for(auto p = Particles.begin(); p != Particles.end(); ++p) { // for all Particle
                if(p->second) continue;

                /* check for collisions */
                for(auto pp=p+1; pp != Particles.end(); ++pp){
                        if(pp->second) continue;

                        float t=collide(&(p->first), &(pp->first));
                        if(t!=-1){ // collision
                                p->second = pp->second = true;
                                interact(&(p->first), &(pp->first), t);

                                /*****************************
                                 * For LAB 5 TOTALVIEW Debug *
                                 *****************************/
                                if(boundaryCheck((p->first), wall)) {
                                        if(calcRowRank(p->first.y) < rank)
                                                sendParticlesUp.emplace_back(p->first);
                                        else
                                                sendParticlesDown.emplace_back(p->first);

                                        //std::cout << "Erasing... \n" ;
                                        Particles.erase( p );
                                }
                                if(boundaryCheck(pp->first, wall)) {
                                        if(calcRowRank(pp->first.y) < rank)
                                                sendParticlesUp.emplace_back(pp->first);
                                        else
                                                sendParticlesDown.emplace_back(pp->first);

                                        //std::cout << "Erasing... \n" ;
                                        Particles.erase( pp );
                                }


                                break; // only check collision of two Particle
                        }
                }
```

```
1  main_sequential.cpp#144  main+0xb97
2  main_sequential.cpp#154  main+0xd00
```

Fig. 1: Totalview GUI showing the debugger environment. This picture illustrates an segmentation fault in the code.

the debugger has successfully created an automatic breakpoint at the place of the segmentation fault and it to left there is a list of every process running with process 6 returning an E, which stands for error. Since it stopped on an iterator access the cause is likely the deletion of vector elements in the if-statement. Moreover, a manual breakpoint is then set before the point of the deletion of elements and segmentation fault so the user can accurately pinpoint the exact cause of the segmentation fault. This is manual breakpoint is seen as a red "STOP" in the row numbering.

III. ITAC

ITAC is an interactive visualization tool which is used to analyze and debug parallel programs. In this report, ITAC is used to find performance bottleneck in an auxiliary MPI program. In this case, ITAC is used on the threshold filter for the first lab in this course. In figure 2 from previous lab report, one can see that the elapsed time increases after 16 cores. Therefore there is a hot spot and performance bottleneck in the code and ITAC was used to find out the exact cause of this problem.

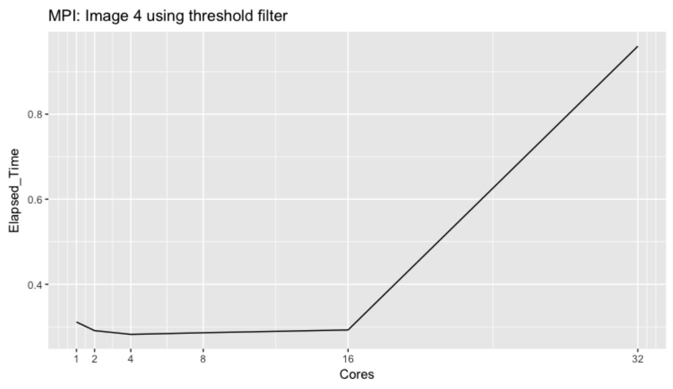
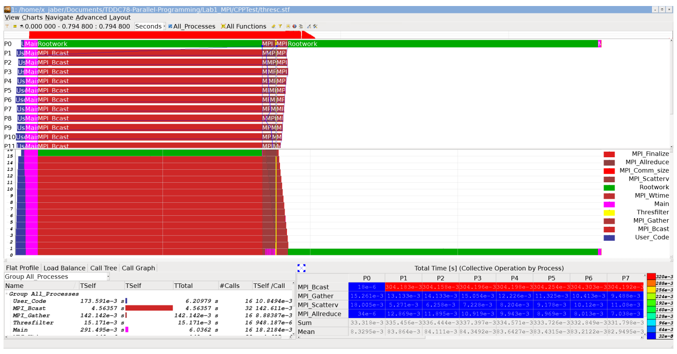


Fig. 2: Linear plot showing dependence between number of processors and elapsed time for image 4.

For this to work, the program needs to initialize the Intel Trace Collector and the underlying communication. This is done in order to generate traces showing how much time the program spends working on different parts of the algorithm. In this report, VT functions are used for the whole main code, thresfilter and root work which can be seen in figure 3.



Name	TSelf	TSelf	TTotal	#Calls	TSelf /Call
Group All_Processes					
User_Code	173.591e-3 s		6.20979 s	16	10.8494e-3
MPI_Bcast	4.56357 s		4.56357 s	32	142.611e-3
MPI_Gather	142.142e-3 s		142.142e-3 s	16	8.88387e-3
Thresfilter	15.171e-3 s		15.171e-3 s	16	948.187e-6
Main	291.495e-3 s		6.0362 s	16	18.2184e-3

Total Time [s] (Collective Operation by Process)

	P0	P1	P2	P3	P4	P5	P6	P7
MPI_Bcast	18e-6	304.183e-3	304.158e-3	304.196e-3	304.198e-3	304.254e-3	304.303e-3	304.192e-3
MPI_Gather	15.261e-3	13.133e-3	14.133e-3	15.054e-3	12.226e-3	11.325e-3	10.413e-3	9.488e-3
MPI_Scatterv	18.005e-3	5.271e-3	6.258e-3	7.228e-3	8.204e-3	9.178e-3	10.12e-3	11.08e-3
MPI_Allreduce	34e-6	12.869e-3	11.895e-3	10.919e-3	9.943e-3	8.969e-3	8.013e-3	7.038e-3
Sum	33.318e-3	335.456e-3	336.444e-3	337.397e-3	334.571e-3	333.726e-3	332.849e-3	331.798e-3
Mean	8.3295e-3	83.864e-3	84.111e-3	84.3492e-3	83.6427e-3	83.4315e-3	83.2122e-3	82.9495e-3

Fig. 3: ITAC GUI showing event timeline, quantitative timeline, function profile and collective operations profile. Root work is defined as green, thresfilter as yellow and main as pink. The red bar charts are MPI calls.

IV. Discussion

Debugging with the TotalView debugger is fairly straightforward if you have previously used another debugger. It does however give more sophisticated tools for handling multiple processes at once compared to the GBD debugger that is a common tool for debugging sequential C/C++ code. In the case of a segmentation fault the TotalView debugger is perfect since running the program normally just causes a crash.

Figure 3 shows that the majority of work was done by MPI. This is certainly a performance bottleneck because the communication operations was more expensive than actual threshold filter operation. The threshold filter which is shown as yellow is barely even seen as having any impact at all