

1 Finding a suitable resolution

For comparing different code approaches a good resolution is needed. If a low resolution is chosen, so that Δx is low, the program will finish very quick. If we want to make a comparison this wont work very well. The difference in results based on the change of code will be overwhelmed by the difference in results due to changes in cpu-load. A too high resolution is also unwanted, for a good comparison multiple runs are required. If the program takes too long too finish less runs can be done. The resolution is experimentally found. Starting at 0.005 we found that the program run times were just too low. Eventually we settled at a value of 0.002, with this resolution running on normal computers a good difference could be seen. However when we ran the program at das4 the multiple cores kicked in, so we had to lower the resolution to 0.001 to be in the clear.

2 Work schedulers

The mandelbrot application isn't an application with a fixed amount of instructions in each loop. In the loop a while statement sits, which runs a couple of times depending on the data. So if each thread get a fixed amount of points to process some threads will get points that exit the while loop quickly, while a other thread can get points that take a long time to exit the while loop. It will be clear that these threads wont finish their portion of points at the same time, one thread will be finished before the other one. In the static scheduling all the work is divided before starting the threads. The thread that is finished before the other threads will just sit there and do nothing because it cannot steal points from other threads. The dynamic scheduler doesn't divide all the point in the begin. Each thread gets a couple of points assigned. if a thread finishes it can request more points to process. The dynamic scheduler will therefore be perfect for our goal. We can see from table 1 that this works in practice as well. The dynamic scheduler is much faster than the static scheduler. The guided scheduler also works quite good, but not as good as the dynamic one.

Table 1: Running mandel with output using different schedulers.
The resolution is 0.001.

Dynamic	Static	Guided
0,470178	0,757189	0,501167
0,47692	0,768196	0,503846
0,481399	0,784334	0,5018
0,482518	0,806455	0,502887
0,490624	0,797248	0,523842
0,508222	0,782431	0,539778
0,504364	0,821781	0,539267
0,533801	0,805446	0,552913
0,528836	0,816622	0,577115
0,481194	0,818687	0,554339
0,479515	0,765153	0,501227
0,479229	0,771336	0,500563
0,482719	0,765416	0,487623
0,464592	0,786886	0,490257
0,500019	0,749763	0,540523
0,482475	0,764995	0,530378
0,521944	0,769605	0,552142
0,523241	0,818235	0,554799
0,533844	0,814846	0,503505
0,483639	0,769063	0,50325
0,495047	0,786786	0,522027

Table 2: Running mandel without output and compiler flag O3.
The resolution is 0.001.

Sequential	Dynamic
0,651594	0,0818824
0,651586	0,0818495
0,651614	0,0818448
0,651548	0,0834362
0,651608	0,0818953
0,651556	0,0818747
0,651563	0,081833
0,651443	0,0840247
0,651496	0,0818772
0,651625	0,0818726
0,651571	0,082067

3 Best run times

In table 2 we can see the runtimes of the program without any output. In this case the O2 compiler flag is used. As we can see the program finishes very quickly using the parallel approach and the dynamic scheduler. This is not a valid result because the compiler optimizes the code by removing unused code. In our case of no output at all, all the parallel code can be removed. Most of the remaining time will be starting up the threads which terminate immediately. The sequential code doesn't finish instantly, this could be caused by the OpenMP compiler. The OpenMP compiler could be more reserved for optimizing sequential code than the normal compiler.

To get some real results we disabled the compiler optimizations by removing the O3 flag.

Table 3: Running mandel without output using different schedulers and sequential.
The resolution is 0.001 and no compiler optimizations are used.

Sequential	Dynamic	Static	Guided
1,18568	0,768784	0,82672	0,772624
1,18586	0,769449	0,84205	0,776406
1,18567	0,76848	0,842205	0,77863
1,18552	0,77718	0,829137	0,78726
1,18568	0,774852	0,852485	0,788156
1,18566	0,791338	0,840201	0,782334
1,1856	0,771359	0,83925	0,782231
1,18547	0,773696	0,853402	0,782055
1,1856	0,796883	0,926374	0,814374
1,18549	0,771051	0,849094	0,776001
1,1857	0,770216	0,807327	0,780678
1,18569	0,7672	0,836319	0,787131
1,18566	0,770971	0,843178	0,780753
1,1857	0,776165	0,849376	0,784131
1,1855	0,770631	0,851354	0,776077
1,18562	0,775439	0,860626	0,789332
1,18573	0,795864	0,85321	0,778719
1,18562	0,797003	0,856143	0,779836
1,18549	0,771624	0,85225	0,785575
1,18554	0,770685	0,847866	0,768145
1,185619	0,775815	0,845826	0,781552