Supercomputers TMA 4280
P1
Tobias Johansen - tobiaj

NB!
- I have run some of my programs on the supercomputer, but the screenshots in this report is all run on my computer, I hope this is not a problem, contact me if you want me to redo this.
- As I never have made makefiles before and was unsure how to do this I simply made one makefile in P1. make builds all files in the project, make test runs all computations (for all tasks), make utest runs the unit test that was made in question 2 and make vtest runs the verification test specified in question 3. I wasn't sure if I was supposed to make any unit tests for the parallelized programs, so didn't include this but I'm sure you understand that I'm able to do this based on the unit test I already wrote.
-Feel free to contact me if there is anything missing as I would be happy to provide this.

Question 1

```
zeta0/test.o 100000
Serial approximation of pi using the Riemann Zeta function after 100000 iterations: 3.14158310432645616

mach0/test.o 100000
Serial approximation of pi using the Machin formula after 100000 iterations: 3.14159265358979400
```

Question 2

```
zeta0/utest.o
Serial unit test for Riemann Zeta function passed, expected and computed value after 3 iterations: 2.85774
mach0/utest.o
Serial unit test for Machin formula passed, expected and computed value after 3 iterations: 3.14162
```

Why do you think such a test may be useful when parallelizing a computational code?

Such a test may be useful to ensure that the code executes as expected. That the expected number of iterations is performed and that the values from the different processes is added together correctly.

Question 3

Riemann Zeta function vtest:

Difference between PI and PI approximated by Riemann Zeta function after 2^1 iterations: 0.40298
Difference between PI and PI approximated by Riemann Zeta function after 2^2 iterations: 0.21898
Difference between PI and PI approximated by Riemann Zeta function after 2^3 iterations: 0.114295
Difference between PI and PI approximated by Riemann Zeta function after 2^4 iterations: 0.0583996
Difference between PI and PI approximated by Riemann Zeta function after 2^5 iterations: 0.0295188
Difference between PI and PI approximated by Riemann Zeta function after 2^6 iterations: 0.0148399
Difference between PI and PI approximated by Riemann Zeta function after 2^7 iterations: 0.00744013
Difference between PI and PI approximated by Riemann Zeta function after 2^8 iterations: 0.00372513
Difference between PI and PI approximated by Riemann Zeta function after 2^9 iterations: 0.00186383
Difference between PI and PI approximated by Riemann Zeta function after 2^10 iterations: 0.000932232
Difference between PI and PI approximated by Riemann Zeta function after 2^11 iterations: 0.000466195
Difference between PI and PI approximated by Riemann Zeta function after 2^12 iterations: 0.000233117
Difference between PI and PI approximated by Riemann Zeta function after 2^13 iterations: 0.000116564
Difference between PI and PI approximated by Riemann Zeta function after 2^14 iterations: 5.8283e-05
Difference between PI and PI approximated by Riemann Zeta function after 2^15 iterations: 2.91418e-05
Difference between PI and PI approximated by Riemann Zeta function after 2^16 iterations: 1.4571e-05
Difference between PI and PI approximated by Riemann Zeta function after 2^17 iterations: 7.28552e-06
Difference between PI and PI approximated by Riemann Zeta function after 2^18 iterations: 3.64276e-06
Difference between PI and PI approximated by Riemann Zeta function after 2^19 iterations: 1.82138e-06
Difference between PI and PI approximated by Riemann Zeta function after 2^20 iterations: 9.10692e-07
Difference between PI and PI approximated by Riemann Zeta function after 2^21 iterations: 4.55346e-07
Difference between PI and PI approximated by Riemann Zeta function after 2^22 iterations: 2.27673e-07
Difference between PI and PI approximated by Riemann Zeta function after 2^23 iterations: 1.13837e-07
Difference between PI and PI approximated by Riemann Zeta function after 2^24 iterations: 5.69228e-08

The accuracy of the Riemann Zeta function seems to converge rather slow and even after 2^24 iteration, the difference is as big as 10^-8. The calculations are however quick to perform and doesn’t require a lot of FLOP.

Machin formula vtest:

Difference between PI and PI approximated by Machin formula after 2^1 iterations: 0.000995624
Difference between PI and PI approximated by Machin formula after 2^2 iterations: 8.81408e-07
Difference between PI and PI approximated by Machin formula after 2^3 iterations: 1.1906e-12
Difference between PI and PI approximated by Machin formula after 2^4 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^5 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^6 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^7 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^8 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^9 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^10 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^11 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^12 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^13 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^14 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^15 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^16 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^17 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^18 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^19 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^20 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^21 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^22 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^23 iterations: 8.88178e-16
Difference between PI and PI approximated by Machin formula after 2^24 iterations: 8.88178e-16

The accuracy of the Machin formula however converges extremely fast and the precision for double precision floats becomes the reason it doesn't get better after 2^4 iterations. Iterations after this doesn't make the approximation more precise, it only generates noise. The Machin formula is extremely slow compared to the Riemann Zeta formula as the FLOP required to calculate v increases for each iteration.

Question 4

Can you comment on the limitation of such approach for the data distribution and a possible improvement?

The suggested solution for distributing data, having one process responsible for calculating each vector element and making the others sum the results, is problematic as it creates a bottleneck on the master/root process. As summing only requires one FLOP and calculating vector elements requires more than that, the master/root will never be able to provide enough work for the other processes.

An improvement would be splitting the workload amongst the processes, letting each calculate a partial sum and then combining the partial sums on the master/root. This is not ideal either, as the program wouldn't be load balanced because the workload is split into fixed pieces. The execution time of the program will always depend on the slowest process.

The processes should iterate backwards, adding the smallest numbers to each others first to avoid losing accuracy (which happens when dealing with FLOP of values with big difference in size). The iterations should also be done with steps of the number of threads, such that each process' partial sum is somewhat the same, for the same reason we start iterating from behind.

Question 5

```
mpirun -n 2 zeta1/mpi.o 10
Sum of process 0: 0.36590277777777780
Sum of process 1: 1.18386495338876285
Parallel approximation of PI using the Riemann Zeta function after 10 iterations, with 2 processes: 3.04936163598206988
Difference between PI and PI approximated by Riemann Zeta function: 0.09223101760772323
Wall time: 0.028 ms.

mpirun -n 4 zeta1/mpi.o 10
Sum of process 2: 0.07812500000000000
Sum of process 0: 0.28777777777777780
Sum of process 1: 1.05234567901234577
Sum of process 3: 0.13151927437641722
Parallel approximation of PI using the Riemann Zeta function after 10 iterations, with 4 processes: 3.04936163598206988
Difference between PI and PI approximated by Riemann Zeta function: 0.09223101760772323
Wall time: 0.083 ms.

mpirun -n 8 zeta1/mpi.o 10
Sum of process 0: 0.26000000000000001
Sum of process 1: 1.01234567901234573
Sum of process 2: 0.01562500000000000
Sum of process 4: 0.02777777777777778
Sum of process 6: 0.06250000000000000
Sum of process 7: 0.11111111111111110
Sum of process 3: 0.02040816326530612
Sum of process 5: 0.04000000000000000
Parallel approximation of PI using the Riemann Zeta function after 10 iterations, with 8 processes: 3.04936163598206988
Difference between PI and PI approximated by Riemann Zeta function: 0.09223101760772323
Wall time: 0.204 ms.

mpirun -n 2 zeta1/mpi.o 1000
Sum of process 0: 0.41073401637872353
Sum of process 1: 1.23320055030283626
Parallel approximation of PI using the Riemann Zeta function after 1000 iterations, with 2 processes: 3.14063805620599279
Difference between PI and PI approximated by Riemann Zeta function: 0.00095459738380033
Wall time: 0.079 ms.

mpirun -n 4 zeta1/mpi.o 1000
Sum of process 0: 0.10255887851134962
Sum of process 1: 0.15861772806205862
Sum of process 2: 0.30817513786737394
Sum of process 3: 1.07458282224077761
Parallel approximation of PI using the Riemann Zeta function after 1000 iterations, with 4 processes: 3.14063805620599279
Difference between PI and PI approximated by Riemann Zeta function: 0.00095459738380033
Wall time: 0.090 ms.

mpirun -n 8 zeta1/mpi.o 1000
Sum of process 2: 0.03959211965853468
Sum of process 3: 0.05301861261853706
Sum of process 4: 0.07698128505016234
Sum of process 5: 0.12740265105542908
Sum of process 6: 0.26858301820883923
Sum of process 7: 1.02156420962224059
Sum of process 0: 0.02557759346118727
Sum of process 1: 0.03121507700662953
Parallel approximation of PI using the Riemann Zeta function after 1000 iterations, with 8 processes: 3.14063805620599323
Difference between PI and PI approximated by Riemann Zeta function: 0.00095459738379988
Wall time: 0.208 ms.
```

```
mpirun -n 2 zeta1/mpi.o 100000
Sum of process 0: 0.41122851676205624
Sum of process 1: 1.23369555013616994
Parallel approximation of PI using the Riemann Zeta function after 100000 iterations, with 2 processes: 3.14158310432644061
Difference between PI and PI approximated by Riemann Zeta function: 0.00000954926335250
Wall time: 0.643 ms.

mpirun -n 4 zeta1/mpi.o 100000
Sum of process 2: 0.30842263753404281
Sum of process 0: 0.10280587922801349
Sum of process 1: 0.15886497800447547
Sum of process 3: 1.07483057213169442
Parallel approximation of PI using the Riemann Zeta function after 100000 iterations, with 4 processes: 3.14158310432644106
Difference between PI and PI approximated by Riemann Zeta function: 0.00000954926335206
Wall time: 0.393 ms.

mpirun -n 8 zeta1/mpi.o 100000
Sum of process 3: 0.05314223709125693
Sum of process 2: 0.03971561951986902
Sum of process 6: 0.26870701801417379
Sum of process 7: 1.02168833504043755
Sum of process 0: 0.02570084484450220
Sum of process 4: 0.07710503438351128
Sum of process 1: 0.03133845250507639
Sum of process 5: 0.12752652549939911
Parallel approximation of PI using the Riemann Zeta function after 100000 iterations, with 8 processes: 3.14158310432644106
Difference between PI and PI approximated by Riemann Zeta function: 0.00000954926335206
Wall time: 0.688 ms.
```

The zeta1 program seems to run the fastest on 4 processes. 100 000 iterations is not accurate enough in my opinion and could probably be higher, 10 000 000?

```
mpirun -n 2 mach1/mpi.o 10
Sum of process 1: 3.18428850955703613
Sum of process 0: -0.04269585596724425
Parallel approximation of PI using the Machin formula after 10 iterations, with 2 processes: 3.14159265358979178
Difference between PI and PI approximated by Machin formula: 0.00000000000000133
Wall time: 0.033 ms.

mpirun -n 4 mach1/mpi.o 10
Sum of process 0: -0.04266659878943459
Sum of process 1: 3.18326450854981591
Sum of process 2: -0.00002925717780966
Sum of process 3: 0.00102400100722026
Parallel approximation of PI using the Machin formula after 10 iterations, with 4 processes: 3.14159265358979178
Difference between PI and PI approximated by Machin formula: 0.00000000000000133
Wall time: 0.110 ms.

mpirun -n 8 mach1/mpi.o 10
Sum of process 0: -0.04266656900034368
Sum of process 1: 3.18326359832759387
Sum of process 3: 0.00000000100824615
Sum of process 4: -0.00000002978909091
Sum of process 5: 0.00000091022222222
Sum of process 6: -0.00002925714285713
Sum of process 2: -0.00000000003495253
Sum of process 7: 0.00102399999897411
Parallel approximation of PI using the Machin formula after 10 iterations, with 8 processes: 3.14159265358979178
Difference between PI and PI approximated by Machin formula: 0.00000000000000133
Wall time: 0.252 ms.

mpirun -n 2 mach1/mpi.o 1000
Sum of process 1: 3.18428850955703791
Sum of process 0: -0.04269585596724432
Parallel approximation of PI using the Machin formula after 1000 iterations, with 2 processes: 3.14159265358979356
Difference between PI and PI approximated by Machin formula: 0.00000000000000044
Wall time: 0.265 ms.

mpirun -n 4 mach1/mpi.o 1000
Sum of process 0: -0.00002925717780972
Sum of process 2: -0.04266659878943459
Sum of process 3: 3.18326450854981591
Sum of process 1: 0.00102400100722186
Parallel approximation of PI using the Machin formula after 1000 iterations, with 4 processes: 3.14159265358979356
Difference between PI and PI approximated by Machin formula: 0.00000000000000044
Wall time: 0.305 ms.

mpirun -n 8 mach1/mpi.o 1000
Sum of process 0: -0.00000000003495253
Sum of process 1: 0.00000000100824615
Sum of process 2: -0.00000002978909091
Sum of process 3: 0.00000091022222222
Sum of process 4: -0.00002925714285719
Sum of process 5: 0.00102399999897571
Sum of process 6: -0.04266656900034368
Sum of process 7: 3.18326359832759387
Parallel approximation of PI using the Machin formula after 1000 iterations, with 8 processes: 3.14159265358979312
Difference between PI and PI approximated by Machin formula: 0.00000000000000000
Wall time: 0.289 ms.
```

```
mpirun -n 2 mach1/mpi.o 100000
Sum of process 1: 3.18428850955703791
Sum of process 0: -0.04269585596724432
Parallel approximation of PI using the Machin formula after 100000 iterations, with 2 processes: 3.14159265358979356
Difference between PI and PI approximated by Machin formula: 0.00000000000000044
Wall time: 22.893 ms.

mpirun -n 4 mach1/mpi.o 100000
Sum of process 1: 0.00102400100722186
Sum of process 0: -0.00002925717780972
Sum of process 3: 3.18326450854981591
Sum of process 2: -0.04266659878943459
Parallel approximation of PI using the Machin formula after 100000 iterations, with 4 processes: 3.14159265358979356
Difference between PI and PI approximated by Machin formula: 0.00000000000000044
Wall time: 12.993 ms.

mpirun -n 8 mach1/mpi.o 100000
Sum of process 3: 0.00000091022222222
Sum of process 7: 3.18326359832759387
Sum of process 5: 0.00102399999897571
Sum of process 1: 0.00000000100824615
Sum of process 4: -0.00002925714285719
Sum of process 0: -0.00000000003495253
Sum of process 6: -0.04266656900034368
Sum of process 2: -0.00000002978909091
Parallel approximation of PI using the Machin formula after 100000 iterations, with 8 processes: 3.14159265358979312
Difference between PI and PI approximated by Machin formula: 0.00000000000000000
Wall time: 7.018 ms.
```

The mach1 program seems to run the fastest on 2 processes on few iterations and 8 processes on many iterations. The accuracy is however good enough after around 16 iterations (see vtest), which means that 2 processes and 16 iterations might be ideal.
The MPI calls used was reduce to sum the partial sums from all processes. I also used MPI_Init and MPI_Finalize (obviously). The wall time was computed by saving a double at start of the execution containing MPI_wTime and subtracting that from current wTime at program completion.

Question 6

The answers from the single-process program and multi-process program with P=2 and P=8 is approximately the same. The differences in values is a cause of how double precision numbers are added together. The single-process program always add the numbers together in the same order, while I assume the multi-process program will add the partial sums based on which order they complete. The difference in values which are to be added together are also bigger in the multi-process programs (as far as I have implemented mine), which may cause more accuracy loss than in the single-process program. The values can also differ based on the hardware its run on because of how the doubles are stored in cache etc.

Question 7

```
tobias@tobias-laptop:~/Documents/TMA4280v2018/P1$ mpirun -n 2 reduc/machinAllreduce.o 1000000
Sum of process 1: 3.18428850955703791
Sum of process 0: -0.04269585596724432
Parallel approximation of PI using the Machin formula after 1000000 iterations, with 2 processes: 3.1415926535897936
Difference between PI and PI approximated by Machin formula: 4.4408920985006262e-16
Wall time: 216 ms.

tobias@tobias-laptop:~/Documents/TMA4280v2018/P1$ mpirun -n 8 reduc/machinAllreduce.o 1000000
Sum of process 1: 0.00000000100824615
Sum of process 7: 3.18326359832759387
Sum of process 3: 0.00000091022222222
Sum of process 2: -0.00000002978909091
Sum of process 5: 0.00102399999897571
Sum of process 6: -0.04266656900034368
Sum of process 0: -0.00000000003495253
Sum of process 4: -0.00002925714285719
Parallel approximation of PI using the Machin formula after 1000000 iterations, with 8 processes: 3.1415926535897931
Difference between PI and PI approximated by Machin formula: 0
Wall time: 66.4 ms.

tobias@tobias-laptop:~/Documents/TMA4280v2018/P1$ mpirun -n 2 reduc/machinRDS.o 1000000
Sum of process 1: 3.18428850955703791
Sum of process 0: -0.04269585596724432
Parallel approximation of PI using the Machin formula after 1000000 iterations, with 2 processes: 3.1415926535897936
Difference between PI and PI approximated by Machin formula: 4.4408920985006262e-16
Wall time: 217 ms.

tobias@tobias-laptop:~/Documents/TMA4280v2018/P1$ mpirun -n 8 reduc/machinRDS.o 1000000
Sum of process 7: 3.18326359832759387
Sum of process 5: 0.00102399999897571
Sum of process 1: 0.00000000100824615
Sum of process 3: 0.00000091022222222
Sum of process 4: -0.00002925714285719
Sum of process 0: -0.00000000003495253
Sum of process 6: -0.04266656900034368
Sum of process 2: -0.00000002978909091
Parallel approximation of PI using the Machin formula after 1000000 iterations, with 8 processes: 3.1415926535897931
Difference between PI and PI approximated by Machin formula: 0
Wall time: 65.5 ms.
```

It seems like both all reduce on recursive doubling sum has about the same wall time and the same precision. They are both done in log2 P iterations, which means they both become increasingly slower the higher number of processes added.

Question 8

Unfortunately I forgot to do save the results of the analysis on the OpenMP solution before I made changes to make it use MPI as well. The differences here varied from the MPI solution. The Machin program forexample hit 0 difference at 11 iterations and stayed at 4.4*10^16 difference at all iterations past that. I can only assume that the addition of the threads is done differently for OpenMP than it is done in MPI and has probably something to do with OpenMP has a shared memory model.

Question 9

This configuration is slower on few cores, but faster on multiple. There are a reduced memory requirement, because the MPI processes require more memory than the threads provided by OpenMP. There is also a load balancing component in OpenMP which makes better use of the parallelization.

Question 10

The multiprocess-programs require more memory than the single process program because it has to share it. Queues, locks and mutexes may occur if there is too little memory available, something that doesn't happen for the singleprocess-program.

The number of FLOP needed to generate v for the Riemann Zeta function is 2, one for multiplying i by i and one for dividing 1 by that product.
The number of FLOP needed to generate v for the Machin formula is 3i+5 where i is the number of the current iteration. i-1 FLOP for (-1)^(i-1), 2i-1 FLOP for x^(2i-1), 3 FLOP for calculating the exponents, 2 FLOP for calculating (2i-1) and lastly 2 FLOP for multiplying and dividing the results. This number gets very high as the iterations go up and is why the Machin formula is so expensive.

The number of FLOP needed to generate S for the Riemann Zeta function is 3i (calculate the vector and add it to sum)
The number of FLOP needed to generate S for the Machin formula is 2*(number of FLOP to generate v)+3FLOP (multiply by 4, subtract one arctan and add to sum):
$2*(\sum i=1 \Rightarrow i=n\ (3i+5))+3$

The multiprocess program in question 9 is semi load balanced. The processes are assigned a fixed amount of iterations to calculate, while the threads inside each process are load balanced.
The other multiprocess-programs however are not load balanced and will each calculate the same number of iterations.

Question 11

Parallel processing surely makes the computations faster at high numbers of iterations. The Machin formula does however not need a lot of iterations to converge, making the way double precision numbers behave be the reason a better estimate is not possible to get. Surely you get a faster way to plow through all the iterations, but the precision doesn't get better, so theres really no use. This means that using a multiprocessor program to calculate PI using the machin formula seems unattractive. A singleprocessor-program would do the job just as good. When it comes to the Riemann Zeta function however, which converges slowly and has vector elements which are easy to

calcuate, a multiprocessor-program is not a bad idea. This function requires a high amount of iterations making it a good target for multiprocessor-programs.