Project 1 tma4280

Trygve Aune

14. mars 2018

1 Serial implementation

The serial implementation is added to mach0.f90 and zeta0.f90

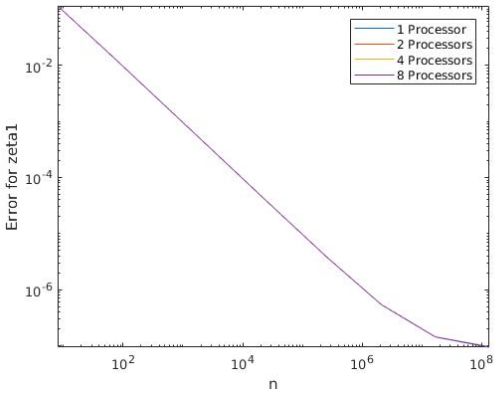
2 Unit test

A unit test is added to the scripts, it compares the scripts' values for the series with a handcrafted S_3. Beware that this comparison only holds for $n = 3$, n being introduced as the first argument, the unit test is run with 0 as argument to the executable. A unit test might be useful to see if you are on the right way, if the code do not work for small n you should change it. Can be run by make utest.

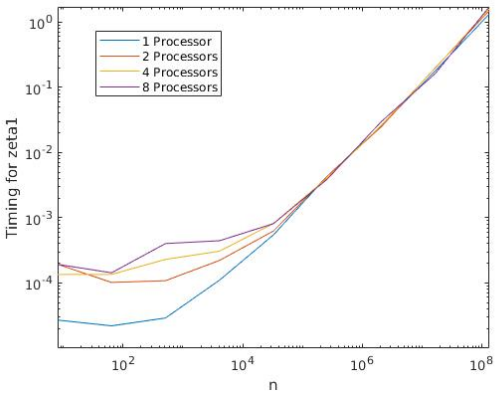
3 Verification test

I am now interested to find the error $|\pi - \pi_n|$ of $\pi = \sqrt{6S_n}$, (with $S_n = \sum_{i=1}^{n} \frac{1}{i^2}$), for the zeta0 and $\pi = 16S_n^{(1)} - 4S_n^{(2)}$, (with $S_n^{(1)} = \sum_{i=1}^{n} (-1)^{i-1} \frac{(\frac{1}{5})^{2i-1}}{2i-1}$ and $S_n^{(2)} = \sum_{i=1}^{n} (-1)^{i-1} \frac{(\frac{1}{239})^{2i-1}}{2i-1}$), for the mach0. I will use $n = 2^k$ with $k \in \{1, 2, ..., 24\}$. The results of vtest can be found in the text files `error_mach.txt` and `error_zeta.txt`. The error of zeta0 seem to be diveded by approximately 2 for each k. The error of mach0 can be seen to stop improving after $n = 2^3$, the reason is the limitation of double precision. The smallest number that can be added to 1 is $2^{-52} = 2.220446 \times 10^{-16}$ and 2 is $2^{-51} = 4.440892 \times 10^{-16}$, $S_n^{(1)}$ and $S_n^{(2)}$ are both smaller than this for $n > 10$. The verification test can be run with make vtest, or with argument input 1 to the executable.

Figur 1: Error for zeta1



Figur 2: Time for zeta1



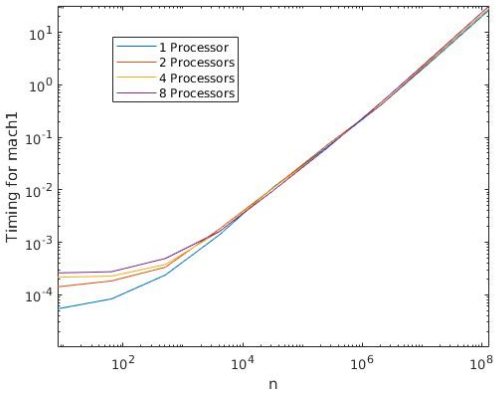
4 Data distribution

I generate vectors to store the ν_i elements in process 0 and distribute the elements equally between the processes. It can be found in zeta1 and mach1. The obvious limitation to such a data distribution is that for the vector generation the root do all the work, I could spread the amount of work around even more.

5 MPI implementation

I include mpi to zeta1 and mach1. A couple of errors and the timing for π^k, $k \in \{3, 6, .., 27\}$ can be found in the tables in the next exercise, and in the logplots in figures 1,2 and3, the errors for mach is redundant. The calls MPI_ Init, MPI_ Finalize, MPI_ Comm_ size and MPI_ Comm_ rank where obvious calls in these scripts, aswell I needed send and receive calls to get the partitioned vectors from the root to the other processes, (and the sums back), I simply used MPI_ Send and MPI_ Recv. Results can be found with make mpi.

Figur 3: Time for mach1



6 Analysis

Some of the errors and timings, (from Lille), using mpi are presented in the tables underneath. The errors should be somewhat similar, but not entirely necessarily, since the error can depend on the order of which the vector elements are added, different roundings might occur. To compute the wall time I used MPI_ Wall time. The wall time mostly increases when using more than one processor, sending elements between processes for little work is not necessarily effective. I can see from figure 3 and 2 that many processors slows the script down when little work is done, but more surprisingly does not help us when n is increased either. For one run of the mach script for $n = 2^{27}$ I found that 22.1 of the 22.9 seconds runtime was used for generating and distributing elements.

Time for zeta1:

Time	k=3	k=12	k=18	k=24
P=1	$2.698972 \cdot 10^{-5}$	$1.103710 \cdot 10^{-4}$	$4.357044 \cdot 10^{-3}$	0.178480
P=2	$1.930650 \cdot 10^{-4}$	$2.217833 \cdot 10^{-4}$	$4.353721 \cdot 10^{-3}$	0.199385
P=8	$1.930669 \cdot 10^{-4}$	$4.441682 \cdot 10^{-4}$	$3.920201 \cdot 10^{-3}$	0.162326

Error for zeta1:

$\lvert\pi - \pi_n\rvert$	k=3	k=12	k=18	k=24
P=1	0.1142949	$2.332047 \cdot 10^{-4}$	$3.730185 \cdot 10^{-6}$	$1.443456 \cdot 10^{-7}$
P=2	0.1142948	$2.332047 \cdot 10^{-4}$	$3.730185 \cdot 10^{-6}$	$1.443406 \cdot 10^{-7}$
P=8	0.1142948	$2.332047 \cdot 10^{-4}$	$3.730186 \cdot 10^{-6}$	$1.443410 \cdot 10^{-7}$

Timing for mach1:

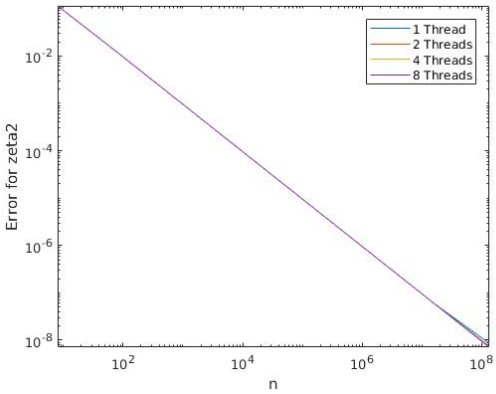
time	k=3	k=12	k=18	k=24
P=1	$5.384907 \cdot 10^{-5}$	$1.391388 \cdot 10^{-3}$	$6.649106 \cdot 10^{-2}$	3.278655
P=2	$1.416169 \cdot 10^{-4}$	$1.797767 \cdot 10^{-3}$	$7.214313 \cdot 10^{-2}$	3.528427
P=8	$2.607591 \cdot 10^{-4}$	$1.609365 \cdot 10^{-3}$	$6.290021 \cdot 10^{-2}$	3.842326

The error is similar, but not identical, that is because addition of floating point numbers is not commutative

$$(a + b) + c \neq a + (b + c),$$

so you are almost guarateed to get at least small differences as numbers are added and subtracted in different ways.

Figur 4: Error for zeta2



7 Global reduction

I modified zeta1 and mach1 using the calls MPI_ Sum with MPI_ Allreduce since all processes are meant to store the final sums. Neither the errors nor the timings, (almost all is generating and distributing the vector anyway), are significantly different from what I got without reduction, they can be found through "make reduc".

8 OpenMP implementation

OpenMP is implemented in zeta2.c and mach2.c, they are basically derived from the data distribution part of the mpi scripts zeta1.c and mach1.c, (from last time I did this project, they are essentially equivalent to zeta1.f90 and mach1.f90). Run with number of processes equaling 1 these will be OpenMP scripts. I used the calls

#pragma omp parallel for schedule(static)

in the vector generation, and

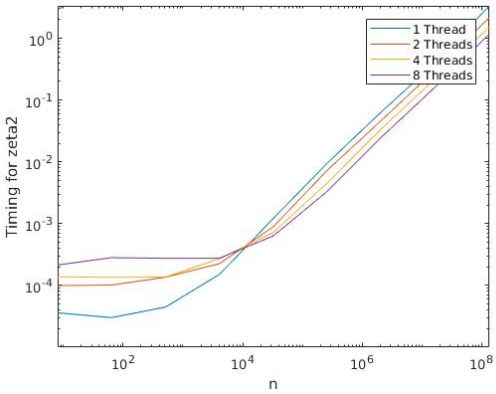
#pragma omp parallel for reduction(+:sum)

in summing the vector elements. The errors and timings can be found in figure 4,5 and 6.

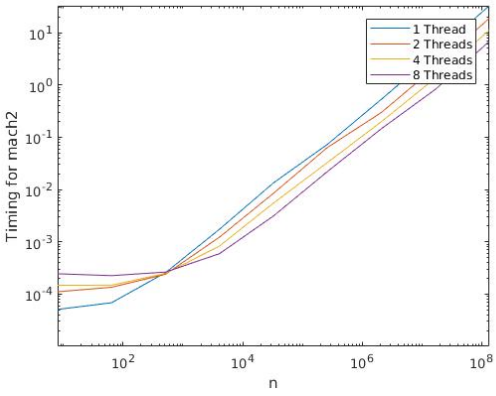
I can clearly see that as for mpi fewer processors are good for small n, but for larger n it is clearly advantageous with more threads. The reason for this is that the time consuming generation and distributing part is open for many threads while in the mpi script it only was for processor 0. We might want to have a closer look on the error of the zeta2 function:
Error for openmp zeta2:

$\lvert\pi - \pi_n\rvert$	k=3	k=12	k=18	k=24
thread=1	0.1142948	$2.331173 \cdot 10^{-4}$	$3.642763 \cdot 10^{-6}$	$5.692283 \cdot 10^{-8}$
thread=2	0.1142948	$2.331173 \cdot 10^{-4}$	$3.642763 \cdot 10^{-6}$	$5.691878 \cdot 10^{-8}$
thread=8	0.1142948	$2.331173 \cdot 10^{-4}$	$3.642763 \cdot 10^{-6}$	$5.691821 \cdot 10^{-8}$

Figur 5: Time for zeta2



Figur 6: Time for mach2



Comparing this to the errors of zeta1 and error_ zeta.txt, it can be seen that the error gets smaller faster than it should, especially for larger n, this is probably caused by some error in the summing. This can all be found with "make openmp".

9 Hybrid MPI/OpenMP implementation

The zeta2 and mach2 scripts run also when increasing with number of processors in mpirun the call

OMP_ NUM_ THREADS=(# threads) mpirun -n (# processors) ./outpat3 (k).

The advantage of using mpi is reduced time from distributing work, and with added OpenMP we can also reap the benefits of parallel computing when the data and workload is badly distributed.

10 Discussion

The memory requirements does not necessarily need to be much greater, without reduction most of the memory goes to the vector with the serial elements, which can be $\frac{n}{size}$ elements in size per process and therefore unchanged regardless of the number of processes. In the zeta script I need $2n$ floating point operations to generate the elements $\frac{1}{i \cdot i}$ for vector v. For mach it all depends on how many FLOP the function $pow(x, y)$ uses, which is y floating point operations if $y - 1$ is a positive integer, remember that $v_i = (-1)^{i-1} \frac{x^{2*i-1}}{2*i-1}$ the number of floating point operation to generate v becomes approximately $3 \sum_{i=1}^{n} i = \frac{3n(n+1)}{2} = \mathcal{O}(n^2)$. Number of floating point numbers needed to compute S_n are adding $n - 1$ plus the the number of floating point numbers required to compute the vector elements. The multi-process program is load balanced for everything apart the generation of v, this is something for which openMP could be of help.

11 Conclusion

Parallel processing is attractive if I have loads of work to do, the problem here is that MPI and openMP designates effort to copying memory and sending messages between processes.