Project 1 in TMA4280

723818

February 20, 2015

0.1 Introduction

The goal of this exercise was to sum

$$\sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}$$

in serial and parallel, using both **OpenMP** and **MPI**, and compare the results.

0.2 Serial code

A program designed to run in serial was made. It generates a vector of elements $v(i) = 1/i^2$, $\max(i) = n = 2^k$, $k \in \{3, \cdots, 14\}$. It then computes the sum in double precision, and compares it to the limit of the sum. The program made to do this is called *single.c*.

0.3 parallel code

When writing the program in parallel there are two choices, using **OpenMP**, or using **MPI**. We are going to explore both, working individually and together.

0.3.1 OpenMP

The program from 0.2 was rewritten to use **OpenMP** to allow parallel computations. The resulting program is called *openmp.c*.

0.3.2 MPI

The program *mpi.c* was written to use **MPI** to be able to run in parallel. Here processor 0 is responsible for dividing the workload to all the other processors, and writing out the resulting difference between the limit and the finite sum.

It was neccesary to use **MPI_Send**, and **MPI_Recv** to tell the different processors which elements they where responsible for summing. Summing everything from all the processors was done convenient by the function **MPI_Sum**.

0.3.3 OpenMP and MPI

The program *mpi.c* was rewritten to allow usage of both **OpenMP** and **MPI**. The resulting program was called *openmpi.c*. Also a program called *superkjapp.c* was made, where dividing the workload is done more dynamic. It also allows greater precision.

0.4 Running the programs

All programs are made by the makefile, and the command: `make`. The run–commands are given in table 1.

Program	Run–command
single.c	`./single`
openmp.c	`OMP_NUM_THREADS=P ./openmp k`
mpi.c	`mpirun -np P mpi k`
openmpi.c	`OMP_NUM_THREADS=P1 mpirun -np P2 mpi k`
superkjapp.c	`OMP_NUM_THREADS=P1 mpirun -np P2 mpi k`

Table 1: Run–commands for the different programs

0.5 Results

	Single	**OpenMP**	**OpenMP**	**MPI**	**MPI**	
		$P = 2$	$P = 8$	$P = 2$	$P = 8$	
2^3	1.175120	1.175120	1.175120	1.175120	1.175120	$e-01$
2^4	6.058753	6.058753	6.058753	6.058753	6.058753	$e-02$
2^5	3.076680	3.076680	3.076680	3.076680	3.076680	$e-02$
2^6	1.550357	1.550357	1.550357	1.550357	1.550357	$e-02$
2^7	7.782062	7.782062	7.782062	7.782062	7.782062	$e-03$
2^8	3.898631	3.898631	3.898631	3.898631	3.898631	$e-03$
2^9	1.951219	1.951219	1.951219	1.951219	1.951219	$e-03$
2^{10}	9.760858	9.760858	9.760858	9.760858	9.760858	$e-03$
2^{11}	4.881621	4.881621	4.881621	4.881621	4.881621	$e-04$
2^{12}	2.441108	2.441108	2.441108	2.441108	2.441108	$e-04$
2^{13}	1.220629	1.220629	1.220629	1.220629	1.220629	$e-04$
2^{14}	6.103329	6.103329	6.103329	6.103329	6.103329	$e-05$

Table 2: Difference in limit and sum, from solving the problem with different methods.

As we can see from table 2, the summing is precise with 6 decimals accuracy. In general the sum will differ because of a round-off error when adding large numbers with small numbers. In this case, it will occur in the reduction part of the programs.

0.6 Computational complexity

Some numbers that might be of interest while dealing with parallel processing is memory usage per processor, due to the small size of the cache, and to see if we need to preform any additional FLOP per iteration.

0.6.1 Memory usage

Memory usage per processor, for large n is:

- For serial code: n doubles.

- For parallel code: n/P doubles.

0.6.2 Floating point operations

Number of floating point operations needed to make the vector $v[i]$ for the serial code: $13{+}n \cdot 7$.
Number of floating point operations needed to make the sum S_n for the **OpenMP** code: $13{+}n \cdot 8$.

Multiprocessor is load balanced because the division and adding costs no more with large numbers than with small numbers.

0.7 Conclusion

Due to the lack of precision in doubles, the sum cannot be approximated with more than about 7 digits. This takes less than a second with any program. Thus parallel is not really necessary in this case.