TMA4280 Super Computers

Jakob Hovland
Jørgen Grimnes
Assignment 4

Spring 2015

1 Introduction

This report presents a solution to homework project number four in TMA4280 Super Computers, spring 2015, at Norwegian University of Science and Technology. The assignment is a introduction to implementing efficient and parallel programs.

2 Write a program (either in C or in FORTRAN) which will do the following

- generate the vector v
- compute the sum S_n in double precision on a single processor
- compute the difference $S - S_n$ for $n = 2^k$, with $k = 3, .., 14$
- print out the difference $S - S_n$ in double precision.

The implemented program was written in C++ and tested on a intel compiler. The lightweight program of only about 25 SLOC uses the build-in C++ vector data structure.

k	$S - S_n$	**k**	$S - S_n$
3	0.11751	9	0.0019512
4	0.060588	10	0.00097609
5	0.030767	11	0.00048816
6	0.015504	12	0.00024411
7	0.0077821	13	0.00012206
8	0.0038986	14	6.1033e-05

Table 1: Error measurements for the linear program

3 Do the necessary changes to utilize shared memory parallelization through OpenMP

We included OpenMP parallelization by using the compiler pragma *omp parallel for*. Please see refer to the file *summation openmp.cpp*

4 MPI program

Write a program to compute the sum S_n using P processors where P is a power of 2, and a distributed memory model (MPI). The program should work as follows: Only processor 0 should be responsible for generating the vector elements. Processor 0 should partition and distribute the vector elements evenly among all the processors. Each processor should be responsible for summing up its own part. At the end, all the partial sums should be added together and made available on processor 0 for printout. Report the difference $S - S_n$ in double precision for different values of n.

Please see the file *summation mpi.cpp*.

5 Confirm that your program also works when you are using OpenMP/MPI in combination.

Please see the file *summation mpi openmp.cpp* for compiling example of using MPI and OpenMP in combination.

6 Which MPI calls are convenient/necessary to use?

We have been operating with the *MPI::COMM_WORLD* communication group in this project.

Call	**Description**
MPI::Init	Initializes the MPI execution environment
MPI::COMM_WORLD.Get_size	Returns the size of the group associated with COMM_WORLD.
MPI::COMM_WORLD.Get_rank	Determines the rank of the calling process in COMM_WORLD.
MPI::COMM_WORLD.Send	Performs a standard-mode blocking send.
MPI::COMM_WORLD.Recv	Performs a standard-mode blocking receive.
MPI::COMM_WORLD.Reduce	Reduces values on all processes within COMM_WORLD
MPI::Finalize	Terminates MPI execution environment.

Table 2: Executed MPI calls

7 Compare the difference $S - S_n$ from the single-processor program and the multi-processor program when $P = 2$ and when $P = 8$. Should the answer be the same in all these cases?

The error rate of our calculation $S-S_n$ will increase as we distribute our program over an increasing number of processors due to the floats and rounding error. This is clearly visible in Table 3 and Table 4. The tables compares the error measurments between the linear and the dual/octo processed programs as we increase k.

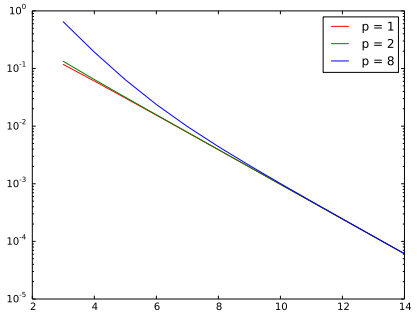

Figure 1: Comparison of the error reduction

k	$error_{P=2}$	$error_{P=1}$	**k**	$error_{P=2}$	$error_{P=1}$
3	0.13314	0.11751	9	0.001955	0.0019512
4	0.064494	0.060588	10	0.00097704	0.00097609
5	0.031743	0.030767	11	0.0004884	0.00048816
6	0.015748	0.015504	12	0.00024417	0.00024411
7	0.0078431	0.0077821	13	0.00012208	0.00012206
8	0.0039139	0.0038986	14	6.1037e-05	6.1033e-05

Table 3: Error measurements for $P = 2$

k	$error_{P=8}$	$error_{P=1}$	**k**	$error_{P=8}$	$error_{P=1}$
3	0.64493	0.11751	9	0.00208	0.0019512
4	0.19244	0.060588	10	0.0010083	0.00097609
5	0.063731	0.030767	11	0.00049621	0.00048816
6	0.023745	0.015504	12	0.00024612	0.00024411
7	0.0098423	0.0077821	13	0.00012257	0.00012206
8	0.0044137	0.0038986	14	6.1159e-05	6.1033e-05

Table 4: Error measurements for $P = 8$