Multicore Processors: Architecture & Programming, Programming Assignment - 2

Tulsi Jain, TJ983

1 Solve Parallel Linear Equations

There are n linear equations and n unknowns. Given this set of n equations with n unknowns (x_1 to x_n), write a program that will calculate the values of x_1 to x_n to within an error margin of e%. The format of the file is:

2 Approach

This problem is solved using iterative approach, until difference between newly calculated and old value is less than pre-specified error (0.001, in this assignment) for every x_i, iteration is repeated else x_i is output as an acceptable solution.

3 Experiments

Two main experiments have been evaluated. First, when each process read input file separately. This approach did not provide any speedup. In the second approach, every process read only the part of file required for computation. Disk reading is major source of blocking hence this provided a huge speed up with respect to one process version. For large input, latter approach is also scaleable, means with increase in number of processes real-time decreases when number of equations and number of unknowns is kept the same. This is because to take benefit of mutlicore processors an application should be inclined towards CPU bound rather IO bound.

4 Results

Time (in seconds)

Unknown/ Process	1	2	10	20	40
10	0.3762	0.3826	0.4924	-	-
100	0.3792	0.384	0.4734	0.5922	0.9522
1000	0.7164	0.577	0.5594	0.6818	1.1468
10000	37.3842	21.561	5.709	3.616	3.254

Speedup

Unknown/ Process	1	2	10	20	40
10	1	0.983	0.764	-	-
100	1	0.987	0.801	0.64	0.398
1000	1	1.241	1.28	1.05	0.624
10000	1	1.733	6.548	10.338	11.488

5 Result Explanation

5.1 a

In the following case no speedup is observed.
1. If problem size is small, 10 and 100 irrespective of number of processes.
2. If problem size is 1000 with and run with 40 processes.

5.2 b

Primary, reasons behind no speedup is more process creation overhead and communicating cost than parallel computation benefits. Like for small problem size, 10 and 100 unknowns, work per process is very small. Calculation part contains only updating unknowns parameters which is very less. Transferring updated values to each process take more time. With increase number of processes calculation part is becoming less per process and communication cost is increased as all processes should have access to updated X array to check if x_i are within error rate. With problem size 1000 for processes 2, 10 and 20 a little speedup is observed but with processes 40 overhead overshadow the parallel computation cost.
So when we won't get speedup:
a. If overhead (process creation and communication cost) part weighs more than parallel computation benefits.

5.3 c

If problem size is sufficiently large that cost of process creation and communication can be overcome by parallel computation benefits. In the following case speed up is observed.
1. With problem size is 1000 run with 2, 10, and 20 processes.
2. If problem size is 10000 with any number of process given in the assignment.

5.4 d

When the work per process is large enough, that it can overcome the overhead incurred. As shown in the speedup table we can gained speed up when number of unknowns are 10000 irrespective on number of processes. While running with 40 processes maximum speedup is observed, 11.488. For the overall program, The only benefit we can get from parallelizing is the calculation part thus we have to gain performance on it.
So when we would get speedup:
a. For the calculation part, the work per process has to be large enough, so that the performance gain could cover the communication overhead and process creation cost between each iterations.
b. The calculation part has to weight enough to bring speed up to the whole program.

6 Machine Information

Above shown results are achieved on **Crunchy5** server and following is machine configuration retrived by running cat proc cpuinfo.

Siblings: 16
Model name: AMD Opteron(TM) Processor 6272
Number processors: 64
Cpu cores: 8
Bogomips: 4199.44
TLB size: 1536 4K pages
Clflush size: 64
Cache alignment: 64
Address sizes: 48 bits physical, 48 bits virtual
Power management: ts ttp tm 100mhzsteps hwpstate cpb

There are 64 logical processors present in the system. So, 64 threads can execute in parallel. For this problem statement we would not run more than 10 threads in parallel.

7 Miscellaneous

I evaluated my program with respect to reference file provided and it passes all the test cases outputting all the x_i within acceptable error range.
Load module MPI for program to work
module load mpi/openmpi-x86_64
Programming code is structured as follows
Each process reads only the specific part of the input file which is required for computation.
Iteration is divided almost equally amongst all the processes.
After each iteration updated **X** are transferred and received by all the processes by MPI **All_Gatherv** API
In last step, I evaluated if solution is reached otherwise iteration again.