COE 379L: Homework 9

Ashton Cole

April 3, 2024

1 Introduction

In this assignment, the performance improvement of *first touch* is investigated on a Jacobi matrix equation solution algorithm.

First touch refers to how data is physically stored in memory. When values are assigned to an allocated data structure, for example, a vector, those values are stored in memory physically closest to the process that initialized them. Thus, if initialization happens in parallel, data will be split, with a chunk of data close to each process. If those same chunks are then only used by the same respective processes, a small amount of time can be saved.

2 Methodology

The Jacobi algorithm is intended to solve a matrix system of equations $\mathbf{Ax} = \mathbf{b}$. In this assignment, it is specifically implemented for a tridiagonal matrix, with 1 on the off diagonals and 2.01 on the main diagonal.

The program is written in C using the OpenMP library. Two versions of the code are made. In one, `HW7.c`, vectors for the unkown, right hand side, and analytical solution are initialized in a serial for loop. In the other, `HW9.c`, the vectors are initialized in a parallel for loop.

The program accepts a single argument, the size of the matrix system of equatoins. All other variables are hard-coded, because the main goal is to evaluate parallel performance.

2.1 Jacobi Algorithm

The Jacobi algorithm is an iterative method to solve a matrix system of equations. Starting with an initial guess $\mathbf{x}^{(0)}$, additional iterations $\mathbf{x}^{(n)}$ are calculated using the following formula.

$$x_i^{(n+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(n)} \right) \tag{1}$$

This formula is applied elementwise in a static parallel for loop. Since the initialization loop is also a static for loop of the exact same size, iterations should be allocated between threads in the same way. This should mean that if the ith elements of the vectors are initialized on a certain process, this formula will be applied for that element on the same process.

2.2 Execution

Testing of the two programs was done on the Lonestar 6 supercomputer at the Texas Advanced Computing Center. A single node and then two nodes were used, each allowing up to 128 tasks. Environment variables were modified as follows.

- `OMP_PLACES=cores`: Threads are fixed to a specific core.
- `OMP_PROC_BIND=spread`: Threads are intentionally spread out across not only cores, but sockets. This makes them physically farther from one another, with the intention of maximizing the latency of pulling data from "other thread's" memory.

In addition, system size and number of threads are manipulated as independent variables to measure the dependent execution time.

3 Results

3.1 Single Node

Table 1: Initializing in serial, the time elapsed for various problem sizes and parallelizations on one Lonestar 6 node.

Vector Size	100	100,000	1,000,000	10,000,000
Threads	Elapsed Time (s)			
1	0.002	1.815	17.991	180.853
2	0.015	0.927	9.047	90.914
4	0.020	0.485	4.558	45.238
8	0.028	0.262	2.312	22.664
16	0.036	0.167	1.197	11.361
32	0.053	0.136	0.697	6.226
64	0.088	0.136	0.418	3.201
128	0.163	0.194	0.338	2.016

Table 2: Initializing in parallel, i.e. taking advantage of first touch, the time elapsed for various problem sizes and parallelizations on one Lonestar 6 node.

Vector Size	100	100,000	1,000,000	10,000,000
Threads	Elapsed Time (s)			
1	0.002	1.794	17.769	178.605
2	0.014	0.920	8.937	90.088
4	0.020	0.478	4.497	44.803
8	0.027	0.260	2.273	22.355
16	0.035	0.161	1.179	11.215
32	0.051	0.129	0.683	6.122
64	0.092	0.142	0.424	3.211
128	0.160	0.188	0.359	1.933

3.2 Two Nodes

Table 3: Initializing in serial, the time elapsed for various problem sizes and parallelizations on two Lonestar 6 nodes.

Vector Size	100	100,000	1,000,000	10,000,000
Threads	Elapsed Time (s)			
1	0.002	1.815	17.989	180.857
2	0.013	0.929	9.048	91.170
4	0.019	0.503	4.554	45.536
8	0.025	0.264	2.313	22.657
16	0.036	0.164	1.193	11.360
32	0.053	0.126	0.686	6.208
64	0.088	0.130	0.414	3.212
128	0.152	0.190	0.343	1.954

Table 4: Initializing in parallel, i.e. taking advantage of first touch, the time elapsed for various problem sizes and parallelizations on two Lonestar 6 nodes.

Vector Size	100	100,000	1,000,000	10,000,000
Threads	Elapsed Time (s)			
1	0.003	1.794	17.782	178.728
2	0.014	0.920	8.937	89.477
4	0.020	0.480	4.491	44.661
8	0.026	0.262	2.273	22.359
16	0.037	0.160	1.173	11.228
32	0.051	0.127	0.705	6.134
64	0.079	0.133	0.407	3.151
128	0.156	0.200	0.338	1.936

4 Conclusions

The first touch adjustment does not speed up execution time significantly, however, it does do so consistently. The only difference between the two codes is the use of first-touch, so this would suggest that a performance improvement is happening. The relative improvement is small, though, suggesting either that physical location is not a major determinant of the time to access data from memory, or that the calculations themselves take far more time.

Interestingly, the improvement still occurs for the serial case. It may be that OpenMP creates a single new serial thread on a different processor.