Assignment 2: Multithreading and Solving System of Linear Equations Algorithms

Tai Duc Nguyen
ECEC 622: Parallel Computer Architecture

May 6, 2020

Abstract

Knowledge from the previous assignment about multithreading's quirks and benefits is leveraged to develop a common algorithm used in solving system of linear equations called Iterative Gaussian Elimination.

1 Experimental Setup

Gaussian Elimination, also known as row reduction, is an algorithm in linear algebra for solving a system of linear equations. It is usually understood as a sequence of operations performed on the corresponding matrix of coefficients. This method can also be used to find the rank of a matrix, to calculate the determinant of a matrix, and to calculate the inverse of an invertible square matrix.

The multithreaded version of this algorithm is implemented as follows:

```
Initialize barrier sync

Create threads
Allocate memory on heap for threads' data

Start all threads. For each thread:
  for k = 0 -> n_rows:
    chunk_size = floor((n_cols-k-1)/num_threads);
    offset = tid*chunk_size + k + 1;
    start = offset;
    if tid < thread_data->num_threads - 1:
      finish = offset + chunk_size;
    else:
      finish = n_cols;

    for j = start -> finish:
      A[k,j] /= A[k,k];  // Division

    BARRIER SYNC

    for i = k + tid + 1 -> n_rows:
      for j = k + 1 -> n_cols:
        A[i,j] -= (A[i,k] * A[k,j]); // Elimination
      A[i,k] = 0;
```

```
25          i += num_threads; (striding method)
26
27        A[k,k] = 1;
28
29        BARRIER SYNC
30
31    Stop all threads
```

Note: running `make all` *will generate the report in* `rpt.txt`

2 Experimental Result

Matrix size (MxM)	Num Threads	Serial	Multithreading
512	4	0.00925	0.032711
512	8	0.007225	0.03313
512	16	0.007203	0.053966
512	32	0.009438	0.076227
1024	4	0.056045	0.141708
1024	8	0.058058	0.120481
1024	16	0.05552	0.132559
1024	32	0.055949	0.24562
2048	4	1.208647	0.791054
2048	8	1.233114	0.630565
2048	16	1.213595	0.484355
2048	32	1.214305	0.477111
4096	4	11.096719	7.261336
4096	8	11.617353	6.808351
4096	16	11.54519	6.696231
4096	32	11.086354	6.248624

Figure 1: Results of execution times in seconds of the serial vs the multi-threaded version using the pthread interface

3 Discussion

From Figure 1, it is apparent that multithreading provides significant improvement only when the problem size is larger than 1024 ($\approx$ 2.4x for 2048, and $\approx$ 2.3x for 4096). In the case where multithreading enhances the program, 32 threads will provide the best speed up. It is not understood yet why the speed up for 4096 is not significantly greater than 2048. This is hypothesized to be the effect of false sharing during the chunking part of the code. Improvements could be made to convert that part into striding instead.