Konstantin Zelmanovich, Claudio Del Valle A

Dr. Nagarajan Kandasamy

ECEC 413 – Introduction to Parallel Computer Architecture

Assignment 1

April 21, 2019

Part 1 - Pthreads: Gaussian Elimination

Below is a modified method for gaussian elimination using pthreads:

```
void
gauss_eliminate_using_pthreads (float *U_mt)
{
    unsigned int elements;

    for (elements = 0; elements < num_elements; elements++) // perform Gaussian elimination
    {
        pthread_t thread_arr[num_threads];
        struct my_struc* entry_point = malloc(num_threads * sizeof(struct my_struc));

        unsigned int i, j, k, m;
        for (i = 0; i < num_threads; i++)
        {
            entry_point[i].elements = elements;
            entry_point[i].id = i;
            entry_point[i].mat = U_mt;

            pthread_create(&thread_arr[i], NULL, row_reduction, (void *)&entry_point[i]);
        }

        for (j = 0; j < num_threads; j++)
        {
            pthread_join(thread_arr[j], NULL);
        }

        U_mt[num_elements * elements + elements] = 1;

        for (k = 0; k < num_threads; k++)
        {
            entry_point[k].elements = elements;
            entry_point[k].id = k;
            entry_point[k].mat = U_mt;

            pthread_create(&thread_arr[k], NULL, elimination, (void *)&entry_point[k]);
        }
```

```
        for (m = 0; m < num_threads; m++)
        {
            pthread_join(thread_arr[m], NULL);
        }

        free(entry_point);
    }
}
```

Below is a method for row reduction:

```
void *row_reduction(void *s)
{
    unsigned int idx_r;
    struct my_struc* myStruct = (struct my_struc*) s;
    int elements = myStruct->elements;
    int id = myStruct->id;
    float* U_mt = myStruct->mat;

    for (idx_r = elements+id+1; idx_r < num_elements;)
    {
         /* Chunking */
        float num = U_mt[num_elements * elements + idx_r];
        float denom = U_mt[num_elements * elements + elements];
        float div_step = num / denom;
        U_mt[num_elements * elements + idx_r] = div_step;

        idx_r = idx_r + num_threads;
    }

    pthread_exit(0);
}
```

Below is a method for the actual elimination:

```
void *elimination(void *s)
{
    struct my_struc* myStruct = (struct my_struc*) s;
    int elements = myStruct->elements;
    int id = myStruct->id;
    float* U_mt = myStruct->mat;
    unsigned int  idx_el_1, idx_el_2;

    for (idx_el_1 = (elements + id)+1; idx_el_1 < num_elements; )
    {
        for (idx_el_2 = elements+1; idx_el_2 < num_elements; idx_el_2++)
        {
            float first_part = U_mt[num_elements * idx_el_1 + idx_el_2];
```

```
            float last_part = (U_mt[num_elements * idx_el_1 + elements] * U_mt[num_elements *
    elements + idx_el_2]);
            float elim_step = first_part - last_part;
            U_mt[num_elements * idx_el_1 + idx_el_2] = elim_step;
        }

        U_mt[num_elements * idx_el_1 + elements] = 0;
        idx_el_1 = idx_el_1 + num_threads;
  }
  pthread_exit(0);
}
```

Table 1: Timing results

# of Threads	Matrix Size	Serial Time (s)	Parallel Time (s)	Speedup
4	512	0.5	0.3175	1.574
4	1024	3.45	1.3264	2.601
4	2048	28.72	9.3806	3.061
8	512	0.49	0.435	1.126
8	1024	3.44	1.5364	2.239
8	2048	28.12	6.2903	4.47
16	512	0.46	0.6873	0.669
16	1024	3.6	1.7892	2.012
16	2048	28.74	8.5025	3.38
32	512	0.49	1.4382	0.34
32	1024	3.57	3.097	1.152
32	2048	28.94	13.2548	2.183

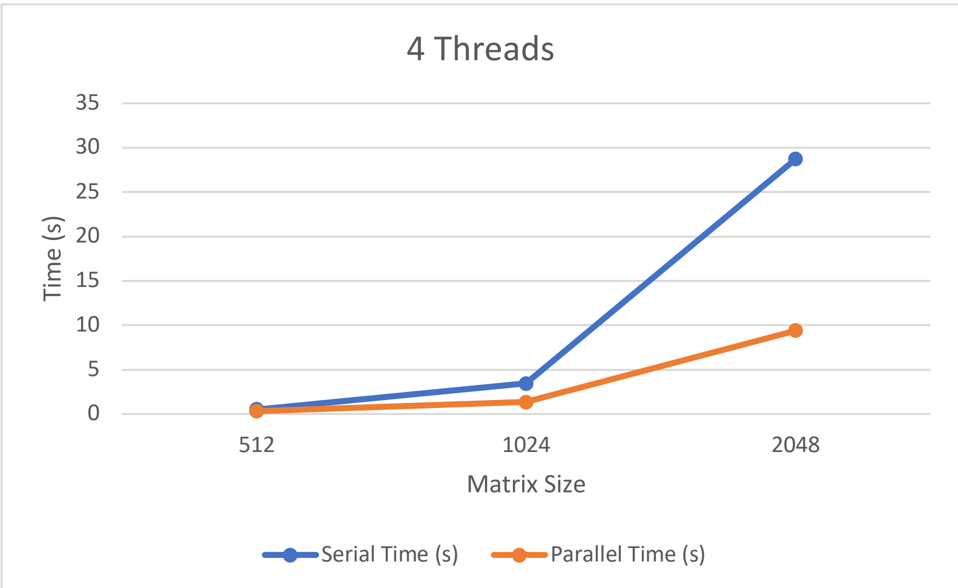

Figure 1: Time vs Matrix Size using 4 Threads

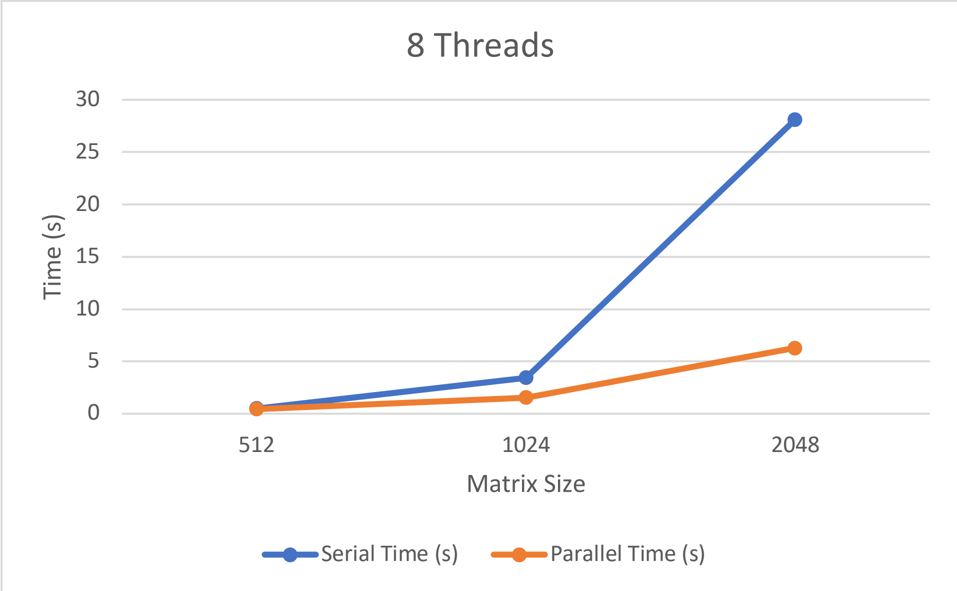

Figure 2: Time vs Matrix Size using 8 Threads

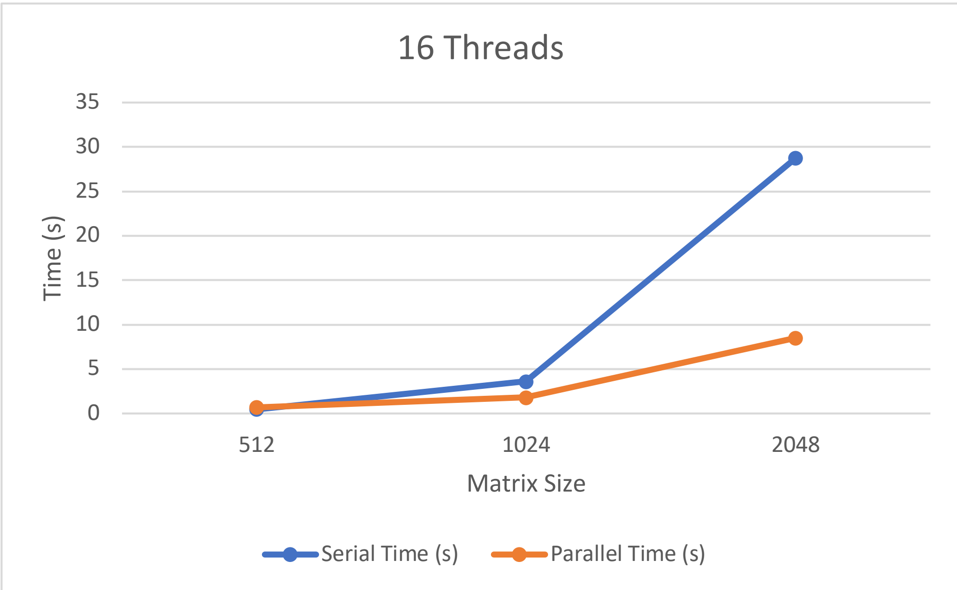

Figure 3: Time vs Matrix Size using 16 Threads

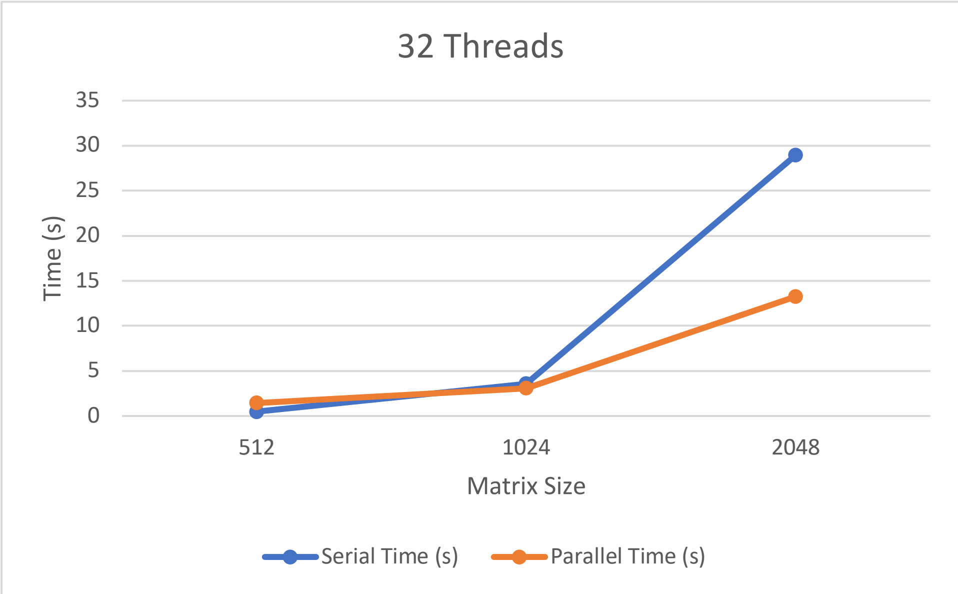

Figure 4: Time vs Matrix Size using 32 Threads