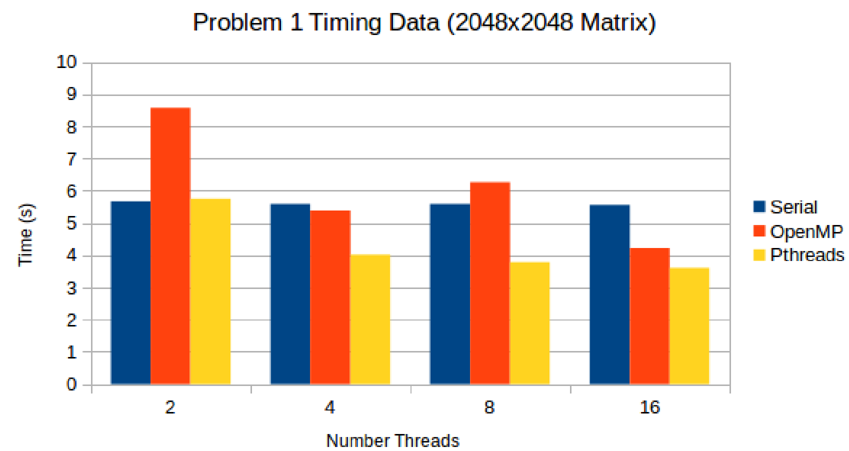
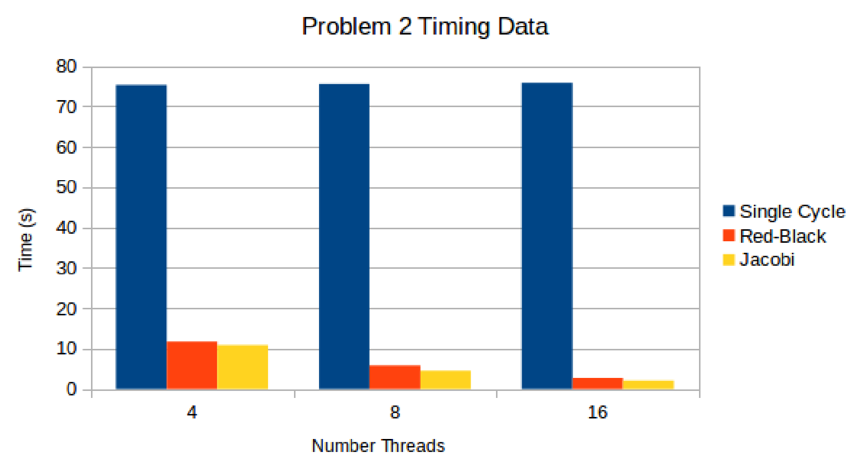
Parallel Computing Midterm

By: Alex Kazantsev and Hanumant Mahida

Date: February 19th, 2017

Results





Problem 1

Note: Compile using gcc -fopenmp -o chol chol.c chol_gold.c -lpthread -lm -std=gnu99

Timing data

Matrix Size	Number of Threads	Serial Time (s)	OpenMP (s)	PThreads (s)	Serial vs OpenMP	Serial vs PThreads
512	2	0.10	0.15	0.10	0.67	1
512	4	0.09	0.10	0.09	0.9	1
512	8	0.09	0.13	0.08	0.69	1.125

Matrix Size	Number of Threads	Serial Time (s)	OpenMP (s)	PThreads (s)	Serial vs OpenMP	Serial vs PThreads
512	16	0.10	0.11	0.09	0.91	1.11
1024	2	0.72	1.09	0.72	0.66	1
1024	4	0.69	0.70	0.50	0.98	1.38
1024	8	0.71	1.09	0.47	1.06	1.37
1024	16	0.70	0.66	0.51	1.06	1.37
2048	2	5.67	8.58	5.76	0.66	0.98
2048	4	5.59	5.39	4.02	1.04	1.39
2048	8	5.59	6.27	3.78	0.89	1.47
2048	16	5.56	4.22	3.61	1.31	1.54

Cholesky Decomposition (PThreads)

Barrier Data Structure Initialized

```
struct s1* para = (struct s1*) malloc(num_threads *  sizeof(struct s1));
for (i = 0; i < num_threads; i++)
{
  para[i].mat = U.elements;
  para[i].id = i;
  // creating num_threads pthreads
  pthread_create(&threads[i], NULL, pthread_wrapper, (void *)&para[i]);
}
```

Main PThread Row Loop

```
for(row = 0; row< num_elements; row++)
{
  row_number = row;
  U.elements[row * U.num_rows + row] = sqrt(U.elements[row * U.num_rows + row]);
  pthread_barrier_wait(&barrier_main);
  pthread_barrier_wait(&barrier_main);
}
```

PThread Wrapper to handle functions rowReduction and eliminationStep with barriers to make sure data is in sync

```
void* pthread_wrapper(void* s)
{
    struct s1* myStruct = (struct s1*) s;
    while(row_number < MATRIX_SIZE)
    {
        pthread_barrier_wait(&barrier_main);
        rowReduction(s);
        pthread_barrier_wait(&barrier_threads);
        eliminationStep(s);
        pthread_barrier_wait(&barrier_threads);
        if (row_number == MATRIX_SIZE-1)
            setZeroes(s);

        pthread_barrier_wait(&barrier_main);
    }
    pthread_exit(0);
}
```

Note: rowReduction and eliminationStep can be found in midterm/cholesky/chol.c

Cholesky Decomposition (OpenMP)

OpenMP used to parallelize algorithm. OMP Parallel and barrier's used to ensure Cholesky decomposition takes place in place on the U Matrix

```
	#pragma omp parallel default(none) shared(U) private(k, i, j)
	{
		for(k = 0; k < U.num_rows; k++){
			 // Take the square root of the diagonal element
			# pragma openmp master
			U. elements [k * U.num_rows + k] = sqrt(U.elements [k * U.num_rows + k]);

			# pragma openmp barrier

			 // Division step
			 for(j = (k + 1)+omp_get_thread_num (); j < U.num_rows; j += num_threads)
					U. elements [k * U.num_rows + j] /= U.elements [k * U.num_rows + k]; // Di
			# pragma openmp barrier

			 // Elimination step
			 for(i = (k + 1)+omp_get_thread_num (); i < U.num_rows; i += num_threads)
					for(j = i; j < U. num_rows; j++)
						U.  elements [i * U.num_rows + j] -= U.elements [k * U.num_rows

			# pragma openmp barrier
		}
	}
```

Problem 2

Note: Compile using gcc -o solver solver.c solver_gold.c -fopenmp -std=c99 -lm -lpthread

Timing data

Matrix Size	Number of Threads	Serial Time (s)	Red-Black (s)	Jacobi (s)	Serial vs Red-Black	Serial vs Jacobi
8192	4	75.36	11.73	10.89	6.42	6.92
8192	8	75.60	5.89	4.51	12.84	16.76
8192	16	75.86	2.79	2.16	27.19	35.12

Red-Black Decomposition

Call OpenMP parallel on while loop

```
   #pragma omp parallel
	for (int i = 1; i < (grid_2->dimension- 1); i++)
	{
	    ....
	}
```

Computing X (Red) for decomposition

```
  // Compute "X" (odd)
	xStart = (id* 2) + i%2;
	#pragma omp for reduction(+: diff_x) private(temp, j, xStart)
	for (int j = xStart; j < (grid_2->dimension- 1); j+=num_threads)
	{
	    temp = grid_2->element[i * grid_2->dimension + j];
```

```
            grid_2->element[i * grid_2->dimension + j] =  0.20*(grid_2->element[i * grid_2->dimension + j] +
                    grid_2->element[(i -  1) * grid_2->dimension + j] +
                    grid_2->element[(i +  1) * grid_2->dimension + j] +
                    grid_2->element[i * grid_2->dimension + (j +  1)] +
                    grid_2->element[i * grid_2->dimension + (j -  1)]);
            diff_x = diff_x +  fabs(grid_2->element[i * grid_2->dimension + j] - temp);
        }
```

Computing Y (Black) for decomposition

```
    // Compute "Y" (even)
        yStart = id* 2 + (1 - id%2);
        #pragma omp for reduction(+: diff_y) private(temp, j, yStart)
        for (int k = yStart; k < (grid_2->dimension- 1); k+=num_threads)
        {
            temp = grid_2->element[i * grid_2->dimension + j];
            grid_2->element[i * grid_2->dimension + j] =  0.20*(grid_2->element[i * grid_2->dimension + j] +
                    grid_2->element[(i -  1) * grid_2->dimension + j] +
                    grid_2->element[(i +  1) * grid_2->dimension + j] +
                    grid_2->element[i * grid_2->dimension + (j +  1)] +
                    grid_2->element[i * grid_2->dimension + (j -  1)]);
            diff_y = diff_y +  fabs(grid_2->element[i * grid_2->dimension + j] - temp);
        }
```

Element-Based Decomposition

Using Reduction to parallelize data and also using a copy of the grid

```
        while(!done){
            diff = 0;
            #pragma omp parallel for reduction(+: temp)
            for (int i = 1; i < (grid_3->dimension- 1); i++)
            {
                int id = omp_get_thread_num ();
                for (int j = id; j < (grid_3->dimension- 1); j+=num_threads)
                {
                    temp = grid_3->element[i * grid_3->dimension + j];
                    grid_3->element[i * grid_3->dimension + j] =  0.20*(grid_3_copy->element[i * grid_3_copy->
                            grid_3_copy->element[(i -  1) * grid_3_copy->dimension + j] +
                            grid_3_copy->element[(i +  1) * grid_3_copy->dimension + j] +
                            grid_3_copy->element[i * grid_3_copy->dimension + (j +  1)] +
                            grid_3_copy->element[i * grid_3_copy->dimension + (j -  1)]);
                    diff = diff +  fabs(grid_3->element[i * grid_3->dimension + j] - temp);
                }
            }
```