# Assignment 2

Siddhant S. Aphale,
Department of Mechanical and Aerospace Engineering,
University at Buffalo,
Person # - 50164327

October 20, 2016

## Problem 1

The matrix addition problem with matrix size 20000 each composed of random numbers from 0 to 9 was used for performance profiling using `gprof`. Both the loop orders were studied viz., looping over rows first and the column as well as looping over column first and then rows. The profiling results obtained over these two cases are stored in `row.txt` and `col.txt` respectively.
Initially when the matrix addition was looped over rows first, we can see that the main function spent 8.00 seconds. Each sample hit covered 2 byte(s) for 0.08% of 12.58 seconds.
Next, the profile performance was recorded with column loop first. It can be seen in `col.txt` that the main function spent 65.47 seconds. It was 93.48% of the total execution time. Each sample hit covers 2 byte(s) for 0.01% of 70.05 seconds.
However, we can see in both the results that `gprof` does not provide any other information about cache misses, floating point operations, etc. Thus, we use PAPI library to study these details with both the looping orders as above. The PAPI results are summarized in the below tables. It can be observed that the number of cache misses in column wise addition are more compared to row wise addition. As the code is in C++, the data is row oriented and row wise strides of data in loaded in the cache.
**Row loop first**

| Matrix Size | L1 cache misses | L2 cache misses | Floating Point Operations | MFLOPs |
|---|---|---|---|---|
| 10000 | 33033533 | 8240975 | 103229315 | 182.377 |
| 20000 | 130580296 | 37076680 | 414703941 | 178.976 |

**Column loop first**

| Matrix Size | L1 cache misses | L2 cache misses | Floating Point Operations | MFLOPs |
|---|---|---|---|---|
| 10000 | 727673749 | 605958490 | 100325281 | 13.2084 |
| 20000 | 2911644213 | 2838664618 | 402006984 | 9.69 |

The results obtained are in `.dat` files.

## Problem 2

In this problem, a code was developed to find out the area of the Mandelbrot set. The grid was generated with varying grid sizes, 100, 500, 1000 and 1500 grid points. The area of the Mandelbrot set was found out to be 1.50734 when the grid was $500 \times 500$. The code performance was studied using PAPI and the results are mention in the table below.

| Grid Size | Time | Area | L1 Cache Misses | L2 Cache Misses | Floating Point Operations | MFLOPs |
|---|---|---|---|---|---|---|
| 100 | 0.757349 | 1.48883 | 14351 | 1530 | 888647536 | 1173.5 |
| 500 | 19.1549 | 1.50734 | 454042 | 70260 | 23729860493 | 1192.69 |
| 1000 | 76.7116 | 1.50599 | 1838824 | 350202 | 115222928956 | 1192.86 |
| 1500 | 172.791 | 1.50684 | 4443980 | 836718 | 321320445228 | 1192.92 |

Now the code was parallelized using OpenMP. Three types of scheduling were experimented, static, dynamic and guided. The chunk size was set to 50 in all the three scheduling strategies. Strong scaling was studied for all the three scheduling strategies. The speedup factor was computed as defined by Amdahl's Law. This was studied for the grid size of 500. CPU constraint was set to `CPU-E5-2660` which has 16 Cores/Node. Time taken to calculate the area of Mandelbrot set was studied first with serial code and then with varying threads from 1 to 16. Figure 1 shows the plot for speedup factor vs the number of threads.

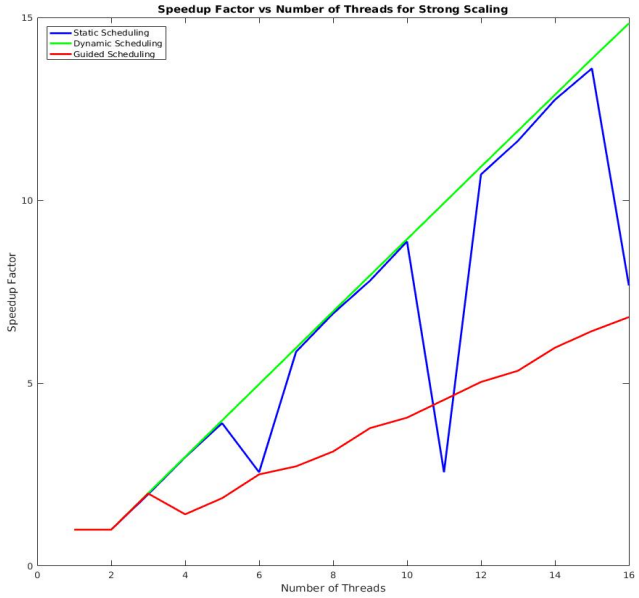


Figure 1: Speedup Factor vs Number of Threads-Mandelbrot Set

Furthermore, the Karp-Flatt serial fraction metric was calculated. The serial fraction metric as defined by Karp-Flatt is given as

$$f = \frac{1/s - 1/p}{1 - 1/p} \tag{1}$$

where, $s$ is the Speedup factor, $p$ is the number of processors/threads. The speedup factors were calculated for three scheduling strategies as discussed above. The Karp-Flatt serial fraction metric was computed for all these three experiments. The plot was generated for these three strategies as in Figure 2

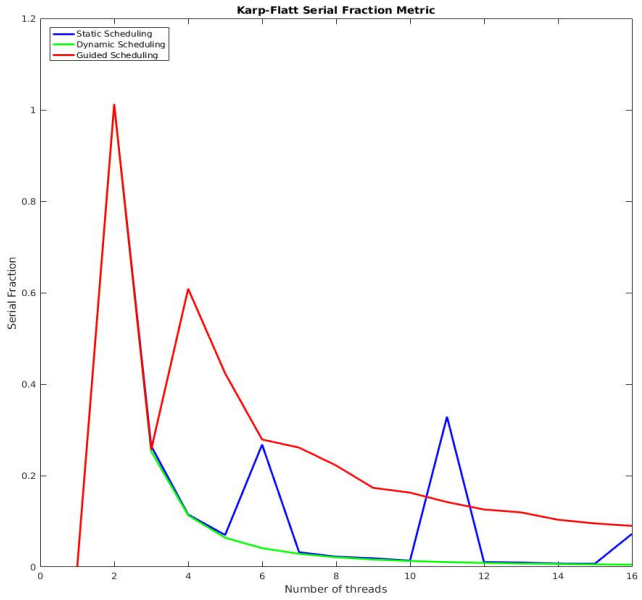


Figure 2: Serial Fraction Metric vs Number of Threads-Mandelbrot Set

## Problem 3

This code was developed to verify the Goldbach conjecture. In this code, the Goldbach conjecture was verified for various numbers 100, 1000, 10000, 100000, 5000000. It was observed that the conjecture holds true for even number upto 5000000. The performance of the code was studied using PAPI. The results obtained are listed below.

| Number | Time | L1 Cache Miss | L2 Cache Miss | Floating Operations | MFLOPs |
|---|---|---|---|---|---|
| 100 | 0.0071148 | 204 | 103 | 35 | 1.75 |
| 1000 | 0.00711267 | 223 | 89 | 35 | 1.75 |
| 10000 | 0.00711429 | 205 | 74 | 31 | 1.55 |
| 100000 | 0.00707558 | 198 | 77 | 33 | 1.5714 |
| 5000000 | 0.00702359 | 208 | 96 | 38 | 1.80952 |

Next the code was parallelized with OpenMP. Various scheduling strategies, static, dynamic and guided scheduling were implemented. The number checked in this experiment was 1000000. The chunk size was set to 10000. Strong scaling was studied for these three scheduling strategies. The speedup factor was computed for these three strategies and was plotted as in Figure 3

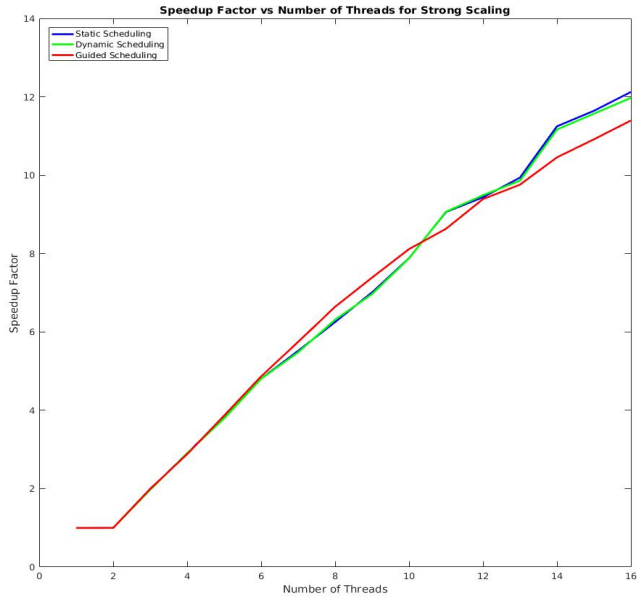


Figure 3: Speedup Factor vs Number of Threads-Goldbach Conjecture

It can be observed from the plot that the code achieves linear speedup till 16 processors.
Karp-Flatt serial fraction was also computed for the above three strategies. It was plotted against the number of threads as in Figure 4

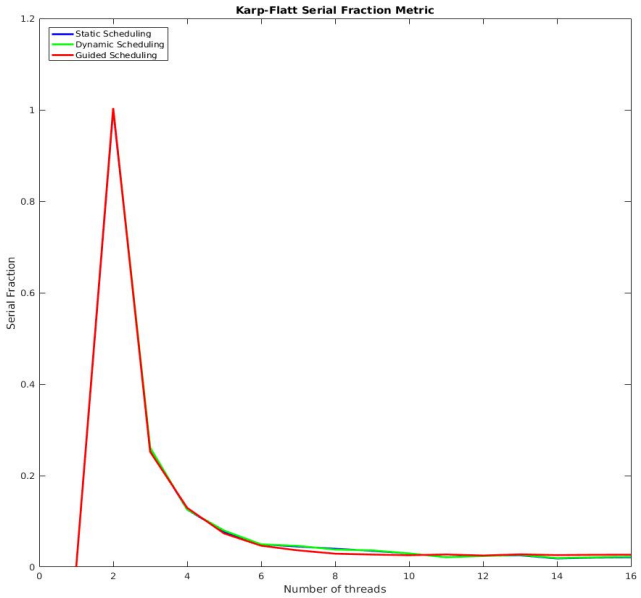


Figure 4: Serial Fraction Metric vs Number of Threads-Goldbach Conjecture

## Problem 4

In this problem, the value of `pi` was estimated using the midpoint rule. The value of `pi` was calculated with different number of iterations from $1 \times 10^9, 2 \times 10^9, 3 \times 10^p$. The performance of the code was studied using PAPI and is summarized below.

| Iterations | L1 Cache Misses | L2 Cache Misses | Floating Point Operations | MFLOPs |
|---|---|---|---|---|
| $1 \times 10^9$ | 170 | 56 | 4941023062 | 727.342 |
| $2 \times 10^9$ | 166 | 53 | 9883535372 | 727.975 |
| $3 \times 10^9$ | 190 | 91 | 14947490864 | 733.523 |

Now the code was parallelized using OpenMP. Scheduling strategies like static, dynamic and guided were studied. This code was run on 12 processors. Strong scaling was studied by calulating the speedup factor for each strategy and was plotted against the number of processors as shown in Figure 5

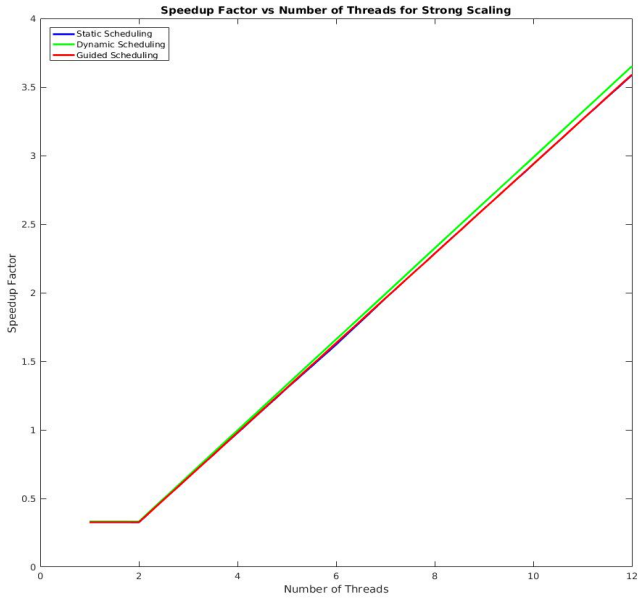


Figure 5: Speedup Factor vs Number of Threads-Pi Value Estimate

Furthermore, the serial fraction was computed based on Karp-Flatt proposed metric. The plot for this metric is as shown in Figure 6

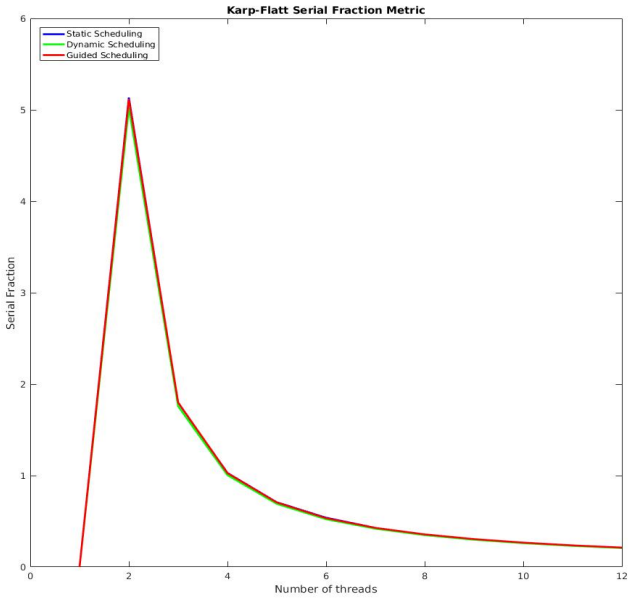


Figure 6: Serial Fraction Metric vs Number of Threads-Pi Value Estimate

It can be observed that the scheduling strategies do not make much difference in this problem.