High Performance Computing

Sagar Bhatt
Person Number: 50170651
Department of Mechanical and Aerospace Engineering,
University at Buffalo

I. Matrix Addition Performance Study

I. Profiling performance using gprof:

Looping over rows first:

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 61.32      2.44     2.44        1     2.44     2.44  main
 38.70      3.98     1.54                             __intel_ssse3_rep_memmove
  0.00      3.98     0.00    30000     0.00     0.00  std::vector<double,
   std::allocator<double> >::vector(std::vector<double, std::allocator<double> > const&)
  0.00      3.98     0.00        3     0.00     0.00  std::vector<std::vector<double, std::allo
   std::allocator<std::vector<double, std::allocator<double> > > >::vector(unsigned long, std::
  0.00      3.98     0.00        3     0.00     0.00  std::vector<double, std::allocator<double
   std::allocator<double> const&)
  0.00      3.98     0.00        2     0.00     0.00  get_time
  0.00      3.98     0.00        1     0.00     0.00  diff
```

Looping over columns first:

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 91.68     16.58    16.58        1    16.58    16.58  main
  8.35     18.09     1.51                             __intel_ssse3_rep_memmove
  0.00     18.09     0.00    30000     0.00     0.00  std::vector<double, std::allocator<double
  0.00     18.09     0.00        3     0.00     0.00  std::vector<std::vector<double, std::allo
  0.00     18.09     0.00        3     0.00     0.00  std::vector<double, std::allocator<double
  0.00     18.09     0.00        2     0.00     0.00  get_time
  0.00     18.09     0.00        1     0.00     0.00  diff
```

II. Instrumenting the code with PAPI

Looping over rows first:

```
Time taken when looping over rows first:0.850615

The total instructions executed for addition are 600666184
The total cycles used are 1247329260
The total cache misses are 37982311

Real_time: 5.974195
Proc_time: 5.960402
Total flops: 303715595
MFLOPS: 50.955555
```

Looping over columns first:

```
Time taken when looping over columns first:14.9937

The total instructions executed for addition are 1000861067
The total cycles used are 30122730315
The total cache misses are 723035278

Real_time: 18.859709
Proc_time: 18.812635
Total flops: 300962949
MFLOPS: 15.997914
```

II. Mandlebrot Set

Estimated area of the Mandlebrot set: 1.5095
Resolution: $\frac{4}{1000}$ for comparing different schedulings. For the plot, resolution was $\frac{4}{10000}$

Iterations: 10000
$Real\ Axis_{max} = 2$
$Real\ Axis_{min} = -2$
$Imaginary\ Axis_{max} = 2$
$Imaginary\ Axis_{min} = -2$

Code Performance:

```
Time without omp: 5.99114 s
Area of the mandlebrot set =1.5095
Read successfully
```

```
The total instructions executed are 20876783386
The total cycles used are 14252196016
The total cache misses are 274536
```

Plot:

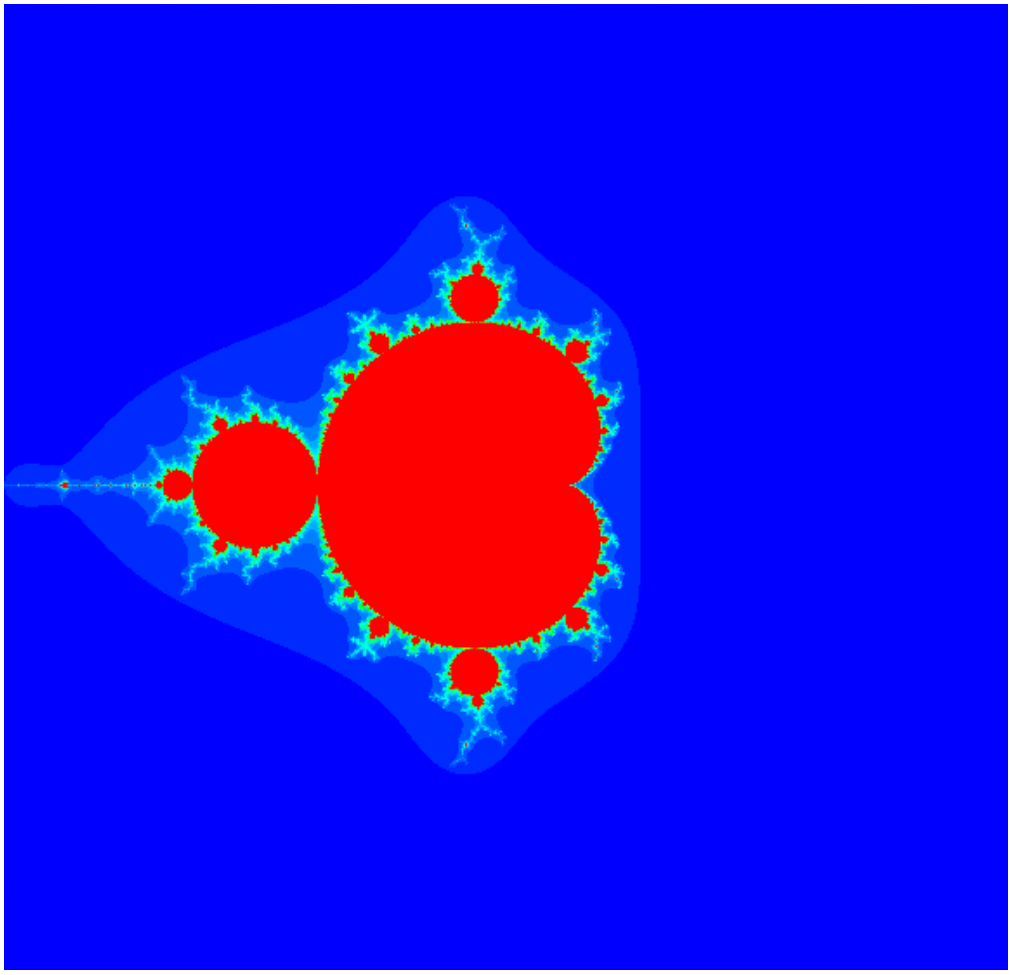

Figure 1: *Plot of Mandlebrot Set*

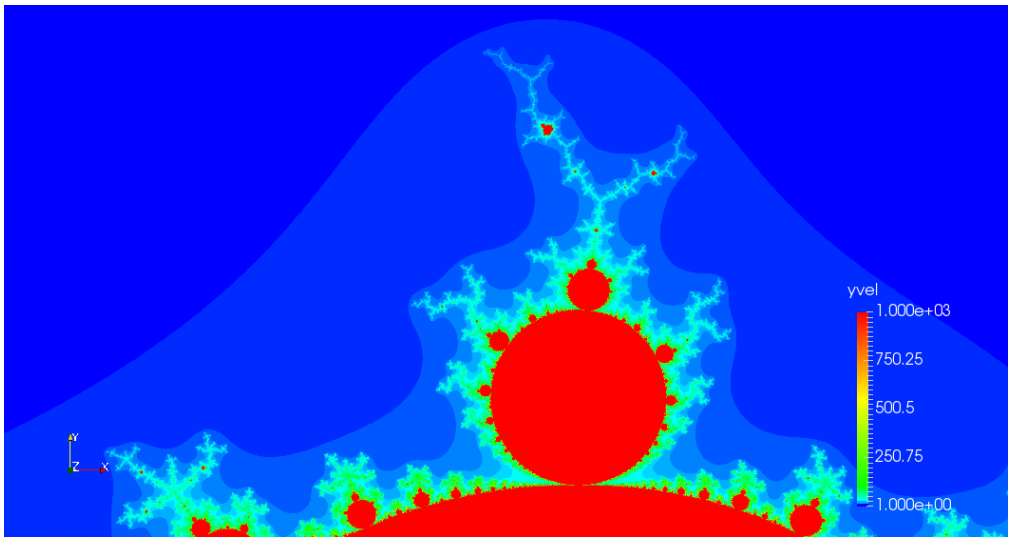

Figure 2: *Fringes of the mandlebrot set*

Speedup was computed using the formula:

$$s = \frac{T(1)}{T(p)}$$

The serial fractions were computed using the Karp-Flatt metric: $f = \dfrac{1/s - 1/p}{1 - 1/p}$

Serial fractions for different scheduling:

Threads	Static	Dynamic	Guided
1	NA	NA	NA
2	0.502304	0.502482	0.507148
3	0.947342	0.34342	0.706208
4	0.497382	0.25828	0.552642
5	0.70475	0.219027	0.393479
6	0.477003	0.208744	0.379274
7	0.551515	0.210188	0.324986
8	0.417825	0.208535	0.267883
9	0.450383	0.208958	0.258204
10	0.357674	0.208973	0.231806
11	0.376339	0.208549	0.213792
12	0.311293	0.208032	0.212929

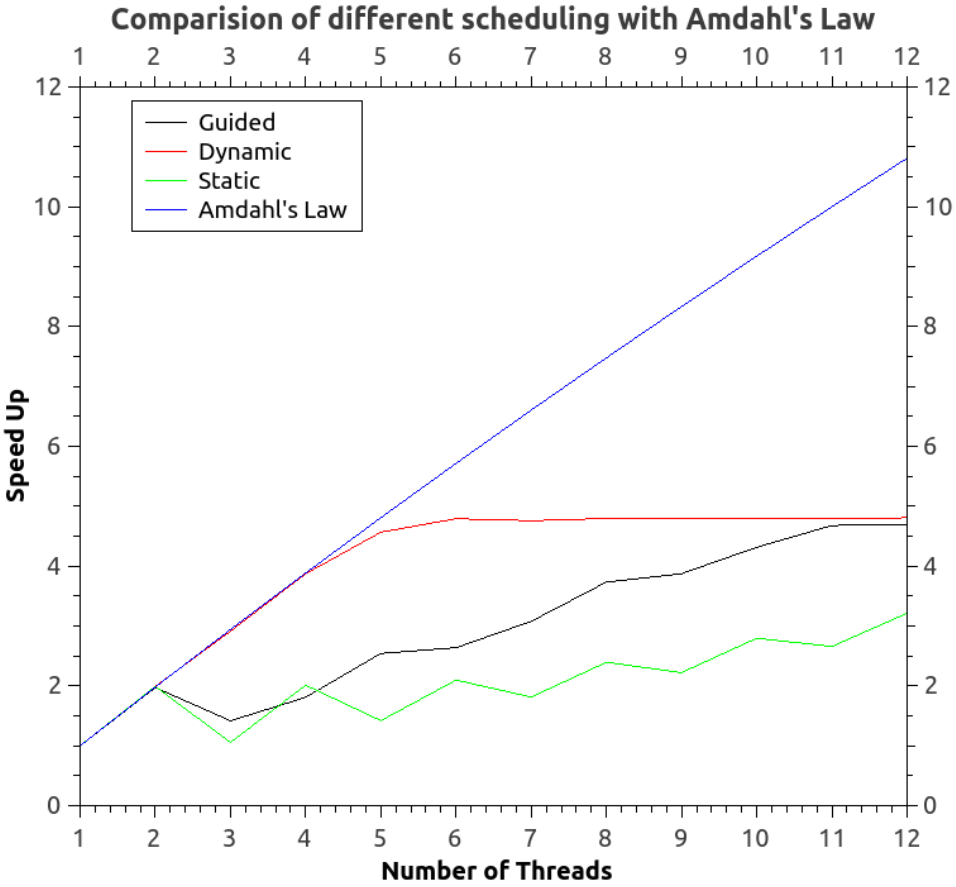

Figure 3: *Comparison of scheduling with Amdahl's law*

III. Goldbach Conjecture

A code was written to verify that Goldbach conjecture was true upto any arbitrary number of integers. For this study, the code was used to verify the conjecture for all even numbers less than 1000.

Code Performance:

```
The total instructions executed are 51546
The total cycles used are 126342
The total cache misses are 1277
Total flops: 672
MFLOPS: 2.507463
```

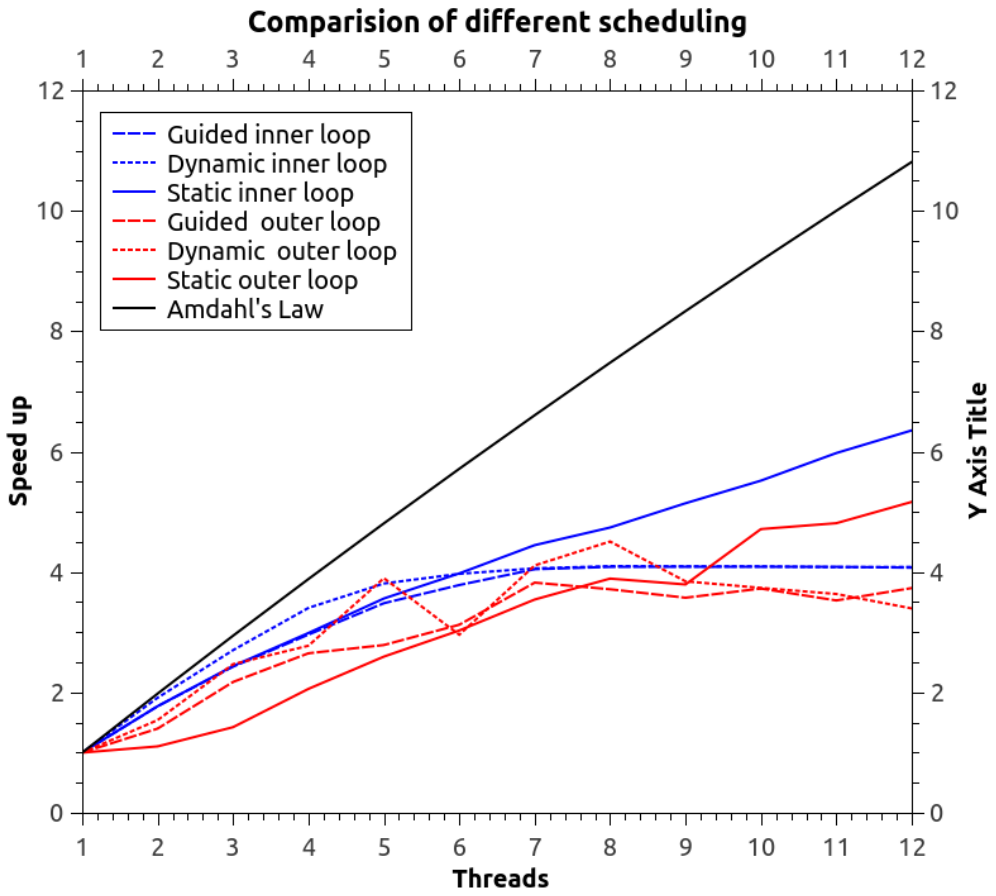

Figure 4: *Comparison of scheduling with Amdahl's law*

Serial fractions for different scheduling:

Parallelizing outer loop:

Threads	Static	Dynamic	Guided
1	NA	NA	NA
2	0.906107	0.648986	0.714653
3	0.702961	0.405036	0.460048
4	0.485634	0.360256	0.377353
5	0.385431	0.256561	0.358821
6	0.330181	0.338079	0.320094
7	0.282251	0.243374	0.261548
8	0.257039	0.221895	0.269296
9	0.263636	0.260092	0.279905
10	0.212191	0.267685	0.268455
11	0.207875	0.275105	0.283498
12	0.193629	0.294559	0.26786

Parallelizing inner loop:

Threads	Static	Dynamic	Guided
1	NA	NA	NA
2	0.564533	0.523178	0.564874
3	0.410179	0.370116	0.411896
4	0.334757	0.293796	0.33783
5	0.280915	0.262536	0.287219
6	0.251457	0.251904	0.264308
7	0.224929	0.246262	0.247159
8	0.211026	0.244044	0.244788
9	0.194478	0.244254	0.244667
10	0.181242	0.244281	0.244685
11	0.167339	0.245056	0.244719
12	0.157428	0.244818	0.245595

Inference: It is evident from the aforementioned results, that parallelizing inner loop has resulted in a much lower serial fraction hence a much better speed up. This can be also observed in the plot where the general trend for all schedulings shows that speed up is higher when inner loop is parallelized.

IV. Calculating π

A program was written to compute π using midpoint rule.

Result:

```
Static Scheduling:
Pi=3.14159305347414097
Time taken with static scheduling on 12 threads: 0.510051985000000041 s

Dynamic Scheduling:
Pi=3.14159305347418005
Time taken with dynamic scheduling on 12 threads: 0.588054434999999986 s

Guided Scheduling:
Pi=3.14159305347416096
Time taken with guided scheduling on 12 threads: 0.565029471000000005 s
```

Code Performance:

```
The total instructions executed are 12000071835
The total cycles used are 54511050117
The total cache misses are 4035
Total flops: 9500601275
MFLOPS: 415.487762
```

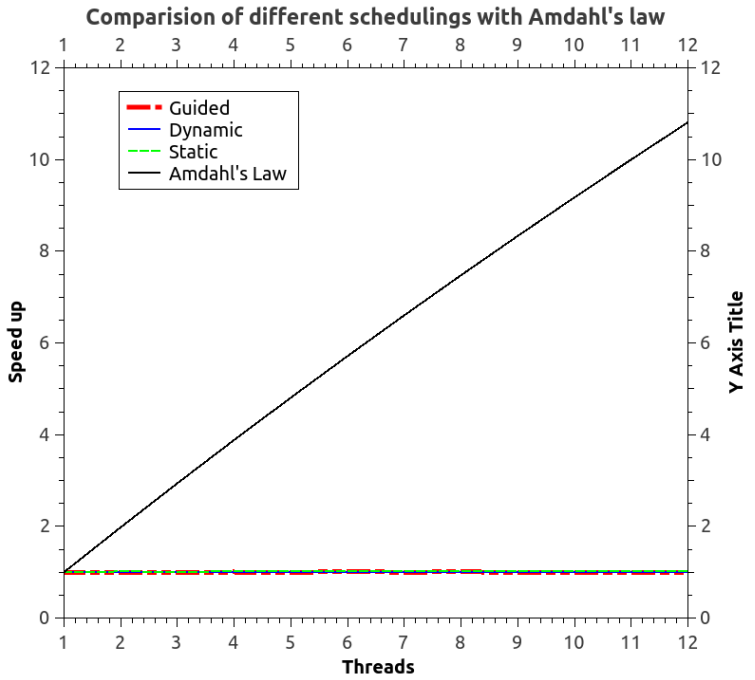

Figure 5: *Comparison of scheduling with Amdahl's law*

The above plot does not show a lot of difference between the schedulings since the speedup was not good enough. The following plot shows a comparison among different strategies :

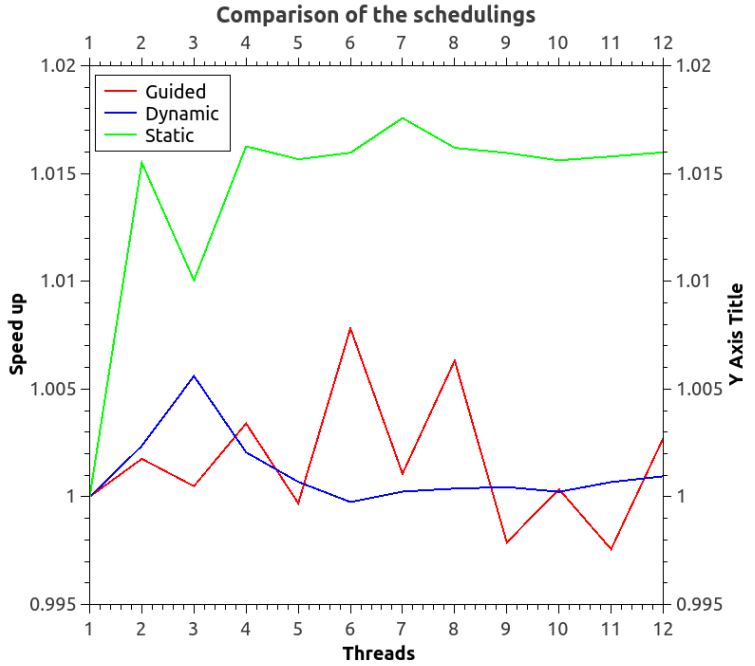

Figure 6: *Comparison of scheduling*

Although there was not much of a difference between different schedulings, the above plot shows that there was a little (though insignificant) better speedup with static scheduling.

Serial fractions for different scheduling:

Threads	Static	Dynamic	Guided
1	NA	NA	NA
2	0.984737	0.997641	0.998241
3	0.99006	0.994427	0.999511
4	0.983998	0.99794	0.996614
5	0.984577	0.999306	1.0003
6	0.984287	1.00024	0.992252
7	0.982735	0.999757	0.998932
8	0.984064	0.99962	0.993718
9	0.984305	0.999552	1.00212
10	0.984625	0.999767	0.999647
11	0.984454	0.999316	1.00244
12	0.984267	0.999049	0.997283