Comparison of Multi-threading between C++ and Rust (OpenMP vs Rayon/Crossbeam)

Project Checkpoint 1

Authors:
Nishal Ancelette Pereira <napereir@andrew.cmu.edu>
Supradeep Rangarajan <strangar@andrew.cmu.edu>

Summary

At this point, we have compared and performed preliminary analysis on the multithreading performance on a simple load imbalance workload (Mandelbrot) utilizing two multithreading libraries, Rayon and Crossbeam, which are implemented specifically for Rust to that of OpenMP implementation for C++.

Progress

We were successfully able to implement multithreaded version of Mandelbrot using OpenMP in C++ and using both Crossbeam and Rayon in Rust. We have finished the implementation of serial version of Matrix multiplication in Rust as well. We initially had some issues with the non-standard version of Rust installed on the GHC machines. Therefore, we had to resort to cross-compilation of the code written in Rust

After overcoming the learning curve of Rust, we started out by implementing a serial version of Mandelbrot. We took an object dump of the C++ and Rust code to make sure that they are using the same assembly instructions and that one version of the code had no unfair advantage over the other. After establishing our baseline, we implemented the multithreaded versions of the code in 2 styles. In one version, we ran the threads parallel over the rows and in the other version, we ran the threads parallel over the pixels of the image. We performed analysis by varying the number of threads from 2, 4, 6, 8, 12 to 16 on the GHC machines. Additionally, we ran experiments by spawning just a single thread to quantify the overhead of thread creation for both Rayon and OpenMP adding timing code for profiling.

Schedule

The progress made so far is according to the schedule proposed earlier and we would not be making any adjustments to the original schedule and project goals.

We have finished benchmarking of Mandelbrot using OpenMP, Rayon and Crossbeam. We have also started the implementation of Matrix multiplication for both languages. Since we have overcome the learning curve of Rust and have written the testing scripts already, we should be able to make faster progress with other benchmarks.

Towards the final week we want to spare more time to perform deeper analysis on the results obtained and learn some of the behavior of Rayon.

Preliminary Results

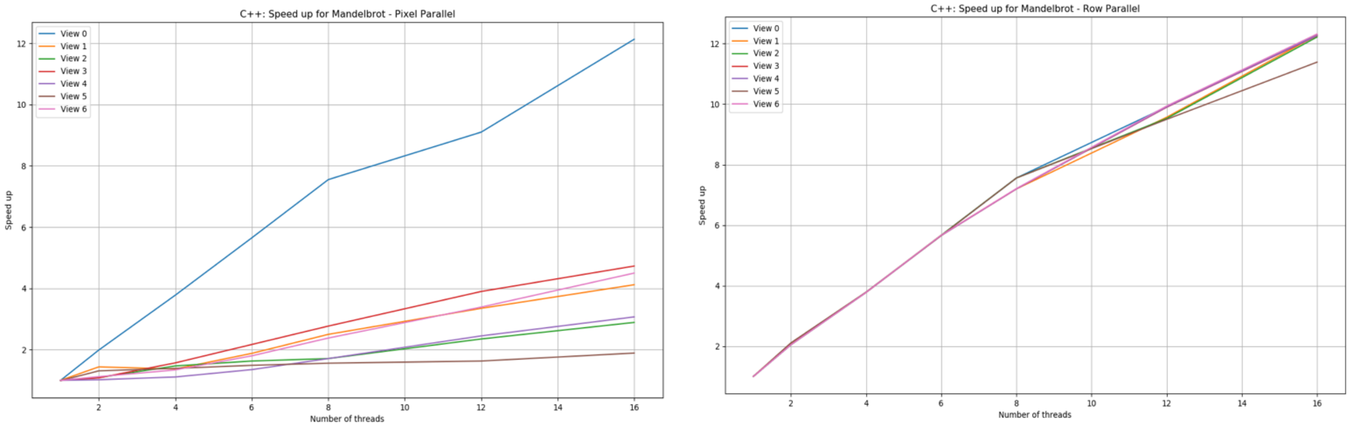

Figure 1: Speedup obtained on different views of Mandelbrot implemented using OpenMP for C++ for (a) Threads running parallel over the pixels (b) Threads running parallel over rows of the image

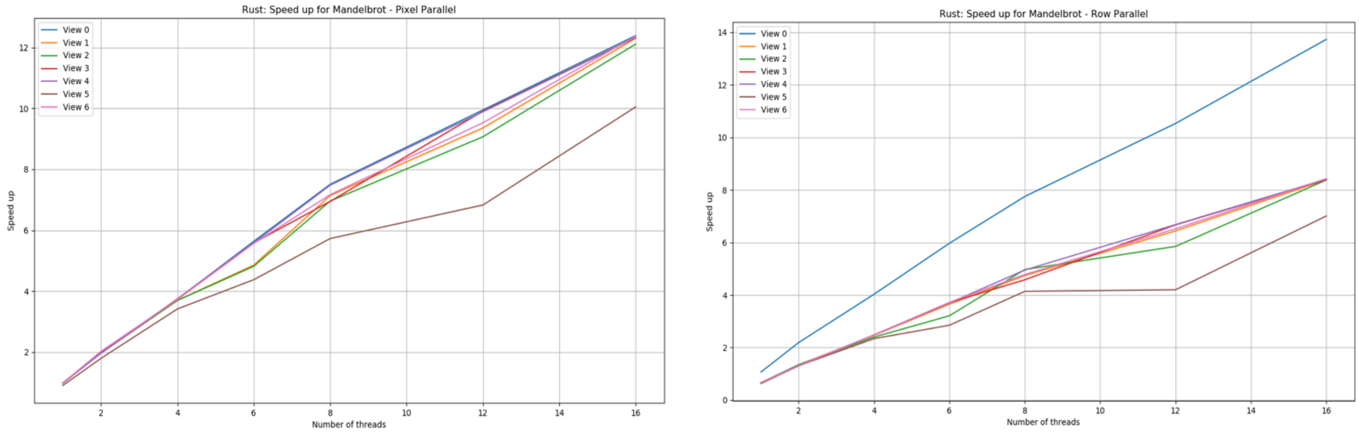

Figure 2: Speedup obtained on different views of Mandelbrot implemented using Rayon in Rust for (a) Threads running parallel over the pixels (b) Threads running parallel over rows of the image

Figure 1 and Figure 2 show the speedup that was obtained on different views of Mandelbrot for threads ranging from 2 to 16 for both C++ and Rust implementations. It is evident that Rust does not perform all that poorly in comparison to C++ version of Multithreading. The speedup obtained for Multithreaded version of Rust with threads running parallel over the rows of the image scales very similar to C++ version of Mandelbrot with threads running parallel over the pixels. Although the speedup obtained with Rust is a more non-linear when compared to C++. After some analysis we found that this is due to the dynamic scheduling in Rust. The threads in Rust steal work from other thread when their load is idle. Since the problem of parallelizing Mandelbrot is embarrassing parallel, we do not observe the benefits that Rust provides in making the functionality of code data-race safe.

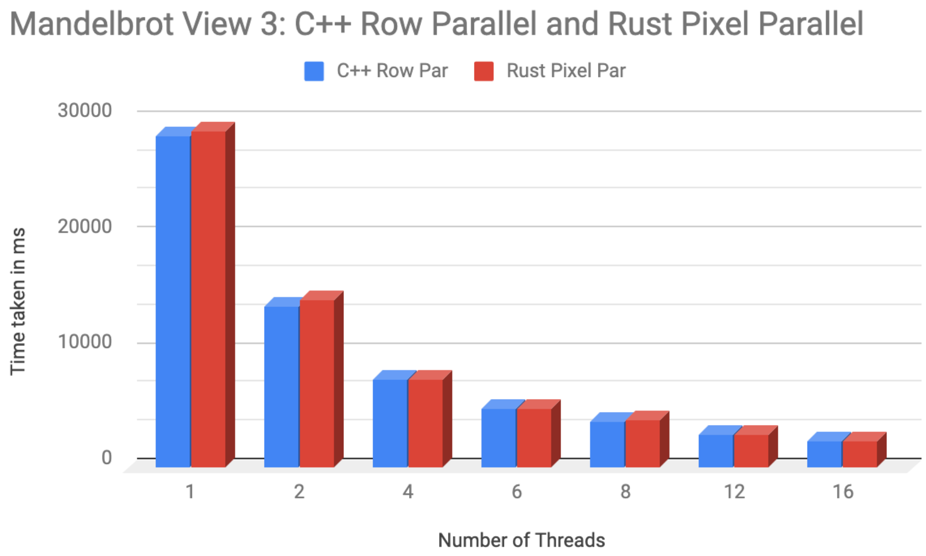

Figure 3: Comparison of time taken by multithreaded version of C++ and Rust Code for Mandelbrot

Figure 3 further iterates the points highlighted in figure 1 and figure 2. The time taken by the Rust implementation of Mandelbrot code lies within 1% of the time taken by C++ implementation.

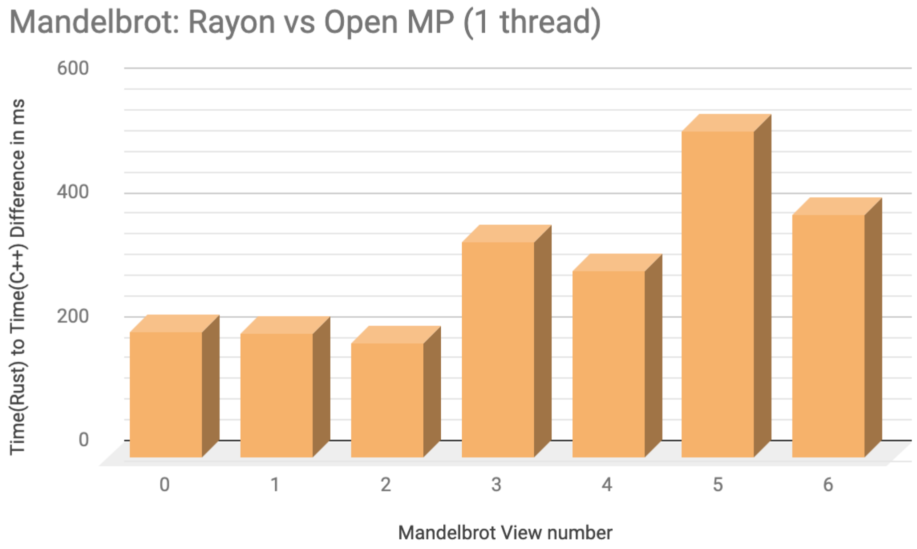

Figure 4: Difference in time taken by single threaded version of Mandelbrot in Rust and C++

Figure 4 analyzes the overhead introduced in spawning a single thread in both Rust and C++. From the timing information showed in the graph, it is observed that for a single threaded version of Mandelbrot, Rust takes more time than the C++ version (Indicated by the positive bar graphs) across all views of the image. This result shows that Rust introduces significant overhead to the boilerplate code.

Challenges

Rust Serial speed issue:

At first, we weren’t able to achieve the same speeds in Rust as in C++ even for the serial code. This leaked into the parallel code. For view 1 - 4096x4096, Rust took 19319.043 *ms* and C++ took 18813.754 *ms*, approximately 500 *ms* more than C++.
Solution: We looked into the assembly, where we found that Rust has 4 extra instructions for Complex32 structure it internally uses. After eliminating it to perform the calculations similar to C++, we got the same speed as C++ (18758.684 *ms*).

Rust Cross Compilation:

We had issues with the non-standard version of Rust installed on the GHC machines. We weren’t able to compile our code with it because the cargo tool was missing, and the non-standard version did not let us install additional crates of Rust for profiling our code.
Solution: We had to opt out for cross compilation for the GHC machine.

Failure of operf on GHC machines:

*operf i*s a performance profiler tool for Linux which we were planning on using to profile our benchmarks. However, we were unable to do so on the GHC machines because of permission issues.
Solution: We plan of exploring tools such as *perf* or alternatively we would love to have the non-standard version of Rust removed on the GHC machines so that we can install crates of Rust that can help in profiling.

Issues and Concerns

One of the issues we faced was with running and compiling Rust successfully on the GHC machine. The problem was because of the non-standard installation of Rust on these machines. We weren’t able to compile our Rust code because the Cargo tool was missing. We would like it if the non-standard version of Rust could be removed from the GHC machines so that this could allow us to install Rust in User Space.

Presentation

For the final presentation, we will be presenting charts and graphs that provide interesting insights for Comparison of Multithreading. Alternatively, we can also demo parts of our code.