# Serial optimizations for the serial 5-point stencil code

Xiaoyue Xiao 1920991

## I. Introduction

This assignment aims to optimize the serial 5-point stencil in order to improve performance. This report will first introduce the influences of different GCC versions. Then, the programmer will test some compiler flags for choosing the best one. Next, the compiler will vectorize the code using 'restrict' keyword. This report also describes the results of modifying the data layout and data type respectively. Finally, This article will introduce other optimizations that cannot successfully improve code performance in this assignment.

## II. GCC version

The default GCC version on Bluecrystal4 is 4.8.5. The latest compiler version improves performance to some extent. According to Table 1, it is clear that the runtimes of three input problems are all reduced after the GCC version is updated to 9.1.0.

Table 1: GCC versions can influence runtimes

| GCC version | 1024×1024 Runtimes(s) | 4096×4096 Runtimes (s) | 8000×8000 Runtimes(s) |
|---|---|---|---|
| **4.8.5** | 4.617649 | 110.034755 | 507.495510 |
| **9.1.0** | 4.584941 | 105.331533 | 481.602353 |

## III. Optimization options

The latest GCC version can reduce runtimes to some extent, but without any optimization option, the compiler just aims to reduce complication cost rather than complication time [1]. In order to speed up runtime, turning on optimization options is necessary. GCC compiler provides some options that can turn on various sorts of optimizations. When users set an optimization option for code, the compiler attempts to reduce compilation time and the debugging ability for the improvement of performance [1].

Table 2 shows the three input runtimes of each option. '-Oo' flag is the default option if users do not set any optimization level for code, which cannot do any optimization. '-Ofast -mtune=native' option turns on the maximum number of flags among five options. Therefore, the runtimes significantly decreased after the optimization level is specified.

Table 2: runtimes of different optimization levels

| | -O0 | -O1 | -O2 | -Ofast | -Ofast -mtune=native |
|---|---|---|---|---|---|
| **1024×1024 Runtimes(s)** | 4.508 | 1.495 | 1.504 | 0.905 | 0.897 |
| **4096×4096 runtimes(s)** | 103.248 | 42.765 | 34.617 | 34.468 | 33.908 |
| **8000×8000 runtimes(s)** | 480.012 | 151.898 | 153.434 | 138.468 | 135.999 |

## IV. Restrict keyword

It is obvious that GCC compiler improves the performance with optimization options. However, there is still more optimization needed.

The vectorization is a critical tool for high performance. With vectorizations, the compiler generates packed SIMD instructions that operate multiple elements at a time [2]. Hence, the loops can execute more efficiently. In the previous section, the optimization option '-Ofast -mtune=native' is specified in our Makefile file and the option turns on the auto-vectorization flag. Due to this flag, the compiler will seek vectorization opportunities. The loop will look for alias

at runtime during vectorization. If the 'restrict' keyword is used in pointer declarations, the compiler will not do any checks for aliasing [2]. Therefore, runtimes will be shortened. The following table 3 demonstrates that runtimes of three different inputs decrease respectively.

Table 3: 'Restrict' keyword helps the compiler ignore aliasing.

| | Before using restrict keyword | After using restrict keyword |
|---|---|---|
| **1024×1024 Runtimes(s)** | 0.900453 | 0.741756 |
| **4096×4096 runtimes(s)** | 33.891886 | 33.524914 |
| **8000×8000 runtimes(s)** | 140.521839 | 122.268852 |

## V. Data layout

The gprof profiler can find out where the costly functions are. The profiler generates a profile report for 5-point stencil code. According to Figure 1, stencil function is an important part of four functions using approximately 98% of total running time. The stencil function is found that the array data is not read continuously. To read data continuously, the order of two iteration variables need to be exchanged. Table 4 proves that the loop interchange can obviously improve the running performance.

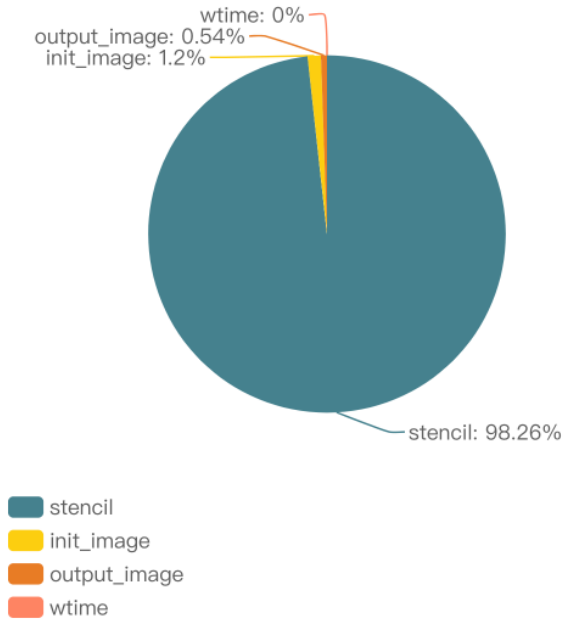


Figure 1: the percentage of total runtimes for each function.

Table 4: The influence of Loop interchange.

| | Before loop interchange | After loop interchange |
|---|---|---|
| **1024×1024 runtimes(s)** | 0.719312 | 0.181445 |
| **4096×4096 runtimes(s)** | 31.960942 | 5.849982 |
| **8000×8000 runtimes(s)** | 124.914562 | 22.270323 |

## VI. Data type

In addition to the data layout, the data type can also influence the code performance. It is clear that smaller types use less memory and provide faster calculations. In this experiment, the experimenter cast double pointers to float pointers. According to Table 5, a smaller data type saves runtimes.

Table 5: runtimes of different data types.

| | double | float |
|---|---|---|
| **1024×1024 Runtimes(s)** | 0.181445 | 0.110686 |
| **4096×4096 runtimes(s)** | 5.849982 | 2.878536 |
| **8000×8000 runtimes(s)** | 22.270323 | 10.348350 |

## VII. Other optimizations

Some optimizations which are not mentioned above are also effective. For example, the intel compiler is a good compiler that turns on some flags by default. Prefetching is a feasible way to bring the data into the cache. Although tiling is an optimization method worth considering, the main function of 5-point stencil code does not have three or more loops, which results in the unavailability of tiling.

## VIII. Conclusion

In this report, a large number of experiments were tested in order to obtain target runtimes. Finally, five optimizations are applied in the code which enables 5-point stencil code can be executed in 0.11 seconds, 2.88 seconds and 22.27 seconds respectively.

## REFERENCE

[1] Gcc.gnu.org. (2019). *Optimize Options - Using the GNU Compiler Collection (GCC)*. [online] Available at: https://gcc.gnu.org/onlinedocs/gcc-4.4.2/gcc/Optimize-Options.html [Accessed 24 Oct. 2019].

[2] Software.intel.com. (2019). *A Guide to Vectorization with Intel® C++ Compilers*. [online] Available at: https://software.intel.com/sites/default/files/m/4/8/8/2/a/31848-CompilerAutovectorizationGuide.pdf [Accessed 25 Oct. 2019].