Parallel Programming Lab Assignment: Diffusion Algorithm

Jan-Hendrik Niemann and Andreas Radke

October 26, 2017

1 Operations and Complexity

We chose the Diffusion Algorithm for our lab assignment. The main part of the algorithm is determined by the loops. We have a triple nested loop in *sum_values* (called twice) for summing all the values of the 3D-matrix, a triple nested loop in *init* (twice) for setting the initial values and the diffusion function with a four-fold nested loop (called once) which is the main part of the computation. Allocating the memory for the array with malloc is in regard to the other functions rather negligible.

Therefore we get for a 3D-matrix with the size of $N_x \times N_y \times N_z$ the number of operations:

$$2N_xN_yN_z + 2N_xN_yN_z + TN_xN_yN_z$$

where $T = \frac{N_xN_yN_z}{1000}$ (in the code it is named *count*). The first two summands are for bigger input sizes way smaller and hence can be neglected. This leads us to the complexity of $\mathcal{O}(N_x^2N_y^2N_z^2)$.

2 Base Version's Performance and Correctness

As we can see in figure (1) the baseline version is very slow. For an input size of $N = 200$ the improved version is nearly 12.6 times faster. It can handle $569 \cdot 10^6$ operations per second. The baseline version scores only $45 \cdot 10^6$ operations per second.

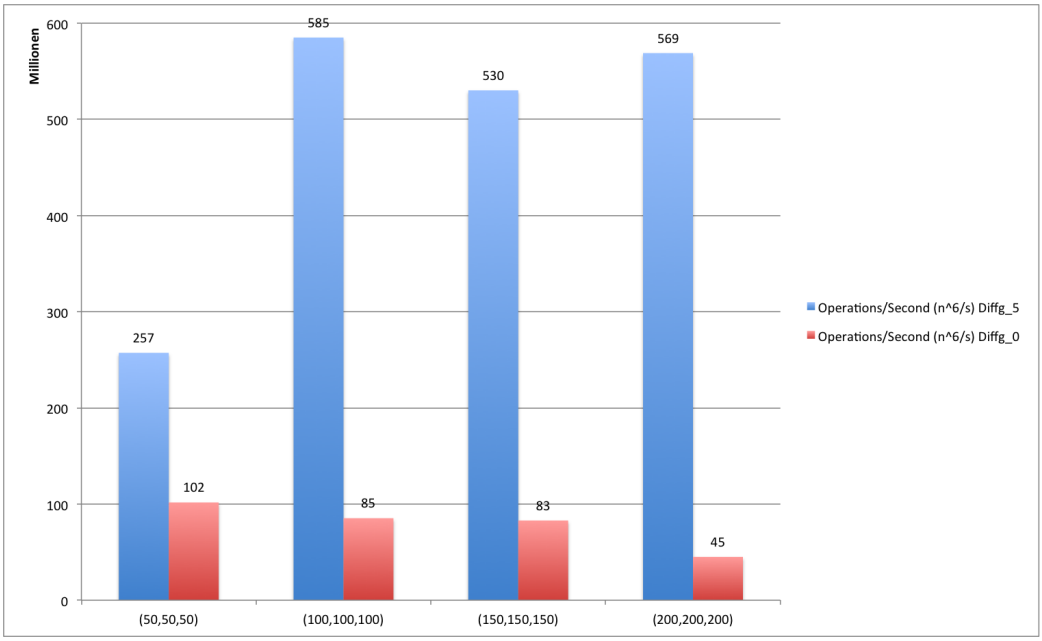

Figure 1: Million Operations per Second

3 Effect of Problem Size

As derived in chapter (1) we can see in figure (2) that the computation time indeed grows exponentially fast. Note that the y-axis has a logarithmic scale. We can see as well that the baseline code is slower than the improved one. It is independend of the input size.

In figure (1) we can see the operations per second in $[N^6/s]$. Note that the baseline code slows down for higher input size. In contrast the improved version "jumps" when doubling the input size from $N = 50$ to $N = 100$. There are almost no differences for higher input sizes. A small variation can be seen for $N = 150$.

4 Compilers

There is a slight difference between the GCC and the ICC compiler. However, in general there cannot made a statement whether GCC or ICC is better. In figure (3) we can see that ICC reaches a higher value for instructions per cycle than GCC. If we take a look at the improved version, we see that the GCC compiler reaches a higher value. The result looks similar if we compare values like instructions, cycles or time.

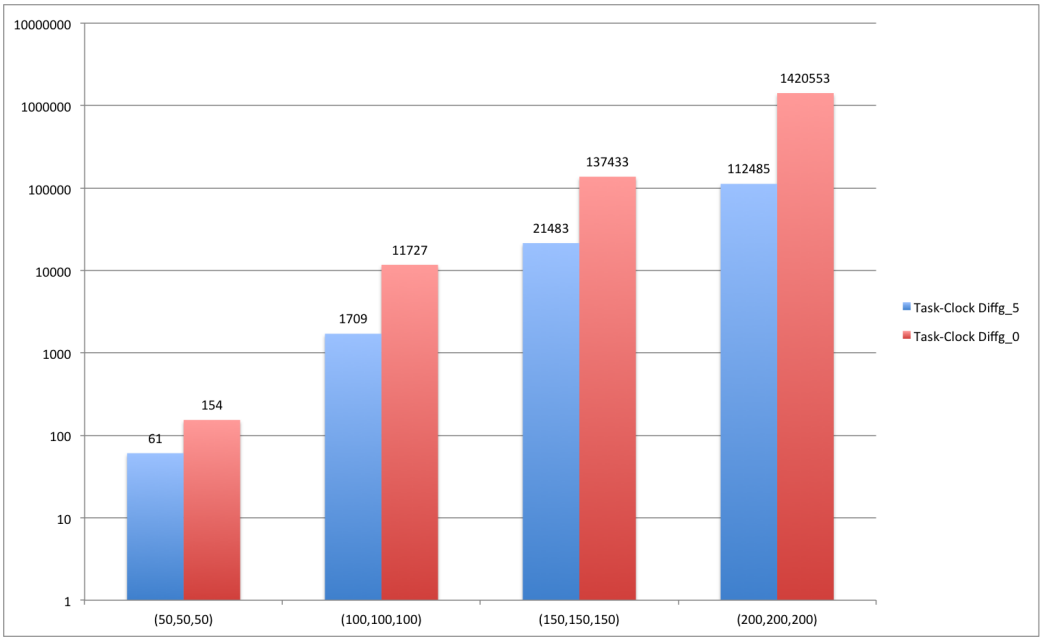

Figure 2: Task-Clock (msec)

5 Optimized Version without SIMD

6 Optimized Version with SIMD

7 Performance Bottlenecks

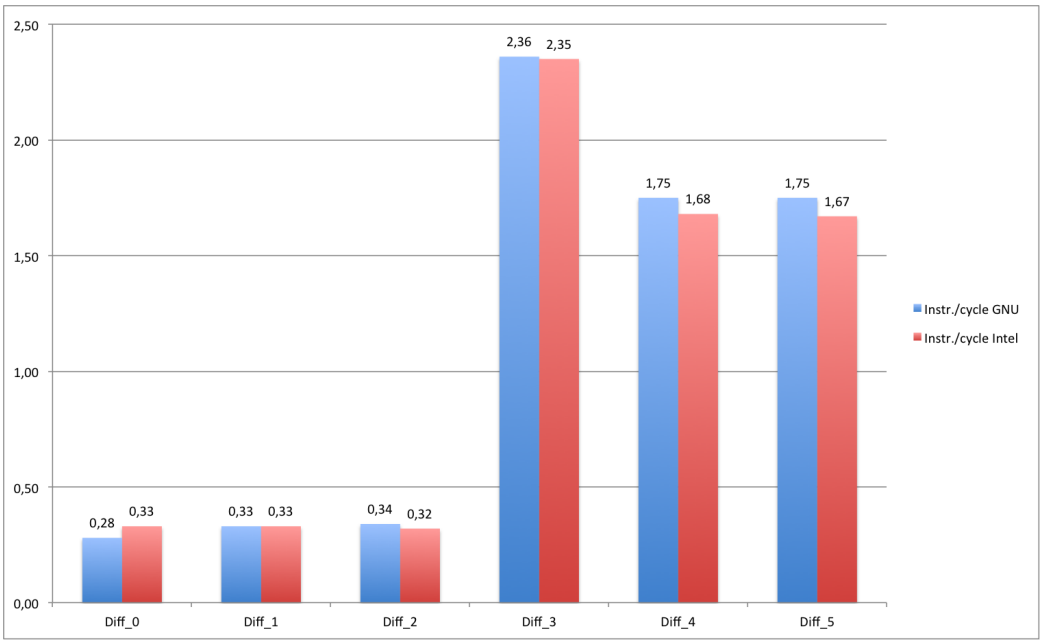

Figure 3: Instructions per cycle