OBHPC TD2
Performance Report
November 2022

R. THIENARD

Contents

1 Introduction & Objectives

The objective of this TP is to discover what performance measurement is. In order to do this, we must use 3 basic version functions and use their implementations. From the most basic to optimized versions, we have to implement and to measure dgemm, reduc and dotprod. Each functions will be running on 3 differents compilers, Gcc (gnu compiler), Clang and Icx (Intel compiler), to measure the impact of the compiler on the performances. I will use only 2 flags to show the differences between compilers and in the annex : Flag Comparaison, I show all my test with differents flag on gcc compiler. I used gcc to highlight the flags because ICX is available for Intel users and will be more efficient with the cblas implementation.

2 Unit test

During my implementation of these functions, I decided, to maximize the process, to use unit test. In fact, unit tests allowed me to check quickly and easily the operation of each fonction. Unit test verify for a small sample if the output corresponds to what I expect. I give small sample and I manually calculate the output so that there is no risk of error.

3 Performance capturing

To measure all functions, I have configured my CPU's to be as stable as possible. In order to do this, I fixed their frequency while making sure that the execution would be on the same core. I used cpupower and taskset. Cpupower allows user to set the cpu frequency if it's possible or just choose the gouvernor. The gouvernor has several possibility like "powersave", "performance" or "userspace". For some CPU, the "usrspace" is not reachable and only "powersave" and "performance" could be chosen. this is my case on my laptop, but on my server, I can fix manualy my frequency with the "userspace". You must disable the boost on your CPU.
Now, to run the program, I only have to fix it on 1 core of my cpu using taskset.

4 CPUs used

To realize this performance report, I used 2 processors. Both commercialized by Intel. The first one is a I7-1165G7 and the second a I5-3570. It's a Tigerlake's processor released in 2020. It is on my laptop and has 4 cores with 8 threads. It has a maximum frequency of 4.7GHz. It has also 12MiB of L3 cache and has AVX512. The cons of this processor is that I can not be configured manualy its frequency because of P-STATE.
The second cpu is a I5-3570. It is an older version belonging to the Ivy Brige generation. Released in 2012, this CPU has 4 cores and 8 threads too. Its maximum frequency is 3.8GHz and it has 6MiB of L3 cache. It hasn't AVX512 but on this CPU I can manage easily the frequency to generate more stable measure.
All the information concerning the characteristics are available in the files I7-1165G7.txt and I5-3570.txt.

For all future measurement values, Graphs have N which is a multiple of 8 because i don't manage garbages of the function unroll 8. Measurement values were chosen arbitrarily. For x-axis, i put the bandwidth. I also decided that the bandwidth will always be in MiB/s to show the gap between 2 implementations. For my future values, I expect to have ICX better than GCC and CLANG. I also expect unroll 8 and CBLAS to perform better than other implementations. Selected values have an stddev of less than 2%. 5% stddev being the maximum accepted to keep the values consistent. I think that ICX will be the most efficient because it use as clang backend LLVM. I except too that the most efficient will be CBLAS for ICX.

5 Dotproduct

The Dotproduct is a function to sum the product of two vectors. Dotprod is a function with complexity in $O(N)$. I expect to have approximatly the same bandwidth for all N given. I have set up 4 versions of the dotproduct. Each of the performance measures was done respecting the part Performance capturing. The measured versions are :

- The basic version written in C.
- The version with an unroll 8, also written in C.
- The x86 assembler SSE version.
- And finally the one provided by CBLAS.

The compilation flags are "-O1" and "-O3".

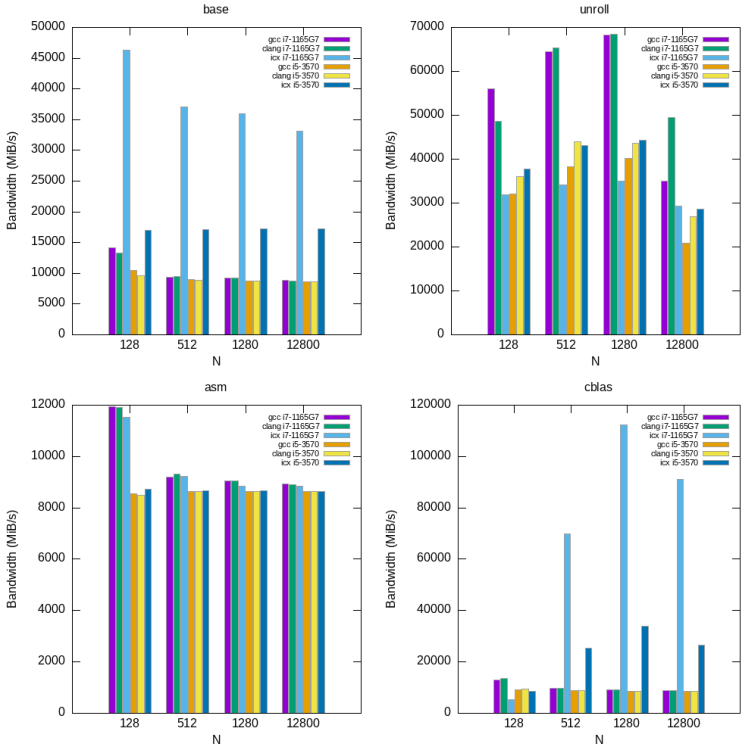

Figure 1: Dotprod O3

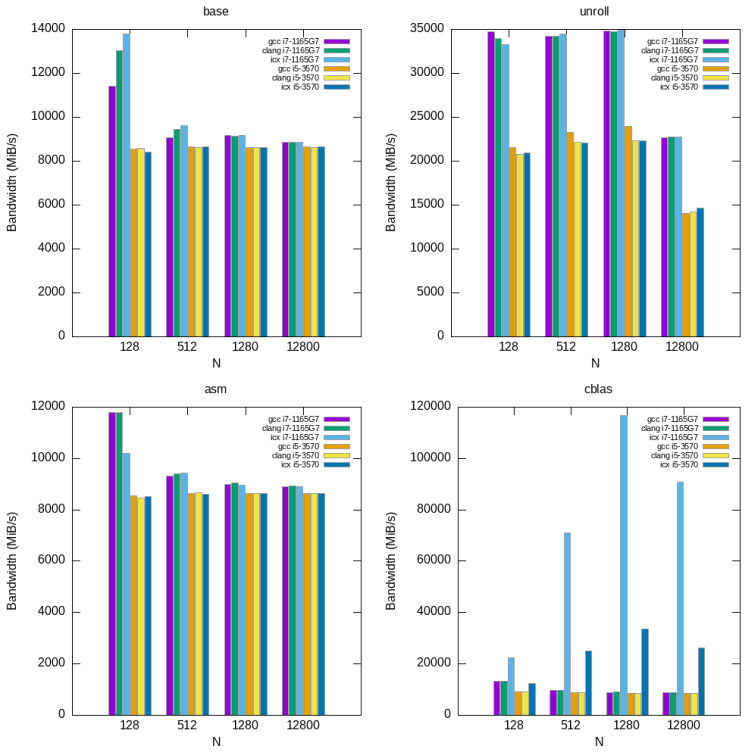

Figure 2: Dotprod O1

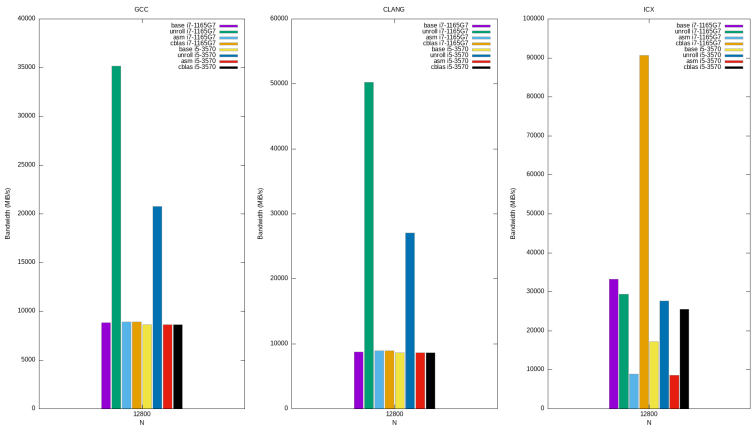

Figure 3: Dotprod O3

6 Reduction

The Reduction is a function to sum one vector. Reduction is a function with complexity in $O(N)$. I expect to have approximatly the same bandwidth for all N given. I have set up 4 versions of the reduction. Each of the performance measures was done respecting the part Performance capturing. The measured versions are :

- The basic version written in C.
- The version with an unroll 8, also written in C.
- The x86 assembler SSE version.
- And finally the one provided by CBLAS.

The compilation flags are "-O1" and "-O3".

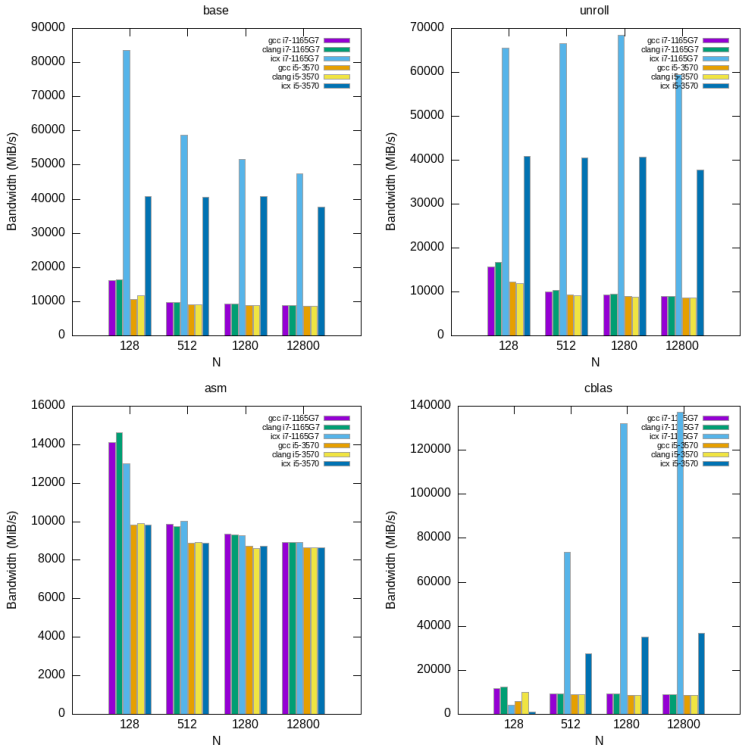

Figure 4: Reduc O3

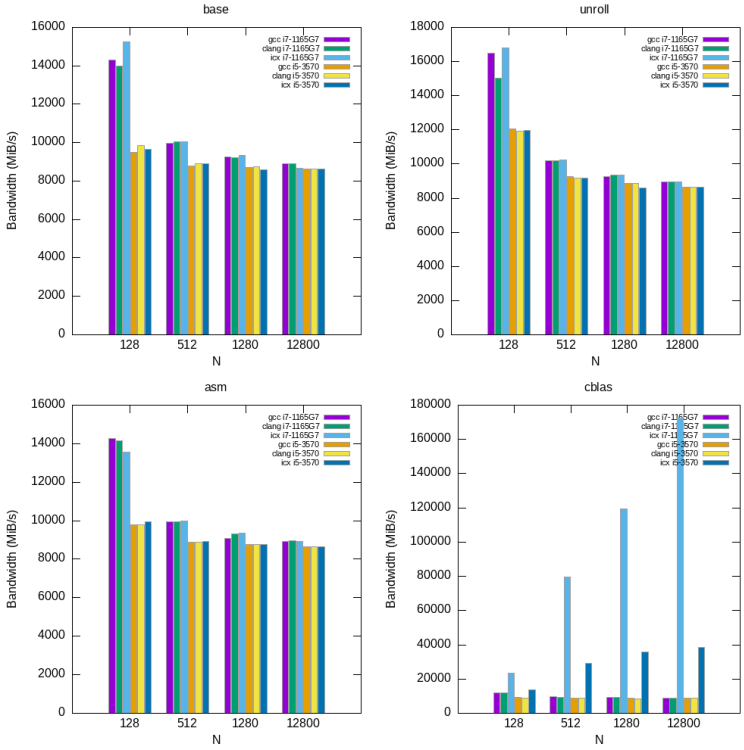

Figure 5: Reduc 01

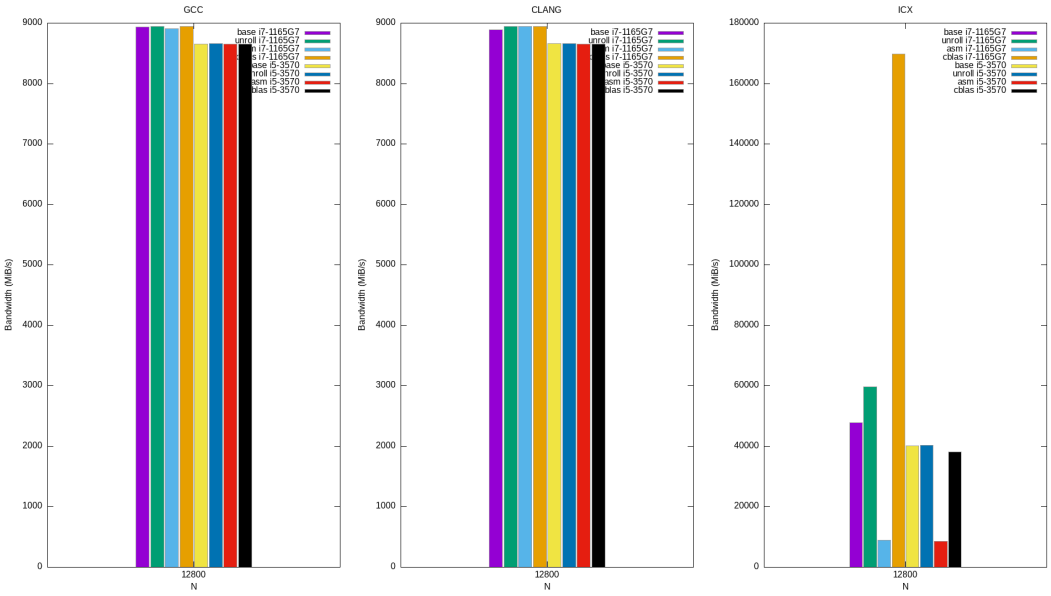

Figure 6: Reduc O3

7 Dgemm

The Dgemm is a function for multiplying two matrix. Dotprod is a function with complexity in $O(N^3)$. I expect to have the bandwidth decreasing in increasing N. I have set up 6 versions of the dgemm. Each of the performance measures was done respecting the part Performance capturing. The measured versions are :

- The basic version written in C which is also call IJK.
- The first optimized version written in C which is also call IKJ. This optimization requires only to move 2 loops of the basic version.
- The second optimized version written in C which is also call IEX.
- The version with an unroll 4, also written in C. Like the unroll 8 but smaller.
- The version with an unroll 8, also written in C.
- And finally the one provided by CBLAS.

The compilation flags are "-O1" and "-O3".

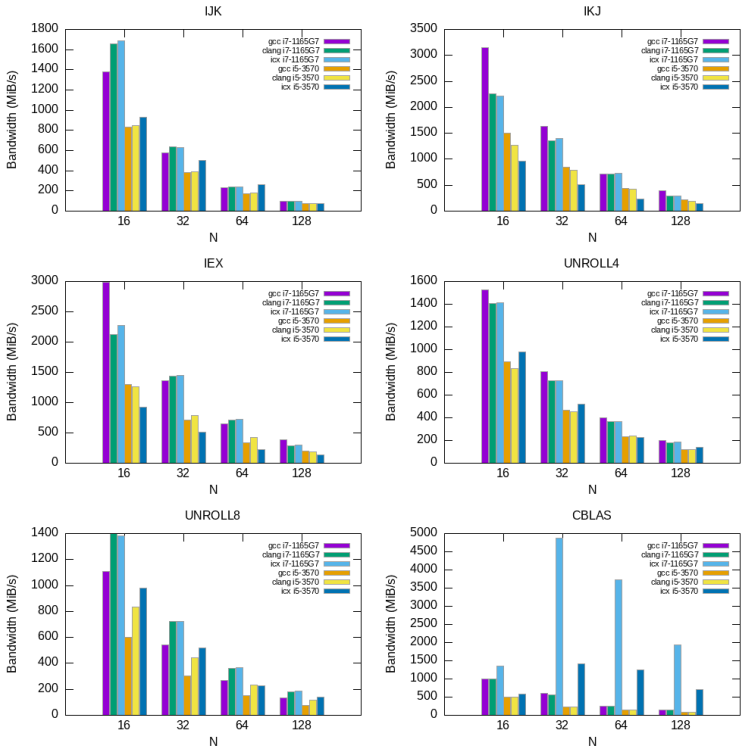

Figure 7: Dgemm O3

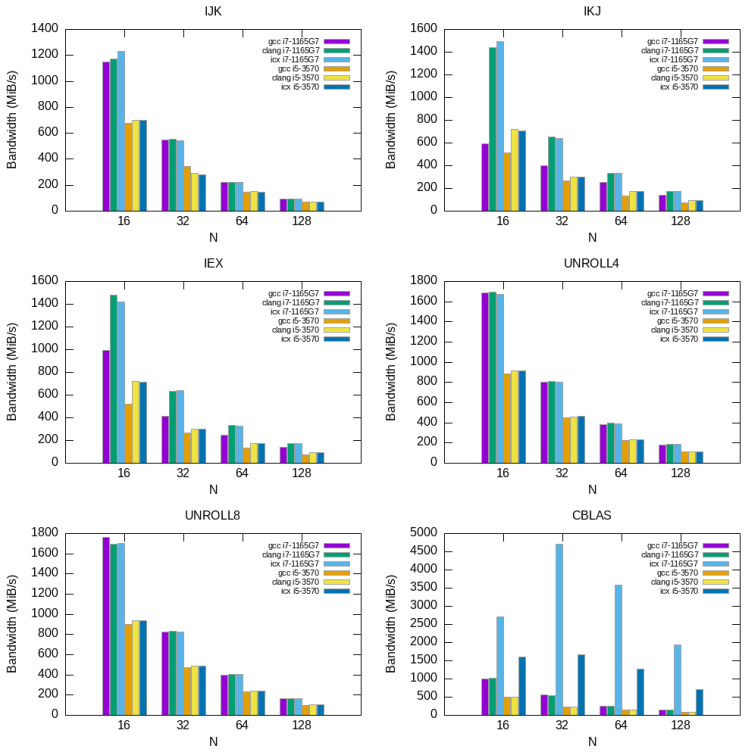

Figure 8: Dgemm O1

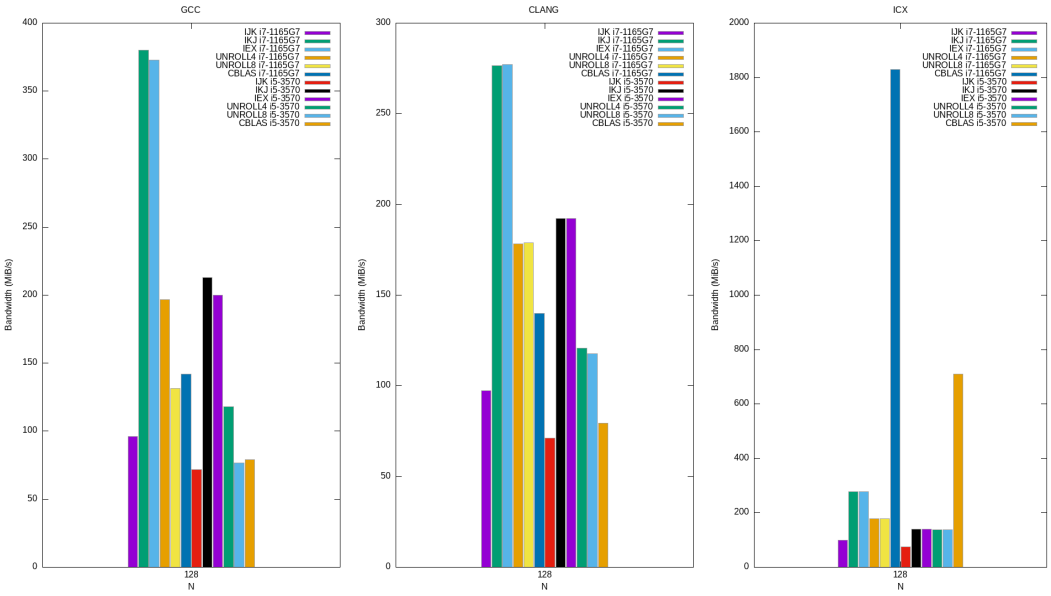

Figure 9: Dgemm O3

8 Flag Comparaison

I decided to compair these 4 flags after trying 8 flags, as -march, funroll-loops, -mtune. But the differences with and without -march or funroll-loops is not visible. I decided to highlight the flags I decided to highlight the flags and see the optimizations through the flags. To know what they contain, i recommand you to see the man.

8.1 Dgemm

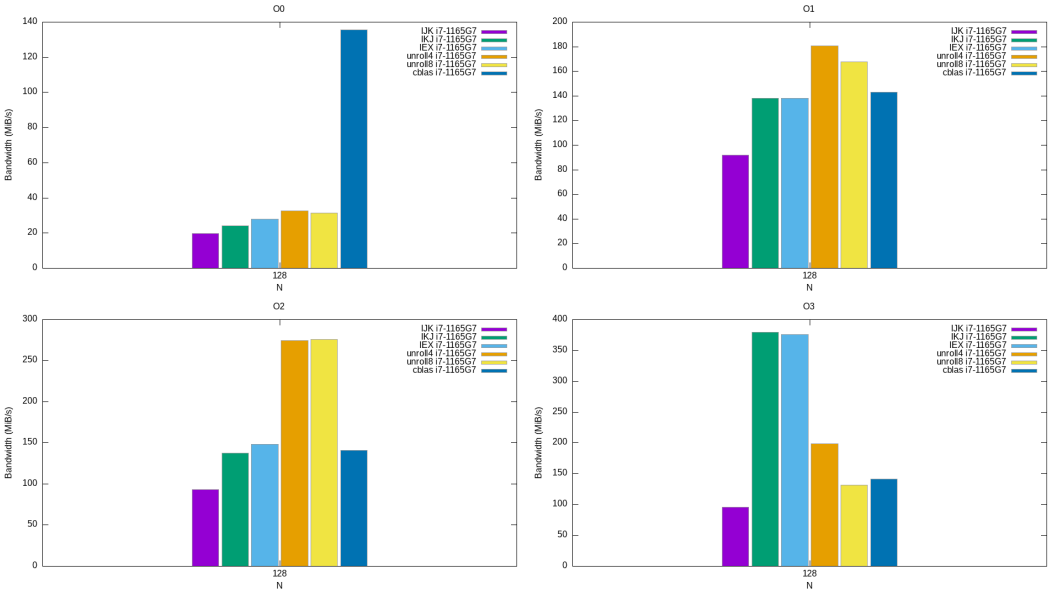

Figure 10: Gcc Dgemm Flags

8.2 Dotprod

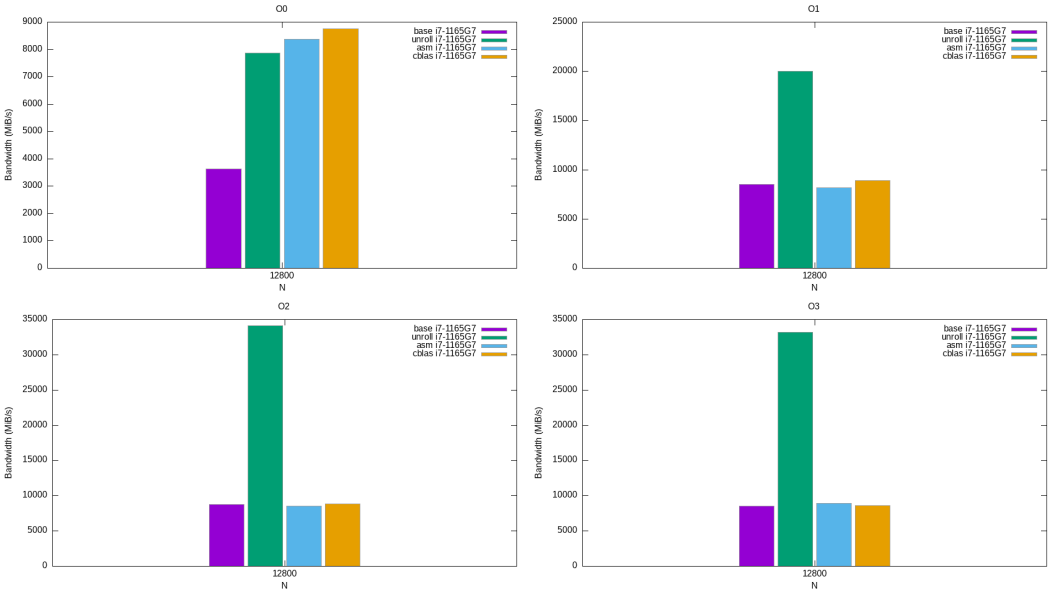

Figure 11: Gcc Dotprod flags

8.3 Reduction

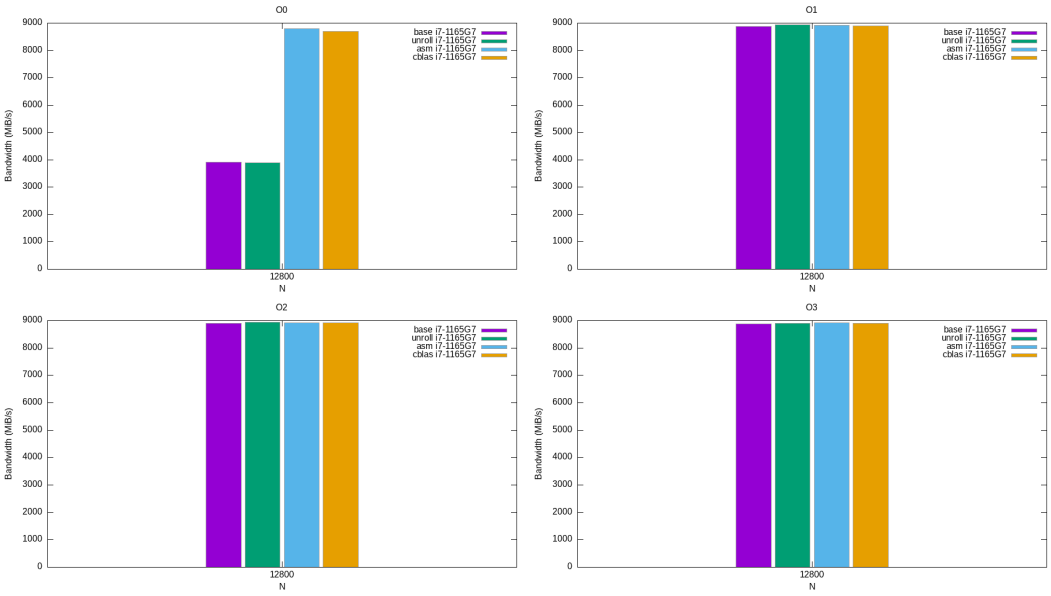

Figure 12: Gcc Reduc Flags

9 To conclude

With some exceptions, CBLAS using MKL lib for ICX is the most efficient. However, I am surprised that the DGEMM function of the MKL library is so efficient whatever the given N. ICX is not hugely more efficient for other implementations as I expected. I think MKL is more efficient because it is the only which is able to use entirely my cpu and all the shortcut designed by intel.
For Dgemm, I asked myself about IEX and Unroll, Unroll is a iex version with forced unroll. I supposed at the beggining that it was the unroll4 and unroll8 which were to large. I tried the Unroll2 implementation and the efficiency was like unroll4 and 8. I don't know why IEX is more efficient than Unroll but i guess it's due to to the O3 flags and precisly "-fpredictive-commoning" and "-floop-unroll-and-jam".

For Dotprod, unroll8 is the most efficient. I tried to implement unroll16 and i noticed that it's was 2 times more efficient.