Performance measurements
OBHPC-info

Zakaria EJJED
M1 CHPS

14 novembre 2022

Introduction
The goal of this project is to measure performances from three different programs –dgemm, dotprod and reduc– with diverse compiler and optimization flags. In the following parts, we'll test gcc, clang and icx compilers and analyze there impact on our programs.

The programs
The programs that we're gonna use are:
-dgemm: Multiply two matrix,
-dotprod: Sum the product of two vectors,
-reduc: Sum a vector.

They all set the vectors and matrix lenght to 80 and do 100 repetition. The number repetition is arbitrary and the lenght is a multiple of 8 so unroll8 won't be disadventaged.

My CPU and caches config
Before analyzing the results, it's important to know on which machine we'll do our benchmarks. The cpu we'll use for those test is an Intel Celeron N4020. It has two cores and two threads (one on each core). We're gonna use the second core and fix it at 2.5GHz of frequency, just a little under the maximum of 2.8GHz to be more stable. It only have two caches, 2 instances of L1 measuring 48KiB and an instance of L2 of 4MiB. All the informations about the cpu and caches can be found in the README.md.

Dgemm

Description

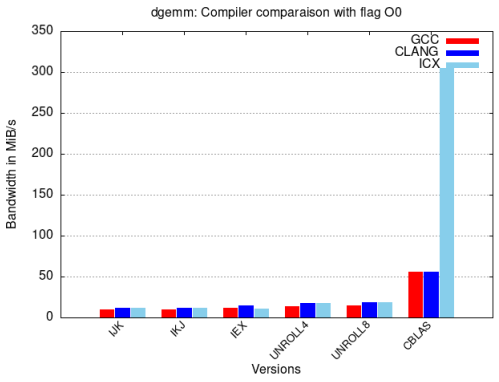
Complexity: O (n^3)

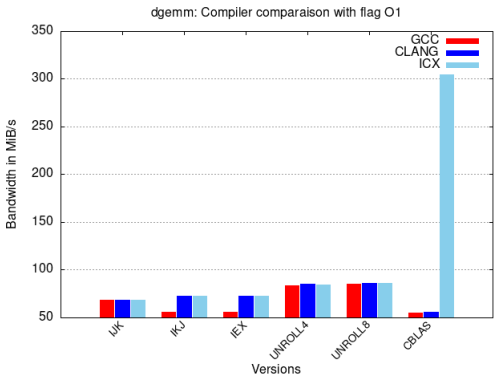
The programm dgemm multiply two matrix of size nxn with r repetition. Like said previously we'll assign by default a value of 80 to n and 100 to r.

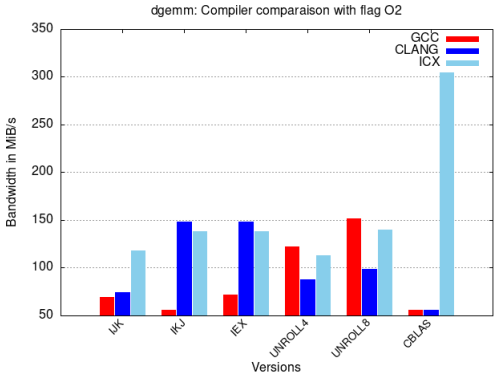
The different version will be:
-IJK: The naive implementation.
-IKJ: Same as IJK but with a loop interchange to easier the memory access
-IEX: Same as IKJ but with an invariant extraction
-UNROLL4: Same as IEX but doing 4 operations by iteration to reduce the number of instructions
-UNROLL8: Same as UNROLL4 but with 8 operations by iteration this time
-CBLAS: Using the cblas function

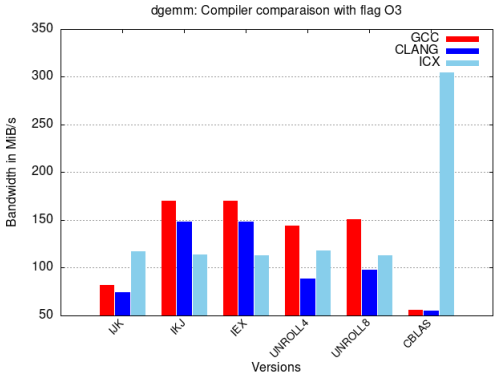
Overall results analysis

First, no matter the flags, the icx compiler using cblas is far more performant than the other versions or compilers. Then, after O0, clang is at its lowest when using cblas, and it's the same for gcc in O3. It doesn't mean that the performances on cblas crashed, but rather the results of the other versions get better. In general, cblas aside, IJK is the worst version and is far behind IKJ and IEX which are quite equivalent. Without surprise the unroll8 is better than the unroll4 but is not twice better, and is quite the same on the first optimizations.





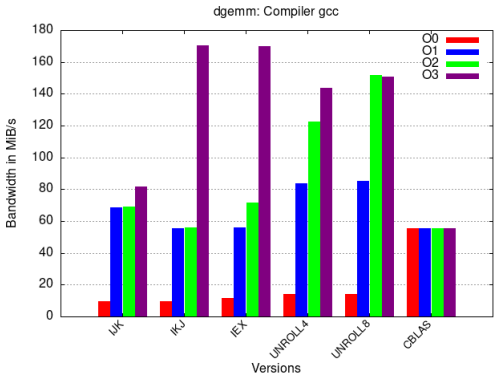




Compilers

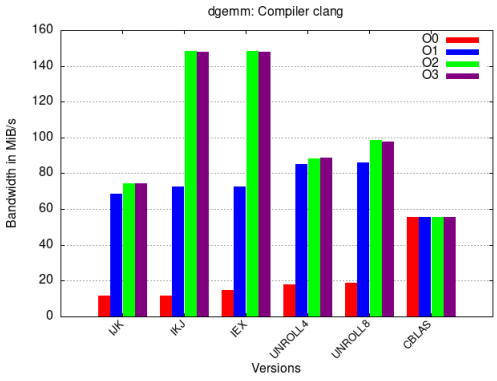
-gcc

When looking at the plot, we can obviously see that the flags have no impact on the cblas version. However, we can see the results skyrocket on the other versions. Especially on IKJ and IEX when using O3.
Going from O0 to O1 then to O2 have a clear repercussion on unrolls but O3 doesn't make a lot of changes. With gcc the IKJ and IEX versions are globally equivalent, except when using O2, IEX seems to get slightly better than IKJ.



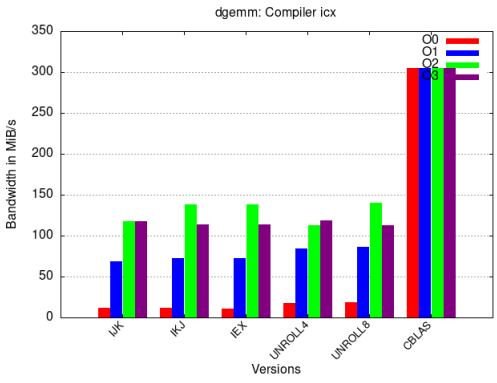
-clang

Just like gcc, clang have almost no variation with cblas while changing the flags. Also, O2 and O3 seems equivalent for all versions here, and while the O0 provide mediocre results, it gets a lot better with O1 for all versions and nearly twice higher on IKJ and IEX using O2.



-icx

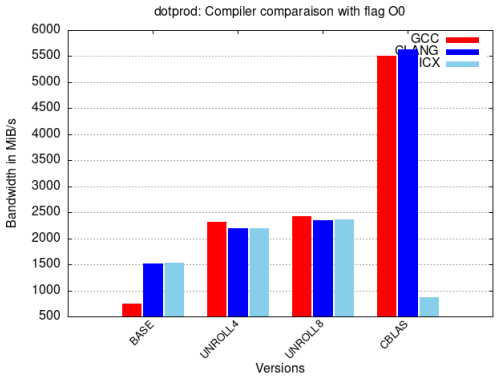
Like the others, the flags induce no alteration on the cblas version, but icx produce high results with cblas. This compiler give low results with O0 but raise significantly with O1 and O2, and except of IJK, even if it's not lower than O1 it surprisingly decrease when using O3.

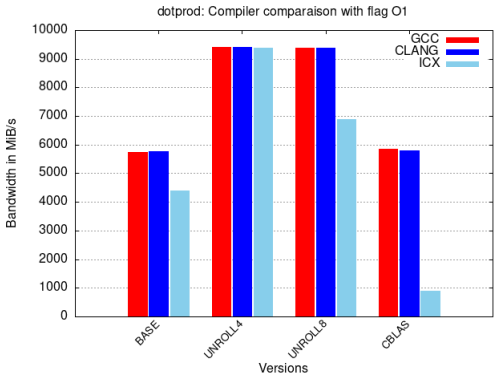


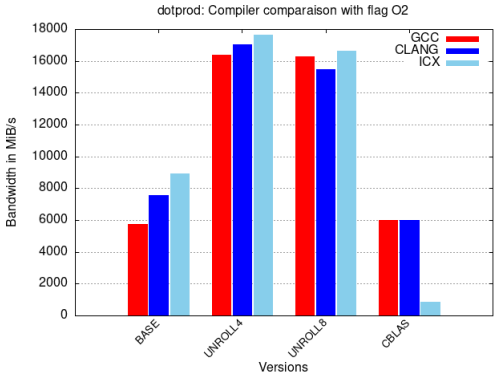
Dotprod
Description
Complexity: O (n)
The programm dotprod sum the product of two vectors of size n with r repetition. Like dgemm we'll assign by default a value of 80 to n and 100 to r for the same reason.

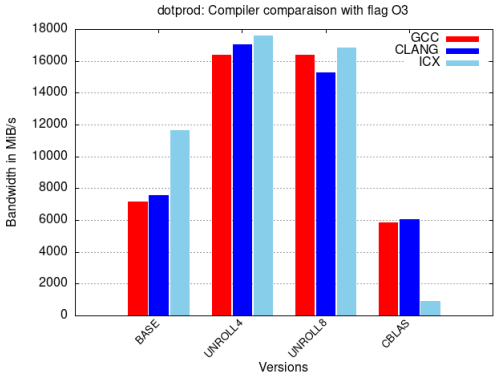
The different version will be:
-BASE: The naive implementation.
-UNROLL4: Same as BASE but doing 4 operations by iteration to reduce the number of instructions
-UNROLL8: Same as UNROLL4 but with 8 operations by iteration this time
-CBLAS: Using the cblas function ćblas_ddot´

Overall results analysis
For this program, cblas with icx is always the worst versions, but with gcc and clang, it seems to be equivalent to the base version. Also, the unrolls are better than the base version but the unroll8 isn't better than the unroll4. Furthermore, gcc and clang produce similar results.





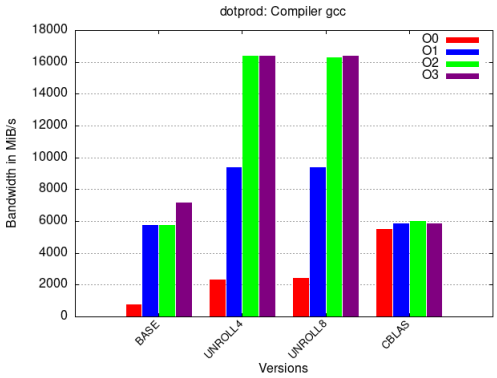


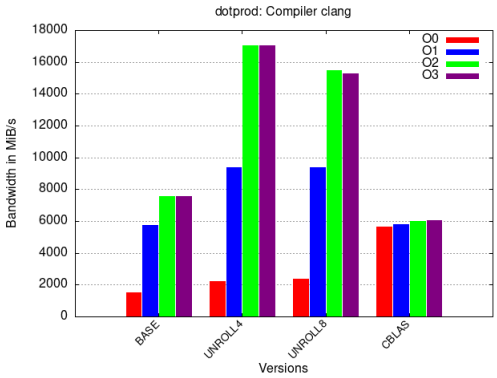


Compilers

-gcc & clang

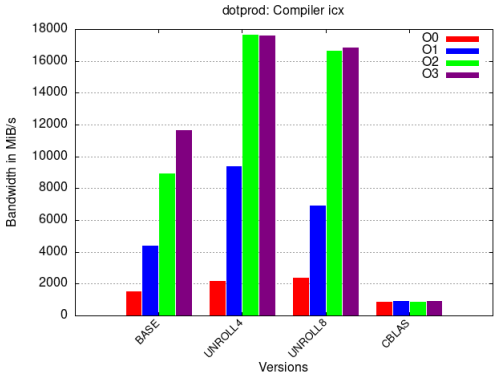
In view of the similarity between gcc and clang results for dotprod, we'll treat them at the same time. This time too, the flags seems to have no great impact on cblas, except a slight increase. Like said before, the unrolls are always better than the base version, and better than cblas starting from O1. The only real difference between gcc and clang, is the weird little lowering between unroll4 and unroll8 while using clang.





-icx

ICX have really low results which doesn't raise or drop with flags on cblas but skyrocket on the other versions while passing throught the flags, except for the unrolls between flag O2 and O3.



Reduc

Description

Complexity: O (n)

The programm reduc sum the values inside vector of size n with r repetition. Like dgemm and dotprod we'll assign by default a value of 80 to n and 100 to r for the same reason.

The different version will be:

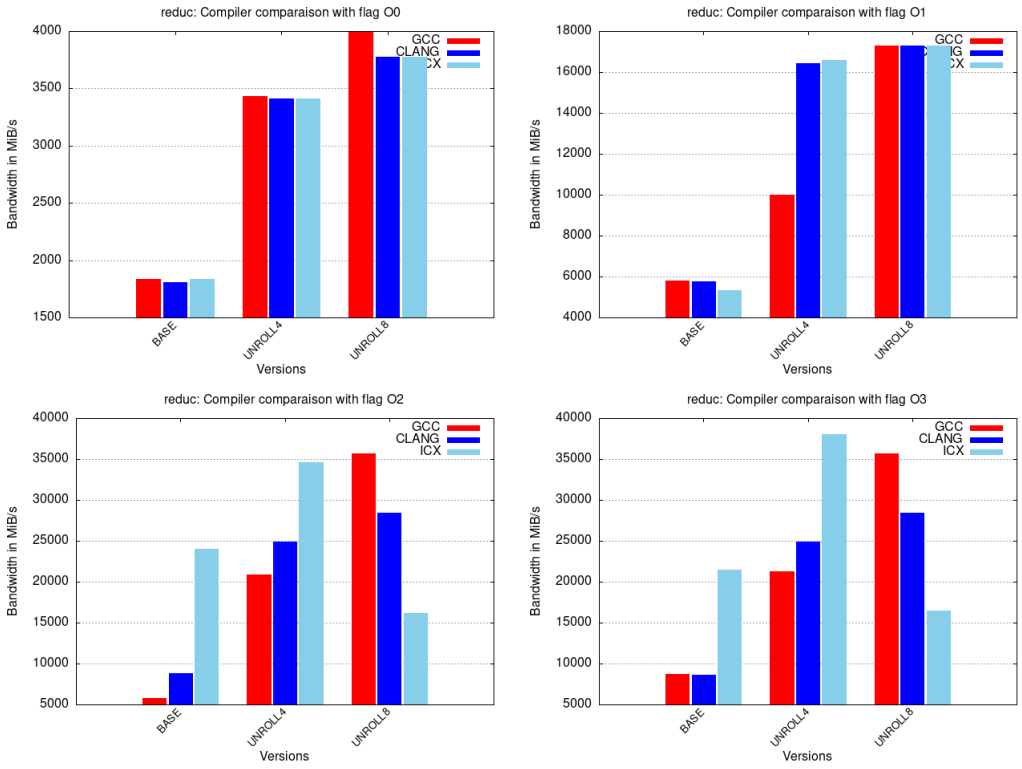
-BASE: The naive implementation.

-UNROLL4: Same as BASE but doing 4 operations by iteration to reduce the number of instructions

-UNROLL8: Same as UNROLL4 but with 8 operations by iteration this time

Overall results analysis

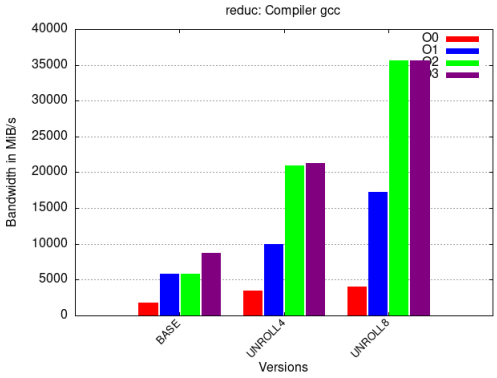
Globally, for O0 and O1, unrolled versions are way better than the base and the three compiler are similar. But when we use O2 and O3, the results get a lot different. First, for base and unroll4 icx is a lot better than the two others but it crashes on O3, while gcc is really low using the base version but skyrocket throught the flags.

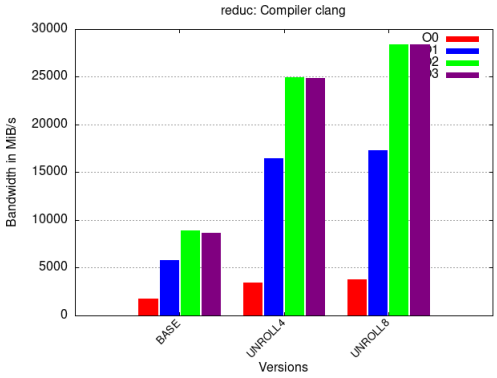


Compilers

-gcc & clang

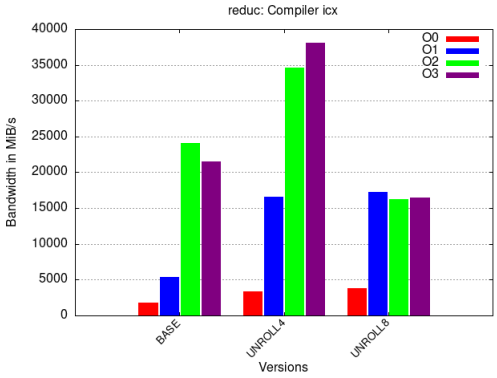
Once again we'll treat gcc and clang the same way because, even if the values are different, the overall growth are similar. On the unrolled versions, they both greatly increase with O1 and O2 but stay equivalent with O3. But on the base version, clang follow the same logic and gcc get almost no increase between O1 and O2 but raise with O3.





-icx

The intel compiler (icx) have a really particular growth on this program, on the base and unroll4 it looks like clang, but it crash on unroll8, also the evolution past O1 is extremely low with the unroll8.



Conclusion

In conclusion, on dgemm cblas is extremely efficient and is not impacted by the optimisation flags. Also, the unroll doesn't always grow like expected. We would normally think that the unroll8 would be twice as efficient as the unroll4, but even if it globally increase, it is not always a incredible raise, and it sometime crash. At the exception of cblas, the optimization flags always have a great impact when going from O0 to O1. Going from O2 to O3 almost never change the unrolls results, probably because of the funroll-loops flag.