HPCTools:
Software profiling and optimizations

Assignment due 29/12/2023

Romaric Vandycke

2023/2024

Valgrind Memcheck:

I considered as the Baseline the code dgesvtask1.c

First of all, in order to use valgrind on the dgesv.c file I need to upate the Makefile and add the lines:
CC = gcc
CFLAGS = -Wall -g

I'am using gcc for now, as I will benchmark the different compiler.
After using (valgrind –leak-check=yes ./dgesv 50) I'am faced with this result:

```
==3336358==
==3336358== HEAP SUMMARY:
==3336358==     in use at exit: 80,216 bytes in 6 blocks
==3336358==   total heap usage: 31 allocs, 25 frees, 160,856 bytes allocated
==3336358==
==3336358== 200 bytes in 1 blocks are definitely lost in loss record 2 of 6
==3336358==    at 0x483779F: malloc (in /mnt/netapp1/Optcesga_FT2_RHEL7/2020/gentoo/22072020/usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==3336358==    by 0x109BFE: main (in /mnt/netapp2/Home_FT2/home/ulc/cursos/curso362/HPCTools/HPPCTools-Task1/dgesv)
==3336358==
==3336358== 20,000 bytes in 1 blocks are definitely lost in loss record 3 of 6
==3336358==    at 0x483779F: malloc (in /mnt/netapp1/Optcesga_FT2_RHEL7/2020/gentoo/22072020/usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==3336358==    by 0x1099AC: generate_matrix (in /mnt/netapp2/Home_FT2/home/ulc/cursos/curso362/HPCTools/HPPCTools-Task1/dgesv)
==3336358==    by 0x109B94: main (in /mnt/netapp2/Home_FT2/home/ulc/cursos/curso362/HPCTools/HPPCTools-Task1/dgesv)
==3336358==
==3336358== 20,000 bytes in 1 blocks are definitely lost in loss record 4 of 6
==3336358==    at 0x483779F: malloc (in /mnt/netapp1/Optcesga_FT2_RHEL7/2020/gentoo/22072020/usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==3336358==    by 0x1099AC: generate_matrix (in /mnt/netapp2/Home_FT2/home/ulc/cursos/curso362/HPCTools/HPPCTools-Task1/dgesv)
==3336358==    by 0x109BA7: main (in /mnt/netapp2/Home_FT2/home/ulc/cursos/curso362/HPCTools/HPPCTools-Task1/dgesv)
==3336358==
==3336358== 20,000 bytes in 1 blocks are definitely lost in loss record 5 of 6
==3336358==    at 0x483779F: malloc (in /mnt/netapp1/Optcesga_FT2_RHEL7/2020/gentoo/22072020/usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==3336358==    by 0x109A3F: duplicate_matrix (in /mnt/netapp2/Home_FT2/home/ulc/cursos/curso362/HPCTools/HPPCTools-Task1/dgesv)
==3336358==    by 0x109BBC: main (in /mnt/netapp2/Home_FT2/home/ulc/cursos/curso362/HPCTools/HPPCTools-Task1/dgesv)
==3336358==
==3336358== 20,000 bytes in 1 blocks are definitely lost in loss record 6 of 6
==3336358==    at 0x483779F: malloc (in /mnt/netapp1/Optcesga_FT2_RHEL7/2020/gentoo/22072020/usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==3336358==    by 0x109A3F: duplicate_matrix (in /mnt/netapp2/Home_FT2/home/ulc/cursos/curso362/HPCTools/HPPCTools-Task1/dgesv)
==3336358==    by 0x109BD1: main (in /mnt/netapp2/Home_FT2/home/ulc/cursos/curso362/HPCTools/HPPCTools-Task1/dgesv)
==3336358==
==3336358== LEAK SUMMARY:
==3336358==    definitely lost: 80,200 bytes in 5 blocks
==3336358==    indirectly lost: 0 bytes in 0 blocks
==3336358==      possibly lost: 0 bytes in 0 blocks
==3336358==    still reachable: 16 bytes in 1 blocks
==3336358==         suppressed: 0 bytes in 0 blocks
==3336358== Reachable blocks (those to which a pointer was found) are not shown.
==3336358== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==3336358==
==3336358== For lists of detected and suppressed errors, rerun with: -s
==3336358== ERROR SUMMARY: 5 errors from 5 contexts (suppressed: 8 from 2)
```

I can see I have memory loss coming from the main function, more importantly it seems 5/6 variable have not been freed. I decide to add in the main.c files the lines:

```
  free(a);
  free(b);
  free(aref);
  free(bref);
  free(ipiv);

  return 0;
}
```

After using valgrind again I obtain this result:

```
Time taken by my dgesv solver: 34 ms
==3349304==
==3349304== HEAP SUMMARY:
==3349304==     in use at exit: 16 bytes in 1 blocks
==3349304==   total heap usage: 31 allocs, 30 frees, 160,856 bytes allocated
==3349304==
==3349304== LEAK SUMMARY:
==3349304==    definitely lost: 0 bytes in 0 blocks
==3349304==    indirectly lost: 0 bytes in 0 blocks
==3349304==      possibly lost: 0 bytes in 0 blocks
==3349304==    still reachable: 16 bytes in 1 blocks
==3349304==         suppressed: 0 bytes in 0 blocks
==3349304== Reachable blocks (those to which a pointer was found) are not shown.
==3349304== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==3349304==
==3349304== For lists of detected and suppressed errors, rerun with: -s
==3349304== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 8 from 2)
```

I decide to follow the call and rerun with the line they want me to and obtain this:

```
==3363976==
==3363976== HEAP SUMMARY:
==3363976==     in use at exit: 16 bytes in 1 blocks
==3363976==   total heap usage: 31 allocs, 30 frees, 160,856 bytes allocated
==3363976==
==3363976== 16 bytes in 1 blocks are still reachable in loss record 1 of 1
==3363976==    at 0x483779F: malloc (in /mnt/netapp1/Optcesga_FT2_RHEL7/2020/gentoo/22072020/usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==3363976==    by 0x5BDC7BC: ??? (in /mnt/netapp1/Optcesga_FT2_RHEL7/2020/gentoo/22072020/usr/lib64/gcc/x86_64-pc-linux-gnu/11.2.0/libgomp.so.1.0.0)
==3363976==    by 0x5BEDCEA: ??? (in /mnt/netapp1/Optcesga_FT2_RHEL7/2020/gentoo/22072020/usr/lib64/gcc/x86_64-pc-linux-gnu/11.2.0/libgomp.so.1.0.0)
==3363976==    by 0x5BDA7D4: ??? (in /mnt/netapp1/Optcesga_FT2_RHEL7/2020/gentoo/22072020/usr/lib64/gcc/x86_64-pc-linux-gnu/11.2.0/libgomp.so.1.0.0)
==3363976==    by 0x400FD51: ??? (in /mnt/netapp1/Optcesga_FT2_RHEL7/2020/gentoo/22072020/lib64/ld-2.31.so)
==3363976==    by 0x400FE58: ??? (in /mnt/netapp1/Optcesga_FT2_RHEL7/2020/gentoo/22072020/lib64/ld-2.31.so)
==3363976==    by 0x40010C9: ??? (in /mnt/netapp1/Optcesga_FT2_RHEL7/2020/gentoo/22072020/lib64/ld-2.31.so)
==3363976==    by 0x1: ???
==3363976==    by 0x1FFEFFEA7A: ???
==3363976==    by 0x1FFEFFEA82: ???
==3363976==
==3363976== LEAK SUMMARY:
==3363976==    definitely lost: 0 bytes in 0 blocks
==3363976==    indirectly lost: 0 bytes in 0 blocks
==3363976==      possibly lost: 0 bytes in 0 blocks
==3363976==    still reachable: 16 bytes in 1 blocks
==3363976==         suppressed: 0 bytes in 0 blocks
==3363976==
==3363976== For lists of detected and suppressed errors, rerun with: -s
==3363976== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 8 from 2)
```

When reading this output, it seems the mistake come from the ld and libgomp library which are use for openMp, which I do not yet use for my code. I consider this a False Positive and decide to move on in the exercises.

BenchMarking

First of all, we will study the GCC compiler with different -0X optimization levels -march=native
Compare run time and see which compiler is best for each level.
For every test, we will consider a matrix size of 1024 and will run the code 5 times and take the mean of the value (all data available in the excel file)
Considering the mean of all result I have, I obtained:

Gcc 01 = 2.267,8 ms
Gcc 02 = 1.285,2 ms
Gcc 03 = 973,4 ms
Gcc Ofast = 1.061,2 ms
Gcc march=native = 4550 ms (same result as no optimizations)

Now I will use icx, I uncomment the necessary line in the makefile and load the module, use cc=icx when compiling. I obtain the following result:

Icx 01 = 2.131,6 ms
Icx 02 = 998 ms
Icx 03 = 985,4 ms
Icx march=native = 982,2 ms
Icx Ofast = 1025,6 ms

Auto-Vectorize

For this part It's important to consider when comparing result that using certain flg that I used above are important to enable the compiler to auto vectorize (like using -O2/3 and march=native)

Starting with the icx compiler:
I run for the first time the lines make "FLAGS = -O2 -qopt-report=2" "CC=icx"
I do obtain a run time of 950ms approximately which is an'improvement but it is now the time to modify the code:
I will start by aligning the data is used (solutions, matrice A/B):

```
double *generate_matrix(unsigned int size, unsigned int seed)
{
  unsigned int i;
  double *matrix = (double *)aligned_alloc(64,sizeof(double) * size * size);

  srand(seed);

  for (i = 0; i < size * size; i++) {
    matrix[i] = rand() % 100;
  }

  return matrix;
}

double *duplicate_matrix(double *orig, unsigned int size)
{
  double *replica = (double *)aligned_alloc(64,sizeof(double) * size * size);

  memcpy((void *) replica, (void *) orig, size * size * sizeof(double));

  return replica;
}
```

Which now improved my code to approximately 940ms.
I Modified the Dgedv file to include some pragma for the icx compiler I obtain now a more consistent runtime of 937ms:
In the screenshot provided below you can use I used a #pragma no vector in the first for loop:
It's because there is a if statement as we are looking for a maximum, vectorization does not suits well in those kind of situation.
For the rest there is no problem that I detected an used both #pragma ivdep and vector always

Scren from dgesvIcxvecto.c

```c
void eliminationGauss(double* matriceA, double* matriceB, int n, int nrhs) {
    for (int col = 0; col < n - 1; col++) {
        // recherche du pivot maximal dans la colonne actuelle
        int max_row = col;
        #pragma novector
        for (int i = col + 1; i < n; i++) {
            if (fabs(matriceA[i* n+col]) > fabs(matriceA[max_row* n+col])) {
                max_row = i;
            }
        }
        // échange des lignes pour mettre le pivot maximal en haut
        #pragma vector always
        for (int j = 0; j < n; j++) {
            double temponA = matriceA[col*n+j];
            matriceA[col*n+j] = matriceA[max_row*n+j];
            matriceA[max_row*n+j] = temponA;
        }
        #pragma vector always
        for (int j = 0; j < nrhs; j++) {
            double temponB = matriceB[col*nrhs+j];
            matriceB[col*nrhs+j] = matriceB[max_row*nrhs+j];
            matriceB[max_row*nrhs+j] = temponB;
        }

        // échelonnement des lignes
        for (int i = col + 1; i < n; i++) {
            double factor = matriceA[i*n+col] / matriceA[col*n+col];
            #pragma ivdep
            #pragma vector always
            for (int j = col; j < n; j++) {
                matriceA[i*n+j] -= factor * matriceA[col*n+j];
            }
            #pragma ivdep
            #pragma vector always
            for (int j = 0; j < nrhs; j++) {
                matriceB[i*nrhs+j] -= factor * matriceB[col*nrhs+j];
            }
        }
    }
}
```

Benchmarking Icx: vectorizations only start as 02 lvl of optimizations

Icx-02: mean time of 943ms
Icx-02-march=native = mean of 904,6 ms
Icx -03: mean time of 938ms
Icx -03 -march=native = mean time of 899,2ms

The code is available in the file dgesvIcxvecto.c example of line to make the binaries:

```
make "CFLAGS= -Wall -O3 -qopt-report=2 -march=native" "CC=icx"
```

For the gcc compiler:

I will need to rewrite the code as the pragma are not relevant for the icx compiler and will instead use pragma omp simd for vectorizations:

First of all, I will add the restrict keyword to the function parameters because the memory regions do not overlap, aiding the vectorizations.

I will also Implement in the EliminationGaus function the #pragma omp simd clause. I will now need to modify the makefile to make sure the openmp libraries is enabled during compilation

For the benchmark, I need to consider when compare that I will always use the flag -march=native and vectorization is only supported with 03 (and 02) lvl.

I modified the makefill and now it look like this, it was necessary to modify more than the CFLAGS to like dgesv.c with the other file of the program:

```
# Default Lapacke: Openblas at CESGA
LDLIBS=-lopenblas

# Other systems (my Debian boxes, for example)
#LDLIBS=-llapacke

# Intel MKL at CESGA
# Module needed: imkl
# => module load openblas
# LDLIBS for intel compiler: icx (module needed: intel)
# Just invoke make like this: make CC=icx
#LDLIBS=-qmkl=sequential -lmkl_intel_lp64

CC = gcc
CFLAGS = -Wall -g

SRCS = dgesv.c main.c timer.c

OBJS = $(SRCS:.c=.o)



dgesv: $(OBJS)
	$(CC) $(CFLAGS) -o $@ $(OBJS) $(LDLIBS) -fopenmp

%.o: %.c
	$(CC) $(CFLAGS) -c -o $@ $< -fopenmp

clean:
	$(RM) dgesv $(OBJS) *~
```

The modified code looks like this and can be found in dgesvvecto.c file:

```c
void eliminationGauss(double* restrict matriceA, double* restrict matriceB, int n, int nrhs) {
    for (int col = 0; col < n - 1; col++) {
        // recherche du pivot maximal dans la colonne actuelle
        int max_row = col;

        for (int i = col + 1; i < n; i++) {
            if (fabs(matriceA[i* n+col]) > fabs(matriceA[max_row* n+col])) {
                max_row = i;
            }
        }
        // échange des lignes pour mettre le pivot maximal en haut

        for (int j = 0; j < n; j++) {
            double temponA = matriceA[col*n+j];
            matriceA[col*n+j] = matriceA[max_row*n+j];
            matriceA[max_row*n+j] = temponA;
        }

        for (int j = 0; j < nrhs; j++) {
            double temponB = matriceB[col*nrhs+j];
            matriceB[col*nrhs+j] = matriceB[max_row*nrhs+j];
            matriceB[max_row*nrhs+j] = temponB;
        }

        // échelonnement des lignes
        //#pragma omp parallel for simd
        for (int i = col + 1; i < n; i++) {
            double factor = matriceA[i*n+col] / matriceA[col*n+col];

          // #pragma omp simd
            for (int j = col; j < n; j++) {
                matriceA[i*n+j] -= factor * matriceA[col*n+j];
            }

           // #pragma omp simd
            for (int j = 0; j < nrhs; j++) {
                matriceB[i*nrhs+j] -= factor * matriceB[col*nrhs+j];
            }
        }
    }
}
```

Benchmark gcc:

Gcc -02: meantime of 960,8ms
Gcc -0 march=native: meantime of 909,4ms
Gcc -03: meantime of 965 ms
Gcc -O3 march=native: meantime of 905,4ms

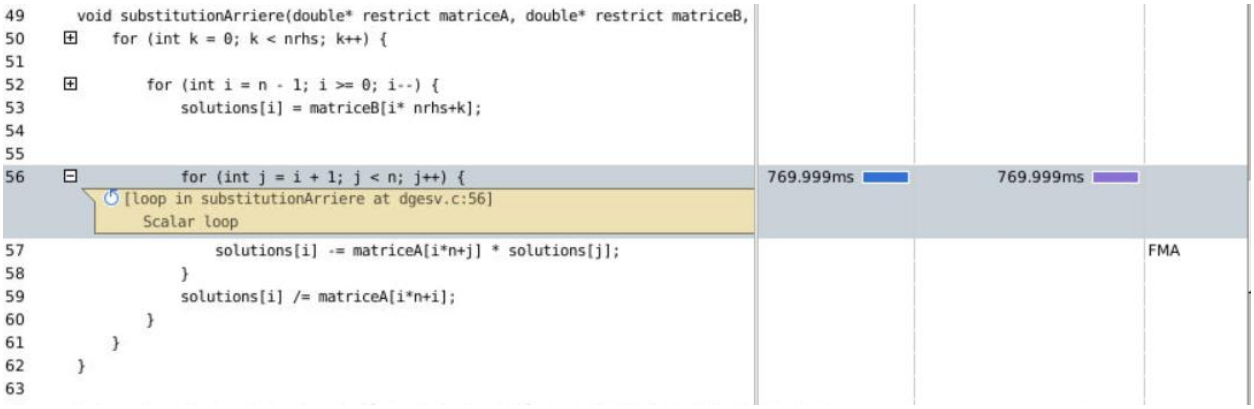
Example of line use to make the binary:

```
make "CFLAGS= -Wall -O3  -march=native -ftree-vectorize -fopt-info-vec"
```

To conclude, I found that Icx is more effective with a optimization level 03 with march=native. However, both compilers have the same performance on the other optimization levels.

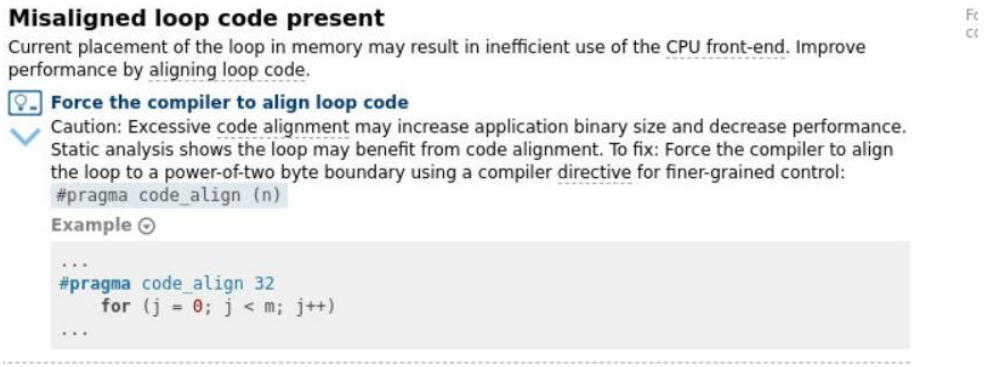
Intel Advisor

After running intel advisor on the latest version of my code, the following result came back:



It informed that this loop was not efficient. It is telling me there is nothing preventing loop vectorization, yet I obtain a higher compute time when I vectorize this part of the code….

It also indicates me to force the compiler to align the code



Intel Advisor does not provide me with more informations concerning the loop that I have vectorized.

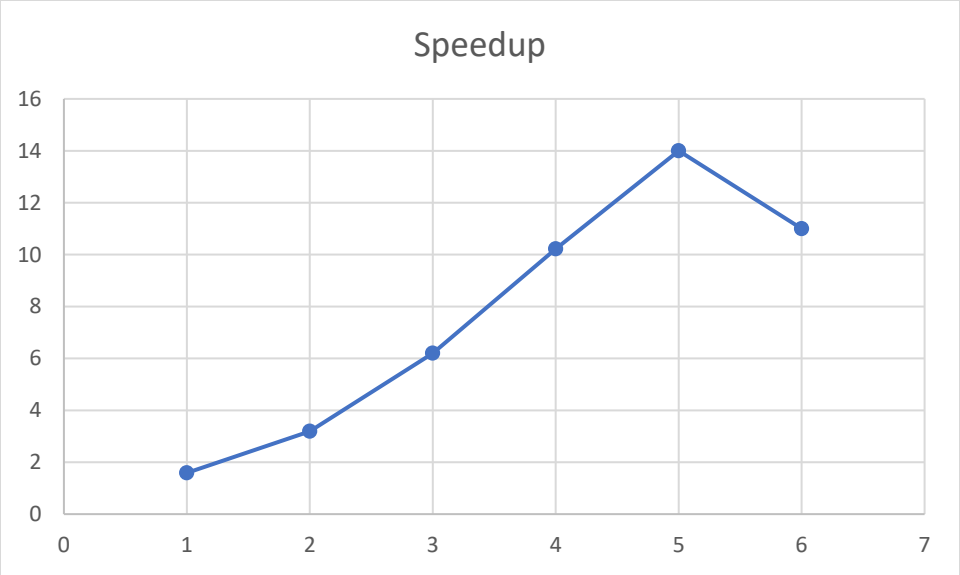
Parallelization

The code is mainly composed of for loop and in one of them we are searching for a max, and there is a open MP clause for that!
Here is the code that can also be find in dgesvOpenMp.c

```
void eliminationGauss(double* restrict matriceA, double* restrict matriceB, int n, int nrhs)
    for (int col = 0; col < n - 1; col++) {
        // recherche du pivot maximal dans la colonne actuelle
        int max_row = col;
        #pragma omp parallel for  reduction(max:max_row)
        for (int i = col + 1; i < n; i++) {
            if (fabs(matriceA[i* n+col]) > fabs(matriceA[max_row* n+col])) {
                max_row = i;
            }
        }
        // échange des lignes pour mettre le pivot maximal en haut
        #pragma omp parallel for
        for (int j = 0; j < n; j++) {
            double temponA = matriceA[col*n+j];
            matriceA[col*n+j] = matriceA[max_row*n+j];
            matriceA[max_row*n+j] = temponA;
        }
        #pragma omp parallel for
        for (int j = 0; j < nrhs; j++) {
            double temponB = matriceB[col*nrhs+j];
            matriceB[col*nrhs+j] = matriceB[max_row*nrhs+j];
            matriceB[max_row*nrhs+j] = temponB;
        }

        // échelonnement des lignes
        #pragma omp parallel for simd
        for (int i = col + 1; i < n; i++) {
            double factor = matriceA[i*n+col] / matriceA[col*n+col];

           #pragma omp simd
            for (int j = col; j < n; j++) {
                matriceA[i*n+j] -= factor * matriceA[col*n+j];
            }

            #pragma omp simd
            for (int j = 0; j < nrhs; j++) {
                matriceB[i*nrhs+j] -= factor * matriceB[col*nrhs+j];
            }
        }
    }
}
```

```
void substitutionArriere(double* restrict matriceA, double* restrict matriceB, double* restrict solutions, int n, int nrhs) {
    #pragma omp parallel for
    for (int k = 0; k < nrhs; k++) {

        for (int i = n - 1; i >= 0; i--) {
            solutions[i] = matriceB[i* nrhs+k];


            for (int j = i + 1; j < n; j++) {
                solutions[i] -= matriceA[i*n+j] * solutions[j];
            }
            solutions[i] /= matriceA[i*n+i];
        }
    }
}
```

When I execute the binary on allocated resources I obtain the following speedup, once again more info In the excel file:



Where the axis corresponds to:
1 = 2 thread
2 = 4
3= 8
4 = 16
5 = 32
6 = 64