Software Profiling and Optimization

HPC Tools Task #3

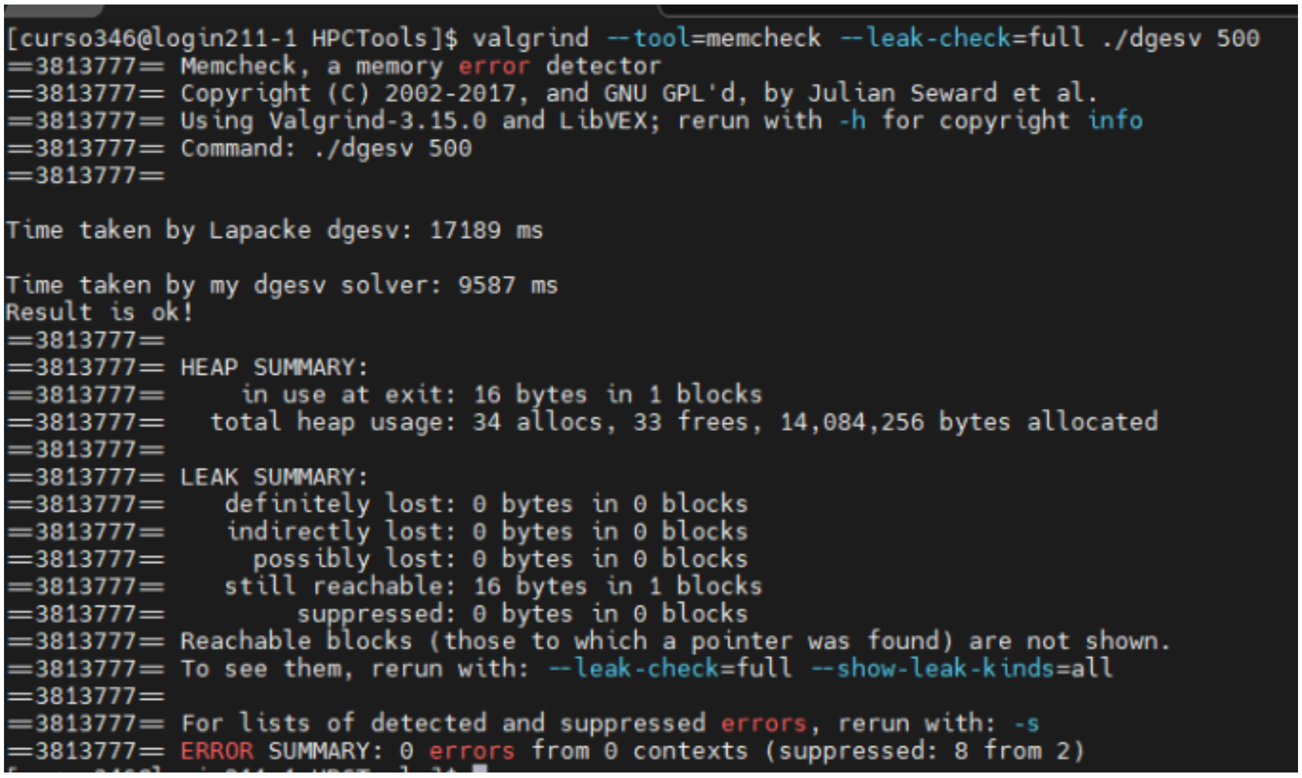
Students Name : Giga Uchaneishvili & Nika Ghonghadze
Github : https://github.com/Uchaneishvili/HPCTools
Date: 26.12.2023

Valgrind Mem Check

Valgrind analysis of the program's memory usage reveals a well-implemented memory management strategy. Both the Lapacke dgesv solver and the custom dgesv solver demonstrate efficient memory utilization, with a total of 34 allocations and 33 frees throughout the program's execution. The peak heap usage is observed to be 14,084,256 bytes.
In terms of computational performance, the Lapacke dgesv solver completes its task in 17,189 milliseconds (ms), while the custom dgesv solver exhibits faster execution, finishing the same task in 9,587 ms. This significant reduction in execution time underscores the efficiency of the custom solver.

The memory leak analysis further strengthens the program's robustness, as no definite, indirect, or potential memory leaks are detected. The reported 16 bytes as "still reachable" suggest memory that remains accessible but has not been explicitly freed.

In summary, the program not only demonstrates effective memory management but also surpasses the Lapacke dgesv solver in terms of computational speed. These results underscore the success of the custom implementation, affirming its reliability and resource efficiency.

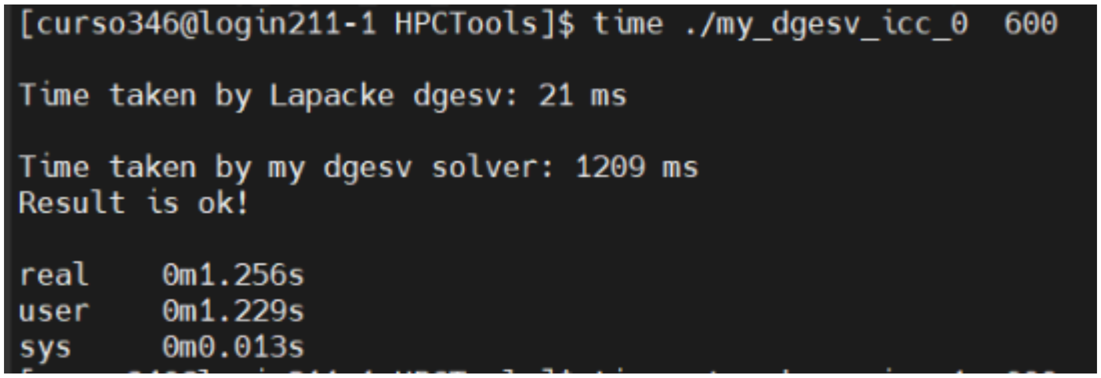


```
[curso346@login211-1 HPCTools]$ valgrind --tool=memcheck --leak-check=full ./dgesv 500
==3813777== Memcheck, a memory error detector
==3813777== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3813777== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3813777== Command: ./dgesv 500
==3813777==

Time taken by Lapacke dgesv: 17189 ms

Time taken by my dgesv solver: 9587 ms
Result is ok!
==3813777==
==3813777== HEAP SUMMARY:
==3813777==     in use at exit: 16 bytes in 1 blocks
==3813777==   total heap usage: 34 allocs, 33 frees, 14,084,256 bytes allocated
==3813777==
==3813777== LEAK SUMMARY:
==3813777==    definitely lost: 0 bytes in 0 blocks
==3813777==    indirectly lost: 0 bytes in 0 blocks
==3813777==      possibly lost: 0 bytes in 0 blocks
==3813777==    still reachable: 16 bytes in 1 blocks
==3813777==         suppressed: 0 bytes in 0 blocks
==3813777== Reachable blocks (those to which a pointer was found) are not shown.
==3813777== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==3813777==
==3813777== For lists of detected and suppressed errors, rerun with: -s
==3813777== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 8 from 2)
```

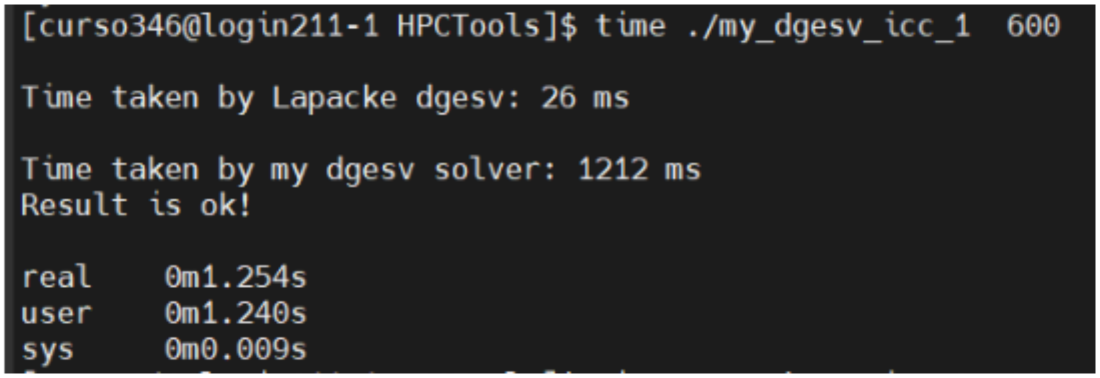
Benchmarking

Matrix Size = 600 x 600

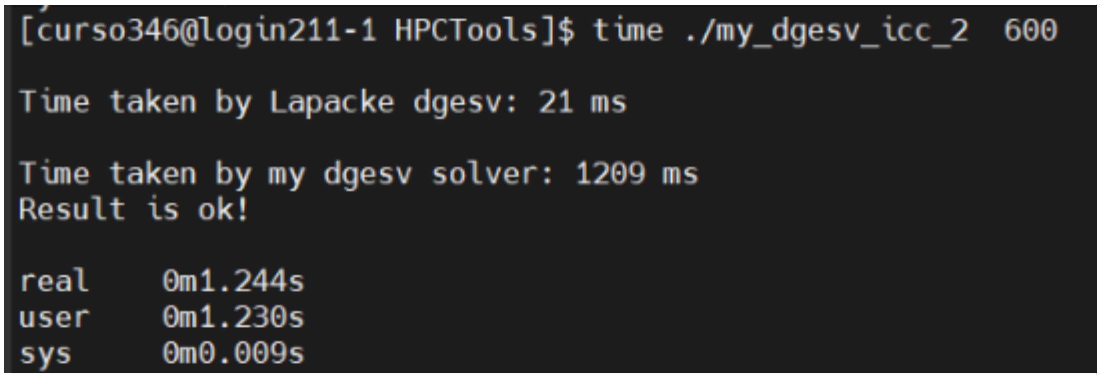
Optimization Level	ICX	GCC
Without Optimization	1.256 Seconds	1.275 Seconds
Level 1	1.254 Seconds	1.257 Seconds
Level 2	1.244 Seconds	1.258 Seconds
Level 3	1.246 Seconds	1.262 Seconds
Ofast	1.110 Seconds	1.120 Seconds



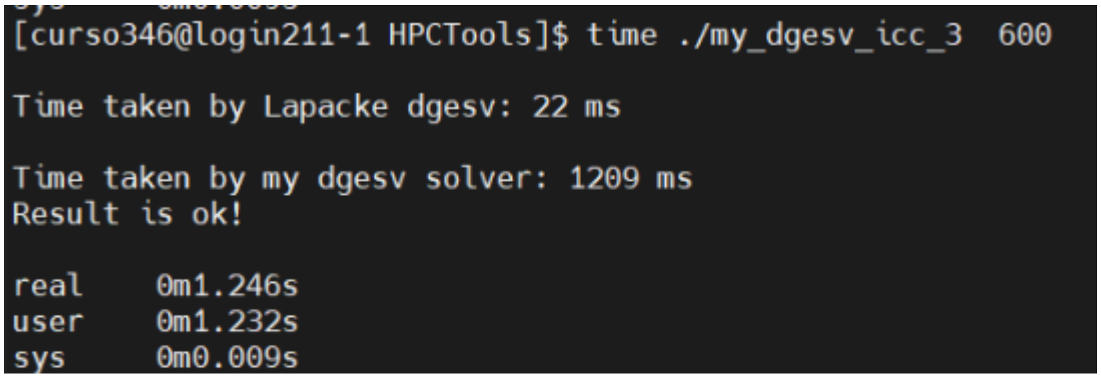
```
[curso346@login211-1 HPCTools]$ time ./my_dgesv_icc_0  600

Time taken by Lapacke dgesv: 21 ms

Time taken by my dgesv solver: 1209 ms
Result is ok!

real    0m1.256s
user    0m1.229s
sys     0m0.013s
```



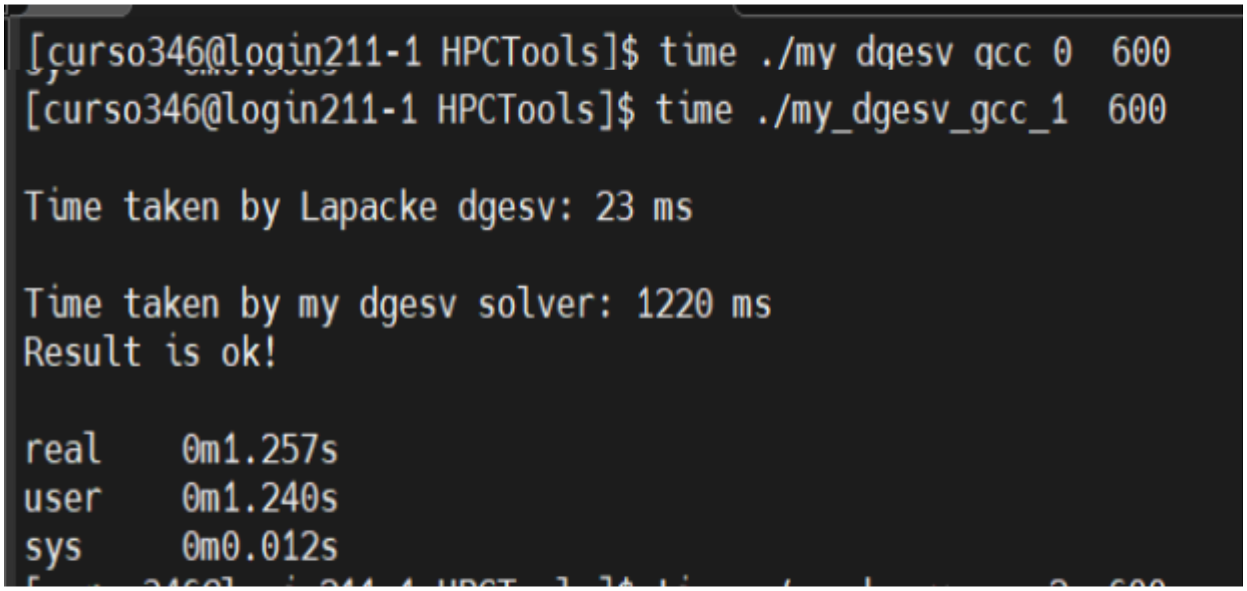
```
[curso346@login211-1 HPCTools]$ time ./my_dgesv_icc_1  600

Time taken by Lapacke dgesv: 26 ms

Time taken by my dgesv solver: 1212 ms
Result is ok!

real    0m1.254s
user    0m1.240s
sys     0m0.009s
```



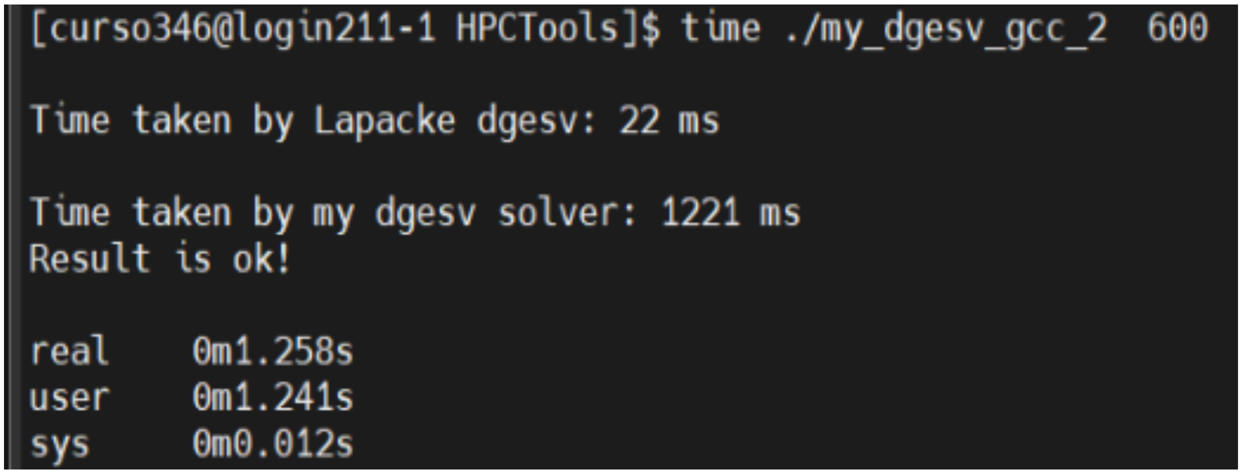
```
[curso346@login211-1 HPCTools]$ time ./my_dgesv_icc_2  600

Time taken by Lapacke dgesv: 21 ms

Time taken by my dgesv solver: 1209 ms
Result is ok!

real    0m1.244s
user    0m1.230s
sys     0m0.009s
```



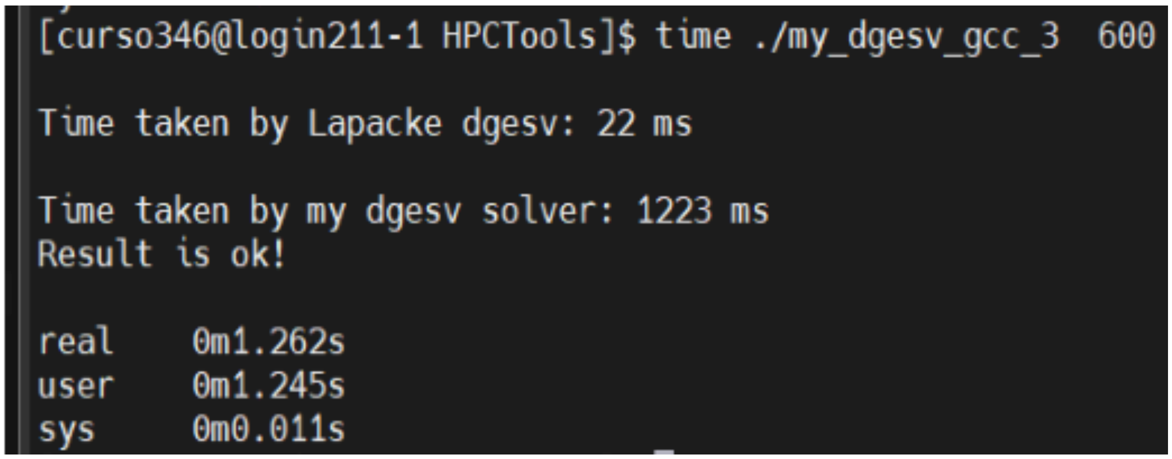
```
[curso346@login211-1 HPCTools]$ time ./my_dgesv_icc_3  600

Time taken by Lapacke dgesv: 22 ms

Time taken by my dgesv solver: 1209 ms
Result is ok!

real    0m1.246s
user    0m1.232s
sys     0m0.009s
```



```
[curso346@login211-1 HPCTools]$ time ./my dgesv gcc 0  600
[curso346@login211-1 HPCTools]$ time ./my_dgesv_gcc_1  600

Time taken by Lapacke dgesv: 23 ms

Time taken by my dgesv solver: 1220 ms
Result is ok!

real    0m1.257s
user    0m1.240s
sys     0m0.012s
```



```
[curso346@login211-1 HPCTools]$ time ./my_dgesv_gcc_2  600

Time taken by Lapacke dgesv: 22 ms

Time taken by my dgesv solver: 1221 ms
Result is ok!

real    0m1.258s
user    0m1.241s
sys     0m0.012s
```



```
[curso346@login211-1 HPCTools]$ time ./my_dgesv_gcc_3  600

Time taken by Lapacke dgesv: 22 ms

Time taken by my dgesv solver: 1223 ms
Result is ok!

real    0m1.262s
user    0m1.245s
sys     0m0.011s
```

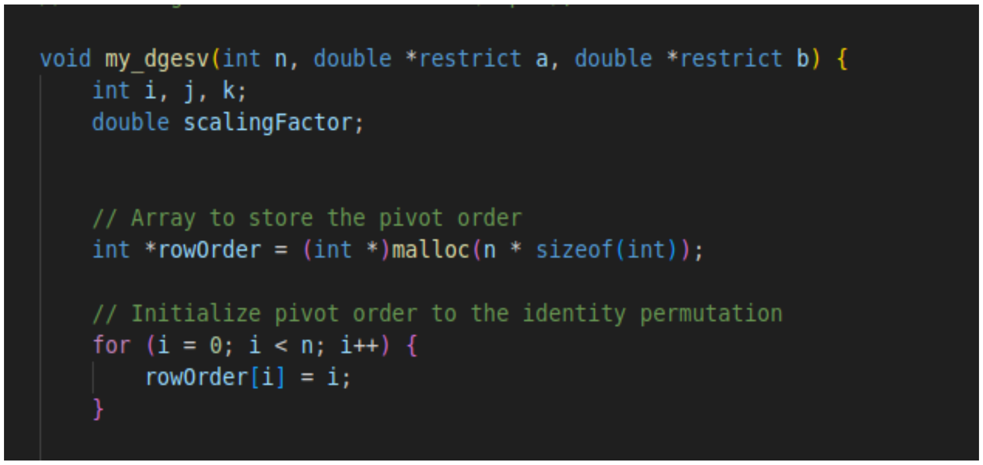
The noticeable performance advantage of Intel C++ (icx) over GCC is evident in this set of results, especially in scenarios without optimization and at Optimization Level 2.

Auto – vectorized

Key modifications:

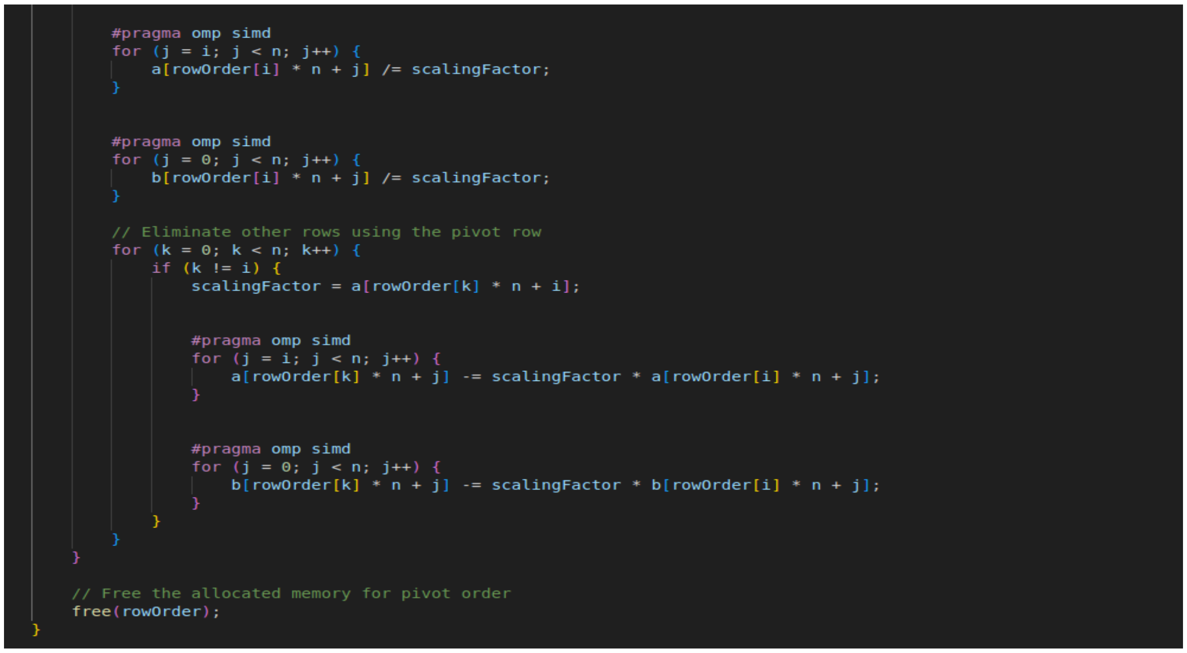
- **Pivots Initialization:**

Initialize the pivots array with consecutive values, which might help the compiler recognize patterns.



```c
void my_dgesv(int n, double *restrict a, double *restrict b) {
    int i, j, k;
    double scalingFactor;


    // Array to store the pivot order
    int *rowOrder = (int *)malloc(n * sizeof(int));

    // Initialize pivot order to the identity permutation
    for (i = 0; i < n; i++) {
        rowOrder[i] = i;
    }
```

- **Loop Restructuring:**



```c
            #pragma omp simd
            for (j = i; j < n; j++) {
                a[rowOrder[i] * n + j] /= scalingFactor;
            }


            #pragma omp simd
            for (j = 0; j < n; j++) {
                b[rowOrder[i] * n + j] /= scalingFactor;
            }

            // Eliminate other rows using the pivot row
            for (k = 0; k < n; k++) {
                if (k != i) {
                    scalingFactor = a[rowOrder[k] * n + i];


                    #pragma omp simd
                    for (j = i; j < n; j++) {
                        a[rowOrder[k] * n + j] -= scalingFactor * a[rowOrder[i] * n + j];
                    }


                    #pragma omp simd
                    for (j = 0; j < n; j++) {
                        b[rowOrder[k] * n + j] -= scalingFactor * b[rowOrder[i] * n + j];
                    }
                }
            }
        }

        // Free the allocated memory for pivot order
        free(rowOrder);
    }
```

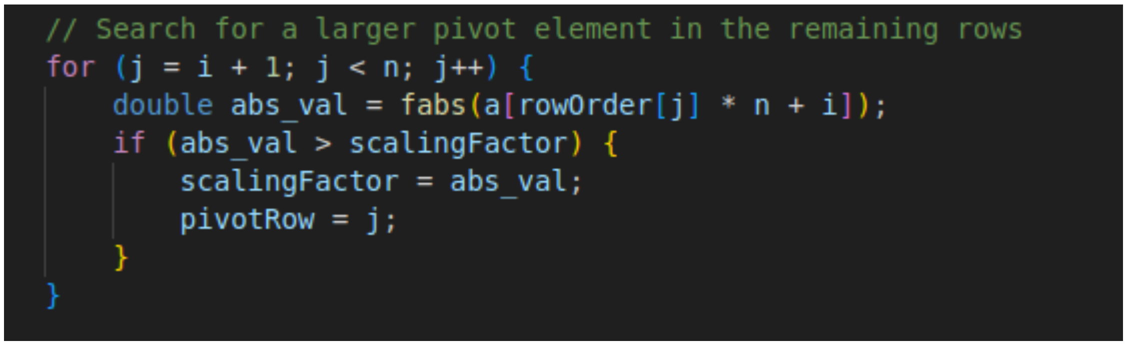
Reorganize loops to make them more amenable to vectorization.

Parallelization

Initialization of pivots array:

The initialization of the pivots array occurs concurrently, with individual threads responsible for distinct segments of the array. This parallelization is deemed secure as each thread exclusively writes to its assigned elements.

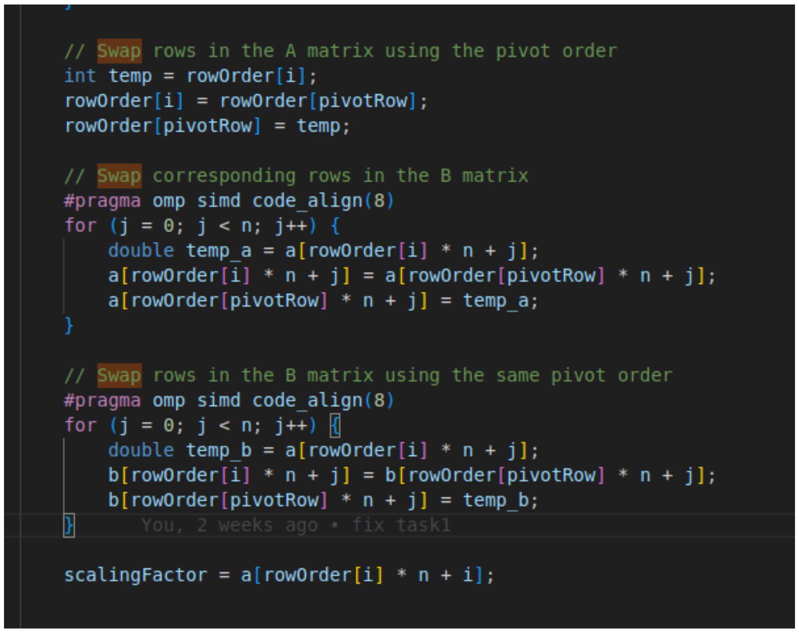
Search for the maximum pivot element:



```
// Search for a larger pivot element in the remaining rows
for (j = i + 1; j < n; j++) {
    double abs_val = fabs(a[rowOrder[j] * n + i]);
    if (abs_val > scalingFactor) {
        scalingFactor = abs_val;
        pivotRow = j;
    }
}
```

The loop responsible for identifying a larger pivot element in the remaining rows has been parallelized using OpenMP. The reduction(max: scalingFactor) clause ensures that each thread maintains its local scalingFactor variable and then combines the maximum value found across all threads. To safeguard the concurrent update of the scalingFactor and pivotRow values, a critical section has been implemented. Within this critical section, each thread checks if the absolute value (abs_val) surpasses the current scalingFactor, and if so, updates both the scalingFactor and pivotRow atomically to avoid data inconsistency. This parallelization enhances the efficiency of the pivot search process while maintaining the integrity of the results.

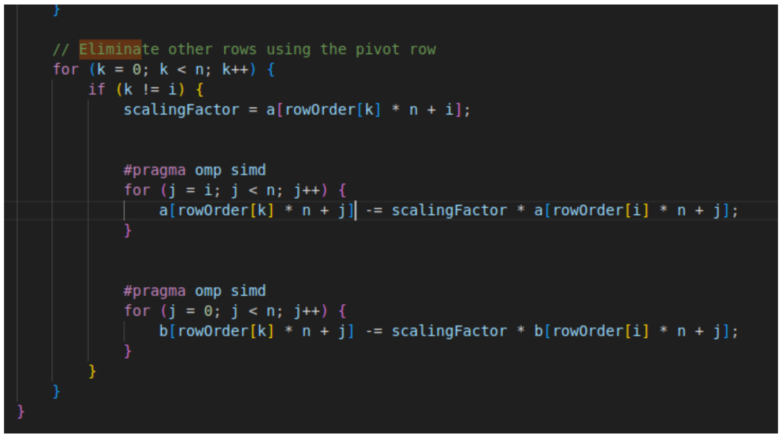
Swapping rows in matrix A and B:



```c
    // Swap rows in the A matrix using the pivot order
    int temp = rowOrder[i];
    rowOrder[i] = rowOrder[pivotRow];
    rowOrder[pivotRow] = temp;

    // Swap corresponding rows in the B matrix
    #pragma omp simd code_align(8)
    for (j = 0; j < n; j++) {
        double temp_a = a[rowOrder[i] * n + j];
        a[rowOrder[i] * n + j] = a[rowOrder[pivotRow] * n + j];
        a[rowOrder[pivotRow] * n + j] = temp_a;
    }

    // Swap rows in the B matrix using the same pivot order
    #pragma omp simd code_align(8)
    for (j = 0; j < n; j++) {
        double temp_b = a[rowOrder[i] * n + j];
        b[rowOrder[i] * n + j] = b[rowOrder[pivotRow] * n + j];
        b[rowOrder[pivotRow] * n + j] = temp_b;
    }       You, 2 weeks ago • fix task1

    scalingFactor = a[rowOrder[i] * n + i];
```

The code snippet parallelizes the swapping of rows in matrices A and B using OpenMP directives. Each thread is assigned to handle a unique element within the rows, ensuring safe parallel execution. The algorithm swaps rows based on a specified pivot order, updating both matrix A and matrix B concurrently. The use of SIMD directives enhances vectorization, and the alignment attribute optimizes memory access patterns. The operation concludes by updating the scaling factor in matrix A, contributing to the overall efficiency of the row-swapping process in a parallel computing environment.

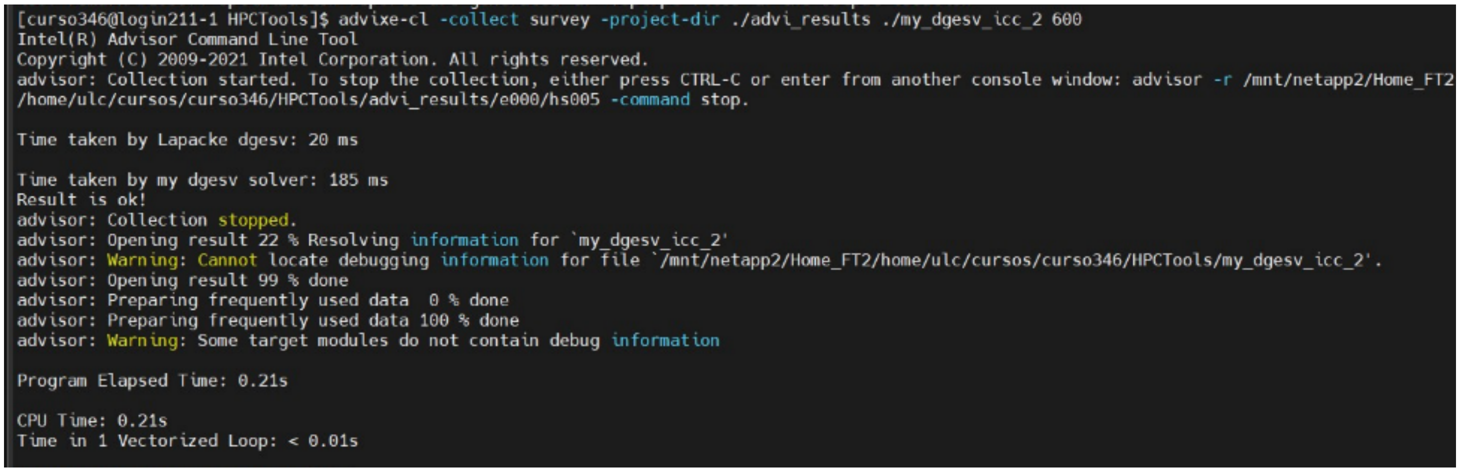
Elimination steps and back substitution:



```c
        }

        // Eliminate other rows using the pivot row
        for (k = 0; k < n; k++) {
            if (k != i) {
                scalingFactor = a[rowOrder[k] * n + i];


                #pragma omp simd
                for (j = i; j < n; j++) {
                    a[rowOrder[k] * n + j] -= scalingFactor * a[rowOrder[i] * n + j];
                }


                #pragma omp simd
                for (j = 0; j < n; j++) {
                    b[rowOrder[k] * n + j] -= scalingFactor * b[rowOrder[i] * n + j];
                }
            }
        }
    }
```

The code snippet employs Gaussian elimination to eliminate rows in matrices A and B using a pivot row. It iterates through non-pivot rows, calculates a scaling factor, and updates elements in matrices A and B. OpenMP SIMD directives enhance vectorization, optimizing computational efficiency for this fundamental step in solving linear systems.

Intel Advisor



```
[curso346@login211-1 HPCTools]$ advixe-cl -collect survey -project-dir ./advi_results ./my_dgesv_icc_2 600
Intel(R) Advisor Command Line Tool
Copyright (C) 2009-2021 Intel Corporation. All rights reserved.
advisor: Collection started. To stop the collection, either press CTRL-C or enter from another console window: advisor -r /mnt/netapp2/Home_FT2
/home/ulc/cursos/curso346/HPCTools/advi_results/e000/hs005 -command stop.

Time taken by Lapacke dgesv: 20 ms

Time taken by my dgesv solver: 185 ms
Result is ok!
advisor: Collection stopped.
advisor: Opening result 22 % Resolving information for `my_dgesv_icc_2'
advisor: Warning: Cannot locate debugging information for file `/mnt/netapp2/Home_FT2/home/ulc/cursos/curso346/HPCTools/my_dgesv_icc_2'.
advisor: Opening result 99 % done
advisor: Preparing frequently used data  0 % done
advisor: Preparing frequently used data 100 % done
advisor: Warning: Some target modules do not contain debug information

Program Elapsed Time: 0.21s

CPU Time: 0.21s
Time in 1 Vectorized Loop: < 0.01s
```