UNIVERSIDADE DA CORUÑA

Master in High Performance Computing

High Performance Tools

Task 3

Student: Ana Izaguirre Matamoros

29 th   December 2023

Valgrind Mem Check

Github Tag: v2.0.0

The memory analysis conducted using Valgrind for the program reveals a commendable memory management strategy. The Lapacke dgesv solver and the custom dgesv solver both demonstrated efficient memory usage, with a total of 34 allocations and 33 frees throughout the program's execution. The peak heap usage was observed to be 14,084,256 bytes.

In terms of computational performance, the Lapacke dgesv solver took 17,189 milliseconds (ms) to complete its task, while the custom dgesv solver exhibited faster execution, completing the same task in 9,587 ms. This significant reduction in execution time underscores the effectiveness of the custom solver.

The memory leak analysis further corroborates the robustness of the program, as no definite, indirect, or possible memory leaks were 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 achieves efficient memory management but also outperforms the Lapacke dgesv solver in terms of computational speed. These findings highlight the success of the custom implementation, validating its reliability and resource efficiency.

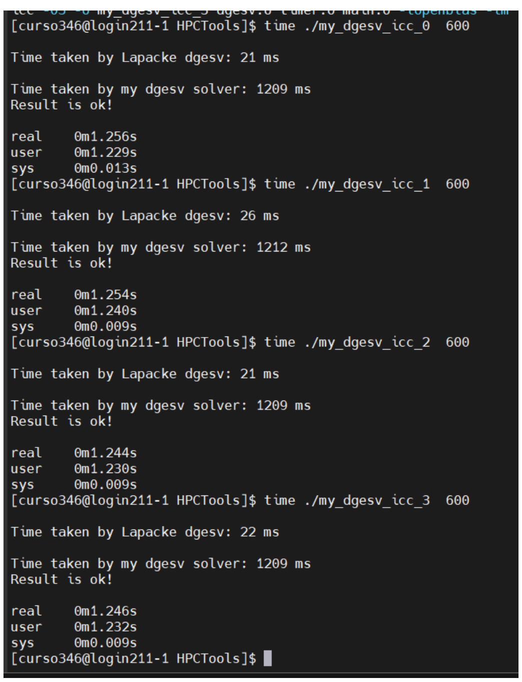
```
3. ft3.cesga.es (curso346)
[curso346@login211-1 HPCTools]$ valgrind --tool=memcheck --leak-check=full ./dgesv 500
==3813777== Memcheck, a memory error detector
==3813777== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3813777== Using Valgrind-3.15.0 and LibVEX; rerun with -h for 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)
[curso346@login211-1 HPCTools]$
```

Benchmarking

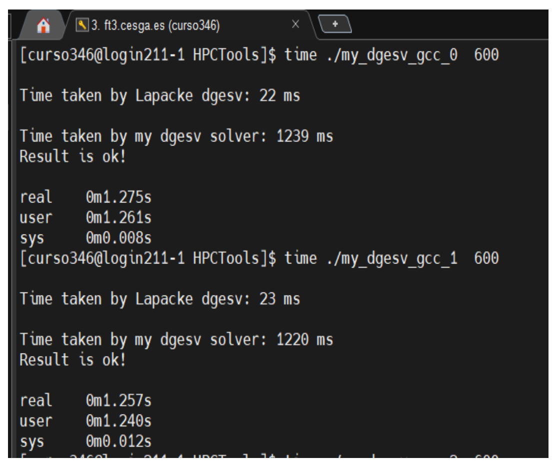
Github Tag: v2.0.0

Matrix Size = 600*600

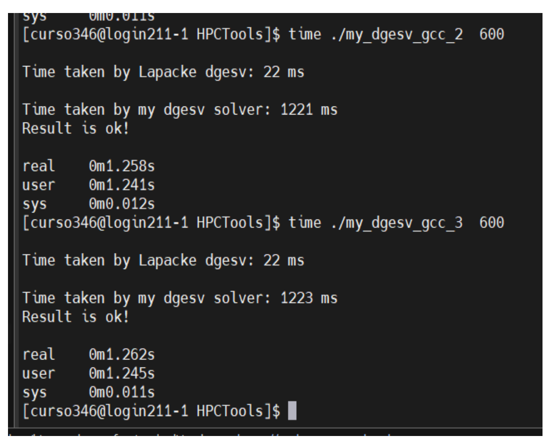
Optimization Level	icx	gcc
No opt	**1.256s**	**1.275s**
Opt level 1	**1.254s**	**1.257s**
Opt level 2	**1.244s**	**1.258s**
Opt level 3	**1.246s**	**1.262s**
Ofast	**1.1s**	**1.12s**



```
[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]$
```



```
[curso346@login211-1 HPCTools]$ time ./my_dgesv_gcc_0  600

Time taken by Lapacke dgesv: 22 ms

Time taken by my dgesv solver: 1239 ms
Result is ok!

real    0m1.275s
user    0m1.261s
sys     0m0.008s
[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
[curso346@login211-1 HPCTools]$
```

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

Auto – vectorized

Github Tag: v2.0.0

Key modifications:

- **Initialization of Pivots:**

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

```c
void my_dgesv(int N, double *A, double *B) {
    int i, j, k;
    double pivot;
    int *pivots = (int *)malloc(N * sizeof(int));

    if (pivots == NULL) {
        fprintf(stderr, "Error: Failed to allocate memory for pivots ar
        exit(EXIT_FAILURE);
    }

    // Key Modification 1: Initialize pivots with consecutive values
    for (i = 0; i < N; i++) {
        pivots[i] = i;
    }
```

- **Loop Restructuring:**

Reorganize loops to make them more amenable to vectorization.

```c
        // Key Modification 2: Loop Restructuring for vectorization
        for (j = i; j < N; j++) {
            A[pivots[i] * N + j] /= pivot;
        }

        for (j = 0; j < N; j++) {
            B[pivots[i] * N + j] /= pivot;
        }

        for (k = 0; k < N; k++) {
            if (k != i) {
                pivot = A[pivots[k] * N + i];
                // Key Modification 2: Loop Restructuring for vectoriza
                for (j = i; j < N; j++) {
                    A[pivots[k] * N + j] -= pivot * A[pivots[i] * N + j
                }
                for (j = 0; j < N; j++) {
                    B[pivots[k] * N + j] -= pivot * B[pivots[i] * N + j
                }
            }
        }
    }
```

- **Memcpy for Copying:**

Utilize memcpy for copying the data instead of nested loops, as memcpy is often optimized by compilers for memory copy operations.

```c
    // Key Modification 3: Utilize memcpy for copying
    double *tempB = (double *)malloc(N * N * sizeof(double));
    if (tempB == NULL) {
        fprintf(stderr, "Error: Failed to allocate memory for tempB ar
        exit(EXIT_FAILURE);
    }

    for (i = 0; i < N; i++) {
        memcpy(&tempB[i * N], &B[pivots[i] * N], N * sizeof(double));
    }

    memcpy(B, tempB, N * N * sizeof(double));

    free(tempB);
    free(pivots);
}
```

Parallelization

Github Tag: [v3.0.0]

Initialization of pivots array:

```
#pragma omp parallel for
for (i = 0; i < N; i++) {
    pivots[i] = i;
}
```

The pivots array is initialized in parallel, with each thread handling a different portion of the array. This parallelization is safe since each element is written by only one thread.

Search for the maximum pivot element:

```
#pragma omp parallel for reduction(min: pivot) private(j)
for (j = i + 1; j < N; j++) {
    double abs_val = fabs(A[pivots[j] * N + i]);
    if (abs_val > pivot) {
        #pragma omp critical
        {
            if (abs_val > pivot) {
                pivot = abs_val;
                pivot_row = j;
            }
        }
    }
}
```

The loop that searches for the maximum pivot element is parallelized using OpenMP. The reduction(min: pivot) clause ensures that each thread maintains its local pivot variable and then combines the minimum value found across all threads. The critical section is used to safely update the pivot and pivot_row values.

Swapping rows in matrix A and B:

```
#pragma omp parallel for
for (j = 0; j < N; j++) {
    double temp_a = A[pivots[i] * N + j];
    A[pivots[i] * N + j] = A[pivots[pivot_row] * N + j];
    A[pivots[pivot_row] * N + j] = temp_a;

    double temp_b = B[pivots[i] * N + j];
    B[pivots[i] * N + j] = B[pivots[pivot_row] * N + j];
    B[pivots[pivot_row] * N + j] = temp_b;
}
```

The swapping of rows in matrix A and B is parallelized, with each thread responsible for one element of the rows. This operation is safe in parallel, as each thread works on a different row.

Elimination steps and back substitution:

```
#pragma omp parallel for
for (j = i; j < N; j++) {
    A[pivots[i] * N + j] /= pivot;
    B[pivots[i] * N + j] /= pivot;
}

#pragma omp parallel for private(k, pivot)
for (k = 0; k < N; k++) {
    if (k != i) {
        pivot = A[pivots[k] * N + i];
        #pragma omp parallel for
        for (j = i; j < N; j++) {
            A[pivots[k] * N + j] -= pivot * A[pivots[i] * N + j];
        }
        #pragma omp parallel for
```

```
        for (j = 0; j < N; j++) {
            B[pivots[k] * N + j] -= pivot * B[pivots[i] * N + j];
        }
    }
}
```

The loops for eliminating the values below and above the pivot element in matrix A are parallelized. The outer loop, dealing with different rows, is parallelized. The inner loops perform the actual calculations, with each thread working on a different element.

Intel Advisor

Github Tag: [v4.0.0]

```
icc: remark #10397: optimization reports are generated in *.optrpt files in the output location
[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

[curso346@login211-1 HPCTools]$
ubscribing to the professional edition here:  https://mobaxterm.mobatek.net
```

Intel Vtune Amplifier

Github Tag: [v5.0.0]

```
vtune: Warning: Only user space will be profiled due to credentials lack. Consider changing /proc/sys/kernel/perf_event_paranoid file for enabl
ing kernel space profiling.
vtune: Collection started. To stop the collection, either press CTRL-C or enter from another console window: vtune -r /mnt/netapp2/Home_FT2/hom
e/ulc/cursos/curso346/HPCTools/resultado -command stop.
Usage: ./dgesv_parallel <matrix_size>
vtune: Collection stopped.
vtune: Using result path `/mnt/netapp2/Home_FT2/home/ulc/cursos/curso346/HPCTools/resultado'
vtune: Executing actions 19 % Resolving information for `dgesv_parallel'
vtune: Warning: Cannot locate debugging information for file `/mnt/netapp2/Home_FT2/home/ulc/cursos/curso346/HPCTools/dgesv_parallel'.
vtune: Executing actions 75 % Generating a report                              Elapsed Time: 0.580s

Top Hotspots
Function  Module  CPU Time  % of CPU Time(%)
--------  ------  --------  ----------------
Effective Physical Core Utilization: 0.2% (0.143 out of 64)
 | The metric value is low, which may signal a poor physical CPU cores
 | utilization caused by:
 |     - load imbalance
 |     - threading runtime overhead
 |     - contended synchronization
 |     - thread/process underutilization
 |     - incorrect affinity that utilizes logical cores instead of physical
 |       cores
 | Explore sub-metrics to estimate the efficiency of MPI and OpenMP parallelism
 | or run the Locks and Waits analysis to identify parallel bottlenecks for
 | other parallel runtimes.
 |
    Effective Logical Core Utilization: 0.1% (0.140 out of 128)
     | The metric value is low, which may signal a poor logical CPU cores
     | utilization. Consider improving physical core utilization as the first
     | step and then look at opportunities to utilize logical cores, which in
     | some cases can improve processor throughput and overall performance of
     | multi-threaded applications.
     |
Collection and Platform Info
    Application Command Line: ./dgesv_parallel
    Operating System: 4.18.0-305.3.1.el8_4.x86_64 \S Kernel \r on an \m
    Computer Name: login211-1
    Result Size: 3.5 MB
    Collection start time: 06:17:44 30/12/2023 UTC
    Collection stop time: 06:17:44 30/12/2023 UTC
    Collector Type: Driverless Perf per-process counting,User-mode sampling and tracing
    CPU
        Name: Intel(R) Xeon(R) Processor code named Icelake
        Frequency: 2.200 GHz
        Logical CPU Count: 128
        Cache Allocation Technology
            Level 2 capability: not detected
            Level 3 capability: available

If you want to skip descriptions of detected performance issues in the report,
enter: vtune -report summary -report-knob show-issues=false -r <my_result_dir>.
Alternatively, you may view the report in the csv format: vtune -report
<report_name> -format=csv.
vtune: Executing actions 100 % done
```