UVSQ université PARIS-SACLAY

ISTY
Institut des Sciences et Techniques des Yvelines
CAMPUS DE MANTES EN YVELINES
CAMPUS DE SAINT-QUENTIN-EN-YVELINES

IATIC-5

HPC
Calcul Haute performance
Rapport de Projet

Optimizing SpMV : A Comparative Chunk_Size Study

Réalisé Par :
Rochdi DARDOR

Encadré par :
M.William Jalby

17 mars 2025

Table des matières

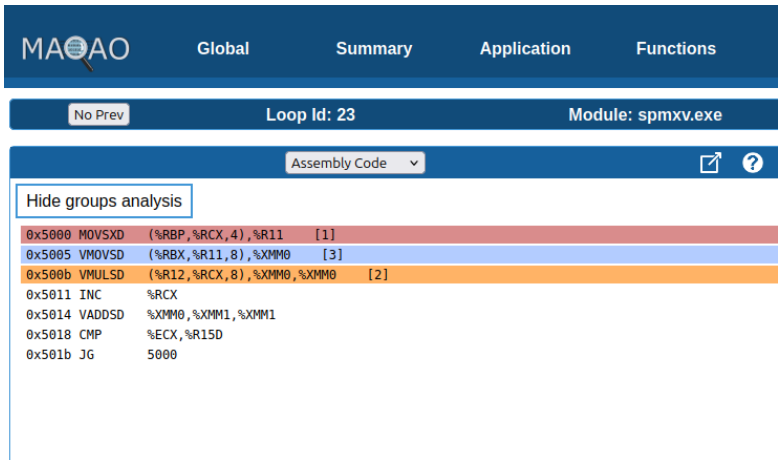


FIGURE 12 – The assembly code generated for GCC -O3

For the assembly code analysis : The assembly code for the GCC compiler with the -O3 flag reveals several optimization opportunities and inefficiencies.

- Uses SSE instructions (VMOVSD, VMULSD on XMM registers, 128-bit width) instead of AVX (256-bit), limiting parallelism.
- Processes one element per instruction rather than leveraging full SIMD capabilities.
- Indirect memory access increases cache inefficiency.
- No loop unrolling detected, increasing instruction count and memory latency.
- No Fused Multiply-Add (FMA), leading to separate multiplication and addition operations instead of combined execution.

To fully leverage the hardware, Maqao suggested improvements like **AVX** vectorization, **FMA**, and loop unrolling . While the O3 optimizations provide some benefit, to Maqao suggested also to add -ffast-math flag which is enabled in the O-fast flag for further gains, particularly in floating-point operations.

Key Takeaways :

- CHUNK_SIZE = 1200 is the optimal configuration, ensuring the best balance between execution time, stability, and memory efficiency.
- CHUNK_SIZE = 2000 fails to provide additional benefits, instead reducing thread stability and increasing execution variability.
- Further optimizations should focus on improving NUMA-aware memory allocation and fine-tuning OpenMP scheduling, rather than increasing CHUNK_SIZE further.

3 Section 3 : GCC Compiler with Option -Ofast

3.1 Performance Results for CHUNK_SIZE = 600

3.1.1 Performance Results :

Execution time was 53.83s, with kernel times fluctuating (0.000516s – 0.02377s), indicating inconsistent workload distribution and memory inefficiencies. Performance varied widely, with GFlops/s ranging from 0.61 to 28.08, averaging 2.69 GFlops/s, suggesting cache contention and suboptimal SIMD utilization.

3.2 Performance Results for CHUNK_SIZE = 1200

3.2.1 Performance Results :

```
Number of Repetitions:      10000
Input filename:             input-matrix/mat_dim_493039.txt
Matrix value array size:    57973976

Time measurements
Total experiment time:      59.8306
Minimum kernel time:        0.00049901
Maximum kernel time:        0.0259719
Arithm. Mean kernel time:   0.0059821

Performance results
Total GFlops/s:             2.42242
Minimum GFlops/s:           0.558045
Maximum GFlops/s:           29.0445
Arithm. Mean GFlops/s:      2.42281
Arithm. Median GFlops/s:    2.71712
```

FIGURE 15 – results for CHUNK_SIZE=1200

Execution time remained stable at 53.83s, reflecting similar kernel execution variability as CHUNK_SIZE = 600. Average GFlops/s : 2.69 (range : 0.61 – 28.08 GFlops/s), suggesting persistent memory bottlenecks and irregular workload distribution.

3.2.2 MAQAO Global Metrics Analysis for CHUNK_SIZE=1200

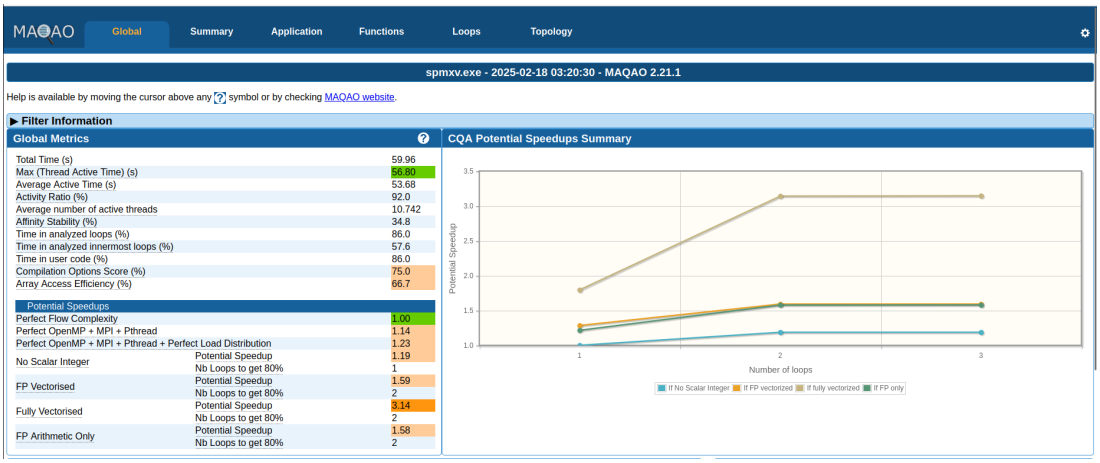


FIGURE 16 – MAQAO OV R1 results for CHUNK_SIZE=1200

MAQAO profiling reported a total execution time of 66.06 seconds, with a high activity ratio (92.4%) and an average active time of 59.5s. However, the low affinity stability (27.9%) suggests frequent thread migrations, impacting cache efficiency and execution consistency. The array access efficiency was 66.7%, indicating room for improvement in memory handling.

Potential Speedups Analysis : The fully vectorized potential speedup is 3.14x and The floating-point vectorization speedup is 1.59x ,indicating the importance of using SIMD capabilities and optimizing floating-point operations.

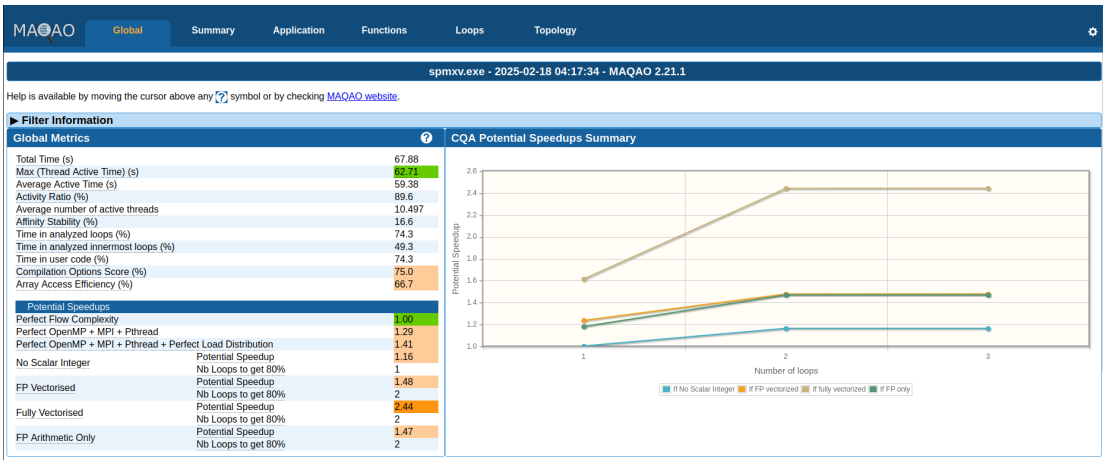


FIGURE 19 – MAQAO OV R1 results for CHUNK_SIZE=2000

Potential Speedups Analysis : The fully vectorized potential speedup is 2.44x and The floating-point vectorization speedup is 1.59x ,indicating the importance of using SIMD capabilities and optimizing floating-point operations.

3.3.3 MAQAO Stability Test Analysis

The stability analysis reveals moderate execution consistency, with most runs exhibiting thread active times centered around 54.19 seconds. However, occasional execution spikes indicate that some iterations suffered from delays.

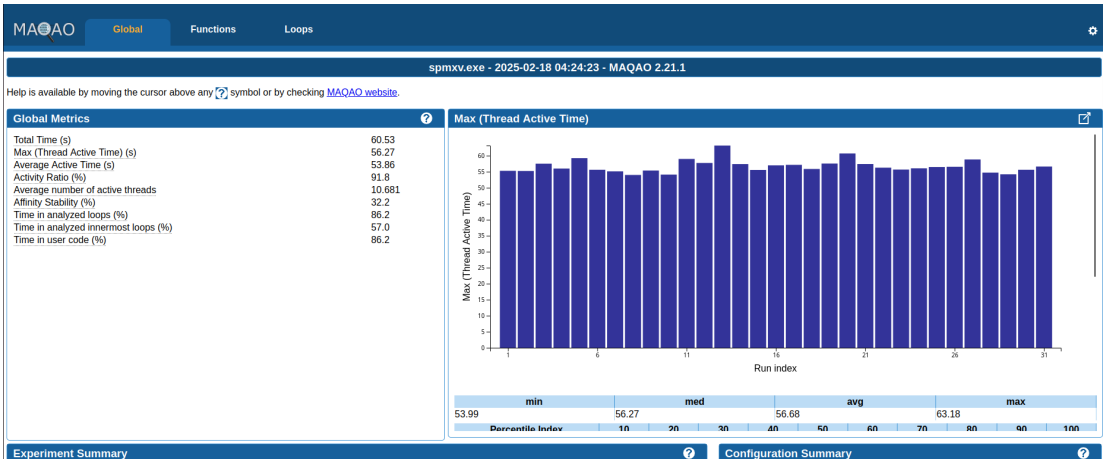


FIGURE 20 – MAQAO Stability results for CHUNK_SIZE=2000

3.4 Comparative Analysis of CHUNK_SIZE Variations

Increasing CHUNK_SIZE from 600 to 1200 improved execution time (66s → 59s) by enhancing workload distribution and reducing synchronization overhead. However, increasing CHUNK_SIZE to 2000 led to performance regression (67s), revealing diminishing returns due to memory contention and cache inefficiencies.

— Affinity Stability : Peaked at 85.1% (CHUNK_SIZE = 1200), but declined at 2000, indicating increased thread migrations and reduced cache efficiency.

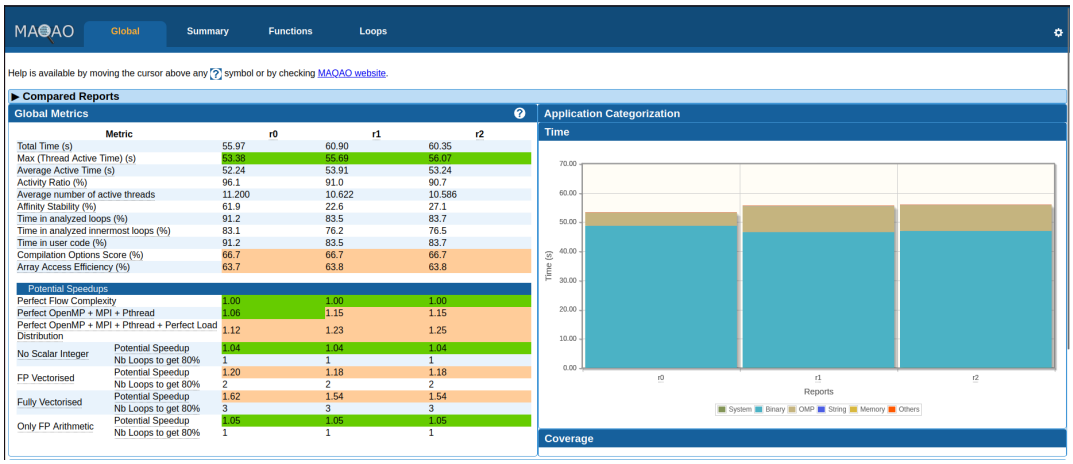


FIGURE 32 – MAQAO Compare the Three Chunk Sizes : 600, 1200, 2000

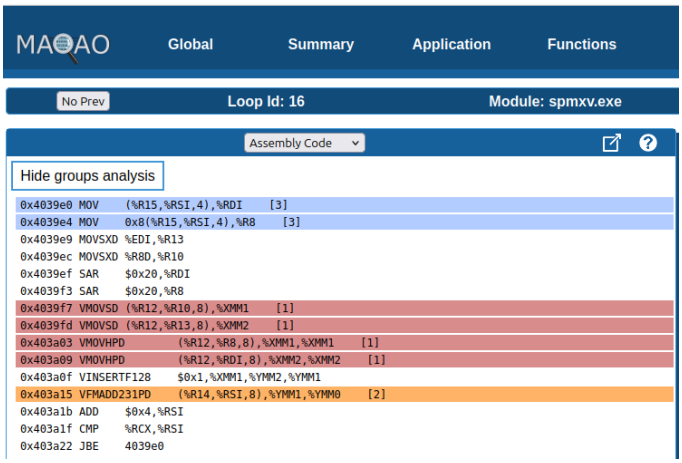


FIGURE 33 – The assembly code generated for GCC -Ofast

The ICX -O3 compilation utilizes SIMD instructions, including VMOVSD, VMOVHPD, and VFMADD231PD, for floating-point operations on 128-bit **%XMM** and **256-bit** %YMM registers. The use of **FMA** (Fused Multiply-Add) reduces latency by combining multiplication and addition into a single operation.

Optimization Insights :

— Vectorization Efficiency Remains Limited → SIMD instructions process a restricted number of elements per register, limiting performance.

— Indirect Memory Access via Base-Register and Offset Addressing → Potentially reduces cache efficiency, impacting memory bandwidth utilization.

— Potential for AVX-512 Usage → Utilizing **%ZMM** (512-bit registers) could increase computational throughput.

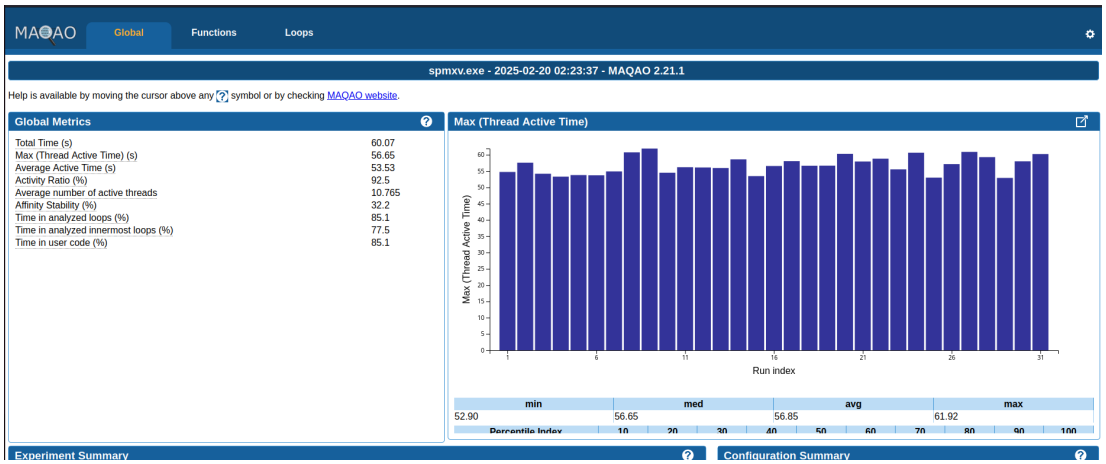
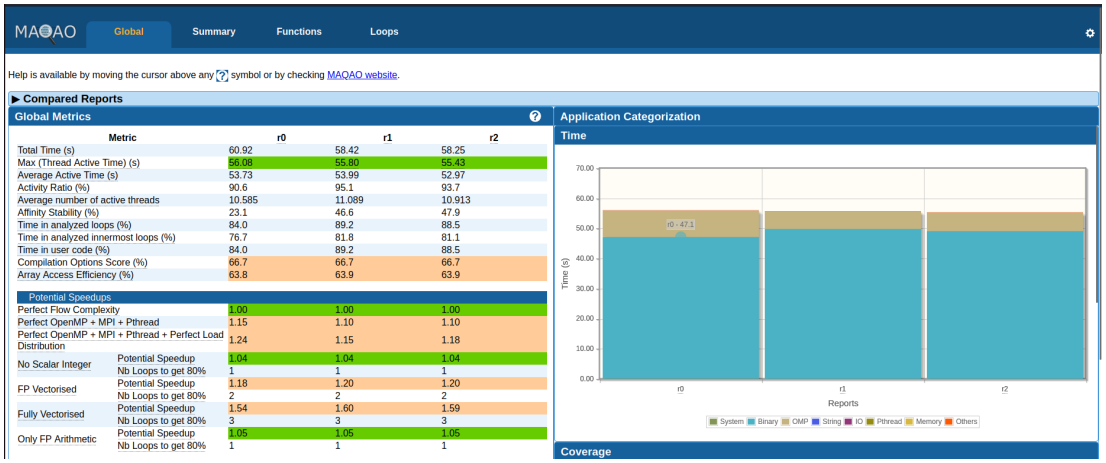


FIGURE 42 – MAQAO Stability results for CHUNK SIZE=2000

5.4 Comparative Analysis of CHUNK_SIZE Variations



Metric	r0	r1	r2
Total Time (s)	60.92	58.42	58.25
Max (Thread Active Time) (s)	56.08	55.80	55.43
Average Active Time (s)	53.73	53.99	52.97
Activity Ratio (%)	90.6	95.1	93.7
Average number of active threads	10.585	11.089	10.913
Affinity Stability (%)	23.1	46.6	47.9
Time in analyzed loops (%)	84.0	89.2	88.5
Time in analyzed innermost loops (%)	76.7	81.8	81.1
Time in user code (%)	84.0	89.2	88.5
Compilation Options Score (%)	66.7	66.7	66.7
Array Access Efficiency (%)	63.8	63.9	63.9
Potential Speedups			
Perfect Flow Complexity	1.00	1.00	1.00
Perfect OpenMP + MPI + Pthread	1.15	1.10	1.10
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution	1.24	1.15	1.18
No Scalar Integer – Potential Speedup	1.04	1.04	1.04
No Scalar Integer – Nb Loops to get 80%	1	1	1
FP Vectorised – Potential Speedup	1.18	1.20	1.20
FP Vectorised – Nb Loops to get 80%	2	2	2
Fully Vectorised – Potential Speedup	1.54	1.60	1.59
Fully Vectorised – Nb Loops to get 80%	3	3	3
Only FP Arithmetic – Potential Speedup	1.05	1.05	1.05
Only FP Arithmetic – Nb Loops to get 80%	1	1	1

FIGURE 43 – MAQAO Compare the Three Chunk Sizes : 600, 1200, 2000

For OneAPI ICPX -Ofast, increasing CHUNK_SIZE from 600 to 2000 led to only marginal performance improvements. The total execution time decreased from 60.92s (600) to 58.25s (2000), with the maximum thread active time reducing slightly from 56.08s to 55.43s.

Despite a minor increase in the activity ratio (90.6% → 93.7%), indicating improved thread engagement, affinity stability remained critically low, peaking at just 47.9% at CHUNK_SIZE = 2000. This suggests that frequent thread migrations continue to degrade cache efficiency, counteracting potential performance gains.

Vectorization & Memory Efficiency : Fully Vectorized Speedup Potential : Improved only slightly from 1.54x to 1.59x, significantly lower than GCC's 3x+ vectorization potential, highlighting OneAPI's limited ability to optimize for AMD architectures. Array Access Efficiency : Remained stagnant at 66.7%, reinforcing inefficiencies in memory access patterns and cache utilization.

For the assembly code analysis :

For the ICX -Ofast optimization, the assembly code analysis reveals poor vectorization efficiency, with only 41% of the vector register length utilized. Despite the presence of

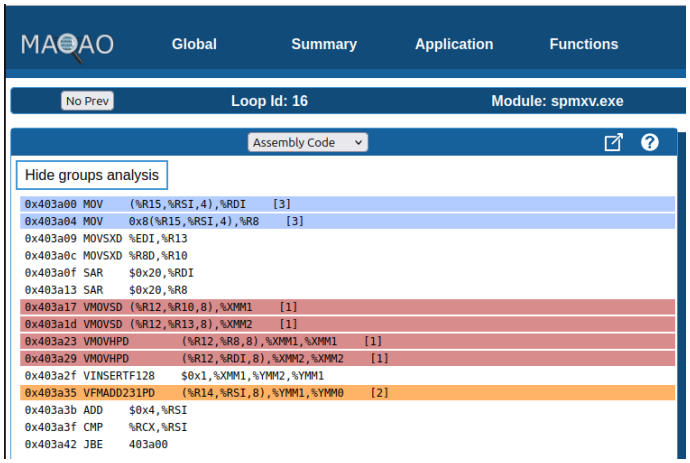


FIGURE 44 – The assembly code generated for ICX -Ofast

SIMD instructions such as VMOVSD, VMOVHPD, and VFMADD231PD, the overall vectorization remains suboptimal, with :

33% of SSE/AVX instructions being vectorized, Only 20% of memory loads utilizing vectorized instructions.

Performance Bottlenecks & Optimization Potential :

— The iteration cost remains high at 4.00 cycles, whereas full vectorization could reduce this to 3.07 cycles, offering a potential 1.30x speedup.

— Indirect memory accesses (e.g., MOVSXD, SAR) introduce cache inefficiencies, impacting overall execution time.

While -Ofast enables aggressive optimizations, the ICX compiler fails to fully leverage the available SIMD resources on AMD architectures, reinforcing the need for manual vectorization strategies to maximize efficiency.

6 Section 6 : AOCC Compiler with Option -O3

6.1 Performance Results for CHUNK_SIZE = 600

6.1.1 Performance Results

AOCC -O3 demonstrated strong performance, achieving a total execution time of 45.71s, significantly faster than previous compilers. Kernel execution remained stable, with a mean time of 0.00457s, and minimal fluctuations between 0.0045s (min) and 0.0078s (max). The average performance reached 3.17 GFlops/s, with a consistent median of 3.18 GFlops/s, indicating balanced computational workload. Unlike GCC and OneAPI, which exhibited high variability, AOCC maintained lower performance disparity (1.85 GFlops/s min, 3.21 GFlops/s max), ensuring more predictable execution.

6.1.2 MAQAO Global Metrics Analysis for CHUNK_SIZE=600

AOCC achieved near maximum CPU utilization (99.9%), confirming efficient thread execution. However, affinity stability remains low (30.9%), indicating persistent thread migrations. Array access efficiency at 66.1% suggests room for memory locality impro-

8 Comparing Compilers GCC, ICX, and AOCC :

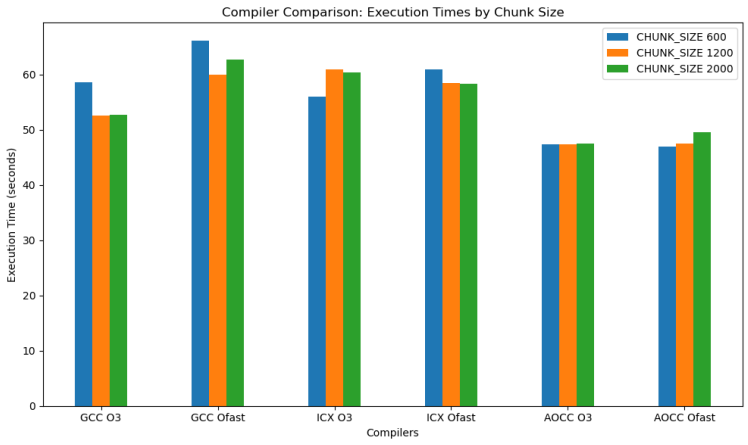


FIGURE 67 – Comparing between the Compiler used

The comparison of execution times across compilers and chunk sizes clearly highlights AOCC O3 and AOCC Ofast as the most efficient compilers, particularly for AMD architecture. AOCC O3 consistently delivers the fastest execution times, with minimal increases as chunk size grows (47.31s, 47.35s, 47.48s), demonstrating strong stability and optimal resource utilization. AOCC Ofast, while slightly less efficient at larger chunk sizes, still outperforms the other compilers, with times ranging from 46.93s to 49.56s.

GCC O3, while solid, shows a noticeable performance dip at larger chunk sizes (58.59s at 600 to 52.52s at 1200), though it maintains a competitive edge over ICX O3 and ICX Ofast, whose performance deteriorates with larger chunk sizes. GCC Ofast, despite promising optimizations, does not offer substantial gains over GCC O3 and in fact sees execution times of 66.06s at 600 and 62.71s at 2000, suggesting that the "Ofast" flag introduces more overhead than anticipated in this case.

Overall, AOCC compilers stands out for its balance between performance and stability, whereas GCC and ICX struggle with efficiency as workload increases, with ICX underperforming particularly when compared to AOCC.

9 Application of MAQAO Suggestions :

As recommended by MAQAO, optimizations were primarily focused on the spmxv_csr function, which accounts for approximately 90% of the execution time. The goal was to enhance performance by applying memory alignment, loop unrolling, and SIMD vectorization.

1. Memory Alignment : One of the first optimizations implemented was ensuring 32-byte memory alignment for critical data structures. Using posix_memalign, key arrays such as y, Aval, Acol, Arow, and x were aligned to enable efficient SIMD operations and improve cache locality.

```
posix_memalign((void**)&y, 32, sizeof(double) * tInput->stNumRows);
// 32-byte aligned
```

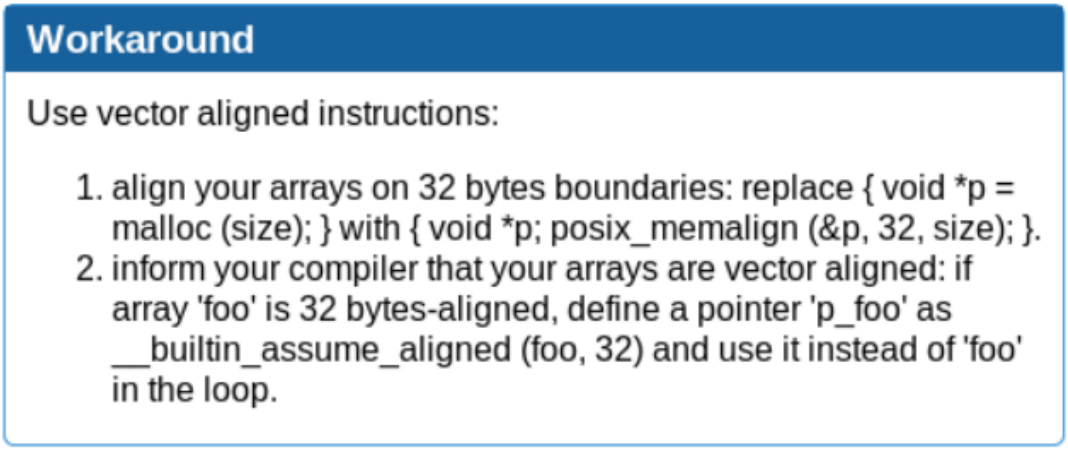


FIGURE 68 – Maqao Suggesting Memory Alignment

Aligning these arrays ensures faster memory access, reducing cache misses and memory access latency.

2. Loop Unrolling : To reduce loop overhead and maximize CPU register utilization,

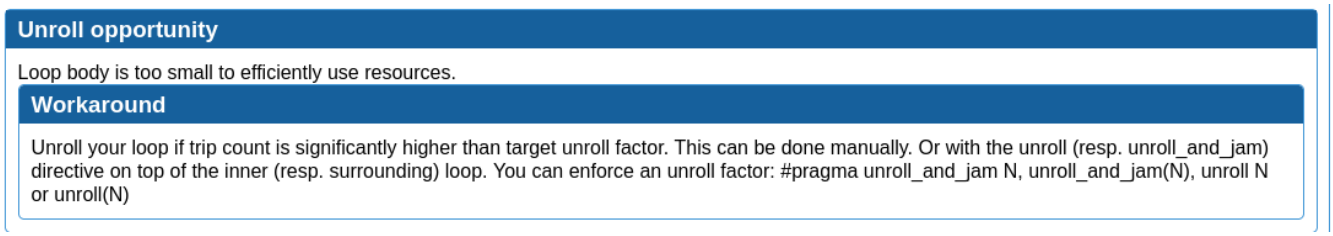


FIGURE 69 – Maqao suggesting loop unrolling

loop unrolling by a factor of 4 was applied to the innermost loop of the matrix-vector multiplication. This approach decreases loop control operations, allowing the processor to execute multiple computations in parallel, thereby improving computational throughput.

```
    // Unrolling loop by 4
for (nz = rowbeg; nz + 3 < rowend; nz += 4)
{
    sum += Aval[nz] * x[Acol[nz]];
    sum += Aval[nz + 1] * x[Acol[nz + 1]];
    sum += Aval[nz + 2] * x[Acol[nz + 2]];
    sum += Aval[nz + 3] * x[Acol[nz + 3]];
}
```

By unrolling the loop, fewer iterations are required, leading to lower branch overhead and better instruction-level parallelism

3. SIMD Vectorization : SIMD vectorization was introduced using AVX2 intrinsics, allowing the processor to operate on multiple data elements simultaneously. 256-bit vector registers (_m256d) were used for processing four double values in parallel, significantly improving floating-point computation.

```
__m256d sum = _mm256_setzero_pd();   // Set sum to zero
__m256d Aval_vals = _mm256_set_pd(Aval[nz+3], Aval[nz+2],
```

```
                    Aval[nz+1], Aval[nz]);
__m256i Acol_vals = _mm256_set_epi32(Acol[nz+3], Acol[nz+2],
                    Acol[nz+1], Acol[nz], 0, 0, 0, 0);
__m256d x_vals = _mm256_set_pd(x[Acol[nz+3]], x[Acol[nz+2]],
                    x[Acol[nz+1]], x[Acol[nz]]);
sum = _mm256_fmadd_pd(Aval_vals, x_vals, sum);
```

The use of FMA (Fused Multiply-Add) operations (VFMADD132SD) reduces instruction count by combining multiplication and addition into a single operation, thus minimizing latency and maximizing throughput.

4. Performance Improvement : After incorporating these optimizations, the execution

```
Input filename:             input-matrix/mat_dim_493039.txt
Matrix value array size:    57973976

Time measurements
Total experiment time:      38.3882
Minimum kernel time:        0.00459003
Maximum kernel time:        0.00811505
Arithm. Mean kernel time:   0.00479845

Performance results
Total GFlops/s:             3.0204
Minimum GFlops/s:           1.786
Maximum GFlops/s:           3.1576
Arithm. Mean GFlops/s:      3.02046
Arithm. Median GFlops/s:    3.1082
```

FIGURE 70 – Applicating Maqao suggestions : loop_unrolling and using AVX intrensics

time for the spmx_csr function was reduced from 46 seconds to 38 seconds, marking a significant performance improvement. This reduction was achieved through better data alignment, efficient use of SIMD instructions, and reduced loop overhead from unrolling. However, the code was not fully vectorized, and further improvements could be made by fully utilizing SIMD for all operations.

Assembly Code Optimization Analysis : After applying MAQAO's suggestions, the

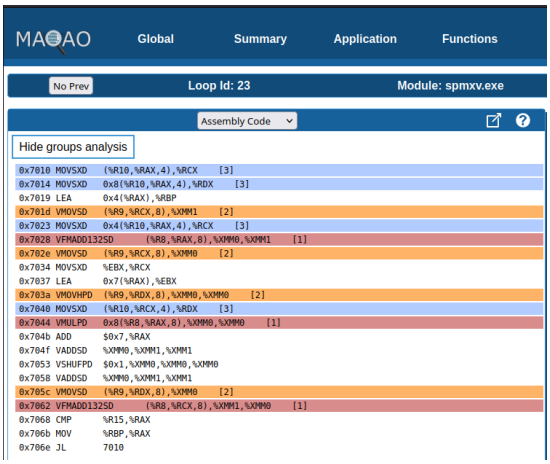


FIGURE 71 – Assembly generated with loop unrolling and using AVX

assembly code was analyzed to assess the effectiveness of optimizations :

— Memory operations **(MOVSD, VMULPD, VADDPS)** now process multiple elements simultaneously, improving data throughput.
— FMA operations **(VFMADD132SD)** replace separate multiplication and addition, reducing instruction count and enhancing performance.
— Loop unrolling ensures efficient iteration handling, preventing performance losses from excessive loop control operations.

These optimizations collectively resulted in a significant performance boost, making better use of the available hardware resources. While further vectorization (e.g., AVX-512) could yield additional gains, the current optimizations have already provided a substantial reduction in computation time.

10 Conclusion

In conclusion, this study provides a comprehensive analysis of sparse matrix-vector multiplication (SpMV) performance across different compilers—GCC, Intel OneAPI, and AMD AOCC—on an AMD architecture. By testing various chunk sizes and optimization flags, we identified key patterns in execution time, thread utilization, and memory efficiency. The results demonstrated that, for AMD systems, AOCC provided a significant advantage in performance, leveraging AMD-specific optimizations, while GCC and Intel compilers showed varying results depending on optimization flags. Despite the general expectation that larger chunk sizes would yield better performance, we observed diminishing returns after a certain point, with CHUNK_SIZE = 1200 emerging as the optimal configuration for most compilers. Furthermore, the assembly analysis revealed that full vectorization is crucial for maximizing performance, as scalar approaches underutilize the potential of the system's execution units. Overall, the study highlights the importance of tailored compiler optimizations, thread management, and memory access patterns for achieving peak performance in computational tasks like SpMV.