Additional Measurements

Kurt Böhm[1]*, Olaf Ippisch[1]

[1]Institute of Mathematics, Clausthal University of Technology,
Erzstraße 1, 38678 Clausthal-Zellerfeld, Germany

*Corresponding author. E-mail: `kurt.boehm@tu-clausthal.de`

Article: LINEAL: An Efficient Linear Algebra Library
Journal: GEM – International Journal on Geomathematics

28th March 2025

1 Sand Samples

1.1 BOOMERAMG: Variant Comparison

The results in Fig. 1 and Table 1 show that using the default parameters with one level of aggressive aggregation is the best of the configurations tested.

1.2 DUNE ISTL: Variant Comparison

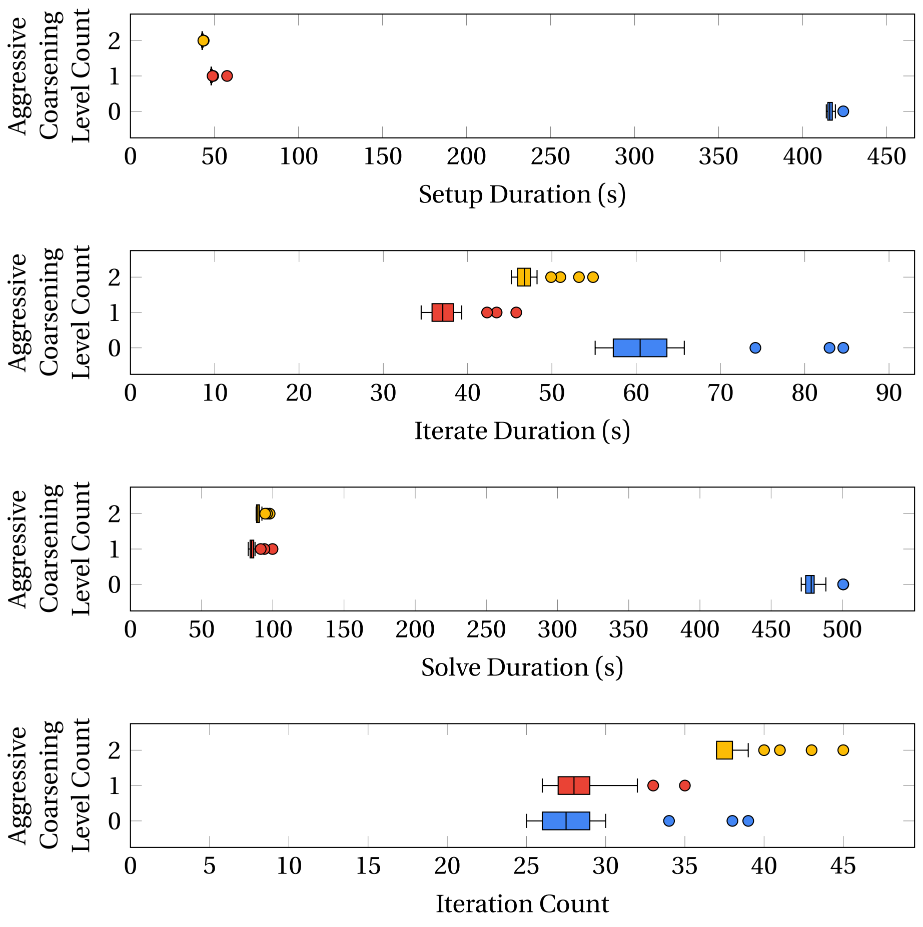
The results in Figs. 2 and 3 and Tables 2 and 3 show that the *iteration-optimized* and *speed-optimized* configuration used in the article are, indeed, the best configurations regarding the specific quantity.

2 SPE10 Model 2

2.1 Feature Comparisons

Here, we present the feature comparison data using SPE10 Model 2, which the notes at the start of the section on the Experimental Results for SPE10 Model 2 in the article apply to as well. Most of the optimization that LINEAL implements have a smaller impact than before due to the smaller problem size and the resulting shorter runtime and reduced memory consumption. In the case of mixed precision, as shown in Fig. 4, both the solve duration and the iteration count are virtually unaffected by switching from `f64`/`f64` to `f64`/`f32` while the memory consumption drops by 20%/12% (*AV-Stencil*/*CSR*) on average. On the other hand, `f32`/`f32` continues not to converge to a relative residual norm of 10^{-10}.

The differences are even less pronounced when it comes to the precision used for the auxiliary vectors of the CG method, as can be seen in Fig. 5: Both the solve duration and the iteration count are virtually the same for all precision combinations



	No aggressive levels	One aggressive level	Two aggressive levels
Min.	437.8	436.3	436.3
Max.	441.3	436.8	436.8

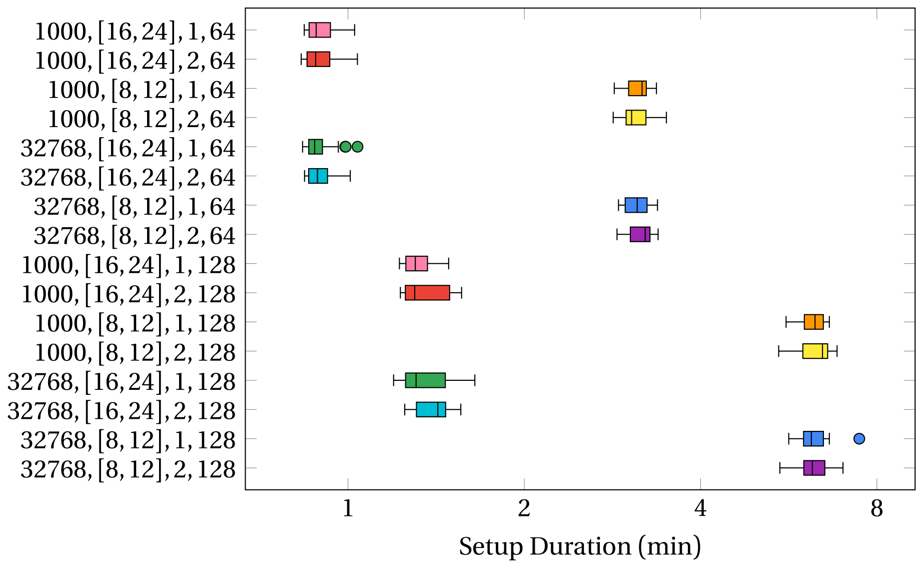
Memory Consumption (GB)

Figure 1 and Table 1: Results gathered using BoomerAMG with various numbers of levels of aggressive coarsening using the Sand Samples.

while the memory savings when using `CG<f32, f32>` are 4%/2% (*AV-Stencil*/*CSR*) compared to `CG<f64, f32>` and 11%/6% compared to `CG<f64, f64>` on average.

Figure 6 shows that different index size combinations make a comparatively small difference as well: Both the solve duration and the iteration count are virtually the same for all precision combinations (apart from 8 B/8 B with the *CSR* variant, which is slightly slower in some cases) while the memory savings for the 4 B/5 B combination are 1%/2% (*AV-Stencil*/*CSR*) compared to 4 B/8 B and 22%/23% compared to 8 B/8 B on average.

Different aggregate size ranges do not influence any aspect of performance significantly for this problem, as shown in Fig. 7.



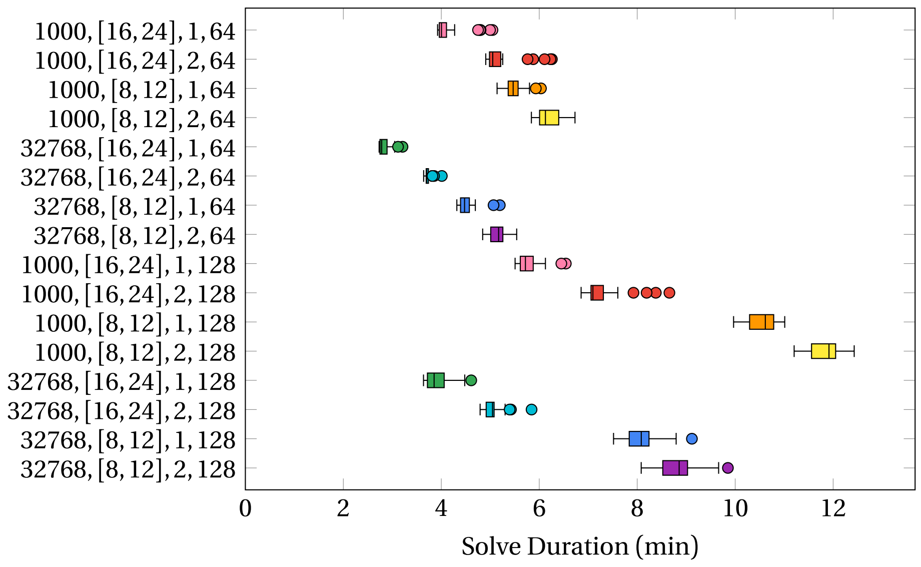
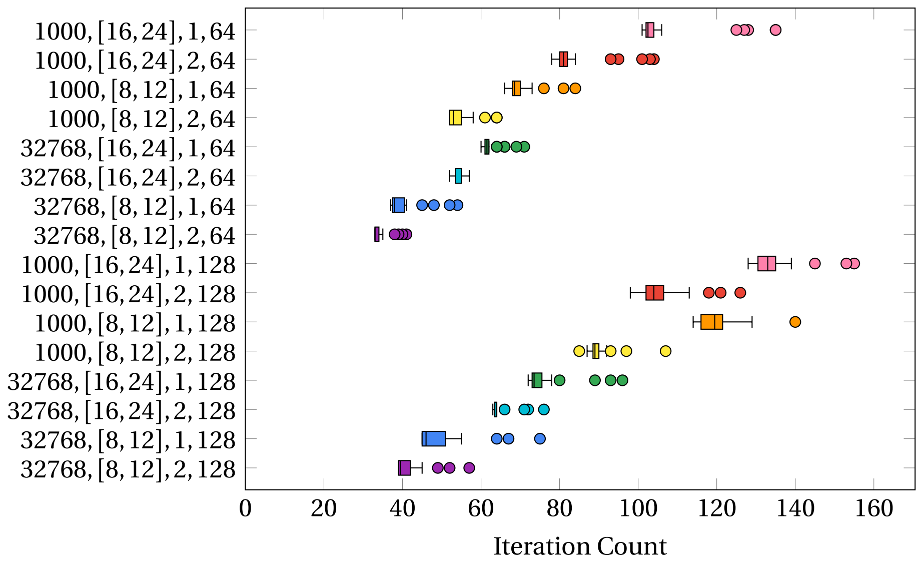


Figure 2 and Table 2: Setup and solve durations gathered using DUNE ISTL with the Sand Samples using configurations characterized by three parameters: the coarsening target, the aggregate size range, and the pre-/post-smoother iteration count. Additionally, both 64 and 128 processes were tested (represented by the last component of the tick labels).

Figure 8 show that explicit SIMD operations and tiling have a very small impact on performance, too, where using 2-component SIMD vectors is slightly slower than no explicit SIMD operations and the other SIMD variants are slightly faster for the *AV-Stencil* variant.



	64 Processes	128 Processes
Min.	225.8	240.7
Max.	226.1	241.7

Memory Consumption (GB)

Figure 3 and Table 3: Iteration counts and memory consumptions gathered using DUNE ISTL with the Sand Samples using configurations characterized by three parameters: the coarsening target, the aggregate size range, and the pre-/post-smoother iteration count. Additionally, both 64 and 128 processes were tested (represented by the last component of the tick labels).

Conversely, Huge Pages have a relatively significant impact due to the miniscule memory consumption: While the solve duration is barely affected, the memory consumption grows by 4%/2% (*AV-Stencil*/*CSR*) on average when enabling Huge Pages. This is because larger memory pages generally result in more unused memory around the memory areas that are actually used, which is more noticeable with a total memory usage of less than a quarter of a gigabyte than with tens or hundreds of gigabytes.

2.2 BoomerAMG: Variant Comparison

The results in Fig. 10 illustrate that using the default parameters with one level of aggressive aggregation yields the best overall results for BoomerAMG among the configurations tested, although all configurations result in very similar solve durations.

2.3 Ginkgo: Variant Comparison

The results in Fig. 11 demonstrate that Ginkgo's default configuration from the article is the fastest of Ginkgo's configurations tested. These results also shows

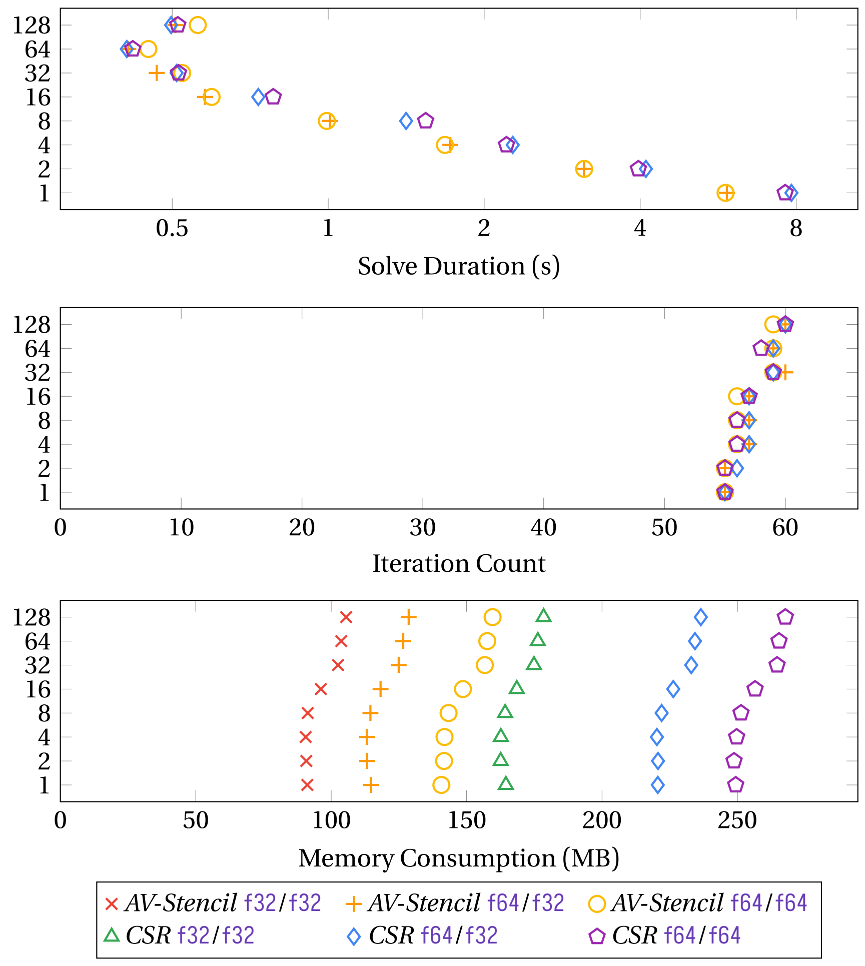


Figure 4: Results gathered with various mixed-precision settings using SPE10 Model 2. Neither runtimes nor iteration counts are shown for `f32`/`f32` because this variant did not converge within 256 iterations.

that Ginkgo's implementation of the Gauß-Seidel method is significantly slower than that of the Jacobi method when executed sequentially, scales very poorly with increasing thread count, and requires 5% more memory on average.

2.4 DUNE ISTL: Variant Comparison

Figure 12 shows that none of the configurations of DUNE tested is clearly the best, as well as problematic scaling behaviour. However, it also shows that the *iteration-optimized* and *speed-optimized* configuration from the article perform the best regarding the respective quantity among the configurations tested.

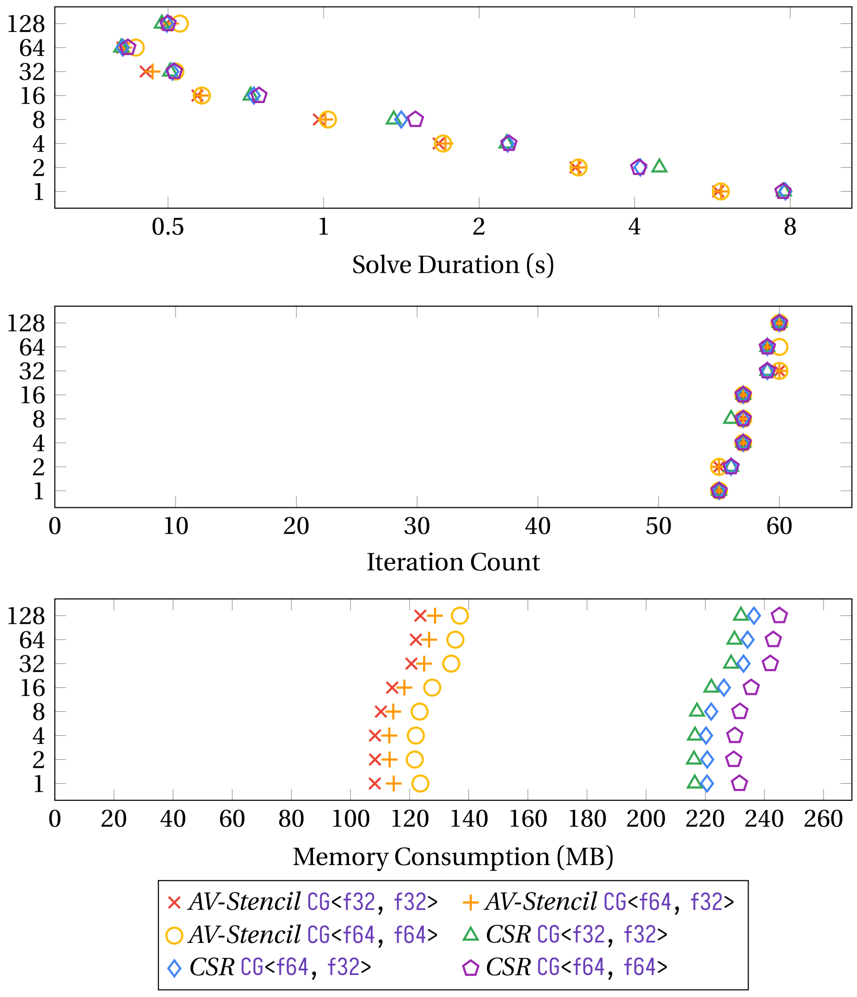


Figure 5: Results gathered with various precision combinations for the auxiliary vectors of the CG method using SPE10 Model 2, where the first type denotes the type used for the residual vector and the second denotes the type used for the remaining auxiliary vectors.

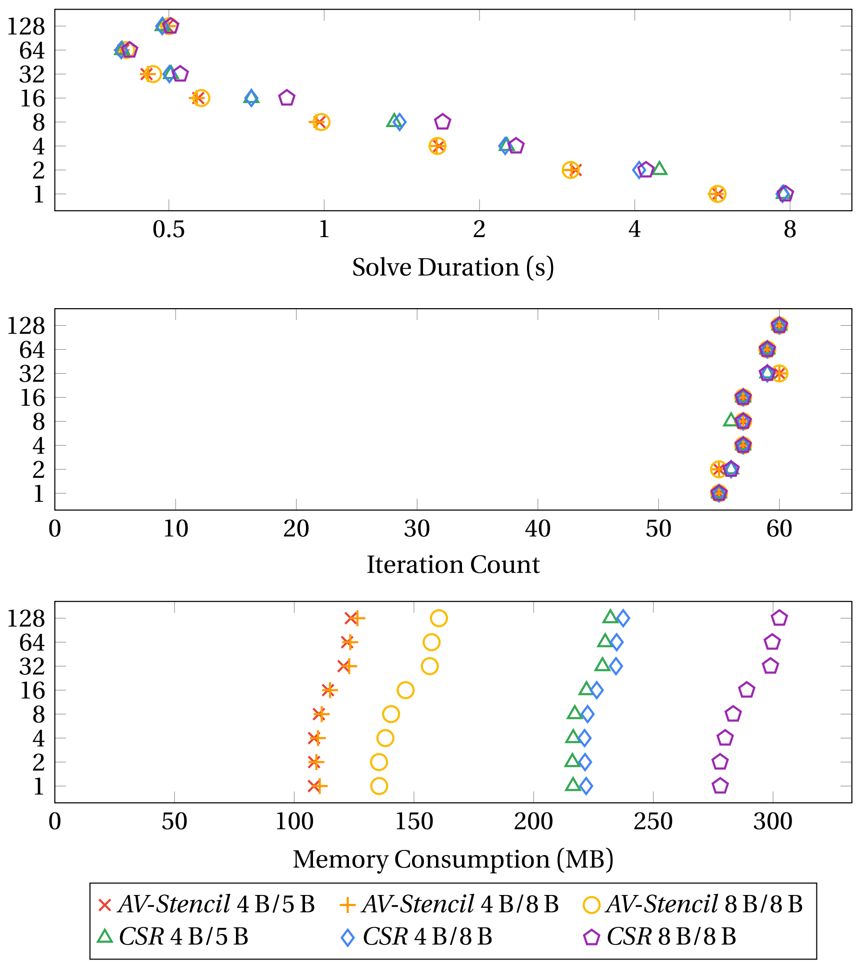


Figure 6: Results gathered with various index size combinations using SPE10 Model 2, where the first size applies to row/column indices and the second to non-zero indices.

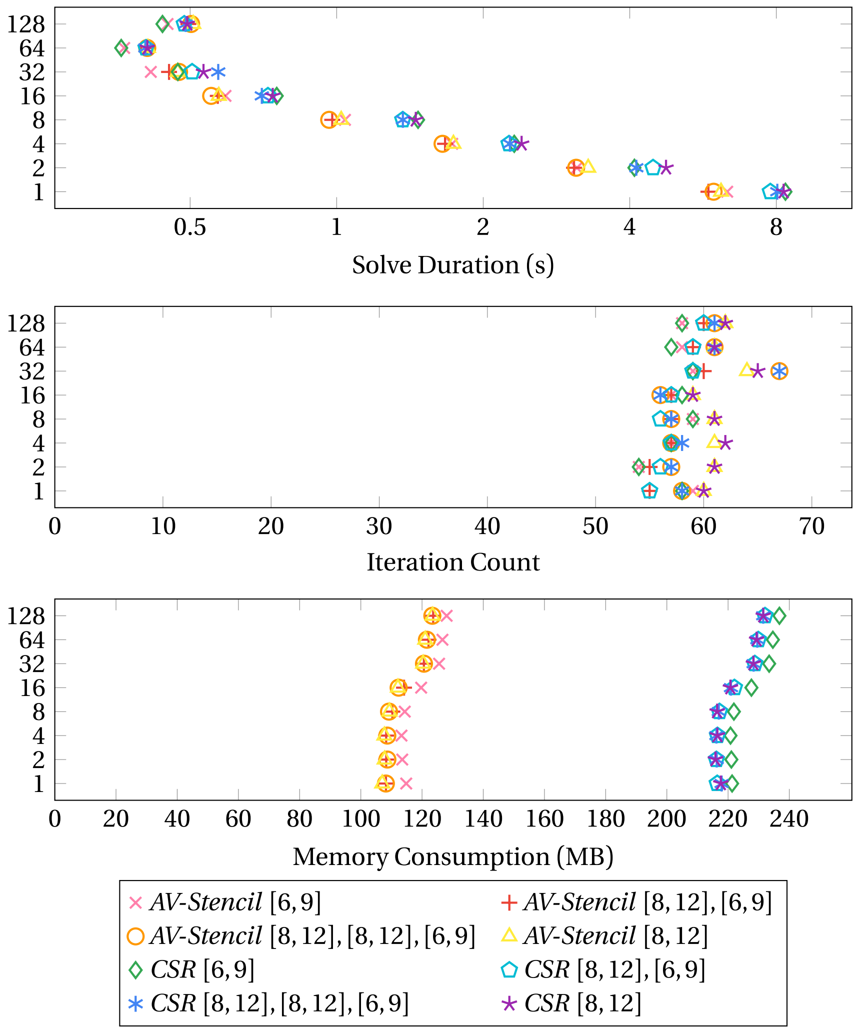


Figure 7: Results gathered with various aggregate size ranges using SPE10 Model 2. Here, the first range specified applies to the finest level, the second to the second-finest level, and so on, while the last range specified is also applied to all coarser levels.

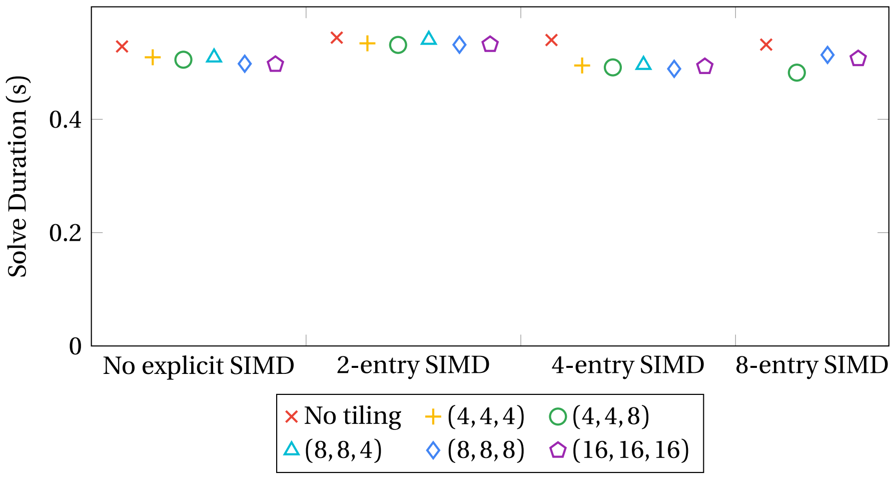


Figure 8: Results gathered with various tile and SIMD vector sizes using the *AV-Stencil* variant with SPE10 Model 2.

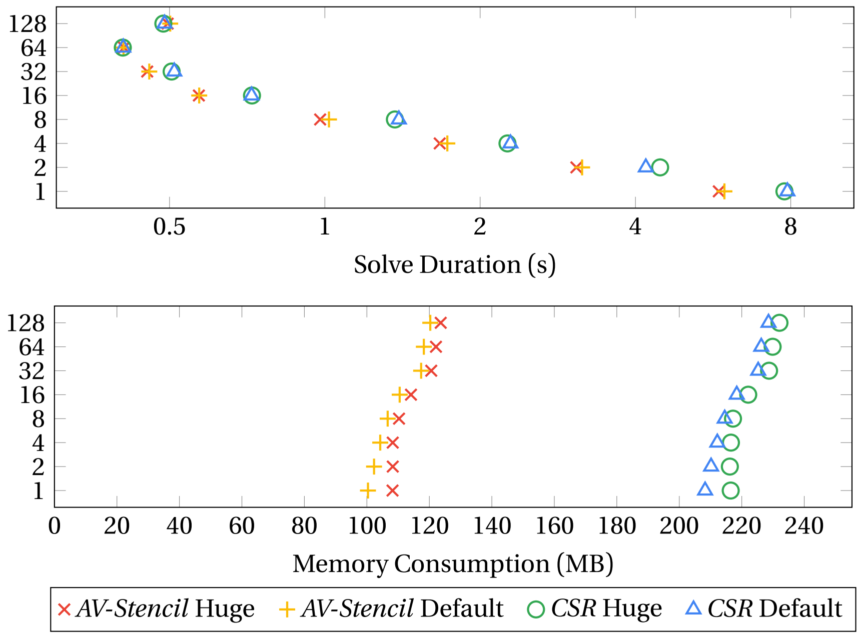


Figure 9: Results gathered with and without huge pages using SPE10 Model 2.

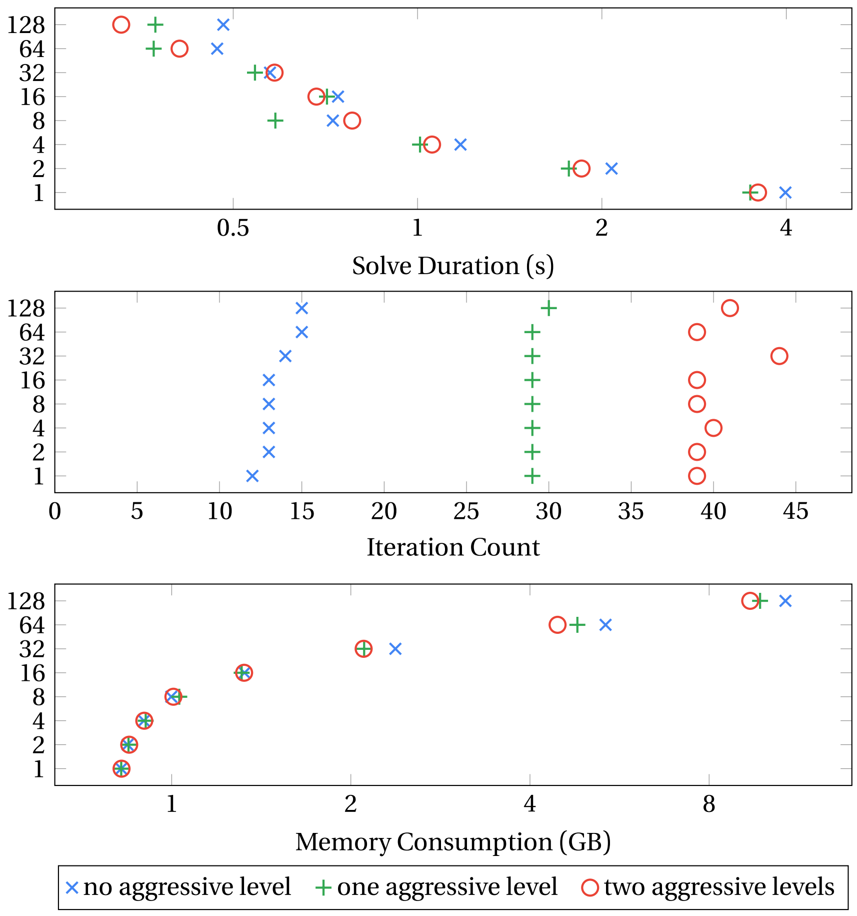


Figure 10: Results gathered using BoomerAMG with various numbers of levels of aggressive coarsening using SPE10 Model 2.

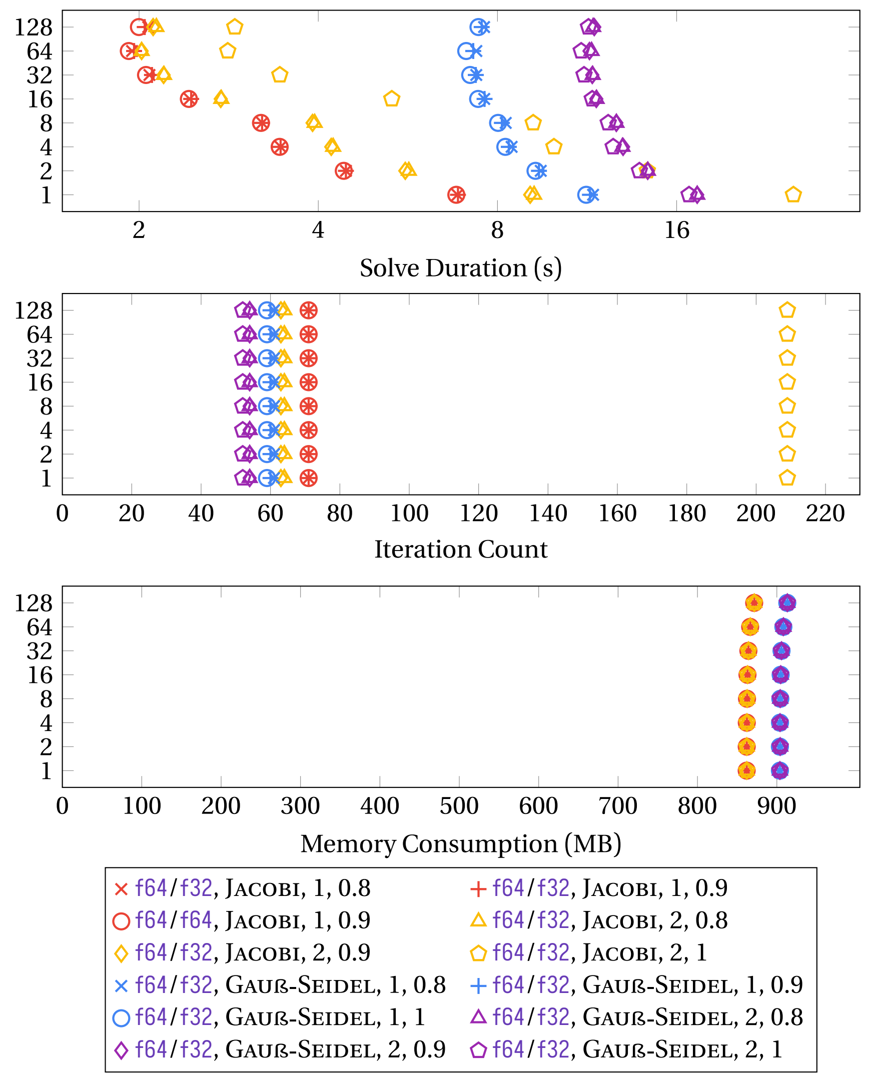


Figure 11: Results gathered using GINKGO with various fine-/coarse-level floating point types, smoothers, smoother iteration counts, and smoother relaxation factors using SPE10 Model 2. The configurations "f32/f32, JACOBI, 1, 0.9" and "f64/f32, JACOBI, 1, 1" are not included in the comparison because they did not converge to a relative residual norm of 10^{-10} within 256 iterations.

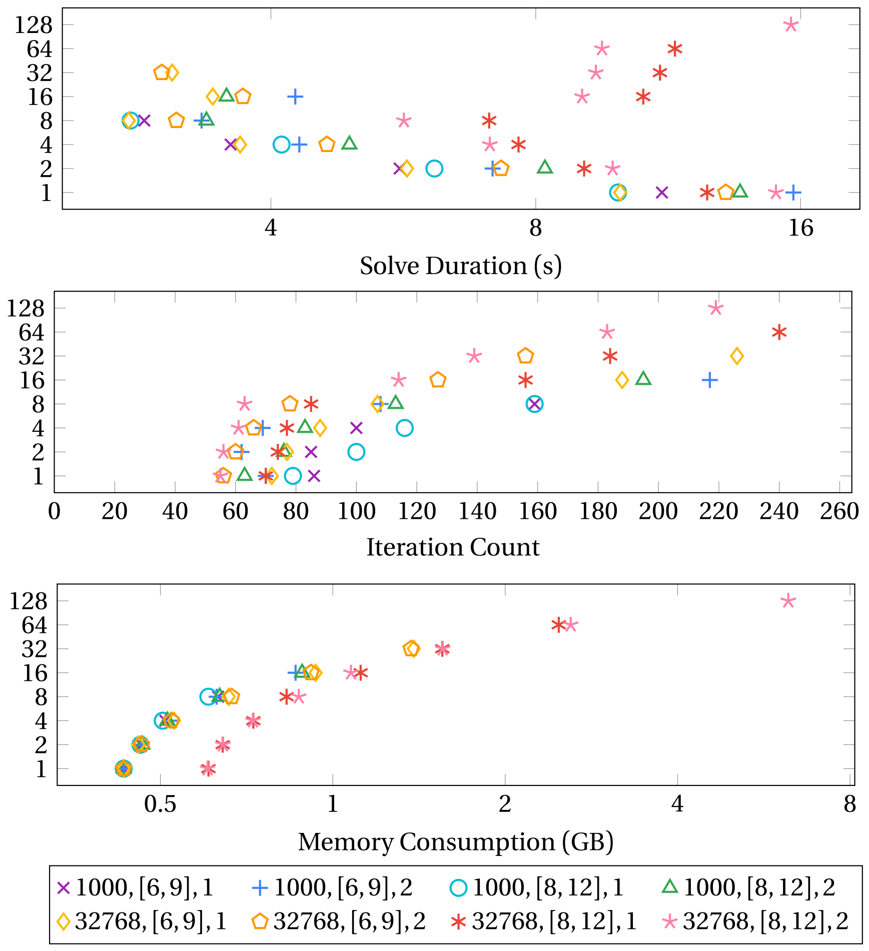


Figure 12: Results gathered using DUNE ISTL with different configurations using SPE10 Model 2. These configurations are characterized by three parameters: the coarsening target, the aggregate size range, and the pre-/post-smoother iteration count.