11th World Congress on Computational Mechanics (WCCM XI)
5th European Conference on Computational Mechanics (ECCM V)
6th European Conference on Computational Fluid Dynamics (ECFD VI)
July 20–25, 2014, Barcelona, Spain

OPTIMIZING THE MEMORY ACCESS PERFORMANCE OF FASTEST'S SIPSOL ROUTINE

Michael Burger[1] **and Christian Bischof**[1]

[1] TU Darmstadt, Mornewegstrasse 30, 64293 Darmstadt, `www.sc.tu-darmstadt.de`,
{michael.burger, christian.bischof}@sc.tu-darmstadt.de

Key words: *Data Structures, Performance Optimization, Parallelization, Cache Optimization*

In this article the runtime behavior of the simulation software Fastest [FNB13] is investigated and its single core performance is optimized through a new cache-friendly data layout. The main performance bottleneck of Fastest is the Sipsol subroutine, which implements a "‘Strongly Implicit Solver” and accounts for over 50% of the total instructions. Fastest arranges the data in so called 3D diagonals, on the right side of figure 1 three example 3D diagonals are visualized: Imagine a pyramid - 3D diagonals are equivalent to a face, and subsequent 3D diagonals lie on top of each other. The blue diagonal covers the green one and the red one covers the blue one. Sipsol updates the grid one 3D diagonal at a time and all elements within a diagonal are independent of the others.

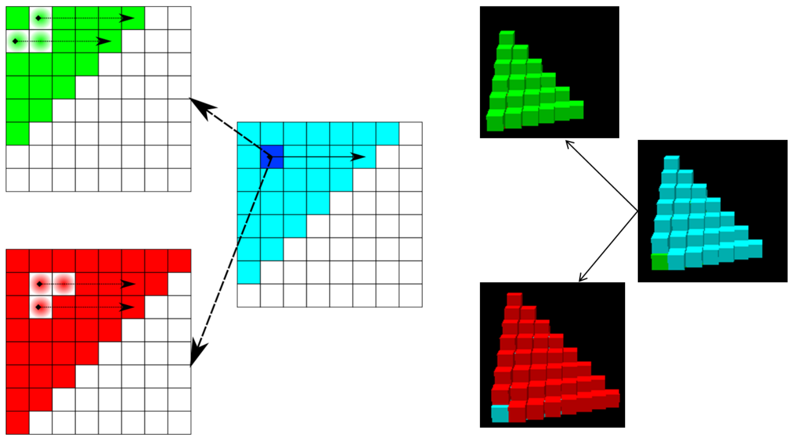
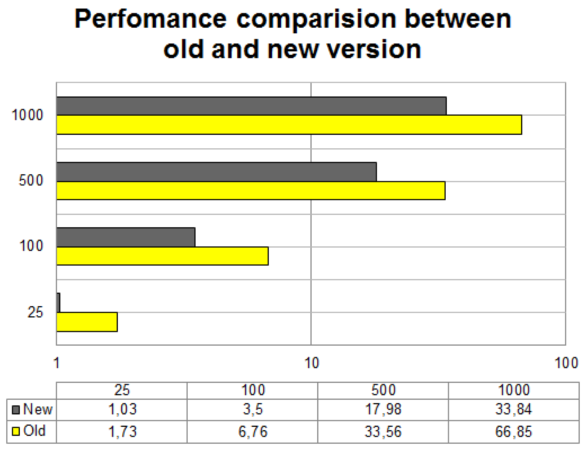


Figure 1: Projecting 3D diagonals to 2D arrays

Analyses we performed with Intel VTune Amplifier and Vampir show that memory access cost dominates the runtime. As a consequence of transforming the element numbering in the diagonals to spatial coordinates within Sipsol. memory is not accessed in a consecutive manner. To address this issue, a new data structure is developed which on the one hand increases the speed, but on the other hand preserves the ability to parallelize the calculations on the data grid executed within Sipsol. To that end, the 3D diagonals are projected into quadratic, fixed size 2D grids. This process is indicated on the left side of figure 1. Assuming that the dark blue item should be calculated, the pale elements in the red and green neighbor-diagonals are required. The arrows indicate the elements that will be needed in subsequent iterations. As figure 1 shows, all data for the update of the element right to the dark blue element either has already been copied to cache or follows directly after the last ones. The ratio between calculation and memory reads/writes is improved significantly, and the performance doubles in many cases. Detailed results are show in figure 2 where the yellow bar shows the performance of the original Sipsol, the gray bar the performance of the new code. For an iteration count bigger than 50 the overhead for creating the new data layout is negligible.



	25	100	500	1000
■ New	1,03	3,5	17,98	33,84
□ Old	1,73	6,76	33,56	66,85

Figure 2: Performance results.

	System configuration:
CPU:	2 x Intel Xeon E5-2670
RAM:	128 GB
OS:	Ubuntu 11.04
Compiler:	Intel ifort v. 13.2
Flags:	- OX (full optimization)
	-avx (vectorization)
Legend for figure	
x-axis	Total runtime in seconds logarithmic scale
y-axis	Count of inner iterations

Table 1: Test configuration

Finally, approaches to efficiently implement a shared-memory parallelization are proposed. It is possible to compute all lines within a 2D grid in parallel, since all elements of a 3D diagonal are independent from the other. Additional points which should be taken into account to take most advantage of the new data layout are also presented, e.g. data structure initialization and adjustment of the inner iteration count.

REFERENCES

[FNB13] FNB: Project Site for Fastest / Technische Universtiaet Darmstadt. 2013 (1). – Forschungsbericht. – `http://www.fnb.tu-darmstadt.de/forschung\_fnb/software\_fnb/software\_fnb.de.jsp`