Execution Environment

Our measurements were run on the SuperMUC supercomputer of the Leibniz Supercomputing Center. The technical parameters of the system are gathered into the table below. The sources we used were Prof. Gerndt's slides about SuperMUC from Parallel Programming, Intel's website about CPU characteristics, LRZ's website, and by logging in to SuperMUC and “looking around”.

Parameter	Value
Number of Processors per Node	**2**
Number of Cores per Processor	**8**
Hyperthreading	**Yes**
Threads per Core	**2**
Number of Threads per Node	**32**
Peak Frequency per Core	**2.70 GHz** **Max Turbo: 3.50 GHz**
Peak Performance (Mflops)per Core	**21600 MFlops @ 2.70 GHz**
Peak Performance (Mflops) per Node	**345600 MFlops @ 2.70 GHz**
Frequency Scaling	**Yes**
L3 Cache per Processor (shared)	**20 MB**
L2 Cache per Core	**256 KB**
L1 Cache per Core	**32 KB**
Compiler	**ICC 13.1.3**
MPI Library	**IBM MPI**

Profiling metrics

We used the PAPI library to gather information about performance metrics; in our case L2 cache miss rate, L3 cache miss rate, floating-point operations executed and wall-clock time. The `papi_avail` command-line utility shows information about the available performance counters.

The cache miss rate was not available directly, but we calculated using available counters as follows:

$$\text{cache miss rate} = \frac{\#\text{ cache misses (PAPI\_Lx\_TCM)}}{\#\text{ cache accesses (PAPI\_Lx\_TCA)}}$$

It turned out that we cannot measure the cache profiling counters together with floating-point operations, therefore we had to make every measurement twice: once for cache profiling and once for counting the number of floating-point operations.

We also had to calculate utilization. Since we measured the execution time and the number of floating-point operations, and we already had had the peak performance, we got the utilization as:

$$\text{utilization (\%)} = \frac{\text{MFLOPS (measured)}}{\text{execution time (s)}} / \text{MFLOPS (peak core performance)}$$

Discussion of utilization and optimization flags

The utilization the CPUs floating-point units is quite low, peaking slightly below 7% at optimization level `-O1` for about all datasets. This suggests that the GCCG solver is rather memory-intensive than computation-intensive. This is also supported by the high L2 cache miss rates, varying from 1/3 to 1/2 for most cases. The relatively low L3 cache miss rates are, on the other hand, not surprising, since the memory requirement of the computations exceeds L3 cache size at most by a factor of a few.

Another notable aspect is that the utilization usually peaks at optimization level `-O1`. Considering elements of utilization formula above, the execution time of the computation phase radically decreases from `-g` to `-O1`, while further optimization has only moderate benefit on time. Although the number of floating-point operations continue to drop significantly from `-O1` to `-O2`, they don't have much effect on the execution time, since the computation is memory bound. However, reducing MFLOPS does have its advantages, since less work for the CPU may help reducing power-consumption.

Discussion of I/O performance

Unlike on the computation phase, the use of different optimization flags had relatively little effect on the I/O performance. At the highest optimization levels, the (textual) input/output phases combined take about 40-70% of the total execution time. Therefore, we had to look for ways of manually optimizing input/output. To do so, we made our GCCG solver capable of using binary input files, which are more compact, easier to read, and their reading requires almost no data processing.

Binary input files turned out to have significant advantages: they are roughly three times smaller and reading them takes roughly 10 times less time. (The actual numbers vary from dataset to dataset, see the spreadsheet document for details.)

We expect similar benefits from using a binary output format instead of the textual VTK file format.