Plasma scaling study

Simon Haendeler, Severin Staudinger

January 30, 2017

Abstract

We are examining the scaling of the PLASMA library in case of the Cholesky Decomposition. We compare the runtime when using a different amount of cores and compare this to the runtime of LAPACK.

1 Introduction

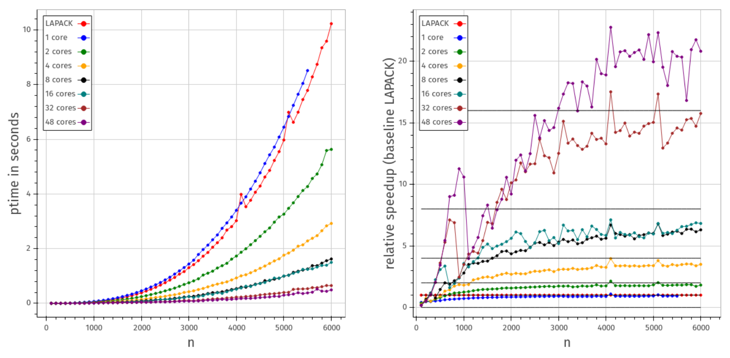
In this paper we demonstrate how a driver function of PLASMA [1] scales. Therefor we decided to show this by the cholesky factorization, which is an important decomposition of symmetric positive definite matrices in the field of scientific computing. To achieve this we implemented PLASMA's *DPOTRF* routine and measured the processing time of its execution on two, four, eight, sixteen, 32 and 48 cores. For time measuring we used the library PAPI [2].
All measurements were done on a system with an x86_64 architecture, 48 CPUs, one thread per core, AMD Opteron™ Processor 6174 with a theoretical peak performance of 2200.00 MHz. Furthermore there are 8 NUMA nodes, which combine six cores per each node. In the end we compared PLASMA's *DPOTRF* runtimes to the sequential cholesky decomposition implemented in LAPACK [3] and calculated absolute and relative speedups.

2 Implementation

In order to build PLASMA we installed OpenBLAS [4] and LAPACK [3] from the latest stable releases. Since on our system is just one thread per core we built OpenBLAS single-threaded with *MAKE USE_THREAD=0*. To guarantee portability we decided to make a bash-script, which install every necessary library fully automatically and also auto-generate a Makefile, which compiles our *main.c* programm. This is very useful, because anybody who is interested in this PLASMA scaling study can simply run this script with *bash plasma_lapack_blas.sh*. After small adjustments of the generated template_Makefile the user can compile the programm with *make*.
To verify a correct execution of the test programm we used valgrind.

3 Evaluation

Because the measurement was done on a shared system we could not guarantee that our program was the only one running. Measurement errors through context switches and alike are possible. Each datapoint represents 5 measurements that were averaged so that small measurement errors cancel out. Systematic error could still be present i.e. long running processes which affect all measurements are still possible. Although we tried to control for those manually via htop and similar tools, we cannot exclude those with 100% certainty.



(a) Process time (b) Relative Speedup

Figure 1: Runtime of PLASMA vs LAPACK

In figure 1a we see the runtime of the LAPACK version and of the PLASMA version with different amount of cores/threads used. As expected the runtime mostly increases like x^3. The single threaded PLASMA version is slightly slower than the sequential LAPACK version. The other PLASMA versions are faster the more cores there are, at least for n large enough. The versions for 8 and 16 threads are very close together. 32 and 48 threads are also close, although we expect that this gap should be relativly narrow. There are bumps in the graph of the LAPACK version for $n = 4100$ and $n = 5100$.

In figure 1b we see the relative speedup for each version. For comparison there are black lines at 2, 4, 8, 16, 32, 48 which represent the theoretical possible speedups. We can see that the parallel versions start slower than the sequential version. Then for relatively low n each version archieves a local maxima. For a higher thread number the n is higher when the maxima is reached.

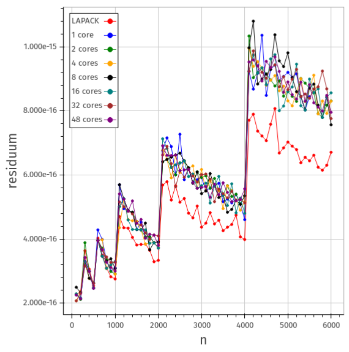
Only 2 and 4 threads nearly reaches its theoretical performance, the higher the amount of threads, the worse the performance relatively to the theoretical performance.

The residuum is shown in figure 2a. All PLASMA versions have the same residuum behavior, but LAPACK is always a bit better. There are certain thresholds after which the residuual gets worse (200, 500, 1000, 2000, 4000). After these sudden increases the residuum gets better with higher n.

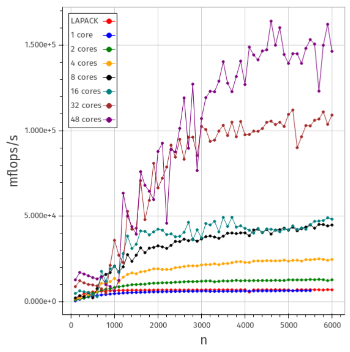
In figure 2b we see the Mflops per second. We see that both LAPACK and PLASMA with one core are very similar.

4 Conclusions

When looking at the runtime we can see that PLASMA scales with additional cores. For a lower amount of cores the scaling is very good and nearly multiplies the performance with the amount of cores. As soon as we go higher than 8 the



(a) Residuum



(b) MFlops per second

Figure 2: Residuum and Performance comparisons

scaling gets worse. The test server has 4 CPUs with each 12 cores, so when the number of cores is higher than 12 additional overhead should come into play. 32 cores reaches the performance we would have wished for 16 cores. This trend also goes on with 48 cores, the maximum on the system.

LAPACK has consistently a better residuum than PLASMA. This is very surprising, because LAPACK is used internally of PLASMA. Maybe the merging of the subsolutions of the multiple LAPACK invocations affect the accuracy. In that case we would expect the residuum to grow with the number of cores, which we cannot see here. Maybe we could see this if the residuum was more stable, i.e. when averaging over a higher number of trails. One could also do additional statistic analysis of the result. But because the residuum is good enough in the PLASMA case we will not further discuss it here.

It is also surprising that the residuum gets better with higher n between the jumps. The jumppoints have a very systematic distribution, so it is likely that LAPACK changes some parameters when around these thresholds.

For mflops we roughly get 150 Gflops. This is $\tilde{7}0\%$ of the theoretical peak performance of 211 Gflops/s. The numbers seems very high when compared to other publications e.g. [2].

References

[1] Plasma. .

[2] Jakub Kurzak, Piotr Luszczek, Mathieu Faverge, and Jack Dongarra. Lu factorization with partial pivoting for a multi-cpu, multi-gpu shared memory system. Technical Report 266, LAPACK Working Note, April 2012.