Advanced Parallel Computing: Exercise #6

Due on Tuesday, June 2, 2015

Svend Dorkenwald, Günther Schindler

Reading

Software Transactional Memory: Why Is It Only a Research Toy?

In this paper the authors investigate the fact that transactional memory is only used theoretical in research and not in an industrail sense. The focus of this paper is software-only TM, which offers flexibility and no hardware cost. They explore the performance of a highly optimized STM and observe that the overall performance of TM is significantly worse at low levels of parallelism, which is likely to limit the adoption of this programming paradigm.
The authors advise the elimination of dynamically unnecessary read and write barriers, which is probably the most powerful lever toward further reduction of STM overheads. They are deeply skeptical of any proposed solutions that require extra work by the programmer.
As far as we can see it is very optimistical to simply analyse only a software implementation of a paradigm like TM. Sure, it is difficult to develop and simulate hardware or hybrid TM but STM leads to too much overhead in excess. Altough, this paper shows some open issues about STM or TM in general which have to be solved anyhow. We weakly accept this paper.

Why STM can be more than a research toy

This paper is a contrary of Cascavals paper 'Software Transactional Memory: Why Is It Only a Research Toy?' in which STM is declared as a research toy because STM perfromed worse than sequential code. The authors of this paper comparing now STM performance to sequential code using a larger and more diverse set of benchmarks than Cascavals and real hardware supporting higher levels of concurrency. Also, they used a state-of-the-art STM implementation more than those used by Cascaval.
The authors show that parallel applications with high contention are not the primary target for STM. They expose that STM with support for compiler instrumentation and explicit, nontransparent privatization outperforms sequential code in almost every workload they used.
Their results contradict Cascavals and suggest STM is usable for a range of applications. We can only trust in the correctness of their results but are still very skeptical of STM. However, if it's possible to improve performance with software we believe that further research should be performed. Hence, we strongly accept this paper.

Parallel Prefix Sum - Development

We implemented the parallel prefix sum according to the algorithm suggested by Blelloch. To ensure that our code generates correct results we compared the results of smaller input arrays with that of a one core straight forward implementation.
To reach reasonable timings we used arrays of length 16777216 ($= 2^24$) with 20-fold repetition.

Parallel Prefix Sum - Analysis

Abbildung 1 shows our results. The runtimes for different numactl settings where measured for Thread counts (TC).
The fact that the results for the default and the membind configurations are mostly the same leads us to the conclusion that per default the memory is bind to one socket. Hence, one should enable interleave=all or similar options when running multi-threaded implementations as shown by the performance of this configuration.
All settings show a peak at about TC=12 which is the number of cores per die. This may be expected for the

TC	$T_{default}$[s]	$T_{membind}$[s]	$T_{interleave}$[s]
1	0.499	0.455	0.414
2	0.419	0.0485	0.397
4	0.400	0.405	0.358
8	0.478	0.478	0.327
12	0.673	0.681	0.378
16	0.552	0.553	0.264
24	0.576	0.582	0.272
32	0.561	0.577	0.244
40	0.606	0.620	0.293
48	0.601	0.609	0.275

Abbildung 1: Performance comparison of different numactl settings

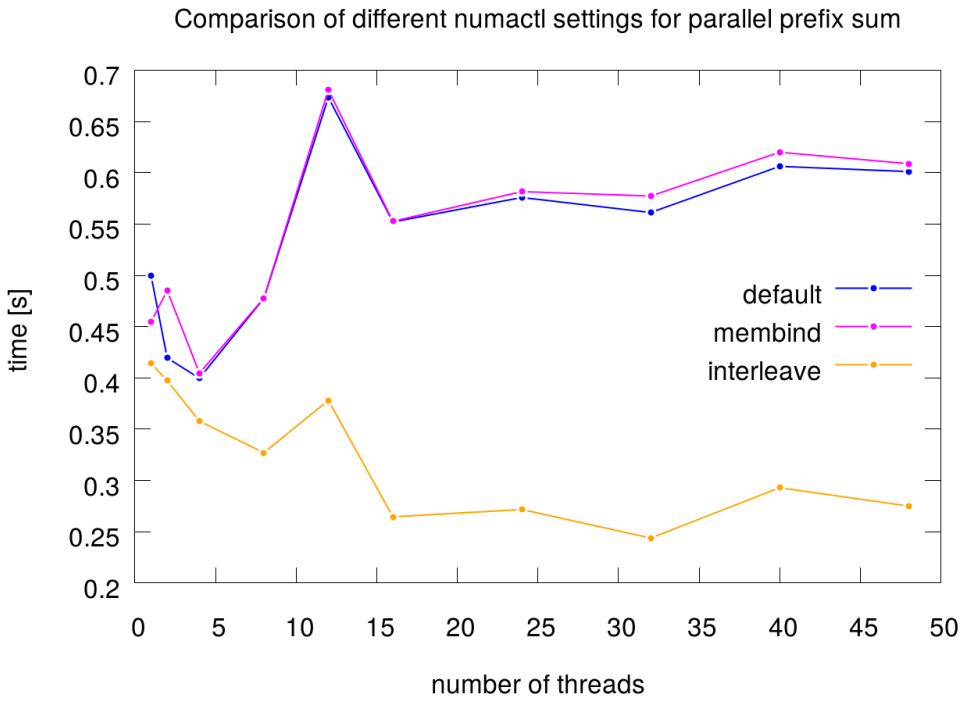

Abbildung 2: Runtimes for the parallel prefix sum.

default and membind configurations, but not for the interleaved one. Since it makes usage of the distribution of the array different cores on different dies seem to be used... This remains unclear to us.