6.895 Theory of Parallel Systems Project Proposal

A Space-Efficient Global Scheduler for Cilk

Jason Hickey and Tyeler Quentmeyer

We propose to implement a scheduler for Cilk based on the work presented in *Space-Efficient Scheduling of Nested Parallelism* by Narlikar and Blelloch [3]. We will begin by implementing the serial AsyncDF scheduler for Cilk and comparing its performance with the randomized work-stealing scheduler. We will then experiment with different methods for grouping threads for locality with the Cilk syntax. Next we will implement the parallel AsyncDF scheduler and compare its performance with both the serial AsyncDF scheduler and the randomized work-stealing Cilk scheduler. Finally, we will experiment with optimizations to the AysncDF implementation and with other techniques to improve resource utilization in Cilk.

The AsyncDF thread-scheduling algorithm is a deterministic provably good (space and time efficient) global parallel scheduling algorithm. Threads tend to allocate a large chunk of memory, perform some computation, and then free the memory. The principal behind AsyncDF is that you get better performance by exploiting parallelism inside the computation portion of the code rather than in code that allocates large amounts of memory. When a thread tries to allocate a large amount of memory, it should be put to sleep until other threads that do not allocate large blocks of memory are exhausted. In other words, the AsyncDF is a P wide depth first search of the computation DAG.

AsyncDF contrasts with the Cilk scheduler both in terms of time and space bounds and in terms of how strongly those bounds are guaranteed. The theoretical performance bounds for AsyncDF are better than Cilk's randomized work-stealing scheduler. AsyncDF is deterministic and its performance bounds are therefore guaranteed. The authors of the AsyncDF algorithm also claim that AsyncDF outperforms Cilk in practice. The "drawback" of AsyncDF is that the algorithm has a tunable parameter that allows the scheduler to make a tradeoff between time and space efficiency. AsyncDF also requires a non-trivial algorithm to "group" threads together to exploit locality.

We will begin our project by implementing the serial (simplest) version of the AsyncDF scheduler and integrating it with the Cilk compiler. We will also experiment with algorithms for grouping threads into blocks that should be run on the same processor to exploit locality. We will implement the tunable parameter to AsyncDF as a parameter to the Cilk compiler.

Next we will begin our performance analysis of AsyncDF in Cilk. Narlikar et al claim that serial AsyncDF is faster than parallel AsyncDF for machines with 32 or fewer processors so performance results at this stage are important. We will compare the performance of several parallel algorithms coded in Cilk (i.e., recursive matrix multiplication, Strassen multiplication, the fast multipole method for solving the n-body problem, etc.) using the randomized work-stealing scheduler and AsyncDF. We will also make performance comparisons over a range of inputs for AsyncDF's tunable parameter. Narlikar et al claim that AsyncDF is faster than Cilk's scheduler based on tests they performed utilizing hand optimized code and a custom parallel runtime based on pthreads. Our tests will provide a much more honest comparison in that they will compare AsyncDF to randomized work-stealing with all else being equal.

We will then implement the parallel AsyncDF scheduler in Cilk and repeat our performance comparisons. We will also compare the performance of the serial and parallel schedulers on different machine architectures.

Finally, we will experiment with improvements to AsyncDF, global schedulers in general, and other techniques to improve resource utilization in Cilk. We will look for incremental improvements in our AsyncDF implementation and for improvements in the AsyncDF algorithm. We will also look at other global scheduling techniques in Cilk. We will take a careful look at methods to introduce memory saving techniques into the original Cilk scheduler. Finally, we will look at extensions to the Cilk syntax to provide cues to the compiler to help it utilize resources more effectively (for example, in the context of AsyncDF, to help the compiler group threads for locality).

Our project proposal is safe in that there are enough almost guaranteed results to ensure that we do not need a sophisticated contingency plan. The AsyncDF algorithm is accessible and integrating it with Cilk is straight forward. Performance comparisons are straightforward and useful. However, our project still has plenty of avenues for novel research should they prove to be fruitful during the course of the semester.

References

[1] Robert D. Blumofe and Charles E. Leiserson. Space-Efficient Scheduling of Multithreaded Computations. *25th Annual ACM Symposium on the Theory of Computing (STOC '93),* May 16-18 1993, San Diego, California, pp. 362-371.

[2] Matteo Frigo, Charles E. Leiserson, and Keith H. Randall. The Implementation of the Cilk-5 Multithreaded Language. *1998 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, Montreal, Canada, June 1998.

[3] Girija J. Narlikar and Guy E. Blelloch. Space-Efficient Implementations of Nested Parallelism. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 21(1), January 1999.

[4] Supercomputing Technology Group, Massachusetts Institute of Technology. *Cilk 5.3.2 Reference Manual*, November 2001. Available on the World Wide Web at URL "`http://supertech.lcs.mit.edu/cilk`".