Roman Kazarin
Michael Nekrasov
CS290B – HW3 Paper Summary

Cilk: An Efficient Multithreaded Runtime System – Summary

Cilk is a runtime system for multithreaded parallel programming. It uses a work-stealing scheduler and gives the user an algorithmic model of application performance. A Cilk program consist of Cilk procedures, all of which consist of nonblocking threads. Threads can then spawn child threads to run child procedures. Parent threads can continue execution and not wait for child threads to return values. Instead, the parent thread must spawn a successor thread to receive the results of a child thread.

The execution time of a Cilk program depends on the number of processors, and the *work* and *critical path* parameters. Work is the time it took by a one-processors execution of the program (sum of the execution times of all threads). Critical path is the amount of time it takes for an infinite-processor execution. Cilk uses "work stealing" to achieve space, time, and communication bounds all within a constant factor of optimal.

Cilk uses a *closure* data structure. A closure points to a C function for a thread to run, a slot for each argument, and a join counter which keeps track of all arguments that are still needed before the thread can run the function. Therefore a closure can either be *waiting* or *ready* if it has received all of its arguments. A Cilk parent procedure must consist of 2 threads – the first spawn a child procedure, the second is a successor thread that waits for a return value from the child.

Cilk uses a work-stealer, which sees a thief processor select a victim processor from whom to steal work when it runs out of work to do. Work stealing is implemented using a request-reply communication protocol between the thief and victim processors.

The paper then goes in depth about some applications of Cilk: solving fibbonaci numbers, the N Queens problem, and a protein-folding program. It observes that the space per processor is usually small and does not grow with the number of processors. It also demonstrates that a nearly perfect linear speedup is obtainable, when the critical path is short compared with the average amount of work per processor.

In conclusion, the authors argue that a programmer should think about work and critical path instead of focusing on communication costs and execution time, which allows characterization of performance of an algorithm regardless of machine configuration. Additionally, Cilk guarantees performance as a function of work and critical path. As of the writing of the paper, the authors admit that Cilk is best at executing asynchronous, tree like computations and that they are still working on applying Cilk to more tradition parallel applications.