Multithreaded Programming in Cilk

Matteo Frigo
Cilk Arts
12 Waltham Street, Lexington, MA 02421

Categories and Subject Descriptors

D.3.2 [**Software**]: Programming Languages

General Terms

Languages

Keywords

`cilk`, `spawn`, `sync`

1. INTRODUCTION

Cilk is a C-based algorithmic, multithreaded language for parallel programming which was developed at the MIT Laboratory for Computer Science over the past 15 years under the leadership of Charles Leiserson. Cilk minimally extends the C programming language to allow interactions among computational threads to be specified in a simple and high-level fashion.

Cilk's features a provably efficient runtime system that dynamically maps a user's program onto available physical resources, freeing the programmer from concerns of communication protocols and load balancing. In addition, Cilk provides an abstract performance model that a programmer can use to predict the multiprocessor performance of an application from its execution on a single processor. Both the runtime system and the performance model are based upon a study of the scheduling of multithreaded computations [1, 2], which proved that well-structured multithreaded programs could be scheduled efficiently using a "work-stealing" scheduler.

Cilk programs not only scale up to run efficiently on multiple processors, they also "scale down": Cilk programs execute on on one processor with minimal overhead compared to an equivalent sequential C program. This efficiency was attained by employing a two-clone compilation strategy and a Dekker-like protocol for work stealing [7].

Cilk is easy to use. Recently, Cilk won the *2006 HPC Challenge Class 2* competition at the *ACM Supercomputing 2006* conference [8]. The Cilk entry implemented and achieved good speedups on all six of the Challenge benchmarks and was cited for "Best Overall Productivity" (see `http://www.hpcchallenge.org/`). No other entrant implemented all benchmarks.

PASCO'07, July 27–28, 2007, London, Ontario, Canada.
ACM 978-1-59593-741-4/07/0007.

Cilk is especially suited for problems with irregular dynamic parallelism, such as combinatorial search. For example, several world-class parallel chess programs were written in Cilk [5], winning prizes in several international computer chess competitions.

A fundamental shift in semiconductor technology is moving the computer industry en-masse to multicores, and as a consequence research systems like Cilk are maturing into fully supported commercial offerings. Cilk Arts was incorporated in the Fall of 2006 and is developing an industrial-strength version of Cilk, which will support C and C++ on Windows and Linux/Unix platforms.

In this talk, I will provide a tutorial on the Cilk language for people with a basic background in computer programming. I will explain how to program multithreaded applications in Cilk and how to analyze their performance. I will also briefly sketch how the software technology underlying Cilk actually works.

2. THE CILK LANGUAGE

The philosophy behind Cilk development has been to make the Cilk language a true parallel extension of C, both semantically and with respect to performance. On a parallel computer, Cilk control constructs allow the program to execute in parallel. If the Cilk keywords for parallel control are elided from a Cilk program, however, a syntactically and semantically correct C program results, which we call the ***C elision*** (or more generally, the ***serial elision***) of the Cilk program. Cilk is a ***faithful*** extension of C, because the C elision of a Cilk program is a valid implementation of the semantics of the program, although it is not the only permitted semantics. With respect to performance, a parallel Cilk program "scales down" to run on one processor nearly as fast as its C elision.

The basic Cilk language can be understood from an example. Figure 1 shows a Cilk program that computes the nth Fibonacci number.[1] Observe that the program would be an ordinary C program if the three keywords `cilk`, `spawn`, and `sync` were elided.

The keyword `cilk` identifies `fib` as a ***Cilk procedure***, which is the parallel analogue to a C function. Parallelism is created when the keyword `spawn` precedes the invocation of a procedure. The semantics of a spawn differs from a C function call only in that the parent can continue to ex-

[1]This program uses an inefficient algorithm which runs in exponential time. Although logarithmic-time methods are known [3, p. 850], this program nevertheless provides a good didactic example.

```
#include <stdlib.h>
#include <stdio.h>
#include <cilk.h>

cilk int fib (int n)
{
    if (n<2) return n;
    else {
        int x, y;
        x = spawn fib (n-1);
        y = spawn fib (n-2);
        sync;
        return (x+y);
    }
}

cilk int main (int argc, char *argv[])
{
    int n, result;
    n = atoi(argv[1]);
    result = spawn fib(n);
    sync;
    printf ("Result: %d\n", result);
    return 0;
}
```

Figure 1: A simple Cilk program to compute the nth Fibonacci number in parallel (using an inefficient exponential-time algorithm).

ecute in parallel with the child, instead of waiting for the child to complete as is done in C. Cilk's scheduler takes the responsibility of scheduling the spawned procedures on the processors of the parallel computer.

A Cilk procedure cannot safely use the values returned by its children until it executes a `sync` statement. The `sync` statement is a local "barrier," not a global one as, for example, is used in message-passing programming. In the Fibonacci example, a `sync` statement is required before the statement `return (x+y)` to avoid the anomaly that would occur if `x` and `y` are summed before they are computed. In addition to explicit synchronization provided by the `sync` statement, every Cilk procedure syncs implicitly before it returns, thus ensuring that all of its children terminate before it does.

Cilk also provides a limited number of advanced features for nondeterministic programming. Its "inlet" feature, inspired in part by TAM [4], allows a value returned by a child to be incorporated into the parent's frame when the child is done, but before the parent has synched. In order to simplify reasoning about inlet behavior, Cilk provides implicit atomicity of inlets without requiring locking, declaration of critical regions, and the like. Sometimes, a procedure spawns off parallel work which it later discovers is unnecessary. This "speculative" work can be aborted in Cilk using the `abort` primitive inside an inlet.

The basic Cilk language contains little else that is not already in C. A library of mutual exclusion locks is provided, as well as extensions of C's storage allocation routines `malloc()`, `free()`, `alloca()`, etc. Cilk is deceptively simple, as complexity is handled by the runtime in order to enable the programmer to take advantage of parallel computing power with minimal effort.

Cilk's compiler and runtime platform work together to execute Cilk programs efficiently on multicore CMP's and other shared-memory multiprocessors. The Cilk development environment provides a compiler, runtime platform, and tool chain which allow fast development of high-performance applications for CMP's, while providing an effective software-release strategy in which Cilk programs can be easily regression-tested. Unlike traditional high-performance computing tools, Cilk program scale down to run as efficiently as C programs on a single processor and scale up with near-perfect linear speed-up. By employing divide-and-conquer recursion, Cilk programs also exploit the cache hierarchy well [6].

3. REFERENCES

[1] R. D. Blumofe and C. E. Leiserson. Space-efficient scheduling of multithreaded computations. In *Proceedings of the Twenty Fifth Annual ACM Symposium on Theory of Computing*, pages 362–371, San Diego, California, May 1993.

[2] R. D. Blumofe and C. E. Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM*, 46(5):720–748, Sept. 1999.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1990.

[4] D. E. Culler, A. Sah, K. E. Schauser, T. von Eicken, and J. Wawrzynek. Fine-grain parallelism with minimal hardware support: A compiler-controlled threaded abstract machine. In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 164–175, Santa Clara, California, Apr. 1991.

[5] D. Dailey and C. E. Leiserson. Using Cilk to write multiprocessor chess programs. *The Journal of the International Computer Chess Association*, 2002.

[6] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science*, pages 285–297, New York, New York, Oct. 17–19 1999.

[7] M. Frigo, C. E. Leiserson, and K. H. Randall. The implementation of the Cilk-5 multithreaded language. In *Proceedings of the ACM SIGPLAN '98 Conference on Programming Language Design and Implementation*, pages 212–223, Montreal, Quebec, Canada, June 1998. Proceedings published ACM SIGPLAN Notices, Vol. 33, No. 5, May, 1998.

[8] B. C. Kuszmaul. A Cilk response to the HPC Challenge (Class 2, productivity). Presented at the the HPC Challenge BOF at Supercomputing '06, Tampa, FL. `http://bradley.csail.mit.edu/~bradley/hpcc06.`, Nov. 2006.