6.895 Theory of Parallel Systems Alexandru Caracaş

Fast Serial-Append File I/O Mode Support for *Cilk*

Project Proposal

1 Introduction

Apart from resources, such as memory and processors, parallel computations also require file I/O. There exist a number of interesting file I/O modes, depending on the specific application. One particularly interesting file I/O mode is serial-append, described in [5]. In this mode the file output of a parallel computation, executed on any number of processors, is "the same"[1] as the output of the sequential, single processor execution. *Cheerio*, an implementation of serial-append presented in [5], uses a special formated file where ordering metadata about the parallel execution of the computation is saved together with the actual data. This allows for later reconstruction of the output of the sequential execution. The algorithm described in [5] uses a single mutex-ed counter that allows a worker processor to obtain a new node to write to, when a steal operation occurs in *Cilk*'s scheduler. Additional locking mechanisms are also performed by *Cilk*'s Runtime System in order to correctly maintain the order of the structure required by *Cheerio*. Due to the high overhead of the steal operation this method does not scale up with the number of processors.

2 Proposal

I propose analyzing different methods of efficiently[2] performing serial-append file I/O from *Cilk* programs. I will investigate the possibility of using (concurrent) B-trees as an underlying data structure. In my research, I would also consider using the information provided by a journal file system such as ReiserFS. The ReiserFS Linux-filesystem (see [1]) implements B-trees for its file and directory structure and according to benchmarks it is one of the fastest journal file systems for Linux. Another factor that influenced my decision towards a particular file system is that the new version of ReiserFS (v.4) will allow plugins.

Subsequently, implement a low overhead serial-append algorithm that scales up with the number of processors. The algorithm (and its implementation) requires a close interaction with *Cilk*'s Runtime System and Scheduler are required.

2.1 Goals

The project will concentrate on serial-append as a file I/O mode for *Cilk* programs. The goals of the project are:

1. Devise an efficient algorithm for performing serial-append from *Cilk* programs.
2. Analyze three different possibilities of implementing the algorithm:
   (a) directly at kernel level by using the ReiserFS API
   (b) using a special formated file on top of an existent file system
   (c) designing a new *Cilk* file system
3. Implement the algorithm using one of the above methods.

[1] By *same* is meant which allows for an easy reconstruction, read and seek operations

[2] By *efficient* is meant with relatively low overhead on Cilk's Runtime System and Scheduler and which scales up with the number of processors.

4. Port *Cheerio* (written for *Cilk* 5.2) to the new version of *Cilk* (5.4).
5. Compare my implementation with the *Cheerio* solution.
6. Parallelize serial applications (e.g. bz2 compression program), to use serial-append and analyze their performance.

3 Plan

The order of the goals suggests the planned schedule. Each task would have ten days to be completed. Some tasks (e.g. step 1) may be allowed to take up to three extra days.

3.1 Backup

In case that the implementation of the algorithm (step 3) would prove to be difficult (e.g. due to systems issues) its priority will be lowered and it will be deferred. Also implementation problems might arise with step 6. In this case this part would be skipped or other, simpler programs will be used (e.g. the compress or gzip utilities). If the most important part, namely step 1, were to fail then parts 4 and 6 could still be accomplished.

Also it would be worth spending effort in making a Linux-kernel support for the *Cheerio* file format.

4 Background

As background to support me with this project I have read the work stealing algorithm described in [2], which is implemented in *Cilk*'s scheduler. The serial-append I/O mode also conforms to the dag memory consistency model described in [3]. For an efficient implementation of serial-append a concurrent B-tree implementation might be necessary as described in [4] and [10] . The *Cilk* reference manual [6] will serve as a primary programming guide. Other readings include the documentation of the journal file system ReiserFS [1] and Linux Programming guides such as [7] and [8]. Also I have background in operating system and file system concepts as described in [9].

References

[1] ReiserFS v.4. Available at http://www.namesys.com/.

[2] Robert D. Blumofe, Matteo Frigo, Chrisopher F. Joerg, Charles E. Leiserson, and Keith H. Randall. An analysis of dag-consistent distributed shared-memory algorithms. In *Proceedings of the Eighth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 297–308, Padua, Italy, June 1996.

[3] Robert D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 356–368, Santa Fe, New Mexico, November 1994.

[4] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, Massachusetts, 1990.

[5] Matthew S. DeBergalis. A parallel file I/O API for Cilk. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 2000.

[6] Supercomputing Technology Group MIT Laboratory for Computer Science. *Cilk 5.3.2 Reference Manual*, November 2001. Available at http://supertech.lcs.mit.edu/cilk/manual-5.3.2.pdf.

[7] Bill O. Gallmeister. *Linux Device Drivers.* O'Reilly & Associates, Inc., 1995.

[8] Alessandro Rubini and Jonathan Corbet. *Linux Device Drivers.* O'Reilly, second edition, 2001.

[9] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts.* John Wiley & Sons, Inc., sixth edition, 2002.

[10] Paul Wang. An in-depth analysis of concurrent B-tree algorithms. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, February 1991. Also available as MIT Laboratory for Computer Science Technical Report MIT/LCS/TR-496.