PARALLEL ACTIVITY ROADMAPS

Daniel Citron, **Dror G. Feitelson***, and **Iaakov Exman**
Institute of Computer Science
Hebrew University of Jerusalem
Jerusalem - 91904 - Israel
Email: ianex@cs.huji.ac.il

Parallel Roadmaps are simple visual constructs, useful for displaying the evolution of large-scale parallel programs with dynamic parallelism. Roadmaps remain intelligible and provide invaluable debugging clues even as program complexity increases. A key design feature is the selective omission of details to be displayed, coupled with the option to focus on part of the program to obtain additional details.
Keywords: massive parallelism, monitoring, debugging, animation.

1. INTRODUCTION

This paper introduces *parallel roadmaps* of activities for debugging and monitoring of parallel programs. Such roadmaps are especially useful in systems that employ dynamic parallelism, with activities being created and terminating during execution.

Parallel roadmaps are a central feature of the `mkmonitor` program development environment. This environment is oriented towards the logical structure of parallel programs, in terms of the activities that compose them. Consistent with this approach, its front-end is machine independent and is driven only by the types of defined events (activity creation and termination, blocking for synchronization, etc.). Nevertheless, some of its tools may also display hardware system structure. By providing a set of coordinated interactive tools, `mkmonitor` allows users to exploit these apparently naive, but powerful visual constructs.

The reader is forewarned that the static, disconnected black and white pictures don't do justice to interaction with the live system. A video tape shown at the conference, and available upon request, is a closer approximation.

2. PARALLEL ROADMAPS

Parallel activities — generated by parallel programs — have a sort of autonomous life. They can exist in a certain PE (processing element), be suspended, restart running, spawn children, or even migrate to other PEs. In our system, parallel programs are written in *ParC*, a parallel version of *C* [1]. Activities are defined by the blocks of a `parblock` construct, or by the iterations of a `parfor`, which are executable in parallel. These are closed constructs based on the parbegin/parend paradigm. They can be mixed and nested as desired, creating a tree of activities in which the leaves are active.

*Current address: IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598

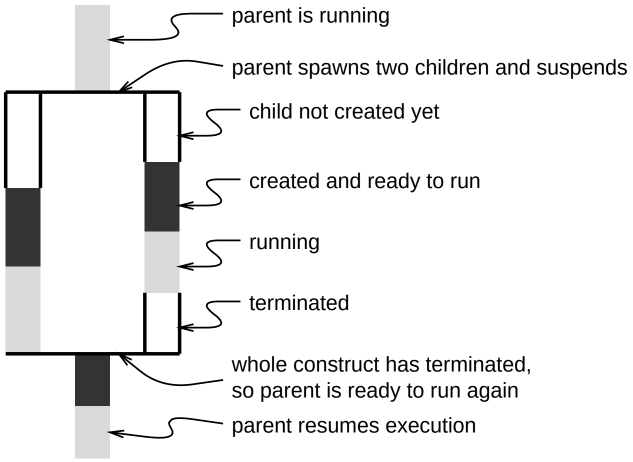

Figure 1. *The basic elements of a parallel roadmap.*

A *history road* is a display of how the activity changes from one state to another, along the time axis. The different states are marked by different colors. For instance in Fig. 1 one sees two activities in parallel, which were spawned by a common ancestor. The time increases from top to bottom. *White* stands for a ghost state: the parent is suspended, due to the spawn, but the children have not been created yet (or similarly before resuming the parent). *Black* means ready to run. *Gray* is actually running. The parent is not shown when it is suspended: it is effectively colored *transparent*.

A *parallel roadmap* is just a collection of history roads for all the activities in the program. It allows the viewer to capture the structure of program execution in one glance. An example is given in Fig. 2, which shows the execution of a Fibonacci calculation.

One can traverse the map interactively, using the mouse. For example, it is possible to display the activity tree or the histogram of loads on the different PEs for a certain instant in time. Moving continuously along the time axis, one creates an animation of these two views. Other options in the tool-set allow measurements of the effective parallelism at each instant — see the ruler in the left-hand side of Fig. 2.

3. VISUALIZING LARGE-SCALE PARALLELISM

The naive appearance of roadmaps is misleading. With large increase of the degree of parallelism, they remain uncluttered, and thus easily comprehensible. Only the mode of use changes. One loses sight of detailed structure and observes more global forms at a glance.

In Fig. 3, with slightly different color conventions from previous figures, one can see the result of a parent spawning 100 activities, executing in a system with 11 PEs. One clearly perceives a "wavefront" of activity startups. All these activities contain a `sync` command, which is reflected in the strictly horizontal line at two thirds from the top — the instant of synchronization.

Roadmaps aid in debugging by their visual emphasis on anomalies of execution. For

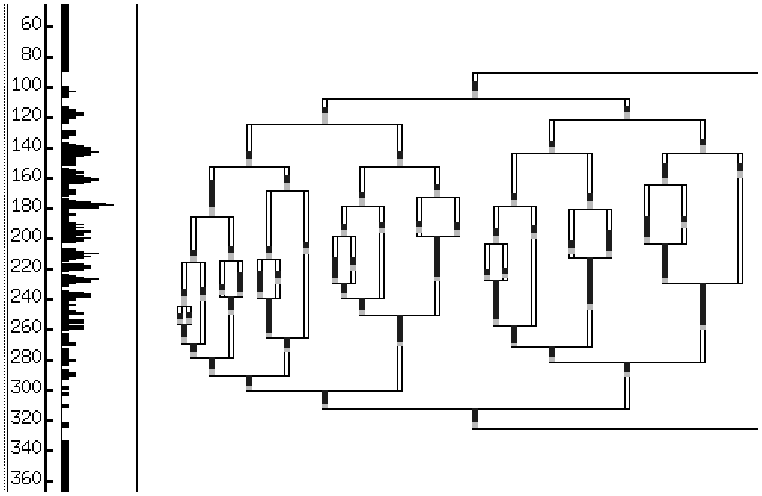

Figure 2. *Example of the roadmap for a Fibonacci calculation.*

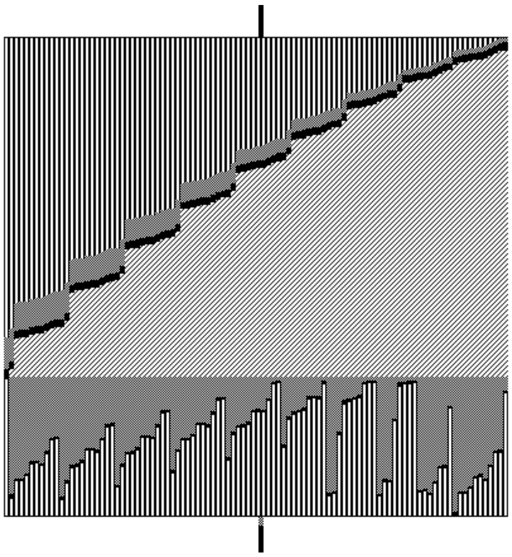

Figure 3. *Global view at a glance: activities wavefront and their barrier synchronization.*

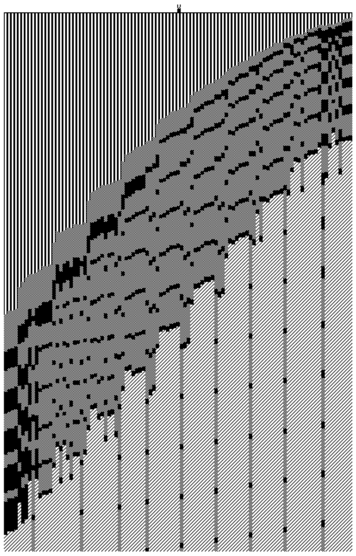

Figure 4. *Anomalies easily discerned in complex computations: infinite loop detected by the unbounded "teeth" projecting downwards from the wavefront.*

example, in Fig. 4 an infinite loop is easily discerned by the unbound "teeth" projecting out of the normally terminating activities wavefront.

4. DISCUSSION

`mkmonitor` was implemented to monitor the execution of *ParC* programs on the Makbilan shared memory parallel processor, which was assembled in our laboratory. Currently in use are logs on disk file for post-mortem processing, display and animation tools. The display driver uses X-lib and X-intrinsics, for portability.

The *roadmaps* of `mkmonitor` are especially suitable to systems with parbegin/parend parallelism. They are not that important for systems where the number of activities is constant, e.g. one per PE. Most other graphical visualization systems do not have views that handle dynamic parallelism effectively. They usually lack views of the logical structure of the program. At best, they have some view that shows the mapping of activities to PEs.

One of the reasons for monitoring is to understand the performance of the application. Performance depends on the product of two factors: the percentage of the time the code is running, after subtracting overhead in the runtime system, and the efficiency when running, after subtracting wait states for synchronization and shared memory access. `mkmonitor` places an emphasis on the first factor. Monitoring the efficiency of execution requires more instrumentation and additional graphical views. For example, it becomes important to know about the mapping of data structures to activities and to PEs. Most other monitoring systems emphasize this aspect of program visualization [2–4]. The price is that it might be harder to interpret the results, and correlate the multiple views.

The most important characteristic of *roadmaps* is their broad range of usefulness with respect to degree of parallelism, from the detailed structure of a few tens of activities to a global view of a complex computation with hundreds of activities. Roadmaps do not get cluttered as program complexity increases. This stems from the parsimonious and very intuitive display of activities as straight lines parallel to the time-axis — any other angle shows up as a clue to computation behavior.

REFERENCES

1. Y. Ben-Asher, D. G. Feitelson, and L. Rudolph, "ParC — an extension of C for shared memory parallel processing". *Software — Pract. & Exp.*, to appear.
2. M. T. Heath and J. A. Etheridge, "Visualizing the performance of parallel programs". *IEEE Software* **8(5)**, pp. 29–39, Sep 1991.
3. C. Kilpatrick and K. Schwan, "ChaosMON — applications-specific monitoring and display of performance information for parallel and distributed systems". *ACM/ONR Workshop on Parallel and Distributed Debugging*, pp. 57–67, May 1991.
4. D. Socha, M. L. Bailey, and D. Notkin, "Voyeur: graphical views of parallel programs". *ACM Workshop on Parallel and Distributed Debugging*, pp. 206–215, May 1988.