Kremlin: Like gprof, but for Parallelization

Donghwan Jeon Saturnino Garcia Chris Louie Sravanthi Kota Venkata Michael Bedford Taylor
University of California, San Diego
{djeon, sat, cmlouie, skotavenkata, mbtaylor}@cs.ucsd.edu

Abstract

This paper overviews Kremlin, a software profiling tool designed to assist the parallelization of serial programs. Kremlin accepts a serial source code, profiles it, and provides a list of regions that should be considered in parallelization. Unlike a typical profiler, Kremlin profiles not only work but also parallelism, which is accomplished via a novel technique called *hierarchical critical path analysis*. Our evaluation demonstrates that Kremlin is highly effective, resulting in a parallelized program whose performance sometimes outperforms, and is mostly comparable to, manual parallelization. At the same time, Kremlin would require that the user parallelize significantly fewer regions of the program. Finally, a user study suggests Kremlin is effective in improving the productivity of programmers.

Categories and Subject Descriptors D.2.2 [*Software Engineering*]: Design Tools and Techniques; D.1.3 [*Programming Techniques*]: Concurrent Programming—parallel programming

General Terms Measurement, Performance

1. Introduction

As multicore processors dominate mainstream computing, more programmers are forced to parallelize their serial programs. Although tools such as OpenMP and Cilk++[4] have been proposed to help programmers with this task, one important question has been largely overlooked: "which parts of the program should I spend time parallelizing?"

`gprof` and similar profilers provide a solution to a similar problem in serial optimization. A conventional profiler models a program as a hierarchical region where a region typically represents a loop or a function and possibly contains other subregions. The profiler produces a list of regions ordered by their work coverage. We call this ordered list a "plan" because it focuses the programmer's efforts on the regions where optimization is likely the most fertile: those regions with the largest percentage of work. Unfortunately, traditional profilers have limited value during parallelization because they do not profile parallelism, leaving the programmer to manually determine if a region has any parallelism.

In this paper, we overview Kremlin, a profiling tool that is designed to help during the parallelization of a serial program. Kremlin produces a plan based on not only work but also parallelism. In accordance with Amdahl's Law, Kremlin calculates the speedup from parallelizing a region by using both its parallelism and its work, ordering regions by their calculated speedup.

```
$> make CC=kremlin-cc
$> ./tracking data
$> kremlin tracking --openmp

     File (lines)          Parallelism   Cov.(%)
1    imageBlur.c (49-58)   145.3         9.7
2    imageBlur.c (37-45)   145.3         8.7
3    getIPatch.c (26-35)   25.3          8.86
...  ...                   ...           ...
```

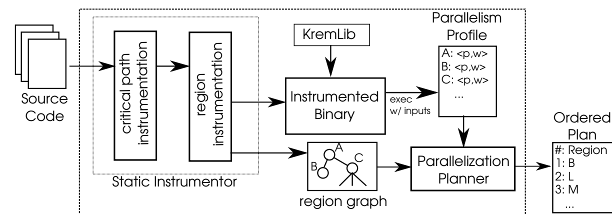


Figure 1. (top) Kremlin Usage Model and (bottom) the Overview of Kremlin System Architecture

The major challenge in a profiler for parallelization is to extract the region-localized parallelism for each region. The region-localized parallelism represents the maximum speedup in a region when only that region–not its subregions–are parallelized. Previous work in critical path analysis (CPA) [3] measured the amount of parallelism in a program but did not localize parallelism to specific regions. Kremlin overcomes this problem by employing a novel technique called *hierarchical critical path analysis, or HCPA*. Based on the program's region hierarchy and CPA results for each region, *HCPA* extracts region-localized parallelism.

From our preliminary evaluation with NAS Parallel Bench (NPB) [2], Kremlin turns out to be very effective. Kremlin reduces the number of regions parallelized by 1.59X on average compared to a third-party parallelization, while outperforming the third-party's performance by as high as 1.85X, with a geometric mean of 1.08X. A user study shows that Kremlin greatly reduces the amount of time that users waste on trying to parallelizing regions that offer little-to-no benefit.

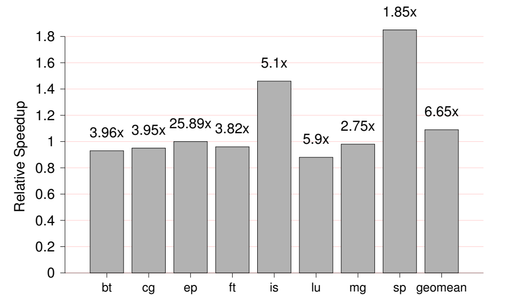
2. Kremlin Overview

Figure 1 illustrates the Kremlin system. Kremlin's usage model is similar to `gprof` (see Figure 1-top). The user compiles input code with a drop-in replacement compiler called `kremlin-cc` to generate an instrumented binary. After the user executes the generated binary, Kremlin outputs a plan consisting of the regions that should be parallelized based on the parallelism profile gathered at runtime. Internally, Kremlin consists of two major modules: the static instrumentor and the parallelization planner (see Figure 1-bottom).

Copyright is held by the author/owner(s).
PPoPP'11, February 12–16, 2011, San Antonio, Texas, USA.
ACM 978-1-4503-0119-0/11/02.

Benchmark	Total	Third-Party	Kremlin	**Reduction**
bt	447	54	27	**2.00x**
cg	135	22	9	**2.44x**
ep	41	1	1	**1.00x**
ft	447	6	6	**1.00x**
is	59	1	1	**1.00x**
lu	509	28	11	**2.55x**
mg	653	10	8	**1.25x**
sp	1366	70	58	**1.21x**
Overall	3657	192	121	**1.59x**

(a) Plan Size Comparison



(b) Relative (bar) and Absolute (number) Speedup of Kremlin

Figure 2. Evaluation of Kremlin based Parallelization. Compared to a third-party manual parallelization [1], Kremlin-based parallelization achieves comparable or better speedups with less regions parallelized.

Static Instrumentor The instrumentor inserts function calls in the source code so that Kremlin correctly captures the regions hierarchy and gathers parallelism-relevant information. The instrumented code is linked to `KremLib`, which implements the instrumentation functions. HCPA specially treats false dependences and easy-to-break dependences as they can create a false impression of seriality in the region. Although the overhead of HCPA instrumentation could be high, the use of a compression technique significantly lowers the overhead. In NPB, the compression technique reduces the average log file size from 17.9GB to 150KB.

Parallelization Planner The parallelization planner produces a plan based on the information gathered from instrumentation. From localized parallelism and work, the planner can calculate the impact on application speedup when a region is parallelized. However, parallelizing a region could affect the potential parallelization benefit in other regions, complicating the design of the planner. For example, consider a doubly-nested loop where both inner and outer loops are parallel. Parallelizing the outer loop directly impacts the benefit of parallelizing the inner loop as the work coverage of the loop nest in the program has already shrunk when the outer region is parallelized. The planner iteratively selects the region with the maximum application speedup, updating the speedup of other regions based on the region hierarchy and the parallelization status of each region. This greedy algorithm finishes when the calculated speedup does not meet a minimum threshold value.

3. Preliminary Results

Our preliminary results examine three key aspects of Kremlin: its plan size, the speedup gained from following it, and its impact on a programmer's productivity. We examined NPB because a third-party OpenMP parallelization is available [1]. We ran Kremlin on the serial run of each benchmark to get a parallelism plan, and created a Kremlin-based parallel implementation. All results were gathered on a 32-core system (8x AMD Opteron 8380 processors).

Kremlin Plan Size To determine Kremlin's ability to reduce the number of regions considered during parallelization, we measured the number of parallelized regions in the third-party parallel version and compared it to Kremlin's plans. Figure 2(a) shows this information. Kremlin recommends 1.59X fewer regions on average compared to the third-party version. Furthermore, Kremlin never recommends more regions than the third-party version, suggesting its value in reducing a programmer's effort.

Kremlin Plan Performance While the smaller plan produced by Kremlin can reduce the effort to parallelize a program, the plan would be of limited utility if it leads to poor performance. To determine the performance of Kremlin based parallelization, we compared it with the performance of the third-party parallel version. Figure 2(b) shows Kremlin significantly outperforms the manual parallelization in `is` (1.46X) and `sp` (1.85X) while offering competitive performance in remaining benchmarks, resulting in 1.08X geometric mean speedup against the third-party. The improved performance in `is` and `sp` resulted from Kremlin uncovering parallelism at a coarser-grain than was exploited by the third-party.

Impact on the Productivity of Programmers In order to see Kremlin's impact on the productivity of programmers, we performed a user study involving seven students in a graduate–level course at UCSD. Students were split into two groups (A and B) and asked to parallelize two programs. For program 1, group A had access to both `gprof` *and* Kremlin while group B had access to only `gprof`. For program 2, the tool access was alternated to normalize group differences in the study. We identified the critical regions, those that bring a speedup over 5%, of the two programs via extensive manual parallelization before the experiment. We measured the time students spent in parallelizing these critical regions as opposed to other regions. In both programs, the group who had access to Kremlin spent a much larger percentage of time on critical regions (84% vs 56% in program 1, 92% vs 49% in program 2). Although the sample size is admittedly small, this result suggests that Kremlin improves programmer productivity by reducing the time spent on regions that do not have a large benefit.

4. Conclusion and Future Work

Kremlin helps parallelization by providing a parallelism plan that ranks regions in the order of importance in parallelization. From our experiments with NPB, Kremlin was able to significantly reduce the number of regions parallelized compared to a manual parallelization, without sacrificing the performance. Also, our user study demonstrates a programmer with Kremlin tends to focus on regions that actually bring a speedup from parallelization, suggesting Kremlin would improve a programmer's productivity. We are currently striving to improve the quality of the parallelization planner by incorporating a wider range of information such as program structure, parallelization platforms, and target machine parameters. *This work was funded in part by NSF Award 0725357.*

References

[1] "NAS Parallel Benchmarks 2.3; OpenMP C." `www.hpcc.jp/Omni/`.

[2] Bailey et al. "The NAS parallel benchmarks." In *SC*, 1991.

[3] M. Kumar. "Measuring parallelism in computation-intensive scientific/engineering applications." *IEEE TOC*, Sep 1988.

[4] C. E. Leiserson. "The Cilk++ concurrency platform." In *DAC*, 2009.