Computational Science and Engineering
(Int. Master's Program)

Technische Universität München

Master's Thesis

Parasite: Local Scalability Profiling to Assist Parallelization

Author:	Nathaniel Knapp
1st examiner:	Prof. Dr. Michael Gerndt
2nd examiner:	Prof. Dr. Michael Bader
Thesis handed in on:	November 15, 2016

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

July 27, 2016 Nathaniel Knapp

Acknowledgments

Andreas Wilhelm

Abstract

The Lehrstuhl für Rechnertechnik und Rechnerorganisation at TUM has developed an interactive parallelization tool, Parceive, which analyzes data dependencies and profiling information collected from a trace that uses dynamic binary instrumentation [1]. Using the visualization provided by this tool, programmers can easily relative execution time of function calls and data accesses to memory nodes. This information allows them to more easily identify areas of the code that are parallelization candidates, at multiple granularity levels [1]. The object of this thesis is to design and implement Parasite, a local scalability profiling tool that will extend the capabilities of Parceive.

Parasite will extend the approach of a recent master's thesis in the Lehrstuhl für Rechnertechnik und Rechnerorganisation that uses user annotations of parallel sections and information from the Parceive trace to provide scalability estimates for the entire program [2]. It will include a feature that ranks functions for the user in order of their potential speedup from parallelization, given the number of cores available. The visualization will also be extended to add a feature where users can select shared memory locations which could require lock synchronization, and then see the effect of these synchronizations on scalability. Test programs have been selected from the NASA Advanced Supercomputing (NAS) parallel benchmark suite, to guide the development of the tool [3].

Contents

Outline

Part I: Introduction and Theory

CHAPTER 1: INTRODUCTION
This chapter presents an overview of the thesis and its purpose.

CHAPTER 2: THEORY
This chapter discusses the parallel programming theory relevant to local scalability profiling, including lock synchronization.

CHAPTER 3: RELATED WORK
This chapter discusses related research to the Parasite tool.

Part II: The Parasite Scalability Profiler

CHAPTER 3: LOCAL SCALABILITY PROFILING
This chapter presents and describes the development of the part of Parasite which provides estimates of speedup for individual functions.

CHAPTER 4: LOCKS
This chapter presents and describes the development of the part of Parasite which allows the user to estimate speedup including lock synchronization.

CHAPTER 5: IMPLEMENTATION
This chapter describes the implementation of Parasite in C++, including how the Parasite tool retrieves necessary information from the database used for the Parceive tool, and how the algorithms in chapters 3 and 4 are implemented to analyze this information.

Part III: Results and Conclusion

CHAPTER 6: RESULTS
This chapter presents and describes tests of Parasite using benchmark programs selected for their attributes.

CHAPTER 7: CONCLUSION
This chapter summarizes the capabilities of the tool developed, and provides an outlook describing the possibilities for future tools that could implement further scalability profiling for parallelization.

Part I.

Introduction and Theory

1. Introduction

As multicore processors have become widespread, so has the need to parallelize legacy code so that it fully uses the scalabilty potential of these processors. However, parallelization is time-consuming and error-prone, so most legacy software "still operates sequentially and single-threaded" [1]. Several challenges of parallel programming explain the gap in development between hardware and software.

One challenge is successful design of a parallel program that operates concurrently. The program's computation may be split into components that run on different threads. Without proper synchronization, operations that access the same memory locations easily lead to nondeterministic behavior due to race conditions. Hence, parallelization of serial programs requires identifiying dependencies and refactoring to use multiple threads in a way in which these dependencies do not create race conditions. Another challenge is load balancing: evenly dividing a computation into threads.

The Lehrstuhl für Rechnertechnik und Rechnerorganisation at TUM has developed an interactive parallelization tool, Parceive, which helps programmers overcome these design challenges for shared memory systems [1]. Figure 1.1 illustrates the high-level components of Parceive. The Parceive tool takes an executable as input, and using Intel's Pin tool, dynamically instruments predefined instructions, including function calls and returns, memory accesses, memory allocation and release, and calls from threading APIs such as Pthread. This instrumentation inserts callbacks that are used to write trace data into a database at runtime. The visualization component of Parceive, described next, uses the trace data to generate views of the application useful for parallelization. Then, a developed interpreter reads the trace stored in the database sequentially in chronological order. An API allows the user to acquire information from events of interest generated from reading the database. The Parasite tool, described after the visualization, interfaces directly with these events to acquire information it needs to analyze the application's scalability.

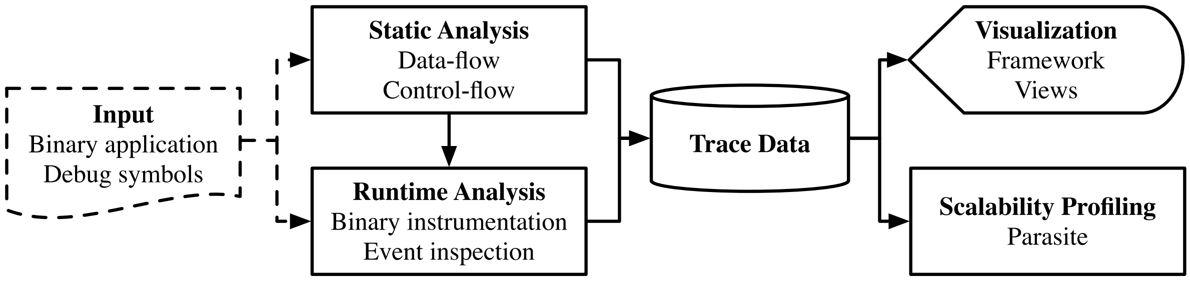

Figure 1.1.: Steps of the Parceive tool.

The visualization component of Parceive consists of a framework that allows the integration of multiple views. For example, the function view displays data accesses to memory nodes, so that programmers can see where dependencies exist, and how they might structure the program to avoid race conditions. The performance view displays the relative execution time of function calls, as shown in Figure 1.3, so that programmers can identify ways in which they can evenly split function workloads between threads. Finally, the calling context tree view, shown in Figure 1.4, displays a call tree consisting of call nodes, loop nodes, and memory nodes, so that programmers can more easily comprehend how dependencies relate to the dynamic behavior of an application.

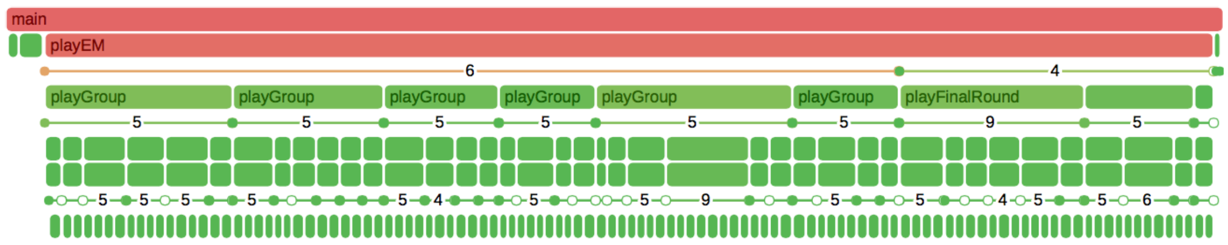

Figure 1.2.: The performance view of Parceive showing an European Championships simulation trace.

The goal of this thesis is to extend Parceive with a scalabiliy profiling tool called Parasite. This tool is useful in two ways. First, it helps programmers improve existing parallelizations to attain higher speedups. Second, it helps the user analyze hypothetical parallelizations: the user can visually annotate applications with parallel regions and synchronization features, and Parasite provides a forecast for the resulting scalability. Before fully explaining Parasite's purpose, it is useful to introduce the upper speedup bounds on parallel programs and the concept of parallelism.

"P processors can execute at most P instructions in unit time" [4], which creates the first speedup constraint, the work law. Here, **work** "is the total time of all instructions executed by a computation" and $T_{parallel}$ is the parallel execution time:

$$T_{parallel} \geq work/P$$

The maximum speedup from parallelization increases linearly with the number of procesors at first, because it is determined by the work law. However, as the number of processors increases, they eventually cannot affect the speedup, because at least one of the processors must execute all instructions on the critical path, or **span**: the length the "longest (in time) path of dependencies –in the computation" [5]. This upper bound for the speedup possible on any number of processors is called the parallelism. **Parallelism** "is the ratio of a computation's work to it's span" [5].

The goal of Parasite is to help the programmer identify why programs that are correctly parallelized do not reach their theoretical upper bound on speedup. In the following paragraphs four types of limitations on speedup are listed, along with descriptions of how the programmer can use Parasite to observe and analyze these limitations:

1. **Insufficient parallelism**: The program contains serial bottlenecks that inhibit its scalability. Parasite calculates the parallelism of every call site in the program. A call site is a specific line in the program where a function is called, and measurements for this call site include all child function calls of the function. The programmer can therefore use Parasite to see which call sites may have insufficient parallelism by looking at their parallelism, and the role of these call sites within the program, using Parceive's visualization. They can then focus their effort on improving the parallelization of problem call sites.

2. **Scheduling overhead**: The work that can be done in parallel is too fine-grained to be worth distributing to other processors. Call sites causing scheduling overhead can be identified using Parasite by comparing the number of processors used for each call site to the parallelism of the call site. The parallelism for the entire program and each call site should greatly outnumber the number of processors in use for the call site to avoid scheduling overhead [5].

3. **Contention**: A processor is slowed down by simultaneous interfering accesses to synchronization primitives, such as mutex locks, or by the true or false sharing of cache lines. The programmer can use Parasite to investigate the impact of contention involving locks on parallelism, using an interactive visualization that allows easy selection of shared memory locations to lock. The tool will then automatically calculate an upper bound on speedup with locks on these shared memory locations, without the programmer changing the source code. Calculation of an upper bound on speedup is possible here without parallelization of the source code because the tool solely operates on a trace that contains information about memory accesses.

4. **Insufficient memory bandwidth**: The processors access memory too quickly for the bandwidth of the machine's memory network to sustain [5]. If the program's overall measured speedup is not close to the parallelism, and use of the Parasite tool has ruled out insufficient parallelism, scheduling overhead, and contention as possible causes, insufficient memory bandwidth is likely preventing the parallelization from achieving its potential speedup.

This thesis will provide a background for the Parasite tool, describe its algorithms and implementation, and show its application to relevant use cases. It is structured as follows. Chapter 2 will describe relevant theory to the Parasite tool, including the DAG model for multithreading, and the theory behind contention for locks. Chapter 3 will describe other scalability profiling tools that have been developed, some of which use approaches similar to Parasite. Chapter 4 will descibe the algorithm Parasite uses to calculate the parallelism for the program and each call site, and chapter 5 will describe the algorithm used to calculate the parallelism including locks. Chapter 6 will describe how these algorithms are implemented, particularly how they interface with the Parceive tool. It will also describe the interactive visualization used for analyzing the effects of locks on speedup. Chapter 7 will describe the impact of applying the Parasite tool on a wide diverse sample of test cases and developer use cases. Finally, chapter 8 will discuss the impact of the Parasite tool, possible future extensions to the tool, and the outlook for future scalability profiling tools.

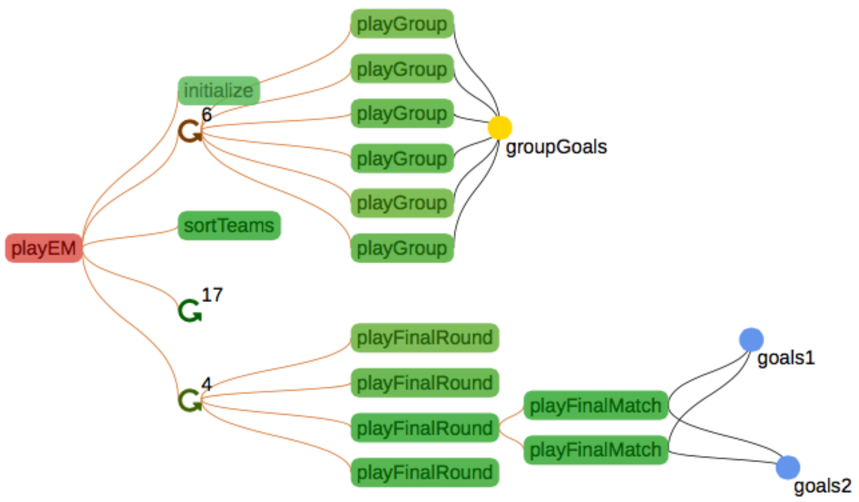

Figure 1.3.: The calling context tree view of Parceive.

2. Theory

3. Related Work

Part II.

The Parasite Scalability Profiler

4. Local Scalability Profiling

5. Locks

6. Implementation

Part III.

Results and Conclusion

7. Results

8. Conclusion

Bibliography

[1] A. Wilhelm, B. Sharma, and M. Gerndt, "Parceive: Interactive parallelization based on dynamic analysis," *Program Comprehension through Dynamic Analysis (PCODA), 2015 IEEE 6th International Workshop on*, March 2015.

[2] R. Guder, "Performance prediction to assist parallelization," Master's thesis, Technische Universität München, 2016.

[3] N. A. S. Division, "Nas parallel benchmarks." http://www.nas.nasa.gov/publications/npb.html.

[4] W. M. L. Yuxiong He, Charles E. Leiserson, "The cilkview scalability analyzer," *SPAA '10*, 2010.

[5] C. E. L. e. a. Tao B. Schardl, William M. Leiserson, "The cilkprof scalability profiler," *SPAA '15*, 2015.