Calum McCartan
cm4114

Revised Project Proposal

Overview

I would like to implement an extension for Visual Studio Code that assists with introducing parallel workers into JavaScript programs. Although JavaScript is a highly asynchronous language, Web Workers are required to introduce true parallelism which takes advantage of multiple cores on a machine's hardware. Like my midterm paper, this project is about parallelism refactoring tools, but focuses on JavaScript / VSCode which were not used in any of the papers I covered.

Motivation

A tool for introducing workers to JavaScript may have quite niche use cases compared to tools discussed in my paper, however the choice of target language should be quite novel, and I expect it to be a more feasible challenge with the given time constraints.

A laptop/PC running a web browser client will be unlikely to have a large number of cores available compared to a backend server where a greater potential speedup may be possible. However, I still believe there are scenarios on a front-end client (such as image processing) where even on just a quad-core machine, some parallelism could be significantly faster than none.

Traditional deadlocks and race conditions are almost non-existent in JS Web Workers due to the way in which JS communicates with workers, however there are still major advantages of using a tool to generate worker code rather than performing a manual implementation. While using web workers in JS is easier than using parallelism in other languages, it will still be beneficial for guiding beginners, and even experienced users should find that the tool can save them some substantial time.

Implementation

I have chosen VS Code both because it is a popular IDE choice for JavaScript / web applications, but also because it has significant support for extensions. Babel (https://babeljs.io/) is a library for generating and manipulating the AST of a JS program, and I plan on using it extensively in my implementation for locating and analysing sites to introduce workers.

Other refactoring tools covered in my paper made use of parallel libraries in their automated refactorings. I will likely do the same with the catiline library (http://catilinejs.com/), which reduces the code required to make use of JS web workers.

I intend on targeting for/while loops, and automatically breaking tasks down into groups which can be split amongst several workers. In addition I would like to perform some safety checks to determine if the tasks within a loop are safe to be performed by a worker, as there are some restrictions on what data a worker can use.

Evaluation

In my midterm paper I identified some goals of parallel refactoring tools. I will evaluate the extent to which my tool covers these, and compare it to others. I will also do some small experiments to discover the magnitude of any obtained speedup on some example programs. Finally, I may conduct a small user study to collect comments on the general usability of the tool.

Revised Proposal Update

I plan to continue the project as described in the original proposal. I have set up a small test program that uses Babel so that I know I am able to install and use the library without problems. I have also learned more about JavaScript Web Workers, and created a small program with them to investigate potential performance improvements that might be obtained with the tool which I am developing.