Heartbeat Scheduling

Provable efficiency for nested parallelism

Research performed at Inria and Indiana University

Team

- Umut Acar
- Arthur Charguéraud
- Adrien Guatto
- Mike Rainey
- Filip Sieczkowski

Overview

A classic problem in parallel computing is to take a high-level parallel program written, for example, in nested-parallel style with fork-join constructs and run it efficiently on a real machine. The problem could be considered solved in theory, but not in practice, because the overheads of creating and managing parallel threads can overwhelm their benefits. Developing efficient parallel codes therefore usually requires extensive tuning and optimizations to reduce parallelism just to a point where the overheads become acceptable.

In this paper, we present a scheduling technique that delivers provably efficient results for arbitrary nested-parallel programs, without the tuning needed for controlling parallelism overheads. The basic idea behind our technique is to create threads only at a beat (which we refer to as the "heartbeat") and make sure to do useful work in between. We specify our heartbeat scheduler using an abstract-machine semantics and provide mechanized proofs that the scheduler guarantees low overheads for all nested parallel programs. We present a prototype C++ implementation and an evaluation that shows that Heartbeat competes well with manually optimized Cilk Plus codes, without requiring manual tuning.

- In the proceedings of PLDI'18 (Acar et al. 2018); Authors' copy
- Video of talk at PLDI'18
- Slides from talk
- Coq proof scripts

Running and extending our expriments

We encourage interested parties to evaluate the findings of our empirical study.

The source code of our prototype implementation is hosted on a Github repository.

Prerequisites

To have enough room to run the experiments, your filesystem should have about 300GB of free hard-drive space and your machine at least 128GB or RAM. These space requirements are so large because some of the input data we use are huge.

We use the nix package manager to handle the details of our experimental setup. The first step is to install nix on your machine.

Setting up the environment

First, we need to obtain the source code.

```
$ git clone https://github.com/deepsea-inria/heartbeat.git
$ cd heartbeat/script
```

If the machine does not already store a copy of the input data, we run the following command, which will just build the binaries for the experiments and the benchmark script.

```
$ nix-build
```

Otherwise, if the input data already exists on the machine, run the following.

```
$ nix-build --argstr pathToData <path to the folder with the input data>
```

If it succeeds, then the `nix-build` command should leave behind a new symlink named `result` in the `script` folder. This results folder stores all the files generated by nix build script.

The benchmarking binaries we are going to use are now reachable from `result/bench`. To save time typing, let us add this folder to our `$PATH`.

```
$ export PATH=`pwd`/result/bench/:$PATH
```

Running the experiment

The next and final step is to run the benchmark script. This process involves downloading all the input data (unless it's already present on the machine), running the benchmarks, and generating plots.

```
$ bench.pbench compare -runs 30
```

It will take at least a few hours to complete the thirty of runs for each data point. To get results faster, but with lower statistical significance, reduce the

number of runs. If, after getting the initial results, you wish to add more runs, just run again, but this time, pass the argument `-mode append`.

When the experiment completes, the results table should appear as a pdf file in a newly created `_results` folder. The latex source for the table is also generated and stored in the same folder.

References

Get the bibtex file used to generate these references.

Acar, Umut A., Arthur Charguéraud, Adrien Guatto, Mike Rainey, and Filip Sieczkowski. 2018. "Heartbeat Scheduling: Provable Efficiency for Nested Parallelism." In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 769–82. PLDI 2018. New York, NY, USA: ACM. doi:10.1145/3192366.3192391.