ECE6960 Heterogeneous Programming Programming Assignment 2 Report

Cheng-Hsiang Chiu u1305418
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT-84112
{cheng-hsiang.chiu}@utah.edu

I. Introduction

In this assignment I implemented parallel reduction using static scheduling and guided scheduling.

II. Implementations

There are three implementations in this assignment. They are:

1) Sequential solution. This is considered as the baseline. It is denoted as `Seq` in the experimental results.
2) Parallel solution with static scheduling. It is denoted as `Static` in the experimental results. This solution performs a parallel reduction using static scheduling. Static scheduling means that the chunk size for every worker is constant through the execution.
3) Parallel solution with guided scheduling. It is denoted as `Guided` in the experimental results. This solution performs a parallel reduction using guided scheduling. Guided scheduling means that the chunk size is dynamic for every worker. The chunk size can be formulated in the following algorithm.

ALGORITHM 1: Guided Algorithm

```
threshold = 2 * (default_chunk_size+1) * num_workers
if remaining_iterations ≤ threshold then
    chunk_size = default_chunk_size
end
else
    chunk_size = remaining_iterations/(2*num_workers)
end
```

III. Availability

The whole implementation is available in the Github repository, link.

IV. Build and Run Executable

There is a README in the uploaded compressed file and in the Github repository mentioned above. README provides very detailed information. In this section, I highlight the instructions to build and run the executable `main`. Please make sure CMake and g++ is installed.

To build the executable `main`, use the following commands,

```
mkdir build
cd build
cmake ../
make
cmake ../
make
```

We need to run "cmake ../" and "make" twice to find the installed google benchmark libraries.

To run the executable, use the following command in folder `build`,

```
./main
```

V. Results

The executable was built using g++ with -std=c++17 with and without -O2 optimization on a Linux machine with Intel i7-9700K 8 Cores at 3.6 GHz and 32 GB RA. All results were obtained using Google benchmark [1].

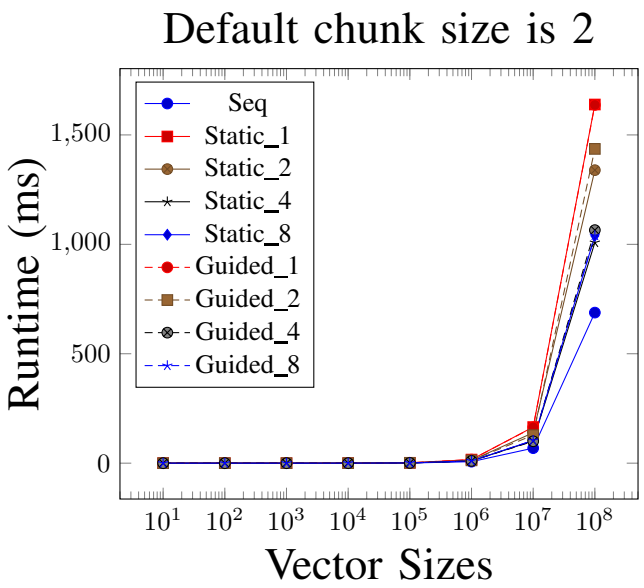

Fig. 1: Runtime comparisons between sequential reduction, parallel reduction with static scheduling, and parallel reduction with guided scheduling under various vector sizes. The default chunk size for parallel solutions is two. The executable is compiled without optimization. The suffixes of legends of parallel solutions refer to the number of threads.

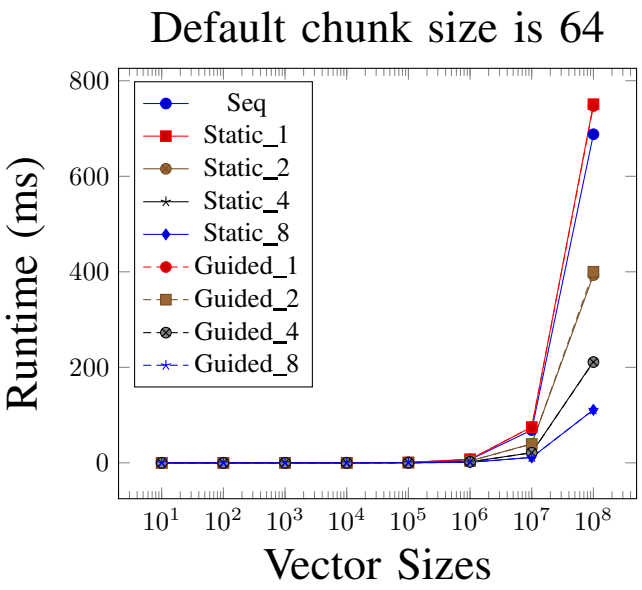

Fig. 2: Runtime comparisons between sequential reduction, parallel reduction with static scheduling, and parallel reduction with guided scheduling under various vector sizes. The default chunk size for parallel solutions is 64. The executable is compiled without optimization. The suffixes of legends of parallel soulutions refer to the number of threads.

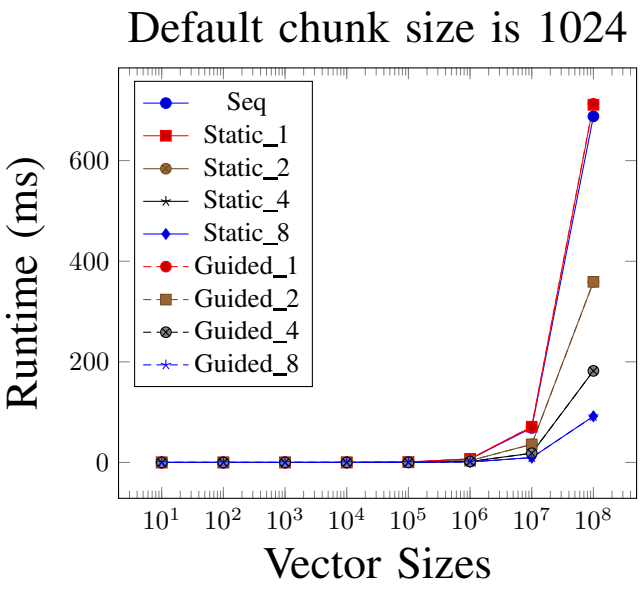

Fig. 3: Runtime comparisons between sequential reduction, parallel reduction with static scheduling, and parallel reduction with guided scheduling under various vector sizes. The default chunk size for parallel solutions is 1024. The executable is compiled without optimization. The suffixes of legends of parallel soulutions refer to the number of threads.

The workload here is the reduction of all integers in a vector. For executions without optimization enabled, in Figures 1, 2, and 3 we can see that parallel reductions using static or guided scheduling perform much better that sequential reduction. The more threads and higher chunk size we use the better performance parallel reductions have over sequential reduction. For example, for one hundred million vector elements and 8 thread, both parallel reductions are 6.5 times and 5.2 times faster than sequential reduction for chunk size with 1024 and 64, respectively. However, at chunk size 2 sequential reduction

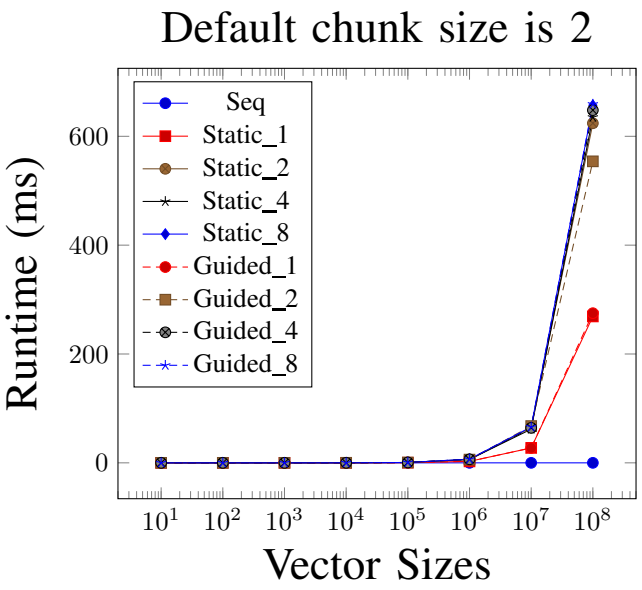

Fig. 4: Runtime comparisons between sequential reduction, parallel reduction with static scheduling, and parallel reduction with guided scheduling under various vector sizes. The default chunk size for parallel solutions is two. The executable is compiled with O2 optimization. The suffixes of legends of parallel solutions refer to the number of threads.

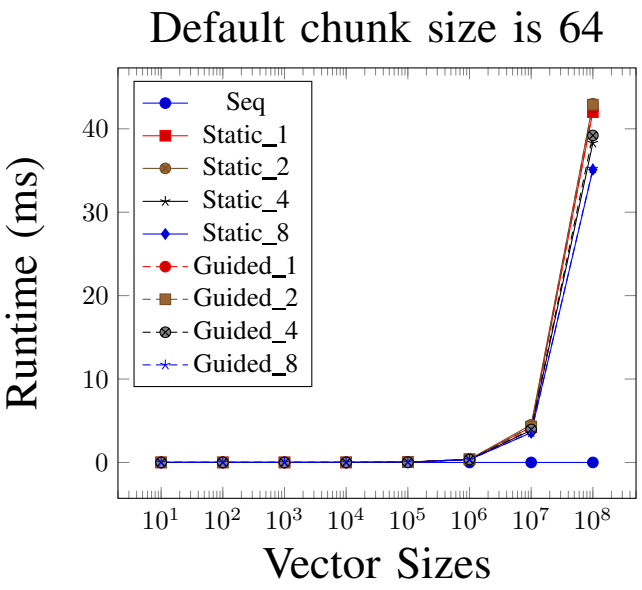

Fig. 5: Runtime comparisons between sequential reduction, parallel reduction with static scheduling, and parallel reduction with guided scheduling under various vector sizes. The default chunk size for parallel solutions is 64. The executable is compiled with O2 optimization. The suffixes of legends of parallel soulutions refer to the number of threads.

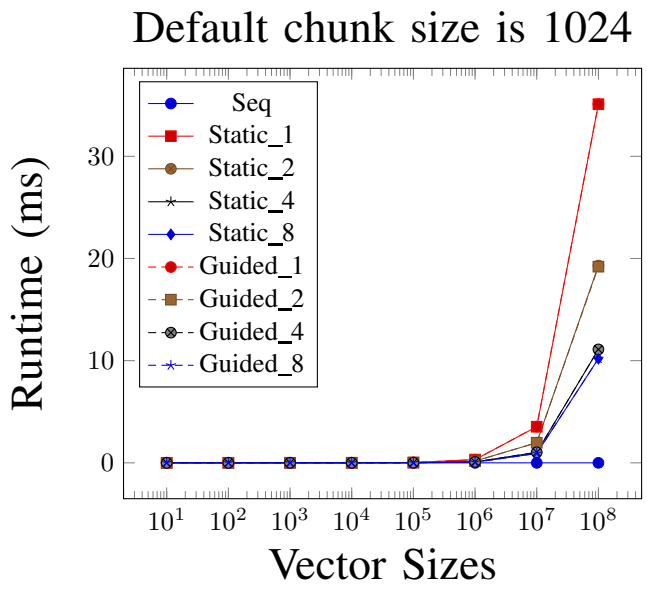

Fig. 6: Runtime comparisons between sequential reduction, parallel reduction with static scheduling, and parallel reduction with guided scheduling under various vector sizes. The default chunk size for parallel solutions is 1024. The executable is compiled with -O2 optimization. The suffixes of legends of parallel soulutions refer to the number of threads.

has better performance because there are too many small tasks created which is a dramtical overhead here.

For executions with O2 optimization enabled, in Figures 4, 5, and 6 we can see that sequential reduction performs better than parallel reductions using static or guided scheduling regardless of the number of threads and the chunk size. That is, for a simple workload that reduces integer elements in a vector, the sequential reduction with an optimization enabled is the better choice than any parallel solutions.

REFERENCES

[1] "Google benchmark," GitHub. [Online]. Available: https://github.com/google/benchmark