Investigating Performance of Parallel Counting and Sorting Algorithms on Dense Vector Spaces In MATLAB

Karan Abraham [ABRKAR004][‡], Bonga Njamela [NJMLUN002][§]
EEE4120F 2024
University of Cape Town
South Africa

***Abstract*—An experiment was conducted to investigate the performance of the high-level constructs provided in the MATLAB Parallel Computing Toolkit, such as `parfor` loop and the `gpuArray`.**

I. Introduction

Parallel computing involves dividing tasks into smaller processes that can be solved simultaneously. The MATLAB Parallel Computing Toolkit allows users to solve problems that require intensive computational and data resources by exploiting parallelism and multi-threading capabilities of processors and GPUs. This experiment leveraged functions and loops in the toolkit such as `parpool` which can initialise a pool of worker threads with one worker per physical CPU core [1]. The built-in multi-threading feature in MATLAB automatically prepares the models and blocks such that they can be executed on multiple threads. On the other hand, the Parallel Computing Toolbox has multiple explicit parallelism mechanisms that are facilitated by a parallel process-based pool or a thread-based parallel pool [2]. This enables the usage of additional hardware resources through threads and processes to improve the performance of a program.

An experiment investigated the improvements in performance when implementing thread-based pools that employed a parallel-for-loop (`parfor`), and a parallel execution of a single program with multiple data (`spmd`). The `parfor` loop was used performing counting and comparing operations on a matrix. Speedup graphs were plotted to evaluate the improvement in the execution time of the parallel program when compared to a sequential counterpart that used the standard `for` loop.

Parallel computing can also be performed on a GPU. MATLAB can accelerate linear algebra operations implicitly by running it on a GPU, provided that two criteria are met. The first criterion is that the time spent on the computation must exceed the time spent on transferring data to and from the GPU memory. Secondly, implicit GPU acceleration requires that computations to be able to be divided into smaller units of work [2]. The Parallel Computing Toolbox includes a special array type that can be initialised using the command `gpuArray` which relocates data from the MATLAB workspace to the GPU memory. A program is executed on the GPU whenever a function call contains at least one `gpuArray` as an input argument and the data is stored on the GPU [3]. While both a GPU and the `spmd` function can apply the same process on multiple data, `spmd` uses multiple workers from a parallel pool whereas programs that do not have a `gpuArray` do not execute on the GPU unless they fulfill the above criteria. However, if a machine has several GPUs, MATLAB assigns a different GPU to each worker by default or a user can initialise a parallel pool with as many workers as available GPUs.

II. Methodology

We expect the parallel algorithm to perform faster than the serial algorithm. We will test this using Matlab.

III. Results and Discussion

IV. Conclusion

References

[1] MathWorks. `parpool`: Creating parallel pool on cluster. [Online]. Available: https://www.mathworks.com/help/parallel-computing/parpool.html
[2] MathWorks. Run matlab on multicore and multiprocessor machines. [Online]. Available: https://www.mathworks.com/discovery/matlab-multicore.html
[3] ——. Run matlab functions on a gpu. [Online]. Available: https://www.mathworks.com/help/parallel-computing/run-matlab-functions-on-a-gpu.htm