Supporting Parallelism in Mat-Cloud

Yongpeng Zhang
Xing Wu

1 Problem Description

This project is an important step to realize an efficient and full-fledged matlab cloud service on GPUs. Based on our previous work that supports coarse-grained matlab scripting, the focus of this project is to efficiently execute fine-grained parallel for loops and sections on GPUs.

The major work of this project includes:

- Parfor: **Parfor** is a keyword in Matlab that allows user to write a loops for a statement or block of code that executes in parallel.
- Parallel Section: **Parallel Section** is a pragma that indicates there is no dependency between multiple code blocks and therefore these blocks can be executed concurrently. This pragma is not available in native Matlab script. But we think it worths to add to MatCloud to further exploit the computation power of GPUs.

2 Approaches

The support of loop structure, the premise of the proposed work, is missing in the current MatCloud implementation. Therefore, the first task of the work is to enhance the existing parser and the Abstract Syntax Tree (AST) traversal algorithm to correctly handle the loop structure inside the user script.

Once loop is supported, user can replace the *for* keyword with *parfor* whenever there is no dependency between loop iterations. In the simplest case, it is the responsibility of the user to guarantee that loops are dependency-free. To parallelize the loop structure, we can assign each iteration of the loop to a single GPU thread. The operand matrices will be stored in the global memory of the device, so that each thread can easily access the data pertaining to the iteration assigned to it. During the code generation, the data placement and memory allocation should be carefully designed such that frequently accessed data is placed in the low-latency shared memory and temporary automatic variable is also created whenever necessary to take advantage of the fast on-chip registers. Furthermore, we also propose to implement the detection of loop dependency, which, in the first place, enables us to warn the user of any abuse

of the *parfor* keyword, and moreover, helps us to automatically detect any potential parallelism in the input script. The dependency detection can be done by analyzing each loop, constructing direction vector and distance vector to identify any RAW/WAR/WAW dependencies between iterations.

Besides the support of the basic *parallel for* style loop parallelism, we also propose to support the reduction across loop iterations, *e.g.*, the accumulation of the results of each iteration. In order to use this feature, user should specify the way the reduction is done, including at least the operation and the target variable. The challenge of reduction over the aforementioned simple loop parallelizing is the generation of the reduction algorithm. A carefully designed reduction algorithm can be several times faster than a naive one. An even larger challenge of reduction is the detection of the reduction logic. We eventually wish to combine the automatic detection of loop parallelism and reduction logic, so that the users can write their script without explicitly specifying the parallelism inside their implementation.

The acceleration of Parallel Section comes from the multi-stream feature in the latest generation of graphic card (fermi) in nVidia. Our idea is to map independent section blocks to multiple stream objects in CUDA. It involves certain amount of code generation and run-time support. Care must be taken to handle synchronizations between streams to guarantee data readiness after the parallel sections. This benefits the most when each section block is relatively small and cannot consume all computation resource by itself alone. Inside the *parfor* loop, we need to consider the case when any use of coarse-grained library call is used inside iteration. Those library calls are mapped into kernels already. Because we cannot call another kernel inside CUDA thread, we will map this kind of for loops into multiple streams if possible.

Finally, for all the proposed topics above, we need to investigate the technique to launch cuda compiler on-the-fly and load binaries dynamically. We think this can be done by compiling CUDA into dynamic library.

3 Milestones

For/while loop	For	YZ	Oct. 31
	While loop	YZ	
Parfor	Detect Reduction	XW	Nov. 12
	Code Generation	XW	Nov. 21
Parallel Section	Code Generation	YZ	Nov. 12
	Run-time support	YZ	Nov. 21
Dependence Detection	XW YZ		Nov. 30
Demo	XW YZ		Dec. 3