**BabelTower: Learning to Auto-parallelized Program Translation**

(GPT: editing grammar and format version)

**What He Did:** He proposed a learning-based framework called BabelTower. He created a large-scale dataset consisting of compute-intensive function-level monolingual corpora. He also used back-translation with a discriminative reranker to cope with unpaired corpora and parallel semantic conversion.

**Why He Did This:** There is a significant difference between the sequential C and the parallel CUDA programming models. Existing approaches fail to address the challenges of auto-parallelized program translation.

**Why It Is Important:** GPUs have become the dominant computing platforms for many applications. Programming GPUs using the widely-used CUDA parallel programming model is difficult because writing parallel programming requires awareness of complex architectural characteristics of GPUs, such as thread/block data organization, on-chip shared memory, and warp synchronization. Conversely, C is easy to program sequentially. Thus, automatically translating to CUDA can relieve the burden of GPU programming.

**Weakness of Previous Research:** Automating this process requires not only syntax-level translation (e.g., generating CUDA-specific keywords like threadIdx.x) but also loop detection, parallel semantic analysis, and sequential-to-parallel conversion. Previous research cannot address the challenges of unpaired corpora and parallel semantic conversion. State-of-the-art auto-parallelization approaches that use code templates or polyhedral models for program transformation require manual efforts for code annotation or face scalability and generality problems.

**Advantages of This Research:** This is the first to provide a publicly available large-scale C-CUDA dataset. It introduces a learning method for translating from C to CUDA, which addresses the key challenges of unpaired corpora and parallel semantic conversion.

**Challenges of This Research:** Program translation is particularly challenging in practice due to the CUDA programming model, which follows the SIMT (Single Instruction, Multiple Threads) model to partition data into different parts with the same code, differing significantly from conventional C programming in both syntax and semantics. It requires loop detection, parallel semantic analysis, and sequential-to-parallel conversion, making auto-parallelized program translation more difficult than traditional automatic parallelization on program translation tasks. Another approach where statistical program translation trains either probabilistic models or neural networks, inspired by recent advances in statistical/neural machine translation using end-to-end automatic code translation with large-scale datasets, fails to address this problem due to two challenges: scarcity of effective datasets and lack of parallel semantics.

**How to Get Through It:** They proposed a novel learning-based framework, BabelTower, specifically for auto-parallelized program translation from sequential C to parallel CUDA. The author created a large-scale dataset consisting of 501,732 C functions and 129,487 CUDA functions, including C-CUDA function pairs for validation and testing. These are compute-intensive to evaluate the effectiveness of parallel semantic conversion. They are using back-translation with a discriminative reranker, which means they first leverage the widely-used data augmentation technique (back-translation) to enable unsupervised translation from C to CUDA based on larger-scale unpaired monolingual corpora. Then the parallel semantics are embedded into a discriminative model for selecting the best hypothesis within the n-best beam search candidates. Experimental results show that BabelTower outperforms the state of the art by scores of 1.79, 6.9, and 9.39 in terms of BLEU, Code-BLEU, and Para-BLEU, respectively. Additionally, 92.8% of the generated CUDA code can be correctly compiled, speeding up execution by 347x over the original sequential C code, and improving developer productivity of real-life CUDA programs by up to 3.8x.

**Weaknesses and Improvements:**

(original version):

**What he did:** he propose a learning-based framework: BabelTower. He creates a large-scale dataset consisting of compute-intensive function-level monolingual corpora. And use the back-translate with a discriminative reranked to cope with unpaired corpora and parallel semantic conversion.

**Why he did this：**. And huge different between the sequential C and the parallel CUDA programming model. There are not approaches fail conduct the challenging: auto-parallelized program translation.

**why it is important:** because GPU become the dominant computing planforms for many applications. And programming GPU with the widely- used CUDA parallel programming model is difficult. because of writing parallel programming model is quite hard since the programmer should be aware of complicated architectural characteristics of GPU and thread/block data organization, on-chip shared memory and warp synchronization. But C is sequential C code is easy to be programming. So, automatically translating to CUDA is relieve the burden of GPU programming.

**What is the weakness of previously research**: automating such process requires not only syntax-level translation (e.g., generating the CUDA-specific keywords such as threadIdx.x), but also loop detection, parallel semantic analysis, and sequential-to-parallel conversion. They cannot address the challenges of unpaired corpora and parallel semantic conversion. And the state-of-the-art auto parallelization approaches employ code template or polyhedral model for program transformation, which they need manual efforts for code annotation or confront scalability and generality problem.

**what is the advantage of this research:** they have the first to provide a publicly available large-scale C-CUDA dataset. And, Introduction a learning method for translating for C to CUDA, which addresses the key challenges of unpaired corpora and parallel semantic conversion.

**What is the challenge of this research:** the challenge is the program translation is greatly challenging in practice, because CUDA programming model, which follows the SIMT (single instruction, Multiple Threads) model to partition the data into different parts with the same code, which is quite different from conventional C programming in syntax and particularly semantics. And it requires loop detection, parallel semantic analysis and sequential-to-parallel conversion. So, that means the auto-prarllelized program translate, is difficulty than traditional automatic parallelization on program translation tasks. (I didn’t understand). There is another approach which the statistical program translation approaches train either probabilistic model or neural networks, they are inspired by recent advances on statistical/neural machine translation which use end-to-end automatic code translation which large-scale dataset, but this approach fail to address this problem because they are face two challenges: scarcity of effective dataset, and lack of parallel semantics

**how to get through it:** they proposed a new novel learning-base framework which is BabelTower. Which is specific for auto-parallelized program translation which is for sequential C to parallel CUDA. Autor create large-scale dataset consisting of 501,732 C function s 129,487 CUDA functions. And have C-CUDA function pairs for validation and test. All of which are compute-intensive (didn’t get it) to evaluate the effectiveness of parallel semantic conversion. And they are using back-translation with a discriminative reranked. Which means they first use leverage the widely used data augmentation technique(back-transition) to enable unsupervised translation from C to CUDA based on larger-scale unpaired monolingual corpora. Then the parallel semantics are embedded into a discriminative model for selection the best hypothesis within the n-best beam search candidates. Which experimental result show that BabelTower outperforms state of the art by 1.79, 6,9, and 9,39 in terms of BLEU, Code-BLEU and Para-BLEU, so 92.8% generated CUDA code can be correctly complied. And speed up 347x over the original sequential C code. And improves developer productivity of real life CUDA programs by at most 3.8x

**What is the weakness of this research? How to be improved it:**