Anton Efremov <aefremov@andrew.cmu.edu>
Kandasamy Chokkalingam <kchokkal@andrew.cmu.edu>

Midway Report for 15-618 Final Project
"Sequence Alignment Using AWS Trainium"

Summary

Over the past two weeks, we've made steady progress toward our project goals:

- We explored various programming options for AWS Trainium and selected nki.language and nki.isa as the most suitable tools
- While awaiting access to Trainium instances, we developed a Python-based Smith-Waterman (SW) algorithm that mirrors NKI's computation model
- Once partial access was granted[1], we set up the environment and began porting our algorithm to NKI
- We also explored relevant NKI primitives and familiarized ourselves with the NKI profiler using reference programs

Currently, we have ported about 95% of our implementation to NKI, and we expect to finalize the first working version within a day. We're actively investigating the limitations and characteristics of the NKI environment to guide our optimization efforts.

Progress vs Plan

We were able to significantly reduce the delay in our plan caused by the late access to Trainium instances, and we now believe that by the end of this week, we will be fully back on track. See https://aefremov88.github.io/NKI-sequence-alignment for an updated schedule.

In the remaining two weeks, we plan to complete the rest of our planned deliverables, including one EXTRA goal:

- Implement optimizations for the Smith–Waterman algorithm (which engines, pipelining, algorithmic changes)
- Start running and profiling on real workloads
- Conduct a holistic analysis across different workload types, including worst-case scenarios
- [EXTRA] Perform a detailed comparison with the CUDA implementation of SW, focusing on workload characteristics and data movement

For this extra deliverable, we have already reviewed the state-of-the-art CUDA implementation described in *CUDASW++4.0: ultra-fast GPU-based Smith–Waterman protein sequence database search*, and examined its workflow and data mapping strategy. We plan to analyze the

[1] 50% of quota was approved, which allows us to run parallel computations on one Trainium chip, but doesn't allow to explore additional parallelism by accessing the whole instance, comprising 16 Trainium chips forming a systolic array

differences between this GPU-based solution and our Trainium-based implementation to highlight architectural trade-offs.

After exploring the problem set up, we're considering replacing our second original EXTRA deliverable, "Implement multiple sequence alignment," with a deeper exploration of Trainium parallelism:

- Running on a larger instance (trn1.32xlarge) with 16 chips forming a 4×4 systolic array
- Optimizing our implementation using **nki.isa**, which allows finer control compared to the higher-level **nki.language** (which is more CUDA-like)

This shift will allow us to explore more interesting parallelization strategies provided by heterogenous Trainium architecture.

Poster Session

The central idea of our poster will be the mapping of the Smith-Waterman algorithm onto **AWS Trainium's unique processor architecture**. This mapping is both non-trivial and highly insightful. Trainium has a deeply heterogeneous, hierarchical structure that enables a lot of different kinds of parallelism:

- A trn1.32xlarge instance contains 16 Trainium chips, physically connected in a 4×4 2D systolic array topology
- Each Trainium chip has 2 NeuronCores
- Each NeuronCore contains 4 engines: scalar engine, vector engine, tensor engine (systolic array), and General Purpose SIMD engine

We will present a clear, layered **visualization of this architecture** alongside a breakdown of how the Smith–Waterman dynamic programming matrix is partitioned and pipelined through this hierarchy.

To make the performance implications of this mapping tangible, we will include **profiling results and visual execution traces**, captured using the NKI profiler. These traces will highlight compute vs. memory bottlenecks, utilization of engine pipelines, and insights into the advantages of Trainium's data and compute model. We think this will be particularly interesting because a lot of optimization challenges on NKI are different in specifics but similar in semantics to the challenges we've discussed in class. For example, data locality is incredibly important in NKI, but so is having the correct physical representation (which way a matrix is oriented, for example).

Additionally, we'll show a **comparison of the workflow on Trainium against GPU-based** SW implementations (CUDASW++ 4.0), focusing on differences in memory layout, synchronization, and data exchange. With this illustration, we will summarize how architectural choices impact algorithm performance and scalability.

Anton Efremov <aefremov@andrew.cmu.edu>
Kandasamy Chokkalingam <kchokkal@andrew.cmu.edu>

Our goal is for the poster to not just present an implementation but to present **new parallel approaches** being unlocked with the emergence of new generation hardware like AWS Trainium.

Challenges

The main external challenge we are currently facing is a **quota limitation**. We were approved for 64 vCPUs, which is enough to run our code on a single Trainium chip (using a trn1.2xlarge instance). However, to fully explore Trainium's parallelism - especially its 16-chip systolic array architecture - we would need 128 vCPUs to access a trn1.32xlarge instance. We've submitted a request for this higher quota and are waiting for a response, but this limitation does not block our core project goals; there is a risk for the extra experiments we planned around additional sources of parallelism.

Internally, the main difficulty is adapting to the new programming environment. Writing low-level code in NKI is different from using standard frameworks like CUDA or even C++. The debugging process takes time because there are fewer sources and limited documentation, and it is a lot of trial and error when it comes to understanding how primitives behave in real programs. One specific thing is running on the 4x4 larger 16-chip architecture - there seems to be a bit of ambiguity about how effective/feasible optimizing for that architecture will be and how much speed up we will see, but we believe we'll have more clarity on that in the next week.

Overall, we feel confident since now that we have a working development environment, things are moving much faster. We've already made strong progress on our implementation - around 95% of the program is debugged.