Speeding up Smith-Waterman using Numba

CPU and GPU (CUDA) implementations

Kevin Jun - June 11, 2020

Introduction

- DP algorithm for accurate and thorough sequence alignment
- Polynomial runtime: $O(n * m * l)$
  - n, m: length of sequences
  - l: number of sequences in database
- Python's Numba package
  - Just-in-time (JIT) compilation compiles functions to machine code at runtime
  - Optimizes loops and computations on numpy arrays

Recurrence relation for Smith Waterman

$$H_{ij} = \max \begin{cases} H_{i-1,j-1} + s(a_i, b_j), \\ \max_{k \geq 1}\{H_{i-k,j} - W_k\}, \\ \max_{l \geq 1}\{H_{i,j-l} - W_l\}, \\ 0 \end{cases} \qquad (1 \leq i \leq n, 1 \leq j \leq m)$$

Wikipedia

Background

- Many implementations to speed up Smith Waterman using computer architecture and engineering
- SIMD architectures for vectorization
- QIAGEN
  - Only company to offer both SSE-vectorized and FPGA solutions
  - Speed-ups of more than 110 over other standard implementations
- SWIPE software
  - Open-source; capable of comparing residues from sixteen different database sequences
  - 106 billion cell updates per second on dual Intel Xeon X5650 6-core system

Methodology/Approach

- Needed to refactor serial code for Numba/CUDA quirks
  - Only compiles Python functions
  - Numba likes loops and numpy arrays
  - Typing cannot be ambiguous
  - No support for many non-primitive Python data types
  - *nopython* mode is faster but stricter
  - arrays cannot be instantiated inside function; required empty arrays to be passed to function
- One thread per sequence in database

Results

- Numba code should be timed after compilation
- CPU Numba
  - 272 speedup over Serial
- CUDA
  - 8,457 speedup over CPU Numba
  - 2,304,163 speedup over Serial
  - $gridDim = 8$, $blockDim = 64$ was the "fastest"; unsure what this means regarding GPU saturation

Implementation	Runtime (seconds)
Serial	853.69264108
Numba CPU	3.13341230
CUDA <16,32>	0.0004447
CUDA <8,64>	0.0003705
CUDA<4,128>	0.0003776
CUDA<2,256>	0.0003848

Discussion

- With so many other faster implementations, now sure how mine will help the field
- Good case study in using HPC and CUDA to achieve speedups for algorithms with “time floors”
- Liked to do
  - Test against entire pdbaa database - unable to because Bio.SeqIO does not error check FASTA files
  - More rigorous CUDA testing - requires larger input and compilation information (can’t find this in the documentation) to calculate GPU saturation
  - Multi-dimensional grid and block kernel launches
  - Is there a better way to divide up work than one thread per sequence?

References

"CUDA Programming." *Introduction to Numba: CUDA Programming*, nyu-cds.github.io/python-numba/05-cuda/.

Harris, Mark, et al. "Numba: High-Performance Python with CUDA Acceleration." *NVIDIA Developer Blog*, 29 Apr. 2020, devblogs.nvidia.com/numba-python-cuda-acceleration/.

Harrism. "Harrism/numba_examples." *GitHub*, github.com/harrism/numba_examples/blob/master/mandelbrot_numba.ipynb.

"Notes on Literal Types." *Notes on Literal Types - Numba 0.50.0.dev0+236.g64fbf2b-py3.7-Linux-x86_64.Egg Documentation*, numba.pydata.org/numba-doc/dev/developer/literal.html.

"Supported Python Features in CUDA Python." *Supported Python Features in CUDA Python - Numba 0.50.0.dev0+236.g64fbf2b-py3.7-Linux-x86_64.Egg Documentation*, numba.pydata.org/numba-doc/dev/cuda/cudapysupported.html.

"Writing CUDA Kernels." *Writing CUDA Kernels - Numba 0.50.0.dev0+236.g64fbf2b-py3.7-Linux-x86_64.Egg Documentation*, numba.pydata.org/numba-doc/dev/cuda/kernels.html.

"A ~5 Minute Guide to Numba." *A ~5 Minute Guide to Numba - Numba 0.49.1-py3.6-Macosx-10.7-x86_64.Egg Documentation*, numba.pydata.org/numba-doc/latest/user/5minguide.html.