CS267 Final Project: Pairwise Alignment Work Stealing CPU/GPU

Brandon Wong, Eric Tang

May 2021

1 Introduction

In this project, we aimed to implement CPU work stealing from a shared work queue for the Smith-Waterman algorithm. Research has showed that the most optimal CPU implementation of Smith-Waterman is competitive with GPU implementations for an instruction set (SSE2) an implementation from 2013 [3] - thus it is important to both be able to take advantage of the parallel computation ability of GPUs, and the lack of the need for transfer latency on CPU, in order to optimize the Smith-Waterman algorithm on high performance compute clusters, where the compute environment is heterogeneous.

For the project, we worked off of an implementation of GPU-BSW [1], which is a batched version of the Smith-Waterman algorithm, for running parallel alignment computations on GPU owning threads. For non GPU owning threads, we used a SIMD version of Smith-Waterman [2]. We integrated the two by updating the kernel calls of the GPU-BSW library to pop work off a shared atomic work queue used by all of the currently running threads. We showed steady performance improvements with this split CPU/GPU approach to tackling Smith-Waterman, and showed how these two implementations can be used in conjunction with one another rather than independently.

In addition to this basic integration of CPU and GPU, we attempted additional optimizations, including work stealing on GPU owning threads, and tuning batch sizes for CPU and GPUs to take or block work off of the shared queue.

2 Prior Work

In recent years, advances in genomic sequencing have yielded large amounts of sequence data, of which computational analysis is required. A key step in this analysis is sequence alignment, in which the most similar regions between a query and reference sequence are found; a subset of this analysis is local sequence alignment, in which the most similar subsequence is computed. Although approximate methods exist, there is still great interest in using advances in high performance and parallel computers in order to find the exact solution to the local sequence alignment problem using the Smith-Waterman algorithm.

Recent work which has explored the space of using GPUs as an intermediate in computing local alignment for includes GASAL2 [5], which provides a CUDA library for various DNA and RNA sequence alignment algorithms, and the basis for this project, GPU-BSW [1], which provides an algorithm for performing batched Smith-Waterman on GPUs. Parallel CPU implementations of Smith-Waterman have also been deeply explored by researchers in the last few decades - prior work includes various versions of Smith-Waterman using various parallel libraries, including SIMD [2], OpenMP, and MPI.

However, there has been a lack of work surrounding adapting Smith-Waterman to the heterogeneous computing environment that shows great promise for further optimizing the speed of the local alignment problem due to the potential of having relatively larger numbers of CPUs available due to global GPU shortages.

3 Methods and Approach

Our solution to integrate the GPU and CPU implementations used a single shared global atomic work queue for both GPU owning and non GPU owning CPU threads to take work off of. Each GPU owning thread would take 20000 alignments to perform in parallel, and each CPU thread would take 10 alignments to perform sequentially (taking just 1 alignment results in thrashing behavior). The exact implementation used OpenMP to specify per thread behavior with a shared pointer and work stolen counter that would only be incremented in atomic sections of the code.

We converged on this approach after realizing that it would be a poor fit to define the load balancing using OpenMP task and target directives, which we discuss a bit more in section 7.1.

4 Results

We found that we were able to achieve significant increases in runtime that scaled relatively similar to projected optimal performance in a heterogeneous GPU-CPU environment. We have also added another environment with a faster CPU and slower GPU to contrast with the Cori environment, which uses top of the line V100 GPUs.

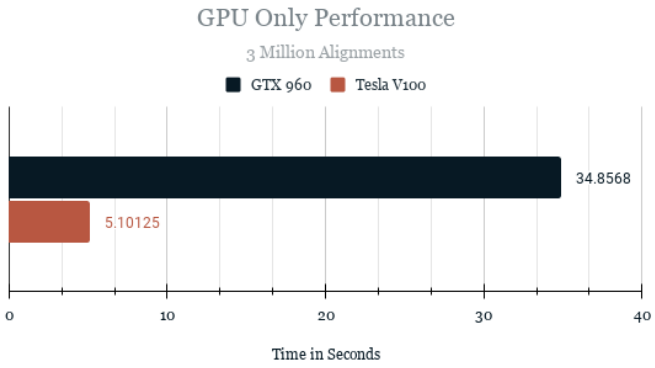

Figure 1: Here we have our baseline performance of GPU-BSW

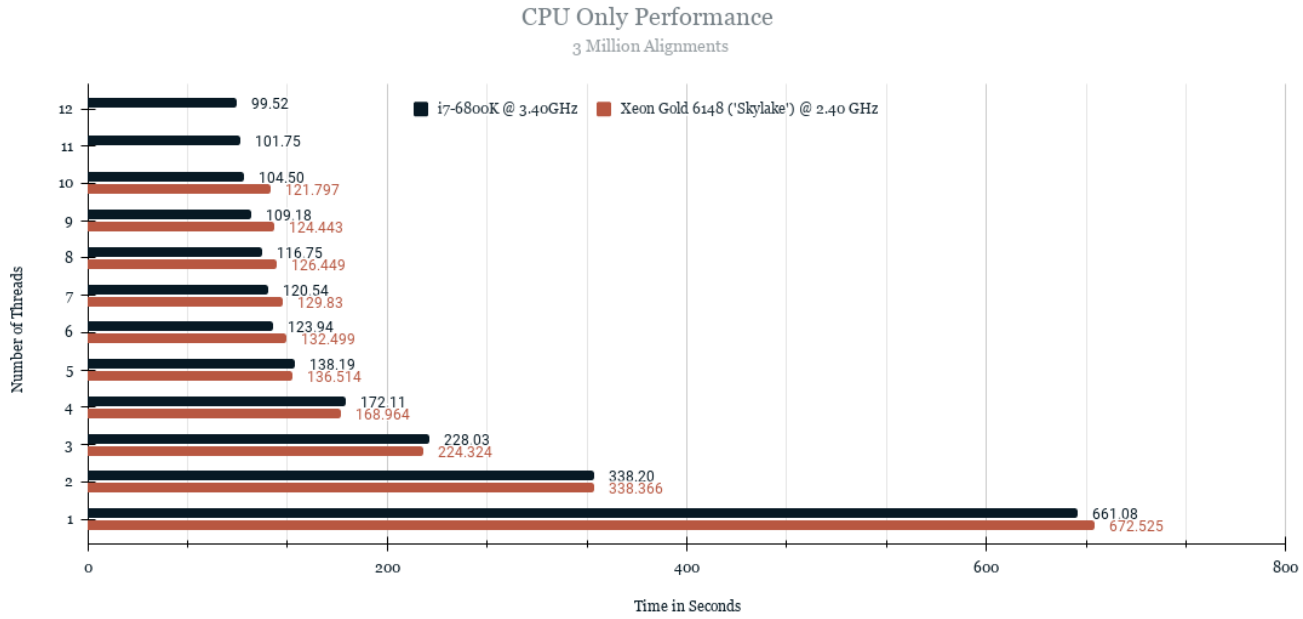

Figure 2: Here we show the performance of CPU-SSW. The performance of this SIMD implementation seems most closely tied to the number of physical cores. On these Intel processors that is half the number of threads for our given allocation, or eight (8) for the i7 and five (5) for the Xeon

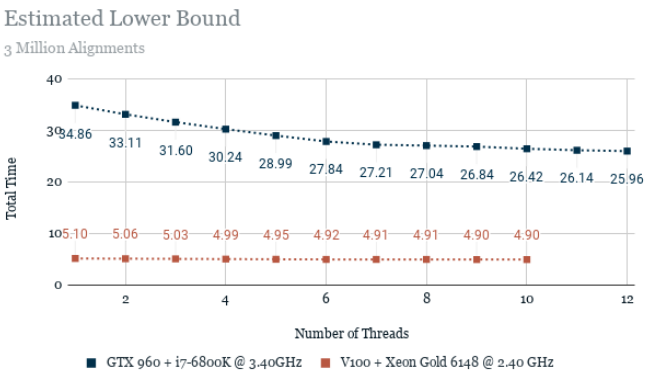

Figure 3: Combining the rate of alignments from both solutions we estimate a lower bound*

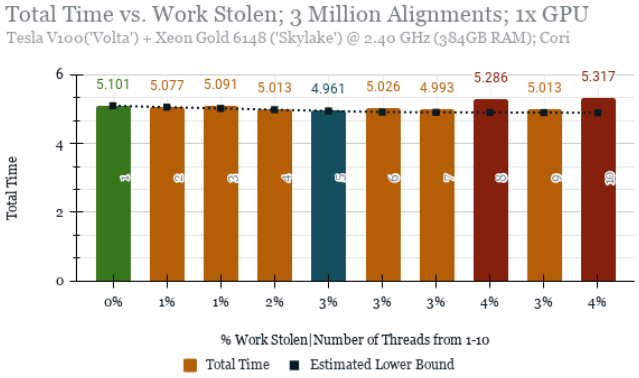

Figure 4: Actual performance approaches our estimated lower bound. The dark red bars actually performed worse than the GPU only solution, which is shown in green. The blue bar represents the quickest.

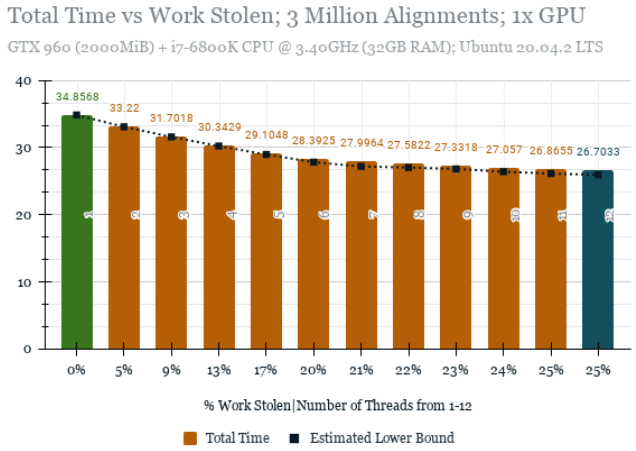

Figure 5: Comparing to the GTX 960 which is more exaggerated with a faster CPU and slower GPU.

5 Optimizations

5.1 Host Work Stealing

In our initial implementation, the thread hosting a GPU does not process alignments. Here we show a small clip of CUDA C that demonstrates how to do concurrent work on the CPU between kernel launches.

```
cudaEvent_t event;
cudaEventCreate(&event)
some_kernel_call<<<BLKS,THRDS,0,streams[0]>>>();
cudaEventRecord(event,streams[0]);
while(cudaEventQuery(event) == cudaErrorNotRead){
    do_work_on_host();
}
```

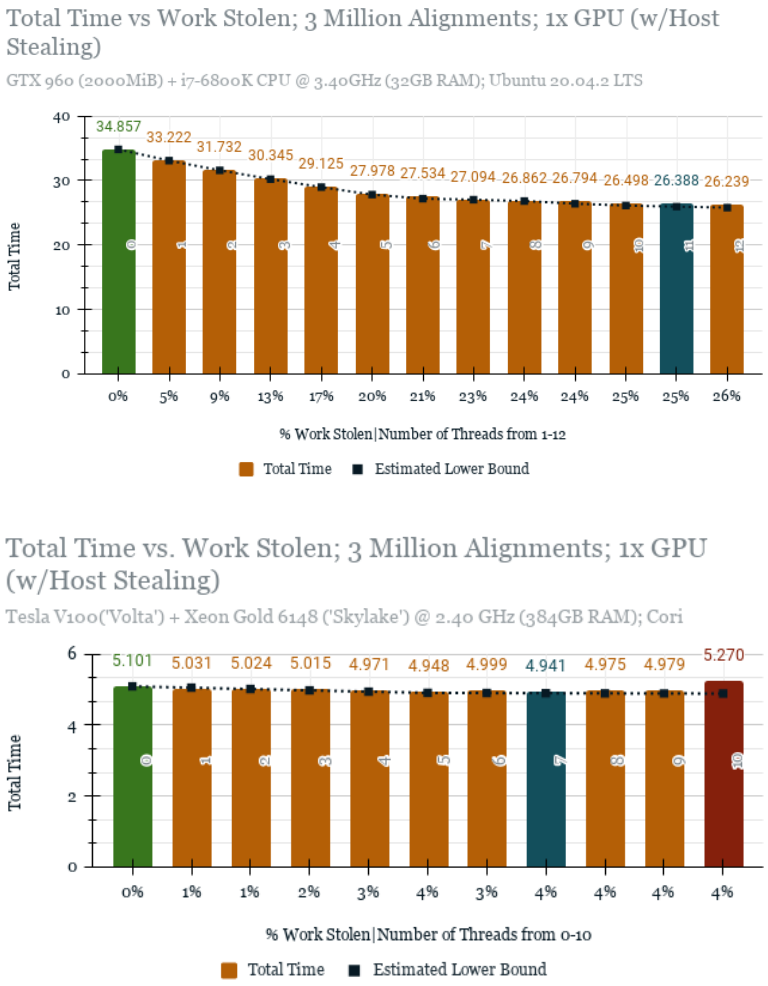

Figure 6: The results were that the added thread actually reduced our total execution time by the rate of an added thread!

5.2 Manual Load Balancing

We attempted to mitigate the difference in work between the GPU and CPU by testing if blocking the "last batch" to either the GPUs or the CPUs would either save processing from an additional GPU launch, or perhaps would not allow the slower CPUs to touch the final bit of work that the GPU could process much quicker. Both attempts yielded increased execution time while testing with our 3 Million alignment data set. It should be noted that our work does not try to attempt to optimize either solution in particular but to demonstrate a work-stealing scenario is feasible.

6 Additional Results

6.1 More CPUs? More GPUs?

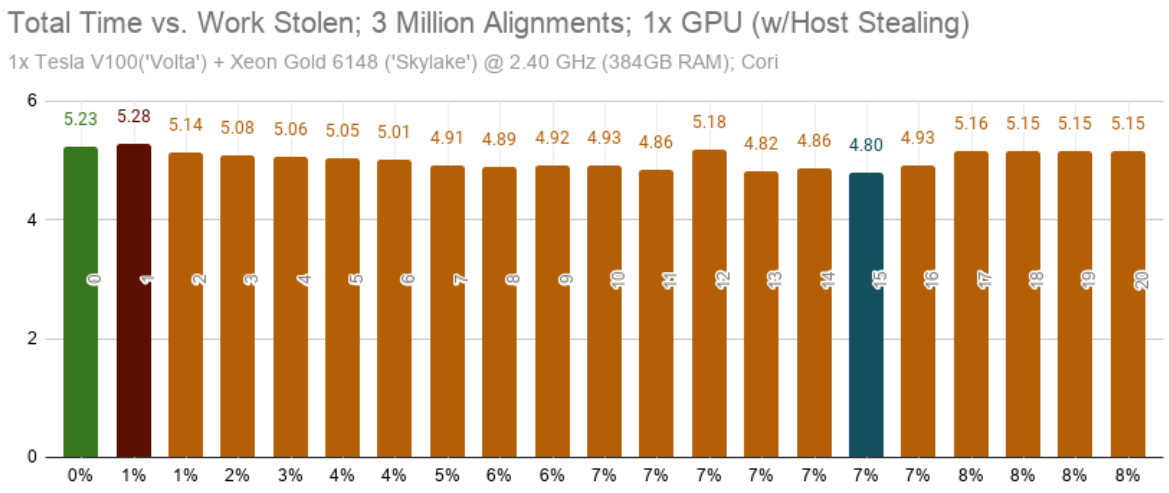

Figure 7: Allocating more CPUs definitely scales, and would be worthwhile if you only have one GPU.

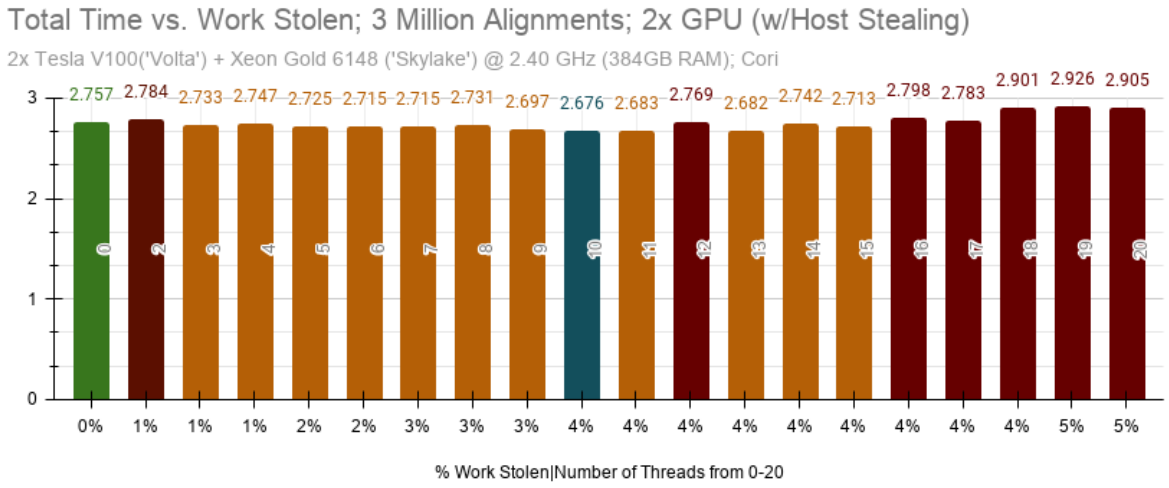

Figure 8: Adding another GPU just seems to scale linearly, see Figure 4 to compare. Again, the CPUs add what they can, but barely make a dent in the total execution time, if not worse.

6.2 Small and Large Data

Where the number of threads were chosen empirically as fastest.

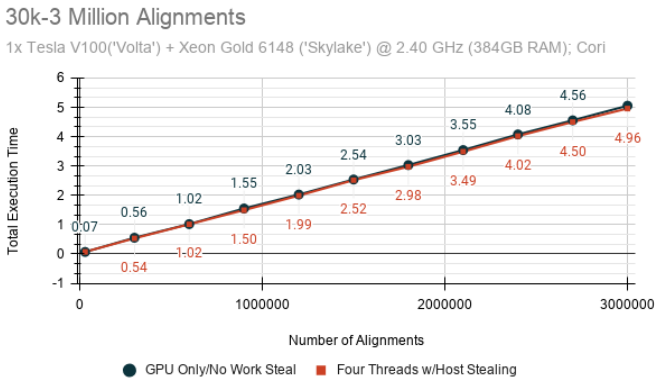

Figure 9: CPU work stealing did not add much performance with the GPU being so much quicker.

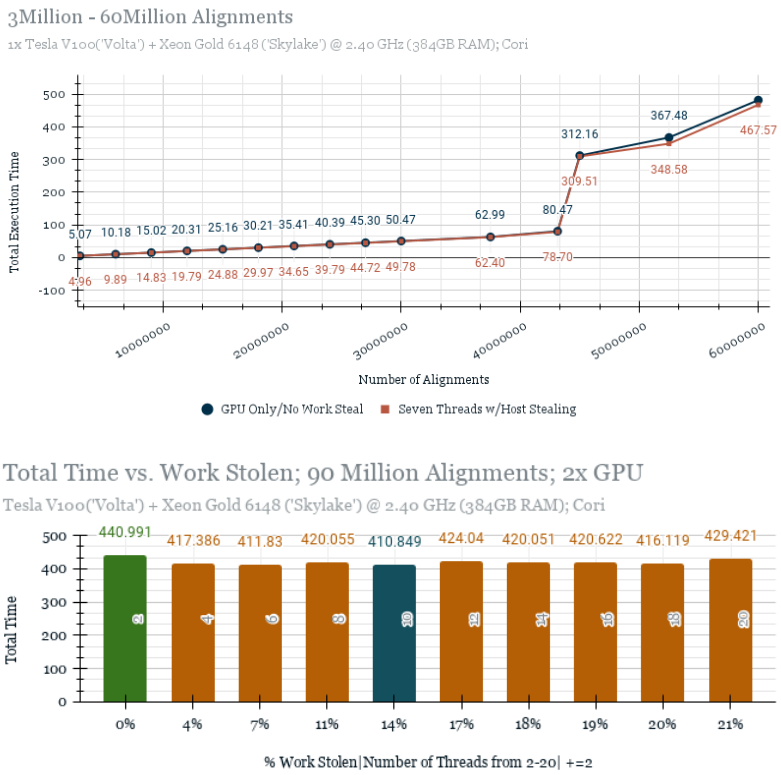

Figure 10: There is a cliff at 45M Alignments per GPU. The CPUs seem to use the opportunity to steal more work.

6.3 Real World Data

The real datasets were benchmarked using code that used host CPU stealing. These datasets were at a larger scale than the toy reference set used for most other testing - the query and reference sequences used in each of the 3 external datsets were all about twice as long as that of the toy reference set.

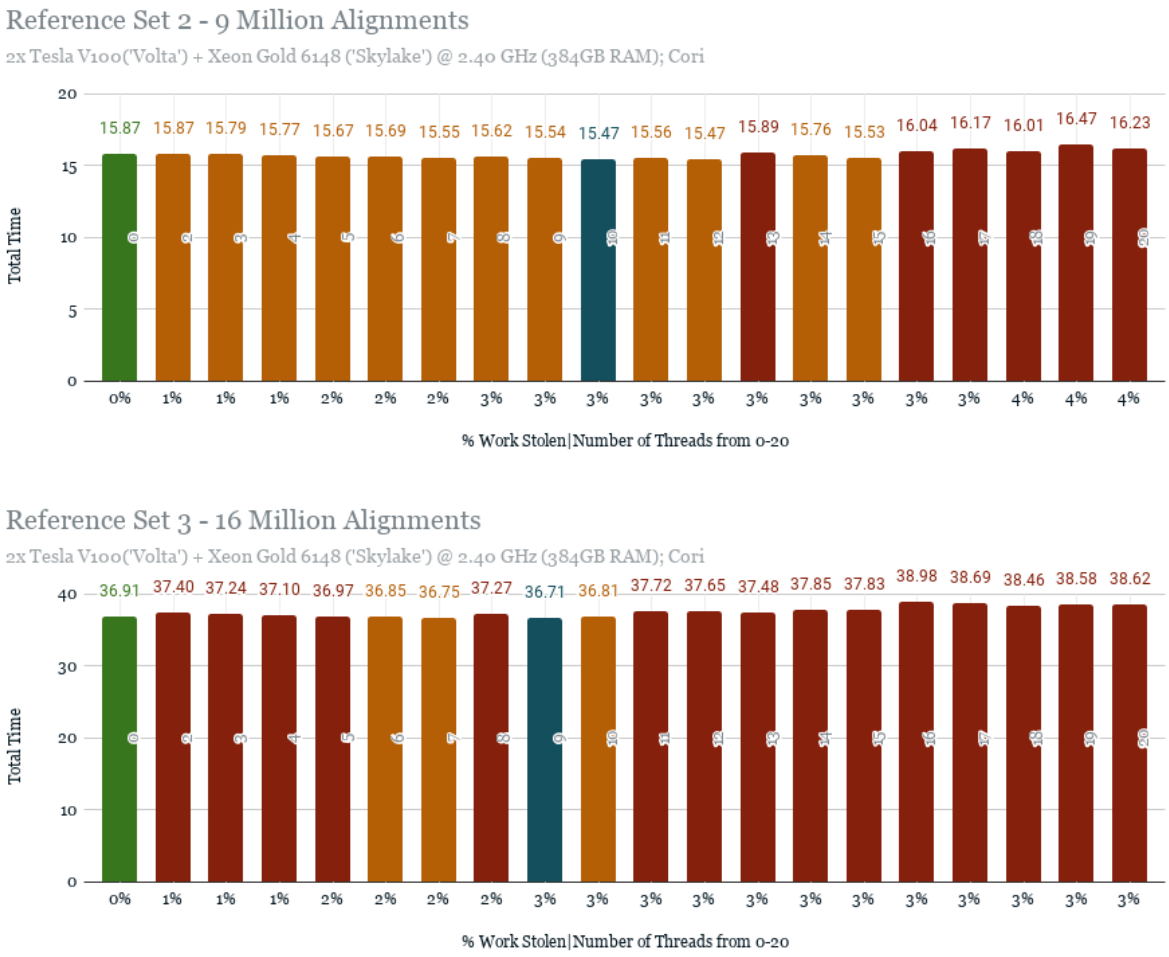

Figure 11: Only a few experiments were better than the GPU only solution.

7 Future Work

For possible future work, a survey using different CPU Smith-Waterman SIMD libraries, such as SWIMM 2.0[6], since our chosen implementation SSW[2] only utilizes SSE2 and our chip supports AVX-512[9] is of high interest. It would be nice to see if larger vector widths would improve performance and help close the GPU gap. Using CPUs such as the KNL processor we have been targeting all semester would be very interesting as it has so many physical cores. Observing that even very old GPUs provide enormous performance boosts in relation to CPUs, a survey of different combinations of CPU-GPU, especially with recycle-bin hardware, would be very timely and may help those with less resource still have access to high throughput computing, during these times where demand outweighs the supply of GPUs. A survey against cloud-computing would be interesting as well, we did poke at it a bit and offerings with GPU Cloud solutions with "vCPUs" seem to be especially problematic towards our experiments.

7.1 Short discussion on Heterogeneous Code

Supporting heterogeneous architectures is a complex problem. Ideally, one version of code can compile to multiple targets, creating a maintainable solution that would require no new work for adding an additional CPU, device or

accelerator. Much work has already been done towards this and one example would be OpenMP 4.5[8]. Unfortunately for us, most solutions for Smith-Waterman are actually quite specialized towards specific architectures[3], which is much the case for our example which utilizes SSE2 and CUDA. Additionally, different implementations of Smith-Waterman may be further customized towards specific sequence types[3]. The codes we are implementing are actually geared towards short sequences. We believe parallelism across devices may lend itself to hand-tuning and require a great deal of work by an actual programmer especially when dealing with high performance solutions.

8 Concluding Remarks

The larger takeaway for us is that in a heterogeneous computing environment, while the GPUs are active, the CPUs are available for and are capable of concurrent work at full capacity.

However, if the discrepancy between the GPU and CPU performance is too great, a worthwhile boost is unlikely. Furthermore, as GPUs become faster and CPUs remain the same, this work-stealing scheme may be ineffective. The code is not free and adds some amount of overhead, complexity, and energy use.

We have shown that on older GPUs there is much to be gained from work-stealing. Our recommendation is to estimate ones gains before implementing.

We also only touch on processing throughput on a single problem. In the scope of a real application, different concurrent work will be available for parallelism.

9 Acknowledgments

The Cori supercomputer, Giulia Guidi for guiding us and keeping our focus in scope, Muaaz Awan for ongoing support and direction, and Professor Yelick for feedback on utilizing our lower bound.

10 Code Availability

`https://github.com/CS267-ADEPT-WORKSTEALING/`

References

[1] Awan, M.G., Deslippe, J., Buluc, A. et al. *ADEPT: a domain independent sequence alignment strategy for gpu architectures. BMC Bioinformatics 21, 406 (2020).* https://doi.org/10.1186/s12859-020-03720-1

[2] Zhao M, Lee W-P, Garrison EP, Marth GT (2013) *SSW Library: An SIMD Smith-Waterman C/C++ Library for Use in Genomic Applications. PLoS ONE 8(12): e82138.* `https://doi.org/10.1371/journal.pone.0082138`

[3] Barnes, Richard *A Review of the Smith-Waterman GPU landscape* `https://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-152.pdf`

[4] *CUDA Runtime API :: CUDA Toolkit Documentation* `https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__EVENT.html`

[5] N. Ahmed, J. Lévy, S. Ren, H. Mushtaq, K. Bertels and Z. Al-ars *GASAL2: a GPU accelerated sequence alignment library for high-throughput NGS data* `MC Bioinformatics 20, 520 (2019) doi: 10.1186/s12859-019-3086-9.`

[6] Rucci, E., Garcia Sanchez, C., Botella Juan, G. et al. *SWIMM 2.0: Enhanced Smith–Waterman on Intel's Multicore and Manycore Architectures Based on AVX-512 Vector Extensions. Int J Parallel Prog 47, 296–316 (2019).* `https://doi.org/10.1007/s10766-018-0585-7`

[7] Mattson, T. G., He, Y., Koniges, A. E. (2019). *The OpenMP Common Core. Amsterdam University Press.*

[8] Pas, R., Stotzer, E., Terboven, C., van der Pas, R. (2017). *Using OpenMP—The Next Step. Amsterdam University Press.*

[9] *Xeon Gold 6148 - Intel - WikiChip* `https://en.wikichip.org/wiki/intel/xeon_gold/6148#Frequencies`