Parallelism and Replicability: Intel® Xeon Phis and Protein Database Querying with the Smith-Waterman Algorithm

Jake Caswell, Brianna Cunniff, Jonathan Lenz, and Phil Nadolny
Department of Mathematics, Statistics, and Computer Science
St. Olaf College
1500 St. Olaf Avenue, Northfield MN, 55057
caswell,cunniff,lenz,nadolnyp@stolaf.edu

Abstract

Prompted by growing interest in study replication by the National Science Foundation, this report aims to produce an experiential replication study that tests the Intel Xeon Phi Many Integrated Core (MIC) platform's performance in a realistic research setting. To demonstrate the MICs' performance, this report replicates the theoretical peak performance of 1 TFLOPS as reported by F. Masci of Caltech. These performance results are then compared to a MIC-enabled Smith-Waterman protein database querying implementation, the SWAPHI (Smith-Waterman Algorithm on Xeon Phi coprocessors) software developed by Y. Liu et al. The results from the SWAPHI paper were not as easily reproducible over the course of the proposed one-month period for a comparable replication research study. The difference between the experience and success of these two replication attempts highlights desired best-practices for researchers who aim to address a perceived "culture of irreproducibility" in computer science.

Introduction

Computational biologists use protein database querying (PDQ) to study the function, structure, and evolutionary relatedness of proteins in both medical and general biology research. However, the computational cost of calculating pairwise alignments over increasingly large protein databases is prohibitive. As such, researchers often resort to algorithms that introduce a heuristic search to find query results more quickly than pairwise alignments (for instance, using BLAST [Camacho, 2009]). But unlike pairwise alignments, heuristics are not guaranteed to always return globally optimal sequence alignments.

To mitigate this performance cost while still finding optimal results, researchers have been adapting the standard Smith-Waterman pairwise alignment algorithm for use on emerging parallel and distributed platforms. One such platform is the Intel Xeon Phi Many Integrated Core (MIC) architecture. Intel first designed the MIC as an accelerator, similar to a GPU, capable of providing additional computational power to a host platform. However, unlike GPUs, which are specifically designed for linear algebra computations, the MICs contain 61 or more generalized x86 compatible cores, depending on the model, each capable of executing four hardware threads.

Since the MIC's performance has been investigated to some degree, this paper first aims to replicate the results from a Caltech technical report that obtained approximately 1 TFLOPS performance per MIC using optimized matrix multiplication as a benchmark [Masci, 2013]. The results from this replication will serve two principal purposes: to verify the MIC's performance claims and to determine under which parallel execution models the maximum performance may be accomplished.

Then, using the results from the performance replication, this paper will attempt to replicate Liu and Schmidt's findings in their MIC optimized Smith-Waterman pairwise alignment implementation [Liu and Schmidt, 2014]. In doing so, this paper aims to understand how researchers may expect the MIC's performance to translate to a typical research use case. This paper will also compare the MIC implementation of Smith-Waterman to a GPU implementation to evaluate the relative performance of different accelerator platforms [Liu and Schmidt, 2014; Liu et al, 2013].

Ultimately, this paper presents an experiential study into the process of computer science research replication, a topic which has seen renewed interest by both the National Science Foundation (NSF) and journal editorial staff alike. The principal concern centers around a lack of sufficient information necessary to facilitate research replication in many studies published to computational journals, to the point that the study's authors themselves cannot replicate their own results months later [Heroux, 2015]. Moreover, replication researchers have found that it is difficult to gain access to code cited in a published study, let alone run it and obtain accurate results [Collberg et. al., 2015]. As the review and replication of findings is foundational to the scientific process, this culture of irreproducibility in computer science research must be addressed.

This paper is the culmination of a month long project aimed at simulating the initial stages of a research replication study, in which an independent team attempts to acquire, compile, and run code from an existing study as quickly as possible. The two related studies highlighted by this project present two drastically different replication experiences. From these experiences, this paper aims to highlight essential best practices necessary for a successful replicable study.

Prior Work

Protein Database Querying and Parallelism

In 2009, Camacho et al presented an improvement over the ubiquitous BLAST which can handle much longer query or database sequences while still relying on the same heuristics. However, other projects have designed parallel pairwise alignment based on Smith Waterman. One such instance is SWIPE, which utilized within-core parallelism and between-core parallelism to achieve speedup [Rognes 2011]. GPU parallelism has also been explored in CUDASW++, currently in version 3.0, achieving speedup by coupling CPU and GPU SIMD instructions [Liu et al, 2013]. This same research group next implemented SWAPHI, which employed MICs to accelerate Smith-Waterman protein database querying [Liu et al, 2014].

Two later papers also utilized MICs to accelerate Smith-Waterman protein database querying. LSDBS employed both the multi-core CPU and many-core MIC hardware to parallelize Smith-Waterman PDQ [Lan et. al., 2015]. LSDBS-mpi incorporated support for a cluster of MICs [Lan et. al., 2016]. Since SWIPE in 2011 and other early parallel SW-PDQ programs, each successive implementation has incorporated new computational approaches and/or greater acceleration and performance. However, no prior replication studies of any of these implementations have appeared.

Intel MIC Performance Benchmarking

According to a technical document produced by the Infrared Processing and Analysis Center at Caltech, each MIC is capable of approximately 1 TFLOPS performance when computing multiplication of two matrices [F. Masci, 2013]. Masci's report also demonstrates the necessity of vector optimization, seeing speedup over the naive implementation performance of 120 GFLOPs. Saule et al. likewise note the theoretical peak performance of 1 TFLOPS in their study highlighting performance comparisons between the MIC architecture and cutting-edge CPUs and GPUs when running sparse linear algebra kernels [E. Saule, 2014]. Teodoro et al. compare CPUs, GPUs and MIC systems when specifically analyzing low-dimensional spatial datasets captured by microscopy apparatuses [G. Teodoro, 2014]. Their results demonstrate the impact of regular versus irregular data access on an accelerated system, noting that GPUs outperform CPUs when accessing data randomly, whereas MICs outperform GPUs when using a first-come-first-served data access strategy.

There are, however, few to no replication studies of these performance testing, which presents a potential gap in the literature. Additionally, many studies rely almost exclusively on testing the performance of OpenMP over the MIC hardware, rather than exploring additional libraries and execution modes. These categories could lead to fruitful future work.

Research Replication in CS

The *ACM Transactions on Mathematical Software* (TOMS) has initiated a Replicated Computational Results (RCR) review process, in which a reviewer must replicate the computational results of the submitted manuscript [Heroux, n.d.]. TOMS RCR occurs in addition to the standard review process at the author's request, and it involves an independent confirmation that the results in a manuscript are correct and replicable. An RCR reviewer will be assigned to replicate the results, and s/he will work with the authors through the RCR process to either independently replicate the results or to review a corpus of computational result artifacts. If the RCR reviewer determines that the results were successfully replicated then the paper will receive a special RCR designation, and an RCR review report will be published in conjunction with the original paper [Heroux, 2015]. Still others suggest that to promote replicability authors should share data, software, workflow, and details of execution. In addition, they should publish sufficient information such that someone in the field could use the shared digital scholarly objects without having to resort to contacting the original authors. Moreover, citation of software should be standard practice to give credit to the original authors [Stodden et. al., 2016]. The consensus view claims that the lack of reproducibility of computational results in the culture of CS research stems from the need for greater awareness of replication as a viable research genre. Common recommendations center on improving incentives for the original authors to make their software replication-friendly. For instance, funding agencies can require data management plans, software productivity, and sustainability plans. In addition, publishers can raise expectations for independent reproducibility of computational results.

Hardware Platform Comparison

As we intend this study to serve as an example of an independent replication study, we here provide information regarding the hardware systems that we used. We also provide similar information for the hardware systems used by the SWAPHI group for direct comparison. The hardware platform from the MIC benchmarking study will not be provided, as their RAM, Xeon E5 product number, and additional hardware specifications were not present in the document. In terms of CPU, our hardware platform and the SWAPHI group's platform were identical, utilizing one Xeon E5-2670 per MIC. However, the two MIC configurations differed slightly. The preproduction MIC hardware on loan to St. Olaf College from Intel had a higher clock speed and more cores than that used by the SWAPHI group. The GPU configuration also differed between the two setups.

	Intel Xeon	**Intel Xeon**	**Intel Xeon Phi (MIC)**	**Intel Xeon Phi (MIC)**	**Nvidia GeForce**	**Nvidia GeForce**
Product number	E5-2670	E5-2670	SE10/7120	B1PRQ-5110 P/5120D	GTX 980 TI	GTX Titan (Titan)
Cores	8	8	61	60	2816	2688
Threads	2	2	4	4	N/A	N/A
Frequency	2.6 GHz	2.6 GHz	1.238 GHz	1.05 GHz	1 GHz	875.5 MHz
L1i Cache	32 KB	32 KB	32 KB	32 KB	N/A	N/A
L1d Cache	32 KB	32 KB	32 KB	32 KB	N/A	N/A
L2 Cache	256 KB	256 KB	30.5 MB	512 KB	3 MB	N/A
L3 Cache	20480 KB	20480 KB	N/A	N/A	N/A	N/A

Table 1: More detailed specifications on the processing units in our setup (white) and that of the SWAPHI group (blue).

Software Environment

To take advantage of the MICs' capabilities, we are using the Intel Manycore Platform Software Stack (MPSS) version 3.3.5. In addition, we are using the Intel Parallel Studio XE 2017 Cluster Edition Update 1, which includes ICC version 17.0.1 and OMP version 4.8.5-11. When running the SWAPHI software, OpenMP using offload mode and guided scheduling as well as a set of Intel C++ compiler functions were used. On the GPU-equipped machine, we are using NVCC version 8.0.44. Similar specifications of the software environment were not present in the original studies. Please contact the authors for a software manifest containing a full list of libraries and utilities.

Data and Results

Benchmarking the MIC

The original report implemented two variations of matrix multiplication: one referred to as the naive approach (test1.c), and the other using sgemm() matrix multiplication from Intel's Math Kernel Library (MKL), which is highly optimized for the MICs (test2.c). To get a sense of the relative performance between the host and MIC hardware, test1.c and test2.c were compiled for both host-only and MIC-only execution. Both implementations facilitate on-node data parallelism using OpenMP. This report will not provide either the source code for the benchmarks or the commands to compile and run them as both are available as an appendix in the original technical report.

Replicating the MIC benchmarks

This report could not locate any measure of the granularity of the original data, as well as the exact values of performance in GFLOPS. In the technical report, only the performance graphs were provided (Figure 1). Thus, our report may only compare the general attributes of the performance graphs.

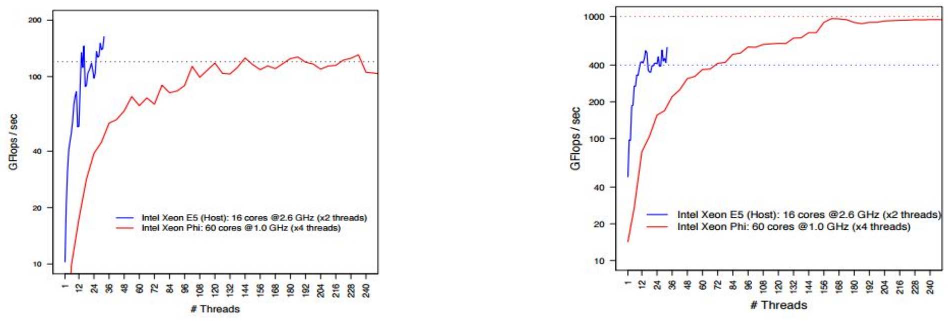

Figure 1. MIC benchmarking performance plots provided by the original report. *Left:* performance as a function of thread count for naive matrix multiplication code. The blue curve represents host only execution up to the maximum number of threads for the host CPU, 32. The red curve represents the performance of native-mic execution up to maximum number of threads, 240. The dotted line approaches a limit of ~120 GFLOPS. *Right:* performance as a function of thread count for optimized matrix multiplication. The blue curve represents host only execution, and the red represents native-mic execution. The lower dotted line represents ~400 GFLOPS, the higher dotted line represents ~1000 GFLOPS.

For naive matrix multiplication, the host consistently outperforms the MIC hardware up until approximately 100 threads, at which point the two perform equally. This trend is not shared in the second graph, where the host-only consistently outperforms the MICs up until 32 threads. Using the Intel optimized matrix multiplication, the host's performance increases towards a limit of 400 GFLOPS. Unlike the host system, the native-mic execution curve outperforms the host-only execution around 72 threads, and continues toward a limit of approximately 1000 GFLOPS.

In an attempt to replicate the curves from the original report, this report executed test1.c and test2.c in host-only execution and mic-native execution for threads proceeding in steps of 2 from n=1 to n=36, and proceeding in steps of size 12 from n=36 to n=240, close to the maximum number of threads present on the MICs.

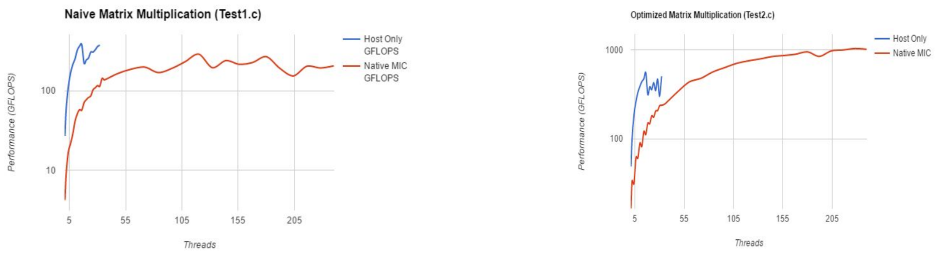

Figure 2. MIC benchmarking performance plots provided by our report. *Left:* performance as a function of thread count for naive matrix multiplication code, test1.c. The blue curve represents host only execution up to the maximum number of threads for the host CPUs, 32. The red curve represents the performance of native-mic execution up to maximum number of threads, 240. Both the host and MIC approach a limit of ~220 GFLOPS. *Right:* performance as a function of thread count for optimized matrix multiplication. The blue curve represents host only execution, and the red represents native-mic execution. The host limit approaches ~450 GFLOPS, whereas the mic limit approaches ~1000 GFLOPS.

According to the data gathered, the general trends persist (Figure 2). We see that, for test1.c, host-only execution again outperforms the native-mic execution for all threads from n=1 to n=32. The performance of the native-mic execution tends toward 220 GFLOPS, rather than 120 GFLOPS, and the host only execution maxes around 220 GFLOPS as well. In the second graph, we see a closer resemblance to the original study. The host only-execution tends toward 400-500 GFLOPS, whereas the native-mic execution tends toward a limit of 1000 GFLOPS.

The discrepancies in performance are to be expected, as the host platforms cannot be directly compared. Unlike our host platform, consisting of 2 Intel Xeon E5-2670 8 core CPUs each with 2 hyperthreads, leading to a total of 32 threads, Masci's host platform consisted of a single unspecified Xeon E5 16 core CPU with 2 hyperthreads, resulting in 32 total threads. Additionally, the original report did not specify further hardware specifications, such as system memory, or software specifications (for instance, the compiler suite or OpenMP versions). Without proper knowledge about these attributes, it is difficult to directly account for how much our results should realistically diverge.

SWAPHI Description

The SWAPHI software utilizes both the MIC's many cores as well as the 512-bit wide SIMD vectors within each processor core [Liu et. al., 2014]. The algorithm performs multiple alignments in individual SIMD vectors, with each vector lane computing one alignment. It also performs intra-sequence alignment, which computes the alignment of a single sequence pair in the SIMD vectors across minor diagonals in the matrix or the query sequence by means of either sequential or striped layout. Both the inter- and intra-sequence models have the same workflow:

1. Construct a query profile for the query if applicable.
2. Perform alignments by creating as many host threads as the number of MICs used (one host thread corresponds to one MIC). The host thread offloads operations to its MIC, and it loads database sequences onto the MIC chunk-by-chunk at runtime
3. Wait for the completion of all host threads.
4. Sort all alignment scores in descending order and output the alignment results.

The SWAPHI group evaluated three variants of their algorithm: Inter-sequence model with a score profile (InterSP), Inter-sequence model with a query profile (InterQP), and Intra-sequence model with a query profile (IntraQP). See the original paper for further descriptions of and differences between the three variants. The SWAPHI group used twenty query protein sequences (ranging from 144 to 5,478 amino acids) from the 2013_08 release of the TrEMBL protein database. For each variant, they analysed the GCUPS and runtime for each query length on one, two, and four MICs. In addition, they compared SWAPHI's performance to that of SWIPE, BLAST+, and CUDASW++3.0.

Replicating the SWAPHI Results

After successfully compiling the source code from the 2014 paper, we ran performance tests in an effort to replicate the results found in the paper. Above all, we sought to compare the GCUPS (billion cell updates per second) and runtime numbers on our MICs. We first downloaded the same 20 proteins used in the original SWAPHI paper, with lengths ranging from 144 to 5,478 amino acids, along with the full TrEMBL database. These proteins can be found in the TrEMBL/Swiss-Prot database with the following accession IDs: P02232, P05013, P14942, P07327, P01008, P03455, P42357, P21177, Q38941, P27895, P07756, P04775, P19096, P28167, P0C6B8, P20930, P08519, Q7TMA5, P33450. Then, we utilized the indexing feature of the SWAPHI program to build an index for the TrEMBL database with which we would compare our protein queries. Next, we used the align feature of the program to align each query protein sequence with the top ten database alignments, taking the average of three trials for each of the InterSP, InterQP, and IntraQP variants. Finally, we recorded the maximum and average results for each of these variants among all protein queries. We focused our analysis on the two MIC setup, which had typical runtimes between 1 and 5 seconds. In our tests, the program would not finish when we used only one MIC, and as of yet we are unable to get the cluster of four MICs to work together. We ran tests on both of the MIC systems.

The SWAPHI group noted that CUDASW++3.0 does not support databases as big as TrEMBL, so they utilized the UniProtKB/Swiss-Prot database (release 2013_08). In addition, they utilized the GPU-only version of CUDASW++3.0, which does not support subject sequences of lengths>3072. Thus, they created a reduced Swiss-Prot database by extracting all subject sequences of lengths <=3072 and only worked with query sequences that were also shorter than length 3072 (i.e. queries 1-15). In order to perform a faithful replication, we utilized the full TrEMBL database on one MIC to compare the SWAPHI variants (Machine 1, M1). And we utilized a reduced Swiss-Prot database on the other MIC to compare the performance between SWAPHI and CUDASW++3.0 (Machine 2, M2). Moreover, like the SWAPHI group we also only used query sequences of length 3072 or less to compare SWAPHI and CUDASW++3.0. Similar to our SWAPHI variant analysis, we took the average GCUPS and runtime of three CUDA trials for each query sequence and then computed the average and maximum GCUPS and runtime for our overall results. Like the SWAPHI group, we used the InterSP model on our two MICs to compare the performance of SWAPHI with that of CUDASW++3.0.

Comparison of SWAPHI and CUDASW++3.0 Results

The SWAPHI group evaluated three variants of their algorithm (InterSP, InterQP, and IntraQP). They measured the runtime and GCUPS for each query with each variant on a single MIC and on four MICs. They also reported the average and maximum GCUPS and runtime for each algorithm-variant/number-of-MICs combination. As of this time, our group was only able to evaluate the three SWAPHI variants on two MICs. However, the SWAPHI group did not report the GCUPS or runtime associated with different query lengths on two MICs, so a direct comparison of our results to theirs is not possible at this time. Figure 3A depicts the GCUPS of each of the three variants on two MICs, and figure 3B depicts the runtime of each of the three variants on two MICs. The average and maximum GCUPS and runtime for each variant on two MICs are summarized in Tables 2 and 3, respectively. For the original SWAPHI results that we were not able to replicate, we direct you to their original paper.

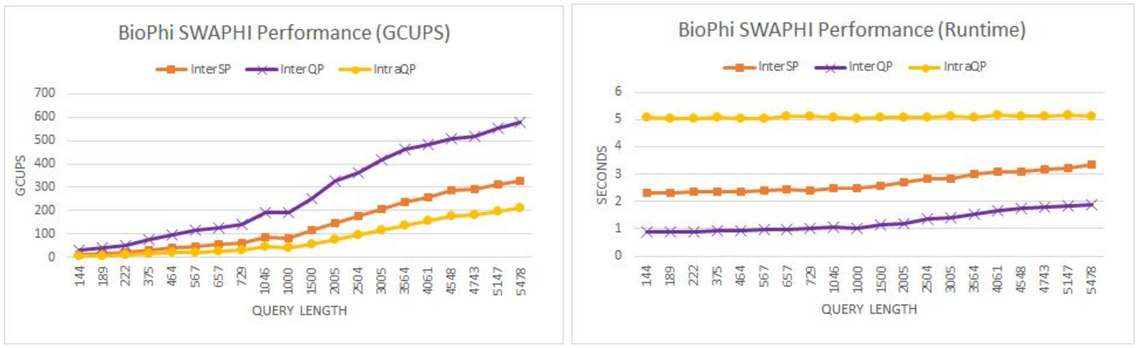

Figure 3. The GCUPS (*3A, left*) and runtime (*3B, right*) of the three SWAPHI variants on two coprocessors. MIC benchmarking performance plots provided by our report.

	InterSP		InterQP		IntraQP		
	M1	**M2**	**M1**	**M2**	**M1**	**M2**	**CUDA**
Average GCUPS	140.0	85.78	276.7	183.72	81.5	46.27	32.6
Max GCUPS	325.1	227.98	576.6	441.99	212.3	136.36	33.0

Table 2: A summary of the results (GCUPS) of running SWAPHI on two MICs and on the local CUDA machine.

	InterSP		InterQP		IntraQP		
	M1	**M2**	**M1**	**M2**	**M1**	**M2**	**CUDA**
Average Runtime	2.69	2.68	1.266	1.30	5.08	4.97	38.72
Max Runtime	3.24	3.31	1.88	1.98	5.14	5.06	101.73

Table 3: A summary of the results (runtime) of running SWAPHI on two MICs and on the local CUDA machine.

The SWAPHI group compared the performance of the SWAPHI software to that of SWIPE, BLAST+, and CUDA++3.0. As of the time of this writing, we have only been able to run the CUDA++3.0 software on our local machine and so will focus our analysis on the SWAPHI and CUDA++3.0 comparison. The SWAPHI group recorded maximum performances of 53.2 GCUPS for one MIC, 90.8 GCUPS for two MICs, and 124.6 GCUPS for four MICs. As for their CUDA implementation, they recorded an average of 108.9 GCUPS and a maximum of 115.4 GCUPS. With two MICs, we recorded a maximum of 227.98 GCUPs and an average of 85.78 GCUPS. As for our CUDA implementation, we recorded an average of 32.67 GCUPS and a maximum of 33.04 GCUPS. In addition, we recorded an average runtime of 22.02 seconds and a maximum runtime of 65.79 seconds. Figure 4 depicts the performance comparison of SWAPHI and CUDASW++3.0 as implemented by the SWAPHI group. Figure 5 depicts the performance comparison of SWAPHI and CUDASW++3.0 as implemented by us.

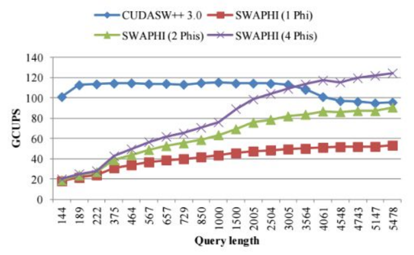

Figure 4: SWAPHI and CUDASW++3.0 comparison as performed by the SWAPHI group.

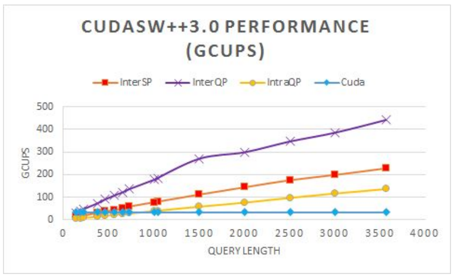

Figure 5: SWAPHI and CUDA++3.0 comparison as performed by our group.

Conclusion

MIC Benchmark Experience

The results of our MIC benchmark tests provided two clear results. First, we demonstrated the same significant discrepancy between optimized and unoptimized execution on the MICs, seeing almost five-fold speedup. Second, we replicated an experimentally-posited performance of 1 TFLOPS per MIC using native-MIC execution. Beyond confirming these key aspects of the original study, our benchmarks also supported general trends seen in the original data. For instance, when using unoptimized, naive matrix multiplication, MIC performance did not exceed that of the host as the number of threads increased towards the hardware limit.

The greatest challenge in replicating the original benchmarking report was the lack of a complete hardware and software manifest of the system. The code was readily available, as were the instructions on how to repeat the compilation and build process. However, without any knowledge of the original target hardware and software environments, we cannot account for any discrepancies in performance. There are two reasons for this. First, the initial report provided no table of the data recorded. Second, the report did not include a comprehensive hardware and software manifest, making any attempt to account for differing results impossible. A more thorough performance analysis could have been conducted had these manifests been provided. Nonetheless, because of the original author's extensive documentation, replicating the general trends in the original report was straightforward, and required little effort and no communication.

SWAPHI Experience

One of the challenging aspects of this comparison was that while the SWAPHI team provided results for their one and four MIC setups, we were only able to attain results from the two MIC setup. Thus, in our comparison, we had to estimate what performance they would have obtained with two MICs based on the other data they provided. A more complete analysis would include data for all three of the MIC setups. Furthermore, due to complications in fully understanding the code, we were only able to produce similar levels of performance. We were not able to fully understand some of the results of our tests, and whether this is due to gaps in our knowledge of the program or some kind of bug or oversight in the code, we cannot know without some form of collaboration with the authors. This collaboration would be particularly necessary because when using available SWAPHI software, our protein database querying did not return the database sequences most well-aligned to our query sequences.

The most challenging part of the replication was getting the desired code from the authors and compiling the code the first time. Making any serious changes to the code was greatly hampered by a lack of documentation. Beginning by seeking papers with judiciously commented code would ease the process of replication for any future studies. In addition, a more fruitful replication would involve actual collaboration with the original authors of a given paper. The authors of the SWAPHI paper did not respond to our emails requesting collaboration, but given the short length of our research timeframe and how late into the process we found the paper, this is understandable. With more time, we would have sought a paper from authors readily willing to collaborate. This would have both expedited our process and led to a greater understanding of the source code, and therefore a more meaningful analysis. In addition, the authors themselves would be rewarded by strengthening the results of their paper thanks to the replication process.

Conclusions

The most significant contribution to replicability is extensive documentation. Reports must provide (either as an appendix or as a link to a maintained location) any and all code developed; data, hardware and software manifests; build and run instructions; and a valid channel for communication with the authors. Doing so, as was evident in our two attempts at replication, provides an easy and relatively painless route to replication of results. The benefits of replication cannot be understated, not only for the authors of a replication study, but for the authors of an original study. If, for instance, a bug is discovered in the algorithms or software packages provided by the original authors, then the replication researches will be able to quickly and painlessly troubleshoot and communicate with the original team. The overall result will be a more robust and accurate software package, which will contribute even further to the computer science community.

Future Work

For a more rigorous set of benchmarks, it would be interesting to run direct comparisons of multiple parallel libraries to see which performed the best. We were able to replicate general performance trends with OpenMP using the method outlined in F. Masci’s 2013 paper, and it would be worthwhile to rewrite this code using other libraries such as TBB, OpenCL, Qthreads, or MPI to better understand how different parallel and distributed execution models perform on the MIC hardware. Further, applying these different parallel execution models to the SWAPHI code could yield significant performance speedup. Taking a step back from the libraries themselves, similar comparisons using different execution modes could prove quite informative for future work. In addition, future researchers could apply the Intel Advisor utilities to see whether they find more opportunities for taking advantage of the user's hardware configuration.

Unfortunately, we were not able to completely replicate the SWAPHI authors' results. This was partially due to our hardware configuration. We had access to four MICs distributed across two different computers whereas the SWAPHI authors had four MICs contained in a single computer. We were, therefore, unable to run tests using four MICs simultaneously. Researchers with this capacity could participate in the replication process in this way. Reworking the SWAPHI code to utilize the MPI+X framework so that it could run on a setup like ours would be an interesting extension of this work as well. Again, initiating contact with the original authors would enhance any future work with this project.

References

C. Camacho, G. Coulouris, V. Avagyan, N. Ma, J. Papadopoulos, K. Bealer, and T.L. Madden, "BLAST+: architecture and applications," BMC Bioinformatics, vol. 10, pp. 421, 2009.

F. Masci, "Benchmarking the Intel® Xeon PhiTM Coprocessor," Infrared Processing and Analysis Center, Caltech (2013).

G.Teodoro, T. Kurc, J. Kong, L. Cooper and J. Saltz, "Comparative Performance Analysis of Intel (R) Xeon Phi (TM), GPU, and CPU: A Case Study from Microscopy Image Analysis," *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, Phoenix, AZ, 2014, pp. 1063-1072. doi: 10.1109/IPDPS.2014.111

H. Lan, Y. Chan, K. Xu, B. Schmidt, Shaoliang Peng, and Weiguo Liu. "Parallel algorithms for large-scale biological sequence alignment on Xeon-Phi based clusters." *BMC Bioinformatics* 17.9 (2016): 267.

Y. Liu, A. Wirawan, and B. Schmidt. "CUDASW++3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions." *BMC Bioinformatics.* 14.1 (2013): 1.

Y. Liu, and B. Schmidt. "SWAPHI: Smith-Waterman protein database search on Xeon Phi coprocessors." In *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*, pp. 184-185. IEEE, 2014.

E. Saule, K. Kaya, and Ü.V. Çatalyürek. (2014) Performance Evaluation of Sparse Matrix Multiplication Kernels on Intel Xeon Phi. In: Wyrzykowski R., Dongarra J., Karczewski K., Waśniewski J. (eds) Parallel Processing and Applied Mathematics. PPAM 2013. Lecture Notes in Computer Science, vol 8384. Springer, Berlin, Heidelberg

T. Rognes, “Faster Smith-Waterman database searches with intersequence SIMD parallelization,” BMC Bioinformatics, vol. 12, pp. 221, 2011.