Inductive-bias-driven Reinforcement Learning for Efficient Schedules in Heterogeneous Clusters (Supplementary Material)

Subho S. Banerjee Saurabh Jha Zbigniew T. Kalbarczyk Ravishankar K. Iyer

A. Extended Motivational Example

Current schedulers prioritize the use of simple online heuristics (Grandl et al., 2016b) and coarse-grained resource bucketing (e.g., core counts, free memory) and require user labeling of commonly used system resources (Hindman et al., 2011; Grandl et al., 2016a) to make scheduling decisions. Those approaches are untenable in truly heterogeneous settings as (i) defining such heuristics is difficult over the combinatorial space of application-processor/accelerator configurations; and (ii) user-based resource usage labeling requires in-depth understanding of the underlying system. This paper demonstrates the use of ML to automatically infer such heuristics and their evolution over time as new user workloads and/or new accelerators are added.

A.1. Dealing with Architectural Heterogeneity

We reiterate that state-of-the-art schedulers do not model the emergent heterogeneous compute platforms that are being widely adopted in data centers and hence leave a lot to be desired (as can also be seen in the performance of our baselines). Consider, for example, the execution of the *forward algorithm on PairHMM models* (Banerjee et al., 2017), a computation that is commonly performed in computational genomics workloads. Fig. 1 shows the significant diversity (nearly $100\times$) in performance of this single workload across CPUs (from Intel and IBM), GPUs (two models of GPUs from NVIDIA) and FPGA implementations. The increasing

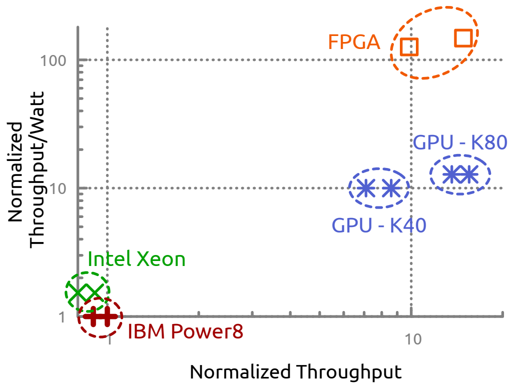

Figure 1. Architectural diversity leading to varied performance for the PairHMM kernel.

heterogeneity necessitates rethinking of the design and implementation of future schedulers, as the current approach will require an extraordinary amount of manual tuning and expertise to adapt to the emergent systems. In contrast, the proposed technique eliminates that work and automates the whole process of learning the right granularity of resources and scheduling workloads in cloud-based, dynamic, multi-tenant environments, thereby improving application performance and system utilization, all with minimal human supervision. Prior work uses microarchitectural throughput metrics such as clock cycles per instruction (Giceva et al., 2014; Delimitrou & Kozyrakis, 2013; 2014; Mars et al., 2011; Mars & Tang, 2013) as proxies for processor affinities. In our case, such metrics are not usable because of the wide diversity in processors, i.e., CPU-centric units cannot describe the performance of GPUs/FPGAs.

A.2. Dealing with Resource Granularity

Traditional schedulers use coarse-grained resource bucketing, i.e., they schedule macro-resources like CPU core counts and GBs of memory. That simplifies the design of the scheduling algorithms (both the optimization algorithms and attached heuristics), resulting in an inability to measure low-level sources of resource contention in the system. The contention of such low-level resources is often the cause for performance degradation and variability. Consider, for example, the concurrent execution of several compute kernels (described in Appendix C.2) on co-located hyper-threads (i.e., threads that share resources on a single core) on an Intel CPU. If we abstract the problem at the level of CPU threads and memory allocated, then those kernels should execute in isolation. The normalized runtime variation is illustrated in Fig. 2. We observe a slowdown of as much as 40% (i.e., the co-located runtime is 60% of the isolated runtime) for some combinations of kernels, and almost no slowdown for others. That problem is further exacerbated by the architectural diversity in processors that we described earlier. The proposed technique accounts for such contention by explicitly collecting information on low-level system state by using performance counter measurements, and by estimating resource usage in the system by explicitly encoding the measurements in its POMDP model.

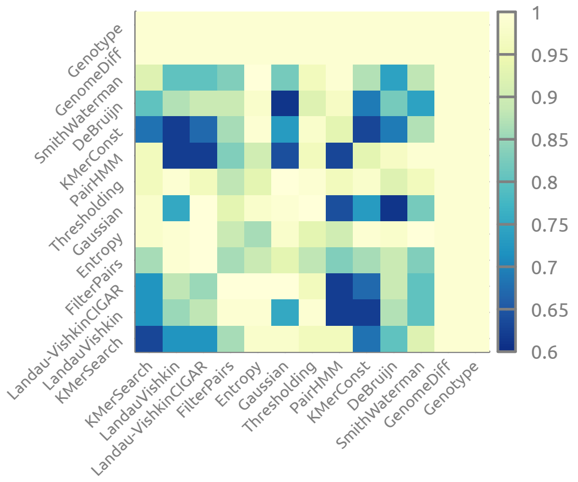

Figure 2. Degradation in runtime of co-located kernels due to shared resource contention.

B. Implementation Details

The scheduling framework functions as follows.

1. The scheduler first makes measurements by using the available processor performance counters (e.g., instructions retired, cache misses).
2. When a processor becomes idle (finishes running the current kernel), it invokes the scheduler.
3. The measurements are fed into the scheduler's BN model as input. Using those measurements, the BN model computes the utilization of different levels of architectural resources in the system (e.g., memory bandwidth utilization, PCIe link utilization). We refer to those utilizations as the *state* of the system.
4. The computed utilization numbers, user programs represented as a DFG, and a system topology graph are fed into an NN. The NN produces a scheduling decision that is actuated in the system. The action space consists of a kernel-processor pair.
5. Finally, the scheduler gets feedback from the system (i.e., the reward) in terms of the time it took for the job to run as a result of its scheduling decision.
6. While in *training mode*, if an incorrect decision is made, Symphony enqueues an update of the policy parameters using back-propagation on the A2C/A3C loss function. An incorrect decision is one where kernel input-output dependencies are not respected, or a kernel-accelerator pair is picked where the accelerator does not provide an implementation of the kernel.

B.1. Graph Network Details

The structure of the graph network used in the proposed model is illustrated in Fig. 3. The numbers of parameters used in the different layers of the graph network are listed in Table 1.

Table 1. Mapping of the graph network layer functions in Fig. 3. We use the notation $FCNN(a, b)$ to denote a 2-hidden fully-connected layers with a and b hidden units, respectively.

Function in GN	Function in 1	Function in 2
ϕ^e	$FCNN(64, 32)$	$FCNN(64, 32)$
ϕ^v	$FCNN(32, 16)$	–
ϕ^u	$FCNN(16, 16)$	$FCNN(32, 16)$
$\rho^{e\rightarrow v}$	$\sum e$	–
$\rho^{v\rightarrow u}$	$\sum v$	–
$\rho^{e\rightarrow u}$	–	$ReLU(e)$

Table 2. Hyperparameters used in the model.

Hyperparameter	**Value**
Learning Rate	0.005
LSTM Unroll Length	20
n_s	20
n_e	2

B.2. Hyperparameters

The hyperparameters used to train the proposed POMDP model are listed in Table 2.

B.3. System Measurement Details

Topology Information. Consider the example of standard NUMA based computing system with PCIe based accelerators shown in Fig. 4. The system contains (i) multiple CPUs which have non-uniform access to memory, (ii) several accelerators (including GPUs and FPGAs) each with their own memory, and (iii) a system interconnect which connects all of the components of the system together. Symphony encodes the system topology as a graph $T = (P, N)$ (also shown in Fig. 4). The nodes of the graph P correspond to the processing elements (and attached memory) and memory/system interconnects. Each of the these nodes $p \in P$ have an attached resource utilization vector. For example, in an Intel processor, the utilization vector would include utilization like that of micro-op issue ports, floating point unit utilization etc. (Doweck, 2016; Intel Corp., 2014).

The scheduler queries the system topology and builds the topology graph T (which is used as an input to the RL agent) using hwloc (Broquedis et al., 2010). hwloc provides information about CPU cores, caches, NUMA memory nodes, and the PCIe interconnect layout (i.e., connections between the PCIe root complex and PCIe switches), as well as connection information on peripheral accelerators, storage, and network devices in the system. The scheduler does not explicitly model the rack-scale or data center network (unlike some previous approaches, e.g., Isard et al. (2009); Chowdhury et al. (2014)), but the BN and RL model can be extended to do so. Our measurements considers injection bandwidth at the network interface card (NIC) to be a proxy for network performance, i.e., the NIC is modeled as an accelerator that accepts data at $\min$(PCIe Bandwidth, Injection Bandwidth).

Performance Counter Measurements. Performance counters' configuration and access instructions require kernel mode privileges, and hence those operations are supported by Linux: system calls to configure and read the performance counter data. Symphony uses a combination of user-space tools, e.g., libPAPI (Terpstra et al., 2010), PMUTools (Kleen, 2010), and perf that wrap around the system call interface to make both system-specific and system-independent measurements. We configure the performance counters to make system-wide measurements (i.e., for all processes). All kernel (i.e., computation) executions are non-preemptive in the context of the proposed runtime, however the OS scheduler can preempt CPU threads. Further we prevent the OS scheduler from re-balance tasks/threads once assigned to a particular CPU, so as to ensure re-balancing actions happen only through the runtime. This is achieved by explicitly setting affinities of threads to cores (i.e., pinning them).

Performance Penalties. Monitoring of performance counters without having to perform interrupts is almost free. In our implementation, we capture on-core performance counters directly before and after a single kernel invocation. Un-core performance counters are measured periodically (every million dynamic instructions on a core) by using a *performance monitoring interrupt*. On an IBM PowerPC processor, the interrupt handler initiates a DMA transfer of the performance counters to memory (Sudhakar & Srinivasan, 2019), thereby incurring no performance penalty (other than the time to service the interrupt). On Intel processors, the interrupt handler has to explicitly read the performance counter registers and write them to memory. In our tests (on Intel processors), we observed a ~3% performance penalty for applications with interrupts enabled. That corresponds to an execution of a usermode interrupt with an average 900-ns latency.

Distributed Execution. In our evaluation we have deployed Symphony in a rack-scale distributed context (over an EDR Infiniband network fabric) as a centralized scheduler controlling all processing resources. Here, all the performance counter measurements are sent over the network to a centralized server that makes scheduling decisions. This approach works well at the scale of a rack, where all resources are essentially one hop away at 0.2-µs latency. Extending Symphony to larger or slower networks might present challenges, where network latency causes stale performance counter data to reach the scheduler. We will address these challenges in future work.

B.4. Dynamically Reconfigurable FPGA Accelerator

Our implementation and evaluation of Symphony uses a custom FPGA accelerator (see Fig. 5). Due to space limitations, here we briefly describe the features of the accelerator.

- *Processing Elements (PEs).* The co-processor is optimized to execute the computational kernels as a single instruction of the application. Sets of four neighboring PEs are directly connected as a systolic element, thereby enabling high bandwidth data transfer in between PEs and forming the quantum of reconfiguration.
- *Host-FPGA Communication.* The board interfaces with the host CPU over PCIe and can be configured to communicate with the host processor over this interface in one of two ways: (i) using direct memory access (DMA) to the hosts memory over the PCIe bus, or (ii) using IBM's coherent accelerator processor interface (CAPI) (Stuecheli et al., 2015).
- *Dynamic Reconfiguration.* The configuration of the accelerator (i.e., which kernels PEs are available at any time) is controlled by Symphony. Symphony treats the reconfiguration of the accelerator as a kernel that has to be dispatched to the FPGA. The state of the accelerator is fed into Symphony along with the system topology T.
- *Launching Kernels.* Remember CPU executors (i.e., threads which are given tasks to execute) are pinned or bound to underlying hardware SMT thread. Accelerators however require the CPUs to initiate their execution. As a

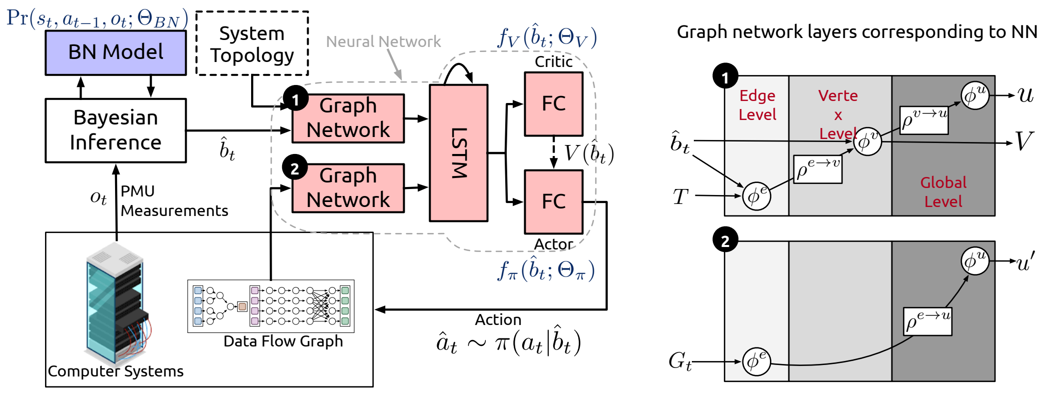

Figure 3. Proposed POMDP model.

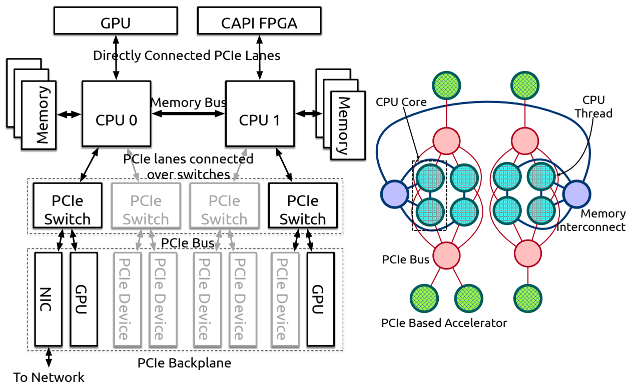

Figure 4. Example of a dual-socket NUMA-based system topology with a PCIe-interconnect and -devices. Figure on the right shows an graph-encoding of the topology.

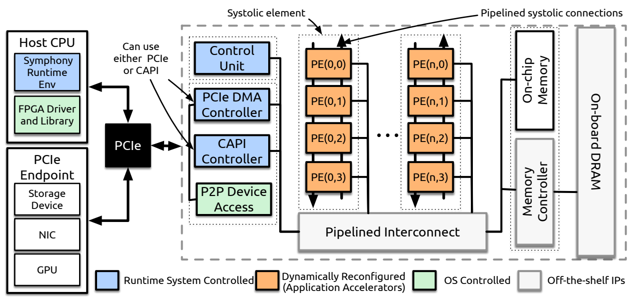

Figure 5. Architecture of the FPGA-based hardware co-processor controlled by Symphony.

result, each accelerator in the system is assigned a proxy executor thread that orchestrates (i.e., launches, polls for completion etc.) its execution. These executors are responsible for managing their own queues for maintaining tasks that are "waiting" for execution.

C. Evaluation Environment

C.1. Evaluation System

All evaluation experiments are performed on an 11 node rack-scale test-bed of IBM Power8 CPUs, NVIDIA K40 and K80 GPUs, as well as FPGAs (listed in Table 3). All the machines in the cluster are connected using a single switch EDR Infiniband network.

C.2. Evaluation Workloads

We illustrated the generality of the proposed approach on a variety of real-world workloads (listed in Table 4) that used CPUs, GPUs, and FPGAs:

1. *variant-calling and genotyping analysis* (Van der Auwera et al., 2013) on human genome datasets appropriate for clinical use (consisting of `Align`, `IR`, and `HC` in Table 4),
2. *epilepsy detection and localization* (Varatharajah et al., 2017) on intra-cranial electroencephalography data; and
3. online *security analytics* (Cao et al., 2015) on network- and host-level intrusion detection system event-streams.

Table 3. Hardware specifications of test cluster.

Name	#	Specifications
`M1`	2	CPU IBM Power8 (SMT 8); 870 GB RAM; GPU NVIDIA K80; FPGA Alpha Data 7V3
`M2`	4	CPU IBM Power8 (SMT 4); 512 GB RAM; GPU NVIDIA K40; FPGA Nallatech 385
`N`	1	Mellanox FDR Infiniband

For the variant-calling and genotyping workload we use the NA12878 genome sample from the GIAB consortium (Zook et al., 2016) for all our experiments as it is representative of human clinical datasets. For the EEG and AT workloads, we use the same datasets as discussed in the original papers.

References

Banerjee, S. S., Athreya, A. P., Mainzer, L. S., Jongeneel, C. V., Hwu, W.-M., Kalbarczyk, Z. T., and Iyer, R. K. Efficient and scalable workflows for genomic analyses. In *Proceedings of the ACM International Workshop on Data-Intensive Distributed Computing*, DIDC '16, pp. 27–36, 2016.

Banerjee, S. S., El-Hadedy, M., Tan, C. Y., Kalbarczyk, Z. T., Lumetta, S. S., and Iyer, R. K. On accelerating pair-HMM computations in programmable hardware. In *Proc. 27th International Conference on Field Programmable Logic and Applications, FPL 2017, Ghent, Belgium, September 4-8, 2017*, pp. 1–8, 2017.

Banerjee, S. S., El-Hadedy, M., Lim, J. B., Kalbarczyk, Z. T., Chen, D., Lumetta, S. S., and Iyer, R. K. ASAP: Accelerated Short-Read Alignment on Programmable Hardware. *IEEE Transactions on Computers*, 68(3):331–346, March 2019a.

Banerjee, S. S., Kalbarczyk, Z. T., and Iyer, R. K. AcMC^2 : Accelerating Markov Chain Monte Carlo Algorithms for Probabilistic Models. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 515–528, 2019b.

Broquedis, F., Clet-Ortega, J., Moreaud, S., Furmento, N., Goglin, B., Mercier, G., Thibault, S., and Namyst, R. hwloc: A Generic Framework for Managing Hardware Affinities in HPC Applications. In *Proc. 2010 18th Eu-*

Table 4. Enumeration of workloads used to evaluation.

Application	Processors			Implementations
	CPU	GPU	FPGA	
Alignment *(Align)*	✓	✓	✓	(Li & Durbin, 2009; 2010; Langmead et al., 2009; Zaharia et al., 2011; Banerjee et al., 2019a; 2016),
Indel Realignment *(IR)*	✓	✗	✗	(McKenna et al., 2010; Nothaft et al., 2015)
Variant Calling *(HC)*	✓	✓	✓	(Li et al., 2009; McKenna et al., 2010; Nothaft, 2015; Rimmer et al., 2014; Banerjee et al., 2017)
EEG-Graph *(EEG)*	✓	✓	✓	(Varatharajah et al., 2017; Banerjee et al., 2019b)
AttackTagger *(AT)*	✓	✓	✓	(Cao et al., 2015; Banerjee et al., 2019b)

romicro Conference on Parallel, Distributed and Network-based Processing, pp. 180–186, Feb 2010.

Cao, P., Badger, E., Kalbarczyk, Z., Iyer, R., and Slagell, A. Preemptive intrusion detection: Theoretical framework and real-world measurements. In *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security*, HotSoS '15, pp. 5:1–5:12, 2015.

Chowdhury, M., Zhong, Y., and Stoica, I. Efficient coflow scheduling with varys. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pp. 443–454, 2014.

Delimitrou, C. and Kozyrakis, C. Paragon: QoS-aware Scheduling for Heterogeneous Datacenters. *SIGPLAN Not.*, 48(4):77–88, March 2013.

Delimitrou, C. and Kozyrakis, C. Quasar: Resource-efficient and QoS-aware Cluster Management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pp. 127–144, 2014.

Doweck, J. Inside 6th generation Intel Core code named Skylake:: New Microarchitecture and Power Management. https://www.hotchips.org/wp-content/uploads/hc_archives/hc28/HC28.23-Tuesday-Epub/HC28.23.90-High-Perform-Epub/HC28.23.911-Skylake-Doweck-Intel_SK3-r13b.pdf, 2016. Accessed 2019-03-05.

Giceva, J., Alonso, G., Roscoe, T., and Harris, T. Deployment of query plans on multicores. *Proc. VLDB Endow.*, 8(3):233–244, November 2014.

Grandl, R., Chowdhury, M., Akella, A., and Ananthanarayanan, G. Altruistic Scheduling in Multi-resource Clusters. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, pp. 65–80. USENIX Association, 2016a.

Grandl, R., Kandula, S., Rao, S., Akella, A., and Kulkarni, J. Graphene: Packing and Dependency-aware Scheduling for Data-parallel Clusters. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, pp. 81–97, 2016b.

Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R., Shenker, S., and Stoica, I. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pp. 295–308. USENIX Association, 2011.

Intel Corp. Intel 64 and ia-32 architectures optimization reference manual. *Intel Corporation, Sept*, 2014.

Isard, M., Prabhakaran, V., Currey, J., Wieder, U., Talwar, K., and Goldberg, A. Quincy: Fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, pp. 261–276, 2009.

Kleen, A. PMU-Tools. https://github.com/andikleen/pmu-tools, 2010. Accessed 2019-03-05.

Langmead, B., Trapnell, C., Pop, M., and Salzberg, S. L. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10(3): R25, 2009.

Li, H. and Durbin, R. Fast and accurate short-read alignment with burrows-wheeler rransform. *Bioinformatics*, 25(14):1754–1760, may 2009. doi: 10.1093/bioinformatics/btp324. URL http://dx.doi.org/10.1093/bioinformatics/btp324.

Li, H. and Durbin, R. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics*, 26(5): 589–595, 2010.

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., et al. The sequence alignment/map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 2009.

Mars, J. and Tang, L. Whare-map: Heterogeneity in "homogeneous" warehouse-scale computers. *SIGARCH Comput. Archit. News*, 41(3):619–630, June 2013.

Mars, J., Tang, L., and Hundt, R. Heterogeneity in “homogeneous” warehouse-scale computers: A performance opportunity. *IEEE Comput. Archit. Lett.*, 10(2):29–32, July 2011. ISSN 1556-6056.

McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., Daly, M., and DePristo, M. A. The genome analysis toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research*, 20(9):1297–1303, jul 2010. doi: 10.1101/gr.107524.110.

Nothaft, F. Scalable genome resequencing with adam and avocado. Master’s thesis, EECS Department, University of California, Berkeley, May 2015.

Nothaft, F., Massie, M., Danford, T., Zhang, Z., Laserson, U., Yeksigian, C., Kottalam, J., Ahuja, A., Hammerbacher, J., Linderman, M., Franklin, M. J., Joseph, A. D., and Patterson, D. A. Rethinking data-intensive science using scalable analytics systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’15, pp. 631–646, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-2758-9. doi: 10.1145/2723372.2742787.

Rimmer, A., Phan, H., Mathieson, I., Iqbal, Z., Twigg, S. R. F., Wilkie, A. O. M., McVean, G., and Lunter, G. Integrating mapping-, assembly- and haplotype-based approaches for calling variants in clinical sequencing applications. *Nature Genetics*, 46(8):912–918, jul 2014.

Stuecheli, J., Blaner, B., Johns, C. R., and Siegel, M. S. CAPI: A Coherent Accelerator Processor Interface. *IBM Journal of Research and Development*, 59(1):7:1–7:7, Jan 2015. ISSN 0018-8646. doi: 10.1147/JRD.2014.2380198.

Sudhakar, A. T. and Srinivasan, M. IBM POWER in-memory collection counters. https://developer.ibm.com/articles/power9-in-memory-collection-counters/, 2019. Accessed 2019-03-05.

Terpstra, D., Jagode, H., You, H., and Dongarra, J. Collecting Performance Data with PAPI-C. In Müller, M. S., Resch, M. M., Schulz, A., and Nagel, W. E. (eds.), *Tools for High Performance Computing 2009*, pp. 157–173, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

Van der Auwera, G. A., Carneiro, M. O., Hartl, C., Poplin, R., del Angel, G., Levy-Moonshine, A., Jordan, T., Shakir, K., Roazen, D., Thibault, J., Banks, E., Garimella, K. V., Altshuler, D., Gabriel, S., and DePristo, M. A. From fastq data to high-confidence variant calls: The genome analysis toolkit best practices pipeline. *Current Protocols in Bioinformatics*, 43(1):11.10.1–11.10.33, 2013.

Varatharajah, Y., Chong, M. J., Saboo, K., Berry, B., Brinkmann, B., Worrell, G., and Iyer, R. EEG-GRAPH: A Factor-Graph-Based Model for Capturing Spatial, Temporal, and Observational Relationships in Electroencephalograms. In *Advances in Neural Information Processing Systems*, pp. 5377–5386, 2017.

Zaharia, M., Bolosky, W. J., Curtis, K., Fox, A., Patterson, D., Shenker, S., Stoica, I., Karp, R. M., and Sittler, T. Faster and more accurate sequence alignment with SNAP. *arXiv preprint arXiv:1111.5572*, 2011.

Zook, J. M., Catoe, D., McDaniel, J., Vang, L., Spies, N., Sidow, A., Weng, Z., Liu, Y., Mason, C. E., Alexander, N., Henaff, E., McIntyre, A. B., Chandramohan, D., Chen, F., Jaeger, E., Moshrefi, A., Pham, K., Stedman, W., Liang, T., Saghbini, M., Dzakula, Z., Hastie, A., Cao, H., Deikus, G., Schadt, E., Sebra, R., Bashir, A., Truty, R. M., Chang, C. C., Gulbahce, N., Zhao, K., Ghosh, S., Hyland, F., Fu, Y., Chaisson, M., Xiao, C., Trow, J., Sherry, S. T., Zaranek, A. W., Ball, M., Bobe, J., Estep, P., Church, G. M., Marks, P., Kyriazopoulou-Panagiotopoulou, S., Zheng, G. X., Schnall-Levin, M., Ordonez, H. S., Mudivarti, P. A., Giorda, K., Sheng, Y., Rypdal, K. B., and Salit, M. Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Scientific Data*, 3:160025, Jun 2016.