Accelerating Simulation-based Inference with Emerging AI Hardware

***Abstract*—Developing models of natural phenomena by capturing their underlying complex interactions is a core tenet of various scientific disciplines. These models are useful as simulators and can help in understanding the natural processes being studied. One key challenge in this pursuit has been to enable statistical inference over these models, which would allow these simulation-based models to learn from real-world observations. Recent efforts, such as Approximate Bayesian Computation (ABC), show promise in performing a new kind of inference to leverage these models. While the scope of applicability of these inference algorithms is limited by the capabilities of contemporary computational hardware, they show potential of being greatly parallelized. In this work, we explore hardware accelerated simulation-based inference over probabilistic models, by combining massively parallelized ABC inference algorithm with the cutting-edge AI chip solutions that are uniquely suited for this purpose. As a proof-of-concept, we demonstrate inference over a probabilistic epidemiology model used to predict the spread of COVID-19. Two hardware acceleration platforms are compared - the Tesla V100 GPU and the Graphcore Mark1 IPU. Our results show that while both of these platforms outperform multi-core CPUs, the Mk1 IPUs are 7.5x faster than the Tesla V100 GPUs for this workload.**

***Index Terms*—Approximate Bayesian Computation, COVID-19, epidemiology, hardware acceleration, high performance computing, likelihood-free inference, simulation-based inference.**

I. Introduction

Models that simulate the natural processes being studied are central to progress in a multitude of scientific domains. The domains span a vast scale of natural phenomena - from particle physics and molecular biology to epidemiology and cosmology [1]. These simulator models encode our current understanding of the processes involved, and allow us to generate data from them. While key for scientific progress, these simulator models are not conducive to statistical inference, which could enable them to learn and modify themselves to better fit real-world experimental data. This incompatibility stems from the intractability of computing the likelihood function for these models - which is a function that provides the probability density of an observation w.r.t the model. To compute this function for these simulator models, one needs to perform an integral over all possible execution trajectories, which is often impossible.

This limitation has led to development of new inference methods, that can leverage the simulation capabilities of the models while also not requiring computation of the likelihood function. Such methods belong to a new class of statistical inference, often known as simulation-based inference or likelihood-free inference. These new methods have been successfully applied to some real-world models, but their use in large-scale applications is still limited due to the computational requirements. One such simulation-based inference method considered in this work is Approximate Bayesian Computation (ABC) [2].

In this work, we develop a parallelized ABC inference implementation for a typical scientific simulator model - a stochastic epidemiology model used to understand and predict the spread of COVID-19 [3]. The parallelization of ABC inference makes it amenable to acceleration with the massively parallel hardware platforms that are specialized in vector computations. Two such hardware acceleration platforms are considered - a Tesla V100 GPU and an emerging AI acceleration solution, the Graphcore Intelligence Processing Unit (IPU). We explore the limits of parallelism provided by the hardware platforms and provide insight over the steps involved in efficient parallelization and hardware acceleration of scientific simulator models. In our analyses, we found that while both GPU and IPU outperform high-core count Xeon Gold CPUs, the IPU based solution performed 7.5x faster than GPU. This suggests that the IPU's architecture may be uniquely suited for this workload. The speedup and efficiency of the parallelized ABC inference algorithm coupled with hardware acceleration could be translated to a wide variety of scientific simulator models, hence providing a new impetus to this exciting field. The paper is organized as follows: Section II provides background and related works in epidemiology, ABC, and a brief overview of the new IPU technology. Since this is a multidisciplinary work, some important terminology will also be introduced. Section III-A shall discuss the process of parallelizing the ABC algorithm for inference for the epidemiology model. Section III-B shall provide insights on the steps involved in supporting the parallelized ABC inference on CPU, GPU, and IPU. Section III shall talk about the experimental methodology used to evaluate the three platforms for their performance on the epidemiology model. In Section IV results are presented and discussed. Section V concludes the paper.

II. Background and Motivation

Since this work spans multiple disciples from epidemiology to Bayesian statistics and computer architecture, it is important to establish the terminology being used. A scientific simulator model is represented as a joint probability distribution $p(\theta, x)$ where θ denotes the set of parameters that we are interested in inferring, and x denotes the observed variables of the model, for which there typically is real-world experimental data to compare to. The prior knowledge about the parameters is encoded in their prior, denoted by $\pi = p(\theta)$. The process of generating simulated data D_s from the model is denoted as $D_s \sim p(x|\theta)$. The updated parameters θ by conditioning on observed data D, known as the posterior, is denoted by $p(\theta|x = D)$, or concisely as $p(\theta|D)$. The likelihood function is denoted by $p(x = D|\theta)$, or concisely, $p(D|\theta)$. Based on this terminology, the epidemiology model and the ABC inference process are discussed next.

A. Stochastic Epidemiology Model

The Covid-19 model [3] used as an example for simulation-based inference, contains 8 parameters:

$$\theta = [\alpha_0, \alpha, n, \beta, \gamma, \delta, \nu, \kappa] \tag{1}$$

Their priors are set to Uniform distributions:

$$\pi = p(\theta) = U(0, [1, 100, 2, 1, 1, 1, 1, 2]) \tag{2}$$

These prior values were taken as-is from the original model description [3]. They signify the reasonable ranges in which the parameters of interest could lie. The model simulation provides the following state vector

$$X = [S, I, A, R, D, R^u], \tag{3}$$

which consists of the sub-populations of **S**usceptible people, undocumented **I**nfected, **A**ctive confirmed cases, confirmed **R**ecoveries, confirmed cases **D**ying, and **U**nconfirmed recoveries/deaths. The model is summarized in Fig.1.

One of the key challenges of this model is that X is *partially* observed; i.e. only the A, R, D values are available from observed data. This makes the likelihood function $P(D|\theta)$ intractable for this model, as the unobserved sub-populations of the model are required to be 'integrated-out'. Instead, simulation-based inference such as ABC is used to perform inference over this model.

The underlying COVID-19 time-series data, provided by Johns Hopkins University [4], contains daily numbers for $[A, R, D]$.

In its first step, the model initializes the remaining variables with $R^u = 0$, $I_0 = \kappa * A_0$, and $S = P - (A + R + D + I)$ with P being the total population count at the first time point.

The second step is to calculate the hazard function h which provides the average update in the model parameters within one day

$$h(S, I, A, R, D, R^u) = \left(gS\frac{I}{P}, \gamma I, \beta A, \delta A, \beta\nu I\right). \tag{4}$$

with $g = \alpha_0 + \frac{\alpha}{1+(A+R+D)^n}$. Depending on the intention of the statistical analysis, g can look more complicated.

The third step is to randomly sample according to these average numbers. Instead of a Poisson sampling with h as parameter, we chose an approximation with normal distributions with mean h and variance $\sqrt{h}$ and use the floor of the numbers.

The fourth step is to apply the simulated numbers to obtain simulated numbers for the next day ($S \to I$, $I \to A$, $A \to R$, $A \to D$, $I \to R^u$, ordering according to h function).

The second to fourth step are repeated in a loop for each day. Eventually, the numbers for A, R, and D can be compared to the real measurements.

Real-world use-cases of epidemiology models: The epidemiology models being used serve two main purposes – providing predictions and understand underlying process to better handle the spread. The former use-case requires a typically simpler model which can be re-trained daily to provide updated predictions[1], while the latter requires more complex models with a nuanced parameterization to for detailed analysis, but these would be updated less frequently[2]. It is possible that providing avenues of acceleration of complex models, such as this work aims to, would allow for more complex models to be updated daily hence providing better analyses and predictions.

B. Approximate Bayesian Computation (ABC)

If we were just interested in the best configuration for θ (see Eq.(1)) that matches the simulation, we could run any optimization algorithm. Instead, we want to quantify the uncertainty in θ for statistical analysis and arguments. The standard Bayesian statistical inference approach of calculating

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} \tag{5}$$

is not applicable. As discussed earlier, the likelihood function $p(D|\theta)$ is intractable, since S_t, I_t, and R^u_t are unknown. Hence, we take the ABC approach, where the ability to simulate from the model is utilized to perform inference on it. First, we sample the parameters θ from their prior $\theta^* \sim \pi$. Next, we simulate the model to generate observations $D_s \sim p(x|\theta^*)$. The simulated observations are then compared to the real-world evidence using a distance function $dist(D_s, D)$. For this model we used the Euclidean distance [3]. Finally, the sampled parameters θ^* are accepted if the distance function is less than a certain tolerance value ϵ, $dist(D_s, D) \leq \epsilon$. This is repeated until we accept the number of samples required. In essence this ABC process is sampling parameters from the approximate posterior of the model given the data while effectively circumventing the likelihood function. From an information-theoretic perspective, as tolerance ϵ approaches 0, the approximate posterior converges to the true posterior [2].

To summarize, in ABC we aim to obtain samples from an approximation to the posterior:

$$p(\theta|D) \approx p(\theta|\text{dist}(D, D_s) \leq \epsilon) \propto \mathbb{P}(\text{dist}(D, D_s) \leq \epsilon)p(\theta) \tag{6}$$

where D is the ground truth data, D_s is simulated data depending on θ, and $p(\theta)$ is the prior [3]. The dist function is the Euclidean distance [3].

[1]https://covid19-projections.com/

[2]https://covid19.healthdata.org/united-states-of-america

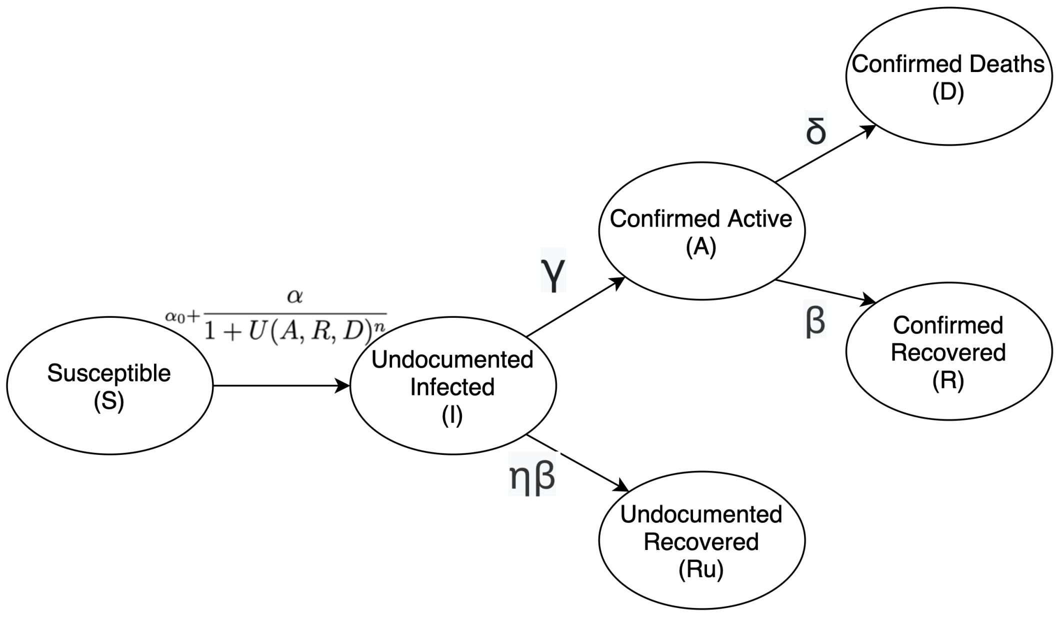
Confirmed Deaths (D)
δ
Confirmed Active (A)
γ
$\alpha_0 + \frac{\alpha}{1+U(A,R,D)^n}$
Susceptible (S)
Undocumented Infected (I)
β
Confirmed Recovered (R)
ηβ
Undocumented Recovered (Ru)

Fig. 1. Overview of the epidemiology model flow. The population of a nation is divided in 6 sub-populations. On a per-day basis, the number of transitions from one sub-population to the other are simulated with a Poisson process, where the rates are governed by the current sub-populations and the transition parameters.

Instead of choosing a fixed threshold, sequential Monte Carlo can be used to transform an initial set of samples to a high quality set with a decreasing sequence of thresholds ϵ and using ABC. This algorithm is called SMC-ABC [3], [5].

C. Hardware Acceleration Platforms

1) Tesla V100 GPU: The current go-to for hardware acceleration for AI applications is the Tesla V100. It is an AI-focused hardware acceleration platform from Nvidia, being widely used in various workloads. It consists of 640 Tensor Cores and 5120 CUDA cores. The solution has a reported 112 TFLOPS of tensor performance and 900GB/s memory bandwidth, with a TDP of 300W.[3]

The Tesla V100 has shown superior performance in a lot of benchmarks like MLPerf [6]. Hence, it is a reasonable choice to test if it can accelerate the ABC algorithm.

2) Intelligence Processing Unit (IPU): For the evaluation we have used a machine with Graphcore's IPU chips. We run the computations on one C2 card, which is a PCIe accelerator card with two Mk1 (first generation) IPU processors in it. A C2 card also has a TDP of 300W.

The IPU is a MIMD (multiple instruction, multiple data) processor. It is well suited for problems that require fine-grain parallelism and high-speed memory access. In particular, models that use autoregressive or sequential elements, employ random memory access patterns, or consist of non-vectorizable parallel paths, can highly benefit from the IPU architecture. Additionally, there is hardware support for random number generation, which is relevant for the application in this paper.

[3]https://www.nvidia.com/en-us/data-center/v100/

Each Mk1 IPU processor can run up to 7,296 independent parallel threads. The bandwidth between the compute and the memory on the chip is 45 TB/s. IPU-Links are used to connect multiple IPUs together for model-parallel and data-parallel execution.

There are three motivations to analyze IPU performance on ABC. First, currently mostly large amounts of CPUs are used for processing. Taking advantage of the independent parallel processes on the IPU can drastically reduce energy consumption and increase performance. Second, the simulation and ABC algorithm fit in the SRAM memory of the Mk1 IPU, which drastically reduces the communication overhead and enables a perfect in-processor computation. Last but not least, there are several new applications with rather complicated computational graphs that are not just based on large scale matrix multiplications, where the IPU showed significant performance gains. Examples are natural language processing [7], image processing [8], bundle adjustment [9], as well as some microbenchmarks [10].

III. Design Methodology

A. Parallelizing ABC inference for Epidemiology Model

In the ABC inference process described in Section II-A and II-B, the computational flow of sampling the parameters to computing the distance function can be performed independently. This is because the specific considerations typically required in concurrently running Bayesian statistical inference algorithms, such as burn-in, auto-correlation, detailed balance, and Independent and Identically Distributed (IID) sampling constraints are either not applicable or easily addressed. This provides us the opportunity to massively accelerate the ABC inference process by following an embarrassingly parallel compute flow to form a batched version of ABC (see Fig.2), without making any fundamental changes to the algorithm itself, while still maintaining asymptotic convergence guarantees on the tolerance. Hence, we express this data-level parallelism in the ABC algorithm by explicitly vectorizing it across a batch of parameter samples, and still have confidence that given sufficiently low tolerance value, the true posterior would be estimated with equal accuracy to the regular ABC. Such vectorized simulation flow is well-supported by the TensorFlow programming style, and allows us to utilize not only the IPU's MIMD architecture, but also the single-instruction-multiple-data (SIMD) architecture of GPUs.

Hence for the epidemiology model discussed in Section II-A, the original ABC algorithm would involve generating a single joint sample of parameters θ of the size [8], and generating data D of size $[3, num\_days]$ (i.e. A, R, D values over the number of days being simulated) and then compare with real data. In the new parallelized ABC algorithm, we sample multiple (batch size of 100k or more) parameters θ by the explicit vectorization size $[100000, 8]$, simulate the resulting data D_s which would be vectorized with size $[100000, 3, num\_days]$, and compare the resulting simulated dataset with the ground truth dataset D to a given tolerance level ϵ. At the end we calculate the number of accepted samples and iterate until a given required total number of accepted samples is obtained.

B. Hardware Optimizations for Accelerating ABC Inference

"XLA (Accelerated Linear Algebra) is a domain-specific compiler for linear algebra that can accelerate TensorFlow models with potentially no source code changes." [4]

XLA runs multiple optimization and analysis passes over the Higher Level Optimization (HLO) representation of the computational graph. The core feature of XLA is fusing operations to more powerful operations. Thus, it can improve memory management as well as execution speed. Apart from target device independent optimizations, there is also a device specific optimization, e.g., to optimize the graph for a GPU or TPU. This optimized XLA representation is also designed as interface to other hardware platforms. Here, the IPU performs its own optimizations and enables a mapping of the computational graph to a lower level code representation that can be eventually executed on the IPU.

Using

```
@tf.function(experimental_compile=True)
```

a TensorFlow function can explicitly be marked for compilation. Alternatively, `TF_XLA_FLAGS` can be used at command-line level. PyTorch can be also used as frontend for XLA. If it is not possible to infer the dimensions of all tensors without actually running the code, a function cannot be XLA compiled. Hence, not every functions is supported by XLA, like for example `tf.unique`.

[4]https://www.tensorflow.org/xla

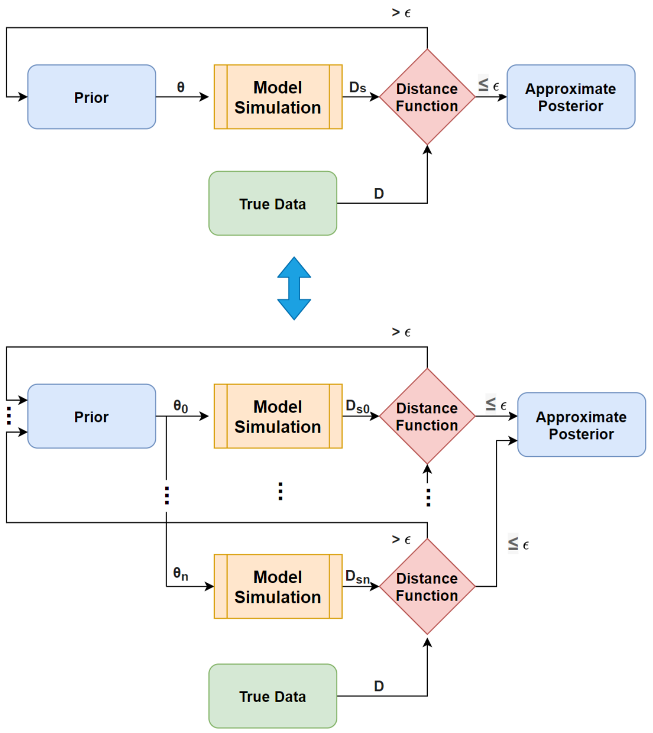

Fig. 2. A conceptual representation of parallelizing ABC inference to batched ABC inference. The sampling from prior, model simulation and distance computation are all embarrassingly parallelized, and this can be accelerated with supporting hardware.

We were interested in comparing computation time for CPU, GPU, and IPU. For the GPU and IPU, XLA compilation is crucial for the performance. Since there is no XLA implementation for Poisson sampling, we used a supported approximation based on the normal distribution.

$$\text{Poisson}(\lambda).\text{get}() \approx \text{floor}(\text{Normal}(\text{mean} = \lambda, \text{std} = \sqrt{\lambda}).\text{get}()) \tag{7}$$

The "floor" term comes from the continuity correction of $+0.5$ and the rounding to integers. The "get" part refers to generating a sample from the distributions. For $\lambda > 10$, this is a common and good approximation. For parameter A, this constraint is fulfilled since day one, for parameter R since day 7 and for parameter D since day 9. The initial days are less relevant because the respective numbers are rather small and influence the error measurement only marginally.

Switching from normal code to XLA compiled code, improved GPU throughput by a factor of 5.

For the IPU, code and processed data are kept in memory. To make optimal use of the processing power, the batch size has to be selected in a way that everything fits in memory. It is crucial to use a large batch size to make sure that the processing power of the IPU is taken advantage of. For the given model, the optimal batch size was 100k. Up to 130k could be used but did not show a performance advantage, since in that case the compiler was optimizing for memory instead

TABLE I
PERFORMANCE COMPARISON

Device	Batch Size	Tolerance	Accepted Samples	Total Time (s)	Time per Run (ms)	Relative *Run* Performance vs. IPU	GPU	CPU
2xIPUs	2x100k	2E+05	100	2.21 ± 0.17	4.63 ± 0.01	1.00	**7.59**	**31.37**
Tesla V100	500k	2E+05	100	14.87 ± 0.01	87.99 ± 0.04	0.13	1.00	4.13
2x Xeon Gold 6248	1M	2E+05	100	67.87 ± 7.27	726.93 ± 6.01	0.03	0.24	1.00
2xIPUs	2x100k	2E+05	**1000**	21.78 ± 0.74	4.63 ± 0.01	1.00	**7.39**	**30.16**
Tesla V100	500k	2E+05	**1000**	154.61 ± 0.04	85.47 ± 0.02	0.13	1.00	4.08
2x Xeon Gold 6248	1M	2E+05	**1000**	660.48 ± 8.74	697.61 ± 4.87	0.03	0.25	1.00
2xIPUs	2x100k	**1E+05**	100	78.34 ± 3.94	4.52 ± 0.00	1.00	**7.55**	**30.42**
Tesla V100	500k	**1E+05**	100	551.57 ± 0.20	85.29 ± 0.04	0.13	1.00	4.32
2x Xeon Gold 6248	1M	**1E+05**	100	2383.10 ± 139.01	693.31 ± 4.71	0.03	0.25	1.00

of speed.

For distributing the processing over two IPUs, two changes were made to the model implementation. First, we use outfeeds to stream the results to the host, which are the sampled parameters and the resulting distance measurements between simulated and real data. Second, we sum the number of accepted samples across the two IPUs after each batch run. We stop iterating as soon as the requested number of samples is obtained.

IV. Results and Discussion

A. Performance

The experimental setup for performance comparison was as follows: For every performance experiment, run *"batch size"* simulations in parallel (vectorized), check how many parameters result in an error lower than the *tolerance* threshold which would give us the number of samples accepted in one *run*. The simulations are repeated until the total number of *accepted samples* exceeds the requested value (100 or 1000).

For our experiments, we use the data of Italy for 49 days, starting at 2020-02-23, the first day with more than 100 active confirmed cases (A_0). We explored inference with two tolerance values, and two different numbers of accepted samples. The optimal batch size for the IPU was selected as the largest batch that can fit the IPU memory, which was 100k per IPU. For GPU, we explored several batch sizes and we chose 500k as the most optimal (see Table II). All experiments were performed 10-fold to capture run-to-run variance. For this performance evaluation, we excluded the time that would be required to collect accepted samples for both GPU and CPU. For the IPU the samples were sent back asynchronously using I/O buffers and this was included in the run-time. The results are displayed in Table I. For time measurements, we provide mean and standard deviation of the 10 recorded values in each setting. The most important number is the total time. We also provide the time per each iteration loop (*Time per Run*). We can see clearly that the CPU performs worst and the IPU shows the best performance. The total time has a large variance because of the randomness in the parameter generation and the simulation. So a better tool for comparison is the more stable *Time per Run*. However, it clearly scales with the batch size. Hence to calculate the *Relative Run Performance*, we normalized the *Time per Run* by the *Batch Size* before determining the speedup. The IPU is around 7.5 times faster than the GPU which itself is around 4 times faster than the CPU.

TABLE II
BATCH SIZE PERFORMANCE FOR GPU, 100 SAMPLES AT $\epsilon = 2.0E + 05$

Batch Size	Total Time (s)
200000	16.92
500000	14.87
1000000	18.66

As a side node, we also observed a significant speed-up in GPU performance with XLA, which is still in experimental stage at the time of writing. While official numbers suggest that XLA could provide up to 50% increase in performance, we observed a ~5x increase in this model. We reported the XLA accelerated times.

Due to the parallelized computation scheme that contains several matrix operations, it is expected that the ABC inference can be accelerated by specialized hardware like GPU and IPU. The superiority of the IPU compared to the GPU is reasonable due to the difference in the architecture. ABC inference has an autoregressive aspect, since the day to day steps have to be performed in sequence. Faster in-processor memory access allows IPU to execute such workloads more efficiently. With all the input data residing on chip, there is no communication overhead to transfer data between external memory and accelerator hardware that is slowing down the data processing. Instead, the IPU can benefit from the fast communication within the chip and exploit localized computation.

B. Simulation and Correctness of Computations

In Figure 3, we compare the resulting model trajectories to the ground truth. We used two different tolerance levels and generated 100 samples from the IPU. Afterwards, we repeated the simulation again with these 100 samples and visualized the resulting statistics of the trajectories (median and percentiles).

We can see clearly that the simulation matches the trajectory on the training data, especially given that we ran a new simulation and did not choose the trajectory generated during the acceptance run. The accuracy of model's predictions holds

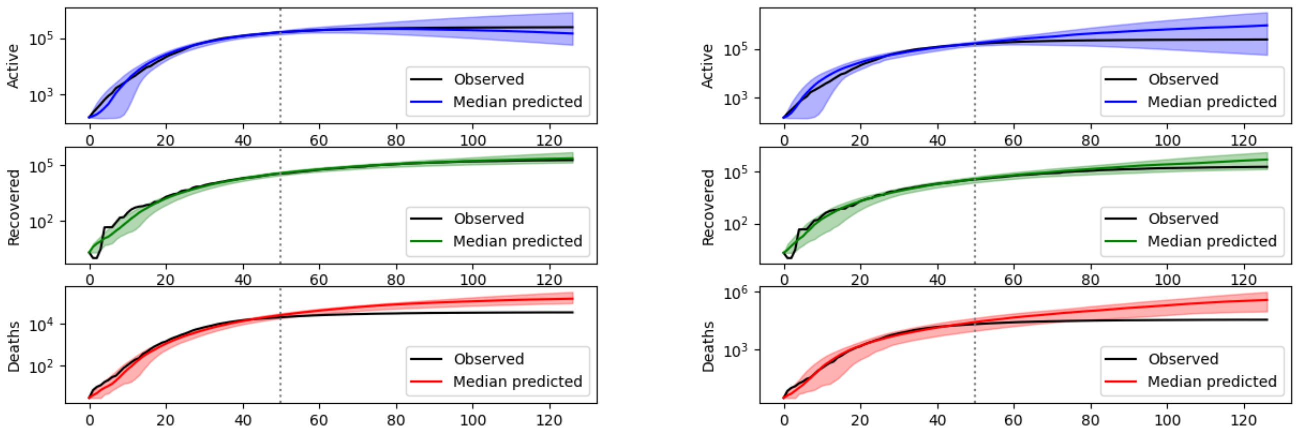

Fig. 3. Trajectories from ABC simulation. This graphic compares simulated and measured cases of tested COVID-19 cases (“Active”), recoveries, and deaths over a time period of 120 days. The x-axis shows the time dimension of the data in days. The dotted line separates between original training data of 49 days (left) and more testing data (right). The prediction is generated with 100 accepted samples and tolerance of $5.00E + 04$ (left) $1.00E + 05$ (right) from the IPU and rerunning the simulation with these parameters. The shaded colored area covers the area between 95% and 5% percentile of the values between the different simulations. The lower tolerance leads to significantly lower bounds and better accuracy in model predictions.

really well for almost a month into the future before there are visible divergences. In case of the predictions in recoveries, the plot with tolerance $5.0E + 04$ exactly matches the observed data through all the three months of the simulation. This verifies the model as well as correctness of our implementation. This plot denotes what the model would have predicted 49 days after 2020-02-23, given the conditions at that time.

The slight divergence after the training suggests that among all the reasonably possible trajectories of COVID-19 cases in Italy, the country managed to employ measures which successfully resulted in more favourable outcomes. However, the objective of the algorithm is not to predict correctly the future but rather provide a distribution of reasonable parameters that describe the training data. The benefit comes from contrasting these distributions between different datasets like a different country or a different time after an intervention.

It is also worth noting that these simulations were generated from a model trained at a tolerances of $1.00E + 05$ and $5.00E + 04$, and have a min-max range of $\sim$ 100x and $\sim$ 20x respectively. Typically, the real-world models would require tighter bounds of $\sim 10 - 5$x. This would require training the model on lower tolerance values, i.e. closer to $5.00E + 04$, which is much more compute intensive, as discussed further in the next subsection.

C. Scalability Analysis

From Table I, we can see how compute time scales with number of accepted samples as well as with reduced tolerance. Increasing the sample size roughly scales linearly. Reducing the tolerance level however drastically increases the total time. The increase in run-time and decrease in target tolerance seems to follow a power-law relationship (super-exponential). We capture this relationship in the plot shown in Figure 4. As discussed in earlier sections, to reduce the confidence bounds of the simulator, a move to lower tolerance would be required. Initial experiments with the IPUs for target tolerance $5.00E + 04$ show that the time to collect 100 samples is $\sim 1.8E + 04$ seconds, i.e. $\sim$ 5 hours. From our performance comparisons from table I, the extrapolated run-time for the same tolerance for GPU and CPU would be $\sim$ 35 hours and $\sim$ 155 hours respectively. This run-time difference can determine the ability to provide daily updates from the model.

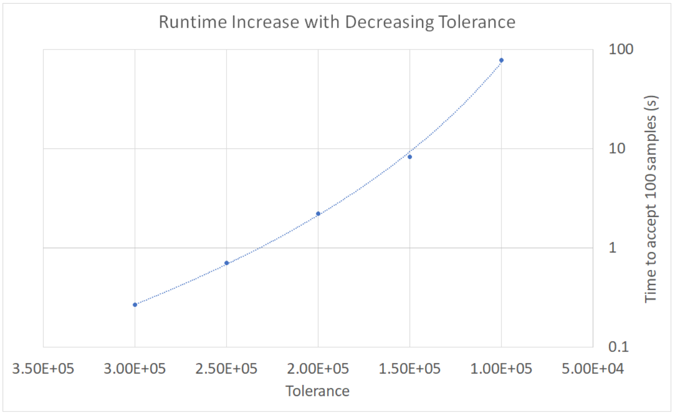

Fig. 4. Scaling of computation time with decreasing tolerance values. As the tolerance decreases, the increase in computation time to accept 100 samples increases super-exponentially. Displayed values are averages across 10 runs on two Mk1 IPUs .

V. Conclusion and Future Work

For exploring different hypotheses, e.g., for contrasting different approaches of COVID-19 response, it is crucial to enable researchers to do fast iterations of simulations and models. In this paper, we showed that the ABC-algorithm, that is used in this field, can be significantly accelerated using hardware that is specialized for processing algorithms that can be described in a form of computational graph. We showed that the IPU can deliver a speedup up to $\sim$30x versus CPU and $\sim$7.5x versus GPU.

In this paper, we focused on the ABC algorithm itself. However, ABC is usually combined with adaptive sequential Monte Carlo sampling in SMC-ABC. Further acceleration through algorithmic innovations such as these is left for future work. Another relevant topic is exploration of the next generation chip, the Mark2 IPU. Its hugely increased memory could allow processing of significantly larger batches of parameters, and result in even higher speedups.

While we consider the epidemiology model and ABC inference for this work, we believe that the proposed approach should generalize to several other scientific models. The models that would benefit the most from this approach would have the following characteristics: i) a relatively small number (< 100) of parameters of interest (even though high numbers of observations would be possible), ii) complex mathematical operations and multiple intermediate stochastic dependencies, and iii) involve simulating the (usually temporal) evolution of a system. There is a large number of simulation models in a wide variety of scientific domains that satisfy those conditions. Examples include elementary particle interaction simulations in particle accelerators [11], protein folding simulations [12], genome-wide association studies [13], macroeconomic simulations [14], galactic and cosmological gravitational simulations [15], and cosmic microwave background studies [16], etc. Most of these applications are compute intensive and typically require use of supercomputers, hence any hardware acceleration benefits achieved would have a significant impact on those domains.

Acknowledgment

The access to IPU servers used to generate the results for IPUs and Xeon CPUs was generously provided by Cirrascale.[5]

References

[1] K. Cranmer, J. Brehmer, and G. Louppe, "The frontier of simulation-based inference," *Proceedings of the National Academy of Sciences*, 2020. [Online]. Available: https://www.pnas.org/content/early/2020/05/28/1912789117

[2] M. Sunnåker, A. G. Busetto, E. Numminen, J. Corander, M. Foll, and C. Dessimoz, "Approximate bayesian computation," *PLOS Computational Biology*, vol. 9, no. 1, pp. 1–10, 01 2013. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1002803

[3] D. J. Warne, A. Ebert, C. Drovandi, A. Mira, and K. Mengersen, "Hindsight is 2020 vision: Characterisation of the global response to the COVID-19 pandemic," *medRxiv*, p. 2020.04.30.20085662, may 2020.

[4] E. Dong, H. Du, and L. Gardner, "An interactive web-based dashboard to track COVID-19 in real time," pp. 533–534, may 2020.

[5] C. C. Drovandi and A. N. Pettitt, "Estimation of Parameters for Macroparasite Population Evolution Using Approximate Bayesian Computation," *Biometrics*, vol. 67, no. 1, pp. 225–233, mar 2011. [Online]. Available: http://doi.wiley.com/10.1111/j.1541-0420.2010.01410.x

[5]https://cirrascale.com/graphcore-cloud.php

[6] P. Mattson, C. Cheng, C. Coleman, G. Diamos, P. Micikevicius, D. Patterson, H. Tang, G.-Y. Wei, P. Bailis, V. Bittorf, D. Brooks, D. Chen, D. Dutta, U. Gupta, K. Hazelwood, A. Hock, X. Huang, A. Ike, B. Jia, D. Kang, D. Kanter, N. Kumar, J. Liao, G. Ma, D. Narayanan, T. Oguntebi, G. Pekhimenko, L. Pentecost, V. J. Reddi, T. Robie, T. S. John, T. Tabaru, C.-J. Wu, L. Xu, M. Yamazaki, C. Young, and M. Zaharia, "Mlperf training benchmark," 2019.

[7] A. Wagner, T. Mitra, M. Iyer, G. Da Costa, and M. Tremblay, "Position Masking for Language Models," *ArXiv*, June 2020. [Online]. Available: http://arxiv.org/abs/2006.05676

[8] I. Kacher, M. Portaz, H. Randrianarivo, and S. Peyronnet, "Graphcore C2 Card performance for image-based deep learning application: A Report," *ArXiv*, feb 2020. [Online]. Available: http://arxiv.org/abs/2002.11670

[9] J. Ortiz, M. Pupilli, S. Leutenegger, and A. J. Davison, "Bundle adjustment on a graph processor," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[10] Z. Jia, B. Tillman, M. Maggioni, and D. P. Scarpazza, "Dissecting the Graphcore IPU architecture via microbenchmarking," *ArXiv*, vol. abs/1912.03413, 2019.

[11] C. Collaboration, S. Chatrchyan, G. Hmayakyan, V. Khachatryan, A. Sirunyan, W. Adam, T. Bauer, T. Bergauer, H. Bergauer, M. Dragicevic *et al.*, "The cms experiment at the cern lhc," 2008.

[12] F. Liang and W. H. Wong, "Evolutionary monte carlo for protein folding simulations," *The Journal of Chemical Physics*, vol. 115, no. 7, pp. 3374–3380, 2001.

[13] B. Peng and C. I. Amos, "Forward-time simulation of realistic samples for genome-wide association studies," *BMC bioinformatics*, vol. 11, no. 1, pp. 1–12, 2010.

[14] J. D. Sachs and N. Roubini, "Sources of macroeconomic imbalances in the world economy: a simulation approach," National Bureau of Economic Research, Tech. Rep., 1987.

[15] R. E. Sanderson, A. Wetzel, S. Loebman, S. Sharma, P. F. Hopkins, S. Garrison-Kimmel, C.-A. Faucher-Giguère, D. Kereš, and E. Quataert, "Synthetic gaia surveys from the fire cosmological simulations of milky way-mass galaxies," *The Astrophysical Journal Supplement Series*, vol. 246, no. 1, p. 6, Jan 2020. [Online]. Available: http://dx.doi.org/10.3847/1538-4365/ab5b9d

[16] M. Liguori, S. Matarrese, and L. Moscardini, "High-resolution simulations of non-gaussian cosmic microwave background maps in spherical coordinates," *The Astrophysical Journal*, vol. 597, no. 1, p. 57, 2003.