

Machine Learning and High-Performance Computing Hybrid Systems, a New Way of Performance Acceleration in Engineering and Scientific Applications

Pawel Gepner

Warsaw Technical University
ul. Narbutta 86, 02-524 Warszawa, Poland
email: pawel.gepner@pw.edu.pl

Abstract—Machine learning is one of the hottest topics in IT industry as well as in academia. Some of the IT leaders and scientists believe that this is going to totally revolutionise the industry. This transformation is happening on both fronts, one is the application and software paradigm, the other is at the hardware and system level. At the same time, the High-Performance Computing segment is striving to achieve the level of Exascale performance. It is not debatable that to meet such level of performance and keep the cost of system and power consumption on reasonable level is not a trivial task. In this article, we try to look at a potential solution to these problems and discuss a new approach to building systems and software to meet these challenges and the growing needs of the computing power for HPC systems on the one hand, but also be ready for a new type of workload including Artificial Intelligence type of applications.

Index Terms—Machine Learning, High-Performance Computing, Exascale performance, HPC systems, IPU

I. INTRODUCTION

TODAY'S High-Performance Computing systems are build out of thousands of nodes containing a couple of CPUs and one-or-many accelerators, mostly GPUs, onto the same node. All these nodes are connected using dedicated fabric to make a cluster, the most common and universal supercomputer of the present day. Clusters definitely dominate the HPC market and occupy 92% of all the systems ranked on the TOP 500 list [1]. From an architecture stand point one-such- node of the system represents a hybrid architecture, including CPU processor and accelerating unit. Basically we find the major CPU architecture commonly used for HPC are Intel/AMD x86, Arm, and Power. According the latest Top 500 list more than 28% of the systems utilize GPUs [1], also on this segment we have a choice between, Nvidia and AMD, as well as, soon to be third player — Intel with their GPU architecture. Integrating hundreds or even thousands of heterogeneous nodes together requires special interconnect technology and advanced topology for the network part. Today we have basically two solution scenarios: InfiniBand with low latency and standard Ethernet with all its advantages and disadvantages. Unfortunately, both new technologies, such as

OPA or BXI, which could become a potential alternative, have not achieved significant market penetration and adoption to be considered real players.

From a programming perspective, such solutions are not trivial, we need to properly handle heterogeneous infrastructure and be fully aware of the hybrid structure of code. Effective use and management of the offload-acceleration parts and utilizing CPU portions of the code is not a simple task and requires experienced and knowledgeable developers.

There is no doubt that in the case of large Exascale systems, homogeneous systems based on CPUs only are unable to provide the required performance in a reasonable size and do not destroy the power budget. It seems that the first few Exascale systems, we are going see in the next couple of years, will be based on a heterogeneous architecture - CPUs plus GPUs, although in diverse combinations of vendors for these components.

Artificial Intelligence-AI and Machine Learning-ML systems have been discussed for decades, but limited access to large data sets, lack of relevant computing architectures and available systems able to execute the AI workloads have restrained AI developments, until the last couple of years.

Artificial intelligence systems are based on one of the combinations of the following types of devices in conjunction with a Central Processor Unit:

- GPU – Graphics Processing Units.
- FPGA – Field Programmable Gate Arrays.
- ASIC – Application Specific Integrated Circuits.

Many of today's AI systems only use modern, efficient multi-core processors to solve ML tasks. Processor vendors are trying to support this segment by optimizing the architecture, extending instruction sets to best address common ML workloads, but also by adding a new data format and special precision for this type of computation.

Despite these efforts by CPU manufacturers, processor-based systems are not considered the most effective solution for processing AI-oriented tasks, especially in the training phase, and dedicated hardware has earned a reputation as the best solution for this type of applications.

Solutions such as GPU, FPGA or specialized ASICs such as Tensor Processing Units-TPU from Google, Inferentia from Amazon, Gaudi and Goya from Habana Labs, Grayskull from Tenstorrent, SambaNova AI Chip, Cerebras Wafer Scale Engine or a special dedicated Intelligent Processor Unit-IPU from Graphcore are gaining great recognition and a reputation for being the best solution for ML workloads.

Initially, the GPU was designed to accelerate multidimensional data processing in computer graphics applications. The GPU consists of thousands of small cores designed to work independently. On the one hand ensures intensive processing, but at the same time requires advanced management of the memory subsystem and operating on the subset of data on which it runs. A GPU can perform complex computational operations for computer graphics such as texture mapping, resizing and cropping images, rotating and flipping, translation and filtering, and the dedicated memory makes these operations fast and efficient.

The HPC community realized the possibilities and benefits of using the GPU to accelerate linear algebra calculations long before the AI researchers started using them to solve machine learning problems. The GPU has been found to benefit wherever we are able to take advantage of the natural parallelism of algorithms and accelerate them by parallel execution. At the same time, from specialized chips dedicated to graphics solutions they became fully programmable graphics processors. Today they are still specialized parallel processors, but also highly programmable and the spectrum of implementation has grown significantly.

Deep neural network calculations are based on a similar type of operation as linear algebra calculations, so the natural consequence of this fact is the use of GPUs to solve deep learning problems. GPUs are throughput processors and can provide high throughput and be an effective solution for accelerating HPC and machine learning type of systems.

Unlike GPUs, FPGAs did not make a stunning career in the HPC market but for Machine Learning specially in inference it is very interesting alternative for already trained model. FPGAs are an array of programmable logic blocks and memory, connected via a hierarchy of reconfigurable interconnects that allow the blocks to be wired together to perform specific complex tasks. Since the FPGA is not a processor, it cannot execute the program stored in memory, but it can perfectly execute the code for which it has been configured. FPGAs are inherently parallel, so they're a perfectly matching parallel computing capabilities of machine learning models. To configure and store the executable program in to the chip and make it operational we use a fully parallel hardware description language - HDL. Results show that FPGA provides superior performance/Watt over CPU and GPU because FPGA's on-chip extracting fine-grained parallelisms and matches the specifics of the machine learning code.

Specialized ASICs for AI solutions, like all customised chips, have their own specific requirements optimized for AI, which radically accelerate the calculations demanded by ML algorithms. They are based on massive parallel operations

that are performed simultaneously, instead of sequentially, to achieve the required performance. Dedicated AI chips require low precision computation in a way that AI algorithms implement them effectively, this approach reduces precision but speeds up execution and reduces the number of transistors required for the same computation. Customised AI ASICs are characterized by super-efficient memory access, where they store all of the data necessary for the correct implementation of the algorithm, therefore, with the growth of models, the memory of such chips also evolves over time. Many companies have brought AI specialised chips to the market and number of new solutions increases every day. Different types of AI chips are useful for different tasks some of them were developed to do training others to do inference others for both domains.

In this article, we propose a hybrid system architecture capable of solving both HPC and AI problems. Such a solution is utilising classic HPC platforms and dedicated AI chip systems, which is particularly important in the context of the challenges of the new types of Exascale systems. We will discuss not only the implications at the chip level, but also the system approach along with the necessary modification of algorithms and software, as well as the performance. The approach of the heterogeneous system presented in the article will allow us to propose a system capable of accelerating the most complex simulation problems. The construction of such a heterogeneous system is based on customized AI chips and systems utilizing Graphcore's Intelligent Processor Unit-IPU for executing a new hybrid algorithm acceleration scenario. Of course on the market we can find other ASICs and dedicated AI type of chips but IPU appears to be the most mature and universal solution available on the market today, with a solid and comprehensive software ecosystem.

This paper is organized as follows. Section 2 presents related works. Section 3 is devoted to the introduction of the Intelligent Processor Unit. It explains architectural details of IPU and describes the way it works and the principles of it's programming model. In Section 4, we concentrate on the details of the proposed system architecture, discuss the benefits and challenges. Section 5 discusses the programming approach and the new way algorithms are tuned to the architecture that enables the use of a hybridization approach. In Section 6, we present the conclusions of the conducted architectural proposal.

II. RELATED WORKS

Over the last few years, there have been many attempts at improving the architecture as well as many discussions on development directions of hardware technology, as well as attempts to use the existing High Performance Computing technology in Machine Learning frameworks and artificial intelligence systems [2]. The convergence of HPC and artificial intelligence [2], [3] offers a promising approach to major performance improvements. As classic HPC simulations are reaching the limits of their progress and slowing down due to the stagnation of Moore's law that has led to the proliferation of various accelerator architectures. These architectures are

still evolving, resulting in very costly, if not harmful, modification of the scientific codes that must be optimised to bring out the last marginal gains of parallelism and efficiency. In many application areas, the integration of traditional HPC approaches with machine learning methods is perhaps the greatest promise to overcome these barriers.

The need to increase performance is the clue to the international efforts behind the Exascale supercomputer projects. Exascale initiatives in the US, Japan and Europe exploring the possible integration of ML with large-scale computing is a very promising way to achieve high performance. There is undoubtedly a close relationship between machine learning and high performance computing as machine learning algorithms are based on the same basic linear algebra operations that are used in HPC. The question that arises is how to efficiently use the existing HPC infrastructure for ML applications and can ML code be incorporated into HPC simulation and is this approach optimal?

Interesting attempts to answer these questions can be found in works such as Jeff Dean's "Machine Learning for Systems and Systems for Machine Learning" [4] and Satoshi Matsuoka's "Convergence of AI and HPC" [5]. However, the most advanced categorization of such systems, and even the development of a specific taxonomy, can be found in Geoffrey Fox's et al. "Learning Everywhere: Pervasive Machine Learning for Effective High-Performance Computation" [2]. The authors attempted to classify a new category of system and the type of calculations associated with them, they proposed a method for their hierarchy, and even defined some performance metrics. They distinguished classical performance as measured by Flops or benchmark results from effective performance, which is achieved by combining learning with simulation and delivering increased performance as seen by the user. This is absolutely crucial in the cases where there is a combination of machine learning and the components of a traditional HPC simulation where classical benchmarks are not representative at all.

On the basis of the proposed classification, several clearly different approaches to the construction of systems using machine learning and HPC solutions have been identified [2]. Basically, two categories of systems are distinguished: HPCforML and MLforHPC:

- HPCforML: Utilising HPC to run and enhance ML performance, or using HPC simulations to train ML algorithms.
- MLforHPC: Deploy ML to enhance HPC applications and systems.

Of course, the proposed split divides the system based on particular type of calculations. If ML and AI is the major target, then we can utilize the HPC installations for acceleration of AI type of code, but if we have a HPC traditional problem and AI and ML seems to be a good way for acceleration run time than, the system belongs to the second class [3]. This is absolutely clear that the main categories of systems have their subclasses and they are defined as follows:

- HPCrunsML: Using HPC to perform ML type of workloads with highest degree of effective using the HPC acceleration capabilities [2], [3].
- SimulationTrainedML: Using HPC simulations to train ML algorithms, which are then used to understand experimental data [2], [3].
- MLautotuning: Using ML to configure autotune HPC simulations. MLautotuning can also be used for simulation mesh sizes and in big data problems for configuring databases and complex systems like Hadoop and Spark [2], [3].
- MLafterHPC: ML analyzing results of HPC simulation [2], [3].
- MLaroundHPC: Using ML for learning from simulations and creating learned solvers replacing classic HPC wrappers [2], [3].
- MLControl: Using ML to control of experiments and simulation run on HPC system [2], [3].

There are many research groups working on all of these HPC and AI hybridization categories, and most of the work is just starting and still in an early stage of development, but by far the most attractive in terms of performance and potential is MLaroundHPC. Some groups have already very impressive results with MLaroundHPC especially in high energy physics, materials science, weather simulation, epidemic forecasting, tissue and cellular simulations, nanoscale and biomolecular simulation, computational fluid dynamics simulation and many others [3].

Another aspect often discussed in the context of combining AI and HPC is the question of measuring the performance of such solutions. In the classic model of measuring the efficiency of HPC systems, the situation is quite obvious "faster is better" which in turn prompts the acceleration of individual work units. For hybrid systems, the ML component should be included, and this requires both hierarchical (vertical) and horizontal (multitasking) parallelism [6].

Much effort has been put into developing a new type of hybrid systems that can scale to very large sizes and bring performance benefits to a whole new level that is impossible to achieve based on classical simulation and HPC. The proposed hybrid approach in this paper incorporating AI and HPC components takes into account these new trends and proposes the use of solutions that have already proved their usefulness in AI solutions.

III. INTELLIGENT PROCESSOR UNIT

Bulk Synchronous Parallel - BSP model is the foundation for the architectural assumptions of the IPU Colossus MK2 GC200 processor and the Poplar programming model. Valiant proposed the Bulk Synchronous Parallel model as a parallel computing abstraction scheme that facilitates the expression of parallel algorithms, helps design large paralleled systems and makes it easier to analyse the performance they achieve while running [7].

This model, proposed in the 1980s, is a parallel counterpart to the Von Neumann model's for sequential computing, and

allows exemplification structure of parallel algorithms and can be used for performance characteristic measurement [8].

BSP is a parallel computing model that IPU use to organize data processing and exchange operations. The way the BSP model arranges computation is based on sequential 3-steps phases [8]. This 3-steps phase is constituted of:

- local computation phase - every process performs computation that operates exclusively on local memory. No communication between processes occurs in this phase.
- communication phase - processes exchange data and each process may send a message to each looked-for destination counterpart. No computation occurs in this phase. Processes can use the communication phase not only to send each other intermediate computation results, but also to request and (at a later communications stage) receive data from remote memories. This mechanism allows each process to use other local memory as a remote memory and to ultimately access the entire aggregate system memory as one larger store.
- barrier synchronization phase - no process continues to the next phase till all processes have reached the barrier. Neither computation nor communication occurs in this phase except for that strictly required by the barrier itself.

The BSP model is ideally suited to describe any parallel algorithms of any complexity and allowing software developers to specify processes in terms of graph vertices that compute on local data. Input operands are fetched to each process by the run-time system before the computation begins, in the communication phase related to the previous cycle [8].

The hardware-assisted programming model not only enforces phase separation but also ensures the coupling of both entities. Since the IPU cores can only access directly the local memories on the chips, this naturally imposes local constraints on the computational phase. The hardware-implemented all-to-all memory exchange mechanism provides native support and enforces control and exchange during the communication and synchronization barriers phases [7].

From the hardware definition IPU is distributed memory, massively parallel, multiple-instruction, multiple-data (MIMD) devices. Each IPU has 1472 cores, each with its own on-chip 624KiB SRAM memory per core, combination of core and associated on-chip memory is named a tile. With 1472 tiles the IPU has just under 900 MB of memory in total. This local memory is the only memory directly accessible by tile instructions. It is used for both the code and data used by that tile. There is no shared memory access between tiles. The tile uses a contiguous unsigned 21-bit address space, beginning at address 0x0. The effect of accessing unpopulated memory addresses is undefined. Memory parity errors can occur when data is read from memory, for example, by a load instruction or an instruction fetch. A parity error detected in a fetched instruction prevents the execution of that instruction. Tiles cannot directly access each other's' memory but can communicate via message passing using an all-to-all high bandwidth exchange (theoretical 8 TB/s). The memory has very low-latency (6 cycles) and ultra-high bandwidth (theoretical 47.5

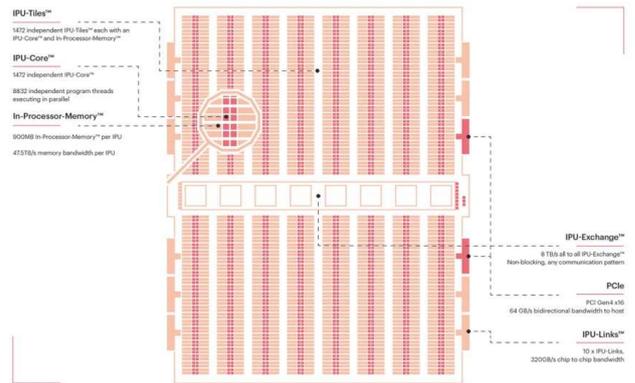


Fig. 1. Simplified version of IPU die

TB/s). The whole chip is built on the budget of 59.4 billion transistors and using the TSMC 7nm manufacturing process [9]. Fig. 1 shows simplified schematic of IPU die.

The IPU is specifically designed for machine learning type of computation, and the tile Instruction Set Architecture-ISA[15] includes focussed hardware elements such as Accumulating Matrix Product-AMP units and Slim Convolution Units-SLICs which allow up to 64 multiply-add instructions to complete per clock cycle. There are also hardware support instructions for random number generation and some selected, generally used in machine learning, transcendental operations. The IPU supports 32-bit single-precision floating point FP32-IEEE, as well as FP16-IEEE 16-bit half-precision floating point numbers of data format with hardware stochastic rounding support. Every tile runs 6 hardware execution threads in a time-sliced round-robin schedule, allowing instruction and memory latency to be hidden. With this mechanism, most instructions, including memory access and vectorised floating-point operations, complete within one thread cycle (6 clock cycles). Every thread represents a truly independent program, there is no restriction that threads run in groups executing the same program in lockstep, and no requirement that memory accesses are coalesced to achieve the high SRAM bandwidth [9].

All these architecture principals have been carefully selected to entirely support the machine learning specific workloads but at the same time they make the IPU one of the most powerful devices on the market. Table 1 shows comparison of existing CPUs and GPUs available on the market. This table clearly illustrates that IPU has a significant advantage in terms of the available resources but also theoretical performance versus other devices uses for machine learning applications.

The programming interface to access the IPU is the Graph-core Poplar programming framework. Poplar is a graph programming environment that essentially extends the functionality of C++ by transforming it into the IPU operation model and is based on three concepts:

- Vertexes are the programs which execute on individual tiles. Vertexes in Poplar are subclasses of the Vertex class. They each have a compute method that is run on the

TABLE I
CHARACTERISTICS OF CPUs, GPUS AND IPU

Architecture	Memory	Capacity	Frequency (GHz)	Bandwidth (GB/s)	FP32-TFLOPS	FP16-TFLOPS
Graphcore IPU	SRAM	900 MB	1.325	47500	61	245
Intel, Xeon 8380	L1/L2/L3/DRAM	48KB/1.25MB/60MB/4TB	2.3	7048/5424/1927/225	5.8	11.6
AMD 7742 (Rome)	L1/L2/L3/DRAM	4MB/32MB/256MB/4TB	2.25	190	4.7	9.4
Nvidia GPU-A10	L1/L2/HBM-2	128KB/6MB/24GB	0.885	600.2	31.2	31.24
Nvidia GPU-A100	L1/L2/HBM-2	192KB/40MB/40GB	0.765	1555	19.5	77.97
Nvidia GPU-V100	L1/L2/HBM-2	128KB/6MB/16GB	1.312	897	16.4	31.33
AMD GPU-MI-100	L1/L2/HBM-2	16KB/8MB/32GB	1	1229	23.07	184.6

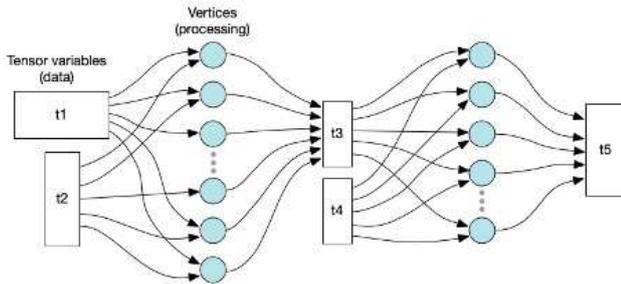


Fig. 2. IPU Computation Graphs concept

tile and returns a bool value. Vertices are defining an interface of inputs and outputs which later allows them to be wired into the Computation Graph. The function performed by a vertex can be anything from a simple arithmetic operation to reshaping tensor data operation or performing a complicated code.

- Computation graph defines the input/output relationship between variables and operations. Poplar provides functionality for constructing, compiling, and serialising the computation graph.
- Control programs administrate arguments, select IPU devices, control the execution of the graph operations.

Fig 2 shows the concept of the IPU computation graphs which defines the input/output relationship between variables and operations.

The graph is made up of tensor variable (variables in the graph), compute tasks (vertices) and edges that connect them. Data is stored in the graph in fixed size multi-dimensional tensors, a vertex is a specific piece of work to be carried out and the edges determine which variable elements are processed by the vertex. A vertex can connect to a single element or a range of elements. Each vertex is associated with a codelet - piece of code that defines the inputs, outputs and internal state of a vertex. Codelet is implemented in standard C++11 [10]. Example 3.1 shows simple example of the Adder vertex (vertex that adds two numbers).

Example 3.1 (C++): The Adder vertex:

```
#include <poplar/Vertex.hpp>

using namespace poplar;
class AdderVertex : public Vertex {
```

```
public:
    Input<float> x;
    Input<float> y;
    Output<float> sum;

    bool compute() {
        *sum = x + y;
        return true;
    }
};
```

The final element that brings all the elements together is the control program, it organizes the selection of devices, loads compiled graphs into the IPU and executes graph programs. An important part of this is the mapping of data transfers between the IPU and the host, memory structures, and initiating transfers. Once the program is deployed, all the code and data structures required to run the program reside in the IPU's distributed memory [10]. The control programs run in order to execute the appropriate vertices.

IV. HYBRID SYSTEM ARCHITECTURE

The next generation of AI systems promises to accelerate computer vision, speech recognition, machine translation systems and increasingly impact our lives. Realizing this promise we need AI systems that can compute massively increasing amounts of data and do it in realistically short time. In the same time the size of the HPC systems is increasing but the level of utilization does not necessarily evolve in the same direction. The most extensive and sophisticated of today's HPC models are so computationally expensive that researchers need to replace parts of the model with approximation mechanisms that transfer accuracy to speed, only to obtain a simulation that can be performed at feasible scale and resolution. It turns out that ML models are universal and accurate approximations, so one direction of research is to use accurate but slow numeric code to generate training data for the ML model that can then be implemented on a large scale, more efficiently, and maybe even more accurately. To make this transformation successful we need the specialized hardware dedicated for ML framework from one hand but should not jeopardise the HPC requirements. Due to different architectural characteristics and the large number of system parameter configurations (such as, the number of threads, thread affinity, workload partitioning

between multi-core processors of the host and the accelerating devices), achieving a good workload distribution that results with optimal performance for HPC 64 bits and ML 32 bit and 16-bit workloads is not a trivial task. An optimal system configuration that results with the highest throughput and performance for HPC may not necessarily be the most effective solution for AI type of solvers. Moreover, the optimal system configuration for AI workloads is likely to change for the different types of applications, sizes of input problems, and available resources that we have in HPC. Taking all these elements into account the proposed approach of hybridisation the system architecture appears to be a creditable solutions.

Suggested systems contain two parts AI and HPC dedicated portion but from the networking, storage and orchestration, administration point of view it is a single instance. This type of the approach provides an optimized solution for the monolithically type of HPC code which is running on the HPC section of the system without any limitation, as well as the AI portion can be utilized only when we have ML workload. The most complicated scenario appears for the hybrid type of code when they have HPC section involved in the preparation phase for the data generating process for AI optimized solver. These new architectures will aim not only to improve the performance, but to simplify the development of the next generation of AI and HPC hybrid applications by providing rich libraries of modules that are easily composable.

Many institutions have already started investigation and development of hybrid systems for example, the University of California, Berkeley Firebox project or Lawrence Livermore National Laboratory - LLNL with their two heterogeneous systems. The LLNL integrated AI-specific systems one from Cerebras and one from SambaNova into two existing LLNL HPC systems (Lassen (23 petaflops) and Corona (10 petaflops)) to achieve system level heterogeneity [11]. Many other works have been carried out and many centres are experimenting with different setups and configurations. Most of today's research is concerned with the allocation of resources and the interaction between different types of resources. One should also consider what is the proper ratio of these different types of resources and what should be a reasonable size of the systems between the HPC and AI components. A particularly important question is the need to provide disaggregated network bandwidth that will be sufficient and network latency low enough to tie the two domains together and meet the needs of each task. Fig. 3 shows the simplified version of the system with separated HPC classic system and ML section. The HPC section is CPU or CPU plus accelerator based. This part is dedicated for typical HPC simulation workload. Second section is ML system responsible for AI type of workloads. Both systems are connected via unified network e.g. Ethernet and they are utilizing the storage subsystem which can be shared simultaneously or as the aggregated bandwidth subsystem for hybrid type of workloads.

The proposed architecture can utilize any type of HPC system based on the existing infrastructure as we see e.g., in LLNL or completely new specially built system. For the

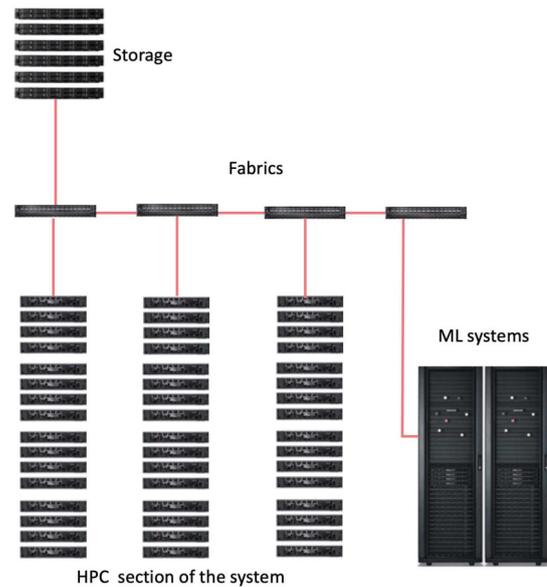


Fig. 3. Schematic of the hybrid system

machine learning component, the proposed system is based on Graphcore architecture for a couple of reasons. The performance of the IPU itself and architecture customisation for ML workloads already discussed in the previous section of the article, also because of the unique approach to the architecture of the IPU systems, based on disaggregated approach for the platform level integration and scalability.

Graphcore IPU-M2000 system is basically a 1U server utilizing 4 IPUs, gateway chip which connects IPUs into compute domain, provides access to the DRAM, two 100Gbps IPU-Fabric Links, a PCIe slot for standard Smart NICs, two 1GbE Open BMC management interfaces, and access to an M.2 slot. Fig.4 shows the block diagram of the IPU-M2000 system. The host system accesses IPU-M2000 platform over 100Gb Ethernet with ROCE (RDMA over Converged Ethernet) with very low-latency access. Such an implementation based on Ethernet avoids the bottlenecks and costs of PCIe connectors and PCIe switches and enables a flexible host CPU to accelerators combination and provides the scaling from single IPU-M2000 system to massive supercomputer scale including 64000 IPUs, all networked over standard networking at lower cost and much more flexibility than using e.g., InfiniBand [12].

The IPU-Fabric is a totally new scale-out fabric designed from the ground up to support the needs of machine intelligence communication. The IPU-Fabric is natively integrated into the IPU processors and IPU-M2000 system. A key difference between IPU-Fabric and other proprietary fabrics are the usage of Compiled Communication and Bulk Synchronous Protocol, both these elements provide deterministic communication behaviour. Every IPU has dedicated IPU-Links providing 64GB/s of bidirectional bandwidth and an aggregate

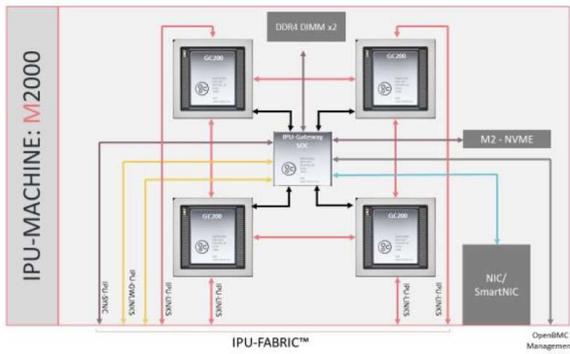


Fig. 4. Schematic and building block of IPU-M2000 Machine

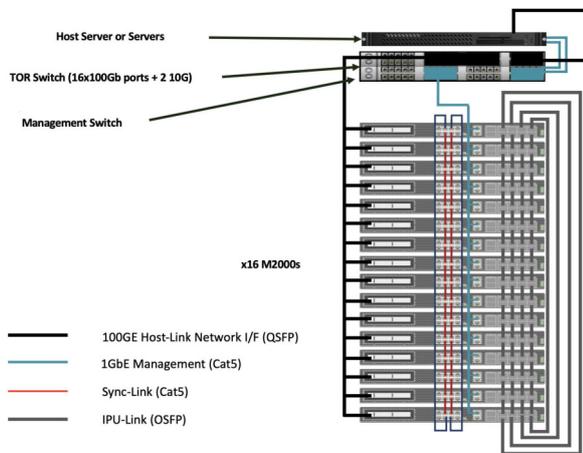


Fig. 5. IPU-POD64 configuration

bandwidth per chip of 320 GB/s. Each IPU-M2000 has 8 external IPU-Links for intra-rack scale out using OSFP copper cables. The intra-rack configuration called IPU-POD64 contains 16 of IPU-M2000's connected into a single instance with 2D ring topology utilizing IPU-Links. Host-Link connectivity is provided from the GW through a PCIe NIC or SmartNIC card. Fig. 5 shows the IPU-POD64 configuration [12].

For inter-rack scale out the IPU-GW provides 100GbE ports that tunnel the IPU-Link protocol over regular Ethernet, these links are named as IPU-GW-Links. Physically every IPU-M2000 system has 2 QSFP ports that support both optical transceivers and copper cables for rack-to-rack connectivity. Each IPU-GW-Link represent a switch plane. In an IPU-POD64 there are 32 such planes and with 32 128-port 1U 100GbE switches it can be scaled to 8000 IPU's in a single switch hop. IPU-Fabric can connect clusters of IPU-PODs in scale-out from a few IPU-PODs to 1,000's of IPU-PODs and can scale to support a cluster of up to 64000 IPU's that can work as a singular AI compute entity or supporting 1000's of different workloads and tenants. The IPU-Fabric

is fully compatible with 100Gb Ethernet using QSFP/OSFP connectors and standard switches. These can be used to connect IPU-Fabric clusters and to build larger systems and to fit in with existing datacentre technology[12].

The memory model for the IPU-Machine is also quite unique and in addition to In-IPU Memory each IPU-M2000 systems has DDR memory available to the four IPU's. This DDR memory is used differently from that found in CPUs or GPUs. Instead of a memory hierarchy that requires swapping data and code from host memory store to the accelerator's memory, the Poplar Graph Compiler creates the deterministic code-memory relationships in both the memory on the IPU tile and the DDR memory. In fact, the IPU-M2000 system can use this additional memory in stand-alone mode for inference processing without any attachment to a host server. And thanks to the BSP model compiling both computation and communication, the network communication overhead is kept to a minimum compared to traditional messaging or shared memory constructs commonly used for parallel processing.

Built-in fabrics are becoming a necessity for AI accelerators since model sizes are increasing dramatically, some containing billions of parameters. These large models must be distributed across hundreds or thousands of processors to solve problems in a reasonable time. Graphcore's hybrid model uses a proprietary IPU-Link fabric to communicate across the tiles in an IPU and adjacent rack IPU's, while tunnelling the IPU-Link protocol across standard 100GbE for rack-to-rack scale-out supporting larger configurations [12].

This disaggregated scaling model is the most important feature of IPU-M2000 based systems and, together with IPU-Fabric, enables a flexible disaggregation model, allowing the user to configure multiple accelerators on the fly without constraints by a predetermined scenario. It is also an important architectural element in the context of the hybrid type of system discussed in this article, where the HPC and AI part can be dynamically reconfigured based on the requirements and specifics of the code allocating HPC processors and accelerators in combination with the ML component based on the task requirements and code characteristics. It would be ideal to implement this in a multidimensional backbone that is efficiently supported by IPU-Fabric and an HPC network topology that would allow direct 1 to 1 communication with very low latency.

V. PROGRAMMING HETEROGENEOUS SYSTEMS

High performance scientific computing has traditionally focused on scaling and increasing the performance of a single large task. In fact, they were designed and optimized for the efficient execution of a few large-scale simulations, instead of the large number of smaller scale simulations necessary e.g., to train accurate ML models. However, even a new hybrid system as proposed in previous section of the article without a radical redesign of algorithms and computational methods faces increasingly serious limitations in the ability to scale single monolithic applications and achieve significant performance gains on large parallel machines. In response,

HPC scientific applications are looking for new methods to reduce time to obtain scientific insight, such as the use of team - aggregated simulations [6], [13] and the integration of artificial intelligence and machine learning methodologies with traditional HPC applications. The coupling of ML methods and HPC simulations is not trivial and it involves a fundamental reconfiguration of classic HPC algorithms and the use of appropriate data sets. In particular, there are two design strategies:

- Sequential implementation of the simulation followed by AI / ML training and subsequent inference runs.
- Stream implementation of the AI / ML component that enables independent and concurrent running of simulation and AI / ML tasks, but with the possibility of data exchange by individual tasks at runtime. Tasks can still be interdependent, depending on how the simulations are selected using the AI / ML methods to run next.

The second strategy, although even more attractive and innovative, requires a complete rebuild of existing codes and the development of a completely new application, so most of the work and efforts of the researchers currently focuses on a strategy ensuring interaction between artificial intelligence and classical solvers for much faster analysis and reduced simulation time. Although using machine learning methods to speed up simulations can significantly improve the performance of scientific applications, there are many limitations that must be considered. ML systems require many examples (data samples) to build accurate surrogate models, and HPC systems are designed to perform as few as possible simultaneous instances of very complex tasks. This underlying tension between ML and HPC requires the problem of multiple simulations to be solved, unfortunately HPC is optimized for a few, meaning that the creation of HPC simulation datasets used to train ML models must be done with care. Standard HPC workflow tools may not be the most effective way to make the large simulation datasets required to train ML models. Also, batch scheduling systems are typically not designed to run thousands or millions of simulations. Parallel file systems can degrade performance when presented with a large number of simultaneous reads and writes that overload the metadata servers. Dynamically loaded shared objects can pose similar problems. Essentially, creating ML-ready HPC simulation datasets requires workflow technology that can efficiently coordinate asynchronous heterogeneous simulation tasks at scales well beyond the design operation of HPC systems [14].

Taking all this into account and being aware that this is the beginning of an arduous road, it should be emphasized that there are already several works and articles that can boast interesting results combining ML components with the classic HPC simulation.

Fundamentally these methods have been used to replace, accelerate, or enhance existing solvers via AI/ML solution. These methods are based on the fact that solvers compute a set of iterations to achieve the convergence state of the simulated

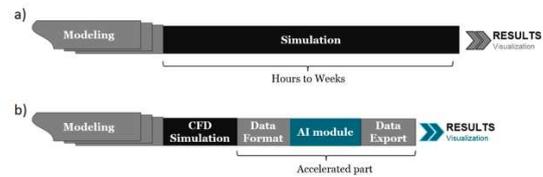


Fig. 6. Scheme of replacing classical solver via ML enabled solution

phenomenon. These methods are based on the assumption of the convergence of ML models on the basis of several initial iterations generated by the classic solver. This way, intermediate iterations do not have to be computed to get the final result, and therefore the time needed to solve is significantly reduced. Fig. 6 shows the simplified scheme of replacing classical solver via ML enabled solution for a CFD workload [15].

The presented approach includes the initial results computed by the CFD solver and the AI-accelerated part executed by the proposed AI module. The CFD solver produces results sequentially, iteration by iteration, where each iteration produces intermediate results of the simulation. All intermediate results wrap up into what is called the simulation results. The proposed method takes a set of initial iterations as an input, sends them to AI module, and generates the final iteration of the simulation. The AI module consists of three stages:

- data formatting and normalization,
- prediction with AI model (inference),
- data export

The advantage of this method is that it does not require to take into account a complex structure of the simulation, but focus on the data. Such an approach lowers the entry barrier for new adopters compared with other methods, such as a learning aware approach [15], which is based on the mathematical analysis of solver equations [15]. The AI-accelerated simulation is based on supervised learning, where a set of initial iterations is taken as an input and returns the last iteration. For simulating the selected phenomenon with conventional non-AI approach, it is required to execute 5000 iterations. At the same time, only the first iterations create the initial iterations that produce input data for the AI module. The accelerated part utilized a dedicated part of the system specialized for ML workloads, when CFD simulation portion uses HPC portion of the system. The proposed approach to accelerating CFD simulation allows shortens the simulation time almost ten times compared to using only a conventional CFD solver. The proposed AI / ML module uses 9.6% of the initial solver iterations and predicts a convergent state with an accuracy of 92.5% [15].

Undoubtedly, the 10-fold reduction in simulation time is impressive, and there are also studies in which this type of approach provides up to a 100-fold improvement in simulation related to protein folding. Over 100 times faster protein folding and 1.6 times more simulations per time unit, improving resource utilization compared to the classic HPC solver, even

more interesting is that this type of approach is not used as an experiment, but is running on platforms and workloads in production [16].

In parallel to the hybrid approach that tries to combine HPC solutions with ML, there are several initiatives that use typical ML frameworks to solve classic HPC problems. This approach can be seen in the attempt to use TensorFlow to solve problems typical for HPC. TensorFlow was designed for creating ML applications, but it must be noted that it reduces the difficulty of programming accelerators in distributed environments, avoiding the low-level programming interfaces typical for HPC, such as CUDA, OpenCL and MPI. This concept allows the use of cloud systems with accelerators such as GPU, TPU, IPU, but also allows the use of typical HPC supercomputers. While TensorFlow was originally designed to solve machine learning problems, it can be generalized to solving a much wider range of numerical problems. Although TensorFlow is not a typical library for numerical computation, there are several examples of using TensorFlow to solve them. TensorFlow is a complete development environment with a high-level API that allows easy development and implementation of distributed algorithms without the need for in-depth knowledge of concepts such as CUDA and MPI. Although TensorFlow is a dynamically developed platform that can be used to create HPC applications on supercomputers with accelerators, it does not seem to be a good candidate to replace classic solvers due to the mismatch of the hardware architecture typical for ML (e.g. TPU or IPU) solutions, different from those for HPC [17].

Considering the proposed hybrid system architecture proposed in section IV where the ML component was based on the IPU-M2000, it should be noted that there are significant limitations in using the IPU for typical HPC tasks. Firstly, Poplar graphs are static, making it difficult to implement techniques such as dynamic grids and adaptive mesh refinement. Secondly, the graph compile time is very high compared to compilation of typical HPC kernels. It is important to emphasize that for small problems, graph compilation may take longer than executing the resulting programs. Thirdly, code developed for the IPU is not portable to other platforms. Fourth and most importantly, the IPU is limited to 32-bit precision, which is absolutely the biggest obstacle for some HPC scientific applications.

VI. CONCLUSIONS

The horizon of Exascale HPC projects is a set of challenges and opportunities. On the one hand, HPC methods and platforms are becoming general and necessary for scientific advances. On the other, classical HPC computations are reaching limits. The HPC community confidently expected that as long as improvements in hardware performance were promising, traditional simulation-based methods would continue to deliver improved performance. The ramifications of this belief are obvious, achieving productivity gains is becoming increasingly difficult, while at the same time requiring significant and unsustainable, investment in software and algorithmic

reformulation. However, it is clear that traditional simulations may not represent the optimal approach for Exascale systems and for the next generations of supercomputers, which could consequently lead to complete stagnation.

The new discussed hybrid HPC and machine learning model enables a new approach to performance, scalability and execution time. In this new paradigm, a specialized ML system in conjunction with classical simulation replaces single large units, which requires both hierarchical and multitasking parallelism. The current research trend shows that hybrid systems, which are a combination of ML and HPC solutions, outperform simulation based approaches. The exact optimal point or intersection point is not trivial: it will be application specific, will depend on the complexity of the learned models, the amount of the data, and the effectiveness and cost of the simulation among others. However, the fundamental idea, that surrogate learned models will represent effective performance improvements over traditional simulations, is powerful and is an important generalization of the multi-scale, coarse-grained approaches used in many disciplines of science such as: computation fluid dynamics simulation, molecular science, climate change, materials simulation, and many others.

Presented details of surrogate system based on machine learning approach with high performance computing infrastructure that is widely available will have important implications for the cyberinfrastructure developed and deployed for the science of tomorrow. While a great deal of effort has gone into making the most of the HPC infrastructure, this article only describes a few fasteners and construction methods that seem easy to implement. From a future system architecture perspective ML methods have and will have an increasingly visible and important role in smarter computational systems. The main reason for the success of such techniques is that they offer simple, scalable and fairly general means to deal with high-dimensional, scientific datasets [14].

While new hybrid systems push the boundaries of effective simulation, this work demonstrates above all that a confluence of disaggregated ML components and traditional HPC technologies represents a promising path towards the realization of next generation integrated scientific computing.

REFERENCES

- [1] <https://top500.org/statistics/list/>
- [2] Geoffrey Fox, James A. Glazier, JCS Kadupitiya, Vikram Jadhao, Minje Kim, Judy Qiu, James P. Sluka, Endre Somogyi, Madhav Marathe, Abhijin Adiga, Jiangzhuo Chen, Oliver Beckstein, and Shantenu Jha. "Learning Everywhere: Pervasive machine learning for effective High-Performance computation: Application background". Technical report, Indiana University, February 2019. [http://dsc.soic.indiana.edu/publications/Learning Everywhere.pdf](http://dsc.soic.indiana.edu/publications/Learning%20Everywhere.pdf).
- [3] Geoffrey Fox, Shantenu Jha, "Understanding ML driven HPC: Applications and Infrastructure", Invited talk to "Visionary Track" at IEEE eScience 2019.
- [4] Jeff Dean. "Machine learning for systems and systems for machine learning". In Presentation at 2017 Conference on Neural Information Processing Systems, 2017.
- [5] Satoshi Matsuoka. "Post-K: A game changing supercomputer for convergence of HPC and big data" / AI. Multicore 2019, February 2019.
- [6] Kadupitiya Kadupitige. "Intersection of HPC and Machine Learning". ENGR-E 687 IND STUDY INTEL SYS: FINAL REPORT

- [7] <https://docs.graphcore.ai/projects/ipu-overview/en/latest/about-ipu.html>
- [8] Leslie G. Valiant. 1990. "A bridging model for parallel computation". *Commun. ACM* 33, 8 (August 1990), 103-111.
- [9] <https://www.graphcore.ai/products/ipu>
- [10] Zhe Jia, Blake Tillman, Marco Maggioni, Daniele Paolo Scarpazza, "Dissecting the Graphcore IPU Architecture via Microbenchmarking", <https://arxiv.org/abs/1912.03413>
- [11] Ion Stoica, Dawn Song, Raluca Ada Popa, David Patterson, Michael W. Mahoney, Randy Katz, Anthony D. Joseph, Michael Jordan, Joseph M. Hellerstein, Joseph Gonzalez, Ken Goldberg, Ali Ghodsi, David Culler, Pieter Abbeel. "A Berkeley View of Systems Challenges for AI". EECS Department. University of California, Berkeley. Technical Report No. UCB/EECS-2017-159. October 16, 2017.
- [12] Karl Freund, Patrick Moorhead. "The Graphcore Second-Generation IPU". <https://moorinsightsstrategy.com/research-paper-the-graphcore-second-generation-ipu/>
- [13] Thorsten Kurth, Sean Treichler, Joshua Romero, Mayur Mudigonda, Nathan Luehr, Everett Phillips, Ankur Mahesh, Michael Matheson, Jack Deslippe, Massimiliano Fatica, Prabhat, Michael Houston. "Exascale Deep Learning for Climate Analytics". Super Computing Conference November 11-16, 2018, Dallas, TX, USA
- [14] J. Luc Peterson, Ben Bay, Joe Koning, Peter Robinson, Jessica Semler, Jeremy White, Rushil Anirudh, Kevin Athey, Peer-Timo Bremer, Francesco Di Natale, David Fox, Jim A. Gaffney, Sam A. Jacobs, Bhavya Kailkhura, Bogdan Kustowski, Steven Langer, Brian Spears, Jayaraman Thiagarajan, Brian Van Essen, Jae-Seung Yeom. "Enabling Machine Learning-Ready HPC Ensembles with Merlin". Lawrence Livermore National Laboratory, Livermore, California 94550, USA. <https://arxiv.org/pdf/1912.02892.pdf>
- [15] Krzysztof Rojek, Roman Wyrzykowski, Pawel Gepner. "AI-Accelerated CFD Simulation Based on OpenFOAM and CPU/GPU Computing" International Conference on Computational Science -2021
- [16] Alexander Brace, Hyungro Lee, Heng Ma, Anda Trifan, Matteo Turilli, Igor Yakushin, Todd Munson, Ian Foster, Shantenu Jha, Arvind Ramanathan. "Achieving 100X faster simulations of complex biological phenomena by coupling ML to HPC ensembles". <https://arxiv.org/abs/2104.04797>
- [17] Steven W. D. Chien, Stefano Markidis, Vyacheslav Olshevsky, Yaroslav Bulatov, Erwin Laure, Jeffrey S. Vetter. "TensorFlow Doing HPC". <https://arxiv.org/abs/1903.04364>