Steering Customized AI Architectures for HPC Scientific Applications

Hatem Ltaief, Yuxi Hong, Adel Dabah, Rabab Alomairy, Sameh Abdulah, Chris Goreczny, Pawel Gepner, Matteo Ravasi, Damien Gratadour, David Keyes

▶ **To cite this version:**

Hatem Ltaief, Yuxi Hong, Adel Dabah, Rabab Alomairy, Sameh Abdulah, et al.. Steering Customized AI Architectures for HPC Scientific Applications. International Supercomputing Conference, May 2023, Hamburg, France. pp.125-143, 10.1007/978-3-031-32041-5_7 . hal-04286101

HAL Id: hal-04286101
https://hal.science/hal-04286101v1

Submitted on 15 Nov 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Steering Customized AI Architectures for HPC Scientific Applications

Hatem Ltaief[1], Yuxi Hong[1], Adel Dabah[1], Rabab Alomairy[1], Sameh Abdulah[1], Chris Goreczny[2], Pawel Gepner[3], Matteo Ravasi[1], Damien Gratadour[4], and David Keyes[1]

[1]Extreme Computing Research Center
Division of Computer, Electrical, and Mathematical Sciences and Engineering
King Abdullah University of Science and Technology
Thuwal, Jeddah 23955 Saudi Arabia
Hatem.Ltaief, Yuxi.Hong, Adel.Dabah.1, Rabab.Alomairy, Sameh.Abdulah, Matteo.Ravasi, David.Keyes@kaust.edu.sa
[2]Graphcore, Poland
chrisgo@graphcore.ai
[3]Warsaw University of Technology, Poland
pawel.gepner@pw.edu.pl
[4]Paris Observatory, France
damien.gratadour@obspm.fr

Abstract. AI hardware technologies have revolutionized computational science. While they have been mostly used to accelerate deep learning training and inference models for machine learning, HPC scientific applications do not seem to directly benefit from these specific hardware features unless AI-based components are introduced into their simulation workflows, for instance, as a replacement of their numerical solvers. This paper proposes to take another direction in an attempt to democratize customized AI architectures for HPC scientific computing. The main idea consists in demonstrating how legacy applications can leverage these AI engines after a necessary algorithmic redesign. It is critical that the resulting software implementations map onto the underlying memory-austere hardware architectures to extract the expected performance. To facilitate this process, we promote the matricization technique for restructuring codes (1) by exploiting data sparsity via algebraic compression and (2) by expressing the critical computational phases in terms of tile low-rank matrix-vector multiplications (TLR-MVM) and batch matrix-matrix multiplications (batch GEMM). Algebraic compression enables to reduce memory footprint and to fit into small local cache/memory, while batch execution ensures high occupancy. We highlight how we can steer the Graphcore AI-focused Wafer-on-Wafer Intelligence Processing Units (IPUs) to deliver high performance for both operations. We conduct a performance benchmarking campaign of these two matrix operations that account for most of the elapsed times of four real applications in computational astronomy, seismic imaging, wireless communications, and climate/weather predictions. We report bandwidth and execution rates with speedup factors up to 150X/14X/25X/40X, respectively, on IPUs compared to other systems.

Keywords: BLAS for Graphcore IPU, Low-rank matrix computations, Batch matrix operations, HPC scientific applications.

1 Introduction

Heterogeneity is ubiquitous in today's hardware landscape. From large distributed data centers supporting cloud computing to on-premise HPC Supercomputers, the scientific community has witnessed major deployments of mainstream system configurations composed of CPU hosts (e.g., x86/ARM) with accelerator/vector devices. In fact, the adoption of hardware heterogeneity is a clear matter and smartphones equipped with a myriad of specific hardware features (with a neural engine, graphic cores, processing cores, etc.) may represent the ultimate example, which may indicate a pathfinder of where high-end HPC architectures may be heading. While this hardware heterogeneity trend legitimately raises serious concerns on general software development, productivity and sustainability, the advancements of AI-focused hardware technologies have been tremendous during the last decade. These have been supported by a business market continuously expanding in size with applications in natural language processing (with transformers) and computer vision (with convolutional neural network).

This paper demonstrates the capabilities of AI-focused architectures in solving some of the HPC grand-challenge scientific problems in computational astronomy [28], seismic imaging [24, 34], wireless communications [10], and climate/weather predictions [5, 6, 11, 13]. In particular, we extend Graphcore AI-focused Wafer-on-Wafer Intelligence Processing Units (IPUs) functionalities [20, 21, 38] to address the computational challenges raised by the aforementioned applications with regards to real-time constraints and memory-bound mode of execution. The programming efforts required to steer these customized AI accelerators for supporting HPC scientific applications are part of a general call for action. Indeed, while it is true that fast matrix engines may not be inherent to real applications for which sustained bandwidth is the main metric for performance [17], algorithmic innovations are key to make compatible HPC workloads with the underlying AI architectures. This necessitates to express the algorithms in terms of matrix structures to be compliant with the specific features provided by hardware accelerators (e.g., NVIDIA Tensor Cores). Although this may sometimes come at the price of performing more floating-point operations (flops), the computational power of the fast matrix engines may compensate the flops increase. Here, we instead redesign the algorithms that drive the simulations of these four real applications by means of tile low-rank matrix-vector multiplication (TLR-MVM) for the computational astronomy and seismic imaging applications and batch matrix-matrix multiplications (batch GEMM) for the wireless communication and climate/weather applications. The former operation actually reduces memory footprint using algebraic compression to fit in memory-austere environment of IPUs and leverage the high bandwidth of local on-processor memory, while improving time complexity. The latter operation permits to cast original memory-bound operations into batch GEMM operations

to improve the hardware occupancy. In particular, MVM accounts for 90% of the real-time controller required on major ground-based deployed telescopes (e.g., the Very Large Telescope [4], the Keck Observatory [2], the Subaru telescope [3], the European Extremely Large Telescope [1], etc.) to compensate for the atmospheric turbulence. For the seismic imaging application, TLR-MVM accounts for 90% of the total elapsed time, as highlighted in Figure 1 of [24]. As for the batch GEMM kernel, it accounts for 80% of the total elapsed time of the wireless communication [10,15] as well as the climate/weather applications [13,14]. These algorithmic changes turn out to be key in extracting performance across various architectures [11, 13, 24, 28], but most importantly, makes IPUs (and potentially similar wafer-on-wafer chip technologies with limited on-chip memory) and their resource disaggregation compatible with these HPC scientific applications.

We develop TLR-MVM and batch GEMM operations on IPUs and report time-to-solution, sustained bandwidth, and execution rate. The two latter metrics permit to assess how some of the memory-bound workloads on standard x86 or GPUs can translate into compute-bound mode of operation (from an absolute performance perspective) once deployed on IPUs. We compare our implementations against other hardware architectures and highlight the performance superiority of our numerical algorithms. We achieve on IPUs speedup factors up to 150X/14X/25X/40X for computational astronomy, seismic imaging, wireless communication, and climate/weather predictions, respectively, against a myriad of hardware systems. These speedups correspond to the performance improvement of the most time-consuming kernels from these applications that are offloaded to IPUs, while the remaining ones run on the host, similar to the hybrid CPU/GPU trend observed in the HPC community.

The remainder of the paper is as follows. Section 2 presents related work and list our main contributions. Section 3 recalls the batch execution model and the TLR-MVM algorithm, while emphasizing on the importance of the matrization when designing numerical algorithms to remain on par with the AI hardware evolution. We present the Graphcore IPU hardware technology in Section 4. Section 5 describes the four HPC scientific applications of interest in this paper. We provide implementations details of our numerical algorithms in Section 6. Section 7 reports the performance results in time-to-solution, sustained bandwidth, and execution rate obtained on IPUs and compare our implementation against other hardware architectures. Section 8 discusses current IPU hardware limitations and gives some perspectives moving forward to further steer IPUs as a general-purpose chip. We conclude in Section 9.

2 Related Work and Research Contributions

Leveraging AI-focussed hardware architectures for general-purpose HPC workloads is still at its infancy due to the significant efforts it requires to map the existing numerical algorithms on these chips, originally designed for machine learning workloads (i.e., training and inference) as studied in [9, 26]. Previous work for accelerating breadth-first graph traversals on IPUs [12] have demonstrated

performance improvement for the class of graph algorithms. Stencil computations for solving the 3D wave equation using finite-difference method in seismic imaging applications have been ported into Graphcore IPU [26] and Cerebras Wafer-Scale Engine (WSE-2) [25]. The authors in the latter work rely on a localized communication strategy to mitigate internal data movement overheads, while ensuring data locality for maximum bandwidth extraction from local flat on-chip memory. This matrix-free algorithmic approach that represents the core engine for PDE solvers is friendly to Graphcore and Cerebras hardware technologies thanks to its minimal memory footprint.

In this paper, we revisit the numerical algorithms of real HPC applications, as originally introduced in computational astronomy [28], seismic imaging [24, 34], wireless communications [10], and climate/weather predictions [5, 13]. We integrate low-rank matrix compressions and batch executions to reconcile them with IPUs hardware design. Based on the numerical kernel primitives used for machine learning on IPUs, we develop these operations by composing higher-level APIs for linear algebra operations, i.e., Level-2 and Level-3 BLAS routines, that were not available natively on the vendor-optimized numerical library. We further tune memory accesses of our algorithms on IPUs and achieve significant performance improvement.

We emphasize the three main contributions of this paper: (1) the matricization approach that enables these HPC applications, otherwise intractable, to exploit IPUs by casting computations on compressed data structures, (2) the batch mode of execution to maintain high hardware utilization, and (3) the reported significant performance speedups that may further democratize AI hardware and accelerate their adoption into the wide HPC application landscape on heterogeneous environments.

3 Batching/Compression or Why Matricization Matters?

We are interested in leveraging IPUs computational and throughput capabilities to accelerate HPC scientific applications that rely on matrix formulations. We employ batched kernel executions to map the matrix operation onto IPUs' local memory and address the resource disaggragation challenge. In addition, if the dataset is large and do not fit on the chip, we exploit Tile Low-Rank (TLR) matrix approximation, an algorithmic technique that consists in splitting the matrix operator into tiles and compressing them using an algebraic method of choice (e.g., rank-revealing QR, randomized SVD, etc.), while enabling matrix algebra on the compressed data structures. Figures 1-6 highlight the compression procedure for a 4×6 tiled matrix followed by the Matrix-Vector Multiplication (MVM) applied to its compressed form. We refer to [6, 7, 28] for more technical details. The approximation error introduced is controlled by an accuracy threshold that maintains the application's numerical integrity [5, 13, 24, 34].

Matricization is a possible approach in maintaining numerical algorithms on par with the AI hardware evolution so that the specific hardware features (e.g., fast matrix engines with support for mixed-precision computations) can be eas-

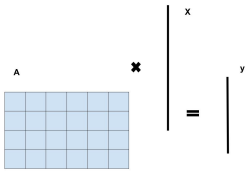


Fig. 1: Dense MVM.

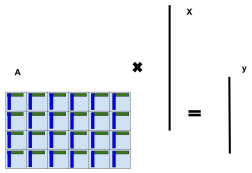


Fig. 2: Compress.

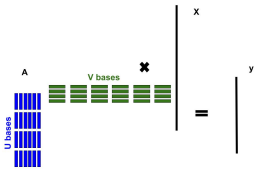


Fig. 3: Stack the bases.

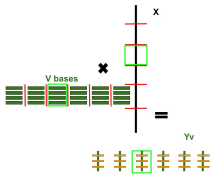


Fig. 4: V-Batch of MVM.

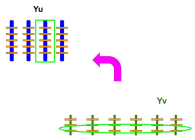


Fig. 5: Slicing V->U.

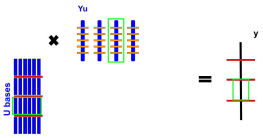


Fig. 6: U-Batch of MVM.

ily integrated and adopted by HPC scientific applications. There is no free lunch and matricization may not be straightforward for all applications. However, when possible, the redesigning efforts may be worth it and these efforts are usually upfront. The main benefits of matricization are twofold: (1) significant performance improvement in bandwidth for memory-bound codes and higher execution rates for compute-bound kernels thanks to dedicated matrix engines (e.g., Intel AMX, NVIDIA Tensor Cores, Graphcore AMP), while being on par with the overall hardware evolution, and (2) high user-productivity when deploying on new hardware architectures across vendors.

4 The Graphcore IPU Hardware Technology

4.1 Architecture Principles and Hardware Details

The Bulk Synchronous Parallel (BSP) model forms the basis for the hardware architecture of Graphcore's Bow Intelligence Processing Unit (IPU) and Poplar graph framework software. The BSP model is fundamental to the operation of IPU processors, which use this parallel computing scheme to schedule data processing and exchange operations. BSP involves a three-step process alternating compute, communication and data synchronisation.

Asynchronous computation. Each process performs local computations using only local memory. This phase does not involve any communication between processes.

Communication. Data is exchanged by the processes and each process may communicate with its target counterpart. In addition to exchanging intermediate computation results, processes may also engage in remote direct memory access in which they request access to data from remote memories. This remote data is then received in a subsequent communication phase. Each process can therefore

access other local memory as a remote memory, effectively enabling it to retrieve any memory from the entire aggregate system memory.

Synchronization. The synchronization phase acts as a check point or barrier. Once a process reaches this phase, it will only continue to the next phase once all processes have reached this check point. This stage does not involve any computation or communication unless the barrier itself specifically requires this.

In terms of its hardware architecture, the IPU is defined as a massively parallel, distributed memory, multiple-instruction, multiple data (MIMD) processor. The IPU has been designed from the ground up to process machine learning algorithms, with explicit programming instructions. The IPU's tile Instruction Set Architecture-ISA [37] comprises of hardware elements such as Accumulating Matrix Product-AMP units (i.e., dot product) and Slim Convolution Units-SLICs, which enable the IPU to complete up to 64 multiply-add instructions per clock cycle. These AMP units are eventually used to compose the necessary kernels and accelerate the HPC applications studied in this paper.

4.2 Programming Model and Poplar Development Kit

Graphcore's Poplar software is designed alongside the IPU to serve as a programming interface. Poplar is a graph programming frameworks that enables direct programming in Python and C++, building on the capability of C++ to form a new IPU operation model founded on three elements – vertices, computation graphs and control programs. The IPU computation graphs define the input/output relationship between variables and operations. Within the computation graph, there is the tensor variable (i.e., the variables in the graph), the compute tasks (vertices) and the edges that connect them. In terms of the tensor variable, data is stored in the graph in fixed-size multi-dimensional tensors. A vertex is a specific task to be performed and the edges determine which variable elements the vertex should process. A vertex can connect to a single element or multiple elements. A codelet is associated with every vertex: this is a piece of code that defines the inputs, outputs and internal state of a vertex. The codelet is implemented in standard C++11 [26]. Finally, we have the control program, which organizes the selection of processors, loads compiled graphs into the hardware and then executes graph programs. This includes the mapping of data transfers between the IPU and the host, memory structures, and initiating transfers. As soon as the program has been implemented, all the code and data structures required to run the program sit in the IPU's distributed memory [26]. Thanks to the control programs, the appropriate vertices can be executed. On top of low-level Poplar framework, Graphcore provides a PopLibs C++ library that contains higher-level mathematical and machine-learning functions. These underlie the Graphcore implementation of industry-standard ML frameworks, such as TensorFlow and PyTorch, but can also be used for other purposes. The massive parallelism and memory locality of IPU processor is well abstracted by PopLibs. For large data operations like matrix-matrix multiplications, Graphcore software handles the work distribution between IPU tiles to keep even memory and compute utilisation across all cores for load balancing purposes.

5 HPC Scientific Applications

This section describes the background of four major HPC applications in computational astronomy [28], seismic imaging [24, 34], wireless communications [10], and climate/weather predictions [5, 11, 13]. We identify the necessary algorithmic changes before deploying the most time-consuming computational kernels on Graphcore IPUs.

5.1 Adaptive Optics in Computational Astronomy

Using Deformable Mirrors (DM), arranged in closed-loop feedback control with wavefront sensors (WFS) in Adaptive Optics (AO) systems [16], correcting for optical aberrations introduced by atmospheric turbulence, giant optical telescopes are able to acquire sharp high-contrast images of faint and distant targets. As shown in Fig. 7, the real-time controller (RTC) [18] is responsible for interpreting measurements from WFS into commands to the DM actuators, adapting in real-time to the rapidly changing atmospheric turbulence conditions. Thanks to advances in computing, WFS and DM technologies, AO systems are becoming more capable and can be deployed on today's largest ground-based telescopes. However, classical AO correction is only valid in a very small patch of sky. Multi-Conjugate Adaptive Optics (MCAO) solves this by using a series of DMs to compensate the turbulence in volume [35]. This increase in AO complexity inevitably translates into a significant additional load on the RTC sub-system.

A robust control scheme for AO is based on regular dense Matrix-Vector Multiplication (MVM), which has a low arithmetic intensity and is thus limited by sustained memory bandwidth. Assuming a typical atmosphere coherence time of a few ms and in order to compensate for most of the accessible frequency content of the turbulence, the AO RTC latency should be kept below 250 μs [22]. This 250 μs specification leads to a memory bandwidth requirement of about 1600 GB/s for single precision floating-point MVM, i.e., several times larger than what is achievable on current high-end dual-socket CPU servers and typically even higher than on a single high-end GPU. Can the IPUs stand as an alternative hardware solution to outsmart the atmospheric turbulence and meet the real-time computational challenges of ground-based giant optical telescopes? A batched dense MVM is necessary to evenly split the matrix across local memories of the IPUs and match the underlying IPU hardware architecture.

5.2 Seismic Processing and Imaging

Reflection seismology is a remote sensing technique that uses principles of wave propagation to image the Earth's subsurface from reflected seismic waves. Most algorithms for processing and imaging of seismic data, originally developed in the 80's and 90's, operate on individual shot gathers (i.e., ensemble of traces recording the energy produced by a single source at the time); this naturally lends to embarrassingly parallel implementations that loop over the dataset once per processing step. A paradigm shift has however emerged in the early 2000s, with

a large portion of modern algorithms relying on wave-equation, inversion-based formulations [42]: such algorithms require repeated access to the entire seismic data in order to evaluate the so-called Multi-Dimensional Convolution (MDC) operator and its adjoint and solve an underlying inverse problem. Examples of such a kind are closed-loop SRME [27], estimation of primaries by sparse inversion [23], multi-dimensional deconvolution [8, 33, 39], and Marchenko-based processing and imaging [30, 31, 40, 44].

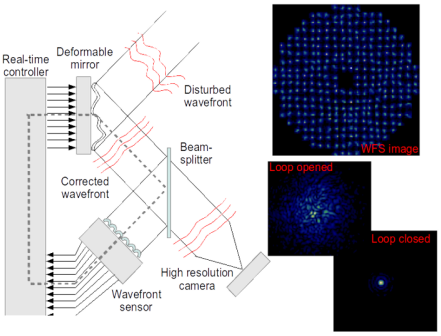


Fig. 7: End-to-end AO simulation [19] relies on MVM in the RTC to outsmart the atmospheric turbulence.

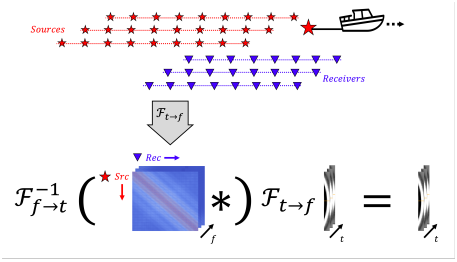


Fig. 8: From seismic acquisition to processing data for the MDC operator that involves TLR-MVM operations.

From a practical standpoint, the MDC operator can be viewed as the chain of the following three linear operations: a Fast Fourier Transform (FFT) to convert the input seismic data from the time to the frequency domain, followed by a batched dense Matrix-Vector Multiplication (MVM) with the frequency representation of the kernel of the MDC operator, and by an Inverse FFT (IFFT) to bring back the output to the time domain (see Fig. 8 and [24, 32, 34] for more details). Whilst the kernel of the MDC operator varies from application to application, its sheer size renders the batched MVM to be the main computational bottleneck of all of the above mentioned algorithms. As discussed in Section 3, TLR-MVM can be used in an attempt to reduce both the memory requirements and computational cost. When performing MDC with dense frequency matrices, the arrangement of sources (along the rows of each matrix) and receivers (along the columns of each matrix) can be arbitrary as long as they remain consistent with that of the input vector. Such reordering becomes much more relevant in the context of TLR algebraic compression as it may lead to better or worse block compression capabilities. Following [34], the Hilbert space-filling curve algorithm has been chosen as the best performing re-arrangement approach. We refer to [24, 34] for a detailed study on the impact of accuracy. Compression and reordering are key algorithmic aspects to consider, when deploying big data applications on hardware with limited memory capacities, e.g., IPUs.

5.3 Climate/Weather Prediction Applications

Geostatistical emulations for climate/weather prediction applications rely on computational statistics methods based on the maximum likelihood estimation. The optimization model requires solving a large system of linear equations. This involves a Cholesky factorization of the covariance matrix of dimension the number of geospatial locations, at every iteration of the optimization process. To reduce algorithmic complexity and memory footprint, we exploit the data sparsity structure of the operator and perform TLR matrix approximation based on algebraic compression, as originally introduced in [5, 6, 13, 14]. This necessitates the development of new kernels composed of several successive calls to BLAS/LAPACK functions (e.g., QR/GEMM/SVD), including the most time-consuming, i.e., TLR-GEMM operating on thin-shaped pairs of U/V matrices using the lower part of the symmetric matrix. To increase hardware occupancy, a left-looking variant of the TLR-Cholesky is employed in [11], which then permits to expose opportunities for batched GEMM kernel executions. For instance, Fig. 9 shows the updates on the matrix tile in red that requires batched GEMMs involving the compressed tiles located in the green/yellow/blue (overlapped) regions. These algorithmic steps are critical to make IPU compatible with such big data applications. otherwise intractable.

5.4 Wireless Communications

The increased number of connected devices and data demand under extreme low latency puts today's base station under a huge burden. Massive Multiple-Input Multiple-Output (M-MIMO) technology uses hundreds of antennas at base-station to fulfills the requirement of next-generation networks in terms of data rate and service quality while supporting a huge number of connected devices. However, this technology suffers from high signal detection complexity and accuracy, which is critical for several applications, such as self-driving cars.

Indeed, reducing M-MIMO detection latency to meet the real-time requirement while guaranteeing good detection accuracy represents a challenging problem. Linear detection algorithms maintain low complexity. However, their lack of reliability cannot be accepted [43]. Optimal non-linear detection approaches [41] have high accuracy, but they are not scalable due to the M-MIMO exponential complexity. The multi-level detection approach proposed in [10, 15] is a promising scalable and accurate approach for M-MIMO detection problem. It iteratively extends a single path with several symbols within L levels until reaching a complete solution path with the shortest distance among all existing paths. These symbols represent the best combination of aggregating multiple levels. This technique increases the accuracy in terms of error rate performance since it uses coefficients from multiple levels to better distinguish the optimal path. As a result, the more levels used, the more confident we are in getting near-optimal solutions.

The computation of these distances can be casted in terms of small matrix-matrix multiplication operations (i.e., GEMM) with dimensions M=K=L and N

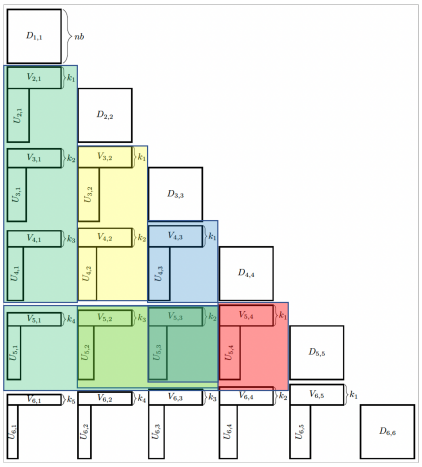


Fig. 9: Tile low-rank Cholesky factorization powered by a batch GEMM for climate applications.

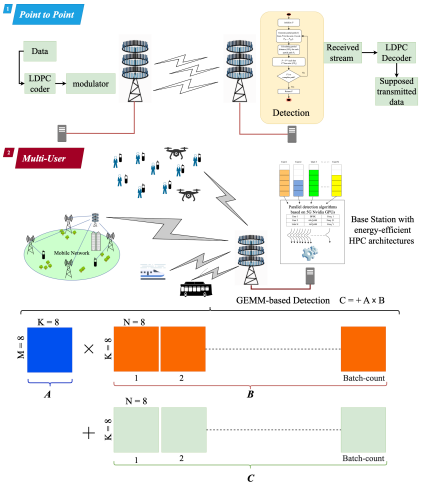


Fig. 10: Massive MIMO workflow powered by a batch GEMM detection for wireless communication.

the number of paths within the window. These well-established GEMM kernels increase the arithmetic intensity of the algorithm and may account up to 80% of the global execution time of the method. However, as highlighted in Fig. 10, the resulting matrix generated by this multi-level algorithm has a short and wide shape, which may prevent it from extracting the full hardware potential, especially in presence of disaggregated memory resources as in IPUs. The single short and wide GEMM must be redesigned into a batched GEMM, while mapping each B_i and C_i blocks along with A onto the local memory.

6 Implementation Details

In the case of the Graphcore Poplar SDK, all linear algebra operations like MVM and GEMM are exposed in form of the PopLibs C++ API that was built from the ground up as a foundation for AI frameworks such as PyTorch and TensorFlow. It is well optimized for this task and allows the user to leverage a high number of independent tiles without needing to manually split the workload among them. It is also based on the concept of computation graphs whereby all compute operations are first compiled into one or more graphs and only then run in this form on the IPU. Those two assumptions make the PopLibs API very different from the standard BLAS interface where parallelism is handled outside of BLAS and compute kernels are run as soon as they are called. It is not straightforward to create a translation layer of BLAS calls which the CPU implementation of TLR-MVM is dependent on to be used on the PopLibs API.

The logical equivalent of MVM and GEMM BLAS calls in FP32 in PopLibs is the matMul function. For the computational astronomy applications, the real

datasets fit in IPUs' local memory so a single call to PopLibs matMul function can be issued, while ensuring proper mapping is done onto the disaggregated memory resources to achieved the required throughput. For the seismic imaging application and its large datasets, the TLR-MVM algorithm comes to the rescue to reconcile the IPUs architectures with the application. TLR-MVM algorithm performs multiple MVM kernels on stacked tile columns (Fig. 4) and rows (Fig. 6), with in-between intermediate slicing phase (Fig. 5). PopLibs offers a matMulGrouped call that aggregates multiple independent MVM or GEMM into a single call and schedules all of them to be performed in parallel, distributed amongst IPU tiles. The TLR-MVM implementations on IPUs comes down to a sequence of three functions, as described in Algorithm 1: (1) perform batched MVM on the group of matrix-vector pairs where each pair consists of stacked tile column V_j and corresponding nb portion of the input vector x to get the set of output vectors Yv, (2) project/slice the set of output vectors Yv from V_j bases to U_j bases to get the set of output vectors Yu, and (3) perform batched MVM on group of matrix-vector pairs where each pair consists of stacked tile row U_i and corresponding output vector column of Yu. However, the dimensions in all multiplications must be equal. In TLR-MVM, each stacked tiles column/row can have different dimensions after matrix compression, as seen for the seismic imaging application. To leverage the parallelism capabilities of PopLibs, it is then necessary to make all dimensions equal to avoid overheads from stragglers due to the BSP model of IPU, as explained in Section 4. Therefore, as shown in Algorithm 2, we need to pad stacked tiles columns/rows with zeros at least up to the size of the biggest element. The actual size of padding is then determined empirically to deliver the best performance with acceptable overheads on memory utilization.

Algorithm 1: Poplar pseudo-code of TLR-MVM.

```
1: Yv = poplin::matMulGrouped(V,X) (i.e., batch MVM of V bases, see Fig. 4)
2: Yu = popops::multiSlice(Yv) (i.e., project from Yv to Yu via slicing, see Fig. 5)
3: Y = poplin::matMulGrouped(U,Yu) (i.e., batch MVM of U bases, see Fig. 6)
```

For the climate application, the large dense data-sparse matrix needs to be compressed first using TLR algebraic compression [6,13,14]. The TLR Cholesky factorization can then be redesigned in a left-looking variant [11] to further expose batched GEMM operations that account for most of the elapsed time. For the wireless communication application, the redesign of the detection algorithm into an efficient GEMM-based approach [10, 15] enables to leverage high throughput of customized hardware features for such a massively parallel operation. Both applications can offload their batched GEMM on IPUs by using PopLibs matMulGrouped function to address the computational and curse of dimensionality challenges for the former and to meet the real-time constraints for the latter by achieving high hardware occupancy.

Algorithm 2: Pseudo-code of the offline zero padding step.

Require: Compress A
Ensure: max = 0
for each tile column **do**
  if sum of ranks > max **then**
    max = sum of ranks
  end if
end for
for each tile row **do**
  if sum of ranks > max **then**
    max = sum of ranks
  end if
end for
mod = 200 (identified empirically)
max = max + mod - max % mod
for each stacked tile column V_i **do**
  append (max - sum of ranks) zeros
end for
for each stacked tile row U_i **do**
  append (max - sum of ranks) zeros
end for

7 Performance Results

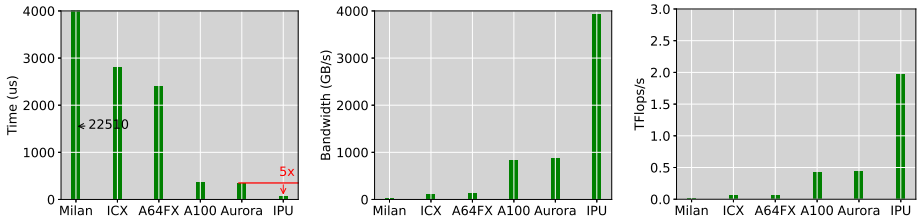
Vendor	Intel	AMD	Fujitsu	NEC	NVIDIA	Graphcore
Family	Cascade Lake	EPYC Milan	Primergy A64FX	SX-Aurora TSUBASA	Ampere GPU	IPU
Model	6248	7713	FX1000	B300-8	A100	Bow
Node(s)/Card(s)	1	1	16	8	1	1
Socket(s)	2	2	4	N/A	N/A	1
Cores	40	128	48	8	6912	1472
GHz	2.5	2.0	2.2	1.6	2.6	1.85
Memory Sustained BW	384GB DDR4 232GB/s	512GB DDR4 330GB/s	32GB HBM 800GB/s	48GB HBM2 1.5TB/s	40GB HBM2e 1.5TB/s	3.6GB 261TB/s
LLC Sustained BW	27.5MB 1.1TB/s	512MB 4TB/s	32MB 3.6TB/s	16MB 2.1TB/s	40MB 4.8TB/s	N/A
Compiler	Intel 19.1.0	GCC 7.5.0	Fujitsu 4.5.0	NEC 3.1.1	NVCC 11.0	POPLAR 2.6
BLAS library	Intel MKL 2020	BLIS 3.0.0	Fujitsu SSL II	NEC NLC 2.1.0	cuBLAS 11.0	N/A
MPI library	OpenMPI 4.0.3	OpenMPI 3.1.2	Fujitsu MPI 4.0.1	NEC MPI 2.13.0	NCCL 2.0	N/A
			x86 - ARM - Vector		GPU	
			MPI + OpenMP		CUDA	

Fig. 11: Hardware/software descriptions and programming models.

The experiments are carried on six architectures, i.e., Intel IceLake (codenamed ICX), AMD Epyc Milan (Milan), Fujitsu A64FX (A64FX), NEC SX-Aurora TSUBASA (Aurora), NVIDIA A100 GPU (A100), and Graphcore Bow IPUs (IPUs). A detailed hardware and software descriptions along with the programming models are illustrated in Fig. 11. We report performance from the median obtained out of 1000 runs on IPUs. All computations are performed using IEEE 754 FP32 arithmetic.

Figure 12 shows the performance of batched dense MVM for the astronomy application. The main numerical kernel, i.e., FP32 Level-2 BLAS MVM, is deployed on single IPU using the real datasets from the MAVIS flagship MCAO instrument for ESO's Very Large Telescope [36], which engenders batched MVM operations on a 5K X 20K matrix size [28] to be performed in real-time. Our batched dense MVM implementation on IPUs achieves 5X speedup factor against Aurora and up to 150X against Milan (DDR4 memory). In terms of absolute performance, our batched dense MVM implementation scores 2 Tflops/s for a

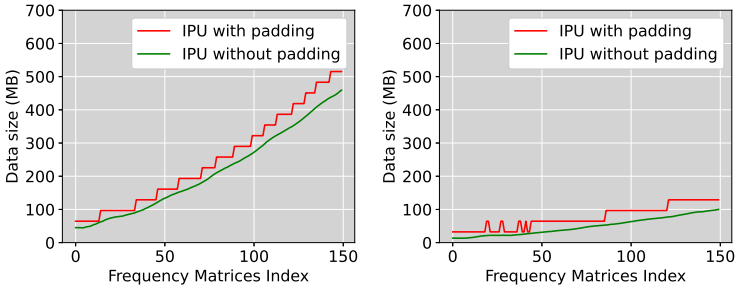
Level-2 BLAS operation that is usually limited in performance by the bus bandiwdth. To give a perspective, the obtained performance is equivalent to half of LINPACK benchmark (FP32) on the two-socket 26-core Intel IceLake system.



(a) Time-to-solution. (b) Sustained bandwidth. (c) Execution rate.

Fig. 12: Performance of Dense MVM for the astronomy application.

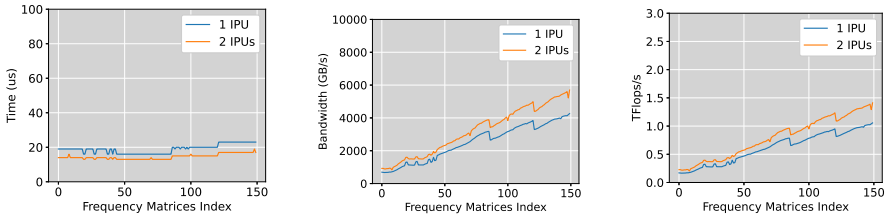
For the seismic imaging application, the batched dense MVM is not an option anymore for IPUs since the large matrix size in addition to the single complex precision do not allow the matrix to fit on the on-chip memory. Therefore, we have to use TLR algebraic compression on the matrix and deploy our TLR-MVM kernel. We design our single complex TLR-MVM into two FP32 TLR-MVM for handling the real and imaginary parts. With TLR matrix approximations, the matrices can now fit into the local memory of the IPUs. To further improve performance, we apply two optimizations: reordering and padding, as explained in Sections 5.2 and 6, respectively. The former reduces memory footprint and time complexity, while the latter ensures load balance on IPUs. Figure 13 shows the impact of padding (represented by the stairs shape since matrices are clustered into bins to mitigate the padding overheads) in terms of MB with limited overhead on all frequency matrices using the default and Hilbert ordering schemes.



(a) Normal ordering. (b) Hilbert ordering.

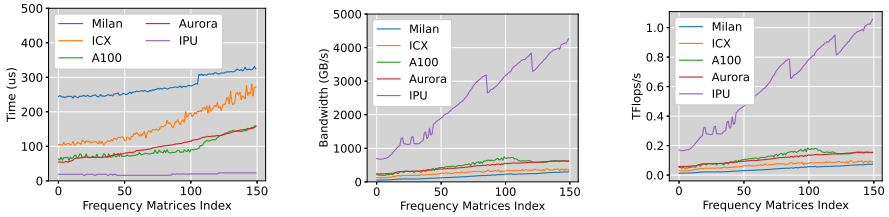
Fig. 13: Overhead (in MB) of padding for the seismic imaging application.

Figure 14 shows the performance of TLR-MVM for the seismic imaging application using Hilbert ordering on two IPUs. We do not show the slower performance obtained for the default ordering due to space limitation. The scalability on two IPUs is a bit limited but this is also expected due to the small memory footprint after applying Hilbert ordering that does not permit saturation.



(a) Time-to-solution. (b) Sustained bandwidth. (c) Execution rate.

Fig. 14: TLR-MVM performance for the seismic application on two IPUs.



(a) Time-to-solution. (b) Sustained bandwidth. (c) Execution rate.

Fig. 15: TLR-MVM performance comparisons for the seismic application.

Figure 15 highlights performance comparisons of TLR-MVM on 150 frequency matrices for the seismic imaging application against other hardware architectures. Our TLR-MVM implementation achieves up to 14X performance speedup against Milan. In terms of absolute performance, our implementation scores more than 1 Tflops/s for a kernel that is intrinsically memory-bound.

Figure 16 shows the performance of batched GEMM for wireless communication and climate prediction applications. We only compare against GPUs since x86/ARM/Vector are not meant for compute-bound kernels. The wireless communication batch size (M=N=K=8) comes from aggregating four tree levels, resulting in a real matrix A with eight rows and columns. The batch count, on the other hand, refers to the total number of possible combinations. For instance, aggregating four levels with 32-QAM modulation generates 8M combinations computed resulting in a batch count close to 1M. The batched

GEMM in the climate application needs to be grouped until each single matrix block fits the local memory, while ensuring an even workload distribution. Compared to [10] (but rerun on NVIDIA A100 with 1.1 Tflops/s) and [11] (results obtained on NVIDIA V100 with 1.1 Tflops/s), we achieve 25X and 40X for wireless communication and climate applications, respectively. By launching these kernels in batched mode, we activate all tiles on the IPUs, allowing high absolute performance, while preserving the integrity of the IPU hardware resource disaggregation.

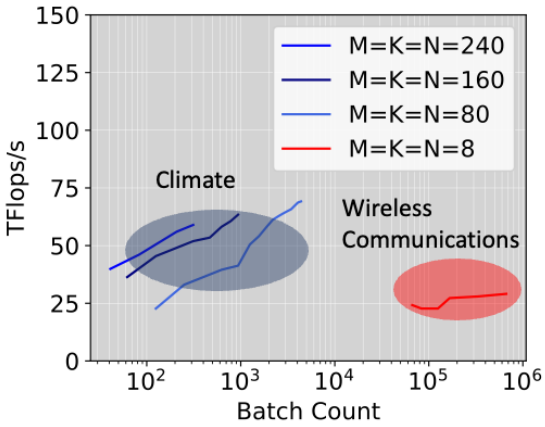


Fig. 16: Batched GEMM performance for MIMO and climate applications.

8 Limitations and Perspectives

While the paper demonstrates IPUs' capabilities, there are some areas for improvement for Poplar SDK, e.g., enabling support for a standard BLAS/LAPACK interface. Currently, porting HPC applications based on TLR-MVM kernels requires the developer to write a separate implementation for IPU that is fundamentally different from industry-standard solutions. This type of support is a challenging task to accomplish considering how the Poplar SDK is designed, however Graphcore is already working on delivering this. For instance, this will enable to run the compression phase on IPUs instead of the host and ensure the whole computational pipeline is resident on the chip. While this may not be a problem for seismic imaging application since the compression is needed only once upfront, it may raise performance bottlenecks for the climate/weather applications application that requires matrix factorization and solve at every iteration of the optimization procedure. One element that has not been mentioned yet is the graph compilation time. Poplar builds one compute graph that contains all the operations instead of running small compute kernels. This allows

for greater runtime performance as there is very little communication required between the x86 host and the IPU. It also allows Poplar to apply multiple graph-level optimisations that further improves performance, but at the cost of graph compilation time increase. In some cases, this can become problematic and reach minutes of x86 host time to perform a fraction of a second of compute on the IPU. Whereas it can work very well for AI tasks when the same set of operations is run thousands of times, this can become problematic for the one-time compute kernels which are commonplace in HPC.

9 Conclusion and Future Work

This paper presents necessary algorithmic techniques to make the Graphcore Bow IPUs compliant with state-of-the-art HPC scientific applications. Based on low-rank matrix approximations and batched matrix-matrix multiplication, we leverage the high bandwidth and throughput of IPUs and deliver high performance with four different applications that share common matrix algebra operations. We report speedup factors up to 150X/14X/25X/40X for computational astronomy, seismic imaging, wireless communication, and climate/weather predictions, respectively, against a myriad of hardware architectures. This highlights the need to pursue the matricization efforts to ensure HPC applications can keep up with latest AI hardware advancements. In terms of algorithmic innovation, algebraic compression and batched execution appear to be critical ingredients with a significant impact on performance and throughput, not only on IPUs as studied herein, but also on a myriad of hardware architectures from a relative performance perspective. For future work, we would like to explore FP16 for some of these HPC applications and demonstrate the applicability of mixed-precision computations [29].

References

1. The European Extremely Large Telescope (2023), `https://elt.eso.org`
2. The Keck Observatory (2023), `https://www.keckobservatory.org`
3. The Subaru Telescope (2023), `https://subarutelescope.org/en/`
4. The Very Large Telescope (2023), `https://www.eso.org/public/teles-instr/paranal-observatory/vlt/`
5. Abdulah, S., Ltaief, H., 0002, Y.S., Genton, M.G., Keyes, D.E.: ExaGeoStat: A High Performance Unified Software for Geostatistics on Manycore Systems. IEEE Trans. Parallel Distributed Syst **29**(12), 2771–2784 (2018), `http://doi.ieeecomputersociety.org/10.1109/TPDS.2018.2850749`
6. Akbudak, K., Ltaief, H., Mikhalev, A., Keyes, D.: Tile Low Rank Cholesky Factorization for Climate/Weather Modeling Applications on Manycore Architectures. In: 32nd International Conference on High Performance, Frankfurt, Germany. pp. 22–40. Springer (2017)
7. Amestoy, P., Ashcraft, C., Boiteau, O., Buttari, A., L'Excellent, J.Y., Weisbecker, C.: Improving multifrontal methods by means of block low-rank representations. SIAM Journal on Scientific Computing **37**(3), A1451–A1474 (2015)

8. Amundsen, L.: Elimination of Free-surface Related Multiples Without Need of a Source Wavelet. Geophysics **66**, 327–341 (2001), doi: 10.1190/1.1444912
9. Arcelin, B.: Comparison of Graphcore IPUs and Nvidia GPUs for Cosmology Applications (2021). https://doi.org/10.48550/ARXIV.2106.02465
10. Arfaoui, M.A., Ltaief, H., Rezki, Z., Alouini, M.S., Keyes, D.: Efficient Sphere Detector Algorithm for Massive MIMO Using GPU Hardware Accelerator. Procedia Computer Science **80**(C), 2169–2180 (2016). https://doi.org/10.1016/j.procs.2016.05.377
11. Boukaram, W., Zampini, S., Turkiyyah, G., Keyes, D.E.: H2OPUS-TLR: High Performance Tile Low Rank Symmetric Factorizations using Adaptive Randomized Approximation. CoRR **abs/2108.11932** (2021)
12. Burchard, L., Moe, J., Schroeder, D.T., Pogorelov, K., Langguth, J.: iPUG: Accelerating Breadth-First Graph Traversals Using Manycore Graphcore IPUs. In: High Performance Computing - 36th International Conference, ISC High Performance 2021, Virtual Event, June 24 - July 2, 2021, Proceedings. vol. 12728, pp. 291–309. Springer (2021). https://doi.org/10.1007/978-3-030-78713-4_16
13. Cao, Q., Abdulah, S., Alomairy, R., Pei, Y., Nag, P., Bosilca, G., Dongarra, J., Genton, M., Keyes, D., Ltaief, H., Sun, Y.: Reshaping Geostatistical Modeling and Prediction for Extreme-Scale Environmental Applications. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (2022), https://dl.acm.org/doi/abs/10.5555/3571885.3571888
14. Cao, Q., Pei, Y., Akbudak, K., Mikhalev, A., Bosilca, G., Ltaief, H., Keyes, D., Dongarra, J.: Extreme-scale Task-based Cholesky Factorization Toward Climate and Weather Prediction Applications. In: Proceedings of the Platform for Advanced Scientific Computing Conference. pp. 1–11 (2020)
15. Dabah, A., Ltaief, H., Rezki, Z., Arfaoui, M.A., Alouini, M.S., Keyes, D.: Performance/complexity Trade-offs of the Sphere Decoder Algorithm for Massive MIMO Systems. arXiv preprint arXiv:2002.09561 (2020). https://doi.org/10.48550/arXiv.2002.09561
16. Davies, R., Kasper, M.: Adaptive optics for astronomy. Annual Review of Astronomy and Astrophysics **50**(1), 305–351 (2012). https://doi.org/10.1146/annurev-astro-081811-125447
17. Domke, J., Vatai, E., Drozd, A., Chen, P., Oyama, Y., 0001, L.Z., Salaria, S., Mukunoki, D., Podobas, A., Wahib, M., Matsuoka, S.: Matrix Engines for High Performance Computing: A Paragon of Performance or Grasping at Straws? In: IPDPS. pp. 1056–1065. IEEE (2021). https://doi.org/10.1109/IPDPS49936.2021.00114
18. Ferreira, F., Sevin, A., Bernard, J., Guyon, O., Bertrou-Cantou, A., Raffard, J., Vidal, F., Gendron, E., Gratadour, D.: Hard Real-time Core Software of the AO RTC COSMIC Platform: Architecture and Performance. In: Schreiber, L., Schmidt, D., Vernet, E. (eds.) Adaptive Optics Systems VII. vol. 11448, p. 1144815. International Society for Optics and Photonics, SPIE (2020). https://doi.org/10.1117/12.2561244
19. Ferreira, F., Gratadour, D., Sevin, A., Doucet, N.: Compass: an Efficient GPU-based Simulation Software for Adaptive Optics System. In: 2018 International Conference on High Performance Computing & Simulation (HPCS). pp. 180–187 (2018). https://doi.org/10.1109/HPCS.2018.00043
20. Gepner, P.: Machine Learning and High-performance Computing Hybrid Systems, a New Way of Performance Acceleration in Engineering and Scientific Applications. In: 2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS). pp. 27–36 (2021). https://doi.org/10.15439/2021F004

21. Graphcore: Tile Vertex ISA (march 2022), https://docs.graphcore.ai/projects/isa/en/latest/_static/Tile-Vertex-ISA_1.2.3.pdf
22. Gratadour, D., Bernard, J., Doucet, N., Ferreira, F., Sevin, A., Biasi, R., Rigaut, F.: MAVIS Real-time Control System: a High-end Implementation of the COSMIC Platform. In: Schreiber, L., Schmidt, D., Vernet, E. (eds.) Adaptive Optics Systems VII. vol. 11448, p. 114482M. International Society for Optics and Photonics, SPIE (2020). https://doi.org/10.1117/12.2562082
23. van Groenestijn, G.J., Verschuur, D.J.: Estimating Primaries by Sparse Inversion and Application to Near-offset Data Reconstruction. Geophysics **74**, 1MJ–Z54 (2009), doi: 10.1190/1.3111115
24. Hong, Y., Ltaief, H., Ravasi, M., Gatineau, L., Keyes, D.: Accelerating Seismic Redatuming Using Tile Low-Rank Approximations on NEC SX-Aurora TSUBASA. Supercomputing Frontiers and Innovations **8** (2021), doi: 10.14529/jsfi210201
25. Jacquelin, M., Araya-Polo, M., Meng, J.: Scalable Distributed High-Order Stencil Computations. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (2022), https://dl.acm.org/doi/abs/10.5555/3571885.3571924
26. Jia, Z., Tillman, B., Maggioni, M., Scarpazza, D.P.: Dissecting the Graphcore IPU Architecture via Microbenchmarking (2019). https://doi.org/10.48550/arXiv.1912.03413
27. Lopez, G.A., Verschuur, D.: Closed-loop Surface-related Multiple Elimination and Its Application to Simultaneous Data Reconstruction. Geophysics **80**, V189–V199 (2015). https://doi.org/10.1190/geo2015-0287.1
28. Ltaief, H., Cranney, J., Gratadour, D., Hong, Y., Gatineau, L., Keyes, D.: Meeting the Real-Time Challenges of Ground-Based Telescopes Using Low-Rank Matrix Computations. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (2021). https://doi.org/10.1145/3458817.3476225
29. Ltaief, H., Genton, M.G., Gratadour, D., Keyes, D.E., Ravasi, M.: Responsibly Reckless Matrix Algorithms for HPC Scientific Applications. Computing in Science & Engineering **24**(4), 12–22 (2022). https://doi.org/10.1109/MCSE.2022.3215477
30. van der Neut, J., Vasconcelos, I., Wapenaar, K.: On Green's Function Retrieval by Iterative Substitution of the Coupled Marchenko Equations. Geophysical Journal International **203**, 792–813 (2015). https://doi.org/10.1093/gji/ggv330
31. Ravasi, M.: Rayleigh-marchenko Redatuming for Target-oriented, True-amplitude Imaging. Geophysics **82**, S439–S452 (2017). https://doi.org/10.1190/geo2017-0262.1
32. Ravasi, M., Vasconcelos, I.: An Open-source Framework for the Implementation of Large-scale Integral Operators With Flexible, Modern HPC Solutions - Enabling 3d Marchenko Imaging by Least-squares Inversion. Geophysics **86**, WC177–WC194 (2021). https://doi.org/10.1190/geo2020-0796.1
33. Ravasi, M., Vasconcelos, I., Curtis, A., Kritski, A.: Multi-dimensional Free-surface Multiple Elimination and Source Deblending of Volve OBC Data. 77th Conference and Exhibition, EAGE, Extended Abstracts (2015). https://doi.org/10.3997/2214-4609.201413355
34. Ravasi, M., Hong, Y., Ltaief, H., Keyes, D., Vargas, D.: Large-scale Marchenko Imaging With Distance-aware Matrix Reordering, Tile Low-rank Compression, pp. 2606–2610 (2022). https://doi.org/10.1190/image2022-3744978.1
35. Rigaut, F., Neichel, B.: Multiconjugate Adaptive Optics for Astronomy. Annual Review of Astronomy and Astrophysics **56**(1), 277–314 (2018).

https://doi.org/10.1146/annurev-astro-091916-055320, https://doi.org/10.1146/annurev-astro-091916-055320
36. Rigaut, F.e.a.: MAVIS Conceptual Design. In: Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, vol. 11447, p. 114471R (Dec 2020). https://doi.org/10.1117/12.2561886
37. Rojek, K., Wyrzykowski, R., Gepner, P.: AI-Accelerated CFD Simulation Based on OpenFOAM and CPU/GPU Computing. In: Computational Science – ICCS 2021: 21st International Conference, Krakow, Poland, June 16–18, 2021, Proceedings, Part II. p. 373–385. Springer-Verlag, Berlin, Heidelberg (2021). https://doi.org/10.1007/978-3-030-77964-1_29
38. Valiant, L.G.: A Bridging Model for Parallel Computation. Commun. ACM **33**(8), 103–111 (aug 1990). https://doi.org/10.1145/79173.79181
39. Vargas, D., Vasconcelos, I., Ravasi, M., Luiken, N.: Time-domain Multidimensional Deconvolution: a Physically Reliable and Stable Preconditioned Implementation. Remote Sensing **13**, 3683 (2022). https://doi.org/10.3390/rs13183683
40. Vargas, D., Vasconcelos, I., Ravasi, M., Sripanich, Y.: Scattering-based Focusing for Imaging in Highly-complex Media From Band-limited, Multi-component Data. Geophysics **Submitted** (2021). https://doi.org/10.1190/geo2020-0939.1
41. Viterbo, E., Boutros, J.: A Universal Lattice Code Decoder for Fading Channels. IEEE Transactions on Information Theory **45**(5), 1639–1642 (1999). https://doi.org/10.1109/18.771234
42. Wapenaar, C.P.A., Berkhout, A.J.: Elastic Wave Field Extrapolation: Redatuming of Single- and Multi-Component Seismic Data. Elsevier Science, Philadelphia (2014), https://www.elsevier.com/books/elastic-wave-field-extrapolation/berkhout/978-0-444-88472-5
43. Xie, Z., Short, R.T., Rushforth, C.K.: A Family of Suboptimum Detectors for Coherent Multiuser Communications. IEEE Journal on Selected Areas in Communications **8**(4), 683–690 (1990). https://doi.org/10.1109/49.54464
44. Zhang, L., Thorbecke, J., Wapenaar, K., Slob, E.: Transmission Compensated Primary Reflection Retrieval in the Data Domain and Consequences for Imaging. Geophysics **84**, Q27–Q36 (2019). https://doi.org/10.1190/geo2018-0340.1