Enabling Large Scale Simulations for Particle Accelerators

Konstantinos Iliakis, Helga Timko, Sotirios Xydis, *Member, IEEE*, Panagiotis Tsapatsaris, and Dimitrios Soudris

Abstract—International high-energy particle physics research centers, like CERN and Fermilab, require excessive studies and simulations to plan for the upcoming upgrades of the world's largest particle accelerators, and the design of future machines given the technological challenges and tight budgetary constraints. The Beam Longitudinal Dynamics (BLonD) simulator suite incorporates the most detailed and complex physics phenomena in the field of longitudinal beam dynamics, required for providing extremely accurate predictions. Modern challenges in beam dynamics dictate for longer, larger and numerous simulation studies to draw meaningful conclusions that will drive the baseline choices for the daily operation of current machines and the design choices of future projects. These studies are extremely time consuming, and would be impractical to perform without a High-Performance Computing oriented simulator framework. In this article, at first, we design and evaluate a highly-optimized distributed version of BLonD. We combine approximate computing techniques, and leverage a dynamic load-balancing scheme to relax synchronization and improve scalability. In addition, we employ GPUs to accelerate the distributed implementation. We evaluate the highly optimized distributed beam longitudinal dynamics simulator in a supercomputing system and demonstrate speedups of more than two orders of magnitude when run on 32 GPU platforms, w.r.t. the previous state-of-art. By driving a wide range of new studies, the proposed high performance beam longitudinal dynamics simulator forms an invaluable tool for accelerator physicists.

Index Terms—Distributed systems, GPU acceleration, approximate computing, network traffic optimisation, accelerator physics

1 Introduction

At CERN, the European Organization for Nuclear Research, physicists probe the fundamental structure of the universe, using instruments of unparalleled magnitude and complexity. Particle accelerators are used to accelerate and then collide charged particles at close to the speed of light, to gather traces of particle interactions, and insights into the principal laws of nature. The Large Hadron Collider (LHC) is currently the world's largest and most powerful particle accelerator. The discovery of the Higgs boson [1] in 2011 in the LHC is one of the Nobel-prize winning discoveries at CERN. Presently, the biggest challenge is to obtain hints for physics beyond the Standard Model, such as theories including gravitons, extra dimensions [2], and the matter-antimatter asymmetry [3].

The LHC Injector Upgrade (LIU) [4], the upcoming High-Luminosity LHC project [5], and the studies of future machines such as the Future Circular Collider (FCC) [6] are CERN's most important R&D projects at present. Despite of vast experience with the present infrastructure, future machines cannot simply be scaled in size and energy to achieve the desired beam quality and intensity. Studies are required to overcome known limitations, and potentially discover new ones. The studies are often complex and require precision modelling in the domain of accelerator physics and beam dynamics simulations to model the relevant physics phenomena and machine-specific features.

The simulator software has to be flexible enough to include a wide range of synchrotrons, energy regimes and particle types. To fulfill these critical requirements, the Beam Longitudinal Dynamics simulation suite (*BLonD*) [7], [8] was developed. As its name suggests, *BLonD* focuses on the longitudinal motion and tracks the energy and time coordinates of beam particles in synchrotrons. It features a modular structure that allows the user to combine a variety of physics phenomena according to the study requirements. *BLonD* is an open-source project[1] that is increasingly gaining popularity among the world's largest accelerator laboratories, including CERN, Fermilab, BNL, and KEK.

The outcome of *BLonD* simulation studies continuously guides the baseline choices for machine upgrades and future

- *Konstantinos Iliakis is with the Microprocessors and Digital Systems Laboratory, Electrical and Computer Engineering Department, National Technical University of Athens, 15780 Athens, Greece, and also with the Accelerator Systems Department, European Organization for Nuclear Research (CERN), 1211 Geneva, Switzerland. E-mail: kiliakis@microlab.ntua.gr.*
- *Helga Timko is with the Accelerator Systems Department, European Organization for Nuclear Research (CERN), 1211 Geneva, Switzerland. E-mail: helga.timko@cern.ch.*
- *Sotirios Xydis is with the Digital Technology Department, Harokopio University of Athens, 17671 Athens, Greece. E-mail: sxydis@hua.gr.*
- *Panagiotis Tsapatsaris and Dimitrios Soudris are with the Microprocessors and Digital Systems Laboratory, Electrical and Computer Engineering Department, National Technical University of Athens, 15780 Athens, Greece. E-mail: panagiotistsap97@gmail.com, dsoudris@microlab.ntua.gr.*

Manuscript received 18 February 2022; revised 20 June 2022; accepted 15 July 2022. Date of publication 20 July 2022; date of current version 23 August 2022.
This work was supported by the European Organization for Nuclear Research (CERN) through a full-time doctoral Studentship in Geneva, Switzerland.
(Corresponding author: Konstantinos Iliakis.)
Recommended for acceptance by D. Tiwari.
Digital Object Identifier no. 10.1109/TPDS.2022.3192707

1. The source code is hosted at: https://github.com/blond-admin/BLonD

machines. Modelling the motion of charged particles in synchrotrons is a computationally challenging task [9], [10]. A very large number of simulated particles is needed, and a vast variety of physics phenomena – e.g., beam feedbacks, machine impedance induced voltage, synchrotron radiation – and their interactions need to be taken into account to precisely simulate the beam motion in synchrotrons. In addition, future challenges in the field of accelerator physics contribute to an ever-growing need for greater simulation sizes. In particular, the High-Luminosity LHC [5] upgrade project, and the FCC [6], a 100 km long collider currently under study, will push the input requirements of *BLonD* simulations by one to two orders of magnitude. This, combined with the extended simulation periods that are required to model the physics phenomena related to these two new projects, will skyrocket the execution time and complexity of beam dynamics simulations.

Due to the inherently parallel nature of the equations of motion (see Section 2.1), these workloads fit naturally in a distributed-memory environment, where MPI can be used for the necessary inter-process communication. To undertake this agenda, in this paper we extend and further optimize *HBLonD* [10], a hybrid, MPI-over-OpenMP library for running large scale longitudinal beam dynamics simulations. To overcome the main limiting factor of distributed, scientific applications [11], that is the inter-node communication time, two physics-inspired approximate computing techniques are proposed. The approximation methods are thoroughly verified in terms of prediction accuracy, with the aid of domain experts in the field of beam dynamics, confirming satisfactory agreement with exact, non-approximate simulations. Next, a highly customizable, dynamic load-balancing scheme is introduced to mitigate multiple sources of performance variability among remote computing nodes that lead to imbalanced execution. The refined *HBLonD*, demonstrates up to 43-57× speedup when deployed on 32 computing nodes, compared to the prior state-of-art. Finally, to satisfy the ever-growing need for larger workloads and longer simulations, we further extend *HBLonD* by combining CUDA and MPI to build a GPU-accelerated, distributed version of *BLonD*, called *CuBLonD*. *CuBLonD*, designed with high performance and usability in mind, taking advantage of modern GPU features. *CuBLonD* is evaluated in a GPU-enabled supercomputing facility, demonstrating greater than two orders of magnitude speedups when using 16 or 32 GPU platforms. The suggested implementations are the key to satisfying the needs of near-future large and ultra-large scale beam dynamics simulation studies. *HBLonD* and *CuBLonD* are openly available[2].

The remainder of this paper is organized as follows: Section 2 provides an overview of the *BLonD* code structure and the physics phenomena simulated with it. Section 3 reviews prior work on beam dynamics simulators. In Section 4, a detailed description of the distributed implementation and dynamic load-balancing is provided. Section 5 presents the physics-motivated approximate computing techniques used for inter-node synchronization relaxation. *CuBLonD*, the GPU-accelerated version of *BLonD*, is detailed in Section 6. The multi-node scalability is evaluated experimentally in Section 7, and finally, Section 8 concludes this paper.

2. Hosted at: github.com/kiliakis/BLonD-1/tree/TPDS

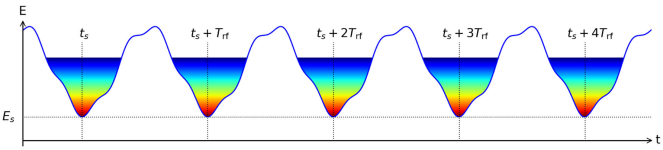


Fig. 1. Periodic potential well in a synchrotron for several RF systems. Without intensity effects, the synchronous point (t_s, E_s) is periodic with T_{rf} along the ring.

2 Background

2.1 Longitudinal Beam Dynamics in Synchrotrons

Longitudinal beam dynamics is the field of physics that models the beam motion in particle accelerators and focuses on the longitudinal plane, i.e., alongside the beam pipe, in contrast to transverse beam dynamics that focuses on the transverse plane. Synchrotrons are circular particle accelerators, in which Radio-Frequency (RF) cavities accelerate the beam, and magnets curve the beam. The particles circulate at a relatively fixed 'reference' orbit. A particle circulating on the synchronous orbit and arriving to the RF cavity at the synchronous RF phase is referred to as 'synchronous particle'.

'Beam' refers to all the charged particles of a given species in the accelerator. In the presence of RF voltage, the beam is condensed around the centres of the RF potential wells, forming a train of 'bunches,' as in Fig. 1. Particles are described in a 2-D phase space, by the longitudinal coordinate and its conjugate variable. A widely-used phase-space coordinate pair is $(\phi, \Delta E/\omega_0)$, where ϕ is the phase w.r.t. RF voltage wave and ΔE is the relativistic energy of the particle w.r.t. the energy E_s of the synchronous particle. The longitudinal equations of motion can be derived, for a single-RF system, to be [12]

$$\frac{d\phi}{dt} = \frac{h\omega_0^2\eta}{\beta^2 E}\left(\frac{\Delta E}{\omega_0}\right), \quad and \tag{1}$$

$$\frac{d}{dt}\left(\frac{\Delta E}{\omega_0}\right) = \frac{1}{2\pi} eV(\sin\phi - \sin\phi_s), \tag{2}$$

where e is the unit charge, V is the RF voltage amplitude and $\beta = v/c$ is the relativistic beta of the particle, with v the velocity and c the speed of light. The slippage factor η is a property of the synchrotron, which describes how much RF phase slippage a particle will undergo after one turn. In *BLonD*, the η used in Eq. (1) is replaced by the momentum compaction factor α of at least zero-th, and up to second order. When multiple RF systems exist, several voltage terms appear on the right-hand-side of Eq. (2).

Since the accelerator is composed of conducting devices, the beam particles interact with their surroundings electromagnetically. This leads to trailing particles being affected by leading particles, which is called collective effects. A given particle experiences thus an induced voltage by itself and the particles in front. If the overall impedance of the machine is $Z(\omega)$, the corresponding wake field $W(t)$ can be defined as the Fourier transform of the impedance,

$$W(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} d\omega e^{i\omega t} Z(\omega). \tag{3}$$

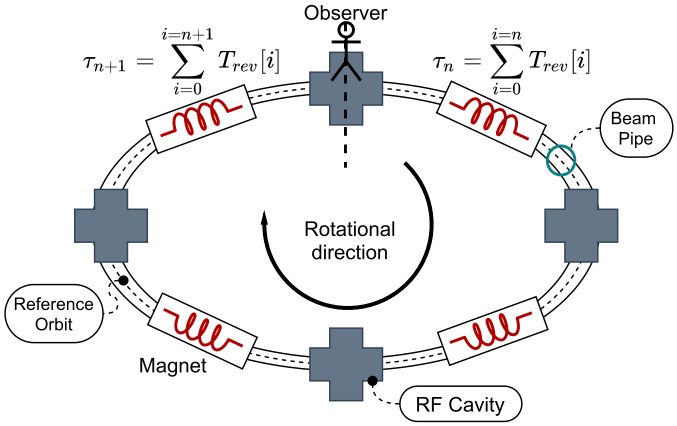


Fig. 2. Simple synchrotron model. The ring, the beam and the RF sections are the main components modelled in *BLonD*. Their interactions are simulated on a turn-by-turn basis.

Denoting the longitudinal profile of the beam particles with $\lambda(t)$, a particle at the coordinate Δt will experience the induced voltage

$$V_{\text{ind}}(\Delta t) = -q\,N_p \int_{-\infty}^{+\infty} \lambda(\tau)\,W(\Delta t - \tau)\,d\tau\,, \tag{4}$$

where N_p is the number of particles in the beam. The induced voltage is applied to the particles as an energy kick $E_{\text{ind}}(\Delta t) = q\,V_{\text{ind}}(\Delta t)$ added to Eq. (2), where q is the particle charge.

2.2 The *BLonD* Simulator Suite

The three main components modeled in *BLonD* can be seen in Fig. 2: the synchrotron or 'ring,' the Radio-Frequency (RF) cavities, and the beam that circulates inside the beam pipe. The modelling of these components in *BLonD* is described below.

In reality, a bunch contains trillions of particles; in simulations however, macro-particles are used which represent many real particles to reduce the memory footprint. In this paper the terms particles and macro-particles are used interchangeably. The user is responsible for determining the number of macro-particles required for sufficient resolution of a certain physical phenomenon. The computational complexity of most operations in *BLonD* scales linearly with the number of macro-particles, which typically ranges from a few millions to 100 s of millions.

Particles are described using the coordinates $(\Delta t_{(n)}, \Delta E_{(n)})$, which are the particle's arrival time and energy at the RF section with respect to the reference time and design energy, respectively. RF sections are placed in fixed locations along the ring. The number of RF stations depends on the case and ranges from one to a dozen. The so-called *kick* equation of motion is used to update the ΔE coordinate of a given particle from time step n to $n+1$, based on the particle's $\Delta t_{(n)}$ coordinate and the RF voltage energy kicks k received in the corresponding RF station,

$$\begin{aligned}\Delta E_{(n+1)} = \Delta E_{(n)} &+ \sum_{k=0}^{n_{\text{rf}}} qV_{k,(n)} \sin\left(\omega_{\text{rf},k,(n)}\Delta t_{(n)} + \varphi_{\text{rf},k,(n)}\right) \\ &- \left(E_{\text{s},(n+1)} - E_{\text{s},(n)}\right) + E_{\text{other},(n)},\end{aligned} \tag{5}$$

where q is the particle's charge, V_k the voltage amplitude, $\omega_{\text{rf},k}$ the revolution frequency, and $\varphi_{\text{rf},k}$ the phase of the RF system k, and $E_{\text{d},(n+1)} - E_{\text{s},(n)}$ the change in synchronous energy from one turn to another. The last term $E_{\text{other},(n)}$ is used to model energy changes due to effects such as intensity effects or synchrotron radiation.

The beam motion from one RF station to another is modeled by the *drift* equation of motion, that updates the time coordinate, using the updated energy of the particle,

$$\begin{aligned}\Delta t_{(n+1)} = \Delta t_{(n)} + T_{\text{rev},(n+1)} \Bigg[&\left(1 + \alpha_{0,(n+1)}\delta_{(n+1)} + \alpha_{1,(n+1)}\delta_{(n+1)}^2\right. \\ &\left. + \alpha_{2,(n+1)}\delta_{(n+1)}^3\right) \frac{1 + \frac{\Delta E_{(n+1)}}{E_{d,(n+1)}}}{1 + \delta_{(n+1)}} - 1 \Bigg],\end{aligned} \tag{6}$$

where $\delta_{(n)}$ is the relative momentum offset, T_{rev} is the revolution period, and α_i are the i-th order momentum compaction factors. One revolution of the beam in the ring corresponds to one simulation iteration. The number of iterations required for a given testcase can range from a few thousands to a few millions.

In frequency domain, the induced voltage (Eq. (4)) is then the product of the impedance and the beam spectrum $V_{\text{ind}}(\omega) = S(\omega)Z(\omega)$, where the beam spectrum is given by the equation $S(\omega) = \int_{-\infty}^{+\infty} \lambda(\tau)e^{-i\omega\tau}d\tau$, and $\lambda(\tau)$ is the beam line density. As discussed in Section 2.1, the ring's impedance produces *collective* effects, that couple the motion of one particle to another, and limit the exploitable parallelism degree of the code.

Since the number of macro-particles in simulations is typically large ($>10^6$), *BLonD* uses a histogram of the beam time coordinates to optimize the computation of the induced voltage. Particles are grouped in bins and the amplitude corresponding to the bin is given by the number of particles inside. A linear interpolation is applied between bins, similarly to particle-in-cell codes [13]. The minimum amount of bins required is fixed by the Nyqvist sampling theorem but depending on the phenomena modelled, a larger amount of bins might be necessary.

BLonD is a modular and flexible library. Other physics phenomena, such as space-charge effects, synchrotron radiation, or impedance-reducing control circuits can be included in a given simulation. A *BLonD* simulation scenario is an assembly of multiple machine, physics and beam dynamics components. Their effects and interactions are modelled on a turn-by-turn basis. Some optional features of *BLonD* include a complete tool-set for data analysis, storage and plotting.

2.3 Real-World Testcases

Three real-world testcases, representative of typical *BLonD* workloads are used for the experimental evaluation of *HBLonD*, and its GPU accelerated version *CuBLonD*. Each testcase targets a different synchrotron of CERN's accelerator complex, and has its own distinct characteristics. Together, they cover a wide range of beam dynamics features. Starting from the smallest machine to the largest, a short description of the three testcases follows.

TABLE 1
Qualitative Comparison of the Four *BLonD* Versions

Version	Language	Multi-threaded	Distributed	Accelerator	Load-Balancing	Approximate
BLonD	Python	—	—	—	—	—
BLonD++ [9]	Python/C++	OpenMP	—	—	—	—
HBLonD [10]	Python/C++	OpenMP	MPI	—	Empirical	SRP, RDS
CuBLonD	Python/C++/CUDA	OpenMP	MPI	GPU/Multi-GPU	Rigorous	SRP, RDS, F32, Verifiable

1. Proton Synchrotron (PS): The PS is the second synchrotron of the LHC injector chain and CERN's oldest synchrotron in use. It has a circumference of 628 m, and can accelerate protons up to an energy of 26 GeV, before injecting them into the SPS. Due to its relatively small size and closely spaced bunches, the PS is dominated by collective effects, meaning that leading particles of the beam affect the dynamics of trailing particles. The PS can accelerate up to 18 bunches simultaneously. The aim of the PS testcase is to study and control the beam instabilities that can manifest due to collective effects. The characteristic timescale of the beam motion in the PS is longer than in the other two testcases, resulting in slower moving particles.
2. Super Proton Synchrotron (SPS): The SPS receives bunches of charged particles from the PS, accelerates them to 450 GeV and delivers them to the LHC. With a circumference of 7 km, it is one of the largest machines worldwide. For the LHC-type beam, the SPS can receive up to four batches of 72 bunches, or 288 bunches in total. In the testcase used for the evaluation [14], due to the very high number of particles and bunches in the machine, collective effects were an important limitation. Furthermore, the detailed impedance model requires very fine frequency sampling rate, calling for time-consuming FFT operations on large inputs. Finally, the beam phase loop, a dynamic feedback system required to correct the bunch phase with respect to the RF system, adds to the overall simulation complexity.
3. Large Hadron Collider (LHC): With a circumference of 27 km and a collision energy of 13 TeV or more, the LHC is the world's largest and most powerful particle collider. The LHC can fit up to 2808 bunches. The testcase used in the evaluation [15], [16] describes one of the critical elements of machine operation, which is the controlled emittance blow-up during the acceleration ramp, required for beam stability. An RF noise is generated and applied turn-by-turn for the so-called controlled emittance blow-up, which is a stochastic process that increases the bunch size and largely affects the simulation results. As the LHC uses superconducting magnets, the acceleration ramp to 6.5 TeV takes about 13 million simulation steps, making the simulation extremely time consuming, even for the smallest sizes using a single bunch.

3 Related Work

ESME [17] is a longitudinal beam dynamics code developed at Fermilab in 1984. The code was written in Fortran and was compatible with Unix-based operating systems like Solaris. The lack of new features, active support and maintenance led the ESME project to disappear from the scientific community. Py-Orbit [18] is another beam dynamics simulator. Py-Orbit is a Particle-In-Cell, 6-D tracker code, modelling both transverse and longitudinal beam dynamics. For longitudinal purposes, the computational complexity of a Py-Orbit simulation is significantly heavier than that of a *BLonD* simulation. Good agreement between Py-Orbit and *BLonD* has been reported in the literature [16], [19]. Similarly to Py-orbit, Elegant [20] is a 6-D tracking code, too. Elegant has been parallelized with MPI [20], [21] and features a GPU accelerated version [22]. However, being more general-purpose oriented code, both Py-Orbit and Elegant lack many of the specific longitudinal and RF features available in *BLonD*.

BLonD differs from the aforementioned frameworks in numerous ways. *BLonD* emerged in 2014 to respond to the need for a highly customizable tool required by scientists at particle accelerator complexes to drive a large amount of longitudinal beam dynamics studies. *BLonD* introduces several unique features not found in any other prior simulator. Featuring a Python front-end, *BLonD* is easy-to-use and attractive to new users. The modular structure allows rapid prototyping of new features that extend its capabilities. Contrary to Py-Orbit and Elegant, *BLonD* specializes in the longitudinal plane and as a consequence, it contains more detailed physics models and is less computationally heavy. In terms of flexibility and precision, *BLonD* has been tested successfully on a plethora of real-world simulation scenarios [14], [19], [23], [24], [25], [26], [27], [28] and it is generic enough to cover a wide spectrum of beam dynamics simulation scenarios ranging from existing to future machines and from relatively small to very large synchrotrons accelerating protons, electrons or ions.

3.1 BLonD Evolution Over Time

The *BLonD* suite has undergone several re-iterations since first developed in 2014. The main HPC features included in the four *BLonD* variants are summarized in Table 1 and discussed bellow.

The original *BLonD* version was written in python, with its main goal to incorporate a detailed beam dynamics model in a language that allows for easy prototyping and rapid development. *BLonD++* [9] integrated an optimised C++ computational back-end, keeping the python front-end. The back-end of *BLonD++* uses extensively the SIMD vector units, and supports multi-threading with OpenMP. *BLonD++* demonstrated a speedup of 18× compared to the python-only *BLonD*. *BLonD++* was shown to be memory bound when increasing the thread count and could only exploit vertical scaling. This motivated the development of *HBLonD* [10], a

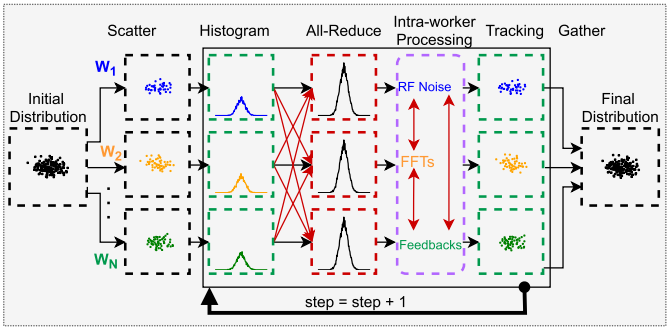


Fig. 3. The *HBLonD* workflow. A data-parallel model is used among the workers to parallelize the histogram and tracking phases. Task-parallelism is used in the intra-worker processing stage.

distributed code that combines MPI with OpenMP, to allow remote processes to communicate, and to profit from the fast shared-memory. *HBLonD* introduced two approximate computing techniques, SRP and RDS (see Section 5), to reduce inter-node communication and relax synchronization. A dynamic load-balancing (DLB) scheme was implemented to mitigate execution imbalances among the MPI workers. In this paper, the DLB scheme is further extended by adopting a more rigorous, experimentally verified, run-time estimation model, and exposing several configurable parameters for fine tuning. Furthermore, the two approximate computing techniques, SRP and RDS, are combined with reduced floating point datatype length, and are validated for the first time in terms of prediction accuracy in Section 5.1. The quantitative evaluation of the improved DLB scheme and approximation methods with respect to prior-art [10] is demonstrated in Table 4. Finally, *CuBLonD*, presented in Section 6, combines *HBLonD*'s distributed architecture with an optimized CUDA core to enable GPU accelerated beam dynamic simulations. *CuBLonD* performance and scalability is evaluated in a supercomputing cluster, demonstrating greater than two orders of magnitude speedup w.r.t. a non-approximate, single-node, 20-core, *HBLonD* instance.

4 Scale-Out Longitudinal Beam Dynamics

HBLonD [10], integrates MPI on top of the single-node optimized code *BLonD++* [9], to allow for multi-node, large beam dynamics simulation scenarios. These large scale simulations are needed for studies that involve upgrades of existing synchrotrons [5], research for future machines [29], and in general high resolution simulations. This section discusses the base implementation of *HBLonD*, and the newly added features that improve its scalability.

4.1 Baseline Distributed Implementation

In Fig. 3, the baseline *HBLonD* execution flow is displayed, showing three MPI workers for simplicity. Each box corresponds to a separate stage of the simulation, and the cross-worker arrows indicate communication. The *Histogram* and *Tracking* stages are then ones that perform the useful computations that are distributed in a data-parallel way, while the operations shown in the intra-worker processing stage are parallelised in a task-parallel manner. In the beginning, the 2-D particle distribution is generated using one of the many distribution generation methods included in the *BLonD* library. This initialization distribution step is not parallelized with MPI since it takes place only once. The particle distribution is scattered equally among the MPI worker processes.

In the histogram stage, each worker runs a histogram operation to calculate the so-called beam profile of its assigned particles. The beam profile is essentially a 1-D histogram of the particles' time coordinate. The amount of histogram bins is a configurable size that affects the simulation's precision and execution time. The local profiles of all workers must be summed to generate the global beam profile, which is needed for the subsequent stages. The global profile is used as input to the intra-worker processing stage. This stage is composed of operations that are only effectively parallelizable among the threads of a worker but not across workers. In this step, task-level parallelism is exploited. Typically three or more tasks are performed in this phase, depending on the use case. These tasks are distributed among the workers that share the same computing node. When all neighboring workers complete their calculations, they broadcast the calculated results, profiting from fast shared-memory communication.

Then, the workers perform the particle tracking, which essentially updates the particle coordinates according to the equations of motion, for all particles. The *kick* and *drift* are the two routines that always need to be applied during the tracking stage, as discussed in Section 2. Their computational complexity is linear w.r.t. the number of simulated macro-particles. This succession of operations, also referred to as the 'main loop,' from histogram to tracking is repeated hundreds of thousands or millions of times, before the final gather operation that assembles the resulting particle distribution that is one of the main outcomes of a *BLonD* simulation.

The steps discussed above are the backbone of a *BLonD* simulation. In addition to these steps, other operations related to data-analysis, statistics collection or data visualization might take place inside the main computational loop.

4.2 Dynamic Load-Balancing

Despite the fact that *HBLonD* was originally developed and deployed onto an homogeneous cluster, performance degradation due to load imbalance was observed, i.e., nodes that require more time than others to execute the same amount of workload. This increased latency appeared to be spontaneous and temporal, and in most of the cases persisted for small or medium periods of time. Since in every iteration all the workers need to synchronize (upper half of Fig. 4), the latency of the slowest worker is experienced by all workers in each simulation step. This spontaneous load imbalance was an artifact of the cluster, and not a feature of the workload, due to the uniform nature of the computations involved and the randomized workload distribution.

While slow workers experience near-zero waiting times at the synchronization barriers, faster workers experience a larger waiting time. By measuring the time the workers spent synchronizing, we calculate the spread of the workers' execution time and how imbalanced the workload is. Fig. 5a shows the time spread, calculated as the difference in run time among the MPI workers, normalized to the total execution time for three *HBLonD* testcases (described in

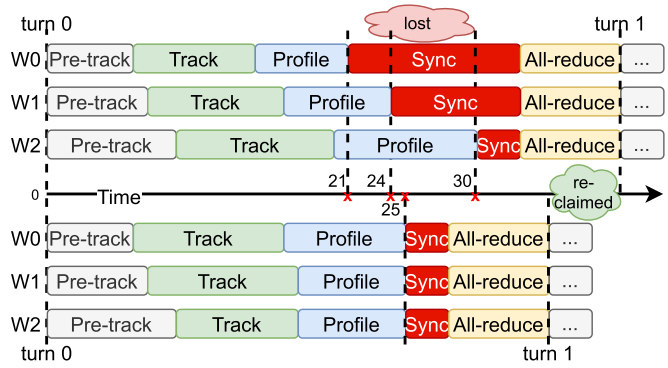
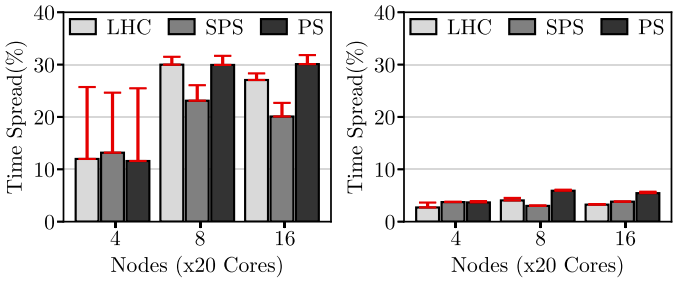


Fig. 4. The per-turn latency is defined by the slowest worker. By controlling the number of particles assigned to each worker and re-balancing the workload, the lost time is re-claimed.

Section 2.3) when running two-, four-, eight- or 16-node simulations, with two MPI processes per node. The detailed node and cluster configurations are provided in Section 7. Multiple runs per testcase and number of nodes were executed, and the red capped lines show the standard deviation of these runs. As shown, the time spread ranges from 11.6% to 30.1%. In some of the lengthier LHC simulations, this would translate to an additional 17 hours to 43 hours of execution time. Moreover, from Fig. 5a we observe that more nodes result to higher time spread because, statistically, the chances that one or more workers will undergo a delayed phase increase. On the other hand, while in the smaller node configurations the average time spread is low, the standard deviation is much larger. This means that most of the runs did not experience any increased latency phases, but there were still few cases with severe load imbalance.

To mitigate this issue, the Dynamic Load-Balancing (DLB) scheme originally presented in the early version of *HBLonD* [10], has been extended. The proposed DLB scheme is generic enough to alleviate imbalances that originate from various sources such as the cluster's topology, the interconnection network, and the involvement of heterogeneous hardware. The proposed DLB scheme has been customized to the following *HBLonD*'s specific workload properties:

1. $tComp_i = p_i \times m_i + c_i$: The computation time of worker i is linearly associated with the number of particles (p_i). The linear relation is defined by the slope (m_i) and y-intersect (c_i) coefficients. The computation time is considered the time needed to perform the particle tracking (*kick* and *drift* methods) as well as to calculate the beam profile (*histogram* operation). All these methods have a linear time complexity with the number of input particles. The first subplot of Fig. 6 validates experimentally this assumption.
2. $tComm_i = const, tIntra_i = const$: The nodes only need to communicate the beam profile and not the beam coordinates, thus the communication ($tComm$) and intra-node ($tIntra$) processing time are independent of the number of particles. The length of the beam profile is equal to the number of histogram bins, which is typically much smaller, of the order of 1‰, than the number of particles. The second and third subplots of Fig. 6 shows that the communication and intra-node processing time are basically independent of the particles' size.
3. Perfect load balance $\Leftrightarrow tSync_i \rightarrow 0$: In a perfectly load-balanced scenario, all workers are in sync and experience near-zero synchronization time ($tSync_i$).
4. Once a worker enters a slower than usual execution phase, it maintains this status for a noticeable period of time.



(a) Time spread without DLB. (b) Time spread with DLB.

Fig. 5. The difference in run time among the workers, normalized to the total execution time. Without DLB, the time spread ranges from 11.6%-30.1%. With DLB, it is limited to 2.7%-5.9%.

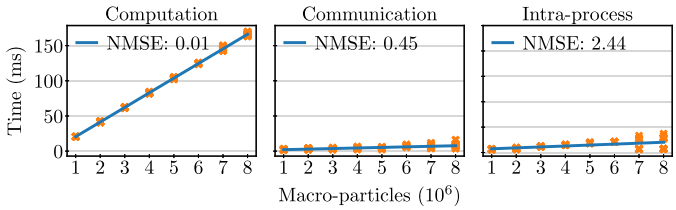


Fig. 6. Experimental verification of the two first workload assumptions on which the DLB scheme is based.

According to the first property, the workers by measuring their computation time, they can apply a least-squares, 1st-degree polynomial fit method to calculate the slope (m_i) and y-intersect (c_i) coefficients. A weighted polynomial solver is used, that takes into account a fixed number of past measurements. To allow for swift reaction to developing imbalances, greater weight is given to the more recent data points. An exponential decay function is used to calculate the weights W, given by the following equation:

$$W[k] = e^{-(H-k)/DC}, k \in \{1 \ldots H\}, \tag{7}$$

where H is the number of historic points considered and DC is the decay coefficient.

Due to the synchronization barriers before the communication phase, all workers will experience the same per-turn latency T:

$$tComp_i + tIntra_i + tSync_i = T, \xrightarrow{\text{property 1}} \tag{8}$$

$$p_i \times m_i + c_i + tIntra_i + tSync_i = T. \tag{9}$$

The $tIntra_i$ can be also measured by each worker, and since it is a constant independent of the particles, let us incorporate it in c_i:

$$p_i \times m_i + c_i + tSync_i = T. \tag{10}$$

Faster workers will experience longer synchronization time so that the turn latency of all workers is equal to T.

TABLE 2
DLB Scheme Configurable Parameters

Name	Default	Description
Rebalance Period	1000	Num. of turns between two re-balancing operations.
Tx_{min}	3%	Min. percentage of particles in a transaction.
P_{min}	10%	Min. percentage of particles assigned to a worker.
History length	20	Number of points to consider in polynomial fitting.
Decay Coefficient	5	Used in the exponential weight function.

According to the 3 rd property, we want to find a new set of (p'_i, T') that makes $tSync'_i = 0$:

$$p'_i \times m_i + c_i = T', \text{with } P = \sum_{i=0}^{N} p'_i, \tag{11}$$

where P is the total number of particles that remains constant, and N is the number of workers. This is a set of $N+1$ equations with $N+1$ variables. By solving this set of equations we get:

$$p'_i = \frac{P + sum_1 - c_i \times sum_2}{m_i \times sum_2}, \text{ where} \tag{12}$$

$$sum_1 = \sum_{i=0}^{N} \frac{c_i}{m_i}, \text{and } sum_2 = \sum_{i=0}^{N} \frac{1}{m_i}. \tag{13}$$

Knowing the number of particles each worker should be assigned in order to simultaneously arrive at the synchronization barrier (bottom half of Fig. 4), slower-than-average workers have to offload a portion of their workload to faster-than-average workers. This set of transactions is calculated by minimizing data traffic [30] and prioritizing transactions within the same node. The workers calculate the particles they need to send or receive from other workers. After completing these transactions, the workers continue with the next iteration of the simulation. This process is repeated periodically. The DLB scheme is highly customizable, and able to cover a wide range of load-imbalance scenarios. The key configurable parameters and their default values, that were identified by exhaustive exploration, are summarized in Table 2. For optimal and portable performance, these DLB parameters must be fine-tuned under a specific workload, load imbalance scenario, and cluster configuration.

The rebalance period is the interval in turns the DLB takes place. A long interval will make the DLB unable to respond swiftly to imbalanced execution, but a too short one will increase the DLB overhead. The Tx_{min} defines the minimum transaction size in terms of particles. The intuition is that the system should not be hyper-sensitive to marginal imbalances, which have insignificant effect on the execution time. P_{min} is used for overall system stability. Without it, a very slow worker would be assigned no particles, which can have negative effects in the system's stability and lead to crashes. Finally, the history length H and decay coefficient DC are used in the exponential decay function in Eq. (7), that define the sensitivity of the system to developing imbalances.

Fig. 5b shows the normalized spread in time among the MPI workers when the DLB mechanism is enabled. It is evident that the imbalance among the workers has been minimized both in smaller and larger node configurations. The time spread is limited to 2.7%-5.9%, i.e., $5\times$ lower than without the DLB mechanism. Furthermore, the more balanced workload brings 17% gain in execution time on average across three real-world simulation cases. The overhead of the DLB scheme is limited to 0.4% for the polynomial solver and p'_i calculation, and 1.1% for the particle transactions, so in total 1.5% of the total execution time.

5 Approximate Computing Methods

Minimizing the communication and synchronization time is essential to enable higher scalability in distributed codes [11]. *HBLonD*, specifically, spends about 50% of the total execution time for inter-node communication when using 16 computing nodes. For this purpose, *HBLonD* utilizes two approximate computing techniques, namely Representative Distribution Subset (RDS) and Smoothly Revolving Profile (SRP) [10]. The approximation techniques are physics-motivated and trade off accuracy, which is strongly case dependent, for latency gain through reduced inter-node traffic. In this paper, the approximations have been extended with more aggressive numerical approximations, thoroughly evaluated and verified by domain experts in beam dynamics that deeply comprehend the simulated phenomena and can decide which measure of error should be calculated and whether a loss in prediction accuracy is affordable.

Representative Distribution Subset (RDS). In the beam distribution scatter phase, a large number of particles, in the order of 100 s of millions, is distributed randomly among a small number of MPI workers. Consequently, every worker is assigned 10 s or 100 s of million particles. RDS is based on the assumption that each worker is assigned a representative subset of the particles that describes the overall distribution adequately. As expressed in Eq. (14) to approximate the global beam profile, worker i can simply scale up the local beam profile by the ratio of all particles to the assigned particles:

$$globalProfile \equiv \sum_{i=0}^{N} profile_i \simeq profile_k \times \frac{\sum_{i=0}^{N} p_i}{p_i} \tag{14}$$

where N is the total number of workers, p_i are the particles assigned to worker i, and $profile_i$ is the local beam profile of worker i. As a consequence, the costly all-to-all beam profile reduction is avoided, the workers' execution is disengaged, and the time needed for communication and synchronization is reduced.

Smoothly Revolving Profile (SRP). In the 2-D time and energy phase space, the particles are slowly revolving around the synchronous point with every simulation step.

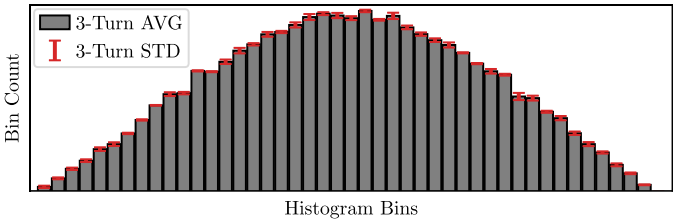


Fig. 7. SRP: Three-turn average and standard deviation of slowly-changing beam profile.

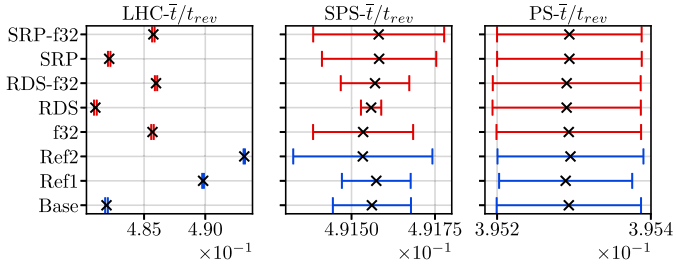


Fig. 8. Accuracy loss in the time coordinate. *Base* is the approximate-free baseline. *Ref1*, *Ref2* are used as reference points.

This approximation is based on the assumption that the beam profile is not changing rapidly between consecutive steps, which is generally true for the slow synchrotron motion of particles, in the absence of for instance fast beam instabilities, i.e., effects that develop rapidly and perturb violently the beam motion. Indeed, as depicted in Fig. 7, in real-world scenarios with sufficient number of simulated particles per histogram bin, the per-turn variation of the beam profile is limited to 1.3% on average. This condition needs to be verified by the user before enabling the approximation. SRP updates the beam profile every K iterations as shown in Eq. (15), which is generally tolerable in terms of precision for small K values, e.g., two or three.

$$profile^{t} = profile^{t+1} = \cdots = profile^{t+K-1}, t \bmod K = 0, \tag{15}$$

where $profile^{t}$ is the beam profile of turn t. By doing so, the need to perform the histogram calculation and the costly, communication-heavy global reduction in every turn is eliminated, reducing the simulation latency significantly.

Floating Point Datatype. In addition, we also evaluate one of the most typical performance-accuracy trade-offs in scientific codes, the floating point datatype size. *BLonD* traditionally uses 64-bit floating point numbers to ensure maximum accuracy in the calculations involved. However, using 32-bit floating point numbers can provide significant latency gains, without major accuracy loss. In the 32-bit *HBLonD* version, the datatype length of all floating point variables is reduced from 64-bit to 32-bit. Additionally, all the routines that were previously operating on 64-bit floats have been adjusted to operate on shorter, 32-bit wide floats. The two previous approximate computing techniques, i.e., RDS and SRP, can be combined with 32-bit floating point numbers for additional gains in latency, as we will see in Section 5.2.

5.1 Accuracy Evaluation Process

In this section, we evaluate the approximate computing techniques described above in terms of accuracy loss. RDS and SRP can be combined with 32-bit floating point size (f32), therefore in total five different approximate variations are considered: SRP, SRP-f32, RDS, RDS-f32 and f32. Naturally, these five methods produce different simulation results, which are compared to the results of the baseline, approximate-free version. To be able to evaluate the magnitude of this difference, we put it in perspective with the statistical fluctuations induced by altering the input seed of the pseudo-random number generator (PRNG) used in the particle distribution generation. This statistical fluctuation is considered acceptable, when using a large enough number of simulated macro-particles. We note that the adopted evaluation process was driven by beam dynamics domain experts. Additionally, while in other applications, verification of the accuracy in small scale experiments is a good estimate for large scale experiments, this is not the case for RDS and SRP since the assumptions these techniques are based on (Section 5) only hold on large datasets, and are severely hindered in small input sets.

In Fig. 8, there are three data points in addition to the five approximate variations. *Base* corresponds to the approximate-free, exact simulation. The *Ref1* and *Ref2* data-points are also non-approximate, but have a slightly altered initial particle distribution depending on the seed value passed to the PRNG method. By using a large enough number of particles, the statistical deviation of two particle distributions with a different seed is minimized. The only difference between *Ref1*, *Ref2* and *Base* is the seed value used for the generation of the input particle distribution. To limit the statistical fluctuation deriving from the exact initialisation of the particle coordinates, large numbers of particles (four million) were used in the experiments shown in Fig. 8.

The cross points in Fig. 8 correspond to the distribution's average after a relatively large period (approximately 40 thousand) of simulated turns. The error bars show the per-turn fluctuation of the distribution's average in the last few turns. As described above, the *Base*, *Ref1*, and *Ref2* data points are basically equivalent, and their difference reflects the statistical fluctuation noise.

The best agreement can be observed in the PS testcase, where both the average values and the error bars of the base, reference and approximate data points are essentially identical. This behavior results from the fact that in the PS simulation scenario the particle distribution is slowly moving and more resilient to micro-approximations. As a consequence, the PS testcase is a good candidate for more aggressive approximations in order to save execution time.

On the other hand, in the LHC testcase, we observe the greatest variation between the base, reference and approximate values. This is related to the RF noise diffusion phenomenon present in this specific testcase. The RF noise is also generated based on a random number generator, and the final particle distribution strongly depends on the generated noise sequence. As a consequence, it overshadows the final result variation coming from both the seed derived statistical fluctuation, and the approximate techniques. Moreover, by observing the error bars of all three approximate versions using 32-bit float (*SRP-f32*, *RDS-f32*, and *f32*), we see that reducing the data-type precision introduces a significant loss. This is due to the fact that 32-bit floats are

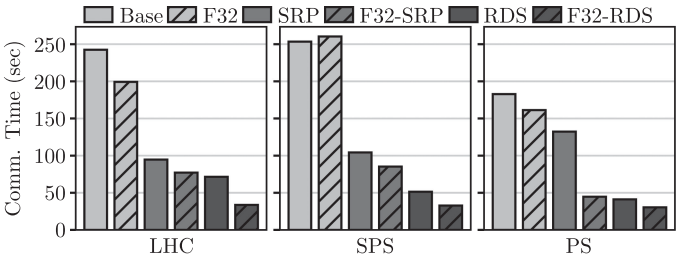


Fig. 9. Time spend for inter-node communication with the various approximate computing techniques.

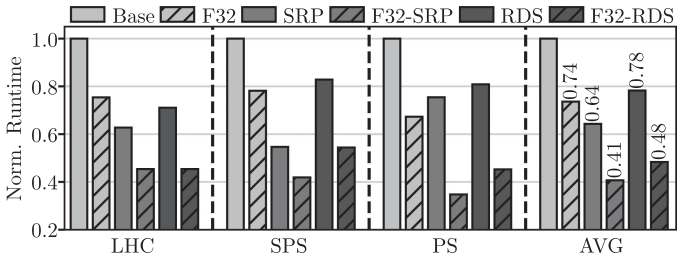


Fig. 10. Approximate computing latency gain evaluation in three test-cases. Lower values correspond to greater gains in latency.

used for the RF noise, which results in a different beam distribution than its 64-bit equivalent. Nevertheless, Fig. 8 shows that the base point is closer to the approximate points than the two reference points for the LHC testcase. This means that the deviations coming from the approximations are less significant than the deviations coming from the simulated phenomenon itself, and thus it is safe to apply the approximate methods.

Finally, the SPS testcase falls within the two extremes, and shows mostly good agreement between the approximate and reference points, and the base simulation results. In the SPS testcase, it can be seen that the *RDS* approximation method reduces the per-turn variation error bars. The *RDS* method decouples the MPI workers and lets them approximate the global beam profile based on their local beam profile. This effectively reduces the per-worker distribution size, which is possibly the reason why the pet-turn variation error bars appear to be narrower.

In conclusion, all the five approximate computing techniques presented above provide acceptable agreement with the exact, non-approximate simulations. Their error magnitude is strongly testcase dependent. Approximate computing is therefore a valuable tool for *BLonD* users, and can be used to reduce the run-time of lengthy simulations, as shown in Section 5.2.

5.2 Latency Gain Evaluation

The RDS and SRP approximation techniques were designed to reduce inter-worker communication time, and relax synchronization. Reducing the floating point data-type length is widely used to accelerate compute intensive code regions and minimize the memory footprint. Fig. 9 demonstrates the total time allocated for inter-node data exchange in each of the three test-cases when applying the aforementioned approximation techniques, as well as the combination of SRP and F32 (F32-SRP) and RDS with F32 (F32-RDS). Eight nodes with 160 cores in total were used for this experiment. Using smaller datatype size is not affecting greatly the communication time, since most of the traffic has to do with the beam profile, which is an array of 32-bit integers. The SRP and RDS approximations reduce the communication time by a factor of 2.5 $\times$ -5$\times$. As expected, the RDS affects more the inter-node traffic since it eliminates the costly all-to-all collective operation, while the SRP technique only applies the all-to-all operation less frequently than in the non-approximate baseline.

Fig. 10, summarizes the gain in run time resulting from the approximation mechanisms. The y-axis shows the run time, normalized to *HBLonD* without any approximation applied. Lower values correspond to greater gains in latency. As expected, the combined approximate versions, F32-SRP and F32-RDS demonstrate the greatest gains in latency, 59% and 52% on average, respectively. SRP updates the beam profile once every three turns, which is one of the most time consuming calculations. Furthermore, the induced voltage, another time consuming operation, and the all-to-all beam profile reduction do not need to be re-calculated unless the beam profile has been updated.

We observe that the characteristics of the simulated scenario affects the latency gains of the various approximate techniques. In the PS testcase, using 32-bit floats instead of 64-bit floats provides 33% reduced run time. This is due to the fact that the particle tracking related operations, such as kick and drift, allocate a large percentage of the total execution time, and these operations profit the most from the reduced data-type length. In the SPS testcase, the run time is reduced by 45% when updating the beam-profile periodically instead of in every turn. This is because the all-reduce and the induced voltage operations are calculated periodically, which jointly account for 40% of the execution time in the SPS testcase. Finally, in the LHC testcase the RDS method offers 21% latency gain. The RDS method allows the workers to operate independently by approximating the global beam profile based on their local beam profile. This completely eliminates the cost of the all-to-all reduction and any time lost in synchronization among the workers. In the LHC testcase, these two categories were allocating 39% of the total execution time.

An alternative approximation method combining RDS and SRP could be also considered. The idea is to do local profile scaling like in RDS, but periodically run the global reduction like in SRP. This combination of RDS and SRP is potentially more accurate than RDS, but less fast than RDS and SRP. However, as RDS was evaluated in terms of accuracy, demonstrating acceptable agreement with the non-approximate simulations (see Section 5.1), we focused our analysis on RDS, SRP and their 32-bit float variations. Since the accuracy of the approximation methods is testcase dependent, in the future, given new simulation scenarios, it is not unlike to implement new hybrid approaches to explore different trade-offs between accuracy and performance. In summary, the approximation techniques provide significant latency gains with minimal loss in accuracy. They are an invaluable tool to the users of *BLonD*.

6 CuBLonD: Multi-GPU BLonD Simulations

Graphic Processing Units (GPUs) were originally designed for efficient image and video processing in computer

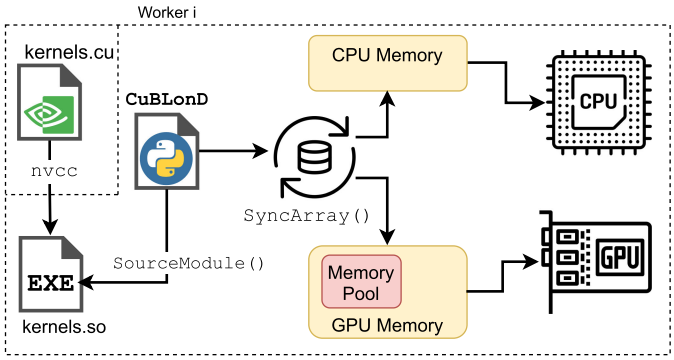


Fig. 11. Software architecture of the GPU-enabled *CuBLonD* code.

graphics applications. Over time, their massive computing capacity as well as the emergence of intuitive programming models, such as CUDA [31] and OpenCL [32], lead to the widespread use of GPUs for general purpose applications [33], [34]. Nowadays, almost all TOP500 supercomputers use GPU cards for acceleration. In this section we examine the details and performance of the GPU-accelerated version of *BLonD*, called *CuBLonD*, which we developed.

Not every application is well-fitted for GPU acceleration. GPUs are throughput oriented, meaning they are optimized for processing large data sets at the expense of increased latency in smaller-scale data sets. *BLonD* simulations can reach up to 1 billion of simulated macro-particles, providing enough work to the GPU hardware to sustain high levels of parallelism. Furthermore, frequent CPU-GPU communication is often a performance limiting factor for GPU accelerated applications. In *CuBLonD*, the largest structures, which are the energy and time coordinate arrays, are transferred once from the CPU to the GPU memory, where they reside for the entire simulation. Only smaller arrays, such as the beam profile, are transferred more often between the host CPU and GPU memory. Finally, one of the challenges in a code like *BLonD*, is the variety of kernels required to model all the physics effects in typical simulation scenarios. *CuBLonD* accelerates almost the entirety of operations that take place during the main computational loop.

6.1 Seamless CUDA Integration

Native CUDA Kernels. The CUDA enabled variation of *HBLonD*, *CuBLonD* was designed with performance as well as ease-of-use in mind. The PyCUDA [35] and Scikit-CUDA libraries were used to simplify the GPU memory allocation and the integration of native CUDA code in Python. In Fig. 11, we can see the high-level architecture of *CuBLonD*. The CUDA kernels are merged in one source file (`kernels.cu`) which is then compiled using the Nvidia C compiler to a shared library (`kernels.so`). This library is exposed to the Python front-end using the `SourceModule()` method of PyCUDA, which allows for direct calls to native CUDA code with seemingly zero performance overhead.

Auto-Synchronized, CPU-GPU Arrays. To simplify the usability of the code, we developed the `SyncArray()` class. This class extends the Numpy [36] array interface, and basically provides to the user a single array reference that can be used as is by CPU and GPU code. Under the hood, this requires the array to be allocated both in the CPU

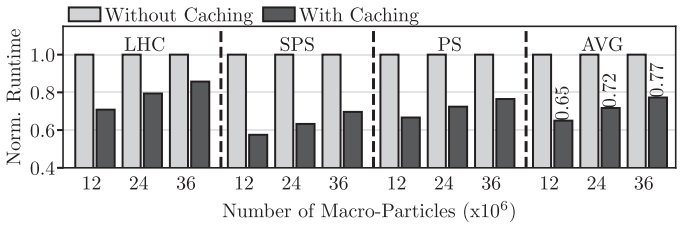


Fig. 12. Effect of caching common arrays in GPU main memory.

and GPU memory. When a modification to the CPU copy is made, the GPU copy is invalidated, and vise-versa. The invalidated copy is updated lazily, so that consecutive modifications will cause only a single data movement, when the invalidated copy will be first accessed. Since the greatest part of computing operations has been ported to the GPU, this process rarely needs to happen, especially inside the main computation loop. In a typical distributed, multi-GPU simulation scenario, data need to be transferred only once per iteration from the GPU to the CPU and back, during the collective `all_reduce()` MPI operation. When using two computing nodes, the percentage of the execution time spent on CPU-GPU data exchange is less than 2% in all our test-cases. However, there are scenarios, where a user of *BLonD* would need to access data stored in GPU for reporting, plotting, storing in files. In this case, the `SyncArray()` mechanism hides the underlying complexity from the end-user, while ensuring the correctness of the simulation.

Efficient Main Memory Management. Another mechanism we developed is the GPU memory pool. There are certain operations in a *BLonD* simulation, such as the evaluation of multiple FFTs, that require the allocation of temporary memory regions. The size of these memory regions often remains constant for long simulation periods. To avoid frequent allocation/de-allocation of these memory regions, we designed a software-managed memory pool, on top of the `SyncArray()` class. This means that the end-user does not need to be aware of the memory pooling mechanism. The memory pool caches frequently used memory structures, and returns them when requested. Since the memory pool manages a fixed amount of memory, it uses a Least Recently Used policy to replace old memory allocations with recent ones. In Fig. 12, we can see the effect on the run time of the GPU pooling mechanism with varying input sizes. The run time is normalized to that of *CuBLonD* without pooling, therefore lower values correspond to greater performance gains. We observe that the performance gain decreases in larger input sizes. This happens since the memory allocation/de-allocation time is amortized across lengthier calculations, due to the higher number of particles. Furthermore, we see that the SPS testcase profits the most from memory pooling. The SPS testcase runs the most FFTs among the three testcases, and subsequently has the most allocation/de-allocation requests among the three testcases. On average, we see a 23% to 35% performance gain from the memory pooling mechanism.

Thread-Block Caching. With the use of CUDA libraries like CuRand and CuFFT, porting most of the computationally heavy code regions from C++ to CUDA was a straightforward process. Apart from the functions found in the third-party libraries, in total 104 CUDA kernels were developed, 32

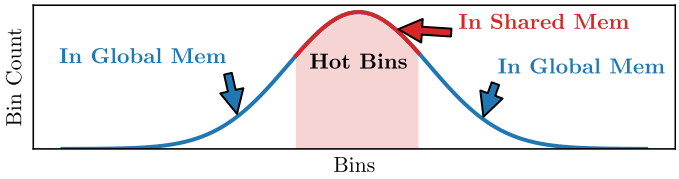


Fig. 13. Using the thread block shared memory to cache the most frequently accessed histogram bins.

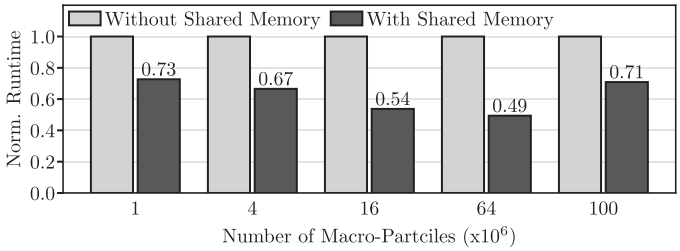


Fig. 14. Latency gain by making use of the thread block shared memory in the `histogram` operation.

of which implemented physics-related operations and the remaining 72 were mathematical and auxiliary functions. Two benchmarks in particular, the `histogram()` that generates the beam profile and the `linear_interpolation_kick()` required adjustments to make proper use of the shared memory. More specifically, to avoid costly atomic operations on global memory structures, the histogram kernel first allocates a thread-block private beam profile. Then the thread-block private profiles are reduced to generate the global beam profile (or worker-wide beam profile). However, since the shared memory is a limited resource, in large simulations only a portion of the beam profile fits in the shared memory. In this case, we take advantage of the Gaussian-like shape of the beam, to store in memory only the "hottest" bins, those around the center of the Gaussian distribution, as can be seen in Fig. 13. The remaining bins reside in global memory and are updated using atomic operations. Since the bins around the center of the distribution are accessed more frequently than the others, the average memory access latency approaches the latency of the fast shared memory.

The performance gain of this hybrid implementation is shown in Fig. 14. The y-axis shows the run time normalized to the global memory only implementation, therefore lower values correspond to greater run time gains. The x-axis shows different numbers of input particles. We observe that the performance gain gradually increases from 27% with 1 million particles, to 51% with 64 million particles. Then, it reaches a saturation point, where the portion of the beam profile that fits in the shared memory is too small to capture the hottest bins, therefore the performance gain is reduced to 29%. We note that 100 million macro-particles is the largest input size that we expect to use per GPU platform, since for very large simulations we use multiple GPU platforms and computing nodes.

6.2 *CuBLonD* Single-Node Performance Evaluation

This section evaluates the single-node performance of *CuBLonD* in comparison with *HBLonD*. Later, in Section 7, the scalability of the code is examined. The specifications of the CPU server used for *HBLonD* are shown in the last column of Table 3, while the specifications of the two GPU platforms evaluated are shown on the second and third column of Table 3.

The y-axis of Fig. 15, shows the throughput of *CuBLonD* using either one GPU K40 platform (*CuBLonD-1xK40*), or two GPU K40 platforms (*CuBLonD-2xK40*) in one node, or one GPU V100 platform (*CuBLonD-1xV100*), normalized to the throughput of *HBLonD* on a single 20-core node. Higher values correspond to greater speedups. *CuBLonD* with one K40 provides approximately 5× higher throughput than the CPU-only *HBLonD*. We observe varying speedup values for the three testcases, since each testcase is a unique application with specific characteristics. By utilizing both GPU platforms, the average speedup is 9.3× compared to the CPU baseline, or 1.82× higher than when using one platform. The speedup is not doubled when going from one to two GPU platforms, mainly because the code is not entirely GPU-accelerated and parallelized. In fact, on average 96.2% of the code runs on the GPU and is also parallelized with MPI. In addition, using two GPU platforms instead of one adds a communication and synchronization overhead to the total execution time.

The K40 GPU, when first released in 2013, it was a high-end, server class GPU. Nowadays it has been superseded by newer generations. Fig. 15 shows the performance of *CuBLonD* using a more recent GPU generation – the Nvidia Tesla V100. The model's specifications can be found in Table 3. On average, V100 is 4× faster compared to the K40, providing 20× faster execution compared to a 20-core CPU node. These results demonstrate the potential of *CuBLonD* to efficiently take advantage of both older and more recent GPU platforms and provide great speedups w.r.t. the previous state-of-art CPU-only implementation.

TABLE 3
Benchmarking Platforms' Specifications

Model	Nvidia Tesla K40	Nvidia Tesla V100	Intel Xeon E5-2630v4
Cluster	ARIS	–	ARIS/ CERN-HPC
Generation	Kepler (2013)	Volta (2018)	Broadwell (2016)
Process size	28 nm	12 nm	14 nm
RAM	12 GB GDDR5	32 GB HBM2	2 x 64 GB DDR4
Bandwidth	288 GB/s	897 GB/s	2 x 68 GB/s
Cores	15	80	2 x 10
Frequency	0.75 GHz	1.2 GHz	2.2 GHz
Cache	1.5 MB (L2)	6 MB (L2)	2 x 25 MB (L3)
Compiler	NVCC (v10.1)	NVCC (v10.1)	GCC (v7.3)

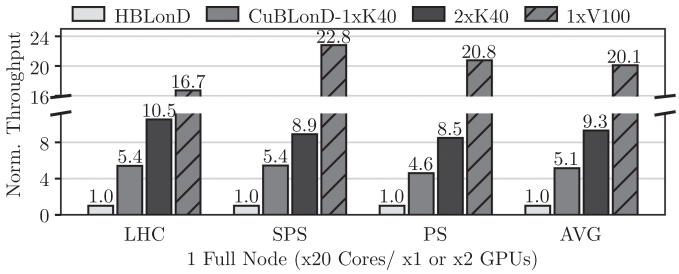


Fig. 15. *CuBLonD* single-node normalized throughput with one K40 GPU, two K40 GPUs in one node, and one V100 GPU.

7 Multi-Node Scalability Stressing

This section evaluates the scalability of *HBLonD* and *CuBLonD* on multiple computing nodes. *HBLonD* was evaluated on an HPC cluster hosted at CERN, while *CuBLonD* was evaluated on the ARIS, Greek national supercomputing infrastructure[3]. The nodes of both clusters contain two 10-core Intel servers. The ARIS nodes contain in addition two Nvidia K40 GPU cards. The specifications of all platforms are shown in Table 3. Both clusters use Infiniband FDR14 [37] interconnect. The largest *CuBLonD* experiments used 32 GPU cards on 16 nodes, and the largest *HBLonD* experiments contained 640 cores on 32 nodes. CUDA version 10.1 was used in the GPU-enabled nodes and GCC version 7.3.

Three MPI implementations were tested experimentally: i) MVAPICH2 [38], ii) OPENMPI3 [39] and iii) MPICH3 [40]. MVAPICH2 performed on average 11% better than MPICH3 and 17% better than OPENMPI3, therefore MVAPICH2 was preferred for the remaining scalability experiments.

Apart from the MPI implementation, the number of MPI workers-per-node (WPN) is another important configurable value that greatly affects the performance. We experimentally tested configurations with one, two, four or ten WPN. Generally, a very large number of workers might increase the communication time and affect the performance. On the other hand, a very small number of workers might also perform poorly due to sub-optimal memory usage [41]. The clusters used in the experimental evaluation are both composed of dual-socket servers. We observed that using one worker per socket performed the best (being on average 13% faster than using one WPN). The second best performing configuration was having four WPN, which was 8% faster than the single WPN. In the following experiments we use span two MPI processes per node, or equivalently one per socket.

The scalability of *HBLonD* and *CuBLonD* was evaluated under strong-scaling workload scenarios. In strong-scaling experiments, the input size is held constant while the number of computing resources increases. The purpose of this study is to determine whether the code can effectively scale in multiple computing nodes and discover the saturation point, i.e., the point after which the addition of more resources is not increasing the performance. As formulated by Amdahl's law, this happens when a portion of the workload is not parallelized.

3. This work was supported by computational time granted from the National Infrastructures for Research and Technology S.A. (GRNET S.A.) in the National HPC facility - ARIS - under project pa200702.

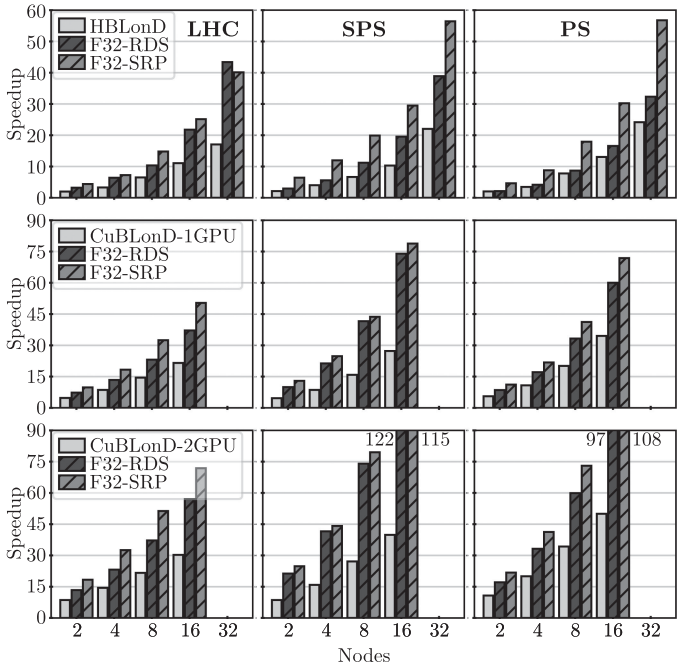


Fig. 16. From the top: The speedup of *HBLonD*, and *CuBLonD* with one or two GPUs per node.

Each of the three columns of Fig. 16 corresponds to one of the three real-world testcases described in Section 2.3. The top row shows the performance of *HBLonD* with up-to 32 nodes or 640 CPU cores. The middle row shows the performance of *CuBLonD*, when using one GPU card per node and up to 16 nodes, while the bottom row shows the scalability of *CuBLonD* with two GPU cards per node. Each subfigure of Fig. 16 demonstrates the scalability of the baseline non-approximate version, the 32-bit RDS approximation and the 32-bit SRP approximation. These approximate computing techniques have been described in detail in Section 5.

The y-axis of all sub-figures shows the speedup w.r.t. a non-approximate, single-worker, 20-core instance of *HBLonD*, which is basically equivalent to *BLonD++* [9]. The DLB mechanism is not applicable to the single-worker *HBLonD*, since the DLB is used to mitigate imbalances among two or more MPI workers. We observe that without enabling any approximate variations, the scalability of the code saturates around eight nodes, due to the excessive communication time. In *HBLonD*, with the aid of the

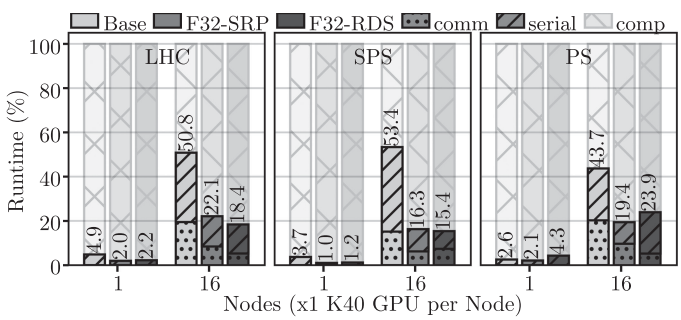


Fig. 17. *CuBLonD* execution time breakdown. As the node count increases, less time is spent on parallelized computations.

TABLE 4
Real-World Simulation Cases Run-Times

Testcase	Turns	Particles	*BLonD++* [9]	*HBLonD*-CF [10]	*HBLonD*-TPDS	*CuBLonD*
	$\times 10^6$	$\times 10^9$	Time	Time (Speedup)	Time (Speedup)	Time (Speedup)
LHC	14	0.8	110 days	4.3 days (25.6×)	2.5 days (43.4×)	1.5 days (71.8×)
SPS	0.43	1.7	8.8 days	9.8 hours (21.4×)	3.7 hours (56.4×)	1.7 hours (121.8×)
PS	0.38	0.7	2.3 days	1.8 hours (30.2×)	1 hour (56.7×)	0.5 hours (108.4×)
HLLHC	10	4	1.1 years	18 days	7 days	3.3 days
FCC	10	100	28 years	1.3 years	180 days	84 days

approximate computing techniques, a speedup of 43× to 58× is extracted in the three test-cases with 32 nodes. In the SPS and PS testcases, the non-approximate *HBLonD* with two computing nodes, achieves super-linear speedup of 2.13× and 2.03× compared to the baseline. This is a result of the task-parallelism exploited by neighboring workers. With task-parallelism, each worker calculates a subset of the non-parallelisable with MPI operations, and then neighboring workers exchange the calculated results profiting from fast, shared memory communication. These tasks demonstrate limited scalability, due to the nature of the operations or the relatively small input size. It is therefore most preferable to assign less cores in each task and calculate them in parallel, as is done in *HBLonD*, than to use all the available cores in every task and calculate the various tasks sequentially, as in the baseline, single-worker *HBLonD*.

The middle row of Fig. 16 corresponds to *CuBLonD*'s speedup, using one GPU platform per node. Similarly to *HBLonD*, the two approximate computing techniques allow for far greater performance scalability compared to the non-approximate baseline. The performance gain the SRP and RDS techniques is comparable, with SRP being slightly better, demonstrating speedups of 50×, 78× and 72× in the LHC, SPS and PS testcase, respectively. Finally, in the bottom row we can see the performance of *CuBLonD* when using both GPU devices available in every computing node. The largest configuration uses 32 GPUs in 16 nodes. In this case, the speedup compared to a 20-core reaches or surpasses two-orders of magnitude in the SPS and PS testcase, i.e., 122× and 108×, respectively. In the LHC case, the larger and more time consuming FFT operations during the induced voltage calculation, that are not parallelized across the MPI workers but only within every worker, the achieved speedup is slightly lower, at 72×.

When examining the scalability of the F32-SRP-*CuBLonD* in 16 nodes, w.r.t. a single-node, single-GPU instance of F32-SRP-*CuBLonD*, we get a speedup that ranges from 10.6× to 13.1×. To better demonstrate the factors that limit the code's scalability, we examine the run-time breakdown of *CuBLonD* in three categories: communication and synchronization (*comm*), time spent in regions not globally parallelized with MPI, but only locally with CUDA (*serial*), and the remaining fully parallelized computation time (*comp*). Ideally, the MPI workers should spend their time mostly in the parallelized regions.

Fig. 17 shows the time breakdown of the non-approximate *Base CuBLonD*, and *CuBLonD* with the *F32-SRP* and *F32-RDS* approximations. In the single-node configuration, no time is spent on communicating, since there is only a single worker, and less than 5% of the execution time of each testcase is spent for serial processing. When the number of computing nodes increases to 16, approximately 15% to 20% of the *Base* version run time is used for communication and synchronization among the remote MPI workers, while the percentage of the run time spent for serial processing ranges from 23% to 38%. The most time consuming, non-globally parallelized computation is the FFTs that take place during the induced voltage calculation. These are one-dimensional FFTs that do not scale effectively on a distributed environment due to the nature of the algorithm. In all three testcases, without applying approximate computing, about half of the total run time is spent running useful parallel operation, and the other half is used for cooperation and non-scalable operations. This is limiting the scalability that can be exploited as the node count increases, as shown previously in Fig. 16. The RDS approximation relaxes the synchronization required among workers, therefore reducing the communication time to 5-7%. The serial processing time is also reduced greatly with the approximate versions, ranging from 10–18%. Since not more than a quarter of the total run time is spent in non-parallelizable operation, we expect the performance of the approximate variations to scale with a further increase in the number of computing nodes.

8 Conclusion

We presented an efficient and scalable parallel implementation of *BLonD*, combining MPI, OpenMP and CUDA. The synchronization was relaxed and the network traffic was reduced by applying two physics-motivated approximate computing techniques. In addition, a dynamic load-balancing scheme was developed to automatically distribute the load among workers and mitigate imbalances. The *HBLonD* implementation demonstrated 43–56× speedup in three real-world testcases when run on 32 nodes, compared to the previous state-of-art code. Finally, a distributed, GPU-accelerated version, called *CuBLonD* was proposed, by integrating a user-friendly, Python front-end with an MPI-over-CUDA back-end, optimized for efficiency and scalability. *CuBLonD* reduced the execution time by up to two orders magnitude when run on 16 GPU-enabled nodes, compared to the previous state-of-art OpenMP-based *BLonD* simulator.

Table 4 summarizes the run-times, using the largest node configurations evaluated in Section 7, of five real-world testcases coming from CERN's accelerator complex. The four *BLonD* variations displayed correspond to: *BLonD++* [9],

prior-art *HBLonD* [10], *HBLonD* as presented in this paper, and *CuBLonD*. Typical studies are composed of 1000 s such simulations. The two last lines show optimistic projections of the run-times of two ultra-large testcases corresponding to future upgrades (HL-LHC [5]), and machines (FCC [29]). The FCC case is expected to be particularly challenging due to the very large number of particles required.

The dramatic reduction in execution time brought by *HBLonD* and *CuBLonD* has enabled scientists to simulate beam longitudinal dynamics scenarios that combine more complex physics phenomena with finer resolution and larger number of simulated particles, leading to discoveries [14], [27] impossible to observe in the past, and design specifications that would not have been realised. These complex, accurate and fast simulations are essential in the field of beam dynamics to overcome current technological limitations, plan the upcoming upgrades of particle accelerators, and design future machines that will help science advance further.

References

[1] F. Englert and R. Brout, "Broken symmetry and the mass of gauge vector mesons," *Phys. Rev. Lett.*, vol. 13, no. 9, 1964, Art. no. 321.

[2] Extra dimensions, gravitons, and tiny black holes, 2022. [Online]. Available: https://home.cern/science/physics/extra-dimensions-gravitons-and-tiny-black-holes

[3] The matter-antimatter asymmetry problem, 2022. [Online]. Available: https://home.cern/science/physics/matter-antimatter-asymmetry-problem

[4] H. Damerau *et al.*, "LHC injectors upgrade, technical design report, vol. I: Protons," CERN, Meyrin, Switzerland, Tech. Rep. CERN-ACC-2014–0337, Dec. 2014.

[5] G. Apollinari *et al.*, "High-luminosity large hadron collider (HL-LHC): Technical design report V. 0.1," Fermi Nat. Accelerator Lab., Batavia, IL, U.S., 2017, doi: 10.23731/CYRM-2017-004.

[6] A. Ball *et al.*, "The future circular collider study," *CERN Courier*, vol. 54, no. 3, pp. 16–18, Apr. 2014.

[7] S. Albright, K. Iliakis, A. Lasheen, D. Quartullo, J. Repond, and H. Timko, "CERN beam longitudinal dynamics code BLonD," 2014. [Online]. Available: https://blond.web.cern.ch/

[8] H. Timko *et al.*, "Beam longitudinal dynamics simulation suite blond," 2022, *arXiv:2206.08148*.

[9] K. Iliakis, H. Timko, S. Xydis, and D. Soudris, "BLonD: Performance analysis and optimizations for enabling complex, accurate and fast beam dynamics studies," in *Proc. 18th Int. Conf. Embedded Comput. Syst. Archit. Model. Simul.*, 2018, pp. 123–130.

[10] K. Iliakis, H. Timko, S. Xydis, and D. Soudris, "Scale-out beam longitudinal dynamics simulations," in *Proc. 17th ACM Int. Conf. Comput. Front.*, 2020, pp. 29–38.

[11] F. Betzel, K. Khatamifard, H. Suresh, D. J. Lilja, J. Sartori, and U. Karpuzcu, "Approximate communication: Techniques for reducing communication bottlenecks in large-scale parallel systems," *ACM Comput. Surv.*, vol. 51, no. 1, Jan. 2019, Art. no. 1.

[12] S. Y. Lee, *Accelerator Physics*, 3rd ed. Singapore: World Scientific, 2012. [Online]. Available: https://cds.cern.ch/record/1425444

[13] C. K. Birdsall and A. B. Langdon, *Plasma Physics via Computer Simulation*. New York, NY, USA: Taylor and Francis, 2005.

[14] M. Schwarz *et al.*, "Flat-bottom instabilities in the CERN SPS," in *Proc. 10th Int. Partile Accel. Conf.*, 2019, pp. 3224–3227.

[15] H. Timko, E. Shaposhnikova, P. Baudrenghien, and T. Mastoridis, "Studies on controlled RF noise for the LHC," in *Proc. 54th ICFA Adv. Beam Dyn. Workshop High-Intensity High Brightness High Power Hadron Beams*, 2015, Art. no. THO4LR03.

[16] H. Timko, D. Quartullo, A. Lasheen, and J. E. Müller, "Benchmarking the beam longitudinal dynamics code BLonD," in *Proc. 7th Int. Part. Accel. Conf.*, 2016, Art. no. WEPOY045.

[17] J. MacLachlan, "Particle tracking in E-ϕ space for synchrotron design and diagnosis," *Proc. Int. Conf. Appl. Accelerators Res. Ind.*, vol. 24, no. 11, 1992.

[18] A. Shishlo, S. Cousineau, J. Holmes, and T. Gorlov, "The particle accelerator simulation code PyORBIT," *Procedia Comput. Sci.*, vol. 51, pp. 1272–1281, 2015.

[19] V. Forte, E. Benedetto, A. Lombardi, and D. Quartullo, "Longitudinal injection schemes for the CERN PS booster at 160 MeV including space charge effects," in *Proc. 6th Int. Part. Accel. Conf.*, 2015, Art. no. MOPJE042. [Online]. Available: http://accelconf.web.cern.ch/AccelConf/IPAC2015/papers/mopje042.pdf

[20] M. Borland, "ELEGANT: A flexible SDDS-compliant code for accelerator simulation," Argonne Nat. Lab., IL, USA, 2000, doi: 10.2172/761286. [Online]. Available: https://www.osti.gov/biblio/761286

[21] Y. Wang and M. Borland, "Pelegant: A parallel accelerator simulation code for electron generation and tracking," in *Proc. AIP Conf.*, 2006, pp. 241–247.

[22] J. King, I. Pogorelov, K. Amyx, M. Borland, and R. Soliday, "GPU acceleration and performance of the particle-beam-dynamics code elegant," 2017, *arXiv:1710.07350*.

[23] E. Shaposhnikova, J. Repond, H. Timko, T. Argyropoulos, T. Bohl, and A. Lasheen, "Identification and reduction of the CERN SPS impedance," in *Proc. 57th ICFA Adv. Beam Dyn. Workshop High-Intensity High Brightness High Power Hadron Beams*, 2016, Art. no. TUAM3X01.

[24] D. Quartullo *et al.*, "Studies of longitudinal beam stability in CERN PS booster after upgrade," in *Proc. 8th Int. Part. Accel. Conf.*, 2017, pp. 4469–4472.

[25] D. Quartullo, E. Shaposhnikova, and H. Timko, "Controlled longitudinal emittance blow-up using band-limited phase noise in CERN PSB," *J. Phys.: Conf. Ser.*, vol. 874, 2017, Art. no. 012066.

[26] A. Lasheen, E. Radvilas, E. Shaposhnikova, T. Roggen, T. Bohl, and S. Hancock, "Single bunch longitudinal instability in the CERN SPS," in *Proc. 7th Int. Part. Accel. Conf.*, 2016, Art. no. TUPOR009.

[27] J. Repond, K. Iliakis, M. Schwarz, and E. Shaposhnikova, "Simulations of longitudinal beam stabilisation in the CERN SPS with BLonD," in *Proc. 13th Int. Comput. Accel. Phys. Conf.*, 2018, Art. no. TUPAF06.

[28] E. Métral *et al.*, "Beam instabilities in hadron synchrotrons," *IEEE Trans. Nucl. Sci.*, vol. 63, no. 2, pp. 1001–1050, Apr. 2016.

[29] FCC Collaboration, "FCC physics opportunities: Future circular collider conceptual design report," *Eur. Phys. J.*, vol. 79, 2019, Art. no. 474.

[30] Minimize cash flow among a given set, 2022. [Online]. Available: https://www.geeksforgeeks.org/minimize-cash-flow-among-given-set-friends-borrowed-money/

[31] D. B. Kirk and W. H. Wen-Mei, *Programming Massively Parallel Processors: A Hands-On Approach*. San Mateo, CA, USA: Morgan Kaufmann, 2016.

[32] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," *Comput. Sci. Eng.*, vol. 12, no. 3, 2010, Art. no. 66.

[33] S. Chetlur *et al.*, "cuDNN: Efficient primitives for deep learning," 2014, *arXiv:1410.0759*.

[34] M. J. Abraham *et al.*, "GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers," *SoftwareX*, vol. 1/2, pp. 19–25, 2015.

[35] A. Klöckner *et al.*, "PyCUDA: GPU run-time code generation for high-performance computing," 2009, *arXiv:0911.3456*.

[36] S. V. D. Walt, S. C. Colbert, and G. Varoquaux, "The NumPy array: A structure for efficient numerical computation," *Comput. Sci. Eng.*, vol. 13, no. 2, pp. 22–30, 2011.

[37] G. F. Pfister, "An introduction to the InfiniBand architecture," *High Perform. Mass Storage Parallel I/O*, vol. 42, pp. 617–632, 2001.

[38] W. Huang, G. Santhanaraman, H.-W. Jin, Q. Gao, and D. K. Panda, "Design of high performance MVAPICH2: MPI2 over InfiniBand," in *Proc. 6th IEEE Int. Symp. Cluster Comput. Grid*, 2006, pp. 43–48.

[39] E. Gabriel *et al.*, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Proc. Eur. Parallel Virtual Mach./Message Passing Interface Users' Group Meeting*, 2004, pp. 97–104.

[40] W. Gropp, "MPICH2: A new start for MPI implementations," in *Proc. Eur. Parallel Virtual Mach./Message Passing Interface Users' Group Meeting*, 2002, pp. 7–7.

[41] R. Rabenseifner, G. Hager, and G. Jost, "Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes," in *Proc. 17th Euromicro Int. Conf. Parallel Distrib. Netw.-Based Process.*, 2009, pp. 427–436.

Konstantinos Iliakis received the diploma in electrical and computer engineering (ECE) from the National Technical University of Athens (NTUA), Greece. He is currently working toward the PhD degree in the area of HPC architectures and data complex applications' acceleration. In his diploma thesis, he designed and implemented a novel Map-Reduce runtime for shared-memory multi-/many-cores that managed to relax the synchronization between the map and reduce phases. He received the best paper award in the 17th ACM International Conference on Computing Frontiers (CF'20).

Helga Timko received the PhD degree in theoretical physics from the University of Helsinki, Finland, in 2012. Since then, she has been working with CERN in the field of beam dynamics and is responsible for the radio-frequency system operation in the Large Hadron Collider as well as theoretical, simulation, and measurement studies preparing for future operation with higher beam intensities. She started developing the Beam Longitudinal Dynamics code BLonD in 2014 and has been leading the developer team thereafter.

Sotirios Xydis (Member, IEEE) received the PhD degree in electrical and computer engineering from the National Technical University of Athens (NTUA), Greece, in 2011. He is an assistant professor with the Department of Informatics and Telematics, Harokopio University of Athens. He has worked for two years as post-doctoral researcher with the Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, and for several years as a senior research associate with the Microprocessors and Digital Systems Laboratory, NTUA. He has published more than 100 papers in international journals and conferences, and he has received three best paper awards.

Panagiotis Tsapatsaris received the diploma in electrical and computer engineering (ECE) from the National Technical University of Athens (NTUA), Greece. In his diploma thesis, he worked on the development and evaluation of the GPU-accelerated BLonD software used for beam dynamics simulations. His research interests include high performance computing. He is currently employed in Capture One.

Dimitrios Soudris received the PhD degree in electrical engineering from the University of Patras, in 1992. He was a professor with the Department of Electrical and Computer Engineering, Democritus University of Thrace for 13 years. He is currently a professor in the ECE School, Department of Computer Science, National Technical University of Athens, Greece. His research interests include embedded systems design, reconfigurable architectures, reliability, and low power VLSI design. He has published more than 340 papers in international journals and conferences, and co-authored seven books.