Software performance, portability and development *(report)*

HEP Software Fundation, 2017 : ... the physics programs of the planned and/or upgraded HEP experiments over the next 10 years will require the HEP community to address a number of challenges in the area of software and computing. It is expected that the computing models will need to evolve and ***a significant software upgrade is required*** *...*

Gathered on June 2022, on behalf of the whole Reprises team, with direct written contributions from : David Chamont, Hadrien Grasland, Bogdan Vulpescu, Pierre Aubert, Claude Mercier, Arnaud Beck.

Summary

As usual, we expect more and more performance from scientific software. In particular, we hope to absorb part of the massive influx of data, by writing "better" code, to "better" take advantage of the hardware. But this hardware has come up against the wall of unsurpassable frequency, and it is changing profoundly. The cores are multiplying, specializing, and being equipped with accelerators, making obsolete the distributed-sequential computing model of the high energy physics community. The programming of this hardware is becoming more complex, and the marriage between performance and portability is becoming a permanent challenge, especially within our multi-site grids.

At the same time as the HEP Software Fundation was making this observation, and following an IN2P3 Computer School on heterogeneous programming, the institute validated the DecaLog master-project, and in particular its sub-project Reprises, focused on portable performance (speed and accuracy). The aim is to evaluate the myriad of technologies that claim to answer this problem, and to provide physicists with recipes to select and use them autonomously.

After several years of exchanging feedback, the ten or so engineers involved have begun to deliver some collective products, starting with a strong contribution to the institute's prospective. A server dedicated to teaching computing has made its first appearance during a face-to-face workshop, and we are working on a guide that we hope to inaugurate for the Journées Informatique in the fall of 2022.

Among the main performance areas to be exploited, we have particularly explored CPU vectoring, GPU and FPGA acceleration, and precision reduction. After collaborations with colleagues from CEA or EDF, with a "high performance computing" culture, we have also now established strong relationships with computer science laboratories, around co-supervised theses.

Whether it's a question of writing portable vectorized code, exploiting various GPUs, or practicing hybrid precision, many solutions come from expert-like technologies (C++/Fortran), even though the younger generations are more and more addicted to Python. Should we be concerned about this?

If the commitment of the participants makes it possible to keep up to date with the latest developments, if solid links have been established with the HPC community and then with the computer science research community, and if the first collective successes are achieved, one

can be concerned about the fragility of an ecosystem that depends on the opportunistic use of equipment financed by others, and on the good will (generally present) of the management of the multiple laboratories where the too rare specialists are located.

Motivation and stakes

In high-energy physics, we have lived through a long reign of ordinary hardware-based computing grids, where we practiced "embarrassing parallelism", distributing our batches of events to thousands/millions of cores, each running isolated sequential programs. But the hardware has changed a lot and continues to change, towards more and more cores (and less memory per core), longer and longer vectors, and an increasingly heterogeneous and specialized mix of cores and accelerators (CPU/GPU/FPGA). The bottlenecks are shifting, and our model of distributed-sequential computing has to be revisited, in favor of a **parallel approach in all its forms: multi-threading, vectorization, distributed memory, GPGPU**... Our software has to adapt at a forced pace, and our physicists are asked to write thread-safe, functional, vectorized code... without always having clear instructions on how to do it.

Of course, this evolution of computing hardware affects all scientific communities, but we cannot simply observe and imitate what is happening elsewhere, for example in the world of intensive computing. Our disciplines have their own specificities and constraints, which legitimize targeted R&D:

- Rather than "high performance computing" (HPC), focused on the punctual realization of very large simulations or treatments, we practice "**high throughput computing**" (HTC), with the objective of exploiting our computational farms with the maximum efficiency over a long time.
- As the grid is made up of multiple sites, financed by independent sponsors, we end up with a computing infrastructure with very heterogeneous hardware (as opposed to supercomputers). We must therefore favor the most portable software approaches.
- Given the **longer and longer life span** of our experiments, as opposed to the ever faster changes in the computing world, we must favor the most durable approaches, i.e. the ones that are the least close to the hardware, i.e. favoring programming languages with a high level of abstraction, and/or "Domain Specific Languages" (DSLs).
- After a few decades of pushing object-oriented programming to its limits, we handle "natural" **data** structures that are generally hard to manipulate for modern hardware, and particularly **unsuitable for parallelization**. To put it simply, we have a lot of arrays of structures, where we should have structures of arrays.
- A lot of our code is too big (~MLOCs) to be migrated entirely: we need to be able to migrate only the critical sub-parts, within a set that remains "traditional". Or at least, we need to be able to **migrate in stages**.

Master-project and sub-projects

Genesis

Following the 2016 IN2P3 Computer School, dedicated to "Parallelism on Heterogeneous Hardware",
and sharing the observation made by the HEP Software Fundation in 2017, that physics software needed a "refoundation", a group of engineers (and some researchers) proposed to

IN2P3 to create a set of projects around the themes of portable heterogeneous programming, hybrid precision and containers. The first two themes will finally be taken in charge by a single sub-project *Reprises*, and the containers by the sub-project *ComputeOps*, this within a master-project *Decalog*.

A few global recommendations have been made:

- Among the plethora of software technologies available, favor those that preserve portability and durability of code and its performance.
- Strive to collaborate with computer science and applied mathematics laboratories. Publish with them. Look for interns. Imagine co-supervised theses.
- Strive to have some physicists in the team. Contribute to the collection of tutorials that will be used to disseminate the know-how to the largest possible technical and physics audience. Organize collective face-to-face sessions.
- Organize face-to-face meetings once or twice a year, on the model of the first IN2P3 HPC-HTC day, which brought together about 30 people who were already working on these topics individually.

In the rest of this document, we will mainly talk about activities related to the "Reprises" sub-project and its sympathizers. If the central theme is the *Portability of Performance* (facing more and more heterogeneous hardware), it quickly appeared other *P* in the problematic, intricate one to another:

- Precision: new computational hardware, often oriented towards automatic learning, reinforces the return of reduced floating-point types for computation, and one must make sure that this does not jeopardize the results, and does not exacerbate the numerical discrepancies that can appear from one hardware to another.
- Productivity (of the developer): as most of the code is written by non-expert researchers (in computer science), it is necessary to select affordable technologies.
- Perenniality: beyond the simple portability from one hardware to another, we are of course sensitive to the fact that a code can last without the need for too frequent updates or rewriting over time.
- Profiling: no performance is possible without profiling of computing and data exchange times.

The central objective is to study the new hardware used for computation (CPU, GPU), and the software evolution necessary to take advantage of it efficiently, while preserving the accuracy of the computation, the portability, and the durability of the written code. By portability, we do not mean the simple possibility to compile, but above all to have reasonable performances considering the hardware capabilities.

The scope of *Reprises* does not include:

- hardware not available in the short term in our computing centers, such as quantum computers,
- machine learning, but rather the lower-level programming of GPUs that makes it possible,
- distributed computing (we limit ourselves to optimization at the level of a single machine),
- operating systems other than Linux.

First period (2017-2019)

The activities of Reprises started around 11 engineers, from 7 laboratories: IPHC, IPNO, LAL, LAPP, LLR, LPC, LUPM. The equivalent in FTE/year is around 1.5, and probably doesn't pay full tribute to the energy and time devoted by all of them, often in the context of their official physics collaborations. Half of Decalog's budget, about 6 k€/year, essentially allowed them to meet once or twice a year, as well as a few participations in conferences and workshops.

At that time, the main external partners were neighbors and natural partners in the framework of the Labex P2IO of the University Paris-Sud: IAS, SERMA (CEA), EDF. P2IO also financed the ACP (Accelerated Computing for Physics) platform with 43 k€ (AMP EPYC server with U280 Xilinx FPGA board), which the Reprises engineers largely benefited from for their tests.

The activities of the group were essentially based on monthly tele-meetings where participants offered feedback on the tools and technologies evaluated in their respective physics collaborations. Three technical themes quickly emerged, around what are the main performance pools to be exploited, while striving to remain portable:

- CPU vectorization,
- GPU and FPGA acceleration,
- the reasoned reduction of precision.

The group has thus been able to increase its competence, to the point of collectively producing a significant contribution to the IN2P3's 2019 prospects. In terms of dissemination of these skills outside the group, we will also note:

- 5 presentations at the JIs 2018,
- contributions to the IN2P3 computer school on functional programming,
- contributions to the 3rd ASTERICS-OBELICS International School,
- 2 presentations at the GPU@CC-IN2P3 workshop
- 7 publications (a majority in conference proceedings).

Note that during this same period, HPC teams are also mobilized across the Atlantic on the theme of portable performance, as evidenced by a site linked to the DOE (https://performanceportability.org/) and the P3HPC workshop included in the SuperComputing conference (https://p3hpc.org/). There are also some HPC-like activities at IN2P3, such as plasma simulation with Smilei, which primarily focused on MPI distributed computing, but must also now address the node level optimization, and the performance portability accross the french and european super-computers.

Second period (2020-2022)

As the group communicated with colleagues of the institute, and as participants moved, the list of involved laboratories was extended to Subatech, LPNHE and L2I, while keeping the number of participants and the FTE equivalent rather stable.

The usual budget has been partially renewed, but the pandemic having impacted the travels beyond the forecasts, we have been authorized to reconvert a part of this budget in some hardware investments:

- in 2020, a "mobile" server dedicated to training in computational optimization (Intel 18-core CPU, NVidia Turing generation GPU)
- in 2021, a server with an ARM processor (Ampère Altra Q32-17), which is still being installed.

In addition to regular exchanges on new technologies, which are published at a steady pace, the group has expanded its collective production for other engineers and researchers at the Institute:

- the effort made for the prospectives has been pursued to write a guide in web format, with the ambition of keeping up to date recommendations to physicists; this is still in progress, with the objective of a first communicable version for the JIs of autumn 2022;
- the list of available tutorials is growing, and they can now be given in person, in the most remote vacation villages of the country, thanks to the mobile server that has been carefully configured as a mini computing center, with its "front-end nodes" and its "computing nodes" served by a batch system;
- contributions to doctoral/post-doctoral nuclear physics school have been given [PhyNuBE school] to raise awareness about parallel programming and precision issues among the new generation of physics software developpers.
- a new school on heterogeneous programming is planned for 2023, currently submitted to the CNRS as a thematic school, 7 years after the one where everything started.

On the side of external partners, the links have weakened for the moment with SERMA and EDF, but we have pursued our collaboration with IAS and our policy of rapprochement with computer science laboratories, and succeeded in launching three co-supervised theses:

- LUPM 2019-2022: Optimization of the simulation of atmospheric cascades for gamma-ray astronomy experiments. With Philippe Langlois (LIRMM).
- IJCLab 2021-2024: Configuration and control of the accuracy of the calculation, application to low-energy gamma-ray measurements. MITI funding. With Fabienne Jezequelle (LIP6).
- IJCLab 2021-2024: GPU and performance portability - heterogeneous approaches and applications. Funded by UPSaclay. With Joel Falcou (LISN).

Some other highlights:

- participation to JIs'20, JIs'21 and CHEP'21 ;
- 8 publications (3 journals, 4 proceedings, 1 invited presentation).

Technical highlights

Below, we detail a little more the major topics that have caught our attention, namely vectorization, GPUs, FPGAs and floating-point accuracy.

Vectorization

Vectorization is one method to perform multiple identical computations on different data at the same time in a single CPU core.
Nowaday, vectorization is crucial to use CPU effectively. Since 2018, CPUs can perform up to 16 floating point operation per core at the same time. Thus, not using this computing opportunity leads to a direct lost of more than 93% of the archivable peak performance, even with multi-threading.

Recent compilers are able to vectorize simple computations but generally fail if it becomes too complex. However, many solutions are developped to solve this issue. Solutions such as SYCL, Eve, xtensor, xsimd provide high or low level API to address both vectorization and

architectures abstraction. Libraries such as LAPACK, BLAS or ATLAS provide vectorized functions for linear algebra computation.

When the issue of the vectorization is solved, comes the variety of vectorisable architectures (i.e. Intel sse2, ssse3, sse4.1, sse4.2, AVX, AVX2, AVX512f, AVX512wb, etc). Precompiling binaries for almost all previous architectures is needed but tedious and generally can lead to bugs. Again, compilers are able to create binaries specialised on several architectures, but only for computation they can well optimise. Otherwise, linking has to be done manually, or the application has to be compiled on the fly (this is generally the choice of python libraries such as numba, rapids or cunumeric). This solution works, but compilers have to be deployed on computing nodes, and executed when other jobs are computing or performing I/O operations. HPC Proxy, developped by Reprises' members, allows to generate an intermediate library, allowing to choose at runtime the implementation adapted to the current hardware. Libraries generated with HPC Proxy are able to work together in order to diminish the number of targeted architectures, and simplify installation managing.

GPU/FPGA

Much virtual ink has been spilled on the appropriateness of using so-called accelerators, and the right way to program them. Of those, the most advanced studies right now concern GPUs and FPGAs.

GPU

Pioneers have generally acquired expertise in CUDA hardware and software, and prefer to cultivate that expertise and take advantage of the new features that NVidia keeps adding to keep ahead of the competition. This has the obvious drawback of strengthening NVidia's monopoly on GPU computing, enabling one company to single-handedly direct the evolution of GPUs in directions that may not be beneficial to scientists, for example by increasing hardware prices or priorizing features that are hard to apply to scientific research like neural network inference.

OpenMP, a hallmark of HPC, has also been adapted to GPU use for a long time, and HPC tools such as Smilei currently favor this approach, especially for the use of AMD GPUs, and soon Intel ones. Yet, OpenMP's directive-based programming model and very low-level configuration mechanism (where matters such as loop unrolling must be tuned explicitly) makes it hard to use in higher-level programming languages like C++, especially if good performance on multiple hardware targets is desired. A higher-level alternative called OpenACC was devised, but it was never well supported outside of the NVidia ecosystem, and remains tedious to use from C++. True to their directives approach, the Smilei team mixes OpenMP and OpenACC with the help of preprocessor instructions. As is almost always the case, simplification and portability for the developer comes at the cost of increased complexity of the compilation machinery.

OpenACC can also be used for Fortran environment where the Matlab-like array expressions gracefully capture vector-type parallelism. At IJCLab, it has proved efficient on a theoretical nuclear astrophysics code to compute electron capture cross-section in nuclei during neutron star collapse. Moving to GPU provided some speedup, strongly depending (from one to five) on hardware generation and amount of memory, but a speedup of two already came from the rewriting of code in vectorizable array expression, on top of a speedup of fifty from optimizing the naive sequential code. Many tracks can yet be explored: newer generations of GPU have new compute capabilities, such as direct matrix products, as well as larger memory, splitting

the work across multiple GPU should also be assessible in OpenACC's high-level language. Operational issues have been a burden, when sharing the GPU between different Operating Systems in containers: GPU systems version are changing often, and should live with a recent OS and software stack for development purposes, while the lab central servers have to stay with older and more stable OS.

Advocates of less proprietary solutions initially focused on OpenCL, invoking easier portability to Intel's then-popular Xeon Phi platform. But OpenCL is another tedious API to program, and Intel has since abandoned the Xeon Phi hardware line. This, together with the need for direct hardware manufacturer support (which NVidia will only provide with extreme moderation) makes OpenCL a less popular choice these days, though it is still useful for embedded computing and FPGAs.

The Khronos group, who maintain the OpenCL specification, has also introduced SYCL, a single-source C++ programming model whose look and feel is closer to that of CUDA. Initially meant to be an abstraction/convenience layer atop OpenCL, SYCL has more recently evolved into a backend-agnostic direction, allowing direct implementation over manufacturer-specific APIs like CUDA, which bypasses the need for direct manufacturer support that held OpenCL back.

SYCL's popularity has been on a steady rise since Intel has made it a central part of its new oneAPI programming model, as part of its ongoing return to the GPU manufacturing business. As long as one pays close attention to Intel's tendency to liberally promote extensions that are not part of the standard SYCL specification when said specification does not progress fast enough for their taste, this is a very positive evolution.

The use of SYCL for HEP particle tracking is being investigated at IJCLab, among other places and notably led to the "Comparing SyCL data transfer strategies for tracking use cases" publication at ACAT 2021.

Before SYCL became popular, another approach that was widely taken was to have an abstraction layer that can dispatch to either OpenMP, CUDA, or HIP (an AMD technology that provides a CUDA-like API on top of either CUDA itself or AMD's software stack). Examples include Raja, Kokkos and Alpaka, as well as offerings taking a more aggressive code generation approach like SkePU. Many of these frameworks also attempt to provide higher-level programming primitives (data structures and/or algorithms), which means that there might still be a benefit to using them, especially if they managed to support SYCL as a backend (which so far has proven very difficult as they make lots of CUDA-specific assumptions in their implementation).

But these same high-level constructs also often make it harder to port existing CUDA codebases into code that make idiomatic use of the framework's higher-level constructs, and introduces a higher risk of ending up betting on a software project that will ultimately prove to be a dead end. Especially given that at this point in time, there is no clear winner emerging among all these options : while Kokkos is generally most popular, some projects also have a strong political stance against it as a result of it being engineered by proud atomic bomb manufacturers.

As programming language standards evolve, it also becomes possible to envision making GPU programming part of the language's standard capabilities, without any need for supplementary frameworks or APIs. An early example of this is provided by NVidia's newer compilers, which allow standard C++ or Fortran programs using the language-provided abstract parallel programming facilities to be transparently ported to the GPU just by rebuilding them with the vendor-supplied compiler, often with reasonable execution performance.

This standards-driven approach does not yet provide a full GPU programming model matching the capabilities of other approaches, however. For example it does not yet include explicit data movement primitives, instead relying on the GPU driver's ability to automatically move data between the CPU and the GPU, which may result in suboptimal performance. Mostly because programming languages themselves do not yet have such primitives in their standard vocabulary.

Given SYCL's intent to ultimately become part of C++'s standard feature set, there is hope that this will eventually change, but at the pace of C++ evolution such a change could take many years to materialize, and even longer to be supported by compilers other than those of each individual GPU manufacturer. We will come back to why the latter is a problem later on.

An even higher-level alternative is provided by the use of ready-made libraries, mostly provided by GPU manufacturers, implementing common computations (linear algebra, fast Fourier transform, neural network training...) using the GPU under the hood. This is the approach that is most commonly used in programming languages like Python, and that should be especially encouraged for programmers who are not GPU experts, although it only works well where a large chunk of the computation to be performed is already available as a pre-made library.

Finally, a section on GPU computing would not be complete without mentioning the growing issue of vendor software stacks. Unlike in CPU computing, where widely available compilers such as gcc and clang can target pretty much every popular CPU architecture out there, GPU computing tends to be tied to software that is directly provided by GPU vendors. Such software is often less mature than equivalent CPU offering, for example it tends to be significantly harder to install and to feature compilers that will reject valid standard code constructs. Combined to the general unavailability of vendor support staff, this raises new software portability challenges.

Vendors also remove support for older hardware in newer software, partly to encourage new purchases, which raises questions of planned obsolescence and computation repeatability. Here, it would be beneficial to investigate avenues for archiving older vendor software stacks before they disappear, and for streamlining the general process of installing GPU computing stacks.

FPGA

In the HEP, the FPGAs are generally known as parts of the read-out chains, treating the detector signals coming from the read-out cards and applying more or less complex algorithms, being capable to generate selection decisions like the event triggers. This can already be seen as a non-standard calculus and it was the starting point to think about asking more from this kind of hardware resources. Two technologies existed on the market, Altera and Xilinx, the latter created by the inventor of the FPGA. OpenCL, a product of the Khronos group, was quickly adopted as the standard for implementing real computing on FPGAs, and enthusiastic volunteers – and specialists in “direct” FPGA programming - started to work by the side of the FPGA manufacturers to build the so called Board Support Packages, containing the necessary basic FPGA firmware allowing to translate the OpenCL language units in an optimal way (an FPGA is tunable in almost every aspect, thus quite tricky to adjust).

In 2017, the electronics department of CERN organized a seminar followed by a hands-on workshop with experts from Intel (fresh owner of Altera at that time), which represented the start of a partnership between CERN and Intel. The trigger-less read-out of the LHCb detector was the first one to test a combination of Intel Xeon and FPGA, on a two socket server. On the

Machine Learning side, Xilinx occupied quickly the place with the HLS4ML package (High Level Synthesis for Machine Learning, with a Python interface) for fast inference of deep neural networks used for level 1 trigger and data acquisition.

This event was pushing forward the interest for this domain and established this subject within the Reprises project from the very beginning.

Unfortunately, the harsh competition between Intel and Xilinx (now part of AMD) put an end to the “romantic epoch” of playing around by programming all kind of code on the FPGAs using plain OpenCL and studying the performances in speed and numerical precision. Today the flagship is Machine Learning and other AI applications and the two companies have narrowed their activity towards the production of powerful computing solutions (and quite expensive), with targeted software and applications (AMD/Xilinx seems to be slightly ahead). The dynamic on the market of their products and their prices demands a serious commitment and important resources in order to keep track with the rapid evolution in this field (CERN was able to raise such a task force).

Many things have been learned during those years and their large majority is still valid. The accumulated expertise allows us to efficiently follow what is happening now and what will come in the next years and, more useful, to provide reasonable counsels for starting to invest in and use such hardware resources for the HEP projects in the IN2P3 laboratories.

Precision

While reducing the precision may keep accuracy of the computation good enough for the scientific goal and provide some speedup at the same time, this equilibrium must be tested experimentaly: beyond the measurement of speedup, the code has to be converted, but also sometimes tuned, for other precisions than the default one and the achieved accuracy for these other precisions must be evaluated.

Yet another aspect is to obtain a definite statement on the accuracy goal of the collaboration: the computing aspect of the knowledge extraction process must be assessed in the same way as, for instance, the detector aspect: the uncertainties, both systematic and statistical, have to be studied carefully. In our case, uncertainties exist at the levels of the numerical method, of implementation and specific set of libraries, of the rounding settings as well as the selected level of precision. An article has been written (`Revisiting "What Every Computer Scientist Should Know About Floating-point Arithmetic"`) to provide examples of good numerical practices relevant to the physics community. Furthermore, these numerical aspects have been included in a lecture about good computing practices given at PhyNuBE school (Low Energy Nuclear Physics) (https://indico.in2p3.fr/event/20625/).

As for the precision conversion, we usually rely on the generic programming paradigm, with varied ease of use within different languages ecosystems, but this is only the tip of the iceberg: the numerical methods may also change with the precision level (such as expansion to higher order requiring new coefficients). C++ provides easy ways to access genericity, and even modern Fortran allows enough simple genericity to adress these issues.

Concerning the control of the accuracy of floating-point computing, the French computer science community seems to be at the forefront, in particular on stochastic arithmetic tools. The authors of the three best-known tools (Cadna, Verrou and Verificarlo) are now cooperating within the InterFLOP consortium (https://www.interflop.fr/).

The engineers of IN2P3 are following this movement closely. After having cooperated with the authors of Verrou (`Floating-point profiling of ACTS using Verrou`), a thesis is co-supervised with Fabienne Jezequel, around the use of CADNA on a nuclear physics code (Agata collaboration Pulse Shape Analysis). Our first works around CADNA have focused on the compatibility of the tool with some of the most important libraries of the domain : Eigen, ROOT, Eigen being a template library proved easy to instrument. Basic comparisons for matrix inversion and diagonalisation have confirmed the stability of the algorithms, even for ill-conditioned matrices.

The work on Pulse Shape Analysis confirmed the memory-boundness of the algorithm: 90% of time is lost in cache-misses. Usual speedup strategies such as vectorization have no effect in this case. We have plans to reduce the precision to half-precision (FP16 soon and even BF16 for a future GPU version) knowing that we can assert the resulting accuracy with CADNA.
What is more, CADNA has shown a number of numerical instabilities in the algorithm. We also plan to test fixed point computations, which should prove useful in online computing.

Some recommendations are easy to implement for non-experts, and particularly on new projects: define all floating point variables with a generic type, and then be able to change it for every compilation between FP32 (float), FP64 (double), FP80 (long double). For Precision, as well as for Performance, some dedicated metrology has appeared necessary, and non-experts should contact reference members of the collaboration.

SWOT-like final comments

Internal strengths

The design, construction and implementation of very large equipment, especially in particle physics, allows IN2P3 to have a large population of engineers, and also very large equipment in scientific computing, where the heterogeneity of the material is very large, and the portability of the code a primary concern. The institute can and must be at the forefront of this subject and work in concert with CERN and the major laboratories in the field.

External opportunities

The persistent competition between NVidia and Intel allows the former to avoid definitively installing its monopoly on accelerators, and the latter to indirectly advance more portable solutions. Let's hope it lasts...

The dynamism of French computer science research is both an opportunity (especially in floating-point computing) and a threat, in the sense that it controls teaching and attracts the best students in the field. We advocate a systematic cooperation with computer science laboratories, and a concretization of these cooperations around interns and co-supervised PhD students, while remaining vigilant on the fact that computer science researchers may be less concerned than we are about the concrete and short-term application of the latest advances. It is also necessary to offer courses in scientific computing to M2 students.

We must also emphasize that the particle physics community is very organized, notably through the HEP Software Fundation, or European projects such as Aida200 and AidaInnova. By participating in these activities, we can meet international experts, and better motivate industrialists to take our priorities into account.

Internal weaknesses

The management of IN2P3 R&D projects is inspired by the management of physics projects, which leads to some difficulties, starting with the counting of contributions in FTEs. Because they are not always encouraged by their management to spend a significant part of their time on projects of general interest (more general than the laboratory), most of the engineers contribute within the framework of the physics collaborations to which they are attached, and will always prefer to declare most of their FTEs in these collaborations, because this conditions the budgets granted by the institute. This is even more true for physicists. The system thus encourages isolated work, with certainly exchanges of feedback, but to the detriment of transverse productions. Would a more matrix-based counting system allow a better account of R&D contributions?

More generally, since there are many specialties and few specialists, we may dream the institute to find a way to create cross-laboratory teams (if possible without forcing all the institute's engineers to migrate to the CC). In a world that is converting to telework, why not tele-teams IN2P3? But how to (re)compensate the laboratories for the time that these engineers would spend outside, for example in a task force *profiling, performance, portability*? One can also wonder about the fragility of activities that rely on a single specialist at the level of the institute, on subjects as fundamental as floating-point computing for example.

As far as test equipment (and sometimes software) is concerned, we can see that the project has been able to use opportunistically equipment acquired by others (P2IO), or to make some purchases in the very particular context of COVID. But it would be much more efficient to have a centralized platform somewhere, with its own dedicated operating engineer and a permanent budget for acquiring new equipment as it becomes available. CERN had such a structure, the TechLab... which it seems to have stopped. It would be instructive to know why.

Finally, we know the benefits of working in cooperation with computer laboratories. It is a long-term work, which is only fully motivating for all if this cooperation is done through shared doctoral students and interns. This raises the problem of funding. The funding of the trainees is a priori entirely based on thco-supervised doctoratee proreps of the laboratories. Isn't this a handicap for the smallest of them? HdR and technical theses are strongly encouraged... as long as their funding does not come in deduction of the one of pure physics or pure computer science theses, and the windows for inter-disciplinary no ssuejts are few. The current "good" phase is perhaps only a positive statistical fluctuation.

External Threats

While GPU portability tools are more and more focused on advanced languages such as C++, the young recruits of the institute arrive more and more with a basic Python culture, and a reluctance to use other languages, perceived as more complex and dedicated to specific niches in which they do not want to be locked. As time goes by, we could lose our ability to develop powerful frameworks tailored to our problems, and depend on what players such as NVidia, Intel, Google... will provide us.

Also, the ease of writing portable code often comes with increased complexity of backend installation, build mechanics and code distribution. Those who have been around SYCL know this. The institute definitively need more specialists in advanced server administration, software distribution, DevOps and containers.

References

Publications

2021

- A Common Tracking Software Project (https://arxiv.org/abs/2106.13593), Xiaocong Ai et al. (dont **H.Grasland**).
- A C++ Cherenkov photons simulation in CORSIKA 8 (https://doi.org/10.1051/epjconf/202125103011), Matthieu Carrère, **Luisa Arrabito**, Johan Bregeon, David Parello, Philippe Langlois and Georges Vasileiadis.

2020

- Optimizing Cherenkov Photons Generation and Propagation in CORSIKA for CTA Monte–Carlo Simulations (https://arxiv.org/abs/2006.14927), **Luisa Arrabito**, Konrad Bernlöhr, Johan Bregeon, Matthieu Carrère, Adnane Khattabi, Philippe Langlois, David Parello, and Guillaume Revy.
- Revisiting "What Every Computer Scientist Should Know About Floating-point Arithmetic" (https://doi.org/10.48550/arXiv.2012.02492), **Vincent Lafage**

2019

- Adaptive SIMD optimizations in particle-in-cell codes with fine-grain particle sorting (https://arxiv.org/abs/1810.03949) , **Arnaud Beck**, Julien Dérouillat, Mathieu Lobet, Asma Farjallah, Francesco Massimo, Imen Zemzemi, Frédéric Perez, Tommaso Vinci, Mickael Grech.
- A deep neural network method for analyzing the CMS High Granularity Calorimeter (HGCAL) events (https://zenodo.org/record/3599481), **G. Grasseau**, Abhinav Kumar, Andrea Sartirana, Artur Lobanov, Florian Beaudette, Pierre Ouannes.
- Performance optimization of the air shower simulation program for the Cherenkov Telescope Array (https://doi.org/10.1051/epjconf/201921405041), Matthieu Carrère, **Luisa Arrabito**, Konrad Bernlöhr, Johan Bregeon, Gernot Maier, Philippe Langlois, David Parello, Guillaume Revy.

2018

- Polynomial data compression for large-scale physics experiments (https://arxiv.org/abs/1805.01844), **Pierre Aubert**, Thomas Vuillaume, **Gilles Maurin**, **Jean Jacquemier**, Giovanni Lamanna, Nahid Emad.
- Data Analysis with SVD for Physical Expriments, Application to the Cherenkov Telescope Array (http://meetings.siam.org/sess/dsp_talk.cfm?p=89494), **Pierre Aubert**, Thomas Vuillaume, Florian Gaté, **Gilles Maurin**, **Jean Jacquemier**, Nahid Emad, Giovanni Lamanna.
- Floating-point profiling of ACTS using Verrou (https://www.epj-conferences.org/articles/epjconf/abs/2019/19/epjconf_chep2018_05025/epjconf_chep20
Hadrien Grasland, **David Chamont**, François Févotte, Bruno Lathuilière.
- Deployment of a Matrix Element Method code for the ttH channel analysis on GPU's platform (https://www.epj-conferences.org/articles/epjconf/abs/2019/19/epjconf_chep2018_06028/epjconf_chep20
G. Grasseau, F. Beaudette, A. Zabi, C. Martin Perez, A.Chiron, T. Strebler, G. Hautreux.

Before

- Matrix element method for high performance computing platforms (https://iopscience.iop.org/article/10.1088/1742-6596/664/9/092009), G. Grasseau, D. Chamont, F. Beaudette, L. Bianchini, O. Davignon, L. Mastrolorenzo, C. Ochando, P. Paganini, T Strebler.

Reports

- Rapport de prospective pour l'IN2P3 (https://reprises.in2p3.fr/ressource/docs/prospective-gt09-reprises-rapport-v3.pdf)
- Présentation lors du GT09 (https://reprises.in2p3.fr/ressource/docs/prospective-gt09-reprises-presentation-pppp.pdf)

Webography

Reprises

- Webinaire : Retour sur la conférence de Nvidia GTC 2022 (https://indico.in2p3.fr/event/27394/)
- Webinaire : Retour sur la conférence de Nvidia GTC 2021 (https://indico.in2p3.fr/event/25027/)
- Retour sur la conférence de Nvidia GTC 2020 (https://gitlab.in2p3.fr/CodeursIntensifs/Reprises/-/wikis/reunion-2020-05-27)
- Guide Reprises (https://reprises.in2p3.fr/)
- Quatrieme Reprise (2022) (https://indico.in2p3.fr/event/25485/)
- Prospectives IN2P3, GT09 : le point de vue du collectif Reprises (2019) (https://reprises.in2p3.fr/ressource/docs/prospective-gt09-reprises-rapport-v3.pdf)
- Troisieme Reprise (2019) (https://indico.in2p3.fr/event/20057/)
- Deuxieme Reprise (2019) (https://indico.in2p3.fr/event/18881/)
- Premiere Reprise (2018) (https://indico.in2p3.fr/event/17470/)

IN2P3 Computing

- 13e Journées Informatiques IN2P3/IRFU (2021) (https://indico.in2p3.fr/event/25008/)
- Journées R&T IN2P3 (2021) (https://indico.ijclab.in2p3.fr/event/6256/)
- 12èmes Journées Informatiques IN2P3/IRFU (2020) (https://indico.in2p3.fr/event/21046/)
- 11èmes Journées Informatique IN2P3/IRFU (2018) (https://indico.in2p3.fr/event/17206/)
- Projets IN2P3 R&D transverse "Calcul&Données" (2018) (https://indico.in2p3.fr/event/16883/)
- Activités et vision pour le domaine HTC / HPC (2017) (https://indico.in2p3.fr/event/14008/)(un talk de ma part)
- Journée Calcul (2017) (https://indico.in2p3.fr/event/14075/)

Plateforms

- GridCL (http://polywww.in2p3.fr/projet-gridcl)
- ACP (https://gitlab.in2p3.fr/grasseau/acp-public/-/wikis/home)
- CERN Techlab (https://twiki.cern.ch/twiki/bin/view/HardwareLabs/HardwareLabsPublic/TechLab)

CNRS, INRIA, ORAP

- 45e Forum : Quelles précisions pour le HPC ? (2020) (http://orap.irisa.fr/45e-forum-quelles-precisions-pour-le-hpc/)
- 44e Forum : Concilier pérennité et performance, quels défis pour l'Exascale? (2019) (http://orap.irisa.fr/44e-forum-concilier-perennite-et-performance-quels-defis-pour-lexascale/)
- Precision, Reproductibilité en Calcul et Informatique Scientifique (2017) (https://precis.sciencesconf.org/)
- Interflop (https://github.com/interflop/interflop)

AIDA european projects

- AidaInnova WP12 (https://aidainnova.web.cern.ch/wp12)
- Aida 2020 WP (http://aida2020.web.cern.ch/activities/wp3-advanced-software)

CERN

- CERN Compute Accelerator Forum (https://indico.cern.ch/category/12741/)
- ATLAS Heterogeneous Computing and Accelerator Forum (HCAF) (https://indico.cern.ch/category/7001/) (HCAF)
- ACTS Parallelization (https://indico.cern.ch/category/7968/)

HEP Software Fundation

- HEP Software Foundation (http://hepsoftwarefoundation.org/)
- HSF WLCG Virtual Workshop on New Architectures, Portability, and Sustainability (2020) (https://indico.cern.ch/event/908146/timetable/)
- HSF Training (https://indico.cern.ch/category/10294/)
- HSF C++ (https://github.com/hsf-training/cpluspluscourse)

HEP Conferences

- ACAT 21 (https://indico.cern.ch/event/855454/)
- vCHEP 21 (https://indico.cern.ch/event/948465/)
- JLab Software and Computing Round Table (https://www.jlab.org/software-and-computing-round-table)

High Performance Computing

- P3HPC (https://p3hpc.org/)
- Portability Across DOE Office of Science HPC Facilities (https://performanceportability.org/)
- IOWCL (https://www.iwocl.org/)
- ISC-HPC (https://www.isc-hpc.com/id-2020.html)

Schools

- PhyNuBE school: Parallel programming (https://indico.in2p3.fr/event/20625/contributions/101997/)
- PhyNuBE school: Good computing practices for physicists and PhD students (https://indico.in2p3.fr/event/20625/contributions/97682/)
- Ecole informatique IN2P3 "Utilisation de la programmation fonctionnelle dans nos environnements scientifiques" (2019) (http://formation.in2p3.fr/Info19/ProgFonct19.html)
- 3rd ASTERICS/OBELICS international school (https://www.asterics2020.eu/event/third-asterics-obelics-international-school)

- Ecole informatique IN2P3 "Parallélisme sur Matériel Hétérogène" (2016) (https://indico.in2p3.fr/event/13126/)

Lectures

- CADNA (https://www-pequan.lip6.fr/~jezequel/AFAE.html)
- EVE (https://jfalcou.github.io/eve/simd-101.html)
- PERF (https://grasland.pages.in2p3.fr/tp-perf/)
- Introduction au C++ (https://lappweb.in2p3.fr/~paubert/introductioncplusplus/index.html)
- Introduction to code optimisation (https://lappweb.in2p3.fr/~paubert/ASTERICS_HPC/index.html)
- Introduction to Valgrind (https://lappweb.in2p3.fr/~paubert/INTRODUCTION_VALGRIND/index.html)
- Introduction to GDB (https://lappweb.in2p3.fr/~paubert/INTRODUCTION_GDB/index.html)
- Development and optimisation (https://lappweb.in2p3.fr/~paubert/DEVELOPMENT_AND_OPTIMISATION/index.html)
- Performance with Nan and other exotic values (https://lappweb.in2p3.fr/~paubert/PERFORMANCE_WITH_NAN/index.html)
- Introduction à Gitlab (https://lappweb.in2p3.fr/~paubert/INTRODUCTION_GITLAB/index.html)
- Introduction to Maqao (https://lappweb.in2p3.fr/~paubert/INTRODUCTION_MAQAO/index.html)
- Performance with stencil (https://lappweb.in2p3.fr/~paubert/PERFORMANCE_WITH_STENCIL/index.html)

Industrials

- "HPC a clever mix between hardware and software", Pierre Aubert, 2020 for DELL Swiss