Integration of Modern HPC Performance Analysis in Vlasiator for Sustained Exascale

Camille Coti[1], Yann Pfau-Kempf[3], Markus Battarbee[3], Urs Ganse[3], Sameer Shende[2], Kevin Huck[2]
Jordi Rodriquez[2], Leo Kotipalo[3], Allen D. Malony[2], Minna Palmroth[3,4]

[1]École de Technologie Supérieure
Montréal, Québec, Canada

[2]University of Oregon
Eugene, Oregon, USA

[3]University of Helsinki
[4]Finnish Meteorological Institute
Helsinki, Finland

***Abstract*—Key to the success of developing high-performance applications for present and future heterogeneous supercomputers will be the systematic use of measurement and analysis to understand factors that affect delivered performance in the context of parallelization strategy, heterogeneous programming methodology, data partitioning, and scalable algorithm design. The fact is that the evolving complexity of future exascale platforms makes it unrealistic for application teams to implement their own tools. At the same time, it is naïve to expect available robust performance tools to work effectively out-of-the-box, without integration and specialization in respect to application-specific requirements and knowledge. Our work reported here took advantage of an opportunity to integrate the TAU Performance System technologies (TAU and APEX) with a leading plasma physics code, Vlasiator, which is being ported to the LUMI HPC system for advanced modeling of the Earth's magnetosphere and surrounding solar wind. Building on a preexisting performance API, we successfully enhanced the performance measurement and analysis capabilities in Vlasiator with the TAU and APEX tools. Results from initial LUMI performance experiments are presented and lessons learned from our experience offered as possible guidance to other related endeavors.**

Introduction

The dawn of exascale computing brings significant challenges to the development and optimization of advanced scientific applications incorporating state-of-the-art multiphysics modeling and parallel simulation. The convergence of heterogeneous architectures, massive parallelism, highly-scalable interconnects, and integrated I/O capabilities is delivering facilities of extreme computational power. However, harnessing exascale systems for real-world workloads will require sustained application performance. In many cases, codes will need to be (re-)developed with hybrid programming methods to address heterogeneity complexities and performance portability objectives. Furthermore, the factors contributing to performance behavior and scaling efficiency will make performance variability more acute. Thus, to achieve sustainable exascale applications, it will be important to integrate performance analysis technologies and engineering methods that have been built to work in exascale environments.

The paper reports on an ongoing collaboration to integrate exascale performance tools with a leading scientific application being ported to the EU's flagship supercomputer in Finland. The *Large Unified Modern Infrastructure (LUMI)* is an HPE Cray EX supercomputer that is the #3 machine on the Top500 [1] at this time. In addition to helping to produce a sustainable exascale-targeted application, we believe the contributions anticipated from this integration will offer useful guidance and approaches for other exascale application projects.

The TAU project at the University of Oregon has developed a rich suite of performance measurement and analysis tools for high-performance computing (HPC), specifically the *TAU Performance System®* composed of the *Tuning and Analysis Utilities (TAU)* [2] and the *Autonomic Performance Environment for eXascale (APEX)* [3]. These tools are continuously being updated and made ready for exascale systems. The *Vlasiator* project [4]–[6] at the University of Helsinki is developing a next-generation simulation software for modeling the space plasma environment of the Earth. The two project teams have been working together since summer 2022 to incorporate TAU and APEX in Vlasiator and conduct performance analysis and tuning on LUMI. An early result from our efforts included adapting a legacy performance API (Phiprof) to work with TAU and APEX measurement systems. This was an important step that enabled the Phiprof events instrumented by the Vlasiator team to be seen as meaningful events with semantic context by which to organize and interpret the now broader scope of performance data that could be collected.

The paper makes the following research contributions:

- Design and implementation of the Vlasiator Phiprof API for performance instrumentation to operate with TAU and APEX measurement systems.
- Integration of TAU and APEX with Vlasiator and validation of Phiprof measurement with performance observation support TAU and APEX can provide (e.g., MPI, OpenMP, hardware counters, profiling/tracing).
- Running of initial Vlasiator performance experiments on HPC systems, in particular LUMI-C (CPU partition), to gain experience in using the tools, explore interesting performance outcomes, and inform directions for further performance analysis.
- Progress on Vlasiator porting to LUMI-G (GPU partition) and performance tool requirements.

The organization of the paper is as follows. Section§I describes the Vlasiator application and outlines its performance analysis challenges. Section§II describes the TAU Performance System. Our work to adopt the Phiprof API is discussed in Section§III. Some of the early experiments with Vlasiator and its new integrated performance support on LUMI are reported in Section§IV. We emphasize here more of the process of learning about the tools and how best to apply them, instead of focusing necessarily on optimization outcomes. Related work is presented in Section§V and Section§VI gives conclusions and outlines future work.

I. Vlasiator

A. Overview

Vlasiator is a simulation software for modeling the collisionless space plasma physics of the Earth's magnetosphere and surrounding solar wind. The understanding of the interaction of the solar wind and its perturbations such as coronal mass ejections originating from the Sun is critical in improving the prediction capabilities for space weather. The latter comprises all phenomena in and from space that have potential impacts on human life and technology, such as GNSS perturbations or ground-induced currents.

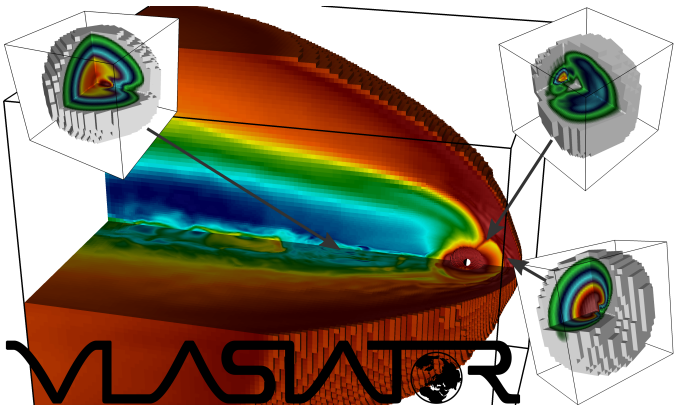

Fig. 1. Vlasiator simulation of the interaction of the supersonic solar wind with the Earth's magnetosphere. A shock and sheath form, encompassing the magnetosphere, in three spatial dimensions. The insets highlight the phase-space density distribution in the three velocity dimensions at selected locations.

B. Numerical model

Describing space plasma phenomena requires equations modeling the behavior of charged particles in dynamic electromagnetic fields. Direct numerical simulation based on Newton's and Maxwell's equations is intractable for a macroscopic system, hence the use of statistical approaches to derive more suitable equations. Vlasiator simulates the evolution of ion phase-space density using the collisionless Boltzmann equation under the exclusive action of electromagnetic fields, also known as the Vlasov equation. By neglecting the short space- and time-scale dynamics of the much lighter electrons, the method becomes what is called the hybrid-Vlasov model of plasma physics, used to simulate the behavior of ions, and in the context of the solar wind primarily protons (H^+ ions). [4].

The Vlasov equation can be understood as a six-dimensional advection equation, where flows in 3D position space are governed by the velocity of plasma, and flows in 3D velocity space (i.e., acceleration) are governed by the acceleration provided by the Lorentz force acting on the ions. This logical decomposition of the physics is reflected in the solution that is split in separate spatial (translation) and velocity (acceleration) steps that are intertwined in a leapfrog fashion [4], [7], [8]. These translation and acceleration solution steps use a semi-Lagrangian approach [9] which helps in relieving the CFL condition imposed by finite-volume methods. The ion velocity distribution function is represented and propagated on an adaptive Cartesian 3D-3V grid [10]. The spatial grid can be cell-based refined to increase resolution in regions of interest [11], either using parametrized regions or refining adaptively during runtime [12]. The velocity grid is sparse, storing and propagating ions only in regions of velocity space with non-negligible phase-space density [5].

The closure of the equation system is provided by Maxwell's equations, solved by an upwind constrained-transport electromagnetic field solver [13] that is computationally lightweight compared to the 3D-3V Vlasov propagation. Therefore the choice was made to keep the field solver acting on a separate, uniform grid at a resolution matching the AMR spatial grid's finest resolution [14], [15]. The closure of the Maxwell equation system enforces a fluid description of electrons, the assumption of quasineutrality – the modeled system cannot exhibit or sustain large-scale charge gradients, and is represented through Ohm's law with the Hall and electron pressure gradient terms included [16].

C. Software

Vlasiator is written in C++17, hybrid parallelized, using CPU SIMD vectorization, OpenMP threading, and decomposition of simulation space over MPI domains, with the Zoltan [17] library performing recursive coordinate bisection or, more recently, recursive inertial bisection load-balancing in the spatial domain based on the number of phase-space cells to propagate in each spatial cell. Support for SIMT GPU instructions is under development for both the Vlasov solvers and the field solver.

D. HPC-relevant characteristics

Compared to many other kinetic plasma simulation codes, which are typically based on the Particle-in-Cell (PiC) methodology [18], which make up a significant fraction of HPC resource use, Vlasiator's Eulerian grid structure and cell-based mesh refinement approach mean that the memory access patterns and scalability challenges it faces are quite distinct. The 3D-3V space would require massive resources in terms of memory and computations if sampled uniformly to the full physically potentially accessible extents. Therefore, Vlasiator uses a sparse approach whereby the phase-space density is stored only in regions above a set threshold (see Fig. 1 and

[5]). This means however that data is constantly moving in physical space as well as memory, leading to continual dynamics and e.g. the necessity to redefine MPI datatypes to communicate the distribution function parts that move from one task's domain to the next at every step.

A further characteristic of (space) plasma simulations are the large contrasts in plasma density and temperature, which directly map to large contrasts in the absolute number of sample points retained in 3V velocity space at different 3D locations. Typically, these contrasts range about three orders of magnitude between different regions of the simulation, directly mapping to equivalent disparities in the computational load distribution across the simulation domain. Figure 2 illustrates the difference in domain size between computationally heavier and lighter regions of the domain.

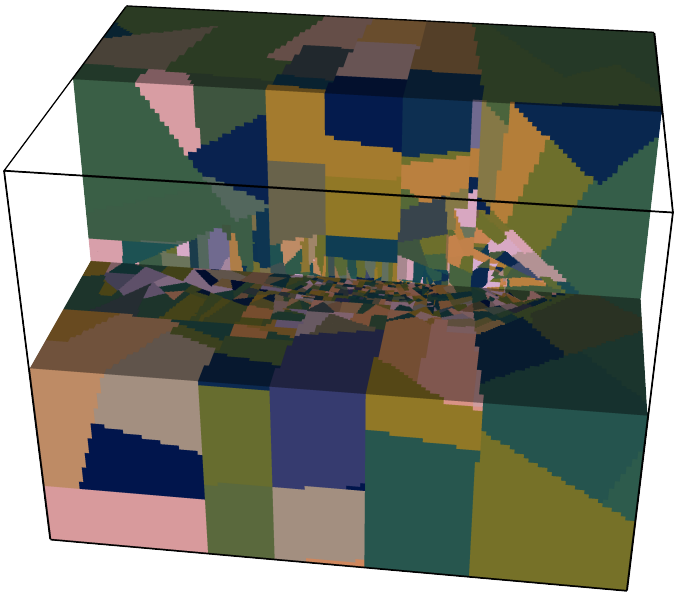

Fig. 2. Illustration of the 3D domain decomposition in a 3D-3V Vlasiator run using recursive inertial bisection with Zoltan [17].

The historical development strategy of Vlasiator has been to target the next-generation supercomputers, so that current algorithmic development efforts allow to leverage planned architectures. Figure 3 illustrates how the major milestones in algorithm development have allowed to reach performance levels compatible with the modeling of the Earth's magnetosphere on a coarse 2D-3V grid first, reaching higher resolutions thanks to the use of the semi-Lagrangian solver, before finally reaching the goal of modeling on a full 3D-3V grid after the development of the adaptive grid solver. The next critical breakthroughs, non-uniform timestepping across the domain and the use of GPUs to accelerate the major Vlasov and field solvers, will allow to extend the total physical time of the simulations, so that typical magnetospheric events such as the development of geomagnetic substorms will be achievable.

II. TAU Performance System

One truism of *high-performance computing* (HPC) is the importance of assessing how well HPC systems meet the

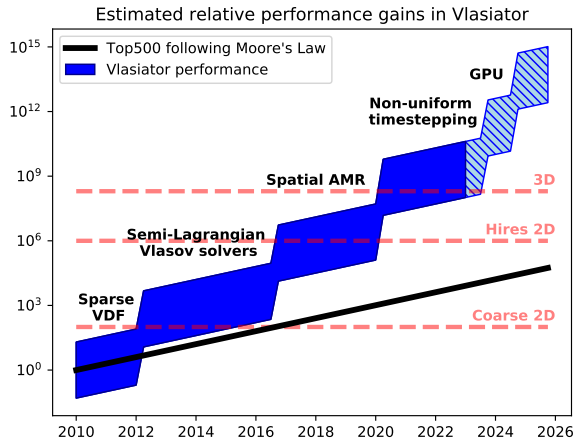

Fig. 3. Estimated evolution of the relative performance of Vlasiator as a function of time and major algorithmic improvements allowing to reach levels required to model Earth's magnetosphere in coarse and high-resolution 2D-3V, and finally 3D-3V setups.

performance expectations of the applications that run on them. It is about performance afterall. Fundamentally, parallelism in its various forms in HPC hardware and software is what make performance observability of actual parallel computations on real HPC machines a hard problem. State-of-the-art performance tools play an important role in helping to understand application performance, diagnose performance problems, and guide tuning decisions on modern parallel platforms. However, performance tool technology must also respond to the growing complexity of next-generation parallel systems and how they are programmed in order to help deliver the promises of HPC.

Presently, the advances in HPC hardware and systems have made it possible to develop parallel applications of greater sophistication and power for purposes of achieving more ambitious objectives in computational and data science domains. With the potential for scalable parallelism, heterogeneous execution, massive concurrency, and low-latency/high-bandwidth interconnection, applications try to maximize the advantage these advances bring. Clearly, the evolution of HPC technology and integration have increased the complexity of this challenge. While new features in parallel languages, programming tools, and runtime system environments can help to transform existing applications or to develop new ones in ways that leverage HPC's strengths, they can also introduce complexities of their own. At the end of the day, the goal of gaining high performance is paramount, but productivity and performance portability concerns are important as well.

The TAU Performance System consists of two toolkits: the *Tuning and Analysis Utilities (TAU)* and the *Autonomic Performance Environment for Exascale (APEX)*. TAU and APEX are based on different performance observability models and target different forms of parallel computation. They are complementary and can be used standalone or together. Each toolkit is described below. More attention is given to

TAU since it is used more in the Vlasiator experiments and has a longer heritage.

A. TAU Technology

The TAU model of performance observability is based on a "worker" ("first-person") perspective. Essentially, each *thread of execution* in a program will make performance measurements with respect to its operation exclusively. All performance data obtained will be stored within each thread's context during execution and produced at the end. From TAU's perspective, the execution of a program is regarded as a sequence of significant performance *events*. TAU can observe these events through *probes* inserted in the application code. The combination of a flexible event model and alternative instrumentation techniques developed in TAU results in a powerful ability to capture events and their semantics that might otherwise be intractable.

TAU supports several instrumentation mechanisms based on the code type (e.g., source, binary/dynamic, interpreter, and virtual machine) and is distinguished by its support for combining different instrumentation methods. Over time, TAU expanded its observation approach to include event-based sampling (EBS) methods, where the "event" here is an interrupt to the application's execution. TAU's EBS support extends its observational fidelity to finer-grained code regions. Both probes and statistical sampling (i.e., EBS) can be used simultaneously in TAU.

Once "events" are made visible (via probes or sampling) they can be measured. The TAU *event interface* allows events to be defined, their visibility controlled, and their runtime data structures to be created. Each event has a *type* (*atomic* or *interval*), a *group*, and a unique *event name*. The event name is a character string and is a powerful way to encode event information. At runtime, TAU maps the event name to an efficient *event ID* for use during measurement. Events are created dynamically in TAU by providing the event interface with a unique event name. This makes it possible for runtime context information to be used in forming an event name (*context-based events*), or values of routine parameters to be used to distinguish call variants, (*parameter-based events*). TAU also supports *phase* and *sample* events.

The measurement system is the heart and soul of TAU. It has evolved over time to a highly robust, scalable infrastructure that is portable to all HPC platforms. The instrumentation layer defines which events will be measured and the measurement system selects which performance data metrics to observe. Performance experiments are created by selecting the key events of interest and by configuring measurement modules together to capture desired performance data. TAU's measurement system provides support for portable timing, integration with hardware performance counters (e.g., PAPI [19] parallel profiling, parallel tracing (with OTF-2 [20]), and runtime monitoring). Given the rise of heterogeneous computing in HPC, TAU's measurement infrastructure has been extended to support performance observation of accelerator devices, primarily GPUs [21].

TAU's measurement system has two core capabilities. First, the *event management* handles the registration and encoding of events as they are created. Second, a runtime representation called the *event callstack* captures the nesting relationship of interval performance events on each thread. It is a powerful runtime measurement abstraction for managing the TAU performance state for use in both profiling and tracing. In particular, the event callstack is key for managing execution context, allowing TAU to associate this context with the events being measured.

The final component of TAU is analysis tools. TAU includes support for parallel profile data management (*TAUdb* [22]), analysis (*ParaProf* [23]), and data mining (*PerfExplorer* [24]). It leverages existing trace analysis functionality available in robust external tools, including Vampir [25], Jumpshot [26], and Expert/CUBE [27].

TAU's observation model, measurement technology, and analysis tools make it highly flexible and configurable, allowing it to be ported to different HPC and systems and applied in a variety of HPC applications. It takes advantage of state-of-the-art performance interfaces for accessing hardware data (e.g., PAPI) and capturing events (e.g., PMPI, OMPT, Level Zero, ROCprofiler, ROCtracer, OpenCL, CUPTI, and Kokkos [28]). Furthermore, TAU has been complemented by additional work by our team to develop technology for a plugin interface [29], support for MPI Tools interface [30], and generic performance interfaces that can be utilized by multiple tools (i.e., PerfStubs [31]).

Mature software engineering practices ensure that the TAU software remains portable, scalable, compatible, stable, and ultimately viable for the long term. For instance, the *TAU API* is a thin layer implemented as C++ macros that are inlined and easily integrated in instrumented code. Without special compilation flags, the API can be expanded to no-ops and so instrumented code can retain the API. While much of its implementation has evolved over the past two decades, its user-facing interface has remained consistent and stable. The core TAU API is accessible through wrappers for C, Fortran, and Python and TAU supports other languages such as Chapel, UPC++, and Java as well. This backward and forward compatibility of software is one of the features that sets TAU apart in preserving compatibility between versions and allows data generated at multiple sites with different versions of TAU to be analyzed by a user on some other system.

TAU's ability to insert probes in the code and perform measurement in un-modified binaries using the *tau_exec* tool has helped its broad adoption. This tool preloads the TAU dynamic shared object (DSO) in the address space of the application and allows the tool to intercept runtime system calls and enable other features of TAU (such as callstack unwinding, event-based sampling, and tracing). A similar tool, *tau_python*, supports instrumentation of Python applications by acting as a drop-in replacement for the Python interpreter. In both these cases, TAU's ability to generate performance data without modifying the application source code, build system, scripts, or the binary helps users easily generate performance

data by simply modifying the launch command.

TAU's components are integrated in a modular way where one module does not impact the other and well defined interfaces allow the user to configure and assemble a set of modules that match the runtime systems, compilers, and languages used by the application to allow TAU instrumentation to easily blend into the application and extract the performance data. The low-level systems skills and mature software engineering practices developed by the TAU team has also helped in merging its technology with application development.

B. APEX Technology

APEX [3] is based on a "task" ("third-person") performance observability perspective, with event-based (*synchronous*) and sample-based (*asynchronous*) measurements. It was originally designed to support the specific needs of HPX, an asynchronous manytask runtime system implemented in C++. HPX exposed difficult observability problems associated with emerging programming models of this type: untied task execution and migration, runtime thread control and execution, state sampling, and runtime performance tuning. These present problems for measurement tools that are designed to handle critical paths with respect to a calling context tree. While APEX can and does keep track of calling context trees, it also has the ability to track task dependencies across threads and devices.

APEX uses an event API and event listeners to observe when a task is created, started, yielded or stopped, and update timers for measurement of these actions. (Note, this is with respect to what constitutes a task, not necessarily its thread of execution.) Dependencies between tasks are also tracked, using globally unique identifiers (GUID). APEX periodically and on-demand interrogates (samples) OS, hardware, or runtime states (e.g., CPU utilization, resident set size, memory "high water mark"). APEX measurement includes background buffer processing to record GPU kernel execution and memory transfers to and from GPUs. Available runtime counters (e.g., idle rate, queue lengths) are also captured on-demand or on a periodic basis.

With its performance observability model, APEX provides much of the measurement and programming model coverage offered by TAU. APEX is integrated with PAPI for both CPU and GPU hardware counter access. Additionally, it takes advantage of Linux *perf* event counters, uncore counters, LM sensors, power and energy counters. APEX supports GPU memory tracking for both CUDA and HIP programs. It keeps track of all GPU memory allocations and records the number of bytes allocated, the address returned by the allocator, and the backtrace from when the allocation occurred.

APEX has native support for performance profiling, in which all tasks scheduled by the runtime are measured and a report is output to disk and or the screen at the end of execution. The profile data contains the number of times each task was executed and the total time spent executing that type of task. The profile data also contains all of the sampled counters encountered during execution. Each process maintains its own profile data, and writes a different output in various optional formats, including TAU profiles or CSV files. In addition, APEX provides a concurrency view with the concurrency listener, which will periodically sample the timer stack on all known OS threads. The samples are written to a file at the end of execution. APEX can also capture the task dependency relationships. The dependencies can be captured as a graph or a tree, and the data can be stored as a TAU callpath profile, task dependency trees in ASCII text, and other forms. APEX is integrated with the OTF2 library for tracing.

Both C++ threads `std::thread`, `std::async`, and POSIX C threads can be measured by APEX. Like TAU, it incorporates the profiling interfaces for OpenMP, OpenACC, CUDA, and HIP. In addition to timing events, APEX will capture how many bytes were transferred to and from events as well as the asynchronous device activity. APEX also implements support for Kokkos and RAJA.

III. Profiling interfaces in Vlasiator

A. The Phiprof interface

Phiprof [32] is a simple library that can be used to profile parallel programs using either MPI, OpenMP or both. It can be used to produce a hierarchical report of time (average, max, min) spent in different timer regions. It supports C, C++ and Fortran 2008. It also supports registering phiprof ranges as NVTX ranges. A key feature is its low overhead (less than $1\,\mu$s per call), allowing its use in loops. It prints the timings as a human-readable hierarchical report when user code requests it and automatically handles cases where groups of processes execute different codepaths.

Phiprof has been integrated into Vlasiator early during code development to enclose important sections of execution. These include, for example, initialization of the run, the main propagation loop, IO operations, as well as important algorithmic blocks such as spatial propagation and field solving steps. The instrumentation of the code was guided by estimations of regions that would be computationally expensive, as well as a striving to obtain profile reports that did not include significant fractions of timers not covered by named regions. Those get collected automatically under the last sub-timer "Other" in each timer level. However, as it relies on user-defined timer regions, it is not capable of drawing the user's attention to unexpected performance discrepancies. A selection of hierarchical profiling regions from Vlasiator is shown in Table I.

B. Support in TAU and APEX

We integrated the Phiprof interface in TAU and APEX to make it possible to take advantage of their infrastructure while using the Phiprof interface. Among these features, we can mention hierarchical performance profiling, and its visualization using TAU's tool *paraprof*. In the screenshot presented Figure 4, we can see that the Vlasiator programmers defined high-level timers that have a meaning in the lifetime of the execution: *initialization*, *simulation*, *finalization*, and *report*. In the simulation itself, they defined timers for operations related

Level	Label	Brief description
1	main	Program `main()` function
2	Initialization	Grid and solver setup
2	report-memory-consumption	Function reporting node memory usage
2	Simulation	Main loop
3	IO	Data and bookkeeping IO operations
3	Propagate	Actual plasma and electromagnetic field propagation
4	Spatial-space	Position space advection
4	Update system boundaries (Vlasov post-translation)	Post-advection update of boundary cells
4	Compute interp moments	Interpolation of density/velocity/pressure between advection and acceleration
4	Propagate Fields	Electric and magnetic field update
4	Velocity-space	Velocity space advection a.k.a. acceleration
4	Update system boundaries (Vlasov post-acceleration)	Post-acceleration update of boundary cells
4	ionosphere-solve	Ionospheric potential update
4	Other	Remaining, non-instrumented sections in level 3 region “Propagate”
3	compute-timestep	Determination of time step limits, update of time step if necessary
3	Balancing load	Rebalancing of the computational load across MPI domains
3	Other	Remaining, non-instrumented sections in level 2 region “Simulation”
1	Other	Remaining, non-instrumented sections in level 1 region “main”

TABLE I
SELECTION OF HIGH-LEVEL CODE REGIONS INSTRUMENTED WITH PHIPROF IN VLASIATOR, ILLUSTRATING THE MAJOR PHASES OF CODE EXECUTION. THE CURRENT VLASIATOR CODE INSTRUMENTS SOME REGIONS DOWN TO A LEVEL OF 9.

to *load balancing*, *IO*, and operations that correspond to parts of the physics simulation: *propagate*, *shrink to fit*, *compute timestamp*...

These timers need to be *nested* to be hierarchical. Moreover, they can use strings or integer labels, as presented by listing 1.

```
int label;
phiprof::initialize();
label = phiprof::initializeTimer( "Propagate" );
/* ... */
phiprof::start( "Simulation" );
/* ... */
phiprof::start( label );
/* ... */
phiprof::stop( label );
/* ... */
phiprof::stop( "Simulation" );
/* ... */
```

Listing 1. Phiprof timers example

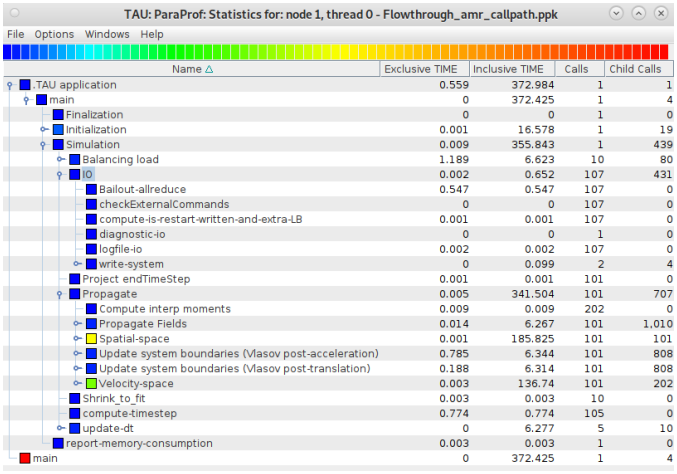

Fig. 4. Hierarchical performance profiling and visualization with TAU.

While performance profiling provides how much time is spent in given operations, *tracing* puts these events on a timeline. Such traces can be visualized using a tool such as Vampir [33] (commercial) or Perfetto (open source)[1]. Figure 5 shows a screenshot of the trace of an execution of Vlasiator, obtained using Apex and displayed using Perfetto.

IV. EXPERIMENTS

Integration of TAU with Vlasiator allowed new insights into the runtime behaviour of the simulation. During a three-day workshop hosted by CSC–IT Center for Science in Espoo, Finland, the Vlasiator and TAU teams worked together to obtain high-fidelity profile and trace data of Vlasiator runs on EuroHPC’s LUMI-C HPE Cray system at different scales.

Since Vlasiator’s scientific focus is the simulation of near-Earth plasma flows, the chosen run test cases were global setups encompassing Earth’s magnetosphere. The basic setup corresponded to the global runs described in [11], with the run resolution adapted to fit on different node counts (concretely, runs on 1, 16, 80 and 250 nodes were conducted).

Compared to the plain-text phiprof output summary, that had been Vlasiator’s main source of performance information before, it became immediately apparent that TAU’s profiling output as presented by the `ParaProf` tool provided a much more comprehensive and versatile view. In particular, TAU’s ability to show sample-based performance events alongside the counters defined by phiprof groups provides a significant gain in fidelity at performance hotspots of the code. Before, a question like “what in this region is it that *actually* takes most of the time” had to be answered by informed guesswork and/or more detailed instrumentation with more profiling calls; now it is readily presented in the hierarchical view of ParaProf (compare Figure 6).

Another useful addition was the concurrent gathering of both user-defined range measurements and probes associated with the MPI subsystem. Efficient exascale performance of HPC codes is extremely reliant on efficient exploitation of high-bandwidth communication between compute nodes, so it

[1]https://github.com/google/perfetto

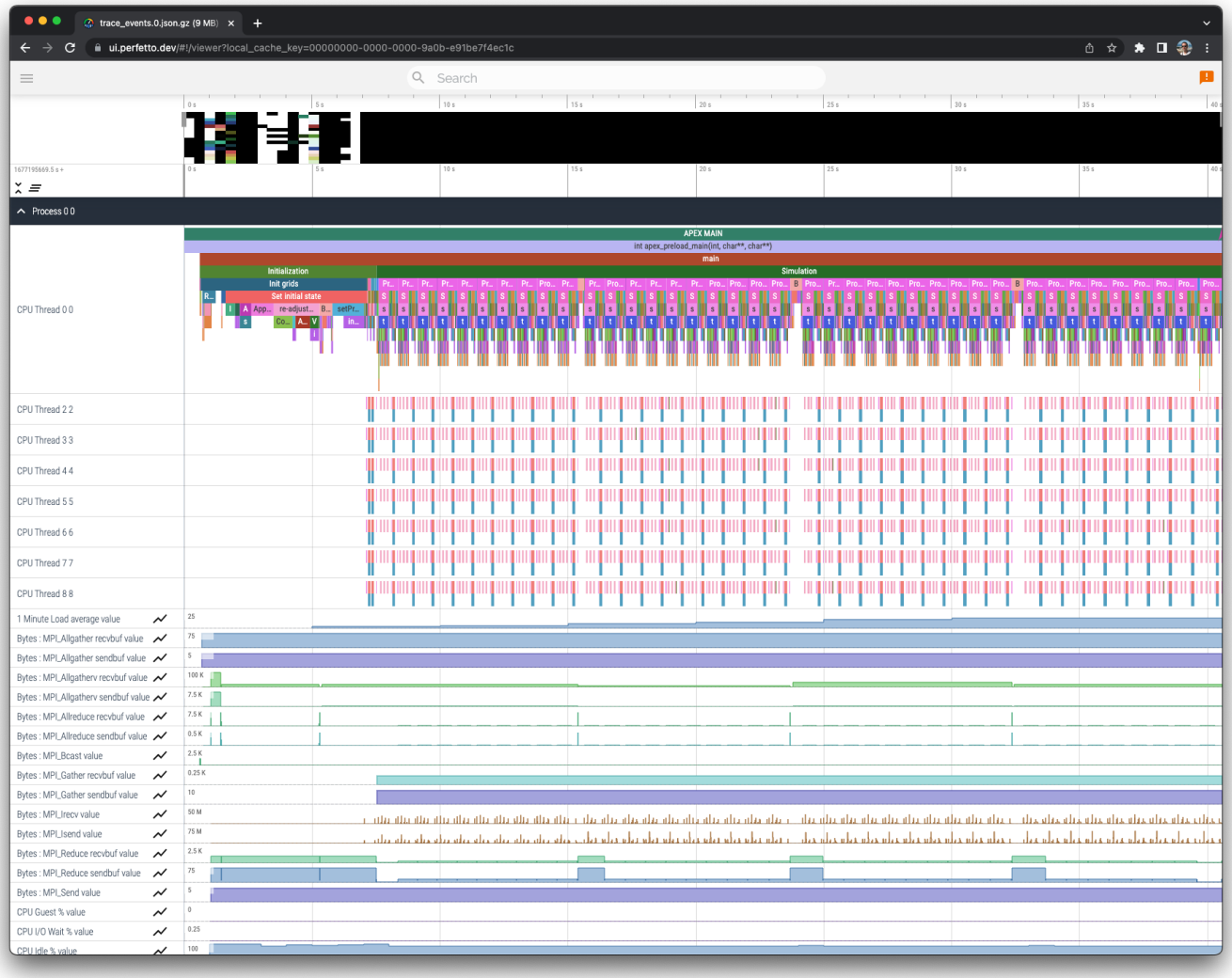

Fig. 5. Visualization of a trace using Perfetto.

Name	Exclusive TIME	Inclusive TIME ▽	Calls	Child Calls
TAU application	0.001	1,844.43	1	1
taupreload_main	0.001	1,844.429	1	17
main	0.002	1,842.288	1	4
Simulation	0.007	1,662.342	1	296
Propagate	0.003	1,558.669	71	513
Propagate Fields	0.054	668.607	71	16,330
Spatial-space	0.001	626.189	71	71
Velocity-space	0.003	164.332	71	213
Update system boundaries (Vlasov post-acceleration)	0.096	52.675	71	15,700
Update system boundaries (Vlasov post-translation)	0.099	34.415	71	15,753
Ionosphere-solve	10.31	10.326	4	4
fieldtracing-ionosphere-fsgridCoupling	0.023	2.063	4	64
Ionosphere-mapDownMagnetosphere	0.007	0.035	4	12
Compute interp moments	0.013	0.013	142	0
Ionosphere-calculateConductivityTensor	0.011	0.011	4	0
IO	0.008	80.645	72	364
Balancing load	0.032	16.713	4	924
compute-timestep	3.071	5.146	70	70
Ionosphere-updateIonosphereCommunicator	0	1.162	4	20
Shrink_to_fit	0.001	0.001	4	0
Project endTimeStep	0	0	71	0
Initialization	0.001	179.906	1	10
report-memory-consumption	0.023	0.038	1	14
Finalization	0	0	1	0
MPI_Init_thread()	2.112	2.112	1	1
MPI_Finalize()	0.028	0.028	1	0
MPI_Comm_free()	0	0	13	0
MPI_Comm_rank()	0	0	1	0

Fig. 6. TAU hierarchical profile view output of a magnetospheric simulation on 250 nodes of LUMI. The three main solvers of Vlasiator (`Spatial-space`, `Velocity-space` and `Propagate fields`) are taking up the majority of the simulation loop's time, but their relative contributions at this scale came as a surprise.

becomes imperative to pinpoint any sources of node-seconds wasted in waiting for MPI communication to complete.

A. *Insights gained*

During a focused investigation of Vlasiator trace data thus obtained, it became quickly apparent that some parts of the code (specifically, the mapping stage of the acceleration solver) were slower than expected, and spending lots of time waiting for completion of individual threads' OpenMP workloads. As it turns out, the particular OpenMP parallel loop at this point in the code was using the `guided` schedule [34]. In this schedule, half of the loop's work is distributed over the available threads, and the remaining work is given to threads as they finish their work batches. Vlasiator's sparse velocity space structure [5] adapts the velocity mesh resolution to the local physical requirements in every point in real space. In the velocity space grid, physical cells size can vary up to three orders of magnitude in typical production

Fig. 7. Perfetto renderings of TAU traces showing the OpenMP thread imbalance that had been identified in Vlasiator's translation solver. **Top panel:** The original code scheduled work groups to all OpenMP threads according to the `guided` strategy with no batch limit, occasionally causing single threads to recieve a disproportionate amount of work. **Bottom panel:** By limiting the batch size to 8 elements, the imbalance was alleviated.

runs of magnetospheric conditions. Due to this inhomogeneity, computational cost of the individual simulation cells, and thus the time requirement of OpenMP work units, likewise differs by multiple orders of magnitude. In some cases, single threads were assigned an initial workload that was far exceeding that of others, such that even after redistribution of the second half of work, they were still ongoing. The straightforward patch for this behavior was to limit the maximum batch size that the OpenMP loop was to distribute (see Fig. 7), but without the TAU tracing insight, this issue would not have been possible to identify (and indeed, had before probably been present and undiagnosed for years).

A second performance bottleneck that was noticed when investigating outputs from event-based sampling was the unusually large amount of time spent in searching for spatial neighbors of simulation cells (through the adaptive grid's `get_face_neighbors_of` function [10]). The function was implemented in such a way that it iterated over nearby cells in a three-cell-wide stencil multiple times, leading to a worst case of thousands of iterations. As the list of neighbors is static as long as the grid doesn't change it was decided to cache the neighbors, updating the list only when necessary. This promptly provided a performance benefit of up to five percent in test runs (Figure 8). Pinpointing this performance bottleneck would have been extremely unlikely were it not for the event-based sampling provided by TAU, which enabled honing in on time spent within too loosely or not-instrumented ranges.

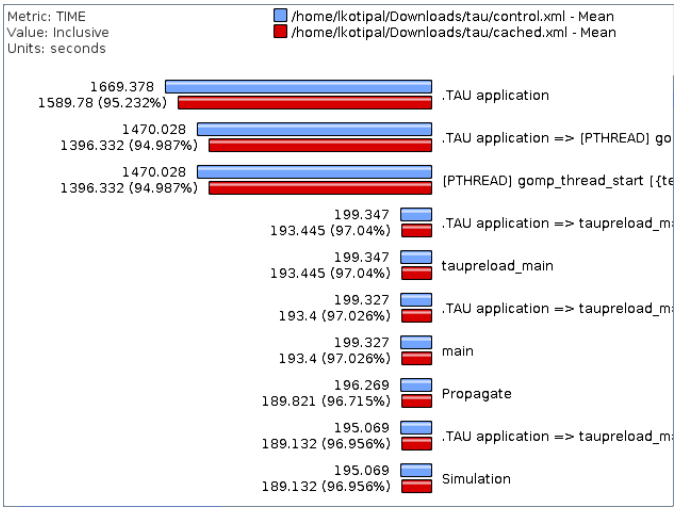

Fig. 8. Tau comparison view of mean time taken per thread without (blue) and with (red) cached neighbors.

B. Future improvements

A major insight gained from instrumenting Vlasiator with TAU was how much time was spent in `MPI_Waitall` functions due to computational imbalances in the code, despite the fact that dynamic load balancing is already employed extensively in the simulation. A number of potential improvements to reduce communication overhead have been identified, where in some cases MPI communication can be avoided outright by performing some extra redundant computation for ghost domain cells (See Fig. 9 for an example). Implementation and re-profiling of these is on the Vlasiator development roadmap. However, most of these identified improvements require significant algorithmic refactoring. As illustrated by Figure 3, the trend of Moore's law supports the performance improvement of the code, and continual performance evaluation and profiling allow for gradual performance improvements that have been omitted from the figure for simplicity, yet the decisive progress is achieved by targeted major algorithmic changes. TAU and APEX promise great help in focusing the attention to the relevant parts of the code when drawing the roadmap of the next generations of performance improvements.

The positive experience of using TAU to profile Vlasiator in a pre-Exascale Cray HPC environment has made it crystal clear how thorough comparison of thread-parallel performance, hardware counters, and time spent in MPI communication is crucial to properly utilizing next-generation computational resources. TAU offers unique perspectives for exascale machines as it combines all of the aforementioned with the awareness of heterogeneous memory systems and the capability to profile GPU kernels.

V. Related Work

There is a rich history of performance technology research and development that has been driven by rapid advances in HPC hardware, software, and systems architecture. Indeed, the

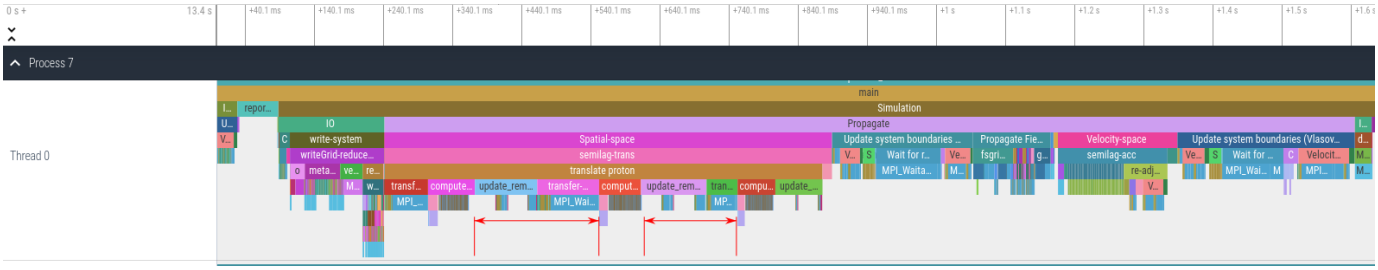

Fig. 9. Zoom-in of a TAU trace of a Vlasiator magnetospheric simulation, visualized using Perfetto (compare Fig. 5). In this view, a single simulation timestep on one MPI task is shown on the width of the figure, allowing the individual contributions of physics solvers to be seen. The two regions highlighted by red arrow ranges have been identified as MPI neighbour communication that could potentially be removed, thus offering a path to significant speedup of the spatial transport solver.

performance tools community is replete with robust performance measurement and analysis tools, several of which have been developed for HPC systems and applications for many years. A significant number come from academic research, national laboratories, and HPC research centers, including HPCToolkit [35], Score-P [36], Scalasca [27], Extrae/Paraver [37], Caliper [38], and more. The HPC industry has also contributed productive tools such as Intel's VTune [39], HPE's CrayPat [40], and NVIDIA's Nsight Systems [41]. All of these tools, like TAU and APEX, face the challenges of keeping up with the aggressive innovations in the HPC field that make the problem of performance observability difficult.

While each tool has its own set of relevant capabilities that make it worthwhile when well-matched to the performance analysis circumstances, no tool is created entirely from scratch. There are important technologies that are key building blocks for creating comprehensive performance systems. The PAPI [42] library for accessing CPU and GPU hardware counters is ubiquitous in performance tools and provides a production-level solution to a complex problem. In general, interfaces to retrieve performance data, either in hardware components, network interfaces, accelerator devices, or the operating system, become invaluable to the performance tool developer. A relevant example for heterogeneous computing are the performance tool interfaces for GPUs, such as provided by NVIDIA CUPTI [43] and AMD GPUPerfAPI [44].

The importance of interfaces for performance observability extends to the general problem of how to make parallel software events visible to the performance measurement system. The profiling interface for MPI (PMPI) [45] is the classic example of a portable technology that has enabled practically every performance tool to observe MPI operation. It uses a library interposition approach that enables "instrumented" versions of MPI routines to be called. Similarly, the OpenMP performance tool interface (OMPT) [34] provides a portable solution to capture thread operations, examine OpenMP internal state, map calling contexts to source, and more. It uses a callback interface that enables a tool to receive notification of OpenMP events. Equivalent functionality is provided in the OpenACC profiling interface (ACCPI) [46]. Both OMPT and ACCPI are part of the standards and able to be supported in OpenMP and OpenACC implementations, as was done in LLVM [47].

Two key features provided by performance interfaces generally are the *semantic context* associated with events (i.e., what do the event mean) and access to internal *performance state*. The interest in parallel programmming abstractions for heterogeneity and performance portability will necessarily result in further separation of low-level performance measurements from their association with high-level context and its execution semantics. To address this concern, performance interfaces are being designed into high-level programming systems such as Kokkos [48], RAJA [49], and StarPU [50]. This makes it possible for performance tools to organize performance information with respect to computational context known only to the high-level software.

The approach is applied as well in motif-based libraries and frameworks. The PETSc [51] profiling interface is a good example. In fact, such work is motivated by the need for better performance analysis of applications using libraries and frameworks. Here, it is certainly reasonable to use interposition techniques and callback methods found in other cases to make sure performance interfaces accessible by tools. For instance, the PerfStubs library [31] is a thin, stubbed-out, "adapter" interface for instrumenting library or application code. The PerfStubs library itself does not do any measurement, it merely provides access to an API that performance tools can implement. The instrumentation function calls are "stubs" in the form of function pointers, initialized to `nullptr`. Our approach with re-implementing the original Phiprof work [32] is along these lines.

VI. Conclusion and Future Work

A common, yet frustrating conundrum facing HPC application developers concerns the need for performance measurement and analysis to characterize execution inefficiencies and inform tuning decisions for better outcomes, versus the choice of which performance tool(s) and their associated technologies to use for that purpose. In many cases, a default solution is adopted, which all too often amounts to just measuring total execution time. In other cases, homegrown measurement support might be developed specific to the application, for instance, to measure time in specific code regions, but will lack significant support for observing anything else. The challenge

to maintain, support, and port such one-off solutions will ultimately isolate them from more modern techniques and make them unsustainable. Our simple objective with this paper is to show that it does not have to be this way.

There are compelling reasons for application teams to try to understand what state-of-the-art HPC performance tools have to offer and to take the time to integrate those technologies in their code development and performance engineering processes. Vlasiator is a perfect case study to demonstrate the potential benefits of such efforts. It is an advance next-generation simulation software for modeling space plasma physics in near-Earth space. The MPI+OpenMP code has been developed over several generations and involves complex data structures and algorithms for 6-dimensional hybrid-Vlasov calculations. We successfully integrated the TAU Performance System with the Vlasiator code and demonstrated the advanced capabilities offered beyond its earlier Phiprof support. Indeed, the enlightened design of the original Phiprof interface made it possible to realize a compatible backend implementation for connecting to the TAU and APEX measurement libraries. The effect was to significantly expand Vlasiator performance observability and scope of analysis. We present several examples attesting to this fact.

Looking to the future, Vlasiator now has access to a robust, portable, and configurable environment for performance analysis and engineering. The TAU and APEX tools work on heterogeneous HPC platforms, across different processor architectures, and at scale. Importantly, it helps to address the challenges that come with measurement of leading-edge CPU and GPU technologies present in supercomputer systems like LUMI. This will be particularly important as Vlasiator transitions to GPU operation and AMD GPU technology.

Beyond the current outcomes, our continuing integration strategy will include tighter support in Vlasiator's build system and development of more substantial performance engineering features built around the TAU Performance System ecosystem, including parametric performance studies, cross-architecture performance characterization, performance regression testing, and continuous integration. The TAUdb performance database and other performance analysis support are relevant to this objective. They provide a performance experimentation platform that can be easily used for testing with new code versions.

The Vlasiator team has an aggressive plan that includes algorithm updates (e.g., dynamic adaptive mesh refinement), GPU development, and porting to new architectures (e.g., AMD CPU and GPU in LUMI), as well as extensions and improvements to the science enabled by the model and the solvers. The robust features of the integrated performance tools should cover every aspect of investigation. We will work closely on defining experiments that coincide with Vlasiator code developments, testing, and performance studies. The measurement setup will be captured for each experiment as well as the performance data generated. We will use the TAU performance database (TAUdb) to store the results from performance studies.

VII. Acknowledgments

The authors gratefully also acknowledge the Academy of Finland (grant numbers 1347795 "HISSA", 1335554 "ICT-SUNVAC" and 339756 "KIMCHI").

This work has received funding from the European High Performance Computing Joint Undertaking (JU) under grant agreement No 101083261 "Plasma-PEPSC". The Finnish Centre of Excellence in Research of Sustainable Space, funded through the Academy of Finland grants 312351 and 1336805, supports Vlasiator development and science as well.

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative, as well as the Scientific Discovery through Advanced Computing (SciDAC) program funded by DOE, Office of Science, Advanced Scientific Computing Research (ASCR) under contract DE-SC0021299.

The simulations for this publication were run on the EuroHPC "LUMI" supercomputer in Kajaani, Finland and the "Mahti" supercomputer at CSC Centre for Scientific Computing. The authors wish to thank the Finnish Grid and Cloud Infrastructure (FGCI) for supporting this project with additional computational and data storage resources.

Code availability

The Vlasiator simulation code is distributed under the GPL-2 open source license at https://github.com/fmihpc/vlasiator [6]. TAU and APEX are open source under a BSD-style license and available at https://tau.uoregon.edu and https://uo-oaciss.github.io/apex.

References

[1] E. Strohmaier, J. Dongarra, H. Simon, and M. Meuer, "November 2022 top500 list of the world's fastest supercomputers." [Online]. Available: https://www.top500.org/lists/top500/2022/11/

[2] S. Shende and A. Malony, "The TAU Parallel Performance System," *International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 287–331, 2006.

[3] K. Huck, "Broad performance measurement support for asynchronous multi-tasking with apex," in *IEEE/ACM 7th International Workshop on Extreme Scale Programming Models and Middleware (ESPM2)*, 2022.

[4] M. Palmroth, U. Ganse, Y. Pfau-Kempf, M. Battarbee, L. Turc, T. Brito, M. Grandin, S. Hoilijoki, A. Sandroos, and S. von Alfthan, "Vlasov methods in space physics and astrophysics," *Living Reviews in Computational Astrophysics*, vol. 4, 2018. [Online]. Available: https://doi.org/10.1007/s41115-018-0003-2

[5] S. von Alfthan, D. Pokhotelov, Y. Kempf, S. Hoilijoki, I. Honkonen, A. Sandroos, and M. Palmroth, "Vlasiator: First global hybrid-Vlasov simulations of Earth's foreshock and magnetosheath," *Journal of Atmospheric and Solar-Terrestrial Physics*, vol. 120, pp. 24–35, 2014.

[6] Y. Pfau-Kempf, S. von Alfthan, U. Ganse, A. Sandroos, M. Battarbee, T. Koskela, O. Hannuksela, I. Honkonen, K. Papadakis, L. Kotipalo, H. Zhou, M. Grandin, D. Pokhotelov, and M. Alho, "fmihpc/vlasiator: Vlasiator 5.2.1," Jun. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.6782211

[7] G. Strang, "On the construction and comparison of difference schemes," *SIAM Journal on Numerical Analysis*, vol. 5, no. 3, pp. 506—517, 1968.

[8] F. Valentini, P. Trávníček, F. Califano, P. Hellinger, and A. Mangeney, "A hybrid-vlasov model based on the current advance method for the simulation of collisionless magnetized plasma," *Journal of Computational Physics*, vol. 225, no. 1, pp. 753–770, 2007.

[9] M. Zerroukat and T. Allen, "A three-dimensional monotone and conservative semi-lagrangian scheme (slice-3d) for transport problems," *Quarterly Journal of the Royal Meteorological Society*, vol. 138, no. 667, pp. 1640–1651, 2012.

[10] I. Honkonen and Vlasiator Team, "Dccrg – a distributed cartesian cell-refinable grid." [Online]. Available: https://github.com/fmihpc/dccrg

[11] U. Ganse, T. Koskela, M. Battarbee, Y. Pfau-Kempf, K. Papadakis, M. Alho, M. Bussov, G. Cozzani, M. Dubart, H. George, E. Gordeev, M. Grandin, K. Horaites, J. Suni, V. Tarvus, F. Kebede, L. Turc, H. Zhou, and M. Palmroth, "Enabling technology for global 3d + 3v hybrid-vlasov simulations of near-earth space," *Physics of Plasmas*, vol. 30, 2023.

[12] L. Kotipalo, "Adaptive mesh refinement in vlasiator," Master's thesis, Helsingin yliopisto, 2023. [Online]. Available: URN:NBN:fi:hulib-202303031442;http://hdl.handle.net/10138/355463

[13] P. Londrillo and L. Del Zanna, "On the divergence-free condition in Godunov-type schemes for ideal magnetohydrodynamics: the upwind constrained transport method," *Journal of Computational Physics*, vol. 195, no. 1, pp. 17–48, 2004.

[14] K. Papadakis, Y. Pfau-Kempf, U. Ganse, M. Battarbee, M. Alho, M. Grandin, M. Dubart, L. Turc, H. Zhou, K. Horaites, I. Zaitsev, G. Cozzani, M. Bussov, E. Gordeev, F. Tesema, H. George, J. Suni, V. Tarvus, and M. Palmroth, "Spatial filtering in a 6D hybrid-Vlasov scheme to alleviate adaptive mesh refinement artifacts: a case study with Vlasiator (versions 5.0, 5.1, and 5.2.1)," *Geoscientific Model Development*, vol. 15, no. 20, pp. 7903–7912, 2022. [Online]. Available: https://gmd.copernicus.org/articles/15/7903/2022/

[15] Vlasiator Team, "fsgrid – a lightweight, static, cartesian grid for field solvers." [Online]. Available: https://github.com/fmihpc/fsgrid

[16] Y. Pfau-Kempf, "Vlasiator – from local to global magnetospheric hybrid-vlasov simulations," Ph.D. dissertation, University of Helsinki, 2016.

[17] E. G. Boman, U. V. Catalyurek, C. Chevalier, and K. D. Devine, "The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering, and coloring," *Scientific Programming*, vol. 20, no. 2, pp. 129–150, 2012.

[18] K. Nishikawa, I. Duţan, C. Köhn, and Y. Mizuno, "PIC methods in astrophysics: simulations of relativistic jets and kinetic physics in astrophysical systems," *Living Reviews in Computational Astrophysics*, vol. 7, no. 1, Jul. 2021. [Online]. Available: https://doi.org/10.1007\%2fs41115-021-00012-0

[19] D. Terpstra, H. Jagode, H. You, and J. Dongarra, "Collecting performance data with papi-c," in *Tools for High Performance Computing 2009*. Springer, 2010, pp. 157–173.

[20] D. Eschweiler, M. Wagner, M. Geimer, A. Knüpfer, W. Nagel, and F. Wolf, "Open trace format 2: The next generation of scalable trace formats and support libraries," in *International Conference on Parallel Computing*, 2011, pp. 481–490.

[21] A. Malony, S. Biersdorff, S. Shende, H. Jagode, S. Tomov, G. Juckeland, R. Dietrich, D. Poole, and C. Lamb, "Parallel performance measurement of heterogeneous parallel systems with GPUs," in *Parallel Processing (ICPP), 2011 International Conference on*, 9 2011, pp. 176–185.

[22] K. Huck, A. Malony, R. Bell, and A. Morris, "Design and Implementation of a Parallel Performance Data Management Framework," in *International Conference on Parallel Processing (ICPP 2005)*. IEEE Computer Society, Aug. 2005, (Chuan-lin Wu Best paper award). (Acceptance rate 28.6% (69/241).

[23] R. Bell, A. Malony, and S. Shende, "ParaProf: a portable, extensible, and scalable tool for parallel performance profile analysis," in *Proc. EUROPAR 2003 Conference*, 2003.

[24] K. A. Huck, A. D. Malony, S. Shende, and A. Morris, "Knowledge support and automation for performance analysis with perfexplorer 2.0," *Scientific Programming*, vol. 16, no. 2–3, pp. 123–134, 2008.

[25] M. S. Müller, A. Knüpfer, M. Jurenz, M. Lieber, H. Brunst, H. Mix, and W. E. Nagel, "Developing scalable applications with vampir, vampirserver and vampirtrace," *Advances in Parallel Computing*, vol. 15, pp. 637–644, 2008.

[26] O. Zaki, E. Lusk, W. Gropp, and D. Swider, "Toward scalable performance visualization with jumpshot," *The International Journal of High Performance Computing Applications*, vol. 13, no. 3, pp. 277–288, 1999.

[27] F. Wolf, B. Wylie, E. Ábrahám, D. Becker, W. Frings, K. Fürlinger, M. Geimer, M. Hermanns, B. Mohr, S. Moore, M. Pfeifer, and Z. Szebenyi, "Usage of the SCALASCA toolset for scalable performance analysis of large-scale parallel applications," in *Proc. of the 2nd HLRS Parallel Tools Workshop*. Stuttgart, Germany: Springer, July 2008, pp. 157–167.

[28] S. Shende, N. Chaimov, A. Malony, and N. Imam, "Multi-level performance instrumentation for kokkos applications using tau," in *2019 IEEE/ACM International Workshop on Programming and Performance Visualization Tools (ProTools)*, 2019, pp. 48–54.

[29] A. Malony, S. Ramesh, K. Huck, N. Chaimov, and S. Shende, "A plugin architecture for the tau performance system," in *48th International Conference on Parallel Processing*, Aug. 2019, pp. 1–11.

[30] S. Ramesh, A. Mahéo, S. Shende, A. D. Malony, H. Subramoni, A. Ruhela, and D. K. D. Panda, "MPI Performance Engineering with the MPI Tool Interface: The integration of MVAPICH and TAU," *Parallel Computing*, vol. 77, pp. 19–37, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167819118301479

[31] D. Boehme, K. Huck, J. Madsen, and J. Weidendorfer, "The case for a common instrumentation interface for hpc codes," in *2019 IEEE/ACM International Workshop on Programming and Performance Visualization Tools (ProTools)*, 2019, pp. 33–39.

[32] S. von Alfthan, "Phiprof – parallel hierarchical profiler," https://github.com/fmihpc/phiprof, 2019.

[33] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. S. Müller, and W. E. Nagel, "The Vampir performance analysis toolset," in *Tools for High Performance Computing*. Springer, 2008, pp. 139–155.

[34] OpenMP Architecture Review Board, *OpenMP Specification*, Nov. 2023. [Online]. Available: https://www.openmp.org/specifications

[35] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent, "HPCToolkit: Tools for Performance Analysis of Optimized Parallel Programs," *Concurrency and Computation: Practice and Experience*, 2010, to appear.

[36] A. Knüpfer, C. Rössel, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, W. E. Nagel *et al.*, "Score-p: A joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir," in *Tools for High Performance Computing 2011*. Springer, 2012, pp. 79–91.

[37] V. Pillet, J. Labarta, T. Cortes, and S. Girona, "Paraver: A tool to visualize and analyze parallel code," in *Proceedings of WoTUG-18: Transputer and occam Developments*, vol. 44. mar, 1995, pp. 17–31.

[38] D. Boehme, T. Gamblin, D. Beckingsale, P.-T. Bremer, A. Gimenez, M. LeGendre, O. Pearce, and M. Schulz, "Caliper: performance introspection for HPC software stacks," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 550–560.

[39] Intel, "Vtune profile," 2023, https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html.

[40] H.-P. Enterprise, "Cray performance measurement and analysis tools user guide," 2023. [Online]. Available: https://support.hpe.com/hpesc/public/docDisplay?docLocale=en_US&docId=a00113914en_us&page=About_the_Cray_Performance_Measurement_and_Analysis_Tools_User_Guide.html

[41] NVIDIA, "Nsight systems," 2023. [Online]. Available: https://developer.nvidia.com/nsight-systems

[42] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, "A Portable Programming Interface for Performance Evaluation on Modern Processors," *International Journal of High Performance Computing Applications*, vol. 14, no. 3, pp. 189–204, 2000.

[43] "CUDA Profiling Tools Interface," August 2020. [Online]. Available: https://docs.nvidia.com/cupti/Cupti/index.html

[44] "ROCm System Management Interface," August 2022. [Online]. Available: https://github.com/RadeonOpenCompute/rocm_smi_lib

[45] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, Jun. 2021. [Online]. Available: https://www.mpi-forum.org/docs/

[46] OpenACC Community, *OpenACC Application Programming Interface Version 3.3*, Nov. 2022. [Online]. Available: https://www.openacc.org/specification

[47] C. Coti, J. Denny, K. Huck, S. Lee, A. Malony, S. Shende, and J. Vetter, "OpenACC Profiling Support for Clang and LLVM using Clacc and TAU," in *Workshop on Programming and Performance Visualization Tools (ProTools)*, 2020, pp. 38–48.

[48] S. D. Hammond, C. R. Trott, D. Ibanez, and D. Sunderland, "Profiling and debugging support for the kokkos programming model," in *ISC Workshops*, 2018.

[49] D. A. Beckingsale, J. Burmark, R. Hornung, H. Jones, W. Killian, A. J. Kunen, O. Pearce, P. Robinson, B. S. Ryujin, and T. R. Scogland, "RAJA: Portable performance for large-scale scientific applications," in *2019 ieee/acm international workshop on performance, portability and productivity in hpc (p3hpc)*. IEEE, 2019, pp. 71–81.

[50] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures," *CCPE - Concurrency and Computation: Practice and Experience, Special Issue: Euro-Par 2009*, vol. 23, pp. 187–198, Feb. 2011. [Online]. Available: http://hal.inria.fr/inria-00550877

[51] PETSc, *Portable, Extensible Toolkit for Scientific Computation*, 2022. [Online]. Available: https://petsc.org/release