The OLCF GPU Hackathon Series: The Story Behind Advancing Scientific Applications with a Sustained Impact

Sunita Chandrasekaran*, Guido Juckeland†, Meifeng Lin‡, Matthew Otten§,
Dirk Pleiter¶, John E. Stone ‖, Juan Lucio-Vega *, Michael Zingale**, and Fernanda Foertter††
*University of Delaware, USA; {schandra,jluciove}@udel.edu
†Helmholtz-Zentrum Dresden-Rossendorf, Germany; g.juckeland@hzdr.de
‡Brookhaven National Laboratory, USA; mlin@bnl.gov
§Cornell University, USA; mjo98@cornell.edu
¶Forschungszentrum Jülich, Jülich Supercomputing Centre, Germany; d.pleiter@fz-juelich.de
‖University of Illinois at Urbana-Champaign, USA; johns@ks.uiuc.edu
**Stony Brook University, USA; michael.zingale@stonybrook.edu
††Oak Ridge National Laboratory, USA; foertterfs@ornl.gov

***Abstract*—The OLCF GPU Hackathons are a one-week code-development/learning event to better enable attendees to utilize GPUs. It only took three years to grow from a "Let's give this a try"-event to a repeatedly copied format with several spin-offs that inspired HPC centers around the world. Sticking to a few fundamental principles—work on your own code, learn from your mentors just what you need and when you need it, stay flexible in achieving your goal—the week long hackathon format created at Oak Ridge Leadership Computing Facility (OLCF) has been just the spark needed by many groups of scientists to light the fire of a wider GPU adoption in leading-edge as well as university-scale HPC environments. Most interestingly, the format enabled both ends of the experience spectrum—graduate students vs. postdoc fellows—the same kind of progress and chance of success.**

***Keywords*-Parallel Programming; Hackathon; GPU applications; Agile development; Mentoring; Profilers**

I. Introduction

How can experts prepare future computational scientists for the upcoming challenges in heterogeneous parallel computing when every University and college curriculum still teaches sequential programming concepts? How can well established domain scientists refactor their applications to keep up with the current hard- and software trends? The radical changes in the compute node architectures in the last decade, which have been introduced by adding hardware accelerators, have brought established programmers back to square one in using these novel devices. In that regard it is also an opportunity where students with their classroom experiences, scientists and programmers with their domain knowledge and GPU programming environment developers can come together and learn from one another.

In 2014, the Oak Ridge Leadership Compute Facility (OLCF) was faced with the question of increasing the number of projects successfully utilizing the GPUs in the fast supercomputer available for research. It became obvious that there was a missing component between the six applications that were chosen by the Center for Accelerated Application Readiness (CAAR) and the general trainings both from OCLF and the vendors. Fernanda Foertter suggested a combination of agile programming and intense mentoring over the course of one week to close this middle ground – and the first such GPU Hackathon took place in the fall of the same year. It featured seven teams that all met in one location for one week to jointly work on their individual applications. The success of the teams as well as the extremely positive and encouraging feedback from the participants, proved that the proposed education format was exactly what was needed to introduce new HPC concepts to experienced programmers. Since then the format has grown both in itself—2017 featured a total of five OCLF supported GPU Hackathons—as well as in similar formats. Other sites, e.g. the Barcelona Supercomputing Center, run similarly structured hackathons and a shorter format—the Mini-Hackathon—has been created as well.

This paper explains the training format adopted for the OLCF GPU Hackathon, presents trends observed, and discusses reasons for successes and failures of teams using selective case studies from over 15 hackathons that have taken place so far. It also summarizes the outcome and takeaway for the participants and looks at the Hackathon format from an educator's perspective. Anyone could take this hackathon format and adopt it in their respective institutions.

II. The GPU Hackathon Format

The OLCF hackathon format is centered around very few, but firm principles:

- You must sign up as a team of at least three members working on the same code. This is to ensure that there is a broader developer base behind a project.
- You as a team must specify the goal you want to reach with the hackathon. That way your team will be working towards that goal over the course of the hackathon.

- Your goal must include GPU usage of some kind – after all, this is what the GPU Hackathons are all about.

Since 2014 the GPU Hackathon Series has expanded to five OLCF endorsed hackathons a year which are spread throughout the year and around the globe. The GPU Hackathons are formally treated as OLCF training events[1] and are part of OLCF's outreach to create exciting proposals that use "Titan"—the veteran GPU computing HPC system at ORNL[2].

Another fixture of the GPU Hackathons is the large pool of mentors that support the teams. Each team will be assigned (at least) two mentors such that they will typically have a domain expert and a GPU programming expert to guide them towards reaching their set goal.

A. Proposal Submission

Interested teams can submit their proposed work through an online submission form where they are asked to describe their targeted application, state their goal for the hackathon, and briefly introduce the team members and their levels of expertise. Every Hackathon site has one or two local organizers that lead the review process for the submissions. The reviewers are the pool of local organizers from all Hackathons since they have the experience of how previous, potentially similar submissions worked out. Furthermore, the pool of potential mentors also reviews the proposals to provide feedback on the technical possibilities and science domain relevance. Due to the increased popularity of the GPU Hackathons the acceptance rate has dropped to about 50% in 2017, albeit all sites increasing the number of supported teams to about 10 per site. Rejected proposals typically have one of the following issues:

- The code is not open source, so that the community impact is questionable. Especially if teams have no license for the code, they are encouraged to choose an open source license. If the license issue cannot be resolved quickly, the team is encouraged to reapply at a later Hackathon.
- There are other software projects available that are already GPU enabled and that accomplish the same task.
- There are teams whose chance of success in achieving their set goal or the impact of the software advancement is judged to be higher. The former is typically the case if some or all team members already have some parallel programming expertise. The latter emphasizes that the Hackathon also works with teams from the ground up, if the impact of a GPU enabled code is very promising for the targeted science domain.

[1]https://www.olcf.ornl.gov/training-event/2017-gpu-hackathons
[2]https://www.olcf.ornl.gov/computing-resources/titan-cray-xk7

B. Preparation

The selected teams are introduced to their mentors about one month prior to the actual Hackathon week. The team and the mentors then discuss a preparation strategy. This includes:

- Receiving access to the compute system(s) to be used during the hackathon week so that the teams can ensure their application runs as it is.
- Online training courses on GPU programming basics (e.g. OpenACC introduction, CUDA basics), so that teams have some fundamental understanding of the used techniques during the Hackathon week.
- In case of a GPU port of a non-GPU accelerated application, the team are also asked to perform an initial profiling of the program so that the hot spots of the code are known.
- The organizers typically also set up a git or svn repository so that the teams and their mentors both have access to the source code and a revision system for changes during the week of the Hackathon.

C. The Hackathon Week

The teams meet up with their mentors in person for one week at the hosting site. They share one table in one room which holds all teams, so that team members but also teams can easily communicate with one another. From a software development point of view the week could be best described as a series of three one-day sprints with a planning day in front and a wrap-up day at the end. Fernanda Foertter, as the Hackathon chair, also employs other agile programming techniques: Daily SCRUM sessions where all teams are forced to summarize their current status and strategy.

While the chair and the mentors can provide a rough guideline as to how fast teams should make progress over the week, the main strength of the Hackathon format is the flexibility for every team to move at their own pace. The mentors can teach required techniques exactly when the team members need them, and the teams apply the learned items directly to their code. Hence, they also have the gratification of improving something they care about instead of yet another Jacobi solver. The Hackathon chair is responsible for keeping teams on track to their goal for the week which can sometimes also lead to the suggestion of abandoning the current strand of work and focus on a new approach.

The constant self reflection enforced by the SCRUM sessions makes sure that the teams set themselves rather small work packages which also leads to a more direct feedback on their progress. The measured speed-up compared to the original code is a very good progress indicator, which also leads to a competitive atmosphere in the room.

D. Post-Hackathon Activities

After the Hackathon week the teams and mentors typically work together remotely for a period of time to tie up the loose ends from the hackathon rush. Furthermore. the local organizers encourage the teams to report about their code improvements at suitable conferences and workshops. Lastly, the pool of Hackathon attendees is at the same time a pool of potential future mentors. The idea here is to repeat spread the knowledge even further in a sustainable manner and to reduce the burden on the system and compiler vendors who supplied most of the mentors at the initial hackathons.

III. Hackathon Trends

Over 60 scientific codes have been ported to heterogeneous platforms via these hackathons.

Team Profile: Hackathons have had teams with senior scientists and expert programmers —to teams comprising of all students starting to learn to use GPUs —to teams consisting of a balanced mix of faculty, students and staff. There have also been teams with only domain scientists clearly seeking help from computer scientists and have managed to use GPUs on Day 1 and also gotten noticeable speedup by Day 5 of the hackathon week.

Collaborative Model: Typically in a team, the domain scientist explains the code, the programmer breaks down the problem into several pieces and start refactoring the code to adapt to architectures, a tool developer points out if the issue is with the code or the tool itself and a mentor drives the team with proposed solutions and prototypes. This also shows that participants of all kinds are stakeholders in advancing science.

HPC Applications: A variety of applications spanning Astrophysics, Quantum Chemistry, Molecular Dynamics, Biophysics, Particles physics, Computational Fluid Dynamics, Climate/Weather/Ocean modeling and more have been ported to GPUs in these hackathons. The lines of code of the applications have ranged from 1000 to 100,000. Section IV highlights some of the cherry-picked case studies.

Programming Languages/Models: The teams typically bring along applications written in C/C++ or Fortran, at an increasing rate also Python and sometimes Kokkos [1] too. Some codes use libraries like Thrust. Sometimes there are applications that are already demonstrating reasonable speedup and show potential to achieve more and sometimes the code is a bare sequential code begging for acceleration on powerful GPUs.

Communication: For fast interactions between and across team members these hackathons have switched to Slack-based communication channel. In such intense hacking setting, there is little room for slow email-based communications. Sometimes not all members of a scientific code can be present at the hackathon due to travel restrictions, for example and the teams need to collectively interact with another that may be in different geographic locations. Using slack has allowed participants to stay in touch with their even after hackathons.

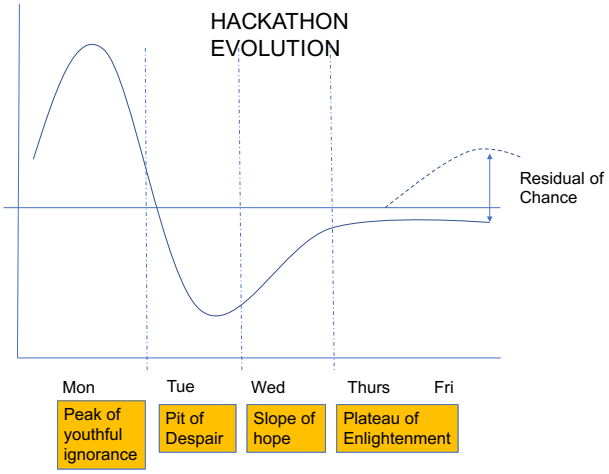

Figure 1. The various stages the hackathon participants find themselves in over the course of the week

Figure 1 captures the trend of a variety of emotions that the participants go through during the 5-day hackathon time period. Early in the week, the teams typically run into all sorts of problems such as their sequential code not even compiling or executing and if it does, they show inaccurate results. Towards the middle of the week, teams are hopeful and enlightened to learn that problem has a solution and are already running their codes on GPUs. Towards the end of the week, teams retrospect different lessons learned in that week and share the outcome with each other further elaborated in Section V. Teams either leave quite delighted with their progress or leave with at the very least the next set of possibilities to explore.

IV. Case Studies

This section will highlight some of the cherry-picked case studies out of 60 applications ported to large scale systems via these hackathons. Each case study is a short story from the local hackathon organizer or the team lead of the application discussing the effort, challenges and outcome. **Machine Setup(s):** Participants in these hacakathons have used large heterogeneous clusters such as TITAN (K20x) and recently Summit-Dev (P100) at ORNL, a new 108 compute node (K80) GPU-accelerated institutional cluster at BNL among others. Teams at the Eurohack access clusters at CSCS consisting of JURECA nodes with K80 GPU and Haswell processors as well as the OpenPOWER JURON system of POWER8 processors and NVLink attached P100 GPUs. The software harness of these machines include a number of compiler tools, profilers, performance analysis tools and other scientific libraries.

A. PALM4GPU: Simulating Turbulent Flows

PALM is a finite-difference large-eddy simulation (LES) model to study turbulent atmospheric and oceanic flows[3]. It is used both for basic research (e.g. turbulent transport above heterogeneous terrain, effect of turbulence on the growth of cloud droplets) and applied research questions (e.g. flow within and behind wind farms, effects of turbulence on airplanes during takeoff and landing, turbulent wind fields in cities). The flow model is coupled with a Lagrangian particle model (LPM) PALM consists of about 100.000 lines of Fortran 95-based code with some Fortran 2003 extensions and external libraries such as netCDF and FFTW are used. Parallelization is based on two-dimensional domain decomposition using MPI and hybrid parallelization with MPI and OpenMP is also available. Even though the goal seemed realistic, over the course of the hackathon the team struggled heavily with this legacy code. The nature of the "organically" grown code base—a result of multiple students' theses plus grant-projects—caught up during the refactoring for adding OpenACC directives. The team struggled with incorrect results that have lain dormant in the code and for a lack of a constant code maintainer have stopped the effort after the hackathon [2].

B. JuRor: Real-Time Fire Propagation Simulation

The JuRor team is part of a German research project called "ORPHEUS" which targets improved safety in underground facilities in case of fire. One part of the project is the real-time simulation of the propagation of the fire in the underground structure to guide fire fighters to where they are needed most. The simulation was compressed to a miniapp of about 2,500 lines of C++11 code. The weakly compressible Navier-Stokes equations are numerically solved using finite differences in a fractional step method using an orthogonal projection. The time marching scheme is split into the three major steps: advection, diffusion and pressure calculation. The advection equation is calculated using a semi-Lagrangian scheme, the diffusion equation solver is based on an implicit Jacobi scheme, and the pressure equation is handled by a (geometric) multi-grid method. The team used the week to add OpenACC directives to the code in the most computationally intense parts early on and spend most time in optimizing data movement, results published in [3].

C. AstroGPU: Astrophyics

The Brookhaven National Lab (BNL) hackathon hosted a team that comprised of present and past students (at different levels of GPU coding experience) from Stony Brook University (SBU). The team used the structured adaptive mesh refinement library AMReX [4], applicable to the astrophysics codes Maestro [5] and Castro [6]. By stripping down version of Castro from 100K lines of code to only 8K lines of code, StarLord, the team found it much easier to explore different ideas. StarLord and AMReX use C++ for the memory management, parallelism, and drivers and Fortran for the compute kernels. The choice was made to use CUDA Fortran for this work, since it seemed to work well with both new and old Fortran codes. Prior to the hackathon, the team spent a lot of time cleaning up interfaces in StarLord and splitting some kernels into multiple pieces to allow them to start offloading the work piece by piece to the device. Over the course of the hackathon week, the team progressed from having their GPU code being slower than a single CPU core to performing about 20x faster on a single GPU compared to a single CPU core on SummitDev—the precursor for the "Summit" system to be installed at OLCF in 2017/2018[4].

D. Kinectic Model Builder: Parameter Estimation of Chemical Processes

A team comprising of Ph.D. and Master's students from the Chemical and Biomolecular Engineering, University of Delaware, with no prior GPU programming experience learned a great deal by participating in the hackathon. They learned to use profilers that indicated that LU decomposition within their ODE solver code, if run on GPU could give them performance gains on large systems. The team although did not achieve a noticeable speedup while at the hackathon, they continued to build their code on the university's local cluster post the hackathon and witnessed great performance gains on a GPU. The LU decomposition workload of an explicit ODE solver was transferred to the GPU. In a complex molecular system of 904 species, for which LU decomposition is performed on a 904x904 matrix at every time step within the ODE solvers algorithm (5000 time steps), the GPU ODE solver implementation (after profiling) took 5.9s and also demoed better scalability for larger matrix sizes. These results indicate that the solvers will scale even better on larger systems.

E. NekCEM:Higher Order Computational Electromagnetics

NekCEM (Nekton for Computational ElectroMagnetics) is a spectral element discontinuous Galerkin solver for electromagnetics featuring geometric flexibility, spectrally accurate numerical convergence, and an efficient parallel MPI and (with the help of the Hackathons) OpenACC implementations. Prior to the hackathon, the NekCEM code had limited OpenACC and GPU support, all implemented by a former student. The hackathon was a chance for many members of the NekCEM team to quickly attain the knowledge needed to implement and optimize OpenACC code. Though there are resources available explaining OpenACC and how to efficiently use it (including examples), they do not cover the myriad of compiler bugs and OpenACC bugs. Many of these can be solved with enough searching, but it is quite

[3]http://palm.muk.uni-hannover.de

[4]https://www.olcf.ornl.gov/summit

discouraging to make only minor changes and then spend 30 minutes to find a solution. Working off of the knowledge of the mentors and experts at the hackathon, these problems can quickly be solved, allowing for the implementation to continue unimpeded. For the NekCEM team, the Hackathon was key to getting an efficient OpenACC version of our Maxwell solver, resulting in more than a 2.5 $\times$ speedup (compared to 16 MPI ranks) on over 16k nodes of Titan [7]. The GPU performance was used as a part of their successful R&D100 application.

F. CoMD: Molecular Dynamics (MD) Exascale Proxy App

The BNL hackathon hosted a team that included a diverse group of students spanning computational biology and computer science backgrounds from UDEL and UH, with a key team member participating solely remotely. The team's application code was CoMD, a molecular dynamics exascale proxy application developed by the DoE Exascale Co-Design Center for Materials in Extreme Environments [8]. The team's two major goals were to create and evaluate a GPU-accelerated version of CoMD using OpenACC, and to investigate extending the CoMD implementation to incorporate support for Particle Mesh Ewald (PME) long-range electrostatics. The original CoMD application was written in C, with some existing adaptations for MPI, OpenMP, and CUDA. Since the internals of CoMD were unfamiliar to several team members, and there was a general disparity in individual programming expertise, this presented an early challenge that led to a bifurcation of the team into a subgroup focused on OpenACC GPU acceleration of CoMD Lennard-Jones (L-J) force calculation loops, and a subgroup focused on adaptation of CoMD to support PME.

One subgroup focused on acceleration of L-J force calculations interacted heavily with the remote team member who was most familiar with CoMD at the outset, and who had prepared an initial starting point OpenACC implementation prior to the start of the Hackathon. Over the course of the hackathon, the speedup obtained for the L-J force kernels was improved substantially, through careful use of directives to cause parallel reductions to be done increasingly efficiently without serialization bottlenecks in complex loop nests, giving an overall GPU speedup of $2\times$ vs. OpenMP on the host CPUs, a result that was also observed in small scale MPI runs.

The other subgroup focused on adaptation of CoMD to implement PME electrostatics faced challenges associated with both algorithmic adaptation issues as well as refactoring of the existing library interfaces for the PME implementation they chose to use. Over the course of the Hackathon, the PME subgroup narrowed the scope of their activities to match their time constraints, focusing on adaptation of key data structures for efficient access on the GPU, minimizing host-GPU data transfers, and adapting individual loop nests for OpenACC.

Overall, the team as a whole had an excellent real-world experience working with third-party source code, gained hands-on experience with multiple tools for profiling and performance analysis, debugging, revision control, compilers, and team messaging and collaboration.

G. VMD: Analysis of MD Simulations

The National Center for Supercomputing applications (NCSA) hackathon in 2015 encouraged teams to submit projects that would be appropriate for GPU acceleration using the early OpenACC tools available at that time. A team of biomolecular modeling domain scientists from U. Illinois proposed a project to use OpenACC for GPU acceleration of existing MD analysis algorithms in the molecular visualization and analysis package VMD [9]. The algorithms of interest were originally implemented in a mixture of C++, Tcl, and Python scripts for processing simulation trajectories by unwrapping or rewrapping atomic coordinates according to periodic boundary conditions, aligning molecular structures, and computing root mean squared deviations of aligned structures.

The selected algorithms were known to present some implementation challenges for OpenACC acceleration due to a variety of complex loop nests and parallel reduction patterns, while involving relatively few arithmetic operations per memory reference. The team's use of the early OpenACC toolchain exposed a variety of unexpected compatibility problems between the nascent OpenACC runtime implementation and applications that already contained code that initialized and used CUDA on the GPUs. To link the newly-developed OpenACC code into VMD, the existing CUDA subsystem in VMD had to be disabled until the OpenACC runtime was later improved. The team had written a straightforward C/C++ based serial implementations of their target algorithms prior to the start of the hackathon, this made adding OpenACC to the key subroutines of interest easier while at the hackathon. The team discovered that significant kernel launch and data transfer overheads in the early OpenACC runtime implementation were preventing them from achieving their performance goals, so the team changed tack and re-implemented the kernels a second time at a lower level using CUDA, which allowed them to achieve performance levels limited by primarily by PCIe host-device transfer bandwidth and GPU global memory bandwidths.

By the end of the hackathon the team had become familiar with both OpenACC and CUDA programming, and the kernels they developed were structured well enough to become useful to the molecular modeling community with a minor amount of additional work. A little over a year after the end of the hackathon, more robust and polished versions of the CUDA versions of the kernels that the team developed were integrated into VMD.

H. QUDA: Lattice QCD Library

Simulating the theory of strong interactions, which is called Quantum Chromodynamics (QCD), requires significant compute resources. Scientists from this area have always been keen to adopting compute devices providing exceptional compute performance. GPUs started to be used for accelerating the performance critical kernel i.e. linear solver, well before CUDA or directive based GPU programming models became available (see, e.g., [10]). As more efficient yet complex solvers become available and as GPUs-based systems continue to evolve, the efforts of implementing such solvers start to become heavy for individual research groups and therefore community efforts to provide shareable implementations are becoming more important. One such effort goes under the name QUDA [11], https://lattice.github.io/quda/

During the hackathon at Jülich Supercomputing Centre in 2017, two different teams of QCD researchers worked on augmenting QUDA. The "Quantum Tornados" team collected researchers both from the US and Europe to integrate a new multigrid-type solver, namely DD-αAMG [12]. During the hackathon the team accomplished the adaptation of the DD-αAMG algorithm into QUDA. At the last day the team reported significant improvement in the number of solver steps compared to a multigrid solver that was already available in QUDA. The code was tested on compute nodes with Intel Xeon processors and K80 GPUs as well as on nodes with IBM POWER8 processors and NVLink-attached P100 GPUs.

The "Lattice-QCD Bielefeld" team focused on integrating there routines for simulating QCD at non-zero temperatures into QUDA. This required implementation of missing routines and changes in the library interface. This code was also running on multiple GPUs concurrently.

I. Asynchronator: Asynchronous Linear Solver

One of the teams, which attended the hackathon at the Jülich Supercomputing Centre (JSC), focused on GPU-enabling a new solver that belongs to an emerging class of highly scalable, fault tolerant methods that are based on asynchronous iterations (see, e.g., [13]). Asynchronicity can help to significantly improve scalability. The team comprised of PhD students of the Marie Skłodowska-Curie training network HPC-LEAP: http://www.hpc-leap.eu/. The team systematically explored different technologies like Thrust, CUB or cuBLAS and targeted different GPU-accelerated platforms, including IBM Minsky nodes with POWER8 processors and P100 GPUs. After having issues with compilers fixed, the team could report on performance improvements due to use of GPUs for selected kernels, which were in the range of 2.6 to 7.3.

J. NASA Langley HPC Incubator CNDE team

The Computational Nondestructive Evaluation team was focused on porting an elastodynamic finite integration technique code to GPUs. The code simulates ultrasound waves in solid materials and uses a second order accurate finite difference type approach with an explicit leap-frog time stepping method to solve a 3D PDE for the elastodynamic motion. The code calculates nine variables on a staggered grid. The starting point of the code was in C++ and was a code version optimized to run on Knights Landing cores. The initial goals for the hackathon were to get the code running on GPUs and to achieve speed increases by running on GPU. The code was successfully ported to GPUs and was tested for a grid size of 1024 x 1024 x 16 and run for 4096 time steps. The initial plan was to use only OpenACC, however the OpenACC version showed a 3x slowdown (30 ms runtime) compared to running on KNL (15 ms runtime). During the hackathon it was determined that improved performance was achieved by using CUDA for more complex programming options (compared to OpenACC). It was also surmised that starting with a more simplified kernel than the code that had been optimized for MIC hardware may have led to better results on the GPU hardware.

K. Hackathon Takeaway and Proposal:

The takeaway from the above stories is that almost all teams have had either students or postdocs participate. They get mentored by professionals in the field, learn to practically apply their classroom theoretical knowledge on real problems and learn to work in a goal-oriented team-based setting. From an education perspective, such strong evidence of students' successful involvement begs us to revisit our lecture-based teaching patterns especially for courses such as parallel programming, which is being mandatory in universities.

The authors of this paper encourage universities to consider policies that will allow universities to give credits to students who participate in such hackathons. Universities could consider awarding credits based on the students' level of engagement in the effort. Students could be required to submit a comprehensive report on what they learned with respect to the technical challenges they faced, newer tools and techniques they learned and how they plan to apply them in their projects or assignments; report evaluated by their advisors. Hackathons do not need to be necessarily on GPUs, it would even be a C/C++ hackathon for example. Section IV-D discusses a somewhat similar hands-on based program that some universities are already adopting.

V. Hackathon Outcomes and takeaway

This section discusses some of the outcomes and takeaways from the hackathons conducted so far. **Outcome**

- Large computing facilities are opening up their systems to scientists and programmers thus building a large customer pool
- Compiler bugs are being identified in software tools and due to the on-site participation of key OpenACC

vendors and developers, these errors are immediately flagged for correction
- The gap between tool developers and its users are narrowed down
- Teams benefit from closer proximity to expert mentors
- Large legacy scientific code are ported to the state-of-the-art software and hardware
- Hardware and software are stress tested by using varieties of scientific code bases
- Next-generation workforce is being trained and prepared to use future machines
- Collaborations are established between teams from national labs, academia and organizations leading to innovative ideas
- Mentors are being created and developed
- An "inverted hierarchy" is created, where experienced students tell the faculty what to do, which further strengthening collaborative effort
- Knowledge is gained outside of an individual's comfort zone while learning how people work together in teams to achieve a common goal or purpose.

Some of the takeaways include:

- Profile the code, if possible ahead of time, to understand performance bottlenecks; various profilers exist such as TAU, Score-P
- Use a local machine if you have one to compile and execute your code where you have full control; this can be very helpful for preliminary experiments
- Start with a smaller kernel in the large code base, or start with a miniapp For example, a team at the BNL hackathon created a stripped down version of Castro [6] called StarLord that was only 8k lines of code instead of 100k lines of code. The smaller code base made it much easier to try out different ideas.
- Identify a good starting point before applying parallelization concepts, if the code is efficiently parallelized for X86, it may not really be a good starting point for GPUs so one may need to consider the sequential version of the code as the starting point to parallelize for GPUs.
- Debugging can be a challenge, especially when you get CUDA kernel errors. Sometimes the simplest techniques work the best. One of the teams adopted the motto 'when in doubt, comment it out.

VI. Hackathon from educator's perspective

Observing students' performance in various hackathons point out that the gap between strong needs of quality programmers in national labs, other organizations like NSA, NASA and industries and the lack of relevant and sufficient education in computer science graduate and undergraduate programs needs to be identified and addressed.

To that end, a critical step that several universities are taking currently is to mandate the parallel programming course at the undergraduate level. This course typically covers parallel programming languages, parallel architectures, parallel algorithms, profilers, performance analysis tools and steps to optimize and analyze parallelized code. This is crucial for the next-generation workforce to be able to think parallel and use current and future large scale platforms. These courses are typically offered to junior and senior level but need to be introduced in the introductory programming classes as well.

In additional to parallel programming, there are other important tools that the students need to learn to use and these may not be always taught as part of a curriculum. Software development is not easy for any domain scientists, as also mentioned in the practical book by Scopatz and Huff [14] that focuses on how to use Python to help collect, analyze, build and publish results for research in physics-based field. Some of the fundamental scientific and technical computing tools that the students should know include Makefile, Python, CMake, BASH, linking, debuggers, Software Licensing, Distribution; Programming Languages (at least one) and Mathematical libraries for linear algebra, Version control. More importantly, these topics should be supported with hands-on training.

A model that supports such hands-on training-based course is Georgia Tech's Vertically Integrated Program (VIP). The author, Chandrasekaran, from UDEL runs a multi-year VIP-HPC program for the undergraduate students in her university, UDEL [15]. It allows students to work on research projects with faculty for an extended period of 3 years. Unlike a typical classroom and lecture-based course, this VIP course almost functions like a hackathon. The students work in teams. Each team is assigned a project. Students are given access to dedicated clusters consisting of state-of-the-art hardware, scientific computing tools and compilers. The students enter their notes into physical VIP notebooks that the instructor supplies. They detail on their approaches, thoughts, proposals, action items and summaries of their projects. The class meets once a week. Students stay in touch with each other and the instructor via slack communication channel. The students are also paired up with senior research students (Masters or Ph.D. students) thus creating a 'learn from mentor' environment and that seems to facilitate faster and easier progress on projects.

The students use Git version control system to commit their codes and at the end of each semester, each team prepares, presents posters and demos their project to faculty from other department, technical managers and recruiters from industries in a poster showcase event. The students' projects are reviewed by peers and faculty. Depending on the final grades, the students are allowed to continue to enroll into the forthcoming semesters. The students receive credits (typically 1 for sophomore and 2 for junior and senior) but this varies from person to person depending on his/her project quality, their aptitude, applicability and

approachability to a given problem.

Such a practical training based model or a similar model could be adopted by universities to encourage students to allocate specific time to apply theoretical knowledge to a real problem and collect credits towards their degree program. This model does not increase the time taken to degree, instead builds a strong foundation at the undergraduate level for students to be aware and knowledgeable of key topics critical in the field of HPC. Such a long-term problem-based learning environment could also be considered instead of capstone course where the students get very limited time (1-2 semesters) to explore and build a complete, functional, high-quality open-source product.

VII. Summary and Outlook

The ORNL-led GPU Hackathons have proved to be quite effective in helping the domain scientists jump start their initial GPU porting efforts or getting them on track to further optimize their codes. In this regard, this form of HPC training should be considered for the major computing facilities where the users are expected to use the computing resources efficiently. On the other hand, the Hackathon format could also be generalized to help train the next-generation HPC users, especially the students enrolled in computer science or scientific computing courses, or early-career scientific researchers. One could imagine that as part of the course work, term projects could be given to the students in a one or two-day "mini-hackathon" setting, provided sufficient preparation similar to the Hackathon preparation discussed earlier in this paper is given. The most crucial point from an education's perspective is offer more hands-on training to students.

VIII. Acknowledgements

These hackathons would not have been successful without the help of some very important players that include Duncan Poole, Pat Brooks, Julia Levites, Mat Colgrove, Michael Wolfe, Brent Leback, Robert Searles, Kyle Friedline, Sandra Wienke among others. Special thanks to Dana Hammond, Cara Campbell Leckey, Elizabeth Gregory and William Schneck of NASA for contributing case studies from their hackathon. J.E. Stone acknowledges support from NIH grant 9P41GM104601. S. Chandrasekaran is grateful to Andy Novocin, Director of the VIP program for bringing this program to UDEL.

References

[1] H. C. Edwards, C. R. Trott, and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," *Journal of Parallel and Distributed Computing*, vol. 74, no. 12, pp. 3202–3216, 2014.

[2] H. Knoop, T. Gronemeier, C. Knigge, and P. Steinbach, *Porting the MPI Parallelized LES Model PALM to Multi-GPU Systems – An Experience Report*. Cham: Springer International Publishing, 2016, pp. 508–523. [Online]. Available: https://doi.org/10.1007/978-3-319-46079-6_35

[3] A. Ksters, S. Wienke, and L. Arnold, "Performance portability analysis for real-time simulations of smoke propagation using openacc," in *Proceedings of the 2nd International Workshop on Performance Portable Programming Models for Accelerators (P^3MA), currently being published with Springer*, 2017.

[4] "New block-structured amr framework (amrex)," https://github.com/AMReX-Codes/amrex, 2016.

[5] A. Nonaka, A. Almgren, J. Bell, M. Lijewski, C. Malone, and M. Zingale, "Maestro: An adaptive low mach number hydrodynamics algorithm for stellar flows," *The Astrophysical Journal Supplement Series*, vol. 188, no. 2, p. 358, 2010.

[6] A. Almgren, V. Beckner, J. Bell, M. Day, L. Howell, C. Joggerst, M. Lijewski, A. Nonaka, M. Singer, and M. Zingale, "Castro: A new compressible astrophysical solver. i. hydrodynamics and self-gravity," *The Astrophysical Journal*, vol. 715, no. 2, p. 1221, 2010.

[7] M. Otten, J. Gong, A. Mametjanov, A. Vose, J. Levesque, P. Fischer, and M. Min, "An mpi/openacc implementation of a high-order electromagnetics solver with gpudirect communication," *The International Journal of High Performance Computing Applications*, vol. 30, no. 3, pp. 320–334, 2016.

[8] J. Mohd-Yusof and N. Sakharnykh, "Optimizing comd: A molecular dynamics proxy application study," in *GPU Technology Conference (GTC)*, 2014.

[9] W. Humphrey, A. Dalke, and K. Schulten, "VMD – Visual Molecular Dynamics," *J. Molecular Graphics*, vol. 14, no. 1, pp. 33–38, 1996.

[10] G. I. Egri, Z. Fodor, C. Hoelbling, S. D. Katz, D. Nogradi, and K. K. Szabo, "Lattice QCD as a video game," *Computer Physics Communications*, vol. 177, no. 8, pp. 631 – 639, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010465507003025

[11] M. A. Clark, R. Babich, K. Barros, R. C. Brower, and C. Rebbi, "Solving Lattice QCD systems of equations using mixed precision solvers on GPUs," *Comput. Phys. Commun.*, vol. 181, pp. 1517–1528, 2010.

[12] A. Frommer, K. Kahl, S. Krieg, B. Leder, and M. Rottmann, "Adaptive Aggregation Based Domain Decomposition Multigrid for the Lattice Wilson Dirac Operator," *SIAM J. Sci. Comput.*, vol. 36, pp. A1581–A1608, 2014.

[13] A. Frommer and D. B. Szyld, "On asynchronous iterations," *Journal of Computational and Applied Mathematics*, vol. 123, no. 1, pp. 201 – 216, 2000, numerical Analysis 2000. Vol. III: Linear Algebra. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S037704270000409X

[14] A. Scopatz and K. D. Huff, *Effective Computation in Physics*, 1st ed. O'Reilly Media, May 2015.

[15] U. of Delaware, "Vertically integrated program - high performance computing," http://vip.udel.edu/hpc, 2017.