Journal Title
XX(X):2–13

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

A Survey of Software Implementations Used by Application Codes in the Exascale Computing Project

Thomas M. Evans[1]*, Andrew Siegel[2], Erik W. Draeger[3], Jack Deslippe[4], Marianne M. Francois[5], Timothy C. Germann[5], William E. Hart[6], and Daniel F. Martin[4]

Abstract

The US Department of Energy Office of Science and the National Nuclear Security Administration (NNSA) initiated the Exascale Computing Project (ECP) in 2016 to prepare mission-relevant applications and scientific software for the delivery of the exascale computers starting in 2023. The ECP currently supports 24 efforts directed at specific applications and six supporting co-design projects. These 24 application projects contain 62 application codes that are implemented in three high-level languages—C, C++, and Fortran—and use 22 combinations of GPU programming models. The most common implementation language is C++, which is used in 53 different application codes. The most common programming models across ECP applications are CUDA and Kokkos, which are employed in 15 and 14 applications, respectively. This paper provides a survey of the programming languages and models used in the ECP applications codebase that will be used to achieve performance on the future exascale hardware platforms.

Keywords

Exascale Computing Project, GPU, computational physics applications, programming models

---

*This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. DOE will provide access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

Introduction

Leadership-class high-performance computing (HPC) systems could enable game-changing advances in science, engineering, and national security applications that are critical to the US Department of Energy (DOE) mission. In the 2023–2024 time frame, the DOE computing centers at the Oak Ridge Leadership Computing Facility (OLCF), Argonne Leadership Computing Facility (ALCF), and Lawrence Livermore National Laboratory (LLNL) will stand up three new exascale systems: Frontier, Aurora, and El Capitan. This paper defines an *exascale system* as a computer capable of greater than or equal to 1 EFlops for general 64-bit floating point operations. The planned exascale systems, along with the 200 PFlops Summit computer at the OLCF[†], use GPUs for the majority of their performance ($\gtrsim 94\,\%$). As discussed in Alexander et al. (2020), DOE began the Exascale Computing Project (ECP) in 2016 to ready a suite of mission-critical applications for deployment at the DOE leadership computing facilities in time for the arrival of the exascale platforms in 2023. In addition to the target applications, the ECP is developing a complete supporting software ecosystem consisting of mathematics, visualization, linear and nonlinear solvers, and performance tuning libraries and utilities through the ECP Software Technology (ST) focus area, as summarized in Heroux et al. (2020).

The ECP Application Development (AD) focus area currently contains 24 applications that span chemistry and materials, energy production and transmission, earth and space science, data analytics and optimization, and national security. Each application has a formally defined challenge problem, the details of which are found in Siegel et al. (2021). The ECP is a formal project with quantitative metrics for success that are measured through project-defined key performance parameters (KPPs). AD projects are grouped into two objective categories, generically referred to as first and second KPP (KPP-1 and KPP-2). The 24 applications projects are listed by objective category in Tables 1 and 2.

The KPP-1 applications have a quantitative performance figure of merit (FOM). The FOMs are defined as a ratio of performance work rates on the current platform relative to a baseline measurement from the start of the project. In each case, these baseline measurements were performed on the OLCF's Titan or ALCF's Mira computers. Each

---

[1] Oak Ridge National Laboratory, USA
[2] Argonne National Laboratory, USA
[3] Lawrence Livermore National Laboratory, USA
[4] Lawrence Berkeley National Laboratory, USA
[5] Los Alamos National Laboratory, USA
[6] Sandia National Laboratories, USA

Corresponding author:
Thomas M. Evans, Oak Ridge National Laboratory, 1 Bethel Valley Rd, Oak Ridge, TN 37831
Email: evanstm@ornl.gov

[†] https://www.olcf.ornl.gov/summit

Table 1. ECP applications targeting KPP-1.

Project name	Category	Short description	Lead lab
LatticeQCD	Chemistry and materials	Exascale lattice gauge theory opportunities and requirements for nuclear and high-energy physics	Fermilab
NWChemEx	Chemistry and materials	Stress-resistant crops and efficient biomass catalysts	Ames
EXAALT	Chemistry and materials	Molecular dynamics at the exascale	LANL
QMCPACK	Chemistry and materials	Find, predict, and control material properties	ORNL
ExaSMR	Energy production	Coupled Monte Carlo neutronics and fluid flow simulation of small modular reactors	ORNL
WDMApp	Energy production	High-fidelity whole device modeling of magnetically confined plasmas	PPPL
WarpX	Energy production	Plasma wakefield accelerator design	LBNL
ExaSky	Earth and space science	Cosmological probe of the Standard Model	ANL
EQSIM	Earth and space science	Seismic hazard risk assessment	LBNL
E3SM-MMF	Earth and space science	Regional assessments in earth systems models	SNL
CANDLE	Data analytics and optimization	Accelerate and translate cancer research	ANL

FOM is defined uniquely by the application project. For example, the work rate for the ExaSMR project Monte Carlo (MC) particle transport application is particles per wall-clock time. Full descriptions of the application FOMs can be found in Siegel et al. (2021). The ECP KPP-1 objective is for $50\,\%$ of the applications to achieve an FOM ≥ 50 on their defined challenge problems.

The second class of applications are categorized as KPP-2 projects. This metric is intended to assess the creation of new science and engineering capabilities that can fully exploit exascale resources. Each KPP-2 project has a work plan that defines the computational capabilities that are needed to execute a science and engineering campaign using leadership-class computers. At the end of the project, $50\,\%$ of these applications must demonstrate these capabilities on their project challenge problems. The work plans for the KPP-2 projects are listed in Siegel et al. (2021).

The six AD co-design projects provide support for applications, focusing on novel capabilities that are not supported by the ECP software ecosystem. The co-design projects are software middleware oriented around computational motifs as defined

Table 2. ECP applications targeting KPP-2. Note that SPARC and EMPIRE are considered a single project within ECP even though they are separate national security projects.

Project name	Category	Short description	Lead lab
GAMESS	Data analytics and optimization	Biofuel catalyst design	Ames
ExaAM	Chemistry and materials	Additive manufacturing of qualifiable metal parts	ORNL
ExaWind	Energy production	Predictive wind plant flow modeling	NREL
Combustion-Pele	Energy production	Combustion engine and gas turbine design	SNL
MFIX-Exa	Energy production	Multiphase flow reactor design	NETL
ExaStar	Earth and space science	Demystify the origin of chemical elements	LBNL
Subsurface	Earth and space science	Carbon capture, fossil fuel extraction, waste disposal	LBNL
ExaSGD	Data analytics and optimization	Reliable and efficient planning of the power grid	PNNL
ExaBiome	Data analytics and optimization	Metagenomics	LBNL
ExaFEL	Data analytics and optimization	Light source-enabled analysis of molecular structure	SLAC
Ristra	National security	High-energy density physics and materials under extreme conditions	LANL
MAPP	National security	Inertial confinement fusion and pulsed power applications	LLNL
SPARC	National security	Virtual flight test of reentry vehicles	SNL
EMPIRE	National security	Electromagnetic plasma physics	SNL

in Alexander et al. (2020). The original 13 motifs are dense linear algebra, sparse linear algebra, spectral methods, particles, structured grids, unstructured grids, MC, combinatorial logic, graph traversal, graphical models, finite-state machines, dynamic programming, backtrack, and branch-and-bound. Additionally, the co-design projects include fundamental computational motifs in finite element methods (FEM), adaptive mesh refinement (AMR), artificial intelligence (AI), and data reduction, compression, and analysis. The complete list of ECP AD co-design projects is given in Table 3.

This paper briefly summarizes the programming models employed and the current performance measurements attained by ECP applications. Each year, the AD application portfolio undergoes a rigorous review, the results of which are extensively documented and publicly released at `https://exascaleproject.org`. Full details on

Table 3. ECP AD co-design projects.

Project name	Principal motifs	Applications
CEED	Unstructured grids, FEM	ExaAM, ExaSMR, Ristra, MARBL, SPARC
AMReX	Structured grids, AMR	ExaWind, Combustion-Pele, MFIX-Exa, WarpX, ExaSky, ExaStar
CODAR	Data reduction and analysis	WDMApp, NWChemEx, CANDLE
CoPA	Particles, spectral methods	EXAALT, ExaAM, WDMApp, ExaSky, WarpX, MFIX-Exa
ExaGraph	Graph traversal, combinatorial logic	ExaBiome, ExaWind, NWChemEx, SPARC, EMPIRE,
ExaLearn	Machine learning, AI	ExaSky, CANDLE

challenge problem definitions, capability plans, and performance on pre- and early exascale hardware are available in the 2021 AD review report by Siegel et al. (2021).

Programming Models Used in ECP

Intranode implementations of each application require programming models to build code that can use the GPUs for each architecture. The ECP ST ecosystem supports several programming models and compilers for GPU programming. One principal consideration for this approach is to provide performance portability across a range of GPU architectures that will constitute the exascale landscape (e.g., NVIDIA, Intel, AMD), each with its own underlying assembly languages (e.g., PTX on NVIDIA, GCN on AMD).

Table 4 lists the full suite of application codes organized by AD project. The primary DOE platform portability software efforts in ECP are Kokkos, as described in Edwards et al. (2014); RAJA, as described in Beckingsale et al. (2019); and Legion, as described in Bauer et al. (2021). Several projects are also using or experimenting with third-party vendor implementations, including OpenMP[‡], OpenACC[§], OpenCL[¶], CUDA[‖], HIP[**], and SYCL[††]. The SYCL implementation used throughout these applications is part of the Intel Data Parallel C++ (DPC++) framework that provides a standard SYCL implementation with several Intel-specific extensions[‡‡]. Some applications are built using AI frameworks (e.g., PyTorch, TensorFlow). Nearly all projects use Python

---

[‡] https://www.openmp.org
[§] https://www.openacc.org
[¶] https://www.khronos.org/opencl
[‖] https://docs.nvidia.com/cuda/cuda-runtime-api/index.html
[**] https://github.com/ROCm-Developer-Tools/HIP
[††] https://www.khronos.org/sycl
[‡‡] https://software.intel.com

for various purposes, particularly workflows, postprocessing, and data analysis. Several of these technologies—including OpenMP, OpenACC, and the LLVM compiler suite that supports them—are supplemented by projects within the ECP ST focus area.

Table 4. AD application codes. Custom abstraction layers built by projects are marked with an asterisk (⋆).

Application project	Code	Main language	GPU programming model
LatticeQCD	Chroma	C++	Kokkos, QUDA library⋆
	CPS	C++	OpenMP, GRID, QUDA libraries⋆
	GRID	C++	HIP, SYCL, CUDA
	MILC	C	GRID, QUDA libraries⋆
	QUDA	C++	HIP, SYCL, CUDA
NWChemEx	NWChemEx	Python, C++	CUDA, HIP, SYCL
GAMESS	GAMESS	Fortran	libcchem⋆, libaccint⋆
	libcchem	C++	CUDA, HIP, DPC++
EXAALT	ParSplice	C++	-
	LAMMPS	C++	Kokkos
	SNAP	C++	Kokkos
	LATTE	Fortran	OpenMP, CoPA (BML/Progress)
ExaAM	AMPE	C++	RAJA
	Diablo	Fortran	OpenMP
	ExaCA	C++	Kokkos
	ExaConstit	C++	RAJA, CEED (MFEM)
	ExaMPM	C++	Kokkos
	MEUMAPPS-SS	C++	Kokkos
	TruchasPBF	Fortran	AMReX
	Tusas	C++	OpenMP, Kokkos, CUDA
QMCPACK	QMCPACK	C++	OpenMP, CUDA, HIP
	RMG	C++	CUDA, HIP
ExaWind	Nalu-Wind	C++	Kokkos
	AMR-Wind	C++	AMReX
	OpenFAST	Fortran	-
Combustion-Pele	PeleC	C++	AMReX
	PeleLM	C++	AMReX
	PelePhysics	C++	AMReX
ExaSMR	NekRS	Fortran, C++	CEED (libParanumal/OCCA)
	OpenMC	Python, C++	OpenMP
	Shift	C++	CUDA, HIP
WDMApp	GENE	Fortran, C++	gtensor⋆
	GEM	Fortran, C++	OpenACC, OpenMP
	XGC	C++	CoPA (Cabana), OpenMP, Kokkos
MFIX-Exa	MFIX-Exa	C++	AMReX
WarpX	WarpX + PICSAR	C++	AMReX
ExaSky	HACC	C++	CUDA, HIP, OpenCL

	CRK-HACC	C++	CUDA, HIP, OpenCL
	Nyx	C++	AMReX
ExaStar	FLASH-X	Fortran, C++	OpenMP
	CASTRO	C++	AMReX
	SEDONA	C++	AMReX
EQSIM	SW4	C++	RAJA
Subsurface	Chombo-Crunch	C++	Proto*
	GEOSX	C++	RAJA
E3SM-MMF	E3SM	Fortran, C++	YAKL*, OpenACC, OpenMP
ExaSGD	ExaGO	C, C++	RAJA
	HiOP	C++	RAJA
	ExaPF-jl	Julia	CUDA
CANDLE	CANDLE	Python	TensorFlow, PyTorch
ExaBiome	MetaHipMer	C++	CUDA, HIP, SYCL/DPC++
	HipMCL	C++	CUDA, HIP, SYCL/DPC++
ExaFEL	M-TIP	C++	CUDA, HIP, OpenCL
	PSANA	Python, C++	Legion, OpenMP
	CCTBX	Python, C++	CUDA
Ristra	Symphony	C++	Kokkos
	FUEL	C++	Kokkos
	Portage	C++	Kokkos
MAPP	MARBL	C++	RAJA
	Miranda	Fortran	OpenMP
SPARC	SPARC	C++	Kokkos
EMPIRE	EMPIRE	C++	Kokkos

Many ECP applications achieve platform portability through the application programming interfaces (APIs) provided by co-design middleware, particularly AMReX, CEED, and CoPA. In this mode, the role of the co-design efforts broadens to provide primary data structure, performance optimization, and algorithmic support. Furthermore, although most applications have internal APIs that manage device-based code and data structures, several applications have built custom external-facing frameworks to support platform portability. Examples of these include Proto* in the Subsurface project, Yet Another Kernel Launcher (YAKL)† in E3SM-MMF, and gtensor‡ in WDMApp. All these custom frameworks ultimately use a platform-specific native layer (e.g., CUDA, OpenMP) in their instantiations. In Table 4, the co-design, ST, and custom layers

---

[*] `https://github.com/applied-numerical-algorithms-group-lbnl/Proto`
[†] `https://github.com/mrnorman/YAKL`
[‡] `https://github.com/wdmapp/gtensor`

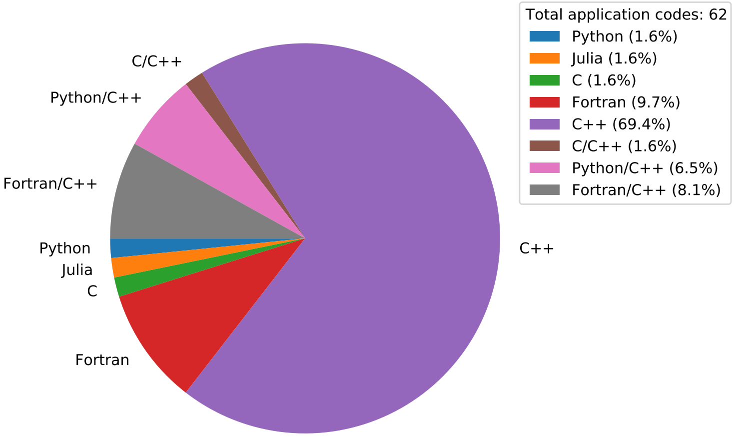

Figure 1. Principal code languages used in ECP application codes.

do not include the underlying implementations (e.g., CUDA, OpenMP, SYCL); if a programming model is listed, it is used directly in the core codebase.

Figure 1 shows the current distribution of principal coding languages and programming models used in the 62 ECP AD codes. The majority of codes are written in C++ (e.g., C++11, C++14, C++17), although at the beginning of the project, there was a more even split between Fortran and C++, as shown in Figure 2. Several project codes began life as Fortran codes but have since have been partly or completely rewritten by using C++ over the intervening three years. Although each project might assess its own risks and path forward based on the specific needs of its application, the most common reasons for moving from Fortran to C++ are the ability to leverage programming abstractions that require closures (e.g., Kokkos, RAJA) and the perception that Fortran support for advanced architectures has lagged significantly behind C++. Today, C++ is the primary implementation in $> 69\,\%$ of codes, and it is used in a total of $\sim 85\,\%$ of all applications in ECP AD. Conversely, Fortran was used in $\sim 31\,\%$ of all codes at the beginning of the project but is only used in $\sim 18\,\%$ today. Python is included as a principal language for several application codes; however, many additional application codes employ Python at the code construction and input processing steps. In these cases, because Python is not formally used at runtime, it has not been listed as a principal language.

The programming models used in ECP AD codes are illustrated in Figure 3. These data show that CUDA, Kokkos, OpenMP, and HIP are the most commonly used tools for achieving GPU performance in the ECP AD codebase; they are used in roughly 24, 23, 19, and $19\,\%$ of the codes, respectively. Platform portability provided by the co-design projects is used in approximately $23\,\%$ of all application codes, and ST programming

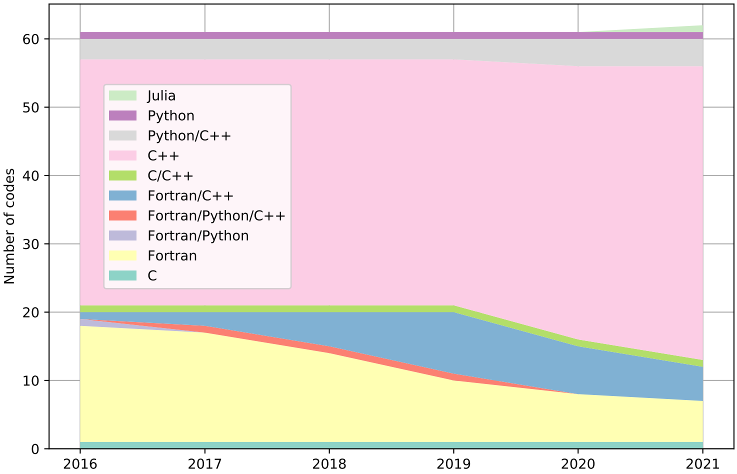

Figure 2. Distribution of primary languages in ECP AD applications over the lifetime of the project.

models account for another $\sim 35\,\%$. This represents a significant benefit of the ECP because much of the fine-scale architectural details of algorithm implementation have been leveraged through these projects. Nonetheless, $\sim 55\,\%$ of the application codes in the ECP are using either native implementations (e.g., CUDA, HIP) or their own custom implementations built on top of these languages. This reflects the difficulty of developing universal platform-portable programming models that span a diverse set of scientific applications.

Internode parallelism is primarily handled by using the message passing interface (MPI); a small number of codes use UPC++. Summit supports both device-to-device and host-to-host parallel communication operations. Device-to-device, or *CUDA-aware MPI*, allows the client to directly transfer data between GPUs. In theory, this should provide performance benefits because data can be transferred directly using network interface controller memory, bypassing the host. However, except for very large message sizes, the communication performance on Summit has been mixed in practice, and the principal benefit of this approach is the convenience of not needing to manually transfer data back to the host to perform communications. Additionally, this programming strategy will have future benefits because the exascale platforms will likely only allow direct device-to-device communication.

The papers in this special issue are focused on the coupling implementations and, in some cases, frameworks used by six application projects in the ECP: WDMApp, E3SM-MMF, EQSIM, ExaStar, ExaAM, and MFIX-Exa. These coupling strategies encompass aspects of both inter- and intranode parallelism. As such, they generally rely on both MPI and GPU programming models. Full details on the interaction of coupling and programming models are discussed in these papers.

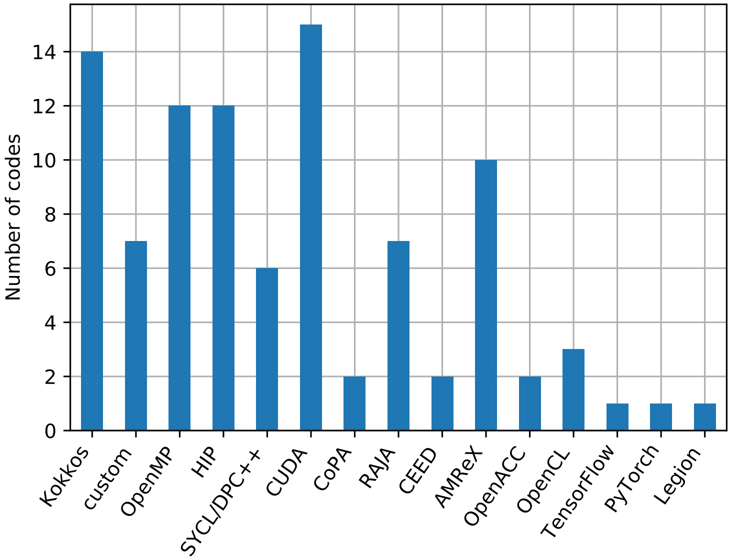

Figure 3. GPU programming models used in ECP application codes. The SYCL implementations imply the version supported by Intel's DPC++ framework.

Current Performance of ECP Applications

The ECP tracks regular performance progress on all KPP-1 applications. In the first two years of the project, each KPP-1 application generated benchmark FOM measurements on either Titan at the OLCF or Mira at ALCF. As new measurements are performed on Summit, the current FOM values are updated and posted in a dashboard.

Current performance measurements for AD projects are shown in Figure 4. The latest measurements indicate that five out of the eleven KPP-1 projects have already achieved an FOM increase of 50 or greater on Summit, and two additional projects observe FOM improvement well over 50 when extrapolated to the full machine. Although these early results are exceptionally encouraging, we stress that these measurements have been performed on the NVIDIA architecture on Summit. The exascale platforms will use AMD (Frontier at OLCF and El Capitan at LLNL) and Intel (Aurora at ALCF), and the software environments and architectures for both systems are less mature than NVIDIA technology. The work to prepare these applications to efficiently use AMD and Intel GPUs is ongoing, and this represents the crucial final stage of the ECP.

Conclusion

The ECP AD focus area supports 62 application codes in 24 projects with the objective of preparing these applications to use the exascale platforms that will be delivered in 2023. Over half of the KPP-1 applications have already seen $\geq 50\times$ performance

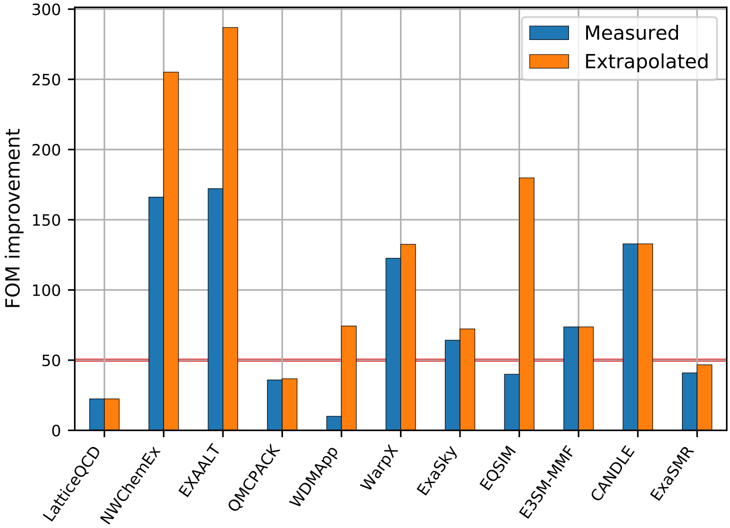

Figure 4. Current performance measurements on Summit at OLCF. The extrapolated results project the measured results to the full machine assuming linear scaling. The red line shows the end-of-project objective of a $50\times$ improvement in FOM.

improvements on the pre-exascale Summit computer. Thus, a 3–4 year investment in code optimization for GPUs can realize significant performance benefits. Because the ECP has also produced many supporting technologies to help in this task, the authors expect that this time frame will be shorter in future efforts.

Some of the interesting trends that can be seen across the ECP code landscape are the contraction of core implementation languages and the proliferation of programming model approaches. The use of Fortran has been significantly impacted by the increasing diversity of GPU hardware. No new Fortran code efforts have been initiated since the start of the ECP, and the use of C++ has proliferated from 41 to 53 codes over the course of the project, all at the expense of Fortran. Fortran-only codes have shrunk from seventeen codes at the beginning of the project to six, primarily due to the move toward programming abstractions requiring closures and concerns over consistent and timely Fortran support across architectures. The usage trends of GPU programming models do not show any clear favorites, although significant benefits have accrued from the use of ECP co-design and ST technologies. This suggests that the choice of programming models is still largely dictated by the application's algorithmic requirements, and there is no unified approach that will serve all scientific computing domains.

Acknowledgements

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the US Department of Energys Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nations exascale computing imperative. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the US Department of Energy under Contract No. DE-AC05-00OR22725. Work for this paper was supported by Oak Ridge National Laboratory (ORNL), which is managed and operated by UT-Battelle, LLC, for the US Department of Energy (DOE) under Contract No. DEAC05-00OR22725. This work was performed under the auspices of the US Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LANL is operated by Triad National Security, LLC, for the National Nuclear Security Administration of US Department of Energy (Contract No. 89233218CNA000001). This research used resources of the National Energy Research Scientific Computing Center (NERSC), a US Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the US Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the US Department of Energy or the United States Government.

References

Alexander F, Almgren A, Bell J, Bhattacharjee A, Chen J, Colella P, Daniel D, DeSlippe J, Diachin L, Draeger E, Dubey A, Dunning T, Evans T, Foster I, Francois M, Germann T, Gordon M, Habib S, Halappanavar M, Hamilton S, Hart W, (Henry) Huang Z, Hungerford A, Kasen D, Kent PRC, Kolev T, Kothe DB, Kronfeld A, Luo Y, Mackenzie P, McCallen D, Messer B, Mniszewski S, Oehmen C, Perazzo A, Perez D, Richards D, Rider WJ, Rieben R, Roche K, Siegel A, Sprague M, Steefel C, Stevens R, Syamlal M, Taylor M, Turner J, Vay JL, Voter AF, Windus TL and Yelick K (2020) Exascale applications: skin in the game. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 378(2166): 20190056. DOI:10.1098/rsta.2019.0056. URL https://royalsocietypublishing.org/doi/10.1098/rsta.2019.0056.

Bauer M, Lee W, Slaughter E, Jia Z, Renzo MD, Papadakis M, Shipman G, McCormick P, Garland M and Aiken A (2021) Scaling implicit parallelism via dynamic control replication. In: *PPoPP '21: Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. New York, NY, USA: Association for Computing Machinery. ISBN 9781450382946. URL https://dl.acm.org/doi/proceedings/10.1145/3437801.

Beckingsale D, Burmark J, Hornung R, Jones H, Killian W, Kunen A, Pearce O, Robinson P, Ryujin B and Scogland T (2019) RAJA: Portable performance for large-scale scientific applications. In: *2019 IEEE/ACM International Workshop on Performance, Portability, and Productivity in HPC (P3HPC)*. pp. 71–81.

Edwards HC, Trott CR and Sunderland D (2014) Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and*

Distributed Computing 74(12): 3202–3216. DOI:https://doi.org/10.1016/j.jpdc.2014.07.003. URL `http://www.sciencedirect.com/science/article/pii/S0743731514001257`. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

Heroux M, McInnes L, Thakur R, Vetter J, Li S, Ahrens J, Munson T and Mohror K (2020) ECP software technology capability assessment report—public. Technical Report ECP-RPT-ST-0002-2020, Exascale Computing Project. URL `https://www.exascaleproject.org/reports/`.

Siegel A, Draeger E, Deslippe J, Evans T, Francois M, Germann T, Martin D and Hart W (2021) Map applications to target exascale architecture with machine-specific performance analysis, including challenges and projections. Technical Report PM-AD-1110, Exascale Computing Project. URL `https://www.exascaleproject.org/reports`.