ECP Milestone Report

A Survey of MPI Usage in the U. S. Exascale Computing Project

WBS 2.3.1.11 Open MPI for Exascale (OMPI-X) (formerly WBS 1.3.1.13), Milestone STPM13-1/ST-PR-13-1000

David E. Bernholdt[1,*], Swen Boehm[1], George Bosilca[2], Manjunath Gorentla Venkata[1], Ryan E. Grant[3], Thomas Naughton[1], Howard P. Pritchard[4], Martin Schulz[5,6], and Geoffroy R. Vallee[1]

[1]Computer Science and Mathematics Division, Oak Ridge National Laboratory
[2]Innovative Computing Laboratory, University of Tennessee, Knoxville
[3]Center for Computing Research, Sandia National Laboratories
[4]Ultrascale Research Center, Los Alamos National Laboratory
[5]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory
[6]Institut für Informatik, Technical University of Munich, Germany

[*]Author for correspondence. Email: bernholdtde@ornl.gov

Originally released 2017-10-13, last updated 2018-06-27

DOCUMENT AVAILABILITY

Reports produced after January 1, 1996, are generally available free via US Department of Energy (DOE) SciTech Connect.

Website http://www.osti.gov/scitech/

Reports produced before January 1, 1996, may be purchased by members of the public from the following source:

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
Telephone 703-605-6000 (1-800-553-6847)
TDD 703-487-4639
Fax 703-605-6900
E-mail info@ntis.gov
Website http://www.ntis.gov/help/ordermethods.aspx

Reports are available to DOE employees, DOE contractors, Energy Technology Data Exchange representatives, and International Nuclear Information System representatives from the following source:

Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831
Telephone 865-576-8401
Fax 865-576-5728
E-mail reports@osti.gov
Website http://www.osti.gov/contact.html

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

ECP Milestone Report
A Survey of MPI Usage in the U. S. Exascale Computing Project
WBS 2.3.1.11 Open MPI for Exascale (OMPI-X) (formerly WBS 1.3.1.13), Milestone STPM13-1/ST-PR-13-1000

Office of Advanced Scientific Computing Research
Office of Science
US Department of Energy

Office of Advanced Simulation and Computing
National Nuclear Security Administration
US Department of Energy

Originally released 2017-10-13, last updated 2018-06-27

ECP Milestone Report
A Survey of MPI Usage in the U. S. Exascale Computing Project
WBS 2.3.1.11 Open MPI for Exascale (OMPI-X) (formerly WBS 1.3.1.13), Milestone STPM13-1/ST-PR-13-1000

Approvals

Submitted by:

David E. Bernholdt, Oak Ridge National Laboratory STPM13-1/ST-PR-13-1000	Date

Approval:

Rajeev Thakur, Argonne National Laboratory Director, Software Technologies Exascale Computing Project	Date

Revision Log

Version	Creation Date	Description	Approval Date
1.0	2017-10-13	Original, based on ExasMPI Workshop paper submission, adding original questionnaire and raw free text responses.	
1.1	2018-06-27	Updated to reflect version of paper accepted for publication (DOI: 10.1002/cpe.4851) in a special issue of Concurrency and Computing: Practice and Experience for the 2017 ExaMPI Workshop proceedings. Light revisions from v1.0	

EXECUTIVE SUMMARY

The Exascale Computing Project (ECP) is currently the primary effort in the United States focused on developing "exascale" levels of computing capability, including hardware, software and applications. In order to obtain a more thorough understanding of how the software projects under the ECP are using, and planning to use the Message Passing Interface (MPI), and help guide the work of our own project within the ECP, we created a survey. Of the 97 ECP projects active at the time the survey was distributed, we received 77 responses, 56 of which reported that their projects were using MPI. This paper reports the results of that survey for the benefit of the broader community of MPI developers

TABLE OF CONTENTS

LIST OF TABLES

1. INTRODUCTION

This paper summarizes the results of a survey of current and planned Message Passing Interface (MPI) usage patterns among applications and software technology efforts that are part of the Exascale Computing Project (ECP).

The ECP [1] is currently the primary project in the United States developing "exascale" levels of computing. It is a collaborative effort of two organizations within the the U.S. Department of Energy (DOE), the Office of Science and the National Nuclear Security Administration (NNSA), though applications originating from other U.S. agencies are represented as well. The ECP is officially chartered with accelerating the delivery of a capable exascale computing ecosystem to provide breakthrough modeling and simulation solutions that address the most critical challenges in scientific discovery, energy assurance, economic competitiveness, and national security[1]. In the context of ECP, *exascale* is defined as computing systems 50 times faster than the nation's most powerful supercomputers in use in 2016, when the project was started.

ECP is focused on three areas of activity:

Application Development (AD) supports application and cross-cutting co-design activities to advance applications readiness for exascale problems, exascale software stacks and exascale hardware environments.

Software Technology (ST) aims at building a comprehensive software stack to support the productive development of (performance) portable applications across diverse exascale architectures.

Hardware Technology (HT) supports the vendor and national laboratory research and development activities required to develop node and system designs for at least two capable exascale systems with diverse architectural features.

Two of these three areas (AD and ST) include a wide range of efforts that touch on, interface with or rely on MPI (or alternative inter-node communication mechanisms). Hence, it is critical for us, as one of the providers of MPI to ECP, to characterize the usage of existing MPI mechanisms within this particular exascale community. Further, to spur a more focused development of future MPI capabilities it is equally critical to identify what types of new constructs applications may need on their quest towards exascale.

MPI is a critical communication API for applications in the high performance computing (HPC) area, and is used extensively by applications of interest to the Department of Energy and ECP. Since its initial introduction in 1994, the MPI standard has been regularly updated to increase its relevancy to parallel applications, to include more versatile and scalable constructs, but also to standardize the best practices put forward by application developers. The most recent version of the MPI standard, 3.1 [2], was released in 2015, and the MPI standardization body, the MPI Forum, is actively working towards future versions. MPI provides different communication techniques for point-to-point communication as well as a rich suite of collective operations. Point-to-point communication occurs between two processes, while collective operations involve all processes in a given application/job or subsets thereof. Such sets of processes are managed using a concept called communicators, which are structures that allow for communication isolation and software encapsulation. Point-to-point communications can take two different approaches to communication: the traditional and most commonly used send/receive semantics, which is referred to as two-sided communication, requires explicit participation by both the source and the target process. The source calls a send operation that sends the message to the target and the target must call a receive operation, which either can dictate where the incoming message is coming from or can receive messages from any eligible process. The other main point-to-point communication method is called Remote Memory Access (RMA), which is a one-sided communication method. RMA only requires the involvement of either the source or the destination process to move data to or from the calling node.

The ST area of ECP includes two efforts specifically focused on providing implementations of MPI with exascale capabilities: OMPI-X, which is includes five institutions, led by Oak Ridge National Laboratory, and includes the authors of this paper, focuses on extending the Open MPI [3, 4] implementation and ExaMPI, which is led by Argonne National Laboratory, targets MPICH [5, 6]. In order to gain a general perspective of how ECP projects are using MPI and how they plan to use it in their exascale versions, we undertook a survey and summarize its results here.

[1] `https://exascaleproject.org/exascale-computing-project/`

Table 1: Summary information about the ECP MPI survey. For convenience, the table also indicates the specific sections of this paper and tables where the corresponding results are presented.

Survey Group	Questions	Results Presented	
Project demographics	1–17	Sec. 3	Table 2
Application demographics	18–26	Sec. 4.1	Table 4
Non-MPI applications	27–30	Sec. 3	Table 3
Basic performance characterization	31–34	Sec. 4.2	Table 5
MPI usage patterns	35–41	Sec. 4.3	Tables 6, 7
MPI tools ecosystem	42–47	Sec. 4.4	Table 8
Memory hierarchy details	48–49	Sec. 4.5	Table 9
Accelerator details	50–52	Sec. 4.6	Table 10
Resilience	53–55	Sec. 4.7	Table 11
Use of other programming models	56–57	Sec. 4.8	Table 12
MPI with threads	58–64	Sec. 4.9	Table 13

2. MOTIVATION

The results of a broad survey of current MPI and planned exascale MPI usage patterns among many different U.S. DOE applications and software technologies efforts as part of ECP provide a unique perspective. They highlight MPI integration issues and challenges into target applications and are useful to both ECP projects and MPI implementors and researchers. The results of our investigation show that MPI will not only remain relevant in the exascale era, but will continue to be embraced by the HPC community for its flexibility, portability and efficiency. Nevertheless, multiple new additions to the standard and optimizations are highly desired by the exascale application community. The survey also shows the most commonly used functions as well as those that are in the greatest need for further optimization, providing a wealth of information to researchers on topics of interest to MPI applications. In addition to the open research questions on how to solve challenges facing MPI for exascale applications, there are a number of interesting engineering challenges that must be overcome. The best solutions for issues facing MPI on its path to exascale capabilities are those that match the semantics needed for next generation applications. Understanding the needs of these applications and their methods, for which they are intending to use MPI, is crucial for innovating practical communication library solutions. This survey provides the information necessary to determine how MPI extension proposals should be structured for a maximum long term utility, as well as highlighting areas in which further investigation is needed to determine solutions that continue to reflect the needs of applications.

Motivation for some of the questions on the survey was found in current open questions before the MPI Forum. Significant topics under consideration for MPI standardization include fault tolerance, thread usage with MPI interfaces and the requirements for thread network addressability, GPU integration in MPI as well as the refinement of Remote Memory Access (RMA) capabilities. This paper describes the current state of the art in terms of needs for these features, as well as capturing feature set requests that are not currently in debate for future MPI specification versions, like active messages.

3. OVERVIEW OF SURVEY

The survey was designed to gather information on a per-project basis from across the ECP[2]. It included 64 questions, organized into 11 groups, as summarized in Table 1.

Each respondent's path through the survey depended on their answers to certain questions. For example, the "non-MPI applications" section was completed only by those projects indicating that they were *not* using

[2]Formally, the ECP is a single large project with a work breakdown structure (WBS) that runs four, and in some places five, levels deep. The scope and funding levels of these lowest levels of the WBS are in line with what most researchers would think of as a standalone research and development project. As such, we tend to think of the ECP as a coordinated collection of individual projects. Hence, for simplicity, we use the term *project* to refer to a level-4 or level-5 element of the Exascale Computing Project and *ECP* to refer to the overarching effort as a whole.

Table 2: A breakdown of the numbers of ECP projects and number of those projects responding to the survey, based on the organizational breakdown used within the ECP. The "Using MPI" column lists the number of projects reporting that they are actually using MPI, which is the set analyzed in the remainder of this paper.

Project Category	ECP efforts	Number of Responses	Using MPI
Application Development (total)	**36**	**28**	**28**
Science and Energy Appl,	25	20	20
NNSA Applications	5	3	3
Other Agency Applications	1	1	1
Co-Design	5	4	4
Software Technologies (total)	**61**	**49**	**28**
Prog. Models and Runtimes	13	8	2
Tools	13	8	4
Libraries and Frameworks	13	13	10
Data Mgmt. and Workflows	10	10	7
Data Analysis and Vis.	5	3	2
System Software	6	6	3
Resilience and Integrity	1	1	0
Overall Total (AD+ST)	**97**	**77**	**56**

MPI. While the majority of questions were multiple choice with either one or multiple responses allowed, some questions allowed for free-form responses. Individual questions will be presented as part of the survey results below.

The survey was constructed and made available to participants using Google Forms, and received responses between 19 May 2017 and 28 July 2017. The survey was not anonymous. Principal investigators of ECP AD and ST projects were contacted and asked to have someone knowledgeable from their team complete the survey. Respondents were asked to identify both themselves and the effort for which they were responding. They were also asked if they were willing to be contacted for follow-up discussions. A secondary purpose for the survey was to identify projects that might be useful partners in the co-design of exascale MPI capabilities and implementations. As our work progresses, we envision contacting willing project teams for deeper requirements gathering, testing of appropriate capabilities or enhancements.

Table 2 summarizes the number of projects and the number of survey responses received, organized according to the ECP's structure. The number of efforts listed in each category is based on the efforts for which we were able to obtain contact information for the principal investigator, and excluded efforts that were not directly technical. It is also worth noting that the ECP is fluid, in that additional efforts are being launched as gaps are identified, so this survey effectively represents a snapshot of the ECP taken in May 2017.

Naturally, not all of the ECP efforts use MPI. Table 3 summarizes the responses to several questions we asked of those efforts that indicated that they were not using MPI. All 21 projects meeting this criteria were in the ST area of ECP (Q30). We consider this natural, as the "software technologies" area of the ECP includes numerous efforts focused on runtimes, tools, and other software that provides alternatives to MPI, that are node-local, or that target system software used by multiple applications. Responses to Q27 confirm this expectation, listing primarily node-local parallel programming models. In response to a question as to whether they had been "driven away" from using MPI (Q28), the only direct response indicated that "The lack of standard or consistent ABIs is an ongoing problem for performance tools." Only one MPI-based third-party library, MDHIM[3] was mentioned as a dependency by these non-MPI projects (Q29).

To provide the most useful statistics for the survey results, in the remainder of this paper we exclude the non-MPI projects from our analysis, and focus on the efforts corresponding to the last column of Table 2.

[3]Because we cannot know for certain if we've identified the correct software, we do not attempt to provide bibliographic citations for software mentioned in responses to the survey.

Table 3: Non-MPI Applications

No.	Question and Responses	AD	ST	Overall
27	**What parallel programming models are you using (node level and global)?** * *(text)*			
	CUDA (3); Current prototype does use MPI. Production version will use Mercury/Margo (1); Kokkos (2); Most of our work is thread-local (1); Node level (1); None (2); Not applicable (4); OpenMP (3); Project is primarily about a build tool. We *may* use MPI for parallel builds, but more likely we'll coordinate through parallel filesystem. (1); Qthreads (1); TBB (1); UPC++ (1); threads (1);			
28	**Are there issues with the MPI standard or implementations which have pushed you away from using MPI in your application?** *(text)*			
	No (6); Not applicable (3); Separation of concerns: intra-node vs. inter-node parallelism (1); The lack of standard or consistent ABIs is an ongoing problem for performance tools (1); We are not an application, so we don't need MPI (1);			
29	**What third-party libraries that your application depends upon use MPI?** * *(text)*			
	None (8); Unknown (1); Not applicable (2); QEMU (1); MDHIM (1);			
30	**Do you expect the exascale version of your application to use MPI?** * *(single)*			
	Yes	0	1	1
	No	0	20	20

4. SURVEY RESULTS

The following sections present and discuss the results of the survey following the structure of the survey itself. The detailed tables of results follow a common format. Question numbers are listed primarily to provide a short, unique identifier for each question. The main text of the question is shown in **bold face** font. An asterisk (*) following the question indicates that a response was required. In some cases, questions included additional context or explanations, which is not shown here, for the sake of space, but can be found in full in Appendix A.After that, in parenthesis, is the type of response allowed. "Single" means that only a single answer could be chosen. "Multiple" indicates that any number of answers (including zero) could be marked. "Text" denotes text fields that allowed a free-form response. In some cases, multiple choice questions also had an "other" option, which allowed free-form text responses in addition to the choices provided. These are denoted as "single+text" or "multiple+text".

Results for multiple choice questions are presented as percentages of the number of projects in the category (AD, ST and Overall). If answers were not required or multiple answers were accepted, totals may be over or under 100%.

Results for free-form text responses were analyzed and summarized by the authors, attempting to preserve the key features of the response while consolidating similar responses as much as possible. We present separate sets of results for the AD and ST projects on the expectation that they may be qualitatively different. In the interest of space, we do not attempt to provide an "Overall" summary that merges the AD and ST summaries. Text summaries include the number of times a point was mentioned in all of the responses, rather than percentages of the numbers of projects. Raw versions of the free-form responses are included in Appendix B.

4.1 Application Demographics

We asked eight questions to get some basic background information about the various ECP projects, as shown in Table 4, in order to better understand the context in which the projects were using MPI.

Overall, a significant majority (61%) of the projects involve multiple applications or libraries versus single applications or libraries (Q18). Though the AD projects were more strongly dominated by multiple applications (68%). From this point onward, we use "application" as a generic term, whether the project involves one or more distinct applications, libraries, tools, etc.

The vast majority of projects expect the exascale version of their application to use MPI as well (Q20). Only one project is transitioning to the Legion runtime environment as a goal of their project.

Table 4: Application Demographics

No.	Question and Responses	AD	ST	Overall
18	**Does your project involve a single application/library or several that you consider to be distinct?** * *(single)*			
	Single application/library	32%	46%	39%
	Multiple distinct applications/libraries	68%	54%	61%
20	**Do you expect the exascale version of your application to use MPI?** * *(single)*			
	Yes	96%	89%	93%
	No	4%	11%	7%
21	**If your anwer above was "no", why not?** *(text)*			
	AD: Project plan is to use Legion, with MPI as a backup.			
	ST: More interested in remote procedure calls than message passing.			
22	**Do you have an abstraction layer that hides the MPI calls? Or do most of your developers write MPI calls directly?** * *(single)*			
	Abstraction layer	79%	46%	62%
	Direct MPI calls	21%	54%	38%
23	**Do you have mini-applications that capture the MPI behavior of your application? If so, are they available to the community?** *(single)*			
	Yes, and they are available to the community	43%	39%	41%
	Yes, but they're limited availability	29%	11%	20%
	No	29%	50%	39%
24	**What programming languages do you call MPI from?** * *(multiple+text)*			
	C	50%	71%	61%
	C++	82%	68%	75%
	Fortran	46%	25%	36%
	Python	2%	2%	3%
	Other responses			
	PETSc	4%	0%	2%
	Tcl	4%	0%	2%
25	**What additional languages would you like to be able to call MPI from?** *(text)*			
	AD: Julia (2 mentions), Python (1), C/C++ (1)			
	ST: *none*			
26	**What third-party libraries that your application depends upon use MPI?** * *(text)*			
	AD: ADIOS (1 mention), ADLB (1), AMReX (1), basic graph libraries (1), CGNS (1), CNTK (1), Global Arrays (1), HDF5 (6), hypre (3), MFEM (1), Metis (1), mpi4py (1), NetCDF (1), PETSc (1), ParaView/Catalyst (1), pio (1), pNetCDF (2), Silo (1), SuperLU-Dist (1), Trilinos (3)			
	ST: ADIOS (1), BLAS (1), CGNS (1), clang (1), DIY (2), HDF5 (2), hypre (2), MKL-Cluster (1), Mercury (2), NetCDF (1), PETSc (1), pNetCDF (2), PTScotch (1), Pardiso (1), PatMetis (1), Polly (1), ROMIO (1), SLATE (1), ScaLAPACK (BLACS, PBLAS) (1), SuperLU_dist (1), TSan (1), Trilinos (4), zlib (1)			

We see that most applications (62%) use an abstraction layer to hide MPI calls (Q22), though this is much stronger in the AD projects (79%) than ST (46%).

Since mini-applications have become recognized as a very useful way for researchers outside of a given project to interact with applications that might otherwise be too large or complex to deal with, we asked the ECP projects whether they had mini-applications that reflected the MPI-related behaviors of their applications (Q23). We found that 71% of AD projects have mini-applications, but only 50% of ST projects do. For both groups, approximately two-thirds of the mini-applications are made available to the community, while the remainder have limited availability.

We asked all projects which programming languages they use to make MPI calls (Q24) and found some different patterns. Among the AD projects, 82% report using C++, and roughly half report using each of C and Fortran (multiple answers were accepted for this question, so the implication is that many projects use more than one programming language and make MPI calls from more than one). For the ST projects C dominated (71%) with C++ as a close second (68%). Although Python is in general quite popular in HPC programming, only a small fraction of projects reported making MPI calls from Python. When asked what additional languages they would *like* to be able to call MPI from (Q25), Julia was mentioned twice. Additional responses to Q25 mentioned Python and C/C++, which we interpret as indicating that projects are not currently calling MPI from these languages, but would like to in the future.

In Q26, we asked about MPI-based third-party dependencies for the responding projects. Not surprisingly, since this was a free-text question, we received a wide range of responses. However, several points seem noteworthy. First, for both AD and ST projects, HDF5 (6 mentions in AD and 2 in ST) and I/O libraries in general (ADIOS, CGNS, pNetCDF, pio, Silo) figured prominently. Second, various numerical libraries (AMReX, hypre, PETSc, SuperLU-Dist, Trilinos, MKL-Cluster, Paradiso, SLATE, ScaLAPACK) were also frequently mentioned. We observe that although we asked specifically about MPI-based third-party libraries, many of the responses do not, in fact, utilize MPI, although we list them for completeness.

4.2 Basic Performance Characterization

Table 5 presents four questions intended to obtain some very basic information related to the performance of applications using MPI.

Q31 asked whether production usage of the application was dominated by small or large messages, with 8 kB being the dividing line. In this case, responses were split roughly equally between small, large and both as dominating.

Half of the AD projects consider their application to be limited by message latency (Q32), with roughly one-third indicating that both bandwidth and/or message rate were limiting (multiple answers were allowed). For the ST projects, the results are somewhat different, with bandwidth being the top bottleneck (46%), followed by latency (39%). Of the free-text additional responses offered, we note that 11% of ST projects indicated MPI-based I/O was a bottleneck. These came from projects specifically working on I/O tools, under the "Data Management and Workflows" area of ECP.

We asked an open-ended question about one aspect of MPI that could be optimized to improve the performance of the project's application (Q33). Among the AD projects, improvements in latency received the most mentions (5), while being mentioned only once among the ST projects. Various aspects of collective operations were also prominent among the requests of both AD (4) and ST (7) projects. A number of other requests pertained in some way to threading (6 AD, 2 ST).

Finally, we asked what impact network topology had on applications (Q34). Only one AD project reported that they were actively mapping MPI ranks to resources based on network topology. Many projects have not found network topology to be an issue (13 AD, 11 ST), while a significant minority have found it to be an issue (9 AD, 9 ST).

4.3 MPI Usage Patterns

We asked a total of seven questions aimed at understanding which aspects of the MPI standard applications are using, and in what ways. Table 6 presents the first three questions. These pertain to the aspects of the MPI standard applications are currently using (Q35) and expect to use in their exascale versions (Q37), as well as which areas of the standard they're currently using in performance-critical sections of their applications

Table 5: Basic Performance Characterization

No.	Question and Responses	AD	ST	Overall
31	**Is your application (when run in production) typically dominated by small or large messages?** *(single+text)*			
	Small (< 8 kB)	29%	25%	27%
	Large (>= 8 kB)	36%	39%	38%
	Other responses			
	Both	32%	32%	32%
	All to all in subcommunicators	4%	0%	2%
	Unknown	0%	4%	2%
32	**Do you consider your application (when run in production) to be constrained by** *(multiple+text)*			
	Message latency	50%	39%	45%
	Message bandwidth	36%	46%	41%
	Message rate	32%	14%	23%
	Don't know	11%	1%	7%
	Other responses			
	Depends	0%	14%	7%
	I/O	0%	11%	5%
	Load imbalance	4%	4%	4%
33	**If there were one aspect of MPI that could be optimized to improve the performance of your application, what would you prioritize?** *(text)*			
	AD: ability to saturate network bandwidth from a single MPI rank (1 mention); asynchronous collectives (1); barrier in shared memory context (2); better fine grain support and dynamic tasking (1); collectives (2); convex partitions (1); don't know (2); fault tolerance (1); hardware all-reduce (1); interoperability with local threading and global task-based runtimes (1); latency (5); memory hierarchy support (1); MPI+X abstraction (1); multithreaded asynchronous (1); optimize data block size (1); predicted latency and bandwidth, available during execution (1); RMA (1); shared-memory MPI (1); thread multiple (1); thread support (2)			
	ST: all-reduce (blocking and non-blocking) (3 mentions); all-to-all (1); bandwidth for repartitioning of data (1); collectives (2); controlling rendezvous threshold (1); CPU use makes MPI untenable for remote procedure calls (2); dealing with large numbers of outstanding requests (1); don't know (2); latency (1); local collectives (1); MPI-IO aggregations based on topology (1); relaxing 2 GB limit on I/O (1); startup time at extreme scale (1); task support (1); thread support (2);			
34	**Is the performance of your application particularly sensitive to the network topology and the specific mapping of MPI processes to that topology?** *(text)*			
	AD: don't know (2 mentions); has been measured (1); measurement variability is too high to give a clear picture (1); no (12); only expected effects of topology on collectives (1); using topology-aware mapping may give up to 15% improvement (1); yes (7); yes, due to latency (1); yes, for non-contiguous/non-convex partitions (1);			
	ST: don't know (3 mentions); expect application/user to handle topology issues (1); no (11); yes (6); yes, for I/O forwarding or direct I/O operations (2); yes, for non-contiguous/non-convex partitions (1);			

Table 6: MPI Usage Patterns – Part A

No.	Questions
35	**What aspects of the MPI standard do you use in your application in its current form?** * *(multiple)*
36	**What aspects of the MPI standard appear in performance-critical sections of your current application?** * *(multiple)*
37	**What aspects of the MPI standard do you anticipate using in the "exascale" version of your application?** * *(multiple)*

	Q35: Current Usage			Q37: Exascale Usage			Q36: Performance Critical		
Responses	**AD**	**ST**	**Overall**	**AD**	**ST**	**Overall**	**AD**	**ST**	**Overall**
Point-to-point communications	96%	79%	88%	89%	71%	80%	93%	75%	84%
MPI derived datatypes	25%	21%	23%	21%	21%	21%	14%	7%	11%
Collective communications	86%	75%	80%	96%	68%	82%	64%	64%	64%
Neighbor collective communications	14%	14%	14%	32%	25%	29%	7%	11%	9%
Communicators and group management	68%	54%	61%	61%	50%	55%	29%	7%	18%
Process topologies	14%	7%	11%	32%	11%	21%	4%	4%	4%
RMA (one-sided communications)	36%	7%	21%	50%	36%	43%	21%	7%	14%
RMA shared windows	18%	7%	12%	21%	18%	20%	7%	7%	7%
MPI I/O (called directly)	25%	18%	21%	21%	18%	20%	4%	7%	5%
MPI I/O (called through a third-party library)	32%	21%	27%	36%	25%	30%	7%	11%	9%
MPI profiling interface	11%	0%	14%	11%	21%	16%	0%	4%	2%
MPI tools interface	0%	4%	2%	0%	18%	9%	0%	0%	0%

Table 7: MPI Usage Patterns – Part B

No.	Question and Responses	AD	ST	Overall
38	**What is the dominant communication in your application?** Check all that apply, recognizing that many applications have different communication patterns in different phases. *(multiple)*			
	Each process talks to (almost) every other process	25%	21%	23%
	Processes communicate in fixed "neighborhoods" of limited size	46%	46%	46%
	Processes communicate in "neighborhoods" of limited size that may change in different phases of the application or evolve over the course of a run	42%	36%	39%
	Communication is largely irregular	18%	21%	20%
39	**Can your application take advantage of non-blocking point-to-point operations...** *(multiple)*			
	To overlap communication with computation?	89%	71%	80%
	To allow asynchronous progress?	64%	64%	64%
	To allow event-based programming?	43%	29%	36%
40	**Can your application take advantage of non-blocking collective operations...** *(multiple)*			
	To overlap communication with computation?	71%	46%	59%
	To allow asynchronous progress?	46%	29%	38%
	To allow event-based programming?	25%	14%	20%
41	**Is MPI providing all the communication semantics required by your application? If not, what is missing?** *(multiple+text)*			
	MPI covers all my needs	43%	43%	43%
	Active messages	32%	14%	23%
	Persistent communications (communications with the same arguments, repeatedly executed, potentially allowing additional optimization if exposed to the MPI layer)	21%	14%	18%
	Other responses			
	Don't know	4%	4%	4%
	Exploring Legion as an alternative	4%	0%	2%
	GPU-to-GPU communications	4%	0%	2%
	Improved threading support	0%	4%	2%
	Interoperability with GASNet	4%	0%	2%
	Job-to-job communications	0%	7%	4%
	Load balancing	4%	0%	2%
	Reductions with variable sized abstract types	4%	0%	2%
	Remote procedure calls	0%	4%	2%

(Q36). The responses allowed for these questions are mostly based on chapters of the MPI specification, with a few additional, more specific features added.

Although the proportions differ between the AD and ST projects, we see that point-to-point communications (88% overall), collective communications (80%) and communicators and group management (61%) strongly dominate the MPI features used in the current versions of applications (Q36). In the exascale versions of the applications (Q37), the same three dominate, though at somewhat different levels (80%, 82% and 55%, respectively). At exascale, we also note a marked rise in the (planned) use of RMA (43% for communications, 20% for shared windows, up from 21% and 12%) as well as neighbor collectives (29%, up from 14%) and process topologies (21%, up from 11%). MPI I/O tends to be called more often through third-party libraries than directly (27% vs. 21% for current applications, 30% vs 20% for exascale). Overall 35% of projects report using MPI I/O in *either* form, and 12% of projects use it in *both* forms. Looking at which features of the specification appear in current performance-critical sections of code (Q36), we see that point-to-point and collective communications strongly dominate (84% and 64%), which is not surprising.

Comparing AD to ST responses in Table 6, we see noticeable drops in both point-to-point and collective communications for the ST projects compared to the AD projects. Since these drops are not compensated by an uptick in an alternative communication mechanism (RMA), we have to assume that there is a qualitative difference in the perception of MPI usage within the applications of the two groups. Comparing other responses, there appears to be significantly less use of some features by the ST projects compared to AD (process topologies, RMA) in current applications, though it appears that many ST projects plan to use RMA in the exascale version of their applications. Comparing the MPI features appearing in performance-critical code sections (Q36) between AD and ST projects, we observe that derived data types, communicators and group management, and RMA are notably more prevalent in AD usage than ST.

Table 7 presents the remaining four questions in this section. In terms of communications patterns (Q38), fixed neighborhoods of limited size dominate (46% overall), though more dynamic, but still limited neighborhoods run a close second (39%). All-to-all and irregular patterns represent a significant minority of the patterns reported (23% and 20%, respectively).

Non-blocking operations in MPI can be used to achieve several different goals, which we inquired about relating to both point-to-point (Q39) and collective (Q40) operations. In both cases, the primary goal in using non-blocking versions of these operations was to overlap communication with computation (80% overall for point-to-point, 59% for collectives). For point-to-point operations, allowing asynchronous progress was also a strong motivation (64%), though for collectives, it was not as strong (38%). There are also a significant minority of projects using non-blocking operations to facilitate event-based programming (36% for point-to-point, 20% for collectives). Comparing AD and ST responses for these questions, we see that ST projects overall report less usage of non-blocking point-to-point operations than AD projects, however, for collective operations, their usage patterns are more similar.

We also asked an open-ended question about whether MPI was providing the communications semantics required by the applications (Q41). We included two possible responses, active messages and persistent communications, based on historical and recent interest in these topics in the MPI Forum. We also allowed projects to add their own responses. Overall, 43% of projects responded that MPI covered all of their communication needs. 23% expressed interest in active messages, and 18% in persistent communications (with AD projects notably more interested in both than ST). Since persistent point-to-point operations have been part of the MPI standard for the last decade, this interest might be interpreted as specifically focusing on persistent collectives, which are still under discussion by the MPI Forum. Alternatively, it is possible that users are not sufficiently aware of the established capabilities, and some outreach would be useful.

The only additional topic that appeared more than once in the free-text responses pertained to job-to-job communications. This is a challenging topic, but the MPI Sessions working group is considering changes to the standard that might make it easier to address this issue [7]. Sessions could make it possible to assemble inter-communicators using information obtained from the system job manager. Other responses indicated interest in improved threading support, more capabilities for reductions and remote procedure calls.

4.4 MPI Tools Ecosystem

Table 8 presents the questions and responses received to six questions focusing on the use of tools with and/or for MPI applications. These questions were motivated primarily by discussions underway in the MPI Forum

Table 8: MPI Tools Ecosystem

No.	Question and Responses	AD	ST	Overall
42	**If you are using performance tools with MPI, how often do you encounter the need to use multiple tools during the same application run?** *(single)*			
	Often	0%	4%	2%
	Occasionally	36%	14%	25%
	Never	18%	25%	21%
	Don't use performance tools with MPI	43%	54%	48%
43	**Are there any MPI-related improvements you'd like to see for debuggers and other "correctness" tools?** *(text)*			
	AD: ability to call out to external tools (1 mention); better resolution for tracking message sizes (1); better scalability (1); current tools give either too much or too little detail (1); information about collective and reduction operations (1); mixed programming modes (1); MPI core file for offline replay within debug tools (1); no (1); race-condition analysis (1);			
	ST: better scalability (1 mention); no (1); no opinion (1);			
44	**Are there any MPI-related improvements you'd like to see for performance tools?** *(text)*			
	AD: better scalability (2 mentions); current tools give either too much or too little detail (1); differentiation between time in MPI library and time on the wire (1); MPI reduction for user profiling tools (1); no (1); upgrade to mpiP (1);			
	AST: analysis of time messages spend in each state (1 mention); autotuning in MPI-T (planned) (1); better scalability (1); like MPE+ jumpshot, other tools are too complicated (1); no (1);			
45	**Are you using the MPI Profiling Interface for anything other than running performance analysis tools? If so, please briefly describe your use.** *(text)*			
	AD: extract communication matrix for topology-aware mappping (1 mention); no (4);			
	ST: capture MPI traces for use in simulations (1 mention); Darshan (1); no (3); noise injection for debugging (1); record/replay for debugging (1);			
46	**What is your level of interest in having access to internal MPI performance data? Information could include items like function call time, load balance information, memory use, message queue information, network counters, etc.** *(single)*			
	1 (Little or no interest)	11%	18%	14%
	2	21%	7%	14%
	3	11%	25%	18%
	4	39%	18%	29%
	5 (Very interested)	18%	29%	23%
47	**Which particular types of internal MPI information would you find useful?** *(multiple+text)*			
	Function call time	50%	46%	48%
	Load balance	68%	57%	62%
	Memory use	54%	57%	55%
	Message queue information	57%	46%	52%
	Network counters	39%	43%	41%
	Other responses			
	Wait time information	4%	0%	2%
	Information for different message sizes	4%	0%	2%
	Time spent in different states	0%	4%	2%
	Tag matching times	0%	4%	2%
	MPI-IO two-phase aggregator locality	0%	4%	2%
	MPI-IO communication between processes and aggregators	0%	4%	2%

Table 9: Memory Hierarchy Details

No.	Question and Responses	AD	ST	Overall
48	**Do you expect to explicitly manage the memory hierarchy in your application?** *(single)*			
	Yes, I expect to explicitly allocate or migrate data in different memory regions	79%	68%	73%
	No, I expect to rely on system mechanisms to place and migrate data	21%	21%	21%
49	**Do you expect to exchange data between different memory regions on different nodes, using MPI?** For example, data in main memory on one node is sent to non-volatile memory on the other? Or directly to the memory of an accelerator device on the other node? *(single)*			
	Yes	50%	43%	46%
	No	14%	21%	18%
	Don't know	11%	14%	12%

targeting new interfaces to support tools as well as our own ECP project's plan to improve tool support within Open MPI.

From Q42, we first learn that only about half of all projects use performance tools at all, which is clearly a disappointing number and requires additional dissemination efforts — the use of performance tools will be critical if one wants to achieve exascale level performance. However, this is not the target of this survey or the task of the MPI implementation projects and requires additional efforts in other areas of ECP.

Of the 52% of the projects that do use performance tools, however, more than half (or 27% of all projects) have at least the occasional need to use *multiple* performance tools during a single application run. This is especially noteworthy, since this is a feature that the interfaces in the current MPI standard do not support and matches an ongoing discussion in the MPI Forum to extend and modernize the MPI profiling interface.

Q43 and Q44 ask about MPI-related improvements desired for correctness and performance tools, respectively. There were a number of requests for better scalability, which we interpret as more about the tools themselves than MPI *per se*. There were a number of other interesting suggestions, but overall, the low number of responses suggests that users are relatively satisfied with what MPI provides to support tools.

In Q45, we asked for uses of the MPI Profiling Interface (PMPI) other than traditional performance analysis tools. One AD project reported using it to obtain the application's communication matrix in order to map MPI ranks to resources with an awareness of the network topology. On the ST side, several other tools or capabilities utilizing the interface were mentioned. This question is also related to the question above on multiple tool uses (Q42), since the use of the PMPI interface for "internal" use within the application would require multi-tool capabilities to either combine these uses or to add performance monitoring on top of it.

Q46 and Q47 were focused on the recently introduced MPI tools information interface (`MPI_T`) and what types of information users would like to see available through it[4]. In Q47, we see that load balance, memory use and message queue information were of interest to more than half of the projects, with function call time and network counters were requested by slightly less than half of the projects. The additional free-form responses to this question were interesting in that there was no commonality between the AD and ST projects, although the "wait time information" and "time spent in different states" might overlap. Q46 shows that overall 52% of projects are interested or very interested in having access to internal MPI performance data.

4.5 Memory Hierarchy Details

Modern node architectures for extreme-scale systems contain a variety of memory technologies, including DRAM, high bandwidth memory (HBM) and non-volatile memory, and these memories have varying performance and scaling attributes. The amount of memory and how the memories are organized varies between systems and generations of systems. This memory architecture trend is expected to continue into the exascale era. To understand the needs of MPI applications such as how they expect to move data between these memories, and what they expect from the MPI standard and implementations, we asked two questions

[4]The `MPI_T` interface only provides the API to access MPI internal information; each MPI implementation decides what information is offered through the interface

Table 10: Accelerator Details

No.	Question and Responses	AD	ST	Overall
50	**Does your application currently run on GPU accelerators, or do you expect the "exascale" version to?** *(single)*			
	Yes	93%	68%	80%
	No	7%	25%	16%
51	**Do you want to be able to make MPI calls directly from within your GPU kernels (as opposed to only from the host CPU)?** *(single)*			
	Yes	43%	29%	36%
	No	11%	21%	16%
	Don't know	39%	18%	29%
52	**In productions runs, how do you expect to deploy your application?** *(single)*			
	One MPI rank per GPU, with CPU resources potentially being shared or divided among multiple MPI ranks	25%	11%	18%
	MPI ranks assigned to CPUs (or CPU cores), GPUs potentially shared or divided among multiple MPI ranks	36%	32%	34%
	Don't know	32%	25%	29%

about the extent to which this architectural characteristic might need to be addressed more explicitly in MPI, as shown in Table 9.

First, we asked if they expect to explicitly manage memory hierarchy in their software or expect the system mechanisms to manage the memory hierarchy for them. Overall, projects indicated strongly (73%) that they expect to explicitly manage memory placement and movement. This suggests that MPI users may desire some control over memory allocations done internally by MPI implementations, both in terms of placement and perhaps in limiting memory usage as part of higher level management of those resources. Additionally, MPI implementations should expect to be called with data residing in various locations within the memory system, and should sensibly handle operations that involve multiple memory areas, be it the locations of different data objects or the operational need to move data to different memory areas.

Then, we asked, if they expected to move data between local node and remote node memory using MPI. For example, moving data from main memory on one node to a remote node's non-volatile memory or accelerator memory. Most users (46%) responded that they do expect to be able to communicate data between different memory areas via MPI.

4.6 Accelerator Details

As GPUs became prevalent in the extreme-scale systems, MPI implementations have enabled several optimizations to aid the efficient movement of data between CPU and GPU memories. The objective of the survey questions in this section was to learn the extent of GPU usage by applications. Then, we were interested to understand how applications tend to use GPU resources and what MPI optimizations the users expect. We asked three questions, which are presented in Table 10.

An overwhelming 93% of AD projects and 68% of ST projects responded that their application either currently runs on GPU accelerators, or the exascale version is expected to. However, a significant minority of ST projects (25%) do not have plans to port to GPU accelerators.

For teams who were using the GPUs for their software, we asked more detailed questions on their usage and optimizations they expected from the MPI implementations. Q51 asks whether applications expect to make MPI calls from within GPU kernels. Among AD projects, 43% responded affirmatively, and 39% responded that they didn't know, with only 11% indicating that they did not expect to make MPI calls from GPU kernels. The ST projects were more conservative, with only 29% expressing their desire to call MPI from within the GPU, 18% aren't sure, and 21% do not plan to call MPI from the GPU. These responses have significant implications for the need to better integrate MPI into the GPU environment.

In GPU-based systems, the MPI operations might be performed on buffers that are either in the CPU or the GPU memory. Without any optimizations for data transfer operations, the contents of the buffer on the

GPU memory is required to be copied to the CPU memory. Also, the message preparation and triggering of data transfer is currently done only by a thread or process on the CPU, i.e., a CUDA thread cannot send the message. Currently, MPI implementations take advantage of hardware capabilities and support mainly two optimizations: GPUDirect and GPU-Async. The GPUDirect enables the MPI implementations to post the data into the GPU memory. The message has to be prepared by the CPU (process or thread on the CPU), while the data can reside in either the CPU or GPU memory [8]. The GPU-Async capability, which improves upon GPUDirect, relaxes the requirement that the CPU thread has to trigger the message transfer. With GPU-Async, the CPU prepares the message while CPU or GPU thread can trigger the message transfer [9]. Another optimization explored by researchers is a capability where the GPU thread prepares and triggers the data transfer [10, 11]. This capability removes the need for an application to switch from GPU execution (CUDA kernel) to CPU execution context for data exchange, potentially leading to huge performance improvements.

The last question of this section (Q52) was focused on the deployment of applications in a multi-GPU system. There is a trend in accelerator-based systems to incorporate more than one accelerator per node (as in the coming Summit and Sierra systems at ORNL and LLNL). For such systems, MPI jobs can be configured with one MPI process per GPU (sharing or partitioning the CPU resources on the node), or to tie the MPI processes to CPU resources (e.g., cores, NUMA domains or sockets) and share the GPUs among the MPI processes on the node. Both AD and ST projects indicated a preference for sharing the GPUs (35% AD, 32% ST), while a minority anticipated deploying one MPI process per GPU (25% AD, 11% ST). A sizable fraction of respondents, however, do not yet have clear plans in this area (32% AD, 25% ST).

4.7 Resilience

Resilience has been a long-standing concern for the HPC community overall, and the topic of discussions and proposals in the MPI community for a number of years now. In an effort to get some basic information about how the ECP community plans to deal with resilience, and some of the features of their applications that might be exploited in order to provide greater resilience, we asked three questions, which are shown in Table 11.

The first of these questions (Q53) asks directly how projects plan to deal with fault tolerance in their applications. We offered four pre-set answers, and allowed respondents to add their own. Only one project indicated that their application was already fault tolerant. The overwhelming majority of AD projects (61%) plan to use checkpoint/restart for resilience, though only half as many ST projects (32%) plan to use it. Many (18% AD, 25% ST) don't have clear plans for resilience at present, and a fair number (7% AD, 18% ST) don't plan to worry about fault tolerance at all. Of the user-provided responses, several indicated the use of checkpoint/restart in combination with other approaches (6% overall). Both the User-Level Fault Mitigation (ULFM) [12] and "ReInit" approaches under discussion in the MPI Forum were also mentioned as solutions. Besides that, a number of other approaches to resilience were mentioned. One response identified MPI itself as a vulnerability and indicated their intent to avoid using it as a strategy for resilience.

A common resilience strategy when running MPI applications is to allocate "spare" nodes to the job so that if one fails, a spare can be utilized to restart the job without having to return the job to the queue. However, this strategy may be undesirable to the extent that the spare nodes add to the cost of the job, but may sit idle for the duration of the job if no node failures occur. Q54 asks whether applications have flexibility to utilize different numbers of processes, either dynamically, during execution, or when restarting from a checkpoint. Just 16% of applications indicated that they can dynamically adapt the number of MPI processes they use, which might allow them to run through a node failure. 54% of AD projects, but only 25% of ST projects indicated the ability to restart from checkpoints on a different number of nodes. This would allow the job to shrink on failures rather than paying for spare nodes that might be mostly idle. And many (46% AD, 68% ST) indicated no flexibility in the number of processes, leaving the sparing strategy as their only option to allow an immediate restart without re-queuing.

Finally, we asked whether applications could continue past a data loss or corruption without an explicit restart (Q55). Overall, most projects indicated they could not (36% AD, 57% ST). Some could, but only in specific sections of the code (39% AD, 21% ST), while a smaller number could do this more broadly (25% AD, 18% ST).

Table 11: Resilience

No.	Question and Responses	AD	ST	Overall
53	**How do you plan to make your application fault tolerant?** *(single+text)*			
	It is already fault tolerant	4%	0%	2%
	I plan to use checkpoint/restart	61%	32%	46%
	Don't know	18%	25%	21%
	I'm not going to worry about fault tolerance	7%	18%	12%
	Other responses			
	Avoid use of MPI	0%	7%	4%
	Data checksums between memory and storage	0%	4%	2%
	Legion capabilities in addition to checkpoint/restart	4%	0%	2%
	Local-failure/local-recovery	0%	4%	2%
	MPI Reinit	4%	0%	2%
	MPI ULFM	4%	0%	2%
	MPI fault tolerance feautres in addition to checkpoint/restart	4%	0%	2%
	Selective reliability	0%	4%	2%
	Skeptical programming	0%	4%	2%
	Task-based capabilities in addition to checkpoint/restart	4%	0%	2%
	Task-based rollback/recovery, replication	0%	4%	2%
	Treat as proper distributed system, with group membership	0%	4%	2%
54	**Is your application "malleable" with respect to the number of processes (assuming MPI can "run through" faults, as needed)?** *(multiple)*			
	Yes, it can change the number of processes dynamically, during execution	18%	14%	16%
	Yes, it can change the number of processes when restarting from a checkpoint made on a different number of processors	54%	25%	39%
	No	46%	68%	57%
55	**Can your application be organized to continue past (limited) data loss or corruption without an explicit restart?** *(single)*			
	Yes	25%	18%	21%
	Only limited sections	39%	21%	30%
	No	36%	57%	46%

4.8 Use of Other Programming Models

In order to better understand how MPI is being combined with other programming models, we asked two questions, which are shown in Table 12.

Q56 asks the core question, seeking information on both node-level and global programming models alongside MPI, with both being "active" at the same time. This was intended to remove from consideration cases such as coupled multiphysics applications in which components using different programming models run in succession. The pre-set responses represent the various programming models that are the subject of various ECP ST projects. We also accepted additional free-text responses. Among the AD projects, OpenMP is the most prominent response (57%), with Kokkos or RAJA (25%), and UPC++ (21%) as the next two. After that, comes CUDA or CUDA Fortran (14%), followed by Pthreads, Global Arrays, and OpenACC, each with 11% of responses. Among the ST projects, Pthreads (36%) beats out OpenMP (32%) for the top spot, followed by Kokkos or RAJA (18%) and Legion (11%). Note that Kokkos and RAJA can utilize a variety of the other programming models on the backend, and some responses may have listed both Kokkos/RAJA *and* the backends they typically utilize.

We also asked projects an open-ended question about what their experience has been to date with mixing programming models (Q57). From the AD projects, we received very few responses, generally indicative of success. There were more responses from the ST projects, and more indications of problems (4 mentions), particularly around resource sharing (3). Five responses indicated success with MPI+OpenMP (2) and other combinations. However of those responses also noted that they rarely saw MPI+threads outperforming MPI-only runs.

4.9 MPI with Threads

MPI has many different modes of threading support, concentrating on thread serialization techniques with MPI. Full multi-threaded support in MPI implementations lags single-threaded modes in terms of performance. Improvements are currently being proposed or are in discussion for MPI inclusion that aim to address performance and usability of multiple threads with MPI libraries. The seven questions shown in Table 13 are designed to determine the current state of multi-threaded use of MPI and determine if applications needs are currently met by MPI multi-thread support. In addition to this, applications developers were asked about how they want to use threads with MPI, specifically if MPI needs to be called in parallel regions (e.g., calling MPI inside an OpenMP loop). This is important to understand as current best practices use parallel loops followed by serialized access to MPI. In some cases algorithms may more easily use MPI if they can call MPI from individual threads. These questions are meant to probe the usability of the current interface. The Endpoints proposal [13] being considered by the MPI Forum allows for per thread addressability on the target and allows each individual thread to have its own rank. This significantly expands the MPI process space of a given application, and therefore questions were included to determine if this functionality is of great use to applications. If this is not required other alternatives such as Finepoints [14] could be useful to applications as it allows highly concurrent threading without targeting thread addressability (only processes are addressable).

The overwhelming majority of both AD (79%) and ST (93%) projects either currently use, or plan to use, multiple threads within an MPI process (Q58). When asked which of the MPI threading options they're currently using (Q59), results for all three modes were represented at similar levels (18% each), with the exception of `MPI_THREAD_MULTIPLE`, which is used by 32% of ST projects. 25% of projects report that they don't know which threading mode they're using, which makes it likely they're using a regular `MPI_Init` call rather than an `MPI_Init_thread`, meaning that they're using a non-multi-threaded mode. It is worth observing that only 79% of AD projects answered this question at all, so there a number of projects for which we do not have information.

In response to the follow-up question about whether they would prefer to be using a different threading mode than they're currently using (Q60), the most common preference was `MPI_THREAD_MULTIPLE` (9 mentions each among AD and ST projects), with performance cited as the most common reason for not using it (5 AD, 0 ST). In Q61, we asked whether applications would benefit from being able to change threading modes in different sections of an application. Interestingly, among the AD projects, more would like to be able to change modes (39% vs. 29%), while among ST projects most would not benefit from changing threading modes (46% vs 25%).

Table 12: Other Programming Models

No.	Question and Responses	AD	ST	Overall
56	**Do you use any other programming models (node-level or global) along side MPI (i.e. both are active at the same time), currently or in your "exascale" version?** *(multiple+text)*			
	None	7%	7%	7%
	OpenMP	57%	32%	45%
	Kokkos or RAJA	25%	18%	21%
	Pthreads	11%	36%	23%
	Global Arrays	11%	4%	7%
	Legion	4%	11%	5%
	UPC++	21%	4%	11%
	PaRSEC	7%	4%	4%
	Other responses			
	Agency	4%	0%	2%
	Argobots	0%	4%	2%
	CUDA or CUDA Forrtan	14%	7%	11%
	Charm++	0%	4%	2%
	DARMA	0%	4%	4%
	Mercury	0%	7%	4%
	OCCA	0%	4%	2%
	OpenACC	11%	4%	7%
	Swift	4%	0%	2%
	TBB	0%	4%	2%
	Thrust	7%	0%	4%
57	**If you already have tried to combine MPI with task-based programming models, please comment on your experience to date.** *(text)*			
	AD: Currently using block-synchronous hand-off between Legion and MPI. Thread safety would be most helpful MPI improvememnt (1 mention); Good success so far, would lke to be able to run efficiently with `THREAD_MULTIPLE` (1); Ok (1); Swift supports multiple MPI task configurations (1);			
	ST: Experimenting with node-level runtimes (OpenMP, Kokkos, C++17, etc.) (1 mention); Huge problem, particularly resource management across programming models, including CPUs, NUMA domains, GPUs, etc. (1); Legion+MPI working (1); Many gotchas and pitfalls. More interoperbaility/integration is needed. Better methods for joint resource management are needed (1); No trouble combining MPI outside of parallel regions with OpenMP tasks (1); No trouble combining OpenMP tasks with MPI (1); Performance is difficult to achieve. Hard to get MPI progress while asynchronous tasks are executing (1); QUARK+MPI (MPI seems to do a better job of overlapping/pipelining) (1); Very messy. Have to stop one programming model worlrd to switch to the other. Slow, error-prone, difficult to code (1); Works reasonably well for strong scaling if there is a sufficiently large amount of data. However MPI+threads rarely out-performs MPI-only (1);			

Table 13: MPI with Threads

No.	Question and Responses	AD	ST	Overall
58	**Do you currently use or plan to use multiple threads within an MPI process?** *(single)*			
	Yes	79%	93%	86%
	No	21%	7%	14%
59	**Which MPI threading option are you using?** *(single)*			
	`MPI_THREAD_MULTIPLE`	18%	32%	25%
	`MPI_THREAD_FUNNELED`	18%	18%	18%
	`MPI_THREAD_SERIALIZED`	18%	18%	18%
	I don't know	25%	25%	25%
60	**Would you prefer to be using a different MPI threading option? If so, which one, and what forced your current choice?** *(text)*			
	AD: Depends on adopted programming model (1 mention); No (1); Not threading now. Would prefer Multiple due to small messages, but not always performant (1)l THREAD MULTIPLE would be great, but we can deal with a more limited model if it provides performance and interoperability (1); We are exploring different options (1); Would prefer `MPI_THREAD_MULTIPLE` (3); Would prefer `MPI_THREAD_MULTIPLE`. But not using it because of poor performance. (4);			
	ST: Currently funneling, would prefer thread multiple (2 mentions); Not currently threading in MPI, using OpeMP, OpenACC, and CUDA (1); Not mixing now. Would prefer Multiple for greatest flexibility (1); Sometimes run thread multiple because required by a dependency (1); We don't care too much, but want something that gives performance (1); Would like to be able to do MPI-IO `MPI_File_Write_At` in `MPI_THREAD_MULTIPLE` (1); Would prefer `MPI_THREAD_MULTIPLE` (2); Would prefer `MPI_THREAD_MULTIPLE` if it became standard on all supercomputers (1); Would prefer that `MPI_THREAD_MULTIPLE` worked with non-blocking collectives (2);			
61	**Would your application benefit from using different MPI threading options in different sections of the code?** *(single)*			
	Yes	39%	25%	32%
	No	29%	46%	38%
62	**Is it important for you to be able to make MPI calls from within multi-threaded regions of your application? If so, which types of communications are performed?** *(multiple)*			
	No communication in multi-threaded regions	25%	11%	18%
	Point-to-point	46%	43%	45%
	RMA (one-sided)	25%	32%	29%
	Collectives	11%	29%	20%
63	**Do you require thread addressability on the target side (delivery of data to specific threads) for certain kinds of operations?** *(multiple)*			
	No communication in multi-threaded regions	54%	50%	52%
	Point-to-point	18%	11%	14%
	RMA (one-sided)	14%	11%	12%
	Collectives	4%	11%	7%
64	**Do you need high-level control over the placement of threads and MPI processes?** *(multiple)*			
	Yes, using mpirun (or equivalent) on the command line	61%	43%	52%
	Yes, via the job manager	29%	18%	23%
	No	0%	32%	16%

Our goal of determining if Endpoints and/or Finepoints solutions could be viable exascale MPI concurrency interfaces led to several questions in the survey as well. In Q62, we asked respondents if it was important to be able to make MPI calls from multi-threaded regions of code. This would indicate an interest in Finepoints/Endpoints types of interfaces. Overall, 45% of respondents said that they would make point-to-point MPI calls from within multi-threaded regions, 29% would make RMA calls, and 20% would call collectives. 18% of projects did not consider it important to be able to make MPI calls from within multi-threaded regions. There were fairly significant differences between the AD and ST projects in their desire to use collectives within multi-threaded regions (11% vs 29%), and in the fraction of projects not interested in MPI in multi-threaded regions (25% vs 11%). A majority (52%) of projects indicated that they did not need thread-level addressability on the target side of communications (Q63), indicating that the Finepoints proposal might be a suitable solution for them, while much smaller fractions indicated the need for thread-level addressability in point-to-point, RMA and collective operations (14%, 12%, and 7%). Finepoints and Endpoints are complimentary solutions, therefore these results motivate pursuing both for exascale.

Finally, a clear majority indicated the need for high-level control over placement of threads and MPI processes via either the command line (`mpirun` or equivalent) (52%) or the job manager (23%).

5. CONCLUSIONS

This paper presents a summary of a survey conducted within the ECP community to gain information about how the MPI standard is currently used and how the various ECP projects are planning on using it to achieve exascale. Additionally, the survey captured requirements stated by the ECP projects as well as potential improvements needed in MPI implementations.

The first conclusion of the survey is the confirmation that MPI remains a critical building block in the exascale ecosystem, for both application (AD) and software stack (ST) projects. However, the survey also highlights that most ECP projects do not interact with MPI directly, but instead use it through libraries or an abstraction layer. Our survey further confirms that the capabilities of interest are, in most cases, covered by point-to-point and collective communications, even if one-sided communication (RMA) is gaining interest in the context of exascale.

Many of the results of our survey are of particular interest to specific audiences.

MPI Forum. C++ has gained a lot of traction in the HPC community and became the most common language from which to invoke MPI among the ECP projects surveyed. This is not necessarily a plea to reenact the MPI C++ bindings removed in MPI 3.0. but it clearly indicates a shift in the programming languages used, at least in the ECP community, which will need to be addressed by the MPI standardization body. Key follow-up questions that might help inform further consideration on this topic would include whether C++ programmers are simply using the C binding of MPI and consider that adequate, or whether they prefer an approach that is more consistent with the features and capabilities of C++ in comparison to C. For those using a C++ binding layer, it would be useful to know the origin and completeness of the binding, in order to better understand the level of effort users are putting into developing and maintaining C++ bindings outside of the standard.

Nearly half of projects reported that MPI covered all of their communication needs. There was significant interest in active messages and persistent communications (the latter may refer to persistent collectives, or it may be an indication that users are not familiar with the persistent point-to-point capabilities already available). Job-to-job communications were also requested.

Checkpoint/restart remains the most widely used resilience strategy among the projects surveyed, though a significant number of projects do not yet have clear plans for resilience. Some responses expressed interest in both the User-Level Fault Mitigation and ReInit approaches. Few projects can adapt the number of processes on which they run, either dynamically or on restart, so for most users, a strategy involving spare nodes is the only way to restart without re-queuing.

Current or planned use of threads (especially OpenMP, Pthreads, Kokkos or RAJA) together with MPI is nearly universal among the responses. Nearly half of responses indicated that they would like to be able to make MPI calls from within multi-threaded regions of code. However a majority of responses also indicated that they did *not* need thread-level addressability on the target side of communications. These results motivate continued pursuit of both of the complementary Finepoints and Endpoints approaches for exascale.

MPI Implementors. Developers of ECP applications seem ready to embrace some of the latest additions or improvements to the MPI Standard and expect to make significantly more use of RMA, neighbor collectives and process topologies in exascale versions of their codes. Exascale-ready implementations of these capabilities are therefore necessary in the near term.

The relevance of MPI+threads, the strong desire of many projects to use `MPI_THREAD_MULTIPLE`, and complaints about the current performance overheads suggest that improving the performance of MPI implementations for multi-threaded use cases would widely be beneficial and appreciated.

Tool Developers. The most significant message for tool developers is that half of the projects responding do not actually use any performance tools. This suggests that latest efforts by the MPI Forum to improve tools support have not been yet embraced by the user community and therefore additional effort on dissemination of and education about available tools may be warranted.

MPI Users. The majority of the projects responding to the survey have produced mini-application reflecting MPI-related behaviors of their applications. These are considered a useful way to interact with researchers outside the project, and a critical tool to assess the capabilities and performance of MPI implementations.

Crosscutting. The increasing use of accelerator-based node architectures (GPUs or other types) and the growth of complex memory architectures, which are not simply hierarchical, are both very significant in the responses. Users expect to explicitly manage memory placement and movement, including the ability to move data directly into different memory spaces while moving it between nodes. Similarly, there is significant interest in being able to make MPI calls directly from accelerator kernels (though similar numbers of projects don't know whether they will need this capability). These results certainly have implications for **MPI implementors** and **hardware vendors**, and possibly also for the **MPI Forum**.

However, it is also important to note there was a great deal of variation in the answers to many of the questions in our survey, which points to the richness and complexity of the exascale ecosystem, with many languages, diverse requirements and expectations for both the MPI specification and for implementations. Is that complexity a risk? How does the MPI standard need to evolve to address it and still guarantee performance and scalability? The same can be said about resilience where checkpoint/restart is still the preferred option: can checkpoint/restart scale to exascale? Can checkpoint/restart handle the many failures that some predict for exascale systems?

Effectively addressing the needs of applications as they transition to adapt to increasingly complex hardware capabilities is critical for the future of any programming paradigm, including MPI. While this task is certainly broader than the specific types of platforms and/or applications associated with the ECP, it provides an exciting opportunity to address some of the most extreme requirements in terms of scale and heterogeneity. In this context, providing support to application developers in their quest to become exascale-ready requires a detailed understanding of their needs and how these needs will evolve. At the same time, we must also be a partner for application teams to help them efficiently explore existing MPI constructs and, when necessary, provide them with missing capabilities critical for their success.

Nevertheless, we still face the same old dilemma on what should come first: should the application take the risk of trying experimental MPI constructs that are not yet standardized, or should the MPI standardization community provide constructs that might not be helpful to application developers for the foreseeable future. With this survey we tried to cover both fronts and to build a solid middle-ground between the MPI community and application developers. We hope that this forms a basis to begin a more concerted discussion on how MPI should evolve to retain its role of a portable and efficient environment for scientific applications at extreme scales and to continue to be the defining force in parallel computing.

ACKNOWLEDGEMENTS

This research was supported by the Exascale Computing Project (ECP 17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

This work was carried out in part at Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U.S. Department of Energy under contract number DE-AC05-00OR22725.

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 U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

This work was performed in part at Los Alamos National Laboratory, supported by the U.S. Department of Energy contract DE-FC02-06ER25750.

Part of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

REFERENCES

[1] Exascale computing project. `https://exascaleproject.org/`, September 2017.

[2] Message Passing Interface Forum. MPI: A message-passing interface standard version 3.1. available at `http://mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf`, June 2015.

[3] Open MPI: Open source high performance computing. `https://www.open-mpi.org/`, June 2017.

[4] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, September 2004.

[5] MPICH high-performance portable MPI. `https://www.mpich.org/`, September 2017.

[6] Ken Raffenetti, Abdelhalim Amer, Lena Oden, Charles Archer, Wesley Bland, Hajime Fujita, Yanfei Guo, Tomislav Janjusic, Dmitry Durnov, Michael Blocksome, Min Si, Sangmin Seo, Akhil Langer, Gengbin Zheng, Masamichi Takagi, Paul Coffman, Jithin Jose, Sayantan Sur, Alexander Sannikov, Sergey Oblomov, Michael Chuvelev, Masayuki Hatanaka, Xin Zhao, Paul Fischer, Thilina Rathnayake, Matt Otten, Misun Min, and Pavan Balaji. Why is mpi so slow? analyzing the fundamental limits in implementing mpi-3.1. *SC'17: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Denver, Colorado, USA*, 2017. to appear.

[7] Daniel Holmes, Kathryn Mohror, Ryan E Grant, Anthony Skjellum, Martin Schulz, Wesley Bland, and Jeffrey M Squyres. Mpi sessions: Leveraging runtime infrastructure to increase scalability of applications at exascale. *Proceedings of the 23rd European MPI Users' Group Meeting*, pages 121–129, 2016.

[8] NVIDIA. Gpudirect. https://developer.nvidia.com/gpudirect, 2015.

[9] Davide Rossetti. Gpudirect: Integrating the gpu with a network interface. http://on-demand.gputechconf.com/gtc/2015/presentation/S5412-Davide-Rossetti.pdf.

[10] Sreeram Potluri, Davide Rossetti, Donald Becker, Duncan Poole, Manjunath Gorentla Venkata, Oscar Hernandez, Pavel Shamis, M. Graham Lopez, Mathew Baker, and Wendy Poole. Exploring openshmem model to program gpu-based extreme-scale systems. *Revised Selected Papers of the Second Workshop on OpenSHMEM and Related Technologies. Experiences, Implementations, and Technologies - Volume 9397*, 2015.

[11] Sreeram Potluri, Anshuman Goswami, Davide Rossetti, Manjunath Gorentla Venkata, Neena Imam, and Chris J. Newburn. Gpu-centric communication on nvidia gpu clusters with infiniband: A case study with openshmem (to appear). December 2017.

[12] Wesley Bland, Aurelien Bouteiller, Thomas Herault, George Bosilca, and Jack Dongarra. Post-failure recovery of MPI communication capability: Design and rationale. *International Journal of High Performance Computing Applications*, 27(3):244–254, 2013.

[13] James Dinan, Ryan E Grant, Pavan Balaji, David Goodell, Douglas Miller, Marc Snir, and Rajeev Thakur. Enabling communication concurrency through flexible mpi endpoints. *The International Journal of High Performance Computing Applications*, 28(4):390–405, 2014.

[14] Ryan Grant, Anthony Skjellum, and Purushotham V Bangalore. Lightweight threading with mpi using persistent communications semantics. Technical report, Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), 2015.

A. COMPLETE SURVEY QUESTIONS

Tables 14 through 24 present the full set of questions from the survey (excluding the section that establishes the identity of the project and respondant).

The tables follow a common format. The main text of the question is shown in **bold face** font. An asterisk (*) following the question indicates that a response was required. In some cases, questions included additional context or explanation. After that, in parenthesis, is the type of response allowed. "Single" means that only a single answer could be chosen. "Multiple" indicates that any number of answers (including zero) could be marked. "Text" denotes text fields that allowed a free-form response. In some cases, multiple choice questions also had an "other" option, which allowed free-form text responses in addition to the choices provided. These are denoted as "single+text" or "multiple+text".

Table 14: Application Demographics

No.	Question and Responses
18	**Does your project involve a single application/library or several that you consider to be distinct?** * This refers to applications/libraries actively developed within your project, not to third-party software you may be using. *(single)* Single application/library Multiple distinct applications/libraries **Important Note:** *Hereafter, we'll use the term application to mean the aggregation of all applications and/or libraries being developed within your ECP project, but excluding third-party libraries that you depend upon but do not contribute to from your project.*
19	**Does your application currently use MPI?** * *(single)* Yes No
20	**Do you expect the exascale version of your application to use MPI?** * *(single)* Yes No
21	**If your anwer above was "no", why not?** *(text)*
22	**Do you have an abstraction layer that hides the MPI calls? Or do most of your developers write MPI calls directly?** * *(single)* Abstraction layer Direct MPI calls
23	**Do you have mini-applications that capture the MPI behavior of your application? If so, are they available to the community?** *(single)* Yes, and they are available to the community Yes, but they're limited availability No
24	**What programming languages do you call MPI from?** * *(multiple+text)* C C++ Fortran Python *Other*
25	**What additional languages would you like to be able to call MPI from?** *(text)*
26	**What third-party libraries that your application depends upon use MPI?** Please list library names, or respond "none". *(text)*

Table 15: Non-MPI Applications

No.	Question and Responses
27	**What parallel programming models are you using (node level and global)?** * *(text)*
28	**Are there issues with the MPI standard or implementations which have pushed you away from using MPI in your application?** *(text)*
29	**What third-party libraries that your application depends upon use MPI?** * Please list library names, or respond "none" or "unknown" if appropriate. *(text)*
30	**Do you expect the exascale version of your application to use MPI?** * *(single)* Yes No *Stop filling out this form.*

Table 16: Basic Performance Characterization

No.	Question and Responses
	We realize that applications often exhibit a mixture of characteristics. Please answer based on what is most important to the performance of your application.
31	**Is your application (when run in production) typically dominated by small or large messages?** *(single+text)* Small ($<$ 8 kB) Large ($\geq$ 8 kB) *Other*
32	**Do you consider your application (when run in production) to be constrained by** *(multiple+text)* Message latency Message bandwidth Message rate Don't know *Other*
33	**If there were one aspect of MPI that could be optimized to improve the performance of your application, what would you prioritize?** *(text)*
34	**Is the performance of your application particularly sensitive to the network topology and the specific mapping of MPI processes to that topology?** If so, please describe the platform(s)/network(s) where you have experienced issues. If not, please just answer "no" *(text)*

Table 17: MPI Usage Patterns – Part A

No.	Question and Responses
35	**What aspects of the MPI standard do you use in your application in its current form?** * *(multiple)*
	Point-to-point communications
	MPI derived datatypes
	Collective communications
	Neighbor collective communications
	Communicators and group management
	Process topologies
	RMA (one-sided communications)
	RMA shared windows
	MPI I/O (called directly)
	MPI I/O (called through a third-party library, such as HDF5 or pNetCDF)
	MPI profiling interface
	MPI tools interface
36	**What aspects of the MPI standard appear in performance-critical sections of your current application?** * *(multiple)*
	Point-to-point communications
	MPI derived datatypes
	Collective communications
	Neighbor collective communications
	Communicators and group management
	Process topologies
	RMA (one-sided communications)
	RMA shared windows
	MPI I/O (called directly)
	MPI I/O (called through a third-party library, such as HDF5 or pNetCDF)
	MPI profiling interface
	MPI tools interface
37	**What aspects of the MPI standard do you anticipate using in the "exascale" version of your application?** * *(multiple)*
	Point-to-point communications
	MPI derived datatypes
	Collective communications
	Neighbor collective communications
	Communicators and group management
	Process topologies
	RMA (one-sided communications)
	RMA shared windows
	MPI I/O (called directly)
	MPI I/O (called through a third-party library, such as HDF5 or pNetCDF)
	MPI profiling interface
	MPI tools interface

Table 18: MPI Usage Patterns – Part B

No.	Question and Responses
38	**What is the dominant communication in your application?** Check all that apply, recognizing that many applications have different communication patterns in different phases. *(multiple)*
	Each process talks to (almost) every other process
	Processes communicate in fixed "neighborhoods" of limited size
	Processes communicate in "neighborhoods" of limited size that may change in different phases of the application or evolve over the course of a run
	Communication is largely irregular
39	**Can your application take advantage of non-blocking point-to-point operations...** *(multiple)*
	To overlap communication with computation?
	To allow asynchronous progress?
	To allow event-based programming?
40	**Can your application take advantage of non-blocking collective operations...** *(multiple)*
	To overlap communication with computation?
	To allow asynchronous progress?
	To allow event-based programming?
41	**Is MPI providing all the communication semantics required by your application? If not, what is missing?** *(multiple+text)*
	MPI covers all my needs
	Active messages
	Persistent communications (communications with the same arguments, repeatedly executed, potentially allowing additional optimization if exposed to the MPI layer)
	Other

Table 19: MPI Tools Ecosystem

No.	Question and Responses
42	**If you are using performance tools with MPI, how often do you encounter the need to use multiple tools during the same application run?** *(single)* Often Occasionally Never Don't use performance tools with MPI
43	**Are there any MPI-related improvements you'd like to see for debuggers and other "correctness" tools?** *(text)*
44	**Are there any MPI-related improvements you'd like to see for performance tools?** *(text)*
45	**Are you using the MPI Profiling Interface for anything other than running performance analysis tools? If so, please briefly describe your use.** *(text)* **Note:** *MPI 3.0 introduced a new interface for tools (MPI_ T) that's intended to allow a view into information about the MPI implementation itself and related low-level information. The next two questions are designed to gauge interest in expanding those capabilities. The capabilities suggested below would potentially be accessible to applications, making calls directly on the MPI_ T interface, or to tools (e.g., performance tools) that could be extended to access the new information.*
46	**What is your level of interest in having access to internal MPI performance data? Information could include items like function call time, load balance information, memory use, message queue information, network counters, etc.** *(single)* 1 (Little or no interest) 2 3 4 5 (Very interested)
47	**Which particular types of internal MPI information would you find useful?** *(multiple+text)* Function call time Load balance Memory use Message queue information Network counters *Other*

Table 20: Memory Hierarchy Details

No.	Question and Responses
48	**Do you expect to explicitly manage the memory hierarchy in your application?** *(single)* Yes, I expect to explicitly allocate or migrate data in different memory regions No, I expect to rely on system mechanisms to place and migrate data *Skip to question 50.*
49	**Do you expect to exchange data between different memory regions on different nodes, using MPI?** For example, data in main memory on one node is sent to non-volatile memory on the other? Or directly to the memory of an accelerator device on the other node? *(single)* Yes No Don't know

Table 21: Accelerator Details

No.	Question and Responses
50	**Does your application currently run on GPU accelerators, or do you expect the "exascale" version to?** *(single)* Yes No *Skip to question 53.*
51	**Do you want to be able to make MPI calls directly from within your GPU kernels (as opposed to only from the host CPU)?** *(single)* Yes No Don't know
52	**In productions runs, how do you expect to deploy your application?** We anticipate that different accelerator-based architectures may offer different ratios of CPUs (or CPU cores) to GPUs. Assuming that the offload programming model continues to be used, do you expect your deployment model to be driven by the GPUs, or the CPUs (or CPU cores)? *(single)* One MPI rank per GPU, with CPU resources potentially being shared or divided among multiple MPI ranks MPI ranks assigned to CPUs (or CPU cores), GPUs potentially shared or divided among multiple MPI ranks Don't know

Table 22: Resilience

No.	Question and Responses
53	**How do you plan to make your application fault tolerant?** *(single+text)* It is already fault tolerant I plan to use checkpoint/restart Don't know I'm not going to worry about fault tolerance *Other*
54	**Is your application "malleable" with respect to the number of processes (assuming MPI can "run through" faults, as needed)?** *(multiple)* Yes, it can change the number of processes dynamically, during execution Yes, it can change the number of processes when restarting from a checkpoint made on a different number of processors No
55	**Can your application be organized to continue past (limited) data loss or corruption without an explicit restart?** For example, an iterative method that still converges, or a domain decomposition with fewer samples, etc. *(single)* Yes Only limited sections No

Table 23: Other Programming Models

No.	Question and Responses
56	**Do you use any other programming models (node-level or global) along side MPI (i.e. both are active at the same time), currently or in your "exascale" version?** *(multiple+text)* None OpenMP Kokkos or RAJA Pthreads Global Arrays Legion UPC++ PaRSEC *Other*
57	**If you already have tried to combine MPI with task-based programming models, please comment on your experience to date.** What works and doesn't work? What needs to change on the MPI side to make it better? *(text)*

Table 24: MPI with Threads

No.	Question and Responses
58	**Do you currently use or plan to use multiple threads within an MPI process?** For example, using OpenMP, Pthreads, Qthreads, Cilk, Argobots, or other threaded programming models *(single)* Yes No *(Stop filling out this form.)*

59	**Which MPI threading option are you using?** *(single)*
	`MPI_THREAD_MULTIPLE`
	`MPI_THREAD_FUNNELED`
	`MPI_THREAD_SERIALIZED`
	I don't know

60	**Would you prefer to be using a different MPI threading option? If so, which one, and what forced your current choice?** *(text)*
61	**Would your application benefit from using different MPI threading options in different sections of the code?** *(single)* Yes No
62	**Is it important for you to be able to make MPI calls from within multi-threaded regions of your application? If so, which types of communications are performed?** *(multiple)* No communication in multi-threaded regions Point-to-point RMA (one-sided) Collectives
63	**Do you require thread addressability on the target side (delivery of data to specific threads) for certain kinds of operations?** *(multiple)* No communication in multi-threaded regions Point-to-point RMA (one-sided) Collectives
64	**Do you need high-level control over the placement of threads and MPI processes?** *(multiple)* Yes, using mpirun (or equivalent) on the command line Yes, via the job manager No

B. RAW FREE TEXT RESPONSES

The following tables present the raw responses to those questions that allowed free-response inputs. The tabular structure corresponds to those used earlier in the document, but *only* those question with text responses are included. Individual responses are separated with semicolons (;). In some cases, small changes have been made to the case and punctuation of the responses, and they have been reordered to group similar responses together.

The sections on Memory Hierarchy Details and Accelerator Details have no free-response questions.

Table 25: Application Demographics

No.	Question and Responses
21	**If your anwer above was "no", why not?** *(text)* AD: We are supposed to use Legion for parallelization, with MPI as backup if Legion turns out to be inappropriate. ST: MPI supports message passing, I am more interested in RPCs; MPI supports message passing, I am more interested in RPCs; This is one option;
24	**What programming languages do you call MPI from?** * *(multiple+text)* AD: petsc interface; Tcl ST: *none*
25	**What additional languages would you like to be able to call MPI from?** *(text)* AD: C/C++; Julia; none at present; Perhaps directly from python and julia; none; none; ST: We supply interfaces for users in C, C++ and Fortran. While we do perform MPI calls for our portion of the solves, we expect that users also call MPI for problem-specific communication. Therefore we would like that all of C, C++ and Fortran be able to call MPI directly; none, I prefer not to use MPI; none, I prefer not to use MPI; none;
26	**What third-party libraries that your application depends upon use MPI?** Please list library names, or respond "none". *(text)* AD: ADLB; AMReX; Basic graph libraries; CNTK; Global Arrays; GridPACK, PETSc; HDF5; Hypre, Metis, MFEM; MFEM relies on hypre, Nek relies only on MPI; Petsc; TRILINOS, PARAVIEW/CATALYST, HDF5, HYPRE; Trilinos, HDF5, ADIOS, Silo; Trilinos, SuperLU-Dist, HDF5, NetCDF, pNetCDF, CGNS; mpi4py; none; none; none; none; none; none; none; none; none; none; parallel HDF5; petsc interface to other libraries; pio, pnetcdf, hdf5; trilinos; ST: BLAS; DIY; Hdf5, ADIOS, DIY; I am not sure, but anything Trillinos is linked with, e.g., ScaLAPACK; Mercury currently has a MPI implementation; Mercury currently has a MPI implementation; NetCDF, pNetCDF, HDF5, CGNS, MKL-Cluster; PnetCDF, ROMIO; ScaLAPACK (BLACS, PBLAS), ParMetis, PTScotch; Trilinos is itself a collection of libraries. We provide access to other MPI-enabled libraries, specifically hypre, SuperLU_dist, Pardiso; Trilinos, PETSc, SLATE; Trilinos; We do not "depend" on these per-se, but we support interfaces to PETSc, HYPRE, SuperLU_MT, KLU and LAPACK. Of these, PETSc and HYPRE use MPI; We're OS and runtime project. We evaluate many MPI applications to evaluate how well our OS and runtimes work. These MPI apps often depend on frameworks such as Trilinos and Sierra and the many third party libraries they incorporate (HDF5, NetCDF, libmesh, DTK, ...); hypre; none (at present); none directly; none; none; none; none; none; none; none; none; none; none; zlib, Polly, TSan, clang;

Table 26: Non-MPI Applications

No.	Question and Responses
27	**What parallel programming models are you using (node level and global)?** * *(text)* AD: *none* ST: CUDA, TBB; Current prototype does use MPI. Production version will use Mercury/Margo; Currently none, tools run serial (databases, metadata capture, etc); Kokkos layered on OpenMP, Threads, CUDA, Qthreads; Kokkos, OpenMP, CUDA; Most of our work is thread-local and and then is run in parallel using any parallel programming model; None. The Tools & Dev Env does not have software like this; OpenMP, CUDA; The main project in HPCDE is Spack. It's written in Python. For Parallel builds we *may* use MPI but more likely we'll coordinate through the parallel filesystem and not require a particular MPI or resource manager; We are implementing a new PGAS Model: UPC++; We work on the operating system, and interface with the parallel programming model from the bottom; na; node level; not applicable; system software / middleware, not application;
28	**Are there issues with the MPI standard or implementations which have pushed you away from using MPI in your application?** *(text)* AD: *none* ST: The lack of standard or consistent ABIs is an ongoing problem for performance tools; We are not an application, so we don't need MPI; n/a; n/a; na; no, We provide node level kernels for MPI+X codes; no, but I currently have no need for MPI; no, this project is specifically focusing on node-level parallel; no. Spack has 139 packages that directly depend on MPI and 248 that can depend on it transitively; no; no; separation of concerns: intra-node vs. inter-node parallelism;
29	**What third-party libraries that your application depends upon use MPI?** * Please list library names, or respond "none" or "unknown" if appropriate. *(text)* AD: *none* ST: "abinit, adept-utils, adios, adlbx, alquimia, amrex, arpack-ng, automaded, bertini, boost, boxlib, caliper, callpath, cbench, cgm, charm, chombo, cntk, conduit, converge, cosmomc, cp2k, cram, dakota, darshan-runtime, dealii, dtcmp, elemental, elk, elpa, esmf, espresso, espressopp, everytrace, exodusii, extrae, fastmath, fenics, fftw, flann, foam-extend, funhpc, gasnet, globalarrays, gmsh, grackle, gromacs, h5hut, h5part, hdf5, hmmer, hoomd-blue, hpctoolkit, hpl, hpx5, hypre, icet, ior, isaac, julia, kripke, lammps, lbann, libcircle, libmesh, libnbc, libquo, libsplash, lulesh, lwgrp, lwm2, mdtest, meep, meme, mesquite, mfem, mitos, moab, mpe2, mpibash, mpifileutils, mpileaks, mpip, multiverso, mumps, muster, netgauge, netlib-scalapack, npb, nwchem, octopus,ompss, opencoarrays, openfoam-com, openfoam-org, osu-micro-benchmarks, p4est, pagmo, panda, paradiseo, parallel-netcdf, paraview, parmetis, parmgridgen, parpack, petsc, pfft, pflotran, phasta, pidx, plumed, portage, pruners-ninja, pumi, py-h5py, py-meep, py-mpi4py, py-pypar, r-rmpi, relion, rempi, samrai, scalasca, scorec-core, scorep, scotch, scr, simul, stat, sundials, superlu-dist, swiftsim, tau, trilinos, valgrind, vampirtrace, wannier90, xsdktrilinos, zoltan" MDHIM, but this project is also moving to Mercury/Margo; QEMU; We don't so much depend on libraries that use MPI, but we do intend to be used in larger software systems using our software in conjunction with MPI, and our work dovetails with that development. This software includes VTK, ParaView, VisIt, ALPINE, Catalyst, Libsim, and DIY. Where appropriate we look other software systems and simulation code.; n/a; na; none. We are node level kernel; none; none; none; none; none; none; none; unknown;

Table 27: Basic Performance Characterization – Part A

No.	Question and Responses
31	**Is your application (when run in production) typically dominated by small or large messages?** *(single+text)* AD: 10 kB to 10 MB depending on problem size; Depends on mini batch sizes; In the interesting strong scale limit, all messages are small; MIXED. AS A SUITE OF MULTI-PHYSICS CODES WE ARE EXPLORING; We have a wide spread of message sizes and latency requirements; all to all in subcommunicators; both depending on the problem run and whether BLAST or Miranda; both; both; both; large; unsure. We have both. Both can bottleneck scaling; ST: Both, depending; Depends on the application; Full-range from global collectives (8 bytes) to neighborhood collectives where we bulk up messages to neighbors, and multi-grid where message sizes vary from very small coarse grid neighborhoods to the full-scale fine grid; HDF5 mainly uses MPI-IO routines; some metadata aggregation calls to exchange data, which is usually $<$ 8 KB; Our portion entirely consists of small messages, but user codes will typically send larger messages to perform problem-dependent communication; depends on what we are doing (and we have not formally measured it). I would speculate that most of our messages are at least 8KB; depends; depends; don't know, but suspect messages are large; large; mix of both small (control) messages and large (data) messages; mixture;
32	**Do you consider your application (when run in production) to be constrained by** *(multiple+text)* AD: Different parts of the code have different requirements and can be constrained by any of these; Don't know, problem scaling on Cori past 2048 nodes; Load imbalance; Message rate, but this is very problem and setup dependent; Message rate, on node contention (e.g., in KNL) or lack of concurrency at strong scale; Somewhat dependent on problem size or part of the code used; Unconstrained, roughly, since our application is mostly perfectly-parallel until results are gathered from the cores; application dependent; application dependent; ST ADIOS is disk I/O constrained but otherwise message bandwith is the most constraining; Again, it depends on what part of the application we are talking about. The code I am most familiar with is the rendering, which uses MPI as basically a large, specialized reduction operation. When you scale very large, the reduction is dominated by a gather to get the pieces of array that have been distributed over the processes during the reduction operation; Any/all, depending on application behavior; Don't know, memory requirements of the application, load imbalance; I/O latency more than network latency; In the future, as we restructure for fine-grain tasking, we will start to be impacted by message rate (we think); Likely constrained by I/O and compute; Our portion is constrained by message latency; I do not know the constraints on our users' codes; depends on the application; depends on the parameters: if the number of detector pixels is large and there are lots of energy bins, MPI bandwidth will most likely be a limiting factor. However, for smaller images, or lower #bins, the triangle intersection computations dominate; depends; depends;

Table 28: Basic Performance Characterization – Part B

No.	Question and Responses
33	**If there were one aspect of MPI that could be optimized to improve the performance of your application, what would you prioritize?** *(text)* AD: ?; Asynchronous communication; Better fine grain support and dynamic tasking; Fault tolerance; INTEROPERABILITY WITH OTHER RUNTIMES, BOTH LOCAL THREADING, AND TASK-BASED DISTRIBUTED MEMORY SYSTEMS SUCH AS LEGION; Latency of small messages. We would like to strong scale out further; MPI+X abstraction; Memory barrier in on-node shared memory programming in MPI+MPI; Memory barrier in on-node shared memory programming in MPI+MPI; Multithreaded asynchronous; Optimize data block size; Predictability of message latency and bandwidth based on machine status and configuration information that can be interogated at runtime; Probably MPI one sided, async collectives; Shared memory MPI; Thread Multiple; You tell me; collective operations; collectives; hardware all-reduce, convex partitions (sorry, I realize these aren't really MPI issues); low latency; latency; latency; message latency; multi-thread support; "i) Core counts are growing and growing. Hybrid threading intranode and message passing internode is the most efficient scheme in principle, because internal memory copies are eliminated. However, all MPI implementations remain to date poor at accepting and concurrently processing concurrent requests from multiple threads. Thread concurrency is important, and needs to be worked on. ii) Although NVIDIA is moving toward a single address space with the host CPU they are not the only GPU or accelerator vendor in the world. Understanding how best to address single copy between âĂIJoffloadâĂİ memory spaces would be good. How to capture non-uniform things like the fast links between Nvidia GPUâĂŹs in the same box would be good. iii) In order to saturate the network bandwidth with nonblocking sends, it may be necessary for an MPI implementation to do internal threading and enlist more cores. Implementations should provide such measures to make saturation possible within just a single MPI rank. Presumably, the standard should specify an environment variable to control the number of internal threads."; ST: Ability to be used in service implementations: robustness, connect/attach, etc.; All_reduce, non-blocking All_reduce; CPU use makes MPI untenable for RPCs; CPU use makes MPI untenable for RPCs; Collective computation routines – specifically MPI_Allreduce. We may eventually transition to asynchronous reduction operations, but for now this is the main MPI-related function we need (for inner-product calculations); Controlling rendezvous threshold; Fast local collectives; I don't know; In my experience, collective operations are the most problematic at large scales; MPI-IO aggregation optimizations based on topology; Relaxing the limit of I/O for larger than 2 GB blocks; MPI_THREAD_MULTIPLE or a suitable replacement for supporting MPI usage by multiple threads/tasks; Message latency; Non-blocking all-reduce; Performance in the presence of tasks and threads within MPI ranks; Threading support; Very fast collectives; all-to-all; bandwidth for repartitioning of data; dealing with a large number of outstanding requests; no opinion; start-up time at extreme scale;

Table 29: Basic Performance Characterization – Part C

No.	Question and Responses
34	**Is the performance of your application particularly sensitive to the network topology and the specific mapping of MPI processes to that topology?** If so, please describe the platform(s)/network(s) where you have experienced issues. If not, please just answer "no" *(text)* AD: ?; It is little sensitive when we distributed data in 2D checkerboard fashion. We observed it on NERSC/Edison; Miranda can be sensitive when using spectral methods, but these are not common in production anymore. Otherwise messages tend to be smaller and not a large percentage of runtime; Our applications work well on both a 4D torus network and an all-to-all network. With the former, of course, it is crucial to get the MPI mapping right; We do not know, because measurement variability is too high to be of use; We see large variability on Aries; network aware topology mapping (by SciDAC SUPER Institute) may sometimes give 15% improvement on Titan. However, since Titan batch scheduler may not give contiguous partitions, the mapping may not always give improvement; no; no; no; no; no; no; no; no; no; no; no; no; not much, except as far as topology affects collective behavior; not sure; we have measured this on current LCF production platforms; yes – If other applications are interfering with the network and introducing contention, that's a bad thing. So, A#1 priority is convex partitions; yes but we have not studied this in detail; yes, due to latency; yes. The network topology is highly heterogeneous; yes; ST: I do not know, but am very interested of knowing; don't know; it may. At the core is matrix-vector multiplication, which may have poor performance for sparse cluster allocators; no; no; no; no; no; no; no; no; no; not in recent years. This used to be an issue; not the SUNDIALS-specific portion – we expect that the user has already mapped the parallel decomposition to the machine, so if they have specific issues/needs, we rely on them to work those out; not to my knowledge; somewhat sensitive to job allocation (non-contiguous allocation on Cray machines); we have observed that topology-aware I/O gains significant performance, especially for a large number of small (size) I/O requests; yes, in some cases; yes. Halo exchanges on Cray systems can usually be optimized with careful MPI process placement. For example, S3D on Cray runs best with an optimized MPICH_RANK_REORDER mapping file. The first order benefit is minimizing the surface area communicated off each node. Second order benefit is mapping better to the physical network topology; yes. Think I/O forwarding; yes; yes; yes; yes;

Table 30: MPI Usage Patterns

No.	Question and Responses
41	**Is MPI providing all the communication semantics required by your application? If not, what is missing?** *(multiple+text)* AD: GPU to GPU; Mixed with gasnet. Reductions with variable sized abstract types; That does not mean that future improvements will not improve application performance in particular in the strong scaling limit new features and more overlap would be useful; WE ARE ACTIVELY EXPLORING LEGION AS AN EXEMPLAR OF A MORE DATA-CENTRIC TASK-BASED PROGRAMMING SYSTEM; details of the implementation, such as the use of Active Messages, should be hidden behind the API; ST: As workflows become more complex, we would like a mechanism to communicate between MPI jobs with an uber-communicator; Easy job-to-job communication (eg workflows w/ jobs that come and go); Improved MPI_THREAD_MULTIPLE or improved semantics for MPI+threads usage; Missing RPC between independent processes; Not sure yet...;

Table 31: MPI Tools Ecosystem

No.	Question and Responses
43	**Are there any MPI-related improvements you'd like to see for debuggers and other "correctness" tools?** *(text)* AD: Better Scalability; Handlers that can call out to monitoring tools outside batch system. "Mpi" core file for offline replay within debug tools; No; OUR PROJECT IS STILL IN START UP MODE. I ANTICIPATE DEBUGGING CODES WITH MIXED PROGRAMMING MODELS (MPI, OPENMP, LEGION) WILL BE CHALLENGING FOR TOOLS; Race-condition analysis; The tools we've tried tended to give access to either too little or too much detail to be useful; better resolution for tracking message sizes; mipP give information about collective and reduction operations; ST: Better scalability; no opinion; no;
44	**Are there any MPI-related improvements you'd like to see for performance tools?** *(text)* AD: Better Scalability; It would be useful to break out time in the MPI library from the time on the wire to understand where the performance is being lost; Mpi reduction for user profiling tools; No; SCALABILITY; The tools we've tried tended to give access to either too little or too much detail to be useful; upgrade to mpiP; ST: Better scalability; I liked MPE+ jumpshot, but it doesn't work anymore. Other tools are too complicated to use; ability to analyze time messages spend in each state; autotuning in MPI-T is planned; no;
45	**Are you using the MPI Profiling Interface for anything other than running performance analysis tools? If so, please briefly describe your use.** *(text)* AD: extract communication matrix (process I send total of X messages, Y bytes to process J) for use in topology aware mapping; no; no; no; no; ST: Capturing MPI traces to feed into external simulation tools; Darshan; ReMPI (MPI record-and-replay) and NINJA (Noise Injection agent tool) use PMPI for debugging; no; no; no;
47	**Which particular types of internal MPI information would you find useful?** *(multiple+text)* Function call time Load balance Memory use Message queue information Network counters *Other* AD: If different algorithms are used depending on message size; More detailed access to wait time information; Note its not clear how useful the queue information or network counters would be. They might be useful, but for now we consider this a research topic that needs to be investigated with the hardware vendors and they are not always exposing useful information from an application viewpoint; ST: MPI-IO two-phase aggregator locality; communications between processes and the aggregators; Match list walk times; time spent in states;

Table 32: Resilience

No.	Question and Responses
53	**How do you plan to make your application fault tolerant?** *(single+text)* AD: CHECKPOINT/RESTART PLUS RESEARCH INTO LEGION PROGRAMMING SYSTEM PERSISTENCE AND RESILIENCE; In addition to checkpoint/restart, we will be incorporating task based management and hope to have some handling at that level; Some functionalities use checkpoint. Others will follow; Provided the runtime does not kill us and basic MPI reinit is possible to adjust communicators etc. we can be fully fault tolerant; We are hoping that MPI FT features will emerge that we can use. We currently use checkpoint/restart; We are looking at ULFM; ST: I plan to not use MPI; I plan to not use MPI; Looking into calculating checksums to provide end-to-end integrity of data between memory and storage; SZ will run inside an application that may have its own fault tolerance; Task-based rollback/recovery and task-based replication; This is a post-processing library, which we hope evolves into an in-situ capability, but how we will deal with fault tolerence is unclear at this time; Treat as proper distributed systems with group membership, etc.; We are exploring several fault tolerance approaches: local-failure, local-recovery; "skeptical" programming, selective reliability; checkpoint/restart for now, but we would like more tools...; not yet clear what fault tolerance mechanism(s) will be needed;

Table 33: Other Programming Models

No.	Question and Responses
56	**Do you use any other programming models (node-level or global) along side MPI (i.e. both are active at the same time), currently or in your "exascale" version?** *(multiple+text)* AD: CUDA (Thrust); CUDA Fortran and OpenACC; CUDA; CUDA; OpenACC (currently); OpenACC; Swift; THRUST/AGENCY; none; none; paRSEC is in progress/planned; ST: Argobots; CUDA, OpenACC; DARMA/Charm++; DARMA; May be Pthread at some point; Mercury Suite; Mercury Suite; OCCA; TBB, CUDA; none; none;
57	**If you already have tried to combine MPI with task-based programming models, please comment on your experience to date.** What works and doesn't work? What needs to change on the MPI side to make it better? *(text)* AD: CURRENTLY WE ARE USING BLOCK SYNCHRONOUS HANDOFF BETWEEN LEGION AND MPI. THREAD SAFETY SEEMS TO BE THE MPI FEATURE THAT WOULD HELP US MOST; No; OK; We had good success so far. Would be nice to be able to run efficiently with THREAD_MULTIPLE to parallelize event handling; Yes, Swift supports multiple MPI task configurations; in plan, but not yet executed; ST: no trouble using OpenMP tasks with MPI; Very messy. More like stop one programming model world and switch to another than back again. Slow. Error prone. Difficult to code; We use OpenMP tasks, but MPI calls are done outside the OpenMP parallel region. So no issues there; QUARK+MPI (MPI seems to do better job overlapping/pipelining); We have Legion/MPI interoperability working. We are experimenting with various node-level runtimes (OpenMP, Kokkos, C++17, etc.); Performance is difficult to achieve. It is hard to get MPI to make progress while asynchronous tasks are executing; So far this has worked reasonably well for strong-scaling as the number of threads increases, but only if there is a sufficiently large amount of data (most of our data access is of BLAS level 1 type). That said, we rarely notice any instance where MPI+Threads outperforms MPI-only parallelism; This is a HUGE problem for us. Right now there is no mechanism to manage what CPU/GPU resources our application should use with a given launch of an MPI job. Setting up the OpenMP configuration, for example, to execute each MPI processes in its own NUMA domain is a huge heartache. And then when that gets combined with a library using a different programming model like TBB, which happens frequently for us, things get even worse. Likewise, if we are using a system with multiple GPUs on a node, it is very difficult to manage which GPU each MPI process should use. What our applications desperately need is a library (either built into MPI or separate) that allow users to launch MPI jobs with simpler high level options (e.g. 1 MPI process per GPU or 4 MPI processes per CPU processor) and then either automatically set up resources or at least provide a query mechanism to get the appropriate parameters for each programming model (e.g. set up CUDA to use the GPU assigned to my rank or set up OpenMP to run threads in my quadrant of a KNL processor); There are many gotchas and pitfalls that need to be avoided. A more integrated solution for interoperability is desired. Better methods for avoiding resource oversubscription and undersubscription are needed;

Table 34: MPI with Threads

No.	Question and Responses
60	**Would you prefer to be using a different MPI threading option? If so, which one, and what forced your current choice?** *(text)* AD: MPI_THREAD_MULTIPLE would be better especially for overlapping computation and communication; Mpi threaded is a better fit but performance is not robust; THREAD MULTIPLE WOULD BE GREAT, BUT WE CAN DEAL WITH A MORE LIMITED MODEL IF IT PROVIDES PERFORMANCE AND INTEROPERABILITY; Want to be using MPI_THREAD_MULTIPLE but it can be slow on some machines (Cray and Intel have good implementations); We are exploring different options; Would prefer MPI_THREAD_MULTIPLE. But not using it because of poor performance; Would prefer MPI_THREAD_MULTIPLE. But not using it because of poor performance; Would prefer MPI_THREAD_MULTIPLE; we would be interested in trying mpi_thread_multiple; Right now threading is not in the code, but is being added. The best method will be picked based on performance. With small message Multiple would be the preference, but its not always performant; Depends on adopted programming model; No; ST: I would simply prefer that MPI_THREAD_MULTIPLE worked with non-blocking collectives; I would simply prefer that MPI_THREAD_MULTIPLE worked with non-blocking collectives; MPI_THREAD_MULTIPLE or a suitable replacement; MPI_THREAD_MULTIPLE would be better. We currently use independent parallel POSIX I/O. We expect to rework this to use MPI I/O with MPI_File_write_at. An individual thread processing a profile would call this; MPI_THREAD_MULTIPLE would be preferred. But only if it becomes standard on all supercomputers; MPI_THREAD_MULTIPLE; MPI_THREAD_MULTIPLE would be interesting to look at. Current version of the code funnels all communications to a dedicated thread; Thread multiple was preferred but forced by performance to funnel; We sometimes run with MPI_THREAD_MULTIPLE because it is required by PTScotch, when configured to use pthreads; Presently, the SUNDIALS-specific portion of applications does not "mix" MPI and OpenMP in any complex manner. However, we'd prefer that the MPI library support MPI_THREAD_MULTIPLE, since we want to support the widest variety of applications as possible; We don't care too much, but want something that gives performance; not currently using MPI for threading. Using OpenMP, OpenACC and CUDA;