An International Survey on MPI Users

By Atsushi Hori

An International Survey on MPI Users

Atsushi Hori[a], Emmanuel Jeannot[b], George Bosilca[c], Takahiro Ogura[a], Balazs Gerofi[a], Jie Yin[a], Yutaka Ishikawa[a]

[a]Riken Center for Computational Science, 7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, 650-0047, Japan
[b]INRIA Bordeaux Sud-Ouest, 200, Avenue de la Vielle Tour, Talence, 33405, France
[c]Innovative Computing Laboratory, University of Tennessee, Suite 203 Claxton, 1122 Volunteer Blvd, , Knoxville, 37996, USA

Abstract

The Message Passing Interface (MPI) plays a crucial part in the parallel computing ecosystem, a driving force behind many of the high-performance computing (HPC) successes. To maintain its relevance to the user community—and in particular to the growing HPC community at large—the MPI standard needs to identify and understand the MPI users' concerns and expectations, and adapts accordingly to continue to efficiently bridge the gap between users and hardware. This questionnaire survey was conducted starting from February 2019 by using two online questionnaire frameworks, and more than 850 answers from 42 countries has gathered at the time of this writing. Some of preceding work in surveying MPI uses are questionnaire surveys like ours, while others are conducted either by analyzing MPI programs to reveal static behavior or by analyzing dynamic runtime behavior of MPI jobs by using profiling tools. Our survey is different from the other questionnaire survey in terms of geographically wide-spread and the much larger number of participants. As a result it is possible to illustrate the current status of MPI users more accurately and with a wider geographical distribution. In this report, we will show some interesting findings, comparing the results with preceding studies when possible, and conducting some recommendations for MPI Forum based on the findings.

Keywords:
2020 MSC: 68-02, Message Passing Interface (MPI), survey

1. Background

Existing studies on MPI uses are focused on a restricted target domain, such as the Exascale Computing Project (ECP) [1] study conducted in 2017 [2] that focused on MPI usage in the context of ECP applications; and/or are generally geographically constrained to a single laboratory, funding agency or at best, country. As such they provide sporadic, disconnected views on the real uses of MPI across the world. Interestingly enough, and mostly by coincidence, simultaneously with the ECP study another survey was conducted in Japan targeting HPCI [3] users which included several questions asking about MPI [4]. HPCI is an infrastructure for HPC users in Japan connecting major supercomputers owned by universities and governmental research institutes. If both questionnaire surveys would have the same questions, we could have compared the answers to reveal the differences between US and Japan MPI user communities. Unfortunately, a single question was similar in both studies, limiting the correlations between the two surveys.

These studies highlighted the need to conduct a larger, more comprehensive, study, reaching across many diverse community of MPI users, and therefore inspired our effort. Unlike it's predecessors we shifted the study's focus from the high-end HPC community, and targeted a wider audience and involved a larger spectrum of geographically distinct users. Since MPI has been a widely used vehicle for high-performance computing for decades, this larger-scale questionnaire survey would be beneficial not only for deciding the future direction of MPI, but also for understanding the feature differences of MPI users among countries and/or regions of the world.

Our team started to conduct such a study as a project at JLESC [5] which is an international research collaboration framework. The international nature of this survey matches the concept of JLESC. Co-authors are a member of JLESC and responsible for the country and/or region where they belong. For the design of the questionnaire, we consulted two social scientists, Prof. Marshall Scott Poole at Illinois Univ., and Prof. Iftekhar Ahmed at Univ. of North Texas, participating JLESC workshops to investigate how researchers can collaborate in the JLESC framework.

Table 1: Comparison of ECP and HPCI Surveys

	ECP	HPCI	ours
Concern	MPI usage in Exascale Computing	Computing Environment	MPI (w/o MPI-IO)
Target	USA	Japan	World
#Questions	64 (max)	75 (max)	30
#Participants	77	105	851

Email addresses: ahori@riken.jp (Atsushi Hori), emmanuel.jeannot@inria.fr (Emmanuel Jeannot), bosilca@icl.utk.edu (George Bosilca), t-ogura@riken.jp (Takahiro Ogura), bgerofi@riken.jp (Balazs Gerofi), jie.yin@riken.jp (Jie Yin), yutaka.ishikawa@riken.jp (Yutaka Ishikawa)

To give an order of comparison with preceding studies, our MPI International Survey, ECP survey, and HPCI survey are summarized in Table 1.

2. Related Work

The existing MPI-related surveys can be categorized in three survey classes; questionnaire survey asking MPI users questions specifically crafted toward a target goal (*Q*) and reflecting more the human understanding or knowledge of MPI capabilities, application-oriented statistical surveys statically analyzing MPI programs and classifying occurrences of MPI calls (*S*), and application-oriented statistical surveys analyzing MPI applications behavior at runtime by using a profiling tool (*R*).

Our survey and the ECP survey are examples of the *Q* category, and highlight, as mentioned above, the user understanding of MPI capabilities and knowledge of MPI features. They can more easily identify what new MPI features are become known by the users community, well before they start appearing in MPI applications.

In the *S* category, [6] statically investigated 110 open-source MPI programs. [7] investigated 14 MPI programs chosen from the ECP Proxy Applications Suite 2.0 [8]. They offer a pragmatic view on the usage patterns of MPI function in existing applications, and can serve as an indicator of what MPI features translates into real usages.

In the *R* category, [9] collected and analyzed the runtime behavior by running more than 100K MPI jobs, with a smaller but still significantly distinct number of different applications. [10] takes a similar approach, but focuses on HPC applications and analyzed the behavior of DOE Mini-apps based on the trace data which DOE made public. It is interesting to note that the target community for these 2 studies is significantly different, the second one looking at applications developed by a user community more inclined to use advanced features of MPI.

The target of the questionnaire surveys are MPI users, the target of *S* is MPI programs, and the target of *R* is MPI jobs. In spite of these target differences, we dare to compare some results of those non-questionnaire-based surveys and ours in the following sections as appropriate.

3. Survey

Design

The social scientists suggested that the number of questions must be limited around 30, to keep participants engaged and not to loose their concentration and focus. This number is significantly smaller that those of ECP and HPCI surveys, forcing us to restrict the scope of the questions, and focus on few, critical aspects to the future of the MPI effort. As an example, we deliberately excluded some topics, such as MPI-IO, and instead focused on MPI communications. We designed the questionnaire so that participants can answer questions as easy as possible, and the questions to force participants doing some extra work to answer the questions, such as counting the lines of code of their programs, are eliminated.

Similarly to the ECP questionnaire, we initially started with Google Forms to develop ours. Later in our project, and mostly for geopolitical reasons, we replicated the same questionnaire using Microsoft Forms for those who cannot access Google Forms. All graphs in this paper were generated using the aggregated data from both forms (Google and Microsoft) exported using a CVS format, and then manipulated using statistical tools developed in Python and R.

The draft questionnaire was tested and validated by several active members of the MPI standardization body, as well as researchers from Inria and Riken Center for Computational Science (R-CCS). The questionnaire was was available online and receiving answers from February 17, 2019 until recently. In fact, the two forms remain open to additional answers, but taking in account the rate of the contributions we do not expect the outcome to drastically change. All questions, their choices, and abbreviations of the choices used in this report are listed in Appendix A.

Distribution

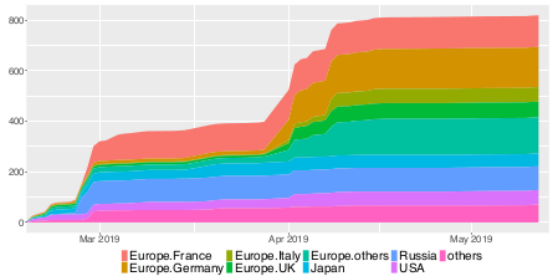

Figure 1: Time series in first 90 days

One of the first questions we had to ask was how to reach a largely international community of researchers and users, quickly and efficiently while hoping for a significant contribution. The survey was initially announced via several major mailing lists in the community such as `hpc-announce@mcs.anl.gov`, but the contributions were extremely slow to arrive. In order to improve participation, we decided to approach the problem more locally and reached out to different collaborators and asked them to locally distribute the questionnaire inside their institutions, via their own distribution process (mailing list, forums, or different form of social platforms). As highlighted in Fig. 1, more localized means of distribution were highly beneficial, each one of the steps in the figure corresponding to a new distribution campaign to a new set of institutions.

This local distribution strategy worked well on some regions but did not work universally. Table 2 shows the number of participants of top 11 countries (all countries are listed in Appendix B). Comparing with Table 3 listing the top 10 countries in the performance share in the Top500 [11], the three major countries, USA, China and Japan in Top500, are not even in the top 5 in our survey. Especially China has only 18 participants including Taiwan (2). We tried to increase the number

Table 2: Top 11 Contributors

#	Contributor	#Ans	[%]
1	Germany	159	18.7
2	France	125	14.7
3	Russia	94	11.1
4	UK	67	7.9
5	Japan	64	7.5
6	USA	58	6.8
7	Italy	57	6.6
8	Switzerland	40	5.8
9	South Korea	27	3.2
10	Austria	26	3.1
11	China (incl. Taiwan)	18	2.1

42 contributors, 851 participants

Table 3: Top50[5] erformance Share (Nov. 2020)

#	Country	[%]
1	USA	27.5
2	China	23.3
3	Japan	24.4
4	Germany	5.4
5	France	3.7
6	Italy	3.2
7	UK	1.4
8	Canada	1.1
9	Netherlands	1.0
10	Switzerland	1.0

of participants of those countries as much as we could, making and distributing fliers at several conferences, with little positive outcome. While the root cause is still unclear, this pinpoints to the need for alternative distribution schemes, especially in these locations.

Major contributors

For the remaining of this report, geographical regions, either countries or regions, having more than 50 participants, are called **major contributors** and are the object of cross-tab analysis. Such major contributors are Germany, France, Russia, UK, Japan, USA, Italy and the rest of European countries. It should be noted that the information used to define the major contributorwas the workplaces in the last 5 years and not the nationality of the individual participants. There is a trade-off between the number of participants from each major contributorand the number of the major contributorsin the cross-tab analysis. The threshold of 50 participants was selected to balance this trade-off but it was not our intent to define this number as a satisfactory participation limit. Hence, some cross-tab analysis may not be reliable enough.

Participants' profile

Fig. 2 shows the graph of Q1 regarding participants' occupation. As shown, the majority, roughly 80% of participants are working at universities or governmental research institutes.

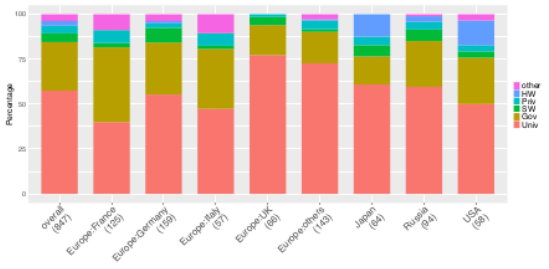

Figure 2: Q1: Occupations *(single)*

Fig. 3 shows the major field the participants are involved with, field selected among a set of provided choices. Roughly speaking, most participants are working on numerical applications and/or libraries, which can either be interpreted as a confirmation that most of the government sponsored MPI usages are in numerical applications or libraries, or that it was the most encompassing field among the proposed choices. It is interesting to note that in 2 major contributors, Japan and US, the percentage of parallel languages and OS/runtimes participants is significantly higher compared with the rest of major contributors.

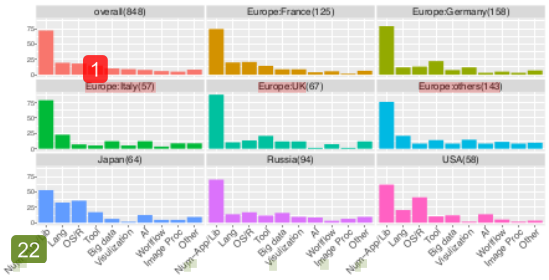

Figure 3: Q7: Working Fields *(multiple)*

4. Comparison with the ECP survey

Although the ECP questionnaire and our questionnaire were designed independently, there are several comparable questions. Before going into the details, we need to clarify some points about the profiles of the participants in our survey. Due to the target audience, we can assume that some of the participants of our survey also participated in the ECP survey. However, significant differences between the outcome of the two surveys arise.

First, and this mainly due to the larger ratio of participants from universities and national laboratories, it seems likely that the ECP survey contains more answers from experts MPI users. Fig. 4 shows the results of self-evaluation of the participants MPI skill in our survey. It is worth raising attention to the US case (right most bar), where almost half of participants rate

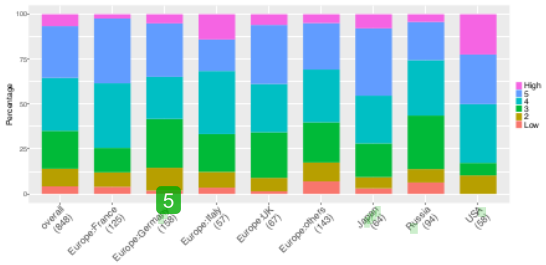

Figure 4: Q3: Self assessment of MPI Skill *(single)*

themselves as highly skilled MPI users (*5* or *High*), significantly ahead that any other major contributors. Not only that, but none of the participants indicated a low MPI-related skill.

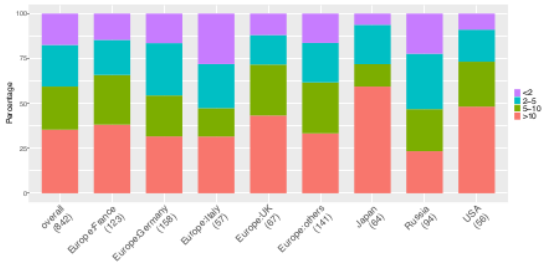

Figure 5: Q6: MPI Experience *(single)*

Fig. 5 shows the more interesting result. This graph is the result of asking *How long have you been writing MPI programs?* and the choices are *more than 10 years* (denoted as *>10*), *between 5 and 10 years* (denoted as *5-10*), *between 2 and 5 years* (denoted as *2-5*) and *less than 2 years* (denoted as *<2*). Interestingly 9% of US participants have less than 2 years MPI experience, but they do no rank their MPI expertise the lowest (Fig. 4). Japan followed by USA has the highest percentage in more than 10 years and has the lowest percentage in less than 2 years experience. Russia followed by Italy has the highest percentage in less than 5 years experience (including the less than 2 years experience case).

We chose three questions from both of out survey and ECP survey that are very similar and thus the results are comparable (Table 4). We will discuss those results in the following subsections.

4.1. Layering MPI calls

Fig. 6 shows the result of our survey and Table 5 shows the comparison between ours and the ECP survey. In the ECP survey, the participants are categorized into two groups; application development (AD) and system technology (ST). It is interesting that the percentage of the participants having MPI layer(s) in our survey is roughly 50% even in the US, whilst the ratio of yes and no is approximately 6 : 4 in the ECP survey.

Table 4: Comparable Questions

Our Survey	ECP Survey
Layering MPI calls (subsection 4.1)	
Q21: In most of your programs, do you pack MPI function calls into their own file or files to have your own abstraction layer for communication? *(single)*	Q22: Do you have an abstraction layer that hides the MPI calls? Or do most of your developers write MPI calls directly? *(single)*
Using MPI Aspects (subsection 4.2)	
Q17: What aspects of the MPI standard do you use in your program in its current form? *(multiple)*	Q35: What aspects of the MPI standard do you use in your application in its current form? *(multiple)*
Multi-threading (subsection 4.3)	
Q18: Which MPI thread support are you using? *(multiple)*	Q59: Which MPI threading option are you using? *(single)*

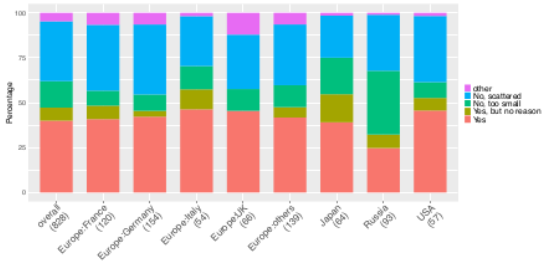

Figure 6: Q21: Layering MPI calls *(single)*

Having a closer look at Fig. 6, the answer, *No, my program is too small to do that*, dominates in Russia. In the other major contributors, the participants having a packing layer occupies 40-50%.

4.2. Using MPI Aspects

The Q35 in the ECP survey and Q17 in our survey are almost equivalent questions, although choices are somewhat different. Fig. 7 shows the result of our survey and Table 6 shows the comparison between ECP's and ours on the same choices. As shown in Fig.7, the using aspects can be categorixed in three groups; A) more frequently used (point-to-point and collectives), B) second frequently used (*Datatypes*, *with OpenMP*, *Communicator*, and *One-sided*), and C) less frequently used (*PMPI*, *Persistent*, and *dyn. process* (dynamic process). It should be noted that all these less frequently used features were already introduced and standardized in MPI 2.2 which was release in 2009. Despite the 10-year appearance, those features failed to get popular.

Most notable difference between our survey and the ECP survey is the datatype (Table 6). The percentage in datatype

Table 5: Layering MPI calls

Choice		Our Survey [%]		ECP [%]		
		overall	USA	AD	ST	AD+ST
Yes	-	40	46			
	no reason	7	7			
	(sum)	47	53	79	46	62
No	too small	15	9			
	-	33	37			
	(sum)	48	46	21	54	38
Other	-	5	2	-	-	-

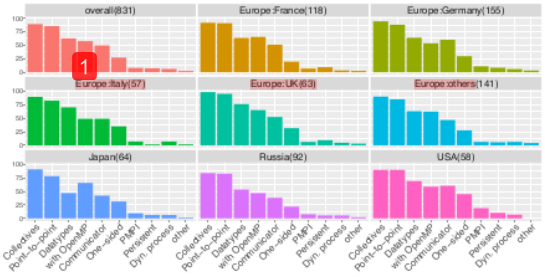

Figure 7: Q17: Using MPI Aspects *(multiple)*

in ECP is 23% and the percentage in our survey is more than 60% in both overall and USA. In the ECP survey, there are three questions asking using MPI aspects in different usage scenarios; **a)** current, **b)** exascale, and **c)** performance critical. In all three questions, the percentages of using datatype are low.

Table 6: Using MPI Aspects

Choice	Ours [%]		ECP (current usage) [%]		
	overall	USA	AD	ST	AD+ST
Collectives	89	90	86	75	80
Point-to-point	85	90	96	79	88
Datatype	63	69	25	21	23
Communicator	50	60	68	54	61
One-sided (RMA)	27	45	36	7	21
PMPI	8	19	11	0	14

* Both are multiple answer questions
** Common choices in both surveys are shown

Fig. 8 is the addition. This graph is a heatmap of the cross-tab analysis between Q3 asking MPI skill and Q16 asking unknown MPI features. The darker the color of a cell, the higher the frequency (Legend combined with color bar can be found at the bottom of the figure. The numbers in the legend cells are percentages). Less frequent rows (1 is the lowest skill and 6 is the highest skill) in this figure are omitted to increase readability. Natural thinking may conclude that the higher the MPI skill, lesser the unknown MPI features. The result is quite interesting, the less used features in Fig. 7, PMPI, persistent, and dynamic process, are almost independent from the MPI skill.

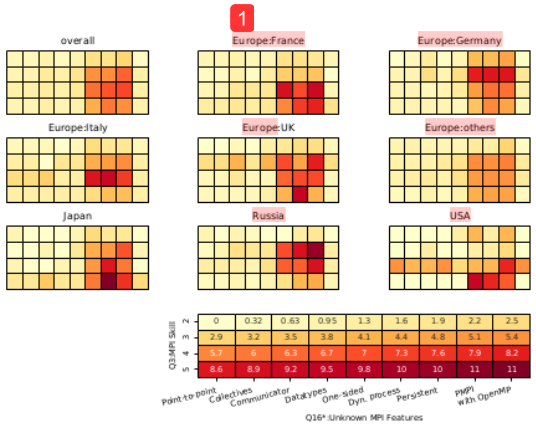

Figure 8: Q3-Q16: MPI Skill *(single)* and Unknown MPI Features *(multiple)*

The similar situation can be seen in the cross-tab analysis between Q6 and Q16 (Fig. 9). It would be plausible, the longer the MPI experience, the less unknown features. However, some major contributors (France, UK and Japan) in these heatmaps show that the longer the experience, the higher the percentage of having unknown features. This may indicate that the long experienced MPI users may not catch up the newly introduced MPI features.

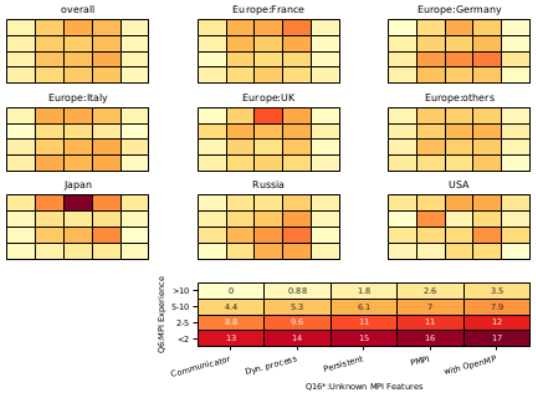

Figure 9: Q6-Q16: MPI Experience *(single)* and Unknown MPI Features *(multiple)*

This may indicate that the MPI standard is very profound. Even the most basic send/receive functions, although they look very simple and natural, they require deep knowledge such as possibility of deadlock, timing of buffer access, blocking/non-blocking, and so on.

Fig. 10 is the result of Q27 asking *What MPI feature(s) are NOT useful for your application?*. Although many participants think MPI has no *not useful features*, fairly amount of participants think the dynamic process feature not useful. Thinking of

the fact that the dynamic process feature is not used by the most participants (Q17, Fig. 7), this result is quite interesting. This tendency is also reported in [6].

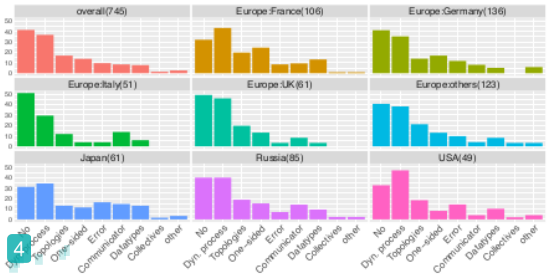

Figure 10: Q27: Useless Features *(multiple)*

The dynamic process feature is on the border of process management and communication, since the process creation itself is obviously out of the scope of the MPI standard, while the communication between the existing (MPI) processes and newly create (MPI) processes must be defined in the standard. Indeed, the implementation of dynamic process creation spreads many parts of a computing system: MPI library, process manager, job scheduling system and system operation. This complexity might make the use of the dynamic process creation hard and impractical for MPI users and the un-usefulness of dynamic process may not come from the MPI standard.

4.3. Multi-threading

The difference between our survey and the ECP survey can also be found on the question asking multi-threading support. Fig. 11 shows the result of our survey and Table 7 shows the difference. Note that our question is multiple-answer and the ECP question is single-answer. In both surveys (USA and ECP), the percentage of using `MULTIPLE` is the highest excepting the AD case. However, the percentage of the choice *I don't know* is fairly larger than ours. This may sound contradictory because the ECP participants would be more experienced MPI users.

Table 7: Multi-threading

Choice	Our Survey [%]		ECP *(single)* [%]		
	overall	USA	AD	ST	AD+ST
SINGLE	29	22	(no corresponding choice)		
FUNNELED	18	13	18	18	18
SERIALIZED	12	10	18	18	18
MULTIPLE	22	31	18	32	25
never used	23	16	(no corresponding choice)		
not know	14	8	25	25	25

The usage of `MULTIPLE` in US is also the highest among the major contributors (Fig. 11). France and Germany have the same trend. In Italy, Japan, Russia and the other European countries, the percentages of *I don't know* are the highest. In UK, the percentage of using `SINGLE` is the highest.

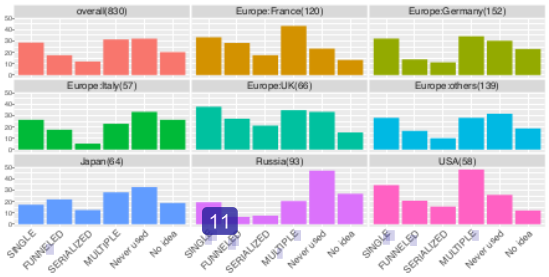

Figure 11: Q18: Multi-threading *(multiple)*

Remember this is a multiple answer question. Table 8 shows the top 7 of the percentages of raw answers (combined answers) of the overall. These top 7 percentages occupy about 85% in total. Nearly half participants answered *never used* or *no idea*. The numbers in parenthesis in this table are the percentage of participants excluding those who answered *never used* or *no idea*. Half of threading-aware participants are using `SINGLE` and/or `MULTIPLE`. Although many participants ignore the thread mode, some participants use a particular thread support (`SINGLE` or `MULTIPLE`) and some other participants select one of supported thread capabilities positively.

Table 8: Multi-threading - Raw Answers

Threading Support	Overall Percentage
never used + no idea	48
`MULTIPLE`	12 (23)
`SINGLE, MULTIPLE`	8 (16)
`SINGLE`	7 (14)
`SINGLE, FUNNELED, SERIALIZED, MULTIPLE`	4 (8)
`SINGLE, FUNNELED`	3 (7)
`SERIALIZED`	3 (5)

Numbers in parenthesis are percentages excluding *never used* and *no idea*

In [9], approximately 75% of their target executables (not number of jobs) on Mira (total of 68) are using `SINGLE`, 15% use `FUNNELED` and 4% use `MULTIPLE`. In [6], approximately 60% of their target programs use `FUNNELED`, 30% use `MULTIPLE`, 20% use `SINGLE` and only few percent use `SERIALIZED`. Thus, the thread support usage varies on each survey and further investigation is needed to state a result.

5. Other Findings

5.1. MPI Implementations

Fig. 12 shows the Q12 result asking which MPI implementation(s) using. The top 3 implementations, Open MPI, Intel MPI and MPICH, dominates in all major contributors followed by MVAPICH. A large disparity can be seen on the other implementations. Taking a look at the *other* choice, there are four (4) answers raising the *bullx MPI* and another four (4) using

MadMPI [12] in France, and 10 answers raising ParaStation MPI in Germany. The frequency of using Fujitsu MPI, ParaStation MPI, bullx MPI and others heavily depend on countries of participants and the countries where the MPI was developed.

Figure 12: Q12: Using MPI Implementations *(multiple)*

Q13 is asking *why did you choose the MPI implementation(s)* and its answers are shown in Fig. 13. The highest percentage of US participants selected *Familiar*. Many Italian participants also selected *Familiar.* Many Russian participants selected *No reason.* The largest part of UK and Germany participants selected *No choice.* In general, more than half of participants excepting US select MPI implementation(s) without any reason nor freedom of choice.

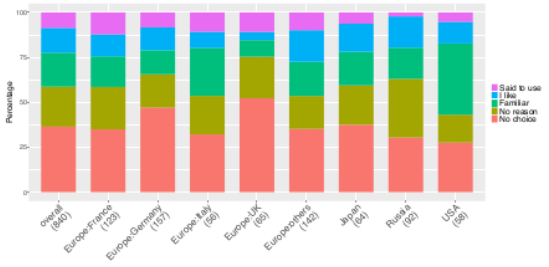

Figure 13: Q13: Choosing MPI Implementations *(single)*

5.2. MPI+X and Alternatives

Fig. 14 shows the result of Q22 asking *Have you ever written MPI+"X" programs?*. Most participants have the experiences of writing MPI+OpenMP programs. In US, *CUDA* is the second largest and the percentage of *No* is the lowest among the others. Considering the low percentage of *No* in overall (approx. 25%), 3/4 participants using MPI with something else. [6] also reported that the approximately 3/4 of the target programs use the hybrid model of MPI+OpenMP.

When the participants were asked the question *What, if any, alternatives are you investigating to indirectly call MPI or another communication layer by using another parallel language / library?*, they exhibited as shown in Fig. 15. In overall, almost half participants do not investigate the alternatives. The second largest answer was *Framework* (i.e. a framework or library

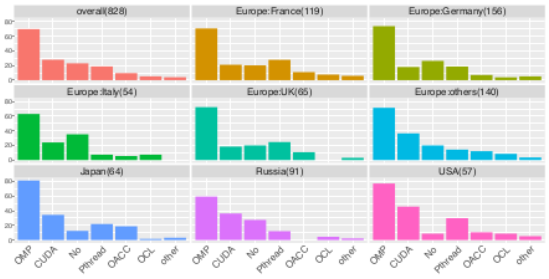

Figure 14: Q22: MPI+X *(multiple)*

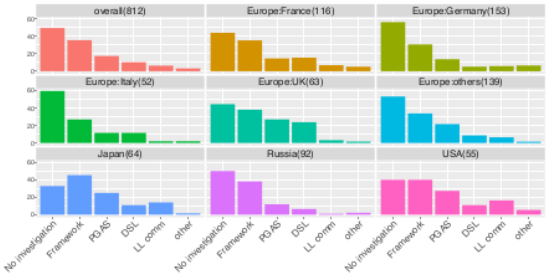

Figure 15: Q24: MPI Alternatives *(multiple)*

using MPI) followed by *PGAS*. The differences over the major contributors are not so big.

Fig. 16 shows the cross-tab analysis of the above Q22 and Q24. A certain percentage of participants of Germany, Italy, Russia and other European countries using MPI+OpenMP without investigating the MPI alternative (upper right corners of the heatmaps).

5.3. Compatibility vs. Performance

In somewhat long history of MPI, maintaining the backward compatibility can be obstacles when introducing new features to enhance MPI capabilities. Fig. 17 shows the result of the question asking which is more important, performance or compatibility. Fig. 18 shows the percentages of Q28 asking backward compatibility.

In Fig 17, let's consider three groups; *performance group* choosing *Performance* or *4*, *compatibility group* choosing *Portability* or *2*), and *grey group* who chose *3*. Apparently the grey group dominates in most major contributors (excepting Russia), however, the performance group occupies more percentage. Regarding to Russia, a certain percentage of Russian participants answered *my program is too small* in Q21 (Fig. 6). If program is small enough, then the compatibility does not cause a big problem.

As shown in Fig. 18, around 40% of participants answered that the compatibility is very important, while the rest of participants may accept the incompatibility conditionally. The incompatibility forces users to update their programs. The result

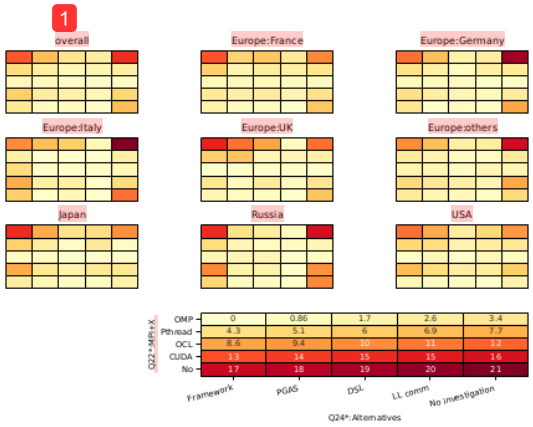

Figure 16: Q22-Q24: MPI+X *(multiple)* and MPI Alternatives *(multiple)*

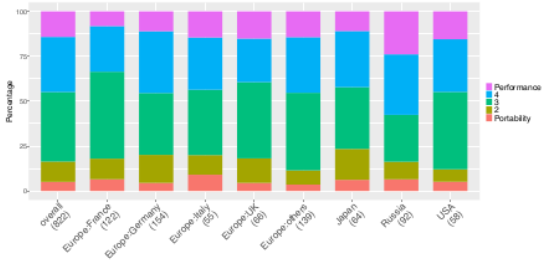

Figure 17: Q29: Performance vs. Compatibility *(single)*

of Q28 may suggests that users would accept incompatibility if the users could get some kind of benefit from the incompatible updates of MPI specification.

5.4. Learning MPI

Fig. 19 shows the percentages of how participants learned MPI. In this graph, *Other lec.* indicates the choice *Other lectures or tutorials (workplace, conference)*. The UK and Russia participants preferred to learn by Internet. The participants of Germany and other European countries preferred to have other lectures. The percentage of reading *Books* in US is the highest. Taking a look at the other answers, 18 participants learned by reading existing code and 8 participants learned by doing[1].

Fig. 20 shows the graph of Q9, asking if participants have read the MPI standard document. Not surprisingly, around 60% of participants, independent from contributors, read the standard partly. Most interesting contributor is UK, where the percentage of participants reading all document and, at the same time, the percentage of the answer *No* are the highest among the major contributors.

[1]One participant answered *reverse engineering!*

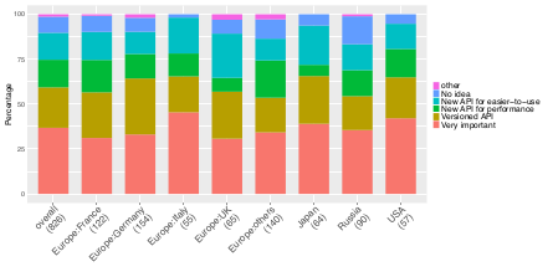

Figure 18: Q28: Backward Compatibility *(single)*

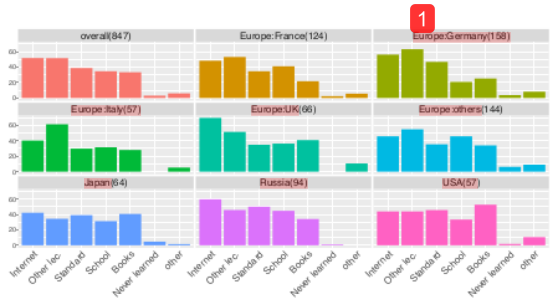

Figure 19: Q10: Learning MPI *(multiple)*

Now let's see how people are writing MPI programs. Fig. 21 shows the percentages of Q14 asking *How do you check MPI specifications when your are writing MPI programs?* Most users are checking MPI specifications by reading online documentations (e.g., man pages), searching Internet, and reading the standard. As shown in the previous figure (Fig. 20), users are reading the standard partly because of checking the MPI specifications.

It would be nice to have the cross-tab analysis between Q3 (MPI skill) and Q9 (reading MPI standard). Unfortunately the participants reading the standard partly dominates and the cross-tab analysis between them did not give us any clear evidence. Instead here we have Fig. 22, cross-tab heatmaps between Q3 and above Q14. There can be seen a a weak correlation, those

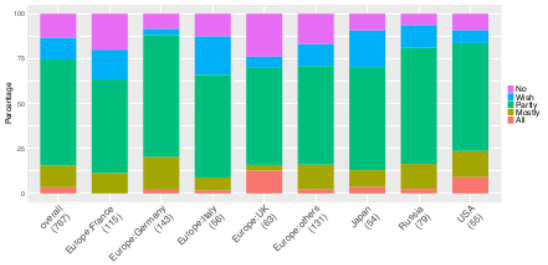

Figure 20: Q9: Reading MPI Standard *(single)*

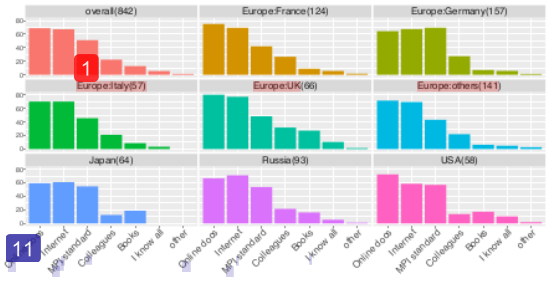

Figure 21: Q14: Checking Specification *(multiple)*

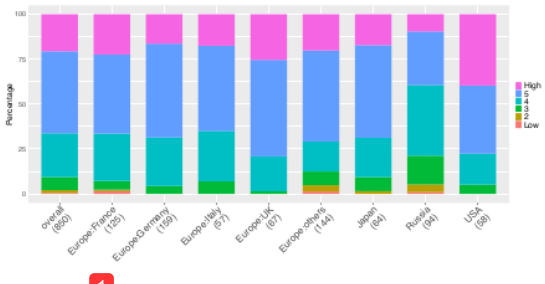

Figure 23: Q2: Rate your overall programming skill (non-MPI programs)

who check MPI standard have higher MPI skill, in some major contributors (France, Germany, and Japan).

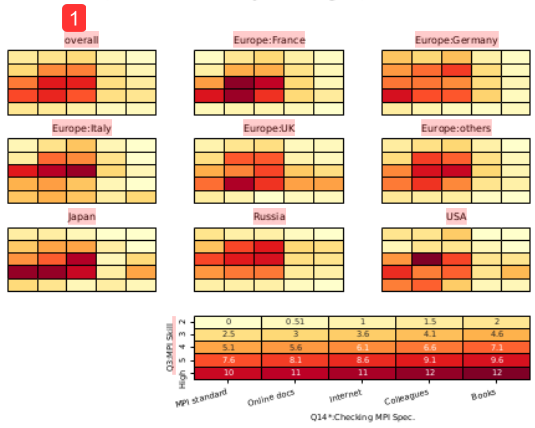

Figure 22: Q3-Q14: MPI Skill *(single)* and Checking Specification *(single)*

Another interesting result can be seen in Fig. 23 asking *Rate your overall programming skill (non-MPI programs)*. People with high programming skills who chose 4 or more account for more than 90%. This indicates that people with high program skills shown interest in our survey and/or that program beginners do not use MPI. This might indicate that MPI programming requires specific skills whom basic developers do not necessarily master. By contributor comparison, Russia shows a different tendency from other contributors.

Generally speaking, there can be a case where a particular question in a survey may ignite for participants to explode their dissatisfaction by writing messages into a free text field. The largest number of *other* inputs in our survey can be found at Q19 asking *What are your obstacles to mastering MPI?* (Fig. 24). Although the largest answer is the choice of *No obstacles,* we got 111 *other* inputs exceptionally. Q4 and Q7 are the second largest (70), but these are because of the variety of answers. More than 20 participants answered *other* raise *time (to master MPI)*. Many other participants pointed out the need of MPI programming guideline (*clear doc.*, *internal doc.*, *implementation doc.*, *performance guideline*, and so on, in their

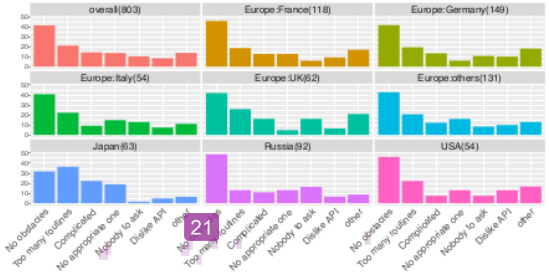

Figure 24: Q19: Learning Obstacles *(multiple)*

words). Some participants complaints about MPI implementations and the (performance or specification) differences among implementations.

As shown in Fig. 25, the cross-tab heatmap graphs between Q6 (Fig. 5) and Q3 (Fig. 4), there is a strong correlation, from lower-left to higher-right, between those two questions regardless to major contributors. And these graphs tell that it takes more than 10 years of MPI programming experience to reach the high MPI skill (*4* to *High*) in most major contributors. Many MPI users in US can reach the high (*4* or higher) MPI skill in 5 to 10 years. Hence, it takes more than 5 or 10 years to reach the high MPI skill mostly. Considering this fact and the profound nature of MPI specification (Subsection 4.2), MPI could be said to be very difficult library to master.

5.5. MPI Programming Difficulty and Tuning

Fig. 26 shows the result of Q15 asking *What is the most difficult part of writing an MPI program?* and Fig 27 shows the result of Q23 asking *Is there any room for performance tuning in your MPI programs?* The largest part of US and UK participants chose *algorithm design* whilst the participants of the other contributors chose *Debugging*. In US, the second largest choice was *Domain Decomposition*. In Japan, the second largest is *Tuning*.

Fig. 27 has more divergence than Fig 26. The participants selected *my MPI programs are well-tuned* is only around 10% excepting Japan and Russia. There seems to be a lots of room to tune MPI programs in general, however, around 40% of participants said they do not have enough resource to do that. In

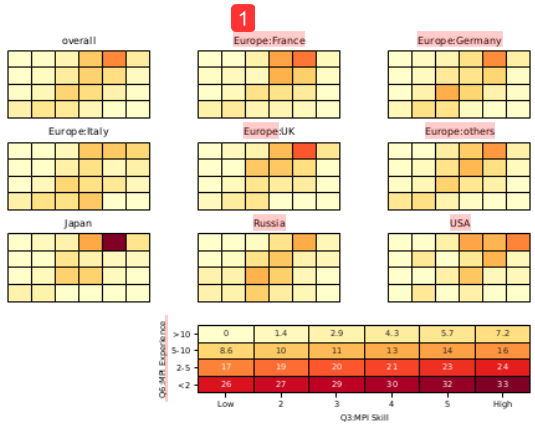

Figure 25: Q6-Q3: MPI Experience *(single)* and MPI Skill *single*

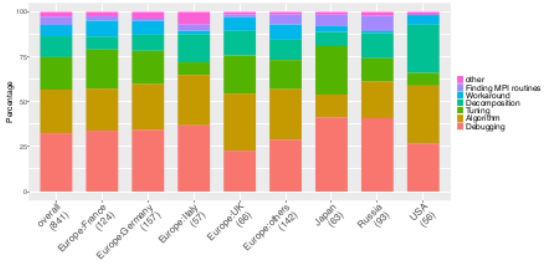

Figure 26: Q15: MPI Programming Difficulty *(single)*

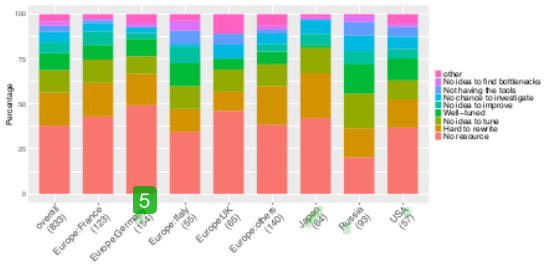

Figure 27: Q23: Performance Tuning *(single)*

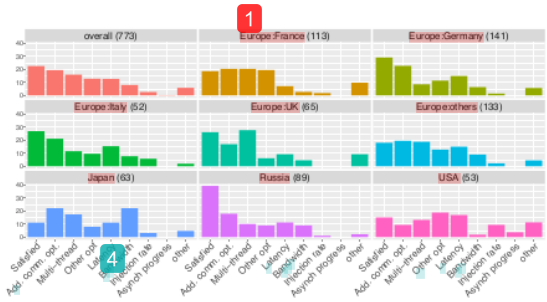

Figure 28: Q25: Features to improve *(single)*

Japan, the percentage of well-tuned program is only few percent. Contrastingly, Russian percentage of choosing *well-tuned* is the highest among the contributors.

5.6. Missing Features and Semantics

It is a general concern how MPI provides optimization opportunities in terms of hardware capabilities such as being able to handle the various topologies of hardware components more efficiently. To answer this, Q25 asking *If there were one communication aspect which is not enough in the current MPI could improve the performance of your application, what would you prioritize? Or* ... (Fig. 28), and Q26 asking *Is MPI providing all the communication semantics required by your application? If not, what is missing?* (Fig. 29) were prepared.

In Fig. 28, only 23% of overall MPI users are satisfied with the current situation. Interestingly enough the second largest percentage is *additional comm. opt.* (*Additional optimization opportunities in terms of communication (network topology awareness, etc.)*, followed by *Multi-thread* and *Other opt.* (*Optimization opportunities except communication (architecture awareness, dynamic processing, accelerator support, etc.)*).

Q26 is somewhat similar to Q25, but looking for more precise answers. This question tackles the issue on which semantic feature is missing from MPI. Overall a very similar picture emerges with Q25, almost one third of the participants are satisfied with the current situation. There is a high discrepancy between Japan where users are the least satisfied with the current situation and Russia which is the most satisfied. The same situation can be seen in Q25. The highest given answer concerns *Add. opt* (*Additional optimization opportunities in terms of communication (topology awareness, locality, etc.)*). This is coherent with Q25 and managing efficiently the topology and the locality seem a major concern to many users. Then comes the concerns about the lack of resilience, a concern shared by more than 20% of the participants. It is very interesting that most major contributors concern about the resilience. Hiding latency through generalization of asynchrony over the whole set of functions is another point raised repeatedly. 16% of the users think that a simpler and easier API would be desirable. Although there are relatively big disparity in the satisfaction (answering *MPI provides all*), the disparities of the other answers are smaller.

Finally, the least desired feature concerns the notion of endpoints, as discussed in the MPI standardization effort. However taking into account the extremely technical aspect of this question, and it's intricate evolution in the standard, it might be possible that most people answering this question knew little, and possible imprecisely, what this feature was exactly about.

5.7. Notes on Contributors

In this subsection, we summarize our findings on some contributorsshowing somewhat different results with the others.

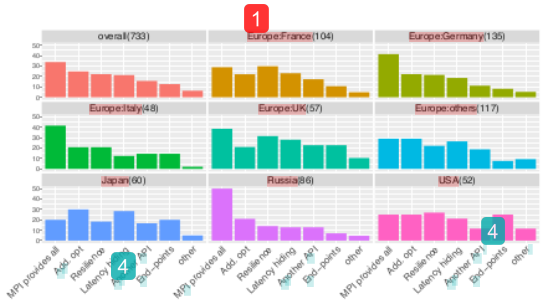

Figure 29: Q26: MPI Missing Semantics *(multiple)*

USA

US has the highest percentages; **a)** having the high MPI skill (Fig. 4), **b)** using MPI more than 10 years (Fig. 5), **c)** using the `MULTIPLE` threading support (Fig. 11), **d)** choosing familiar MPI implementations (Fig. 13), and **e)** reading MPI books (Fig. 19) among the major contributors. All these results indicates that the MPI users in US are most advanced.

Russia

Russia is; **a)** having the second largest percentage of MPI users less than 5 years of MPI experience, as well as Italy (Fig. 5), **b)** having the largest percentage of (non-MPI) programming beginners (Fig:23), **c)** having the highest percentage of thinking their MPI programs are well-tuned (Fig. 26), **d)** having the highest percentage not knowing which thread level they are using (Fig. 11), and **e)** the second highest contributor, next to US, choosing the MPI+CUDA (Fig. 14).

These findings may indicate that Russia is relatively younger in terms of MPI usage than the other major contributors. The high concern on MPI+CUDA, however, is very interesting.

Japan

In this survey, Japan shows the most unique results (this is already reported in [13]). Despite the high MPI skill (Fig. 4) and long MPI experience (Fig. 5), many Japanese MPI users seem to be suffered from debugging and tuning (Fig. 26), whilst many participants of the other contributors raise *Algorithm*.

Most notably, more than 50% of Japanese MPI users have the MPI experience more than 10 years exceptionally. This would sound good, however, the equally distributed answers is desired, thinking the continuous alternation of generations. The percentage of 5-to-10-year MPI experience in Japan is the smallest among the contributors. If this lack of mid-level is true, then the future of Japanese HPC community would not be bright.

6. Discussion

6.1. Inflating standard

As this survey reveals that some MPI features, which were already introduced almost a decade ago, are not widely accepted by MPI users yet (Subsection 4.2). Here a question may be raised, if this gap between the MPI features defined by the standard and the acceptance of the features will become wider or narrower?

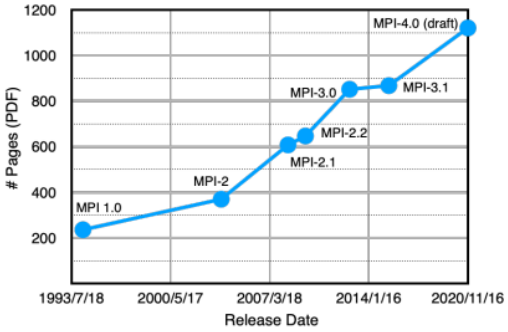

Figure 30: Page sizes of MPI Standards

Fig. 30 shows the number of pages (in terms of PDF, not the content) plotted over the released dates. Not surprisingly, the number of pages increases every time newer standard is released. It is a natural thinking that the number of pages and the number of features are proportional.

In many cases, the higher functionalities introduced by newer MPI standard yield more degree of implementation freedom. An MPI implementation can be optimized its performance by exploiting hardware resources without having the MPI users large effort. If only the most basic communication functions, send and receive, are supported by MPI then users have to write collective functions which are not easy if users wants optimize for the parallel machines they are using.

Another reason of the inflation is that MPI standard is the standard as a library. There is no way for library functions to know how the library functions are called in which context. The higher abstracted functions can give a library more information of how and which and thus the higher-level functions can be optimized. Traf, et al., gives a formal analysis on this point[14]. This situation, introducing higher-level functions into the standard, inflates the standard and always will be. And the gap will not be narrowed.

Hoefler et al., reported their idea to extract collective operation patterns from a series of communication primitives, send and receive[15], at run time. Applying this technique, a communication library will be able to optimize various communication patterns without introducing higher-level functions. Although their idea is at the experimental stage, however, this seems to be a good solution not introducing new functions but narrowing the gap.

6.2. Recommendation as a result of this survey

Currently the MPI standard documents are available in PDF format and hardcover books [16]. There are some MPI tutorial web sites ([17] as an example). [18] pointed out most of such web pages are out-dated and not thorough.

As already shown in Subsection 4.2, some MPI features, which are not newly introduced, have failed to be well-known. Further many MPI users have not enough time to master MPI as described in Subsection 5.4. These suggest that there is something wrong for people to learn MPI and writing MPI programs.

Thus, it is very important to narrow the gap described in the previous subsection by helping MPI users to learn and write MPI programs. We believe this is the responsibility of MPI Forum, since the other, volunteer-based approach would not be efficient and sufficient. If MPI Forum agrees with us for narrowing the gap, then MPI Forum should

- **slow the pace of introducing new features,** and
- **create a new working group to prepare and maintain web pages for tutorial, guidelines for MPI programming, and good (and bad) MPI examples**.

7. Summary

We have conducted a questionnaire survey and succeeded to gather more than 850 participants from more than 40 countries and regions. By analyzing the collected data, we could get several findings. As for the MPI features, the dynamic process feature is considered not only as a less-used feature but also a useless feature (highlighting that the MPI programming model is seen as *static*). By asking several questions how participants obtain MPI knowledge and experiences, it is revealed that MPI is a very difficult-to-use library. Many MPI users desire to have practical programming guideline, online documents in hypertext form, and useful sample programs. Most important (and most difficult) thing is those supplemental documents in any form must be up-to-dated and thorough. For the compatibility, many MPI users may accept to sacrifice the compatibility to get more performance.

All collected answers, the programs to analyze the survey data and to generate graphs, and all published reports are available at `https://github.com/bosilca/MPIsurvey.git`.

Acknowledgments

We thank to those who participated in this survey and those who helped us to distribute the questionnaire to their local communities. We especially thank to MPI Forum members who gave us many significant comments on the draft questionnaire. This research is partially supported by the NCSA-Inria-ANL-BSC-JSC-Riken-UTK Joint-Laboratory for Extreme Scale Computing [5].

References

[1] Exascale Computing Project, Exascale Computing Project, `https://exascaleproject.org`, 2021.

[2] D. E. Bernholdt, S. Boehm, G. Bosilca, M. Gorentla Venkata, R. E. Grant, T. Naughton, H. P. Pritchard, M. Schulz, G. R. Vallee, A survey of mpi usage in the us exascale computing project, Concurrency and Computation: Practice and Experience 32 (2020) e4851. E4851 cpe.4851.

[3] Research Organization for Information Science and Technology (RIST), High-Performance Computing Infrastructure, `http://www.hpci-office.jp/folders/english`, 2018.

[4] RIST, Report of the fourth survey on the K computer and the other HPCI systems, `http://www.hpci-office.jp/materials/k_chosa_4th`, 2018. (in Japanese).

[5] JLESC, Joint Laboratories for Extreme-scale Computing, `https://jlesc.github.io/`, 2021.

[6] I. Laguna, R. Marshall, K. Mohror, M. Ruefenacht, A. Skjellum, N. Sultana, A large-scale study of mpi usage in open-source hpc applications, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '19, Association for Computing Machinery, New York, NY, USA, 2019. URL: `https://doi.org/10.1145/3295500.3356176`. doi:10.1145/3295500.3356176.

[7] N. Sultana, M. Ruefenacht, A. Skjellum, P. Bangalore, I. Laguna, K. Mohror, Understanding the use of message passing interface in exascale proxy applications, Concurrency and Computation: Practice and Experience n/a (2020) e5901.

[8] D. F. Richards, O. Aaziz, J. Cook, H. Finkel, B. Homerding, P. McCorquodale, T. Mintz, S. Moore, A. Bhatele, R. Pavel, Fy18 proxy app suite release. milestone report for the ecp proxy app project (2018).

[9] S. Chunduri, S. Parker, P. Balaji, K. Harms, K. Kumaran, Characterization of mpi usage on a production supercomputer, in: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, 2018, pp. 386–400. doi:10.1109/SC.2018.00033.

[10] B. Klenk, H. Fröning, An overview of mpi characteristics of exascale proxy applications, in: J. M. Kunkel, R. Yokota, P. Balaji, D. Keyes (Eds.), High Performance Computing, Springer International Publishing, Cham, 2017, pp. 217–236.

[11] TOP500.org, Top 500, `https://www.top500.org`, 2021.

[12] A. Denis, NewMadelein, `http://pm2.gforge.inria.fr/newmadeleine`, 2021.

[13] A. Hori, G. Bosilca, E. Jeannot, T. Ogura, Y. Ishikawa, Is Japanese HPC another Galapagos? - Interim Report of MPI International Survey -, Technical Report 34, Information Processing Society of Japan, SIGHPC, 2019.

[14] J. Larsson Traff, W. D. Gropp, R. Thakur, Self-consistent mpi performance guidelines, IEEE Transactions on Parallel and Distributed Systems 21 (2010) 698–709.

[15] T. Hoefler, T. Schneider, Runtime detection and optimization of collective communication patterns, in: 2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT), 2012, pp. 263–

[16] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, Version 3.1, High Performance Computing Center Stuttgart (HLRS), 2015.

[17] W. Kendall, D. Nath, W. Bland, A Comprehensive MPI Tutorial Resource, `https://mpitutorial.com`, 2021.

[18] W. Kendall, MPI Tutorial Introduction, `https://mpitutorial.com/tutorials/mpi-introduction/`, 2021.

[dataset][19] A. Hori, T. Ogura, E. Jeannot, G. Bosilca, MPI International Survey GitHub Repository, `https://github.com/bosilca/MPIsurvey.git`, 2021.

Appendix A. List of Questions and Choices

The followings are the list of all questions associated with choices. The question numbers suffixed by asterisks (*) are multiple-answer questions. The choices are followed by corresponding abbreviations in square brackets, if any.

Q1: What is your main occupation **C**1) College/University [Univ] **C**2) Governmental institute [Gov] **C**3) Hardware vendor [HW] **C**4) Software vendor [SW] **C**5) Private research institute [priv] **C**6) Other

Country: Select main country or region of your workplace in past 5 years. Choose one from the country list.

Q2: Rate your overall programming skill (non-MPI programs). Choose one in the range of 1 to 6. [Low-High]

Q3: Rate your MPI programming skill. Choose one in the range of 1 to 6. [Low-High]

Q4*: What programming language(s) do you use most often? **C**1) C/C++ [C(++)] **C**2) Fortran 90 or newer [>=F90] **C**3) Fortran (older one than Fortran 90) [<F90] **C**4) Python [Py] **C**5) Java [Java] **C**6) Other

Q5: How long have you been writing computer programs (incl. non-MPI programs)? **C**1) more than 10 years [>10] **C**2) between 5 and 10 years [5-10] **C**3) between 2 and 5 years [2-5] **C**4) less than 2 years [<2]

Q6: How long have you been writing MPI programs? **C**1) more than 10 years [>10] **C**2) between 5 and 10 years [5-10] **C**3) between 2 and 5 years [2-5] **C**4) less than 2 years [<2]

Q7*: Which fields are you mostly working in? **C**1) System software development (OS, runtime library, communication library, etc.) [OS/R] **C**2) Parallel language (incl. domain specific language) [Lang] **C**3) Numerical application and/or library [Num-App/Lib] **C**4) AI (Deep Learning) [AI] **C**5) Image processing [Image Proc] **C**6) Big data [Bg Data] **C**7) Workflow and/or In-situ [Workfflow] **C**8) Visualization [Visualization] **C**9) Tool development (performance tuning, debugging, etc.) [Tool] **C**10) Other

Q8*: What is your major role at your place of work? **C**1) Research and development of application(s) [Apps] **C**2) Research and development software tool(s) [Tools] **C**3) Parallelization of sequential program(s) [parallelize] **C**4) Performance tuning of MPI program(s) [Tuning] **C**5) Debugging MPI programs [Debug] **C**6) Research and development on system software (OS and/or runtime library) [OS/R] **C**7) Other

Q9: Have you ever read the MPI standard specification document? **C**1) I read all. [All] **C**2) I read most of it. [Mostly] **C**3) I read only the chapters of interest for my work. [Partly] **C**4) I have not read it, but I plan to. [Wish] **C**5) No, and I will not read it. [No]

Q10*: How did you learn MPI? **C**1) I read the MPI standard document. [Standard] **C**2) I had lecture(s) at school. [School] **C**3) I read articles found on Internet. [Internet] **C**4) I read book(s). [Books] **C**5) Other lectures or tutorials (workplace, conference). [Other lec.] **C**6) I have not learned MPI. [Never learned] **C**7) Other

Q11*: Which MPI book(s) have you read? **C**1) Beginning MPI (An Introduction in C) [Beginning MPI] **C**2) Parallel Programming with MPI [Parallel Programming] **C**3) Using MPI [Using MPI] **C**4) Parallel Programming in C with MPI and OpenMP [Parallel Programming in C] **C**5) MPI: The Complete Reference [MPI: Complete Ref] **C**6) I have never read any MPI books [(no book)] **C**7) Other

Q12*: Which MPI implementations do you use? **C**1) MPICH **C**2) Open MPI [OMPI] **C**3) Intel MPI [Intel] **C**4) MVAPICH [MVA] **C**5) Cray MPI [Cray] **C**6) IBM MPI (BG/Q, PE, Spectrum) [IBM] **C**7) HPE MPI [HPE] **C**8) Tianhe MPI [Tianhe] **C**9) Sunway MPI [Sunway] **C**10) Fujitsu MPI [Fujitsu] **C**11) NEC MPI [NEC] **C**12) MS MPI [MS] **C**13) MPC MPI [MPC] **C**14) I do not know [No idea] **C**15) Other

Q13: Why did you choose the MPI implementation(s)? **C**1) I like to use it. [I like] **C**2) I was said to use it. [Said to use] **C**3) I could not have any choice (the one provided by a vendor). [No choice] **C**4) I am familiar with it. [Familiar] **C**5) I have no special reason. [No reason]

Q14*: How do you check MPI specifications when you are writing MPI programs? **C**1) I read the MPI Standard document (web/book). [MPI standard] **C**2) I read online documents (such as man pages). [Online docs] **C**3) I search the Internet (Google / Stack Overflow). [Internet] **C**4) I ask colleagues. [Colleagues] **C**5) I read book(s) (except the MPI standard). [Books] **C**6) I know almost all MPI routines. [I know all] **C**7) Other

Q15: What is the most difficult part of writing an MPI program? **C**1) Algorithm design [Algorithm] **C**2) Debugging [Debugging] **C**3) Domain decomposition [Decomposition] **C**4) Finding appropriate MPI routines [Finding MPI routines] **C**5) Implementation issue workaround [Workaround] **C**6) Performance tuning [Tuning] **C**7) Other

Q16*: Which MPI features have you never heard of? **C**1) Point-to-point communications [Point-to-point] **C**2) Collective communications [Collectives] **C**3) Communicator operations (split, duplicate, and so on) [Communicator] **C**4) MPI datatypes [Datatypes] **C**5) One-sided communications [One-sided] **C**6) Dynamic process creation [Dyn. process] **C**7) Persistent communication [Persistent] **C**8) PMPI interface [PMPI] **C**9) MPI with OpenMP (or multithread) [with OpenMP] **C**10) Other

Q17*: What aspects of the MPI standard do you use in your program in its current form? **C**1) Point-to-point communications [Point-to-point] **C**2) Collective communications [Collectives] **C**3) Communicator operations (split, duplicate, and so on) [Communicator] **C**4) MPI datatypes [Datatypes] **C**5) One-sided communications [One-sides] **C**6) Dynamic process creation [Dyn. process] **C**7) Persistent communications [Persistent] **C**8) MPI with OpenMP (or multithread) [with OpenMP] **C**9) PMPI interface [PMPI] **C**10) Other

Q18*: Which MPI thread support are you using? **C**1) MPI_THREAD_SINGLE [SINGLE] **C**2) MPI_THREAD_FUNNELED [FUNNELED] **C**3) MPI_THREAD_SERIALIZED [SERIALIZED] **C**4) MPI_THREAD_MULTIPLE [MULTIPLE] **C**5) I have never called `MPI_INIT_THREAD` [never used] **C**6) I do not know or I do not care. [Np idea] **C**7) Other

Q19*: What are your obstacles to mastering MPI? **C**1) I have no obstacles. [No obstacles] **C**2) Too many routines. [Too many routines] **C**3) No appropriate lecture / book / info. [No appropriate one] **C**4) Too complicated and hard to understand. [Complicated] **C**5) I have nobody to ask. [Nobody to ask] **C**6) I do not like the API. [Dislike API] **C**7) Other

Q20: When you call an MPI routine, how often do you check the error code of the MPI routine (excepting MPI-IO)? **C**1) I rely on the default Errors abort error handling [Default] **C**2) Always **C**3) Mostly **C**4) Sometimes **C**5) Never **C**6) Other

Q21: In most of your programs, do you pack MPI function calls into their own file or files to have your own abstraction layer for communication? **C**1) Yes, to minimize the changes of communication API. [Yes] **C**2) Yes, but I have no special reason for doing that. [Yes, but no reason] **C**3) No, my program is too small to do that. [No, too small] **C**4) No, MPI calls are scattered in my programs. [No, scattered] **C**5) Other

Q22*: Have you ever written MPI+X programs? **C**1) OpenMP [OMP] **C**2) Pthread **C**3) OpenACC [OACC] **C**4) OpenCL [OCL] **C**5) CUDA **C**6) No **C**7) Other

Q23: Is there any room for performance tuning in your MPI programs? **C1**) No, my MPI programs are well-tuned. [Well-tuned] **C2**) Yes, I know there is room for tuning but I should re-write large part of my program to do that. [Hard to rewrite] **C3**) Yes, I know there is room for tuning but I do not have enough resources to do that. [No resource] **C4**) I think there is room but I do not know how to tune it. [No idea to tune] **C5**) I do not have (know) tools to find performance bottlenecks. [Not having the tools] **C6**) I have no chance to investigate. [No chance to investigate] **C7**) I do not know how to find bottlenecks. [No idea to find bottlenecks] **C8**) I do not know if there is room for performance tuning. [No idea to improve] **C9**) Other

Q24*: What, if any, alternatives are you investigating to indirectly call MPI or another communication layer by using another parallel language/library? **C1**) A framework or library using MPI. [Framework] **C2**) A PGAS language (UPC, Coarray Fortran, OpenSHMEM, XcalableMP, ...). [PGAS] **C3**) A Domain Specific Language (DSL). [DSL] **C4**) Low-level communication layer provided by vendor (Verbs, DCMF, ...). [LL comm] **C5**) I am not investigating any alternatives. [No investigation] **C6**) Other

Q25: If there were one communication aspect which is not enough in the current MPI could improve the performance of your application, what would you prioritize? Or is MPI providing all the communication semantics required by your application? If not, what is missing? **C1**) Latency [Latency] **C2**) Message injection rate [Injection rate] **C3**) Bandwidth [Bandwidth] **C4**) Additional optimization opportunities in terms of communication (network topology awareness, etc.) [Additional comm. opt.] **C5**) Optimization opportunities except communication (architecture awareness, dynamic processing, accelerator support, etc.) [Other opt.] **C6**) Multi-threading support [Multi-thread] **C7**) Asynchronous progress [Asynch progress] **C8**) MPI provides all semantics I need [Satisfied] **C9**) Other

Q26*: Is MPI providing all the communication semantics required by your application? If not, what is missing? **C1**) Latency hiding (including asynchronous completion) [Latency hiding] **C2**) Endpoints (multi-thread, sessions) [End-points] **C3**) Resilience (fault tolerance) [Resilience] **C4**) Additional optimization opportunities in terms of communication (topology awareness, locality, etc.) [Additional opt] **C5**) Another API which is easier and/or simpler to use [Another API] **C6**) MPI is providing all the communication semantics required by my application [MPI provides all] **C7**) Other

Q27*: What MPI feature(s) are NOT useful for your application? **C1**) One-sided communication [One-sided] **C2**) Datatypes [Datatypes] **C3**) Communicator and group management [Communicator] **C4**) Collective operations [Collectives] **C5**) Process topologies [Topologies] **C6**) Dynamic process creation [Dyn. process] **C7**) Error handlers [Error] **C8**) There are no unnecessary features [No] **C9**) Other

Q28: Do you think the MPI standard should maintain backward compatibility? **C1**) Yes, compatibility is very important for me. [Very important] **C2**) API should be clearly versioned. [Versioned API] **C3**) I prefer to have new API for better performance. [New API for performance] **C4**) I prefer to have new API which is simpler and/or easier-to-use. [New API for easier-to-use] **C5**) I do not know or I do not care. [No idea] **C6**) Other

Q29: In the tradeoff between code portability and performance, which is more or less important for you to write MPI programs? Choose one in the range of 1 to 6. [Portability-Performance]

Appendix B. Contributors

Table B.9: Country

Country	Region Category	# Participants
Germany	Europe	159
France	Europe	125
Russia	Russia	94
UK	Europe	67
Japan	Japan	64
USA	USA	58
Italy	Europe	57
Switzerland	Europe	40
Korea, South	South Korea	27
Austria	Europe	26
China	China	16
Sweden	Europe	15
Spain	Europe	14
India	India	12
Poland	Europe	10
Netherlands	Europe	8
Brazil	Central and South America	6
Denmark	Europe	6
Czech Republic	Europe	5
Luxembourg	Europe	5
Canada	North America	4
Finland	Europe	3
Argentina	Central and South America	3
Australia	Australia	3
Taiwan	China	2
Serbia	Europe	2
Pakistan	Asia	2
Egypt	Africa	2
Greece	Europe	2
Belgium	Europe	2
Tunisia	Africa	1
Peru	Central and South America	1
Singapore	Asia	1
Norway	Europe	1
Mexico	Central and South America	1
Denmark, Austria	Europe	1
Croatia	Europe	1
Portugal	Europe	1
Estonia	Europe	1
Saudi Arabia	Asia	1
UAE	Asia	1
Ukraine	Europe	1
42 contributors		851

An International Survey on MPI Users

ORIGINALITY REPORT

11%

SIMILARITY INDEX

PRIMARY SOURCES

1	hal.inria.fr Internet	859 words — 8%
2	"Driving Scientific and Engineering Discoveries Through the Convergence of HPC, Big Data and AI", Springer Science and Business Media LLC, 2020 Crossref	43 words — < 1%
3	Miłosz Ciżnicki, Krzysztof Kurowski, Jan Węglarz. "Energy and performance improvements in stencil computations on multi-node HPC systems with different network and communication topologies", Future Generation Computer Systems, 2020 Crossref	40 words — < 1%
4	Nathanaël Cheriere, Matthieu Dorier, Gabriel Antoniu. "How fast can one resize a distributed file system?", Journal of Parallel and Distributed Computing, 2020 Crossref	34 words — < 1%
5	mafiadoc.com Internet	34 words — < 1%
6	Ian Karlin, Yoonho Park, Bronis R. de Supinski, Peng Wang et al. "Preparation and optimization of a diverse workload for a large-scale heterogeneous system", Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2019 Crossref	33 words — < 1%
7	Xi Luo, Wei Wu, George Bosilca, Yu Pei, Qinglei Cao, Thananon Patinyasakdikul, Dong Zhong, Jack	31 words — < 1%

Dongarra. "HAN: a Hierarchical AutotuNed Collective Communication Framework", 2020 IEEE International Conference on Cluster Computing (CLUSTER), 2020
Crossref

8	archiv.ub.uni-heidelberg.de Internet	30 words — < 1%
9	arxiv.org Internet	22 words — < 1%
10	link.springer.com Internet	20 words — < 1%
11	es.scribd.com Internet	20 words — < 1%
12	rmets.onlinelibrary.wiley.com Internet	16 words — < 1%
13	Lecture Notes in Computer Science, 2016. Crossref	14 words — < 1%
14	"Euro-Par 2016: Parallel Processing", Springer Science and Business Media LLC, 2016 Crossref	14 words — < 1%
15	www.csc2.ncsu.edu Internet	13 words — < 1%
16	Heike Jagode, Anthony Danalis, Reazul Hoque, Mathieu Faverge, Jack Dongarra. "Evaluation of dataflow programming models for electronic structure theory", Concurrency and Computation: Practice and Experience, 2018 Crossref	13 words — < 1%
17	hal.univ-lorraine.fr Internet	12 words — < 1%
18	Alexander Huck, Joachim Protze, Jan-Patrick Lehr, Christian Terboven, Christian Bischof, Matthias S. Muller. "Towards compiler-aided correctness checking of adjoint	12 words — < 1%

MPI applications", 2020 IEEE/ACM 4th International Workshop on Software Correctness for HPC Applications (Correctness), 2020
Crossref

19 Corrado, Cesare, Jamila Lassoued, Moncef Mahjoub, and Néjib Zemzemi. "Stability analysis of the POD reduced order method for solving the bidomain model in cardiac electrophysiology", Mathematical Biosciences, 2016.
Crossref
11 words — < 1%

20 hdl.handle.net
Internet
8 words — < 1%

21 epdf.pub
Internet
8 words — < 1%

22 d-nb.info
Internet
8 words — < 1%

EXCLUDE QUOTES	OFF	EXCLUDE MATCHES	OFF
EXCLUDE BIBLIOGRAPHY	OFF		