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 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

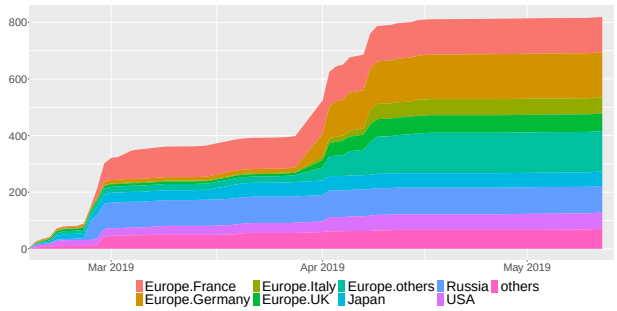

Figure 1: Time series in first 90 days

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 of participants of those countries as much as we could, making

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: Top500 Performance 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

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 contributor was 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 contributor and 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.

Fig. 3 shows the major field the participants are involved with, field selected among a set of provided choices. Roughly

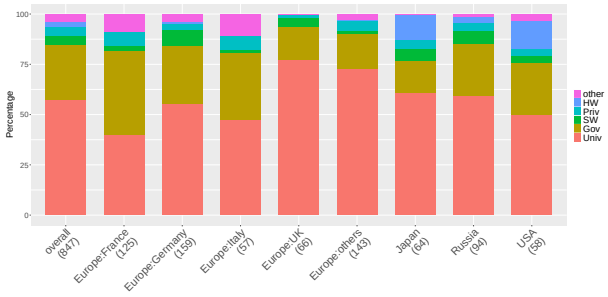

Figure 2: Q1: Occupations *(single)*

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.

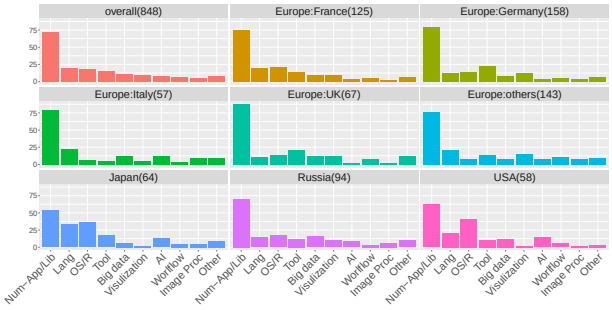

Figure 3: Q7: Working Fields *(multiple)*

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 reached by the survey propagation means (emails, HPC related mailing lists, and community based word-of-the-mouth), 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 highly HPC-centered participants or 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 themselves as highly skilled MPI users (*5* or *High*), significantly ahead that any other major con-

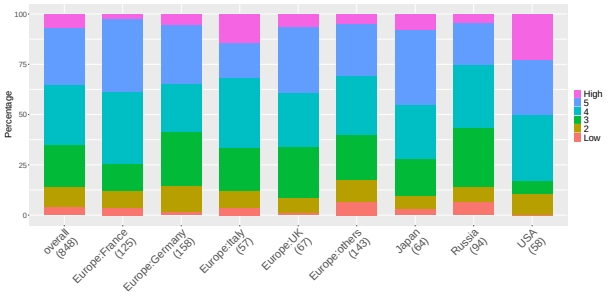

Figure 4: Q3: Self assessment of MPI Skill *(single)*

tributors. Not only that, but none of the participants indicated a low MPI-related skill.

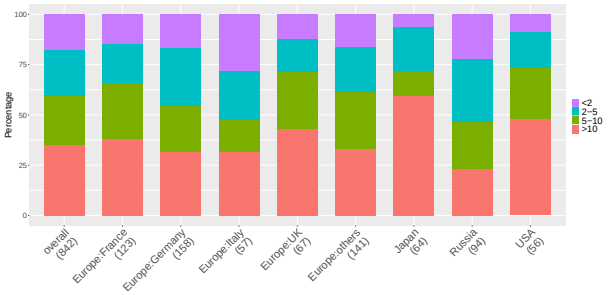

Figure 5: Q6: MPI Experience *(single)*

Fig. 5 shows an interesting result, picturing the answers about participants' expertise via the length of the interactions with the MPI world. The question was *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 only 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 closely has the highest percentage in participants with more than 10 years of experience and also the lowest percentage in those with 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).

A second set of questions (Table 4) with strong similarities between the ECP and our survey relates to the software stack where the MPI code is included. 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 focuses on the comparison between our 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. Having a closer look at Fig. 6, the answer, *No, my program is*

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)*

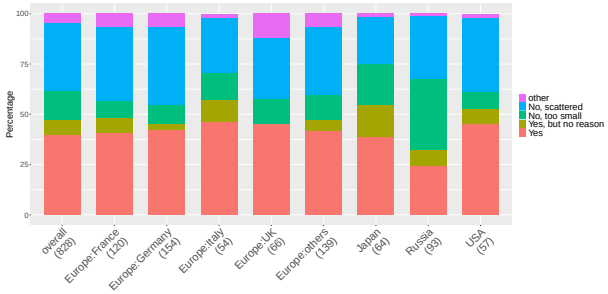

Figure 6: Q21: Layering MPI calls *(single)*

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 Features

The Q35 in the ECP survey and Q17 in our survey are equivalent questions, although the answer 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 categorized 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. This is clearly a concerning factor for the popularity of some of the MPI features as despite the 10-year existence many of the features failed to get any traction outside a small, certainly dedicated crowd.

The most notable difference between the two surveys relates to the use of datatypes (Table 6). The percentage of datatype us-

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	-	-	-

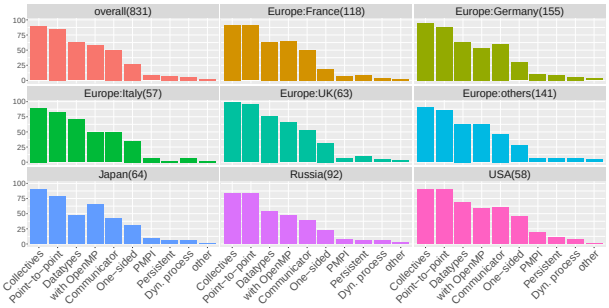

Figure 7: Q17: Using MPI Aspects *(multiple)*

age in the ECP was around 23% while in our survey it is significantly higher, at more than 60% in both overall and USA contributors. Looking at the USA data, as it is difficult to imagine that the common participants changed their mind between the two surveys, it seems that the datatype usage is more developed outside national laboratories. But at this point this conclusion is conjectural, more thorough analysis is needed to gain a better understanding.

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 a heatmap representing the cross-tab analysis between participants' MPI skills (Q3) and the knowledge or use of MPI features (Q16). 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. The result is interesting in the sense it goes against the expected outcome, where MPI experts will know and use more features. What we observe here is that the less used features in Fig. 7, PMPI, persistent, and dynamic process, are almost independent from the MPI skill.

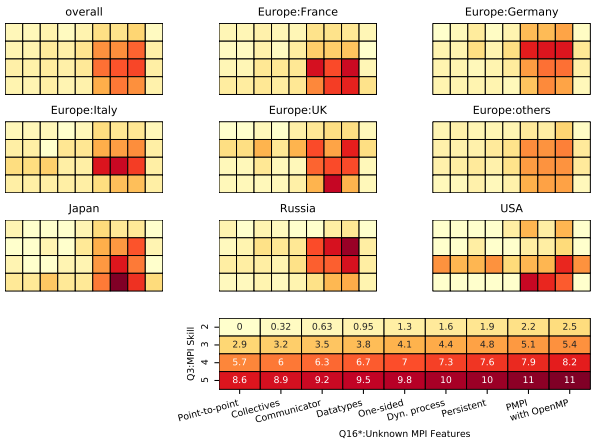

Figure 8: Q3-Q16: MPI Skill *(single)* and Unknown MPI Features *(multiple)*

A similar situation can be seen in the cross-tab analysis between Q6 and Q16 (Fig. 9). It would be natural to expect that the longer the MPI experience, the more familiar the participant should be with different MPI features and thus less unknown features. However, some major contributors (France, UK and Japan) show that in some cases there is no relationship between these two, and that a longer experience could evolve around the same, limited set of MPI features being used. This may also indicate that experienced MPI users may not easily catch up the newly introduced MPI features.

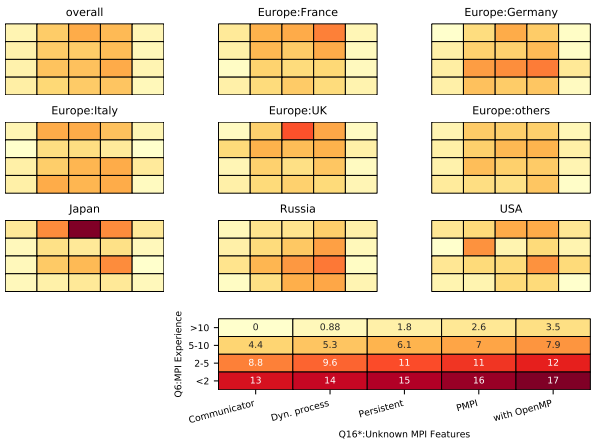

Figure 9: Q6-Q16: MPI Experience *(single)* and Unknown MPI Features *(multiple)*

This may indicate that the MPI standard is complex, and its understanding by the general developer population remains limited. Even the most basic send/receive functions, although their API looking simple and natural, require deep knowledge such as possibility of deadlock, timing of buffer access, blocking/non-blocking, and so on.

Fig. 10 represent the answers regarding the MPI features perceived as unnecessary by the participants (Q27: *What MPI feature(s) are NOT useful for your application?*). Although most participants believe MPI has little *unnecessary features*, a fair amount of participants seem convinced that the dynamic process features are not useful. There is certainly a correlation between this and the fact that dynamic process feature is not being used by the most participants (Q17, Fig. 7). It should be noted that this tendency is also reported in [6].

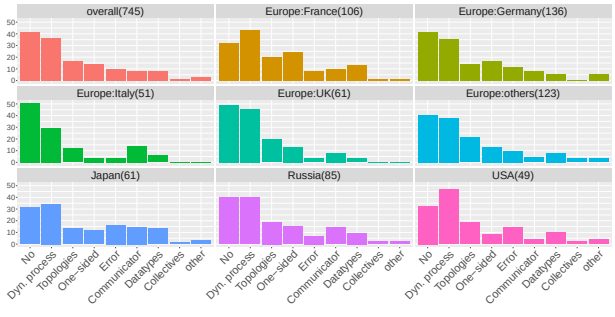

Figure 10: Q27: Useless Features *(multiple)*

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. But, we also need to look at applications and their demands. Most of the scientific applications look at the scientific process on a set of fixed boundaries and condition, and thus required a fixed number of processes in a completely static world, one that does not grow or shrink. Few applications exit this mold, and the small number of developers working around these applications seem to not have been reached by the survey.

4.3. Multi-threading

Similar to dynamic processes, the outcome of the question related to threading support in MPI has widely divergent answers between the two surveys. Fig. 11 shows the result of our survey and Table 7 shows the difference with the ECP survey. Note that our question is multiple-choice and the ECP question is single-choice. In both surveys, the percentage of using `MULTIPLE` is the highest among the valid choices, but the percentage of the choice *I don't know* remains the largest. This may sound contradictory because the ECP participants would be more experienced MPI users.

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.

Keeping in mind this question was a multiple-choice question Table 8 shows the top 7 raw answers (combined answers),

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

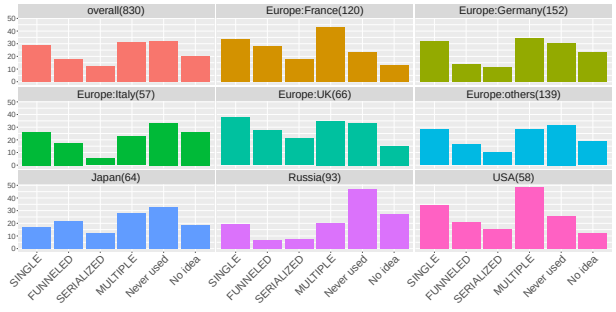

Figure 11: Q18: Multi-threading *(multiple)*

with a coverage of about 85% of the total answers. As nearly half participants answered *never used* or *no idea*, we will ignore these two choices in the remaining of this analysis. This is reflected in the numbers in parenthesis in this table which 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 willingly, which might indicate a well established knowledge of the MPI features.

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*

A similarly scattered trend has also been confirmed by other studies. Indeed, [9] indicates 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`. Similarly, [6] indicate that 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 usage of the different MPI implementations, (Q12 asking specifically which MPI implementation(s) the participants are using regularly). This result presents a coherent picture across the board, as Open MPI, Intel MPI and MPICH, dominates in all major contributors followed by MVAPICH. Outside these top contenders, a large disparity can be seen on the other implementations. Taking a look at the *other* choice, there are four (4) answers naming 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.

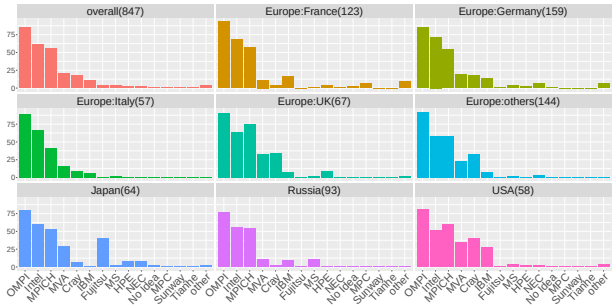

Figure 12: Q12: Using MPI Implementations *(multiple)*

In order to understand how the usage of a particular MPI implementation came to happen, we specifically asked the participants in Q13 *why did you choose the MPI implementation(s)* and the answers are shown in Fig. 13. One of the interesting outcomes from this question is the fact that more than half of the participants outside US have little to no choice in the selection of the MPI implementation, they have to use what is made available to them with the platform. For the rest of participants their choice seem to favor their familiarity with a particular implementation or past experiences with the community supporting their choice MPI implementation. This clearly suggest that MPI implementors must carefully build and support their users community in order to increase the usage of their particular implementation.

5.2. MPI+X and Alternatives

Fig. 14 shows the result of Q22 asking *Have you ever written MPI+"X" programs?*. As a constant across the board, most participants have experienced writing MPI+OpenMP programs. An interesting highlight, in US *CUDA* is the second largest and the percentage pure MPI applications is the lowest. Considering the low percentage of *No* in overall (approx. 25%), 3/4 participants are using MPI in combination with another, node-level or accelerator-focused, programming paradigm. A similar finding

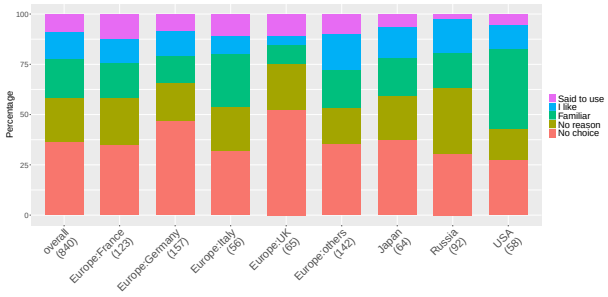

Figure 13: Q13: Choosing MPI Implementations *(single)*

was highlighted in [6], where it has been reported that the approximately 3/4 of the target programs use the hybrid model of MPI+OpenMP.

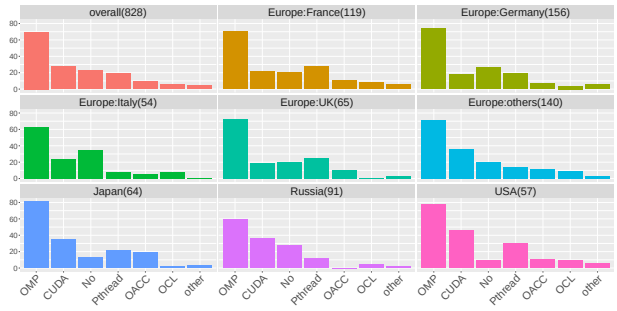

Figure 14: Q22: MPI+X *(multiple)*

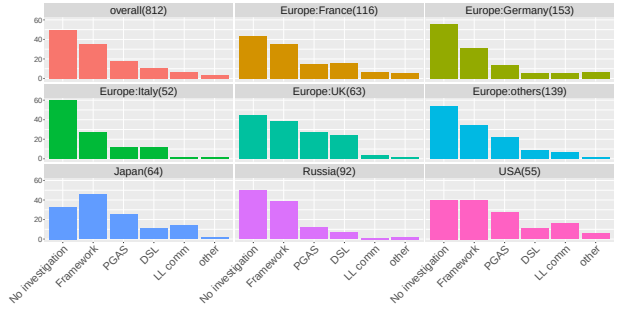

Figure 15: Q24: MPI Alternatives *(multiple)*

Without going in details about the features provided by MPI, it could be natural to assume that all types of data movements can be provided by other message passing paradigms. We specifically asked the participants to indicate if they have investigated any of these alternative message passing libraries (Q24 *What, if any, alternatives are you investigating to indirectly call MPI or another communication layer by using another parallel language library?*). The Fig. 15 highlights that almost half of the participants are totally satisfied with MPI, and have not investigated any replacement message passing paradigm. Out of the remaining participants, *PGAS* seems to be the most used alternative to MPI, a result that is similar across all major contributors.

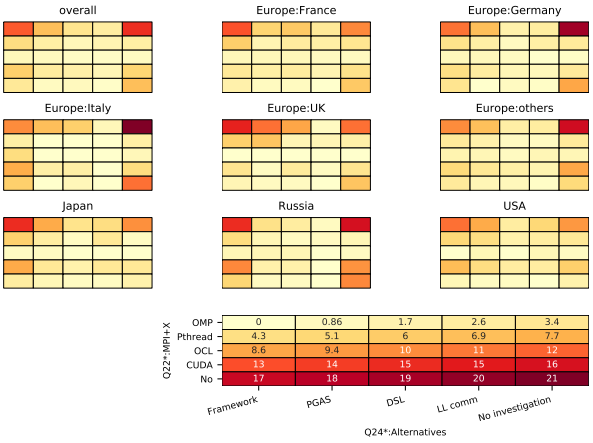

Figure 16: Q22-Q24: MPI+X *(multiple)* and MPI Alternatives *(multiple)*

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

5.3. Compatibility vs. Performance

In the history of MPI, portability, which translates into maintaining backward compatibility across versions, has been of paramount importance. From the MPI standard point of view it can also be an obstacle to the introduction of new features to enhance MPI capabilities, and to the deprecation of features that proved inconsistent or were replaced by better alternative. Fig. 17 shows the result of the question asking which is more important, performance or compatibility on a scale, while Fig. 18 shows the expressed need for backward compatibility (Q28).

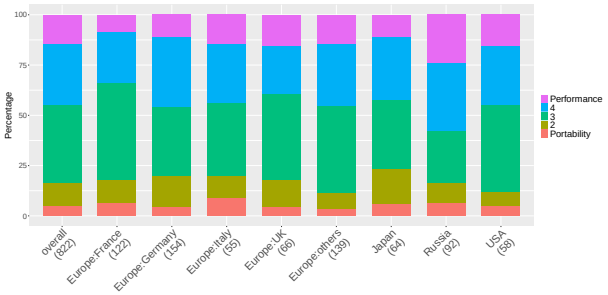

Figure 17: Q29: Performance vs. Compatibility *(single)*

In Fig 17, let's consider three groups; *performance group* focused on *Performance* or *4*, *compatibility group* focused on *Portability* or *2*), and *middle group* who chose *3*. While the middle group, striking a balance between performance and portability dominates in most major contributors (excepting Russia), the performance group tend to occupy a larger percentage. Regarding Russia, a large percentage of Russian participants answered *my program is too small* in Q21 (Fig. 6), in which case the lost of compatibility does not cause a considerable burden.

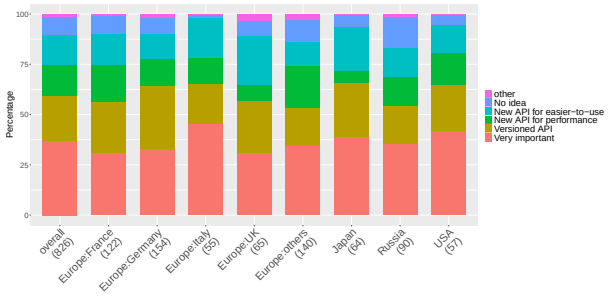

Figure 18: Q28: Backward Compatibility *(single)*

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 of Q28 may suggests that users would accept incompatibility in exchange of a substantiated benefit, either in terms of performance or productivity.

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 from online sources. The participants of Germany and other European countries preferred to have other form of 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 writing MPI applications.

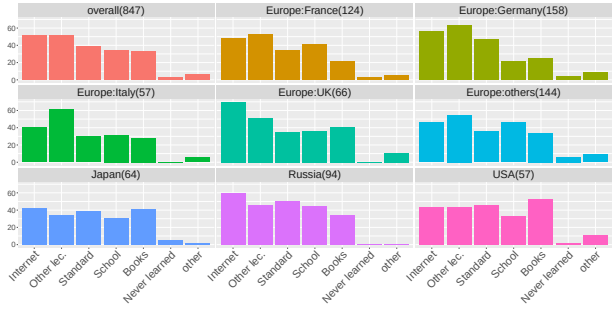

Figure 19: Q10: Learning MPI *(multiple)*

Fig. 20 shows the familiarity of the participants with the official MPI standard document (asking if participants have read the MPI standard document). Not necessarily surprising, around 60% of participants, independent from contributors, have partially read the MPI standard. Interestingly in UK, the percentage of participants having read the entirety of the MPI standard is similar to the percentage of users having never read the standard. In same time, UK participants overwhelmingly learned MPI from online sources, which usually translate by via practical examples.

While the MPI standard is certainly not the best document for learning MPI, it is the most valid and trusted source for

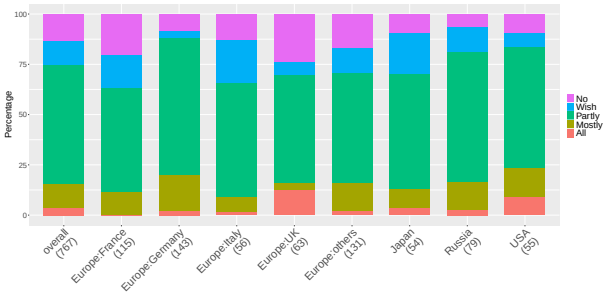

Figure 20: Q9: Reading MPI Standard *(single)*

checking the specification of the MPI API. 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.

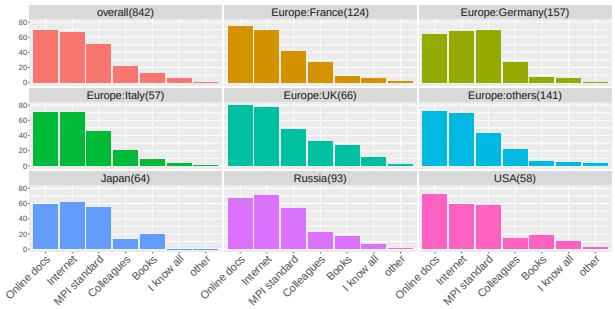

Figure 21: Q14: Checking Specification *(multiple)*

It would have been interesting 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 would not give us any clear evidence. Instead Fig. 22 presents a cross-tab analysis between Q3 and Q14. There can be seen a weak correlation, those who check more regularly the MPI standard have higher MPI skill, in some major contributors (France, Germany, and Japan).

Another interesting result can be seen in Fig. 23 asking *Rate your overall programming skill (non-MPI programs)*. People who auto-evaluate high their programming skills (basically those who chose 4 or more on the skills grade) account for more than 90%. This indicates that MPI users are seasoned developers, or at least programmers with high programming skills. This might indicate that MPI programming requires specific skills whom basic developers do not necessarily master, or that before starting to write parallel applications (where MPI is necessary) most developers have already become acquainted with programming. By contributor comparison, Russia shows a different tendency from other contributors.

Generally speaking, allowing participants to freely answer questions in text boxes lead to a large variety of disparate answers, making difficult to find commonality between mostly

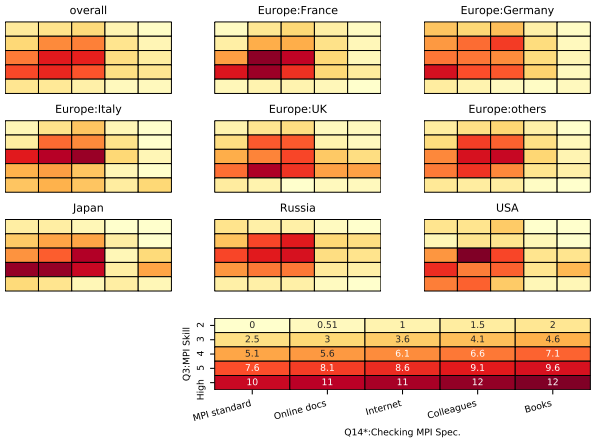

Figure 22: Q3-Q14: MPI Skill *(single)* and regular checking of the MPI Specification *(single)*

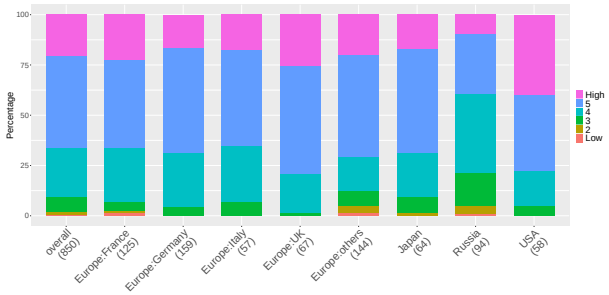

Figure 23: Q2: Rate your overall programming skill (non-MPI programs)

subjective answers and to put forward a consistent answer. Despite this, we had one particular question in our survey, where we felt that predefined answers would have been led to unsatisfactory results, and where more diverse information could be gathered with a combination of preselected answers and the opportunity to enter a different answer in a text box. This particular question, Q19, relates to *What are your obstacles to mastering MPI*, and is represented in Fig. 24. Although the largest answer was one of the provided choices, *No obstacles,* we got an exceptional 111 *other* inputs. After post-analysis, we can report that out of these, more than 20 participants, 18%, answered pinpointing to how time consuming master MPI is. Many other participants pointed out the need for more clear MPI programming guideline (*clear doc.*, *internal doc.*, *implementation doc.*, *performance guideline*, and so on, in their 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), highlight a strong correlation, from lower-left to higher-right, between those two questions regardless of major contributors. In fact, these graphs confirm a prior answer regarding mastering MPI, indicating that it takes more than somewhere between 5 and 10 years of MPI programming experience to reach a high MPI skill (*4* to *High*). The answer

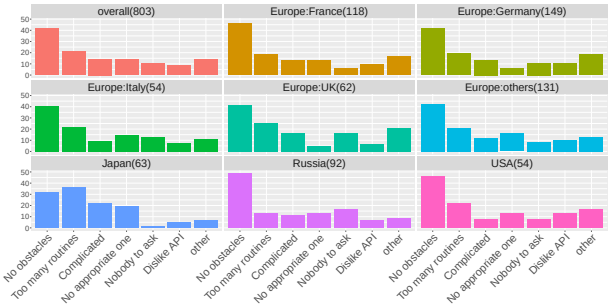

Figure 24: Q19: Learning Obstacles *(multiple)*

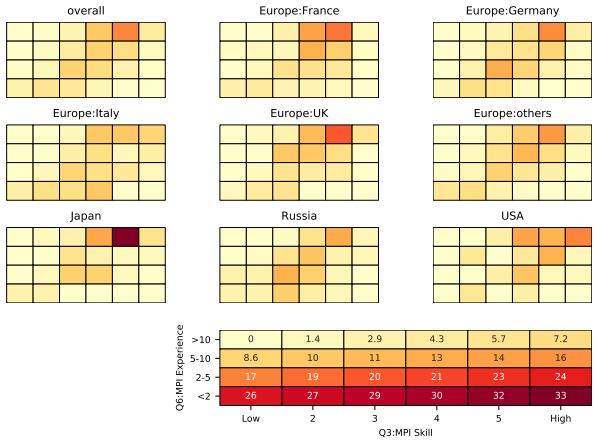

Figure 25: Q6-Q3: MPI Experience *(single)* and MPI Skill *single*

is confirmed across the entire spectrum and in most major contributors. Considering this fact and the nature and size of the MPI specification (Subsection 4.2), it is apparent that there is a, widely spread, belief that MPI could be said to be very difficult specification to master, and that the standardization body would need to put forward some serious efforts to facilitate the adoption and help MPI become more mainstream.

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 having selected *my MPI programs are well-tuned* account for only around 10%, with the exception of 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 the necessary resources to do that. In Japan, the percentage of well-tuned program is only few percent, highlighting the fact that as parallel machines become more complex, users are feeling that an increasing percentage of performance become unobtainable.

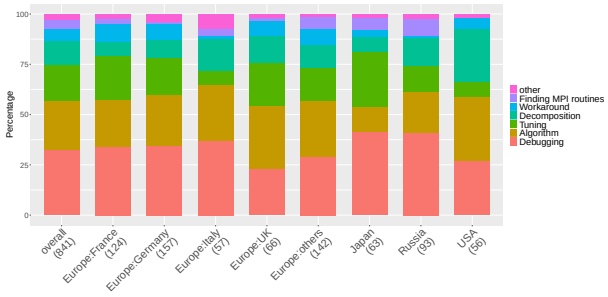

Figure 26: Q15: MPI Programming Difficulty *(single)*

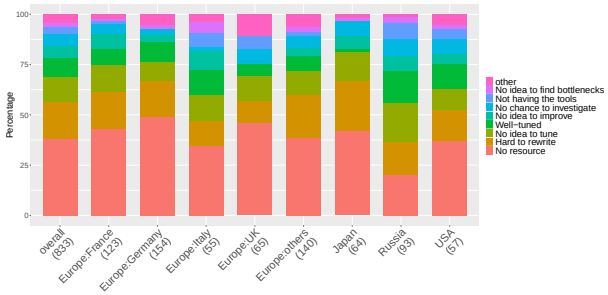

Figure 27: Q23: Performance Tuning *(single)*

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 asked *If there were one communication aspect which is not enough in the current MPI that 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).

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

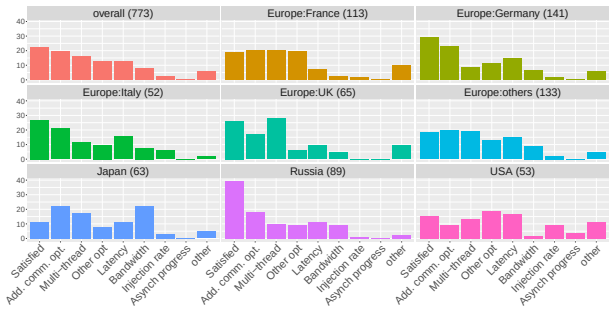

Figure 28: Q25: Features to improve *(single)*

Q26 is somewhat similar to Q25, but looking for more precise answers. This question tackles the issue on which semantic features are missing from MPI. Overall a very similar picture emerges with Q25, almost one third of the participants are satisfied with the existing MPI features. There is a high discrepancy between Japan, where users are the least satisfied with the current situation, and Russia which hosts the most satisfied MPI users. The situation here is similar to what we seen in Q25, with the highest answer being *Additional optimization opportunities in terms of communication (topology awareness, locality, etc.)*. Thus, it appears that efficiently managing 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 to note that most major contributors have expressed concerns about resilience, but we do not have enough information to understand the root cause. 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.

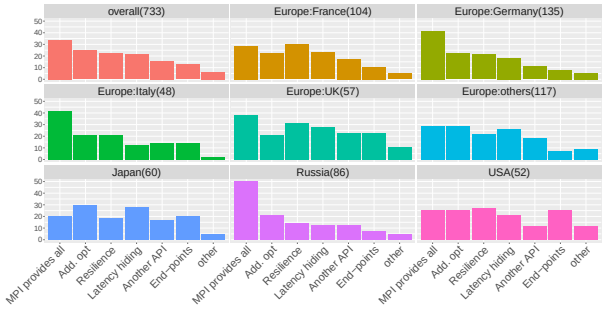

Figure 29: Q26: MPI Missing Semantics *(multiple)*

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 possibly imprecisely, what this feature was exactly about.

5.7. Notes on Contributors

In this subsection, we summarize our findings where some contributorshave showed somewhat different results than the others.

USA

US has the highest percentages: **a)** of high MPI skill (Fig. 4), **b)** of seasoned users, with more than 10 years of MPI experience (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, by some standards the most advanced.

Russia

Russia is: **a)** having the second largest percentage of MPI users with less than 5 years of MPI experience, a position it shared with Italy (Fig. 5), **b)** having the largest percentage of (non-MPI) programming beginners (Fig:23), **c)** having the highest percentage of users assessing 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 compared with 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 a high level of MPI skill (Fig. 4) and a long MPI experience (Fig. 5), many Japanese MPI users seem to be displeased with the current status of debugging and tuning (Fig. 26), whilst many participants of the other contributors are more concerned about *Algorithm*.

Most notably, more than 50% of Japanese MPI users have an extensive MPI experience, with more than 10 years. Having such a large mass of well seasoned MPI users sound promising, however, it might also point to an imbalance in generations of users, and to a potential lack of younger MPI developers that will continue the work in the future. Indeed, 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 might be in a difficult spot over the next decade.

6. Discussion

6.1. A constantly increasing standardization document

This survey reveals that some MPI features, which by most standards are not new being introduced almost a decade ago, are not yet widely accepted by MPI users (Subsection 4.2). An interesting question may be raised regarding the evolution of the gap between the MPI features defined by the standard and the acceptance of the features.

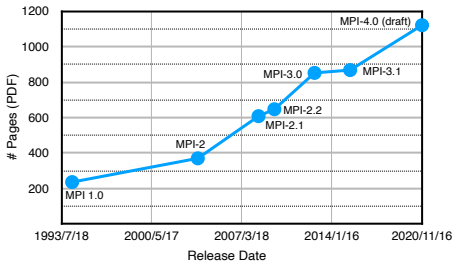

Figure 30: Page sizes of MPI Standards

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 with every new version of the standard. 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 by exploiting hardware resources without imposing significant effort on the MPI users. 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. Träff, et al., gives a formal analysis on this point [14]. This situation, introducing higher-level functions into the standard, will keep increasing the standard.

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. Recommendations for MPI Forum

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 kept in sych with today's web standards.

As already shown in Subsection 4.2, some, rather old, MPI features have failed to gain traction and be widely accepted by the users. Furthermore, as indicated in Subsection 5.4, many MPI users complain of a lack of time to hone or master MPI, and also to a lack of clear and easy understandable documentation. These suggest that there is a real difficulty for people to learn MPI and to write, efficient and error-free MPI programs.

However, this can be addresses with a stronger educational effort from the MPI standardization body. Indeed, 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

- **raise the bar on potential user adoption for all new features in order to slow the pace of introducing new features,** and
- **create a new working group focused on educational resources, and tasked to prepare and maintain web pages for tutorials, guidelines for MPI programming, and good (and certainly 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 have put forward few interesting findings regarding the current status of MPI adoption. As for the MPI features, the dynamic process feature is considered not only as a less-used feature but also mostly as 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, at least perceived as a very difficult-to-use library. Many MPI users point to a lack of documentation and would wish to have a practical programming guideline, online documents in hyper-text form, and useful sample programs, put forward and maintained by the MPI standardization committee. Most important (and most difficult) thing is those supplemental documents in any form must be up-to-dated and thorough. Regarding backward compatibility, many MPI users may accept to sacrifice some level of portability in exchange for more performance, an outcome at odds with the current thinking in the MPI Forum.

All collected answers, the programs to analyze the survey data and to generate graphs, and all published reports are freely 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], with additional funding from different natioanl science agencies.

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. 263–272.

[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-choice 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? **C**1) No, my MPI programs are well-tuned. [Well-tuned] **C**2) Yes, I know there is room for tuning but I should re-write large part of my program to do that. [Hard to rewrite] **C**3) Yes, I know there is room for tuning but I do not have enough resources to do that. [No resource] **C**4) I think there is room but I do not know how to tune it. [No idea to tune] **C**5) I do not have (know) tools to find performance bottlenecks. [Not having the tools] **C**6) I have no chance to investigate. [No chance to investigate] **C**7) I do not know how to find bottlenecks. [No idea to find bottlenecks] **C**8) I do not know if there is room for performance tuning. [No idea to improve] **C**9) Other

Q24*: What, if any, alternatives are you investigating to indirectly call MPI or another communication layer by using another parallel language/library? **C**1) A framework or library using MPI. [Framework] **C**2) A PGAS language (UPC, Coarray Fortran, OpenSHMEM, XcalableMP, ...). [PGAS] **C**3) A Domain Specific Language (DSL). [DSL] **C**4) Low-level communication layer provided by vendor (Verbs, DCMF, ...). [LL comm] **C**5) I am not investigating any alternatives. [No investigation] **C**6) 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? **C**1) Latency [Latency] **C**2) Message injection rate [Injection rate] **C**3) Bandwidth [Bandwidth] **C**4) Additional optimization opportunities in terms of communication (network topology awareness, etc.) [Additional comm. opt.] **C**5) Optimization opportunities except communication (architecture awareness, dynamic processing, accelerator support, etc.) [Other opt.] **C**6) Multi-threading support [Multi-thread] **C**7) Asynchronous progress [Asynch progress] **C**8) MPI provides all semantics I need [Satisfied] **C**9) Other

Q26*: Is MPI providing all the communication semantics required by your application? If not, what is missing? **C**1) Latency hiding (including asynchronous completion) [Latency hiding] **C**2) Endpoints (multi-thread, sessions) [End-points] **C**3) Resilience (fault tolerance) [Resilience] **C**4) Additional optimization opportunities in terms of communication (topology awareness, locality, etc.) [Additional opt] **C**5) Another API which is easier and/or simpler to use [Another API] **C**6) MPI is providing all the communication semantics required by my application [MPI provides all] **C**7) Other

Q27*: What MPI feature(s) are NOT useful for your application? **C**1) One-sided communication [One-sided] **C**2) Datatypes [Datatypes] **C**3) Communicator and group management [Communicator] **C**4) Collective operations [Collectives] **C**5) Process topologies [Topologies] **C**6) Dynamic process creation [Dyn. process] **C**7) Error handlers [Error] **C**8) There are no unnecessary features [No] **C**9) Other

Q28: Do you think the MPI standard should maintain backward compatibility? **C**1) Yes, compatibility is very important for me. [Very important] **C**2) API should be clearly versioned. [Versioned API] **C**3) I prefer to have new API for better performance. [New API for performance] **C**4) I prefer to have new API which is simpler and/or easier-to-use. [New API for easier-to-use] **C**5) I do not know or I do not care. [No idea] **C**6) 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: Contributors

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