Parallel Computing 108 (2021) 102853

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

An international survey on MPI users

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

[a] Riken Center for Computational Science, 7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, 650-0047, Japan
[b] INRIA, LaBRI, Univ. Bordeaux, 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
[d] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430, Japan
[e] Fujitsu Limited, 9-3, Nakase 1-chome, Mihama-ku, Chiba, 261-8588, Japan

ARTICLE INFO

MSC:
68-02

Keywords:
Message Passing Interface (MPI)
Survey

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 adapt accordingly to continue to efficiently bridge the gap between users and hardware. This questionnaire survey was conducted using two online questionnaire frameworks and has gathered more than 850 answers from 42 countries since February 2019. Some of preceding surveys of MPI uses are questionnaire surveys like ours, while others are conducted either by analyzing MPI programs to reveal static behavior or by using profiling tools to analyze the dynamic runtime behavior of MPI jobs. Our survey is different from other questionnaire surveys in terms of its larger number of participants and wide geographic spread. 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, compare the results with preceding studies when possible, and provide some recommendations for MPI Forum based on the findings.

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 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. 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, which connects major supercomputers owned by universities and governmental research institutes. If both questionnaire surveys would have had the same questions, we could have compared the answers to reveal the differences between US and Japan MPI user communities. Unfortunately, only 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 communities of MPI users. Unlike earlier studies, 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 is 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, where most co-authors are active participants. 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

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

* Corresponding author.
E-mail addresses: ahori@nii.ac.jp (A. Hori), emmanuel.jeannot@inria.fr (E. Jeannot), bosilca@icl.utk.edu (G. Bosilca), t.ogu@fujitsu.com (T. Ogura), bgerofi@riken.jp (B. Gerofi), yinj@nii.ac.jp (J. Yin), yutaka_ishikawa@nii.ac.jp (Y. Ishikawa).

https://doi.org/10.1016/j.parco.2021.102853
Received 19 February 2021; Received in revised form 18 June 2021; Accepted 16 September 2021
Available online 2 October 2021

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

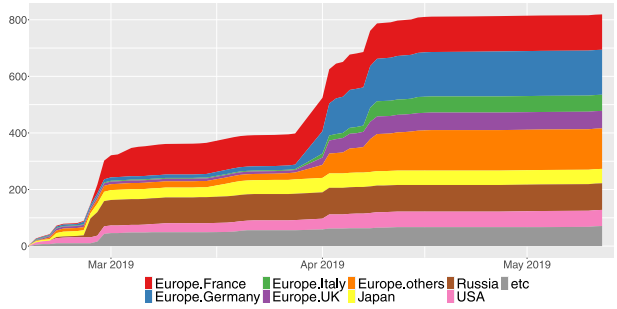

Fig. 1. Time series in first 90 days.

2. Related work

The existing MPI-related surveys can be categorized in three survey classes;

Questionnaire (target: MPI users) Questionnaire surveys asking MPI users questions specifically crafted toward a target goal and reflecting more the human understanding or knowledge of MPI capabilities.

Static Analysis (target: MPI programs) Application-oriented statistical surveys statically analyzing MPI programs and classifying occurrences of each MPI call.

Runtime Analysis (target: MPI jobs) Application-oriented statistical surveys analyzing the behavior of MPI applications at runtime by using a profiling tool.

Our survey and the ECP survey are examples of the *Questionnaire* 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 becoming known by the users community, well before they start appearing in MPI applications.

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

In the *Runtime Analysis* category, Chunduri et al. [9] collected and analyzed the runtime behavior by running more than 100 K MPI jobs, with a smaller but still significantly distinct number of different applications. Klenk et al. [10] took 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 two studies is significantly different, the second one looking at applications developed by a user community more inclined to use advanced features of MPI.

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

Prof. Poole and Ahmed, our consulting social scientists, suggested that the number of questions must be limited to around 30, to keep participants engaged and not to lose 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 easily as possible, and the questions which force participants to do 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 CSV 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 beginning February 17, 2019 and the most recent answer was June, 2020. 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

One of the first challenges we had to consider was how to reach a largely international community of researchers and users quickly and efficiently while expecting significant contributions and feedback. The survey was initially announced via several major mailing lists in the community (`hpc-announce` for example), 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 in top 11 countries (all countries are listed in Appendix B). The three major countries in Top500, USA, China and Japan, are not even in the top four in our survey. China, which, according to Top500, holds the most compute platforms among all countries, had only 18 participants including Taiwan (two). We tried to increase the number of participants from these countries as much as we could, making and distributing flyers at several conferences, with little positive outcome. While the root cause is still unclear, this pinpoints 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**

Table 2
Our Contributors and Top500 Performance Share.

Contributor	Top 11 contributors in our survey			Top500 performance share	
	Rank	#Ans	[%]	Rank	[%]
Germany	1	159	18.7	4	5.4
France	2	125	14.7	5	3.7
Russia	3	94	11.1	18	0.4
UK	4	67	7.9	8	1.4
Japan	5	64	7.5	2	24.4
USA	6	58	6.8	1	27.5
Italy	7	57	6.6	6	3.2
Switzerland	8	40	5.8	10	1.1
South Korea	9	27	3.2	13	0.8
Austria	10	26	3.1	27	0.1
China (+Taiwan)	11	18	2.1	3	23.3
42 contributors, 851 participants				(Nov. 2020)	

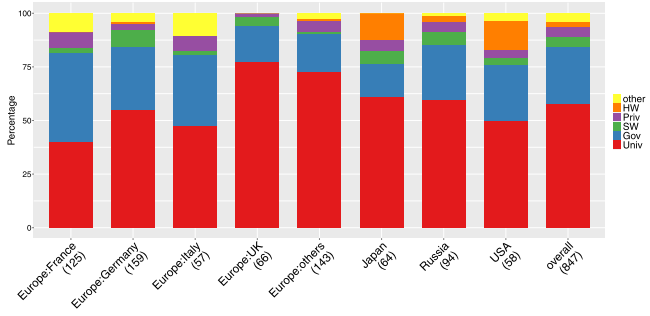

Fig. 2. Q1: Occupations *(single)*. *HW*:Hardware vendor, *Priv*:Private research institute, *SW*:Software vendor, *Gov*: Governmental institute, and *Univ*: College/University.

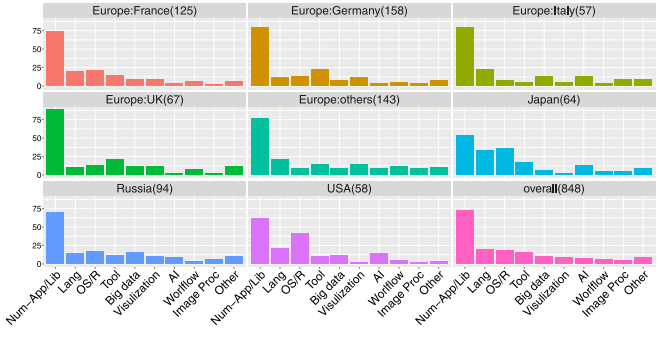

Fig. 3. Q7: Working fields *(multiple)*.

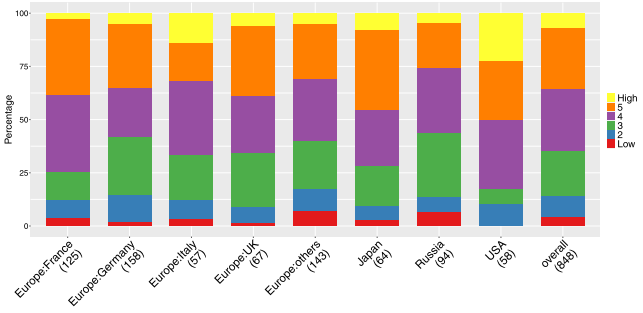

Fig. 4. Q3: Self assessment of MPI skill *(single)*.

contributors and are the object of cross-tab analysis. Such major contributors are Germany, France, Russia, UK, Japan, USA, Italy and the rest of the European countries (denoted as "*Europe others*". Table B.5 lists all of them.). It should be noted that the information used to define the major contributor was not the nationality of the individual participants, but rather the respondents' workplace in the last five years. The threshold of 50 participants was selected so that USA and Japan which have the big performance share in Top500 can be the major contributors (Table 2).[1]

The larger number of participants in our survey enables us to conduct cross-tab analysis between two questions to identify any correlation between the two questions. A number of cross-tab heatmap graphs of every possible combination of the two questions in our questionnaire survey are generated by our developed Python program and analyzed. The cross-tab graphs showing obvious correlations will be shown in this paper. It was very unfortunate that the number of such meaningful cross-tab graphs was surprisingly small.

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. There are two type of questions in our survey, single-answer and multiple-answer questions. This is single-answer question. Hereinafter "*(single)*" or "*(double)*" is denoted to distinguish them.

Fig. 3 shows the major field the participants are involved with; participants selected the field from 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 two major contributors, Japan and US, the percentage of parallel languages and OS/runtimes participants is significantly higher compared with the rest of major contributors.

4. Comparison with the ECP survey

Although the ECP questionnaire and our questionnaire were designed independently, there are several comparable questions. We are going to clarify some points about the profiles of the participants in our survey. Due to overlapping survey propagation means (emails, HPC-related mailing lists, and word-of-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, due to the larger ratio of participants from universities and national laboratories, it seems likely that the ECP survey contains more answers from expert MPI users or highly HPC-centered participants. Fig. 4 shows the results of the self-evaluation of the participants' MPI skill in our survey. It is worth raising attention to the US case (rightmost bar), where almost half of participants rate themselves as highly skilled MPI users (*5* or *High*), significantly ahead of any other major contributors. Additionally, none of the US participants indicated a low MPI-related skill.

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 two years MPI experience, but they do not rank their MPI expertise the lowest (Fig. 4). Japan followed closely and has the highest percentage in participants with more than ten years of

[1] The sampling number of 50 is a little more than that of the case satisfying 80% confident level and 20% error margin.

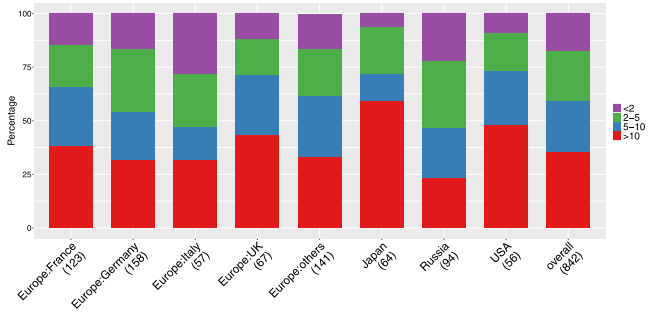

Fig. 5. Q6: MPI experience *(single)*.

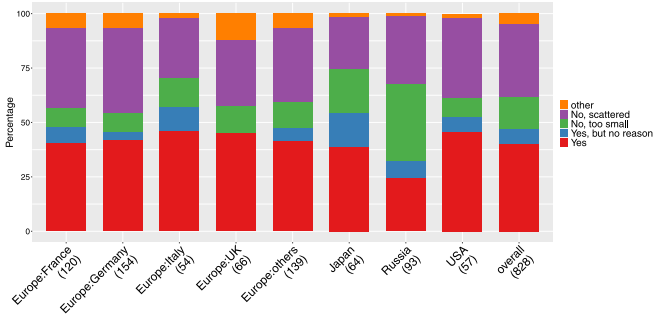

Fig. 6. Q21: Layering MPI calls *(single)*.

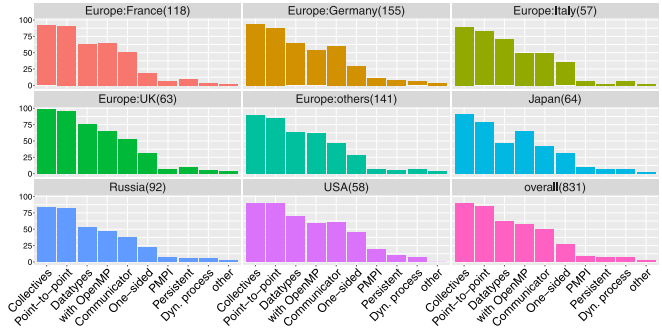

Fig. 7. Q17: Using MPI aspects *(multiple)*.

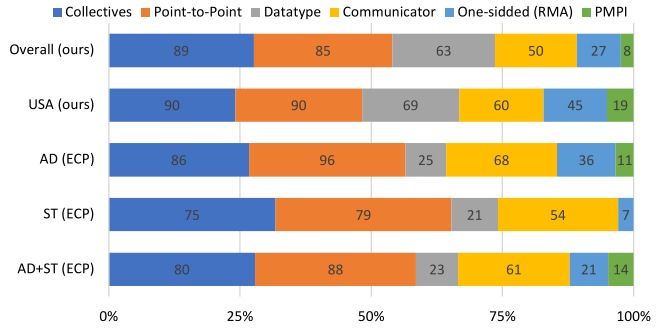

Fig. 8. Using MPI aspects.

experience and also the lowest percentage in those with less than two years experience. Russia, followed by Italy, has the highest percentage in less than five years experience (including the less than two years experience case).

A second set of questions (Table 3) 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 4 focuses on the comparison between our survey 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 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 Fig. 8 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 released 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 (Fig. 8). The percentage of datatype usage 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 participants are unlikely to change their minds between the two surveys, it seems that the datatype usage is more developed outside national laboratories. At this point, this conclusion is conjectural as more thorough analysis is needed to gain a better understanding.

Fig. 9 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 (The legend combined with a color bar can be found at the bottom of the figure. The numbers in the legend cells are percentages). Less frequent rows (one is the lowest skill and six is the highest skill) in this figure are omitted to increase readability. The result is interesting in the sense it contradicts 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.

A similar situation can be seen in the cross-tab analysis between Q6 and Q16 (Fig. 10). It would be natural to expect that the longer the MPI experience, the more familiar the participant should be with different MPI features and encounter fewer 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.

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 looks simple and natural, require deep knowledge of topics such as the possibility of deadlock, timing of buffer access, blocking/non-blocking, and so.

Fig. 11 represents the answers regarding the MPI features perceived as unnecessary by the participants (Q27: *What MPI feature(s) are NOT*

Table 3
Comparable questions.

Topic	Our Survey	ECP Survey
Layering MPI calls (Section 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 (Section 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 (Section 4.3)	Q18: Which MPI thread support are you using? *(multiple)*	Q59: Which MPI threading option are you using? *(single)*

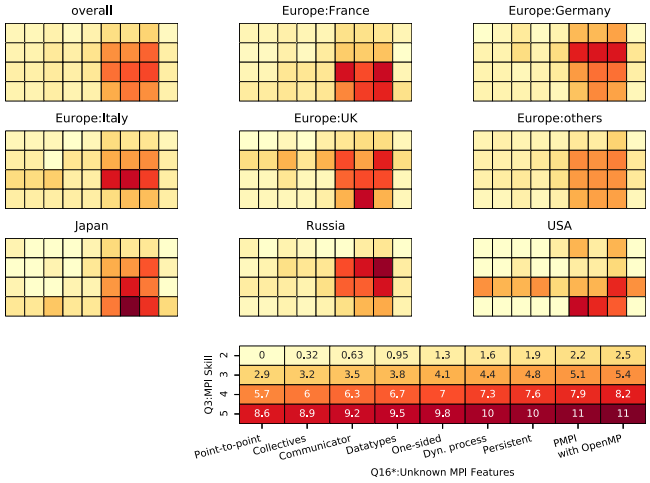

Fig. 9. Q3–Q16: MPI skill *(single)* and unknown MPI features *(multiple)*.

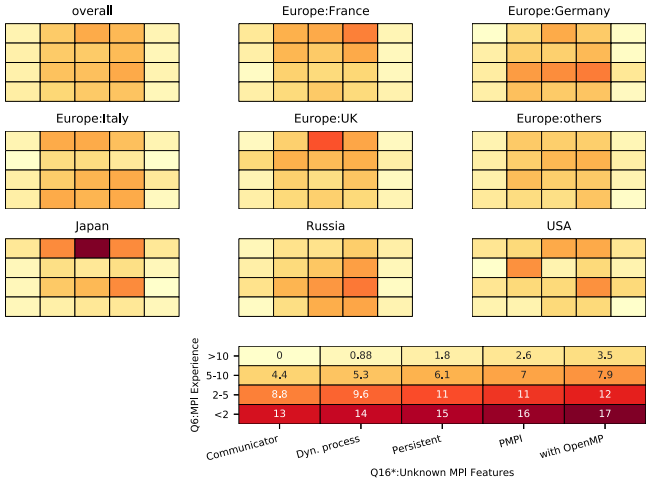

Fig. 10. Q6–Q16: MPI experience *(single)* and unknown MPI features *(multiple)*.

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 features are not being used by most participants (Q17, Fig. 7). It should be noted that this tendency is also reported in [6].

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 created (MPI) processes must

Table 4
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	–	–	–

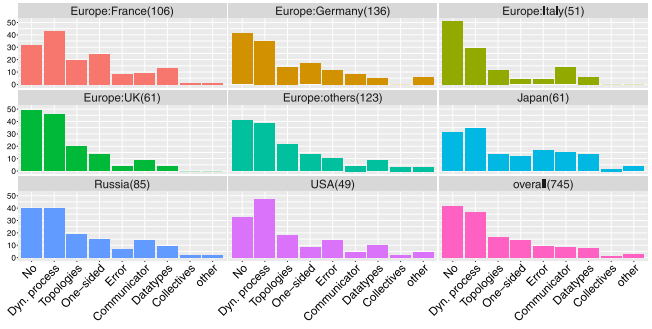

Fig. 11. Q27: Useless features *(multiple)*.

be defined in the standard. Indeed, the implementation of dynamic process creation involves many parts of a computing system: MPI library, process manager, job scheduling system and system operation. 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 conditions, 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 received widely divergent answers between the two surveys. Fig. 12 shows the result of our survey and Fig. 13 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 *No idea* 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. 12). France and Germany have the same trend. In

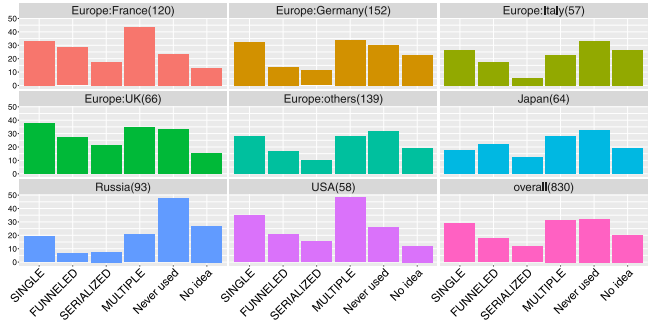

Fig. 12. Q18: Multi-threading *(multiple)*.

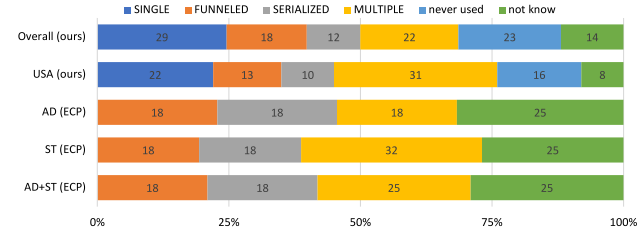

Fig. 13. Multi-threading comparison (ECP does not have the choices *SINGLE* and *never used*).

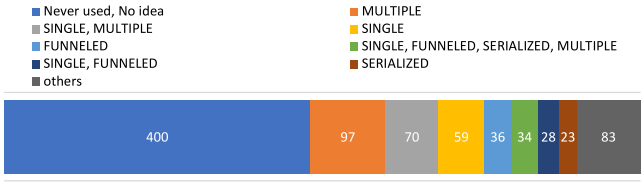

Fig. 14. Multi-threading — Raw answers.

Italy, Japan, Russia and the other European countries, the percentages of *No idea* are the highest. In UK, the percentage of using `SINGLE` is the highest.

Keeping in mind this question was a multiple-answer question Fig. 14 shows the top seven raw answers (combined multiple answers, excluding *never user* and *no idea*), with a coverage of about 85% of the total answers. As nearly half of participants answered *never used* or *no idea*, we will ignore these two choices in the remainder 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.

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 determine a clear result.

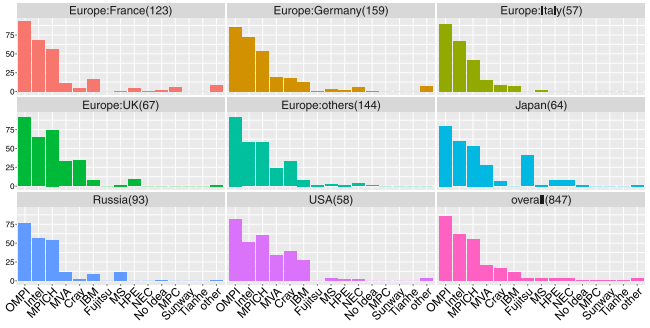

Fig. 15. Q12: Using MPI implementations *(multiple)*.

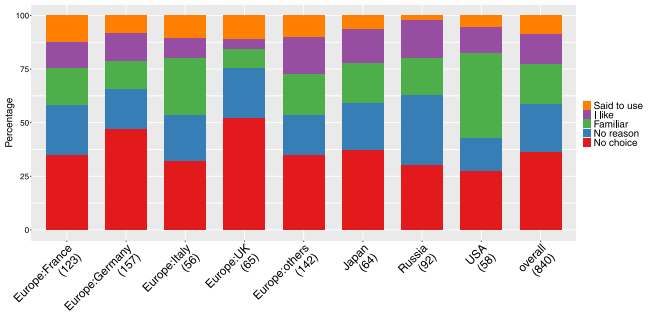

Fig. 16. Q13: Choosing MPI implementations *(single)*.

5. Other findings

5.1. MPI implementations

Fig. 15 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 for 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 [11] 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.

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. 16. 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 suggests that MPI implementors must carefully build and support their user communities in order to increase their implementation adoption.

5.2. MPI+X and alternatives

Fig. 17 shows the result of Q22 asking *Have you ever written MPI+"X" programs?*. As a constant across the board, most participants have experience with writing MPI+OpenMP programs. An interesting highlight, in US *CUDA* is the second largest and the percentage of pure MPI applications is the lowest. Considering the low percentage of *No* in overall (approx. 25%), 3/4 participants are using MPI in combination

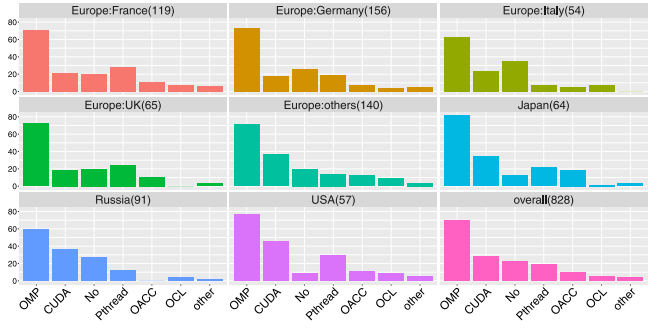

Fig. 17. Q22: MPI+X *(multiple)*.

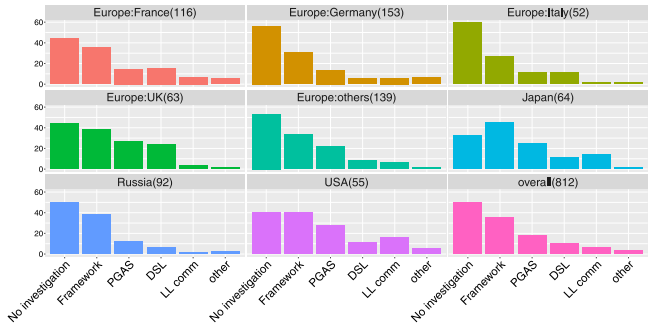

Fig. 18. Q24: MPI alternatives *(multiple)*.

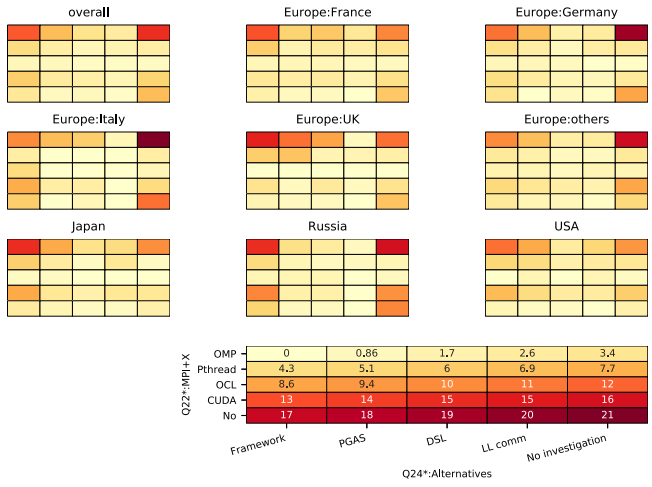

Fig. 19. Q22–Q24: MPI+X *(multiple)* and MPI alternatives *(multiple)*.

with another node-level or accelerator-focused programming paradigm. A similar finding 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.

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. 18 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.

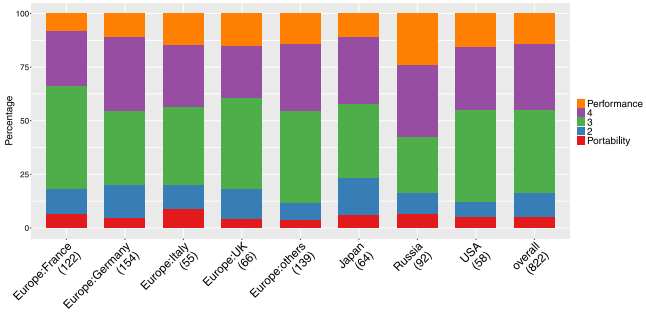

Fig. 20. Q29: Performance vs. Compatibility *(single)*.

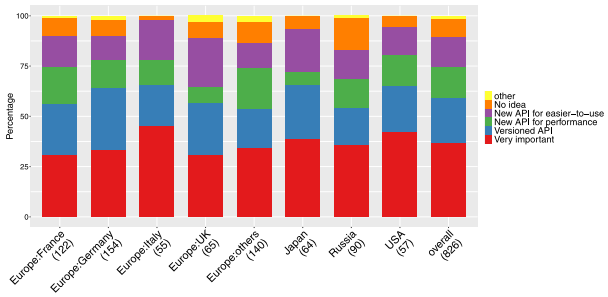

Fig. 21. Q28: Backward compatibility *(single)*.

Fig. 19 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 an MPI alternative (upper right corners of the heatmaps).

5.3. Compatibility vs. Performance

From the MPI standard point of view, the backward compatibility 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 a better alternative. Fig. 20 shows the result of the question asking which is more important on a scale, performance or compatibility, while Fig. 21 shows the expressed need for backward compatibility (Q28).

In Fig. 20, let us consider three groups; "performance group" focused on *Performance* or *4*, "compatibility group" focused on *Portability* or *2*), and "middle group" that chose *3*. While the middle group, striking a balance between performance and compatibility dominates in most major contributors (except 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 loss of compatibility does not cause a considerable burden.

As shown in Fig. 21, around 40% of participants answered that the compatibility is very important, while the rest of participants may accept the incompatibilities conditionally. The incompatibility forces users to update their programs. The result of Q28 may suggest that users would accept incompatibility in exchange for a substantiated benefit, either in terms of performance or productivity.

5.4. Learning MPI

Fig. 22 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

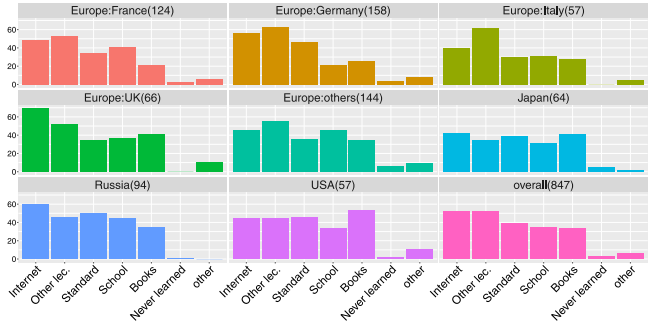

Fig. 22. Q10: Learning MPI *(multiple)*.

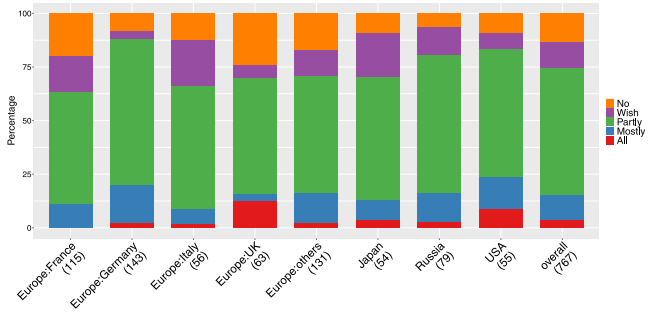

Fig. 23. Q9: Reading MPI standard *(single)*.

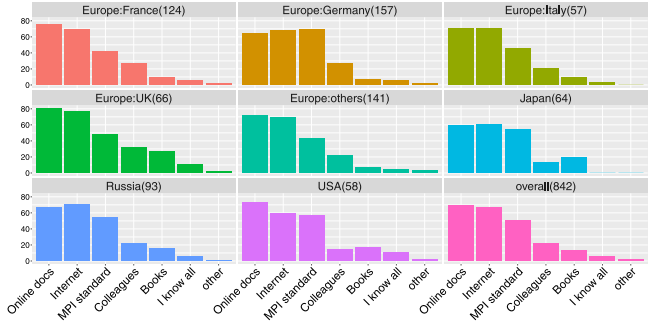

Fig. 24. Q14: Checking specification *(multiple)*.

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.

Fig. 23 shows the familiarity of the participants with the official MPI standard document by 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 and the percentage of users having never read the standard are the highest among the major contributors. At the same time, UK participants overwhelmingly learned MPI from online sources, which usually translates 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 checking the specification of the MPI API. Fig. 24 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 the Internet, and reading the standard. As shown in the previous figure (Fig. 23), users are reading the standard partly because of checking the MPI specifications.

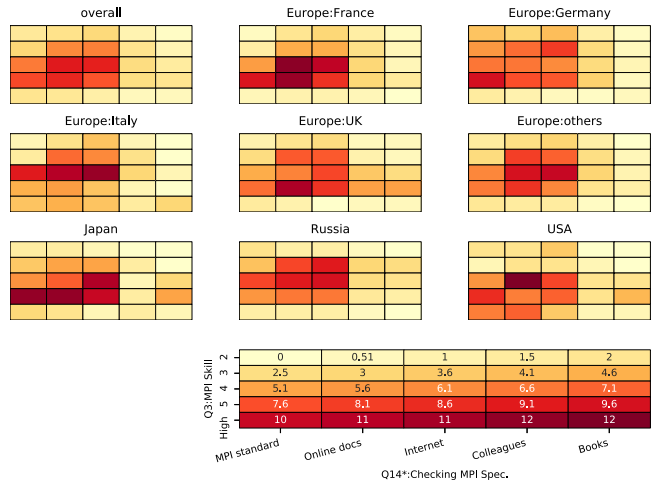

Fig. 25. Q3–Q14: MPI skill *(single)* and regular checking of the MPI Specification *(single)*.

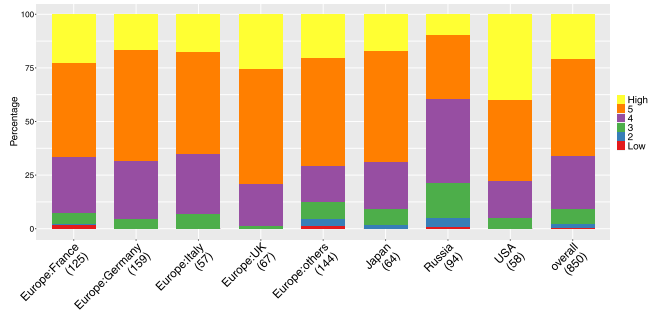

Fig. 26. Q2: Rate your overall programming skill (non-MPI programs).

It would have been interesting to have the cross-tab analysis between Q3 (MPI skill) and Q9 (reading MPI standard). Unfortunately the participants partly reading the standard dominates and the cross-tab analysis would not give us any clear evidence. Instead, Fig. 25 presents a cross-tab analysis between Q3 and Q14. There is a weak correlation in those who check the MPI standard more regularly and have higher MPI skills in some major contributors (France, Germany, and Japan).

Another interesting result can be seen in Fig. 26, which asks participants to *Rate your overall programming skill (non-MPI programs)*. People who auto-evaluate their programming skills higher (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 which many 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 leads to a large variety of disparate answers, making it difficult to find commonality between mostly 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 led to unsatisfactory results; 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. 27. Although the largest answer was one of the provided choices, *No obstacles,* we got an exceptional 111 *other* inputs.

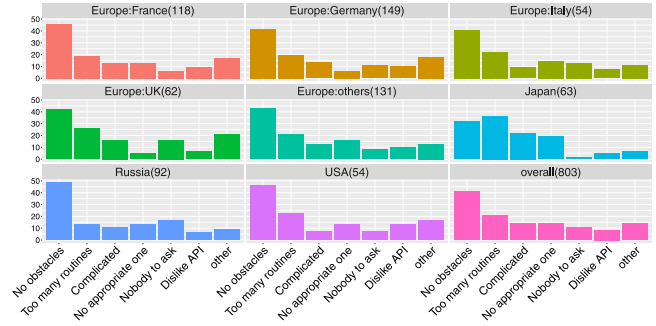

Fig. 27. Q19: Learning Obstacles *(multiple)*

Fig. 28. Q19: Learning obstacles — Text mining of the *Other* free text.

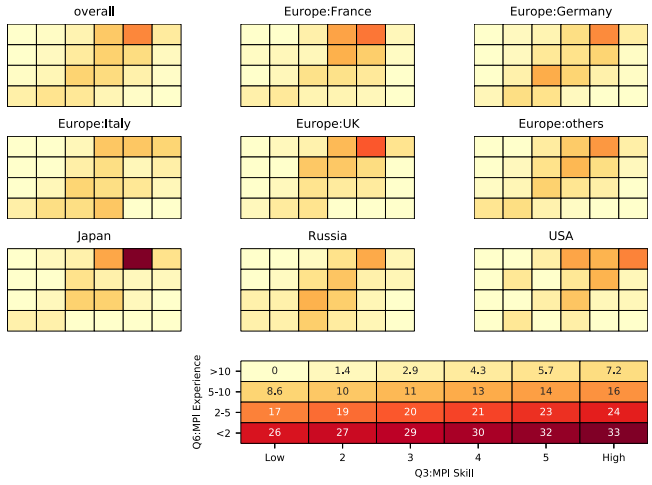

Fig. 29. Q6–Q3: MPI experience *(single)* and MPI skill *single*.

After post-analysis, we can report that out of these, more than 20 participants, 18%, answered pinpointing to how time consuming mastering MPI is. Many other participants pointed out the need for more clear MPI programming guidelines (*clear doc.*, *internal doc.*, *implementation doc.*, *performance guideline*, and so on, in their words), while some participants complained about MPI implementations and the performance or specification differences among implementations. Fig. 28 shows the text mining result of the free text inputs (using WordCloud[2] [12]). As shown, "lack (of) time" is most highlighted. 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 complaint about MPI implementations and the (performance or specification) differences among implementations.

As shown in Fig. 29, 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 is confirmed across the entire spectrum and in most major contributors. Considering this fact and the nature and size of the MPI specification (Section 4.2), it is apparent that there is a widely spread belief that MPI could be said to be a very difficult specification to master and that the standardization body would need to put forward significant efforts to facilitate the adoption process and help MPI become mainstream.

[2] WordCloud parameters: `stopwords=None, min_word_length=4, collocations=True, collocation_threshold=5, max_words=30`

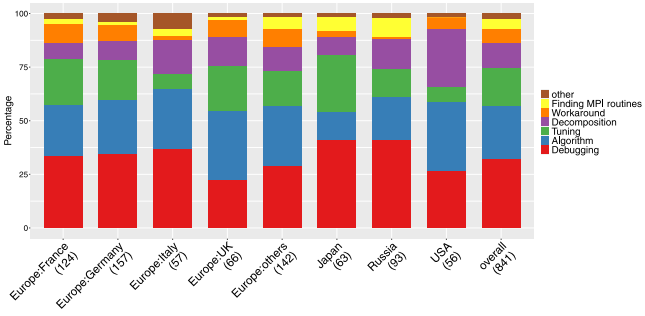

Fig. 30. Q15: MPI programming difficulty *(single)*.

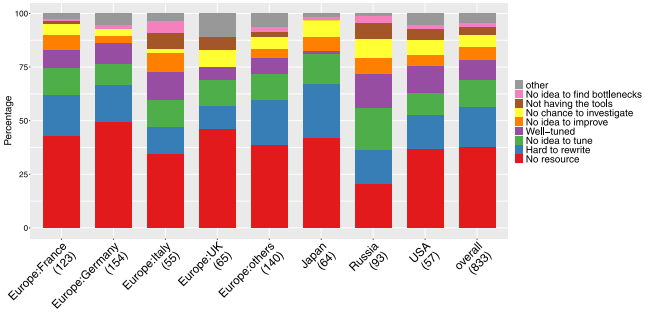

Fig. 31. Q23: performance tuning *(single)*.

5.5. MPI programming difficulty and tuning

Fig. 30 shows the result of Q15 asking *What is the most difficult part of writing an MPI program?* and Fig. 31 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. 31 has more divergence than Fig. 30. 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 lots of potential for tuning 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 programs is only a few percent, highlighting the fact that as parallel machines become more complex, users are feeling that increasing performance becomes unobtainable.

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. 32), and Q26 asks, *Is MPI providing all the communication semantics required by your application? If not, what is missing?* (Fig. 33).

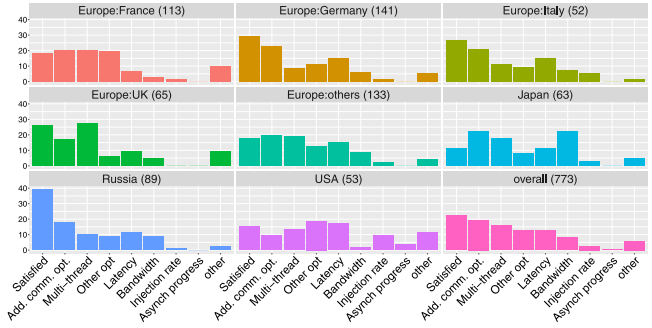

Fig. 32. Q25: Features to improve *(single)*.

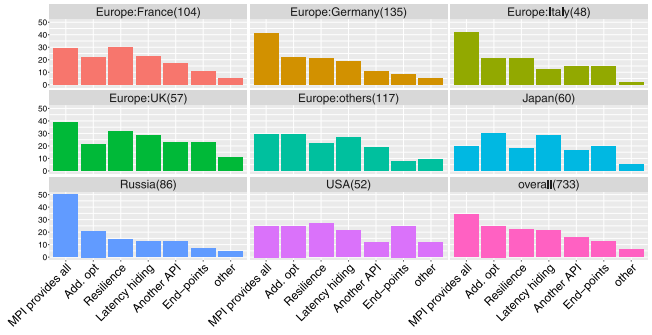

Fig. 33. Q26: MPI missing semantics *(multiple)*.

Fig. 32 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.)*.

Q26 is somewhat similar to Q25 but looking for more precise answers. This question addresses the issue on which semantic features are missing from MPI. Overall a picture very similar to Q25 emerges, as 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 have 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 to be 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 disparities in the satisfaction (answering *MPI provides all*), the disparities among 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 its intricate evolution in the standard, it might be possible that most people answering this question know little about this feature.

5.7. Notes on Contributors

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

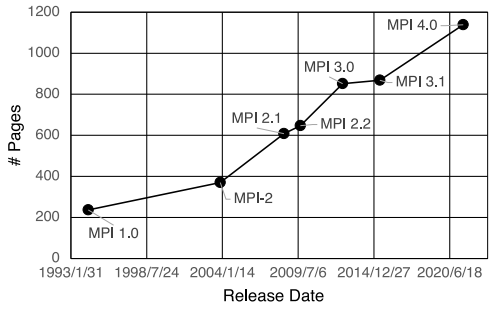

Fig. 34. Page sizes of MPI standards.

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. 13); **(d)** choosing familiar MPI implementations (Fig. 16); and **(e)** reading MPI books (Fig. 22) among the major contributors. All these results indicates that the MPI users in US are, in a sense, 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 shares with Italy (Fig. 5); **(b)** having the largest percentage of (non-MPI) programming beginners (Fig. 26); **(c)** having the highest percentage of users assessing their MPI programs are well-tuned (Fig. 30); **(d)** having the highest percentage not knowing which thread level they are using (Fig. 13); and **(e)** the second highest contributor, next to US, choosing the MPI+CUDA (Fig. 17).

These findings may indicate that Russia is relatively younger in terms of MPI usage compared with the other major contributors. The high focus 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. 30), 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 sounds 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 the 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 adopted by MPI users (Section 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.

Fig. 34 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 a higher 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, were provided by the MPI standard users would have to write their own higher-level capabilities to cover other useful parallel constructs, such as collective functions, a difficult and challenging task even for experienced developers.

Another reason for the inflation is that MPI standard is the standard as a library. There is no way for the implementation of low-level communication procedures to know whether they are employed as part of a higher-level data exchange pattern. The higher abstracted functions can give a library more information of the higher-level information and thus the higher-level functions can be optimized. Träff, et al. gives a formal analysis on this point [14]. This situation, to introduce higher-level capabilities 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 to introduce new functions but to narrow the capabilities 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 sync with today's web standards.

As already shown in Section 4.2, some rather old MPI features have failed to gain traction and be widely adopted by the users. Furthermore, as indicated in Section 5.4, many MPI users complain of a lack of time to hone or master MPI and of a lack of clear and understandable documentation. These findings 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 addressed 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 nor sufficient. To narrow the adoption gap, the 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 to make sure these new additions are needed by the majority of MPI user community, 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.

6.3. Lessons learned

Design Strategy Striking a balance between the number of questions and their coverage. We followed the advice of social scientists to maintain only 30 easy-to-answer questions, a number much smaller than prior surveys (Section 1). The "no answer" rate of each question varies from zero (Q2) to 22% (Q11, asking reading MPI books), and is only 2% for the last question. This may indicate that the questionnaire design was successful in maintaining the participants involved. The drawback is a reduced coverage of the wide spectrum of the MPI topics.

Questionnaire Design In general, when designing a good questionnaire, designer must have some insights beforehand how the participants will react. However, this is very difficult in most cases and this survey is no exception. In retrospect, Q11 asking reading MPI books was the most useless, since we could not conclude any meaningful results from it. Conversely we should have had a question asking age ranges of participants. This would have revealed the generation distributions of contributors.

Reaching Target Community First, the means (mailing lists, MLs) for announcing a survey must be selected in such a way that the target community must be in reach. In many cases, major mailing lists, must be complemented with more local, as an example university based, where the traffic is more limited and users more attentive to the content of the messages.

Online Forms Unfortunately, even scientific surveys are subject to geopolitical restrictions. Surveys with a worldwide scope must ensure the hosting platform can be accessed freely without difficulties or restrictions from all locations. Otherwise the number of answers might be biased.

7. Summary

We have conducted a questionnaire survey and gathered 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 about 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 prefer to have a practical programming guideline, online documents in hypertext form, and useful sample programs, put forward and maintained by the MPI standardization committee. The most important (and most difficult) thing is maintaining thorough and up-to-date supplemental documents. Regarding backward compatibility, many MPI users may be willing to 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. These questionnaire forms are still open and we will be able to compare results over time.

CRediT authorship contribution statement

Atsushi Hori: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Visualization, Supervision. **Emmanuel Jeannot:** Conceptualization, Methodology, Validation, Investigation, Writing – review & editing, Visualization. **George Bosilca:** Conceptualization, Methodology, Software, Validation, Investigation, Resources, Writing – review & editing. **Takahiro Ogura:** Methodology. **Balazs Gerofi:** Formal analysis, Writing – review & editing. **Jie Yin:** Data curation. **Yutaka Ishikawa:** Funding acquisition, Project administration, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We thank those who participated in this survey and those who helped us to distribute the questionnaire to their local communities. We especially thank the 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 national science agencies.

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], and (**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], and (**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], and (**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], and (**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], and (**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], and (**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], and (**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], and (**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)], and (**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], and (**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], and (**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], and (**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], and (**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], and (**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], and (**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. [No idea], and (**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], and (**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, and (**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], and (**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, and (**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], and (**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], and (**C**6) Other

Table B.5
Contributors.

	Contributor	#Ans		Contributor	#Ans
1	Germany†	159	22	Finland ‡	3
2	France†	125	23	Argentina	3
3	Russia	94	24	Australia	3
4	UK†	67	25	Taiwan	2
5	Japan	64	26	Serbia‡	2
6	USA	58	27	Pakistan	2
7	Italy†	57	28	Egypt	2
8	Switzerland ‡	40	29	Greece‡	2
9	Korea, South	27	30	Belgium‡	2
10	Austria‡	26	31	Tunisia	1
11	China	16	32	Peru	1
12	Sweden‡	15	33	Singapore	1
13	Spain‡	14	34	Norway‡	1
14	India	12	35	Mexico	1
15	Poland ‡	10	36	Denmark, Austria‡	1
16	Netherlands‡	8	37	Croatia‡	1
17	Brazil	6	38	Portugal‡	1
18	Denmark‡	6	39	Estonia‡	1
19	Czech Republic‡	5	40	Saudi Arabia	1
20	Luxembourg‡	5	41	UAE	1
21	Canada	4	42	Ukraine‡	1
†: Europe, ‡: Europe:others			42 contributors, 851 answers		

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], and (**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], and (**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], and (**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], and (**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

See Table B.5.

References

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

[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, Concurr. Comput.: Pract. Exper. 32 (3) (2020) e4851, e4851 cpe.4851. http://dx.doi.org/10.1002/cpe.4851.

[3] Research Organization for Information Science and Technology (RIST), High-performance computing infrastructure, 2018, http://www.hpci-office.jp/folders/english.

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

[5] JLESC, Joint laboratories for extreme-scale computing, 2021, https://jlesc.github.io/.

[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, http://dx.doi.org/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, Concurr. Comput.: Pract. Exper. (2020) e5901, http://dx.doi.org/10.1002/cpe.5901.

[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, http://dx.doi.org/10.2172/1482870.

[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, http://dx.doi.org/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] A. Denis, NewMadelein, 2021, http://pm2.gforge.inria.fr/newmadeleine.

[12] A. Mueller, Wordcloud for python documentation, 2020, https://amueller.github.io/word_cloud.

[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 Träff, W. Gropp, R. Thakur, Self-consistent MPI performance guidelines, IEEE Trans. Parallel Distrib. Syst. 21 (5) (2010) 698–709, http://dx.doi.org/10.1109/TPDS.2009.120.

[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, 2021, https://mpitutorial.com.

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