MP net as Abstract Model of Communication for Message-passing Applications

Martin Šurkovský
IT4Innovations
VŠB – Technical University of Ostrava
Ostrava, Czech Republic
martin.surkovsky@vsb.cz

ABSTRACT
MP net is a formal model specifically designed for the field of parallel applications that use message passing interface. The main idea is to use MP net as a comprehensible way of presenting the actual structure of communication within MPI applications. The goal is to provide users with the kind of feedback that can help them to check quickly whether or not the actual communication within their application corresponds to the intended one. This paper introduces MP net that focuses on the communication part of parallel applications and emphasizes its spatial character, which is rather hidden in sequential (textual) form.

KEYWORDS
MP net, Message Passing Interface, MPI, communication, abstract model, code comprehension, reshaping

ACM Reference Format:
Martin Surkovsky. 2019. MP net as Abstract Model of Communication for Message-passing Applications. In Proceedings of the 16th conference on Computing Frontiers (CF ’19), April 30-May 2, 2019, Alghero, Italy. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3310273.3322824

1 INTRODUCTION
MP net (Message Passing net) is a formal model specifically designed to capture the communication within MPI1 (Message Passing Interface) applications. MPI is widely used in the field of HPC (high-performance computing) as a means of communication between nodes of a computer cluster. Generally speaking, a programming parallel application is significantly harder in comparison to the sequential ones, and the new element which makes it more difficult is the involved communication. Therefore, a good understanding of communication is important.

One could ask why care about the communication, the authors know exactly how they have written the application, don’t they? Everyone who has ever written an application knows how easy it is to make a mistake. The code of MPI applications actually consists of several parts. Such a program is then distributed among the nodes of a computer cluster where each node can perform a different part of the program. In other words, MPI application describes a set of programs that are spatially distributed, but recorded in a linear form (linear stream of code). This is one of the aspects which makes programming of distributed applications more difficult. Moreover, communication involved in such kind of programs gives rise to a new sort of problems and errors, and programmers have to bear them in mind. Therefore, it can be very useful to have a tool that provides a different view of the MPI applications, the view that stresses their spatial character.

As the MP net is going to be presented back to users, one of the main requirements is that the model has to have a “nice” visualization. However, once such a model is available it can be used for other supportive activities than just the visualization (see Figure 1).

Figure 1: “Big picture” with the highlighted part considering this paper.

This paper shows how MP nets can be used to describe the communication within an MPI program. Even though the presented example is rather simple, it shows that a good understanding of communication can be crucial. The tool which will implement the presented ideas is currently under development.

The text is organized as follows. At the beginning, the state of the art is summarized. Then a motivational example is presented with the usage of MP net demonstrated upon this example. The conclusion sums up the article and discusses the possible utilization of MP nets. The more detailed information, including the formalization of MP net and the transformation of an MPI program can be found in the extended version of this text [16].

---

1https://www.mpi-forum.org

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPS II) project ‘IT4Innovations excellence in science’ - LQ1602” and partially supported by the SGC grants No. SP2019/108 “Extension of HPC platforms for executing scientific pipelines”, and No. SP2018/142 “Optimization of machine learning algorithms for HPC platform II”, VSB - Technical University of Ostrava, Czech Republic.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CF ’19, April 30-May 2, 2019, Alghero, Italy.

© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM. ACM ISBN 978-1-4503-6685-4/19/05. $15.00.

https://doi.org/10.1145/3310273.3322824
2 RELATED WORK

The motivation and original idea for the work presented here comes from the work on the tool KAIRA [2–4]. KAIRA is a development environment for applications using MPI. It strictly separates the communication and computational layers of the application. The former one is specified via a visual programming language while the latter one is written in C++. The idea of using a visual form for describing communication seems to be right because it emphasizes the spatial character of the communication. However, it turned out that using this visual form as an input which is provided and created by programmers, is similarly demanding as writing the application using MPI.

The current work aims to reverse the process and instead of modeling the communication via a visual language, the goal is to provide an abstract model of communication for an existing code of an MPI application. In a fully automatic way without need of any modification in the analyzed code. Hence, it shifts the paradigm from the field of visual programming to program visualization. In this way, the programmer is a reader of such model and not its creator.

The genre of visual programming has been defined by Grafton and Ichikawa [7] and includes three distinct areas: graphics techniques that provide both static and dynamic multidimensional views of software, graphics-based high-level programming languages, and animation of algorithms and software [8]. Their work was later refined by Myers [13]. He strongly distinguished visual programming, program visualization, and other high-level programming paradigms. Moreover, he also provided two taxonomies for the classification of programming systems, one for visual programming and the other for program visualization. According to Myers taxonomy MP nets fall into the category of static-code analyses.

Since this work mainly considers HPC applications, let us look at the supportive tools in this field. There is a collection of applications that perform either profiling or trace analysis. As the representatives of these categories can be named: TAU [14], SCALASCA [6], VAMPIRE [2] and the environment SCORE-P [9]. These performance measurement tools can however be identified as data-static in Myers taxonomy because the results are some statistical information or static charts based on the data collected from the run time of the applications. As an example of a parallel debugger ARM DDT [3] can be named.

As for the static-code analyzers whose results are a kind of visualization presented back to the users, tools such as SOURCETRAIL [4] or CODE MAP [5] can be named. These sort of tools are called code explorers and they help users with orientation in large projects by inspecting the complicated relationships in the application. The tool that also belongs to this category and concerns MPI is MPI-CHECK[11]. MPI-CHECK, however, checks the correct usage of MPI calls.

3 MP NET

MP net is a formal model specifically designed for the field of parallel applications that use MPI. It is built on Coloured Petri nets [8] (CPNs) and a simplified version 6 of Queuing Petri nets [1, 10] (QPNs). The goal is to propose a model that is, on one hand, powerful to fully capture the semantics of MPI [5], and, on the other hand, simple enough to be understood relatively easily by people. The balance between these two main requirements is the reason why a new model is proposed. The full formalization is out of the scope of this text. Nevertheless, it can be found in the extended version [16].

For the sake of readability, briefly, MP net is a finite set of addressable areas, where each area is a pair of an annotated sequential program fragment and communication net. Each pair describes a part of an original program that is devoted to a specific rank or set of ranks. The original program is split according to the ranks, if possible. From the MPI calls an appropriate communication net is generated and the sequential code is enriched with annotations that establish data exchange between these two compound parts. Both of these are generated from the code of the program.

The communication net is built on the simplified version of QPN. It defines four types of queuing input arcs that determine the strategy used to access tokens in a corresponding place. These are distinguished via an arrow head and can be categorized into two groups, single-headed (▷, ◁) and double-headed (▷▷, ◁◁). Single-headed arcs can access tokens only on the head of a queue while

2https://docs.microsoft.com/en-us/visualstudio/modeling/map-dependencies-across-your-solutions?view=vs-2017
3https://www.arm.com/products/development-tools/server-and-hpc/forge/ddt
4https://vampir.eu/
5https://www.vampir.eu/
6The stochastic and timed aspect of QPN is discarded. From the perspective of MP net, only scheduling of tokens in queuing places is important.
double-headed arcs can access tokens in the queue. These strategies influence the enabling rule. Another categorization is whether the arrow head is full or empty; empty arcs describe read-only arcs. This differentiation influences the occurrence rule.

4 MOTIVATIONAL EXAMPLE

In this section, we will look at three different implementations of one particular example. For each, we will look closely at the communication and discuss the differences. It will show that the first idea we implement might be different from the one we want.

Before we start with an example, let us remind a “standard” workflow when developing a program. Every programmer has to think through a problem at the beginning, come up with an idea of what the main components are, and how they interact with each other. They have to create a mental image of the program that is going to be developed.

When developing parallel programs programmers have to consider the communication done during computation. MPI often used to deal with that communication, is quite low-level. Then it is not surprising that the actual communication model of the resulting program may differ from the intended one. The difference between the intended communication model and the actual one – written in an application’s code – is the main motivation of this work. Therefore, the standard workflow is enriched with the possibility to extract an actual model of communication from the source code of an MPI application. At the moment, the programmer can use that model and confront it with their mental image.

4.1 All-send-one example

Let us demonstrate this workflow on a very simple example. Imagine we have a task to create a parallel program that works on \( n \) processes, \( p_0, p_1, \ldots, p_{n-1} \). Each process except the \( 0^{th} \) one sends a message (e.g. its process ID) to \( p_0 \) and \( p_0 \) receives all these messages. Our mental image may look like the one in Figure 2.

![Figure 2: Mental model of the simple task.](image)

With this image in our mind we can make a program (Listing 1) called all-send-one, that may look as follows. We can see, there is a loop on the \( 0^{th} \) process (\( rank = 0 \)) that receives all the messages sent by other processes. But, does the program behave (communicate) the way we expected? To answer this question, we can examine the code and state whether it does or does not. Nevertheless, it would be nice to have a way that presents communication in the program in a more comprehensive form than just looking at the source code.

Listing 1: First version of all-send-one program.

```c
int rank, size;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
if (rank == 0) {
    int ns[size-1];
    for (int i=0, src=1; src < size; i++, src++) {
        MPI_Recv(&ns[i], 1, MPI_INT, src, TAG_A, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
}
else {
    MPI_Send(&rank, 1, MPI_INT, 0, TAG_A, MPI_COMM_WORLD);
}
```

Let us look at the MP net for this program. It consist of two parts. The first one (see Figure 3) represents all the processes that send its rank to process zero. It can be seen, that the MPI_Send call actually consists of two parts. Sending the request and waiting for its completion (green and blue part of the picture). There are two special places: Active Receive Requests and Completed Receive Requests. The former one represents a queue of requests that is visible for the MPI message broker. It is labeled by compound place label ASR, i.e. all such places form different views of the same place. The latter place then represents a local place where the message broker sends the information about completion of the request.

The second part of MP net (see Figure 4) represents the \( 0^{th} \) process that receives all the messages. It is visible that both sending the receive request and waiting for its completion are done within the cycle. As we can see, before another request starts being received, the receive of the previous one has to complete. The receive loop we have used in the program introduces causal dependencies between individual sends that are not present in our original mental image (cf. Figure 2).

Suppose we want to get rid of these causal dependencies. The very first thing we can notice is that the receiving operation waits for a specific source (argument src). Instead, we can use wild card MPI_ANY_SOURCE if we do not need to wait for a specific process (see Listing 2). However, such a change does not make the causal dependencies disappear. The only change is that the data is received in a non-deterministic order. An appropriate block of MP net looks almost the same as the one in Figure 4, but the source entry in the request would be unset, i.e. the request is less restricted. Nevertheless, each receive is still completed within the cycle, i.e., one by one.
Figure 4: MP net for all-to-one program – rank zero; 1st version.

Listing 2: all-send-one using MPI_ANY_SOURCE wild card.

```c
for (int i = 0, src = 1; src < size; i++, src++) {
    MPI_Recv(&ns[i], 1, MPI_INT, src, TAG_A, MPI_ANY_SOURCE, TAG_A, &requests[i]);
    if (size > 0) {
        MPI_Waitall(size - 1, requests, MPI_STATUSES_IGNORE);
    }
}
```

Another change we can make is to send the receive requests in a non-blocking way and then wait for all of them (Listing 3). In this way we still have to wait for all the messages, but the way how the messages are received is left on the used implementation of MPI. So, we can say, that the last version is the closest one to the original idea. An appropriate addressable area of MP net for the 3rd version (cf. Listing 3) of the program can be seen in Figure 5. It shows that all the receive requests are sent first (green part) while the waiting for the completion is done out of the cycle.

Listing 3: all-send-one using non-blocking communication to break causal dependency.

```c
for (int i = 0, src = 1; src < size; i++, src++) {
    MPI_Recv(&ns[i], 1, MPI_INT, src, TAG_A, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    if (size > 0) {
        MPI_Waitall(size - 1, requests, MPI_STATUSES_IGNORE);
    }
}
```

5 CONCLUSION

The main goal of this work is to provide programmers of MPI applications with a kind of feedback that can help them to quickly check whether or not the actual communication within their application corresponds to the intended one. For this purpose MP net is established as a model that should be relatively easily understood by people and is capable of capturing the general semantics of MPI. At the current state, only point-to-point communication is covered. Nevertheless, the proposed approach does not disable to cover other, more advanced, routines from MPI. This may be one of a possible future work. However, the near future is devoted to the implementation of the presented approach into a working tool.

One could object that the model will increase with the size of the input program. That is not necessarily true, because the size is directly related to the number of MPI calls used within the program. Moreover, no matter of the program complexity MP nets can be still used locally to identify local issues.

Another objection could be that the model is still difficult to read. But the author is convinced that with a little practice it will be manageable. There is definitely a learning curve, but in comparison to visual programming tools, users just have to learn “to read” and can benefit from it. Moreover, it may be even more interesting to use this model further. For example, once it is available, performance data can be projected into the model, like coloring the most used transitions or stress the paths that are used most often during the real run. It can also be used for simulation of the application run and, for example, stop the program in a situation when two transitions are enabled to see the source of non-determinism.
REFERENCES

[1] Bause, F. Queueing petri nets—a formalism for the combined qualitative and quantitative analysis of systems. In Proceedings of 5th International Workshop on Petri Nets and Performance Models (Oct 1993), pp. 14–23.

[2] Böhm, S. Unifying Framework For Development of Message-Passing Applications. PhD thesis, FEI VŠB-TUO Ostrava, 17. listopadu 15, Ostrava, 11 2013. http://verif.cs.vsb.cz/sh/thesis.pdf.

[3] Böhm, S., Běhálek, M., Meca, O., and Šurkovský, M. Application and Theory of Petri Nets and Concurrency: 35th International Conference, PETRI NETS 2014, Tunis, Tunisia, June 23-27, 2014. Springer International Publishing, Cham, 2014, ch. Kaira: Development Environment for MPI Applications, pp. 385–394.

[4] Böhm, S., Běhálek, M., Meca, O., and Šurkovský, M. Visual programming of mpi applications: Debugging, performance analysis, and performance prediction. Computer Science and Information Systems 11 (2014), 1315–1336. IF: 0.477 (2014).

[5] Forum, M. P. Mpi: A message-passing interface standard. Tech. rep., Knoxville, TN, USA, 2011. Version 3.1.

[6] Geimer, M., and et al. Scalable collation and presentation of call-path profile data with CUBE. In Proc. of the Conference on Parallel Computing (ParCo), Aachen/Jülich, Germany (September 2007), pp. 645–652. Minisymposium Scalability and Usability of HPC Programming Tools.

[7] Grafton, R., and Ischkawa, T. Guest Editor’s introduction to the Special Issue on Visual Programming 18, 8 (1985), 6–94.

[8] Jensen, K., and Kristensen, L. M. Coloured Petri Nets - Modelling and Validation of Concurrent Systems. Springer, 2009.

[9] Knüpfer, A., and et al. Score-p: A joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir. In Tools for High Performance Computing 2011 (Berlin, Heidelberg, 2012), Springer Berlin Heidelberg, pp. 79–91.

[10] Kounev, S., Spinnler, S., and Meier, P. Introduction to queuing petri nets: Modeling formalism, tool support and case studies. In Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (New York, NY, USA, 2012), ICPE ’12. ACM, pp. 9–18.

[11] Luecke, G., Chen, H., Coyle, J., Hoekstra, J., Kraeva, M., and Zou, Y. Mpi-check: a tool for checking fortran 90 mpi programs. Concurrency and Computation: Practice and Experience 15 (02 2003), 93–100.

[12] Murata, T. Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77, 4 (Apr 1989), 541–580.

[13] Myers, B. A. Visual programming, programming by example, and program visualization: A taxonomy. SIGCHI Bull. 17, 4 (Apr. 1986), 59–66.

[14] Shende, S. S., and Malony, A. D. The tau parallel performance system. Int. J. High Perform. Comput. Appl. 20, 2 (May 2006), 287–311.

[15] Taylor, R. P., Cunniff, N., and Uchiyama, M. Learning, research, and the graphical representation of programming. In Proceedings of 1986 ACM Fall Joint Computer Conference (Los Alamitos, CA, USA, 1986), ACM ’86, IEEE Computer Society Press, pp. 56–63.

[16] Šurkovský, M. MP net as Abstract Model of Communication for Message-passing Applications. arXiv e-prints (Mar 2019), arXiv:1903.08252.