# MTF: A Scalable Exchange Format for Traces of High Performance Computing Systems

Luay Alawneh and Abdelwahab Hamou-Lhadj
*Software Behaviour Analysis Group*
*Department of Electrical and Computer Engineering*
*Concordia University*
*1455 de Maisonneuve Blvd. West*
*Montreal, QC, Canada H3G 1M8*
*{l_alawne, abdelw}@ece.concordia.ca*

*Abstract*— **Execution traces generated from running high performance computing applications (HPC) may reach tens or hundreds of gigabytes. The trace data can be used for visualization, analysis and gathering profiling information about the target system. However, in order to make the utilization of this data efficient, the trace needs to be represented in a structure that facilitates the access to its data. One important factor that should be considered when representing trace data is the scalability of its schema; the trace metamodel should be able to represent the trace in a compact form that enables scalability of the analysis tools. Additionally, a trace file needs to be available in a format that is well-known in the software engineering arena supported by its openness. In this paper, we propose a metamodel for representing dynamic information generated from HPC that use the MPI as the inter-process communication model. MPI Trace Format (MTF) is meant to meet the aforementioned requirements and is intended to facilitate the interoperability among the different trace analysis tools.**

***Keywords: HPC-High Performance Computing systems; Inter-process communication traces; MPI-Message Passing Interface; Standard exchange format; Trace Metamodel***

## I. INTRODUCTION

High Performance Computing (HPC) systems are used extensively in various domains including bioinformatics, telecommunications, computation-intensive scientific systems, and others. Although the benefits of these applications are numerous, they tend to be difficult to analyze and understand which often hinders maintenance and other software engineering tasks [1]. These applications involve a large number of inter-communicating processes which is one of the main obstacles to their effective analysis. Several techniques and tools have emerged to facilitate the analysis of HPC applications (e.g. [2,3]). These tools come with many features including trace analysis algorithms, visualization layouts, optimization algorithms, pattern detection methods, and others that can help in studying the run-time behavior of these applications for performance analysis, debugging, deadlock detection, and so on. These tools, however, do not interoperate due to a lack of a common exchange format for representing HPC traces. The only way to take full advantage of these tools is to convert the trace file from one format into another. This is often cumbersome and impractical. Clearly, a common trace format that enables synergy and sharing of data among tools is needed, which reduces the cost to represent HPC traces.

The objective of this paper is to present MTF (Message Passing Interface Trace Format), an exchange format that we have developed to represent run-time information generated from HPC applications. The focus is on inter-process communication traces based on the message passing paradigm, with a particular interest in the MPI (Message Passing Interface) standard, which is the de-facto standard used in most today's high performance computing distributed systems [4]. There exist some exchange formats in the literature for HPC-generated traces (e.g., [5, 6]), but most of them do not scale up to large traces. Many of them are also proprietary and represent traces in binary format which hinders their portability. MTF is built with several requirements in mind to facilitate its adoption and enable it to become a standard exchange format for traces generated from HPC applications. One of the key requirements that we addressed carefully is the ability for MTF to support very large traces. This work is a continuation of the work presented in [7] in which we presented the first version of MTF where it failed to support large traces due to the fact that it does not consider any technique to compact the traces. The compaction scheme used in this paper is inspired by the work found in [8]. MTF was also limited to representing MPI operations. Several improvements have been made to the initial version of MTF such as a new mechanism for compacting MPI traces. We also added support for routine calls (user calls) which makes MTF more expressive.

The rest of the paper is organized as follows. In Section 2, we briefly discuss the shortcomings found in the literature. In Section 3, we present the MTF and its main components. Section 4 presents some empirical results showing the compaction gain. The paper is concluded in Section 5.

## II. RELATED WORK

In our previous work [7], we surveyed execution trace formats for traces generated from HPC applications that use the message passing model for inter-process communication. Most of these formats are being used in proprietary tools which hides the details of their metamodels. This contradicts the openness requirement for a common exchange format. Also, none of these formats proposes an approach for compacting traces of HPC systems, which hinders their ability to scale up to large traces.

## III. MPI Trace Format (MTF)

In this section, we start by describing the principles that we used to guide the design of MTF followed by a description of the MPI traces domain with a particular focus on the compaction scheme we used to reduce the number of model elements that need to be represented.

### *A. Guiding Principles*

A trace format should meet certain requirements, such as expressiveness, scalability, openness, simplicity and transparency, in order to qualify as a common exchange format [12]. In this paper, we only focus on the following key requirements as guiding principles in the design of MTF.

***Expressiveness***: An exchange format should be expressive enough to capture the needed information to enable various types of analyses such as the MPI operations, routine calls, exchanged data, the sender, receiver, the timestamp, partner processes, and others.

***Scalability***: An exchange format should be scalable to support a large amount of information efficiently and in a way that does not degrade access to the instance data for analysis purposes.

***Extensibility***: Exchange formats should be easily extended in order to support new or different data types. Also, they should be extended without affecting previous versions of the trace data.

### *B. The Domain of MPI Traces*

An MPI trace depicts the execution of the running processes in the program along with the messages exchanged among them. HPC applications often follow the Single Program Multiple Data (SPMD) paradigm in which the program tasks are run in parallel on multiple processors to maximize performance. Communication among processes is based on executing MPI operations supported by the MPI environment. MPI supports two communication modes: point-to-point and collective communications. The MPI specifications [4] provide detailed description of the various MPI operations. An MPI trace can be considered as a set of streams of data, where each stream corresponds to one process in the program. Each trace contains the routines executed by the process, the MPI operations invoked by the process to communicate with other processes, the messages sent and received, and many other details such as timestamps.

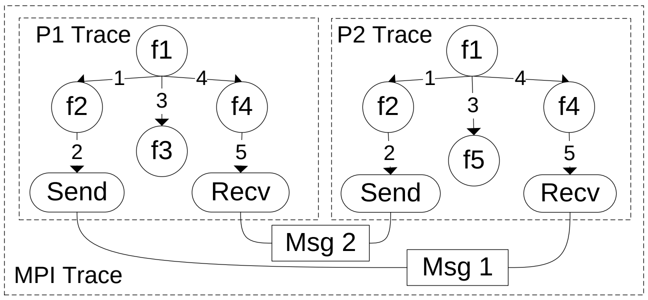

**Figure 1. MPI Trace Representation**

Figure 1 shows an example of two processes that execute in parallel five functions *f1*, *f2*, *f3*, *f4*, and *f5*. The interaction between these two processes is also shown as typical Send and Receive MPI operations along the exchanged messages.

Another important aspect that distinguishes HPC programs from other systems is the existence of communication patterns that characterize the process communication topology of the application [9]. Examples of such patterns include the butterfly and wavefront patterns [9]. MTF supports the description of these patterns. It also supports the ability for the user to define new patterns. In [10], we presented an approach for automatically detecting patterns in MPI execution traces expressed in MTF.

We reduce the amount of data in the original trace by representing repetitive sequence of events that occurs in a trace only once. Our techniques vary depending on whether the repetitions appear contiguously in the trace or not. Contiguous repetitions are often caused by the presence of loops and recursive calls in the code or the way the scenario is executed. They can be removed by collapsing the repetitions into one node and keeping an array of timestamps extracted from the original nodes. Our second compaction mechanism consists of representing repetitions that appear non-contiguously in the trace (also known as trace patterns) only once in a trace. For this purpose, we adapted the compactness scheme presented in [8] and in which the authors proposed a variant of Valiente's algorithm [8] to transform a call tree into an ordered Directed Acyclic Graphs (DAG) where similar subtrees are represented only once [11]. The authors showed that this transformation provided maximum compactness of the trace data while it preserved the order of calls in the original trace.

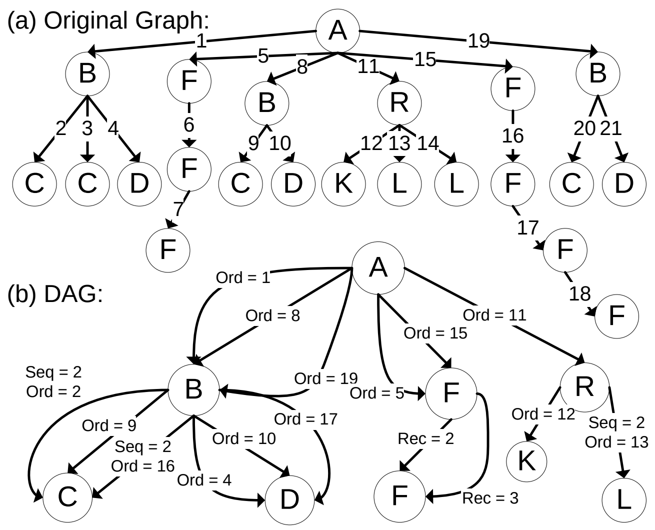

**Figure 2. Tree to DAG Conversion Example**

Figure 2 shows an example of converting a tree into an ordered DAG after removing contiguous repeats. We keep the original label (order) on each edge in order to keep track of the changes in the original tree. Figure 2b shows the final DAG which contains 9 nodes and 16 edges compared to 22 nodes and 21 edges in the original tree. Moreover, in the

MPI trace, if the same message between two processes is being exchanged repeatedly, then we can collapse these messages into one node while keeping track of the timestamps in an array.

### C. *MTF Metamodel*

The MTF metamodel is represented as the class diagram shown in Figure 3. The metamodel supports the description of the usage scenario (class *Scenario*), the ability to express different trace types (class *Trace*), information about processors and processes that run on these processors (classes *Processor* and *Process*), various traceable units including the program function calls (*RoutineCall*), and MPI operations. The MPI operations that are supported by the model cover a large range of the MPI standard including the point-to-point and collective operations. We believe that this makes MTF a very rich and expressive language. It also subsumes a number of existing exchange formats for MPI applications, and which only provide partial support for MPI traces. The class Edge models the edges between various traceable units. An Edge can be a regular edge, a sequence edge, or a recursive edge, which is represented by the attribute *type*. It has a number of repetitions. It should be noted that a traceable unit which characterizes routine calls can have multiple incoming and outgoing edges to support the concept of ordered directed acyclic graphs. Several constraints are added to the model (not shown in this paper) to limit the traceable units that can have multiple incoming and outgoing edges to routine calls only (e.g. instances of the MPOperation class cannot have an outgoing edge). The communication patterns are represented by the classes *TracePattern* and *PatternOccurrence*, which depict, respectively, the pattern itself and its occurrences in the trace.

## IV. Empirical Evaluation

In order to show the ability of MTF to represent MPI traces generated from large systems in a compact form, we tested it on several trace files generated by the VampirTrace tracing tool [12] in the OTF format [5]. The OTF format does not apply any compaction on the trace events themselves. It uses *zlib* [13] to compress the trace file into several streams. However, the number of traces in the uncompressed OTF file map exactly to the number of events generated from the target system. We take OTF traces and apply our compaction techniques on them. More precisely, we load OTF traces as call trees. Each call tree represents the calls executed by one process. The point-to-point messages are linked to their corresponding MPI calls as was shown previously in Figure 1. Then, we perform our collapsing rules on the nodes in the tree as well as on the messages. Finally, we convert each tree into a DAG which is the MTF representation of the original OTF trace.

We targeted four programs provided by the NAS Parallel Benchmark [14]. In this case study, we target four programs that are part of the NPB suite (CG, MG, LU, and SP). Table 1 shows the empirical results and the compaction rate (CR) gained after applying our compaction scheme which reached 78% in some cases.

## V. Conclusion and Future Work

In this paper, we presented an exchange format for MPI traces generated from HPC applications, called MTF. MTF is built with the requirements for a standard trace exchange format, which we believe can facilitate its adoption. We provided the abstract syntax (metamodel) of MTF in the form of a UML class diagram. MTF is extensible to acquire new trace types. Also, MTF is scalable to very large traces. It enables representing each process trace as an ordered directed acyclic graph. We provide the specifications of MTF in an open framework in order to promote it to be adopted as a standard exchange format.

We provided 8 traces generated from 4 different HPC programs. We applied our compaction method using DAG on traces in the OTF format. The results show that by using our approach, a trace could be 78% more compact than its original version. An immediate future direction is to continue to demonstrate how MTF can represent very large traces. We also need to further investigate ways to reduce the number of edges between the nodes of the ordered DAG. This can be made possible by investigating ways to group similar (but not identical subtrees) as instances of the same pattern. We need to create converters that would convert the formats used by other tools into MTF to encourage tool vendors to adopt it. Finally, we will continue working on the tool support for MTF by enriching its query language.

Acknowledgment

This research is supported by NSERC (Natural Sciences and Engineering Research Council of Canada)

**Table 1 Empirical Results (#P is number of Processes, N is number of Nodes, E is number of Edges, CR is the Compaction Rate = (1 – B / A) * 100%, M is number of Messages, 0: before compaction, c: after compaction**

|  | #P | $N_0$ | $E_0$ | $M_0$ | $A = \sum(N_0, E_0, M_0)$ | $N_c$ | $E_c$ | $M_c$ | $B = \sum(N_c, E_c, M_c)$ | CR |
|---|---|---|---|---|---|---|---|---|---|---|
| **CG** | 16 | 3509121 | 3509105 | 47104 | 7065330 | 561 | 1479281 | 42716 | 1522558 | 78% |
| **CG** | 32 | 7139585 | 7139553 | 134656 | 14413794 | 1121 | 3039969 | 119252 | 3160342 | 78% |
| **MG** | 16 | 609874 | 609858 | 11024 | 1230756 | 648 | 608280 | 7588 | 616516 | 49% |
| **MG** | 32 | 692690 | 692658 | 21728 | 1407076 | 561 | 689428 | 15001 | 704990 | 50% |
| **LU** | 32 | 10473947 | 10473915 | 1644936 | 22592798 | 1518 | 6009007 | 986054 | 6996579 | 69% |
| **LU** | 64 | 18310623 | 18310559 | 3542924 | 40164106 | 2990 | 12088359 | 2046493 | 14137842 | 68% |
| **SP** | 64 | 9525649 | 9525585 | 1232256 | 20283490 | 2881 | 4340289 | 1112352 | 5455522 | 73% |
| **SP** | 100 | 14359525 | 14359425 | 2406600 | 31125550 | 4501 | 8465901 | 2188443 | 10658845 | 66% |

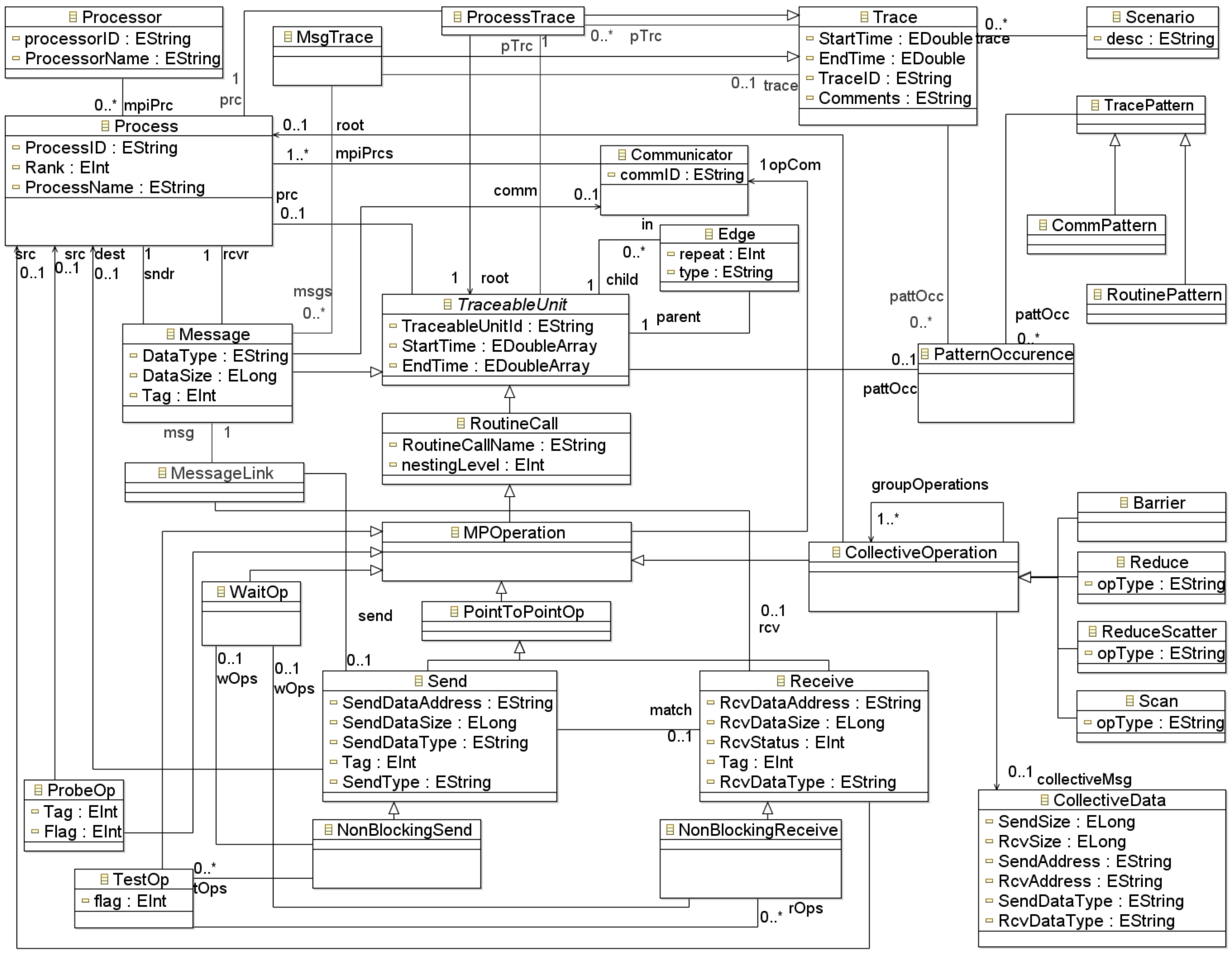

**Figure 3. The Modified MTF Metamodel**

## REFERENCES

[1] D. Becker, . Wolf, W. Frings, M. Geimer, B.J.N. Wylie, B. Mohr, “Automatic tracebased performance analysis of metacomputing applications,” *In Proc. of the International Parallel and Distributed Processing Symposium*, IEEE Computer Society, 2007.

[2] TAU User’s Guide. URL: http://www.cs.uoregon.edu

[3] Vampir Visualization tool. URL: http://vampir.eu

[4] Message Passing Interface forum. MPI: a message passing interface standard, June 1995

[5] A. Knüpfer, R. Brendel, H. Brunst, H. Mix, W. Nagel, “Introducing the Open Trace Format (OTF)”, *In Proc. of the International Conference on Computational Science*, 2006.

[6] A. I. Margaris, “Log File Formats for Parallel Applications: A Review,” *International Journal of Parallel Programming*, 37(2), 2009.

[7] L. Alawneh and A. Hamou-Lhadj, “An exchange format for representing dynamic information generated from High Performance Computing applications,” *Future Generation Computer Systems*, 27(4), 2011.

[8] A. Hamou-Lhadj, T. Lethbridge, “A metamodel for dynamic information generated from object-oriented systems,” *In Proc. of WCRE’04*, Electronic Notes in Theoretical Computer Science, vol. 94, 2004.

[9] N. Palma, “Performance Evaluation of Interconnection Networks using Simulation: Tools and Case Studies” *PhD Dissertation*, Department of Computer Architecture and Technology, University, Spain, 2009.

[10] L. Alawneh and A. Hamou-Lhadj, "Pattern Recognition Techniques Applied to the Abstraction of Traces of Inter-Process Communication", *In the Proc. of CSMR 2011*, Oldenburg, Germany, 2011.

[11] Downey J.P., Sethi R., and Tarjan R.E., “Variations on the Common Subexpression Problem”, *Journal of the ACM*, Volume 27, Issue 4, pages 758-771, 1980.

[12] VampirTrace, ZIH, Technische Universitat, Dresden. http://tu-resden.de/die_tu_dresden/zentrale_einrichtungen/zih.

[13] J. L. Gailly and M. Adler, “zlib 1.1.4 Manual”, 2002. URL: http://www.zlib.net/manual.html.

[14] NAS Parallel Benchmarks 3.3, URL: http://www.nas.nasa.gov/Resources/Software/npb.html.