Chapter 5
Gleaming the Cube: Online Performance Analysis and Visualization Using MALP

Jean-Baptiste Besnard, Allen D. Malony, Sameer Shende, Marc Pérache and Julien Jaeger

Abstract Multi-Application onLine Profiling (MALP) is a performance tool which has been developed as an alternative to the trace-based approach for fine-grained event collection. Any performance and analysis measurement system must address the problem of data management and projection to meaningful forms. Our concept of a *valorization chain* is introduced to capture this fundamental principle. MALP is a dramatic departure from performance tool dogma in that is advocates for an online valorization architecture that integrates data producers with transformers, consumers, and visualizers, all operating in concert and simultaneously. MALP provides a powerful, dynamic framework for performance processing, as is demonstrated in unique performance analysis and application dashboard examples. Our experience with MALP has identified opportunities for data-query in MPI context, and more generally, creating a "constellation of services" that allow parallel processes and tools to collaborate through a common mediation layer.

5.1 Introduction

Scalable performance measurement on modern supercomputing systems inevitably becomes a problem of scalable data management. Whether parallel profiles or traces are collected, large, distributed performance data are a fundamental part of an application's execution. If the performance data is accessed during execution (we use the term "performance monitoring" in this case), the data access and transformation (i.e., analytics) becomes an important concern. Ultimately, this becomes a question of how performance information is eventually processed by a human, requiring a *projection* to an intelligible state that acknowledges the end-user's cognitive

J.-B. Besnard (✉)
ParaTools SAS, Bruyeres-le-chatel, France
e-mail: jbbesnard@paratools.fr

A.D. Malony · S. Shende
ParaTools Inc., Eugene, USA

M. Pérache · J. Jaeger
CEA, DAM, DIF, 91297 Arpajon, France

A. Knüpfer et al. (eds.), *Tools for High Performance Computing 2015*,
DOI 10.1007/978-3-319-39589-0_5

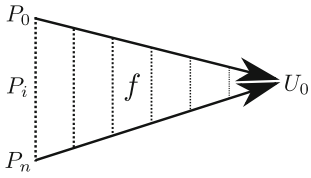

Fig. 5.1 Illustration of a measurement chain from processes ($P_{0\to n}$) to a single end-user (U_0)

capabilities. This defines a parallel measurement (processing) chain with the purpose to allow the exploration of a distributed performance state, for example, to understand it (debugging), to derive potential inefficiencies (performance) or to pinpoint misuses of standardized interfaces (validation). In this section we propose to discuss the trade-offs guiding the design of such measurement chains first from a global point of view and then by going through alternative approaches developed by tools.

5.1.1 From Performance Data Management to Performance Valorization

In order to define more formally the role of such measurement chain for HPC tools, consider Fig. 5.1. On the left we identify several processes $P_{0\to n}$ which are *spatially* distinct, being scattered in various nodes within the parallel machine. On the right, we modeled the end-user who desires the transformed performance information in more meaningful forms. Thus, the data have to be *projected* to global metrics which are easier to interpret, a process modeled by function f. This function is not necessarily statically defined and could be taking user-defined parameter. Moreover, this function is by definition performing data analytics associated with a computational cost. Dealing with the *projection* it can be done on any dimension defining a parallel computation: spatial (per process/thread/node), temporal, per procedures, per file, per programming model, and so on. However, in all cases it has to be doing some form of transformation (reduction, aggregation, analysis, …) in order to provide actionable, semantic information to the end-user.

Starting from this simple model we can see that the measurement chain is closely associated with a *valorization*[1] function f which role is to transpose numerous events to an unified view interpretable by an human. It is interesting to note that this function also plays role in instrumentation chain's scalability by defining what are the valuable informations. Indeed, by definition a parallel computation will generate a large number of instrumented events from several processes, leading to a data-management problem. In the mean time, we argued that the user cannot take advantage of individual events and therefore that a tool was needed to explore these data either working on projections or subsets. It is then possible to envision projections valorizing data

[1] *Valorize* means to give or ascribe value or validity to something.

before display, for example, by reducing spatially during measurement—overcoming in the meantime data-management problematics.

It is this relationship between performance data and how and when that data are processed which gave rise to the landscape of performance tool approaches and infrastructures we see today.

5.1.2 In-Place Performance Data Processing

When processing data in-place, events of interest are projected at the moment they are instrumented. It has the advantage of processing data locally, avoiding to generate any event-related data movement (all can be done on the stack). The counterpart of this locality is the projection function cost which directly affects the application. As a consequence, this projection cost has to remain limited in terms of both computational power and memory footprint as it shares resources with the target. In such conditions, it is then generally prohibitive to maintain shared state. The in-place approach is among the most scalable, overcoming data-management problematic by performing an immediate projection. The *mpiP* [24] performance tool uses in-place processing to precisely track MPI time. It first performs a statistical temporal reduction before aggregating results spatially when application ends. The *HPCToolkit* [1] performance tool uses in-place sampling and statistical profiling to project execution time, counters, and other events on threads of execution before aggregating profiles spatially when application ends.

5.1.3 Trace-Based Performance Data Processing

The trace-based approach has been retained by several tools. It consists in storing all the events in a file-system trace in order to have them processed in a post-mortem fashion. This has several advantages, first the processing is completely decoupled from the instrumented (measured) program and can therefore have an arbitrary cost without impacting measurement quality. Moreover, as file-system is a non-volatile storage the trace can be analyzed several time at various granularities. Nonetheless, performance traces also bring several problems. First, the file-system is known to be subject to contention as a shared resource, its scalable usage requiring the use of a parallel IO library such as for example SionLib [10] used by Score-P [17]. Similarly, event verbosity can lead to impractical trace sizes. In the meantime as parallel IOs can be relatively expensive, instrumenting events with small durations can lead to prohibitive overheads.[2] A wide range of trace formats were developed, Pajé [20] with a focus on genericity, SLOG [6] which was designed for temporal trace visualization in Jumpshot [26], and the Open Trace Format 2 (OTF2) [9] (which replaced OTF1

[2]Dilation is a function of the duration ratio between events of interest and instrumentation cost.

[15] with several performance improvements). Using this approach, the Vampir [16] tool allows the interactive exploration of parallel application's temporal behavior, up to single event granularity.

5.1.4 OnLine Performance Data Processing

This last approach can be seen as a combination of the previous two. It consists in coupling instrumented processes with processes dedicated to the analysis at runtime. This removes the need for the data-multiplexing required by parallel IO libraries, directly using network coupling. In complement, having dedicated computing resources in the analysis allows more complex event processing, possibly with a distributed state. Nonetheless, as analysis processes are limited in memory, events still have to be either processed fast-enough or sufficiently small, limitation which can be mitigated by increasing processing resources. Such an approach has been used at scale with, for example, the MrNET Tree-Based Overlay Network (TBON) [21], performing a spatial reduction of stack traces in the STAT debugger [2]. This kind of reduction tree is also used to characterize applications in Periscope [3] or to derive a global state in the Generic Tools Infrastructure (GTI) [13] used by MUST [14] for validation. Similarly, commercial debuggers such as DDT and TotalView rely on a tree-based overlay network to control and monitor distributed processes. In MALP [5], we use this approach for fine-grained online trace analysis.

5.1.5 Summary

Table 5.1 summarizes the aforementioned performance processing approaches. While the trace-based approach traditionally decouples the analysis from the application's execution and allows replay, it still has scalability problems when facing large trace volumes that can force online actions (e.g., transferring in-memory trace buffers to disk when they become full).

Table 5.1 Performance coupling-modes characteristics overview

Approach	Bandwidth usage	Overhead decoupling	Analysis type	Replay
Trace-based	File-system	Total (post-mortem)	Arbitrary analysis	Yes
In-place	None (Stack)	None	Lightweight	No
Online	Network	Partial (pipeline)	Bandwidth dependent	No

Dealing with the Online approach, the notion of allocating extra resources in the online approach partially decouples the analysis from the instrumentation (measurement), thereby overcoming file-system limitations thanks to direct network coupling. Such a method supposes that performance events can be processed fast-enough so as not to impact the instrumented application, therefore putting an upper bound on analysis' complexity.

Clearly, nothing prevents these approaches to be combined to mitigate their limitations. For example, when we think of an IO proxy, one would use the online coupling paradigm to perform trace-based storage—delegating the IO overhead to remote processes furnishing their local memory as caches. Performance tools also mix these approaches. For example, Scalasca [12] reduces call-stacks locally and then unifies event identifiers during analysis to derive performance metrics, eventually storing an unified report for post-mortem processing—covering the three approaches we presented. Similarly, the Tuning and Analysis Utility (TAU) [23] has continually explored different approaches, proposing profiles, traces and even online results through MrNET [19]. TAU and Scalasca eventually contributed to the Score-P [17] unified infrastructure which also is able to collect call-stack profiles while generating traces, mixing in-place and post-mortem approaches.

Looking at all these state of the art tools deploying various data-management approaches, one can see the trade-offs implied by supercomputers where a global state is out of reach, context in which tools need to derive models and metrics more than raw data. Naturally, traces describe a parallel execution with the highest level of details thanks to re-playable fine-grained event. However, file-system limitations coupled with the combinatorial complexity of millions of computation units communicating advocate for a *multi-scale* profiling approach. Therefore, starting from a global overview generated through scalable spatial reductions such as TBONs or temporal/functional reduction inside local accumulators, one should be able to change the *focus of attention* to query local data—possibly more verbose as less spatially diffuse. This model supposes that performance data ranging from the local to the parallel state can be queried in a selective manner, somehow reversing the instrumentation chain. In other words, if a trace will always be the most verbose manner to explore a problem either spatially or temporally located, we think that at larger scales in-situ/online analytics and visualization can provide sufficient insights to heuristically extract valuable subsets from the unreachable parallel state.

The following section presents our research on the Multi-Application OnLine Profiling or MALP framework Its online nature was at first motivated by file-system's limitations, requiring the use of a secondary set of processes to perform the reduction while the application was running in order to avoid storing large volumes of redundant data. This allowed us to generate profiling reports without storing an intermediate trace as detailed in Sect. 5.3. Then, as our instrumentation chain was collocated with the application we started investigating in-situ approaches, trying to "see" the application running as illustrated by the application dashboard described in Sect. 5.4.

5.2 MALP Architecture

In order to obtain fine-grained performance data, tracing approaches move analysis to offline. Our work investigates whether it is possible to reduce the impact on the application of online analysis through network-coupling of computing resources dedicated to online data projection. In this section, we detail the steps which turned MALP, designed for online reduction [4, 5], into an in-situ analysis engine.

As described in [5], on the analysis side, we implemented a data-flow engine inspired from the *Blackboard* expert systems, the idea being to have several (knowledge system) plug-ins valorizing incoming performance events in an orthogonal manner by registering themselves on a given event type. In MALP most of these analysis are performing either temporal or functional reductions. The interprocess aspect was originally handled by synchronous MPI reductions when application ended, reducing data accumulated in each module. Due to this intermediate MPI step, we were not taking advantage of the online nature of MALP as unified performance data were only reachable at the end of the execution in a PDF report. The spatial reduction acting as a barrier on our data-collection process, we looked for an alternative design in order to allow runtime interactions. Following the Blackboard idea, we then changed MALP design to integrate a visualization agent interacting with distributed processes. The idea being that reduction plug-ins would produce data which could then be directly consumed by rendering components.

As shown in Fig. 5.2, we added a new visualization agent. It was implemented in JavaScript on top of *Node.js* in order to be modular while taking advantage of built-in networking libraries (TCP and HTTP). In this new infrastructure, each plug-in describes a template-based visualization component which is loaded inside

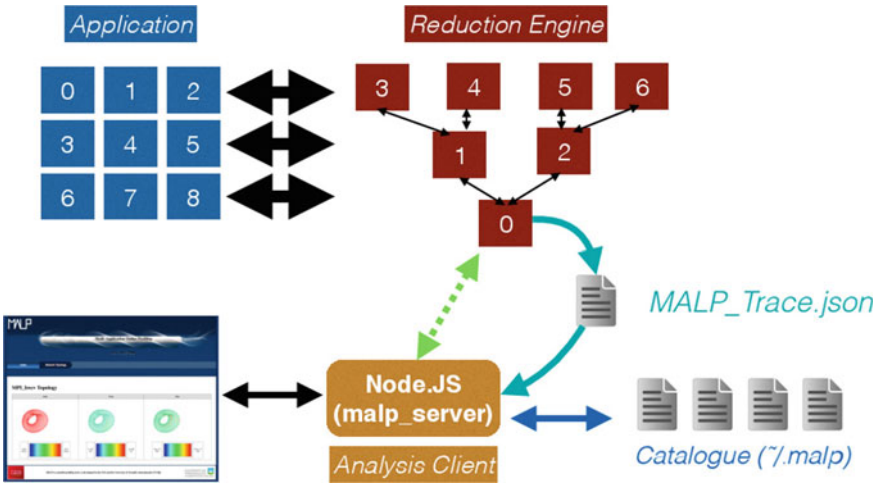

Fig. 5.2 MALP instrumentation chain including the web-based rendering component

a lightweight web-server. Server distributing dynamic web-pages to client web-browsers, opening the way for interactive data-exploration (further described in Sect. 5.3). The coupling between the application and the analysis engine remained unchanged, forwarding fine-grained events. Dealing with the coupling between the server and the analysis engine we decided to adopt a shared data-model in order to provide generic storage capabilities. Our server being in JavaScript and our target rendering environment being a web-browser (also JavaScript enabled), we decided that the plug-ins located in the reduction engine would output *JavaScript Object Notation (JSON)* [7] formated data. As the analysis was implemented in C, we defined serializable C data-structures allowing analyses to store arbitrary data in a hierarchy of JSON objects. Moreover, data for each module is stored inside a common root object, allowing the whole performance data-set to be serialized at once. This root JSON object, storing arbitrary sub-objects defined a common data-structure for performance data, this with almost no constraints unlike in trace-formats—what is stored is at the discretion of the plug-in.

As shown in Fig. 5.2, to deal with the coupling between the analysis engine and the Node.JS web-server, we relied on two methods. The first one uses a JSON file which is obtained after performing a reduction of the per-process performance data in the root analysis task—approach similar to what was previously done for our PDF reports. The JSON file is generated by serializing the root JSON object which contains the data from all plug-ins. Such files can be stored in a performance catalog for later retrieval in the server. The second coupling approach is an online one. Thanks to a TCP socket connected to the root of a binary tree gathering analysis tasks, the web-server is able to send and query JSON objects, relying on serialization and deserialization primitives in both C and JavaScript. We call this coupling the *Node Shared Cache*, making it possible for the C data located in a remote process to be queried from the Node.JS web-server and transitively from a web-browser.

To illustrate MALP's ability to couple performance data with analysis, the following sections describes our new performance-data visualization (Sect. 5.3) and interactive application control (Sect. 5.4).

5.3 Profiling with MALP

In this section we briefly illustrate MALP profiling capabilities. The new web-based interface, replacing a PDF report, allowed the implementation of original performance visualizations, taking advantage of web technologies (D3.js, WebGL, plot libraries …) to provide an interactive user-friendly interface while minimizing development cost. MALP now provides a wide range of performance analysis modules with, for example, profiles, temporal and spatial behavior maps, load-balancing charts in Fig. 5.3a and interactive communication topologies in Fig. 5.3b. JSON profile files generated by the analysis engine can be directly loaded through the web-interface, being stored in a *performance catalog* for latter reference.

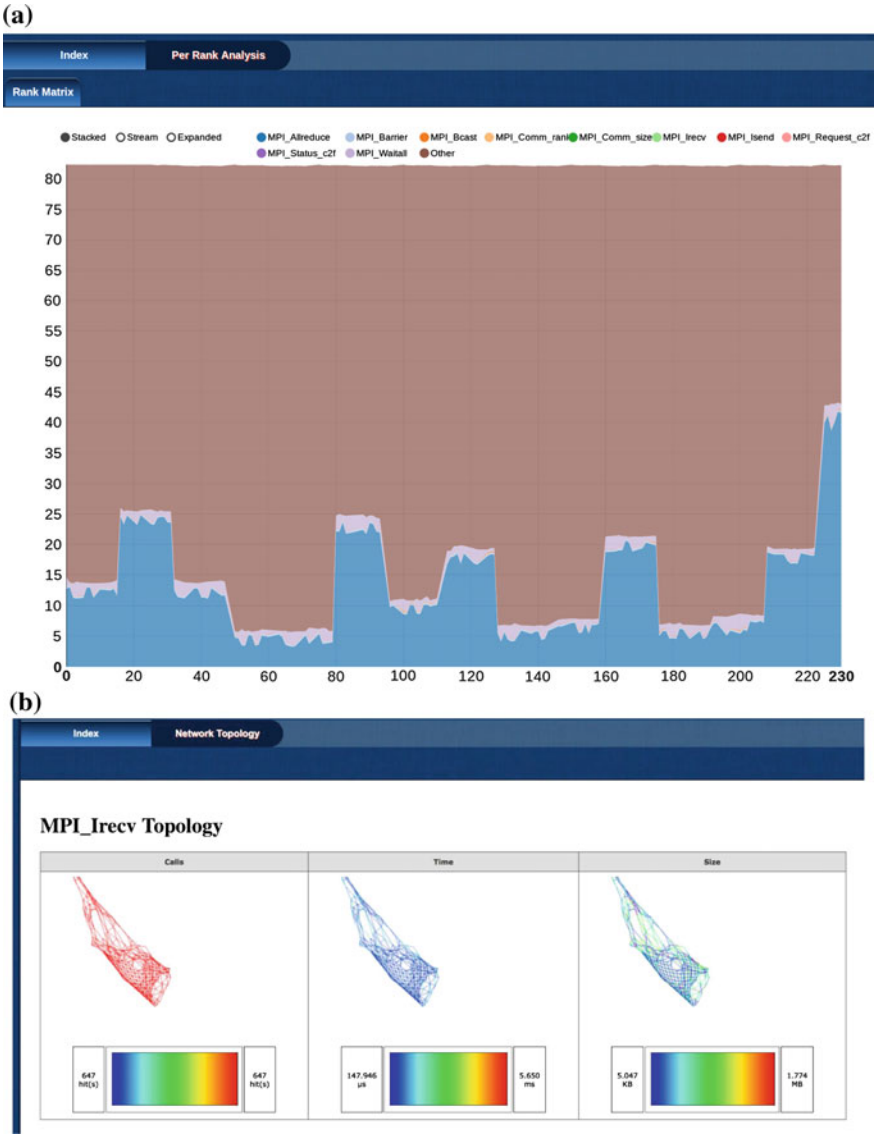

Fig. 5.3 Screenshots from MALP web-interface. **a** Process view of an imbalanced computation. **b** Interactive 3D topology viewer

The templating system developed in the Node.JS part of MALP allows the expression of performance analysis in a straightforward manner, the JSON object exported by the analysis engine being directly addressable programmatically by the JavaScript abstraction. This process can be illustrated with Fig. 5.3a presenting MPI profile balancing and generated from the following per-task MPI profile:

```
[[{"hits":128,"size":1468010000,"name":"MPI_File_read","time":3.65},..],..]
```

This JSON array is first indexed with the MPI rank and then presents a per-rank profile gathering for each instrumented function, the number of calls, the cumulative time and the total size when suitable. From this point, generating the MPI balancing graph was then just a matter of converting the JSON data-layout to match the requirements of the plotting library (here NVD3). Similarly, if we consider Fig. 5.3b, we also start from exported JSON data providing connectivity information for each MPI point-to-point in terms of hits, time and size as follows:

```
{"MPI_Recv":[{"src":1,"hits":4,"size":192,"dest":0,"time":0.0005},..],..}
```

The topology analysis, uses these data to generate a Graphviz [11] graph description file in order to compute the three-dimensional layout of the graph. Then Graphviz's output, enriched with coordinates is reloaded in the server, coordinates are centered around object's center of gravity and normalized before being passed to the interactive three.js WebGL layer inside the client web-browser.

These two simple examples illustrate the advantage of our JSON approach which is directly operated from the JavaScript layer for both pre-processing and rendering in a web-browser. If compared to a binary trace format, this approach is clearly less space efficient but makes data-analysis much simpler. Moreover, adding new events or plugins to the reduction engine requires no modification in the intermediate format unlike trace formats with (in general) a constrained event set. However, the interface was not yet taking advantage of the collocation with the running application, inciting us to explore new use cases exemplified in next sections.

5.4 Introducing Parallel Application Dashboards with MALP

The *Node Shared Cache* is also a way of interacting with running applications, in this section we illustrate its use to monitor a sample parallel code. The motivation for this is that data-management techniques developed for performance monitoring can clearly be applied more globally to application data-management.

Figure 5.4 presents an HTML interface that we designed in a few hours on top of MALP online coupling facilities. We made a simple program computing Pi by integrating $Arctan$ with the Euler approximation method (as known as the rectangle method). In Fig. 5.4a, we use the HTML page hosted in the Node.JS server to first explain our numerical method relying on $\int_0^1 Arctan(x)\mathrm{d}x = \frac{\pi}{4}$. Then we allow the user to *interactively* set the number of intervals (or rectangles) to be used, before clicking on compute. This has the effect of sending a GET operation to the server which then relays the information to the MPI processes by setting a value in the *Node Shared Cache*—parallel computation being triggered on value change. Then, when the processes are done computing the integral on their own sub-intervals,

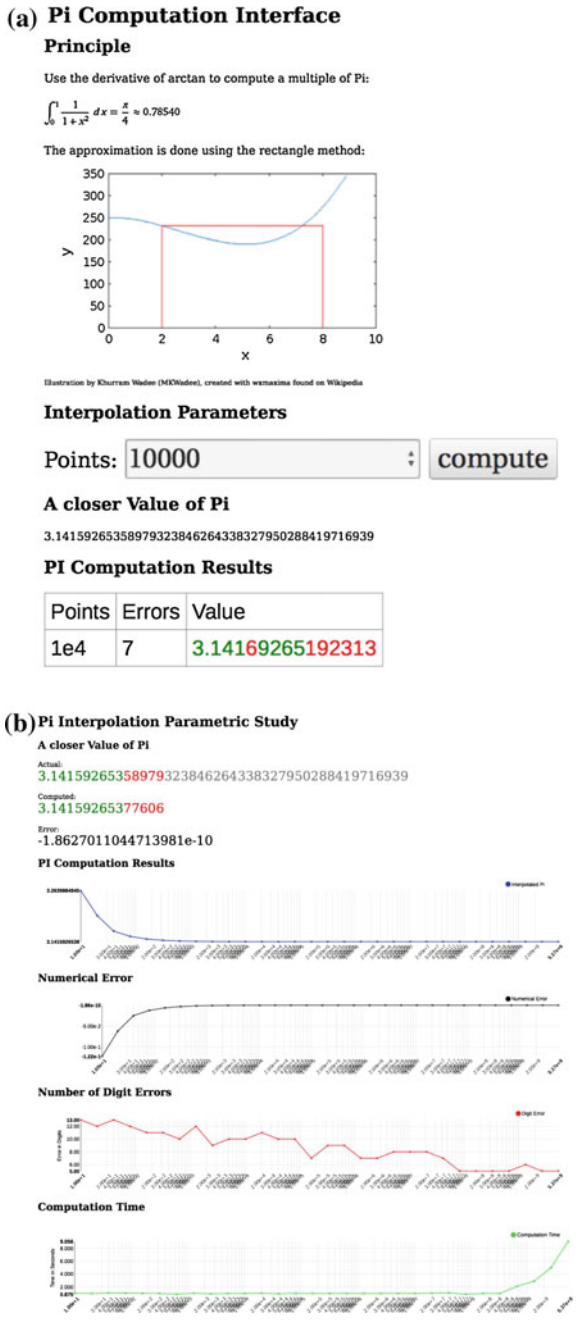

Fig. 5.4 Overview of our on-line application management interface for π computation. **a** Interactive computation interface. **b** Computation dashboard

rank zero stores the approximated π value in the cache, event which is notified in the server. Eventually, on client side, a JavaScript function periodically queries current value, displaying it in a color-coded fashion with correct digits in green and bad ones in red. This first example demonstrated bidirectional online coupling with a running application. The second example of Fig. 5.4b uses a similar approach, however, instead of waiting for the user to input a number of rectangles, a script in the browser does a parametric analysis with an increasing number to study the convergence towards π. In complement of the colored value, we present dynamic curves respectively displaying successive approximations, relative error, number of false digits and computation time.

This example, clearly out of the scope of a profiling tool was chosen to illustrate the potential shifting that exists from complex data-management approaches developed for profiling to the more general problematic of managing code outputs. We shown that starting from our online profiling engine we were able to implement with a reduced development effort an interface providing new interaction opportunities with running applications. We are convinced that hybrid data-management techniques will play an important role in the usability of Exascale machines. Justifying in our opinion the addition of primitives allowing such behavior in runtimes or programming interfaces such as MPI as we further detail in the conclusion.

5.5 Conclusion an Future Work

As discussed in Sect. 5.1.5, we think that interactions with parallel applications will benefit from a *multi-scale* approach, combining all the coupling paradigms that we covered in our introductory content. To do so, suitable mechanisms have to be defined at programming interface level in order to pave the way for such online interactions. We are currently working on MALP capabilities transposition as extensions to the MPI standard.

Dealing with MPI virtualization [5, 22], the standard is already able to join disjoint MPI applications using MPI_Comm_connect, creating an inter-communicator with the exact same features than `MPI_COMM_UNIVERSE`. It is then possible to achieve the virtualized MPI behavior with current standard, connecting several applications or services. As far as the data-stream interface is concerned, it can also be implemented inside MPI as we already did it using only standard calls [5]. However, one point that we see as missing in MPI is *query* support. Indeed, for performance reason, MPI mostly provides either paired communications or RDMAs but no easy manner to implement queries which could be modeled as Remote Procedure Calls (RPC). For us, the ability to *opportunistically* interact with a remote process is a key component. Using current standard, one would need to have a loop doing MPI_Recv, `ANY_SOURCE` to process incoming request which would then be processed by an arbitrary function before returning a result. To do so, the application has to handle a pool of threads processing such messages while making sure that the underlying runtime supports the `THREAD_MULTIPLE` level. If handled by the runtime the use of

RPCs would become more convenient and more importantly standardized between applications, allowing transversal tool services definition.

Such interface has already been investigated in the MPI context and called *active messages* [8], such messages were implemented in AMMPI and AM++ [25] but not standardized yet. One could imagine with such interface, querying performance data from a given process, sending a command to a simulation, initiating a two sided communication phase by creating the remote context through an RPC for in-situ visualization, load-balancing tasks on a set of nodes and gathering results, … In this purpose, we propose a similar RPC interface for *service* abstraction, going a bit further than Active Messages [18] by proposing a global naming (URI and domain):

```
/* Emit a Query to a remote process or a global URI */
MPIX_Query( int domain, char * uri,
            void * post_buff, size_t post_size,
            void ** get_buff, size_t ** get_size,
            MPI_Communicator comm, MPI_Request * req );
/* Register a function to fulfil global or local queries at a
    given URI */
MPIX_Bind( int domain, char * uri,
           int (*mpi_rpc)( void * post_buf,
                           size_t post_size,
                           MPI_Query * handle ));
/* Return a result using a query context */
MPIX_Query_ret(  MPI_Query * handle,
                 void * get_buff,
                 size_t get_size );
/* Return an error code using a query context */
MPIX_Query_error(  MPI_Query * handle,  int code );
```

This interface would provide query support in MPI through the MPIX_Query function call. Either locally or globally using respectively the `MPI_SCOPE_LOCAL` and `MPI_SCOPE_GLOBAL` keywords as `domain` parameter. The `uri` defines the resource to be queried (just like in an HTTP request) and the other parameters define both what is sent in order to be passed as argument in the remote function and the buffer which is returned (note that it is allocated) from the call. Eventually the query can be waited as usual with a request. In order to expose an RPC service, a process has to call the MPIX_Bind function which takes a `domain` argument (local or global), an `uri` to be matched by the calling RPC and a function associated with the RPC. Eventually, we introduce two functions to be used to return a result from an RPC, to do so, they rely on the `MPI_Query` context passed in argument of the RPC handler. The function can then either return an arbitrary buffer using MPIX_Query_ret or an error code (reported on the caller) using MPIX_Query_error.

This small MPI interface extension, is what we consider to be an important component to enable more orthogonality between applications, exposing several data-sources even from pre-loaded libraries through the same query interface. This leads to a more orthogonal vision of an MPI computation with a convenient expression of the master–slave/client–server model which is not present yet in the standard being mostly used for SPMD computation. Thanks to this data-model, it will be much easier to have several MPI applications collaborating around *services* which could be exposed using this interface through an horizontal vision of the parallel computation with IOs, visualization, profiling, logging, output-management and computation processes collaborating in an online fashion, as a *constellation of services*.

References

1. Adhianto L, Banerjee S, Fagan M, Krentel M, Marin G, Mellor-Crummey J, Tallent NR (2010) HPCToolkit: tools for performance analysis of optimized parallel programs. Concurr. Comput.: Pract. Exp. 22(6):685–701
2. Arnold, D.C., Ahn, D.H., de Supinski, B.R., Lee, G.L., Miller, B.P., Schulz, M.: Stack trace analysis for large scale debugging. In: IEEE International Parallel and Distributed Processing Symposium, IPDPS. pp. 1–10. IEEE (2007)
3. Benedict, S., Petkov, V., Gerndt, M.: Periscope: an online-based distributed performance analysis tool. Tools for High Performance Computing 2009, pp. 1–16. Springer, Berlin (2010)
4. Besnard, J.B.: Profiling and Debugging by Efficient Tracing of Hybrid Multi-Threaded HPC Applications. Ph.D. thesis, Université de Versailles Saint Quentin en Yvelines (2014)
5. Besnard, J.B., Pérache, M., Jalby, W.: Event streaming for online performance measurements reduction. In: 42nd International Conference on Parallel Processing (ICPP), pp. 985–994. IEEE (2013)
6. Chan A, Gropp W, Lusk E (2008) An efficient format for nearly constant-time access to arbitrary time intervals in large trace files. Sci. Program. 16(2–3):155–165
7. Crockford, D.: The Application/Json Media Type for Javascript Object Notation (json) (2006)
8. von Eicken, T., Culler, D.E., Goldstein, S.C., Schauser, K.E.: Active messages: a mechanism for integrated communication and computation. In: Proceedings of the 19th Annual International Symposium on Computer Architecture, ISCA '92, pp. 256–266. ACM, New York, NY, USA (1992). http://doi.acm.org/10.1145/139669.140382
9. Eschweiler D, Wagner M, Geimer M, Knüpfer A, Nagel WE, Wolf F (2011) Open trace format 2: the next generation of scalable trace formats and support libraries. PARCO. 22:481–490
10. Frings, W., Wolf, F., Petkov, V.: Scalable massively parallel i/o to task-local files. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, pp. 1–11. IEEE (2009)
11. Gansner ER, North SC (2000) An open graph visualization system and its applications to software engineering. Soft. -Pract. Exp. 30(11):1203–1233
12. Geimer M, Wolf F, Wylie BJ, Ábrahám E, Becker D, Mohr B (2010) The scalasca performance toolset architecture. Concurr. Comput. Pract. Exp. 22(6):702–719
13. Hilbrich, T., Müller, M.S., de Supinski, B.R., Schulz, M., Nagel, W.E.: GTI: A generic tools infrastructure for event-based tools in parallel systems. In: IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS), pp. 1364–1375. IEEE (2012)
14. Hilbrich, T., Schulz, M., de Supinski, B.R., Müller, M.S.: MUST: A Scalable approach to runtime error detection in MPI programs. Tools for High Performance Computing 2009, pp. 53–66. Springer, Berlin (2010)
15. Knüpfer, A., Brendel, R., Brunst, H., Mix, H., Nagel, W.E.: Introducing the open trace format (OTF). Computational Science–ICCS 2006, pp. 526–533. Springer, Berlin (2006)
16. Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Müller, M.S., Nagel, W.E.: The vampir performance analysis tool-set. Tools for High Performance Computing, pp. 139–155. Springer, Berlin (2008)
17. Knüpfer, A., Rössel, C., an Mey, D., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A., et al.: Score-P: a joint performance measurement run-time infrastructure for periscope, scalasca, TAU, and vampir. Tools for High Performance Computing 2011, pp. 79–91. Springer, Berlin (2012)
18. Mainwaring, A.M., Culler, D.E.: Active message applications programming interface and communication subsystem organization. Technical Report UCB/CSD-96-918, EECS Department, University of California, Berkeley (Oct 1996). http://www.eecs.berkeley.edu/Pubs/TechRpts/1996/5768.html
19. Nataraj, A., Malony, A.D., Morris, A., Arnold, D., Miller, B.: A framework for scalable, parallel performance monitoring using TAU and MRnet. In: International Workshop on Scalable Tools for High-End Computing (STHEC 2008), Island of Kos, Greece (2008)

20. de Oliveira Stein, B., de Kergommeaux, J.C., Mounié, G.: Pajé Trace File Format. Technical report, ID-IMAG, Grenoble, France, 2002. http://www-id.imag.fr/Logiciels/paje/publications (2010)
21. Roth, P.C., Arnold, D.C., Miller, B.P.: MRNet: A Software-based multicast/reduction network for scalable tools. In: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, p. 21. ACM (2003)
22. Schulz, M., de Supinski, B.R.: P^nMPI tools: a whole lot greater than the sum of their parts. In: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, p. 30. ACM (2007)
23. Shende SS, Malony AD (2006) The TAU parallel performance system. Int. J. High Perform. Comput. Appl. 20(2):287–311
24. Vetter, J., Chambreau, C.: MPIP: Lightweight, scalable MPI profiling (2005)
25. Willcock, J.J., Hoefler, T., Edmonds, N.G., Lumsdaine, A.: AM++: a generalized active message framework. In: Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques, PACT '10, pp. 401–410. ACM, New York, NY, USA (2010). http://doi.acm.org/10.1145/1854273.1854323
26. Zaki O, Lusk E, Gropp W, Swider D (1999) Toward scalable performance visualization with jumpshot. Int. J. High Perform. Comput. Appl. 13(3):277–288