Ocelotl: Large Trace Overviews Based on Multidimensional Data Aggregation

Damien Dosimont, Youenn Corre, Lucas Mello Schnorr, Guillaume Huard and Jean-Marc Vincent

Abstract Performance analysis of parallel applications is commonly based on execution traces that might be investigated through visualization techniques. The weak scalability of such techniques appears when traces get larger both in time (many events registered) and space (many processing elements), a very common situation for current large-scale HPC applications. In this paper we present an approach to tackle such scenarios in order to give a correct overview of the behavior registered in very large traces. Two configurable and controlled aggregation-based techniques are presented: one based exclusively on the temporal aggregation, and another that consists in a spatiotemporal aggregation algorithm. The paper also details the implementation and evaluation of these techniques in **Ocelotl**, a performance analysis and visualization tool that overcomes the current graphical and interpretation limitations by providing a concise overview registered on traces. The experimental results show that Ocelotl helps in detecting quickly and accurately anomalies in 8 GB traces containing up to two hundred million of events.

Damien Dosimont, Youenn Corre, Guillaume Huard, Jean-Marc Vincent
Inria
Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France
CNRS, LIG, F-38000 Grenoble, France
e-mail: damien.dosimont@imag.fr
youenn.corre@inria.fr
guillaume.huard@imag.fr
jean-marc.vincent@imag.fr

Lucas Mello Schnorr
Informatics Institute, UFRGS, Porto Alegre, Brazil
e-mail: schnorr@inf.ufrgs.br

1 Introduction

Performance analysts commonly use execution traces collected at runtime to understand the behavior of applications running on distributed and parallel systems. These traces are then inspected post mortem using various interactive visualization techniques. They are used by performance analysts to detect unexpected behavior, correlating the events to the application source code and its components, execution settings or the configuration of the runtime platform. A very common example of visualization technique is the Gantt-like space/time visualization, which represents the behavior of the application components along a time axis. The problem is that such visualization technique is incapable to scale properly for an increasing number of events. This issue is mainly due to the bounded screen resolutions, preventing the proper drawing of many graphical objects (see Figure 1 for an example). Another problem is the natural limitation of human perception, which cannot handle too much irregular information. From the technical side, long processing time and interaction delay, caused by the trace event handling, may hinder and discourage the analyst to conduct further behavior investigation.

We present **Ocelotl**, a tool that overcomes the graphical and interpretation limitations of current visualization techniques. It provides a concise overview of the trace behavior, as the result of a user-configurable and controlled data aggregation process. The aggregation process gathers parts of the trace where behavior is homogeneous. Such homogeneity is determined by a user-configurable trade-off between the information lost by the aggregation process, and the complexity reduction of the visual representation. The user dynamically adjusts the balance between both metrics to get a level of details that helps him understand the application behavior.

We propose two different trace visualization overviews based on this trade-off principle: a temporal and a spatiotemporal-based approach. In the former, the aggregation is applied exclusively on the time dimension. The obtained aggregated view is mainly aimed at analyzing applications expected to be regular along time. Examples of such applications are the software of multimedia players or algebraic computation: we easily highlight temporal perturbations and phases on these use-cases. The later, spatiotemporal, is when the aggregation is mutually applied in time and space dimensions. By space we mean the hardware and software component hierarchy. This second technique is efficient to detect problematic behaviors that touch only a subset of resources, which is typical of network sharing issues or unbalanced performances of machines and clusters running a parallel application.

Because of the computational cost derived from the efficient implementation of these techniques, we have designed a clever reuse of results provided by previous analysis sessions. It helps to drastically reduce the computation time and improve the interaction responsiveness and the general utilization of the tool. The experimental results show that Ocelotl helps to quickly and accurately detect anomalies in 8 GB traces containing up to two hundred million of events.

This text is organized as follows. Section 2 presents the state-of-the-art on performance analysis visualization tools, with a focus on the relationship with visual and semantic aggregation mechanisms that are present on related work. Section 3

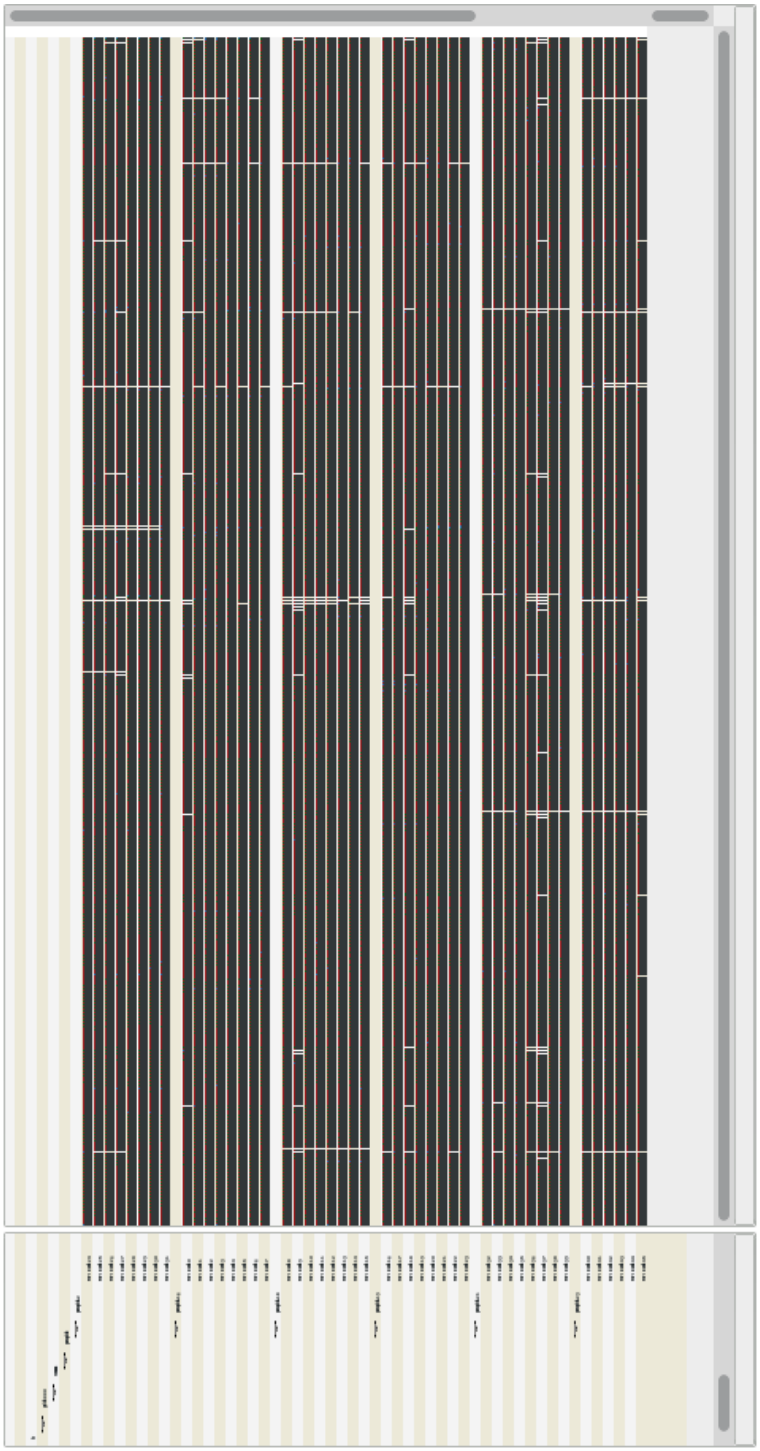


Fig. 1 Example of a cluttered space/time view that is incapable to show correctly a very large trace over temporal and spatial dimensions

reminds two aggregation methodologies based on a trade-off between information loss and complexity reduction that were described in previous documents. Section 4 presents the Ocelotl performance visualization tool that implements the overviewing techniques of our approach. Section 5 closes the paper with a work summary and future work.

2 Trace Overview Common Issues

Different analysis tools propose several methods to improve scalability and provide decent overviews over time and space. In this section, we describe several common issues related to the techniques they employ.

2.1 Pixel Guided Representations

Pixel-guided representations, present in some tools, like Vampir or Paraver [6, 14], associate each screen pixel to a set of data. As the pixel is incapable of representing all the information, the rendering algorithm decides what is shown or hidden. We claim that this aggregation process misguides the user: a pixel might correspond to a raw data or be the aggregation of several data. Moreover, some tools do not provide clear indication about the aggregation operation used to render the pixel. Pixel-guided representations also suffer from a fidelity issue: for instance, resizing the window may modify strongly the visualization content because the pixel allocation changes.

2.2 Visual Aggregation

In a visual aggregation, the rendering tries to preserve the graphical object scales, whose size depends on their contents. When it is impossible, e.g. when the size is less than one pixel, it generates aggregates gathering close objects. In Pajé [5] and LTTng Eclipse Viewer [1], such aggregates are just used to avoid visual clutter, but do not represent the data they contain.

2.3 Spatiotemporal Entity Budget Management

Techniques based on Gantt Chart-like space-time diagrams [6, 14, 5, 1, 15, 12] have an issue to keep a reasonable budget of graphical entities present on the screen. Indeed, since the temporal and the spatial dimensions behave differently, visual-

ization techniques do not treat both axis the same way. KPTrace Viewer [15], for instance, proposes an interactive hierarchical aggregation for the space dimension and a mechanism that compresses time whenever an event lasts too long. However, if the spatial reduction technique is efficient, the time compression does not manage sufficiently the time entity budget. Other space-time tools [6, 14, 5, 1, 12] lack reduction technique on space and disrespect the vertical budget, which forces the user to scroll over y-axis and prevents him from getting a spatial overview.

2.4 Monodimensional Representations

Conversely, other techniques [6, 16] propose unidimensional overviews that better manage the entity budget. Vampir's task profile [6] clusters the most similar processes according to a distance measure based on the duration of the functions executed by each process. However, the temporal dimension is lost in the process, and this technique is thus more adapted for profiling. One of our previous works, Viva [11], provides a treemap view showing the hardware and software component hierarchy. Entities having the most homogeneous behavior are aggregated using a compromise between the representation complexity and the information loss induced by the aggregation. This technique highlights troubles characterized by an heterogeneous behavior. However, time dimension is missing from the representation, although it is used to compute the entities values through a time integration. It is thus difficult to find local temporal perturbations, and this technique is thus mainly aimed at finding bottlenecks that concern the whole execution.

3 Provide a Trace Overview Using Data Aggregation

In this section, we remind two applications of an aggregation methodology we already described in previous documents: temporal [4, 3, 13] and spatiotemporal [2] data aggregations that provide overviews of the trace and address issues evoked in Section 2. For each technique, we designed an algorithm which gathers the parts of the trace where the behavior is homogeneous. The aggregation strength is tuned thanks to a parameter set by the user, which balances a trade-off between the information lost by the aggregation process and the complexity reduction compared to a representation without aggregation. We also designed visualization techniques to represent the algorithm output. They may involve visual aggregation to help providing a clean representation of the algorithm output. With the set of techniques we use to build our overviews, we reach the objective of reducing the complexity of the representation while keeping important information about the application behavior. We thus address in priority the screen limitations and the analyst understanding issues. Nevertheless, the complexity of our algorithms enables to keep reasonable compu-

tation times, and ensures a good interactivity. Both overviews are implemented in the Ocelotl analysis tool, which we describe in Section 4.

3.1 Aggregation Methodology

The *analytic approach* consists in analyzing separately each constituent of a system. However, when the system contains too many entities, it is not possible to understand its global behavior from the behavior of each of its components. On the opposite, the *systemic approach* consists in analyzing the system globally, using a macroscopic point of view, in order to provide the analyst with a quantity of information he is able to work with. Lamarche-Perrin proposes an aggregation methodology [10, 11, 9, 8], based on the systemic approach, that we follow to build our overview techniques. Since this methodology is generic, we can adapt it to trace analysis.

In order to build the macroscopic overview that represents the behavior of our whole system, we start by observing the system from a microscopic point of view. This observation generates microscopic data, which are, in principle, too complex and too numerous to be analyzed using the analytic approach. Collected information are organized using data structures and metrics which can describe the system behavior. In particular, data are structured over temporal and spatial discrete dimensions.

The microscopic data can be considered as a set of entities x. The aggregation process, which leads to a macroscopic point of view from these microscopic data, is constituted by two steps. First, we partition the entities x, which gives us a partition $\boldsymbol{P}$ of the microscopic model. We name $\mathscr{P}$ the set of all possible partitions $\boldsymbol{P}$. Then, we generate a result value V_X associated with each part $X \in \boldsymbol{P}$ using an aggregation operator. In order to give meaning to this process, we propose several steps to fulfill:

1. Choose the dimensions where to perform the aggregation. This choice depends on the behavior to study and the visualization that is planned. In this step, we also structure these dimensions (ordered set, hierarchy, etc.).
2. Choose the operand, e.g. the entities that can be aggregated. In our case, it corresponds to the entities x.
3. Constrain the aggregation: determine the set $\mathscr{P}$ of partitions that are allowed. This constraint has several roles. First, it forbids aggregations that cannot translate into an understandable behavior (for instance, in the case of a temporal analysis, we forbid the aggregation of two discontinuous time part). Second, it insures coherence with the structure of the dimensions (for instance, for a hierarchy, we aggregate the nodes into their parents). And last, it reduces the number of possible partitions to limit the aggregation algorithm complexity.
4. Choose the aggregation operator that will generate the aggregation result. This operator can be a mathematical function but also a visual technique.
5. Choose a condition, provided by the user or the environment, which determines how to aggregate the system, i.e. the way the microscopic model is partitioned.

In our methodology, we will use systematically the sum operation to fulfill the step 4: $V_X = \sum_{x \in X} v_x$

3.2 Trigger the Aggregation with Information Loss and Complexity Reduction

A particularity of Lamarche-Perrin methodology is the use of information theory concepts to build the condition triggering the aggregation (step 5). To bring meaning to the analyst, we want to aggregate the microscopic model by gathering in priority the most homogeneous spatiotemporal areas, while preserving data about the heterogeneous ones. In this way, aggregation consists in optimizing a trade-off between data reduction and information loss.

Measures have been proposed in previous work [10, 11, 4, 3, 13] to express such a trade-off. For each part X obtained from a partition $\boldsymbol{P}$ of our microscopic model, we associate measures of information loss (*loss*), data reduction (*gain*) and a *parametrized Information Criterion* (pIC). We use Kullback-Leibler divergence [7] as a measure of information loss:

$$\text{loss}_X = \sum_{x \in X} v_x \log_2 \left(\frac{|X| v_x}{V_X} \right) \tag{1}$$

Shannon entropy [17] as a measure of data reduction,

$$\text{gain}_X = V_X \log_2 V_X - \sum_{x \in X} v_x \log_2 v_x \tag{2}$$

and define the *parametrized Information Criterion* [10] as follow:

$$\text{pIC}_X = p\,\text{gain}_X - (1-p)\,\text{loss}_X \tag{3}$$

where $p \in [0,1]$ is the parameter used to balance this trade-off.

These measures are additive. The gain, the loss and the pIC associated with a partition $\boldsymbol{P}$ is defined as the sum of the gain, the loss or the pIC of the parts X that compose it. Finally, we define the optimal partition for a given parameter p as the one that has the highest pIC. This process tends to select partitions that aggregates in priority entities with the closest associated values, since it enables to reduce data complexity while minimizing information loss. For $p = 0$, the analyst wants to be as accurate as possible (the microscopic partition is optimal) and, for $p = 1$, he wants to be the simplest (the full aggregation is optimal). When p varies from 0 to 1, a whole class of nested representations arises. The choice of this parameter is deliberately left to the analyst, so he can adapt the entity budget to the analysis purposes.

3.3 Trace Microscopic Models

In order to apply Lamarche-Perrin methodology to trace analysis, it is mandatory to generate a microscopic model structured by discrete dimensions, and providing a metric that can represent the behavior an analyst wants to analyze. This microscopic model is thus generated from the raw trace and can be considered as a first abstraction of it.

We formalize our microscopic models as matrices involving temporal dimension, spatial dimension, and event type dimension.

- The *time dimension* is defined as the set $T = \{t_1, \ldots, t_n\}$ of the observed microscopic time periods. This dimension is discrete, whereas the raw trace time is continuous. To fit with the microscopic model, the raw trace is divided in $|T|$ (regular) time periods and the events are associated with the periods where they are active. Each period t has a duration $d(t) \in \mathbb{R}^+$ and the whole set is naturally ordered by "the arrow of time". Formally, a total order $<$ on T provides the concept of interval: $T_{(i,j)} = \{t \in T \mid t_i \leq t \leq t_j\}$ with $t_i \leq t_j$. We mark $\mathscr{I}(T)$ the set of intervals of T.
- The *spatial dimension* is defined as the set $S = \{s_1, \ldots, s_m\}$ of the platform microscopic resources. For the computing platforms we are interested in, this set has a hierarchical structure: resources are organized in processes, running on cores, each one being associated with a machine, themselves organized in clusters, and so on. Formally, a *hierarchy* is a set $\mathscr{H}(S) = \{S_1, \ldots, S_p\}$ of subsets of S that contains the whole resource set ($S \in \mathscr{H}(S)$), each singleton ($\forall s \in S,\ \{s\} \in \mathscr{H}(S)$), and such that any two parts in $\mathscr{H}(S)$ are either disjoint or included one in another ($\forall (S_i, S_j) \in \mathscr{H}(S)^2$, either $S_i \cap S_j = \emptyset$ or $S_i \subset S_j$ or $S_i \supset S_j$). A hierarchy is thus equivalent to a *rooted tree* where the nodes correspond to these hierarchical subsets, the *leaves* corresponds to the singletons, the *root* to the whole set, and the *tree-order* to the subset relation.
- The *event type* dimension is defined as the set $U = \{u_1, \ldots, u_l\}$ of the different possible types of events that occur during the trace execution.

We define our microscopic models as 3D matrices M^{μ} with dimensions $|S| \times |T| \times |U|$, and we name $v_{(s,t,u)}$ each entry associated with coordinates (s,t,u)

We propose different metrics that can be used to fill the microscopic model from the data contained in the raw trace:

- *Event occurrence number* corresponds to the number of events of a certain type u that occurs during a time interval t and generated by a resource s. This is the simplest approach.
- *State duration* corresponds to the total time passed in a particular state type u (a state is an event, like a function call, associated with start and end timestamps) over a time interval t, for a given resource s.
- *Variable mean* is the mean value of a variable associated with a resource s over a time interval t, and of a given type u.

3.4 Temporal Overview

In order to detect temporal anomalies in the application behaviors, our first contribution was a temporal overview for trace analysis fitting with Lamarche-Perrin methodology.

3.4.1 Methodology Application

1. We aggregate over time dimension.
2. The operands are the time periods $t \in T$. For each time period $t \in T$, we associate a value constituted by the sub matrix $M^{\mu}(t)$, with the dimensions $|S| \times |U|$.
3. We constrain the aggregation by allowing only the aggregation of contiguous time periods ($\mathscr{I}(T)$), without overlapping.
4. The result of time period aggregation is the sum of their associated matrices: $M^{\mu}(T_{(i,j)}) = \sum_{t \in T_{(i,j)}} M^{\mu}(t)$.

The algorithms and complexity measure calculation are presented in our previous works [4, 3, 13]. The aggregation steps are the following: the algorithm first calculates the quality measures of each possible time interval. Then, the user provides a value for p. The aggregation algorithm gives in return the best partition associated with p. To do that, we use dynamic programming and take advantage of the partition lattice structure and the additivity of their gain, loss and pIC, which helps us to decompose this set partition problem.

3.4.2 Visualization

The algorithm output is the set of aggregated time periods as a function of the parameter p. In order to provide a visualization that brings meaning to the user, we show a stacked bar chart, where the time is used as abscissa. Each stack of rectangles is associated with a temporal aggregate, organized following the same order over time. Its width is determined by the number of time slice aggregated. Each layer is associated with a type u. Its amplitude is computed as followed:

$$Q(T_{(i,j)}, u) = \frac{\sum_{t \in T_{(i,j)}} \sum_{s \in S} v_{(s,t,u)}}{j - i + 1} \tag{4}$$

In the case where layers are too small to be printed correctly, e.g., their sizes are less than one pixel, we visually aggregate them and show them differently.

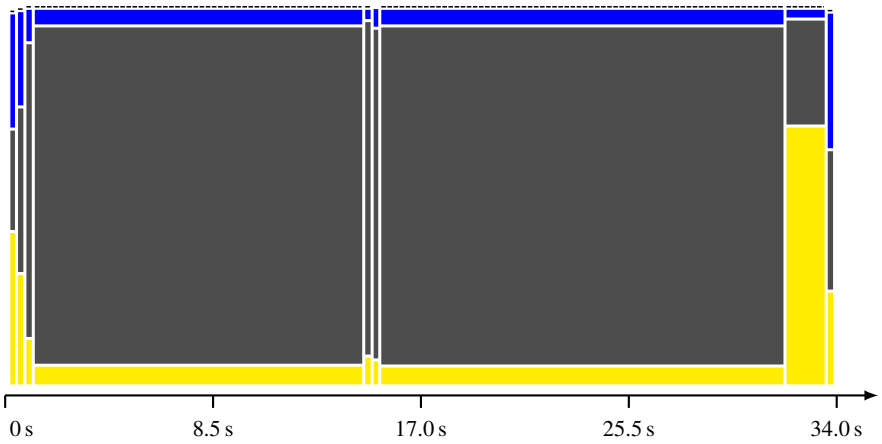


Fig. 2 Temporal overview showing a GStreamer multimedia application perturbed by a CPU stress (around 15 s). State colors correspond to the following GStreamer debug levels: Grey - `LOG`, Blue - `DEBUG`, Yellow - `WARNING`

3.4.3 Examples

In our previous works [4, 3, 13], we describe examples of applications where the temporal overview shows anomalies. For instance, we perturbed a GStreamer multimedia application playing a video with a CPU stress. The perturbation is easily detectable with Ocelotl (Figure 2). We also analyzed MPI applications where we expected a regular temporal behavior. We were able to detect some perturbations provoked by a concurrency access to the network with other user running applications on the same platform (Figure 3 and 6).

3.5 Spatiotemporal Overview

Temporal overview can show temporal anomalies, but is limited to represent phenomena that also involve the application structure. This issue is at the bottom of the design of a new overview, showing both space and time.

3.5.1 Methodology Application

1. We aggregate **simultaneously** over time and space dimension.
2. The operands are the area determined by the Cartesian product of the time periods $t \in T$, and space elements $s \in S$. For each couple (s,t), the associated metric is the row matrix $M^{\mu}(s,t)$ with the dimension $|U|$.

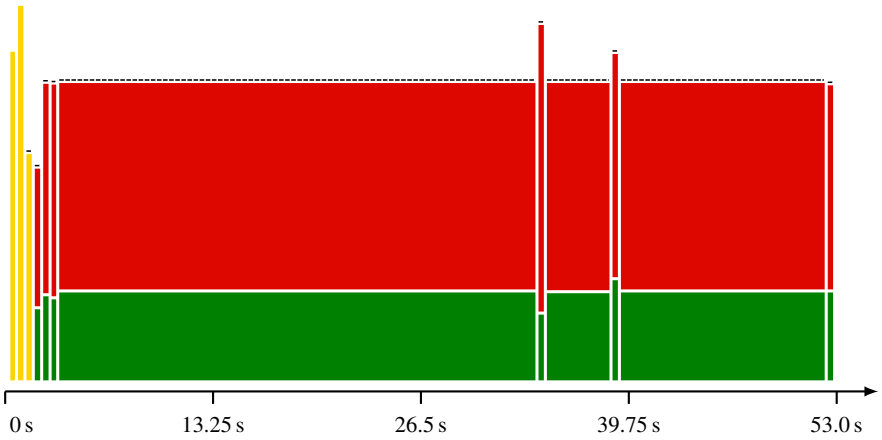


Fig. 3 Temporal overview showing a MPI application (NASPB CG class C, 64 processes) with two perturbations (around 35 and 40 s) provoked by a concurrency access to the network. State colors correspond to following the MPI calls: Yellow - `MPI_Init`, Green - `MPI_Send`, Red - `MPI_Wait`

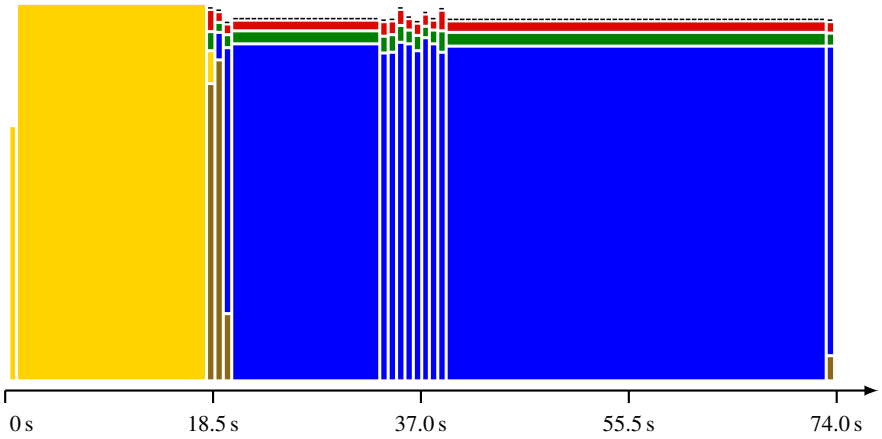


Fig. 4 Temporal overview showing a MPI application (NASPB LU class C, 700 processes) with one perturbation (37 s) provoked by a concurrency access to the network. State colors correspond to the following MPI calls: Yellow - `MPI_Init`, Brown - `MPI_Allreduce`, Green - `MPI_Send`, Red - `MPI_Wait`, Blue - `MPI_Recv`

3. We constrain the aggregation as the Cartesian product of the contiguous time periods $\mathscr{I}(T)$ and the hierarchy $\mathscr{H}(S)$, without overlapping.
4. The result of spatiotemporal elements aggregation is the sum of the associated vectors: $M^{\mu}(S_k, T_{(i,j)}) = \sum_{s \in S_k} \sum_{t \in T_{(i,j)}} M^{\mu}(s,t)$.

The corresponding algorithm and complexity measure calculation are presented in our previous works [2]. Similarly to what we did for temporal aggregation, we first compute the quality measures, but this time, for each time interval associated with an element of the hierarchy (leaves or nodes). Then, the user provides a value for p and the algorithm returns the best associated partition. The algorithm takes advantage of the same properties than for the temporal aggregation (dynamic programming, lattice structure, quality measure additivity). It is mainly constituted by iterative-recursive sequences searching successively for spatial partitions and temporal cuts.

3.5.2 Visualization

We represent as algorithm output the spatiotemporal area that are aggregated, as a function of a parameter p. With each area, we associate a color, that corresponds to the event type $u_{mode} \in U$ such as:

$$M^{\mu}(S_k, T_{(i,j)}, u_{mode}) = \max_{u \in U} \left(\frac{\sum_{t \in T_{(i,j)}} \sum_{s \in S_k} v_{(s,t,u)}}{|T_{(i,j)}| \times |S_k|} \right) \tag{5}$$

When the number of resources $|S|$ is greater than the amount of pixels in the spatial axis, we use visual aggregation (during rendering) to keep a clean visualization: if an aggregate has a visual height inferior to a threshold (in pixels), its parent in the spatial hierarchy is drawn instead. Visually-aggregated areas are marked differently: by a diagonal line, if underlying resources have the same temporal data partitioning, or by a cross otherwise. Figure 6 shows these different aggregation patterns.

3.5.3 Example

We mainly provide MPI application use cases to show the relevancy of our spatiotemporal overview technique. In these cases, we still show temporal perturbations that can be detected with the temporal algorithm (Figure 5), but also spatial heterogeneity that indicates that resources behave differently, which can be a problem (Figure 6).

4 Ocelotl Tool

In this section, we present Ocelotl, an analysis tool that implements the overviews presented in Section 3. We first describe its global architecture. In particular, we give details about the interaction features that provide the capability of a dynamic analysis. We then insist on the interactivity and the performance by showing techniques we use to reduce the computation times.

Fig. 5 Spatiotemporal overview of the application shown by Figure 3. We still see the two temporal perturbations (around 35 and 40 s). State colors correspond to following the MPI calls: Yellow - `MPI_Init`, Green - `MPI_Send`, Red - `MPI_Wait`

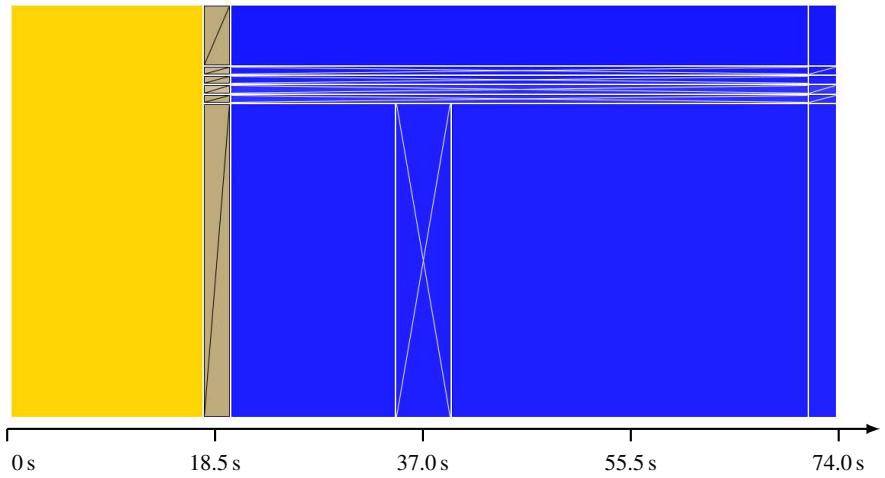


Fig. 6 Spatiotemporal overview of the application shown by Figure 4. We find the temporal perturbation again (37 s) but now, we can see that it concerns only one cluster. Moreover, another cluster behaves heterogeneously during the computation phase. State colors correspond to the following MPI calls: Yellow - `MPI_Init`, Brown - `MPI_Allreduce`, Blue - `MPI_Recv`

4.1 Global Architecture

Ocelotl is written in Java and is an Eclipse plug-in. It is a module of Framesoc [3], a trace and tool manager and analysis infrastructure. Framesoc provides an API to access traces, stored as data bases using a generic data model. Each trace is imported

only once into Framesoc, and remains accessible for the following analysis sessions. Framesoc also provides a graphical interface and a bunch of tools (statistics, Gantt chart-like space-time diagram) classically used in the trace analysis domain.

4.1.1 Component Organization

Ocelotl is composed of a core and a graphical interface. The part related to the aggregation algorithm is a shared library we have written in C++ and interfaced through JNI (*Java Native Interface*). The libraries are named lpaggreg and lpaggregjni. This design choice was motivated by performance reasons (computation time, accurate memory management).

The core is responsible of the computations. The graphical interface provides various settings to the user. An area is dedicated to the rendering of the algorithm output. Another one is used to represent gain and loss curves that help to browse through the different aggregations. A screenshot of the GUI is shown in Figure 7.

Ocelotl is relatively modulable ; we provide three types of extension points:

- *microscopic model*: an extension that generates a microscopic model as a matrix over time, space and type dimensions. The currently implemented metrics to fill the matrices are the *event occurrence number*, the *state duration*, or the *variable mean* (cf. Section 3.3). Other metrics can be added by providing new extensions using the interface we provide.
- *aggregation type*: an extension that determines on which dimension aggregate and which algorithm of the C++ library is chosen. We currently propose *temporal* and *spatiotemporal* aggregations. As for microscopic models, new extensions can be added.
- *visualization*: an extension to draw the algorithm output. For the temporal aggregation, we draw the output as a simple partition using colored rectangles or as a stacked bar chart (cf. Section 3.4.1). For the spatiotemporal aggregation, we represent the aggregated area, or the mode (cf. Section 3.5.1). Other visualizations can also be added.

These extensions are dynamically detected when Ocelotl is launched. The user chooses, thanks to combo boxes, which microscopic model, which aggregation type, and which visualization he wants.

4.1.2 Pipelined Analysis Process

The analysis process is a pipeline composed of different steps:

1. The analyst tunes the different settings (choice of the trace, microscopic model, aggregation type, visualization, time bounds, resources and event types, time slice number and so on.) in the graphical interface. Extensions can also have their own settings. Once done, the user launches the analysis.

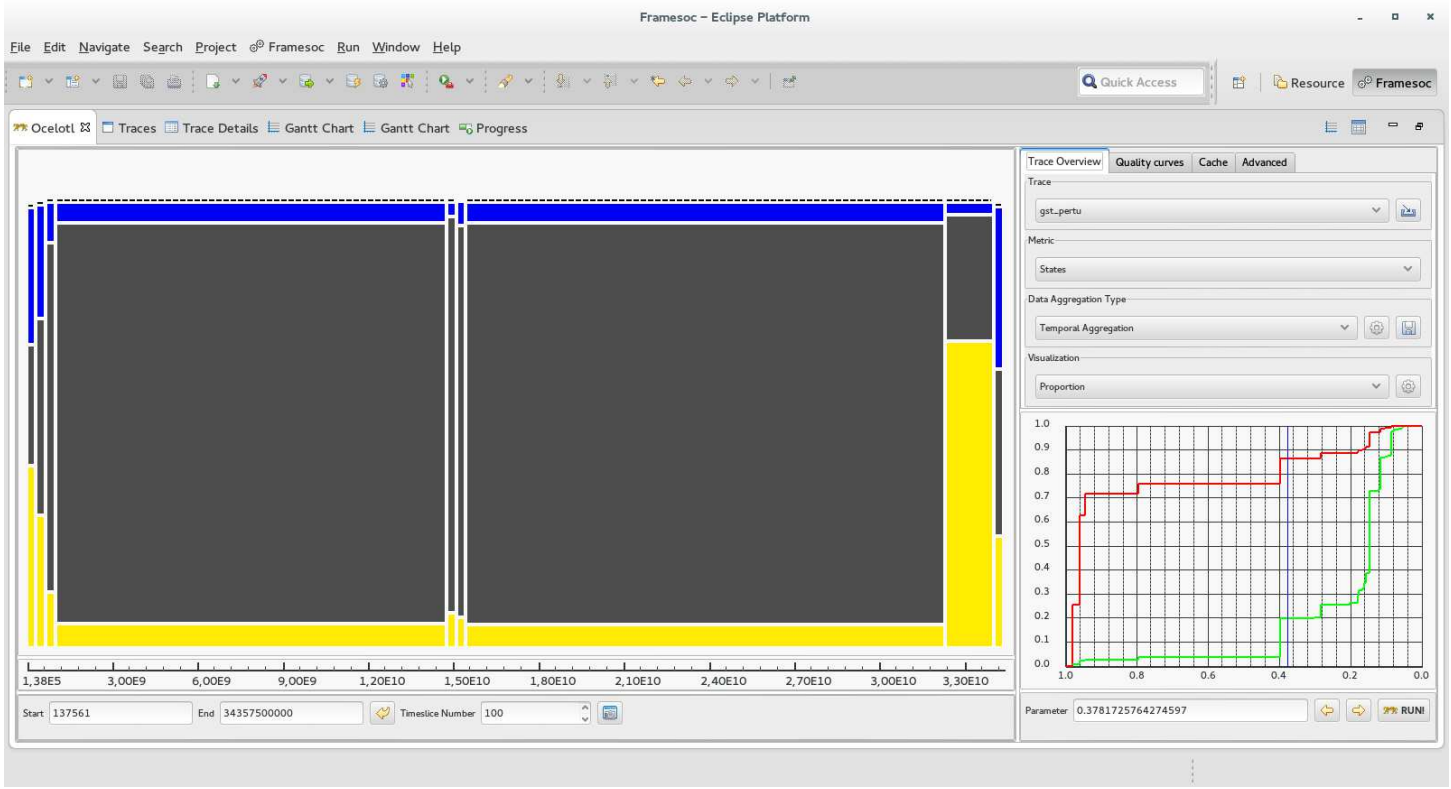


Fig. 7 Screenshot of Ocelotl showing its graphical interface. On the left, the overview composed by a temporally aggregated visualization. On the bottom right, curves show the complexity of the different possible representations (in green) and the information quantity contained inside (in red).

2. Ocelotl generates a query using the parameters defined by the user, provides it to Framesoc and gets a result set containing the events matching with the query.
3. Ocelotl fills the microscopic model, whose dimensions are determined by step 1 settings, while reading the result set. This process is multithreaded.
4. Ocelotl provides the microscopic model to the aggregation algorithm library.
5. Ocelotl calls the quality computation function of the shared library. This function computes the gain and loss associated with each possible aggregates.
6. Ocelotl calls the bissection function of the library, which returns a list of parameter p values, such as each partition $\boldsymbol{P}$ of this list is unique. This list is used to build an interactive graph of the gain and loss associated with $\boldsymbol{P}$ as a function of$p$. This graph is shown on the bottom right of the Figure 7.
7. The user can click on this graph to select a value of p.
8. Ocelotl calls the algorithm function that provides the partition for a value of p in input. Since the user can manually provide a parameter p that is not contained in the list, this partition is recomputed and not retrieved from the step 6.
9. Ocelotl generates a visualization from the algorithm output and the microscopic data.
10. The user can change the partition by choosing another p value. Back to the step 8.
11. The user can change other settings (change time bounds to zoom, modify resources involved, change time slice number to generate microscopic model, etc.). Back to the step 1.

4.1.3 Interaction

We provide different types of interaction in Ocelotl. First, as presented in Section 4.1.2 (step 7), the user can click on the gain and loss curves in order to select a parameter p. The objective is to choose the most interesting values of p thanks to the curve shapes. Second, we provide a zoom mechanism in the visualization. By selecting an area with the mouse, the user changes the time bounds and can relaunch an analysis on this area. The pipelined process is started from the beginning: indeed, the zoom is not graphical, but performs a reiteration of the full process on the selected subpart of the trace. The new visualization obtained this way is thus more accurate. Last, the user can switch to a space-time Gantt chart-like diagram. By clicking on a button, Framesoc opens a new Gantt chart window on the selected time bounds. Here, the objective is to provide more details on an interesting time part detected thanks to Ocelotl. An example of Gantt chart is given by Figure 8, showing a perturbation detected by Ocelotl.

4.2 Performance

Regarding the Ocelotl performances, three points have to be evaluated separately:

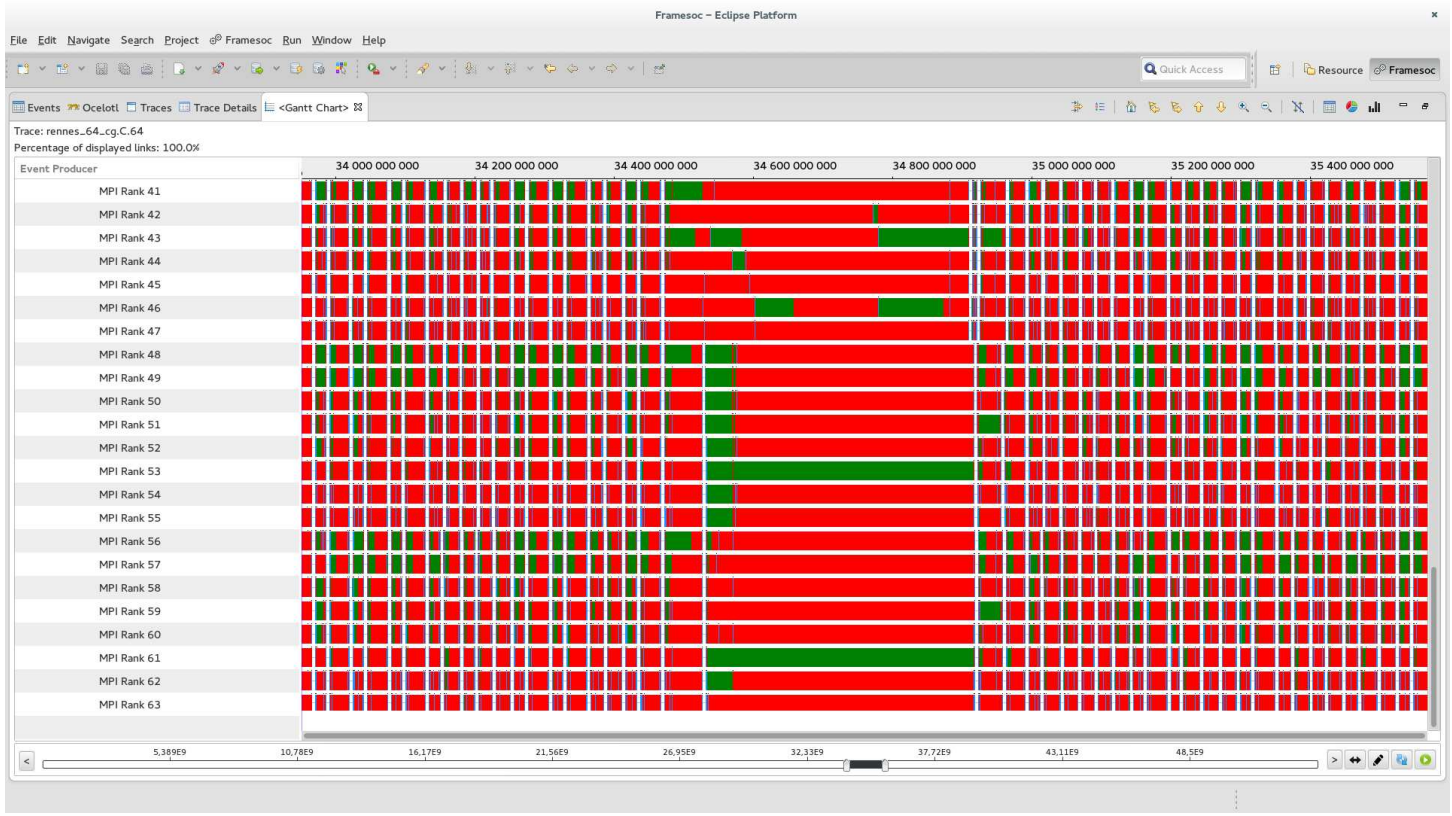


Fig. 8 Example of the Gantt chart provided by Framesoc, focused on one of the temporal perturbations detected by Ocelotl in the use case NASPB CG class C, 64 processes (Figure 3)

- The microscopic model building: it corresponds to the steps 1 to 4 of the pipelined process. It consists in reading the trace and getting its events through a query to the database, and filling the microscopic model using aggregation. Figure 9 shows that for a given time slice number, the time of microscopic model building from the database is linear to the event number. As trace reading is a costly process, we have implemented a way to save microscopic models associated with a trace, which reduces dramatically the microscopic model building time for further analysis sessions (reading the trace is mandatory at least once). We describe this feature in Section 4.2.1.
- The algorithm computation: it depends mainly on the microscopic model dimensions. In the temporal algorithm case, the costliest part is the quality computation, whose complexity is $\mathcal{O}(|T|^2 \times |S| \times |U|)$. The time slice number is the main parameter the user tunes. Increasing its value provides more precision, but leads to longer computation times. In the spatiotemporal algorithm case, quality computation and bissection time have the same order of magnitude. Its total complexity (sum of the different operations) is $\mathcal{O}(|T|^3 \times |\mathcal{H}(S)|)$. In practical, $|T|$ should be left less than 30 time slices to keep reasonable computation delays. In both temporal and spatiotemporal cases, part computation times are marginal compared to quality and bissection times. Figure 10 and 11 show the algorithms execution time as a function of the microsopic model size. Bissections are computed with a maximum difference of 0.001 between two values of p.
- The rendering: the time needed to build the visualization from the algorithm output. It depends mainly on the graphical object number, and is thus related to the time slice number in the case of the temporal overview, and to the space and time dimension elements in the case of the spatiotemporal overview. In the worst cases, this time reaches about 10 seconds. Responsiveness depends partially on the SWT and Draw2D Eclipse libraries performances.

Table 1 shows Ocelotl execution times for real MPI traces of different sizes. All our tests have been performed using a desktop computer, composed of 4 cores Intel Xeon CPU E3-1225 v3 at 3.20 GHz, 32 GB DDR3, 256 GB SSD. All the traces and their caches were stored on the SSD, which had a SATA-3 interface and a reading capacity of 530 MB/s.

4.2.1 Trace Reading Time Reduction

Reuse Generated Microscopic Models

In order to speed up the access to a trace data, we implemented a cache. This cache works by saving in a file the microscopic model the first time it is built. Since all the computations performed by Ocelotl are based on this microscopic model, the cache avoids the necessity to go through the costly process of reading the whole trace in the database in later analysis sessions. The microscopic model is based on an aggregation of the events which depends on the used metric (i.e. event occurrence, state duration, variable mean) and, for a given metric, this microscopic model does

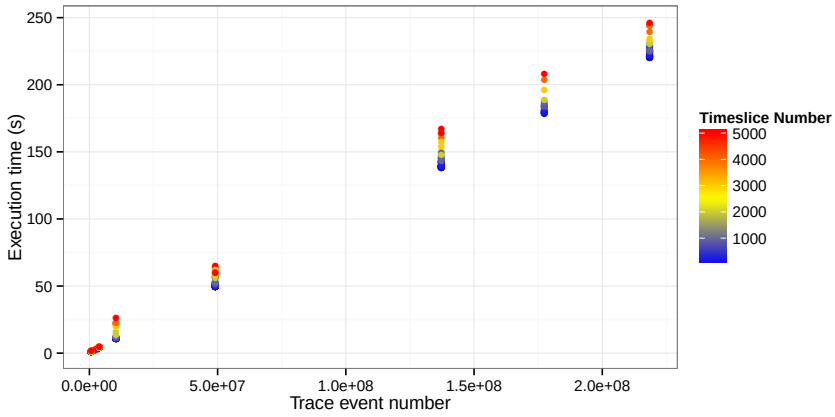


Fig. 9 Microscopic building computation time, including the trace reading from the database, as a function of event number contained in the trace. Time slice number is symbolized by a color gradient. For a given time slice number, execution time is linear to the event number. Most part of execution time is due to database trace reading.

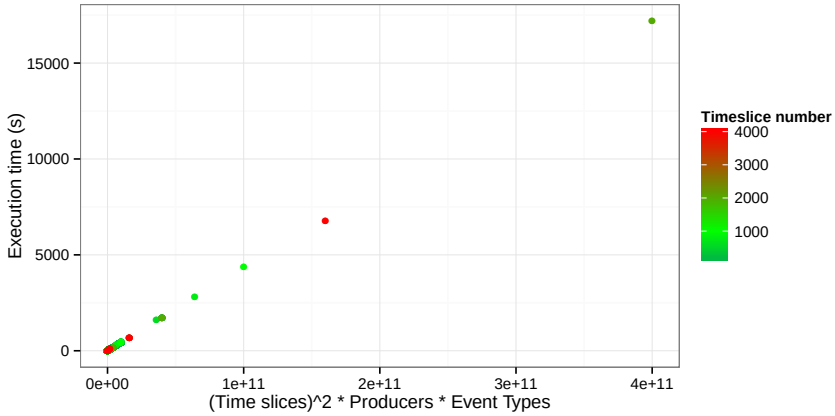


Fig. 10 Total computation time of the temporal aggregation algorithm. We represent execution time as a function of $|T|^2 \times |S| \times |U|$ to highlight the algorithm complexity. As an indication, the complexity of the temporal analysis of NASPB LU class C, 700 processes (Figure 4) is $|T|^2 \times |S| \times |U| = 100^2 \times 700 \times 10 \sim 10^8$

not change from one analysis to another, thus allowing its reuse. Thanks to the aggregation, the size (memory footprint) of a microscopic model is much smaller than the raw trace stored in database and consequently is much faster to load. To further reduce the footprint and improve performance, we only save non-null values of the microscopic model. As an illustration of the obtained footprint reduction, a

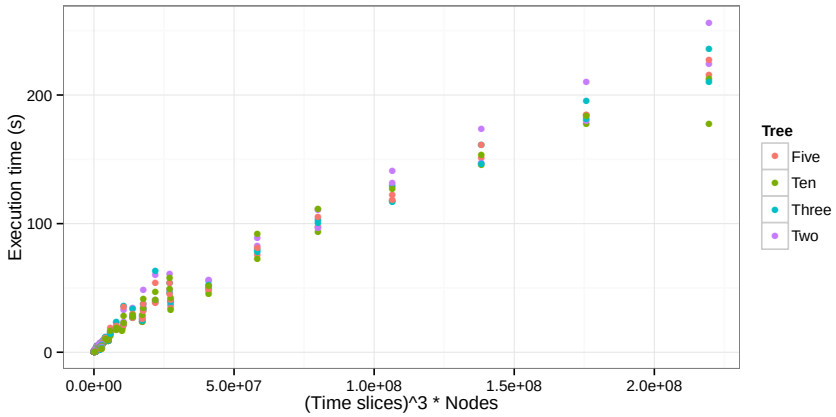


Fig. 11 Total computation time of the spatiotemporal aggregation algorithm. We represent execution time as a function of $|T|^3 \times |\mathscr{H}(S)|$ to highlight the algorithm complexity. Tree corresponds to the hierarchy structure (number of child per node). As an indication, the complexity of the spatial analysis of NASPB LU class C, 700 processes (Figure 6) is $|T|^2 \times |\mathscr{H}(S)| = 30^3 \times 801 \sim 2 \times 10^7$

Table 1 Ocelotl execution times for various trace sizes (temporal analysis, $|T| = 100$)

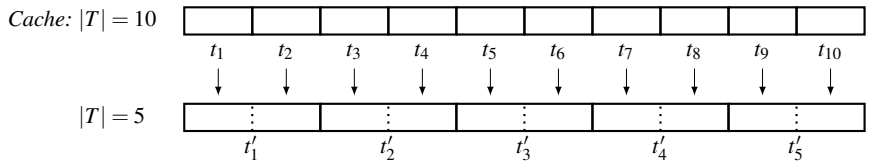
	Case A	**Case B**	**Case C**	**Case D**
Application	CG, class C	CG, class C	LU, class C	LU, class B
Processes	64	512	700	900
Site	Rennes	Grenoble	Nancy	Rennes
Event Number	3,838,144	49,149,440	**218,457,456**	177,376,729
Trace size (in DB)	136.9 MB	1.8 GB	**8.3 GB**	6.7 GB
Ocelotl computation times				
Trace Reading + Microscopic Model	4 s	50 s	**224 s**	181 s
Qualities Computation	<1 s	1 s	**2 s**	3 s
Bissection	<1 s	<1 s	**<1 s**	<1 s
Total Preprocess	4 s	51 s	**226 s**	184 s
Parts Computation + Rendering	<1 s	<1 s	**<1 s**	<1 s

trace with 1 billion events takes 35,6 GB in database but takes only 34.8 MB as a cache file of 1000 time slices.

As explained in Section 3.3, the size of a microscopic model depends on three dimensions: the number of time slices, the number of event producers and the number of event types (i.e. the temporal, the spatial and event type dimensions respectively in Section 3.3). However the number of time slices is the only dimension that has an influence on the values stored in the microscopic model. Indeed, modifications in the number of event producers or in the number of event types can be dealt with in the later computations by simply ignoring the stored values corresponding to the ignored event producers or event types. As a consequence, the saved microscopic model is built with a high number of time slices[1], in order to have a microscopic

[1] This number is configurable through the GUI in Ocelotl.

a) Example of microscopic model rebuilding using perfect cache



b) Example of microscopic model rebuilding with dirty time slices (in red)

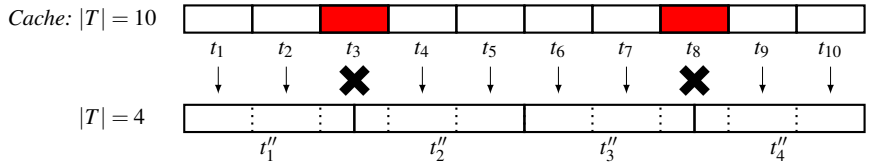


Fig. 12 Illustration of the rebuilding of a microscopic model using a 10 time slice cached microscopic model: a) Cache time slice number fits with generated microscopic model b) Cache time slice number does not fit, which is responsible of dirty time slices

model with a high granularity. This allows us to use the cache for all the analyses performed with a number of time slices lower than the one used in the cached microscopic model. For example, suppose that a cache was generated with 1000 time slices ($|T| = 1000$), and the analyst wants to perform an analysis with 100 time slices ($|T| = 100$). Then a time slice of the microscopic model will be build by using 10 time slices of the cached one. The operation performed for rebuilding is a simple addition of the values of the cached time slices. More formally, for a time slice t_{built} using the cached time slices from t_{cache_i} to t_{cache_j}, the value of t_{built} is:

$$t_{built} = \sum_{n=i}^{j} t_{cache_n} \tag{6}$$

This is illustrated by the Figure 12 a): in this example, each new time slice is composed of two time slices of the cache, e.g. $t'_1 = t_1 + t_2$. Figure 13 shows the improvement when using a presaved microscopic model instead of reading the trace events.

Generate an Approximated Microscopic Model

Rebuilding the microscopic model works well when the number of time slices in the cached microscopic model is divisible by the number of time slices used in the newly built microscopic model. However, when this is not the case, it can lead to problematic cases where some of the cached time slices are split over two time slices. This phenomenon is illustrated in the Figure 12 b). In this figure, we can

see that the time slices t_3 and t_8 each overlaps over two time slices, and thus cannot be used "as is" to build the new microscopic model. We call these cache time slices split across two time slices, "dirty" time slices. This problem can also occur when performing a zoom on a region of the trace. To deal with those problems, we implement an approximate strategy. The strategy is a tradeoff between speed and accuracy. It consists in computing for each dirty time slice, the proportions of the time slice which overlaps the currently built time slices, which gives a value between 0 and 1. We then multiply the values of the dirty time slice, with the corresponding proportion. More formally, with the notations introduced in Section 3.3, for a given dirty time slice t_{dirty} and the two built time slices t_{built1} and t_{built2} it overlaps, we compute $prop_1$ and $prop_2$, with:

$$prop_1 = \frac{\tau_{end}(t_{built1}) - \tau_{start}(t_{dirty})}{d(t_{dirty})} \tag{7}$$

where $\tau_{start}(t)$ and $\tau_{end}(t)$ correspond respectively to the start and the end timestamp of the timeslice t. We can then compute $prop_2$ as $prop_2 = 1 - prop_1$. Then for each value in the time slice t_{built1} in the built microscopic model, we perform the following operation:

$$v_{built}(s,t_{built1},u) = v_{built}(s,t_{built1},u) + v_{cache}(s,t_{dirty},u) \times prop_1 \tag{8}$$

and a similar operation for t_{built2} with $prop_2$.

This strategy provides good results in term of performance (Figure 13), and only has a small overhead compared to using a cache with no dirty time slices. However, the performed approximations can lead to discrepancies between the microscopic model obtained and the original one. In our experiments, we have observed that, when existing, the large majority of the differences are minor, as illustrated by Fig 14, and do not hinder the analysis results. However in some rare cases, especially when the ratio of dirty time slice number to time slice number is high, more problematic cases can appear, leading to different aggregations. This is why this strategy should be used with caution and used to quickly explore the trace. In the case where the user wants to keep maximum fidelity in the microscopic model, an option can be selected to get the events from the database trace when the cache does not fit well, instead of using the approximate strategy.

5 Conclusion

We have presented Ocelotl, a visualization tool that provides multidimensional overviews for trace analysis. These overviews are built upon an aggregation methodology that involves information quantities, and gather trace parts that behave homogeneously. This method enables to reduce the visualization complexity while keeping useful information and highlighting application phases and anomalies. We present various examples showing the relevancy of our temporal and spatiotemporal

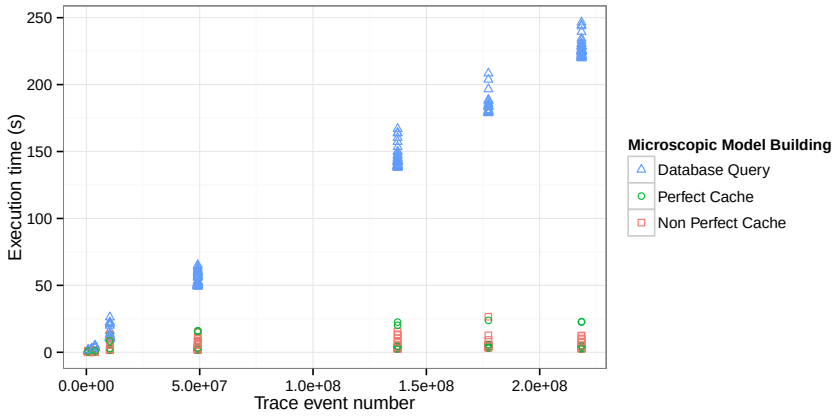


Fig. 13 Microscopic building computation time as a function of event number contained in the trace. Time slice number varies from 10 to 200. We distinguish several cases: events are retrieved by a *query* to the database to build the microscopic model, or microscopic model is generated from a previously saved data cache. We distinguish the case where both microscopic model time slices fit perfectly (*perfect*), and the case where an approximation has to be done (*non perfect*).

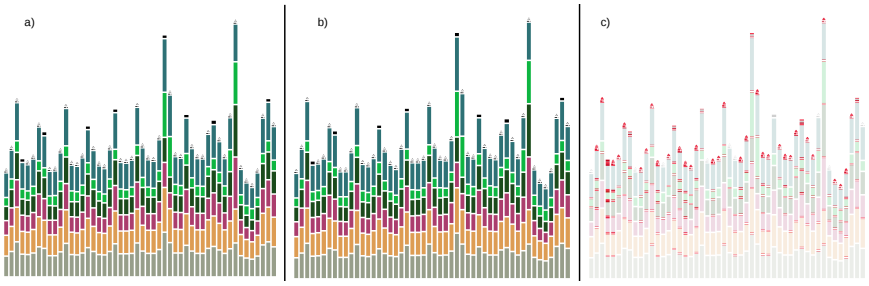


Fig. 14 Example of the differences obtained with the approximate strategy: a) is the view obtained when using no cache, b) is the view obtained when using the approximate strategy and c) highlights in red the differences between the two views.

overviews. We succeed in finding anomalies in 8 GB traces containing up to 218 millions events.

The Ocelotl tool has features that help the analysis. Its modularity allows to add new components adapted to particular needs. Interaction mechanisms like zoom and switches to other representations help to get easily more details on subparts of the trace. Regarding the performance and the interactivity, we profit from the Framesoc framework that stores traces in databases. We succeed in reading 1 billion event traces of 35 GB in 18 minutes. As reading the trace remains globally a costly process, we save intermediate trace abstractions, the *microscopic models*, for further analysis sessions. This enables to get a result in up to 30 seconds, whatever the original trace size and event number.

Our future work concerns many domains. Aggregation methodology could be extended on other dimensions to represent trace aspects like communications, which we are unable to manage for now. Some efforts have to be done on the visualization, in particular by providing dynamically more information about the aggregates. Regarding the performance, the aggregation algorithms could be parallelized, which could help us to be more accurate, in particular on the temporal dimension of spatio-temporal overview.

References

1. Linux Tools Project/LTTng2/User Guide - Eclipsepedia. http://wiki.eclipse.org/index.php/Linux_Tools_Project/LTTng2/User_Guide
2. Dosimont, D., Lamarche-Perrin, R., Schnorr, L.M., Huard, G., Vincent, J.M.: A spatio-temporal data aggregation technique for performance analysis of large-scale execution traces. In: Proceedings of the 2014 IEEE International Conference on Cluster Computing (CLUSTER'14). Madrid, Spain (2014)
3. Dosimont, D., Pagano, G., Huard, G., Marangozova-Martin, V., Vincent, J.M.: Efficient analysis methodology for huge application traces. In: Proceedings of the 2014 International Conference on High Performance Computing & Simulation (HPCS). Bologna, Italia (2014)
4. Dosimont, D., Schnorr, L.M., Huard, G., Vincent, J.M.: A trace macroscopic description based on time aggregation. Tech. Rep. HAL RR-8524 (2014)
5. Chassin de Kergommeaux, J.: Pajé, an Interactive Visualization Tool for Tuning Multi-Threaded Parallel Applications. Parallel Computing **26**(10), 1253–1274 (2000)
6. Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Müller, M.S., Nagel, W.E.: The Vampir Performance Analysis Tool-Set. In: Tools for High Performance Computing, pp. 139–155. Springer-Verlag Berlin (2012)
7. Kullback, S., Leibler, R.: On Information and Sufficiency. The Annals of Mathematical Statistics **22**(1), 79–86 (1951)
8. Lamarche-Perrin, R., Demazeau, Y., Vincent, J.M.: How to Build the Best Macroscopic Description of your Multi-agent System? In: Y. Demazeau, T. Ishida (eds.) Proceedings of the 11th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS'13), *LNCS/LNAI*, vol. 7879, pp. 157–169. Springer-Verlag Berlin, Heidelberg (2013)
9. Lamarche-Perrin, R., Demazeau, Y., Vincent, J.M.: The Best-partitions Problem: How to Build Meaningful Aggregations. In: Proceedings of the 2013 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, pp. 399–404 (2013)
10. Lamarche-Perrin, R., Demazeau, Y., Vincent, J.M.: Building the Best Macroscopic Representations of Complex Multi-Agent Systems. In: Transactions on Computational Collective Intelligence, LNCS. Springer-Verlag Berlin, Heidelberg (2014)
11. Lamarche-Perrin, R., Schnorr, L.M., Vincent, J.M., Demazeau, Y.: Evaluating Trace Aggregation for Performance Visualization of Large Distributed Systems. In: Proceedings of the 2014 IEEE International Symposium on Performance Analysis of Systems and Software, pp. 139–140 (2014)
12. Lusk, E., Chan, A.: Early Experiments with the OpenMP/MPI Hybrid Programming Model. In: OpenMP in a New Era of Parallelism, *LNCS*, vol. 5004, pp. 36–47. Springer-Verlag Berlin (2008)
13. Pagano, G., Dosimont, D., Huard, G., Marangozova-Martin, V., Vincent, J.M.: Trace management and analysis for embedded systems. In: 2013 IEEE 7th International Symposium on Embedded Multicore Socs (MCSoC), pp. 119–122. Tokyo, Japan (2013). DOI 10.1109/MCSoC.2013.28

14. Pillet, V., Labarta, J., Cortes, T., Girona, S.: Paraver: A tool to visualize and analyze parallel code. In: Proceedings of WoTUG: Transputer & Occam Developments, vol. 44, pp. 17–31 (1995)
15. Prada-Rojas, C., Riss, F., Raynaud, X., De Paoli, S., Santana, M.: Observation tools for debugging and performance analysis of embedded linux applications. In: Conference on System Software, SoC and Silicon Debug-S4D (2009)
16. Schnorr, L.M., Legrand, A., Vincent, J.M.: Detection and Analysis of Resource Usage Anomalies in Large Distributed Systems Through Multi-Scale Visualization. Concurrency and Computation **24**(15), 1792–1816 (2012)
17. Shannon, C.E.: A Mathematical Theory of Communication. The Bell System Technical Journal **27**(3), 379–423,623–656 (1948)