

A Trace Macroscopic Description based on Time Aggregation

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

▶ **To cite this version:**

Damien Dosimont, Lucas Mello Schnorr, Guillaume Huard, Jean-Marc Vincent. A Trace Macroscopic Description based on Time Aggregation. [Research Report] RR-8524, INRIA. 2014. hal-00981020v2

HAL Id: hal-00981020

https://inria.hal.science/hal-00981020v2

Submitted on 22 Apr 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



A Trace Macroscopic Description based on Time Aggregation

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

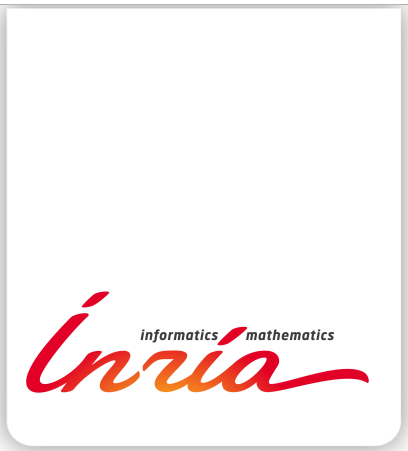
RESEARCH REPORT

N° 8524

April 2014

Project-Teams MOAIS, MESCAL

ISSN 0249-6399 ISRN INRIA/RR--8524--FR+ENG



A Trace Macroscopic Description based on Time Aggregation

Damien Dosimont*†, Lucas Mello Schnorr‡, Guillaume Huard§†, Jean-Marc Vincent§†

Project-Teams MOAIS, MESCAL

Research Report n° 8524 — version 2 — initial version April 2014 — revised version April 2014 — 22 pages

Abstract: Today, because of computing system complexity, it is required to trace application executions to understand their behavior. Visualization techniques provide some help in representing their content, but their scalability is limited both because of human perception and bounded screen resolution. To solve this issue, we propose a visualization based on time aggregation that provides a concise overview of a trace whatever its size. The level of details in this visualization can be configurable by users who can adjust the compromise between concision (gain from aggregation) and information loss. They can then refine their analysis by zooming in an interesting part and choosing a less aggregated overview for this interesting part. This visualization is implemented in our tool, Ocelotl, which enables users to interact with this visualization by changing the selected time interval and its aggregation settings dynamically. The results presented in this paper show that the technique can help users correctly identify anomalies in very large trace files composed of up to forty million events.

Key-words: Trace visualization, trace analysis, trace overview, time aggregation, parallel systems, embedded systems, information theory, scientific computation, multimedia application, debugging, optimization

* INRIA
† first.last@imag.fr
‡ Informatics Institute, UFRGS, Porto Alegre, schnorr@inf.ufrgs.br
§ UJF

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

A Trace Macroscopic Description based on Time Aggregation

Résumé : De nos jours, à cause de la complexité des systèmes actuels, les analystes utilisent le traçage pour comprendre le comportement des programmes. Les techniques de visualisation aident à représenter le contenu de ces traces, mais le passage à l'échelle est limité par la perception humaine des données affichées ainsi que par la résolution des écrans. Dans le but de résoudre ce problème, nous proposons une technique de visualisation faisant appel à une algorithme d'agrégation, fournissant un aperçu du contenu de la trace quelle que soit sa taille. Le niveau de détail peut être ajusté par l'utilisateur, grâce à un compromis entre la réduction de complexité de la représentation (gain dû à l'agrégation) et la perte d'information. L'utilisateur peut ensuite raffiner l'analyse en zoomant sur des parties intéressantes de la trace et en diminuant l'intensité de l'agrégation. Cette technique est implémentée dans notre outil, Ocelotl, qui permet à l'utilisateur d'interagir avec la visualisation en changeant les bornes de temps et les paramètres de l'agrégation de manière dynamique. Les résultats présentés dans ce rapport montrent que notre contribution aide les utilisateurs à identifier des anomalies dans des traces contenant jusqu'à quarante millions d'événements.

Mots-clés : Visualisation de trace, analyse de trace, agrégation temporelle, systèmes parallèles, systèmes embarqués, théorie de l'information, calcul scientifique, application multimédia, débogage, optimisation

Table of contents

1 Introduction **4**

2 Related Work **6**
2.1 Traditional Approaches 6
2.2 Information-based Aggregation 6
2.3 Shortcomings and Discussion 6

3 Macroscopic description of a trace **7**
3.1 Reducing Trace Complexity with Time Aggregation 8
3.1.1 Reducing Complexity of Multi-Agent Systems 8
3.1.2 Characterization of the aggregation process 8
3.1.3 Description of trace contents 9
3.1.4 Provide a microscopic description by timeslicing 10
3.1.5 Extend the best-cut partition algorithm 10
3.2 Applying Shneiderman's Methodology 11
3.2.1 Visual Representation 12
3.2.2 How to Tune Parameter p 12
3.2.3 Completing the Analysis 12

4 Framework: Ocelotl tool **13**

5 Application Analysis **13**
5.1 GStreamer Application 13
5.1.1 Settings 13
5.1.2 Part analysis 14
5.1.3 Traces comparison using state proportions 15
5.1.4 More details using a zoom in and a Gantt chart 15
5.2 NAS Parallel Benchmark 16
5.2.1 Setting 16
5.2.2 Comparison with the Vampir timeline 16
5.2.3 Analysis of two executions 18

6 Conclusion **19**

1 Introduction

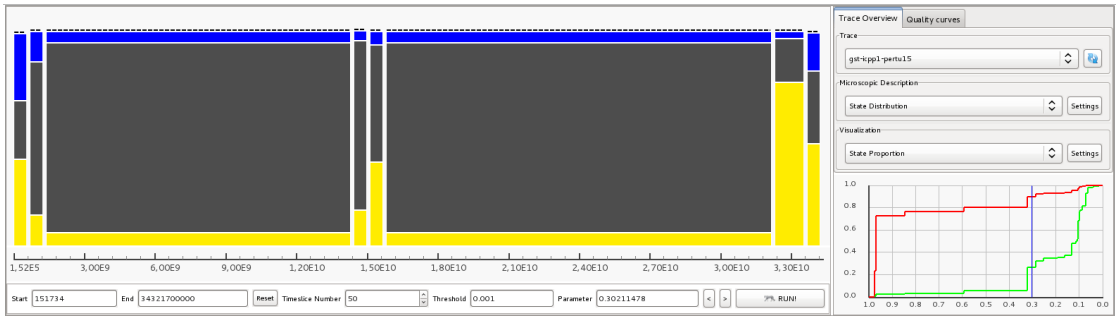


Figure 1: Ocelotl analysis tool overview: the graph shows the detected application phases and highlights a perturbation at $1.5 \cdot 10^{10}$ ns; information and complexity curves help to decide about the aggregation compromise (bottom right)

Nowadays, computer systems are made of increasingly complex hardware and software components. Their hardware architectures are possibly multicore, heterogeneous and distributed. Their software stack is composed of numerous layers including, for example, middleware to abstract the platform. In this context, application debugging and performance optimization become tremendously difficult tasks.

By tracing the application, the analyst gathers low-level information about its execution (function calls, threads or processes execution states, interruptions, CPU load, memory utilization, hardware counters). In debugging scenarii, the objective is to find what is the cause of a perturbation or an undesirable behavior. Whereas in performance optimization, the analyst looks for bottlenecks as well as inefficient algorithms in parts of the implementation.

Because of trace size, which can reach several gigabytes, and the amount of data contained within, up to several millions of events, analysts frequently use compact representation techniques to inspect trace contents. Both statistics and visualizations, temporal or structural, enable them to analyze trace behavior from various angles: from global information to detailed causality relations or application structure.

The problem is that existing solutions handle with difficulty large data sets and their outputs suffer from a lack of scalability. This is particularly present in those related to time and structural aspects. This phenomenon becomes visible with aliasing artifacts and cluttered representations when too much information is displayed on the screen, as depicted by a KPTrace [1] visualization presented in Figure 2. Indeed, there is no data treatment in this tool to reduce trace complexity before rendering it. Schnorr & al. [2] already noticed this issue. Some tools, such as Pajé [3] or LTTng Eclipse Viewer [4] (Figure 3) try to propose a visual aggregation, but without control on the information loss. When this loss is too high, the visualization does not give insights about the application behavior.

The problem of visualization scalability is particularly complicated to address. Even if statistics are often employed, the lack of time and space dimensions makes the link with other representations difficult. According to Shneiderman's methodology [5], analysis using visualization should be composed of successive steps: "overview, zoom and filter, then details on demand". Nevertheless, we consider that the first one is neglected by most analysis tools in the trace visualization domain since they do not provide a sufficiently satisfying overview to apply this methodology.

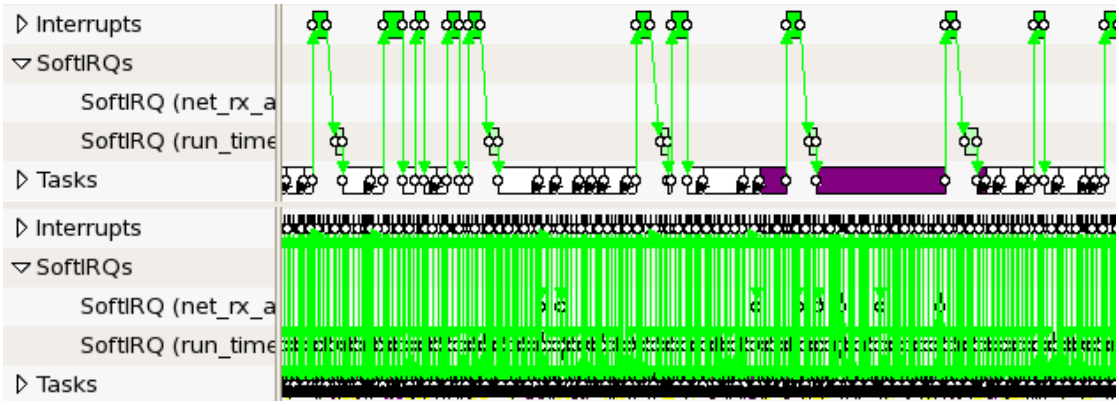


Figure 2: Example of KPTrace Gantt chart [1] showing typical time dimension scalability issues. On the top, the small time scale enables to represent correctly the events. On the bottom, the zoom out provides a larger time interval, but the representation is completely cluttered.



Figure 3: The LTTng Eclipse Viewer [4] uses visual aggregation to avoid aliasing artifacts. However, only states that last long are easy to perceive, and an important part of the information is lost because of the aggregation

To solve this overview problem at scale, we propose an innovative technique based on time aggregation, which represents trace behavior over time with simple colored rectangles. This visualization summarizes an execution trace by highlighting times at which the application behavior changes. This helps determine where the analyst should focus. Moreover, we let the analyst choose the level of details he desires, while providing him with a measure of the information lost by the aggregation. This technique follows Shneiderman methodology by proposing interaction to zoom, filter, and get details. We propose an implementation of this technique in a software of our own design.

This paper is organized as follows. Section 2 evokes related work in trace visualization domains, from traditional visualization techniques to representations that use aggregation mechanisms to improve their scalability. In this state of the art, we detail why we consider that existing solutions are not satisfying. In Section 3, we describe our work in two parts: we first explain how we build our trace aggregation from a theoretical point of view, and then how we represent it and interact with it. Section 4 presents Ocelotl, which is the tool we have designed and that implements our overview technique (see Figure 1 for an example). Section 5 presents several

experiments that demonstrate the relevance of our technique: using scenarios from mainstream multimedia player to parallel applications. Section 6 concludes and describes our future works.

2 Related Work

2.1 Traditional Approaches

Statistics are used as an overview approach in several tools. For instance, PerfExplorer [6] proposes 2D graphs, area charts and scatter plots to compare execution speed-up, function duration distribution or to help in finding correlations between variables of the application. Improving upon the same ideas, 3D techniques found in ParaProf [7] enable to correlate even more variables. Bar charts, found in Paragraph [8] or Paradyn [9], are typically used to show metrics like the number of messages between processes, or their memory utilization. Pie charts, present in Pajé [3], represent, for instance, the proportion of various state processes. The time axis is not represented in such statistics, although this dimension is often needed to understand behavior evolution over time. That is why many analysis tools, such as KPTrace [1], LTTng Eclipse Viewer [4], Vampir [10] or Pajé [3] include Gantt Chart [11] views that show causality relations. States, variables, or communications are associated with hierarchically organized entities and shown over time. Common analysis techniques also include structural representations, such as Vampir's communication matrices [10], function call graphs found in Paragraph [8] and ExTraVis [12] or even resource hierarchy and application topology as in Viva [2]. The Vampir timeline [10] shows an overview over time by using a pixel-guided visualization, where the pixel color represents the state which is the most common one during the time interval related to the pixel position. Paraver [13] [14] also features a Gantt chart with an overview level generated thanks to some rendering algorithm.

2.2 Information-based Aggregation

Another feature of Vampir [10] is its task profile, which clusters most similar processes, according to some distance metric, and shows the mean of their state distribution. However, time dimension and thus causality relation are lost, and this technique is mainly aimed at a profiling usage. Viva [2] implements a treemap view that uses a multidimensional aggregation both in space and time. Hierarchy of resources can be manually collapsed or expanded, while time aggregation is done by choosing a time slice where is applied a configurable operator such as sum or mean. Finally, the states distributions over hierarchical elements are depicted in the treemap by using colored rectangles whose size is related to state proportion. Jumpshot [15] provides a high-level abstraction of the trace over time thanks to SLOG-2 trace format. This technique enables to deal with large traces without loading all the events and to fit all the timeline within the screen size.

2.3 Shortcomings and Discussion

We discuss about the limits of the existing tools and justify our proposal. Traditional tools using statistics neither represent time nor application structure. If they can provide an overview, the lack of these dimensions does not enable the analyst to easily make the link with other representations that involve time or structure. Structural representations also lack the time dimension. Whereas it is sometimes represented with color gradients, like in ExTraVis [12], this information remains difficult to interpret. We insist on time dimension because we consider that an application behavior is characterized by its evolution over time. In particular, problems such

as performance bottlenecks or deadlocks due to some race condition, are strongly related to time. To detect these problems, we have to determine when they might happen. Then only, we can look for a possible cause.

Traditional Gantt Chart views can represent causality relations but they are not suited to a large number of events. A zoom-out often gives a visual clutter where objects placement is done by the graphic rendering pipeline, without any control on the parts of the trace hidden by possible overdrawing. Conversely, a zoom-in causes a context loss and forces the user to scroll and pan in the detailed trace to select right time bounds and hierarchy elements. Solutions that provide visual-based aggregation are not satisfying: their designers use the fact that events with a small size should be aggregated. This relies on the assumption that small sized events are less important than longer ones, which is not always true. Thus, this aggregation reduces visual clutter at the expense of uncontrolled information loss related to these small events. The same problem also appears with filtering techniques, as in Paraver [13], which reduce noise, but also uniformly remove much information. In techniques that use information aggregation like in Jumpshot [15], we are still limited by hierarchy size, which is entirely represented.

To summarize, we consider that the lack of scalability present in existing time representations is responsible for the lack of entry point in the analysis. Usual tools cannot be used in Shneiderman's methodology [5], and it is consequently difficult to understand application behavior from their trace visualization. We claim that this entry point should be clear, simple to understand, reliable and should let the user know how the complexity has been reduced and how much information has been lost. At last, this representation should be meaningful and should guide the analyst to a more precise evaluation of the application behavior.

We solve these issues with an innovative solution that provides a macroscopic description of a trace. It is constituted by a simple time line, which does not represent all the resources associated to events but exploit them to build a simpler visualization, showing which are the different successive macroscopic behaviors that appear during the application execution. These time areas are associated with simple colored rectangles. We fit with Shneiderman's methodology [5] by proposing interaction, zooming, and filtering, but also the tuning of a parameter that controls the compromise between complexity and information loss. Finally, we combine this representation with a Gantt chart, dedicated to the last steps of the analysis.

3 Macroscopic description of a trace

The visualization technique we propose is based on an aggregation algorithm, designed by Lamarche-Perrin & al. [16]. Its originality is that it is configurable, according to the information quantity the analyst accepts to lose during the aggregation process. Visualization can thus bring different levels of details: as the information loss grows, the representation becomes simpler and the number of elements rendered on screen decreases. The objective is thus to find a compromise between representation clarity and information loss by letting the analyst choose the configuration that helps him to understand the application behavior. This decision is dynamic and the level of details is adjusted interactively.

We bring two mains contribution: first, we extend and thus generalize Lamarche-Perrin & al. works, from monodimensional to multidimensional systems, in order to comply with application trace analysis. Secondly, we present an associated visualization technique that enables us to apply the Shneiderman's analysis methodology.

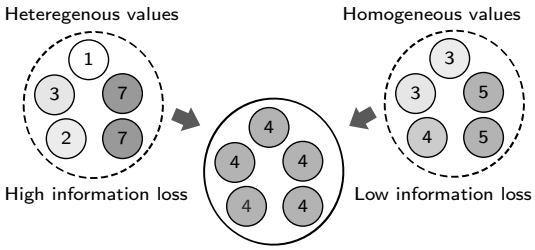


Figure 4: Information loss, when calculating the average, is higher for the left set because its values are more sparse.

3.1 Reducing Trace Complexity with Time Aggregation

3.1.1 Reducing Complexity of Multi-Agent Systems

Lamarche-Perrin & al. works [16] are focused on a way to represent a monodimensional multi-agent system by generating an abstraction to reduce its analysis complexity. The authors propose to partition the system, which defines a microscopic description of this one. Abstraction of the system is then built by aggregating these partitions. To bring semantic to this abstraction, but also reduce the number of combinations the algorithm has to explore, the partition aggregation is constrained, and only aggregates that are meaningful are allowed. For instance, parts might be ordered (in time), or included in a hierarchy (relating to resources), and only contiguous or sibling ones can be aggregated.

For each possible aggregation, we consider two metrics: information loss and complexity reduction. Figure 4 shows how information loss behaves: although the result of an aggregation that averages values is the same for both data sets, the standard deviation is higher for the left set, because its values are more sparse. Consequently, the aggregated value is closer to values in the set on the left than to values in the set on the right. Thus, when performing the aggregation, the information loss is higher for the left set. Figure 5 is an example of complexity reduction: a system comprising five values is first separated into two groups, each one gathering values close to each other. Then we aggregate each of the groups by using the mean. The new system is consequently composed by only two elements instead of five, decreasing its complexity. An additional aggregation makes the system complexity become minimal. Complexity reduction is thus the information we save by representing an aggregate instead of all its elements separately. The idea is to keep the representation that holds a good compromise between information loss and complexity gain.

Lamarche-Perrin & al. [16] propose several algorithms that meet these characteristics, adapted to different types of systems [17]. To aggregate along time dimension, they constrain the system by allowing only consecutive parts aggregation. This way, the associated aggregation algorithm only has a $o(n^2)$ complexity, where n is the part count.

3.1.2 Characterization of the aggregation process

We propose to reuse these works by adapting them to trace analysis. What differs in our case is that our trace contains several dimensions, while the original work was only able to represent a monodimensional system, such as entities ordered in time. So, we define a microscopic model adapted to trace contents, and extend the aggregation algorithm to work with several dimensions. Here, we describe our formalism associated to the aggregation process. We define our trace as an hypermatrix S_H of rank $d \geq 2$. Its size is the product of its size in each dimension

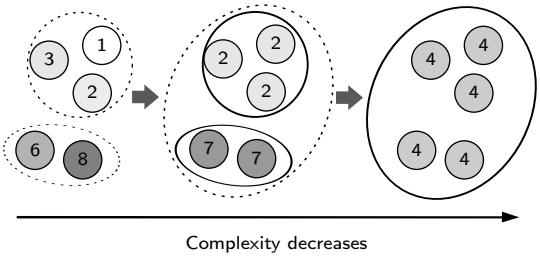


Figure 5: Complexity reduction by aggregating using the average operator.

$n_{dim} = \prod_{i=0}^{d-1} n_{dim_i}$. We divide these dimensions into two categories. The space is the one that gathers the set of hardware or software resources that produce the events in the trace. Space is deduced from the hypermatrix by removing its time dimension, its size is $m = \prod_{i=0}^{d-2} n_{dim_i}$. The time dimension is related to the dates at which events are produced. This is the $(d-1)^{\text{th}}$ dimension of our matrix, its size is $n_t = n_{dim_{d-1}}$. We can simplify the space representation by projecting it to a single dimension and thus reduce the hypermatrix S_H by flattening it as a simple matrix S. Doing this might be necessary for visual rendering (for instance in a Gantt chart) but might imply the loss of contiguity for some elements. The ordered set of time values is noted $\vec{t} = (t_0; t_1; \ldots; t_{n_t-1})$. The set of space values, using an arbitrary order compliant with the hierarchical organization of the resources, is noted $\vec{s} = (s_0; s_1; \ldots; s_{m-1})$. We define an event e as an abstract object produced by a resource at a particular timestamp. Our matrix contains n_e events among its elements such that $n_e \leq n_{dim}$ (they all can be represented by our matrix).. Each event e is characterized by the couple (s_j, t_k) included in the cardinal product of $\vec{s}$ by $\vec{t}$ which is its coordinates in the matrix: s_j is the spatial element, that is the resource which produces the event, while t_k is its timestamp.

Our objective is first to propose a matrix dimension reduction R such that $\mathrm{R}(S_{m,n_t}) = S^{\mu}_{m,n}$. R reduces the size in the time dimension from n_t to n but let the space dimension is invariant. The resulting matrix, which corresponds to what we call a microscopic description, is constituted by n parts $\vec{p_0}, \ldots, \vec{p_{n-1}}$ of dimension $d-2$. To compute the values in these parts, we first cut the trace in fixed time intervals such that $\Delta t_\mu = \frac{t_{n_t-1} - t_0}{n}$, we obtain $\vec{t_\mu} = (0; \Delta t_\mu; \ldots; (n-1) \times \Delta t_\mu)$. Values of $\vec{p_i}$ are computed for each space value from raw trace events by using a timeslicing algorithm TS, what gives $\vec{p_i}[s_j] = \mathrm{TS}(S[s_j], \vec{t_\mu}[i])$.

The final step is to reduce the matrix again by an aggregation $\mathcal{A}$ such that $S^{M}_{m,n_{ag}} = \mathcal{A}(S^{\mu}_{m,n})$ with $n_{ag} \leq n$. This operation is computed by the Best-Cut Partition algorithm. Aggregation $\mathcal{A}$ involves several aggregates: $\vec{A_0}, \ldots, \vec{A_{n_{ag}}}$. An aggregate A_i is defined by $A_i = \sum_{j=a_i}^{a_i + size_i - 1} \vec{p_j}$. Two aggregates $\vec{A_i}$ and $\vec{A_j}$ produced by a same aggregation $\mathcal{A}$ cannot involve a same part $\vec{p}$. Moreover, all partitions are involved in the aggregation, even if they are not aggregated with other partitions. Finally, it is important to notice that only consecutive partitions can be aggregated in an aggregate A_i, to maintain a proper time ordering. All of this means that $a_0 = 0$ (the first aggregate starts at timeslice 0) , $\sum_{j=0}^{n_{ag}} size_j = n$ (all the partitions are part of an aggregate) and $a_{i+1} = a_i + size_i$ (aggregates are made of consecutive partitions, without overlapping).

3.1.3 Description of trace contents

Our trace spatial dimension involves a hierarchy H, constituted by the entities that produce the events. They may be software (functions, thread, processes) or hardware (core, machine, cluster, geographic site) entities. The events they produce are timestamped, and, using Pajé

terminology [18], may be punctual events (synchronization lock, flag, for example), variables (a counter, a semaphore value), states (a process is running, idle, waiting) or links (a communication between two entities). In our work, we only focus on the states, even if the principle remains the same for other types of events. States are represented by a start and an end timestamp, i.e., $\vec{state} = (e_{start}, e_{end})$. We claim that trace behavior is characterized by its state distribution in time: an homogeneous or regular behavior over time matches an uniform distribution in time and, on the contrary, a variation in the behavior translates into a locally non uniform distribution. What makes the analysis particularly difficult when looking at the raw trace, are the multiple event sources that may have different behavior. Nevertheless, if we can understand the behavior of each of them separately, the whole system behavior, as the sum of the behavior of all its entities, is almost impossible to perceive because of its complexity.

3.1.4 Provide a microscopic description by timeslicing

The timeslicing algorithm TS that builds the microscopic description consists in aggregating the values of states associated with a timeslice of size Δt_μ by doing a sum operation. We formalize the calculus:

$$\begin{aligned} S^\mu[s_j][t_k] = \mathrm{TS}(S[s_j], t_k) & \\ = \sum_{i=0}^{n_{state}-1} & (\min(state[s_j][i]_{end}, (t_k+1)\cdot\Delta t_\mu) \\ & - \max(state[s_j][i]_{start}, t_k \cdot \Delta t_\mu)) \end{aligned} \quad (1)$$

Where the *state* matrix is derived from the events in the trace matrix S.

3.1.5 Extend the best-cut partition algorithm

The original time aggregation algorithm is described by Lamarche-Perrin & al. [16]. Because it is only suited to a scalar analysis, we explain our version, adapted to vectors. Our modifications relate to quality measures, described here. First, we have to define all the possible aggregate of our system S^μ. For n partitions $\vec{p}_i$, we get $\sum_{i=1}^{n} i$ possible aggregates:

$$A_{n,n} = \begin{pmatrix} A_{0\ldots(n-1)} & & & \\ A_{0\ldots(n-2)} & A_{1\ldots(n-1)} & & \\ \vdots & \vdots & \cdots & \\ A_{01} & A_{12} & \cdots & \\ A_0 & A_1 & \cdots & A_{n-1} \end{pmatrix} \quad (2)$$

Each aggregate A is characterized by its value, $\vec{v}$, a vector with m dimensions:

$$V_{n,n} = \begin{pmatrix} \sum_{i=0}^{n-1}\vec{v}_i & & & \\ \sum_{i=0}^{n-2}\vec{v}_i & \sum_{i=1}^{n-1}\vec{v}_i & & \\ \vdots & \vdots & \cdots & \\ \vec{v}_0+\vec{v}_1 & \vec{v}_1+\vec{v}_2 & \cdots & \\ \vec{v}_0 & \vec{v}_1 & \cdots & \vec{v}_{n-1} \end{pmatrix} \quad (3)$$

We associate two measures of quality to each of these aggregates:

- the **information quantity** that is lost when we aggregate some elements e in A. This metric is computed from the Kullback-Leibler divergence formula [19]:

$$\text{loss}(A) = \sum_{i=0}^{m} \left(\sum_{e \in A} \vec{v}(e)[i] \times \log_2 \left(\frac{\vec{v}(e)[i]}{\vec{v}(A)[i]} \times |A| \right) \right) \tag{4}$$

  where $\vec{v}(e)$ is the value of the element and $\vec{v}(A)$ the sum of aggregated element values.

- the **complexity reduction** that is the quantity of information we save when we represent an aggregate A instead of the aggregated elements. This formula is calculated from Shannon Entropy [20]:

$$\text{gain}(A) = \sum_{i=0}^{m} (\vec{v}(A)[i] \log_2 \vec{v}(A)[i] - \sum_{e \in A} (\vec{v}(e)[i] \log_2 \vec{v}(e)[i])) \tag{5}$$

The following part of algorithm is now the same as in the original algorithm [16]. With the knowledge of quality measures of aggregates, we are able to determine what is the best aggregation $\mathcal{A}$ according to a compromise between information loss and complexity reduction configurable by the user. For that, we compute a *parametrized Information Criterion*:

$$\text{pIC}(A_i) = p \times \text{gain}(A_i) - (1 - p) \times \text{loss}(A_i) \tag{6}$$

associated with an aggregate A_i. An optimal aggregation $\mathcal{A}_{opt}$ is chosen such that

$$\text{pIC}(\mathcal{A}_k) = \sum_{i \in k} \text{pIC}(A_i) \tag{7}$$

and

$$\text{pIC}(\mathcal{A}_{opt}) = \max \left(\text{pIC}(\mathcal{A}_0), \ldots, \text{pIC}(\mathcal{A}_k), \ldots, \text{pIC}(\mathcal{A}_{n_{\mathcal{A}}}) \right) \tag{8}$$

The analyst influences the aggregation by providing p, which we name aggregation parameter, and which value determines the ratio between loss and gain. Indeed, when $p = 0$, we totally ignore the gain value and only loss is considered. That leads in a representation exactly identical to the microscopic description, i.e., partitions $\mathcal{P}$ are completely disaggregated. On the contrary, when $p = 1$, we ignore the loss and consider only the gain, what provokes a full aggregation of the partition. An intermediate value of p gives a partially aggregated representation, where result depends on part values. Figure 6 describes the aggregation process depending on the p value. We start with a microscopic description (1), and then, make p progressively grows. For each successive configuration, the partition is more aggregated, but, at the same time, more information is lost. This leads to a complexity reduction, from 5 aggregates (2), where information loss is minimal, to only one (6), where information loss is maximal. The objective is thus to find which one is the most meaningful in regard to the behavior we want to describe.

3.2 Applying Shneiderman's Methodology

We explain now how we use the macroscopic description to fit with the Shneiderman methodology [5]. In particular, we describe our aggregated visualization as an overview which serves as an entry point to the analysis. We go further with interaction mechanisms that correspond to the mantra part "zoom and filter", and "get more details on demand".

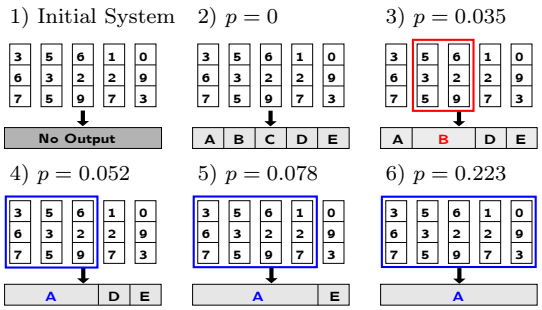


Figure 6: An example of influence of the aggregation parameter p on the aggregation result.

3.2.1 Visual Representation

The macroscopic description we obtain has a reduced complexity, but space dimension is totally preserved. According to space complexity, it can be very difficult to represent the whole set of variables on a visualization without creating visual clutter. That is why we represent only the aggregated intervals, without their contents. The representation is a simple time line, and we associate aggregates to colored rectangles in which width is the duration. Each consecutive rectangle has a different color. The worst case complexity is n rectangles, where n is chosen small enough, less than 1000 elements, to be able to represent all the timeslices on a single computer screen. To summarize the meaning of our representation: we cut the trace in parts, and each part is considered to have a homogeneous behavior. This homogeneity is determined by the combined behavior of all the resources. In other words, all the resources have to behave in a homogeneous way to enable an aggregation of the whole time part.

3.2.2 How to Tune Parameter p

Important elements of our representation are the two characteristic curves of information loss and complexity reduction. By using a search by bisection we identify the set of relevant parameters p_i such that $\mathcal{A}(p_i) \neq \mathcal{A}(p_{i+1})$. In other words, each distinct parameter p_i provides a different aggregation. Our interface also outlines both metric values associated to each aggregation parameter p_i. The quality curves might guide the user in choosing a value for p that gives the most interesting aggregation. When the user clicks on the curves, the parameter p related to their values is automatically provided and the aggregation associated is represented on the screen. Thus, the user focus on curve behavior instead of value of p, which does not provide enough information to determine which aggregations are significant. Typical curve behavior that can help to find relevant aggregations are discontinuities. Complexity variation amplitude is related to the number of additional aggregates between two aggregations, while information variation amplitude translates into the quantity of information the additional aggregates will loose. This last information helps to figure out if new aggregates represent a homogeneous behavior of not (if their associated values are really far from the other aggregates or not).

3.2.3 Completing the Analysis

We propose a second aggregation, related to the spatial dimension, which can help the user to get more information. After the temporal aggregation, for each aggregate, we represent state proportion in the whole entities set by summing values for each state type. We thus obtain a kind of area chart, giving us two metrics: we still distinguish homogeneous parts, but we can also

see the states distribution. We aggregate all states that have too few occurrences to be correctly represented. Still, one can know the content of this aggregate by moving over it with the mouse. This shows a tool tip that lists states names included.

We also propose features corresponding to Shneiderman's approach. Zoom in/out mechanism enables to focus on particular time interval of the trace. User can then recompute aggregations for this trace part until the refinement is precise enough. Filter features helps to remove some events or resources and reduce noise. Finally, the analyst may use a Gantt Chart limited to the time and space bounds where he knows that the potentially problematic behavior is located.

4 Framework: Ocelotl tool

We have implemented our aggregation-based visualization and the different features (zoom, filtering, interaction) in Ocelotl (Figure 1), a tool we have designed. It is written in Java, and generate graphics thanks to Draw2D library. Ocelotl is constituted by a core, which does computations, and a graphical interface, which provides parameter tuning, interaction and visualization. Ocelotl takes advantage of the FrameSoC framework [21], an Eclipse framework that provides traces and tools for traces management. In particular, FrameSoC enables to store traces in a relational-database and proposes an interface to DBMS. Our core is interfaced with FrameSoC, but also with an external native library, LPAggreg, which we have written in C++ for performance reasons, and that implements the time aggregation process (from quality computation to aggregate determination). A specificity of Ocelotl design is the concept of aggregation operator modules. These modules are of two types, a first one in charge of generating the microscopic model from the raw trace by performing queries to FrameSoC and applying the TimeSlicing algorithm on retrieved events. The possibility to add new aggregation modules enable users to extend the TimeSlicing algorithm to other event types (variables, communications, punctual events), to other slice metrics, or to filter some dimensions out before creating the microscopic model. This includes the adaptation to a trace format semantic. A second type of modules is responsible of the representation of additional information, like state proportions computed during spatial aggregation. The Ocelotl graphical interface mainly includes widgets for parameter tuning, timeline and curve visualization.

5 Application Analysis

In this section, we analyze two different use cases with Ocelotl. First, a multimedia player execution based on GStreamer, as a typical example of real-time application, where the problem is characterized by a rupture in the video stream. Secondly, the Lower-Upper Gauss-Seidel solver (LU) of the NAS parallel benchmark, as an example of MPI application doing computation in successive steps.

5.1 GStreamer Application

5.1.1 Settings

This first use case is based on the trace of a GStreamer player execution, in which it plays a MP4 video. We target an application with real time constraints, where a missed deadline provokes a perturbation visible by the user. We trace the execution thanks to the `GST_DEBUG` feature, which prints different timestamped messages on the terminal and, thus, gives information about

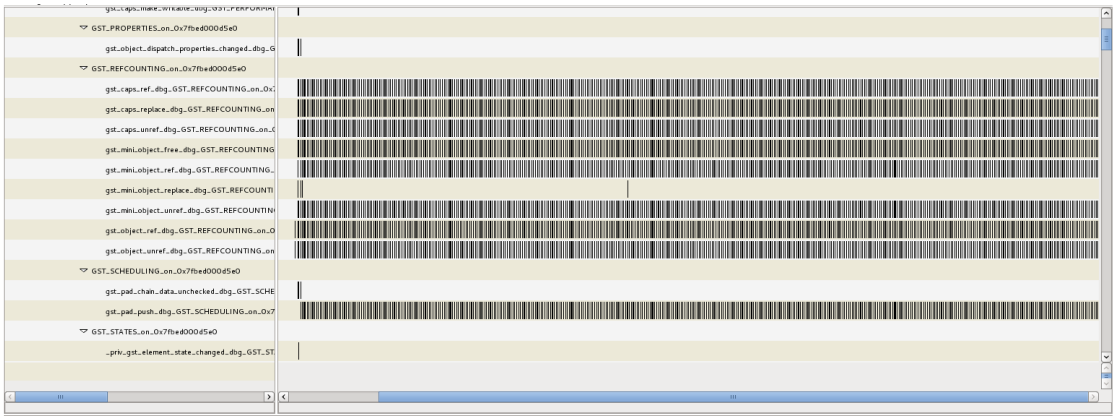


Figure 7: FrameSoC Gantt chart screenshot, showing unreadable representation when we try to analyze GStreamer perturbed trace case B

what is happening. We record several executions of three different videos. A, C, and E[1] are reference cases, i.e., an execution of the player without any perturbation. B and D are obtained by perturbing the execution thanks to the *stress* program, which introduces a CPU load that lasts one second, triggered during the video decoding, and that slightly freezes the video. We use a laptop with a 4 cores Intel Core i7-2760QM clocked at 2.4 GHz and 8 GB of DDRAM for these experiments. The tracing results and analysis times using Ocelotl with 50 timeslices are summarized in the Table 1. We first tried to open the use cases with the Gantt chart provided by FrameSoC (Figure 7). The visualization was unreadable for A and B, and program crashed for C and D because of their large event number.

Table 1: G-Streamer application execution contexts

Use case	A	B	C	D	E
Settings	reference	stress @ 15 s	reference	stress @ 60 s	reference
Video duration	34 s	34 s	314 s	314 s	625 s
Event number	750045	749876	8290576	8302874	14496945
Resources	1581	1581	1528	1593	1592
Trace size	116.1 MB	117 MB	1.3 GB	1.3 GB	2.3 GB
DB query time	12098 ms	12034 ms	117839 ms	122063 ms	213680 ms
Microscopic description	1526 ms	1468 ms	12488 ms	13538 ms	21765 ms
Quality computation	2300 ms	2350 ms	1528 ms	1712 ms	1697 ms
p **list computation**	1 ms	92 ms	1 ms	1 ms	1 ms

5.1.2 Part analysis

We detail how we analyze case D, which is perturbed after 60 s. The objective is to progressively disaggregate the trace and see if a perturbation is detected during the process. To do that, we use both information curves presented in Figure 9. They are made of four stages determined

[1]Only mentioned to give indications about Ocelotl performances. Indeed, video was continuously perturbed by tracing, more intrusive in case of long videos.

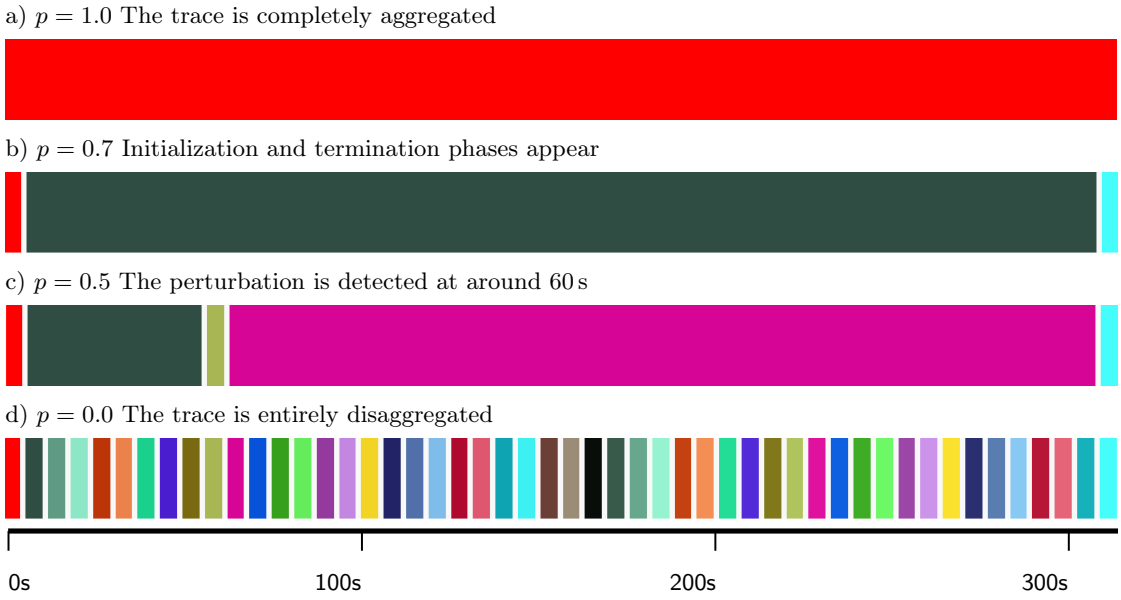


Figure 8: Ocelotl time line screenshots for the GStreamer perturbed case D, highlighting 4 aggregation steps.

by three significant gaps. Each of these stages, corresponding to several aggregations, enables to understand the trace by associating an abstraction level to it. A level will bring a particular type of information to the user. For instance, first abstraction level provides the least complex representation, depicted in Figure 8a, in which the trace is made uniform by the aggregation. The second abstraction level (I-T) matches a few aggregates characterized by a strongly different behavior. Here, we separate initialization and termination phases from the rest of the execution (Figure 8b). Stage P corresponds to an abstraction level where the new aggregates that appear are closer. Here, we see the perturbation provoked by the CPU stress at 60 seconds (Figure 8c), which was not very aggressive. Finally, last stage contains all the information, the trace is completely disaggregated (Figure 8d).

5.1.3 Traces comparison using state proportions

Now, we compare the perturbed case B with the reference case A by using the state proportions representation. Figure 10 shows that trace behaviors are the same, except during the stress when the `DEBUG` level state proportion is larger.

5.1.4 More details using a zoom in and a Gantt chart

We repeat now the same analysis on case B, by zooming in the perturbation (Figure 11). FrameSoC enables to switch automatically to the Gantt chart, by using the bounds chosen with Ocelotl. Thanks to this feature, we notice the corresponding events, showing that we are blocked in the function `gst_clock_id_wait`, because the CPU load has preempted the execution at this time. Finally, this enables to conclude that Ocelotl helped us to analyze these perturbed cases and find which functions were concerned, while it was not possible to do it directly with a Gantt chart.

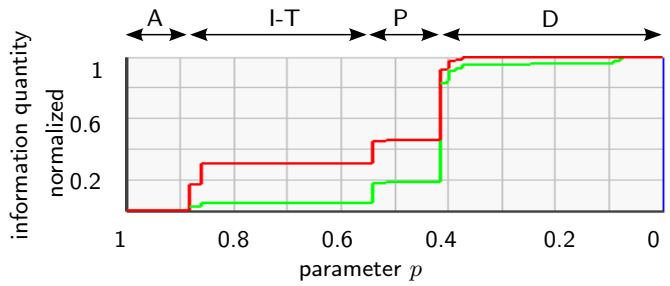


Figure 9: Information (red) and complexity (green) curves related to the perturbed case. We can see 4 main stages.

a) Reference case ($p = 0.44$)



b) Perturbed case ($p = 0.28$)

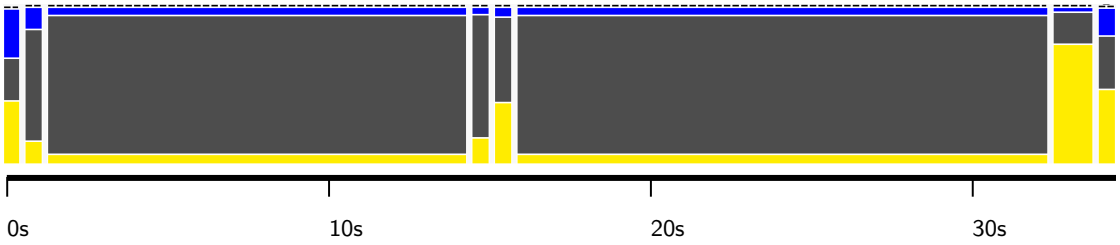


Figure 10: Ocelotl time line screenshot of a GStreamer execution: a) reference A, b) perturbed case B, with states proportions (yellow: `DEBUG`, grey: `LOG`, blue: `TRACE`, black and dashes: aggregation of small states). Behavior are close, excepted during the perturbation

5.2 NAS Parallel Benchmark

5.2.1 Setting

The NAS Parallel Benchmark (NPB) [22] is a well-known benchmark in the high performance computing area. From the set of available applications, we have selected the class A of the Lower-Upper Gauss-Seidel factorization. As our execution platform, we have used two different sites of Grid'5000[2], Lille and Grenoble, with various configurations[3] (please refer to Table 2 for details).

5.2.2 Comparison with the Vampir timeline

The Vampir [10] timeline is quite representative of classic overview techniques, guided by pixels to render proportion of states. Figure 12 shows three distinct renderings of the same case A

[2] `https://www.grid5000.fr/mediawiki/index.php/Special:G5KHardware`

[3] We do not present case D analysis, because its behavior is similar to case C, but mention it to give indications about Ocelot performance for a 40 million event trace.

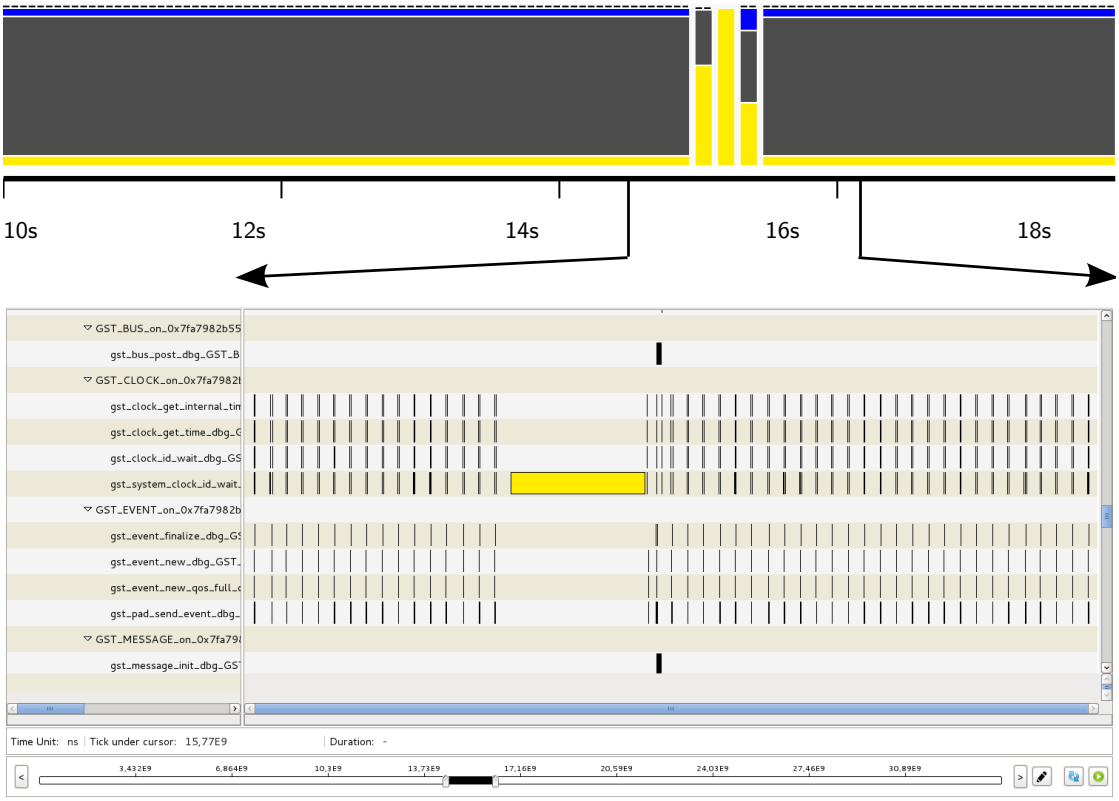


Figure 11: Switch to the Gantt representation after having focused on the perturbation using Ocelotl. By zooming in this area, we discover that the `gst_clock_id_wait` function stays blocked, because GStreamer lost the access to CPU at this time.

Table 2: NAS LU.A application execution contexts

Use case	A	B	C	D
Cores	16	160	152	320
Site	Lille	Lille	Grenoble	Grenoble
Clusters (nodes)	chinqchint(2)	chinqchint(19), chirloute(1)	adonis(6), edel(11), genepi(2)	adonis(1), edel(7), genepi(32)
Event number (MPI only)	1531053	18754155	17669778	38528625
Trace size	427.8 MB	2.7 GB	2.6 GB	5.4 GB
DB query time (MPI only)	111085 ms	959922 ms	908796 ms	1989554 ms
Microscopic description	3028 ms	36153 ms	34762 ms	74997 ms
Quality computation	38 ms	295 ms	321 ms	581 ms
p **list computation**	2 ms	1 ms	1 ms	1 ms

using different resolutions, then resized for the sake of comparison. We notice visible differences between these representations: the visualization depends on the available pixels and only most active states are represented while the others are ignored. When we change the resolution, states

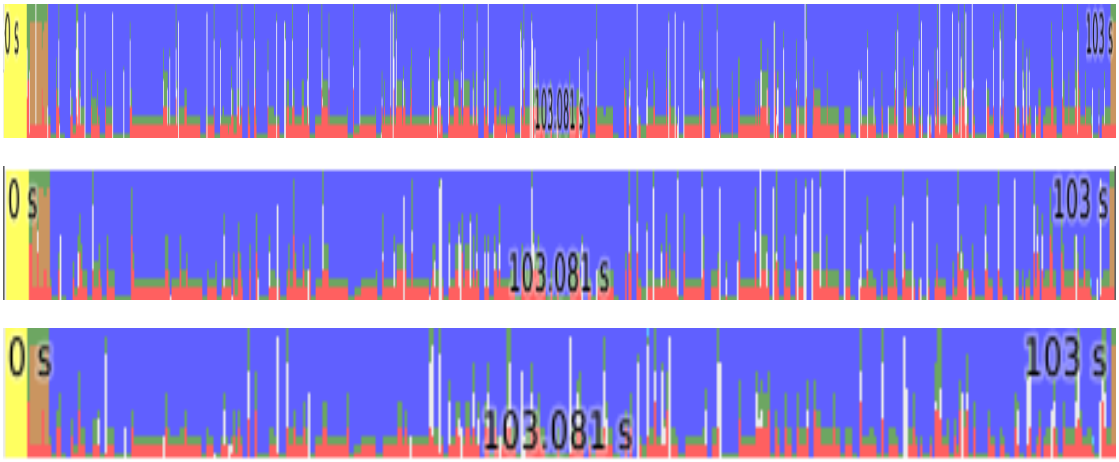


Figure 12: Case A represented with the Vampir Timeline. We have used three different resolutions by resizing the Timeline window (from top to bottom: highest resolution to lowest) and stretched them to the same width to compare their rendering. As the aggregation is guided by pixels, we notice that some information completely disappears when we change the resolution, which corresponds to a coherence loss.

are aggregated differently and some of them completely disappear. The phenomenon is exactly the same with the Vampir Gantt chart, not presented here. Conversely, when using Ocelotl (Figure 13), the user manages by himself the quantity of information that can be represented in the visualization by choosing the time slices number and p. It is more reliable than Vampir timeline, because we represent information loss quantity and because it is insensitive to the number of pixels (which changes when resizing, for instance). Regarding the states too small to be shown, we use dashes to indicate that some information is hidden. But the main interest of this technique is that the partitions match resources behavior, whereas this information is not considered in Vampir aggregation.

Table 3: Grid 5000 cluster hardware information

Cluster	**chirloute**	**ChinqChint**	**adonis**	**edel**	**genepi**
Model	Dell PowerEdge C6100	SGI Altix Xe 310	Bullx R422-E2	Bullx Blade B500	Bull R422-E1
CPU	Intel Xeon E5620	Intel Xeon E5440 QC	Intel Xeon E5520		Intel Xeon E5420 QC
Memory	8 GB		24 GB		8GB
Network	GB Ethernet	Myri-10G	InfiniBand 40G		InfiniBand 20G
Storage	300 GB / SATA	250 GB / SATA	250 GB / SATA	64 GB / SATA	160 GB / SATA

5.2.3 Analysis of two executions

The figure 14 shows several screenshots of case B (Lille) and C (Grenoble) with 160 and 152 processes respectively. As the number of cores used in both cases is close, we want to compare the execution sequences. We choose three levels of abstraction to emphasize the various information Ocelotl can bring to the user. First (on the left), we represent states proportions on all the trace.

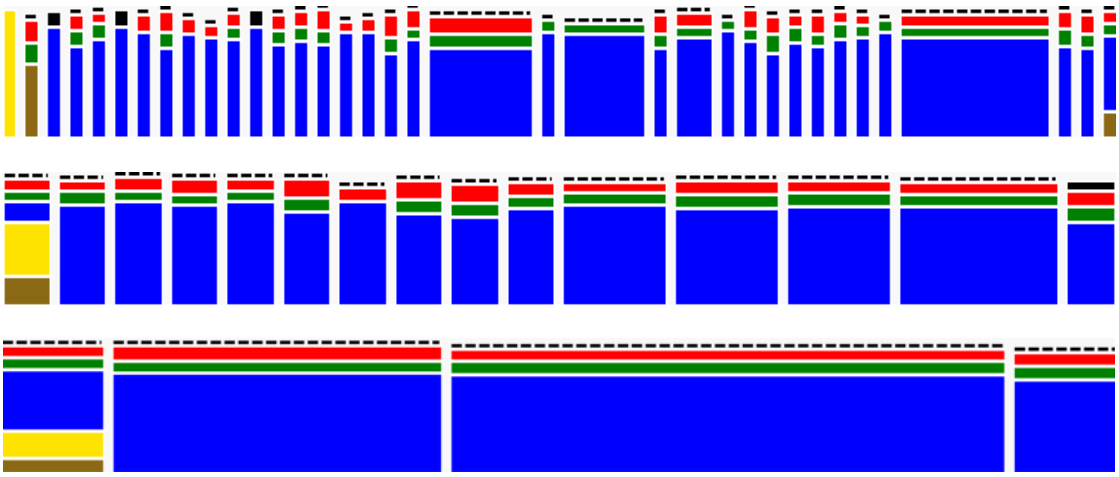


Figure 13: Case A represented with Ocelotl. Ocelotl maintains coherence and does not loose important information when the level of details decrease (from top to bottom). Black dashes indicates the presence of states that are too small to be shown

We discover that the execution time was shorter for Grenoble (5.5 s compared to 10.7 for Lille). Moreover the `MPI_init`:`MPI_*` ratio is higher in case C, but we are not yet able to compare the time spent in the `MPI_init` state using our tool. The second level (in the middle) separates computation phases from the initialization. Surprisingly, the initialization lasts two times more for Grenoble despite its better performance. However, we notice that the computation lasts only 2 s, compared to 9 s for Lille. In particular, the `MPI_Recv` state proportion is higher for Lille. In the third level of abstraction (on the right) details about computation phases appear. We see that the sequence is noticeably different in the two cases: `MPI_Recv` keeps a high proportion during the whole computation for Lille, while it decreases in Grenoble after 4 s. We guess that this matches a bottleneck in communications since `MPI_Wait` stays always active for Lille, which is not the case in Grenoble. Regarding hardware configuration of both platforms, we guess that the better performance of computation phases in Grenoble are related to InfiniBand cards used for communications, while Lille has only Miry-10G inside ChinqChint cluster and Gigabit Ethernet between its both clusters. Network may also influence the initialization behavior.

6 Conclusion

We have designed a time overview technique that provides an entry point to trace analysis. It is based on an aggregation technique [16], which provides the analyst with a macroscopic description of the trace and the ability to control and measure the information quantity lost by this aggregation. Our first contribution is the generalization of the original work [16] by extending the algorithm to multi-resources systems and redefining the microscopic model provided in input. Our second contribution is the visualization, and the associated interaction, implemented in the Oceltol tool.

Although our technique is able to represent efficiently an overview of the time dimension, we must admit that it has several limitations. First, time aggregation algorithm complexity depends on resource and time slice number. We estimate empirically to 10000 the limit of resource number

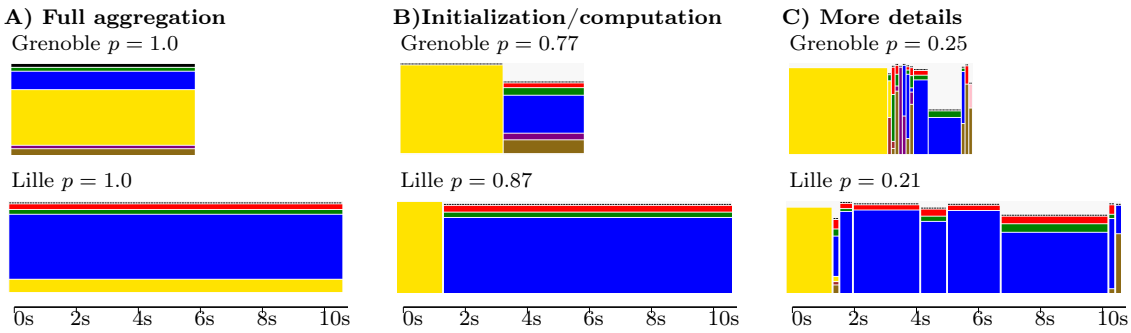


Figure 14: Time-aggregated states of the NPB's LU.A benchmark executions on Grenoble (case C) and on Lille (case B) sites. We show three abstraction levels, which enable to make both qualitative and quantitative comparisons about A) proportion of `MPI_init` state (yellow), B) timestamp of computation start, C) behavior of computation phases (`MPI_Recv` state in blue, `MPI_Wait` in red and `MPI_Send` in green)

we can work with, by using 1000 time slices. However, we could aggregate some resources prior to the analysis to reduce this number. Secondly, our technique seems more useful to analyze applications that have a periodic behavior or behavior organized as a sequence of phases. If the behavior is too chaotic, it might be difficult to find a good compromise between full and no aggregation.

Nevertheless, we present two use cases that highlight the efficiency of this technique to synthesize trace behavior from realistic applications executions. We easily observe a perturbation in a real-time multimedia application that is not visible by using a Gantt Chart. The zoom and interaction features in Ocelotl give the keys to quickly find the functions impacted by the perturbation. Our visualization is also adapted to parallel applications, as the NAS benchmark use-case shows: we can compare several executions and detect the influence of the platform on the application behavior. Regarding the performance, after a preprocess that lasts less than 5 minutes, our implementation is able to provide instantaneous aggregation level changes when analyzing a GStreamer trace with almost 15 million events (2.5 GB). We also succeed in analyzing a NAS Benchmark trace with 40 million event (5 GB), but the preprocess lasts 34 minutes.

We also think that the information contained into the space dimensions should not be limited to state proportions: our future work is focused on the extension of the time-aggregation technique to provide combined space and time aggregations, using the same principle of information and complexity measurement to get the best representation.

Acknowledgments

This work is partially supported by the french Interministerial Unique Fund (FUI) and the project SoC-Trace. We thank Generoso Pagano for his help with FrameSoC and Matthieu Dreher for his help with Grid'5000.

References

[1] KPTrace Trace Format. `http://www.stlinux.com/`.

[2] Lucas Mello Schnorr, Arnaud Legrand, and Jean-Marc Vincent. Detection and analysis of resource usage anomalies in large distributed systems through multi-scale visualization. *Concurrency and Computation: Practice and Experience*, 24(15):1792–1816, 2012.

[3] J Chassin de Kergommeaux. Pajé, an Interactive Visualization Tool for Tuning Multi-Threaded Parallel Applications. *Parallel Computing*, 26(10):1253–1274, August 2000.

[4] Linux Tools Project/LTTng2/User Guide - Eclipsepedia. http://wiki.eclipse.org/index.php/Linux_Tools_Project/LTTng2/User_Guide.

[5] Ben Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, pages 336–343, Washington, DC, USA, 1996.

[6] K. A. Huck and A. D. Malony. Perfexplorer: A Performance Data Mining Framework for Large-Scale Parallel Computing. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 41, 2005.

[7] Robert Bell, Allen D. Malony, and Sameer Shende. ParaProf: A Portable, Extensible, and Scalable Tool for Parallel Performance Profile Analysis. In *Euro-Par 2003 Parallel Processing*, volume 2790, pages 17–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[8] M. T Heath and J. A Etheridge. Visualizing the performance of parallel programs. *IEEE Software*, 8(5):29–39, September 1991.

[9] B. P Miller, M. D Callaghan, J. M Cargille, J. K Hollingsworth, R. B Irvin, K. L Karavanic, K. Kunchithapadam, and T. Newhall. The paradyn parallel performance measurement tool. *Computer*, 28(11):37–46, November 1995.

[10] Andreas Knüpfer, Holger Brunst, Jens Doleschal, Matthias Jurenz, Matthias Lieber, Holger Mickler, Matthias S. Müller, and Wolfgang E. Nagel. The Vampir Performance Analysis Tool-Set. In *Tools for High Performance Computing*, pages 139–155. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[11] J.M. Wilson. Gantt Charts: A Centenary Appreciation. *European Journal of Operational Research*, 149(2):430–437, 2003.

[12] B. Cornelissen, A. Zaidman, D. Holten, L. Moonen, A. Van Deursen, and J.J. van Wijk. Execution trace analysis through massive sequence and circular bundle views. *Journal of Systems and Software*, 81(12):2252–2268, 2008.

[13] Vincent Pillet, Jesús Labarta, Toni Cortes, and Sergi Girona. Paraver: A tool to visualize and analyze parallel code. In *Proceedings of WoTUG-18: Transputer and occam Developments*, volume 44, pages 17–31, 1995.

[14] Jesús Labarta, Judit Gimenez, E Martinez, P González, Harald Servat, Germán Llort, and Xavier Aguilar. Scalability of visualization and tracing tools. *Proceedings Parallel Computing (ParCo)*, 2005.

[15] Ewing Lusk and Anthony Chan. Early experiments with the OpenMP/MPI hybrid programming model. In Rudolf Eigenmann and Bronis R. de Supinski, editors, *OpenMP in a New Era of Parallelism*, volume 5004 of *Lecture Notes in Computer Science*, pages 36–47. Springer, 2008. IWOMP, 2008.

[16] Robin Lamarche-Perrin, Yves Demazeau, and Jean-Marc Vincent. The Best-partitions Problem: How to Build Meaningful Aggregations. In *Proceedings of the 2013 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'13)*, Atlanta, USA, 2013.

[17] Timothée Giraud, Claude Grasland, and Robin Lamarche-Perrin. Identification of International Media Events by Spatial and Temporal Aggregation of RSS Flows of Newspapers. Application to the Case of the Syrian Civil War between May 2011 and December 2012. In *Proceedings of the 18th European Colloquium on Theoretical and Quantitative Geography*, 2013.

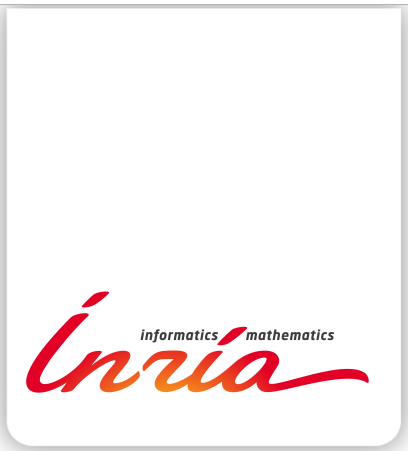
[18] Benhur de Oliveira Stein and Jacques Chassin de Kergommeaux. Paje Trace File Format, 2010.

[19] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, March 1951.

[20] Claude Elwood Shannon. A Mathematical Theory of Communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[21] Generoso Pagano, Damien Dosimont, Guillaume Huard, Vania Marangozova-Martin, and Jean-Marc Vincent. Trace Management and Analysis for Embedded Systems. In *Proceedings of the IEEE7th International Symposium on Embedded Multicore SoCs (MCSoC-13)*, Tokyo, Japan, sep 2013.

[22] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS parallel benchmarks. In *Proceedings of the SuperComputing Conference*, pages 158–165. IEEE Computer Society, 1991.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399