A Time-Coherent Model for the Steering of Parallel Simulations

Aurélien Esnard, Michaël Dussere, and Olivier Coulaud

ScAlApplix Project, INRIA Futurs and LaBRI UMR CNRS 5800,
351, cours de la Libération, F-33405 Talence, France

Abstract. The on-line visualization and the computational steering of parallel simulations come up against a serious coherence problem. Indeed, data distributed over parallel processes must be accessed carefully to ensure they are presented to the visualization system in a meaningful way. In this paper, we present a solution to the coherence problem for structured parallel simulations. We introduce a hierarchical task model that allows to better grasp the complexity of simulations, too often considered as "single-loop" applications. Thanks to this representation, we can schedule in parallel the request treatments on the simulation processes and satisfy the temporal coherence.

1 Introduction

Thanks to the constant evolution of computational capacity, numerical simulations are becoming more and more complex; it is not uncommon to couple different models in different distributed codes running on heterogeneous networks of parallel computers (e.g. multi-physics simulations). For years, the scientific computing community has expressed the need for new computational steering tools to better grasp the complex internal structure of large-scale scientific applications.

The computational steering is an effort to make the simulations more interactive. To reach this goal, we focus on three key functionalities: the *control* which allows the end-user to precisely follow the simulation execution and to suspend its execution; the *on-line visualization* of intermediate results which requires to efficiently access the simulation data at run-time; the *interactions* which allow the end-user to modify steering parameters on-the-fly or to remotely fire steering actions at particular points of the simulation. The complex parallel simulations commonly used in scientific computing raise the crucial issue of the temporal coherence for the steering operations. For instance, an action must occur at the very same moment for all the simulation processes. In the same way, the distributed data collection requires that all the pieces come from the same timestep to remain meaningful.

The different approaches proposed to address the temporal coherence can be compared by considering both the time management and the simulation model. Magellan [1] represents a simulation with an unstructured collection of instrumentation points. To notify the time evolution, a timestamp is explicitly specified

M. Danelutto, D. Laforenza, M. Vanneschi (Eds.): Euro-Par 2004, LNCS 3149, pp. 90–97, 2004.
© Springer-Verlag Berlin Heidelberg 2004

by the end-user on each instrumentation point but nothing ensures their coherence over the parallel processes. CUMULVS [2] considers parallel simulations as single-loop programs with a unique instrumentation point. Thanks to this simplification, it implements synchronization mechanisms relying on the internal loop counter and provides time-coherent accesses to distributed data. Only VASE [3] proposes a high-level model of simulations based on a control-flow graph (CFG) and provides a data-flow coupling model. However, this environment is only intended for sequential simulations. These environments are representative of the state-of-the-art: efforts have to be done to achieve an efficient steering environment which combines both the temporal coherence and a precise model of parallel simulations.

In this paper, we focus on the steering of parallel simulations like SPMD or simple MPMD applications (MPI, PVM, etc.). We first present the basis and the architecture of *EPSN*[1], our computational steering environment. Then, we introduce the high-level model used in *EPSN* to specify the steering of parallel simulations. Finally, we describe a strategy of scheduled request treatments to guarantee the temporal coherence.

2 *EPSN* Steering Environment

EPSN is a distributed environment based on a client/server relationship between user interfaces (clients) and simulations (servers). Both simulations and user interfaces must be instrumented with the *EPSN* API. The clients are not tightly coupled with the simulation; actually, they can interact on-the-fly through asynchronous and concurrent requests. These characteristics make *EPSN* environment very flexible and dynamic.

EPSN uses an internal communication infrastructure based on CORBA which provides the interoperability between applications running on different architectures. We previously discussed this option and other aspects of the architecture in [4]. As shown in Fig. 1, *EPSN* environment runs a permanent thread attached to each process of the simulation. These threads are in charge of all the steering treatments and each one contains a CORBA server waiting for the client requests. In addition, an independent application, called *proxy*, provides an unified view of the simulation and acts as a relay for the requests between the clients and all the *EPSN* threads. Once a request for data is received, the thread accesses the simulation process memory and transfers the data directly to the remote client. This transfer occurs concurrently to the simulation execution, in respect of the access areas placed in the simulation source code. As both the simulation and the client can be parallel applications, we use a MxN redistribution library, called *RedSYM*, with a CORBA communication layer.

Once a simulation has been instrumented with the *EPSN* library, it can be deployed as usually. The clients locate it on the network using the CORBA naming service and then connect it through the proxy. To follow the time evolution

[1] *EPSN* project (`http://www.labri.fr/epsn`) is supported by the French ACI-GRID initiative.

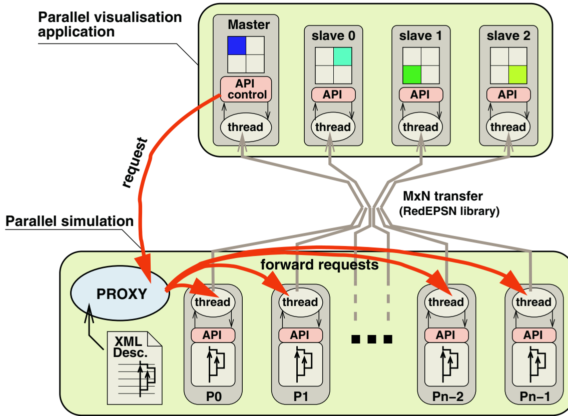

Fig. 1. Architecture of *EPSN* environment.

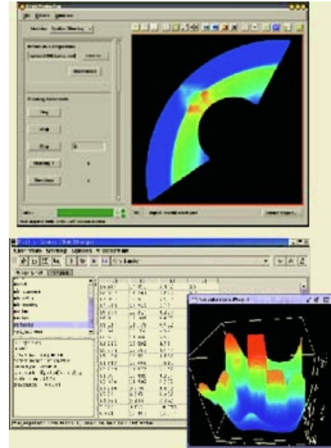

Fig. 2. Visualization clients.

of data, one can use a high-level visualization system based on *AVS/Express* (top of Fig. 2) or *VTK*. In addition, one can monitor a simulation and basically interacts thanks to our *Generic Client* (bottom of Fig. 2).

3 High-Level Model of Steerable Simulations

In order to closely interact with any parallel simulations, we introduce a high-level description model. Traditionally, numerical simulations are described as a collection of steerable items (data, breakpoint, loop, etc.) which are remotely manipulated by a user-interface. In the *EPSN* environment, we organize these items in a hierarchical task model that reflects the essential structure of SPMD and simple MPMD applications. To describe a simulation according to this model, we use an XML file parsed at the initialization of *EPSN* (*epsn_init*). As illustrated in Fig. 3, the XML is divided into three parts, the accessible data for clients (section *data*), the task program structure (section *htg*) and the possible interactions (section *interaction*).

3.1 Data Model

Data currently supported in *EPSN* are *scalars* and *fields* (dense multi-dimensional arrays). Ongoing works will extend the data model for *particles* and *meshes*. All accessible data must be declared, in the XML file (Fig. 3, lines 4–8), with a unique ID, a type (short, long, float, etc.), the number of dimensions for fields, and the distribution mode. Scalars and fields can be either *located* on a single process or *replicated* among all processes. Fields can also be *distributed* over the processes with any generic rectilinear decomposition such as the classical block-cyclic distribution. Finally, in order to make data accessible for clients

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE simulation SYSTEM "epsn.dtd">
3  <simulation id="sample">
4    <data>
5      <scalar id="initial-temp" type="double" distribution="replicated"/>
6      <scalar id="mean-temp" type="double" distribution="located"/>
7      <field id="heat" type="double" dimension="2" distribution="distributed"/>
8    </data>
9    <htg access="protected">
10     <task id="init" access="protected"/>
11     <loop id="main-loop" access="accessible">
12       <task id="init-loop">
13         <data-context ref="mean-temp" context="out"/>
14       </task>
15       <loop id="sub-loop">
16         <data-context ref="heat" context="out"/>
17       </loop>
18       <breakpoint id="bp0" state="up"/>
19     </loop>
20     <task id="ending"/>
21   </htg>
22   <interaction>
23     <writing data="initial-temp">
24       <point node="init" position="end"/>
25     </writing>
26     <action id="restart">
27       <point node="main-loop" position="iterate"/>
28     </action>
29   </interaction>
30 </simulation>
```

Fig. 3. Example of a simulation description in XML.

(*publish*), the user must complete its description with the *EPSN* API and must precise the data storage and the field distribution. Moreover, correlated data may be logically associated in a *group*, and thus *EPSN* will ensure the temporal coherence among the group members.

3.2 Hierarchical Task Model

The basis of our hierarchical task model consists in representing SPMD simulations through a hierarchy of tasks (Fig. 3, lines 9–21). In this paper, we extend the definition of *hierarchical task graphs* [5] or HTGs for the purpose of computational steering. Basically, an HTG is a directed acyclic graph, which captures the control flow between blocks of code and hierarchical nodes (condition, loop).

Here, we consider three types of nodes: (1) *basic tasks* encapsulate logical blocks (e.g. program routine); (2) *loop tasks* represent the iterative structures (for, while); (3) *conditional tasks* represent the conditional structures (if-then-else, switch-case). All these nodes are hierarchical in nature, as they can contain a serie of other nodes. We also introduce *breakpoint* nodes, that are used to explicitly control the execution flow and have no sub-nodes. Initially, a main task encapsulates all the hierarchy which is delimited in the source code by the *epsn_init* and the *epsn_exit* calls. In practice, the program structure can be described at different levels of details and the user can consider only the relevant tasks for the steering. The instrumentation is achieved by manually annotating the begin and the end of each task with the *EPSN* API (*epsn_task_begin*, *epsn_task_end*, etc.). Figure 4 illustrates, on a simple "two-loop" program, that the instrumentation exactly reflects the HTG structure and its XML description. As a result, the instrumentation process can be partially automated.

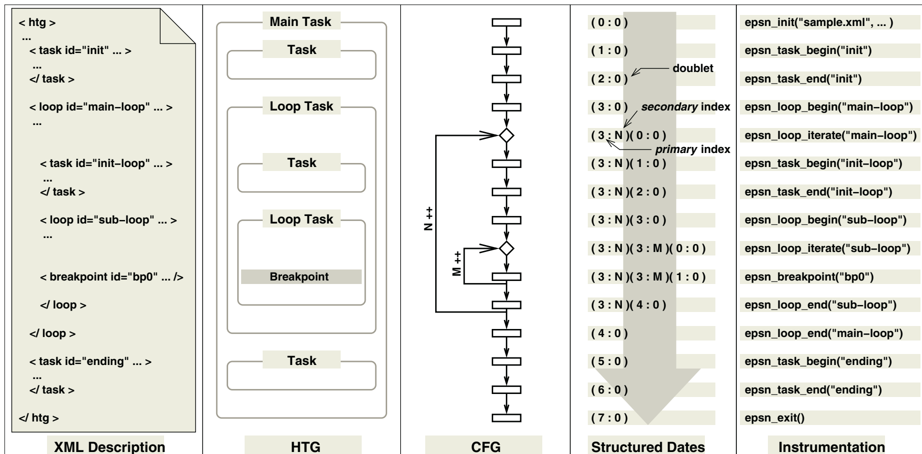

Fig. 4. Comparison of the different representation modes.

In addition, a task may have input and output data, explicitly specified in the XML through a *context*. The input data are purely formal information, whereas the output data are those modified by a task. *EPSN* prevents any client access to output data during all the task execution and automatically releases them at the end. Furthermore, a task may contain a specific access context, specifying the protected and accessible data. Such access areas are fundamental in *EPSN* as they exhibit where and when a data can be safely and asynchronously retrieved.

In simple MPMD cases that we consider, the processes are divided into very few groups (*process-partition*) involved in different programs (e.g. ocean atmosphere coupled models) and running on the same cluster. We assume that the HTGs of the different groups have their upper hierarchy level in common, typically, they share a synchronized main loop. Consequently, we can describe the whole simulation as a unified HTG whose lower levels are distributed in sub-HTGs among the groups (*distributed-sections*). The user must also complete the XML description to precise over which process group the data are replicated or distributed (*location* attribute). The unified HTG provides the end-user with a single representation of the simulation, which can be steered either at an upper level addressing all processes or at a lower level addressing just a group.

3.3 Steering Interactions

As shown in Fig. 3 (lines 22–29), *EPSN* proposes two kinds of steering interactions: the modification of simulation data (*writing*) and the callback of user-defined functions (*action*). As we carefully consider such interactions, we only enable them to occur on instrumentation points, identified in the XML description. Only the actions require to be instrumented in the source code by associating a callback function with the action ID.

4 Temporal Coherence

The efficiency of a steering system for parallel simulations depends on its capability to execute time-coherent operations over all the processes. Different strategies can be used to maintain the temporal coherence, but they all rely on the ability to compare the current positions in the execution of parallel processes.

4.1 Structured Dates

We consider the date of a process as its position in the execution flow. The succession of the dates related to the instrumentation points define a discrete time system. As we deal with structured applications typically involving a hierarchy of loops, a simple timestamp, incremented at each instrumentation point, would be ambiguous. As a result, we define a structured date as a collection of doublets (e.g. $(F_0 : A_0)(F_1 : A_1)...(F_n : A_n)$). As shown in Fig. 4, each doublet corresponds to a level of the hierarchy and it is composed by a *primary* and an *secondary* index. When the execution flow traverses a level of the HTG, the corresponding *primary* index is simply incremented at each instrumentation point. When it enters a sub-level of the HTG (loop, switch), the *secondary* index marks the current iteration or the chosen branch; then, a new doublet is added to follow the progression in the sub-level.

The use of structured dates presents several advantages for a steering environment. It first allows a client to clearly identify the simulation progress. Structured dates are also relevant for elaborating precise requests. Finally, they constitute a robust discrete time system: thanks to the inherent hierarchy of the structured dates, a desynchronization in a sub-level will not propagate into the upper ones.

4.2 Coherence Strategies

The toolkits and libraries developed for the steering of parallel and distributed applications present different strategies to maintain the temporal coherence. In the strategy of *post-collection ordering*, the environment first collect on a central server the informations of the simulation, then these informations are interpreted together. This strategy is mainly used for the ordering of monitoring events like in Falcon [6] or PathFinder [7]. Considering that it requires a systematic centralized data collection, it is poorly adapted to the on-line visualization of large amount of data. In the *centralized request management* strategy, at each step of the simulation, the processes refer to a central server which indicates them if they have to do a treatment. This strategy is used by client-server-client architectures like DAQV [8], but it imposes a very strong synchronization and the simulation processes advance at the server's pace. For *EPSN* environment, we have chosen a third strategy which consists in scheduling the treatments independently on each process of the simulation. The *scheduled treatment* strategy is the one used in CUMULVS [2] to plan data transfers between a simulation and its clients. This strategy does not imply a synchronization with a server and if there is no request,

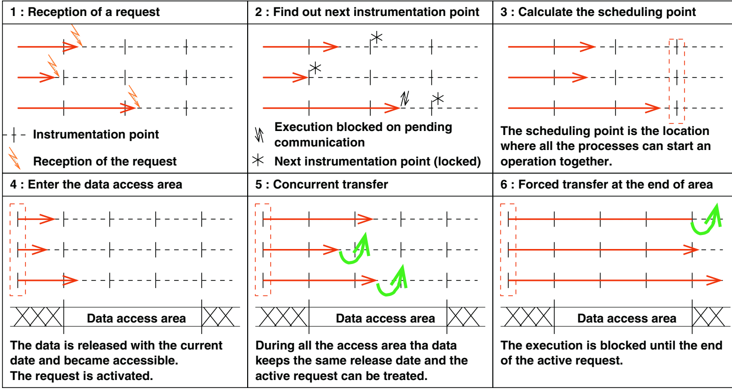

Fig. 5. Scheduled treatment of parallel steering requests.

the simulation is not disturbed. However, to ensure the temporal coherence, the requests must be scheduled at the same date, even asynchronously, in all the processes and these processes must follow the same evolution.

4.3 Scheduled Treatments

We call *scheduling point* the date when the simulation processes schedule the treatment of a request. It is also the earliest instrumentation point that no simulation process has already passed when it get the request. The steps 1 to 3 of Fig. 5 illustrate the choice of the scheduling point. When the instrumentation threads of the processes receive a request, they lock the instrumentation points, they exchange their current dates to determine the scheduling point and then they unlock the instrumentation points. Even if a process is between two instrumentation points, whether it is blocked on pending communications or just computing, its instrumentation thread can communicate its current date to the other threads. As a result, the research of the scheduling point can not lead to a deadlock. When a simulation process goes past the scheduling point, the request is activated independently of the other processes, and it will be treated as soon as its associated constraints are satisfied. For control requests and actions, the constraints are based on the current date. For data access requests, they are based on the data *release*: the date when the data has been lately modified. The release remains unchanged during all the access area after the data modification, and the data can be treated at any time in this area. The constraints based on the release are consistent because the data evolution is linked to the control-flow through the task description and consequently it is the same on all the processes.

The steps 4 to 6 of Fig. 5 illustrate the use of the data release in the case of a simple on-the-fly data access request. When a simulation process enters the access area, the data is released and the request constraint is satisfied. During all the access area, the request can be asynchronously treated because the simulation description ensures that the data will not be modified. At the end of the access

area, the request treatment is forced (flushed) before a process enters in a new modifying area. Figure 5 exhibits all the mechanisms that ensure the temporal coherence. However, in the best cases, the simulation would be synchronized enough for the requests to happen at the same momentof the simulation, and for the scheduling point to be chosen without blocking any process. In the same way, if the access area is long enough, the data transfer would be completely overlapped by the simulation.

5 Conclusion and Future Works

In this paper we address problems raised by the computational steering of parallel simulations. We have introduced a high-level model to better grasp the complexity of these simulations and to closely follow their evolution. In this context, the use of *hierarchical task graphs* has appeared to be very promising. In order to allow the precise positioning in these graphs, we have defined the structured dates which are the basis for our scheduling strategy of the steering treatments. This temporal coherence strategy involves very little synchronization between processes and allows the overlapping of data transfers.

The developments of *EPSN* are now oriented on the support of massively parallel simulations and distributed applications, involving a hierarchy of proxies. Ongoing works also aim at the specification of a generic data model for particles and meshes which would be suitable for the MxN redistribution.

References

1. Vetter, J.: Experiences using computational steering on existing scientific applications. In: Ninth SIAM Conf. Parallel Processing. (1999)
2. Papadopoulos, P., Kohl, J., Semeraro, B.: CUMULVS: Extending a generic steering and visualization middleware for application fault-tolerance. In: Proceedings of the 31st Hawaii International Conference on System Sciences (HICSS-31). (1998)
3. Jablonowski, D., Bruner, J., Bliss, B., , Haber, R.: VASE: The visualization and application steering environment. In: Proceedings of Supercomputing'93. (1993) 560–569
4. Coulaud, O., Dussère, M., Esnard, A.: Toward a computational steering environment based on CORBA. In Joubert, G., Nagel, W., Peters, F., Walter, W., eds.: Parallel Computing: Environments and Tools for Parallel Scientific Computing. Advances in Parallel Computing, Elsevier (2004)
5. Girkar, M., Polychronopoulos, C.: The hierarchical task graph as a universal intermediate representation. International Journal of Parallel Programming **22** (1994) 519 – 551
6. Gu, W., Eisenhauer, G., Schwan, K., J.Vetter: Falcon: On-line monitoring for steering parallel programs. Concurrency: Practice and Experience **10** (1998) 699–736
7. Miller, D., Guo, J., Kraemer, E., Xiong, Y.: On-the-fly calculation and verification of consistent steering transactions. In: ACM/IEEE SC2001 Conference. (2001)
8. Hackstadt, S., Harrop, C., Malony, A.: A framework for interacting with distributed programs and data. In: HPDC. (1998) 206–214