Scaling of Distributed Multi-Simulations on Multi-Core Clusters

Cherifa Dad*, Stephane Vialle*, Mathieu Caujolle†, Jean-Philippe Tavella†, Michel Ianotto*
*CentraleSupelec, University Paris-Saclay, 57070 Metz, France
†EDF Lab Saclay, 91120 Palaiseau, France

***Abstract*—DACCOSIM is a multi-simulation environment for continuous time systems, relying on FMI standard, making easy the design of a multi-simulation graph, and specially developed for multi-core PC clusters, in order to achieve speedup and size up. However, the distribution of the simulation graph remains complex and is still the responsibility of the simulation developer. This paper introduces DACCOSIM parallel and distributed architecture, and our strategies to achieve efficient multi-simulation graph distribution on multi-core clusters. Some performance experiments on two clusters, running up to 81 simulation components (FMU) and using up to 16 multi-core computing nodes, are shown. Performances measured on our faster cluster exhibit a good scalability, but some limitations of current DACCOSIM implementation are discussed.**

I. Introduction

Complex systems are characterized by the interconnection of numerous and heterogeneous cyber components (e.g. controllers) and physical components (e.g. power grids). For EDF (the major French utility company), the smart power grids will extensively rely on new control functions (i) to increase the grid efficiency, reliability, and safety, (ii) to enable better integration of new assets (e.g. distributed generation and alternative energy sources), (iii) to support market dynamics and manage new interactions between established and new energy players. *Cyber-Physical Systems* (CPS) design involves multiple teams working simultaneously on different aspects of the system. Especially, the development of a smarter electrical grid requires to reuse models and tools often based on separate areas of expertise.

This heterogeneity led EDF to investigate coupling standards such as the *Functional Mock-up Interface* (FMI) initiated by Daimler AG within the ITEA2 MODELISAR project and now maintained by the Modelica Association[1]. More precisely, EDF chose its FMI-CS (FMI for co-simulation) part because this operation mode allows to export models as active components called FMUs (Functional Mock-up Units), each FMU being a self-contained archive file including a model and a numerical solver. As an additional benefit, the model IP is readily protected.

For EDF, it is vital to develop agile modelling and simulation in order to design and validate distributed operating functions a long time before performing tests on experimental sites. The *Distributed Architecture for Controlled CO-SIMulation* (DACCOSIM software [1]) developed by EDF and CentraleSupelec is a part of the answer to this problematic. It is aimed at simulating large and complex multi-physics systems on multi-core PC clusters (the standard scalable computing architecture), despite some load unbalance and important communications inherent in our multi-simulations. This paper introduces the parallel and distributed architecture of DACCOSIM, and some scaling experiments on PC clusters (running up to 81 FMUs and using up to 16 multi-core cluster nodes). Finally we list some issues and future works to achieve better speedup and size up.

These researches are carried out by the RISEGrid[2] institute, founded by EDF and CentraleSupelec with the ultimate goal to simulate the smart electric grids of the future.

II. Context and related works

A FMI based multi-simulation is defined by a FMU graph, where all inputs and outputs need to be correctly initialized during a mainly iterative phase called co-initialization. EDF has achieved the co-initialization phase of DACCOSIM reusing its expertise in the Newton-Raphson algorithm (very popular for power flow calculations) in conjunction with recent works done at UC Berkeley on dependency cycles analysis [2] (which are now possible with the latest version 2.0 of the FMI standard). The global dependencies graph automatically deduced from the FMU graph is valuable to solve algebraic loops with a parallel Newton-Raphson algorithm staged at the initialization mode of the multi-simulation. Then, the FMU graph enters a loop of parallel multi-simulation time steps, concurrently running each FMU at each time step, without the need for additional power flow calculations between two consecutive steps. From this point of view, the current version 2.0 of the FMI standard seems sufficient for EDF as its current use cases do not report non convergence examples due to algebraic loops in the step mode.

So, our problematic is to distribute a FMU graph on a multi-core PC cluster. This looks like a task graph distribution problem, which is an important research field. For example, [3] proposes a scheduling algorithm on heterogeneous distributed architectures for static tasks modeled by Directed Acyclic Graph (DAG), and [4] introduces mapping strategies of hierarchically structured multiprocessor tasks on multi-core clusters, which is our target architecture. However, our DACCOSIM FMU graph is not hierarchically structured, and

[1]https://www.fmi-standard.org/downloads

[2]http://www.supelec.fr/342_p_36889/risegrid.html

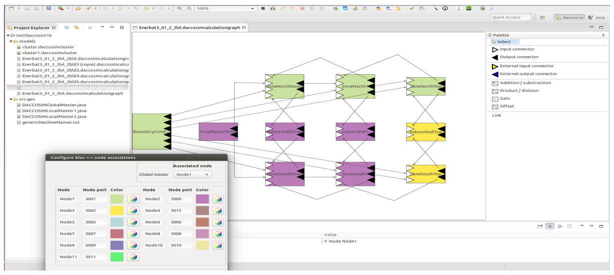

Fig. 1. DACCOSIM GUI: FMU graph definition and configuration

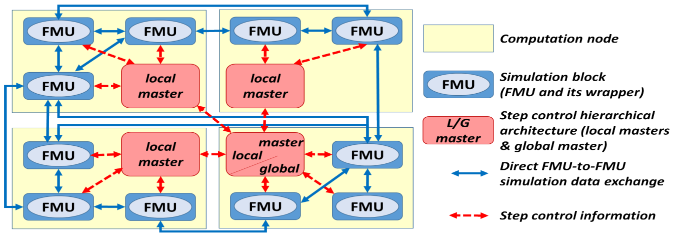
FMU
local master
L/G master
master local global
Computation node
Simulation block (FMU and its wrapper)
Step control hierarchical architecture (local masters & global master)
Direct FMU-to-FMU simulation data exchange
Step control information

Fig. 2. Distributed DACCOSIM architecture, with a hierarchical *master*

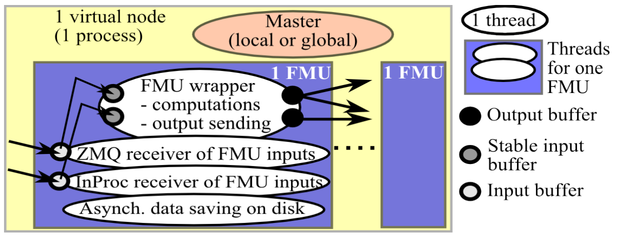
1 virtual node (1 process)
Master (local or global)
1 FMU
FMU wrapper - computations - output sending
ZMQ receiver of FMU inputs
InProc receiver of FMU inputs
Asynch. data saving on disk
1 FMU
1 thread
Threads for one FMU
Output buffer
Stable input buffer
Input buffer

Fig. 3. Multithreaded implementation of a virtual node

is a DAG with very few task dependencies. All FMU tasks can run in parallel when starting a new time step, and when all computations are finished all inter-FMU communications can start and occur in parallel [1]. Then, when all communications are achieved, all FMUs can enter a new time step. There are no dependencies between our FMU task computations, and usual solutions of task graph distribution are not really adapted to our problem. The basic version of our FMU graph working is closer a classical *Bulk Synchronous Parallel* (BSP) model [5], which is well known in parallel computing, but our FMU tasks are heterogeneous and load unbalanced. Moreover, a computationally big FMU can not be split into smaller ones as it would require to design new mathematical models. So, our problem is not a classical *Simple Program Multiple Data* (SPMD) scheme following a BSP model. Our FMU graph has heterogeneity and load unbalance typical of generic task graphs. It is also possible to consider our FMU graph like a kind of time stepped *Multiple Program Multiple Data* application running on a computing cluster and aiming to reach high performances [6]. Identification of an efficient distribution of our FMU graph thus requires to design a specific algorithmic solution.

Some others multi-simulation environments interconnect some continuous time based simulators, like EPOCHS [7] and INSPIRE [8], or even some FMUs, like C2WT [9], through a HLA logical event bus [10]. This bus relies on a Run-Time Infrastructure (RTI), ensuring event routage and right synchronization between simulators. Many RTI implementations are distributed, and can activate concurrently simulators on different computing nodes. However, parallelism of the system depends on the abundance of *safe* events concentrated in a *lookahead* time interval [11], [12], and our time stepped FMU graph contains more potential parallelism.

III. DACCOSIM ENVIRONMENT

A. DACCOSIM software suite

DACCOSIM has been designed to achieve multi-simulations of continuous time systems, discretized with *time steps*, and running solvers using constant or variable time steps (to maintain the requested accuracy with the minimal amount of computations, whatever the dynamic of the system). It consists in two complementary parts: a friendly *Graphic User Interface* (GUI) and a *dedicated computation package*.

The GUI developed in Java facilitates the complex systems studies by designing the multi-simulation graph (Figure 1), i.e. the FMUs involved and the variables exchanged in-between, defining the resources used by the simulation (local machine or cluster), configuring the simulation case (duration, co-initialization method, time step control strategy…) and implementing the graph into DACCOSIM master tasks managing the simulation.

The *dedicated computation package* controls all task execution issues relative to the multi-simulation: co-initialization, local or distributed computation steps, fixed or variable time step control strategies, detection of state events generated inside FMUs, inter-FMU communications, distributed and hierarchical decision process…The Java version of DACCOSIM relies on JavaFMI[3] and is available for both Windows and Linux operating systems, whether 32-bit or 64-bit.

B. Parallel and distributed runtime architecture

The *dedicated computation package* of DACCOSIM includes a *parallel and distributed runtime architecture*, designed to take maximal advantage of any cluster of multi-core nodes. A DACCOSIM simulation executes a series of time steps composed of three stages: the time step computations of the FMUs (independent computations), the communication of the FMU outputs to the connected inputs (many small communications), and the simulation control by the hierarchical control master (information gathering, next operations decision and order broadcasting). All these operations include potential parallelism exploited by DACCOSIM runtime.

As inside one computing step all FMUs can achieve their computation concurrently, DACCOSIM architecture distributes FMUs on cluster nodes, encapsulates FMUs of a same node in different threads and implements a *hierarchical (and*

[3]SIANI, University of Las Palmas, Spain: JavaFMI (2016), https://bitbucket.org/siani/javafmi/wiki/Home

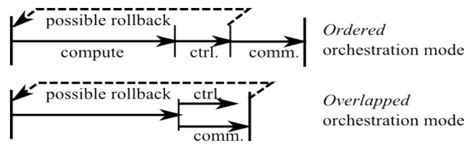

Fig. 4. Two orchestration modes of a multi-simulation step

distributed) control master* to manage all threads and FMU operations. Figure 2 illustrates this architecture. The *hierarchical control master* is composed of a unique global master located on one cluster node, in charge of aggregating the control data coming from the local masters located on the different nodes, and taking decisions based on these information. The global master also assumes the role of the local master on its node. Every local master aggregates control data from the FMUs on its node, and sends synthesized control information to the global master. All master tasks run concurrently.

DACCOSIM uses FMUs in co-simulation mode: they embed their solvers and are implemented as dynamic libraries (enhanced with meta-data XML files). A *FMU wrapper* thread encapsulates each FMU, calls its computation function to achieve a time step progress, and sends each of its outputs to the connected inputs. Three others threads are associated to each FMU, as illustrated on Figure 3: one receipts the output values coming from other computing nodes (currently across ZMQ middleware), one achieves similar receipt but from FMUs located on the same node (not going through the middleware to run faster), and one stores simulation results on disk (asynchronously and per block). Received input values are stored into buffers and distributed to the real FMU inputs before the start of each FMU computation step, so that the inputs remain stable during the computations.

When using variable time steps, the control master gathers and analyses some time step results. It decides if they are valid and if the simulation can enter the next step with the same or a larger time step, or if the simulation has to roll back and continue with a smaller time step (to reach the required accuracy). Two *orchestration modes* are available in DACCOSIM (see Figure 4). The *ordered* mode executes the three phases of each time step in order: (1) FMU computations, (2) communication and control with the hierarchical master, and (3) inter-FMU communications if the master has validated the time step results. The *overlapped* mode overlaps the inter-FMU communications with the control master operations (phases 2 and 3). If the control master requires a rollback, the received input values are forgotten and the FMU states at the beginning of the time step are restored. But when the dynamic of the simulated system is limited (no turbulence), there are few rollbacks and the *overlapped* mode reduces the simulation time. When the dynamic is high, many rollbacks can appear and a lot of inter-FMU communication phases can be achieved needlessly. Currently the orchestration mode has to be set by the user when running the multi-simulation. In the future it could be auto-tuned dynamically depending on the frequency of the rollbacks.

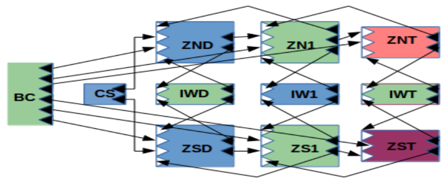

Fig. 5. Best distribution (4 nodes) of *cl5* benchmark with 1 building

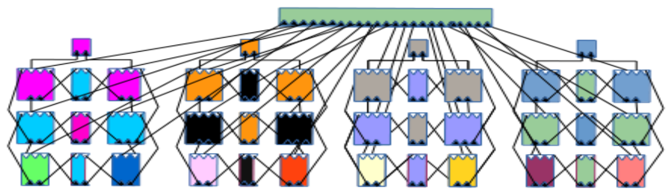

Fig. 6. Scaling on 16 nodes of heat transfer *cl5* benchmark with 4 buildings

A deployment tool (*dacrun*) completes the DACCOSIM software suite, and makes easy the deployment of a multi-simulation on a set of cluster nodes. It is compliant with the *OAR*[4] environment, allowing to allocate nodes and run interactive or batch jobs on a PC cluster, but can be easily adapted to any similar cluster management environment. However, design of a multi-simulation graph distribution on different cluster nodes is the responsibility of the multi-simulation developer. An automatic distribution of the FMUs is under investigation, according to the lessons learnt from our experiments (see next sections).

IV. Parallel and Distributed Experiments

This section introduces our scalable benchmarks and PC clusters, and our methodology to identify the best *distribution patterns* of elementary multi-simulation benchmarks. Then we detail our scaling approach, and our size up experiments on larger benchmarks.

A. Testbed introduction

Our test case is a simplified industrial case provided by EDF R&D, representing heat transfers in a set of n three-floor buildings, with two zones per floor separated by an indoor wall. The entire multi-simulation includes $1 + 10 \times n$ FMUs, interconnected as illustrated in Figures 5 and 6. The unique FMU *BC* is in charge of the thermal boundary conditions (e.g. actual temperatures recorded in a French suburb) which are supposed identical for all the floors in every building. A complementary FMU *CS* models the crawl space temperature for the lower floor of a building, and each building having a specific FMU *CS*. The FMUs *ZNx* (resp. *ZSx*) represent the Northern (resp. Southern) part of every floor, each being designed with Modelica differential equations modeling physical phenomena such as conduction, convection and solar radiation. The FMUs *IWx* detail the behavior of the wall between the floor zones depending on different insulating properties. All these FMUs are equation-based only, and modeled by

[4]https://oar.imag.fr/

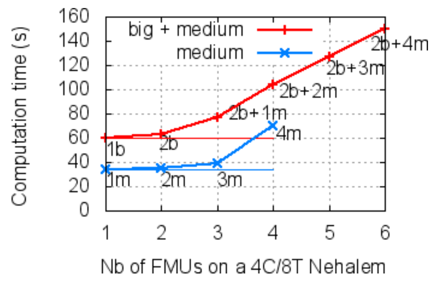

Fig. 7. *cl3* benchmark on one Nehalem node

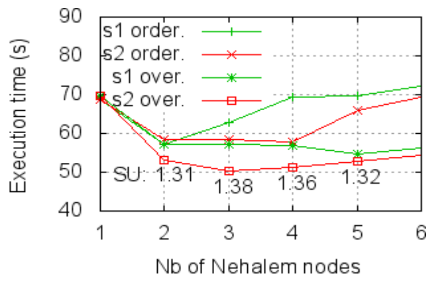

Fig. 8. *cl3* benchmark on our Nehalem cluster

strategy 1: "load"

load:	26.0s; 26.0s; 23.0s
nb intra-comm:	14

strategy 2: "load+comm"

load:	29.8s; 26.0s; 24.0s
nb intra-comm:	16

Fig. 9. Load balancing and number of intra-node communications on 3 nodes

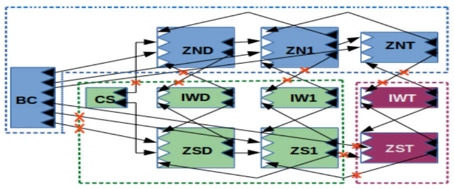

Fig. 10. Best distribution (3 nodes) of *cl3* benchmark with 1 building

encapsulated arrays of records with changeable size to propose 5 levels of complexity (size/weight) for each FMU. No control or temperature regulation is considered in this benchmark, the data exchanged between these FMUs are temperatures and thermal flows. We designed two benchmarks running $1+10\times n$ FMUs. The *cl5* benchmark includes high complexity level FMUs, and exhibits negligible communication times compared to the computation ones. The *cl3* benchmark includes medium complexity level FMUs and has significant communications.

Two PC clusters of CentraleSupelec have been used for our experiments. The first one has a 10 Gigabit/s Ethernet interconnect, and each node includes one Intel Sandy Bridge processor with 6 hyperthreaded physical cores (6 Cores/12 Threads) at 3.2 GHz, and 8 GByte of RAM. The second one has a 1 Gigabit/s Ethernet interconnect, and each node includes one Intel Nehalem processor with 4 hyperthreaded physical cores (4 Cores/8 Threads) at 2.6 GHz, and 6 GByte of RAM.

B. Speedup tracking strategy

Identifying the optimal distribution of a multi-simulation graph on a multi-core PC cluster encounters several locks. First, strong differences appear after measuring the computing load of each FMU. FMUs are heterogenous tasks leading to complex load balancing between computing nodes. Moreover, splitting time consuming FMUs into several smaller ones would require if possible to redesign their mathematical models and would imped to quickly reuse existing simulation components.

Second, installing only one FMU per multi-core node would be a waste, requiring many under-used nodes and increasing the inter-node communication time. But running less than k FMUs in parallel on a k-core node leads to a global computation time greater than the longer FMU one. Figure 7 shows the computation time of a mix of big and medium (half a big) load FMUs on one 4-cores/8-threads Nehalem node. Up to 4 FMUs, the computation time should match the horizontal thin lines, but it appears to follow the thick curves (we also observed this behavior on other multi-core nodes). Most of parallel codes are *memory bound*, excepted when achieving very regular memory accesses and implemented with *tiling* (subdivided and blocked in cache memories). When codes are not tiled, parallel computing time on each multi-core node becomes hard to predict, specially with heterogenous tasks. Computing a load balanced distribution of a FMU set on a multi-core cluster is thus not obvious.

Third, a multi-simulation graph achieves many communications between FMUs, as well as between each FMU and the hierarchical master. Our measures have shown there are numerous and small communications, sensitive to the inter-node network latency. Communication times can be significant compared to computation times (like in our *cl3* benchmark), and inter-node communications remain longer than intra-node communications. So, the distribution strategy of our multi-simulation graphs has to take into account both computations and communications issues.

To achieve our first investigations we adopted an experimental methodology: (1) measuring each FMU computation time (running our real multi-simulation with only one FMU per node to avoid FMUs disturb each others), (2) measuring computation times of different mix of FMUs on one multi-core node, (3) deducing FMU distributions with best load balancing on different numbers of nodes, and (4) tuning these load balanced oriented distribution in order to decrease the number of inter-node communications when communications appear to be important. We applied this methodology to our *cl3* benchmark with 1 building. Figure 8 shows the multi-simulation times we measured on our Nehalem cluster function of the number of used nodes. Two distribution strategies (*s1* focuses on load balancing while *s2* takes into account both computations and communications) and two orchestration modes are considered (see section III). Finally, the best distribution we identified and experimented for our *cl3* benchmark with 1 building uses 3 nodes and is illustrated on Figure 10 (FMUs with the same color will be located on the same node). Figure 9 shows the little tuning (sacrifice) of load balancing successfully achieved by strategy *s2* on 3 nodes to increase the number of intra-node communications (decreasing the number of inter-node ones). For our *cl5* benchmark with 1 building, including huge computations and negligible communications, the best distribution we identified on our Nehalem cluster uses 4 nodes and is illustrated on Figure 5.

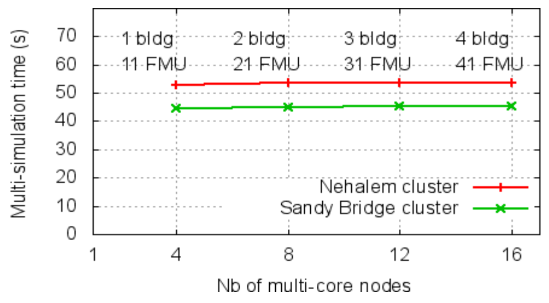

Fig. 11. Size up experiments on *cl5* benchmark

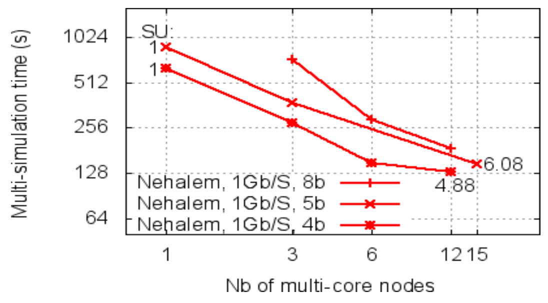

Fig. 13. *cl3* benchmark time with 4, 5 and 8 buildings on 1Gb/s cluster

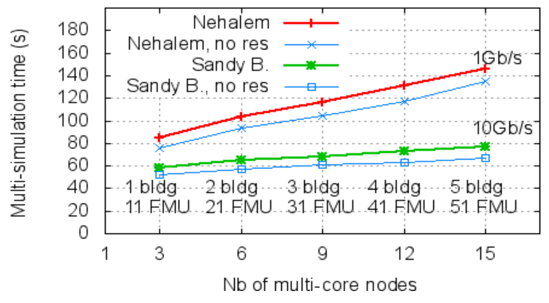

Fig. 12. Size up experiments on *cl3* benchmark

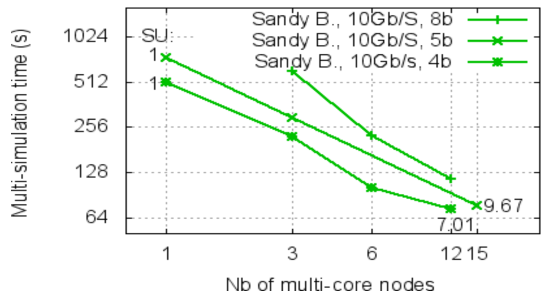

Fig. 14. *cl3* benchmark time with 4, 5 and 8 buildings on 10Gb/s cluster

One of our objectives is now to automatize this investigation methodology, in order to quickly analyze new FMU graphs and define some efficient *distribution patterns*, like the optimal distributions of our *cl3* and *cl5* benchmarks with 1 building. Then, these patterns will be reused in the distribution of larger multi-simulation graphs.

C. Scaling and size up experiments

All experiments introduced in this section use the *overlapped* orchestration mode, and avoid to go through the middleware for internal node communications. This configuration appears the most efficient on our benchmarks and clusters. We measured multi-simulation time with result storage (approximately $850MB$ per run), however we stored these results on the computing nodes local disks, in order to avoid fluctuations due to network traffic on our LAN.

We conducted many experiments to measure *size up* of our multi-simulation benchmarks: we increased both the problem size (number of buildings simulated) and the number of computing ressources (number of cluster nodes), attempting to keep constant the execution time. When processing larger multi-simulations we replicate the best distribution pattern identified for one building (see figure 5 and 6). So we attempt to obtain:

$$T(b\ buildings, b \times n_1\ nodes) = T(1, 1 \times n_1)$$

With n_1: the optimal number of nodes to simulate 1 building

Our previous experiments has shown the optimal distribution for one *cl5* building requires 4 nodes, so we simulated 2, 3 and 4 buildings on 2×4, 3×4 and 4×4 nodes. Figure 11 shows this approach has fully succeeded for the *cl5* benchmark. We obtained approximately the same execution time: $53s$ on our Nehalem cluster with an $Eth\ 1Gb/s$ interconnect and $45s$ on our Sandy Bridge cluster with an $Eth\ 10Gb/s$. These results were expected, as the *cl5* benchmark includes mainly computations and our replication strategy of distribution pattern avoids the building simulations disturb each others.

At the opposite, the *cl3* benchmark includes significant amount of communications compared to the amount of computations. We replicated our 3 nodes best distribution pattern, and we simulated 2, 3, 4 and 5 buildings on 2×3, 3×3, 4×3 and 5×3 nodes. Figure 12 shows we obtained unperfect but interesting performances on our $10Gb/s$ cluster (bottom thick curve). Multi-simulation times increase by $+31\%$: from $58s$ up to $76s$ when processing 1 building on 3 nodes up to 5 buildings on 15 nodes. On our $1Gb/s$ cluster the times increase by $+72\%$ (top thick curve). This experiment shows the impact of the cluster network bandwidth, but we also observed the FMUs exchange mainly small messages. So, a high bandwidth and a low latency seem both necessary to achieve good size up of DACCOSIM simulations.

Thin curves on figure 12 show the execution time when the multi-simulation results are not stored. No result storage means no asynchronous IO thread running in parallel with all others on each node, and no time spent writing on disks. On *cl3* benchmark it leads to smaller execution times but size up benchmarks still exhibit regular increase of execution time. So, we can probably improve our result storage mechanism, but it is not the main limitation of our size up benchmark.

Finally, figures 13 and 14 exhibit multi-simulation times of *cl3* benchmark for 4, 5 and 8 buildings, when increasing the number of computing nodes. We replicated our best distribution pattern (1 building on 3 nodes for *cl3* benchmark), to deploy for example 4 buildings on $12 = 4 \times 3$ nodes. Then we grouped 2 buildings per 3-node block to deploy 4 buildings on 6 nodes, and finally we grouped the 4 buildings on 3 nodes.

So, all our deployments are based on the previously identified optimal deployment of 1 building. This straightforward approach does not track a global optimized deployment, but allows to quickly configure large scale experiments.

Logarithmic scales on both axis lead to expect straight lines with -1 slope for ideal distributions, corresponding to theoretical times $T(n\ nodes) = T(1\ node)/n$. Experimental time curves roughly look like straight lines, but the slope appears stronger on 10Gb/s cluster, showing again the impact of the interconnect performance on our distributed multi-simulations. When simulating 4 or 5 buildings (middle and bottom curves), it was bearable to run and measure execution times on 1 multi-core node and to compute speedup. On 10Gb/s cluster we achieved significant speedup close to 7 on 12 nodes and to 9.7 on 15 nodes (see figure 14), compared to multi-threaded executions on one node. When simulating 8 buildings we did not waste time to run long computations on one node: large problems are not intended for mono-node executions (and perhaps could not fit into one node memory). But we observe the 8-building curves are similar to the 4-building ones with better decrease. So, DACCOSIM can efficiently run larger problems on greater number of nodes and *scale* our multi-simulations (with a fast cluster network).

Executions on 6 nodes (for 4 and 8 buildings) should be 2 times longer than executions on 12 nodes. Computing ressources are 2 times less numerous, but communication achievement is also different. The *BC* FMU is connected to 6 local FMUs instead of 3 and achieves 2 times more intra-node communications, and the network switch is solicited on 6 ports instead of 12. Deep investigations would be required to analyse the origin of these abnormally good performances on 6 nodes, but they point out our straightforward deployment (replicating the 1-building best one) is not optimal. A global deployment of b buildings (i.e. $1+10 \times b$ FMUs) on n multi-core nodes could achieve better performances, but is not currently available.

V. Conclusion and future work

Our FMI based multi-simulation environment (DACCOSIM) is multithreaded and distributed, and allows the user to distribute FMU graphs on clusters of multi-core nodes. According to our distribution strategies, our experiments have achieved significant speedup and satisfying size up on an Ethernet 10Gb/s cluster of hexa-cores, running up to 81 FMUs and using up to 16 nodes to simulate heat transfer inside buildings. Complementary large benchmarks replicating an elementary distribution have shown DACCOSIM can *scale* (efficiently running larger problems on more computing nodes). In order to improve DACCOSIM performances, our next works will consist in: (1) exchanging more compact messages, (2) using high performance MPI communication library instead of 0MQ middleware, (3) exploiting low latency and high bandwidth Infiniband cluster network by relying on MPI.

Automatic distribution of the FMU graph is also under investigation. Numerous experiments have allowed to define a first distribution strategy, that we aim to automatize. DACCOSIM should be able to point out the ideal number of nodes and the associated distribution of the FMU graph, and also to compute the optimal distribution when the number of nodes is constrained by the cluster availability.

Finally, the use case considered in this paper was based on multi-floor buildings with equation-based models only. In the future, EDF R&D intends to run hybrid multi-simulations involving cyber and physical models mixing piecewise constant signals and continuous time signals. As FMI-CS has been designed only for continuous time signals, some extensions to the standard FMI are required and some proposals are on the table (see [13] from UC Berkeley). EDF also leads a French working group on this topic with partners (CentraleSupélec, Dassault Systèmes and CEA List). From the point of view of the optimal distribution of a multi-simulation on clusters, the control FMUs will certainly be very light and ought to be handled by the deployment algorithm without disturbing the distribution of the heavily calculative FMUs.

Acknowledgment

Authors thank Region Lorraine for its constant support.

References

[1] V. Galtier, S. Vialle, C. Dad, J.-Ph. Tavella, J.-Ph. Lam-Yee-Mui, and G. Plessis, "FMI-Based Distributed Multi-Simulation with DACCOSIM," in *Proceedings of the 2015 Spring Simulation Multiconference (TMS/DEVS'15)*, 2015.

[2] D. Broman, C. X. Brooks, L. Greenberg, E. A. Lee, M. Masin, S. Tripakis, and M. Wetter, "Determinate composition of FMUs for co-simulation," in *International Conference on Embedded Software (EMSOFT)*, 2013.

[3] M. I. Daoud and N. Kharma, "A high performance algorithm for static task scheduling in heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 68, no. 4, 2008.

[4] J. Dummler, T. Rauber, and G. Runger, "Mapping algorithms for multiprocessor tasks on multi-core clusters," in *Parallel Processing, 2008. ICPP'08. 37th International Conference on*, Sept 2008.

[5] L. G. Valiant, "A bridging model for parallel computation," *Commun. ACM*, vol. 33, no. 8, pp. 103–111, Aug. 1990.

[6] C.-C. Chang, G. Czajkowski, T. von Eicken, and C. Kesselman, "Evaluating the Performance Limitations of MPMD Communication," in *Supercomputing, ACM/IEEE Conference*, 1997.

[7] K. Hopkinson, X. Wang, R. Giovanini, J. Thorp, K. Birman, and D. Coury, "Epochs: a platform for agent-based electric power and communication simulation built from commercial off-the-shelf components," *IEEE Transactions on Power Systems*, vol. 21, no. 2, May 2006.

[8] H. Georg, S. C. Müller, N. Dorsch, C. Rehtanz, and C. Wietfeld, "Inspire: Integrated co-simulation of power and ict systems for real-time evaluation," in *Smart Grid Communications (SmartGridComm), 2013 IEEE International Conference on*, Oct 2013.

[9] *Model-Based Integration Platform for FMI Co-Simulation and Heterogeneous Simulations of Cyber-Physical Systems*, vol. Proceedings of the 10th International Modelica Conference. Modelica Association and Linkoping University Electronic Press, March 2014.

[10] F. Kuhl, R. Weatherly, and J. Dahmann, *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Prentice Hall PTR, 1999.

[11] R. M. Fujimoto, *Parallel and Distributed Simulation Systems*, A. Y. Zomaya, Ed. Wiley, 2000.

[12] S.-Y. Wang, C.-C. Lin, Y.-S. Tzeng, W.-G. Huang, and T.-W. Ho, "Exploiting event-level parallelism for parallel network simulation on multicore systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 4, April 2012.

[13] F. Cremona, M. Lohstroh, S. Tripakis, C. Brooks, and E. A. Lee, "FIDE - An FMI Integrated Design Environment," October 2015, poster presented at the Eleventh Biennial Ptolemy Miniconference, Berkeley. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/1138.html