Project: **AXIOM - Agile, eXtensible, fast I/O Module for the cyber-physical era**
Grant Agreement Number: **645496**
Call: **ICT-01-2014: Smart Cyber-Physical Systems**

H2020 FRAMEWORK PROGRAMME

ICT-01-2014: Smart Cyber-Physical Systems

PROJECT NUMBER: 645496

Agile, eXtensible, fast I/O Module for the cyber-physical era

D7.1 – Initial AXIOM Evaluation Platform (AEP) definition and initial tests

Due date of deliverable: 31st January 2016
Actual Submission: 9th February 2016

Start date of the project: 1st February 2015 Duration: 36 months

Lead contractor for the deliverable: UNISI

Revision: See file name in document footer.

Project co-founded by the European Commission within the HORIZON FRAMEWORK PROGRAMME (2020)	
Dissemination Level: PU	
PU	Public
PP	Restricted to other programs participant (including the Commission Services)
RE	Restricted to a group specified by the consortium (including the Commission Services)
CO	Confidential, only for members of the consortium (including the Commission Services)

Change Control

Version#	**Date**	**Author**	**Organization**	**Change History**
0.1	31.01.2016	Roberto Giorgi	UNISI	v0.1
0.2	04.02.2016	Paolo Gai, Bruno Morelli, Stefano Garzarella	EVI	Minor typos

Release Approval

Name	**Role**	**Date**
Roberto Giorgi	WP Leader	08.02.2015
Roberto Giorgi	Project Coordinator for formal deliverable	09.02.2015

The following list of authors will be updated to reflect the list of contributors to the document.

Roberto Giorgi
Department of Information Engineering and Mathematics
University of Siena, Italy – (UNISI)

TABLE OF CONTENTS –

TABLE OF FIGURES –

GLOSSARY

ACP – Accelerated Coherency Port: an ARM AXI bus for (one-way) coherent operations
AEP – AXIOM Evaluation Platform
AXI – a proprietary protocol for buses introduced by ARM Ltd
AXIOM-acc – an FPGA accelerated system that performs a given function
AXIOM-arch – the architecture of an AXIOM (module or) board
AXIOM-core – the cores where the computations run in an AXIOM board
AXIOM-fpga – the programmable logic part in an AXIOM board
AXIOM-link – the interconnects that permits board-to-board communication in AXIOM
Bitstream – the binary code used for configuring the PL
BRAM – Block-RAM: a fast RAM that is available in the FPGA slices (in smaller blocks)
BSD -- BroadSword Document – In this context, a file that contains the SimNow machine description for a given Virtual Machine
CUDA – NVIDIA programming model for GPUs
Conduit – A software stub that connects GASNet to a given network protocol or programming model
Device – in this context: it is the physical system that runs a ‘device-tagged’ part of the code
DoA – Description of Action (acronym set by the European Commission)
DTS -- Distributed Thread Scheduler
DMA – Direct Memory Access: a separate master that can take over local memory transfers
DSE – Design Space Exploration
DSM – Distributed Shared Memory
eMMC – Embedded Multi Media Card
FPGA – Field Programmable Gate Array
FPGA-device – a specific accelerator that is implemented on the FPGA
GASNet – Global Address Space over Network: is a language-independent, low-level networking layer that provides network independent communication primitives
Infiniband – a high-performance (costly) NI
IP – Intellectual Property system (either hardware or software)
Mercurium – the OmpSs compiler
Nanos++ -- the OmpSs runtime
MGT – Multi-Gigabit Transceiver
MPI – Message Passing Interface: library for writing portable message-passing programs
PL – Programmable Logic: the purely FPGA part of a SoC like ZYNQ
PS – Processing System: the hardwired IPs of a FPGA-hybrid SoC like ZYNQ
MPSoC – Multi-Processor SoC
NI – Network Interface
OpenCL – Khronos group programming model for heterogeneous architectures
OmpSs – Extension of OpenMP programming model to support task dataflow programming
OmpSs@FPGA – FPGA extension of OmpSs
OmpSs@Cluster – Cluster extension of OmpSs
PCIe – PCI Express – standard for peripherals interconnection
PHY – the physical implementation of the network interface
QSPI – Quad Serial Peripheral Interface
RDMA – Remote DMA: a DMA that can work from one computer to another computer
ROI – Region of Interest
SoC – System on Chip
USB OTG – Universal Serial Bus On The Go
XSMLL – (pronounced “X-SMALL”) eXtended Shared-Memory Low-Level API
X-Thread – a self-contained thread that can be distributed across boards through XSMLL
ZYNQ -- A System-on-Chip commercialized by XILINX, which includes FPGA and CPUs

Executive summary

The main goals of the AXIOM-WP7 are:

• *Definition of the AXIOM evaluation platform (AEP) appropriate for Cyber-Physical Systems*

• *Definition and development of appropriate Design Space Exploration (DSE) tools and methodologies;*

• *Integrate in the evaluation platform the main results from all the all work packages*

During the first year of the AXIOM project, we developed the initial AEP and the DSE tools, while enabling all the partners to perform their experiments in a scientifically rigorous and repeatable way by using those common tools.

The AEP encompasses several tools, namely:

- The HP-Labs COTSon simulator, which is a powerful virtual platform that permits to run off-the-shelf operating systems like Linux and totally decouples the functional modeling and the timing (architectural) modeling; we are in particular interested to develop the timing models for novel AXIOM components and decide the appropriate partitioning of functionalities between software and hardware; the XSMLL API specification and deployment shows here such capabilities;
- The DSE tools, that are the necessary glue to properly manage the experiments, parallelize them, collect and visualize the results;
- The prototyping and development boards from FPGA vendor; our choice is to use boards Xilinx in this project although our platform is in principle portable to other vendor boards.

This document briefly illustrates the context of the AEP tools, describes the capabilities of such tools and shows some initial results that we were able to produce while advancing beyond the state-of-the art. In particular, we show how it is possible to distribute the work in the form of special threads that we call X-Threads across several cores and several boards.

An X-thread follows a novel memory consistency model based on dataflow concepts and permits scalability of our platform from 1 to 4 nodes (4 to 16 cores) in our initial experiments, while permitting a consistent distribution of data across several boards through a low-level API that we call XSMLL (pronounced “X-SMALL”, which stands for eXtended Shared Memory Low-Level API). In this way we overcome some important limitations of traditional Distributed Shared Memory models.

Therefore the objectives of the first year have been met or exceeded (e.g. with the XSMLL API specification and its implementation in the AEP tools and with the visualization tools).

1 Introduction

The main goal of the AXIOM project is to build a reference board for Smart Cyber-Physical Systems. To that end, in this Workpackage we intended to select and use the best suitable methodology and tools in order to assess the performance and to steer the design by exploring the most promising options (Design Space Exploration or DSE).

We illustrate the AXIOM Evaluation Platform and in particular the tools: COTSon, MYINSTALL, MYDSE, ESTRAI, RISGRAPH. The COTSon and MYDSE are by far the most complex ones. We do not describe here the Xilinx XC-706 platform (our reference platform) for which we have a large amount of documentation form the vendor [42].

1.1 Document structure

This document is organized as follows

- In Section 2 we briefly recall the AXIOM platform high-level architecture;
- In Section 3 we explain the motivations behind the AXIOM Evaluation Platform;
- In Section 4 we recall the state of the art and the COTSon simulator capabilities useful for this project;
- In Section 5 we illustrate through a driving example how we can implement in the simulator timing model based on the envisioned low-level support for thread distribution across boards
- In Section 6 we illustrate the DSE tools;
- In Section 7 we show some initial and successful experiment to distribute a single computation (matrix multiplication) across several boards (in total e boards with 4 cores each).

1.2 Relation to other deliverables

Deliverable D3.1 refers to the selection of initial kernels and benchmarks to be used for the initial analysis (this is also detailed in this deliverable).

Deliverable D4.1 refers to the programming model (OmpSs) and other activities to build a coherent software and hardware stack in coordination with WP4, WP5, WP6 (and this Workpackage).

Deliverable D6.1 describes the initial architecture for the first AXIOM board prototype (specified at month 9).

1.3 Tasks involved in this deliverable

This deliverable is the result of the work developed in tasks:

- Task 7.1 (month 1 - 3): Evaluation-Platform Setup (ALL PARTNERS)
- Task 7.2 (month 4 - 30): Continuous development of performance evaluation tools and DSE (Partners: UNISI, BSC, EVI, FORTH, SECO)
- Task 7.3 (month 4 - 30): Evaluation from kernels to benchmarks and final application code T5.3: Parallel programming library (UNISI, BSC, EVI, FORTH)

There is also a close collaboration with the following task (only reported here for reference):

- Task 5.1 Operating System (month 1-18)
- Task 5.2 (month 1 -18): Remote memory access

2 The AXIOM platform

We briefly recall here how the AXIOM platform is architected. The architecture is based on the following pillars (see also Figure 1 and Deliverable D6.1):

P1 FPGA, i.e. large Programmable Logic for acceleration of functions, soft-IPs, implementing specific AXIOM support for interconnects and scaling,

P2 General Purpose Cores, to support the OS and for running parts that make little sense on the other accelerators,

P3 High-Speed, Inexpensive Interconnects to permit scalability and deverticalise the technology, e.g., for toolchains,

P4 Open-Source Software Stack,

P5 Lower-Speed Interface for the Cyber-Physical world, such as Arduino [22] connectors, USB, Ethernet, WiFi.

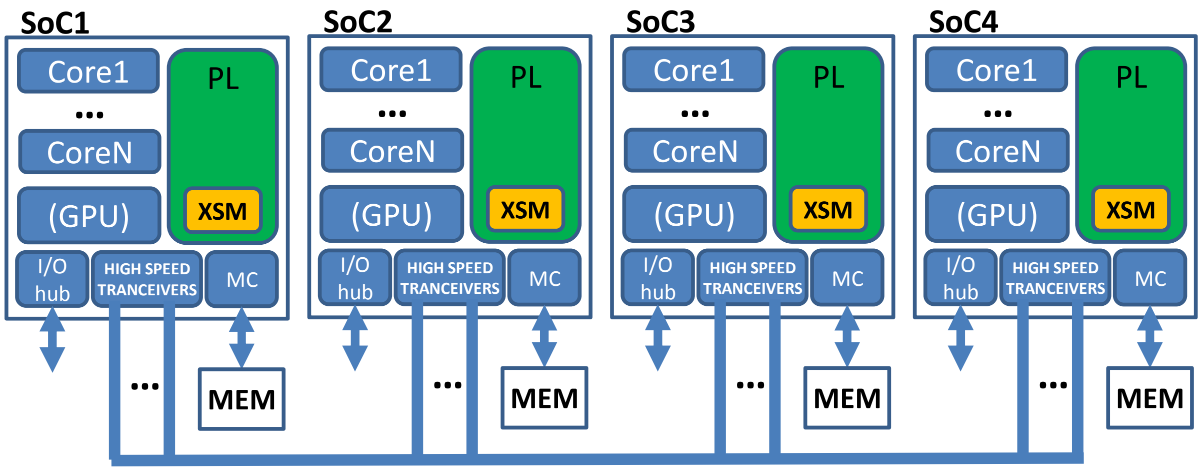


Figure 1: AXIOM Scalable Architecture. An instance consisting of four boards, each one based on the same System-on-Chip (SoC). GPU is an optional component (MC=Memory Controller. PL=Programmable Logic. XSM=eXtended Shared Memory).

Below we illustrate those pillars more in more detail.

[P1] In the first phase we will adopt one of the existing solutions such as the Xilinx Zynq [21], (Zynq is a chip-family, the chip can include a dual ARM Cortex-A9@1GHz, 4@6.25Gbps to 16@12.5Gbps transceivers, low-power programmable logic from 28k to 444k logic cells + 240 to 3020 KB BRAM + 80 to 2020 18x25 DSP slices, PCI express, DDR3 memory controller, 2 USB, 2 GbE, 2 CAN, 2SDIO, 2 UART, 2 SPI, 2 I2C, 4x32b GPIO, security features, 2 ADC@12bit 1Msps). The central hearth of the board is the FPGA SoC, so that it can make possible to integrate all the features, to provide customized and reconfigurable acceleration of the specific scenario where the board is deployed and to provide the substrate for board-to-board communication. In our roadmap, we are also considering other options that may be available soon such as the Xilinx Ultrascale+ [23] [43].

[P2] The general purpose cores are used for supporting a number of activities such as the Operating System (or a system task) but also whenever there is a sequential task which needs for more Instruction Level Parallelism rather than other forms of acceleration.

[P3] To keep the cost low we are initially oriented to use the FPGA transceivers and use standard and inexpensive (multiple) connectors such as the SATA connectors (without necessarily use the SATA protocol). Similar solutions had been adopted in the FORMIC board [24].

[P4] The recent success of SBCs such as the UDOO [25] and RaspberryPi further demonstrated the need for using open-source software. Linux has already become a reference example of how open-source software can widen the benefits at any level. While there is not yet a final consensus on which parallel programming model is best, we believe that adopting OmpSs [2] can easy the programmability by providing techniques familiar to the HPC programmer into the Embedded Computing community.

[P5] In order to interface with the physical world the platform includes support for Arduino connectors for General Purpose I/O and other standard interfaces such as the USB, Ethernet and WiFi. Not less important is the capability of interfacing with sensors and actuators or any other type of external shields as in the Arduino platform.

Moreover, X-Threads (see Section 5) make possible to bring together in a single platform all those elements and tackling cross-issues such as a better real-time scheduling: as the inputs should be available before execution of the X-Threads, the system can be more predictable too.

3 The AXIOM Evaluation Platform (AEP) Setup

During the first three months (Task T7.1) of this project, we had a main task of deciding the best choice at that moment in time for a promising and achievable platform for the AXIOM system and consequently to setup an appropriate DSE methodology. Essentially the AEP is made of the combination of two important tools: the HP-Labs COTSon simulator and the Xilinx Zynq based platforms as explained below.

In the proposal phase we tentatively indicated the Xilinx XC702 development board, which is based on the Zynq-7000 platform. After considering the possible options at that moment, including interesting products from, e.g. Altera Arria-V SoC, we preferred the Xilinx for the following main reasons:

- More mature product
- Existing collaborations, e.g., between BSC and Xilinx
- More promising roadmap including 64-bit products like the Ultrascale+ (not yet available at the time of deciding the FPGA prototyping platform but later on in October 2015 first sampled successfully and under initial distribution at the time of writing January 2016). The Ultrascale+ has been inserted in the AXIOM roadmap for WP6 (cf. D6.1).

The commonly agreed choice was then the Xilinx-XC706 development board. Each of the partners then acquired one or two of those boards (two for those partners directly involved in the experimentation of the multi-board framework and one board in the case of those partners mostly interested in the application setup).

The second fundamental choice was about the simulation platform. Given the research nature of the goals of this project we also needed a more flexible platform for the Design Space Exploration (DSE) in order to better understand e.g. whether some bottlenecks are due to, e.g., the congestion on a bus or to insufficient cache size and this is less flexible or impossible to be seen on the FPGA prototyping platform.

Partner UNISI had a considerable experience (5+years developed during the TERAFLUX project [15] [16] and ERA project [44] [46] [47]) on the HP-Labs COTSon [9][10], therefore partner UNISI immediately provided access to all partners to such platform indicating a road for fast prototyping of our platform: in fact (as explored in the Task T5.2), as of May 2015 we enabled the possibility to run a computations on multiple nodes by specifying a low-level API (called XSMLL [7]) that enables the distribution of threads (called X-Threads) across not only the (AXIOM-)cores on the same board but also across the AXIOM-cores on multiple AXIOM-boards [5]. Also: COTSon include an interface to the HP McPAT tool [20] for estimating the consumed power.

Therefore the second fundamental choice was to rely both on the FPGA and the COTSon simulator. The proposed methodology requires first exploring and modeling parts on the simulator and then, once the DSE is completed, implementing them on the FPGA based prototypes. This has also the considerable advantage of allowing immediately to develop the software stack early (WP3-WP4-WP5) and therefore face more early in the project the possible challenges.

It has to be noted that at the current time COTSon relies for its functional execution on the AMD SimNow virtualizer: this has the advantage of providing a realistic platform that is the same internal tool used by AMD for developing their processors and platform. A special interface is available in COTSon to communicate the internal state of the SimNow so that the two tools can seamlessly cooperate. One objection, which is typically raised at first sight, is that the current SimNow relies on x86 architecture rather than the ARM architecture. However it has to be noted that:

i) previous studies discovered that the performance and power studies of ARM and x86 do not depend on the specific ISA that is used [17] but rather they depend on the type and quantity of resources that are included in each platform (i.e., bus-widths, cache size, cache organization, instruction window size and similar): "We find that ARM and x86 processors are simply engineering design points optimized for different levels of performance, and there is nothing fundamentally more energy efficient in one ISA class or the other. The ISA being RISC or CISC seems irrelevant". Therefore we postponed the detailed analysis of the specific feature details provided by each platform to a later step since the most relevant DSE parameters and choices are conditioned by other aspects which are portable throughout the ISAs such as the availability of certain software components or library such as GASNet [1], MPI [4], OmpSs [2][3];
ii) AMD is developing chips (e.g., the K12 [20] and A1100 [19]) that also rely in ARM and in the foreseeable future there could be extensions of the SimNow to support also that architecture.

As of Milestone 7.1 (in synchronization with the end of task T7.1), we therefore commonly agreed on the following technical aspects in order to simplify and steer the subsequent research steps and prototyping:

1) The Operating System of the target system (AXIOM-board) will be Linux based on the distribution Ubuntu 14.04LTS. This should guarantee reasonable stability and support from the part vendors as well as good support for the peripherals and related drivers. Being this an "LTS" (Long Term Support) distribution means that software updates will be guaranteed until year 14+5= (20)19 so that the AXIOM-board could safely rely on that. A switch to a more recent version could be considered but evaluated carefully at later stages in this project.
2) For simplicity also the supported host systems should use the Ubuntu 14.04LTS. This also simplifies the deployment of applications and tools usage and understanding.
3) We will use both the HP-Labs COTSon simulator and the Xilinx ZC-706 boards as well as the first AXIOM-board prototypes from partner SECO as soon as they are available. At any time the use of the simulator could be helpful to debug problems by producing execution traces of the application or to have a reference comparison.
4) Start the exploration with the following three simple benchmarks: i) matrix-multiplication; ii) CJPEG; iii) face detection. The reason is both due to simplicity and representativeness
5) As a further step we agreed to consider part or all of the ERA benchmark suite [45], which is well representative of the two fundamental applications that we aim to demonstrate in this project, namely Smart-Home and Smart Video-surveillance. For a later phase in the project we will then explore the final application prototypes.

As of aspects 1 and 2, as of month 3 (March 2015) partners UNISI and BSC provided on the common repository pre-installed images that could both:

- Run on the SimNow simulator
- Run on a real x86 machine

This is greatly helpful to run any software or tool needed for our project in a realistic framework.

To even accelerate the start-off of the simulator, UNISI provided the so called Virtual-Machine Snapshots (called "BSD" in SimNow terminology) so that a given benchmark can be launched from the shell prompt (without the boot time).

4 A brief review of the State-of-the-Art in the Simulation and Design Space Exploration Technologies

Design Space Exploration and its automation is an important part of modern performance evaluation and power estimation methodologies [11] [12] [13] [14] [15] [16]. Recent projects like MULTICUBE [12] have proposed systematic approaches for DSE. However some of those approaches require considerable time to be ported to other frameworks. In this case, our project is focusing on a heterogeneous architecture encompassing both programmable logic, i.e., FPGA and off-the-shelf cores such as the ARM Cortex A9. Therefore we decided to adopt a multi-pronged approach that is based on both a well-known architectural simulator (HP-Labs COTSon [9] [10]) and development boards and prototypes based on the Xilinx Zynq SoC [21] [42].

In Table 1 we compare some state of the art simulators (adapted from [27]) which shows that one of the main feature is the ability to separate the functional description and the architectural modeling (called "timing model") through the so called "functional directed" approach. There exist at least four approaches for a (timing) simulation depending on the relationship between the "functional model" (fm) and the "timing model" (tm) [28]:

- "functional-first" or "trace-driven", the fm is run first and separately and the tm is run later on in a completely decoupled fashion (all fm is run before the tm is run);
- "timing directed" or "execution driven", the fm and tm are closely coupled (no decoupling);
- "timing-first", the tm drives the fm, both are completely decoupled, but the function has to checked later on and eventually undone;
- "functional-directed", the fm drives the tm, both are completely decoupled, the function is always the right one but we need a timing feedback from tm to correct the timing. [10].

Table 1: Comparison of main features of several timing simulators

	Sniper	Graphite	Gem5	MARSx86	COTSon
Timing-directed/ integrated			X		
Functional-directed	X	X		X	X
User-level	X	X	X		
Full-system			X	X	X
Supported Architectures	x64	x64	X64 Alpha SPARC	x64	x64
Parallel (in-node)	X	X			
Parallel (multi-node)					X
Shared Caches	X		X	X	X

4.1 A COTSon overview

Compared to other approaches, COTSon uses the so called "functional-directed" approach. The COTSon simulator permitted us to immediately run the chosen platform (see previous section), i.e., the Ubuntu 14.04 distribution of the Linux-based operating system including the OmpSs compiler (cf. Deliverable D4.1) and related applications.

In Figure 2 the green part represents the functional-models. This allows us to run essentially any currently available software as in a real machine. The "mediator" represents the model of a switch and our aim is to modify it in order to model the behavior of our custom interconnects. Another instance of the mediator can already model and Ethernet based communication, e.g., we can perform a "ping-test" with two or more nodes virtually connected through that component as they were in an Ethernet based LAN. The motivation for the multiple interconnects derives from the AXIOM project design that aims to separate the traffic for building a multi-board system and the traffic for the internet-related connection. With the COTSon mediator we can model both cases. The SimNow is the Virtual Machine which models all details of a computer. AMD is also providing a separate SDK to model any specific board that has to be plugged in such as a network card or a GPU.

Again in Figure 2 the blue part represents the timing models. This is what we are mostly interested to develop during the project, i.e., the specific modules or architecture to support the easy programmability in an efficient way (XSMLL and X-Threads – c.f. next section for a driving example). Moreover this part is what can implement simulation acceleration techniques, such as dynamic sampling [29], and the tracing, profiling, statistics collection.

Finally, again in Figure 2 the orange part represents the scripting glue that controls, e.g., the setup of sandboxes for the parallel simulation instances of SimNow, the boot/resume/stop of each virtual machine.

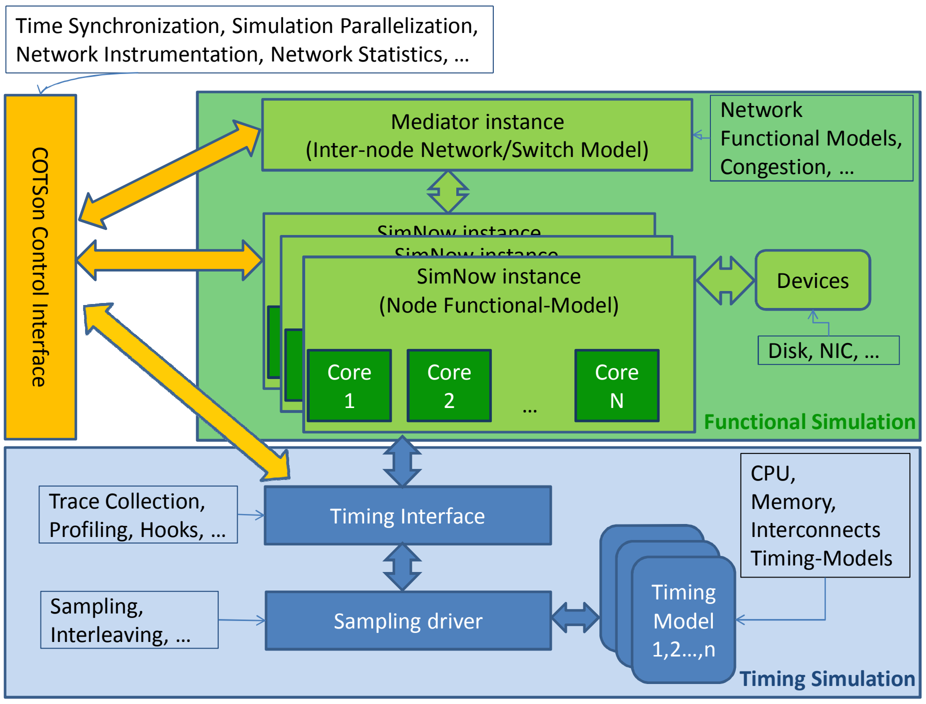

Time Synchronization, Simulation Parallelization, Network Instrumentation, Network Statistics, …
COTSon Control Interface
Mediator instance (Inter-node Network/Switch Model)
Network Functional Models, Congestion, …
SimNow instance
SimNow instance
SimNow instance (Node Functional-Model)
Core 1
Core 2
…
Core N
Devices
Disk, NIC, …
Functional Simulation
Trace Collection, Profiling, Hooks, …
Timing Interface
Sampling, Interleaving, …
Sampling driver
Timing Model 1,2…,n
CPU, Memory, Interconnects Timing-Models
Timing Simulation

Figure 2: Main components of the COTSon simulator.

5 A driving example: adopting a new execution model based on Dataflow Threads (X-Threads)

As a driving example to show how COTSon can model what we aim to design in the AXIOM project, we used the XSMLL (eXtended Shared Memory Low-Level) API (see also Appendix A) to demonstrate how we could easily partition functionalities between software and hardware.

5.1 Description of the Thread Management problem

Using a general paradigm to manage threads can lead to good performance, such as in the case of P-threads, Cilk, OpenMP. However, these models suffer performance penalties when synchronization and distribution of data is not managed properly [30]. By re-organizing the execution is such a way the threads follow more closely the data flow of the program, such as with DF-Threads (in this context extended as X-Threads), better scalability can be achieved [31] (Figure 3).

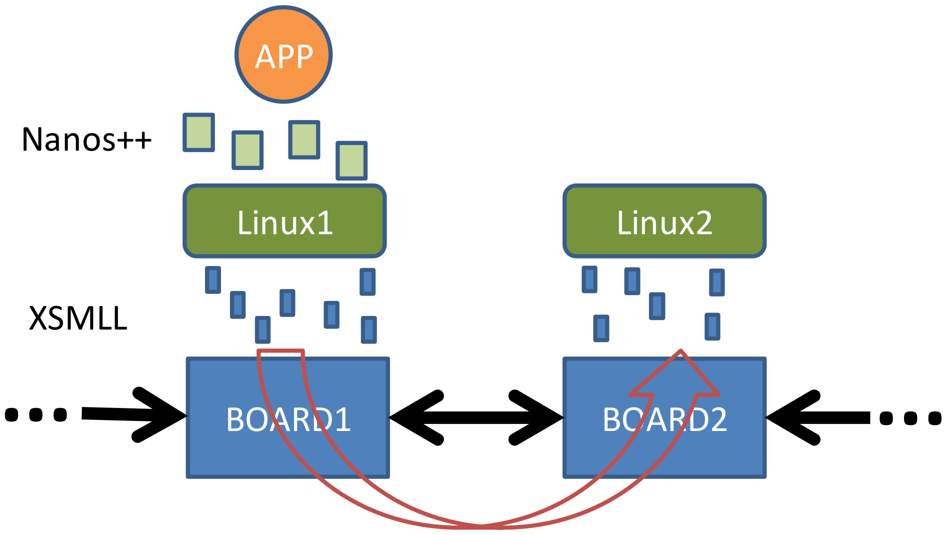


Figure 3: The X-Threads distribution. Nanos++ generates coarser grained threads that could be further distributed as X-Threads across several nodes.

X-Threads are best implemented in hardware through the use of a Distributed Thread Scheduler [32] (DTS). The DTS tries to solve the following challenges:

- at the system level, all the available resources and the healthiness of the whole system must be considered in a distributed fashion: if a part breaks the remaining of the system should continue to work [33];
- at low-level, the fine-grain threads coming from the adoption of the data-flow execution model must be distributed across the computing elements (CPUs, FPGAs).

This means to understand at run-time what is the best resource assignment (scheduling/mapping on CPU or reconfigurable HW) to a task (or thread), according to multiple goals (e.g., performance/QoS, power consumption minimization, thermal hotspots). The policies should operate effectively both in a single application and a mixed workload scenario. The scheduler can be further extended to enable it distributing fine-grain threads across the different boards or MPSoCs.

In order to reduce the thread management overhead, the DTS needs to be accelerated in hardware, by mapping its structure into the FPGA. The hardware thread support is represented in Figure 1 by the eXtended Shared Memory Low-Level (XSMLL) block. Standard high-speed and low-latency interconnections (e.g., PCIe 3.0) may provide enough bandwidth, but the exact interconnects is under exploration [36] (c.f. Deliverable D4.1 1 and Task 6.5).

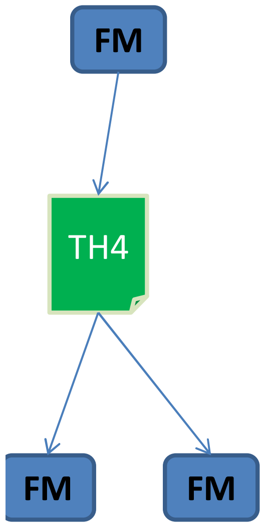


Figure 4: Relation between Frames (FM) and an X-Thread. An X-Thread reads its local data from an input frame and may produce local outputs for other consumer threads by writing in a number of output frames.

Definition 1 -- X-thread (new incarnation of DF-thread [30]) is a function that expects no parameters and returns no parameters (Figure 4).

The body of this function can refer to any memory location for which it has got the pointer through XSM function calls (e.g., `xpreload`, `xpoststor`, `xsubscribe`, ...). An X-thread is identified by an object of type `xtid_t` (X-thread identifier). In other words:

```
typedef void (*xthread_t)(void)
```

Definition 2 -- INPUT_FRAME: A buffer which is allocated in the local memory and contains the input values for the current X-thread (Figure 4).

Definition 3 -- OUTPUT_FRAME: A buffer which is allocated in the local memory and contains values to be used by other X-threads, i.e., consumer X-threads (Figure 4).

Definition 4 -- SYNCHRONIZATION_COUNT: A number which is initially set to the number of input values (or events) needed by an X-thread. The SYNCHRONIZATION_COUNT has to be decremented each time the expected data is written in an OUTPUT_FRAME.

XSMLL basic functions[1] (see Appendix A for a more extensive description of the current implementation):

[1] `xtid_t` is type that combines two objects: a thread identifier (a number from 1 through a `XSM_MAX_THREAD_X_OFF(tid)_ID`) and an offset inside its input frame. Given an `'xtid_t tid'` the thread number is obtained by the `_X_TID(tid)` macro, while the frame offset is obtained by the macro.

XSCHEDULE:

```
xtid_t xschedulez(xthread_t ip, uint32_t sc, uint32_t sz)
```

Schedules an X-thread whose name (instruction pointer) is specified by `'ip'`.

The thread expects `sc` inputs which are stored in its INPUT_FRAME of size `'sz'` (in bytes). Returns an X-thread identifier (with 0 internal offset -- see above definition). `'sc'` is also called the SYNCHRONIZATION_COUNT of the X-thread (see above definition). Implicitly allocates the data frame It only *schedules* the code The code will be activated on availability of ALL input data and resources.

XDESTROY:

```
void xdestroy()
```

Called at the end of an X-thread to signal that any allocated resource belonging to the current X-thread can be freed up.

The aim of the AXIOM project is also towards an energy-efficient improvement of the performance of applications, along with benefits in terms of modular scalability of the platform. In the next sections we will describe the first experiments that enabled us to have more confidence with this approach.

5.1.1 Installation and more examples

XSMLL installation instructions

https://git.axiom-project.eu/?p=XSMLL;a=blob;f=README00-xsmll_install

The internal repository also provides more XSMLL examples (see Appendix B for some code samples)

fibx.c

simple recursive fibonacci program that generates lots of threads (useful for testing xschedule)

mmx.x

block-striped matrix multiplier program that uses different memory consistency on the matrices (useful for testing shared memory)

test1.c

simple test of several functions communicating

test2.cc

graph construction with dynamic nodes that are added and deleted useful to show how to use pointers with X-threads.

6 The Design Space Exploration Tools

The DSE tools developed in the AXIOM project aim at providing the necessary substrate to implement a proper scientific methodology for experimentation. An important aspect is the open–access [36] [37] and reproducibility of the experiments [38] [39] [40].

For a proper simulation methodology at least the following steps have to be followed (and have to be therefore automatized) for carrying out the experiments:

- SET THE MACHINE: we must use the same BSD file with very small variations (ideally just the no. of cores: e.g., 1, 2, 4)
- SET THE ALGORITHM: while comparing different programming models, compilation option, etc. we must keep the variation at the minimum (e.g., sequential vs. Cilk vs. OmpSs vs. DSMs like Jump [41]). A common mistake is to compare something which is highly optimized code vs. a substantially different method to solve the same problem (e.g. Fibonacci Recursive vs. Fibonacci Iterative vs. Fibonacci Lookup)
- SET THE INPUT DATA: the input data MUST be the same across different architectural or software parameters or we may end up in comparing a very different parallelism or, even worse, just get not-comparable results
- PRECALCULATE YOUR EXPECTED OUTPUT AND COMPARE IF OK: the output data should be computed separately from the running test so that we can check if the test has run till completion or there was any error (bug, segmentation or anything else). The common mistake is to get an ultrafast program because it prints an error message after the first instruction.
- DEFINE THE REGION OF INTEREST (ROI) OF THE PROGRAM: any measurement should be taken from a well-defined part of the code that avoids the initial/final print statements - unless you are interested to study effects on the I/O.
- DEFINE THE METRICS YOU ARE INTERESTED IN: e.g., execution time, miss rate, power, faults ...make sure you use the right estimator for each metric; identify any source of "noise"; eventually repeat the same test/experiment several times to filter out any "random error".
- MAKE SURE THAT YOU USE THE RIGHT PROBE/METHODOLOGY: e.g. use of "gettimeofday" vs. "hw counters" vs. "sw probes" to measure the execution time.
- DESCRIBE IN DETAIL ALL THE TEST SETUP: similarly to SPEC (www.spec.org) we need to prepare a sort of identification tag to each experiment and being able to check what were the exact conditions of the tests.
- TAKE NOTE OF THE HOST METRICS TOO: we can't afford to spend months for a single test. Therefore, we need to monitor how many resources in terms of host-wall-clock time, host-memory, host-cores, etc. are necessary to perform a certain test.

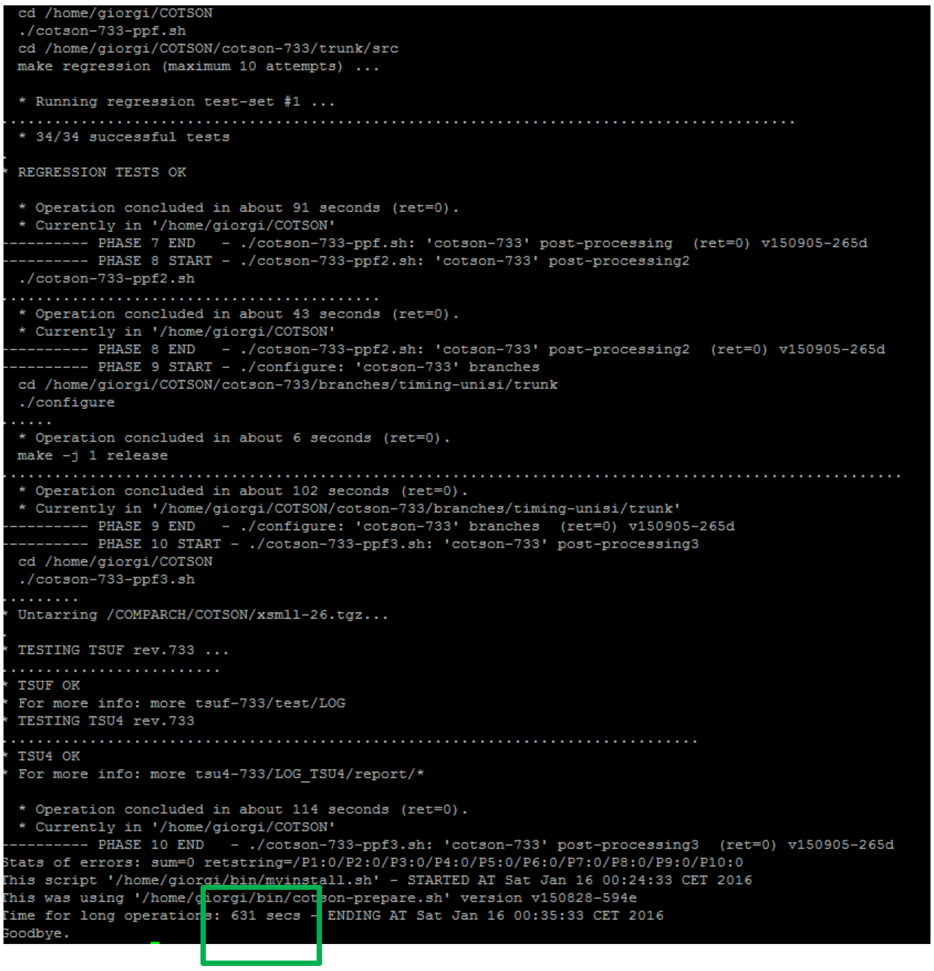
As explained in the next section the DSE tools (MYINSTALL, MYDSE) aim at describing those details in a methodologically correct way.

6.1 MYINSTALL – repeatable and easy tool installation

The purpose of this tool is to easy the installation process of the simulator and the related hard disk. As explained, in AXIOM we need to define the hard-disk image containing the selected OS (Ubuntu 14.04 Linux distribution in our case) with the OmpSs tools and OmpSs applications. This process can be tedious and prone to errors. Therefore we designed a specific tool so that:

i) The basic host software simulator is installed. A specific (previous release) of the simulator can be selected in order to perform regression tests or compare several installations.
ii) Hard-disk image and BSDs are prepared and installed so that the experimentation phase can easily start.
iii) Regression tests are automatically performed at installation time in order to make sure that the software is properly patched, compiled and installed.
iv) Additional plugins and temporary patches can be installed automatically.

Typically this process may now take as much as 10 to 15 minutes and is all automatic (Figure 5). Previously, it could have taken a day or so and might have repeated everytime there was some doubts on the installation. Moreover, this can be easily repeated on a number of (parallel) simulation hosts and by project partners or for future use by the scientific community. This tool has been proved largely useful to improve the productivity of the experimentation and design cycle.



```
 cd /home/giorgi/COTSON
 ./cotson-733-ppf.sh
 cd /home/giorgi/COTSON/cotson-733/trunk/src
 make regression (maximum 10 attempts) ...

 * Running regression test-set #1 ...
.........................................................................................
 * 34/34 successful tests
.
* REGRESSION TESTS OK

 * Operation concluded in about 91 seconds (ret=0).
 * Currently in '/home/giorgi/COTSON'
--------- PHASE 7 END   - ./cotson-733-ppf.sh: 'cotson-733' post-processing  (ret=0) v150905-265d
--------- PHASE 8 START - ./cotson-733-ppf2.sh: 'cotson-733' post-processing2
 ./cotson-733-ppf2.sh
.........................................
 * Operation concluded in about 43 seconds (ret=0).
 * Currently in '/home/giorgi/COTSON'
--------- PHASE 8 END   - ./cotson-733-ppf2.sh: 'cotson-733' post-processing2  (ret=0) v150905-265d
--------- PHASE 9 START - ./configure: 'cotson-733' branches
 cd /home/giorgi/COTSON/cotson-733/branches/timing-unisi/trunk
 ./configure
.....
 * Operation concluded in about 6 seconds (ret=0).
 make -j 1 release
.......................................................................................................
 * Operation concluded in about 102 seconds (ret=0).
 * Currently in '/home/giorgi/COTSON/cotson-733/branches/timing-unisi/trunk'
--------- PHASE 9 END   - ./configure: 'cotson-733' branches  (ret=0) v150905-265d
--------- PHASE 10 START - ./cotson-733-ppf3.sh: 'cotson-733' post-processing3
 cd /home/giorgi/COTSON
 ./cotson-733-ppf3.sh
........
* Untarring /COMPARCH/COTSON/xsmll-26.tgz...
.
* TESTING TSUF rev.733 ...
........................
* TSUF OK
* For more info: more tsuf-733/test/LOG
* TESTING TSU4 rev.733
...............................................................................
* TSU4 OK
* For more info: more tsu4-733/LOG_TSU4/report/*

 * Operation concluded in about 114 seconds (ret=0).
 * Currently in '/home/giorgi/COTSON'
--------- PHASE 10 END   - ./cotson-733-ppf3.sh: 'cotson-733' post-processing3  (ret=0) v150905-265d
Stats of errors: sum=0 retstring=/P1:0/P2:0/P3:0/P4:0/P5:0/P6:0/P7:0/P8:0/P9:0/P10:0
This script '/home/giorgi/bin/myinstall.sh' - STARTED AT Sat Jan 16 00:24:33 CET 2016
This was using '/home/giorgi/bin/cotson-prepare.sh' version v150828-594e
Time for long operations: 631 secs - ENDING AT Sat Jan 16 00:35:33 CET 2016
Goodbye.
```

Figure 5: The several phases of the MYINSTALL tool. This includes, e.g., patches, regression test. This installation took 631 seconds as can be seen from the last lines and it was completely successful ("Goodbye" string).

6.2 MYDSE – design space exploration tool

The SimNow virtualizer is a very complex tool (like a real machine) and is prone to failure or errors. Managing the simulations in case of a large number of design points to be explored is almost unfeasible if done manually. Therefore we found necessary to define a specific tools (MYDSE) with the following goals:

i) Specifying a DSE experiment through a small configuration file ("INFOFILE", see Figure 7);
ii) Distributing the simulation across an arbitrary number of simulation hosts (currently we use a cluster of more than 20 machine at the location of partner UNISI);
iii) Managing automatically cases when a simulations fails or get stuck for a given time threshold by killing the simulation and all related processes;
iv) Properly using the same binary across multiple hosts with different GLIBC libraries and compilation tools binaries (depending on the library and compilation tools the simulation results may differ due to a different benchmark binary that get pulled into the guest);
v) Collect in an ordered way the several files from a single simulation point (Figure 6) ad from several simulation point belonging to the same experiment (Figure 8);
vi) Monitoring and controlling the simulation loop (Figure 9, Figure 10);
vii) Interpreting user formulas to extract aggregated information from several basic statistics (e.g. calculating the miss rate starting from the read miss and write miss);
viii) Automatically trying to re-execute the simulations that eventually fail;
ix) Inserting the code for marking the ROI;
x) Parsing and updating the configuration files of a simulation starting from a given template;
xi) Supporting different execution model such as XSMLL, the standard one or others;

Currently the MYDSE script is more than 200kB of shell scripting.

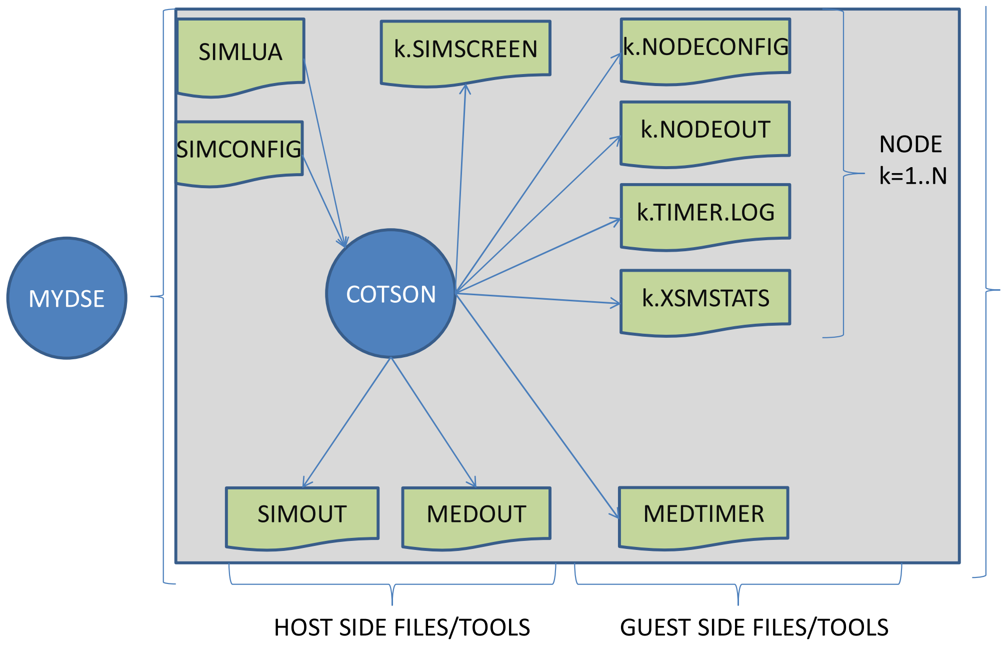
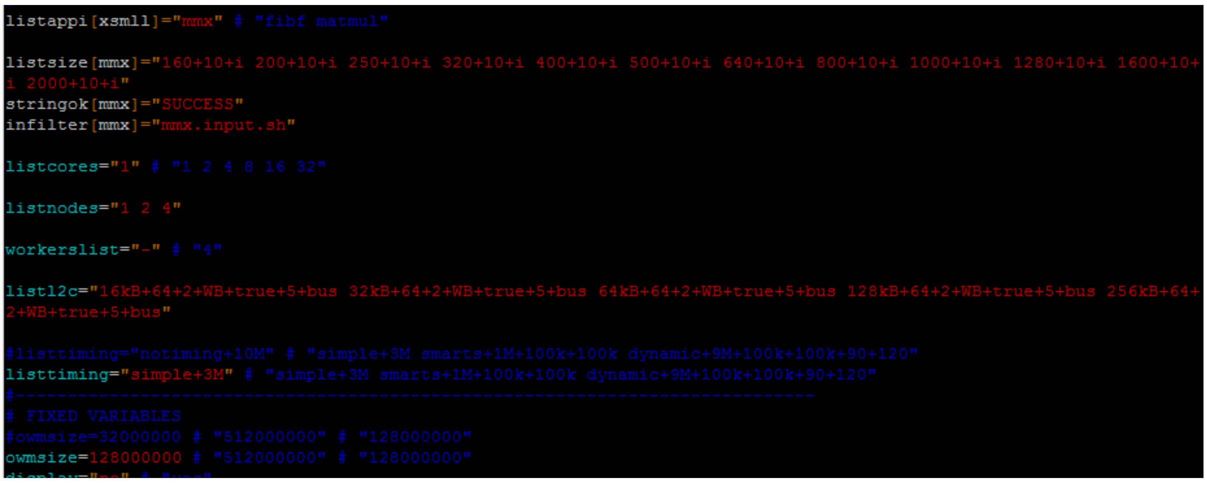
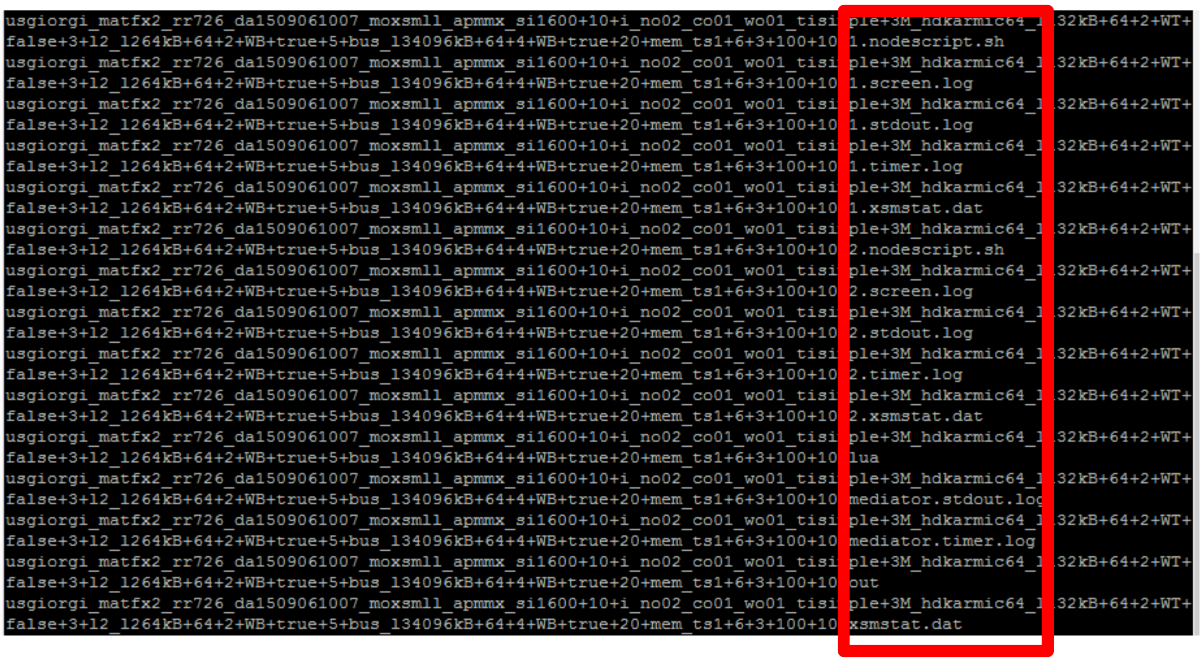


Figure 6: COTSON and MYDSE tool flow. COTSon needs and produces several input and output file for many experiments: this is wrapped and properly managed by the MYDSE tool.



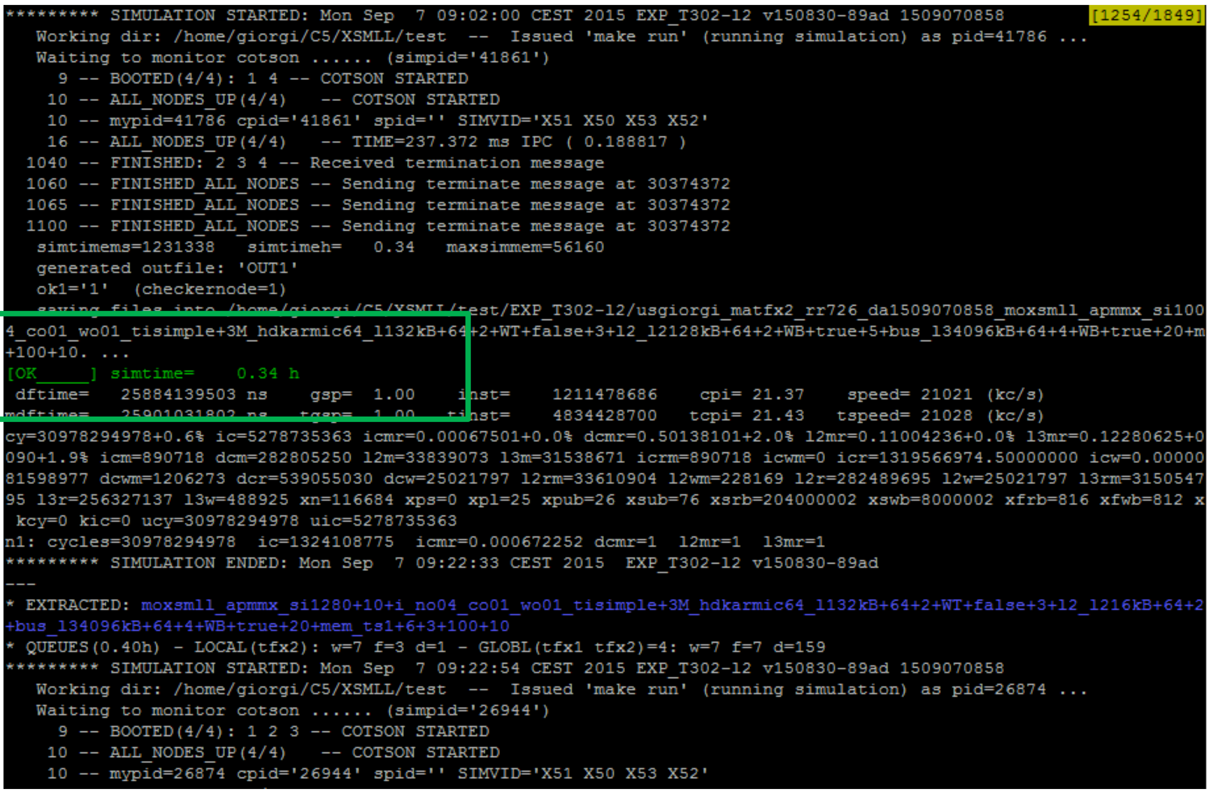
```
listappi[xsmll]="mmx" # "fibf matmul"

listsize[mmx]="160+10+i 200+10+i 250+10+i 320+10+i 400+10+i 500+10+i 640+10+i 800+10+i 1000+10+i 1280+10+i 1600+10+
i 2000+10+i"
stringok[mmx]="SUCCESS"
infilter[mmx]="mmx.input.sh"

listcores="1" # "1 2 4 8 16 32"

listnodes="1 2 4"

workerslist="-" # "4"

listl2c="16kB+64+2+WB+true+5+bus 32kB+64+2+WB+true+5+bus 64kB+64+2+WB+true+5+bus 128kB+64+2+WB+true+5+bus 256kB+64+
2+WB+true+5+bus"

#listtiming="notiming+10M" # "simple+3M smarts+1M+100k+100k dynamic+9M+100k+100k+90+120"
listtiming="simple+3M" # "simple+3M smarts+1M+100k+100k dynamic+9M+100k+100k+90+120"
#-----------------------------------------------------------------------------
# FIXED VARIABLES
#owmsize=32000000 # "512000000" # "128000000"
owmsize=128000000 # "512000000" # "128000000"
```

Figure 7: INFOFILE for a DSE experiment. In this case we are specifying a XSMLL execution model, a benchmark 'mmx', 10 different inputs (each one is a triplet of values), 1 core per node, three node configurations (1, 2 4 nodes) and 5 different caches size together with other simulation parameters.



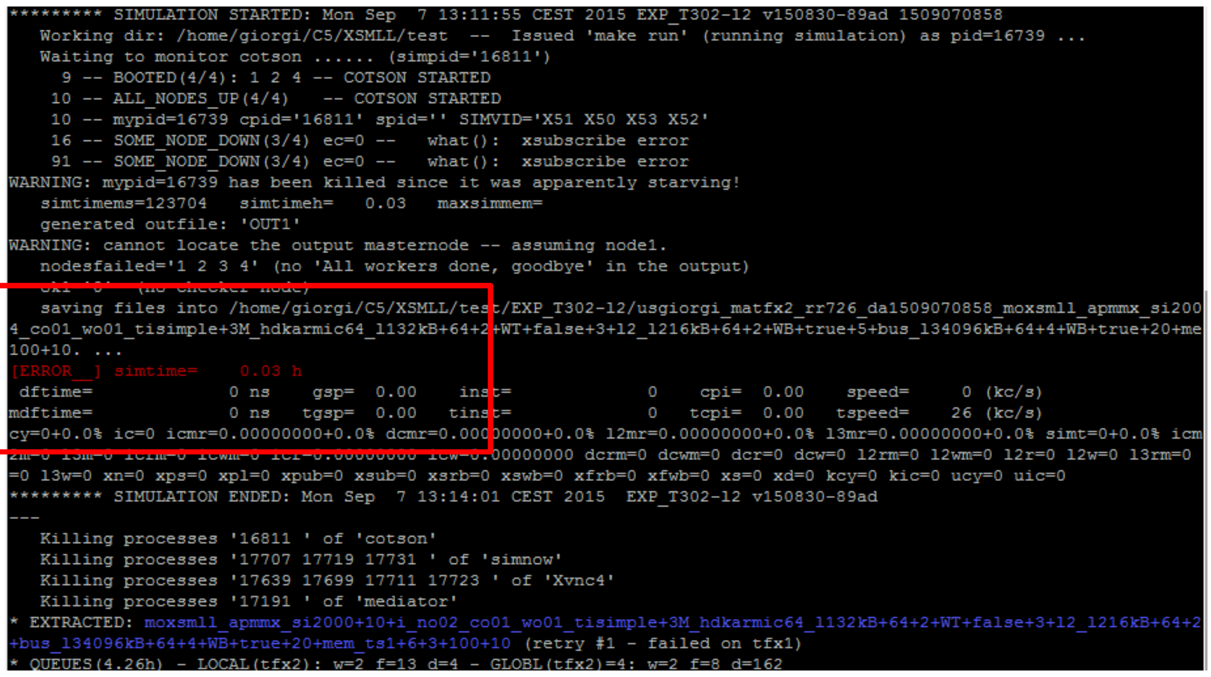
```
usgiorgi_matfx2_rr726_da1509061007_moxsmll_apmmx_si1600+10+i_no02_co01_wo01_tisi ple+3M_hdkarmic64_1 32kB+64+2+WT+
false+3+l2_l264kB+64+2+WB+true+5+bus_l34096kB+64+4+WB+true+20+mem_ts1+6+3+100+10 1.nodescript.sh
usgiorgi_matfx2_rr726_da1509061007_moxsmll_apmmx_si1600+10+i_no02_co01_wo01_tisi ple+3M_hdkarmic64_1 32kB+64+2+WT+
false+3+l2_l264kB+64+2+WB+true+5+bus_l34096kB+64+4+WB+true+20+mem_ts1+6+3+100+10 1.screen.log
usgiorgi_matfx2_rr726_da1509061007_moxsmll_apmmx_si1600+10+i_no02_co01_wo01_tisi ple+3M_hdkarmic64_1 32kB+64+2+WT+
false+3+l2_l264kB+64+2+WB+true+5+bus_l34096kB+64+4+WB+true+20+mem_ts1+6+3+100+10 1.stdout.log
usgiorgi_matfx2_rr726_da1509061007_moxsmll_apmmx_si1600+10+i_no02_co01_wo01_tisi ple+3M_hdkarmic64_1 32kB+64+2+WT+
false+3+l2_l264kB+64+2+WB+true+5+bus_l34096kB+64+4+WB+true+20+mem_ts1+6+3+100+10 1.timer.log
usgiorgi_matfx2_rr726_da1509061007_moxsmll_apmmx_si1600+10+i_no02_co01_wo01_tisi ple+3M_hdkarmic64_1 32kB+64+2+WT+
false+3+l2_l264kB+64+2+WB+true+5+bus_l34096kB+64+4+WB+true+20+mem_ts1+6+3+100+10 1.xsmstat.dat
usgiorgi_matfx2_rr726_da1509061007_moxsmll_apmmx_si1600+10+i_no02_co01_wo01_tisi ple+3M_hdkarmic64_1 32kB+64+2+WT+
false+3+l2_l264kB+64+2+WB+true+5+bus_l34096kB+64+4+WB+true+20+mem_ts1+6+3+100+10 2.nodescript.sh
usgiorgi_matfx2_rr726_da1509061007_moxsmll_apmmx_si1600+10+i_no02_co01_wo01_tisi ple+3M_hdkarmic64_1 32kB+64+2+WT+
false+3+l2_l264kB+64+2+WB+true+5+bus_l34096kB+64+4+WB+true+20+mem_ts1+6+3+100+10 2.screen.log
usgiorgi_matfx2_rr726_da1509061007_moxsmll_apmmx_si1600+10+i_no02_co01_wo01_tisi ple+3M_hdkarmic64_1 32kB+64+2+WT+
false+3+l2_l264kB+64+2+WB+true+5+bus_l34096kB+64+4+WB+true+20+mem_ts1+6+3+100+10 2.stdout.log
usgiorgi_matfx2_rr726_da1509061007_moxsmll_apmmx_si1600+10+i_no02_co01_wo01_tisi ple+3M_hdkarmic64_1 32kB+64+2+WT+
false+3+l2_l264kB+64+2+WB+true+5+bus_l34096kB+64+4+WB+true+20+mem_ts1+6+3+100+10 2.timer.log
usgiorgi_matfx2_rr726_da1509061007_moxsmll_apmmx_si1600+10+i_no02_co01_wo01_tisi ple+3M_hdkarmic64_1 32kB+64+2+WT+
false+3+l2_l264kB+64+2+WB+true+5+bus_l34096kB+64+4+WB+true+20+mem_ts1+6+3+100+10 2.xsmstat.dat
usgiorgi_matfx2_rr726_da1509061007_moxsmll_apmmx_si1600+10+i_no02_co01_wo01_tisi ple+3M_hdkarmic64_1 32kB+64+2+WT+
false+3+l2_l264kB+64+2+WB+true+5+bus_l34096kB+64+4+WB+true+20+mem_ts1+6+3+100+10 lua
usgiorgi_matfx2_rr726_da1509061007_moxsmll_apmmx_si1600+10+i_no02_co01_wo01_tisi ple+3M_hdkarmic64_1 32kB+64+2+WT+
false+3+l2_l264kB+64+2+WB+true+5+bus_l34096kB+64+4+WB+true+20+mem_ts1+6+3+100+10 mediator.stdout.log
usgiorgi_matfx2_rr726_da1509061007_moxsmll_apmmx_si1600+10+i_no02_co01_wo01_tisi ple+3M_hdkarmic64_1 32kB+64+2+WT+
false+3+l2_l264kB+64+2+WB+true+5+bus_l34096kB+64+4+WB+true+20+mem_ts1+6+3+100+10 mediator.timer.log
usgiorgi_matfx2_rr726_da1509061007_moxsmll_apmmx_si1600+10+i_no02_co01_wo01_tisi ple+3M_hdkarmic64_1 32kB+64+2+WT+
false+3+l2_l264kB+64+2+WB+true+5+bus_l34096kB+64+4+WB+true+20+mem_ts1+6+3+100+10 out
usgiorgi_matfx2_rr726_da1509061007_moxsmll_apmmx_si1600+10+i_no02_co01_wo01_tisi ple+3M_hdkarmic64_1 32kB+64+2+WT+
false+3+l2_l264kB+64+2+WB+true+5+bus_l34096kB+64+4+WB+true+20+mem_ts1+6+3+100+10 xsmstat.dat
```

Figure 8: A set of input and output files related to a single DSE point. All files are properly renamed so that they can be associated to the same experiment which is performed by a given user (giorgi) on a given machine (tfx2) with a specific simulator revision *726) at a given date_time (150906 10:07). The suffix of the file identifies its role.



```
********* SIMULATION STARTED: Mon Sep  7 09:02:00 CEST 2015 EXP_T302-12 v150830-89ad 1509070858      [1254/1849]
   Working dir: /home/giorgi/C5/XSMLL/test  --  Issued 'make run' (running simulation) as pid=41786 ...
   Waiting to monitor cotson ...... (simpid='41861')
     9 -- BOOTED(4/4): 1 4 -- COTSON STARTED
    10 -- ALL_NODES_UP(4/4)    -- COTSON STARTED
    10 -- mypid=41786 cpid='41861' spid='' SIMVID='X51 X50 X53 X52'
    16 -- ALL_NODES_UP(4/4)    -- TIME=237.372 ms IPC ( 0.188817 )
  1040 -- FINISHED: 2 3 4 -- Received termination message
  1060 -- FINISHED_ALL_NODES -- Sending terminate message at 30374372
  1065 -- FINISHED_ALL_NODES -- Sending terminate message at 30374372
  1100 -- FINISHED_ALL_NODES -- Sending terminate message at 30374372
   simtimems=1231338   simtimeh=   0.34   maxsimmem=56160
   generated outfile: 'OUT1'
   ok1='1'  (checkernode=1)
   saving files into /home/giorgi/C5/XSMLL/test/EXP_T302-12/usgiorgi_matfx2_rr726_da1509070858_moxsmll_apmmx_si100
4_co01_wo01_tisimple+3M_hdkarmic64_l132kB+64+2+WT+false+3+l2_l2128kB+64+2+WB+true+5+bus_l34096kB+64+4+WB+true+20+m
+100+10. ...
[OK_____] simtime=    0.34 h
 dftime=   25884139503 ns    gsp=  1.00     inst=     1211478686    cpi= 21.37    speed= 21021 (kc/s)
mdftime=   25901031802 ns   tgsp=  1.00    tinst=     4834428700   tcpi= 21.43   tspeed= 21028 (kc/s)
cy=30978294978+0.6% ic=5278735363 icmr=0.00067501+0.0% dcmr=0.50138101+2.0% l2mr=0.11004236+0.0% l3mr=0.12280625+0
090+1.9% icm=890718 dcm=282805250 l2m=33839073 l3m=31538671 icrm=890718 icwm=0 icr=1319566974.50000000 icw=0.00000
81598977 dcwm=1206273 dcr=539055030 dcw=25021797 l2rm=33610904 l2wm=228169 l2r=282489695 l2w=25021797 l3rm=3150547
95 l3r=256327137 l3w=488925 xn=116684 xps=0 xpl=25 xpub=26 xsub=76 xsrb=204000002 xswb=8000002 xfrb=816 xfwb=812 x
 kcy=0 kic=0 ucy=30978294978 uic=5278735363
n1: cycles=30978294978  ic=1324108775  icmr=0.000672252 dcmr=1  l2mr=1  l3mr=1
********* SIMULATION ENDED: Mon Sep  7 09:22:33 CEST 2015  EXP_T302-12 v150830-89ad
---
* EXTRACTED: moxsmll_apmmx_si1280+10+i_no04_co01_wo01_tisimple+3M_hdkarmic64_l132kB+64+2+WT+false+3+l2_l216kB+64+2
+bus_l34096kB+64+4+WB+true+20+mem_ts1+6+3+100+10
* QUEUES(0.40h) - LOCAL(tfx2): w=7 f=3 d=1 - GLOBL(tfx1 tfx2)=4: w=7 f=7 d=159
********* SIMULATION STARTED: Mon Sep  7 09:22:54 CEST 2015 EXP_T302-12 v150830-89ad 1509070858
   Working dir: /home/giorgi/C5/XSMLL/test  --  Issued 'make run' (running simulation) as pid=26874 ...
   Waiting to monitor cotson ...... (simpid='26944')
     9 -- BOOTED(4/4): 1 2 3 -- COTSON STARTED
    10 -- ALL_NODES_UP(4/4)    -- COTSON STARTED
    10 -- mypid=26874 cpid='26944' spid='' SIMVID='X51 X50 X53 X52'
```

Figure 9: The “control interface” of MYDSE. The blue string identified the DSE point. This is a successful simulation point since a “green” OK string has been printed meaning that the output of the benchmark has been successfully compared with the reference output. Some output statistics are printed on the screen to give preliminary information.



```
********* SIMULATION STARTED: Mon Sep  7 13:11:55 CEST 2015 EXP_T302-12 v150830-89ad 1509070858
   Working dir: /home/giorgi/C5/XSMLL/test  --  Issued 'make run' (running simulation) as pid=16739 ...
   Waiting to monitor cotson ...... (simpid='16811')
     9 -- BOOTED(4/4): 1 2 4 -- COTSON STARTED
    10 -- ALL_NODES_UP(4/4)    -- COTSON STARTED
    10 -- mypid=16739 cpid='16811' spid='' SIMVID='X51 X50 X53 X52'
    16 -- SOME_NODE_DOWN(3/4) ec=0 --    what():  xsubscribe error
    91 -- SOME_NODE_DOWN(3/4) ec=0 --    what():  xsubscribe error
WARNING: mypid=16739 has been killed since it was apparently starving!
   simtimems=123704   simtimeh=   0.03   maxsimmem=
   generated outfile: 'OUT1'
WARNING: cannot locate the output masternode -- assuming node1.
   nodesfailed='1 2 3 4' (no 'All workers done, goodbye' in the output)
   ok1='0'   (no checker node)
   saving files into /home/giorgi/C5/XSMLL/test/EXP_T302-12/usgiorgi_matfx2_rr726_da1509070858_moxsmll_apmmx_si200
4_co01_wo01_tisimple+3M_hdkarmic64_l132kB+64+2+WT+false+3+l2_l216kB+64+2+WB+true+5+bus_l34096kB+64+4+WB+true+20+me
100+10. ...
[ERROR__] simtime=    0.03 h
 dftime=             0 ns    gsp=  0.00     inst=              0     cpi=  0.00    speed=    0 (kc/s)
mdftime=             0 ns   tgsp=  0.00    tinst=              0    tcpi=  0.00   tspeed=   26 (kc/s)
cy=0+0.0% ic=0 icmr=0.00000000+0.0% dcmr=0.00000000+0.0% l2mr=0.00000000+0.0% l3mr=0.00000000+0.0% simt=0+0.0% icm
2m=0 l3m=0 icrm=0 icwm=0 icr=0.00000000 icw=0.00000000 dcrm=0 dcwm=0 dcr=0 dcw=0 l2rm=0 l2wm=0 l2r=0 l2w=0 l3rm=0
=0 l3w=0 xn=0 xps=0 xpl=0 xpub=0 xsub=0 xsrb=0 xswb=0 xfrb=0 xfwb=0 xs=0 xd=0 kcy=0 kic=0 ucy=0 uic=0
********* SIMULATION ENDED: Mon Sep  7 13:14:01 CEST 2015  EXP_T302-12 v150830-89ad
---
   Killing processes '16811 ' of 'cotson'
   Killing processes '17707 17719 17731 ' of 'simnow'
   Killing processes '17639 17699 17711 17723 ' of 'Xvnc4'
   Killing processes '17191 ' of 'mediator'
* EXTRACTED: moxsmll_apmmx_si2000+10+i_no02_co01_wo01_tisimple+3M_hdkarmic64_l132kB+64+2+WT+false+3+l2_l216kB+64+2
+bus_l34096kB+64+4+WB+true+20+mem_ts1+6+3+100+10 (retry #1 - failed on tfx1)
* QUEUES(4.26h) - LOCAL(tfx2): w=2 f=13 d=4 - GLOBL(tfx2)=4: w=2 f=8 d=162
```

Figure 10: The output of a failed DSE point. A simulation may file for a number of reasons. A simulation may crash or remain stuck. If one critical situation is detected the DSE point is automatically rescheduled for another attempt wither on a different machine or at a later moment. Several processes have to be tracked (e.g., mediator, simnow, vnc cotson) and killed consequently or the simulation host may finish its resources. The user can also cancel a single simulation by pressing CTRL-C twice or kill the whole experiment by a triple CTRL-C.

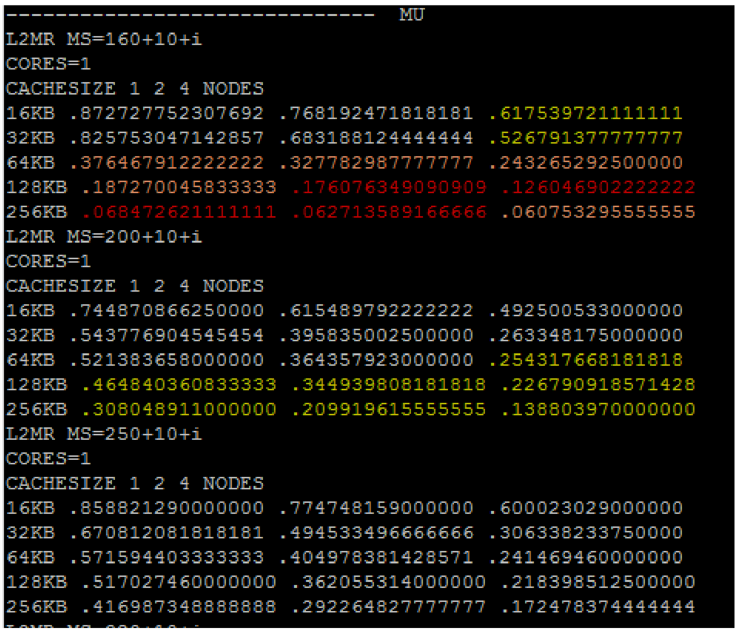
6.3 ESTRAI/RISGRAPH

Once a campaign of experiments had been conducted, the user needs to visualize the results of the experiments. The first step is to extract the data, as specified in the INFOFILE in a tabular format (Figure 11). The data can then be easily imported in a spreadsheet for further analysis.

The ESTRAI tool permits to extract such tables representing the average (calculated for each point of the table) across all the experiment performed or only on a certain subset of them (e.g., located by the date of the experiment). The ESTRAI tool permits also to analyze the Coefficient of Variation (COV) calculate as the ratio between the standard deviation s and the mean μ:

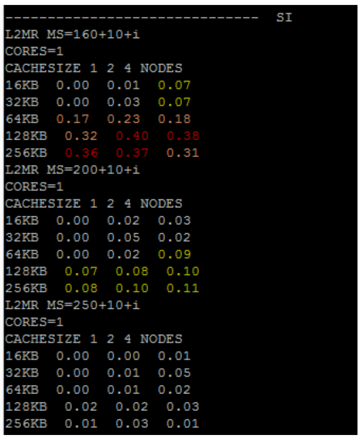
$$\text{COV}=\frac{s}{\mu} \qquad \text{where } s=\sqrt{\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{n-1}} \quad \text{and} \quad \mu=\frac{1}{n}\sum_{i=1}^{n}x_i$$

On the other side, if the set of data to be represented is already well known another tool called RISGRAPH transforms the table automatically in a graph that can be more easily interpreted (Figure 14).



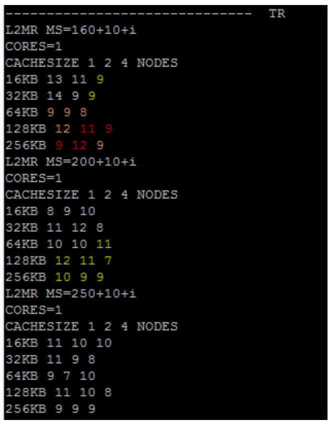
```
------------------------------  MU
L2MR MS=160+10+i
CORES=1
CACHESIZE 1 2 4 NODES
16KB .872727752307692 .768192471818181 .617539721111111
32KB .825753047142857 .683188124444444 .526791377777777
64KB .376467912222222 .327782987777777 .243265292500000
128KB .187270045833333 .176076349090909 .126046902222222
256KB .068472621111111 .062713589166666 .060753295555555
L2MR MS=200+10+i
CORES=1
CACHESIZE 1 2 4 NODES
16KB .744870866250000 .615489792222222 .492500533000000
32KB .543776904545454 .395835002500000 .263348175000000
64KB .521383658000000 .364357923000000 .254317668181818
128KB .464840360833333 .344939808181818 .226790918571428
256KB .308048911000000 .209919615555555 .138803970000000
L2MR MS=250+10+i
CORES=1
CACHESIZE 1 2 4 NODES
16KB .858821290000000 .774748159000000 .600023029000000
32KB .670812081818181 .494533496666666 .306338233750000
64KB .571594403333333 .404978381428571 .241469460000000
128KB .517027460000000 .362055314000000 .218398512500000
256KB .416987348888888 .292264827777777 .172478374444444
```

Figure 11: The output of a series of multiple experiments based on the same INFOFILE (each experiment typically consists of several simulation points). The average among several experiments is also calculated (MU). In the figure we selected to represent as output data the L2-Miss Rate (L2MR) when we vary the number of nodes and the cache size. In yellow we see the numbers with a small variance, in orange those one with a larger variance and in red those numbers with a high variance across the several experiments.



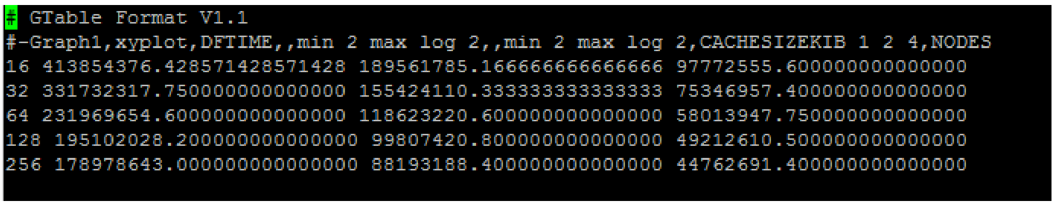
```
-----------------------------  SI
L2MR MS=160+10+i
CORES=1
CACHESIZE 1 2 4 NODES
16KB  0.00  0.01  0.07
32KB  0.00  0.03  0.07
64KB  0.17  0.23  0.18
128KB  0.32  0.40  0.38
256KB  0.36  0.37  0.31
L2MR MS=200+10+i
CORES=1
CACHESIZE 1 2 4 NODES
16KB  0.00  0.02  0.03
32KB  0.00  0.05  0.02
64KB  0.00  0.02  0.09
128KB  0.07  0.08  0.10
256KB  0.08  0.10  0.11
L2MR MS=250+10+i
CORES=1
CACHESIZE 1 2 4 NODES
16KB  0.00  0.00  0.01
32KB  0.00  0.01  0.05
64KB  0.00  0.01  0.02
128KB  0.02  0.02  0.03
256KB  0.01  0.03  0.01
```

Figure 12: The Coefficient of Variation (SI) of the L2 Miss Rate (L2MR) across several experiments.



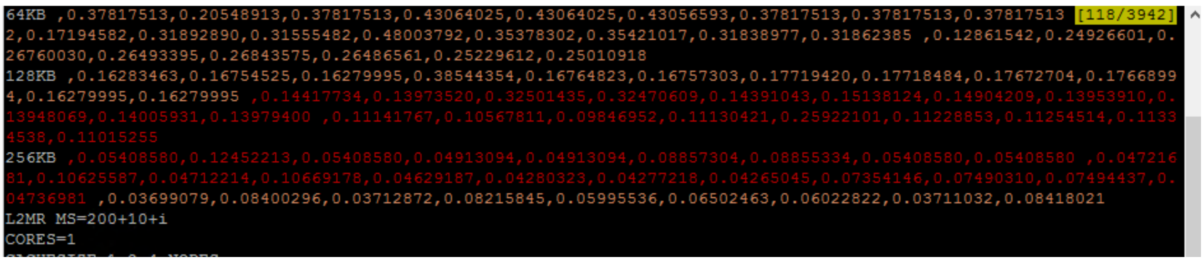
```
-----------------------------  TR
L2MR MS=160+10+i
CORES=1
CACHESIZE 1 2 4 NODES
16KB 13 11 9
32KB 14 9 9
64KB 9 9 8
128KB 12 11 9
256KB 9 12 9
L2MR MS=200+10+i
CORES=1
CACHESIZE 1 2 4 NODES
16KB 8 9 10
32KB 11 12 8
64KB 10 10 11
128KB 12 11 7
256KB 10 9 9
L2MR MS=250+10+i
CORES=1
CACHESIZE 1 2 4 NODES
16KB 11 10 10
32KB 11 9 8
64KB 9 7 10
128KB 11 10 8
256KB 9 9 9
```

Figure 13: The number of repetitions (NR) of the simulations related to a specific number in this L2 Miss Rate (L2MR) table.



```
# GTable Format V1.1
#-Graph1,xyplot,DFTIME,,min 2 max log 2,,min 2 max log 2,CACHESIZEKIB 1 2 4,NODES
16 413854376.428571428571428 189561785.166666666666666 97772555.600000000000000
32 331732317.750000000000000 155424110.333333333333333 75346957.400000000000000
64 231969654.600000000000000 118623220.600000000000000 58013947.750000000000000
128 195102028.200000000000000 99807420.800000000000000 49212610.500000000000000
256 178978643.000000000000000 88193188.400000000000000 44762691.400000000000000
```

Figure 14: The output table can be pre-formatted so that it can also be automatically processed by a next DSE visualization tool (RISGRAPH). The information for appropriate plotting can be specified in the INFOFILE associated to the experiment. The graphical output is similar to the one in Figure 20.



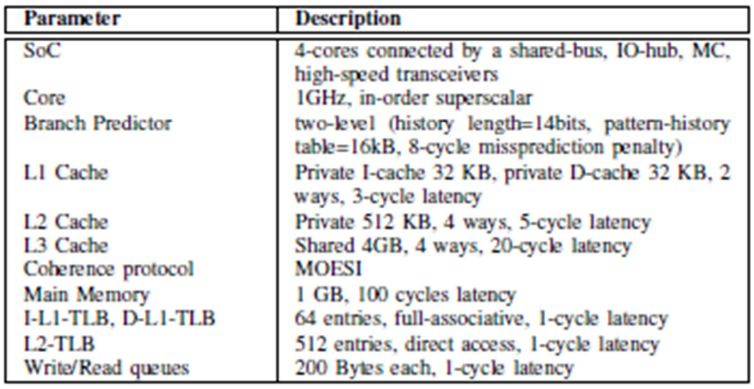
```
64KB ,0.37817513,0.20548913,0.37817513,0.43064025,0.43064025,0.43056593,0.37817513,0.37817513,0.37817513 [118/3942]
2,0.17194582,0.31892890,0.31555482,0.48003792,0.35378302,0.35421017,0.31838977,0.31862385 ,0.12861542,0.24926601,0.
26760030,0.26493395,0.26843575,0.26486561,0.25229612,0.25010918
128KB ,0.16283463,0.16754525,0.16279995,0.38544354,0.16764823,0.16757303,0.17719420,0.17718484,0.17672704,0.1766899
4,0.16279995,0.16279995 ,0.14417734,0.13973520,0.32501435,0.32470609,0.14391043,0.15138124,0.14904209,0.13953910,0.
13948069,0.14005931,0.13979400 ,0.11141767,0.10567811,0.09846952,0.11130421,0.25922101,0.11228853,0.11254514,0.1133
4538,0.11015255
256KB ,0.05408580,0.12452213,0.05408580,0.04913094,0.04913094,0.08857304,0.08855334,0.05408580,0.05408580 ,0.047216
81,0.10625587,0.04712214,0.10669178,0.04629187,0.04280323,0.04277218,0.04265045,0.07354146,0.07490310,0.07494437,0.
04736981 ,0.03699079,0.08400296,0.03712872,0.08215845,0.05995536,0.06502463,0.06022822,0.03711032,0.08418021
L2MR MS=200+10+i
CORES=1
```

Figure 15: The reasons of a large variance in the experiment can be investigated by printing all the numbers that are used to calculate the average and variance. The identifiers of the specific simulation are also printed (not shown in this figure) so that the user can investigate the input/output file to discover the reasons of a large deviation.

7 Initial experiments enabled by the AXIOM Evaluation Platform

In this Section we illustrate the main results already appearing in a recent publication [36]. In order to flexibly fit the need of designing both the hardware and the software of the AXIOM system, we used the COTSon simulator [9] [10]. COTSon can model the main AXIOM components of Figure 1. Among the important features, COTSon performs full-system simulation: the designer can run, e.g., an off-the-shelf Linux distribution and model in a decoupled way the desired functionalities and their timing behavior. This models a more realistic situation where the OS is interacting with the user programs and includes also any interrupts, exceptions, virtual memory management. In particular, the key parameters of the modeled cores are described in Table 2.

Table 2 Multicore architectural parameters.



Parameter	Description
SoC	4-cores connected by a shared-bus, IO-hub, MC, high-speed transceivers
Core	1GHz, in-order superscalar
Branch Predictor	two-level (history length=14bits, pattern-history table=16kB, 8-cycle missprediction penalty)
L1 Cache	Private I-cache 32 KB, private D-cache 32 KB, 2 ways, 3-cycle latency
L2 Cache	Private 512 KB, 4 ways, 5-cycle latency
L3 Cache	Shared 4GB, 4 ways, 20-cycle latency
Coherence protocol	MOESI
Main Memory	1 GB, 100 cycles latency
I-L1-TLB, D-L1-TLB	64 entries, full-associative, 1-cycle latency
L2-TLB	512 entries, direct access, 1-cycle latency
Write/Read queues	200 Bytes each, 1-cycle latency

Additionally, the simulator has been extended to support X-Threads [31]. This means that the simulator is also modeling the Distributed Thread Scheduler [32], which is implemented on the Programmable Logic through the block XSM (eXtended Shared Memory) of Figure 1.

As for the interconnections among SoCs, we are currently exploring several options as offered by the latest technologies. In the COTSon simulator we are performing limit-study experiments assuming that we can achieve enough bandwidth and low latency at a reasonable cost. This part is explored in detail within Deliverable D4.1 (and further explored in the next period activities).

7.1 *Matrix Multiplication Benchmark*

To illustrate the capabilities of the DSE platform, we selected the Matrix Multiplication kernel to test the performance evaluation infrastructure and to verify the feasibility of supporting X-Threads on the AXIOM platform.

The Matrix Multiplication benchmark has the following characteristics:

- Blocked matrix multiplication using the classical 3 nested loops algorithm.
- Square matrices of size n×n, where n=200,250,320,400,500,640,800.
- Block size b=10 and b=25.

Since, in this case, the number of operations is O(n3), the size n of the matrix has been chosen in such a way that the cubed size of each number of the size sequence is approximately the double of the cubed size of the previous number, i.e., 2503≈2×2003 and so on. This is useful to perform the weak scaling tests (Figure 18).

The X-Threads are generated in such a way that each thread performs the matrix multiplication of each block, therefore we can expect a number of threads equal to n/b.

7.2 Initial Experiments through the AXIOM Evaluation Platform

In order to verify the feasibility of running programs not only on the single board but also on multiple boards thanks to the adopted programming model (OmpSs [7]) and the underlying runtime system a key point is being able to schedule and execute the generated X-Threads across the boards. This implies that the memory which is local to each node (see Figure 1) has to be managed in such a way that it appears as shared to the rest of the SoCs/boards.

The XSM block of Figure 1 serves to that goal by bookkeeping the X-Threads and by appropriately moving the data where is needed.

We performed two classical tests to verify that the proposed paradigm can permit the distribution of the threads:

- Strong Scaling tests,
- Weak Scaling tests.

With the strong scaling tests, we increase the number of SoCs (for simplicity we refer to the single SoC as if it were a board) and we want to verify if the speedup t1/tN (being t1 the time to execute the program on a single SoC and tN the time to execute the program on N SoCs) is close to the ideally linear speedup (Figure 16, Figure 17, Figure 19).

With the weak scaling tests, we increase both the number of SoCs and the quantity of work to be executed, in the same proportion.

The number of operations varies as O(n^3) where n is the size of the square matrix. Therefore, we have to increase the size of the matrix by a factor $\sqrt[3]{2}$, as we increase the number of SoCs in order to perform the weak scaling tests (Figure 4). In the latter case, the ideal curve is a horizontal line with value 1, which (ideally) means that as we increase the quantity of work and the SoCs (in the same proportion) the time tN equals the time t1, i.e., the scaled systems keeps up with the increased volume of data.

As we can see in Figure 16, as the number of SoCs is increased from 1 to 2 and then 4, the scalability is good enough (close to ideal), especially for higher matrix sizes (e.g., 320). In fact, for higher matrix sizes, the number of available X-threads is also higher.

The deviation from ideal behavior is mainly due to:

- Too few X-Threads from the program;
- Increased data movement.

In fact, as the number of X-Threads is equal to n/b, in the case of n=200 we only have 5 threads to be assigned to the each of the SoCs; moreover, since our SoC has 4 cores, some of the cores may remain idle. This is visible in Figure 16 for the curve for n=200 and 4 SoCs with a drop in the scalability. We reported the strong scaling curves for some other values (200, 250, 320 and 400) to verify the sensitivity to the input data; in the tests of Figure 16 the block size is kept constant.

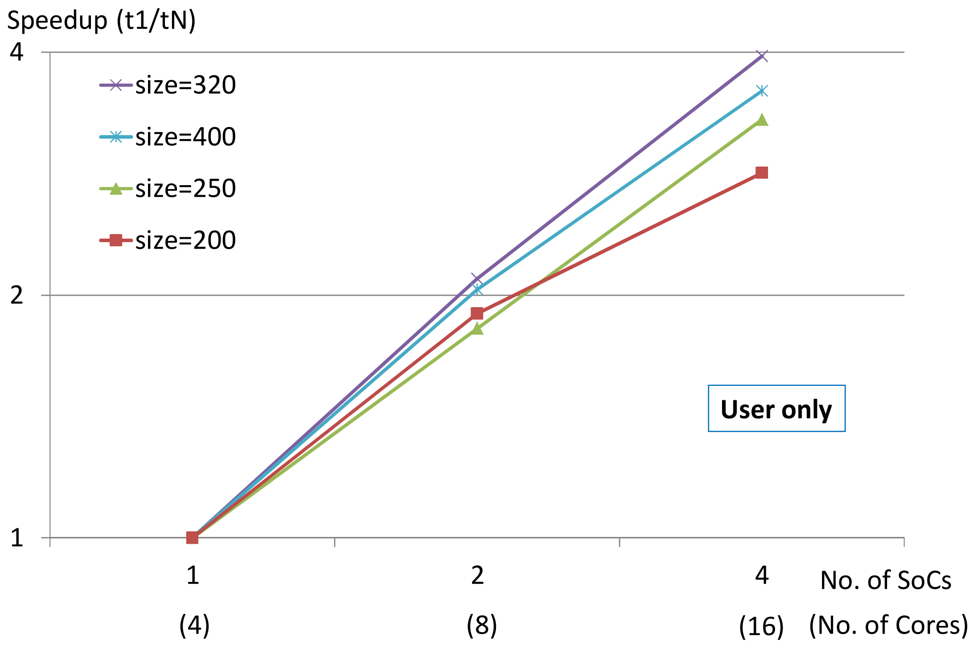


Figure 16: Strong Scaling for benchmark "Dense Matrix Multiplication" (Square Matrices). Matrix size varies: 200,250,320,400. Block size is constant and equal to 10. The time used for calculating the speedup accounts only for the User Time (without Kernel Time).

One other aspect regards the influence of the Operating System. The curves in Figure 16 reflect the execution of only the User part of the program. We extracted also the strong scaling curves that reflect both the User and the Kernel instructions (Figure 17).

As we can see, the strong scaling is affected by the OS: all the curves of Figure 16 are now compressed towards the bottom part of the Figure 17, as the time spent in Kernel mode ranges from 6% to 60% in those tests. This indicates a strong need for using a full-system simulator and not neglecting the OS activities for a proper platform design.

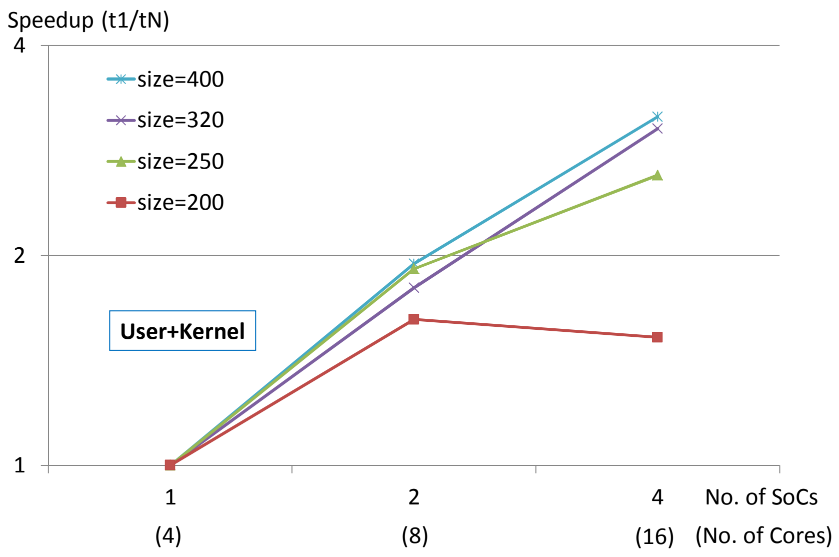


Figure 17: Strong Scaling for benchmark "Dense Matrix Multiplication" (Square Matrices). Matrix size varies: 200,250,320,400. Block size is constant and equal to 10. The time used for calculating the speedup accounts for both the User Time and the Kernel Time.

In the weak scaling tests of Figure 18, we observe that for some matrix size (n=400) we have a high efficiency, as it was for the strong scaling tests.

However, please note that in the weak scaling tests, each curve corresponding to three different matrix sizes for each of the X-axis values (the number of SoCs). For example, for the curve corresponding to n=400 the sequence of data is: 1 SoC →n=400, 2 SoCs →n=500, 4 SoCs →n=640. In particular, for 4 SoCs the efficiency drops as it has an implied matrix size of n=640 and the data set is large enough to cause a significant miss rate increase in the L2 cache (not shown in the figures).

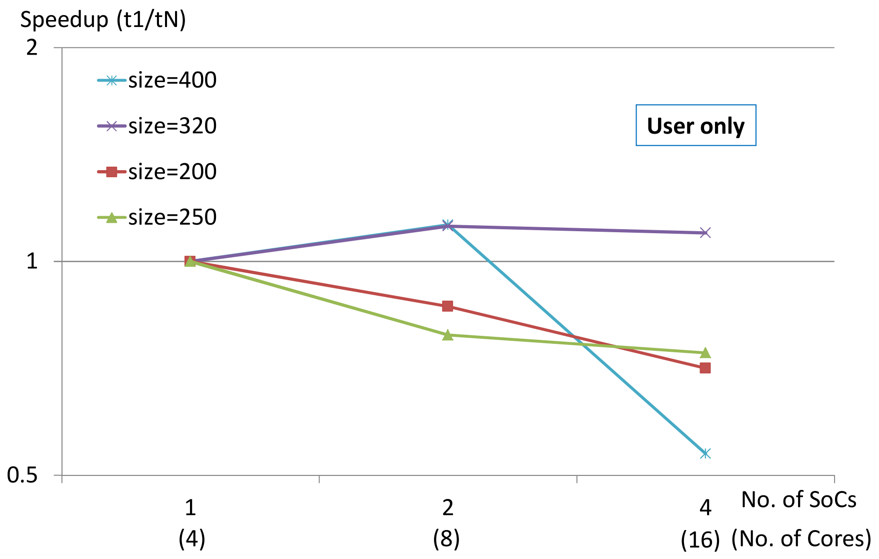


Figure 18: Weak Scaling for benchmark "Dense Matrix Multiplication" (Square Matrices). Matrix size varies: 200, 250, 320 and 400 on each single SoC (to keep the work almost constant on each core/SoC). Block size is constant and equal to 10. The time used for calculating the speedup only accounts User Time (without Kernel Time).

Finally, we explored the sensitivity to the thread granularity, by choosing a larger block size. A larger block generates longer X-threads to process such matrix block. In Figure 5, we analyzed the situation for three matrix sizes (n=200, n=400, n=800) and while the block size b is equal to 10 and 25.

As we can see, having larger threads implies fewer opportunities for parallelism, as the number of X-threads is smaller. This is particularly evident for the curve for n=200 and b=25. Moreover, there are combinations of n and b (e.g., n/b=16) where the scaling is better as the number of available threads is a multiple of the number of cores. For larger matrix sizes (e.g. 800) as noticed, the L2 cache tends to suffer more misses, thus affecting the performance.

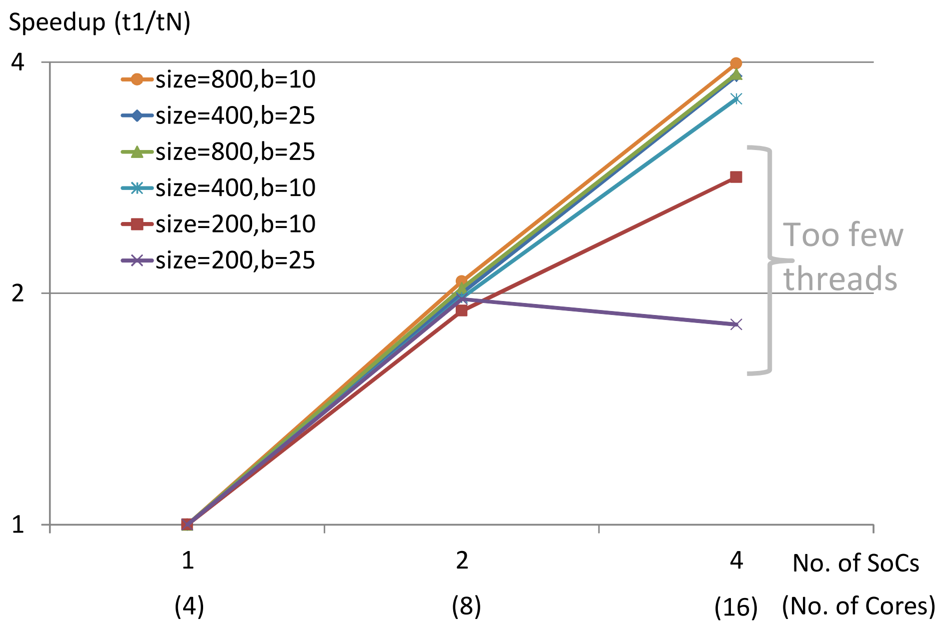


Figure 19: Effect of the thread granularity (through the block size) on strong scaling for benchmark "Dense Matrix Multiplication" (Square Matrices). Matrix size varies: 200,250,320,400. Block size assumes values 10 and 25. The time used for calculating the speedup accounts only for the User Time (without Kernel Time).

Strong and weak scaling tests are therefore useful to analyze the performance of the embedded system constituted of N SoCs. The current results show a good potential for achieving scalability across SoCs.

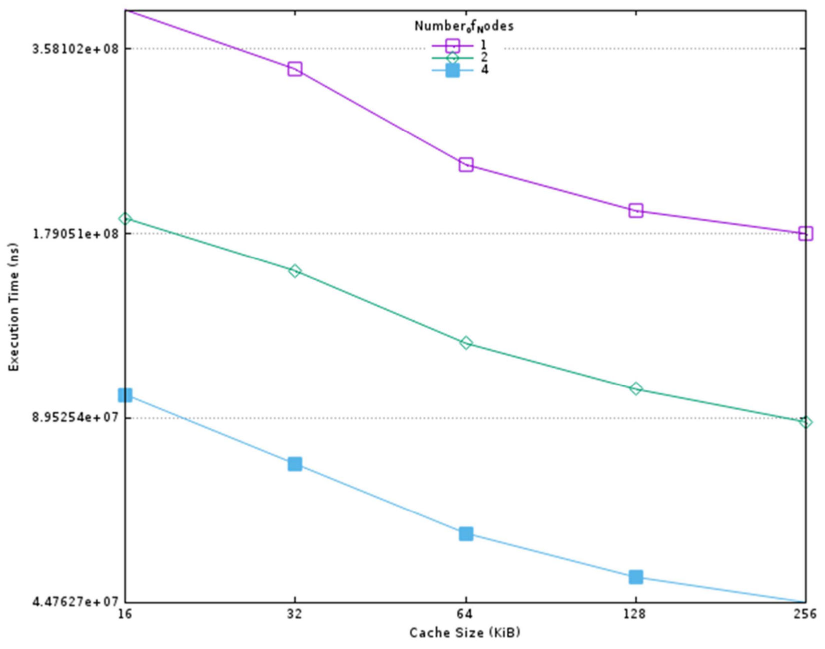


Figure 20: Execution Time dependency on L2-cache miss rate in case of 1, 2, 4 nodes.

8 Confirmation of DoA objectives and Conclusions

Here we describe how the deliverables conform to the DoA stated objectives.

PLANNED	DELIVERED
DELIVERABLE:	
• Initial AXIOM Evaluation Platform (AEP) definition and initial tests	Report

We have presented the AXIOM Evaluation Platform (AEP). Its motivation is driven by the possibility to flexibly experiment the AXIOM system by using off-the-shelf Operating Systems like Linux and the possibility of architectural (timing) and functional modeling for easier and faster prototyping. This should pave the way for FPGA prototyping, which has been already used by some of the partners for the components that were already mature for such experimentation (e.g., the AXIOM-link).

In order to permit any easy deployment we created the MYINSTALL tool that makes the installation process fast and repeatable in about 10 minutes (manually it may be subject to variations and can take a day); this is very useful since we often repeat the installation process.

In order to allow a systematic Design Exploration we created the MYDSE tool (about 200kB of shell scripting) so that we can control the simulations with little effort and we can exclude most of errors during the setup of experiments. This therefore permits a more rigorous and repeatable experimentation.

Some other tools had been prepared for a proper investigation of the results through tables and graphs.

Finally, we show some initial results for the Matrix Multiplication benchmarks: the AEP allows for investigating all details related to the execution of a single program on multiple boards.

Therefore the objectives of the first year have been meet or exceeded (e.g. with the XSMLL API specification and its implementation and with the visualization tools). Further challenges are foreseen before arriving to a possibly commercially exploitable system.

References

1. Dan Bonachea; GASNet Specification, v1.1. Report No. USB/CSD-02-1207. CS Division, EECS Department, University of California, Berkeley; October 2002; http://gasnet.lbl.gov/CSD-02-1207.pdf
2. Javier Bueno, Xavier Martorell, Rosa M. Badia, Eduard Ayguadé, Jesús Labarta; Implementing OmpSs support for regions of data in architectures with multiple address spaces. ICS 2013: 359-368 (2013).
3. Alejandro Duran, Eduard Ayguadé, Rosa M. Badia, Jesús Labarta, Luis Martinell, Xavier Martorell, Judit Planas; OmpSs: a Proposal for Programming Heterogeneous Multi-Core Architectures. Parallel Processing Letters 21(2): 173-193 (2011).
4. Message Passing Interface Forum: MPI: A Message-Passing Interface Standard, Version 3.0; September 2012; http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf
5. R. Giorgi, "Scalable Embedded Systems: Towards the Convergence of High-Performance and Embedded Computing", Proc. 13th IEEE/IFIP Int.l Conf. on Embedded and Ubiquitous Computing (EUC 2015), Oct. 2015.
6. DMA driver for AXIOM: https://git.axiom-project.eu/?p=axiom-dma
7. XSMLL API for AXIOM: https://git.axiom-project.eu/?p=XSMLL
8. OmpSs website: http://pm.bsc.es/ompss
9. COTSon website: http://cotson.sourceforge.net/
10. Argollo, E., Falcón, A., Faraboschi, P., Monchiero, M., and Ortega, D. 2009. COTSon: infrastructure for full system simulation. SIGOPS Oper. Syst. Rev. 43, 1 (Jan. 2009), 52-61
11. Palermo, G.; Silvano, C.; Zaccaria, V., "ReSPIR: A Response Surface-Based Pareto Iterative Refinement for Application-Specific Design Space Exploration," in Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on , vol.28, no.12, pp.1816-1829, Dec. 2009
12. Silvano, C.; Fornaciari, W.; Palermo, G.; Zaccaria, V.; Castro, F.; Martinez, M.; Bocchio, S.; Zafalon, R.; Avasare, P.; Vanmeerbeeck, G.; Ykman-Couvreur, C.; Wouters, M.; Kavka, C.; Onesti, L.; Turco, A.; Bondi, U.; Mariani, G.; Posadas, H.; Villar, E.; Wu, C.; Fan Dongrui; Zhang Hao; Shibin, T., "MULTICUBE: Multi-objective Design Space Exploration of Multi-core Architectures," in VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on , vol., no., pp.488-493, 5-7 July 2010
13. Giovanni Mariani, Aleksandar Brankovic, Gianluca Palermo, Jovana Jovic, Vittorio Zaccaria, and Cristina Silvano. 2010. A correlation-based design space exploration methodology for multi-processor systems-on-chip. In Proceedings of the 47th Design Automation Conference (DAC '10). ACM, New York, NY, USA, 120-125.
14. Mariani, G.; Palermo, G.; Silvano, C.; Zaccaria, V., "Multi-processor system-on-chip Design Space Exploration based on multi-level modeling techniques," in Systems, Architectures, Modeling, and Simulation, 2009. SAMOS '09. International Symposium on , vol., no., pp.118-124, 20-23 July 2009.
15. R. Giorgi, R. Badia, F. Bodin, A. Cohen, P. Evripidou, P. Faraboschi, B. Fechner, G. Gao, A. Garbade, R. Gayatri, S. Girbal, D. Goodman, B. Khan, S. Koliaï, J. Landwehr, N. Minh, F. Li, M. Lujàn, A. Mendelson, L. Morin, N. Navarro, T. Patejko, A. Pop, P. Trancoso, T. Ungerer, I. Watson, S. Weis, S. Zuckerman, M. Valero, "TERAFLUX: Harnessing dataflow in next generation teradevices ", ELSEVIER Microprocessors and Microsystems, Netherlands, Amsterdam, vol. 38, no. 8, Part B, 2014, pp. 976-990.
16. M. Solinas, M. Badia, F. Bodin, A. Cohen, P. Evripidou, P. Faraboschi, B. Fechner, G. Gao, A. Garbade, S. Girbal, D. Goodman, B. Khan, S. Koliaï, F. Li, M. Lujàn, A. Mendelson, L. Morin, N. Navarro, A. Pop, P. Trancoso, T. Ungerer, M. Valero, S. Weis, S. Zuckerman, R. Giorgi, "The TERAFLUX project: Exploiting the dataflow paradigm in next generation teradevices", IEEE Proc. 16th EUROMICRO-DSD, Santander, Spain, no. 6628287, 2013, pp. 272-279.
17. Blem, E.; Menon, J.; Sankaralingam, K., "Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures," in High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on , vol., no., pp.1-12, 23-27 Feb. 2013
18. AMD Opteron A-series: http://www.amd.com/en-us/products/server/opteron-a-series
19. AMD K12 architecture: http://partner.amd.com/Documents/MarketingDownloads/en/AMD-FAD-2015-Codename-Decoder-FINAL.pdf
20. Li, Sheng, et al. "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures." Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture. ACM, 2009.
21. Xilinx Inc., “Zynq Series.” [Online]. Available: http://www.xilinx.com/content/xilinx/en/products/silicon-devices/soc/zynq-7000.html

22. M. Banzi, *Getting Started with Arduino*. Sebastopol, CA: Make Books - Imprint of: O'Reilly Media, 2008.
23. Xilinx Inc., "Xilinx UltraScale Architecture." [Online]. Available: http://www.xilinx.com/support/documentation/white_papers/wp435-Xilinx-UltraScale.pdf
24. S. Lyberis, G. Kalokerinos, M. Lygerakis, V. Papaefstathiou, D. Tsaliagkos, M. Katevenis, D. Pnevmatikatos, and D. Nikolopoulos, "Formic: Cost-efficient and scalable prototyping of manycore architectures," in FCCM, 2012, pp. 61–64.
25. E. Palazzetti, Getting Started with UDOO, Resha Raman, Ed. Packt Publishing, 2015.
26. J. Bueno, L. Martinell, A. Duran, M. Farreras, X. Martorell, R. Badia, E. Ayguade, and J. Labarta, "Productive cluster programming with OmpSs," *Euro-Par 2011 Parallel Processing*, pp. 555–566, 2011.
27. Heirman et. Al. - ISPASS Tutorial The SNIPER multi-core simulator
28. Carl J. Mauer et al. Full system timing-first simulation. In SIGMETRICS02, pp. 108-116, June 2002'.
29. Falcon, A.; Faraboschi, P.; Ortega, D., "Combining Simulation and Virtualization through Dynamic Sampling," in Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on , vol., no., pp.72-83, 25-27 April 2007
30. R. Giorgi, "Transactional memory on a dataflow architecture for accelerating Haskell," WSEAS Trans. Computers, vol. 14, pp. 794–805, 2015.
31. R. Giorgi and P. Faraboschi, "An introduction to DF-Threads and their execution model," in IEEE MPP, Paris, France, Oct. 2014, pp. 60–65.
32. R. Giorgi and A. Scionti, "A scalable thread scheduling co-processor based on data-flow principles," ELSEVIER Future Generation Computer Systems, vol. 53, pp. 100–108, July 2015.
33. S. Weis, A. Garbade, B. Fechner, A. Mendelson, R. Giorgi, and T. Ungerer, " Architectural support for fault tolerance in a Teradevice dataflow system," Springer Int.l Journal of Parallel Programming, pp. 1–25, May 2014.
34. D. Theodoropoulos et al., "The AXIOM project (agile, extensible, fast i/o module)," in IEEE Proc. 15th Int.l Conf. on Embedded Computer Systems: Architecture, MOdeling and Simulation, July 2015.
35. R. Giorgi, "Scalable Embedded Systems: Towards the Convergence of High-Performance and Embedded Computing", Proc. 13th IEEE/IFIP Int.l Conf. on Embedded and Ubiquitous Computing (EUC 2015), Oct. 2015.
36. "Scientific data: open access to research results will boost Europe's innovation capacity." http://europa.eu/rapid/press-release IP-12-790 en.htm.
37. "EU Open Science and Open Access Policies." http://ec.europa.eu/ research/swafs/index.cfm?pg=policy&lib=science
38. R. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. Wiley, May 1991.
39. D. Lilja, "Measuring Computer Performance: A Practitioner's Guide", Cambridge Univ. Press, 2005.
40. L. Eeckout, "Computer Architecture Performance Evaluation Methods", Morgan & Claypool Publishers, 2010.
41. The JUMP Software DSM System. http://www.snrg.cs.hku.hk/srg/html/jump.htm
42. Zynq-7000 Technical Reference Manual : http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf
43. Zynq Ultrascale+ Technical Reference Manual: http://www.xilinx.com/support/documentation/user_guides/ug1085-zynq-ultrascale-trm.pdf
44. S. Wong, A. Brandon, F. Anjam, R. Seedorf, R. Giorgi, Z. Yu, N. Puzovic, S. Mckee, Magnus Sjaelander and Georgios Keramidas, "Early Results from ERA – Embedded Reconfigurable Architectures", 9th IEEE Int.l Conf. on Industrial Informatics (INDIN), Lisbon, Portugal, Jul 2011, pp. 816-822.
45. ERA Benchmark suite: http://www.dii.unisi.it/~giorgi/ebs/
46. S. Wong, L. Carro, S. Kavvadias, G. Keramidas, F. Papariello, C. Scordino, R. Giorgi, S. Kaxiras, "Embedded reconfigurable architectures", ACM Proc. 2012 international conference on Compilers, architectures and synthesis for embedded systems (CASES), New York, NY, USA, 2012, pp. 2.
47. Wong Stephan, Carro Luigi, Rutzig Mateus and Matos Debora Motta, Giorgi Roberto, Puzovic Nikola , Kaxiras Stefanos, Cintra Marcelo, Desoli Giuseppe, Gai Paolo, Mckee Sally A., Zaks Ayal, "ERA - Embedded Reconfigurable Architectures", Springer New York, ISBN:978-1-4614-0061-5, Aug 2011, pp. 239-259.

APPENDIX A -- XSMLL API draft 0.3

This is the initial specification of the XSMLL API internally published in June 2015 and made public with this document as of 10th February 2016; the API is still under development and can therefore be updated during this project.

```
=================================================================================
X S M - L L
=================================================================================
The eXtended Shared Memory (XSM) Low Level Specification (draft 0.3)
Roberto Giorgi - University of Siena, Italy - 01 February 2015
=================================================================================
$Id: README01-xsmll 19 2015-06-20 14:00:32Z giorgi $
DISCLAIMER: this specification is WORK IN PROGRESS -- positive comments are welcome

---------------------------------------------------------------------------------
INTRODUCTION (DO NOT SKIP)
---------------------------------------------------------------------------------

XSM DEFINITIONS
---------------
    X-thread
        A function that expects no parameters and returns no parameters.
        The body of this function can refer to any memory location for which
        it has got the pointer through XSM function calls (e.g., xpreload, xpoststor,
        xsubscribe, ...). An X-thread is identified by an object of type xtid_t
        (X-thread identifier). In other words:
        typedef void (*xthread_t)(void)

    xtid_t
        A type that combines two objects: a thread identifier (a number from 1
        through a XSM_MAX_THREAD_ID) and an offset inside its input frame.
        Given an 'xtid_t tid' the thread number is obtained by the _X_TID(tid) macro,
        while the frame offset is obtained by the _X_OFF(tid) macro.

    INPUT_FRAME
        A buffer which is allocated in the local memory and contains the input
        values for the current X-thread.

    OUTPUT_FRAME
        A buffer which is allocated in the local memory and contains values
        to be used by other X-threads (consumer X-threads)

    SYNCHRONIZATION_COUNT
        A number which is initially set to the number of input values (or events)
        needed by an X-thread. The SYNCHRONIZATION_COUNT has to be decremented each
        time the expected data is written in an OUTPUT_FRAME.

    GUEST (SPACE)
        The simulated space.
        This the environment that is simulated inside a virtual machine (as a
        full-system simulation). This environment does not necessarily know whether
        is running as a virtual machine or as a real system.

    HOST (SPACE)
        The simulation space.
        This environment models both the behavior, the timing, the power
        (or anything else) that is related to the machine that has to be modeled.
    XSMU
        A hardware/software unit which manages the XSM interface at low level.


=================================================================================
C level API for XSM-LL - a set of C callable functions to manage XSM and X-threads
=================================================================================

XSM BASIC FUNCTIONS
-------------------
    xtid_t xschedulez(xthread_t ip, uint32_t sc, uint32_t sz)
        Schedules an X-thread whose name (instruction pointer) is specified by 'ip'.
        The thread expects sc inputs which are stored in its INPUT_FRAME of size 'sz'
        (in bytes).
        Returns an X-thread identifier (with 0 internal offset -- see above
        definition).

    void xdestroy()
        Called at the end of an X-thread to signal that any allocated resource
        belonging to the current X-thread can be freed up.

    void* xsubscribe(
        xtid_t tloc, uint32_t regionstart, uint32_t regionsize, uint8_t type)
        Subscribes a memory region that is going to be shared by several X-threads
        with some given Read/Write protection specified by 'type':
          type = 0x01 -- read-only access
          type = 0x02 -- write-only access
          type = 0x03 -- read-write access
          type = 0x00,0x0x-0xFF (reserved values)
        The region is specified by its offset ('regionstart') inside a preallocated
        shared heap and its size ('regionsize') in bytes.
        Both regionstart and regionsize are aligned to 8 bytes (64 bits).
        The pointer to the region is both returned by this function and stored in
        the tloc (thread-location) indicated (see the examples below). Please note
        that only the specified thread will have write access (if the protection
        bits specified so), all other thread will get an error if they accidently
        try to read that region once it is under "write protection". Read is
        possible if the read-only has been specified.
        Read-write modality is used to implement transactional memory consistency.

    void xpublish(void* regptr)
        Publishes the modifications to a previously subscribed region pointed by
        regptr. The whole subscribed region is published. To publish an arbitrary
        portion use: xpublish_range.

    uint64_t xtmbegin(uint64_t s1, uint64_t s2)
        Begins a transactions on the TM region pointed by s1 and having a lenght s2.
        Returns 1 if the transaction can start or 0 otherwise.

    uint64_t xtmend(uint64_t s1, uint64_t s2)
        Ends a transactions on the TM region pointed by s1 and having a lenght s2.

    xwrite
        TBD
    xread
        TBD
    xsend
        TBD
    xrecv
        TBD


XSM AUXILIARY FUNCTIONS
-----------------------
    void* xpreload()
        Called at the beginning of an X-thread to get the INPUT_FRAME pointer

    void* xpoststor(xtid_t tid)
        Called when an X-thread needs to write output values for a consumer X-thread
        tid (which is probably scheduled right before).

    void xdecrease(xtid_t tid, int n)
        This is used to the decrease the SYNCHRONIZATION_COUNT by n. Typically this
        is issued after each write operation or - cumlatively - after n operations
        (n is always not greater than the inital SYNCHRONIZATION_COUNT)


XSM EXTRA FUNCTIONS
-------------------
    void xconstrain(xtid_t tid, uint64_t nmask)
        Binds the execution of the X-thread tid to the nodes specified by the
        node-mask nmask (limited to XSM_MAX_NODE_MASK nodes, bit0 for node-0, bit1
        for node-1, ...)

    xtid_t xschedule64(xthread_t ip, uint32_t sc)
        Variant of the xschedulez functioni with fixed size (64 bits) inputs.
        Schedules an X-thread whose name (instruction pointer) is specified by ip.
        The thread expects sc inputs of size 64 bits which are stored in its
        INPUT_FRAME.
        Returns an X-thread identifier (with 0 internal offset -- see above
        definition).

    void xpublish_range(void* regptr, size_t sz)
        Publishes the modifications to a previously subscribed region.
        This can be a subregion pointed by regptr with size sz.

    void xacquire(void* regptr)
        Reloads the subscribed region identified by regionprt

    void xacquire_range(void* regptr,size_t sz)
        Reloads the subscribed region identified by regionprt

    void* xalloc(uint64_t tid, uint32_t sz, uint64_t type)
        Allocates a region of size sz bytes with a given type to be used by the
        X-thread indicated by tid. This is a simplified version of the xsubscribe,
        where:
          i) the user does not need to specify where the region is positioned inside
             the preallocated shared heap.
         ii) its up to the user to write the returned pointer into the OUTPUT_FRAME
             belonging to the tid X-thread

    void xfree(uint64_t regprt)
        Frees a region identified by the region pointer regptr

    void xprotect(uint64_t regptr, uint64_t type)
        Changes the protection bits (type -- meaning as in xsubscribe) of the region
        identified by regptr.


XSM PERFORMANCE COUNTERS FUNCTIONS
----------------------------------
    void xzonestart(uint8_t zone)
        Starts metrics collection for the specified zone of the code.

    void xzonestop(uint8_t zone)
        Stops metrics collection for the specified zone of the code.

    void xtimestamp()
        Clears the internal timestamp counters for the metrics/statistics related to
        XSM (these are metrics derived from the HOST metrics/statistics, which are
        not modified by this function, they are only read)


XRT FUNCTIONS (XSM RUN-TIME FUNCTIONS)
--------------------------------------
    IMPORTANT NOTE
    --------------
        The XRT is currently enabled as a wrapper of the C main function.
        The 'int __wrap_main(int argc, char **argv)' calls the
        'int __real_main(int argc, char **argv)' function which maps to the
        application main thanks to the appropriate link-time option (see Makefile).

    void xsm_set_nopf(void* nopf_ptr)
        Sets the run-time 'nop' functions that are called in case of no work to be
        done.

    void xsm_set_worker_stack(void* tstack_ptr)
        Sets the worker stack that is used to allocate a local INPUT_FRAME or
        OUTPUT_FRAME.

    void xsm_inject(uint64_t val)
        Diagnostic/Instrumentation function that can be used to pass a value 'var;'
        from the GUEST environment to the HOST environment.

    void xsm_reset(void* owmptr, uint32_t owmsz, uint32_t stacksz)
        Loads the pointers for several types of memory regions:
           - subscribable regions (base pointer: owmptr, size: owmsz)
           - default size of worker stack (used from frames): stacksz
        Then sends and initial reset to the HOST runtime.

    void xsm_setbuf0(void *buf)
        Sets the internal (HOST) XSM buffer #0 for metrics/statistics to the GUEST
        buffer 'buf'

    void xsm_setbuf1(void *buf)
        Sets the internal (HOST) XSM buffer #1 for metrics/statistics to the GUEST
        buffer 'buf'

    void xsm_exit(void)
        Ends the execution on the node which calls this function.

    void xsm_printstats(void)
        Prints the content of internal metrics/statistics counters on the GUEST
        stdout.


XSM GUEST ENVIRONMENT VARIABLES
-------------------------------
    XSM_NWORKERS
        Number of XSM workers per node.  Default is the number of cores.
        It can be set as the number of cores or a slightly larger value.
        The runtime ties each worker to each core in round robin.

    XSM_NODEID
        Numeric identifier of the execution node, starting from 1.  Default 1.

    XSM_NNODES
        Number of executions nodes.  Default 1.

    XSM_OWMSZ
        Size of the subscribable memory in bytes.  Default XSM_DEFAULT_OWM_SIZE
```

APPENDIX B -- XSMLL coding example

The following examples are intended to show the coding style that could be achieved through source-to-source compilation from higher level programming model. XSMLL is an execution model NOT a programming model.

Sample code for fibx.c using the XSMLL API

```
/*$Id: fibx.c 16 2015-05-22 09:12:04Z giorgi $*/
#include "xsm.h"
#include <stdio.h>

#define DEFAULTN  20
#define THRESHOLD 10
#define RECFIB

// reference version
inline uint64_t serialfib(int n)
{
#ifdef RECFIB
    return n < 2 ? n : serialfib(n-1) + serialfib(n-2);
#else
    uint64_t a=0,b=1;
    int i=0;
    for(; i < n; ++i) {
        uint64_t t = a + b;
        a = b;
        b = t;
    }
    return a;
#endif
}

typedef struct { xtid_t tloc; uint64_t n1; uint64_t n2; } adder_s;
void adder(void)
{
    const adder_s* fp=xpreload();
    xtid_t xloc = fp->tloc;
    uint64_t *xp = xpoststor(xloc);
    xp[_X_OFF(xloc)] = fp->n1 + fp->n2;
    xdecrease(xloc,1);
    xdestroy();
}

typedef struct { xtid_t tloc; uint64_t n; uint64_t t; } fib_s;
void fib(void)
{
    const fib_s* fp=xpreload();
    uint64_t n  = fp->n; // receive n
    uint64_t t  = fp->t; // receive t (threshold)
    xtid_t xloc = fp->tloc; // target location

    if (n < t) {
        uint64_t *tp = xpoststor(xloc);
        tp[_X_OFF(xloc)] = serialfib(n);
        xdecrease(xloc,1);
    }
    else {
        xtid_t xadd = xschedule64(&adder,3); // spawn adder
        adder_s* tadd = xpoststor(xadd);
        tadd->tloc = xloc;  // add.dst is this.dst
        xdecrease(xadd,1);

        xtid_t xfib1 = xschedule64(&fib,3);  // spawn fib1
        fib_s* tfib1 = xpoststor(xfib1);
        tfib1->tloc  = _XREF(xadd,1); // fib1.tloc is add[1]
        tfib1->n     = n-1;           // send fib1, n-1
        tfib1->t     = t;             // send fib1, threshold
        xdecrease(xfib1,3);

        xtid_t xfib2 = xschedule64(&fib,3);  // spawn fib2
        fib_s* tfib2 = xpoststor(xfib2);
        tfib2->tloc  = _XREF(xadd,2); // fib2.tloc is add[2]
        tfib2->n     = n-2;           // send fib2, n-2
        tfib2->t     = t;             // send fib2, threshold
        xdecrease(xfib2,3);
    }
    xdestroy();
}

```

```

uint64_t nn=0; // input, for checking purposes
uint64_t th=0; // input, for checking purposes

typedef struct { uint64_t res; } report_s;
void report(void)
{
    const report_s* fp=xpreload();
    uint64_t res=fp->res;
    xdestroy();
    printf("++report\n");
    printf("xsm fib= %lu\n",res);

    uint64_t ser_res = serialfib(nn);

    printf(res==ser_res?"*** SUCCESS\n":"***FAILURE\n");
//    xdestroy();
}

int main(int argc, char **argv)
{
    nn = DEFAULTN;
    th = THRESHOLD;
    if (argc > 1)
        nn = atoi(argv[1]);
    if (argc > 2)
        th = atoi(argv[2]);
    printf("computing fibonacci(%lu)\n",nn);

    xtid_t xrep = xschedule64(&report,1); // spawn reporter
    xconstrain(xrep,0x1); // constrain execution on node 1
    xtid_t xfib = xschedule64(&fib, 3);  // spawn fib
    fib_s* tfib = (fib_s*)xpoststor(xfib);
    tfib->tloc = _XREF(xrep,0); // edge: fib.out -> report[0]
    tfib->n    = nn;
    tfib->t    = th;
    xdecrease(xfib,3);
    return 0;
}
```

Sample code for mmx.c using the XSMLL API

```
#include "xsm.h"
#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include <math.h>
/*$Id: mmx.c 20 2015-06-21 21:28:11Z giorgi $*/

//#define __DEBUG
//#define N 512
//#define BLOCKSZ 8
//#define XDATA_DOUBLE
#define N 200
#define BLOCKSZ 25
//#define XDATA_SINGLE
#define XDATA_INT64
#include "xdebug.h"

#ifdef XDATA_SINGLE
    #define DATA float
#endif
#ifdef XDATA_DOUBLE
    #define DATA double
#endif
#ifdef XDATA_INT64
    #define DATA uint64_t
#endif
#define SIZE(M,N) (N*M*sizeof(DATA))

#define RB_SZ SIZE(N,N)
#define RA_SZ SIZE(N,BLOCKSZ)
#define RC_SZ SIZE(N,BLOCKSZ)

#define RC_OFF(j) (            SIZE(N,j))
#define RB_OFF(j) (  SIZE(N,N)+SIZE(N,j))
#define RA_OFF(j) (2*SIZE(N,N)+SIZE(N,j))

// xreport = finish barrier tid
typedef struct  { DATA *A; DATA *B; DATA *C; xtid_t xreport; } bmmul_s;
```

```
void bmmul()
{
    const bmmul_s *cfp = xpreload();
    const DATA *A = cfp->A;
    const DATA *B = cfp->B;
    DATA *C = cfp->C;
    int i, j, k;

    printf("Working at C address: 0x%lx\n",(uint64_t)C);
    fflush(stdout);
    for (j = 0; j < BLOCKSZ; j++) {
        for (i = 0; i < N; i++) {
            DATA t = 0;
            for (k = 0; k < N; k++) {
                t += A[j * N + k] * B[k * N + i];
            }
            C[i + j * N] = t;
        }
    }
    xpublish(C);
    xdecrease(cfp->xreport,1); // decrement the barrier count
    xdestroy();
}

typedef struct { uint64_t *scsp; DATA *C; } report_s;
void report()
{
    const report_s *cfp = xpreload();
    const DATA *C = cfp->C;
    const uint64_t *scsp = cfp->scsp;
    int i, j;
    int ok = 1;
    const uint64_t superchecksum = *scsp;
    xzonestop(1);
    xdestroy(); // XSM computations are finished at this point

    print_matrix(C,N,N);

    // Verify the SuperCheckSum
    uint64_t xchecksum = 0L;
    for (i = 0; i < N; i++) {   // i = row pointer
        for (j = 0; j < N; j++) { // j = column pointer
            xchecksum = (xchecksum + (int)(C[i*N + j])) &0xFF;
        }
    }
    if (superchecksum==xchecksum) ok = 1; else ok = 0;
    printf("CHECKSUM=%lu vs %lu  OK=%d\n", xchecksum, superchecksum, ok); fflush(stdout);
    printf("\n*** %s ***\n", ok ? "SUCCESS" : "FAILURE");fflush(stdout);
}

typedef struct { xtid_t xr; } owm_matrix_mul_s;
void owm_matrix_mul() /* OWM version. Compute C=A*B.  */
{
    const owm_matrix_mul_s *fp = xpreload();
    xtid_t xr = fp->xr;

    xtimestamp(); //resets timestamp statistics so we count time from here
    int j,nb;

    for (j=nb=0; j<N; j+=BLOCKSZ,++nb) {

        xtid_t bm = xschedulez(&bmmul, 4, sizeof(bmmul_s));
        /* Region B (size N*N)       stores the entire matrix B */
        /* Region A (size N*BLOCKSZ) stores the block of matrix A  */
        /* Region C (size N*BLOCKSZ) stores the computed result.  */
        xsubscribe(XDST(bm, bmmul, A), RA_OFF(j), RA_SZ, _OWM_MODE_R);
        xsubscribe(XDST(bm, bmmul, B), RB_OFF(0), RB_SZ, _OWM_MODE_R);
        xsubscribe(XDST(bm, bmmul, C), RC_OFF(j), RC_SZ, _OWM_MODE_W);

        // finish barrier
        bmmul_s* bm_fp = xpoststor(bm);
        bm_fp->xreport = xr;
        xdecrease(bm,4);
    }
    xdestroy();
}

typedef struct { DATA *A; DATA *B; DATA *C; uint64_t *scs; xtid_t xm; } fill_matrix_s;
void fill_matrix()
{
    const fill_matrix_s *cfp = xpreload();
    DATA *A = cfp->A;
    DATA *B = cfp->B;
    DATA *C = cfp->C;
    uint64_t *scs = cfp->scs;
    xtid_t xm = cfp->xm;
    int i, j, cr, cc;
    uint64_t superchecksum;
    srand(12345); // start always with same numbers in the matrices (optional)
    //C=0
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            C[i * N + j] = 0;
        }
    }
    //B
    for (i = 0; i < N; i++) {
        cr = 0;
        for (j = 0; j < N-1; j++) {
            int val1 = rand()&0xFF;
            cr = (cr + val1) &0xFF;
            B[i*N+j] = (DATA)val1;
        }
        B[i*N+N-1] = (DATA)cr;
    }
    //A
    for (j = 0; j < N; j++) {
        cc = 0;
        for (i = 0; i < N-1; i++) {
            int val2 = rand()&0xFF;
            cc = (cc + val2) &0xFF;
            A[i*N+j] = (DATA)val2;
         }
         A[(N-1)*N+j] = (DATA)cc;
    }
    // Calculate the SuperCheckSum
    superchecksum = 0;
    for (i = 0; i < N; i++) {
        superchecksum =
           (superchecksum + (uint64_t)(A[(N-1)*N+i]) * (uint64_t)(B[i*N+N-1]))&0xFF;
    }
    *scs = (superchecksum<<2)&0xFF;

    // Publish the Filled Matrices and the expected SuperCheckSum
    xpublish(A); xpublish(B); xpublish(C); xpublish(scs);

    // Start the multiply
    xdecrease(xm,1);
    xdestroy();
}

int main(int argc, char **argv)
{
    xzonestart(1);
    xtid_t xf = xschedulez(&fill_matrix, 5, sizeof(fill_matrix_s));
    xtid_t xr = xschedulez(&report, N / BLOCKSZ, sizeof(report_s));
    xtid_t xm = xschedulez(&owm_matrix_mul, 1, sizeof(owm_matrix_mul_s));
    xconstrain(xr,1); // force on node 1
    xconstrain(xm,1); // force on node 1
    xconstrain(xf,1); // force on node 1

    xsubscribe(XDST(xf, fill_matrix, A), RA_OFF(0), SIZE(N, N), _OWM_MODE_W);
    xsubscribe(XDST(xf, fill_matrix, B), RB_OFF(0), SIZE(N, N), _OWM_MODE_W);
    xsubscribe(XDST(xf, fill_matrix, C), RC_OFF(0), SIZE(N, N), _OWM_MODE_W);
    xsubscribe(XDST(xf, fill_matrix, scs), RA_OFF(N), sizeof(uint64_t), _OWM_MODE_W);
    fill_matrix_s *xf_fp = xpoststor(xf);
    xf_fp->xm = xm;

    owm_matrix_mul_s *xm_fp = xpoststor(xm);
    xm_fp->xr = xr;

    // Schedule final check
    xsubscribe(XDST(xr, report, scsp), RA_OFF(N), sizeof(uint64_t), _OWM_MODE_R);
    xsubscribe(XDST(xr, report, C), RC_OFF(0), SIZE(N, N), _OWM_MODE_R);

    xdecrease(xf, 5);
    return 0;
}
```