

# JubiTool: Unified design flow for the Perplexus SIMD hardware accelerator

O. Brousse, J. Guillot, T. Gil, F. Grize, G. Sassatelli, J. M. Moreno, J. Madrenas, A. Villa, H. Volken and M. Robert

**Abstract**—This paper presents a new unified design flow developed within the Perplexus project that aims to accelerate parallelizable data-intensive applications in the context of ubiquitous computing. This contribution relies on the JubiTool: a set of integrated tools (JubiSplitter, JubiCompiler, UbiAssembler), allowing respectively to extract, compile and assemble parallelizable parts of applications described in Jubi language. Jubi is a modified Java agent based language (JADE) dedicated to the Ubichip (the bio-inspired chip developed within the confines of the Perplexus project). By appending hardware directives to a software agent description, the inherent flexibility of software is combined with the runtime performance of a hardware execution. In the case of typical Perplexus applications such as the Spiking Neural Network Simulator, this contribution takes profit of the intrinsic property of the Ubichip in terms of parallelism resulting in an expected speedup of at least one order of magnitude. Finally, this hybrid (SW/HW) flow could be easily modified and adapted to support other kind of distributed platforms.

## I. INTRODUCTION

The PERPLEXUS project [1] aims to develop a scalable and distributed platform dedicated to the simulation of large-scale phenomena, and to the observation of emerging behaviors. The Perplexus platform is composed of a cluster of communicating nodes called UBIDULES (UBIquitous moDULES). These nodes are built around a XScale microprocessor (ARM based) and a specific reconfigurable integrated circuit (Ubichip presented in [2] and [3]) that both provide support to distributed bio-inspired mechanisms. Furthermore, this specific chip has two operating modes, native (FPGA-like) and multiprocessor. In this paper, we exclusively focus on this last operating mode and use it to accelerate parts of Perplexus applications. In this configuration the Ubichip is composed of multiple Processing Elements (PE) scheduled by a single sequencer and therefore can be viewed as a hardware SIMD (Single Instruction Multiple Data) coprocessor for the Ubidule main processor. Fig.1 gives an overview of the platform which is in the form of a network of Ubidules. Each Ubidule is in turn represented as an unit made of a hardware and a software partition. An embedded Linux operating system runs on top of the

microprocessors and handles most high-level functionality like MANET (Mobile Ad-hoc NETWORK) and Agent Oriented Programming environment.

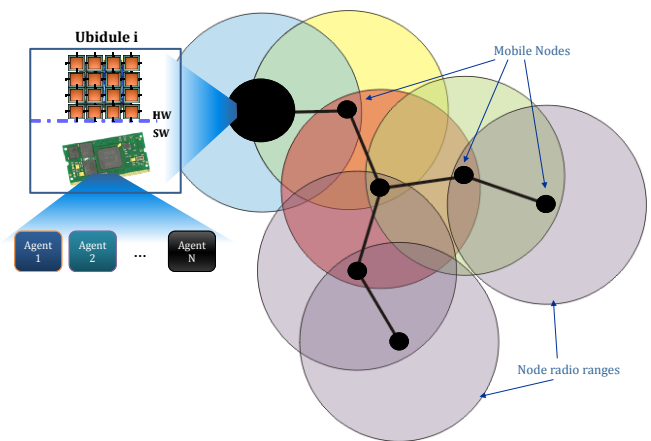


Figure 1. Perplexus platform overview.

Among the many challenges that form this project, this paper puts focus on the work realized at the Ubidule level to enable fast design of hardware accelerated applications. To this purpose we present an agent oriented design flow that allows taking advantage of the parallel processing capabilities of the Ubichip thanks to a dedicated SIMD compiler. The proposed approach is based on a unified description of applications. This accelerates and eases developers work while providing 2 implementations opportunities; software for debugging and reference purposes and functionally equivalent hardware accelerated.

To provide this solution we chose to introduce a unified HW/SW language called Jubi as well as the set of compiling tools used for the Perplexus platform. This approach could easily be extended to other parallel architectures.

This paper is organized as follows section II introduces the Jubi language as an efficient way to describe intrinsically parallel applications to be hardware accelerated on the Ubichip of the Perplexus platform. Then section III presents the compilation flow and tools we propose to exploit this new language in conjunction with the Perplexus platform. Then section IV presents two Perplexus applications that take advantage of the developed tool called JubiTool. Section V concludes on this original contribution and finally opens perspectives for further investigations.

O. Brousse, J. Guillot, T. Gil, G. Sassatelli and M. Robert are with Microelectronic Department of the LIRMM UMR 5506 CNRS, univ. Montpellier 2, France. E-mail: lastname@lirmm.fr. F. Grize is with ISI, Univ. of Lausanne, Switzerland. E-mail: Francois.Grize@unil.ch. J. M. Moreno and J. Madrenas are with AHA-DEE at Univ. Politecnica de Catalunya, Spain. E-mail: lastname@aha-dee.upc.edu. A. Villa is with the Institut de Physiologie at the Univ. Joseph Fourier, France. E-mail: Alessandro.Villa@ujf-grenoble.fr. H. Volken is with IMA at the Univ. of Lausanne, Switzerland. E-mail: Henri.Volken@unil.ch

## II. AUGMENTED JAVA FOR HW/SW UBIQUITOUS COMPUTING

The software layer of the Perplexus Project we introduced in [4], [5] and [6] is based on the well known Java Agent DEvelopment framework JADE [7]. As a direct consequence of this, all users applications are programmed in Java [8] and described according to the *agent* concept. This language is increasingly used and perfectly suits our portability and flexibility needs. Some Java hardware machines have been under studies since the last decades, for the sake of examples we can cite the following works [9],[10], [11] or Sun PicoJava[12]. None of them provide support for parallelism as the original language was not intended to this. Some software parallel classes like JPCL [13] adds software parallelism but even in this case Java Virtual Machine (JVM) are running on one processor and requires a framework to link several JVM running on different targets with a shared global object space to take advantage of those classes. Manta [14] solution relies on compiling Java threads to native x86 assembly code spanned on x86 based platform making the whole application to become hardware dependent.

The fundamental concept behind this Jubi approach relies on the use of directives for flagging parallel sections in a hardware independent language. Agent coded in Jubi can then be executed in SW mode (in such case directives are ignored) or hybrid SW/HW in which case flagged sections are compiled for parallel SIMD execution.

The J2ME standard has been chosen as the base for our Jubi. This comes from the XScale limited resources (and so forth Java virtual machine). As the J2ME standard is a light version of Java 1.4 standard and so supports the JADE light version called LEAP, it perfectly suits our needs in term of framework requirements.

An accelerate section of code presented as a component can be described with its inputs, outputs and internal behavior. Adding this approach to Java requires to set firstly *in* and *out* keywords that will be considered respectively as input and output flags for inputs and outputs *pins* of the component. A *NPE* keyword allows the user to specify the number of processing elements (PE) that will be used for this application. This *NPE* number must be defined as a constant.

The following code gives an example of the transformations:

```
final int NPE = 5;
int a[NPE], b[NPE];
int c[NPE];
```

becomes:

```
final int NPE = 5;
in int a[NPE], b[NPE];
out int c[NPE];
```

Then to describe the behavior of the hardware block we define the `#jubi` keyword that flags the code to be accelerated using the SIMD hardware. The following code gives an example:

```
#jubi
{
    c [NPE]= a[NPE] + b[NPE];
}
```

Finally to enable the parallelization of software sequential loops in the hardware accelerated mode we introduce the “`parallelfor`” keyword. This keyword allows both software and hardware generating implementations from the same unified description.

```
#jubi
parallelfor(int i=0; i<NPE; i++)
{
    c [i]= a[i] + b[i];
}
```

The hardware versus software time chart of the code above is shown in Fig.2, and gives a qualitative idea of the performance improvement that can be obtained using the hardware accelerator.

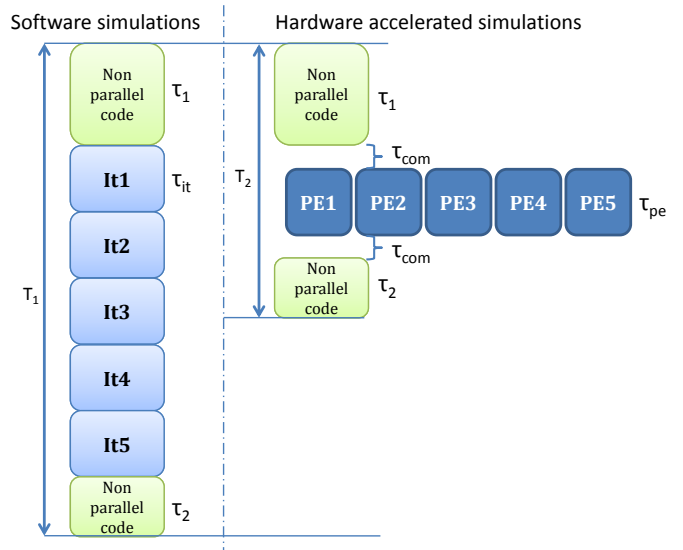


Figure 2. Compared execution time for software (sequential) and hardware (parallel) loops.

We should mention that in this current version of the tool only declarations, assignments and integer arithmetic or logic (Boolean) expressions are supported inside the `#jubi` blocks. This limitation avoid the use of class and pre-defined functions inside the hardware block.

## III. PERPLEXUS COMPILATION FLOW

### A. Flow

Fig.3 presents the compilation flow we propose to allow fast SW and hybrid HW/SW applications development. In this figure the software compilation flow appears on the left side and the hardware compilation flow on the right side.

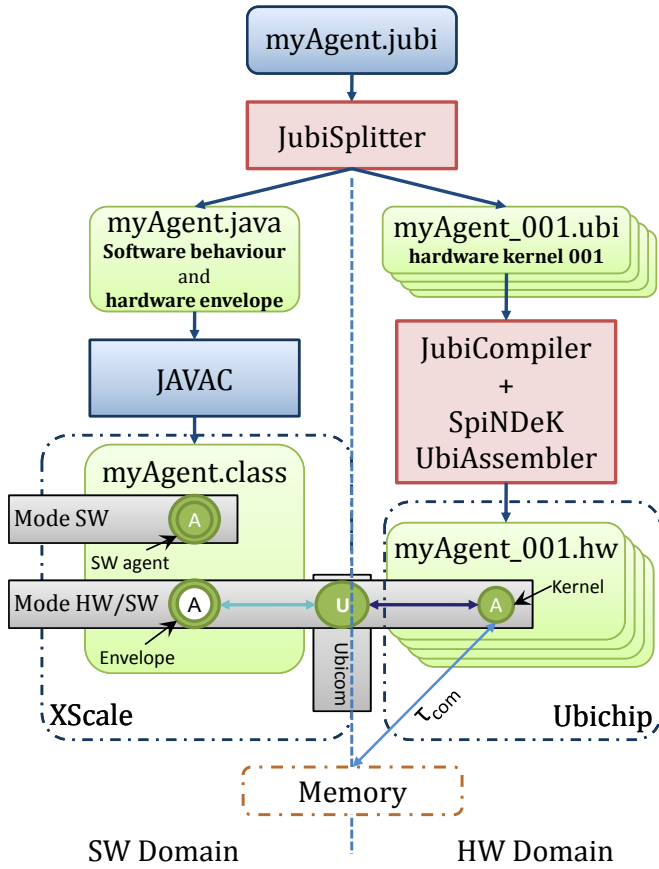


Figure 3. One file for all Compilation Flow

Software applications execution only use software flow whereas hardware accelerated applications require both sides to compile accelerated agents software part (named envelope) and hardware parts (named kernels). The agent envelope is actually a part of the Java file that triggers the hardware execution and feeds hardware kernels with appropriate data. This is done through the Ubicom agent that acts as a wrapper between sequential and parallel sections (i.e. software and hardware parts) of the application code.

As presented in Fig.3 the processing of the Jubi file results in the creation of two distinct file types. Firstly, one Java file that offers the possibility to start the application either in software mode or in hardware accelerated mode. These two options are of interest to benchmark an application speedup between software and hardware executions. The section III-B gives more details on this and the JubiSplitter. Secondly, for each #jubi block described in the Jubi file one “ubi” file is created. Every “ubi” file encapsulates the code to be accelerated. Sections III-C and III-D give more details on the compiler and the assembler that handle these files.

These two types of file are then treated separately. The Java file is compiled thanks to the standard Java compiler (javac), whereas ubi ones are converted into associated hardware kernels in “.hw” files. The last step of the compilation flow is the loading of the HW accelerated code from a “.hw” file into the program memory of the Ubichip. This is done at

runtime by the Ubicom agent.

### B. JubiSplitter

The entry point of the JubiTool compiling environment is the JubiSplitter that splits the Jubi description into a Java description for the software part and several ubi descriptions for the hardware accelerated parts. The JubiSplitter generates “softwareBehaviour” and “hardwareBehaviour” classes. These JADE Behavior classes represents respectively the whole agent functionality in SW mode simulations and the envelope of hardware accelerated kernels, which contains non-parallelizable (non #jubi flagged) code sections in the HW mode. The execution mode is then chosen when the platform is configured. Fig.4 gives an example of this code splitting step that is the first stage of application compiling process.

### C. JubiCompiler

Once a Jubi code has been split into Java and Ubi kernels the JubiCompiler tool compiles every Ubi codes into Ubichip assembly. As a result, the following example:

```
int a [NPE] = {4, 3, 1, 2};
int b [NPE] = {1, 2, 3, 4};
int c [NPE];
c = a + b;
```

is then compiled into:

```
.data
V01="00030004", "00020001"
V02="00020001", "00040003"
V03="00000000", "00000000"

.code
load    r1, V01
load    r2, V02
mov     r1
add     r2
movr    r3
store   r3, V03
```

JubiCompiler is based on a flex[15] and bison [16] description of the Ubi grammar. NPE indicates that every Processing Element of the SIMD processor will receive a different data set. 1-D arrays therefore become PE-local scalar variables. In this way a 1-D NPE dependent variable (in SW) is spanned over the PEs of the Ubichip architecture as scalar variables (in HW), for clarity reasons we named them multi-scalar to differentiate them from vectors.

### D. UbiAssembler

The UbiAssembler is the final tool used in the flow. It is in charge of translating the JubiCompiler generated assembly code into the Ubichip SIMD hardware language. This tool is part of the SpiNDeK environment [17] and was developed to program the Ubichip using assembly. Two main features are provided in this tool in addition to the code translation: the memory layout setup that involves variable to physical

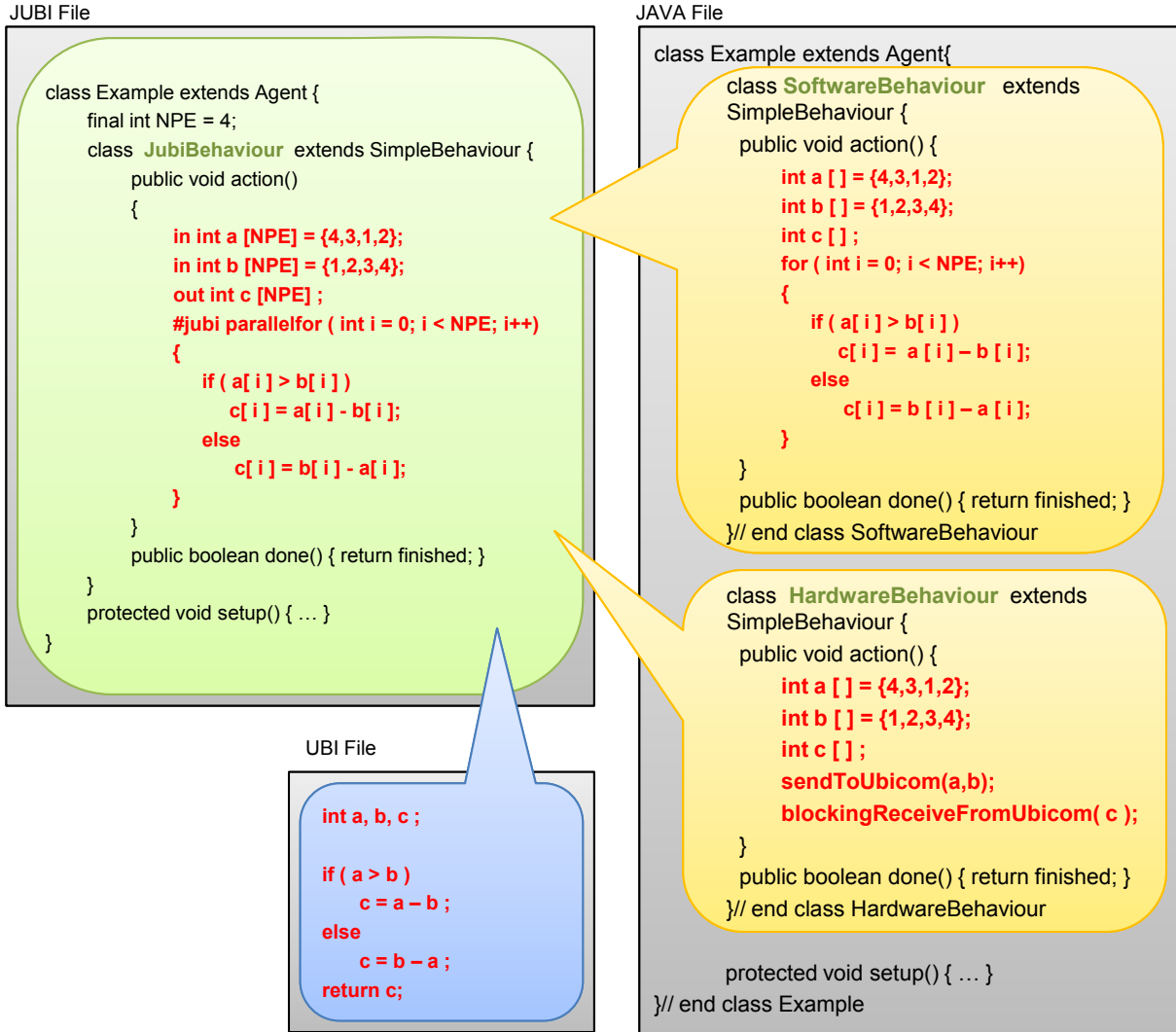


Figure 4. JubiSplitter: Code splitting and cleaning

address translation, and jump linking provided by label to physical address translation.

#### E. Ubicom agent & data transfers

This section deals with the runtime behavior of a typical hardware accelerated application. Fig.2 gives an example of the runtime arrangement of the different stages of this application composed of two sequential parts separated by an intrinsically parallel part. The wrapping of sequential to parallel and parallel to sequential data transfers is performed by the Ubicom agent in the case of the Perplexus platform. This agent will load data in the Ubichip dedicated SRAM and then will receive results from the Ubichip.

As the XScale processor and the Ubichip communicate through a shared bus, data transfers result in latencies (called  $\tau_{com}$ ) between two consecutive parts that have different execution types (sequential and parallel). In fact, applications that have a high level of intrinsic parallelism and do not require frequent changes in execution type will take advantage of the hardware acceleration. The following section gives two

examples of such applications.

#### F. Runtime analysis

As illustrated in Fig.2, runtimes  $T_1$  and  $T_2$  corresponding respectively to a purely SW and a hybrid SH/HW way of computation can be quite different in the case of parallelizable applications. These runtimes are respectively defined by Equations 1 and 2.

$$T_1 = \tau_1 + \tau_2 + NPE \times \tau_{It} \quad (1)$$

$$T_2 = \tau_1 + \tau_2 + 2 \times \tau_{com} + \tau_{PE} \quad (2)$$

The speedup obtained by parallelizing parts of the algorithm can be expressed as follows:

$$S = \frac{T_1}{T_2} = \frac{\tau_1 + \tau_2 + NPE \times \tau_{It}}{\tau_1 + \tau_2 + 2 \times \tau_{com} + \tau_{PE}} \quad (3)$$

Typical applications targeted by the Perplexus platform are data-intensive applications with a high degree of potential

parallelism (like in the Spiking Neural Network (SNN) simulator described in Section IV-A). As an example, with 100 neurons ( $NPE = 100$ ), this application executed in a purely software fashion, spends 95% of its runtime in Neuron state computations. As a result  $\tau_1 + \tau_2$  (representing 5% of the runtime) can be considered as negligible in the total runtime. Therefore the speedup function  $\mathcal{S}_{SNN}$  for this application can be approximated to:

$$\mathcal{S}_{SNN} \approx \frac{NPE \times \tau_{It}}{2 \times \tau_{com} + \tau_{PE}} \quad (4)$$

As the frequency of the Ubichip is lower than the XScale one (95 MHz vs 512 MHz), it is obvious that  $\tau_{PE}$  is longer than  $\tau_{It}$ . Our experiments show that computing on a PE of the Ubichip is approximately 6 times slower than performing the same computation on the XScale processor. Data transfer latency referred as  $\tau_{com}$  represents the time needed to send the data to the different PEs from the XScale. Considering the high number of iterations of the parallelized code - that represents 95% of the application time - in comparison with the small amount of data to be transferred  $\tau_{com}$  can also be considered as negligible with regard to  $\tau_{PE}$ . From this observation, the speedup can be approximated to:

$$\mathcal{S}_{SNN} \approx \frac{NPE \times \tau_{It}}{\tau_{PE}} = \frac{NPE = 100}{6} = 16.66 \quad (5)$$

Obviously, this speedup depends on the targeted type of applications, the number of available resources of the platform to perform computations, and the possibility to extract efficiently the parallelism but at least it gives a reasonable expected gain. Within the confines of the Perplexus project two applications may take advantage of the JubiTool compilation environment. References on the presented model are given as no detailed descriptions are provided in this article.

#### IV. APPLICATION WITHIN PERPLEXUS

##### A. Spiking Neural Network simulator

The article [18] gives precise details on this Spiking Neural Network simulation that aims at studying brain development. As shown on Fig.5(a) the JubiTool will offer the possibility to accelerate the two parts of this algorithm that rely on Neuron state computations. As a typical simulation uses arrays of 100 neurons, this application will take great advantage of the JubiTool.

##### B. Evolutionary Game simulator

Articles [19] and [20] describe an Evolutionary Game-theoretical (EG) model aiming at understanding the interplays between agent connectivity and psychological biases in the cultural evolution of cooperation. Actually this application must provide statistical results taking into account numerous runs for hundreds of parameters combinations therefore this application may take advantage of the Ubichip and more generally the whole Perplexus platform because of the highly parallel aspects of the EG simulations and

the platform possibility to accelerate a single execution as well as to parallelize several runs on different modules. Fig.5(b) gives an abstract description of this application. As for the previously presented application the JubiTool offers the possibility to accelerate these parallel parts given the fact that populations are composed of more than 100 individuals.

#### V. CONCLUSIONS AND PERSPECTIVES

This paper has introduced a novel and original contribution realized within the confines of the Perplexus Project. This solution combines HW independence thanks to the use of Java-based solution with the benefit of parallel execution on a SIMD accelerator. In order to maintain that HW independence, current work aims at porting the compiler to the XScale processor so that HW kernels can be compiled and loaded on the fly on the Ubichip. As the proposed approach and the Jubi language have been designed with broader application fields in mind, it should be appropriate for a wide range of parallelizable applications where SIMD execution brings advantages. The design flow and associated tools are available for Linux x86-based workstations as well as for Windows 32 bits and Mac OSX platforms.

In a short term, we plan to add several optimization features at the compiler level (common subexpression elimination, dead code elimination, partial loop unrolling, constant propagation, etc...). This would increase code efficiency and therefore the performance of the system. It could also be interesting to evaluate the relevancy of implementing a code motion mechanism in the JubiSplitter tool to adapt dynamically the parallelism of the application with regard to the available resources. We also plan to conduct more tests using either Perplexus applications or other applications to precisely bench Perplexus Ubichip SIMD mode speedup for a relevant set of applications.

#### ACKNOWLEDGMENT

This project is funded by the Future and Emerging Technologies program IST-STREP of the European Community, under grant IST-034632 (PERPLEXUS). The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

#### REFERENCES

- [1] E. Sanchez, A. Perez-Urbe, A. Upegui, Y. Thoma, J. M. Moreno, A. Villa, H. Volken, A. Napierlaski, G. Sassatelli, and E. Lavarec, "Perplexus: Pervasive computing framework for modeling complex virtually-unbounded systems," *Proceedings of Second NASA/ESA Conference on Adaptive Hardware and Systems(AHS 2007)*, 2007, IEEE Computer Society.
- [2] A. Upegui, Y. Thoma, E. Sanchez, A. Perez-Urbe, J. M. Moreno, and J. Madrenas, "The perplexus bio-inspired chip," *Proceedings of Second NASA/ESA Conference on Adaptive Hardware and Systems(AHS 2007)*, 2007, IEEE Computer Society.
- [3] J. M. Moreno and J. Madrenas, "A reconfigurable architecture for emulating large-scale bio-inspired systems," *Submitted to Congress on Evolutionary Computation (IEEE CEC 2009)*, 2009.
- [4] O. Brousse, G. Sassatelli, T. Gil, M. Robert, L. Torres, E. Sanchez, J. M. Moreno, A. Villa, H. Volken, A. Napierlaski, and F. Mondada, "Perplexus project modeling framework," *ICESCA'08, Tunis*, 2008.

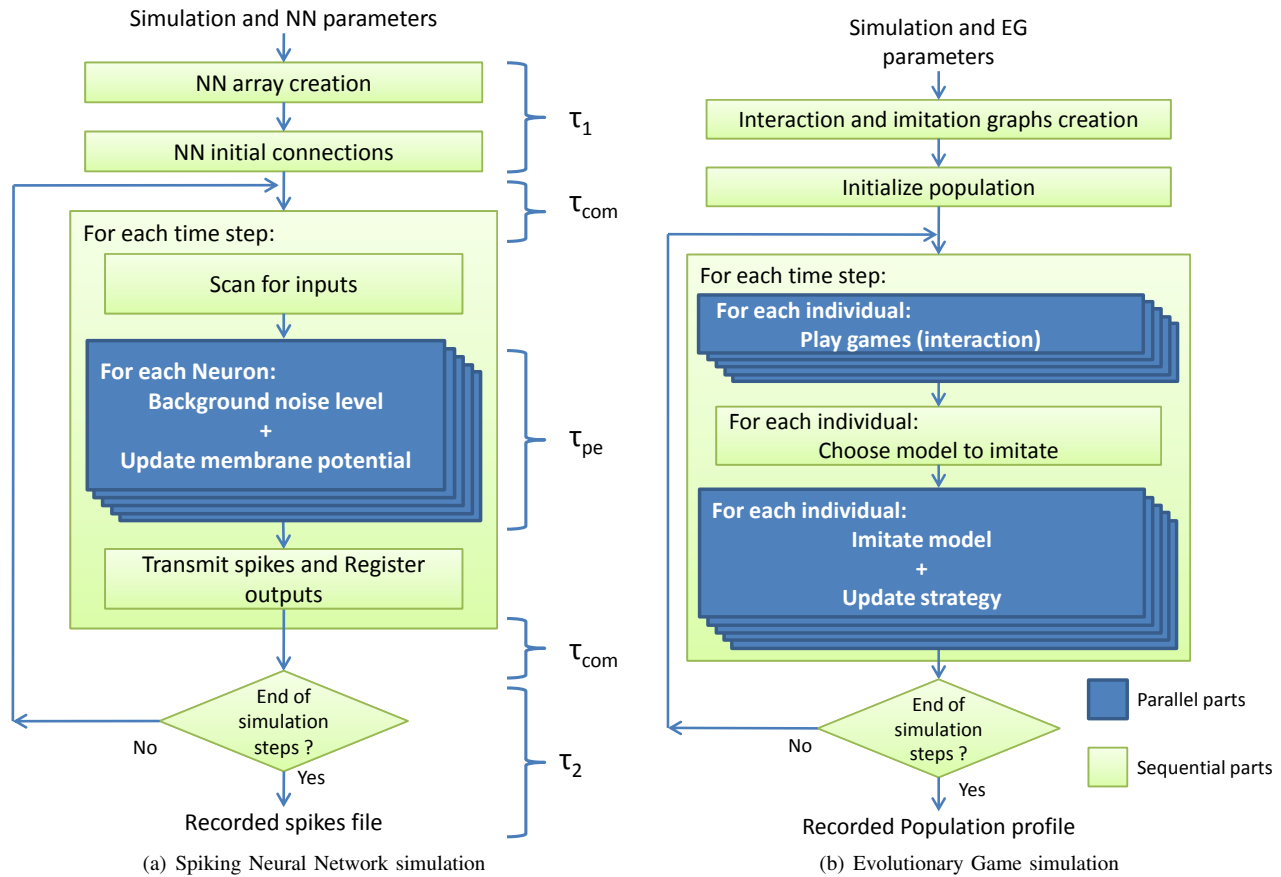


Figure 5. Typical hardware accelerated applications developed by partners involved in the Perplexus project

- [5] O. Brousse, G. Sassatelli, T. Gil, Y. Guilleminet, M. Robert, L. Torres, and F. Grize, "Baf: A bio-inspired agent framework for distributed pervasive applications," *GEM'08, Las Vegas*, 2008.
- [6] O. Brousse, G. Sassatelli, T. Gil, M. Robert, L. Torres, F. Grize, E. Sanchez, A. Upegui, and Y. Thoma, "The perplexus programming framework combining bio-inspiration and agent-oriented programming for the simulation of large scale complex systems," *ICES'08, Prague*, 2008.
- [7] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. Wiley, April 2007.
- [8] S. Microsystems, "Developer resources for java technology," online: 2009-02-11. [Online]. Available: <http://java.sun.com/>
- [9] M. Schoeberl, "Jop: A java optimized processor for embedded real-time systems," Ph.D. dissertation, Vienna University of Technology, 2005.
- [10] D. Hardin, "aj-100: A low-power java processor," *Embedded Processor Forum*, 2000.
- [11] ARM, "Jazelle - arm architecture extension for java applications." 2002, white paper.
- [12] H. McGhan and M. O'Connor, "Picojava: a direct execution engine for java bytecode," *Computer*, vol. 31, no. 10, pp. 22–30, 1998, author affiliation: Sun Microsyst. Inc., Palo Alto, CA.:
- [13] T. Brecht, H. S. M. Shan, and J. Talbot, "Paraweb: Towards world-wide supercomputing," in *In European Symposium on Operating System Principles*, 1996, pp. 181–188.
- [14] J. Maassen, T. Kielmann, and H. E. Bal, "Parallel application experience with replicated method invocation," in *Concurrency and Computation: Practice and Experience*, 2001, pp. 13–8.
- [15] gnu.org, "Flex - a fast lexical analyser generator," online: 2009-02-11. [Online]. Available: [www.gnu.org/software/flex/](http://www.gnu.org/software/flex/)
- [16] —, "Bison - gnu parser generator," online: 2009-02-11. [Online]. Available: [www.gnu.org/software/bison/](http://www.gnu.org/software/bison/)
- [17] M. Hauptvogel, J. Madrenas, and J. M. Moreno, "Spindek: An integrated design tool for the multiprocessor emulation of complex bioinspired spiking neural networks," *Submitted to Congress on Evolutionary Computation (IEEE CEC 2009)*, 2009.
- [18] O. Chibirova, J. Iglesias, V. Shaposhnyk, and A. E. Villa, "Dynamics of firing patterns in evolvable hierarchically organized neural networks," *Lecture Notes in Computer Science*, no. LNCS 5216, 2008.
- [19] J. Peña, "Conformist transmission and the evolution of cooperation," *Proceedings of the Eleventh International Conference on Artificial Life*, 2008.
- [20] J. Peña, E. Pesticelli, M. Tomassini, and H. Volken, "Conformity and network effects in the prisoner's dilemma," *Submitted to Congress on Evolutionary Computation (IEEE CEC 2009)*, 2009.