An FPGA-based supercomputer for statistical physics: the weird case of Janus

M. Baity-Jesi, R. A. Baños, A. Cruz, L. A. Fernandez, J. M. Gil-Narvion, A. Gordillo-Guerrero, M. Guidetti, D. Iñiguez, A. Maiorano, F. Mantovani, E. Marinari, V. Martin-Mayor, J. Monforte-Garcia, D. Navarro,G. Parisi, M. Pivanti, S. Perez-Gaviro, F. Ricci-Tersenghi, J. J. Ruiz-Lorenzo, S. F. Schifano, B. Seoane, A. Tarancon, P. Tellez, R. Tripiccione and D. Yllanes

M. Baity-Jesi
Departamento de Física Teórica I, Universidad Complutense, 28008 Madrid, Spain
Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), Zaragoza, Spain,
e-mail: marcobaityjesi@fis.ucm.es

R. A. Baños
Departamento de Física Teórica, Universidad de Zaragoza, 50009 Zaragoza, Spain
Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), Zaragoza, Spain,
e-mail: raquel.alvarez@unizar.es

A. Cruz
Departamento de Física Teórica, Universidad de Zaragoza, 50009 Zaragoza, Spain
Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), Zaragoza, Spain,
e-mail: andres@unizar.es

L. A. Fernandez
Departamento de Física Teórica I, Universidad Complutense, 28008 Madrid, Spain
Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), Zaragoza, Spain,
e-mail: laf@lattice.fis.ucm.es

J. M. Gil-Narvion
Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), Zaragoza, Spain,
e-mail: jmgil@bifi.es

A. Gordillo-Guerrero
Departamento de Ingeniería Eléctrica, Electrónica y Automática, Universidad de Extremadura, 10071 Cáceres, Spain
Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), Zaragoza, Spain,
e-mail: anto@unex.es

M. Guidetti
Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), Zaragoza, Spain,
e-mail: mguidetti@bifi.es

D. Iñiguez
Fundación ARAID, Diputación General de Aragón, Zaragoza, Spain
Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), Zaragoza, Spain,
e-mail: david.iniguez@bifi.es

A. Maiorano
Dipartimento di Fisica, La Sapienza Università di Roma, 00185 Roma, Italy
Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), Zaragoza, Spain,
e-mail: andrea.maiorano@roma1.infn.it

F. Mantovani

Dipartimento di Fisica Università di Ferrara and INFN - Sezione di Ferrara, Ferrara, Italy,
e-mail: filimanto@fe.infn.it

E. Marinari
Dipartimento di Fisica, IPCF-CNR and INFN, La Sapienza Università di Roma, 00185 Roma, Italy
e-mail: enzo.marinari@uniroma1.it

V. Martin-Mayor
Departamento de Física Teórica I, Universidad Complutense, 28008 Madrid, Spain
Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), Zaragoza, Spain,
e-mail: victor@lattice.fis.ucm.es

J. Monforte-Garcia
Departamento de Física Teórica, Universidad de Zaragoza, 50009 Zaragoza, Spain
Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), Zaragoza, Spain,
e-mail: jmonforte@bifi.es

A. Muñoz Sudupe
Departamento de Física Teórica I, Universidad Complutense, 28008 Madrid, Spain,
e-mail: sudupe@fis.ucm.es

D. Navarro
Departamento de Ingeniería, Electrónica y Comunicaciones and Instituto de Investigación en Ingeniería de Aragón (I3A), Universidad de Zaragoza, 50018 Zaragoza, Spain,
e-mail: denis@unizar.es

G. Parisi
Dipartimento di Fisica, IPCF-CNR, UOS Roma Kerberos and INFN, La Sapienza Università di Roma, 00185 Rome, Italy,
e-mail: giorgio.parisi@roma1.infn.it

M. Pivanti
Dipartimento di Fisica, La Sapienza Università di Roma, 00185 Roma, Italy,
e-mail: pivanti@fe.infn.it

S. Perez-Gaviro
Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), Zaragoza, Spain,
e-mail: spgaviro@unizar.es

F. Ricci-Tersenghi
Dipartimento di Fisica, IPCF-CNR, UOS Roma Kerberos and INFN, La Sapienza Università di Roma, 00185 Rome, Italy,
e-mail: federico.ricci@roma1.infn.it

J. J. Ruiz-Lorenzo
Departamento de Física, Universidad de Extremadura, 06071 Badajoz, Spain
Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), Zaragoza, Spain,
e-mail: ruiz@unex.es

S. F. Schifano
Dipartimento di Matematica e Informatica, Università di Ferrara and INFN - Sezione di Ferrara, Ferrara, Italy,
e-mail: schifano@fe.infn.it

B. Seoane
Departamento de Física Teórica I, Universidad Complutense, 28008 Madrid, Spain
Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), Zaragoza, Spain,
e-mail: seoane@lattice.fis.ucm.es

A. Tarancon

Abstract In this chapter we describe the Janus supercomputer, a massively parallel FPGA-based system optimized for the simulation of spin-glasses, theoretical models that describe the behavior of glassy materials.
The custom architecture of Janus has been developed to meet the computational requirements of these models. Spin-glass simulations are performed using Monte Carlo methods that lead to algorithms characterized by *i)* intrinsic parallelism allowing us to implement many Monte Carlo update engines within a single FPGA; *ii)* rather small data base (2 MByte) that can be stored on-chip, significantly boosting bandwidth and reducing latency. *iii)* need to generate a large number of good-quality long ($\geq$ 32 bit) random numbers; *iv)* mostly integer arithmetic and bitwise logic operations.
Careful tailoring of the architecture to the specific features of these algorithms has allowed us to embed up to 1024 special purpose cores within just one FPGA, so that simulations of systems that would take centuries on conventional architectures can be performed in just a few months.

1 Overview

This chapter describes Janus, an application-driven parallel and reconfigurable computer system, strongly tailored to the computing requirements of spin glass simulations.

A major challenge in condensed-matter physics is the understanding of glassy behavior [1]. Glasses are materials that do not reach thermal equilibrium in human lifetimes; they are conceptually important in physics and they have a strong industrial relevance (aviation, pharmaceuticals, automotive, etc.). Important material properties, such as the compliance modulus or the specific heat, significantly depend on time even if the material is kept for months (or years) at constant experimental conditions [2]. This sluggish dynamics, a major problem for the experimental and

Departamento de Física Teórica, Universidad de Zaragoza, 50009 Zaragoza, Spain
Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), Zaragoza, Spain,
e-mail: tarancon@unizar.es

P. Tellez
Departamento de Física Teórica, Universidad de Zaragoza, 50009 Zaragoza, Spain,
e-mail: ptellez@unizar.es

R. Tripiccione
Dipartimento di Fisica and CMCS, Università di Ferrara and INFN - Sezione di Ferrara, Ferrara, Italy,
e-mail: tripiccione@fe.infn.it

D. Yllanes
Dipartimento di Fisica, La Sapienza Università di Roma, 00185 Roma, Italy
Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), Zaragoza, Spain,
e-mail: yllanesd@roma1.infn.it

theoretical investigation of glassy behavior, places numerical simulations at the center of the stage.

Spin glasses are the prototypical glassy systems most widely studied theoretically [3, 4]. Simulating spin glasses is a computing grand challenge, as their deceivingly simple dynamical equations are at the basis of complex dynamics, whose numerical study requires large computing resources. In a typical spin-glass model, the dynamical variables, one calls them spins, are discrete and sit at the nodes of discrete D-dimensional lattices. In order to make contact with experiments, we need to follow the evolution of a large enough lattice, say a 3D system with 80^3 sites, for time periods of the order of 1 second. One Monte Carlo step – the update of all the 80^3 spins in the lattice – roughly corresponds to 10^{-12} seconds, so we need some 10^{12} such steps, that is $\sim 10^{18}$ spin-updates. One typically wants to collect statistics on several ($\sim 10^2$) copies of the system, adding up to $\sim 10^{20}$ Monte Carlo spin updates. Therefore, performing this simulation program in an acceptable time frame (say, less than one year) requires a computer system able to update on average one spin per picosecond or less.

This analysis shows that accurate simulations of spin glasses have been a major computational challenge; the problem has been attacked in different ways, and the development of application-specific computers has been one of the options considered over the years. This chapter describes the Janus project, which has lead to the development of the Janus reconfigurable computer, optimized for spin-glass simulations. Janus has played a major role in making the simulations described above possible on a reasonable time scale (order of months); it has provided the Janus collaboration with a major competitive advantage, which has resulted in groundbreaking work in the field of spin glasses as will be described later.

There are several reasons that make traditional computer architectures a poor solution for spin-glass simulations and at the same time suggests that a reconfigurable approach may pay very large dividends:

- the dynamical variables describing these systems only take a small number of discrete values (just two in the simplest case); sequences of bitwise logical operations are appropriate to compute most (not all) quantities involved in the simulation;
- a large amount of parallelism is easily identified; a large number of lattice locations can be processed independently, so they can be handled in parallel.
- the structure of the critical computational kernels is extremely regular, based on ordered loops that perform the same sequence of operations on data values stored at regularly stridden memory locations; the control structure of the program can therefore be easily cast in the form of simple state-machines. The control sequence is the same for all lattice locations, so a Single Instruction Multiple Data (SIMD) approach is appropriate and the control structure can be shared by many computational threads.

These points suggest an ideal architecture for a spin-glass engine, based on a very large number of computational cores; cores are extremely slim processors, able to perform only the required mix of logical manipulations and a limited set of arithmetic operations; many cores work concurrently, running the same thread under just one control structure; they process data fetched from memory by just one memory control engine. Seen from a different point of view, one may think of a streaming processor, working on a steady flow of data extracted from and flowing back to memory. As discussed in details later on, the logical complexity of one such computational core is in the order of just a few thousand logical gates, so one can assemble them by the thousands in just one integrated circuit. This promises significant benefits, provided that the huge amount of data needed to keep all these processors busy can be supplied by the memory system; this is a serious problem that can be handled in this case as the size of the simulation data-base is small enough to be accommodated on chip.

The requirements described above are slightly at variance with traditional architectures. On one hand standard CPUs offer features not really exploited by our regular programming paradigm (out of order execution, branch prediction, cache hierarchy); on the other hand they are very limited in the extraction of the available parallelism. Indeed, at the time the Janus project started (early 2006), state-of-the-art simulation programs running on state-of-the-art computer architectures were only able to exploit a tiny fraction of the available parallelism, and had an average update time of one spin every ~ 1 ns, meaning that the simulation campaign outlined above would proceed for centuries.

Curiously enough, the time frame that has seen the development of the Janus project coincides with that in which computer architectures have strongly evolved towards wider and more explicit parallelization: many-core processors with $\mathcal{O}(10)$ cores are now widely available and Graphics Processing Units (GPUs) now have hundreds of what can be regarded as "slim" cores. Today, one might see an ideal spin-glass simulation engine as an *application-specific* GPU, in which i) data paths are carefully tailored to the specific mix of required logical (as opposed to arithmetic and/or floating-point) operations; ii) the control structure is shared by a much larger number of cores than typical in state-of-the-art GPUs; iii) data allocation goes to on-chip memory structures and iv) the memory controller is optimized for the access patterns typical of the algorithm.

Architectures available in 2011-2012 have indeed improved performance for spin glass simulations by about one order of magnitude with respect to what was available when the Janus project started (slightly better than one would predict according to Moore's law, see later for a detailed analysis), but standard commercial computers are even today not a satisfactory option for large-scale spin-glass studies.

We already remarked that, over the years, this state of affairs has motivated the development of several generations of application-driven, spin-glass-optimized systems; this approach has been often taken by computational physicists in several areas, such as Lattice QCD [5, 6, 7] or the simulation of gravitationally coupled systems [8]; early attempts for spin systems were performed more than 20 years ago [9], and – more recently – an approach based on reconfigurable computing was pioneered [10].

The Janus[1] project has continued along this line, developing a large reconfigurable system, based on field programmable gate-arrays (FPGAs). FPGAs are slow with respect to standard processors. This is more than offset by large speedup factors, allowed by architectural flexibility. A more radically application-driven approach would be to consider an ASIC (application-specific integrated circuit), a custom-built integrated circuit, promising still larger performance gains, at the price of much larger development time and cost, and much less flexibility in the design.

The remainder of this chapter is organized as follows: in section 2 we describe the physics systems that we want to simulate, elaborating on their relevance both in physics and engineering; section 3 provides details on the Monte Carlo simulation approach used in our work; section 4 describes the Janus architecture and its implementation, after which section 5 gives a concrete example. Section 6 summarizes the main physics results obtained after more than three years of continuous operation of the machine. Section 7 assesses the performance of Janus on our spin-glass simulations, using several metrics, and compares with more standard solutions. We consider both those technologies that were available when Janus was developed and commissioned and those that have been developed since the beginning of the project ($\approx$ 2006). We also briefly discuss the performance improvements that may be expected if one re-engineers Janus on the basis of the technology available today. Our conclusions and outlook are in section 8.

2 Spin glasses

What makes a spin glass (SG) such a complex physical system is frustration and randomness (see Fig. 1). One typical example is a metal in which we replace some of its metallic atoms with magnetic ones. Qualitatively, its dynamical behavior is as follows: the dynamical variables, the spins, represent atomic magnetic moments, interacting via electrons in the conduction band of the metal and inducing an effective interaction which changes in sign (the RKKY interaction) depending on the spatial location. In some materials, it is easy for the magnetic moments to lie in only one direction (and not in the original three-dimensional space) so we can consider that they only take values belonging to a finite set. Finally we can assume that spins sit at the nodes of a crystal lattice.[2]

At some sites (i and j) in the lattice, neighbor spins (σ_i and σ_j) may lower their energy if they have the same value: their *coupling constant* J_{ij}, a number assigned to the lattice link between i and j, is positive. However elsewhere in the lattice, with roughly the same probability, two neighboring spins may prefer to have different values (in this case, $J_{ij} < 0$). A lattice link is *satisfied* if the two corresponding spins are in the energetically favored configuration. In spin glasses, positive and negative coupling constants occur with the same frequency, as the spatial distribution of pos-

[1] From the name of the ancient Roman god of doors and *gates*.

[2] A typical example of an Ising spin glass is $Fe_{0.5}Mn_{0.5}TiO_3$.

itive or negative J_{ij} is random; this causes *frustration*. Frustration means that it is impossible to find an assignment for the σ_i, that satisfies all links (the concept is sketched in Fig. 1, and explained in the caption).

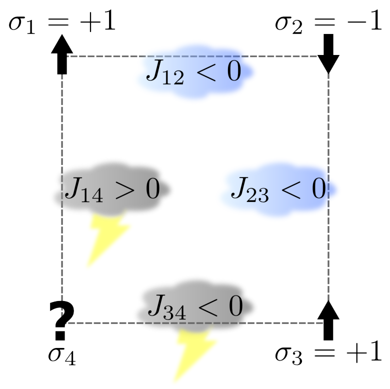


Fig. 1 Four neighboring points of a regular Ising spin lattice (for clarity we show a 2D system). Spins ($\sigma_i = \pm 1$) sit at the edges of the lattice; each link joining two edges has a coupling constant $J_{ij} = \pm 1$. If $J_{ij} > 0$ a link is *satisfied* if $\sigma_i = \sigma_j$, while $J_{ij} < 0$ requires that $\sigma_i \neq \sigma_j$. One can easily check that, for the J_{ij} values in the picture, no assignment of the σ_is exists that satisfies all links (in general, this happens if an odd number of J_{ij} along the circuit has the same sign). This is called *frustation.*

Models that describe this behavior are defined in term of the following energy function:

$$\mathscr{H} = -\sum_{\langle i,j \rangle} J_{ij}\delta(\sigma_i, \sigma_j); \tag{1}$$

σ_i is the spin at lattice site i; it takes discrete values belonging to a finite set of q elements, δ is the Kronecker delta function and J_{ij} are the coupling constants between the two spins; angle brackets mean that the sum is restricted to pairs of nearest neighbors in the lattice

Models described by (1) are usually referred to as Potts spin glasses. One typically considers values of q ranging from just two to eight or ten. The simplest case ($q = 2$) has been especially considered since it was first proposed more than thirty years ago; it is known as the Edwards-Anderson spin glass [11]. One usually writes its energy function in a slightly different form:

$$\mathscr{H} = -\sum_{\langle ij \rangle} \sigma_i J_{ij} \sigma_j; \tag{2}$$

Here, the symbols have the same meaning as in Eq. 1 but in this case $\sigma = \pm 1$; Eq. (2) goes over to (1) – apart from a constant term that does not affect the dynamics – if one appropriately rescales the values of the J_{ij}.

The coupling constants J_{ij} are fixed and chosen randomly to be ± 1 with 50% probability. A given assignment of the $\{J_{ij}\}$ is called a *sample*. Some of the physical properties (such as internal energy density, magnetic susceptibility, etc.) do not depend on the particular choice for $\{J_{ij}\}$ in the limit of large lattices (self-averaging property). However, in the relatively small systems that one is able to simulate, it is useful to average results over several samples.

Frustration makes it hard to answer even the simplest questions about the model. For instance, finding the spin configuration that minimizes the energy for a given set of $\{J_{ij}\}$ is an NP-hard problem [12]. In fact, our theoretical understanding of spin-glass behavior is still largely restricted to the limit of high spatial dimensions, where a rich picture emerges, with a wealth of surprising connections to very different fields [13].

In three dimensions, we know experimentally [14] and from simulations [15] that a spin-glass reaches an ordered phase below a critical temperature T_c. In the cold phase ($T < T_\mathrm{c}$) spins *freeze* in some disordered pattern, related to the configuration of minimal free energy. For temperatures (not necessarily much) smaller than T_c spin dynamics becomes exceedingly slow. In a typical experiment one quickly cools a spin glass below T_c, then waits to observe the system evolution. As time goes on, the size of the domains where the spins coherently order in the (unknown to us) spin-glass pattern, grows.

Domain growth is sluggish, however: in typical spin-glass materials after eight hours at low temperature ($T = 0.73T_\mathrm{c}$), the domain size is only around 40 lattice spacings [16]. The smallness of the spin-glass ordered domains precludes the experimental study of equilibrium properties, as equilibration would require a domain size of the order of $\simeq 10^8$ lattice spacings. However, an opportunity window opens for numerical simulations. In fact, in order to understand experimental systems we only need to simulate lattices sufficiently larger than the typical domain. This crucial requirement has been met for the first time in the simulations made possible by Janus.

3 Monte Carlo simulations of spin glasses

Spin glasses have been heavily studied numerically with Monte Carlo techniques and the Janus architecture has been designed with the main goal of exploiting every performance handle available in this computational area. In this section we provide a simple overview of the relevant algorithms, focusing on those features that will have to be carefully optimized on our reconfigurable hardware. For simplicity, we only treat the Edwards-Anderson model of Eq. (2), defined on a 3D lattice of linear size L; the Monte Carlo algorithms that apply to more general Potts model are similar and – most important in this context – they have essentially the same computational and architectural requirements.

We focus on the Heat-Bath (HB) algorithm [17] that ensures that system configurations $\mathscr{C}$ are sampled according to the Boltzmann probability distribution

$$P(\mathscr{C}) \propto \exp\left(-\frac{H}{T}\right) , \tag{3}$$

describing the equilibrium distribution of configurations of a system at constant temperature $T = \beta^{-1}$. This is one well known Monte Carlo method; see e.g. [18] for a review of other approaches;

Let us focus on a spin at site k of a 3D lattice; its energy is

$$E(\sigma_k) = -\sigma_k \sum_{m(k)} J_{km}\sigma_m \ = \ -\sigma_k \phi_k , \tag{4}$$

where the sum runs over the six nearest neighbors, $m(k)$, of site k; ϕ_k is usually referred to as the *local field* at site k. In the HB algorithm, one assumes that at any time any spin is in thermal equilibrium with its surrounding environment, meaning that the probability for a spin to take the value $+1$ or -1 depends only on its nearest neighbors. Following (3), the probability for the spin to be $+1$ is

$$P(\sigma_k = +1) = \frac{e^{-E(\sigma_k=+1)/T}}{e^{-E(\sigma_k=+1)/T} + e^{-E(\sigma_k=-1)/T}} = \frac{e^{\phi_k/T}}{e^{\phi_k/T} + e^{-\phi_k/T}} , \tag{5}$$

The algorithm then is an iteration of just two steps:

1. Pick one site k at random, and compute the local field ϕ_k (Eq. 4).
2. Assign to σ_k the value $+1$ with probability $P(\sigma_k = +1)$ as in Eq. 5. This can be done by generating a random number r, uniformly distributed in $[0,1]$, and setting $\sigma_k = 1$ if $r < P(\sigma_k = 1)$, and $\sigma_k = -1$ otherwise.

A full Monte Carlo Step (MCS) is the iteration of the above scheme for L^3 times. By iterating many MCS, the system evolves towards statistical equilibrium. Fig. 2 shows a snapshot of a large spin-glass lattice at a later stage of the Monte Carlo evolution for a temperature lower than the critical one, showing the build up of an ordered domain structure.

A further critically important tool for Monte Carlo simulations is Parallel Tempering (PT) [19]. Dealing with frustration (as defined above) means handling rough free-energy landscapes and facing problems such as the stall of the system in a metastable state. PT helps to overcome these problems by simulating many copies of the system in parallel (hence the name) at different (inverse) temperatures β_i and allowing copies, whose β (energy) difference is $\Delta\beta$ (ΔE), to exchange their temperatures with probability equal to $\min\{1, \exp(\Delta\beta\Delta E)\}$. Following PT dynamics, configurations wander from the physically interesting low temperatures, where relaxation times can be long, to higher temperatures, where equilibration is fast and barriers are quickly traversed; they explore the complex energy landscape more efficiently, with correct statistical weights. For an introduction to PT, see for instance [20].

We now sketch the steps needed to implement a Monte Carlo simulation on a computer. First, one maps physical spin variables onto bits by the following transformation, $\sigma_k \to S_k = (1-\sigma_k)/2$, allowing to turn most (not all) steps of the al-

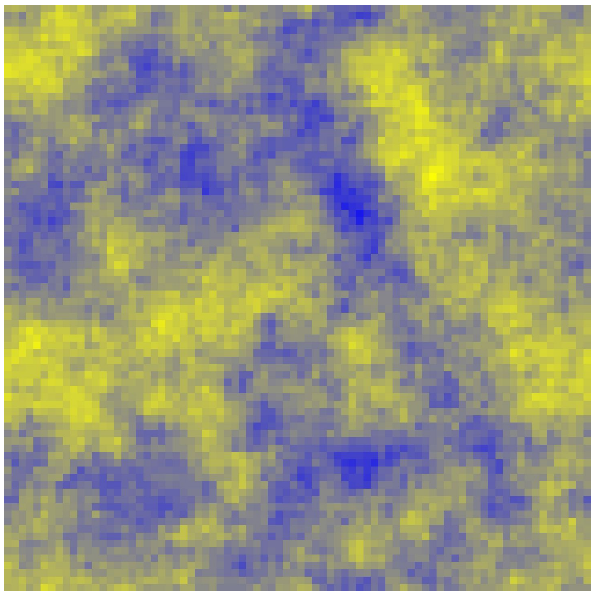


Fig. 2 Domain growth for an Edwards-Anderson spin glass of size $L = 80$ at $T = 0.73T_c$, after 2^{36} Monte Carlo steps, corresponding to a time scale of ≈ 0.1 sec.

gorithm into logic (as opposed to arithmetic) operations. The following points are relevant:

1. The kernel of the program is the computation of the local field ϕ_k, involving just a few logic operations on discrete variable data.
2. The local field ϕ_k takes only the 7 even integer values in the range $[-6,6]$, so probabilities $P(\sigma_k = +1) = f(\phi_k)$ can be stored in a look-up table.
3. High-quality random numbers are necessary to avoid spurious spatial correlations between lattice sites, as well as temporal correlations in the sequence of spin configurations.
4. Under ergodicity and reversibility assumptions, the simulation retains the desired properties even if each Monte Carlo step visits each lattice site exactly once, in any deterministic order.
5. Several sets of couplings $\{J_{km}\}$ (i.e., *different samples*) are needed. An independent simulation has to be performed for every sample, in order to generate properly averaged results.
6. One usually studies the properties of a spin-glass system by comparing the so-called *overlaps* of two or more statistically independent simulations of the *same* sample, starting from uncorrelated initial spin configurations (copies of a sample are usually referred to as *replicas*).

The last three points above identify the parallelism available in the computation; farming easily takes advantage for 5 and 6, while for 4 we need a more accurate analysis. In fact, if we label all sites of the lattice as *black* or *white* in a checkerboard scheme, all black sites have their neighbors in the white site set, and *vice versa*: in principle, we can perform the steps of the algorithm on all white or black sites in parallel.

We will see in the following that the Janus architecture allows us to exploit to a very large degree the parallelism of point 4 above. If one tries the same approach with a standard processor, mapping independent spins of one sample to the bits of one machine word and applying bitwise logical operations, one quickly meets a bottleneck in the number of required random numbers (this approach is known in the trade as Synchronous Multi-Spin coding, SMSC). An alternate approach (known as Asynchronous Multi-Spin coding, AMSC) maps the same spin of independent samples to the bits of the machine word and uses the same random number to decide on the evolution of all these spins (this introduces a tolerable amount of correlation). This strategy does increase overall throughput but does not decrease the time needed to perform a given number of MCS, which is a critical parameter.

4 Janus: the architecture

This section describes the Janus architecture, starting from its overall organization, and then going into the details of its hardware structure, of its reconfigurable components and of its supporting software environment. The idea to develop Janus was born in the early years of this century. After some preliminary analysis helped estimate the level of performance that one could expect, preliminary work really started in late 2005. Early prototypes were available in late 2006, and a large-scale machine was commissioned before the end of 2007. After acceptance tests were completed, Janus become operational for physics in spring 2008. Since then, it has continuously been up and running, and it still provides computer power for Monte Carlo simulations.

4.1 Global structure

The Janus supercomputer is a modular system composed of several Janus modules. Each module houses 17 FPGA-based subsystems: 16 so-called scientific processors (SPs) and one input/output processor (IOP). Janus modules are driven by a PC (Janus host). For our application, the Janus module is the system partition that exploits the parallelism available in the simulation of one spin glass sample. Several modules are then used to farm out the simulation of many spin glass samples, that evolve independently.

We generically refer to SPs and the IOP as *nodes*. The 16 SPs are connected by a 2D nearest-neighbor toroidal communication network, so an application can be mapped onto the whole set of SPs (or on a subset thereof). A further point-to-point network links the IOP to each SP; it is used for initialization and control of the SPs and for data transfer.

The Janus host PC plays a key role of master device: a set of purpose-made C libraries are written using low levels of Linux operating system in order to access the raw Gigabit Ethernet level (excluding protocols and other unhelpful layers adding latencies to communications). Moreover two software environments are available: an interactive shell written in Perl mostly used for testing and debugging or short preliminary runs and a set of C libraries strongly oriented to the physics user, making it relatively easy to set up simulation programs for Janus.

The FPGA panorama was various and the choice of a device for Janus was driven by the simple idea that the only important feature is the availability of memory and logic elements in order to store lattices as large as possible and to house the highest number of update engines. Large on-chip memory size and many logic elements are obviously conflicting requirements; each FPGA family offered different trade-offs at the time of the development phase of Janus.

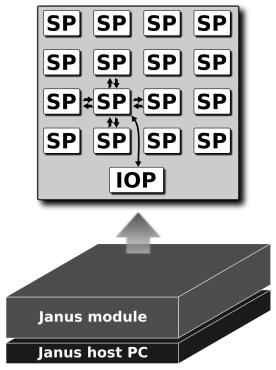
Our preliminary prototype was developed in 2005 using a PCI development kit housing an Altera Stratix S60 FPGA providing ~ 57000 logic elements and ~ 5 MB of embedded memory. The first two Janus prototype boards developed in 2006 had Xilinx Virtex-4 LX160 FPGAs while the final implementation of the system was based on Xilinx Virtex-4 LX200 FPGAs.

The choice between Altera or Xilinx FPGAs has not been fully trivial. While both families had approximately the same amount of logic elements[3], the amount of on-chip memory was different: Altera Stratix-II FPGAs offered ~ 8 Mb organized in three degrees of granularity allowing us to efficiently exploit only $\sim 50\%$ of it. Conversely Xilinx Virtex-4 LX200 FPGAs provided ~ 6 Mb of embedded memories made up of relatively small blocks that we could use very efficiently for our design.

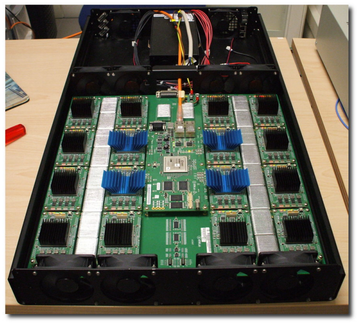
The main clock for Janus is 62.5 MHz; we set a rather conservative clock frequency, trying to minimize time-closure problems when mapping the reconfigurable portion of an application onto the FPGAs. Selected parts of the system use faster clocks: for instance the Gigabit-ethernet interface within the IOP has a 125 MHz clock (called I/O-clock), as needed by Gigabit protocol constraints. Perfect bandwidth balance is achieved: the Gigabit protocol transfers 1 byte per I/O-clock cycle (i.e., 8 bits every 8 ns) and a so called *stream router* forwards data to the Janus world with a rate of 2 bytes per system-clock cycle (i.e., 16 bits every 16 ns). Furthermore link connecting the IOP with the SPs is 8 bit wide and runs a double data rate protocol so the bandwidth continues to be balanced.

The 17 FPGA-based nodes are housed on small daughter-cards plugged into a mother-board (see Fig. 3a for a sketchy block diagram and Fig. 3b for a picture of one Janus module). We used daughter-cards for all nodes to make hardware maintenance easier and also to allow an easier technology upgrade. The first large Janus system, deployed in December 2007 at Zaragoza, has 16 modules and 8 Janus host

[3] We consider the largest devices of both FPGA families available when we had to make a final decision: Altera Stratix-II 180 and Xilinx Virtex-4 LX200.



(a)



(b)

Fig. 3 (a) Topology of a Janus board: each SP communicates with its nearest neighbors in the plane of the board. (b) Janus board housed in a Janus module.

PCs assembled together in a standard 19" rack. More details on the Janus architecture and its implementation are given in [21, 22, 23, 24].

4.2 Programming paradigm

The programming framework developed for Janus is intended to meet the requirements of the prevailing operating modes of Janus, i.e., supporting (re)configuration of SPs, initialization of memories and data structures within the FPGA, monitoring the system during runs, interfacing to memory;

Applications running on Janus can be thought of as split into two sub-applications, one, called *software application* SA, written for example in C, and running on the Janus host. The other, called *firmware application* FA, written for example in VHDL, runs on the SP nodes of a Janus board. As shown in Fig. 4, the two entities, SA and FA are connected together by a *communication infrastructure* CI, which is a logical block including physically the IOP and which allows to exchange data and to perform synchronization operations, directly and in a transparent way.

The CI abstracts the low-level communication between SA and FA applications, implemented in hardware by the IOP and its interfaces. It includes, a C communication library linked by the SA, a communication firmware running on the IOP processor, interfacing both the host PC and the SP processor and a VHDL library linked by the FA.

Firmware running on the IOP communicates with the host PC via a dual Gigabit channel, using the standard RAW-Ethernet communication protocol. To guarantee reliability of the communication in the direction IOP to Janus host, we adopt the *Go-back-N* protocol [25] allowing us to reach approximately the 90% of the full Gigabit bandwidth, when transferring messages of the order of 1 MB, and using the

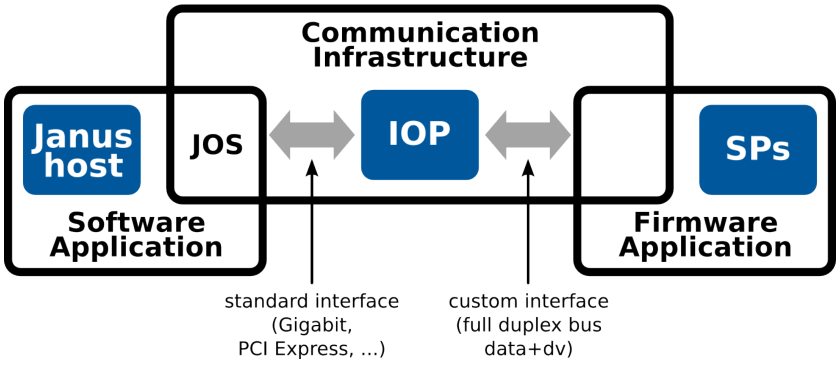

Communication Infrastructure
Janus host
JOS
IOP
SPs
Software Application
Firmware Application
standard interface (Gigabit, PCI Express, ...)
custom interface (full duplex bus data+dv)

Fig. 4 Framework of an application running on the Janus system.

maximum data payload per frame, 1500 bytes. This is enough and safe in a context of spin-glass simulations.

Communications from IOP to host PC do not adopt any communication protocol since the IOP interface, barring hardware errors, ensures that no packets are lost. Incoming frames are protected by standard Ethernet CRC code, and errors are flagged by the IOP processor.

Data coming from the SA application packed as burst of frames are routed to one of the devices supported by the IOP firmware. These devices can be internal to the IOP (e.g., memory interface, internal control registers) or external (e.g., SPs).

Developers of Janus-based applications have to provide their SA and FA relaying on the CI communication infrastructure, to make the two applications collaborative. A typical SA configures the SPs with the appropriate firmware, using functions provided by the communication library, loads input data to the FA, starts and checks the status of the SP logic and waits for incoming results.

4.3 IOP structure and functions

The guidelines for the architectural structure of the IOP come from the original idea of the project that each Janus core is a "large" co-processor of a standard PC running Linux, connected to the host with standard networking interfaces. Spin glass simulations are characterized by long runs with limited interaction with the Janus-host so that each system is loosely coupled with its host. This is different from similar systems in which the FPGA is tightly coupled to a traditional PC and its memory, like in the Maxwell machine [26] or more recently on Maxeler computers [27].

Each simulation starts with the upload of an FA configuring the FPGA of the SPs, followed by the initialization of all the FA data structures (i.e., upload via Gigabit of lattice data, random numbers seeds and other physical parameters). After this step has completed, the SA starts the simulation and polls the status of each Janus module. The SA detects the end of the FA task and initiates the download of the results (e.g., the final physical configurations) and in some cases runs data analysis. All these operations involve the CI and in particular the firmware of IOP under the control of the Janus-host.

In some cases (e.g., when running the parallel tempering) the SA requires data exchange across different SPs during the run: in this case the IOP performs the additional task of gathering data from all SPs, performing a small set of operations on them and re-scattering data to SPs.

From this simplified operation scheme it is clear that the IOP plays a key role between the Janus operating system (JOS) running on the Janus-host and the FA running on each SP.

The IOP, like the SPs, is based on a Virtex 4 XC4LX200 FPGA but, unlike the SPs, has 8 MB static memory, a PROM programming device for FPGA boot and some I/O interfaces: a dual Gigabit channel, a USB channel and a slow serial link for debug.

The current IOP firmware is not a general purpose programmable processor: its role is to allow data streaming from the Janus-host to the appropriate destination (and back), under complete control of the JOS.

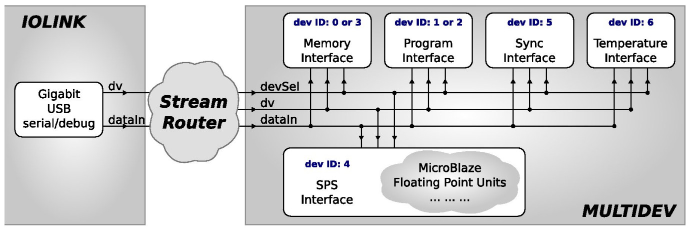


Fig. 5 Block diagram of the IOP architecture.

As shown in figure 5, the IOP structure is naturally split in 2 functional areas called *IOlink* and *MultiDev* blocks.

The IOlink block handles the I/O interfaces between IOP and the Janus-host (Gigabit channels, serial and USB ports). It supports the lower layers of the Gigabit Ethernet protocol and performs CRC checks to ensure data integrity.

The MultiDev block contains a logic device associated to each hardware subsystem that may be reached for control and/or data transfer: a memory interface for the staging memory, a programming interface to configure the SPs, an SP interface to handle communication with the SPs (after they are configured) and several service/debug interfaces are present. Each interface receives a data stream, strips header words and forwards the stream to the target hardware component (memory, SPs, etc.) as encoded in the header.

In order to route the input stream coming from IOlink we implemented a module called *Stream Router* that scans the data stream and recognizes the masks associated to each device within the MultiDev.

The current IOP implementation uses $\sim 4\%$ of the total logic resources and $\sim 18\%$ of the total memory resources of the FPGA; this means that from the point of view of the Janus developers a future expansion of the IOP functionalities is possible and easy to implement simply adding devices to the MultiDev entity. For instance one might add floating point units in order to perform floating point arithmetic directly "on module" or include a MicroBlaze$^{\text{TM}}$ microprocessor opening the possibility to use the Janus core as a standalone computer.

From the point of view of the Janus user, the IOP, together with a software layer running on the host PC, is meant to be part of the CI and therefore a stable structure in the Janus system. What is not fixed on the other hand are the functionalities implemented on the SPs, that can perform in principle arbitrary algorithms (of course taking into account the hardware design/limitations).

In the following sections we will describe the details of the software layer interfacing the SA of a user to the IOP and SPs and later an example of just one specific SP firmware used for spin-glass simulations along the lines described in previous sections.

4.4 Software layer

As part of the CI we developed a software environment running on each Janus host able to cope with any SP-based FA, as long as the latter adheres to a model in which Janus is a memory based coprocessing engine of the host computer; user programs can therefore use load/store instructions to move their data onto the FA storage devices (e.g., FPGA embedded memories) and activate, stop and control Janus processes mapped in the FA.

This model is supported by a host-resident run-time environment that we call Janus operating system (*JOS*). It runs on any Linux-based PC and builds on a low-level C library, based on standard Unix raw network sockets. It implements the protocol needed to communicate with the IOP firmware on the Gbit Ethernet link.

For the application user, JOS consists of:

- a multi-user environment for Janus resource abstraction and concurrent jobs management (`josd`);
- a set of libraries with primitives in order to interact with the CI level (e.g., IOP devices), written both in Perl and C (`JOSlib`);
- a set of FA modules for scientific applications and the corresponding C libraries needed to control them via the josd environment (`jlib`).

josd is a background job running on the Janus host, providing hardware abstraction and a stable interface to user applications. It hides all details of the underlying structure of Janus, mapping it to the user as a simple grid of SPs. It interfaces via Unix socket APIs, so high-level applications may be written in virtually any programming language. Whenever a new FA module is developed, new primitives controlling that module are added to JOSlib. User programs, written in high-level languages, use these primitives in order to schedule and control Janus-enabled runs. For debugging and test, an interactive shell (`JOSH`), written in Perl and also based on JOSlib, offers complete (and potentially dangerous) access to all Janus resources for expert users. It provides direct access to the CI, allowing to communicate with the IOP and drive all its internal devices.

5 SP Firmware: an application example

SPs are fully configurable devices, so they can be tailored to perform any computational task compatible with the available resources and complying with the communication and control protocols of the CI. In this section we discuss one example, taken from a set of several applications that we developed for spin glass simulations.

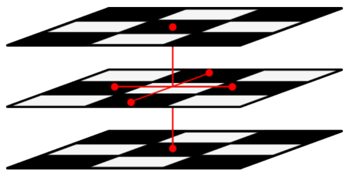
Tailoring our FAs for Janus has been a lengthy and complex procedure, justified by the foreseen long lifetime of each application and by an expectation of huge performance gains. Obviously, a high-level programming framework that would (more or less) automatically split an application between standard and reconfigurable processors and generate the corresponding codes would be welcome. Unfortunately the tools available at the time of the development of the machine do not deliver the needed level of optimization and for Janus this work was done manually using a hardware description language (VHDL).

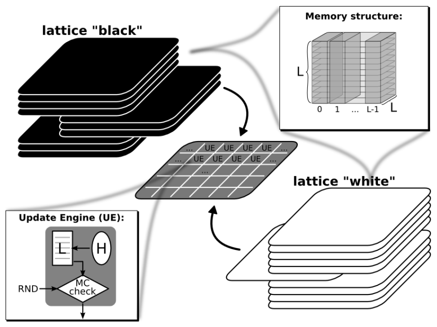
Our implementation of model and algorithm tries to exploit all internal resources in the FPGA in a consistent way (see [21] for a detailed description). Our VHDL code is parametric in several key variables, such as the lattice size and the number of parallel updates. In the following description we consider, for definiteness, a lattice of 80^3 sites corresponding to the typical simulation described in the introduction.

As described in section 3 these algorithms do not allow us to update at the same time spins sitting next to each others in the lattice. On the other hand we can organize the spin update process in two steps such that we process in parallel up to half of the spins at each step. We can in other words split our 3D lattice of spins in a checkerboard scheme and update first all the white spins and then all the black ones (see Fig. 6a).

Virtex 4 LX200 FPGAs come with many small embedded RAM blocks; they can be combined and stacked to naturally reproduce a 3D array of bits representing the 3D spin lattice and making it possible to access data with almost zero latency. We used this memory structure to store the lattice variables, carefully reordered and split in black and white sets. A similar storage strategy applies to the read-only couplings. After initialization, the machinery fetches all neighbor spins of a portion of a plane of spins and feeds them to the update engines.

The flexibility given by the use of FPGAs allow us to implement a number of update engines matching the number of spins processed in parallel. Each update engine returns the processed (updated) spins to be stored at the same given address. For the Edwards Anderson spin glass, we update from 800 to 1024 spins simultaneously; the update logic is made up of a matching number of identical update engines. Each


(a)



(b)

Fig. 6 (a) Checkerboard model used for updating spins in parallel. (b) Logic diagram of the spin update process performed in each SP.

engine receives the 6 nearest-neighbor spins, 6 couplings and one 32-bit random number; it then computes the local field, which is an address to a probability look-up table; the random number is then compared to the extracted probability value and the updated spin is obtained (See Fig. 6b).

Look-up tables are small 32-bit wide memories instantiated as *distributed* RAM. There is one such table for each update cell. Random number generators are implemented as 32-bit Parisi-Rapuano generators [28], requiring one sum and one bitwise XOR operation for each number. To sustain the update rate we need one fresh random value for each update engine; our basic random number engine has 62 registers of 32 bits for the seeds and produces 80 random numbers per clock cycle by combinatorial cascades; 8 to 13 of these random number generators are instantiated in the FA.

The number of $800 \cdots 1024$ updates per clock cycle is a good trade-off between the constraints of allowed RAM-block configurations and available logic resources for the update machinery. Total resource occupation for the design is $75-80\%$ of the RAM blocks (the total available RAM is $\simeq 672$ KB for each FPGA) and $85-94\%$ of the logic resources. The system runs at a conservative clock frequency of 62.5 MHz. At this frequency, the power consumption for each SP is $\simeq 35W$.

Further optimization techniques are used to improve performances and reduce resource usage: for example we can simulate at the same time two replicas of the lattice (sharing the random numbers) to increase the statistic, at a small additional cost in terms of memory consumption. Details on this and other tricks are available elsewhere [21].

6 An overview of physics results

In almost 4 years of continuous operation, Janus has provided the Janus collaboration with a major competitive advantage, allowing us to consider lattice sizes and simulation time scales that other researchers could only dream of. This has resulted in ground-breaking work in the field of spin-glasses; major work has been done on the Potts glass model (for several values of q) and on the Edwards-Anderson model with Ising spins described in section 2. We refer the reader to the original papers for a full account; here we provide only a very short and sketchy panorama of our main results.

As for all problems that are not really understood, spin glasses should be studied with a large variety of tools. Indeed, we do not know where the essential clue will come from, so being able to perform different types of simulations efficiently is very important. FPGA reconfigurability is a major asset in this context.

From the experimental point of view, a major limitation is imposed by the slow dynamics, which makes it impossible to study the equilibrium phase diagram. In order to reproduce and understand experimental results, it is very important to perform simulations that match experimental conditions. Experimentalists work with very large samples (containing some $N \sim 10^{23}$ spins), and follow the dynamical evolution for time scales that span several orders of magnitude. They focus their attention on *self-averaging* quantities. These magnitudes are such that, for large enough samples, they take the same value irrespective of the particular configuration of the couplings (technically, their sample variance scales as an inverse power of the number of spins, N). Examples of self-averaging quantities include the internal energy, the magnetic susceptibility (i.e., the derivative of the magnetization density with respect to the applied field), or some correlation functions. Self-averaging is a most important property: it makes it possible to compare data from different experimental teams, working with different spin-glass samples of nominally identical chemical composition.

Hence, if our *dynamic* simulations are to imitate experiments, we need to follow the dynamics of a single system for a time scale spanning several orders of magnitude. The simulated system should be as large as possible, in order to minimize the artifacts introduced by the finite size of the simulated samples. The only good news come from the self-averaging nature of the quantities that one studies: if the simulated systems are large enough, one may average the obtained results over a moderate number of samples (most of the time experimentalists work with just one or two samples!).

The reader may rightly question about how large is ‘large enough’. The point is that, as time proceeds, glassy domains grow in the system, whose size defines a time-dependent ‘coherence length’ $\xi(t_w)$. As long as $\xi(t_w)$ is much smaller than the lattice size, the system behaves as if its size were infinite and reproduces the experimental evolution. However, when the coherence length begins to approach the lattice size, spurious finite-size effects appear. These systematic errors scale as $\exp(-L/\xi(t_w))$; one should make sure that (relative) statistical errors are much larger than this value. So, the precise meaning of “large enough” depends on time, temperature and accuracy. As a rule of thumb, one is on the safe side if the lattice size is at least seven or eight times larger than $\xi(t_w)$ [29]. Since the coherence length grows with the simulation time, a given lattice size may well be large enough for 10^5 Monte Carlo steps but not for 10^{10} Monte Carlo steps. Janus has proven an excellent compromise in this respect. It has allowed us to follow the dynamics for some 10^{11} Monte Carlo steps, on hundreds of samples containing $N = 80^3 \sim 5 \times 10^5$ spins. Since a single lattice sweep corresponds roughly to a picosecond (i.e., 10^{-12} seconds), this means that we have covered the range from the microscopic time scale to one tenth of a second, which is already long enough to understand what happens in the experimental regime [29, 30]

On the other hand, theoretical physicists prefer a different approach. They like to think about a complex phase space, with many local minima, where the system may get trapped for quite a long time. The natural framework for this way of thinking is equilibrium thermodynamics. Hence, we need to reach thermal equilibrium, meaning that the coherence length is as large as the system size. Under these conditions, almost no magnitude is self-averaging. One needs to describe the physics in terms of probability distributions with the disorder. In practice, one needs to reach thermal equilibrium on several thousands of samples, obtain thermal mean values over each of them, and afterwards study the disorder distributions (i.e., quantify how much the *same* quantity can vary, if computed over different samples). An added difficulty is that thermal equilibrium is terribly difficult to reach. Even worse, the larger the system, the harder the equilibration. And, of course, the larger the system, the more significant the reached results.

Fortunately, when one wants to reach equilibrium, it is no longer important that the computational dynamics resemble, in any way, the physical dynamics. The only important mathematical property is *balance* (see e.g., [17]). This allows an enormous flexibility in the choice of the dynamic rules. In particular, the already discussed parallel tempering dynamics outperforms by orders of magnitude the simple heat-bath algorithm used in the non-equilibrium simulations. Even then, one may need as many as 10^{11} parallel tempering steps (each parallel tempering step is followed by 10 heat-bath full lattice sweeps) in order to reach thermal equilibrium in some samples containing only $N = 32^3$ spins [31].

In our simulations with Janus, we reached equilibrium on thousands of relatively small samples (ranging from $N = 16^3$ to $N = 32^3$, smaller systems were simulated on PCs). This simulation campaign was extremely long: all in all Janus performed 10^{21} spin updates. In the case of the worst samples we estimated that the necessary wall clock time was well over six months. For these samples we have accelerated the simulation by increasing the level of parallelism, by running the PT temperature-assignment procedure on the IOP. This has allowed us to distribute the set of temperatures along several FPGAs on a the same module, speeding up the simulation accordingly. These simulations have opened a new window into the nature of the equilibrium spin-glass phase [31, 32, 33]

Finally, we combined the results of both the non-equilibrium and the equilibrium simulation to clarify, in a quantitative way, the relation between the dynamical evolution and the equilibrium spin-glass phase. We did this by means of a finite-time scaling formalism, with interesting implications for experimental work [34].

Regarding the Potts glass, we studied its phase transition for $q = 4,5,6$, simulating lattices of up to $N = 16^3$ [35, 36]. We found that, in contrast to the mean-field prediction, this transition remained of the second order for all the considered values of q and that ferromagnetic effects were not relevant.

A further example of the benefits of the FPGA reconfigurability is the possibility of simulating the spin glass under an applied magnetic field. In fact, the fate of the spin-glass phase when an external field is switched on is one of the major open questions in the field. Janus is rather efficient in this context, both for the simpler dynamic simulations, or for the equilibrium simulations that need parallel tempering. Our recent results on this problem have been reported in [37].

7 Janus performance

In this section we analyze both computing and energy performances of the Janus system for some of the applications described in Sec. 6, and compare with that of systems based on commodity CPUs and GPUs, available both when Janus was developed as well as today. The tables in this section originally appeared in [40] and are reproduced with kind permission of The European Physical Journal (EPJ).

Let us first estimate the effective computing power delivered by a full Janus system, configured to run an Edwards-Anderson simulation. SPs run at clock frequency of 62.5 MHz, and at each clock cycle conservatively update 800 spins. An equivalent C program running on a commodity CPU architecture could require to perform at least the following mathematical operations for each spin-update:

- 1 32-bit integer sum
- 2 32-bit xor
- 6 3-bit integer sum
- 6 3-bit xor
- 1 32-bit integer comparison

In the above count we have neglected operations to load data and instructions, and to compute memory address, which are obviously necessary during the run. Counting the 6 short xor and sum operations as one single 32-bit integer operation each, we end up with 6 equivalent operations for each spin update. This translates into a required processing power of $6 \times 800 \times 62.5 \times 10^6$ operations per second, corresponding to a sustained performance of 300.0 Giga-ops for a single SP, and 76.8 Tera-ops for a full Janus system running 256 SPs.

At the time the Janus project started, early 2006, state-of-the-art commodity systems where based on dual-core CPUs. Prior to actually building the system we made an extensive analysis of the performance gain that we could expect from the new machine (see for instance [21]). Table 1 contains a short summary of that analysis, listing the relative speed-up for the Ising and the Potts models of one Janus SP with respect to standard processors available in 2006-2007 .

Model	Algorithm	Intel Core 2 Duo	Intel i7
3D Ising EA	Metropolis	45×	10×
3D Ising EA	Heat Bath	60×	-
q=4 3D glassy Potts	Metropolis	1250×	-

Table 1 Speed-up factors of one Janus SP with respect to state-of-the-art CPUs available at the time the project was started.

Let us now make a comparison of energy efficiency between Janus and commodity computing systems based on PCs at the same point in time as above. Let us consider the case of a simulation campaign of a EA model on a lattice size of 64^3 for 10^{12} Monte Carlo steps and 256 samples. In the comparison, we consider both AMSC and SMCS strategies, and estimate the power consumption of one PC at $\simeq 100$ W. Table 2 shows the comparison performance of the PC cluster versus the Janus system in terms of energy dissipated and wall-clock time needed to perform the simulation.

Since the deployment of Janus, in Spring 2008, significant improvements have been made in the architecture and performance of commodity architectures, and in spite of that, Janus is still a very performing machine.

We have extensively compared [38, 39] Janus with several multi-core systems based on the IBM Cell Broadband Engine, the multi-core Nehalem Intel CPU, and the NVIDIA Tesla C1060 GP-GPU. We have made this exercise for the Ising model (as opposed to the Potts model) as in the former case the relative speed-up is much smaller, so we may expect traditional processors to catch up earlier. We consider these results as state-of-the-art comparisons, assuming that within a factor 2 they are still valid for even more recent multi-core architectures, like the Fermi GPUs. This assumption is indeed verified by an explicit test made on the very recent 8-core Intel Sandy Bridge processor.

As discussed in previous sections, for traditional processor architectures we analyzed both SMSC and AMSC strategies and also considered mixes of the two tricks (e.g., simulating at the same time k spins belonging to k' independent samples), trying to find the best option from the point of view of performance.

A key advantage of Janus is indeed that there is no need to look for these compromises: an SP on Janus is simply an extreme case of SMSC parallelization: if many samples are needed on physics ground, more SPs are used. Equally important, if different samples need different numbers of Monte Carlo sweeps (e.g., to reach thermalization), the length of each simulation can be individually tailored without wasting computing resources on other samples, as would necessarily be the case in an AMSC approach.

Performance results for Janus are simply stated: one SP updates ≈ 1000 spins at each clock cycle (of period 16 ns), so the spin-update time is 16 ps/spin for any lattice size that fits available memory. In the cases of $L = 96$ and $L = 128$ there is

	Janus	AMSC	SMSC
processor	1 SP	1 CPU	1 CPU
statistic	1 (16)	1 (128)	1 (4)
wall-clock time	50 days	770 years	25 years
energy	2,7 GJ	2,3 TJ	78,8 GJ
processor	256 SPs	2 CPUs	256 (64) CPUs
statistic	256	256	256
wall-clock time	50 days	770 years	25 years
energy	43 GJ	4,6 TJ	20 (5) TJ

Table 2 Energy comparison between Janus and commodity PCs. The upper part of the table compares the performance of 1 SP versus a PC. The lower part compares the required time and energy to run the simulation on 256 lattice replicas.

3D Ising spin-glass model, SMSC (ns/spin)						
L	Janus SP	I-NH (8-Cores)	CBE (8-SPE)	CBE (16-SPE)	Tesla C1060	I-SB (16 cores)
16	0.016	0.98	0.83	1.17	–	–
32	0.016	0.26	0.40	0.26	1.24	0.37
48	0.016	0.34	0.48	0.25	1.10	0.23
64	0.016	0.20	0.29	0.15	0.72	0.12
80	0.016	0.34	0.82	1.03	0.88	0.17
96	–	0.20	0.42	0.41	0.86	0.09
128	–	0.20	0.24	0.12	0.64	0.09

Table 3 SMSC update time (in ns) for a 3D Ising spin-glass (binary) model of lattice size *L*, for Janus and for several state-of-the-art processor architectures. I-NH (8-Cores) a dual-socket quad-core Intel Nehalem board, CBE (16-SPE) a dual-socket IBM Cell board, and I-SB a dual-socket eight-core Intel Sandy Bridge board.

3D Ising spin-glass model, AMSC (ns/spin)						
L	Janus	I-NH (8-Cores)	CBE (8-SPE)	CBE (16-SPE)	Tesla C1060	I-SB (16 cores)
16	0.001 (16)	0.031 (32)	0.052 (16)	0.073 (16)	–	–
32	0.001 (16)	0.032 (8)	0.050 (8)	0.032 (8)	0.31 (4)	0.048 (8)
48	0.001 (16)	0.021 (16)	0.030 (8)	0.016 (16)	0.27 (4)	0.015 (16)
64	0.001 (16)	0.025 (8)	0.072 (4)	0.037 (4)	0.18 (4)	0.015 (8)
80	0.001 (16)	0.021 (16)	0.051 (16)	0.064 (16)	0.22 (4)	0.011 (16)
96	–	0.025 (8)	0.052 (8)	0.051 (8)	0.21 (4)	0.012 (8)
128	–	0.025 (8)	0.120 (2)	0.060 (2)	0.16 (4)	0.011 (8)

Table 4 AMSC update time for the 3D Ising spin-glass (binary) model, for the same systems as in the previous table. For Janus, we consider one core with 16 SPs. The number of systems simulated in parallel in the multi-spin approach is shown in parentheses

not enough memory in the FPGA to store the lattice and we do not represent the performance in the tables. For standard processors, we collect our main results for the 3D Ising spin-glass in tables 3 and 4 for SMSC and AMSC respectively.

We see that performance (weakly) depends also on the size of the simulated lattice: this is an effect of memory allocation issues and of cache performance. All in all, recent many-core processors perform today much better that 5 years ago: the performance advantage of Janus has declined by a factor of approximately 10 for SMSC: today one Janus SP outperforms very latest generation processors by just a factor $5\times\cdots 10\times$. It is interesting to remark that GP-GPUs are not the most efficient engine for the Monte Carlo simulation of the Ising model: this is so, because GP-GPU strongly focus on floating-point performance which is not at all relevant for this specific problem. There is one point where Janus starts to show performance limits, it is associated to the largest system size that the machine is able to simulate: no significant limit applies here for traditional processor.

All in all, for the specific applications we have presented in this chapter, Janus – after 4 years of operation – still has an edge of approximately one order of magnitude, which directly translates on the wall-clock time of a given simulation campaign.

8 Conclusions

This chapter has described in detail the Janus computer architecture and how we have configured the FPGA hardware to simulate spin-glass models on this architecture. We have also briefly reviewed the main physics results that we have obtained in approximately 4 years operating with this machine.

From the point of view of performance, Janus still has an edge on computing systems based on state-of-the-art processors, in spite of the huge architectural developments since the project was started. It is certainly possible to reach very high performances in terms of spin flips per second using multi-spin coding on CPUs or GP-GPUs (or simply by spending money on more computers), thus concurrently updating many samples and achieving very large statistics. In the Janus collaboration, we have instead concentrated on a different performance goal: minimizing the wall-clock for a very long simulation, by concentrating the updating power in a single sample. This has allowed us to bridge the gap between simulations and experiments for the non-equilibrium spin-glass dynamics or to thermalize large systems at low temperatures, thus gaining access to brand new physics. In particular, one single SP of Janus is able to simulate (two replicas of) an $L = 80$ three-dimensional lattice for 10^{11} MCS in about 25 days.

Janus provides one of the few examples of the development of a successful large scale computing application fully running on a reconfigurable computing infrastructure. This success comes at the price of a large investment in mapping and optimizing the application programs onto the reconfigurable hardware. This has been possible in this case as the Janus group has a full understanding of all facets of the algorithms and every performance gain immediately brings very large dividends in term of a broader physics program. Most potential FPGA-based applications do not have equally favorable boundary conditions, so automatic mapping tools would be most welcome; however further progress is needed in this area in order to support a widespread use of configurable FPGA-based computing.

Focusing again on the spin glass arena, there is still room for substantial progress. Nowadays, the theoretical analysis of temperature-cycling experiments is still in its infancy. Janus has made possible an in depth investigation of isothermal aging (i.e., experiments where the working temperature is kept constant). However, isothermal aging reflects only a minor part of the experimental work, where different temperature variation protocols are used as a rich probe of the spin-glass phase.

Janus is not able to support these analyses, as its performance is not enough in this case, and also because memory limits would quickly become a major problem, as the coherence length grows very fast close to the critical temperature. If one wants to work in this direction a new generation Janus system should be developed; this can be done by leveraging on technology progress of FPGAs in the last 5 years and introducing a few limited architectural changes in the memory structure of the SPs and in the interconnection harness with the host system.

If this system is developed, we should be able to reach the same time scales of 10^{11} lattice sweeps, which is roughly equivalent to a tenth of a second, on systems containing some 5×10^7 spins. In other words, we should be able to simulate systems with lattice size up to $L = 400$, large enough to accommodate a coherence length of up to 50 lattice spacings. After 40 years of investigations, a direct comparison between experiments and the Edwards-Anderson model will finally be possible.

Acknowledgments

We wish to thank several past members of the Janus Collaboration, F. Belletti, M. Cotallo, G. Poli, D. Sciretti and J.L. Velasco, for their important contributions to the project. Over the years, the Janus project has been supported by the EU (FEDER funds, No. UNZA05-33-003, MEC-DGA, Spain), by the MICINN (Spain) (contracts FIS2006-08533, FIS2009-12648, FIS2007-60977, FIS2010-16587, FPA2004-02602, TEC2010-19207), by CAM(Spain), by the Junta de Extremadura (GR10158), by UCM-Banco Santander (GR32/10-A/910383), by the Universidad de Extremadura (ACCVII-08) and by the Microsoft Prize 2007. We thank ETHlab for their technical help. E.M. was supported by the DREAM SEED project and by the Computational Platform of IIT (Italy); M.B.-J. and B.S. were supported by the FPU program (Ministerio de Educacion, Spain); R.A.B. and J.M.-G. were supported by the FPI program (Diputacion de Aragon, Spain); nally J.M.G.-N. was supported by the FPI program (Ministerio de Ciencia e Innovacion, Spain).

References

1. See, for instance: C. A. Angell, Science **267**, (1995) 1924; P. G. Debenedetti, *Metastable liquids* (Princeton University Press, Princeton 1997); P. G. Debenedetti and F. H. Stillinger, Nature **410**, (2001) 259.
2. L. C. E. Struick, *Physical Aging in Amorphous Polymers and Other Materials* (Elsevier, Houston, 1978).
3. J. A. Mydosh, *Spin Glasses: an Experimental Introduction* (Taylor and Francis, London, 1993).
4. A. P. Young (editor), *Spin Glasses and Random Fields*, (World Scientific, Singapore, 1998).
5. P. A. Boyle et al., IBM J. of Research and Development **49**, (2005) 351-365.
6. F. Belletti et al., Computing in Science & Engineering **8**, (2006) 18-29.

7. G. Goldrian et al., Computing in Science & Engineering **10**, (2008) 46-54; H. Baier et al., Computer Science - Research and Development **25**, (2010) 149-154.
8. J. Makino et al., *A 1.349 Tflops Simulation of Black Holes in a Galactic Center on GRAPE-6*, *Proceedings of the 2000 ACM/IEEE conference on Supercomputing* (2000) Article n. 43.
9. A. D. Ogielski and D. A. Huse, Phys. Rev. Lett. **56**, (1986) 1298-1301.
10. J. Pech et al., Comp. Phys. Comm. **106**, (1997) 10-20; A. Cruz et al., Comp. Phys. Comm. **133**, (2001) 165-176.
11. S. F. Edwards and P. W. Anderson, J. Phys. F: Metal Phys. **5**, (1975) 965-974; *ibid.* **6**, (1976) 1927-1937.
12. J. Barahona., J. Phys. A: Math. Gen. **15**, (1982) 3241-3253.
13. M. Mézard, G. Parisi and M. Virasoro, *Spin-Glass Theory and Beyond*, (World Scientific, Singapore, 1987).
14. K. Gunnarsson et al., Phys. Rev. B **43**, (1991) 8199-8203. See also P. Norblad and P. Svendlidh *Experiments on Spin-Glasses* in [4].
15. H. G. Ballesteros et al., Phys. Rev. B **62**, (2000) 14237-14245.
16. F. Bert et al., Phys. Rev. Lett. **92**, (2004) 167203.
17. See for instance D. J. Amit and V. Martin-Mayor, *Field Theory, the Renormalization Group and Critical Phenomena*, (World Scientific, Singapore, 3rd edition, 2005).
18. M. E. J. Newman and G. Barkema *Monte Carlo Methods in Statistical Physics* (Oxford University Press, 1999).
19. H. Hukushima, K. Nemoto, J. Phys. Soc. Japan **65**, (1996) 1604; E. Marinari in *Advances in Computer Simulation*, J. Kerstéz, I. Kondor (eds.) (Springer-Verlag, 1998).
20. H.G. Katzgraber, *Introduction to Monte Carlo methods*, lecture at *Modern Computation Science*, (Oldenburg, 2009).
21. F. Belletti et al., Comp. Phys. Comm. **178**, (2008) 208-216.
22. F. Belletti et al., *IANUS: Scientific Computing on an FPGA-based Architecture*, in *Proceedings of ParCo2007, Parallel Computing: Architectures, Algorithms and Applications* (NIC Series Vol. 38, 2007) 553-560.
23. F. Belletti et al., Computing in Science & Engineering **8**, (2006) 41-49.
24. F. Belletti et al., Computing in Science & Engineering **11**, (2009) 48-58.
25. S. Sumimoto et al., *The design and evaluation of high performance communication using a Gigabit Ethernet*, proceedings of the 13th international conference on Supercomputing, (1999) 260-267.
26. R. Baxter et al, *Maxwell - a 64 FPGA Supercomputer*, Second NASA/ESA Conference on Adaptive Hardware and Systems, (2007) 287-294.
27. M. Flynn et al., *Finding Speedup in Parallel Processors*, International Symposium on Parallel and Distributed Computing ISPDC '08, (2008) 3-7.
28. V. Parisi, cited in G. Parisi and F. Rapuano, Phys. Lett. B **157**, (1985) 301-302.
29. F. Belletti et al., Phys. Rev. Lett. **101**, (2008) 157201.
30. F. Belletti et al., J. Stat. Phys. **135**, (2009) 1121-1158.
31. R. Alvarez Baños et al., J. Stat. Mech. (2010) P06026.
32. R. A. Baños et al., Phys. Rev. B **84**, (2011) 174209.
33. A. Billoire et al., J. Stat. Mech (2011) P10019.
34. R. Alvarez Baños et al., Phys. Rev. Lett. **105**, (2010) 177202.
35. A. Cruz et al. Phys. Rev. B **79**, (2009) 184408.
36. R. Alvarez Baños et al., J. Stat. Mech. (2010) P05002.
37. R. A. Baños et al. Proc. Natl. Acad. Sci. USA (2012) 109, 6452-6456
38. M. Guidetti et al., *Spin Glass Monte Carlo Simulations on the Cell Broadband Engine* in *Proc. of PPAM09*, (Lecture Notes on Computer Science (LNCS) 6067, Springer 2010) 467-476.
39. M. Guidetti et al., *Monte Carlo Simulations of Spin Systems on Multi-core Processors* (Lecture Notes on Computer Science (LNCS) 7133 K. Jonasson (ed.), Springer, Heidelberg 2010) 220-230.
40. M. Baity-Jesi et al. (Janus Collaboration), European Physical Journal - Special Topics (in press) arXiv:1204.4134.