Spin glass simulations on the Janus architecture: a desperate quest for strong scaling

M. Baity-Jesi[1,2], R.A. Baños[2,3], A. Cruz[2,3], L.A. Fernandez[1,2], J.M. Gil-Narvion[2], A. Gordillo-Guerrero[7], M. Guidetti[2], D. Iñiguez[2,8], A. Maiorano[9,2], F. Mantovani[5], E. Marinari[9,10], V. Martin-Mayor[1,2], J. Monforte-Garcia[2,3], A. Muñoz-Sudupe[1], D. Navarro[3], G. Parisi[9,10], S. Perez-Gaviro[2], M. Pivanti[9], F. Ricci-Tersenghi[9,10], J. Ruiz-Lorenzo[7,2], S.F. Schifano[6], B. Seoane[1,2], A. Tarancon[2,3], P. Tellez[3], R. Tripiccione[6], and D. Yllanes[9,2]

[1] Universidad Complutense, Madrid, Spain
[2] Instituto BIFI, Zaragoza, Spain
[3] Universidad de Zaragoza, Zaragoza, Spain
[4] Università di Tor Vergata and INFN, Roma, Italy
[5] Universität Regensburg, Regensburg, Germany
[6] Università di Ferrara and INFN, Ferrara, Italy
[7] Universidad de Extremadura, Badajoz/Cáceres, Spain
[8] Fundacion ARAID, Zaragoza, Spain
[9] Università La Sapienza, Roma, Italy
[10] IPCF-CNR and INFN

Abstract. We describe Janus, an application-driven architecture for Monte Carlo simulations of spin glasses. Janus is a massively parallel architecture, based on reconfigurable FPGA nodes; it offers two orders of magnitude better performance than commodity systems for spin glass applications. The first generation Janus machine has been operational since early 2008; we are currently developing a new generation, that will be on line in early 2013. In this paper we present the Janus architecture, describe both implementations and compare their performances with those of commodity systems.

1 Introduction

A major challenge in condensed-matter physics is the understanding of glassy behavior. Glasses are materials with strong industrial relevance (aviation, pharmaceuticals, automotive, etc.) that do not reach thermal equilibrium in human lifetimes; a quantitative understanding of their behavior is still an open problem.

Spin glasses (SG) [1] are a widely studied category of prototypical glassy models. Spin variables, taking a small set of discrete values (e.g. just two , ± 1) sit at the nodes of a regular D-dimensional lattice, and tend to align (or anti-align) with their neighbors according to the sign of a coupling constant associated to the link connecting neighbor sites.

The deceivingly simple rules (see later) governing the evolution of each spin, translate into complex collective dynamics as soon as the lattice has even a fairly small number of sites (e.g., ≥ 1000 sites). A striking feature of glasses – and of SG models – is that at low enough temperatures (below the *critical temperature*, T_c) important material properties, such as the compliance modulus or specific heat, depend on time even if the sample is kept for months (years) at constant experimental conditions.

If one quickly cools a sample below T_c, glassy domains grow in the system; their size defines a time-dependent coherence length $\xi(t)$; $\xi(t)$ remains very small (e.g., just a few tens of lattice spacings) even for macroscopic times. Good news is that lattices whose size is just a small multiple of $\xi(t)$ reproduce the experimental evolution, so numerical simulations on computationally affordable lattices are an accurate investigation tool. Bad news is that long simulations are necessary. One usually studies SG with Monte Carlo techniques; one Monte Carlo step (the trial of one spin-reversal on all nodes of the lattice), roughly relates to the average spin-flip time in a real sample, $\simeq 1$ ps; real experiments span a time scale of seconds, that is $\geq 10^{12}$ complete lattice updates in the simulated dynamics. If one wants to simulate $\simeq 100$ independent samples of a lattice with 100^3 sites for 1 (equivalent) second the computational burden quickly goes to 10^{20} spin-flip trials. One may also want to study spin glass properties at equilibrium, for which $\xi(t)$ is roughly the size of the system; this is only possible today for small lattices (e.g. of linear size ≤ 32); thermalization may require in this case 10^{12} Monte Carlo steps on thousands of independent samples.

We see that SG simulation is a computational grand challenge. Luckily enough, SG are easy to treat numerically as their computational algorithms (see Section 2) offer a huge amount of easily identified parallelism; however, the size of our simulated system is (and must remain) of limited size: as more computing resources become available, they must speed up the simulation of a problem of (almost) fixed size; we are in a regime governed by Amdahl's law, as opposed to exploiting larger resources to tackle larger problems (where Gustafson's law applies). If one uses standard parallel systems this means that *strong* performance scaling is crucial: this is why an application-driven system is appropriate.

This paper describes *Janus*, a reconfigurable architecture carefully optimized for Monte Carlo simulations of SG. The first generation Janus machine, deployed in early 2008, outperformed standard computing systems by two orders of magnitude; after four years it still retains a non-negligible performance edge. We are now developing a new generation – *Janus2* – planned for early 2013; again we expect performances $\mathcal{O}(100)$ better than commercial systems.

This paper is organized as follows: we first describe the SG models we are interested in, and the Monte Carlo techniques used to investigate them; we then describe the Janus architecture, highlighting the features that make it a very efficient SG number-cruncher. We then discuss the first Janus version and provide details on the new implementation, Janus2. Next, we compare (measured) Janus and (expected) Janus2 performance with other options available over the lifetime of the project. Our conclusions and outlooks end our text.

2 Spin Glass models and their simulation

We start by introducing a widely studied SG model and its associated computational algorithms, highlighting their architectural implications.

The three-dimensional Edwards-Anderson model [2] is a popular SG model, defined on a lattice of L^3 sites; it is easily described in terms of the energy of the system:

$$E = -\sum_{\langle ij \rangle} \sigma_i J_{ij} \sigma_j; \tag{1}$$

σ_i are L^3 spin variables (modeling the magnetic moments at atomic sites); they take values ± 1 and sit at the nodes of the lattice; the sum spans all pairs of nearest neighbors in the lattice. J_{ij} are the strengths of the interaction (couplings) along the edges connecting nearest-neighbor sites; $J_{ij} > 0$ favors alignment of the corresponding spins, a negative value favors misalignment. J_{ij} values are usually extracted from a distribution with zero mean and unit variance; the simplest case is that $J_{ij} = \pm 1$ with equal probability (binary model). A given assignment of $3L^3$ J_{ij} defines a *sample* of the system;

The *local energy* of a spin at site k is $\epsilon(\sigma_k) = -\sigma_k \phi_k$; ϕ_k depends on the value of the neighbor spins:

$$\phi_k = \sum_{j=k\pm x, k\pm y, k\pm z} J_{kj} \sigma_j; \tag{2}$$

for a given configuration of neighbors, $\epsilon(\sigma_k)$ is two valued, $\epsilon(\pm 1) = \mp \phi_k$. With the assumption that σ_k is at equilibrium with its neighbors at a given temperature T, the probabilities that $\sigma_k = \pm 1$ are given by the Boltzmann-Gibbs distribution,

$$P(\sigma_k = 1) = \frac{\exp\left[\beta \phi_k\right]}{\exp\left[\beta \phi_k\right] + \exp\left[-\beta \phi_k\right]}. \tag{3}$$

$\beta = 1/T$ is the *inverse temperature* (in units such that the Boltzmann constant K_B equals 1). This defines the *Heat-Bath* Monte Carlo algorithm (see ref. [3] for a review on Monte Carlo algorithms): we may decide if the spin σ_k is up or down by comparing the probability, Eq. (3), with a (pseudo-)random number ρ, uniformly extracted in $[0, 1)$.

The simulation of a sample starts from an arbitrary initial configuration (usually chosen by randomly assigning ± 1 to all spins) and proceeds as follows:

1. begin a trial spin-flip: pick a site k at random, with uniform probability;
2. compute ϕ_k, Eq. (2), and the spin up probability $P(+1)$, Eq. (3);
3. pick a uniformly distributed pseudo-random number $0 \leq \rho < 1$;
4. if $\rho < P(+1)$, then $\sigma_k = +1$, otherwise $\sigma_k = -1$; end of the trial spin-flip;
5. repeat as many times as needed;

One easily maps spin values to bits by setting $\sigma_k \to S_k = (1 + \sigma_k)/2$ and applying similar transformations for $J_{ij} \to \hat{J}_{ij}$ and $\phi_k \to F_k$; most of the processing

then reduces to logical (as opposed to arithmetic) operations on bits. F_k takes only seven integer values in $[0 \dots 6]$ so the values of $P(+1)$ can be precomputed and stored in a small look-up table, addressed by F_k.

A Monte Carlo step (MCS) is defined as a number of trial spin-flips equal to the number of sites in the lattice; one such step relates to the average spin-flip time for real systems, as discussed in the introduction. Each MCS produces a new spin configuration on the lattice; one can show that the sampled configurations asymptotically follow the Boltzmann-Gibbs distribution (although, in practice the number of required MCS may be far larger than possible). One also shows (see e.g., [3]) that, for a large number of MCS, the statistical properties of the observables (averages over all sites and over MCS) do not depend on the order in which the algorithm visits each lattice sites; one then adopts a fixed sequence, so each site is visited once and only once in each MCS, and always in the same order. This makes it easy to efficiently exploit available parallelism, as we will see shortly.

The algorithms described above are extremely compute-friendly:

- there are only two critical kernels in the computation, the assignment of a new value to σ_k and the correlated generation of one random number;
- both kernels are based on a small number of logical and arithmetic operations on a small set of variables;
- the main computational structure, when repeated over the whole lattice, gives rise to regular loops performing the same set of operation on a regularly structured data base, whose elements are stored in memory in regular patterns that do not depend on the computation itself;
- Control is simple and memory addressing is regular: simple state-machines are enough for both purposes;
- there is a huge amount of available internal parallelism; this is most easily unveiled, by considering a checkerboard decomposition of the lattice: the neighbors of all black sites are white, so each of the two subsets can in principle by operated upon at the same time;
- there is an additional level of available parallelism – external parallelism – associated to the fact that one wants to accumulate statistics on several unrelated samples, or to perform simulations at many different temperatures.

The last point is easily exploited by farming; previous points define the challenge for an efficient SG engine: extract the largest possible amount of available parallelism; The ideal SG machine can be seen as a large collection of identical cores, that perform efficiently the small set of needed logical and arithmetic operations; a single control structure drives all cores; they work concurrently, performing the same thread at the same time. Each core is a slim object, of just $\simeq 1000$ logical gates, so thousands of them can be easily deployed.

Another way to look at the ideal SG machine is to view it as an *application specific* GPU, with data paths tailored to the specific sequence of logical operations, a control structure shared by a number of cores larger than in state-of-the-art GPUs, variables allocated on on-chip memory and a memory controller optimized for the access patterns required by the chosen algorithm.

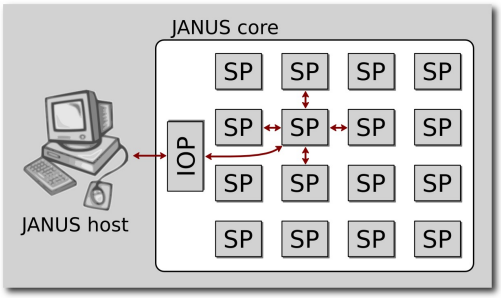

Fig. 1. Conceptual architecture of a Janus system, including the cluster of Simulation Processors (SPs), the Input-Output Processor (IOP) and the host PC.

3 Janus Architecture

As pointed out above, an efficient SG machine is a large collection of *spin-processors*. The spin-processor is a small computing element, tailored to the required mix of operations on bit-valued variables, and to random number generation. Currently available commodity processors are at large variance with these requirements, mainly because their architectures are optimized for long data words. This makes the logic complexity of each data path much larger than needed in SG simulations, and also makes it hard to map efficiently spins onto machine words.

Over the years, *Field Programmable Gate Arrays* (FPGA) have steadily become more powerful and flexible enough to become key building blocks for an SG-computer. FPGAs are reconfigurable at gate level, and the amount of available resources in state-of-the-art FPGAs is enough to accommodate a large cluster of spin-processors. Last but not least, storing the lattice on on-chip memory makes it possible to provide the large memory bandwidth needed to keep all spin-processors at work.

The available parallelism of SG applications is larger than can be exploited by one FPGA: a set of FPGAs can be connected together by a simple network, e.g. a first neighbor torus, and one can process SG samples on this multi-node structure.

A host computer is connected to this structure: contrary to several recent proposed reconfigurable machines [4–6], the host has a minor role, and it is only needed to initialize and start the FPGA array, and to collect simulation results; relatively low bandwidth and long latency are acceptable in our case.

The architecture of the Janus machine [8–10] follows these guidelines; figure 1 shows the Janus concept. Janus is a cluster of computing elements called SP and based on FPGAs; the smallest Janus block is based on 16 FPGAs mounted on a processing-board; our largest installation has 16 processing-boards mounted in a standard 19" rack and controlled by 8 host PCs. Each processing-board hosts a communication hub (the IOP, also using an FPGA) between the SPs

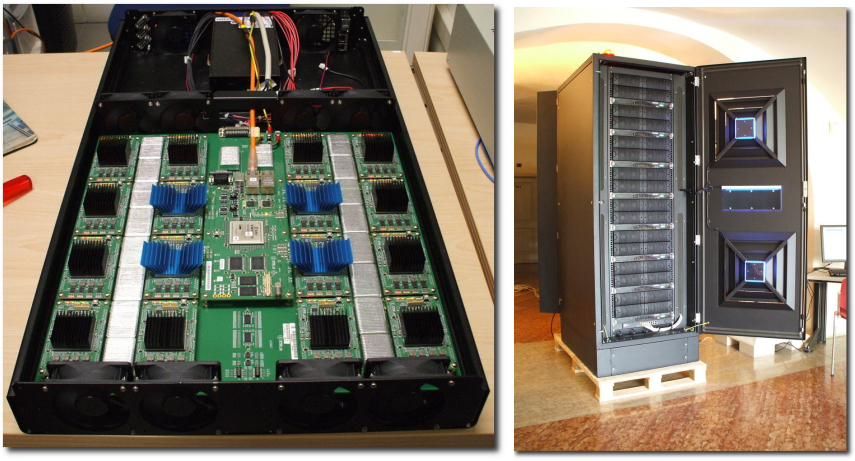

Fig. 2. Left: a Janus processing-board with 16 SPs; Right: a large Janus system with 16 processing-boards, 256 SPs and 8 host PCs.

and the host PC; communication is based on a gigabit Ethernet interface. The IOP handles input-output operations between the SPs and the host PC, and acts as a supervisor for all SPs.

Janus uses Xilinx Virtex-4 LX200 FPGAs, the largest available elements of a state-of-the-art FPGA family when the system was built. The main advantage of FPGAs is that they allow a quick and low cost development cycle; a further step would be to develop an hardwired application-specific processor, as done, for instance, in the Anton project [7]. The main clock is fixed once for all at a conservative 62.5 MHz. This choice reduces the effort needed to successfully map on the FPGA many different and quickly evolving versions of our codes, associated to various possible choices for the physics model and the simulation algorithm.

Figure 2 is a picture of the system. Janus was developed in 2006-2007; it helped obtain significant physics results, as early as mid 2008 [11]; a recent summary is presented in [12].

User applications running on Janus are made of two parts: a firmware module, that defines the operation of the SPs, and a C program running on the host PC; the C program performs data initialization, data input/output and supervises the operations of the SPs. Firmware modules are written in VHDL and compiled by the appropriate Xilinx tools; FPGAs on the SPs are configured on a run-by-run basis, at the start of each application run. Using VHDL, our heaviest SG applications implement up to 1024 spin-processors on each FPGA (and a matching number of random number generators; we use throughout the simple but very efficient Parisi-Rapuano generator [13]); they use approximately 95% of the resources of the FPGA; the actual data-path is very complex and criti-

cally exploits the large number of on-chip memory blocks, providing the needed memory bandwidth; for full details, see [9, 12].

A more user-friendly programming environment is not a goal of the Janus project; some limited experimentation with high level FPGA programming tools ended with only $\leq 10\%$ of the performance of VHDL codes. This scenario – acceptable for JANUS, as only a small set of applications, each running for weeks or months, is needed – is a key problem for more general purpose FPGA machines, like Novo-G [6], whose architecture is not too different from Janus.

Looking again at (strong) performance scaling, each Janus SP is an ideal device, as performance scales linearly as more cores are added within each FPGA. This trend ends as we try to map one physical lattice over several SPs, as the network bandwidth ($\simeq$ 1Gbit/s per link) is not enough (by a factor 3 if we map an 80^3 lattice on a full Janus board).

4 Janus2

We have recently started the development of a new Janus generation, that we call *Janus2*. Its architecture follows the approach described above, with several changes made possible by recent advances in FPGA technology. We plan the following architectural and technology improvements:

- we make the torus network 3D, so one lattice sample can be mapped onto a larger number of SPs; this allows faster processing of a given lattice size, but also simulations of much larger systems;
- we add fast DDR3 memory to each SP node; this is not necessary for SG simulations; however with this improvement Janus2 becomes a more flexible reconfigurable system; for instance, we are already considering graph-coloring algorithms for which the old Janus was poorly suited as enough memory was not available;
- we place the host CPU closer to the SP array, using a PCI-Express (PCIe) interface; this increases bandwidth by a factor 40 and decreases latency to $\approx 1\mu$s, allowing a much closer control of the SP array by the host;
- the IOP module is directly connected to the host PC through an 8X PCI-Express bus; the host PC is a Computer-On-Module (COM) system, directly plugged onto the processing-board;
- we use the VX485T device of the latest Xilinx Virtex 7 FPGA family. This more than doubles the complexity that can be mapped, and preliminary synthesis have verified that the clock frequency of our codes can be increased by a factor 4X; all in all we expect that each Janus2 SP will be 8 times faster than Janus.

As before 16 SPs will be installed on a mother-board, arranged on the edges of a $4 \times 4 \times 1$ 3D-grid. All links of the torus network use high speed serial links directly available on the FPGA. We plan ≥ 20 Gbit/sec for the on-board links and ≥ 5 Gbit/sec for the links in the Z-directions (that run on cables across multiple processing-boards). These figures match the bandwidth requirement of

L^3	Core 2 Duo	CBE (16 cores)	Janus	Tesla C1060	NH (8 cores)	SB (16 cores)	Janus 2
	2007	2007	2008	2009	2009	2012	2013
64^3	1000 ps	150 ps	16 ps	720 ps	200 ps	60 ps	2 ps

Table 1. Spin-update-time (SUT) of EA simulation codes on a 64^3 lattice on several architectures. CBE is a system based on the IBM Cell processor; Tesla C1060 is an NVIDIA GP-GPU with 448 cores; NH (SB) are dual-socket systems based respectively on the 4-core Nehalem Xeon-5560 (8-core Sandybridge Xeon-E5-2680) processors.

linear scaling for a lattice of 160^3 points, that we plan to parallelize on all 16 Janus2 SPs on a board; so, we expect an overall *strong scaling* performance increase of a factor ≥ 100. We will also consider much larger lattices, for which a real 3D system with several processing-boards will be used. We expect working prototypes of this system in late summer, this year.

5 Janus Performances

In this section we analyze Janus performances, comparing them with conventional systems based on recently developed multi- and many-core CPU architectures.

We first consider conventional performance figures: a reference Monte Carlo algorithm for the simulation of the EA model performs the following operations for each lattice site:

- generation of 1 random number; using the Parisi-Rapuano generator this steps performs 1 32-bit sum and 1 32-bit XOR;
- computation of the local field; 6 3-bit XORs and 5 3-bit sums;
- test of the Heat-Bath probability: one 32-bit comparison.

Conservatively equating the 6 short xor and sum operations to one 32-bit integer operation, we end up with 6 equivalent operations for spin update. On Janus each SP has 1000 62.5 MHz spin-processors; each SP then delivers $\simeq 375$ Giga-ops (that is $\simeq 96$ Tera-ops for our large system); Janus has also been a very energy efficient machine, at $\simeq 10$ Giga-ops/W (for comparison, the top entry of the Green500 list in summer 2008 had $\simeq 500$ Mflops/W).

We now consider a performance metrics relevant to the physics user; the *System spin Update Time* (SUT) is the average time needed to update *one* spin of *one* lattice. For each SP in Janus SUT is 16 ps, and we estimate that it will decrease to 2 ps for Janus2; for one full Janus2 processing board working on one lattice, SUT goes down to 0.125 ps.

When comparing with standard computers, one also introduces a *Global spin Update Time* (GUT),appropriate when one simulation job handles several samples of the lattice at the same time; GUT is the SUT value divided by the number of lattices simulate in parallel. This slightly awkward definition is appropriate, since one has been forced, when using traditional CPUs, to combine the variables of several independent samples into the bits of one processor word

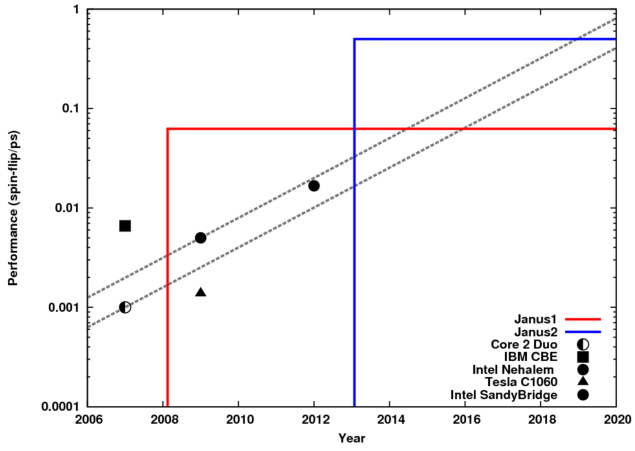

Fig. 3. EA performance (spin-flips/picosec) for optimized programs on several architectures as a function of time. The dotted grey lines scale according to Moore's law. See the text for more details.

and update those samples in parallel, in order to boost overall performance; this trick, usually called multi-spin coding improves the amount of statistically relevant information made available by a run in a given wall-clock time, but does not improve on the time needed to perform a given number of MCS.

When the Janus project started, early 2006, state-of-the-art commodity systems had dual-core CPUs; on those processors carefully optimized codes had a SUT of $\simeq 1000$ ps and GUT of $\simeq 400$ ps. In the following years, processors have changed significantly with the introduction of many-core CPUs and of general purpose GPUs; these are better SG machines that traditional CPUs as one maps the available parallelism on more cores (or on more threads, for GPUs).

Over the years, we have compared [14, 15] Janus with several multi-core systems. A summary of results is collected in Table 1. We clearly see that over the years the large performance gap of Janus has been significantly eroded; an interesting first example was the extremely efficient IBM Cell CPU, for which SUT is 150 ps. In April 2012 the best figure is offered by a 16 cores Sandybridge-based system, for which SUT is ≈ 60 ps.

6 Conclusions and Outlook

We have described two generation Janus machines, that we have developed for Spin Glass applications, and analyzed their measured and expected performances.

An obvious question when developing a custom system is how long it will keep its performance edge over commercial systems. An educated guess to this

question for Janus2 comes from Figure 3 that graphically presents part of the data of Table 1. A reasonably clean pattern emerges:

- while commercial machines increase in performance over the time in a regular way, application-specific projects imply a sequence of step functions over time, as there is no performance gain till a new generation is available.
- Intel processor performance has grown faster that expected by Moore's law.
- we interpret this fact as the consequence of a performance gap that happened when multi-core processors were introduced, followed by a regular Moore's behavior (compare the two Moore's lines in the picture).
- pending new architectural changes, Janus2 should remain a competitive simulation engine at least up to the year 2017.
- a by-product of our analysis shows the poor performance of GPUs for this problem, as well as the outstanding performance of the IBM-Cell processor, whose production has however been discontinued.

Acknowledgments

Janus was supported by the EU (FEDER funds, UNZA05-33-003, MEC-DGA, Spain), by MEC (Spain) (FIS2006-08533, FIS2007-60977, FIS2010-16587, FIS2009-12648-C03, FPA2004-02602, TEC2007-64188, TEC2010-19207), by CAM (Spain), by Junta de Extremadura (GR10158) and by the Microsoft Prize 2007. M. Pivanti was supported by ERC grant agreement N.247328.

References

1. M. Mézard, G. Parisi and M.A. Virasoro, *Spin Glass Theory and Beyond* (World Scientific, Singapore, 1987); A. P. Young (editor), *Spin Glasses and Random Fields*, (World Scientific, Singapore, 1998).
2. S. F. Edwards and P. W. Anderson, J. Phys. F: Metal Phys. **5**, (1975) 965-974; *ibid.* **6**, (1976) 1927-1937.
3. A.D. Sokal, *Functional Integration: Basics and Applications (1996 Cargèse School)* ed. C. DeWitt-Morette, P. Cartier and A. Folacci (1997 New York: Plenum)
4. R. Baxter et al, *Maxwell - a 64 FPGA Supercomputer*, Second NASA/ESA Conference on Adaptive Hardware and Systems, (2007) 287-294.
5. M. Flynn et al., *Finding Speedup in Parallel Processors*, International Symposium on Parallel and Distributed Computing ISPDC '08, (2008) 3-7.
6. A. George, H. Lam, and G. Stitt, Computing in Science & Engineering **13** (2011) 82-86
7. D. E. Shaw, et al., Comm. ACM **51** (2008) 91-97.
8. F. Belletti et al., Computing in Science & Engineering **8** (2006) 41-49.
9. F. Belletti et al., Comp. Phys. Comm. **178**, (2008) 208-216.
10. F. Belletti et al., Computing in Science & Engineering, **11** (2009) 48-58.
11. F. Belletti et al., Phs. Rev. Lett. **101** (2008) 157201.
12. M. Baity-Jesi, et al,, Eur. Phys. J. Special Topics **210** (2012) 33-51.
13. V. Parisi, G. Parisi, F. Rapuano, Phys. Lett. B **157** (1985) 301.
14. M. Guidetti et al., *Spin Glass Monte Carlo Simulations on the Cell Broadband Engine* in *Proc. of PPAM09*, LNCS 6067, 467-476 (Springer, Heidelberg 2010).
15. M. Guidetti et al., *Monte Carlo Simulations of Spin Systems on Multi-core Processors* (K. Jonasson ed.), LNCS 7133, 220-230 (Springer, Heidelberg 2010) .