John von Neumann Institute for Computing

IANUS: Scientific Computing on an FPGA-Based Architecture

F. Belletti, M. Cotallo, A. Cruz, L. A. Fernández,
A. Gordillo, A. Maiorano, F. Mantovani, E. Marinari,
V. Martín-Mayor, A. Munoz-Sudupe, D. Navarro,
S. Pérez-Gaviro, M. Rossi, J. J. Ruiz-Lorenzo,
S. F. Schifano, D. Sciretti, A. Tarancón, R. Tripiccione,
J. L. Velasco

published in

Parallel Computing: Architectures, Algorithms and Applications ,
C. Bischof, M. Bücker, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr,
F. Peters (Eds.),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **38**, ISBN 978-3-9810843-4-4, pp. 553-560, 2007.
Reprinted in: Advances in Parallel Computing, Volume **15**,
ISSN 0927-5452, ISBN 978-1-58603-796-3 (IOS Press), 2008.

http://www.fz-juelich.de/nic-series/volume38

IANUS: Scientific Computing on an FPGA-Based Architecture

Francesco Belletti[1,2], Maria Cotallo[3,4], Andres Cruz[3,4], Luis Antonio Fernández[5,4], Antonio Gordillo[6,4], Andrea Maiorano[1,4], Filippo Mantovani[1,2], Enzo Marinari[7], Victor Martín-Mayor[5,4], Antonio Muñoz-Sudupe[5,4], Denis Navarro[8,9], Sergio Pérez-Gaviro[3,4], Mauro Rossi[10], Juan Jesus Ruiz-Lorenzo[6,4], Sebastiano Fabio Schifano[1,2], Daniele Sciretti[3,4], Alfonso Tarancón[3,4], Raffaele Tripiccione[1,2], and Jose Luis Velasco[3,4]

[1] Dipartimento di Fisica, Università di Ferrara, I-44100 Ferrara (Italy)

[2] INFN, Sezione di Ferrara, I-44100 Ferrara (Italy)

[3] Departamento de Física Teórica, Facultad de Ciencias
Universidad de Zaragoza, 50009 Zaragoza (Spain)

[4] Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), 50009 Zaragoza (Spain)

[5] Departamento de Física Teórica, Facultad de Ciencias Físicas
Universidad Complutense, 28040 Madrid (Spain)

[6] Departamento de Física, Facultad de Ciencia
Universidad de Extremadura, 06071, Badajoz (Spain)

[7] Dipartimento di Fisica, Università di Roma "La Sapienza", I-00100 Roma (Italy)

[8] Departamento de Ingenieria Electrónica y Comunicaciones
Universidad de Zaragoza, CPS, Maria de Luna 1, 50018 Zaragoza (Spain)

[9] Instituto de Investigación en Ingenieria de Aragón (I3A)
Universidad de Zaragoza, Maria de Luna 3, 50018 Zaragoza (Spain)

[10] ETH Lab - Eurotech Group, I-33020 Amaro (Italy)

This paper describes the architecture and FPGA-based implementation of a massively parallel processing system (IANUS), carefully tailored to the computing requirements of a class of simulation problems relevant in statistical physics. We first discuss the system architecture in general and then focus on the configuration of the system for Monte Carlo simulation of spin-glass systems. This is the first large-scale application of the machine, on which IANUS achieves impressive performance. Our architecture uses large-scale on chip parallelism ($\simeq$ 1000 computing cores on each processor) so it is a relevant example in the quickly expanding field of many-core architectures.

1 Overview

Monte Carlo simulations are widely used in several fields of theoretical physics, including, among others, Lattice Gauge Theories, statistical mechanics[1], optimization (such as, for instance, K-sat problems).

In several cases, the simulation of even *small* systems, described by simple dynamic equations requires huge computing resources. Spin systems - discrete systems, whose

variables (that we call *spins*) sit at the vertices of a $D-$dimensional lattice - fall in this category[2]. Reaching statistical equilibrium for a lattice of $48 \times 48 \times 48$ sites requires $10^{12} \ldots 10^{13}$ Monte Carlo steps, to be performed on $O(100)$ copies of the system, corresponding to $\simeq 10^{20}$ spin updates. Careful programming on traditional computers (introducing some amount of SIMD processing) yields an average spin-update time of $\simeq 1$ ns: a typical simulation would use 1 PC for $\simeq 10^4$ years (or $\simeq 10000$ PCs for 1 year, optimistically and unrealistically assuming that perfect scaling takes place!) The structure of simulation algorithms for spin system are however very well suited to some unconventional computer architectures that are clearly emerging at this point in time (see, e.g.[3]).

This paper describes the architecture and the FPGA-based implementation of a massively parallel system carefully tailored for the class of applications discussed above. The system, that we call IANUS, has been developed by a collaboration of the Universities of Ferrara, Roma I and Zaragoza, and of Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), with Eurotech as industrial partner. Our text is structured as follows: Section 2 builds a bridge between the structure of the physical problem and its computing requirements; Section 3 describes our FPGA-based architecture and discusses how we configure it for our application. Section 4 reports on performance and compares with corresponding figures for traditional PC clusters and for at least one recently appeared architecture. Section 5 contains our concluding remarks.

2 Sample Application: Spin Systems on a Lattice

We briefly describe here discrete spin systems, the application that has triggered IANUS development. Consider $N = L \times L \times L$ nodes labeled by $i = \{0, \ldots, N-1\}$ and arranged on a 3D cubic grid. On each node we place a *spin* variable s_i that only takes discrete values $\{+1, -1\}$. An *energy cost* is associated to every pair of *nearest neighbour* sites on the grid (each site has six nearest neighbors, two for each direction x, y and z), which is proportional to the product of the values of the two spins.

$$\epsilon_{ij} = -J_{ij} s_i s_j$$

the proportionality constants J_{ij}, usually called *couplings*, are in general different for each pair of sites, and take the values $+1$ or -1. For a positive coupling $J_{ij} = 1$, the situation in which two neighboring spins are *parallel* (they takes the same value), is energetically favoured. Negative couplings favour *anti-parallel* (mutually opposite) neighboring spins. Periodic boundary conditions are usually applied so the system is a 3D discrete torus. The sum of energy contributions for all pairs of neighboring sites is by definition the energy function of the system.

$$H = \sum_{\langle i,j \rangle} \epsilon_{ij} = -\sum_{\langle i,j \rangle} J_{ij} s_i s_j \tag{2.1}$$

The notation $\langle i, j \rangle$ means that the summation is done only on neighboring sites. The set of all J_{ij} is kept fixed during a simulation (the set is extracted from distribution probabilities that fix the physical properties of the systems described by the model: if $J_{ij} = 1 \; \forall i, j$ we have the Ising model of ferromagnetism, while random equiprobable values $+1, -1$ are considered in the Edwards-Anderson Spin Glass model).

The variables s_i evolve in time according to some specified rules, that we now briefly describe. Given a system with energy function H, one is usually interested in its properties at constant temperature $T = 1/\beta$ (system in equilibrium with a heat source); every configuration of spins $\{S\}$ contributes to thermodynamic properties with a weight factor given by the Boltzmann probability $P \propto \exp(-\beta H[\{S\}])$.

The Monte Carlo Metropolis algorithm is a standard tool to generate configurations according to the Boltzmann probability. As such, it is a key computational algorithm in this field. It may be described by the following steps:

1. choose a site at random in the lattice, and compute the local energy E of the corresponding spin (containing only contributions of nearest neighbour couplings);

2. perform the move: flip the chosen spin and compute the new local energy E' and the energy variation of the move

$$\Delta E = E' - E$$

3. accept the move with probability $P = \min[1, \exp(-\beta \Delta E)]$;

4. go back to 1;

A Monte Carlo *sweep* is usually taken as a number of iterations of the elementary steps described above equal to the volume N of the lattice. Several (in practice, a huge number of) Monte Carlo sweeps are needed in order to reach the equilibrium distribution. It can be shown that the asymptotic behaviour is the same if we choose to visit sites in any given lexicographic order, so a sweep is usually performed by sequentially visiting sites.

Inspection of the algorithm and of equation (2.1) shows that the procedure has several properties relevant for an efficient implementation:

- a large amount of parallelism is available: each site interacts with its nearest neighbors only, so up to one-half of all sites, organized in a checkerboard structure, can be handled in parallel (two neighboring spins may not be updated simultaneously since local transition probability must be well-defined at each step), provided enough random numbers are available;

- the computational kernel has a regular loop structure. At each iteration the same set of operations is performed on data words whose addresses can be computed in advance;

- data processing is associated to bit-manipulation (as opposed to arithmetics performed on long data words), since bit valued variables are involved ;

- the data base associated to the computation is very small, just a few bytes for each site (e.g., $\simeq 100$ KBytes for a grid of 48^3 sites).

In brief, the algorithm is an obvious target for aggressive parallelization, and, for a given budget of logical resources, parallelization can be pursued with greater efficiency if very simple processing elements are available.

3 IANUS Architecture

We try to match the architectural features described in the previous section with an FPGA-based architecture, following earlier attempts[4,5] and leveraging on technology advances. We use latest generation FPGAs, each accommodating several hundreds (see later for details) processing engines. A dramatic memory access bottleneck follows, as all engines process one lattice site each, since several thousands data bits must be moved from memory to the processing engines at each clock cycle. This bandwidth can only be sustained by memory available on-chip that, in current generation FPGAs, is large enough for the data-set.

The system that we have built uses two hierarchical levels of parallelism:

- our system is based on a 4×4 grid of processing elements (that we call SPs) with nearest-neighbour links (and periodic boundary conditions). All SPs are also connected to a so-called IO-Processor (IOP), that merges and moves data to a host computer via 2 Gigabit-Ethernet links. Both the IOP and the SP are implemented by Xilinx Virtex4-LX160 or Virtex4-LX200 FPGAs.
- The SP processor contains uncommitted logic that can be configured at will (fast on the fly reconfiguration of all SPs is handled by the IOP). In our typical application, the SP becomes a *many*-core processor (we use the terminology proposed recently in[3]), each core performing the same algorithm on a subset of the spins (see later for details).

A block diagram of the system is shown in Fig. 1, while a picture of the IANUS prototype is shown in Fig. 2

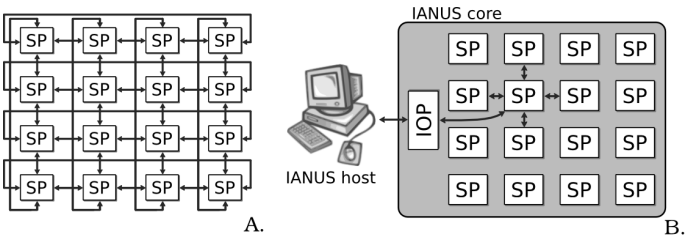

Figure 1. IANUS topology (A) and overview of the IANUS prototype implementation (B), based on a processing board (PB) with 16 SPs and one I/O module (IOP).

The computational architecture that we configure on each SP for the first large-scale IANUS application - the Monte Carlo simulation described above - is shown in Fig. 3.

As a guide to understand the structure, we have a set of processing engines (also called update cells, UC), that receive all the variables and parameters needed to update one spin and perform all needed arithmetic and logic operations, producing updated values of the spin variable. Data (variables and parameters) are kept in memories and fed to the appropriate UC. Updated values are written back to memory, to be used for later updates.

The choice of an appropriate storage structure for data and the provision of appropriate data channels to feed all UCs with the value they need is a complex challenge; designing

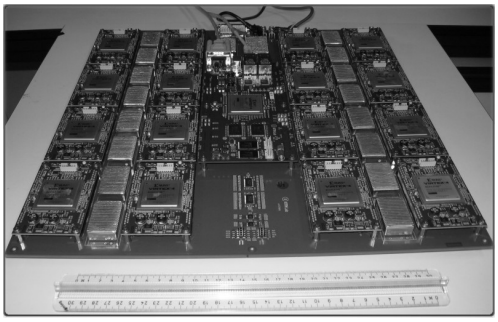

Figure 2. IANUS board prototype: SPs are accommodated in 4 lines of 4 processors each while the I/O processor is placed at the centre.

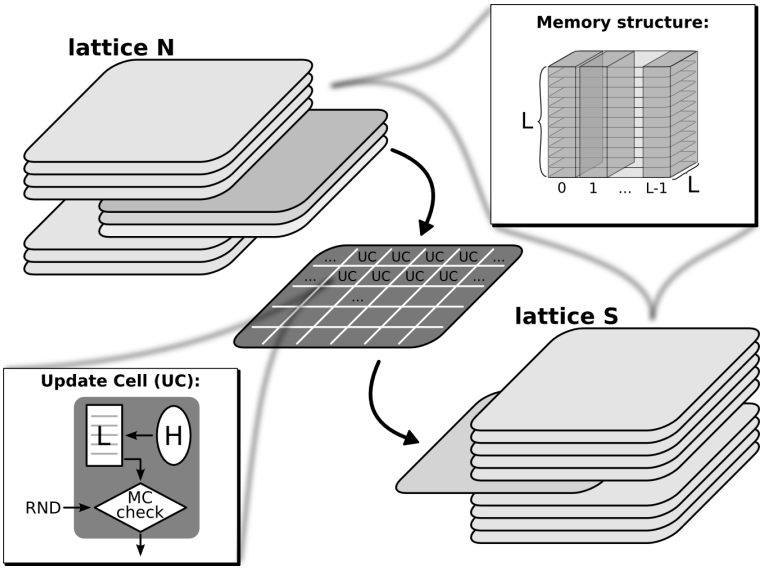

Figure 3. Parallel update scheme. The spins (lattice S) and their neighbors (lattice N) are stored in RAM blocks. All $L \times L$ spins in a plane in the S lattice of vertical coordinate (memory address) z are updated simultaneously. A set of $L \times L$ Update Cells (UC) are implemented in logic; at each clock cycle they receive all needed neighboring spins (the lattice planes of vertical coordinates z, $z-1$, $z+1$). Note that at regime we only need to fetch one plane from the N lattice, as the other two are available from the previous iteration. The outputs of the UCs are the updated spins, stored into the S memories at address z. The structure is slightly more complex than described here as we also need to feed UCs with the coupling values. When all planes in S have been processed, S and N interchange their roles.

the update cells is a comparatively minor task, since these elements perform rather simple logic functions.

We now briefly describe our data structure (the reader may find reference to Fig. 3 helpful to follow the discussion). As pointed out before, we can arrange for parallel update of up to one half of the lattice, processing all *white* (*black*) sites in a checkerboard scheme

at once. In order to maximize performance, we resort to a simple trick, by replicating the system twice: the two *replicas* share the same coupling data base, but follow independent dynamics, doubling the overall statistics. We arrange all white sites of replica 1 and black sites of replica 2 in one lattice (stored in *S*, the *updating spin* block, see Fig. 3) and all black sites of replica 1 and white sites of replica 2 in another lattice (the *neighbors block*, called *N* in Fig. 3). With this arrangement, all spins in the updating block have their neighbors in the neighbors block only. These artificial lattices have the same topology as the original ones, and are mapped directly onto embedded RAM blocks on the FPGA. Consider the simple case of an $L = 16$ system. The updating spin lattice (and of course the neighbors lattice) may be stored in 16 RAM blocks (whose labels are the x coordinates), each 16 bit wide (bit index is the y coordinate) and 16 word deep (memory addresses are z coordinates). We need three similar structures to store the couplings J_{ij} one per each coordinate axis (each site has two links to neighbors in each direction). Fetching one 16 bit word for each RAM block at the same address z corresponds to fetching a full (xy) plane of the updating spin block (or the neighbors or coupling blocks in one direction). This scheme is easily generalized to several values of lattice size L. Of course only fractions of entire planes may be updated for very large lattices (since in this case there are not enough logical resources within the FPGA), but this can be easily managed.

Each update step consumes one (pseudo)-random number. Random number generators use a large fraction of the available hardware resources. We implemented the Parisi-Rapuano shift-register wheel[7], defined by the rules

$$\begin{aligned} I(k) &= I(k-24) + I(k-55) \\ R(k) &= I(k) \otimes I(k-61)\ , \end{aligned} \tag{3.1}$$

where $I(k-24)$, $I(k-55)$ and $I(k-61)$ are elements (32-bit wide in our case) of a shift register initialized with externally generated values. $I(k)$ is the new element of the wheel, and $R(k)$ is the generated pseudo-random value. This generation rule is easily arranged on configurable logic only, so, by exploiting cascade structures, each wheel produces about one hundred values per clock cycle.

After spins are fed to an update cell, the local (integer) energy values are computed and used to address a look-up table whose entries are pre-calculated Boltzmann probability values (normalized to $2^{32}-1$). The probability is compared with the 32 bit random number in order to decide the spin's fate. Look-up tables (LUTs) are implemented in distributed RAM whose availability is one limiting factor in the number of possible parallel updates. On the other side, LUTs are small data sets so arranging them on RAM blocks would be a waste of precious storage. Since two reads are possible from each LUT in one clock cycle, each one is shared between two update cells. A more detailed description can be found in in[8].

We accommodate all memory structures, 512 update cells and a matching number of random generators and LUTs on the XILINX-LX160 FPGA available on the SP. A larger FPGA (the XILINX-LX200 that will be used for the full-scale IANUS system) doubles all values. Independent simulations are carried out on each SP belonging to the IANUS system, since several simulations associated to different parameters of the system are needed anyway. The number of update cells corresponds to the number of updates per clock cycle (512 on the LX160 and 1024 on the LX200). Note that at 1024 updates per clock cycle and 62.5 MHz clock frequency, the random number generator alone runs at 64 Gops (32-bit)

on each node (1 Tops per IANUS board), even neglecting the cost of XOR operations.

In short, we are in the rewarding situation in which: i) the algorithm offers a large degree of allowed parallelism, ii) the processor architecture does not introduce any bottleneck to the actual exploitation of the available parallelism, iii) performance of the actual implementation is only limited by the hardware resources contained in the FPGAs.

4 Performance

The algorithms described above run on our FPGA with a system clock of 62.5 MHz, corresponding to an average update time of $1/(512 \times 62.5 \times 10^6) = 32$ ps/spin (16 ps/spin in the LX200 version, as the number of update cells doubles) per FPGA, that is 2 ps/spin (1 ps/spin) for a full IANUS processing board (16 FPGAs).

It is interesting to compare these figures with those appropriate for a PC. Understanding what exactly has to be compared is not fully trivial. Popular PC codes update in parallel the *same* site of a large number of (up to 128) replicas of the system, each mapped onto one bit of the processor word. The same random number is shared by all replicas. This scheme (we call it Asynchronous Multi Spin Coding, AMSC) extracts reasonable performance from architectures with large word sizes in a context in which the natural variable size is just one bit. This approach is useful for statistical analysis on large samples. On the other hand, it is less appropriate when a *small* number of replicas of a large system must be updated *many* (e.g., $10^{12} \ldots 10^{13}$) times. In this case an algorithm that updates in parallel many spins of the same lattice (exactly what we do in IANUS) is a much better choice. Such codes (that we call Synchronous Multi Spin Coding, SMSC) were proposed several years ago, but never widely used. We put a rather large effort in developing efficient AMSC and SMSC codes on a high-end PC (an Intel Core2Duo - 64 bit - 1.6 GHz processor). A comparison - see Table 1 - shows that one IANUS processing board has a performance of hundreds (or even thousands) of PCs.

Preliminary measurements of these codes ported to the IBM Cell Broadband Engine (CBE) (work is still in progress in this area) show that one CBE (using all 8 synergistic processors) is approximatively $10 - 15$ times faster than an Intel Core 2 Duo PC.

	LX160	LX200	PC (SMSC)	PC (AMSC)
Update Rate	2 ps/spin	1 ps/spin	3000 ps/spin	700 ps/spin

Table 1. Comparing the performances of one IANUS processing board and two different PC codes.

Finally note that the mismatch between processing power available on the system and bandwidth to the host is such that a full simulation run must be executed on IANUS; this is at variance with recent architectures in which FPGA co-processors are directly attached to a traditional CPU and executes relatively fine-grained tasks.

Our planned IANUS installation (16 sub-systems of 16 processors each, expected for fall 2007) has a processing power equivalent to ≥ 10000 PCs or approximately 1000 CBEs, making it possible to carry out the physics program outlined above in about one year time. IANUS will be housed in one standard rack; power dissipation will be $\simeq 4$ KW, a huge improvement with respect to a large PC farm.

5 Conclusions

Our performance are admittedly obtained by carefully handcrafting an appropriate architecture for a parallel-friendly algorithm. This experience teaches, in our opinion, some lessons relevant in a more general context:

- we have put in practice the potential for performance of a *many*-core architecture, exposing all parallelization opportunities available in the application.
- The huge performances that we obtain depend on huge bandwidth to/from memory, only achievable with *on-chip* embedded memories.
- FPGAs can be configured to perform functions poorly done by traditional CPUs, exploiting a large fraction of available resources. This compensates the overheads associated to reconfigurability and inherent to any FPGA structures.

At present we are fine-tuning for IANUS a Monte Carlo procedure for random graph colouring (a prototype K-sat problem), for which we also expect large performance gains with respect to PCs. This code, that we describe elsewhere, will assess the performance potential of FPGA-based processors for algorithms with irregular memory access patterns.

Acknowledgements

IANUS has been partially funded by the UE (FEDER funds) and by Diputación General de Aragón (Spain) and supported by the Spanish MEC (FIS2006-08533 and TEC2004-02545).

References

1. D. P. Landau and K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics*, (Cambridge University Press 2005).
2. See for instance *Spin Glasses and Random Fields*, edited by P. Young, (World Scientific, Singapore, 1998).
3. K. Asanovic et al., *The Landscape of Parallel Computing Research: A View from Berkeley*, Tech. Report UCB/EECS-2006-183, (2006).
4. J. H. Condon and A. T. Ogielski, *Fast special purpose computer for Monte Carlo simulations in statistical physics.*, Rev. Sci. Instruments, **56**, 1691–1696, (1985).
5. A. Cruz, et al., *SUE: A Special Purpose Computer for Spin Glass Models*, Comp. Phys. Comm., **133**, 165, (2001) .
6. F. Belletti et al., *IANUS: An Adaptive FPGA Computer*, Computing in Science and Engineering, **71**, 41, (2006).
7. G. Parisi and F. Rapuano, *Effects of the random number generator on computer simulations*, Phys. Lett. B, **157**, 301–302,(1985).
8. F. Belletti et al., *Simulating spin systems on IANUS, an FPGA-based computer*, `http://arxiv.org/abs/0704.3573`, Comp. Phys. Comm., (2008, in press).