A Reconfigurable Monte-Carlo Clustering Processor (MCCP).

C.P. Cowen and S. Monaghan
Neural and VLSI Systems Group
Department of Electronic Systems Engineering
University of Essex
Colchester, CO4 3SQ England

Abstract

A generic Monte-Carlo problem architecture, implemented earlier in a dedicated FPGA computer, is extended to incorporate a wider range of more complex Monte-Carlo and percolation problems. The new algorithms, which are implemented in the same computer architecture, execute an order of magnitude faster than on comparable DSP hardware, as with the earlier and simpler algorithms.

Introduction

This paper is one of a series [1, 2, 3] concerned with the design and implementation of an inexpensive PC-AT hosted FPGA-based computer (henceforth to be called the processor) which can be configured to implement in hardware a range of Monte Carlo (MC) algorithms currently used in statistical physics. MC simulations offer a good 'laboratory' for investigating the potential of FPGA-based computing machines for several reasons:

1. They are simple enough to be implemented in FPGAs [1].
2. There are a wide range of Monte-Carlo algorithms to exploit the reconfigurability of these machines.
3. MC simulations often require the computational resources of dedicated hardware [4, 5].
4. MC methods are increasingly used in areas other than statistical physics.

In addition, parts of an MC simulation resemble low-level multi-dimensional image processing tasks [2].

The range of MC problems previously implemented in the reconfigurable hardware include (using image processing terminology to describe the character of the MC simulation) : single-bit pixels with local windows with a fixed processing element [1], single-bit pixels with local windows with a pixel dependent processing function [2] and multi-bit pixels with a fixed processing element [3].

In all cases the processing element was based on the classical MC methods [6].

The motivation for this work was to extend the capability of the processor by implementing a class of MC algorithms [7] which are extensively cited in the physics literature and which to our knowledge have not previously been implemented in hardware. As a by-product, the processor can also be used to study the important problem of percolation, which arises in many engineering contexts.

Monte-Carlo Clustering Techniques

Overview

Monte-Carlo (MC) simulations are typically carried out on systems comprising a large number, N, of microscopic particles which interact with their neighbouring particles. For further details see [2, 3] and references therein.

The idea of a Monte-Carlo simulation is to sample the distribution of microscopic states occupied by these N particles to obtain the system's average or macroscopic properties. A good MC simulation uses importance sampling to generate significant and independent samples rapidly [6]. In this respect, the classical MC methods fail for the simulation of some systems under certain conditions, e.g. the simulation of a ferro-magnet at its critical temperature, T_c [8]. Here, the time required to generate statistically independent samples grows in a power-like fashion (critical slowing down) with the size of the system under simulation. This restricts the maximum system size that can be simulated. In recent years, a new class of algorithms have been developed, known as cluster

0-8186-5490-2/94 $03.00

algorithms [7], which to a large extent resolve this difficulty for certain important systems.

This paper will consider only the two-dimensional *Ising Model* [8], where each of the N particles are represented by 1-bit quantities, known as *spins*, and arranged in an $M \times M$ square lattice.

Swendsen-Wang Clustering

The Swendsen-Wang algorithm for the Ising Model [7] was one of the first clustering algorithms to appear. The algorithm consists of the repeated application of two stages, the first of which is the formation of clusters. Figure 1 shows a small example lattice of spins of arbitrary values. During the first stage of the algorithm, the value of each spin in the lattice and those of its nearest neighbouring spins are examined in turn.

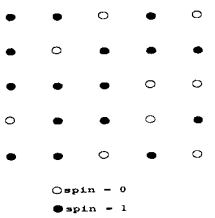

Figure 1: Spin system

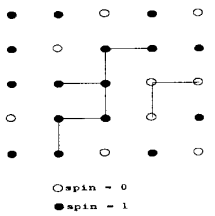

Figure 2: Spin system with clusters

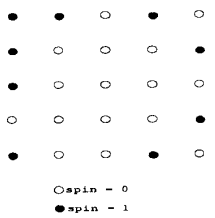

Figure 3: Spin system after clustering

Whenever two neighbouring spins are found to have the same value, a bond (or link) is established between them [7] according to the probability:

$$p = 1 - \exp(-\Delta K) \tag{1}$$

$$\Delta K = \frac{1}{k_B T} \tag{2}$$

where k_B is Boltzmann's constant and T is the system temperature. This construction is related to the well known bond percolation problem.

At the end of this first stage, the lattice will consist of a number of spins bonded together in clusters. In figure 2, two clusters are shown: one consisting of seven spins, the other of three. Any unconnected spins can be considered as clusters with just one spin. The largest possible cluster size contains all the spins in the lattice.

During the second stage of the algorithm, clusters with two or more spins are located and all the spins in the cluster are assigned a common value. Whenever a spin with a bond to another is found, the other spins in this cluster can be found by following all bonds until the cluster has been exhausted. The new value assigned to each of the spins in the cluster is decided with equal probability. Figure 3 shows the state of the lattice after one update. In this example, the larger cluster has been changed, but the spins in the smaller have remained the same. Whether or not the values of the constituent spins of a cluster have changed, the bonds that join them are removed, and the image is ready to be processed again.

For a 64×64 Ising lattice, independent samples can be generated about 25 times [7] faster than with the use of the 'classical' algorithms. For larger systems, the improvement is even greater. Indeed in the problem which originally motivated this research, the lattice size is required to be 1024×1024. This size of lattice is currently used in certain difficult problems associated with Ising Model.

Processor Architecture

The Monte-Carlo Clustering Processor (MCCP) architecture used to implement the Swendsen-Wang algorithm is shown in figure 4. There are two main parts to the processor corresponding to the two stages of the clustering algorithm.

Forming Clusters

The image memory (fig. 4) stores the lattice of spins and the link information. In the two dimensional

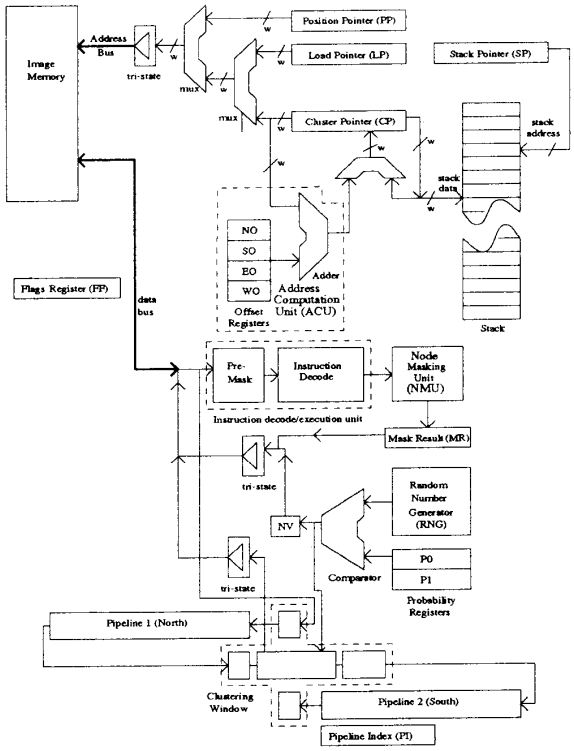

Figure 4: MCCP Architecture

Ising Model, there are four nearest neighbours, north, south, east and west, so the dataword stored for each lattice site looks like figure 5.

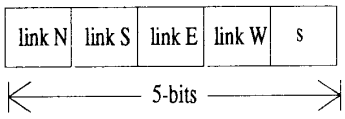

Figure 5: MCCP data representation for 2-dimensional Ising Model.

The first stage of the algorithm is performed sequentially, by passing the entire lattice through a pipeline. This pipelined arrangement and the associated address generators are very similar to that of the processors described in [2, 3]. Two address registers are used in this operation. The Load Pointer fetches the next spin into the pipeline and the Position Pointer writes the processed dataword with link information back to the memory. The clustering window contains a register where the five-bit dataword is formed. During the clustering process, if two neighbouring spins have equal value, a random number is generated and compared with the probability in register *P0*. This register contains the value p given in equation 1. Only north and east links of a given spin need be generated. When the whole image has been processed, a flag instructs the processor to go into the second stage of the algorithm.

Processing the Clusters

The second section of the processor examines the lattice and looks for clusters of spins. Whenever a cluster of spins is found, the cluster is explored by following the links to neighbouring spins. This is a highly recursive process. Since there are four possible links in two dimensions, a stack is provided to store addresses of any spin with links to more than one other spin. *The Cluster Pointer* (CP) is used to address the cluster and can be post-modified by the appropriate amount to allow it to follow the links in any of the four directions as well as being loadable from the stack.

At the core of the MCCP there is a finite-state machine that operates on node (link) information, moving around clusters and masking out the links as it does so. In accordance with the Swendsen-Wang Algorithm, a new value for the spins in each cluster is assigned with a probability of a half, stored in register *P1*. When the end of a cluster is found the processor returns to the point in the lattice where it found this cluster and searches for any others. Thus each cluster is processed in turn by being unlinked and having its spins set to a new value. When all clusters have been processed, the appropriate flag is adjusted and the clustering stage can begin again.

Instruction Set

There are several operations that the finite state machine must carry out, modifying the CP register, masking out links information, setting flags etc. To simplify the design process, these operations have each been given DSP-style instruction mnemonics and these are summarized in table 1. Unlike conventional instruction sets, the MCCP instruction op-codes are formed from a combination of the node information and the machine status, which is governed by the previous instruction and stored in the flags register.

Architectural Parameters

The basic MCCP architecture depicted in figure 4 is scalable to a variety of problem types and sizes. The main factors that influence the actual parameters such as bus and register sizes are the following:

1. The number of spins, N, in the lattice. The width of the lattice will be denoted by M (in the case of the square lattice $N = M^2$)

2. The number of bits representing each spin, B_{bits}. (In the case of the Ising Model the $B_{bits} = 1$).

3. The number of dimensions in the system, d. (In this paper $d = 2$).

4. The number of bits used to represent the generated pseudo-random numbers, R_{bits}.

Table 2 lists the parts of the MCCP architecture and how they are affected by these factors. Note that in figure 4, the internal bus width is denoted by w, where $w = \log_2 N$. It is worth pointing out that the intent is to simulate the relatively large lattice sizes of upto $N = 1024^2$, but in order to test the processor's operation a small, 'toy' 8x8 system was implemented and its parameter values are included in the rightmost column of table 2. The table gives a maximum value for the stack, but in most cases less will be required.

FPGA Implementation

This section considers the arrangement and interconnection of FPGAs and associated memories into which the generic MCCP architecture will be placed. One of the main influencing factors will be pinout cost. As the problem size increases, the bus sizes increase logarithmically (see table 2). The Monte-Carlo processor described in [3] uses the arrangement shown in figure 6, but this is not fully suitable for the 1024^2 problem. An extra memory is required to incorporate the stack if the processor is to operate at its most efficient. Referring to figure 4, it can be seen that the addressing and data processing parts may be separated by placing all the address registers, the address computation unit and the stack controller into one FPGA and the instruction decode, RNGs, comparators, cluster pipeline window and the state machine that provides the clock signals into another. The resulting architecture is shown in figure 7. Bus widths etc are given in table 2.

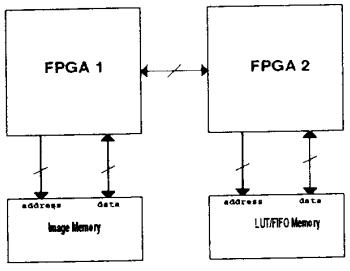

Figure 6: FPGA/memory architecture 1

The highest pin-out cost is for FPGA 1. For the $N = 1024^2$ problem, 60 pins are required to address

FLAGS			Node Data				Instruction Set
CM	DM	SE	N	S	E	W	Instruction Nmemonic
Zero Node Group							
0	1	1	0	0	0	0	CLR DM; (or DM = 0;)
0	1	0	0	0	0	0	POP CP;
Single Node Group							
0	1	X	0	0	0	1	MR = EMASK(CP+EO),(CP) = MR;
0	1	X	0	0	1	0	MR = WMASK(CP+WO),(CP) = MR;
0	1	X	0	1	0	0	MR = NMASK(CP+NO),(CP) = MR;
0	1	X	1	0	0	0	MR = SMASK(CP+SO),(CP) = MR;
Multi-Node Group							
0	1	X	0	0	1	1	PUSH CP,MR = EMASK(CP+WO),(CP) = MR;
0	1	X	0	1	0	1	PUSH CP,MR = SMASK(CP+WO),(CP) = MR;
0	1	X	0	1	1	0	PUSH CP,MR = EMASK(CP+SO),(CP) = MR;
0	1	X	0	1	1	1	PUSH CP,MR = EMASK(CP+WO),(CP) = MR;
0	1	X	1	0	0	1	PUSH CP,MR = NMASK(CP+NO),(CP) = MR;
0	1	X	1	0	1	0	PUSH CP,MR = NMASK(CP+NO),(CP) = MR;
0	1	X	1	0	1	1	PUSH CP,MR = NMASK(CP+NO),(CP) = MR;
0	1	X	1	1	0	0	PUSH CP,MR = NMASK(CP+NO),(CP) = MR;
0	1	X	1	1	0	1	PUSH CP,MR = NMASK(CP+NO),(CP) = MR;
0	1	X	1	1	1	0	PUSH CP,MR = NMASK(CP+NO),(CP) = MR;
0	1	X	1	1	1	1	PUSH CP,MR = NMASK(CP+NO),(CP) = MR;
Cluster Search Mode Group							
0	0	X	0	0	0	0	NV = RNG CMP P1;
0	0	X	X	X	X	1	SET DM; (or DM = 1;)
0	0	X	X	X	1	X	SET DM; (or DM = 1;)
0	0	X	X	1	X	X	SET DM; (or DM = 1;)
0	0	X	1	X	X	X	SET DM; (or DM = 1;)

Table 1: MCCP Instruction Set Summary

Parameter	Value	8x8 example
Register sizes (bits)		
Address Registers (PP, LP, CP)	$\log_2 N$	6
Stack Pointer (SP)	$\log_2 N$	6
Offset Registers	$\log_2 N$	6
Pipeline Index (PI)	$(M-2)$	N/A
Cluster Register (CR)	$2d + B_{bits}$	5
Mask Result Register (MR)	$2d$	4
New Value Register (NV)	B_{bits}	1
Probability Registers (P0, P1)	R_{bits}	8
Bus widths (bits)		
Image Memory Address	$\log_2 N$	6
Image Memory Data	$2d + B_{bits}$	5
Stack Address	$\log_2 N$	6
Stack Data	$\log_2 N$	6
Internal address buses	$\log_2 N$	6
Miscellaneous		
Comparator size (bits)	R_{bits}	8
Stack max. size	$N - 2(M-1)$	6
Pipeline length (total)	$2(M + d + 1)$	22
Ram FIFO sections (size)	$(M-2)$	N/A
Image memory capacity	N	64

Table 2: MCCP architecture scaling parameters.

the stack and the image memory and provide the stack data bus.

The toy problem

An implementation of a 'toy' $N = 8^2$ problem is possible on the board used for previous Monte-Carlo processors, which has the architecture shown in figure 6.

FPGA	CLBs
FPGA1	49
FPGA2	43

Table 3: CLB counts for $N = 8^2$ lattice problem.

Because the pipeline is short enough to be contained entirely within one of the FPGAs, no FIFO memory is needed. SRAM2 can then be used to implement the stack memory and the stack controller placed in FPGA2. The CLB counts for each of these two Xilinx 3030 FPGAs are shown in table 3.

The bus sizes are given in table 2. In addition there are a number of connections between the two FPGAs which include clock signals and control signals from the instruction decoder/sequencer.

Performance

Processor performance for the classical MC algorithms is easy to quantify and is invariably given in terms of million-updates per second. A clustering algorithm is not as straightforward to quantify and although figures can be given in terms of millions-updates per second, a more meaningful quantity is the amount of time taken to process an image of given size and at given temperature.

The first part of the algorithm is a sequential operation and the time taken is given by:

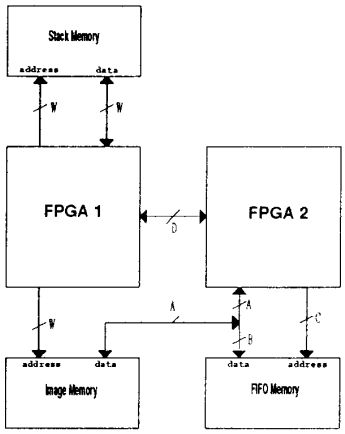

Figure 7: FPGA/memory architecture 2

$$T_{clus} = t_c \; C_{clus} \; N \tag{3}$$

where t_c is the processor's clock period, C_{clus} is the number of clock cycles required to process each spin, and N is the number of spins in the lattice.

The average number of processing steps, $F(\Delta K)$, required by the declustering part of the algorithm is shown for a 40×40 system in figure 8.

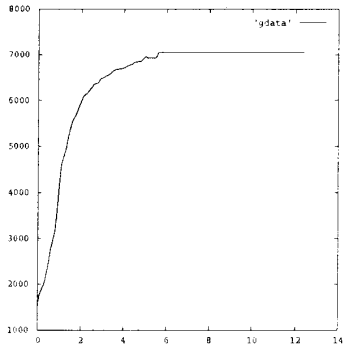

Figure 8: Processing steps, $F(\Delta K)$ vs. Temp for a 40x40 lattice

The asymptotic region of the graph corresponds to a fully linked lattice, where all the spins are linked into one big cluster. Fortunately, values of $\Delta K > 2$ are of little interest since $\Delta K \approx 0.44$ at the point at which the classical Monte-Carlo methods exhibit critical slowing down. At this point, the MCCP is operating at approximately a third of the maximum value of $F(\Delta K)$.

The time taken to decluster and update is therefore given by:

$$T_{declus} = t_c C_{declus} \alpha N F(\Delta K) \tag{4}$$

Total sweep time then is:

$$T_{sweep} = T_{clus} + T_{declus} \tag{5}$$

MCCP versus DSP implementation

To obtain an idea of the MCCP's efficiency, the algorithm was coded in assembly language on a popular Digital Signal Processor (DSP) [9] to do similar tasks.

Processor	N	Temp	$T_{clus}(\mu s)$	$T_{declus}(\mu s)$	$T_{sweep}(\mu s)$
DSP	8^2	$T \to 0$	245.0	721.0	966.0
MCCP	8^2	$T \to 0$	16.9	68.4	85.3
DSP	8^2	$T \approx T_c$	190.0	312.0	502.0
MCCP	8^2	$T \approx T_c$	16.9	39.9	56.8

Table 4: MCCP vs. DSP performance - 8×8 Ising model

Processor	N	Temp	$T_{clus}(s)$	$T_{declus}(s)$	$T_{sweep}(s)$
DSP	1024^2	$T \approx T_c$	1.88	3.36	6.30
MCCP	1024^2	$T \approx T_c$	0.28	0.37	0.65

Table 5: MCCP vs. DSP performance - 1024^2 Ising model

The current implementation of the processor has $C_{clus} = C_{declus} = 4$, and runs from a 15MHz clock giving $t_c \approx 66ns$. The DSP has an instruction cycle time of $80ns$. Table 4 gives a summary of the results obtained for the 8×8 model.

Percolation

In implementing the Swendsen-Wang algorithm, we have also gained the capability to study percolation problems. The declustering part of the MCCP can be used to study the properties of clusters formed, for example to determine if a cluster extends across the entire lattice (percolation). This opens up a number of possible applications of the processor.

Conclusions

It has been shown that the reconfigurable Monte-Carlo processor, described in earlier papers, can accommodate a new class of Monte-Carlo algorithm if the MCCP problem architecture described here is used. This is a necessary step towards developing

a general-purpose Monte-Carlo processor which offers the performance of dedicated hardware to a wide range of Monte-Carlo problems.

It can be seen from the performance analysis that the same order of magnitude performance gain offered by previous MC processors over software is obtained by the clustering processor.

By developing the MC problem architecture (Fig.4) to perform clustering algorithms, the range of applications to which the processor can be put is increased. In particular, it also possible to study percolation problems. In this case, the basic algorithm was implemented in two Xilinx 3000 series FPGAs and three memories, on the Monte-Carlo processor described in this paper. This conclusion should also be true for large FPGA custom computing machines [10, 11].

References

[1] S. Monaghan, T. O'Brien, and P. Noakes. *FPGAs*, page 363. Abingdon CS Books, 1991. Edited by W. Moore and W. Luk.

[2] S. Monaghan. Gate level reconfigurable Monte Carlo processor. *JVSP*, 6:139 - 153, 1993.

[3] S. Monaghan and C.P. Cowen. Multi-bit reconfigurable processor for DSP applications in Statistical Physics. *Proc. FPGA for Custom Computing Machines FCCM'93*, 1993. sponsored by IEEE Computer Society.

[4] R. Pearson, J.L. Richardson, and D Toussaint. A fast processor for monte-carlo simulation. *Journal of Computational Physics*, 51:241–249, 1984.

[5] J. H. Condon and T. Ogielski. Fast special purpose computer for Monte-Carlo simulations in statistical physics. *Rev. Sci. Instrum*, 56(9), 1985.

[6] K. Binder, editor. *Applications of the Monte-Carlo Methods in Statistical Physics.* Springer-Verlag, 1984.

[7] Robert H. Swendsen and Jian-Sheng Wang. Nonuniversal Critical Dynamics in Monte Carlo Simulations. *Physical Review Letters*, 58(2), January 1987.

[8] L.D. Landau and E.M. Lifshitz. *Statistical Physics. Part 1.* Oxford: Pergamon Press, 1980.

[9] Analog Devices. *ADSP-2101 Data Sheet*, 1990.

[10] J.M.Arnold, D.A Buell, and E.G. Davis. Splash 2. *In Proc. 4th Annual ACM Symp. on Parallel Algorithms and Architectures*, pages 316 - 322, 1992.

[11] P. Bertin, D. Roncin, and J Vuillemin. Introduction to Programmable Active Memories. June 1989. Technical Report No3, DEC Paris Research Laboratory.