

The Design of a Specialised Processor for the Simulation of Sintering

Adam Postula* David Abramson† Paul Logothetis†

*Department of Electrical and Computer Engineering, University of Queensland, Brisbane, Australia.
adam@elec.uq.edu.au

† School of Computing and Information Technology, Griffith University, Kessels Rd, Brisbane 4111 Australia.
email: davida@cit.gu.edu.au fax: +61 7 875 5051

ABSTRACT

This paper presents the design decisions encountered during the development of a specialised processor for the Monte-Carlo simulation of metallurgical sintering. Several possible architectures are presented. We show that such a specialised processor using commercially available gate array technology can solve the same problem more than 100 times faster than a modern high-end workstation. Even using slower FPGAs it is possible to achieve a speedup of over 50 times faster than a workstation, which supports the concept of programmable special purpose computers attached to general purpose machines. A prototype of the processor is now being built using Xilinx FPGA and Aptix FPIC switch technology.

1. Introduction

General purpose computers have been optimised to solve a wide range of problems efficiently. However, for certain specific applications, it is possible to obtain much faster and cheaper solutions using specially designed machines. Examples include RSA Cryptography, String Matching, Heat

and Laplace Equation Solution, N-body evolution, Binary 2-d Convolution, Binary Neural Nets, 3D Geometry Graphics Accelerators and Discrete Cosine Transformations [3]. Many other applications of custom build computers can be found in [6],[1],[2],[10],[13],[8]. In this paper we discuss the process of sintering, and show how it is possible to implement an application specific architecture tailored to this problem. We explore a number of design trade-offs and show how they affect the performance of the system.

Sintering is the metallurgical process in which an object is formed by heating a metal powder to a temperature below its melting point. The main mechanism in this process is diffusion, which can be effectively simulated using Monte-Carlo techniques. However, even unrealistically small problems contain millions of atoms whose movements must be calculated in every Monte-Carlo step. As a result, simulations can take days of CPU time on a conventional workstation.

Our goal is to build a specialised processor to accelerate the simulation of sintering and allow quicker experimentation with its governing parameters. At this stage we consid-

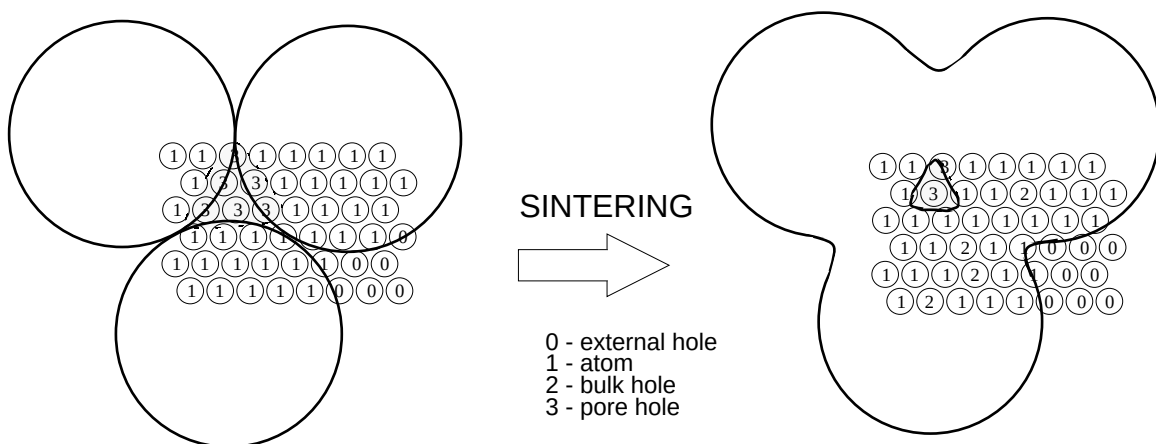


Figure 1. Two-dimensional model of sintering

```

WHILE not finished DO
  read address of hole
  choose random neighbour
  read hole and neighbour
  IF neighbour is atom THEN
    retrieve neighbours of hole
    retrieve neighbours of atom
    calculate deltaN // difference in number of atoms
    calculate value of probability function P(deltaN)
    choose random number R(0-1)
    IF R < P THEN
      write swapped hole and atom
      update hole address
    END IF
  END IF
  increment hole array counter
END WHILE

```

Figure 2. Pseudo-code of the sintering simulation algorithm

er only a two-dimensional (cross-sectional) model. In section 2 of this paper, an atomistic model of the sintering process is presented. Section 3 introduces the architecture and technology considerations involved in mapping this model on to hardware. Sections 4 and 5 present two different possible architectures and various optimisations. In section 6 the performance of these architectures is estimated and compared to that attainable on a high-end workstation.

2. The Sintering Model

Figure 1 shows a two-dimensional model of sintering. Each large circle represents one particle of the sintered powder, while the smaller circles represent atoms and holes. Only three particles are considered since the results will be similar throughout the powder.

Since the number of vacancies in the lattice (holes) is generally less than one percent of the number of atoms, it is more convenient to consider the motion of holes. Initially, all of the holes are in the pore (that is, in the space between the three particles), and during simulation may migrate into the particles or around the surface of the pore. The probability that a hole will swap places with a particular neighbour is influenced by the configuration of atoms and holes in its neighbourhood. This is because atoms have a higher probability of moving to a lower energy state (one in which they have more atoms as neighbours). The pseudo-code in Figure 2 describes the sintering process as implemented in software [11].

In our model of sintering there are two main diffusion mechanisms [11]:

- a) Bulk diffusion, where a hole can move just one

atomic distance at a time;

- b) Surface diffusion, in which holes on the surface of the powder particle can jump several atomic distances to fill a remote hole. This can be modelled using a less constrained probability function than for bulk diffusion [11].

In this paper we only address the simulation of systems which can be represented using a hexagonal grid, thus each atom has six immediate neighbours. This covers a wide range of elements, including Copper, for which experimental data is available for verification of the simulation.

3. Architecture and Technology Considerations

The model depicted in Figure 1 has some features which can be exploited to accelerate the simulation. For example, processing hole rather than atom movement dramatically reduces the number of elements which need to be processed. If the algorithm is executed on a special purpose computer rather than a general purpose machine, then a number of other optimisations can be realised. For example:

- a) The memory word length can be matched to the data structures required by the algorithm, resulting in smaller storage requirements;
- b) Memory accesses can be interleaved, allowing parallel access to a group of atoms;
- c) Low level concurrency in the algorithm can be exploited effectively;
- d) The functions, structure and word size of the ALU can be tailored to the problem.

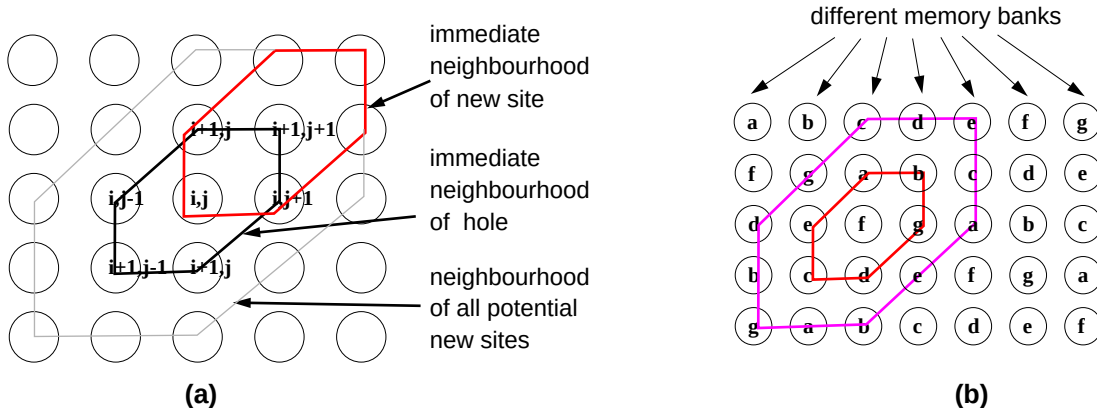


Figure 3. Organisation of hexagonal grid in software (a) and in hardware (b).

Most of these optimisations relate to the memory organisation used in the machine. The memory organisation considered for this work has been designed to store a hexagonal grid efficiently, and is shown in Figure 3a. Three bits are required for each location to encode the type of hole, namely whether it is a pore, bulk or external hole, or the type of atom, making it possible to represent different metals.

The probability that a hole will swap places with a particular atom depends upon the number of neighbour atoms in the new position compared to the current number of neighbours. Therefore, seven accesses are required to fetch a hole and its six immediate neighbours. A further three accesses are required to fetch the neighbourhood of the new location, since four of the required sites must be common with the current position of the hole.

A high performance solution is to use a seven-way interleaved memory, as shown in Figure 3b.

Such a scheme allows any hole and all its immediate neighbours to be fetched in a single access. Thus, only two memory access are required to process each hole. However,

technical considerations regarding the number of address and data lines routable on a PCB make three-way interleaving (described in section 5) more practical.

4. A Scanning Architecture

The simplest implementation of the simulation is to process all sites sequentially, irrespective of whether they contain an atom or a hole, as shown in Figure 4. As discussed earlier, this approach is usually rejected for a software implementation because most of the sites do not change in every simulation step. However, from a hardware perspective, the scheme has some advantages, namely:

- a) It is not necessary to maintain a list of holes, with its associated overheads;
- b) Address generation is extremely simple;
- c) The architecture leads naturally to pipelined implementation;
- d) It is easy to parallelise.

The major problem with this architecture for sintering simulation is that the number of holes is typically less than

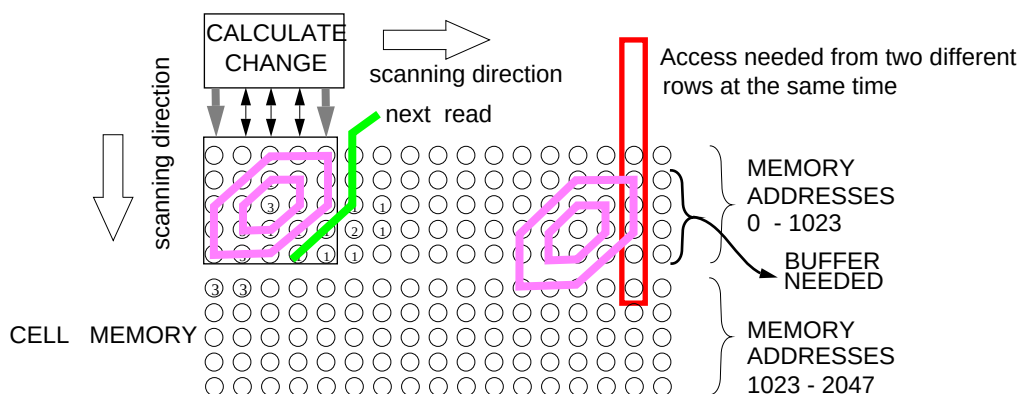
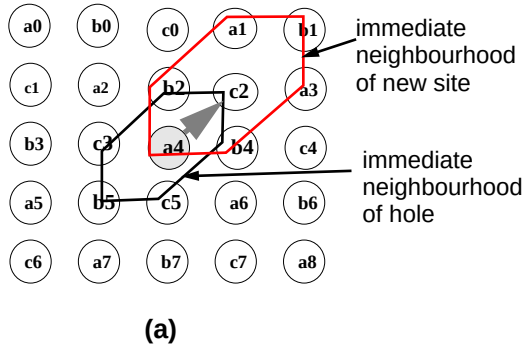


Figure 4. Principle of the scanning method and the need for a buffer.



Operation	Frequency
FETCH 1 : a4, b4, c2	N_p
FETCH 2 : a3, b5, c5	N_{mvbl}
FETCH 3 : a1, b1, c3	N_{mvbl}
FETCH 4 : b2	N_{mvbl}
WRITE : a4, c2	N_{mved}

(b)

Figure 7. Organisation of hexagonal grid on a three way interleaved memory (a) and the access sequence (b)

viously taken by one. The additional complexity over a non pipelined implementation results from the need for a smart buffer scheme between the memory and the state logic.

4.2 Parallelisation

Another way of accelerating the simple scanning architecture is to exploit data parallelism. This parallelism arises because each row can be calculated in parallel. Figure 6a shows a scheme in which N strips of rows are processed in parallel. In this case the memory data word is extended to N sites ($N \times 3$ bits), while the registers are extended to $N+4$ sites. The buffer requirements are unchanged. For a strip of ten cells, the memory word size becomes 30 bits, which is still feasible with current technology. The computation hardware must be replicated ten times, but as mentioned earlier it is relatively small. However, the computation hardware must now resolve potential conflicts, as illustrated in Figure 6b. In order to achieve the same results as for a sequential implementation it is necessary to give higher priority to upper holes.

5. A “Holes Only” Architecture

The main disadvantage of the scanning architecture is that it processes about 100 times more information than the software implementation. It is also possible to implement a machine which only processes holes, using the scheme shown in Figure 2. This algorithm has a number of features which can be exploited by the architecture. First, if the randomly chosen neighbour is also a hole, then the original hole is unmovable and no further action is needed, avoiding a number of memory read operations. Second, if the probability function for a movable hole results in no movement, then again no memory write is necessary and processing can continue with the next hole. In order to analyse the behaviour of a machine which exploits these properties, the following definitions are useful:

Let

MCS = the total number of Monte Carlo steps in the simulation;

N_p = the total number of holes processed during the simulation;

N_{mvbl} = the number of processed holes which are potentially movable;

N_{mved} = the number of holes which were actually moved;

Therefore

N_{mvbl}/N_p = Fraction of processed holes which are potentially movable;

N_{mved}/N_p = Fraction of processed holes which are actually moved;

Table 1 shows the ratios N_{mvbl}/N_p and N_{mved}/N_p for some typical simulations. One observation is that the fraction of holes which are actually moved is very small and does not increase as the problem size grows. However, the number of holes which are potentially movable falls to a small fraction of the total number of holes processed. Consequently, most of the time will be spent processing the holes which are unmovable, and thus the number of cycles spent looking at these holes should be minimised. The two designs discussed in sections 5.2 and 5.3 are focused on minimising this time.

5.1 Memory Organisation

As discussed in section 2, an appropriate memory organisation for this architecture is seven-way interleaved. Unfortunately, address generation for such a scheme is expensive, and the large number of connections to memory makes physical construction difficult. However, a three-way interleaved memory as shown in Figure 7 is more practical. Four read accesses are required to process a movable hole. However, it is important to note that all four reads are

Table 1.

wire radius	MCS	N _p	N _{mvl} /N _p	N _{mvd} /N _p
50	2×10^6	9.42×10^8	0.69	0.0015
100	2×10^6	3.71×10^9	0.22	0.0016
200	2×10^6	1.50×10^{10}	0.07	0.0013

not always required, as illustrated in Figure 7b. If the chosen neighbour is also a hole, no move is possible and processing continues with the next hole. At the start of simulation more than 99 percent of holes are unmovable and therefore require just one read access, though this value decreases as the simulation progresses. The proportion of holes which actually move, requiring the final write access, is even less, as shown in table 1.

5.2 A non pipelined architecture

Figure 8a shows the design of an architecture capable of implementing the “holes only” algorithm. The hole array memory stores the address of each of the holes, and is relatively small. For example, a 1500x1500 problem only requires 47 kilobytes of high speed memory. The address generator produces an address for each of the three main memory banks of the neighbouring sites. Up to five such address combinations (listed in Figure 7b) are produced for each hole. A set of registers is used to store the fetched data for calculation. The move calculation hardware is combinational logic, identical to that used in the scanning method.

Figure 8b shows the state diagram for the machine.

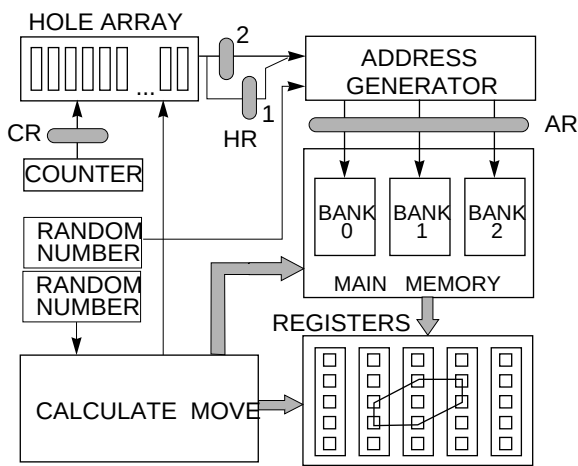
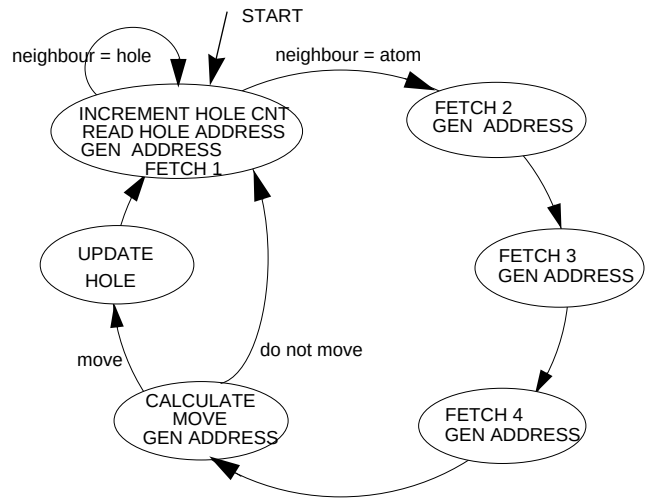
**(a)****(b)**

Figure 8. Architecture (a) and state diagram (b) of the pipelined “holes only” sintering processor

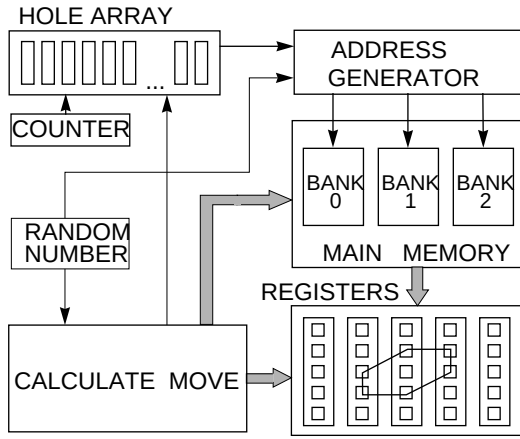
Twelve states are required to process a hole which is moved including the write back to memory. If the hole is unmovable, then the state machine exits after three transitions, and if the hole does not move because of energy considerations then one state is skipped. Thus, based on the simulation results, the processor would spend a significant amount of its time executing only three states.

5.3 A pipelined architecture

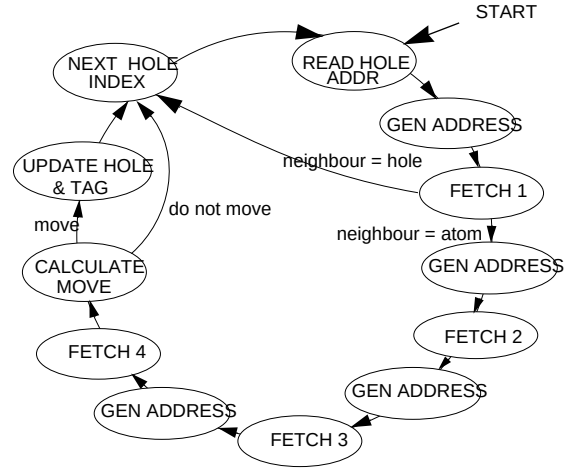
Like the scanning architecture, the “holes only” design can also be pipelined, as shown in Figure 9a. Pipelining exploits the temporal concurrency in processing holes, and can provide significant performance gain. The only new elements are the pipelining registers.

Pipelining allows many states to be merged, resulting in the state diagram of Figure 9b. Addresses can be pre-generated for the next main-memory fetch. Also, the hole array counter can be incremented and the next hole address read in parallel with the main-memory fetch.

Importantly, unmovable holes are now processed in just one cycle instead of three. As table 1 shows, these make up the bulk of the processed holes for large problems, and the



(a)



(b)

Figure 9. Architecture (a) and state diagram (b) of “holes only” sintering processor

advantage of processing them quickly grows dramatically as the problem size increases.

6. Performance Comparison

In this section we analyse the performance of the four architectures presented in this paper, namely the pipelined and non-pipelined scanning machine and the pipelined and non-pipelined “holes only” machine. The following parameters are used for the comparison:

- Radius of each particle: 100 atoms
- Total Lattice size: 442 x 442
- Initial no. of holes: 1854
- Total Monte Carlo steps: 2×10^6

Performance was compared to an efficient optimised Fortran program¹, which used a “holes only” method, executed on a Sun SparcStation 5. This program generated the following statistics, which were then used to estimate the performance of the hardware architectures:

$$N_p = 3,708,000,000$$

$$N_{mvbl} = 801,474,213 \Rightarrow N_{mvbl}/N_p = 22\%$$

$$N_{mved} = 5,964,814 \Rightarrow N_{mved}/N_p = 0.16\%$$

Table 2 summarises the performance of the two architectures for both FPGA and gate array technology.

The cycle times are technology and architecture dependent, and were chosen based on the underlying technology. For the “holes only” architectures, address generation is the most time consuming operation, since it involves a table lookup and three parallel 20-bit additions. Hence the cycle

time is determined by the speed of the address generator. For the scanning architectures the address generator is a simple counter. The memory access time is equal to 15 ns, which is typical for fast static RAMs used in cache memories.

7. Conclusions

In this paper we have presented a number of designs for a specialised machine for modelling sintering.

The scanning architecture is attractive because of its simplicity, however the performance loss through processing inactive sites cannot be ignored. Other machines which have been built to solve problems of this class, like the CAP have ignored this loss of efficiency and have assumed that all sites must be processed [12],[9].

Even pipelining the scanning architecture and using a high speed gate array technology cannot result in a significant speedup over a conventional workstation. The parallel implementation of this architecture is limited by physical constraints of memory width to about ten processing units and would still perform worse than the simple “holes only” architecture.

When implemented without pipelining, the “holes only” architecture only achieved a modest performance improvement, however, pipelining the design resulted in a substantial speed increase using both field programmable and gate array technology.

A prototype sintering processor is currently under construction. A VHDL simulation model of the hardware has been built and design of the processor is underway using Xilinx FPGAs [14],[4],[5].

Acknowledgements

The authors would like to acknowledge the contribution of our colleagues at the University of Queensland, namely

1. The program was written and kindly provided by Dr. Ivan Podolsky of the Mathematics Department, The University of Queensland.

Table 2.

Technology	Architecture	Machine Cycle Time	Total Cycles	Monte Carlo Steps /sec	Speedup
SS5	Fortran holes only	13 ns	--	57	1.0
Xilinx 4010 FPGA	Scanning	50 ns	1.41×10^{12}	28	0.5
	Scanning (pipelined)	50 ns	4.69×10^{11} a	85	1.5
	Scanning (pipelined) 10 parallel	70 ns	4.69×10^{10} b	610	10.7
	Holes only	100 ns	1.96×10^{10} c	1183	20.8
	Holes only (pipelined)	100 ns	6.12×10^9 d	3268	57.3
Gate Array	Scanning (pipelined)	30 ns	4.69×10^{11} a	142	2.5
	Holes only	50 ns	1.96×10^{10} b	2040	35.8
	Holes only (pipelined)	50 ns	6.12×10^9 c	6536	114.7

a - $442 \times 442 \times \text{MCS}$ b - $442 \times 442 \times \text{MCS}$ c - $4\text{Np} + 7\text{Nmvl} + 1\text{Nmvd}$ d - $1\text{Np} + 4\text{Nmvl} + 1\text{Nmvd}$

Professor Kevin Burrage, Dr. Graham Schaffer and Dr. Ivan Podolsky. This project is supported by the Australian Research Council .

References

- 1 D.A. Abramson, "A Very High Speed Architecture to Support Simulated Annealing ", IEEE Computer, May 1992.
- 2 D. Abramson, "High Performance Application Specific Architectures", Proceedings of 26th Hawaii International Conference on System Sciences, Kauai, Hawaii, Jan 1993.
- 3 P. Bertin, Roncin and Vuillemin, "Programmable Active Memories: A Performance Assessment", Technical Report, DEC Paris Laboratories, 1990.
- 4 D. Bursky, Advanced 100-kgate FPGA Family Takes On Gate Arrays, Electronic Design, May1, 1995.
- 5 D.Bursky, FPGA Routing Enhancements Boost the Usability Of Complex Logic, Electronic Design, May1, 1995.
- 6 R.W. Hartenstein, Custom Computing Machines, Copernicus CP 94:0536 Benefit-DMM'95
- 7 R.W. Hartenstein, J.Becker, R.Kress,H.Reinig,K.Schmidt, A Two-level Hardware/Software Co-design Framework for Automatic Accelerator Generation, Copernicus CP94:0536 Benefit-DMM'95
- 8 A. Jantsch, P. Ellervee, J. Vberg, and A. Hemani, "A Case Study on Hardware/Software Partitioning", Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, CA, April 1994.
- 9 G.Milne, P.Cockshott, G.McCaskill, P.Barrie, Realising Massively Concurrent Systems on the SPACE Machine, Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, April 1993, Napa, California.
- 10 R.Razdan, M.D.Smith, A High-Performance Microarchitecture with Hardware-Programmable Functional Units, MICRO-27 -11/94 San Jose, USA.
- 11 T. Sercombe, "A Monte Carlo Simulation of Sintered Microstructures", Bachelor of Engineering Thesis, Department of Mining and Metallurgical Engineering, The University of Queensland, 1994.
- 12 T. Toffoli and N. Margolus, "Cellular Automata Machines", The MIT Press, 1987, ISBN 0-262-20060-0.
- 13 R.Venkateswaran, P.Mazumder, Coprocessor Design for Multilayer Surface-Mounted PCB Routing, IEEE Trans. on VLSI, Vol.1, No.1, March 1993.
- 14 XILINX and APTIX data books