FPGA based custom computing machines for irregular problems

Abramson, David; Logothetis, Paul; Postula, Adam; Randall, Marcus

FPGA Based Custom Computing Machines for Irregular Problems

David Abramson † Paul Logothetis † Adam Postula § Marcus Randall £

† Department of Digital Systems,
Monash University,
Clayton, Victoria, 3168
{davida, paull}@dgs.monash.edu.au

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

£ School of Applied Science
Griffith University, Gold Coast
Queensland, Australia, 4217
m.randall@eas.gu.edu.au

ABSTRACT

Over the past few years there has been increased interest in building custom computing machines (CCMs) as a way of achieving very high performance on specific problems. The advent of high density field programmable gate arrays (FPGAs), in combination with new synthesis tools, have made it relatively easy to produce programmable custom machines without building specific hardware.

In many cases, the performance achieved by a FPGA based custom computer is attributed to the exploitation of massive concurrency in the underlying application. In this paper we explore the sources of speedup for irregular problems in which is difficult to exploit such parallelism. We highlight 5 main sources of speedup that we have observed, namely the provision of high memory bandwidth, the use of flexible address generation hardware, the use of gather-scatter array operations, the use of lookup tables and the use of multiple tailored arithmetic units. By considering some representative examples of such irregular problems, the paper illustrates that good performance is possible given the current generation of FPGA devices and RISC processors.

The paper then explores whether this performance gain will be possible given the next generation of RISC processors and FPGAs. It concludes that the only way to maintain the speedup is to alter the architecture of CCMs in combination with architectural changes to the FPGAs themselves.

1. Introduction

Over the past few years there has been increased interest in building custom computing machines (CCMs) as a way of achieving very high performance on specific problems. The advent of high density field programmable gate arrays (FPGAs), in combination with new synthesis tools, have made it relatively easy to produce programmable custom machines without building specific hardware. This approach is supported by systems such as SPLASH [8], which provide a collection of FPGAs with a reconfigurable interconnection network. Programming tools range from conventional schematic capture through to high level synthesis systems. There are now a number of commercially available CCM products on the market [31, 32, 33, 34]. The approach has spawned a conference series devoted to the application of FPGA based custom computers [13, 14, 15, 16].

There has been relatively little work on analyzing the source of speedup provided by CCMs [2, 6, 5]. In many cases, the speedup over conventional workstations has been impressive. For example the SPLASH II machine achieved a performance improvement of about 40,000 over a Sparc 10 workstation on solving a genetic database problem. The SPLASH system also achieved a speedup of about 1000 times a 32 node CM5. Performing image processing, the same hardware ran between 10 and 100 times faster than a Sparc 10 workstation on an image processing application.

Traditionally, the type of application which can benefit from implementation on a CCM has had the following attributes:

- It is iterative;
- It has high memory bandwidth requirements;
- Fixed point arithmetic is sufficient;
- The memory operations are array based;
- The problem contains concurrency.

The iterative nature of the algorithm means that some core computation is executed repeatedly. Thus, by implementing the core in hardware, the overall process can be accelerated without mapping the entire application into hardware. This approach is exemplified in hardware/ software co-design [23, 17], in which an application is split between a conventional machine running software and a CCM. The requirements for high memory bandwidth mean that a CCM can tailor its memory interface to the problem, and can deliver the data rapidly to the core algorithm. This requirement often becomes a bottleneck when the algorithm is implemented on a Von-Neumann machine, because data elements must often be fetched sequentially through a narrow memory data path. Also, unless the algorithm exhibits significant spatial and temporal locality, the performance of modern cache based RISC microprocessors can be poor. The requirement that the algorithm should operate with fixed point arithmetic is largely because it is difficult to implement floating point arithmetic units on FPGA based CCMs which are competitive with those on commodity microprocessors. Further, if some operations are to be executed in parallel, then multiple ALUs are required, making the CCM extremely large and expensive. Array based memory operations are common in CCM applications because the memory subsystem can be tailored for fetching and operating on array elements. Further, memory pipelining and interleaving can deliver significant performance. Finally, almost all successful CCM applications have exhibited some level of concurrency. Schemes which utilize both spacial concurrency (i.e. parallel functional units) and temporal concurrency (i.e. pipelining) are likely to be effective on all these applications.

There is little doubt that the problems with abundant concurrency can benefit most from implementation on a CCM. In these *regular* applications, the core computation is replicated many times, and the data is delivered to the ALUs in such a way that very high throughput is possible. However, there are also a number of *irregular* problems which only exhibit low levels of concurrency. In this paper we pose three questions:

1. *What are the main opportunities for speedup in irregular applications?*

2 *Is it possible to achieve reasonable speedup using current technology?*

3 *What advances are required in FPGA architecture to maintain the speedup?*

In order to answer these questions, we start by examining some sources of speedup in irregular problems. We then report the speedup which has been achieved by implementing some real problems on CCMs. Since these experiments were performed on the last generation of FPGAs and RISC machines, we examine whether the speed can still be maintained with current technology. Finally, we suggest some architectural changes to the CCM architectures and the FPGAs themselves if the speedup is to be maintained.

2. Speedup for Irregular Problems

From studying a number of irregular problems we have identified 5 main techniques for achieving speedup when the problem is mapped onto a CCM. These are:

- the provision of high memory bandwidth,
- the use of flexible address generation hardware,
- the use of gather-scatter array operations,
- the use of lookup tables and
- the use of multiple tailored arithmetic units

These same techniques can also be applied to regular problems. However, such applications can gain significant performance improvments by exploiting the available concurrency, whereas irreulgar ones do not possess this inherent parallelism. In this section we examine these techniques and illustrate their use by example.

2.1 High Memory Bandwidth

Traditionally, general purpose computers have organized memory as a collection of fixed size words. Accordingly when the data for a problem is mapped onto such machines it must be broken into a number of fixed size words. This mapping process effectively sequentializes the way that the algorithm accesses the data. The need for high memory bandwidth is present in many applications. For example, in graphics and image processing applications an entire image is processed, and thus the individual pixels must be read from memory, processed and written. Whilst spatial locality in these types of operations might be high, because the data may be accessed sequentially, the temporal locality is often very low. Consequently, Von Neumann processors may exhibit poor cache performance.

In the examples of irregular problems considered in Section 3, the data structures do not fit neatly into one word, and thus multiple memory reads are required on a RISC processor. However, on a CCM the memory organization can be tailored to the data size and all related data can be fetched in one operation given sufficient parallel access to the memory. Further, each of the key data structures can be implemented in different memory devices, providing further concurrency in fetching the required data.

Another technique which can be applied on CCMs is to replicate some array structures which must be accessed concurrently at different index values. For example, consider accessing A[i] and A[j] concurrently. This is possible by maintaining two copies of array A, and then accessing them with different key values, as shown in Figure 1. The scheme works well providing the ratio of read accesses to writes is high, because when a write is performed to more than one location, it is necessary to update both memories sequentially. However, on a Von Neumann machine the two read accesses must be sequentialised, leading to limited performance.

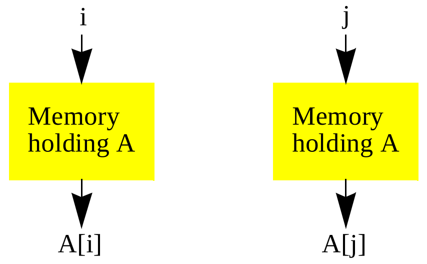

Fig 1. Concurrent access of A[i] and A[j]

2.2 Flexible Address Generation Hardware

Many candidate applications manipulate multi-dimensioned arrays. On a RISC processors, it is necessary to calculate an address in linear memory from the individual index values, which involves a number of addition and multiplication (or shift) operations. For example, to compute the address of the expression A[i][j] and to fetch the data, at least 5 RISC instructions are required.

However, if the array bounds are powers of 2, then it is possible to form the address on a CCM without using any logic, by catenating the two values i, and j, together to form an address for the memory containing the data. This is illustrated in Figure 2, where the data at location A[i][j] can be read in one memory access time. When this type of addressing structure is used to fetch more than one item concurrently it is possible to access large amounts of data very quickly.

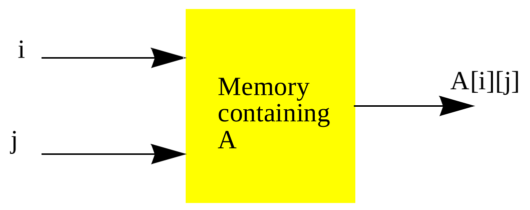

Fig 2. Two dimensional access of memory

2.3 Gather-scatter Array Operations

Some applications access arrays using a mapped version of a simple index rather than the index value itself. This type of addressing pattern is very common in problems which implement sparse array structures, and is supported by a gather or scatter operation depending on whether a read or a write is performed. A gather operation can be represented as the following expression: A[B[i]], where array A is accessed using the value which is retrieved by indexing array B. In many of the applications we have examined, two dimensional gather operations are required, as represented using the expressions: A[B[i]][B[j]]. On a RISC processor, evaluating this expression takes a minimum of 12 instructions using standard compilation techniques. Whilst this can be reduced by using special data structures and instruction sequences, this requires significant low level programming effort.

On a CCM it is possible to use the output of one memory directly as the address to another, and the entire gather operation can be performed in two memory accesses. Figure 3 shows a two dimensional gather implemented with memory devices.

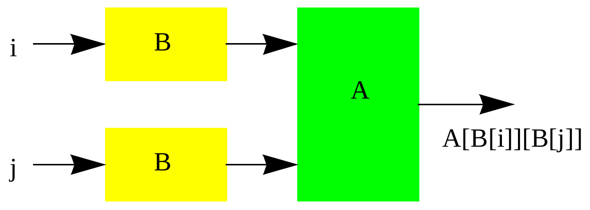

Fig 3. CMM implementation of gather operation

2.4 Lookup tables

One of the more difficult aspects of mapping algorithms onto CCMs involves removing the need for complex expression evaluation hardware. Providing hardware support for evaluating trigonometric and transcendental functions, and even performing relatively simple operations like multiplication and division, can be expensive.

In many cases these operations can be replaced by lookup tables. For example, in the machine described in section 3.2 it is necessary to evaluate the expression exp(-ΔC/T) for different values of ΔC and T. This expression would be extremely expensive to evaluate directly in hardware because it involves a division and computation of the exp() function. However, in practice the value of T is held constant for a large number of iterations whilst ΔC changes on each iteration. Using this information, it is possible to compute a table of values, indexed by ΔC, for fixed values of T. The table data can be computed by a host computer whilst the CCM is running, and the table can be reloaded when the value of T changes. This technique avoids the need for division and exp() on the CCM, and allows the expression to be evaluated in the time taken to read a high speed memory.

Whilst it is also possible to accelerate the computation performed by the software running on a workstation by using lookup tables, the schemes discussed in sections 2.2 and 2.3 mean that it is possible to index the table extremely rapidly on a CCM.

2.5 Multiple Tailored Arithmetic Units

Modern superscalar processors contain more than one arithmetic unit, and can perform simple logical and arithmetic operations in a single cycle. In many potential CCM applications the operations are quite simple, but may not be well supported by conventional ALUs. On a CCM it is possible to build a number of simple ALUs which perform the required task, and then to replicate them to exploit underlying parallelism. For example, the architecture described in section 3.1 requires the computation of a new single bit cell value, based on the values in surrounding array locations. This ALU is implemented using a relatively simple set of logic functions, and because it is small, a number can be implemented on the CCM.

The architecture developed in [1] required a number of values to be incremented and decremented as part of the algorithm. On a Von Neumann processor this would have required the sequential execution of a series of add instructions. However, in the CCM which was developed, it was possible to build a set of ALUs which could only perform increment and decrement. Further, because the data could be accommodated in 6 bits, the ALUs could also be fixed at 6 bits with clamping to handle overflow.

The parallelism available in irregular problems is often quite small and nested in inner loops. By using multiple tailored ALUs it is possible to unwind these loops and achieve high performance on a CCM.

3. Some Example Architectures

In this section we describe two CCMs for irregular problems, and highlight their performance in context of the architectural techniques discussion in section 2.

Both machines are currently implemented on an Aptix AP4 reconfigurable logic board, containing a number of Xilinx XC4010 FPGAs and Aptix FPIC interconnection chips [4]. The designs occupy one or more XC4010s and a number of memory devices. The machines are clocked by an external 10 MHz oscillator on the AP4 board.

The designs are specified in a mix of schematic circuits and VHDL, which is then synthesized using the Autologic tools from Mentor Graphics. Partitioning is performed by the Xilinx Foundry tool set, however, we found it necessary to provide information based on our knowledge of the problems to assist the partitioner. By carefully specifying the individual components, the quality of the resulting circuits is fairly close to those which could have been designed manually.

3.1 The Sintering Problem

Sintering is the metallurgical process in which an object is formed by heating a metal powder to a temperature below its melting point [24]. 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.

Accordingly, we built a specialized processor to accelerate the simulation of sintering and allow quicker experimentation with its governing parameters. The machine which was developed only supported a two-dimensional (cross-sectional) model of the process, however, research is being undertaken to build full three dimensional models [24].

Figure 4 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.

The most natural way to model this problem is by using a regular cellular automata [25], and to model the movement of atoms. Such an approach is ideally suited to implementation on a CCM, and one could expect high performance. Similar examples are found in [25, 19]. However, in this problem, the number of vacancies in the lattice (holes) is generally less than one percent of the number of atoms, so 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 neighbor is influenced by the configuration of atoms and holes in its neighborhood. This is because atoms have a higher probability of moving to a lower energy state (one in which they have more atoms as neighbors).

Because the problem models holes rather than atoms, it is very difficult to extract high levels of parallelism from the algorithm. This is because the movement of one hole can affect the movement of other holes, and, in general, it is not possible to process all holes concurrently. Accordingly, the only available parallelism is in the way that each hole is processed. In Figure 5 we outline the algorithm using pseudo

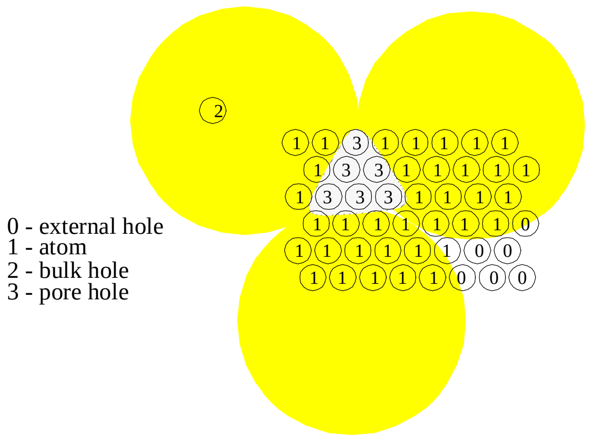

Fig 4. Model of Sintering Process

code.

In another paper, we explored a wide range of implementation techniques [3], and showed that an architecture which processed only the holes was more efficient than one which processed the entire grid, even though the latter ran with a much faster cycle time. A schematic of this architecture is shown in Figure 6.

```
for time = 0 to numberofsteps
    read address of hole
    choose random neighbor
    read hole and neighbor
    if neighbor is atom then
        read neighbors of hole
        read neighbors of atom
        let deltaN = difference in number of atoms
        let P = probability of accepting deltaN
        choose random number R
        if R < P then
            write swapped hole and atom
            update hole array
        end if
    end if
    increment hole array counter
end for
```

Fig 5. Pseudo code for sintering simulation.

The machine utilizes two main memories. One stores the

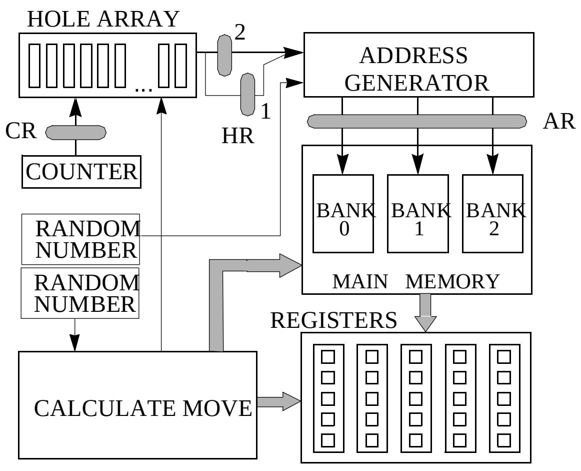

Fig 6. Sintering Architecture

state of each cell in the cellular array (called the MAIN MEMORY in Figure 6), and another stores the positions of the holes (called the HOLE ARRAY in Figure 6). The architecture scans through the holes, as addressed by a counter. An address generator is capable of taking the hole address and producing a set of main memory addresses for the neighboring cells. Thus, the machine utilizes a two dimensional gather operation as discussed in section 2.3. In order to process the information efficiently, three neighboring cells are accessed concurrently, using an interleaved memory organization. Once all of the information about a hole and its neighbors has been extracted, the new position for the hole is computed using a special ALU called the CALCULATE MOVE unit. This ALU takes the state of the 6 neighboring cells and decides whether the hole should move to a randomly chosen position.

Two versions of the CCM were designed, and one was built. The first uses some internal pipelining to overlap the fetch of hole addresses with the processing of movement calculations and memory writeback. The performance of this machine was predicted at about 60 times faster than an optimised Fortran program running on the IBM RS6000/590. The other design, which was actually implemented did not use much internal pipelining, and achieved a speedup of 30 times over the same Fortran program. The CCM utilized a number of 20 nsec memory chips, a Xilinx 4010 FPGA and an Aptix FPIC for interconnecting the FPGA and memory devices.

The architecture discussed above utilizes a number of the techniques discussed in section 2. Most notably, it uses:

- A gather mechanism for addressing the main memory via the holes memory
- Specifically tailored ALUs for computing the new cell value
- Interleaving of memory for high bandwidth

3.2 The Travelling Salesman Problem

The well known travelling salesman problem is to find a tour of cities for a salesman which minimizes the distance travelled [7, 18]. TSP is an important combinatorial optimisation algorithm because it has many of the properties of real world problems, and thus it can be used to test new optimisation algorithms. Among the many solution techniques for TSP, simulated annealing has been applied in the past.

Simulated Annealing (SA) is a heuristic algorithm. It is based on an analogy with the physical process of annealing, which involves the aggregation of many particles in a physical system as it is cooled [9, 18]. It was first introduced as a technique to solve complex non linear optimisation problems. SA has general applicability because it performs optimisation without prior knowledge of problem structure or of any particular solution strategy. Traditionally, it has been easy to apply because *transitions* (in the solution space of the problem being solved) are easy to derive. Furthermore, the formulation of a cost function, however complex, is easily obtained in most cases. Transitions (or *moves*) lead to either *uphill* or *downhill* movements in the solution space. In a minimization problem, moves of the latter type are automatically accepted while moves of the former type are accepted with a certain probability, based on the evaluation of the Boltzmann equation, $\exp(-\Delta C/T)$.

There are many ways of representing the TSP, however, a convenient way is to store the tour as an ordered list, where each element contains the identity of a city, and the ordinal of the city represents its position in the tour. There are a range of transition operators which can be applied. One operator, which is well suited to implementation in a CCM, involves swapping pairs of cities in the tour.

Using this list formalism, the TSP can be defined as fol-

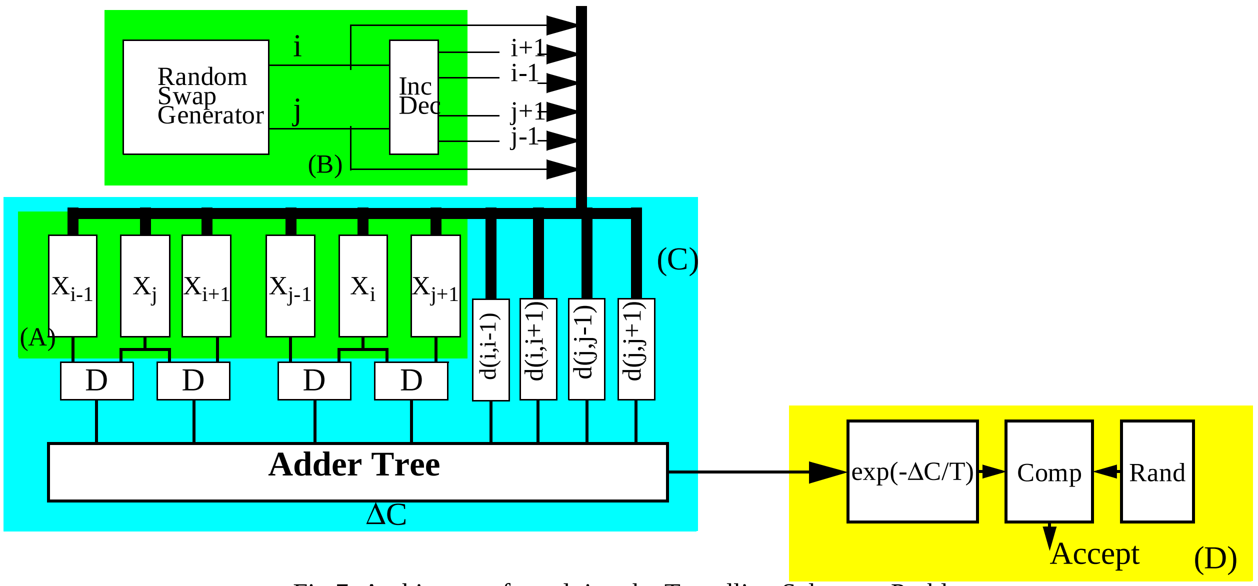

Fig 7. Architecture for solving the Travelling Salesman Problem

lows:

$$\text{minimise} \quad \sum_{i=2}^{N} d(x_i, x_{i-1}) + d(x_N, x_1)$$

subject to:

$$x_i \neq x_j \quad \forall\ i,j \quad i \neq j$$

Where:

x_i is the i^{th} city on the tour

$d(c_1,c_2)$ is the distance between city c_1 and city c_2

Figure 8 shows the design of a simulated annealing algorithm for solving this problem.

```
repeat
    choose two cities at positions  i, j of the tour
    read cities i, i+1, i-1, j, j+1, j-1
    compute inter-city distances
    let detaC = difference in cost
    choose random number R
    if R < exp(-deltaC/T) then
        swap cities i and j
        re-write inter-city distances
    end if
until system is frozen
```

Fig 8. Pseudo code for simulated annealing algorithm.

Figure 7 shows the design of a machine which solves the TSP. It contains memory for storing a solution (A), a unit for generating a move (B), a unit for computing the change in cost (C) and an update unit which applies the Boltzmann equation (D).

Unit (B) is responsible for generating two city indexes, and for computing the neighborhood of the cells. It does this by generating two different random numbers between 0 and N-1, and then computing the increment and decrement of each number modulo N. These 6 numbers are used as direct indexes for the tour memory (X), which is held in unit (A). By reading the cities at each of the 6 positions concurrently and sending the data to a parallel adder tree the change in distance resulting from the interchange can be computed rapidly. To support this, Unit (A) provides 6 identical copies of the X memory. The inter-city distances in the new tour are computed by 4 lookup tables, D. Each D memory is addressed by the concatenation of two city identifiers, and presents the distance between those two cities. This approach works well for small problems because it avoids the need to compute the euclidian distance between cities. For larger problems we plan to explore alternative lookup table implementations of the distance computation hardware. The inter city distances in the old tour are stored in the **d** memories. Each of the 8 distances is then presented to the adder tree and a change in cost is computed.

The change in cost is presented to the update unit (D), which evaluates Boltzmann’s equation to decide whether the swap should be accepted. If ΔC is negative, then it is accepted without any further computation. However, if ΔC is positive, the expression exp(-ΔC/T) is evaluated. To avoid having to calculate an exponential function and a division operation, a lookup table containing the values of exp(-ΔC/T) for various different values of ΔC is used. The table is reloaded each time the temperature is changed, which only occurs after several thousand swaps. This scheme was used in [1] and was quite successful because the table values can be computed by the host workstation in parallel with the execution on the CCM.

In this CCM the architecture achieved a speedup of about 40 times over an RS6000 Model 590 workstation. By examining the algorithm used by this machine, it is clear that the speedup is largely attributed to

- the implementation of two dimensional gather operations,
- the parallel fetching of this data from multiple memory banks
- the use of a special ALU implemented as a parallel tree adder.
- the use of a lookup table for the exponential function.
- A hardware implementation of the random number generator

3.3 Summary

Both of these application achieve a speedup of between 30 and 40 times over a high end workstation. In these experiments there is very little interaction between the CCM and the host computer, and thus the speedup is not degraded by executing software of the host. In the simulated annealing machine, the only interaction occurs when the temperature is decremented, and at this time the exponential table is reloaded. In the sintering machine, many cycles of the simulation are run before the memory state is extracted. Thus, the execution time is determined by the speed of the hardware alone.

The speedup was calculated by computing the CPU time per cycle for the software implementation and the real time for the hardware implementation. This level of speedup would seem to warrant the effort in building a CCM, especially if the problems are to be solved repeatedly. We believe this is typical of many irregular applications. In the next section we provide a more detailed analysis of the speedup using current technology.

4. Performance Trends

The technology used for the experimental work reported in this paper was *freely available* at the start of this project in late 1994. Accordingly, the speedups reported were actually measured by comparing the execution of the code on a 66 MHz IBM RS6000/590 against an Aptix AP4 board, configured with FPIC interconnection chips and Xilinx XC4010-4 FPGAs.

A speedup of this magnitude is significant; at this level it is possible to reduce a run from 35 minutes to 1 minute. Many practitioners would value such a speedup. Another perspective is that a parallel version of the algorithms, running with an efficiency of 80% on a 45 processor system, would achieve a speedup of 35 times. This type of parallel machine is significantly more expensive, and physically larger, than the type of hardware in CCM platforms. Further, given the correct skills, programming a parallel machine is not much easier than the type of high level hardware design used in these case studies.

We are interested in whether a FPGA based CCM would provide the same performance gain for this class of application if it were implemented using more up to date technology. To this end, we mapped the TSP machine onto more recent Xilinx FPGA chips, the 4000E-1 family, and computed the performance using simulation tools. We also chose faster memory devices, namely 10 nsecond memories which are readily available.

An IBM RS6000/595, a more recent version of the model 590, was used to gauge the performance of the algorithm on a current generation RISC. In order to predict the performance of the software implementation on the new processor, since one was not available, we used the ratio of the SPECint95 values for the previous and current machines. This methodology was confirmed by timing the software on a 66 MHz IBM RS6000/590 and a 266 MHz DEC Alpha, and compared this to the ratio of the SPECint95 values for the two machines. Since the answers were very close, we assume that the SPECint95 value serves as an adequate predictor for the performance of the software on future machines.

Given the simulation results and SPECint95 comparison, it would appear that it is possible to build a FPGA based CCM which would deliver comparable performance gains over the last generation of components, as shown in Table 1.

Of more concern, is whether it will be possible to achieve the same performance from the next generation of RISC processors and FPGA chips?

Digital Equipment have recently released the DEC 500/500 which has a SPECint95 value of 15. This appears to be the fastest processor on the SPEC table at present, and one would suspect that by next year many manufacturers will have similar speed systems. In order for a FPGA based CCM to achieve the same speedup, it would be necessary to run the system roughly two times faster than the 4000E-1 implementation. This seems unlikely for a number of reasons. First, it would require 5 nsecond memories, which are available, but are expensive. Second, it would require very fast FPICs and low inter chip delays. Finally, the FPGA would need to run roughly two times faster again. We are unaware of new FPGA devices capable of meeting the timing constraints of this new machine. The results of the analysis is summarized in Table 1. Thus, unless further changes are made to the architectures of the machines, and to the FPGA devices themselves, it is unlikely that the same level of performance gain will be released in the near future. One

Table 1: Performance Trends

	Parameter	1994	1997	1998
Soft-ware	Processor	RS6000 /590	RS6000/ 595	DEC 500/500
	SPECint95	3.33	6.17	15
	Relative Performance	1	1.8	4.5
TSP CCM	Cycle Time	310 nsecs	155 nsecs (simulated)	70 nsecs (required for speedup below)
	FPGA Family	Xilinx XC4010-4	Xilinx XC4000E-1	Unknown
	Interconnect delay	15 nsecs	15 nsecs	5 nsecs
	Memory	20 nsecs	10 nsecs	5 nsecs
	Relative Performance	1	1.5	4.5
Speedup	CCM/RISC	30 - 40	25 - 33	30 - 40

problem with FPGA devices is that they are unlikely to improve in speed as quickly as RISC microprocessors because the variable interconnection structure of the chips limits the speed at which signals can propagate in the device. On the other hand, ASIC CCMs could well compete with commodity micro-processor technology.

5. Conclusion

The analysis in section 4 would suggest that FPGA based CCMs can deliver speedup of the order of 30 times for irregular problems over the last few generations of FPGAs. However, the analysis also questions whether this performance gain can be maintained. In order to achieve similar speed gains as delivered by RISC micro-processors, it will be necessary to double the speed of the FPGA implementation by early next year. Whilst this is can be achieved for highly parallel regular applications by doubling the amount of hardware, irregular applications do not have sufficient concurrency to support this approach.

The most serious source of potential improvement is by altering the architecture of the CCMs themselves. The CCMs built as part of this study did not make any significant use of pipelining. A simple 2 level pipelining scheme should deliver a further speed improvement over the non-pipelined machines of at least 1.5 times. This would deliver a total speedup of about 45 to 60 times on current technology. Indeed, a paper design for a pipelined implementation of the sintering machine suggested that a speedup of 60 was possible using memories which were 4 times slower than those used in the non-pipelined version [21, 22]. Accordingly, more aggressive architectural design may further enhance the performance of CCMs on irregular problems.

Modifications to current FPGA architectures would be of further assistance. First, one of the current limiting features of both CMMs and RISCS is the speed of the memory devices. CCMs require fairly large amounts of very high speed memory to achieve good performance and RISCs require large secondary caches to maintain their performance. The integration of more memory on the FPGA devices would further enhance the performance of CCMs. This is already occurring for RISC machines in the form of large internal cache memories, and the approach has been suggested for FPGAs by a number of other authors [20, 27, 28]. Further, many FPGAs already contain quite large amounts of memory for implementing the arbitrary boolean functions, and this could be expanded to provide more addressable memory cells. The density of these devices should be equivalent to current CMOS static memory technology. This change would avoid the need to transmit signals off chip, and would allow faster memory access times to be realized.

Second, larger FPGAs will reduce the need for interconnection chips like the Aptix FPICs, further reducing the delays. As the FPGA and memory devices become faster, the interconnection delay represents a significant bottleneck.

Third, and more importantly, the provision of some higher level building blocks on the FPGA would make it possible to build faster special purpose ALUs. The current Xilinx family has already made some moves in this direction [29], by providing fast carry look ahead logic for adders. Similar integration could be possible for other arithmetic operations, and this would avoid the routing delays within the chip for these functions.

Another view on this work is that it would be possible for RISC processors to compete with CCMs on irregular problems if they incorporated some of the architectural techniques discussed in section 2. For example, much of the speedup in a CCM implementation comes from the support for multidimensional gather/scatter operations. Hardware support for this is not new, and has been provided on a number of vector supercomputers over the years (See [12], pp 380 - 382). Thus, one could expect such hardware to be incorporated into RISC chips in the future. Further, it would be possible to implement lookup tables and reconfigurable ALUs on chip. The latter approach has already been used to a limited extend in the multi-media extensions to the Ultra Sparc, in which the 64 bit ALU can be viewed as 8, 8 bit ALUs [26]. A reconfigurable ALU which permitted arbitrary dyadic operations would be very powerful for some applications, like the one discussed in section 3.1. Finally, it is clear that RISC processors require more memory bandwidth if they are to compete. The Multiflow Trace machine [10] experimented with a very long instruction word controlling multiple memory ports. If these techniques are integrated into current RISCS then their application specific performance could be improved.

In summary, good performance gains are possible if higher level architectural changes are made to the machine implementations and the FPGA devices. On the other hand, further advances in RISC chips may overtake changes in FPGA technology. The jury is still out!

Acknowledgement

This project is funded by the Australian Research Council Large Grants Scheme. Thanks go to Mohan Krishnamoorthy from the CSIRO, Yazid Sharaiha from Imperial College and our colleagues at the University of Queensland, namely Professor Kevin Burrage, Dr. Graham Schaffer and Dr. Ivan Podolsky. We would also like to acknowledge Phillip Freidin and David Duke for their helpful discussions.

References

1 Abramson, D. "A Very High Speed Architecture to Support Simulated Annealing ", IEEE Computer, May 1992.

2 Abramson, D. "High Performance Application Specific Architectures", Proceedings of 26th Hawaii International Conference on System Sciences, Kauai, Hawaii, Jan 1993.

3 Abramson, D, Logothetis, P., Randall, M and Postula, A. "Application Specific Computers for Combinatorial Optimisation", to appear, the Australian Computer Architecture Workshop, Sydney, Feb. 1997.

4 Aptix Corporation Programmable Interconnect Products System Data Book, November 1993

5 Bergmann, N and Mudge, C. “Comparing the Performance of FPGA-Based Custom Computers with General-Purpose Computers for DSP Applications”, IEEE Workshop on FPGAs for Custom Computing Machines, Napa, California, April 1994.

6 Bertin, P. Roncin and Vuillemin, "Programmable Active Memories: A Performance Assessment", Technical Report, DEC Paris Laboratories, 1990.

7 Bonomi E. and Lutton J.L., “The N-City Travelling Salesman Problem: Statistical Mechanics and The Metropolis Algorithm”, *SIAM Review*, **26**, 551-568, 1984.

8 Buell, D., Arnold, J and Kleinfelder, N. “Splash 2: FPGAs in a Custom Computing Machine”, IEEE Press, 1996, ISBN 0-8186-7413-X.

9 Collins N.E., Eglese R.W. and Golden B.L., “Simulated Annealing: An Annotated Bibliography”, *American Journal of Mathematical and Management Sciences,* **8**(3-4), 209-307, 1988.

10 Colwell, R., Nix, R, O’Donnell, J, Papworth, D and Rodman, P, “A VLIW Architecture for a Trace Scheduling Compiler”, IEEE Transactions on Computers, Vol 37, No 8, August 1988, pp 967 - 979.

11 Hartenstein, R.W. Custom Computing Machines, Copernicus CP 94:0536 Benefit-DMM'95

12 Hennessy, J. and Patterson, D., “Computer Architecture: A Quantitative Approach”, Morgann Kaufmann Publishers, Inc, San Mateo, California, 1990.

13 IEEE Computer Society, "Proceedings of IEEE Workshop in FPGAs for Custom Computing Machines", April, 1993, Napa, California.

14 IEEE Computer Society, "Proceedings of IEEE Workshop in FPGAs for Custom Computing Machines", April, 1994, Napa, California.

15 IEEE Computer Society, "Proceedings of IEEE Workshop in FPGAs for Custom Computing Machines", April, 1995, Napa, California

16 IEEE Computer Society, "Proceedings of IEEE Workshop in FPGAs for Custom Computing Machines", April, 1996, Napa, California

17 Jantsch, A. Ellervee, P. Vberg, J. and Hemani, A. "A Case Study on Hardware/Software Partitioning", Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, CA, April 1994.

18 Kirkpatrick S., Gelatt Jr. C.D. and Vecchi M.P., “Optimization by Simulated Annealing”, *Science*, **220** 671-680, 1993.

19 Milne, G. Cockshott, P. McCaskill, G. Barrie, P. “Realising Massively Concurrent Systems on the SPACE Machine”, Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, April 1993, Napa, California.

20 Ngai, T, Rose, J and Wilton, S. “An SRAM-Programmable Field-Configurable Memory”, IEEE Custom Integrated Circuit Conferemce, oo 499 - 502, 1995.

21 Postula, A., Abramson, D. and Logothetis, P, "Synthesis for Prototyping of Application Specific Processors", Invited Key Note Talk, 3rd Asia Pacific Conference on HDL (APCHDL-96), Jan 8-10, 1996, Bangalore, India..

22 Postula, A., Abramson, D. and Logothetis, P, "The Design of a Specialised Processor for the Simulation of Sintering", Proceedings of the 22nd Euromicro Conference, September 2-5, 1996, Prague, Czech Republic.

23 Razdan, R. Smith, M.D. “A High-Performance Microarchitecture with Hardware-Programmable Functional Units”, MICRO-27 -11/94 San Jose, USA.

24 Sercombe, T. "A Monte Carlo Simulation of Sintered

Microstructures", Bachelor of Engineering Thesis, Department of Mining and Metallurgical Engineering, The University of Queensland, 1994.

25 Toffoli T. and Margolus, N. “Cellular Automata Machines”,The MIT Press, 1987, ISBN 0-262-20060-0.

26 Tremblay, M and O’Connor, M, “Ultra Sparc I: A Four-issue Processor Supporting Multimedia”, IEEE Micro, Vol 16, No 2, pp 42 - 50.

27 Wilton, S, Rose, J. and Vranesic, Z. “Architecture of Centralized Field-Configurable Memory”, ACM/SIGDA International Symposium on Field Programabble Gate Arrays, pp 97 - 103, 1995.

28 Wilton, S, Rose, J. and Vranesic, Z. “Memory-to-Memory Connection Structures in FPGAs with Embedded Memory Arrays”, ACM/SIGDA International Symposium on Field Programabble Gate Arrays, pp 10 - 16, Feb 1997.

29 Xilinx, Inc., The Programmable Logic Data Book, 1994

30 http://www.io.com/~guccione/HW_list.html

31 Giga Ops’ Spectrum REconfigurable Computing Platform, Giga Operations Corporation, http://www.gigaops.com

32 FireFly 6200 reconfigurable PCI Development System, Annapolis MicroSystems, http://www.annapmicro.com

33 RC1000-II, Embedded Solutions, http://www.embedded-solutions.ltd.uk

34 H.O.T.Works - The Complete PCI-XC6200 Development System, Virtual Computer Corporation, Reseda California.