

A Parallel Genetic VLSI Architecture for Combinatorial Real-Time Applications - Disc Scheduling

B.C.H. Turton, T. Arslan

University of Wales, UK

Introduction Parallel Genetic Algorithms (PGAs) have frequently been cited as an important area of research as they provide a means of rapidly developing a solution to a wide range of problems. In particular the Parallel Genetic Algorithm has the potential for solving problems far faster than a conventional Genetic Algorithm (GA). Despite these advantages real-time applications are rarely discussed in the GA literature. In principle a hardware version of the PGA could cope with such real-time problems. Turton et al [1] produced a suitable design for an image registration PGA, however the design considered was based on the conventional two-point crossover operator. Many problems are more suited to combinatorial operators such as order-based crossover. This paper proposes a new hardware based PGA using order-based crossover which will be capable of optimising a new category of real-time combinatorial problems. One of the few references to such a problem in the GA literature is Bennet's Database Query Optimisation [2] which finishes with the intention of developing a PGA solution with the hope of using their algorithm in real-time. The hardware design in this paper could provide an engine for exactly this form of problem.

In order to establish the benefits of the proposed hardware, disk-scheduling has been identified as a common real-time optimisation problem. After discussing the PGA, details of the disk-scheduling problem are discussed along with conventional solutions. This is then followed by a suitable hardware design for the order-based PGA and timing calculations. Simulation results are provided which contrast conventional and PGA results for disc scheduling. Finally conclusions are drawn from the results.

Parallel Genetic Algorithms Parallel genetic algorithms can be categorised into three types, 'standard', 'coarse grained' and 'fine grained'. Standard GAs can be implemented on a parallel architecture by distributing the evaluation process over a number of processors [3]. Coarse grained genetic algorithms run several populations of genes in parallel. After a number of generations (G) the separate populations export a set of individuals (n) to other

neighbouring populations. G generations is termed an 'epoch' or migration period. A variety of topologies have been used to define 'neighbours' typically a simple grid (Figure 1) or hypercube (Figure 2) is used with each node corresponding to a processor [4-6]. Theoretical studies of a coarse grain GA have been done by Petty & Leuze[7].

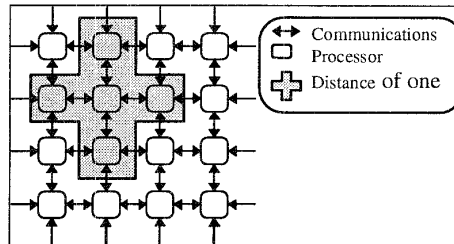


Figure 1: Grid Topology

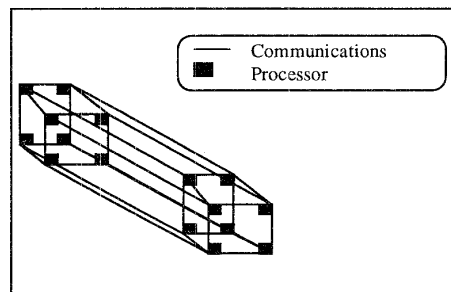


Figure 2: Hypercube Topology

(4 Dimensional)

Fine-grained parallel genetic algorithms act on each member of the population in parallel. Consequently each member of the population performs crossover with its immediate neighbours, where the neighbourhood is defined by the topology and some distance parameter [8-10] (Figure 1). Manderick's [9] work suggests that a distance parameter of two on a simple grid topology is reasonable. There is no artificial separation of the chromosomes into distinct populations with fine-grained PGAs however the neighbourhood restriction permits separation by distance. Tomassini uses a neighbourhood of four neighbours, but also permits 'migration' to greater distances. Spiessens[11] also reports that small neighbourhoods are preferable, when using tournament selection. Schwehm[12] describes a mesh connected array processor which can only successfully use elitism for short

distances, random selection of chromosomes is preferred for large distances. Coarse grain genetic algorithms are more complex than fine-grain genetic algorithms because each processor must control a complete population in the coarse-grain case, whereas for the fine-grain case only one chromosome is processed. However the fine-grain case will typically need far more processors. Goldberg's work[13] suggests that for parallel genetic algorithms the larger the population the faster the GA converges to the optimum result, this is not true of sequential genetic algorithms. Consequently for a single chip VLSI implementation, a large number of simple processors is far more attractive than a few complex processors. If a few complex processors are required then standard devices such as transputers linked together are likely to be highly efficient [3]. This paper will therefore concentrate on the fine-grain GA with massive parallelism on a single chip. The basic algorithm (Tomassini 93)[10] is given below:

Initialise system

while not done **do**

```

    evaluate f(xi)
    get fN, fS, fE, fW
    get xN, xS, xE, xW
    fim = min{fN, fS, fE, fW}
    (xi1, xi2) = xi ⊗ xim
    evaluate f(xi1) and f(xi2)
    fim = min{f(xi), f(xi1), f(xi2)}
    xi = xim
    mutate xi with probability pm

```

end while

- ⊗ - order-based crossover.
- f_i^m - fitness of the minimum valued string.
- x_i :- chromosome i.
- f() - fitness.
- x_i¹ - child 1 of chromosome i.
- x_i² - child 2 of chromosome i.
- x^N, x^S, x^E, x^W - The four chromosomes adjacent to chromosome i assuming a grid topology.
- f^N, f^S, f^E, f^W - The fitness of the four adjacent chromosomes.

The VLSI implementation in this paper incorporates Syswerda's [14] order based crossover operator and order-based mutation.

Order Based Mutation

Select two locations in the chromosome at random and swap the genes in those positions.

Order-Based Crossover Operator

```

-Select alleles (positions) at random
  (probability=0.5 per allele)
Child2=createchild(parent1,parent2,positions)
Child1=createchild(parent2,parent1,positions)
end Order-Based Crossover Operator

```

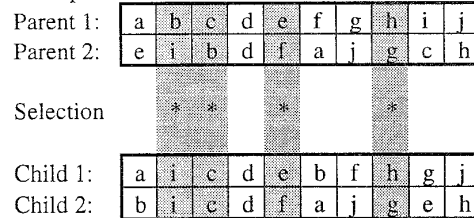
createchild(ParentA,ParentB, Positions)

```

-Identify the values at the selected Positions in ParentA
-Find corresponding values in ParentB
-Copy ParentB to Child
-Reorder the selected values in Child to be the same as the order for parent A, use the same Positions
return Child

```

Example



Disk Scheduling

Disk scheduling first became a serious issue when multiprogrammed computer systems became common-place, which resulted in high disc utilisation. The advent of client/server architectures with high capacity fibre-optic networks has continued to ensure the importance of fast and efficient disc drives. The essence of the problem is to ensure that the average disc access time is small, so that access requests can be dealt with quickly. For heavily loaded systems these requests form a queue and the order in which the requests are serviced have a profound affect on the average disc access time and its variance. The simplest strategy is the First Come First Served strategy (FCFS). In this strategy the requests are dealt with in the order they are received. It has the advantage that little processing is required and the variance is small, however the average access time is very poor compared with other strategies. Consequently it won't be discussed further. Three other strategies will be compared to the proposed PGA scheduling strategy:

SCAN : The disk head sweeps across the disc in either direction processing each request as it comes to the appropriate track. It only sweeps in one direction only until there are no further requests in that direction whereupon it reverses direction.

C_SCAN: The disk arm only sweeps across in one direction. When all requests ahead of the disc head have been serviced it jumps back to the nearest request to the outer track and proceeds inward again.

SSTF : The disk head moves to the request which minimises arm movement.

Further details can be obtained from references [15,16].

In order to use a genetic algorithm for scheduling the following encoding and fitness function is used:

Encoding- The chromosome comprises of a gene per disc access request. The order in which the genes are stored determines the order in which the requests are subsequently performed.

Fitness- The fitness is the sum of seek time plus the latency for the requests in the queue (chromosome) at any given time.

Each time a request arrives the queue is reordered using the PGA to ensure the fitness function is minimised. An upper limit of thirty two requests are dealt with in the genetic algorithm (similar to N-step SCAN where $N=32$).

Hardware Implementation

An overall view of the hardware configuration for a single order-based genetic processor (OGP) is illustrated in figure 3. During the normal operation of the OGP, a typical sequence of events would commence by depositing a chromosome, *chromoi* (representing a sequence of events to be scheduled), into the register REG0. This will represent the individual x_i mentioned earlier. The same chromosome will be copied to register REGP1 simultaneously. The next step will require the selection of the best chromosome from the neighbouring four OGPs. This is performed through the multiplexer MX1 through which the best neighbour will be transferred to the register REGP2. Both of the later two registers will contribute to the crossover operation, performed by the block XOL (see later), resulting in the two children x_i' and x_i'' being deposited in the registers REG1 and REG2 respectively. The random crossover positions will be externally provided to the OGP through the input pins P1, P2, . . . Pn (in our implementation we have restricted n to 32 events only). The likelihood's of both crossover and mutation operations are signalled by the different values on the external input M/X. At this stage, registers

REG0-REG2 will contain x_i , x_i' and x_i'' respectively. The fitness logic circuit will be used to evaluate $f(x)$ for each of the chromosomes (directed through multiplexer MX5) and the resulting values will be deposited in the corresponding fitness register (FREG0-FREG2). Appropriate register selection and path direction are performed by purely combinational blocks (Register Control Logic (RCL) and the logic of block L) which in turn control the demultiplexing of DX1 (since this also controls the transfer of fitness values from neighbouring chromosomes - see above).

Finally, if mutation is signalled at M/X, the mutation operation logic block, MOL, will be activated (see later). The rest of this section will briefly comment on the circuit configuration for the main genetic operations.

Order Based Crossover

A detailed look inside the XOL block, mentioned above, will reveal the circuit configuration illustrated in figure 4. The circuit is a hardware interpretation of the function `createchild(parent1, parent2, positions)`. Parent1 and Parent2 are the contents of the registers REGP1 and REGP2 respectively and the positions are given by P1-Pn.

Crossover will commence by duplicating parent2 into the register REGC1 (child 1) through MX8. Next each position in the REGC1 is compared to the content of *all* the specified positions in REGP1 (using COMP). If the result of the comparison is positive, then the content of position P1 is copied to the considered position in REGC1 (this is tracked by a counter C which directs the copying through DX2 into REGC1). For the next position matching in REGC1, the position corresponding to P2 in REGP1 will be copied. This process will continue till all the specified positions are deposited in REGC1.

The shaded area in figure 4 indicates that a similar circuit configuration is used for `createchild(parent2, parent1, positions)`. Crossover is terminated when the contents of the child registers REGC1 and REGC2 are copied to REG1 and REG2, see figure 3. In addition to the above a number of clock signals are provided for sequencing the crossover operation.

Order Based Mutation

The mutation hardware, MOL, consists of a simple Multiplex-Demultiplex system pair at the inputs/outputs of the final register (REG2) in figure 3. The two random positions P_x and P_y are used to select the appropriate two

locations through the above multiplexing system and later duplicating these in their respective reverse locations.

Fitness Evaluation

As described earlier in the disc scheduling section, the fitness calculation is merely the addition of the differences of consecutive scheduling events (seek time+ latency). Therefore, the fitness logic block, in figure 3, will mainly consist of a sequence of adders. The literature is very rich with different types of adder architectures the choice of which involves a trade-off between speed/chip-area in addition to the precision required[17].

Practical Considerations

In order to evaluate the design a 1 μ m CMOS process is used. A single processor would take approximately 0.016 milliseconds in processing a chromosome. Simple addition operations for calculating the fitness function will not add significant time to this calculation. However more complicated evaluation functions could become the dominant factor (for example image registration [1]). Typically one hundred generations are required to complete an optimisation, which brings the total time to 1.6 milliseconds per optimisation.

In addition the number of bits in a register will depend on the range of values a gene can take and the number of genes held in a chromosome. For the disk scheduling case a gene holds the coordinate of the request (track + rotational position) and the chromosome holds the requests that need to be scheduled. A 24 bit value per request and thirty two requests to be ordered at any one time is reasonable.

The speed of processing is very much dependant on the target silicon process utilised. The 1 μ m CMOS process above is only used as an example of the typical processes which could be accessed relatively easily in both industrial and academic uses. Much improved performance could be obtained by using a process with smaller feature size (such as the mietec 0.7 μ m) or targeting Gallium Arsenide technologies.

Results

Results were obtained by simulating the PGA for 4000 disc access requests. All four of the algorithms mentioned earlier (SCAN, CSCAN, SSTF and the new GA scheduler) were tested against the test data (figures 5-7). The following simulation conditions applied:

- Request interarrival times followed an exponential distribution (4000 requests simulated)
- Requests are evenly distributed across the tracks.
- File request are for fixed sizes of records with negligible transmission time when compared with latency plus seek time.
- Single disk drive with dedicated controller and channel.
- Seek time is a linear function of seek distance, and is equal to $4000+15*(\text{No of tracks moved}) \mu\text{S}$.
- Number of tracks=1200.
- No distinction between read & write requests.
- Rotational speed =3600 rpm.

SSTF and SCAN have uneven distributions across the track (SSTF also has a large access time variance) making them unsuitable. CSCAN behaves well but none of the techniques are as effective as the genetic algorithm. The genetic algorithm has used rotational information to improve its performance. Since the PGA can have a variety of fitness functions it will often be able to out-perform other techniques if only by virtue of optimising the most appropriate parameter.

Conclusions

The hardware implementation of this order-based genetic algorithm is capable of scheduling disc-access requests in approximately 2 milliseconds. Since the minimum seek time is four milliseconds the system can easily cope with real-time scheduling. Figures 5-7 indicate that the PGA is more effective than any of the other techniques for optimising the average access time. Should other criteria become desirable then the fitness function should be replaced by one which reflects the new goals. This requires no fundamental change to the PGA section of the hardware, however the fitness function evaluation time must be considered when developing a new version of the system. In addition to disc scheduling such processing power is quite capable of dealing with many other real-time applications provided the function evaluation can be done in reasonable time. An example of this is database query optimisation. Further work is in progress on developing the hardware for more advanced forms of parallel genetic algorithm and identifying other application areas. This hardware, if mass produced, should provide the potential for cheap self-optimising systems using an algorithm which at present is mainly limited to off-line general purpose computers.

References

1. Turton B.C.H, Arslan T, Horrocks D.H (1994) 'A Hardware Architecture for a Parallel Genetic Algorithm for Image Registration' Colloquium on Genetic Algorithms in Image Processing and Vision Digest No 1994/193 pp 11/1 11/6
2. Bennet K, Ferris M.C, Ioannidis Y.E (1991) 'A Genetic Algorithm for Database Query Optimisation' in Proceedings of the Fourth International Conference on Genetic Algorithms Booker B.L, Belew R.K (Eds) Morgan Kaufmann:California pp 400-407
3. Fogarty T.C and Huang R (1990) 'Implementing the Genetic Algorithm on Transputer Based Parallel Processing Systems' in Parallel Problem Solving in Nature Scwefel H.P & Manner R (Eds) Springer Verlag:NY pp 145-49
4. Tanese R (1989) 'Distributed Genetic Algorithms' in Proceedings of the third International Conference on Genetic Algorithms' Schaffer J.D (Ed) Morgan Kaufmann Publishers pp 434-39
5. Cohoon J.P, Martin W.N and Richards D.S (1990) 'Genetic Algorithms and Punctuated Equilibria in VLSI' in Parallel Problem Solving in Nature Scwefel H.P & Manner R (Eds) Springer Verlag pp 134-141
6. Muhlenbein H, Schomisch M and Born J (1991) 'The Parallel Genetic Algorithm as a Function Optimizer' in Proceedings of the Fourth International Conference on Genetic Algorithms Booker B.L, Belew R.K (Eds) Morgan Kaufmann:California pp 271-78
7. Petty C.C and Leuze M.R (1989) 'A Theoretical Investigation of a Parallel Genetic Algorithm' in Proceedings of the third International Conference on Genetic Algorithms Schaffer J.D (Ed) Morgan Kaufmann Publishers pp 398-405
8. George-Schleuter M (1990) 'Explicit Parallelism of Genetic Algorithms through Population Structures' in Parallel Problem Solving in Nature Scwefel H.P & Manner R (Eds) Springer Verlag:NY pp 150-59
9. Manderick B and Spiessens P (1989) 'Fine-Grained Parallel Genetic Algorithms' in Proceedings of the third International Conference on Genetic Algorithms' Schaffer J.D (Ed) Morgan Kaufmann Publishers pp 428-433
10. Tomassini M (1993) 'The Parallel Genetic Cellular Automata: Application to Global Function Optimization' in Artificial Neural Nets and Genetic Algorithms, Albrecht R.F, Reeves C.R & Steele N.C (Eds) Springer-Verlag:New York pp 385-391
11. Spiessens P and Manderick B, 'A Massively Parallel Genetic Algorithm. Implementation & First Analysis' in Proceedings of the Fourth International Conference on Genetic Algorithms Booker B.L, Belew R.K (Eds) Morgan Kaufmann:California pp 279-86
12. Schwehm M (1993) 'A Massively Parallel Genetic Algorithm on the MasPar MP-1' in Artificial Neural Nets and Genetic Algorithms, Albrecht R.F, Reeves C.R & Steele N.C (Eds) Springer-Verlag:New York pp 503-7
13. Goldberg D.E, (1989) 'Sizing Populations for Serial and Parallel Genetic Algorithms' in Proceedings of the third International Conference on Genetic Algorithms' Schaffer J.D (Ed) Morgan Kaufmann Publishers pp 70-9
14. Syswerda G (1991) 'Schedule Optimization Using Genetic Algorithms' In Davis L (ed) 'Handbook of Genetic Algorithms', Van Nostrand Reinhold, 332-349
15. Teory T.J (1972) 'Properties of Disk Scheduling Policies in Multiprogrammed Computer Systems' Proceedings AFIPS FJCC 41 pp 1-11
16. Teory T.J, Pinkerton T.B (1972) 'A comparative Analysis of Disk Scheduling Policies' Communications of the ACM 15 No 3 pp 177-184
17. Hwang K (1979) 'Computer Arithmetic. Principles, Architecture & Design' J.Wiley:New York.

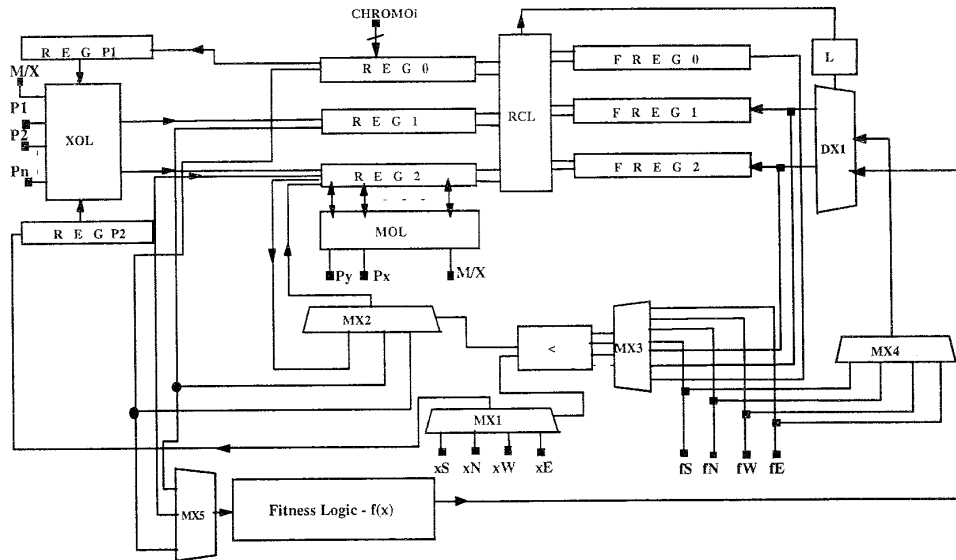


Figure 3: The main blocks of a single processor

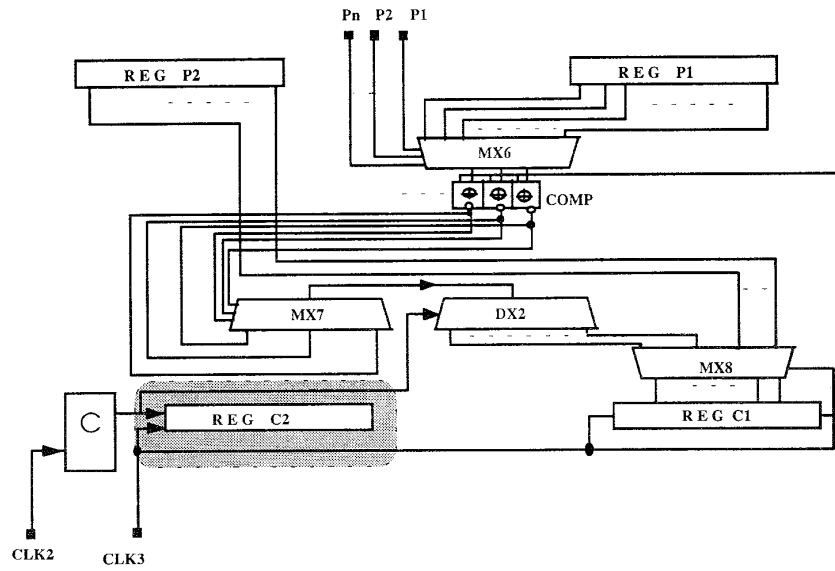


Figure 4: The production of the first child using an order-based-crossover