Application-Specific Memory Interleaving Enables High Performance in FPGA-based Grid Computations[1]

Tom VanCourt and Martin C. Herbordt
Boston University, ECE Department
8 St. Mary's St., Boston MA 02215
{tvancour, herbordt} @ bu.edu

Abstract

Current generations of FPGAs create possibilities for innovative, application-specific computation pipelines. In many cases, the pipeline can fully exploit the FPGA's parallelism only when multiple operands are available concurrently, requiring clusters of values to be fetched from memory. These clusters of values often have fixed organization, as in the eight grid points around an off-grid position that are needed for 3D interpolation of a value at that position. We present a technique for creating custom interleaving of the FPGA's on-chip memories, giving access to the entire cluster of values in one memory cycle. This technique works on grids of 2, 3, or more dimensions, on many non-rectangular grids, and on cluster organization specific to each application. We report the initial version of a design tool that inputs the relative positions of grid points in the access cluster, and produces synthesizable HDL code for the custom-interleaved memory.

1. Introduction

FGPA accelerators are now entering the main stream of high performance computing. To date, these applications have generally been hand coded by skilled logic designers as point solutions to specific problems or families of problems. Unlike software-based application development, with over fifty years of experience and accepted practice, FPGA-based application development suffers a near-total lack of tools and techniques applicable across many families of computing problems.

This work addresses FPGA-based computations that use regular grids of data points, where each step of the computation uses a cluster of nearby points. In the applications of interest, full use of the FPGA's potential parallelism requires concurrent access to all of the points in that cluster. We present an orderly technique that creates custom-interleaved memories for specific clusters, constructed so that all points in that application's access cluster can be fetched in one memory cycle. This technique takes full advantage of the FPGA's many independently addressable on-chip RAMs, its high degree of fine-grained parallelism, its ease of application-specific configurability, and its capacity for data path widths into the thousands of bits. This technique is applicable to grids of 1, 2, 3, or more dimensions and to access clusters of arbitrary layout, making it a widely reusable tool for FPGA-based computing. In order to simplify use of this kind of memory interleaving, we have built a tool for creating application-specific interleaved memories and matching testbenches.

2. Motivating applications

Grid-based calculations with regular access clusters arise in many kinds of applications. Bi- and tri-linear interpolation, based on clusters of points at the vertices of a square or cube, occurs in volume rendering and ray-casting algorithms. Interpolation also occurs in physics and chemistry applications, where forces computed at grid points must be applied to off-grid particles. Red-black relaxation of nonlinear systems examines the four orthogonal neighbors around each point in the grid. Image processing applications use morphological operators and convolutions, which access some (possibly sparse) set of points in an image grid. Cellular automata in one, two, or three dimensions perform updates based on different patterns of neighboring values.

3. Related work

Memory interleaving has been used since the 1960s, either for *broad* parallel access to multiple words at the same time [1], or for *pipelined* parallelism of single accesses with overlapping latency periods [2]. Either way, the intent is to improve memory performance by avoiding concurrent, interfering accesses to any one memory bank. General purpose processors can not take advantage of knowledge about any of their applications, however. Also, their fixed structure makes it impossible to adjust to the different behavior of different applications. In contrast, developers of

[1] This work was supported in part by NIH award #RR020209-01 and was facilitated by donations from Xilinx Corporation

FPGA-based applications have intimate knowledge of each application, and assume that they will reconfigure the FPGA's on-chip memories for each application.

Allocation of multiple memories for FPGA-based systems is also well studied. For example, tools have been developed for automated placement of different arrays into different memories, in order to reduce the number of memory cycles needed [3] for calculations involving multiple arrays and variables. Our work differs in that its goal is to partition a single array so as to guarantee single-cycle access to specific subsets of array elements.

4. Example: Bilinear interpolation

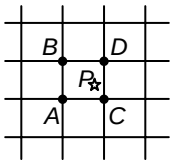

Figure 1. Cluster of grid points for computing *P* by bilinear interpolation

Fig. 1 shows a candidate for memory interleaving, the cluster of points *A*, *B*, *C*, *D* needed for bilinear interpolation of point *P*. One *X* axis coordinate in the cluster is odd and the other is even, and the same is true in the *Y* axis. It is clear, by inspection, that the grid points can be partitioned by their (*X*, *Y*) indices into an array of (even, even), (even, odd), (odd, even), (odd, odd) memory banks. In that case each access to a four-point cluster requires one point from each RAM bank.

If the cluster of Fig. 1 spans *X* coordinates 3 and 4, then points *A* and *B* must come from odd-*X* memory banks, and points *C* and *D* from even-*X* banks. A different cluster covering *X* coordinates 2 and 3 would have the opposite mapping of *A*,*B* and *C*,*D* to even-*X* and odd-*X* memory banks. The same is also true in the *Y* axis, so point *A*, according to the coordinate base of the cluster, may be supplied by any of the four memory banks. This is easily handled with a multiplexer downstream of the memory array for each of the output values *A*, *B*, *C*, *D*.

Since the least significant bits (LSBs) select the RAM from which a value is taken, RAM addresses are based only on the $\lfloor X/2 \rfloor$ and $\lfloor Y/2 \rfloor$ values, i.e. stripped of their LSBs. A cluster that covers $X \in \{3, 4\}$ has the same $\lfloor X/2 \rfloor$ for all *X* coordinates, but a cluster covering $X \in \{3, 4\}$ has must compute RAM addresses using two different values of $\lfloor X/2 \rfloor$. This means that RAM addresses are computed differently for each RAM, according to the LSBs of the indices in each dimension – a conditional increment of the *X* value used for address generation, according to offset of the cluster relative to the RAM array.

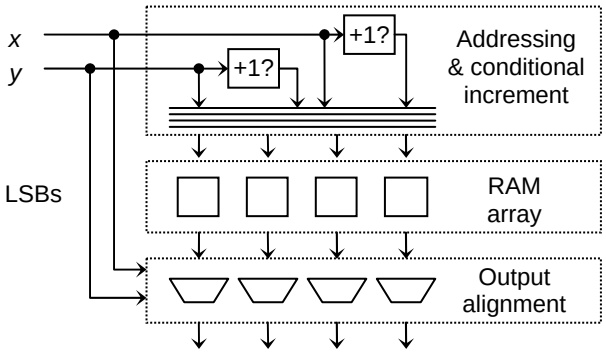

Figure 2. Interleaved memory for broad

Figure 2 shows the general structure of the interleaved memory: address generation for each of the RAMs, the RAM array itself, and output multiplexing to align RAM output to specific members of the access cluster. That structure is the same for any number of dimensions of indexing, and for any organization of output cluster. Different implementations will differ in the number of RAMs, the number of LSBs chosen, and the number and organization of output multiplexers. These specifics depend on the application-specific number of indexing dimensions, the extent of the access cluster in each dimension, and the number of points in the access cluster.

5. Conclusions

We present an orderly technique for creating custom parallel-access memories for a range of applications, using design resources that are plentiful in current FPGAs. These address the needs of grid-based calculations in which a cluster of nearby points must be accessed for one step of a calculation. Such applications occur widely in image processing, data visualization, numerical computing, computational chemistry, and other fields. As a result, memory structures in this family have potentially wide applicability. In order to make this technique widely accessible, we have implemented some of its variations in a Java-based design tool, which is available at `http://www.bu.edu/caadlab`.

6. References

[1] G. H. Barnes, R. M. Brown, M. Kato, D. J. Kuck, D. L. Slotnick, and R. A. Stokes. "The Illiac IV Computer". IEEE Transactions on Computers 17(8), August 1968

[2] Control Data Corporation. *Control Data 6400/6500/6600 Computer Systems Reference Manual*. 1969.

[3] M. B. Gokhale and J. M. Stone. "Automatic Allocation of Arrays to Memories in FPGA Processors With Multiple Memory Banks." Proc. FCCM 1999