Accelerating Floating Point DGEMM on FPGAs.

Martin Langhammer, Tom VanCourt.

Altera Corporation

mlangham@altera.com
tvancour@altera.com

Introduction

Recent generations of FPGAs, together with innovation in synthesis of floating point data paths, open new possibilities in performance computing. For example, Altera's Stratix ® 3SE260 can sustain 50GFLOPs of 64-bit double-precision (DP) IEEE 764 floating point computation. This report describes a matrix multiplication core that approaches that performance, limited by the data rate of the Hypertransport 3.0 channel that delivers results to the host's main memory.

The arithmetic core of this application was built using an experimental floating point core builder. Typical FPGA floating point applications use separate, fixed blocks for each arithmetic operator, irrespective of the expression in which each operator appears. This core generator is different. It starts with entire expressions and sets of expressions written in ANSI C, and creates a fused, fully pipelined floating point data path specific to that application. This lets the core builder examine the context in which each operator appears and generate operator logic specific to that expression context. The result typically reduces latency (in clock cycles) and logic utilization by up to 50%, compared to naïve assembly of operator blocks, while keeping clock rates of 200 MHz or more. The reduction in logic translates directly into a reduction in power for a given computation, or an opportunity for more functions to be implemented in a given amount of FPGA fabric.

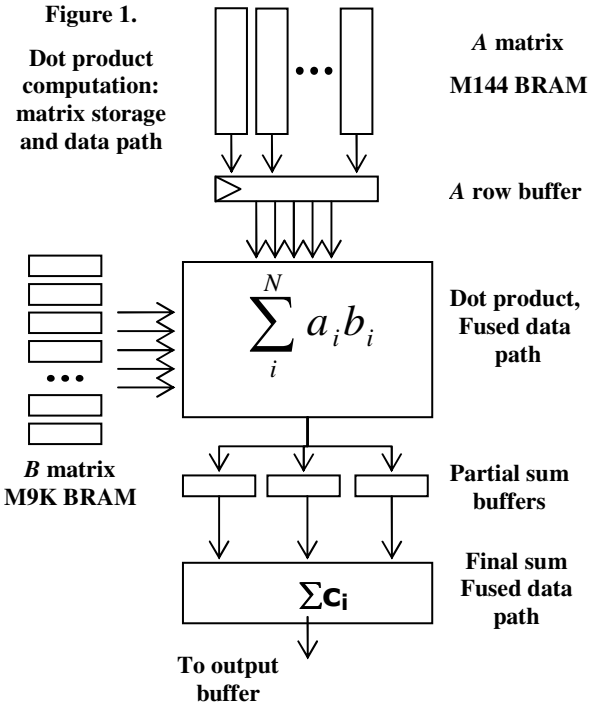
DGEMM Implementation

This implementation can be tailored at compile time to handle matrices of arbitrary size, as long as both of the input matrices can fit into on-chip RAM. The matrix multiplication A□B is subdivided into vector dot products with scalar results. A rows and B columns are subdivided into vectors, and elements of the result matrix are sums of a set of dot products. In order to compute partial sums, one row vector of the A matrix is held steady while successive column vectors of the B matrix are fetched. The partial sum is the dot product of A's row and B's column. Dot product logic, described below, takes a new vector-pair at every cycle and, after pipeline latency, delivers one scalar per cycle. The dot product length can be configured, and the current library supports vector lengths of 32, 64, 96, and 128 DP values. Other lengths could be added, but were not needed for the proof-of-principle implementation.

Matrix sizes are not limited by the size of the dot product length. The implementation uses blocking to support any matrix sizes that are multiples of the dot product length. The A matrix is decomposed into a large number of matrixes, where each row contains multiple matrixes, each one row by dot product length columns. The B matrix is composed of a smaller number of matrixes, each dot product length rows by the number of columns in B. Block results are stored in a local cache, which are summed once the first element of the last block of the current group of blocks has been written to the cache.

Input matrices are interleaved across multiple on-chip RAM banks, allowing concurrent access to multiple elements of each array. Column vectors of the B matrix are interleaved across the 3SE260's on-chip M9K RAM banks so that every element of the row vector is fetched in a single cycle, up to 128 DP values (2K bytes) in the current implementation. Sequencing logic presents one entire column vector from the A matrix to the dot product core on every cycle. At the same time, the next row vector from the A matrix is readied using multi-cycle access to the M144 on-chip RAMs, overlapped with the multi-cycle computation of a row of the output matrix. Figure 1 illustrates organization of the A and B memories with respect to the dot product core. There are 844 independently addressable M9K RAM blocks on the 3SE260, each with

Figure 1.



9K (9216) bits total, configurable in word widths to 36 bits. The 3SE260 also contains 48 independently addressable M144 RAMs, each with 144K (147,456) total bits and word widths up to 72 bits.

The sequencing logic buffers result values to be sent from the accelerator to the host's main memory. This overlaps computation as much as possible, but can stall the floating point pipeline when output buffers fill.

The dot product core

Figure 2 shows the ANSI C code for the dot product (except for normal declarations), using the length-32 core as an example. This core is fully pipelined, so it accepts a new vector-pair (two length-32 vectors, or 64 DP values for this example) per cycle. After pipeline latency, the core delivers one scalar DP result per cycle.

```
dp00 = ((xx00*cc00 + xx01*cc01)  +
        (xx02*cc02 + xx03*cc03)) +
       ((xx04*cc04 + xx05*cc05)  +
         (xx06*cc06 + xx07*cc07));
dp01 = ((xx08*cc08 + xx09*cc09)  +
        (xx0a*cc0a + xx0b*cc0b)) +
       ((xx0c*cc0c + xx0d*cc0d)  +
        (xx0e*cc0e + xx0f*cc0f));
dp02 = ((xx10*cc10 + xx11*cc11)  +
        (xx12*cc12 + xx13*cc13)) +
       ((xx14*cc14 + xx15*cc15)  +
         (xx16*cc16 + xx17*cc17));
dp03 = ((xx18*cc18 + xx19*cc19)  +
        (xx1a*cc1a + xx1b*cc1b)) +
       ((xx1c*cc1c + xx1d*cc1d)  +
         (xx1e*cc1e + xx1f*cc1f));
result = ((dp00+ p01) + (dp02+dp03));
```

Figure 2. Length-32 dot product, input to floating point block builder

This ANSI C code, plus some wrapper declarations also coded in ANSI C, is input to the experimental floating point core builder. The core builder honors parentheses for enforcing order of evaluation. As a result, this code evaluates one dot product by performing 32 multiplications in parallel, then feeding the 32 products to tree adder. Because summation uses a tree adder, total latency of the pipeline grows only as the log of the vector length. Table 1 shows latencies for the library's current dot product cores.

Conclusions

This case study examines FPGA-based acceleration of matrix multiplication, using double precision IEEE floating point arithmetic. It uses an experimental tool for building the floating point core of the computation, and can achieve DP computation rates up to 47.46 GFLOPs, until throttled by HyperTransport's rate of sending results to host memory, a theoretical maximum of 20.8 Gbyte/sec.

Table 1. Dot product latency vs. vector length

Vector length	Latency (cycles)
32	41
64	46
96	51
128	55

FPGAs have traditionally been considered "bad" at IEEE floating point arithmetic. We hope that myth can be laid to rest. At the same time, the FPGA's massive on-chip memory bandwidth and native parallelism allow fast execution of operations that require up to hundreds of operands at a time. We look forward to other applications of the floating point core builder, and to even higher performance on the new Stratix IV generation of FPGAs.