MEMOCODE 2007 Co-Design Contest

Forrest Brewer
ECE Dept. UC Santa Barbara
forrest@ece.ucsb.edu

James C. Hoe
ECE Dept. Carnegie Mellon University
jhoe@ece.cmu.edu

Abstract

New to the 2007 MEMOCODE conference is the HW/SW Co-Design Contest. Members of the technical and steering committees from MEMOCODE 2006 thought that the co-design practice is distinct from conventional hardware or software design practice. A co-design contest was conceived to help elucidate these issues and to foster greater interest in the design aspects of the MEMOCODE conference. The contest would also serve to showcase advances in co-design tools and methodologies. Below, we describe briefly the development of the contest format and the thinking behind the choices. The actual call-for-participation that appeared on the contest website (http://memocode07.ece.cmu.edu/contest.html) is reproduced in the appendix section.

1 Contest Rationale

Guidelines. The purpose of the contest is to popularize co-design as a particular science. For this reason, the contest cannot be very short, like a CADathlon (software) contest, nor can it be too long like a conventional hardware design contest. Further, since there are many ways to do co-design, the contest needs to be organized and graded in a way that helps to develop the aims of the co-design community—not simply a particular technique in a niche market. Since co-design techniques span such a variety of design scales, techniques, and applications, we need a design problem which is widely understandable but for which there are not obvious solutions or well-developed product libraries. The issue here is to try to level the playing field to avoid canned solutions which, although practical, do not document the design rationale that lead to their construction. Lastly, the co-design problem should require meaningful creation of both hardware and software components and allow exploitation of existing co-design flows as they are applicable.

Contest Term. The issue here is ensuring that there is sufficient time to complete a design after learning the issues particular to a given HW/SW co-execution platform. Unlike software, where development can be made portable across many execution platforms, the common practice in co-design is to expose the specific hardware features so that a design can make best use of the platform. On the other hand, few researchers are willing to lend several of their students to participate in a contest for periods longer than a few weeks at the most. To make a contest feasible, we decided to keep the term relatively short and to provide a working code base for at least one inexpensive and easily available platform. We ultimately opted for a problem that we gauged could be done well in a week and allowed the contestants a month lead-time to solve it.

Execution Platform. We did not wish to limit the competition to a particular execution platform since we thought that many design groups would be more familiar with their preferred platforms. Instead, we left the choice of platform to the contestants and provided a grading metric that attempts to equalize the differences. We did, however, select the Xilinx XUP2VP prototyping board as the reference platform. The XUP2VP board provides a large variety of interfaces, is well supported, and is inexpensive. The on-board Virtex 2 FPGA (XC2VP30) has a substantial programmable logic fabric, a fair amount of on-chip memory, and two PowerPC 405 embedded processors. Further, there is a DDR slot which can be used to provide adequate system memory to support a solid software execution platform on the embedded PowerPC 405 processors. For the XUP2VP environment, we provided reference starter design materials, including a software-only reference solution and a reference HW/SW interface library comprising ready-to-use C functions and Verilog modules.

Design Task. The design problem ultimately chosen is the Blocked Matrix-Matrix Multiplication. The basic algorithm, though fundamentally simple to understand, lends itself to a large space of high-leverage optimizations in managing data movement, storage, and bandwidth. We designed the problem to required hardware involvement in the design as well as a software component. To enforce software, execution time is measured on matrix multiplication from from DRAM storage back to DRAM storage, and the location of the input and output matrices is under the soft-

ware's purview. In addition, the implementations are required to handle a range of matrix sizes without hardware reconfiguration. To enforce hardware, the matrix multiplication operates on matrices of 16-bit fixed-point complex data values, thus placing software arithmetics at a significant performance disadvantage relative to hardware. Again, the goal is to force a solution that must balance a mix of software and hardware elements to succeed.

Grading and Judging. In this inaugural contest, we focused on performance as the primary metric of merit. The metric used is the speedup achieved by the contestants' design relative to the official software-only reference design on their platform of choice. This normalized metric attempts to even the scale when dissimilar platforms are used. However, the performance metric is formulated to not give advantage to platforms slower than the XUP2VP, as several important issues disappear if the clock speed and fabric speed is made artificially slow.

We further asked the contestants to describe the design in sufficient detail so that a panel of judges could make subjective decisions about the 'elegance' of the approach as well as the efficacy of a particular design. We felt that it was important for the judging panel to be able to reward a unique or novel solution even if its overall performance is less competitive.

2 Summary of Outcomes

Since this was the first year of the contest, publicity was crucial. In addition to the MEMOCODE audience, we also advertised at FPGA and related EDA conferences in the few months before the March contest period. Posters and fliers were designed and distributed at the conferences. In addition, we contacted select candidates likely to field a competitive team. One issue we found was the time proximity to DAC and to other conferences of interest to designers in this area. (At least 2 groups did not enter because of this). At the start of the contest, seven teams from U.S. and Europe had registered for the reference design materials.

Since the contest was sanctioned by the MEMOCODE conference, it was felt that some recognition of the contestants was necessary. We had planned to invite the top three entrants to come to the conference for presentations and for an open panel of judges to decide the winner at the conference. Ultimately, only two teams, from MIT and Virginia Tech respectively, successfully submitted a final design. Both teams are invited to present at the conference. Brief write-up of their solutions are also included in the formal conference proceedings. In addition to a cash prize to be awarded at the conference, the organizing committee are providing travel grants to offset partially the travel cost of student team members.

Although the contest did not have the number of entrants as we had hoped, the two finished entries are of very high quality and offer real technical contributions to share with the conference audience. We believe the co-design contest adds an important new dimension to the MEMOCODE conference and should remain a valuable component of the MEMOCODE conference in the future.

In retrospect, much can be improved. For the future contest organizers, we offer three suggestions from this inaugural experience. First, the one-week term may be too optimistic. Feedback from one group indicated that they had issues getting the reference design running. The reference tool and platform should be announced well ahead of time to give contestants ample opportunity to become familiar with the design and execution environment. Second, the timing of MEMOCODE07—nearly time-adjacent to DAC, but 8000 miles away—is known to have stopped at least two groups from entering the contest. Careful consideration of time and location is of course crucial to the entire conference. Third, the publicity in both conference and web venues can be strengthened. Greater awareness should also be built through word-of-mouth advertising by the program committee and conference attendees.

3 Acknowledgments

We would like to acknowledge Nokia and Bluespec. Inc for sponsoring this contest. We would like to thank Roland Wunderlich for providing the reference software implementation of the matrix-matrix multiplication.

4 Appendix: Call for Participation

The text from the actual call for participation is reproduced in this section, detailing the design problem and grading criteria.

4.1 Overview

Matrix-matrix multiplication (MMM) is a linear algebra operation familiar to everyone. This design challenge emphasizes your HW/SW co-design methodology's ability to explore algorithmic alternatives, to fine-tune performance, and to create quickly a HW/SW co-designed/optimized implementation.

The implementation of a basic triple-loop square MMM is given below. The first three arguments A, B and C are pointers to the square N-by-N multiplicand, multiplier and product matrices, respectively. The fourth argument N specifies the number of rows/columns in the square matrices. A, B and C are row-major. The implementation below assumes C is zero initialized. This implementation performs N^3 multiplications and N^3 additions.

```
void mmmKernel(Number* A, Number* B,
               Number* C, int N) {
  int i, j, k;
  for (j = 0; j < N; j++)
    for (i = 0; i < N; i++)
      for (k = 0; k < N; k++)
        C[i*N+j] = C[i*N+j] +
                   A[i*N+k]*B[k*N+j] ;
}
```

The above straightforward implementation has poor data reuse locality, which becomes a performance concern when the entire data set cannot be kept in a nearby fast memory. This can occur in a software scenario when the data set cannot fit in the cache of the processor; this can also occur in a HW/SW co-execution scenario when the data set cannot fit within the hardware accelerator. In both scenarios, the data set must be transferred frequently and redundantly between the nearby fast memory and a slower backing-store (e.g., DRAM).

To improve data locality, a 'blocked' MMM breaks down the computation into a series of small MMMs of NB-by-NB regions in the original N-by-N multiplicand and multiplier matrices. Blocked MMM also performs N^3 multiplications and N^3 additions but has better data locality (requiring fewer data transfers between the fast near-by memory and the slower backing store). A blocked MMM implementation is given below with an additional argument, NB, the blocking size. This implementation assumes N is an integer multiple of NB. Again, this implementation assumes C is zero initialized. This implementation assumes the existence of a new MMM kernel (which could be a basic triple-loop implementation) to multiply the sub-blocks. For complete details, please see the reference software-only implementation released with the contest.

```
void mmmBlocked(Number* A, Number* B,
          Number *C, int N, int NB) {
  int j, i, k;
  for (j = 0; j < N; j += NB)
    for (i = 0; i < N; i += NB)
      for (k = 0; k < N; k += NB)
        mmmKernel(&(A[i*N+k]),
                  &(B[k*N+j]),
                  &(C[i*N+j]), N, NB);
}
```

4.2 Design Task

Starting from the reference software-only blocked MMM implementation (see submission instructions), you are to develop a HW/SW co-designed optimized implementation that partitions the computation onto a processor running software and the hardware datapath to be implemented on an FPGA fabric. For starters, `mmmKernel( )` is a good candidate to accelerate in HW, and the choice of NB is a tunable parameter. You might want to explore non-square blocking. You can also consider more effective data management to take advantage of on-FPGA memory capacity. Your implementation should be optimized for 1024-by-1024 and also 256-by-256 sized matrices. Your implementation must be able to support 2-power matrix sizes, N=64, 128, 256, 512, and 1024, without reprogramming the FPGA fabric.

Like the reference implementation, your design must support 16-bit fixed-point, complex data values. That is, each matrix element is 32-bits: the more significant 16 bits are used for the real component and the less significant 16 bits are used for the imaginary component. The multiplier matrix (B) and the product matrix (C) use the same 16-bit 2's-complement fixed-point format in the real and the imaginary component. The multiplicand matrix (A) uses a 16-bit 2's-complement fixed-point formant with 14 bits for fraction. You can assume the multiplicand matrix contains real and imaginary values that are between 1 and -1 inclusive. You can also assume the test cases would not cause overflows in the reference software-only implementation.

4.3 Implementation Platform

You may use any HW and SW design methodology at your disposal. Formal methods are encouraged but not required.

You may use any FPGA development platform at your disposal. The FPGA platform must permit a processor running software (e.g., the embedded PowerPC405 in a Xilinx Virtex-II Pro FPGA or a Xilinx Microblaze soft-core) to communicate with the hardware accelerator datapath on an FPGA fabric. The processor and/or the FPGA should have access to off-chip memory sufficient to hold the multiplicand, multiplier and product matrices, up to 1024-by-1024. In your implementation, the multiplicand matrix and the multiplier matrix must start from the off-chip memory, and the product matrix must complete in the off-chip memory.

The reference platform supported by the contest is the Xilinx XUP development board (http://www.digilentinc.com/Products) with 512MB of DRAM. For those of you using XUP, we provide a reference EDK project for the Xilinx XUP board that implements a basic interface library to enable communication between the software running on the 300MHz embedded PowerPC405 and the 100MHz FPGA fabric through the DSOCM interface (150+MB/sec bandwidth). The interface is based on two circular memory FIFOs, one for each direction of data communication between the hardware and the software. You may develop or acquire any other interfacing scheme you prefer.

4.4 Contest Judging

The contest will be judged in two parts:

First, an objective element of judging is based on the combined execution time (wall-clock) of a 1024-by-1024 MMM and a 256-by-256 MMM. Second, an subjective element of judging is based on the 'elegance' of the solutions.

The execution time will normalized for platform differences. The effective execution time will be:

$$Time_{effective} = (Time_{measured N=1024} + 64 \times Time_{measured N=256}) \times f_{CPUspeed} \times f_{FPGAcapacity} \times f_{FPGAspeed}$$

where

$$f_{CPUspeed} = min(1,\ Time_{SW,\ XUP}/Time_{SW,\ yourCPU})$$
$$f_{FPGAcapacity} = min(1,\ gates_{YourFPGA}/gates_{XC2VP30})$$
$$f_{FPGAspeed} = min(1,\ frequency_{YourFPGA}/100MHz)$$

Note, this normalization penalizes platforms that are more capable than the XUP but deliberately does not benefit less capable platforms. Some factors that the normalization calculation does not take into account are 1. HW/SW communication bandwidth and latency, and 2. off-chip memory bandwidth and latency. This normalization is not perfect; hence 'gaming the system' is fair game. In the case an entry is based on an extraordinary FPGA platform, the judging panel reserves the right to determine a fair normalization on a case-by-case basis.

You need to prepare a brief documentation describing 1. your FPGA platform, 2. your design methodology, 3. the organization of your design and its theory of operation, 4. a brief analysis of its performance (e.g., where are time spent and where are the bottlenecks). A panel of judges will assess subjectively the elegance of the solution and rank the entries accordingly. The decisions of the judges are final. (Documentations in the form of PowerPoint slides are perfectly acceptable. Keep in mind, the judges' subjective sense of elegance is likely to be influenced by the quality of the documentation.)

The entries are ranked overall by the value ($Rank_{Time\,effective} + Rank_{elegance}$). In the case of a tie, Time-effective is the tie-breaker. The top three finishers will have the opportunity to present their design and design methodology in MEMOCODE07's technical program. All entrants are invited to submit a poster for MEMOCODE's Poster Session. We will offer a cash prize for the winning design (from the top three) selected at the conference

We rely on the honor systems for initially reporting the performance data required for $Time_{effective}$ calculation, i.e., $Time_{measured,N=1024}$, $Time_{measured,N=256}$, $Time_{SW,your-CPU}$, $FPGA_{capacity,your-FPGA}$, and $frequency_{your-FPGA}$.

The winning designs must work correctly. In the case you should come in top three, we will need to arrange verification of the correctness of your design and the performance data.

4.5 Eligibility and Submission Instructions

This contest is open to industry and academic institutions (faculty and/or students). A team may include both industry and academic members. There is no limit on the size of a team. There is no limit on the number of teams per institution. Each person can participate in only one team.

If you are interested, please email `forrest@ece.ucsb.edu` with your contact information (team member names and affiliation). We will respond by sending you the SW-only MMM reference design (portable C) and the XUP HW/SW interface design (C and Verilog in an EDK project). There are no obligations associated with requesting the reference designs. You have until March 19, 2007 to submit your design submission (design source+documentation) in a tar or zip file via `http://www.ece.cmu.edu/tools/upload/` (use `memocode07@ece.cmu.edu` as the recipient address). Although you have until March 19, we estimate the design task to be approximately a one-week effort.