Performance modeling of virtualized custom logic computations

Michael J. Hall
Roger D. Chamberlain

Michael J. Hall and Roger D. Chamberlain. “Performance modeling of virtualized custom logic computations,” in *Proc. of 24th Great Lakes Symposium on VLSI* (GLSVLSI), May 2014.

Dept. of Computer Science and Engineering
Washington University in St. Louis

Performance Modeling of Virtualized Custom Logic Computations

Michael J. Hall Roger D. Chamberlain
Department of Computer Science and Engineering
Washington University in St. Louis
{mhall24, roger}@wustl.edu

ABSTRACT

Virtualization of custom logic computations (i.e., by sharing a fixed function across distinct data streams), provides a means of reusing limited hardware resources. This is common practice in traditional processors where more than one user can share processor resources. In this paper, we virtualize a custom logic block using C-slow techniques to support fine-grain context-switching. We then develop and present an analytic model for several performance measures (throughput, latency, input queue occupancy) for both fine- and coarse-grained context switching. Next, we calibrate the analytic performance model with empirical measurements. We then validate the model via discrete-event simulation and use the model to predict the performance and develop optimal schedules for virtualized logic computations.

Categories and Subject Descriptors

B.5.2 [**Register-Transfer-Level Implementation**]: Optimization; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids

1. INTRODUCTION

Virtualization of computational resources provides a way by which hardware resources can be reused or shared. Sharing is a common technique for utilizing available hardware resources for computing such as DSP blocks, memory controllers, memory bandwidth, and IP cores.

We have developed a model of the performance of a virtualized fixed logic computation. Our interest is in supporting a set of distinct data streams that all wish to perform the same computation. Virtualizing the logic computation involves sharing hardware resources and context-switching each distinct data stream into the hardware. This context switch can be either fine-grained (in which context is changed each clock cycle) or coarse-grained (in which the state of the computation is swapped out to a secondary memory). With the model, we can then tune a schedule for optimal performance.

GLSVLSI'14, May 21–23, 2014, Houston, TX, USA.
ACM 978-1-4503-2816-6/14/05.
http://dx.doi.org/10.1145/2591513.2591570.

To virtualize a function, consider a hardware block (HW) with an Nx1 input multiplexer and 1xN output demultiplexer. This block is a function implemented in custom logic. N distinct data streams (each with a dedicated input and output port) share the single instance of the HW block. These data streams are then multiplexed into the custom logic block, processed, and then demultiplexed back into independent streams. When the logic function is purely combinational (i.e., feed-forward), inputs from any data stream can be presented to the HW block at each clock cycle, even if it is deeply pipelined. In this case, there are no constraints on scheduling. When the logic function is sequential (i.e., has feedback) and has been deeply pipelined, this imposes scheduling constraints. Once a data element from a particular stream has been delivered to the HW block, the stream has to wait a number of clock ticks equal to the pipeline depth before it can provide a subsequent data element from that same stream.

Pipelined logic circuits having feedback can be context switched to compute multiple data streams concurrently. The pipelined logic adds latency and decreases single stream throughput since it takes multiple clock cycles (corresponding to the number of pipeline stages) to compute a single result and feed it back to the input. If the number of pipeline stages is C, then this circuit is said to be C-slowed since a single computation takes C times more clock cycles (often mitigated by a higher clock *rate*). C-slow is a technique described by Leiserson and Saxe [2] by which each register is replaced by C registers and then retimed to balance the registers throughout the combinational logic.

When the number of contexts to be supported, N, is greater than the pipeline depth, C, coarse-grained context switching can be used, swapping out whatever state is stored in the circuit to a secondary memory. In general, this will incur some cost, S, representing the overhead of a context switch.

2. MODELING AND CALIBRATION

The queueing model computes the effective service rate for each context to be

$$\mu_s = \frac{R_S}{(R_S N + SN/C) \cdot t_{CLK}} \ {}^{\text{elements}}/_{\text{s}} \qquad (1)$$

where R_S is the number of rounds of C fine-grain contexts that execute in a round-robin schedule before a context switch to a secondary memory (schedule period).

It follows that the total achievable throughput, T_{TOT}, is then $N \cdot \mu_s$. The wait time at the head of the queue is,

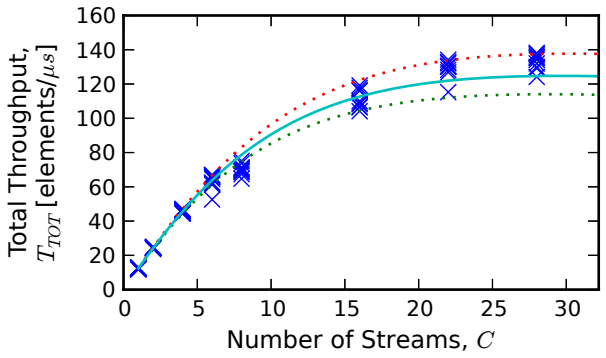

Figure 1: Total achievable throughput plot. The number of streams is the logic pipeline depth, C, and $N = C$). For each value of C, 10 data points were taken.

likewise, determined to be

$$W_h = \frac{(R_S\,(N-C) + SN/C)^2 \cdot t_{CLK}}{2\,(R_S N + SN/C)}$$

which can be used with classic queueing model results to compute latency.

We use an AES encryption cipher in CBC block mode (that has a feedback path) as the experimental vehicle to calibrate the clock period, t_{CLK}, in the model. In our implementation, the individual block cipher is fully unrolled forming a long combinational function up to 14 rounds (the AES 256-bit standard) [3]. We target a Xilinx Virtex-4 XC4VLX100 FPGA and use the Xilinx ISE 13.4 tools for synthesis, place, & route of the hardware designs.

We compute the total throughput, T_{TOT}, as $1/t_{CLK}$, and plot the observed data values from the synthesis, place, & route runs and a model prediction fitted to the data in Figure 1. The solid line represents the model prediction. The dotted lines represent confidence intervals on the model [1] (computed as a 95% confidence for 10 future observations).

The model does a reasonably good job of characterizing the shape of the curve. Most, although clearly not all, of the measured data points are within the confidence intervals. Second, throughput initially increases linearly (at low stream counts) but eventually levels off and adding additional streams does not provide any significant throughput gains. Finally, the maximum throughput achievable is present at 16 streams and higher.

3. RESULTS

We use our model with the calibrated t_{CLK} to predict latency and total throughput performance (shown in Figure 2) as a function of offered load and schedule period (R_S). The latency plot is initially high at low R_S due to queueing delay, but drops steeply as R_S increases due to the total achievable throughput also increasing. Latency then hits a minimum at the knee and next gradually increases due to wait time caused by the larger R_S in the round-robin schedule. The total achievable throughput plot increases as R_S increases because the cost of a context switch, S, is amortized by the higher schedule period. To optimize for both latency and throughput, we computed a figure of merit (FoM) defined

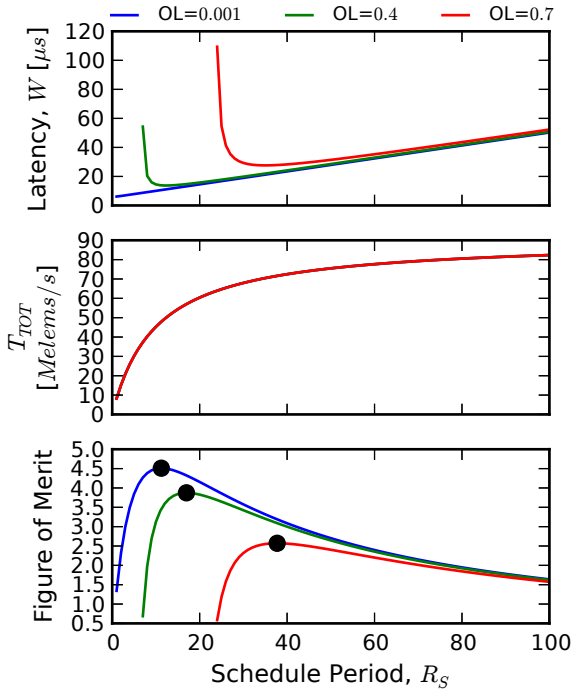

Figure 2: Latency (top), total achievable throughput (middle), and figure of merit (bottom) at three different offered loads (OL) versus the schedule period, R_S. C is 10, N is 100, S is 100, and t_{CLK} is calibrated using model in Figure 1.

as T_{TOT}/W (total throughput divided by latency). A plot of the figure of merit shows that it is initially low (due to low throughput and high latency), increases to a peak (the optimal value), and then decreases gradually (since latency is increasing faster than throughput).

Solving for the maximum FoM optimizes the schedule period, R_S, concurrently for high throughput and low latency.

4. CONCLUSIONS

Analytic models are developed for achievable throughput, latency (including traditional queueing, increased queueing due to hierarchical scheduling, and compute time), and occupancy of the input buffers. Using the models, we demonstrate the ability to choose an optimal schedule period as a function of one (or more) input parameters.

5. ACKNOWLEDGMENT

This research was supported by NSF grant CNS-0931693.

6. REFERENCES

[1] R. Jain. *The Art of Computer System Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. John Wiley & Sons, Inc., New York, 1991.

[2] C. Leiserson and J. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6:5–35, June 1991.

[3] NIST. FIPS-197: Advanced Encryption Standard. *Federal Information Processing Standards (FIPS) Publications*, Nov. 2001.