Using Queuing Theory to Model Streaming Applications

Rahav Dor, Joseph M. Lancaster, Mark A. Franklin, Jeremy Buhler, and Roger D. Chamberlain
Dept. of Computer Science and Engineering, Washington University in St. Louis
{rahav.dor, lancaster, jbf, jbuhler, roger}@wustl.edu

***Abstract*—Queuing theory provides a theoretical framework for quantitative understanding of the performance of streaming applications. There are questions, however, as to the suitability of queuing models to real applications. Here, we investigate the correspondence between a relatively simple queuing model of an FPGA-accelerated BLAST implementation and empirical measurements taken from executions of the actual application.**

I. Introduction

Queuing theory has been used to model many systems, from simple single-server queuing stations such as a bank teller to complex distributed software systems such as web servers [1]. Here, we are interested in whether simple, straightforward queuing models can be effective at describing the performance of high performance streaming data applications executing on architecturally diverse computers. This question is particularly interesting given the fact that we know, a priori, that the queuing models make several simplifying assumptions about the operation of the modeled system that we know to be counter to the actual application. If this modeling paradigm proves viable, then the ease of constructing and solving models under this paradigm should be palatable for system modeling in general, as well as for design space exploration, understanding of a system, and reasoning about its improvements or limitations. We explore this question by constructing a Jacksonian queuing network model of a streaming implementation of BLAST deployed on a combination of CPUs and FPGAs [2], [3]. BLAST is a computationally intensive biosequence alignment application and we compare performance predictions from the model to empirical measurements from the executing application.

The TimeTrial performance monitor [4], [5] is used to both calibrate and validate the performance model. For calibration we measure the input rate at the beginning of the pipeline, λ_{in}, and we use the measured branching probabilities to calculate the input rates to subsequent stages. Service rates are determined by a combination of first principles understanding of how the system operates and measured values. We validate the model by predicting server utilizations and queue occupancies and comparing the predictions to empirical measurements on two distinct input data sets.

II. BLAST Application

BLAST [6], [7] is a software tool widely used in bioinformatics to find areas of biologically meaningful similarity between DNA or protein sequences. BLAST works by comparing one or more *query sequences* to a database of other sequences, using a weighted edit distance computation to determine how similar the query is to each substring of the database. Because this distance is expensive to compute, BLAST uses a pipeline of heuristics to rapidly filter out database regions with a large edit distance to the query.

This work focuses on modeling Mercury BLAST [2], [3], an accelerated BLAST that combines general-purpose processors and FPGAs. Mercury BLAST's three-stage pipeline is illustrated in Figure 1. In the first FPGA stage, BLAST detects *seed matches*, which are exact substring matches of length 11 between the query and the database. Mercury BLAST divides this stage's work into two parts: stage 1a, in which each database word is checked against on-chip Bloom filters [8] built from the query to eliminate most non-matching words, and stage 1b, in which the locations of matching query and database words, if any, are identified using an SRAM-based hash table.

Seed matches are forwarded to the second FPGA stage, *ungapped extension*. This stage checks whether each seed match is part of a larger *ungapped alignment*, i.e. a region in which the query and database differ by a small number of character substitutions. Ungapped alignments passing this stage are forwarded to the third stage, *gapped extension*, which runs in software. Gapped extension determines whether each seed match is part of an even larger region with small edit distance, this time permitting character substitutions, insertions, and deletions. Regions that pass this final test represent strong *gapped alignments* between query and database, which are reported to the user.

III. Queuing Theory Performance Model

Figure 2 shows a queuing network used to model Mercury BLAST. The queuing network is Jacksonian [9], [10], meaning that the individual queuing stations are Markovian (Poisson arrival process with rate λ, exponentially distributed service times with rate μ) and the queues are assumed to have infinite capacity (an M/M/1 queuing model). It is worth pointing out here that the actual application exhibits none of these properties. Arrivals are not Poisson, service times are not exponential, and the physical queues are finite in capacity. Whether or not the M/M/1 model assumptions are crucial is the relevant question addressed in this work.

The model is nominally composed of 5 queuing stations: the PCI-X bus that interconnects the processor and the FPGA,

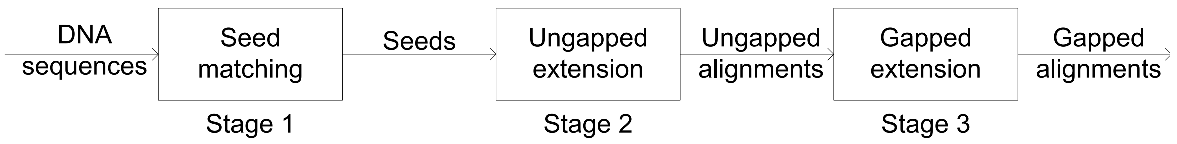

Figure 1. Mercury BLAST.

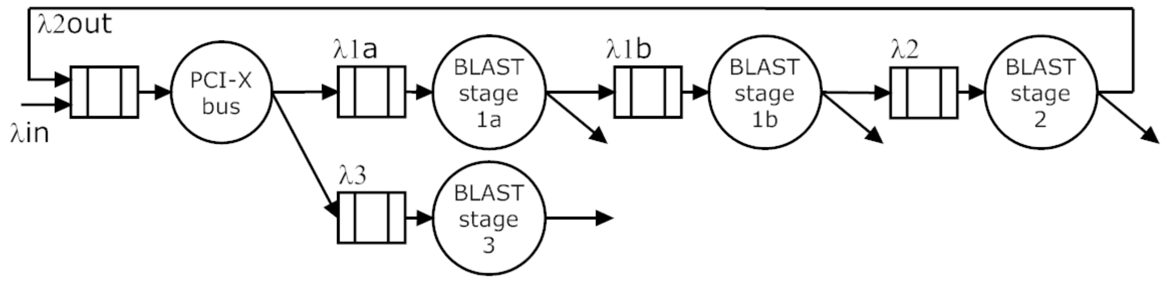

Figure 2. Queuing model of Mercury BLAST.

as well as BLAST stages 1a, 1b, 2, and 3. Our focus is on application stages that are executed on the FPGA, so we will concentrate our attention on the servers modeling BLAST stages 1a, 1b, and 2.

Starting from the left, λ_{in} is the rate at which DNA sequence bases are consumed. They are queued for delivery across the PCI-X bus and then delivered to stage 1a. As a result, $\lambda_{1a} = \lambda_{in}$. The feedback from BLAST stage 2 is sent over the PCI-X bus to stage 3 which is executed on CPUs, therefore $\lambda_3 = \lambda_{2out}$. Stage 1a filters the stream of incoming bases into hits that are passed to stage 1b (with probability p_{1a1b}) or dropped (with probability $(1 - p_{1a1b})$). Stage 1b generates seeds that are passed to stage 2 (with probability p_{1b2}) or dropped (with probability $(1 - p_{1b2})$). Stage 2 subsequently generates alignments that are either passed to stage 3 via the PCI-X bus (with probability p_{23}) or dropped (with probability $(1 - p_{23})$). The above relates the set of λs by the following system of equations:

$$\begin{aligned} \lambda_{1a} &= \lambda_{in} \\ \lambda_{1b} &= p_{1a1b}\lambda_{1a} \\ \lambda_2 &= p_{1b2}\lambda_{1b} \\ \lambda_{2out} &= p_{23}\lambda_2 \\ \lambda_3 &= \lambda_{2out}. \end{aligned}$$

Using classic results from queuing theory [11], the utilization for each server i is

$$\rho_i = \lambda_i/\mu_i,$$

the mean queue occupancy for queue i is

$$N_{Q,i} = \rho_i^2/(1 - \rho_i),$$

and the probability of n or more elements in station i is

$$P_i[N \geq n] = \rho_i^n.$$

IV. Model Calibration and Validation

We use the TimeTrial performance monitor [4], [5] to make empirical measurements on the executing system. TimeTrial is capable of measuring resource utilizations, data throughputs, etc. for streaming applications deployed on a combination of processors and FPGAs. It is engineered to make its measurements with minimal impact on the performance of the application being monitored.

To calibrate the queuing model we use TimeTrial to measure the input arrival rate (λ_{in}) at the beginning of the pipeline, operating values of stage 1b that are data dependent (average number of lookups in SRAM per seed, and the SRAM utilization), and the branching probabilities (p_{1a1b}, p_{1b2}, and p_{23}). To validate the model we compare the model predictions to the empirically measured server utilizations of stages 1a, 1b, and 2 (ρ_{1a}, ρ_{1b}, and ρ_2) and queue occupancies of stages 1b and 2 ($N_{Q,1b}$ and $N_{Q,2}$).

Each of the above measurements is made for 2 distinct test cases. The runs are as follows:

- Run 1: The first dataset is the human chromosome 1 (from build 19 of the human genome) divided into 7,964 65,400-base segments as the query. The database consists of the 9th build of the mouse genome (2.7 GBases).
- Run 2: The second dataset consists of comparing all the non-mammal vertebrate mRNA split into 8,608 65,400-base segments as the query. The queries were searched against all the mammal mRNA in the NCBI RefSeq repository (791 Mbases) as the database.

Table I gives the model inputs, while Table II provides a comparison between model predictions and empirical measurements. Note that the units for the service rates vary as one moves down the pipeline. Where needed, appropriate unit conversions are incorporated into the model (e.g., DNA bases are encoded as 4 bits per base, so one byte of data transfered across the PCI-X bus delivers 2 bases to the input of stage 1a). In addition, the service rate for stage 1b is a non-linear function of whether or not the external memory port is saturated.

Starting with the server utilization results, we observe that there is a close match in virtually every case between the model predictions and the empirical measurements. This validates model input rate, service rates, and branching probabili-

Table I
INPUT PARAMETERS TO QUEUING MODEL.

Parameter	Value for Run 1	Value for Run 2
λ_{in}	900 MB/s	720 MB/s
μ_{PCI}	1 GB/s	1 GB/s
μ_{1a}	2.1 Gbases/s	2.1 Gbases/s
μ_{1b}	130 Mseeds/s	50 Mseeds/s
μ_2	133 Maligns/s	133 Maligns/s
p_{1a1b}	0.018	0.035
p_{1b2}	0.88	0.76
p_{23}	0.0002	0.0004

Table II
MODEL PREDICTIONS VS. EMPIRICAL MEASUREMENTS.

Parameter	Model Prediction	Empirical Measurement	Error
Run 1			
ρ_{1a}	0.84	0.85	0.01
ρ_{1b}	0.25	0.23	0.02
ρ_2	0.21	0.21	0
$N_{Q,1b}$	0.08	approx. 0	0.07
$N_{Q,2}$	0.06	1.2	1.1
Run 2			
ρ_{1a}	0.68	0.68	0
ρ_{1b}	0.999	0.93	0.07
ρ_2	.29	.29	0
$N_{Q,1b}$	7500	580	6900
$N_{Q,2}$	0.12	1.7	1.6

ties. This is not surprising, since these directly determine stage utilizations and are independent of the distributions employed.

Turning to the queue occupancies, for 3 of 4 cases we again have a very close match between the model predictions and the empirical measurements. The one significant discrepancy is for the queue associated with stage 1b in run 2. Here, the high server utilization indicates that this server is the performance limiting bottleneck in the application. The physical queue is of length 600 entries, so the empirical queue occupancy cannot grow larger than that. The model predicts a much larger queue occupancy. Both the model and the empirical results are indicating that the queue will fill; however, the infinite queue capacity in the model is not capped by the length of the physical queue.

Having predicted both the server utilizations and the queue occupancies implies that the assumptions present in the M/M/1 queuing models do not inordinately impact the quality of the model for these two runs. These two runs were selected because they represent two distinct execution circumstances. Run 1 lightly taxes the system while run 2 heavily taxes at least stage 1b.

V. CONCLUSIONS

We have illustrated the use of straightforward queuing models to describe the performance of high performance streaming data applications. In general, they do surprisingly well at predicting the performance properties of the real application. Where there are discrepancies, the model can assist in understanding those discrepancies. For example, in the actual system, there is backpressure being asserted upstream of stage 1b due to the fact that the queue is full. While the notion of backpressure doesn't exist explicitly in the current queuing model, we can estimate it by asking the model for the probability that the queue occupancy is greater than the actual capacity of the queue. For run 2 this gives us $P_{1b}[N \geq 600] = 0.92$, a likely event. Additional details on this proposed paradigm as well as its suitability for an M/M/1/K queuing model can be found in [12].

Our intent is to seek additional evidence for our hypothesis by empirically validating our modeling paradigm with more runs of Mercury BLAST and other applications. Using the suggested queuing models to describe applications that do not necessarily exhibit the corresponding probability distribution requires additional exploration and we intend to look into the reasons that allow these models to work. Lastly we intend to use this modeling paradigm to guide tuning of the Mercury BLAST implementation, to propose design alternatives that will increase its performance, or to examine the potential performance benefits achievable by exploiting alternative accelerators (e.g., graphics engines) for one or more of the pipeline stages.

ACKNOWLEDGMENTS

This work was supported by NSF grants CNS-0931693, CNS-0905368, and CNS-0751212 and NIH grant R42 HC003225 (the latter through BECS Technology, Inc.). R.D. Chamberlain is a principal in BECS Technology.

REFERENCES

[1] G. Casale, M. Ningfang, and E. Smirni, "Versatile models of systems using map queueing networks," in *IEEE Int'l Symp. on Parallel and Distributed Processing*, Apr. 2008.

[2] J. D. Buhler, J. M. Lancaster, A. C. Jacob, and R. D. Chamberlain, "Mercury BLASTN: Faster DNA sequence comparison using a streaming hardware architecture," in *Proc. of Reconfigurable Systems Summer Institute*, Jul. 2007.

[3] P. Krishnamurthy, J. Buhler, R. Chamberlain, M. Franklin, K. Gyang, A. Jacob, and J. Lancaster, "Biosequence similarity search on the *Mercury* system," *Journal of VLSI Signal Processing*, vol. 49, no. 1, pp. 101–121, Oct. 2007.

[4] R. D. Chamberlain and J. M. Lancaster, "Better languages for more effective designing," in *Proc. of Int'l Conf. on Engineering of Reconfigurable Systems and Algorithms*, Jul. 2010.

[5] J. M. Lancaster, J. D. Buhler, and R. D. Chamberlain, "Efficient runtime performance monitoring of FPGA-based applications," in *Proc. of 22nd IEEE Int'l System-on-Chip Conf.*, Sep. 2009, pp. 23–28.

[6] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, pp. 403–10, 1990.

[7] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, W. Miller, and D. J. Lipman, "Gapped BLAST and PSI-BLAST: A new generation of protein database search programs," *Nucl. Acids Res.*, vol. 25, no. 17, pp. 3389–3402, Sep. 1997.

[8] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Comm. of the ACM*, vol. 13, no. 7, pp. 422–426, May 1970.

[9] J. Jackson, "Network of waiting lines," *Management Science*, vol. 5, no. 4, pp. 518–521, 1957.

[10] ——, "Jobshop-like queueing systems," *Management Science*, vol. 10, no. 1, pp. 131–142, 1963.

[11] L. Kleinrock, *Queueing Systems, Volume 1: Theory*. John Wiley & Sons, 1975.

[12] R. Dor, "Against all probabilities: A modeling paradigm for streaming applications that goes against common notions," Dept. of Computer Science and Engineering, Washington University in St. Louis, Tech. Rep. WUCSE-2010-30, 2010.