Numerical solution of polling systems for analyzing networks on chips

Numerical Solution of Polling Systems for Analyzing Networks on Chips

Benny Van Houdt
Performance Analysis of Telecommunication Systems Research Group
Department of Mathematics and Computer Science
University of Antwerp - IBBT
Middelheimlaan 1, B-2020 Antwerp, Belgium
Email:{benny.vanhoudt}@ua.ac.be

***Abstract*—This paper introduces a numerical solution method for a class of discrete-time polling systems by relying on the power method, Kronecker matrix representations and the shuffle algorithm. The class of polling models considered consists of several infinite pseudo queues, deterministic service times, Bernoulli service and Markovian routing and includes exhaustive, 1-limited and k-limited service as special cases. A truncated, large finite state Markov chain is obtained by solving a series of finite state Markov chains with increasing size. The model is motivated by the analysis of networks on chips and its superiority over other existing approximation methods in terms of the accuracy and computation times is demonstrated.**

I. Introduction

Network-on-Chip (NoC) is an emerging paradigm using packet-switched networks for communications within large VLSI systems on-chip. NoCs are poised to provide enhanced performance, scalability, modularity, and design productivity as compared with previous communication architectures such as busses and dedicated signal wires [5], [3]. Since 2007 annual ACM/IEEE symposiums on NoCs are being organized. The delay performance in a NoC with a single sink (meaning all packets are routed towards the same destination) has been studied previously by setting up a queueing network with a tree topology [1]. Each node in this tree consisted of a polling station serving a number of waiting rooms. Packets are allowed to enter the tree at any level and move upwards in the tree until they are served by the root node. A packet that is served by the i-th child of a node in the tree joins the i-th waiting room of its parent node. By assuming deterministic packet lengths and a Head-of-Line (HoL) service for the root node, it was proven in [1] that the mean delay in this queueing network could be obtained by analyzing an equivalent single node queueing system. The HoL service means that the selection of the waiting room that is served next depends on the empty/nonempty state of all the waiting rooms during last b slots, for some finite b. Thus, it includes exhaustive, 1-limited and k-limited service (as well as time-limited due to the fixed length of the packets), but not gated service. In order to study the mean delay in this reduced single node polling system, an approximation was proposed in [2] via an iterative numerical method, where each iteration involved the solution of an M/G/1-type Markov chain that corresponded to a system with one infinite waiting room, while all other waiting rooms had a truncated length of B ($B = 1$ or 2). Increasing B typically lead to more accurate results, but also resulted in increased computation times.

We will study the same single node polling system as in [2], except that we consider a somewhat more general subclass of the HoL service disciplines by relying on pseudo queues. The most significant implication of this relaxation is that our class also includes the k-limited service disciplines. Furthermore, the approach presented in this paper relies on the power method, Kronecker representations and the shuffle algorithm [6], to compute the steady state of a large finite Markov chain and is considerably faster to reach the same or a better degree of accuracy in comparison with [2]. Finally, the presented approach also enables us to compute the per-type interdeparture time distributions in a straightforward manner.

II. System description

We consider a discrete-time polling system consisting of N queues with a waiting room of size Q_1, Q_2, ..., Q_N, respectively. All waiting rooms were considered of infinite size in [2], thus a dynamic truncation procedure to determine Q_1 to Q_N is presented in Section V, such that the loss rate in each queue is negligable without overdimensioning the queue too much as larger queues require longer computation times. The arrivals to queue number i form a batch Bernoulli process characterized by the probabilities $x_i(0), x_i(1), \ldots, x_i(Q_i)$, where $x_i(j)$ represents the probability of having j arrivals. Notice, the probabilities $x_i(j)$ with $j > Q_i$ are irrelevant as at most Q_i packets can be stored in queue i. The service times are deterministic and last for one time slot. The service discipline of the system, that determines the order in which the queues are served, relies on $\bar{N} \geq N$ pseudo queues. With each pseudo queue $i \in \{1, \ldots, \bar{N}\}$ we associate a queue $\psi(i) \in \{1, \ldots, N\}$. At each time slot, the service discipline determines the pseudo queue to be served. If pseudo queue i is selected, queue $\psi(i)$ is served. A pseudo queue i is said to be empty if its corresponding queue $\psi(i)$ is empty.

The service discipline is Bernoulli with Markovian routing characterized by a set of probabilities $q(1), q(2), \ldots, q(\bar{N})$ and an irreducible $\bar{N} \times \bar{N}$ stochastic matrix R with $r_{i,i} = 0$ for all i. The probability $q(i)$ represents the probability that the server remains at pseudo queue i after serving a packet from

pseudo queue i, i.e., queue $\psi(i)$. Thus, the next server position, denoted as j, equals i. On the other hand, with probability $1-q(i)$ the server moves to another pseudo queue, in which case entry $r_{i,j}$ holds the probability that the server moves from pseudo queue i to pseudo queue j (recall, $r_{i,i}=0$). If subsequently pseudo queue j is empty, but not all pseudo queues are empty, the server position evolves according to the matrix R until a non-empty pseudo queue is encountered. Finally, if during a particular time slot no packet is served, because all the queues are empty, the server position remains identical.

The abovementioned service discipline supports 1-limited routing (by setting $\bar{N}=N$ and $q(i)=0$), exhaustive service ($\bar{N}=N$ and $q(i)=1$), and k-limited service ($\bar{N}=Nk$, $\psi(i)=\lceil i/k \rceil$, $q(i)=0$ and R is the stochastic circulant matrix with a one in the lower left corner). The introduction of the probabilities $q(i)$ is only required to support exhaustive service, otherwise we could simply use the stochastic matrix $\hat{R}$ with $\hat{r}_{i,i}=q(i)$ and $\hat{r}_{i,j}=(1-q(i))r_{i,j}$ to determine the next server position j.

III. Transition matrix

Let $L_i(n)$ be the queue length of queue i at time n and $F(n)\in\{1,\ldots,\bar{N}\}$ the position of the server at time n. From the above description it is clear that the process $\{(L_1(n),L_2(n),\ldots,L_N(n),F(n)),n\geq 0\}$, is a Markov chain with $S=\bar{N}\prod_{i=1}^{N}(1+Q_i)$ states. In order to specify its transition matrix P, we will denote it as a product of three size S matrices D, A and C, such that

$$P=DAC.$$

The transition matrix D will capture the departures as well as the immediate change of the server position due to the service completion (i.e., the change from pseudo queue i to j mentioned before). The arrivals are added by the matrix A, while C takes care of the change of the server position such that a non-empty queue is visited (unless all queues are empty).

We will denote the unity matrix of size b, $\prod_{k=1}^{i-1}(1+Q_i)$ and $\prod_{k=i+1}^{N}(1+Q_i)$ as I_b, $I_{<i}$ and $I_{>i}$, respectively. Moreover, we denote $\Delta(x)$ as the diagonal matrix with its diagonal entries equal to the vector x and let e_i be a vector of the appropriate size with all its entries equal to zero, except for the i-th, the value of which is one.

To construct the matrix D, we first define D_i as a size $(1+Q_i)$ matrix of the form

$$D_i=\begin{bmatrix} 1 & & & & \\ 1 & 0 & & & \\ & 1 & 0 & & \\ & & \ddots & \ddots & \\ & & & 1 & 0 \end{bmatrix}.$$

and

$$\hat{D}=\sum_{i=1}^{\bar{N}}\left(I_{<\psi(i)}\otimes D_{\psi(i)}\otimes I_{>\psi(i)}\otimes\Delta(e_i)\right).$$

Hence, $\hat{D}$ captures the effect of a possible service competion. After the service completion the server moves according to

$$R^{(q)}=\Delta(q)+(I_{\bar{N}}-\Delta(q))R,$$

with $q=(q(1),\ldots,q(\bar{N}))$. There is however one exception: if all the queues were empty, the server remains in its initial position. This implies that D can be expressed as

$$D=\left(\hat{D}-\Delta(e_1)\otimes I_{\bar{N}}\right)(I_{S/\bar{N}}\otimes R^{(q)})+\Delta(e_1)\otimes I_{\bar{N}},$$

because the first $\bar{N}$ states of the Markov chain correspond to an empty system.

The matrix A incorporates the arrivals and can clearly be represented as

$$A=(A_1\otimes A_2\otimes\ldots\otimes A_N)\otimes I_{\bar{N}},$$

with $x_i^+(j)=\sum_{k\geq j}x_i(k)$ and A_i an upper triangular matrix of size $(1+Q_i)$ of the form

$$A_i=\begin{bmatrix} x_i(0) & x_i(1) & x_i(2) & \ldots & x_i^+(Q_i) \\ & x_i(0) & x_i(1) & \ldots & x_i^+(Q_i-1) \\ & & \ddots & \ddots & \\ & & & & x_i^+(0) \end{bmatrix}.$$

Finally, the matrix C changes the server position such that a non-empty pseudo queue is served in the next time slot (unless the system is empty). To ease the notation, we denote $bin(i)$ as the binary length N representation of $i\in\{1,\ldots,2^N-1\}$. Let $bin_j(i)$ be the j-th bit of $bin(i)$, which we associate with queue j. Let $N_{bin(i)}=\{j,bin_j(i)=1\}$. Furthermore, denote $I_{bin(i)}$ as the size $S/\bar{N}$, binary, diagonal matrix with ones on the positions corresponding to a state where all queues in $N_{bin(i)}$ are non-empty, while the other queues are empty:

$$I_{bin(i)}=\bigotimes_{j=1}^{N}\left(\Delta(e_1)(1-bin_j(i))+\Delta(e-e_1)bin_j(i)\right).$$

Next, let $V_{bin(i)}$ be a size $\bar{N}$ matrix with its j-th row equal to the first passage probabilities of the Markov chain characterized by R, from state j to the set of states $\{j',\psi(j')\in N_{bin(i)}\}$ (note, $V_{bin(2^N-1)}=I_{\bar{N}}$). This allows us to specify the matrix C that describes the change of the server position as

$$C=\sum_{i=1}^{2^N-1}I_{bin(i)}\otimes V_{bin(i)}.$$

IV. Computing the steady state distribution

To compute the invariant vector of $P=DAC$, we will rely on the power method, i.e., we start with some length S vector x_0 and repeatedly multiply this vector with P. The multiplication with P is split into three steps: first we multiply with D, next with A and finally with C, each time exploiting the structure of these matrices.

For the multiplication with A we simply apply the shuffle algorithm [6] N times together with the identity

$$\bigotimes_{i=1}^{N}A_i=\prod_{i=1}^{N}(I_{<i}\otimes A_i\otimes I_{>i}).$$

We do not exploit the structure of A_i as the shuffle operation requires more time than the product itself and the matrices A_i are small. To speed up the computation we do store N index vectors that specify the length S permutations used by the shuffle algorithm, such that they only need to be determined once.

In order to multiply the vector with D, it should be clear that the multiplication with $\hat{D}$ is the key step, as the $\Delta(e_1)$ matrices only affect the first $\bar{N}$ entries of the vector and multiplying with $(I \otimes R^{(q)})$ is trivial. The multiplication with $\hat{D}$ is found as the sum of $\bar{N}$ multiplications with a matrix of the form

$$I_{<\psi(i)} \otimes D_{\psi(i)} \otimes I_{>\psi(i)} \otimes \Delta(e_i).$$

Due to the $\Delta(e_i)$ matrix, this can be done by selecting the entries on the positions $i + \bar{N}j$, for all j, and multiplying this length $S/\bar{N}$ vector with $I_{<\psi(i)} \otimes D_{\psi(i)} \otimes I_{>\psi(i)}$ (via the shuffle algorithm). As before, we do not exploit the structure of D_i as the shuffle operation is more time consuming and the matrices are small. The N permutations of length $S/\bar{N}$ used by the shuffle algorithm are stored in memory to speed-up the computation.

Finally, we need to multiply a vector x with C, meaning we need a fast product with the matrices $I_{bin(i)} \otimes V_{bin(i)}$ for all i. As the matrices $I_{bin(i)}$ are binary diagonal matrices, this step only involves selecting a subvector of x and a multiplication with $I_{N(i)} \otimes V_{bin(i)}$, where $N(i) = \prod_{j \in N_{bin(i)}} Q_j$. The selection of the $\bar{N}N(i)$ entries of x is implemented by storing $2^N - 1$ vectors with a total length of $\bar{N} \sum_{i=1}^{2^N-1} N(i) = S - \bar{N}$ (the first $\bar{N}$ entries are not affected by C).

V. Selecting the queue lengths Q_i

To determine the sizes Q_i of the waiting rooms, we will use a stepwise procedure. The procedure aims at making the waiting rooms large enough to obtain a good approximation, while avoiding the need to use unnecessarily large queues as this would result in poor computation times. This is a delicate problem as these lengths depend very much on the traffic characteristics and polling discipline under consideration.

In step 1, we set $Q_1 = \ldots = Q_N = 1$ and solve the system as discussed in the previous section, with x_0 the uniform vector of length $\bar{N}2^N$. At the end of step $i \geq 1$, we determine the queue with the highest probability of being full, say queue j, and increase the size of its waiting room Q_j by one, while all the other Q_i values remain identical. Next, in step $i + 1$ we solve this updated system and use the solution of step i as the initial vector x_0 (by setting the entries that correspond to having $Q_j + 1$ packets in queue j equal to zero), thereby considerably reducing the number of iterations required by the power method. This stepwise solution finishes if the probability of having a full queue is below a predefined threshold ϵ (e.g., $\epsilon = 10^{-5}$).

One could also increase the queue length of several queues by one during a single step, this typically reduces the number of steps significantly, but each step requires more time. When the final queue lengths Q_i are similar for different i values, this approach typically realizes the best computation times, while for more asymmetric queue lengths the former approach prevails. In our numerical examples, exhaustive service and 1-limited service with low to medium loads (0.5 and 0.7) often resulted in similar queue lengths, while the 1-limited service examples with high loads (0.9) did not.

VI. Interdeparture time distribution

Using the Markov chain characterized by P and the power method described earlier, it is not hard to devise a simple procedure to compute the interdeparture time distribution of type i packets (i.e., packets belonging to queue i). For this purpose, we define $n_i(x)$ of a size S vector x as another size S vector, with its j-th entry equal to the j-th entry of x if j is a state for which queue i is nonempty and equal to zero otherwise.

Next, we define x_0 as the normalized vector of $n_i(\pi)$, where π is the steady state vector (computed by the power method) of P. Subsequently, we compute $y_k = x_{k-1}P$ and let $x_k = y_k - n_i(y_k)$, while $P[I_i = k]$ the probability of having an interdeparture time of k slots, is found as the sum of the entries in $n_i(y_k)$. We perform s steps such that $\sum_{k=1}^{s} P[I_i = k] > 1 - \epsilon$, for some ϵ small.

VII. Numerical results

In this section we start by comparing our approach to the aproximation method developed in [2]. To compare the accuracy and computation times, we rely on the same example as in [2] and consider a system with $N = 4$ queues, 1-limited service, cyclic routing and Poisson distributed batch sizes with parameter ρ_i, where $(\rho_1, \rho_2, \rho_3, \rho_4) = (0.1, 0.2, 0.3, 0.4)\rho$, meaning the overall load is ρ. We applied the power method until the difference between two consecutive vectors was less than 10^{-7} and increased the queue lengths until the probability of having a full queue was below 10^{-4}. For $\rho = 0.9$ this resulted in solving 59 Markov chains with 64 states for the first and 134640 states for the last chain.

In Table I the improved accuracy of the power method for the queue length distribution of the 4-th queue (this one is the hardest to approximate) is shown by comparing it with the simulation results and approximations for $B = 2$ (and $B = 3$) reported in [2]. The computation times for $B = 2$ and $B = 3$ reported were about 2 to 3 seconds and 50 seconds (per value of ρ), respectively. Thus, increasing B by one augmented the computation times by a factor close to 20, hence, setting $B = 4$ would require many minutes of computation time. It should also be noted that these computation times also depend strongly on the value of ρ, but no detailed timings per ρ value were provided. The power method introduced in this paper uses less than 5 seconds for $\rho = 0.5$ and $\rho = 0.7$ and up to 2.5 minutes for $\rho = 0.9$. About 50% of the computation time was devoted to the multiplication with A, 12% with D and 13% with C. The remaining computation time was mostly spend on setting up the index vectors needed by the shuffle algorithm. This part of the computation could be further improved as they are completely recomputed each time one of the queue

ρ		$P(Q_4=0)$	$P(Q_4=1)$	$P(Q_4=2)$	$P(Q_4=3)$	$P(Q_4=4)$	$P(Q_4=5)$	$P(Q_4=6)$
0.5	$B=2$	.7412	.2109	.0395	.0069	.00126	.00024	.000047
	Simul.	.7411	.2109	.0395	.0069	.00127	.00024	.000049
	Power	.7411	.2109	.0395	.0069	.00127	.00024	.000046
0.7	$B=2$	.5661	.2756	.0994	.0361	.0137	.0054	.00218
	Simul.	.5655	.2754	.0994	.0362	.0139	.0056	.00228
	Power	.5655	.2754	.0994	.0362	.0139	.0056	.00228
0.9	$B=2$	.267	.2159	.1438	.0993	.0712	.0520	.0384
	$B=3$	.265	.2129	.1413	.0974	.0701	.0517	.0387
	Simul.	.263	.2109	.1394	.0959	.0690	.0510	.0383
	Power	.263	.2112	.1396	.0960	.0690	.0510	.0383

TABLE I
ACCURACY IN TERMS OF THE QUEUE LENGTH DISTRIBUTIONS

ρ		Queue 1	Queue 2	Queue 3	Queue 4
0.7	$B=2$	0.615	0.854	1.138	1.462
	Simul.	0.618	0.858	1.145	1.475
	Power	0.617	0.857	1.143	1.473
	BM	0.709	0.938	1.167	1.395
	GL	0.539	0.830	1.152	1.503
0.9	$B=2$	1.172	1.98	3.50	6.46
	$B=3$	1.179	2.01	3.59	6.84
	Simul.	1.181	2.02	3.66	7.21
	Power	1.180	2.01	3.65	7.18
	BM	1.590	2.71	4.70	5.97
	GL	1.168	1.94	3.04	7.71

TABLE II
ACCURACY IN TERMS OF THE MEAN WAITING TIME

ρ		Queue 1	Queue 2	Queue 3	Queue 4
0.7	1-lim	0.617	0.857	1.143	1.473
	2-lim	0.832	0.904	1.114	1.417
	4-lim	1.186	1.130	1.129	1.204
	8-lim	1.413	1.288	1.164	1.041
	16-lim	1.451	1.323	1.182	1.000
	exhaust.	1.452	1.323	1.183	0.999

TABLE III
MEAN WAITING TIME FOR k-LIMITED SERVICE

lengths changes. The simulation times reported in [2] required considerably more time.

In Table II we compare the mean waiting times (via Little's formula) of the Power method with the simulation results, the approach in [2], as well as with the results in BM [4] and GL [7] for $\rho = 0.7$ and $\rho = 0.9$. We find that for $\rho = 0.9$ the power method (in comparison with $B = 3$) significantly improves the mean delay approximation, especially for queue 4. For $\rho = 0.7$ we also observe a clear improvement over $B = 2$, with comparable computation times.

In Table III we used the power metyhod to depict the mean waiting time per queue for the $\rho = 0.7$ scenario above, but for the more general k-limited service for various k values as well as for the exhaustive service. As expected, while k increases the results converge towards the exhaustive service discipline. The size of the final Markov chain grows from 13860 states for $k = 1$ to 276480 states for $k = 16$, while the computation time grew from 4 seconds to 40 seconds, meaning the computation time scales well with k. The exhaustive service case required less than 4 seconds (using 17280 states). The overall mean waiting time was 1.165 in all cases.

Finally, we study the impact of the number of queues N on the computation times by varying N from 2 to 6. In [2], small N values are regarded as practical for the network on chip architecture. As in [2] the loads $\rho_1, \ldots, \rho_N$ of the N queues are chosen proportional to $1, \ldots, N$. For a load of 0.7 with 1-limited cyclic routing the computation times ranged from less than 5 seconds for $N \leq 4$, to about 20 seconds for $N = 5$ and 1.5 minutes for $N = 6$. The dimension of the final Markov chain increased from 13860 states for $N = 4$ to 94500 states for $N = 5$ and towards 403200 states for $N = 6$.

ACKNOWLEDGMENT

The author would like to thank Paul Beekhuizen and Jacques Resing (Technical University of Eindhoven) for their valuable discussions and suggestions related to this work.

REFERENCES

[1] P. Beekhuizen, D. Denteneer, and J. A. C. Resing. Reduction of a polling network to a single node. *Queueing Syst. Theory Appl.*, 58(4):303–319, 2008.

[2] P. Beekhuizen and J. A. C. Resing. Approximation of discrete-time polling systems via structured markov chains. In *Proceedings of SMCTools*, pages 1–10, ICST, Brussels, Belgium, 2009.

[3] L. Benini and G. De Micheli. Networks on chips: a new SoC paradigm. *Computer*, 35:70–78, 2002.

[4] O. J. Boxma and B. W. Meister. Waiting-time approximations in multiqueue systems with cyclic service. *Perform. Eval.*, 7(1):59–70, 1987.

[5] W. J. Dally and B. Towles. Route packets, not wires: on-chip inteconnection networks. In *DAC '01: Proceedings of the 38th annual Design Automation Conference*, pages 684–689, New York, NY, USA, 2001. ACM.

[6] P. Fernandes, B. Plateau, and W. J. Stewart. Efficient descriptor-vector multiplications in stochastic automata networks. *J. ACM*, 45(3):381–414, 1998.

[7] W.P. Groenendijk and H. Levy. Performance analysis of transaction driven computer systems via queueing analysis of polling models. *IEEE Trans. Comput.*, 41(4):455–466, 1992.