# Pre-synthesis Queue Size Estimation of Streaming Data Flow Graphs

S. Mondal, S. Ogrenci Memik, *N. Bellas
EECS Department, Northwestern University, Evanston, IL
*Motorola Labs, Schaumburg, IL

## 1. INTRODUCTION

All synthesis efforts targeting reconfigurable logic face the challenge of creating designs that comply with the resource and storage capacity of the target device. Hence, area cost estimation is of significant importance in all stages of the hardware compilation process, which is a translation of a behavioral specification into a register-transfer level description. Data Flow Graphs (DFGs) are widely used for representing such behavioral descriptions. Area estimation techniques for compilation onto reconfigurable hardware in literature focus mainly on the functional unit (FU) area. In this work, we present an estimation technique to assess the resource requirement for storage elements in pipelined streaming architectures. Specifically, our proposed technique tackles the problem of pre-synthesis estimation of data queuing cost, while incorporating the potential impact of resource and throughput constraints on the final implementation.

With the increasing popularity of portable devices, there is a growing demand for multimedia applications. These applications are computationally intensive (often highly parallel) and are often streaming in nature. Reconfigurable logic is an effective medium for creating pipelined hardware as well as for exploiting parallelism. To create efficient hardware for streaming applications functional pipelining has been shown to be very effective and in such cases it is essential to register the inputs and outputs of FUs. This is because a FU has to retain its results from previous iterations (until they have been passed on to all consumers), while it is busy computing for successive iterations. Therefore, register queues at the outputs of FUs is one of the major building blocks that enable communication between FUs. The Reconfigurable Streaming Vector Processor (RSVP™ II[1]) [1, 2] is such a highly pipelined vector coprocessor architecture that has been implemented on reconfigurable fabric with a limited set of links (which implement the FIFO register queues). Hence, it is imperative to incorporate these register queues in area estimation.

In this paper we propose a pre-synthesis register queue size estimation technique for an unscheduled streaming DFG (sDFG) for pipelined synthesis. Our estimation method first designates a minimum queue size to each communication edge of the sDFG based on the ALAP value of the source node and ASAP value of the sink node of that edge. Our aim is to further refine this initial minimum queue size estimation. Our main tool is based on the likelihood estimation that the source node may actually be producing data before its ALAP time, and likewise, the sink node may actually be consuming data after its ASAP time. The likelihood of the source and sink nodes of an edge being moved up and down respectively during the actual scheduling depends primarily on resource constraints of the design and criticality of the nodes. In addition, it will be affected by the heuristics that a particular scheduler is using to optimize the throughput by reducing the register or interconnect pressure. Based on these modified queues, each node is assigned the longest queue among all its outgoing edge queue sizes. Finally, the queue size at each node is modified based on the estimated iteration interval. Based on such estimation the designer can assess at the pre-scheduling stage whether the target architecture can keep up with the storage requirements of the design. This can also be utilized to evaluate the complexity/cost of implementing various computational kernels on the target reconfigurable fabric. The best candidates can then be identified based on this estimation.

In this abstract, we present our preliminary results motivating the need for estimation. We compare pre-synthesis estimations on a set of industrial image processing applications with the queue sizes determined by Proteus[2] scheduler for RSVP™ architecture, which employs modulo scheduling to maximize the throughput of a sDFG synthesized as a pipelined datapath.

## 2. QUEUE SIZE ESTIMATION

The input to the queue size estimation is an unscheduled streaming DFG (sDFG) and a set of resource constraints. An sDFG is a DFG where I/O and internal communication edges are data streams, and not just simple variables. Also, no pointers, "GOTO" statements, function calls, or recursion is allowed in an sDFG. The total number of registers required for all nodes to implement the design is the final output of the estimation process. Figure 1 shows our register queue estimation flowchart. In the next subsections we will first formulate the queue estimation problem, and then discuss our approach in details. In the next section we will present our experimental results, which will be followed by our conclusions.

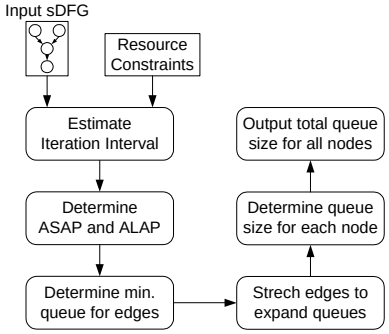


**Figure 1. Register queue size estimation flowchart**

### 2.1 Procedure for Queue Size Estimation

Given, an unscheduled streaming data flow graph $G = (V, E)$, and a set of resource constraints $R$, our goal is to estimate, the total number of registers in the queues of all nodes.

The first step of our estimation scheme is to determine the iteration interval of an sDFG based on the resource constraints. We assume that the sDFG does not have any cycle or in other words no inter-iteration dependencies. The lower bound of the iteration interval is estimated based on the technique presented by Hwang et al. [3]. Let $N_i$ be the number of operations of type $i$ in the sDFG, which can be implemented using a functional unit of type $i$, and let $M_i$ be the number of such functional units, then the

[1] RSVP is a trademark of Motorola Inc. Other product or service names are the property of their respective owners.

[2] Proteus scheduler is developed at Motorola Inc.

lower bound of the iteration interval, *ItIr*, is given by $\max_{1\le i\le t}\lceil N_i/M_i\rceil$, where $t$ is the number of types of functional units.

The next step is to determine the ASAP and ALAP schedules of the given sDFG. We have used the ASAP latency of the sDFG as the upper bound latency of the ALAP schedule. Let *ASAP*($v$) and *ALAP*($v$) be the ASAP and ALAP times of node $v \in V$. Once we have both the ASAP and ALAP schedules, we designate minimum queue sizes to each edge of the sDFG,

$$Q^{edge}Min(i, j) = ALAP(i) - ASAP(j),\ i, j \in V,\ (i, j) \in E \quad (1)$$

Our tool then refines these minimum queue sizes under the given resource constraints. Figure 2 shows the probabilistic ***push-and-pull*** queue expansion of an edge, where each node $n$ is marked with a set of values, [*ASAP*($n$), *ALAP*($n$), *slack*($n$)], and *slack*($n$) is given by *ALAP*($n$) – *ALAP*($n$).

Now let us first consider node $i$. Node $i$ can be pushed up by the scheduler depending on various factors, such as criticality of node $i$, resource constraints, and the number of more critical nodes of the same type within cycles *ASAP*($i$) and *ALAP*($i$).

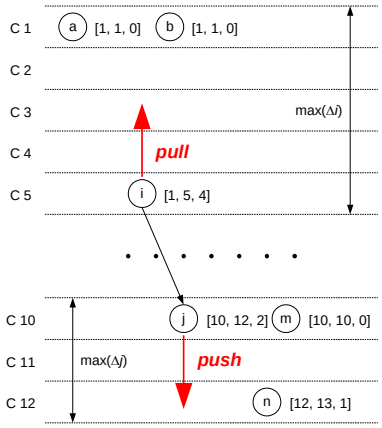


**Figure 2. Probabilistic queue expansion by *push-and-pull***

Let $P(i)_k$ be the probability that node $i$ is scheduled in cycle $k$. So, assuming that $i$ can be pushed up only until *ASAP*($i$), we have,

$$\sum_{k=ASAP(i)}^{k=ALAP(i)} P(i)_k = 1, \quad (2)$$

$$P(i)_k = 0 \quad \forall k < ASAP(i) \vee k > ALAP(i) \quad (3)$$

Assuming we have only one functional unit that can implement operations $a$, $b$, and $i$, in that case, $P(i)_1 \approx P(i)_2 \approx 0$, because, as nodes $a$ and $b$ are more critical than $i$, and we have only one functional unit, it is extremely less likely that node $i$ will be scheduled in cycle 1 or 2. Also, assuming that the scheduler primarily optimizes latency, we will have $P(i)_3 > P(i)_4 > P(i)_5$, since chances are high that the scheduler will ***pull*** up node $i$ as early as possible to minimize latency. We define $\Delta i$ as the amount by which node $i$ is pulled up. We compute the expected value of $\Delta i$, based on $P(i)_k$ as,

$$E[\Delta i] = \sum_{k=0}^{k=slack(i)} kP(i)_{ALAP(i)-k} \quad (4)$$

Similarly for node $j$, the scheduler will most likely ***push*** it down because of resource constraints. In that case, for the same reasons as above, we will have, $P(j)_{10} \approx 0$, and $P(j)_{12} > P(j)_{11}$. We define $\Delta j$ as the amount by which node $j$ is pushed down. Likewise, we calculate the expected value of $\Delta j$ as,

$$E[\Delta j] = \sum_{k=0}^{k=slack(j)} kP(j)_{ASAP(j)+k} \quad (5)$$

Now, the new expanded queue size for each edge $e(i, j)$ will be,

$$Q^{edge}Expand(i, j) = Q^{edge}Min(i, j) + E[\Delta i] + E[\Delta j] \quad (6)$$

Note that up until now we have only mentioned about the queue size of an edge, but what we are really trying to estimate is the queue size at the output of each node. From this the estimated queue size of a node $i$ will be,

$$Q^{node}Expand(i) = \max\{Q^{edge}Expand(i, j) : (i, j) \in E\} \quad (7)$$

Now, that we have $Q^{node}Expand(i)$, because of the iteration interval (*ItIr*) this will be reduced by a factor of *ItIr*. If *ItIr* equals 1, i.e. a new iteration starts every cycle, the queue sizes are maximum. If *ItIr* equals 2, then a new iteration starts every other cycle, and in that case the required queue size of all nodes are halved and so on. Therefore, the final queue size of a node $i$ is given by,

$$Q^{node}(i) = Q^{node}Expand(i) \,/\, ItIr \quad (8)$$

Finally, the total number of registers used by all the queues of the sDFG will be given by,

$$Tot(Q^{node}(sDFG)) = \sum_{\forall i \in V} Q^{node}(i) \quad (9)$$

## 3. EXPERIMENTAL RESULTS

The effectiveness of the proposed estimation scheme is evaluated by comparing the queue sizes determined by the Proteus scheduler and our estimation technique for a set of industrial applications. The results presented in Table I assume $E[\Delta i] = slack(i)$ and $E[\Delta j] = slack(j)$, which is practically the queue sizes if $i$ is scheduled ASAP and $j$ in ALAP. We are currently investigating alternative formulations for $E[\Delta i]$ and $E[\Delta j]$ for better accuracy.

Note that, these estimated queue sizes are for a given sDFG only. In reality, several such kernels may be running concurrently and in that case the queue sizes will be larger.

**Table I. Register Estimation**

| ***Design*** | ***# nodes*** | ***Proteus*** | ***Estimated*** | ***% Error*** |
|---|---|---|---|---|
| dctCol | 85 | 36 | 39 | 8.3 |
| dctRow | 95 | 42 | 43 | 2.4 |
| hpf_med_cc | 157 | 76 | 55 | -27.6 |
| lpf_gc_rgb | 221 | 104 | 57 | -45.2 |
| lpr | 67 | 58 | 38 | -34.5 |
| open | 30 | 44 | 36 | -18.2 |
| quant | 10 | 14 | 14 | 0.0 |
| RsvpLPR | 67 | 58 | 38 | -34.5 |
| RsvpLPR_u2 | 102 | 57 | 76 | 33.3 |
| ***Average*** | 92.7 | 54.3 | 44.0 | **-19.0** |

## 4. CONCLUSIONS

In this work we propose a probabilistic *push-and-pull* register queue size estimation technique. A naïve estimation, which does not take resource constraints into account, can differ from the post-schedule results by as much as 45%. More precise estimations for expected expansion in queue sizes, which take resource constraints and other scheduler strategies into account is in progress as presented in this abstract.

## 5. REFERENCES


[1] S. Ciricescu, R. Essick, B. Lucas, P. May, K. Moat, J. Norris, M. Schuette, and A. Saidi, "The Reconfigurable Streaming Vector Processor (RSVP™)," IEEE/ACM International Symposium on Microarchitecture, 2003.

[2] S. Chiricescu, S. Chai, K. Moat, B. Lucas, P. May, J. Norris, R. Essick, and M. Schuette, "RSVP II: A Next Generation Automotive Vector Processor," 2005.

[3] C. Hwang, Y. Hsu, and Y. Lin, "PLS: A scheduler for pipeline synthesis," *IEEE Transactions on CAD/ICAS*, vol. 12, pp. 1279-1286, 1993.