
Conference or Workshop Item:
Burmester Campos, Pedro orcid.org/0000-0001-6933-3282, Dahir, Nizar, Bonney, Colin Andrew et al. (3 more authors) (2017) XL-STaGe:A Cross-Layer Scalable Tool for Graph Generation, Evaluation and Implementation. In: International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XVI), 17-20 Jul 2016.

https://doi.org/10.1109/SAMOS.2016.7818372

XL-STaGe: A Cross-Layer Scalable Tool for Graph Generation, Evaluation and Implementation

Pedro Campos, Nizar Dahir, Colin Bonney, Martin Trefzer, Andy Tyrrell, Gianluca Tempesti

Department of Electronics, University of York, York, UK
Email: {pedro.campos, nizar.dahir, cab523, martin.trefzer, andy.tyrrell, gianluca.tempesti}@york.ac.uk

***Abstract*—This paper presents XL-STaGe, a cross-layer tool for traffic-inclusive directed acyclic graph generation and implementation. In contrast to other graph-generation tools which focus on high-level DAG models, XL-STaGe consists of a set of processes that generate the task-graphs as well as a detailed process model for each node in each graph. The tool is highly customizable, with many parameters that can be tuned to meet the user's requirements to control the topology, connection density, degree of parallelism and duration the task-graph. Moreover, two use cases are presented, a high-level one, which illustrate the benefit of the developed tool in application mapping and a circuit-level one to verify the accuracy of the XL-STaGe process models when implemented in hardware.**

***Index Terms*—DAG, task-graph, process model, networks-on-chip, CAD tools.**

I. Introduction

Research in many-core systems and parallel computing, with hundreds or thousands of cores, lacks application benchmarks that are required for design-space exploration of such systems. These benchmarks should embed various application characteristics such as the number of tasks, task dependencies and input/output degrees of the tasks. Many synthetic task-graph models previously proposed by literature. For example, the GGen [1] task generator implements a selection of standard task graph generation algorithms (Erdos-Renyi, Layer-by-Layer [2], Fan-in / Fan-out [3], [4] and Random Orders) allowing researchers to choose the generation algorithm that best suits their expected workload.

The majority of application task-graph models express the application as a set of nodes (tasks) and arcs (dependencies) usually in the form of a DAG (directed acyclic graph). Another characteristic that may be added to these graphs is the communication requirements among the tasks. However, these DAG models usually describe the high-level behaviour of the underlying application and they lack models for the temporal behaviour of these tasks. In other words, the process model or the real-time behaviour of these processes is lacking. This narrows the use of these models to high-level design-space exploration such as application mapping and routing and does not allow lower-level explorations such as accurate hotspot identification, buffer-level requirements or identifying the voltage-frequency pair of a node/link in systems with DVFS support. These low-level design objectives require detailed process models that can run on real hardware (or hardware simulation) to test the system in real-time and evaluate real-time design performance.

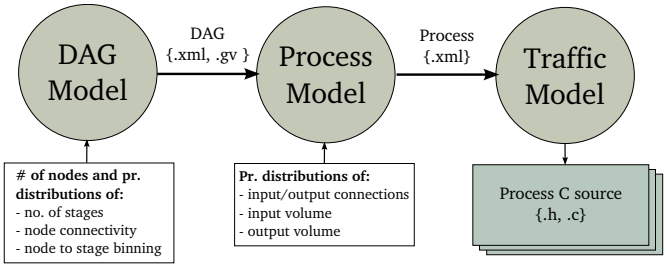


Fig. 1: Illustration of XL-STaGe models flow, input parameters and input/output file formats

In this work we present XL-STaGe, a cross-layer tool for DAG generation and implementation. It consists of a set of tools that generate the task-graph as well as the process models for each node in this graph, as shown in Figure 1. To this end the main contributions of this paper are:

- We present XL-STaGe, a cross layer application task graph generation tool.
- The generated graphs embed high-level tasks and traffic as well as low-level process implementation.
- The tool set is demonstrated through use-cases for both high-level DAG models and low-level process.
- The tool is freely available online [5].

Section II details the process of generating the directed acyclic graphs, Section III describes how the process model is embedded into the generated graph, and Section IV explains how the edge throughputs are calculated based on the process model. Section V presents some use-cases for the developed tool, and a discussion of the contributions of XL-STaGe can be found in Section VI.

II. Graph Model

A. DAG Task Graph Model

The task graph model considered in this work is a directed acyclic graph (DAG), $G(V, E)$ where,

- V: is the set of *vertices* or *nodes*,
- E: is the set of *arcs* or *edges*.

Multiple input and output edges are allowed and the edges are identified by the nodes they are connecting. So, each directed edge $\varepsilon_{i,j} \in E$ connects node $i \in V$ to node $j \in V$. Let $S \subset V$ be the set of the nodes with no predecessors (*sources*)

978-1-5090-3076-7/16/$31.00 ©2016 IEEE

and $T \subset V$ be the set of nodes with no successors (*sinks*). A path of G is the sequence of edges $(i_1, i_2, ..., i_k)$ such that every edge $(i_n, i_{n+1}) \in E$. Using this analogy, the nodes in the generated graph are distributed to *processing stages* $k \in K$. Nodes in higher processing stages have their dependencies in lower stages. The stage of all the sources S is the first stage ($k = 1$) while the stage of all sinks T is last stage ($max\{K\}$).

B. Parameters and Assumptions

The tool starts by generating the task graph with a set of nodes representing the processes and then generating an executable process based on a process model (see Figure 1). The task graph is modelled as a stochastic graph. In other words the graph is a random variable in a set of task graphs. We fix the number of nodes but all other characteristics of the task graph are randomly generated based on a given probability distribution. The parameters used to create the task graph are:

1) **Number of nodes or processes in the graph** - N is fixed in the current tool setup but can be made random within a range based on user preference.
2) **The number of processing stages** - K is a random variable bounded by values that are computed by establishing a relationship between K and N to control the height of the graph (max number of nodes in a stage) and its length (max path length from a source to sink) for the same number of nodes. Figure 2a and 2b shows examples of graphs with different heights and widths. The height of the graph determines to the number of parallel processing threads that exist in this graph while the width determines the length of these threads.
3) **Node to rank distribution** - or $Pr(v = k)\ \forall v \in V$, $\forall k \in K$. This probability follows a normal distribution with μ_k and σ_k set by the user. μ_k and σ_k can be tuned to control the shape of the graph. For example the graph can be made divergent (less sources than sinks), convergent (more sources than sinks) as shown in Figures 2c and 2d, respectively.
4) **Connection probability** - or $Pr(\varepsilon_{i,j} = 1)\ \forall i, j \in V$. A parameter between 0 and 1, set by the user, represents the initial value of this probability. It drops by a factor that is proportional to the difference in processing stages between the nodes. In other words, connections are more likely to exist between nodes with closer processing stages and less likely to exist between nodes with distant processing stages.

C. Output Products

Following a review of available description languages for graphs, the DOT Graph Description Language was chosen for its simplicity and available software [6], [7]. The DOT language is supported by GraphViz which proved to be able to provide a clear, uncluttered visual representation of our graphs. A utility called DOT also provides the means of producing a pdf, eps or png file of the visual representation of a graph using standard free tools. Additionally, the DOT language allows groups of nodes to be defined as having the same rank (processing stage) and provides functionality for controlling the size, shape, colour and text of nodes.

The tool uses two formats for the output DAGs (as illustrated by Figure 1). The first is DOT format (in the form of gv file) while the second is xml that which includes tags describing the nodes, edges and processing stages.

III. Process Model

XL-STaGe provides a framework to incorporate low-level network operational details into a standard directed acyclic graph (DAG). A DAG will consist of edges and nodes, but the typical high-level graph representation does not include the dependencies that exist between inputs and outputs of a particular node, which will ultimately be used to determine the throughput between any two nodes. This notion of causality is a key aspect of the approach undertaken for XL-STaGe. After a directed acyclic graph is generated as part of the DAG model process – as described in section II – an additional layer of information can be added, which describes the behaviour of the application at the data-flow level.

A. Low-Level Model

A typical software application implemented on a many-core system will be split into many tasks or processes that can be mapped to individual cores. These tasks will take in a set of inputs, process them, and generate data which will either be directed to another task, or will be part of the application's output.

The algorithm behind XL-STaGe makes the following assumptions for the operation of a generic application:

- **Data packets** – data travels across the application in the form of packets of a user-specified size.
- **Data buffering** – the network layer should have the ability to store unprocessed data at each of its nodes.
- **Thread/process flexibility** – if all outputs of a single node depend on all its inputs, the task/process consists of a single thread; otherwise the task/process can be multi-threaded.
- **Triggering condition** – in order to activate the output of a task/process, a given volume of data from each connected input must be received.
- **Dependency of inputs** – different tasks or threads will require varying volumes of data from each associated input.
- **Output volume** – different tasks or threads will generate varying volumes of data once the triggering condition is met, and processing is completed.

These assumptions lay the foundation for the cross-layer approach of XL-STaGe. The process model adds information to the application layer, or high-level representation, which can be used to generate traffic at the network layer.

B. I/O Connectivity

The first step of the algorithm used to create the process model in XL-STaGe consists of establishing connections between the outputs and inputs of each node (representation

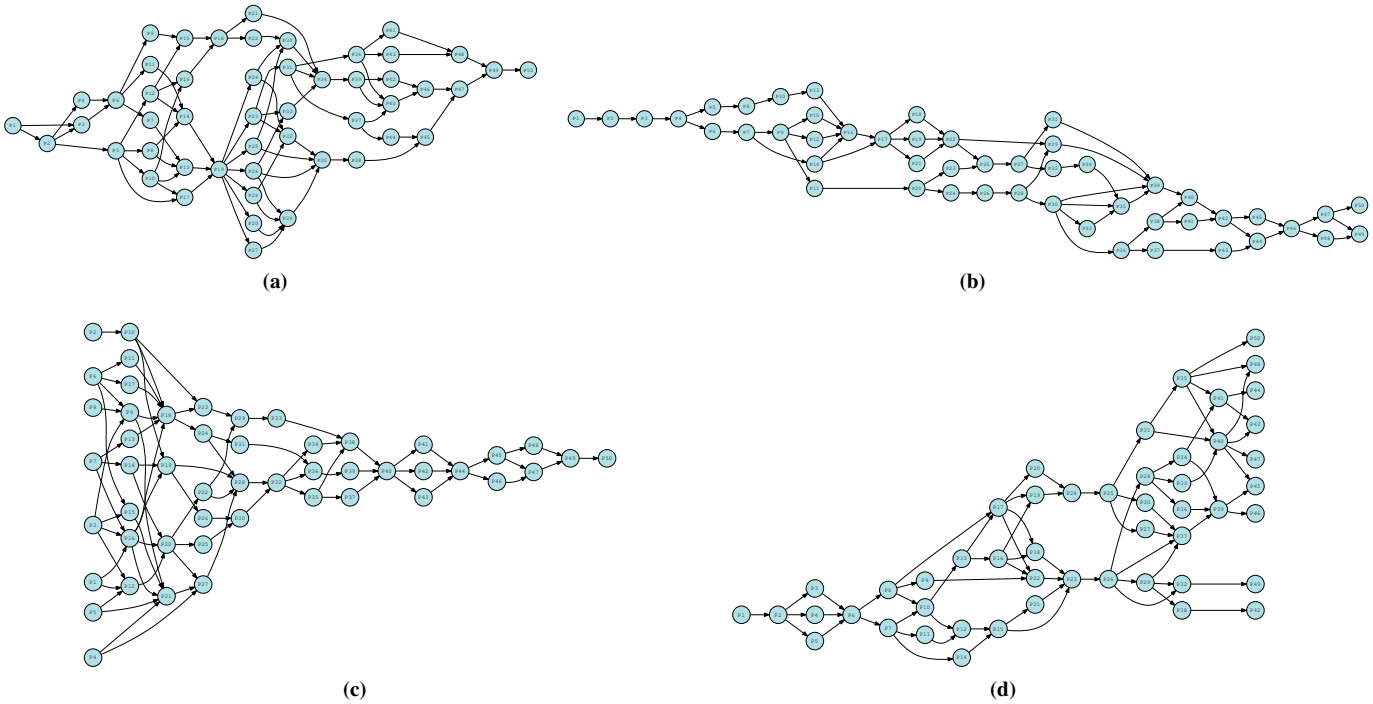


Fig. 2: Various DAG examples, (a) & (b) illustrate high and low number of processing stages, respectively, (c) a convergent graph, and (d) a divergent graph.

of a task) in the DAG that was generated by the previous modelling stage. These connections are established based on a configurable parameter which specifies the probability of a dependency between any input-output pair of a single node. For each output of a particular node, the tool establishes a connection with each of the available inputs based on the specified probability, $p(c)$. To ensure consistency, all outputs are connected, any unconnected inputs are then connected to any of the available outputs, selected at random.

As mentioned in the thread/process flexibility assumption, a node with fully connected I/Os will represent a single-threaded task. Otherwise, each output of the modelled task will represent a single thread.

C. Input Dependencies and Output Volumes

The triggering condition previously described states that for each output of the task or thread, a given volume of data from each connected input must be received before data can be generated. This volume of data, measured in data packets per output/input pair, is defined as the **input dependency**. The amount of data travelling across an output through time is defined as **output volume**, and the expression to calculate it is described in Equation 1, where V_j is the volume of data generated by the task once a triggering condition is met, $d_{i,j}$ is a flag determining whether or not the condition has been met for each input i connected to output j, $b_i(t)$ is the number of unprocessed (buffered) data packets from input i of the node through time, and $D_{i,j}$ is the number of packets from input i **required** by output j to trigger processing. Once these values are established, each node on the graph will have the information depicted in Figure 3.

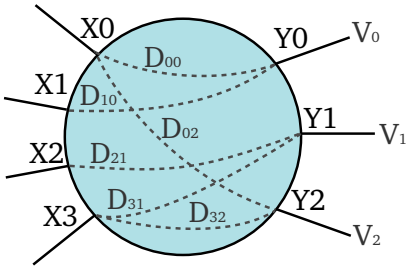


Fig. 3: A 4-input, 3-output node after going through the I/O connectivity processing step of XL-STaGe. Inputs $x_{0\ldots4}$ are connected to outputs $y_{0\ldots2}$ according to probability of connection $p(c)$. Dependencies $D_{i,j}$ and output data volumes V_j are added in the next step of the process.

$$O_j(t) = V_j \prod_{i=0}^{n} d_{i,j}(t), \quad \text{where} \quad d_{i,j}(t) = \begin{cases} 1, \text{if } b_i \geq D_{i,j} \\ 0, \text{otherwise} \end{cases} \tag{1}$$

The condition $d_{i,j}$ for an input i connected to an output j of a particular node is met once enough data is received to allow for processing. As an example, some video streaming applications require a set amount of frames to be received before running image manipulating algorithms.

D. Input and Output Products

The process model takes a directed acyclic graph as an input, in the form of an XML file, along with the specification of the range of values for data volume V and dependency D, as well as the value for the probability of connection, $p(c)$.

The output of this process is an XML file with the values of V_j and $D_{i,j}$, sampled from normal distributions

centered around a user-specified value, and also with a user-configurable range. The standard .gv file is also edited to include the calculated relative throughputs on the graph.

IV. Traffic Model

The process model creates the input dependencies and output volumes for each node on the task graph. The next step in the XL-STaGe algorithm is to extrapolate the resulting throughput across each edge on the graph, based on the $D_{i,j}$ and V_j values. This is a crucial aspect of the XL-STaGe framework, as the output of this step is a task-graph complete with calculated edge throughputs that have a foundation on a detailed process model. This foundation is critical for the layer-crossing reach of XL-STaGe, as it allows for the seamless implementation of a user-generated traffic model from the application layer onto a multi-node platform which can make up a network layer.

A. Throughput Calculation

Based on the values of $D_{i,j}$ and V_j, it is possible to calculate the *relative throughputs* across each edge of the task graph, with respect to the throughput of the data source. For example, a task with one input x_0 and one output y_0, with a D_{00} of 2 data packets and an output volume V_0 of one data packet, will generate data at half the rate of its input, and therefore its *relative throughput* would be 0.5. A more complex example could involve a node with two inputs x_0 and x_1 and one output y_0, with $D_{00} = 2$, $D_{10} = 4$ and $V_0 = 3$. If both inputs receive data at the same rate (*i.e.* they share the same input throughput), then it would take twice as long to receive enough packets to satisfy both d_{00} and d_{10}, and therefore the throughput at output Y_0 would be $\frac{3}{4}$ of the input throughput.

The relative throughput ξ_j for an output j of a given node can then be calculated through the expression described in Equation 2, where the relative throughput ξ_i of each input i connected to node output j is divided by the dependency $D_{i,j}$ and multiplied by the data volume V_j.

$$\xi_j = V_j \min \frac{\xi_i}{D_{i,j}}, \text{subject to } D_{i,j} \geq 0 \tag{2}$$

Based on this concept, it becomes possible to perform the same calculation for all edges in the graph, allowing for an analytical estimation of the throughputs across the graph all the way to the sink nodes. Figure 4 illustrates an example of a 9-node graph, which incorporates information from the process model in order to calculate the edge's relative throughputs. Conceptual source- and sink-nodes are added to illustrate the rate at which data is introduced into the graph and extracted from it, respectively.

B. Input and Output Products

The traffic model takes a directed acyclic graph with the values of V_j and $D_{i,j}$ as an input, in the form of an XML file, and generates a header file with the process model information, written in C, along with the source code that can be incorporated into a template to run on a standard processor.

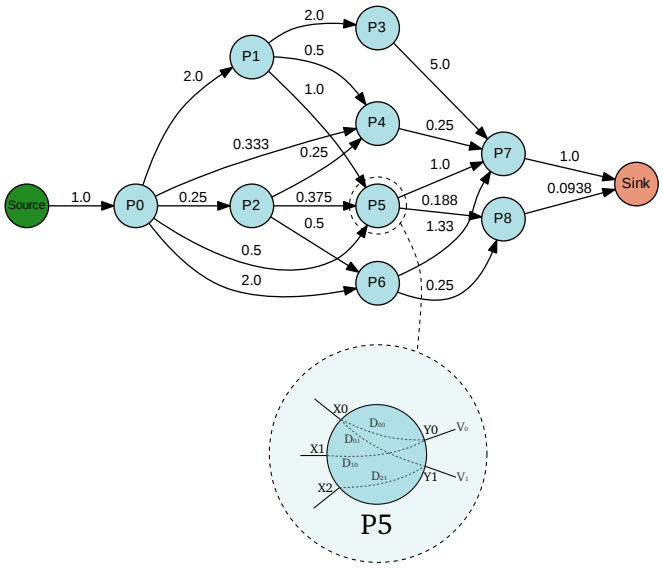


Fig. 4: Example of a 9-node graph, with the additional conceptual source- and sink-nodes also represented. The numbers associated with the edges represent the throughput ratios of node-to-node connections with respect to the source throughput. A zoomed-in representation of node P5 illustrates the meaning of both the input dependencies $D_{i,j}$ and the output volumes V_j, with inputs ranging from 0-2 and outputs from 0-1.

At this point, information regarding which processing node will run which task can also be generated (task mapping), allowing for quick platform evaluations.

V. Use Cases

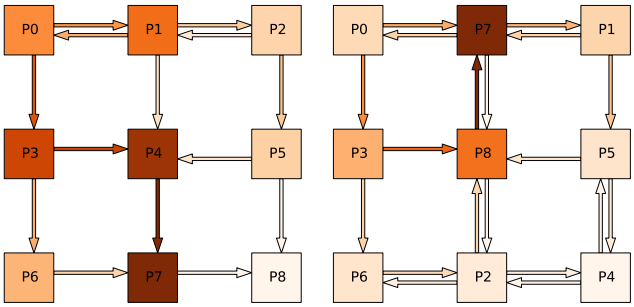
A DAG whose structure is described in the process and traffic models can be implemented either virtually or physically on a hardware platform. In the use cases that follow, the virtual implementation assumes an array of homogeneous processing nodes, and the physical implementation, described later in the text, uses an array of Zynq System-on-Chip, but other hardware implementations are also possible. This approach allows the user to experiment with different task mapping approaches and evaluate their performance.

A. System Level

At the system level, the low-level implementation details are discarded, and only the calculated throughputs between nodes are used to provide estimates of traffic load across a virtual network. This section describes the functionality of XL-STaGe for the virtual implementation case, where an array of processing nodes of a user-defined size is populated with nodes from the traffic model previously developed.

1) Mapping: Depending on which processing nodes the traffic nodes are mapped to, the chosen routing strategy, and the network topology, the load will be distributed differently across the virtual array. As an example, Figure 5 shows a comparison between two different task mappings of the graph illustrated in Figure 4.

2) Evaluation: By providing XL-STaGe with a task mapping associating with the tasks with processing nodes, a topology definition of the virtual node array, and an optional



(a) **(b)**

Fig. 5: An example of two different task mappings (a) & (b) for the same XL-STaGe generated task graph on a 9-node virtual array with 3 rows and 3 columns, using deterministic x-y routing and a Von Neumann neighbourhood for the physical links between nodes. A lighter colour denotes a lower throughput, and darker areas represent virtual links or nodes of higher traffic load.

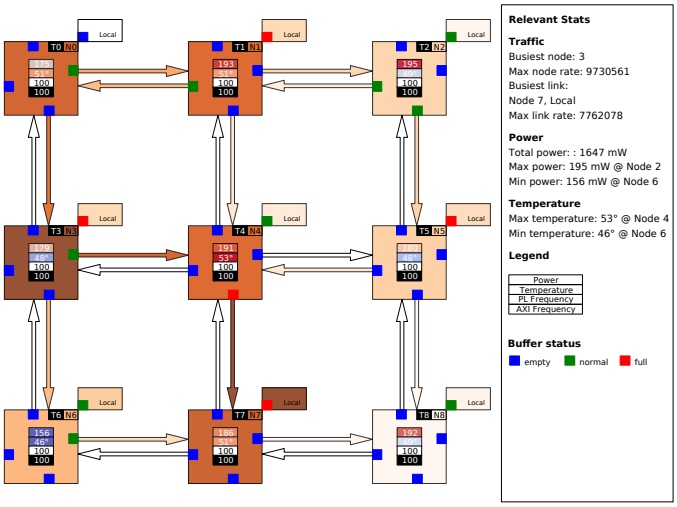


Fig. 6: The traffic distribution and other real-time parameter, including power, temperature, and NoC buffer (fifo) level status, that result from running the task-graph on the hardware platform. The frame is generated by a real-time visualiser that we developed form monitoring of the hardware platform. The parameters are updated every 1 sec.

set of physical constraints – such as maximum node or link throughput – the performance of a particular mapping can be evaluated in terms of load distribution, number of path alternatives, or any other metric that the user may consider relevant.

In order to minimise the workload on the network for a particular task mapping of an XL-STaGe generated graph, a typical approach is to minimise the Hamming distance – or $|X_s - X_d|$ and $|Y_s - Y_d|$, the distances in the x- and y- axis respectively – between any two source (s) and destination (d) nodes running tasks that are connected at the task-graph level. An optimisation goal would then be to explore task mappings which minimise the sum of the Hamming distances for all connected nodes. Inserting the throughputs ξ_g associated with each edge g into the sum and minimising it will result in task mappings which bring nodes with large edge throughputs together, therefore minimising the volume of data circulating across the network, or the *network energy* (ϕ_G). Equation 3 represents the expression describing ϕ_G for a given graph, G, after mapping M is applied.

$$\phi_G(M) = \sum_{g=0}^{N} \xi_g(|X_s - X_d| + |Y_s - Y_d|) \qquad (3)$$

As an example, the expression for ϕ_G for the mapping illustrated in Figure 5b is roughly 20% higher than that of the mapping illustrated in Figure 5a, since the former uses up fewer communication channels than the latter.

This kind of system level metrics are embedded in XL-STaGe and could be exploited to find an optimal task mapping for a task-graph/virtual platform pair, or to evaluate a task mapping technique or algorithm.

B. Circuit-Level

The previous section presented a use-case for high-level analysis of the graph. This section presents the results of experiments for low-level verification of the process model used in the tool. This is done to verify the assumptions made when developing the process model. Namely, that the input-output dependency process model results in a stable traffic distribution when the processes are distributed and run independently across a many-core system. This is achieved through a physical implementation on a dedicated hardware system, which also allows to verify that the resulting traffic distribution should match the calculated one.

We start by mapping the task graph to a previously custom 9-node hardware platform and running the processes on the cores of this platform. The resulting data throughputs are then compared with the expected throughputs computed by analysing the DAG as described in Section IV-A.

1) Architecture Summary: To run our circuit-level experiments, we use a previously developed hardware platform consisting of 9 Xilinx Zynq development boards, based on Zynq-7000 System-on-Chip devices. These boards contain a processing system (PS), that includes a dual-core ARM Cortex-A9 processor, plus programmable logic. Custom connection boards were designed to implement the interconnection network (NoC). The routing logic is implemented on the programmable logic of the Zynq chip. The topology is a 2D mesh with 4 bi-directional channels per node plus the local channel to/from the PS. The platform supports both adaptive and deterministic routing but, in this experiment, we use the deterministic XY routing. Round-robin is used for arbitration if more than one input channel requires access to an output channel at the same time. This resulting architecture of distributed may-core system is shown in Figure 7.

2) Process Model Verification: Here we present an evaluation for the process model through the verification of the accuracy of the XL-STaGe model by comparing the throughput measured for the DAG shown in Figure 4 and when implemented on the physical platform architecture and applying a throughput of 1,024,000 flit/sec at the source. This throughput is bellow the the platform saturation throughput which is

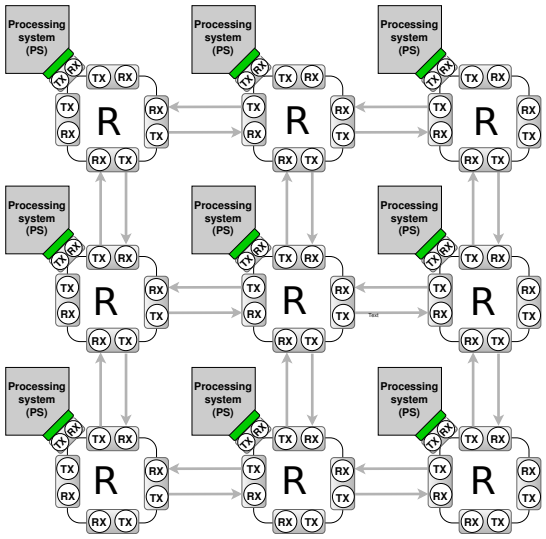


Fig. 7: Illustration of the experimental system platform architecture.

found to be $\sim 2,000,000$ flit/sec. While data packets can have multiple flits, in this experiment, data packet size is set to one flit. Table I compares the measured communication throughputs in each NoC link, including sinks throughputs, with the calculated one. It can be seen that the difference between the calculated throughputs and the measured ones is negligible, which verifies the accuracy of the developed traffic and process models as well as the functionality of the hardware platform. This also verifies the assumptions made when developing the process model.

VI. Discussion & Conclusions

This paper presents XL-STaGe, a cross layer tool for task-graph generation and implementation. The major contribution of XL-STaGe compared to previous works is the inclusion of a detailed process model. As a result traffic pattern results from the dependencies between inputs and outputs in the process and is not completely random. This makes XL-STaGe suitable for low-level evaluation of hardware platforms. Moreover, users can make partial use of the tool flow. For example, if a user provides XL-STaGe with a process model of their real application graph, they can make use of the model generation and evaluation methods described in this paper.

Two use-cases of the tool are presented. A system-level one which shows the benefits of the tool in comparing different application mappings in terms of workload distribution. The second use-case is circuit-level where a task-graph generated by the tool is implemented on a hardware platform. The accuracy of the traffic model is verified by comparing the measured throughputs with the ones calculated using the traffic model where good match is found with error of 0.082%. The results of the circuit-level use-case also verifies the correct functionality of the hardware platform and is used to evaluate the platform throughput capacity as well as power consumption and temperature. These circuit-level parameters highlight the advantages of the detailed implementable process and traffic models that XL-STaGe generates. Other uses of these models include evaluating hardware designs in terms of other parameters. For example throughput capacity (QoS) and power consumption can be evaluated against frequency and voltage in systems with DVFS.

TABLE I: Calculated and measured throughputs for the DAG shown in Figure 4 implemented on a 9 board system and assuming a source throughput of 1,024,000 flit/sec. . The links are denoted as $< start\_node > \rightarrow < end\_node >$. Nodes are denoted as 0 (top left) to 8 (bottom right) and (S) is the sink.

Link	Measured (flit/sec)	Calculated (flit/sec)	Error (%)
0→1	3159676	3157333	0.074
1→2	1792058	1792000	0.003
3→4	5107595	5105008	0.051
4→5	0	0	0
6→7	1615872	1621333	0.337
7→8	255588	256000	0.161
1→0	2552247	2560000	0.303
2→1	765774	768000	0.290
4→3	0	0	0
5→4	1024035	1024000	0.003
7→6	0	0	0
8→7	0	0	0
0→3	4600027	4608000	0.173
1→4	1108393	1109333	0.085
2→5	1918079	1920000	0.100
3→6	2558243	2560000	0.069
4→7	6400017	6400000	0.001
5→8	191214	192000	0.409
3→0	0	0	0
4→1	0	0	0
5→2	0	0	0
6→3	0	0	0
7→4	0	0	0
8→5	0	0	0
7→(S)	1023500	1024000	0.048
8→(S)	95976	96000	0.025
Mean Error			**0.082**

Acknowledgements

The authors would like to thank EPSRC for their support through the Graceful project grant (EP/L000563/1) of which this work is part of, as well as the Bio-inspired Adaptive Architectures and Systems project grant (EP/K040820/1).

References

[1] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J.-M. Vincent, and F. Wagner, "Random graph generation for scheduling simulations," in *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010, p. 60.

[2] T. Tobita and H. Kasahara, "A standard task graph set for fair evaluation of multiprocessor scheduling algorithms," *Journal of Scheduling*, vol. 5, no. 5, pp. 379–394, sep 2002. [Online]. Available: http://doi.wiley.com/10.1002/jos.116

[3] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," in *Proceedings of the 6th international workshop on Hardware/software codesign*. IEEE Computer Society, 1998, pp. 97–101.

[4] K. Vallerio, "Task graphs for free (tgff v3. 0)," *Official version released in April*, vol. 15, 2008.

[5] N. Dahir, P. Campos, and C. Bonney, "XL-STaGe," https://github.com/nizarsd/xl-stage, 2016.

[6] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull, "Graphviz and dynagraphstatic and dynamic graph drawing tools," in *Graph drawing software*. Springer, 2004, pp. 127–148.

[7] E. R. Gansner and S. C. North, "An open graph visualization system and its applications to software engineering," *Softw. Pract. Exper.*, vol. 30, no. 11, pp. 1203–1233, Sep. 2000.