×pipesCompiler: A tool for instantiating application specific Networks on Chip

Antoine Jalabert
CEA
LETI-DSIS
antoine.jalabert@cea.fr

Srinivasan Murali
CSL
Stanford University
smurali@stanford.edu

Luca Benini
DEIS
Univ of Bologna
lbenini@deis.unibo.it

Giovanni De Micheli
CSL
Stanford University
nanni@stanford.edu

Abstract

Future Systems on Chips (SoCs) will integrate a large number of processor and storage cores onto a single chip and require Networks on Chip (NoC) to support the heavy communication demands of the system. The individual components of the SoCs will be heterogeneous in nature with widely varying functionality and communication requirements. The communication infrastructure should optimally match communication patterns among these components accounting for the individual component needs. In this paper we present `×pipesCompiler`*, a tool for automatically instantiating an application-specific NoC for heterogeneous Multi-Processor SoCs. The* `×pipesCompiler` *instantiates a network of building blocks from a library of composable soft macros (switches, network interfaces and links) described in SystemC at the cycle-accurate level. The network components are optimized for that particular network and support reliable, latency-insensitive operation. Example systems with application-specific NoCs built using the* `×pipesCompiler` *show large savings in area (factor of 6.5), power (factor of 2.4) and latency (factor of 1.42) when compared to a general-purpose mesh-based NoC architecture.*

Keywords: Systems on Chips, Networks on Chips, latency-insensitive design, application-specific, SystemC.

1 Introduction

With increasing transistor density, the number of cores on a chip and the communication demands between them is rapidly increasing. System interconnect scalability is limited for state-of-the-art SoC communication architectures based on shared communication resources. *Networks on chip* (NoC) architectures have been proposed to address the scalability challenge [1, 2, 3, 8]. NoCs are scalable and compatible with design and reuse of cores, which is a critical feature required by SoC designers to meet tight time-to-market constraints [4].

An important design decision for NoCs is the choice of topology. Several researchers [10, 11, 12, 13] envision NoCs as regular topologies (such as *mesh* networks and *fat trees*), which are suitable for interconnecting homogeneous cores in a chip multiprocessor (Figure 1(a)). However, many SoCs involve heterogeneous cores having varied functionality, size and communication requirements. If a regular interconnect is designed to match the requirements of few communication-hungry components, it is bound to be largely over-designed with respect to the needs of the remaining components. This is the main reason why most current SoCs use irregular topologies like bridged busses and/or dedicated point-to-point links [14].

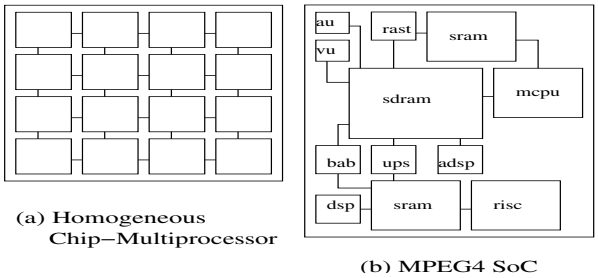


Figure 1. Homogeneous CMPs and heterogeneous SoC applications

As an example, consider the implementation of an *MPEG4 decoder* [5], depicted in Figure 1(b), where blocks are drawn roughly to scale and links represent inter-block communication. First, the embedded memory (SDRAM) is much larger than all other cores and it is a critical communication bottleneck. Block sizes are highly non-uniform and the floorplan does not match the regular, tile-based floorplan shown in Figure 1(a). Second, the total communication bandwidth to/from the embedded SDRAM is much larger than that required for communication among the other cores. Third, many neighboring blocks do not need to communicate. Even though it may be possible to implement MPEG4 onto a homogeneous fabric, there is a significant risk of either under-utilizing many tiles and links, or, at the opposite extreme, of achieving poor performance because of local congestion. These factors motivate the use of an application-specific on-chip network [15].

With an application-specific network the designer is faced with the additional task of designing network components (e.g., switches) with different configurations (e.g., different I/Os, virtual channels, buffers) and interconnecting them with links of uneven length. These steps require

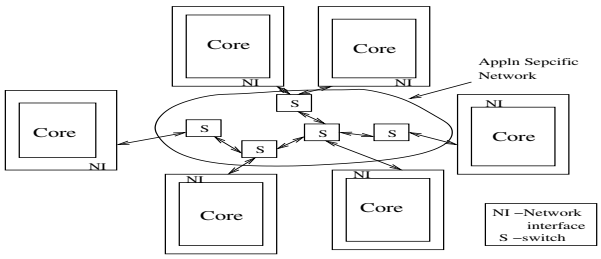


Figure 2. NoC architecture block diagram

significant design time and the need to verify network components and their communications for every design. In this paper we describe a tool that bridges the design gap for heterogeneous application-specific NoCs.

The ×`pipesCompiler` automatically instantiates network components (routers, links, network interfaces) for a specific NoC topology, using the ×`pipes` library of SystemC soft macros defined at the cycle-accurate and signal accurate level [6]. The ×`pipes` library is aggressively designed for high performance: links can be pipelined to an arbitrary degree to decouple clock cycle time from worst-case link delay (this mode of operation has been called *latency insensitive* in recent literature [9]). Latency insensitive link-level error control is fully supported, ensuring robustness against communication errors.

Even though ×`pipes` can be instantiated as a high-performance regular NoC, its most innovative feature is that all its components are highly parameterized, and they can be tailored to the communication needs of a specific architecture. Thus, the ×`pipesCompiler` can instantiate optimized NoCs. Significant improvements in area, power and latency are achieved with respect to regular NoC architectures, as demonstrated by several case studies detailed in the paper.

2 The ×`pipes` Architecture

In this section we present a brief description of the architecture of switches, links and network interfaces that form the ×`pipes` library. We refer the reader to [6] for a detailed description of these components. A conceptual picture of the network architecture is shown in Figure 2. It supports packet-switched communication, with source routing and wormhole flow-control. Source-based routing results in lightweight switch implementations when compared to dynamic routing and wormhole flow-control results in reduced buffering at each switch. Cores can be plugged into the network, provided they are OCP compliant [18] and the network communication protocols are completely hidden to the cores. The individual components are detailed below.

2.1 Network Interface

The *Network Interface* (NI) connects the core to the NoC. It converts the end-to-end OCP transactions into packets that are to be transmitted through the network. The NI builds the packet header using the routing information for

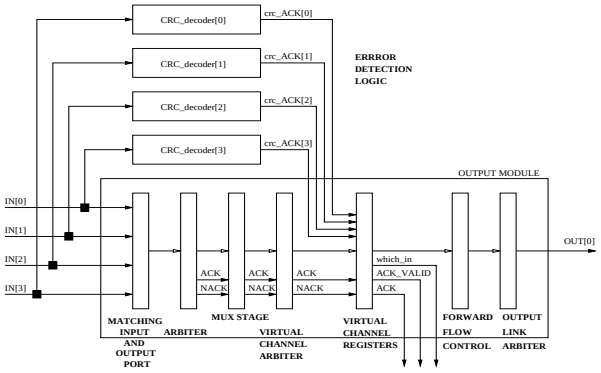


Figure 3. Pipelined architecture of a switch

the destination stored in a look-up table. The packet is broken down into header and payload flits and the flits are injected into the network at the rate of one flit every clock cycle. The NI synchronizes the network requests with the consuming rate of the core. To keep the interface complexity low, the NI supports only a single outstanding read operation, but an arbitrary number of write transactions can be carried out after an outstanding read.

2.2 Switch Architecture

Switches in the ×`pipes` library are deeply pipelined to maximize the operating frequency. The pipeline structure for a single output module is shown in Figure 3, and the structure is repeated for each output. Forward flow control is used and a flit is transmitted to the next switch only when adequate storage is available in that switch. Switches support multiple virtual channels and a physical link is assigned to different virtual channels on a flit-by-flit basis, thereby improving network throughput. The CRC decoders for error detection work in parallel with the switch operation, thereby hiding their impact on switch latency.

For latency insensitive operation, the switch has virtual channel registers to store $2N+M$ flits, where N is the number of link pipeline stages and M is an architecture dependent parameter (12 cycles in our design). The reason is that each transmitted flit has to be acknowledged before being discarded from the buffer. Before an ACK is received, the flit has to travel across the link (N cycles), an ACK/NACK decision has to be taken at the destination switch (a portion of M cycles), the ACK/NACK signal has to be propagated back (N cycles) and recognized by the source switch (remaining portion of M cycles). During this time, other $2N + M$ flits are transmitted but not yet ACKed.

2.3 Link Architecture

The links of an irregular NoC are of varying lengths, and it takes a varying number of clock cycles to traverse the links. In order to maximize throughput, the links are subdivided into basic segments that require a single clock cycle for traversal, making the links *latency insensitive* by pipelining flits through them. This pipelining is applied to both data and control lines. As previously discussed, the

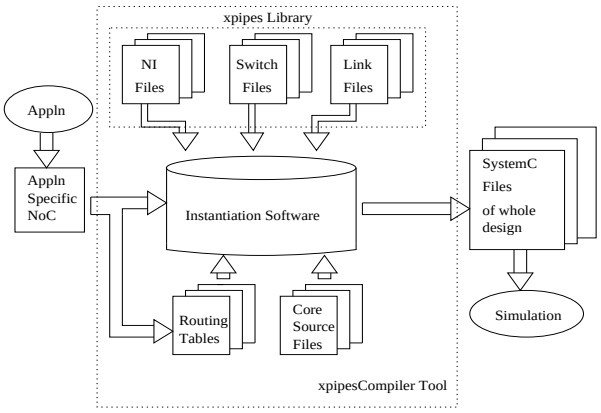


Figure 4. NoC synthesis flow

switches and network interfaces have enough buffering resources and their functional correctness depends only on the flit arriving order and not on timing. This ensures a correct latency insensitive operation of the whole system.

2.4 Instantiation-time Parameters

All network components can be specialized through instantiation-time parameters. Some parameters are global and they affect all component instances. Global parameters are: flit size, address space of cores, degree of redundancy (minimum distance) of the CRC error-detection code to be used on the links, number of bits used for packet sequence count (for end-to-end flow control), maximum number of hops between any two network nodes (used for header sizing), number of flit types (to support special flits, for instance control flits with no payload), number of packet types (which can be used by higher level protocols).

As for the local parameters affecting single network component instances, we have the following. For the network interface: number of data and address lines and maximum burst length in the OCP connection between NI and the core, type of interface (master/initiator, slave or both), flit buffer size in the output port (which enables the network interface to continue packetization even when the network link is congested) and content of the routing table. For the switch: number of ports, number of virtual channels, link buffer size for each port (which relates to the number of pipeline stages in the corresponding link). For each link, the number of stages can obviously be specified.

3 The ×pipesCompiler

The complete NoC design flow is depicted in Figure 4. From the specification of an application, the designer (or a high-level analysis and exploration tool) creates a high-level view of the SoC floorplan, including nodes (with their network interfaces), links and switches. Based on clock speed target and link routing, the number of pipeline stages for each link is also specified. The information on the network architecture is specified in an input file for the ×pipesCompiler. Routing tables for the network interfaces are also specified. The tool takes as additional input the SystemC library of soft components described in the previous section. The output is a SystemC hierarchical description, which includes all switches, links, network nodes and interfaces and specifies their topological connectivity. The final description can then be compiled and simulated at the cycle-accurate and signal-accurate level. At this point, the description can be fed to back-end RTL synthesis tools for silicon implementation (the details of the back-end synthesis process are not covered in this paper).

```
.module_sw -NAME=SW_0 -NPORT=3 -NVC=4
.module_sw -NAME=SW_1 -NPORT=2 -NVC=2
.module_sw -NAME=SW_2 -NPORT=3 -NVC=2
.module_sw -NAME=SW_3 -NPORT=5 -NVC=3
.module_sw -NAME=SW_4 -NPORT=4 -NVC=2
.module_lnk -NAME=lnk_0_1 -MAP=SW_0,0;SW_1,2 -NREP=3
.module_lnk -NAME=lnk_1_0 -MAP=SW_1,2;SW_0,0 -NREP=3
.module_lnk -NAME=lnk_2_0 -MAP=SW_2,3;SW_0,1 -NREP=2
.module_lnk -NAME=lnk_0_3 -MAP=SW_0,2;SW_3,0 -NREP=5
.module_lnk -NAME=lnk_0_4 -MAP=SW_0,3;SW_4,1 -NREP=2
.module_lnk -NAME=lnk_4_0 -MAP=SW_4,1;SW_0,3 -NREP=2
```

Figure 5. Example input specification

In a nutshell, the ×pipesCompiler generates a set of network component instances which are custom-tailored to the specification contained in its input network description file. Network instantiation follows a two-step procedure: first, the input file is parsed in to an internal data structure, then the structure is traversed and the output is generated. In the following subsections we describe these steps in more detail.

3.1 Input Specification and Parsing

The input file describes the cores, switches, links and the relationships between them. From the designer's viewpoint, the implementation of the NI that connects a core to the NoC is transparent. The ×pipesCompiler will instantiate the needed NIs according to the type of the core (Master, Slave, Master/Slave).

Example 1 *Part of a simple input specification is shown in Figure 5. Attributes of a switch are: name (*-NAME*), number of I/O ports (*-NPORT*) and the number of virtual channels for each output port (*-NVC*). The attributes of a link (*.module_lnk*) are: name (*-NAME*), the name and port number of the source and destination switch for the link (*-MAP*), and the number of repeaters it includes (*-NREP*).*

While parsing the design description file, the ×pipesCompiler dynamically allocates a tree data structure to store the necessary information for each object of the design. Once the first step has been executed and no errors have been detected (like invalid parameters, parameters missing, global parameters not defined), the ×pipesCompiler processes the collected information, as described in the following subsection.

3.2 Network instantiation

The ×pipesCompiler identifies the different types of switches, links and network interfaces that are required in the design. Because all components are written in SystemC,

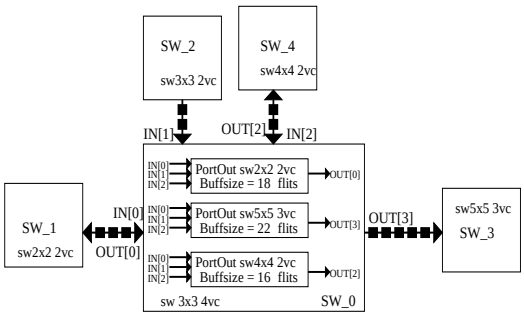
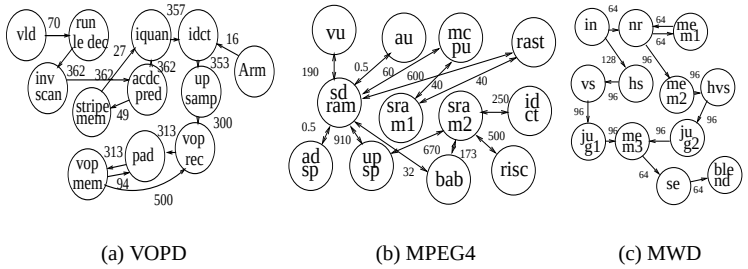


Figure 6. Irregular n/w optimizations



(a) VOPD (b) MPEG4 (c) MWD

Figure 7. Communication pattern of example designs

a single class is created for each type, and multiple objects are instantiated whenever needed.

During network instantiation, the ×`pipesCompiler` performs several optimizations to remove redundant logic from the generic library components. For example, if a switch has only an input link connected to a port, the logic and buffers of the missing output port will not be generated. In the case of a custom-made irregular design, this is a very valuable optimization that drastically reduces hardware complexity. The optimized network component classes are dynamically stored into arrays of structures.

For each object type, a recursive function processes the tree of files from the leaves to the main object file (the root), parsing each file and customizing it according to its type. It is also during this step that the routing tables for each NI are processed and converted into files that are to be included.

The top level (`main.cc`) of the design is then generated. This file instantiates all objects of the design at run-time. It also defines the signals needed to connect the objects according to the design description file. Here again, if possible, the ×`pipesCompiler` will share the signals that are common to all objects. In order to automate tracing of signals, the *debug* command has been implemented, which enables monitoring of any signal in the design.

Example 2 *Referring to Example 1, Figure 6 shows how the network is optimized by removal of redundant logic during instantiation. Because* `port[1]` *of SW_0 has only an input connection the* ×`pipesCompiler` *will not instantiate a* `PortOUT Block` *that would be connected to this output. Similar optimization is performed for an output-only port like* `port[3]`*: in each* `PortOUT Block` *generated, the signals related to this non-existing input are removed from the generic template files. The* ×`pipesCompiler` *will also optimize the size of each output buffer according to the number of repeaters on each output link.*

The time of execution of the ×`pipesCompiler` depends on the design characteristics. For example, a regular topology such as a 16x16 mesh can be generated faster than an application-specific topology with only few cores and switches. But in absolute terms execution time is not a major concern. For instance, for a custom design with 4 switches and 2 cores the execution time is about 2s on a Pentium running at 1.8Ghz. For all our experiments, network generation was completed in few minutes.

4 Experimental Validation and Case Studies

We consider three video processing applications: *Video Object Plane Decoder* (VOPD), *MPEG4 Decoder* (MPEG4) and *Multi-Window Displayer* (MWD), presented in [5, 7], that are mapped onto cores. The communication characteristics of these applications are shown in Figure 7, with the edges annotated with the amount of data transferred between the cores in MB/s. We manually developed customized application-specific topologies that closely match the applications' communication characteristics. For comparison, we also developed a regular mesh NoC for each application. The application designs with different NoC configurations are shown in Figures 8-10.[1]

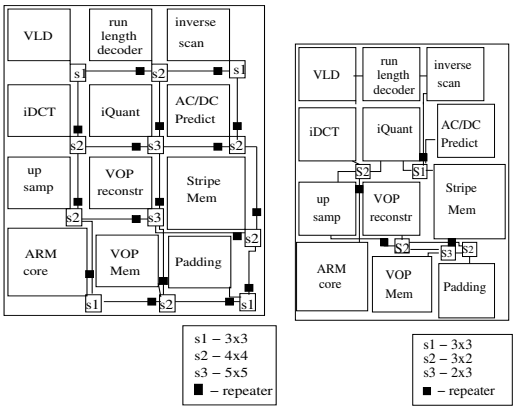
4.1 Application-Specific NoC Characterization

In the VOPD, about half the cores communicate to more than a single core. This motivates the configuration of the custom NoC in Figure 8(b), having less than half the number of switches than a regular mesh NoC. Also these switches are much smaller than the mesh switches. In the MPEG4 design considered, many of the cores communicate with each other through the shared SDRAM. So a large switch is used for connecting the SDRAM with other cores (Figure 9(b)) with smaller switches for other cores. We also consider an alternate custom NoC for MPEG4 (Figure 9(c)) which is an optimized mesh network, with superfluous switches and switch I/Os removed. In the communication pattern of MWD, all but two cores communicate to only one other core. So the custom NoC for MWD (Figure 10(b)) has only two switches.

4.2 Area-Power Analysis

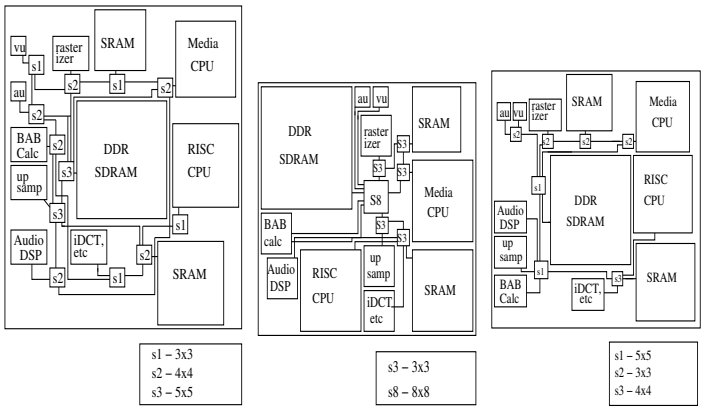
We developed analytical models for estimating the switch areas. The area calculations include the crossbar area, buffer area, logic (including control) area. The models take into account the nuances of individual switch configurations and includes fine granularity of details (like accounting for pipeline registers, cross points, etc).

[1] For clarity, we show link pipelining only in Figure 8 and we assume NI as part of core in the experiments.



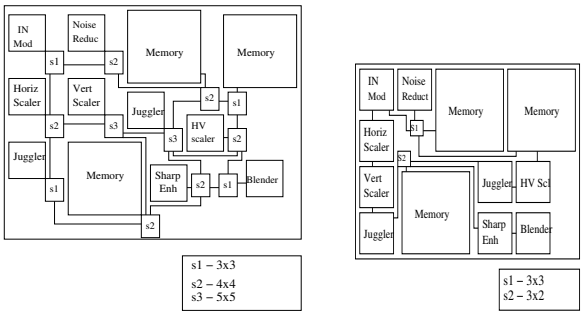
(a) Mesh NoC (b) Appln Specific NoC

Figure 8. Video Object Plane Decoder



(a) Mesh NoC (b) Appln Specific NoC1 (c) Appln Specific NoC2

Figure 9. MPEG4 Decoder



(a) Mesh NoC (b) Appln Specific NoC

Figure 10. Multi-Window Displayer

The area estimates for the various NoC configurations for 0.1μ technology is shown in Table 1. As custom VOPD and MWD NoCs (generated by the $\times$`pipesCompiler`) have relatively small number of switches, we obtain significant area improvement for the custom NoC. But for the MPEG4, as each core communicates to many other cores we have many switches and obtain only $1.69\times$ area improvement with the custom NoCs. We obtain an average of $6.54\times$ area savings for the custom NoCs when compared to the mesh NoC.

We used ORION [16], a power modeling tool, for developing bit energy models for the switches. We use wiring parameters from [17] to estimate link power dissipation. We calculate the power dissipation for each NoC design, based on the average traffic (shown as edge annotations in Figure 7) through each network component for a supply voltage of 1.8 V. The results of the power analysis is summarized in Table 2. For VOPD and MWD we obtain power savings of a factor of three, whereas for the MPEG4 the power savings are smaller. The reason for lower savings in MPEG4 is that most of the traffic traverses the bigger switches connected to the memories. As power dissipation on a switch increases non-linearly with increase in switch size there is more power dissipation in the switches of custom NoC1 of MPEG4 (that has an 8x8 switch) as compared to the mesh NoC. However most of the traffic traverses short links in this custom NoC (Figure 9(b)), thereby giving marginal power savings for the whole design.

Table 1. Area estimates

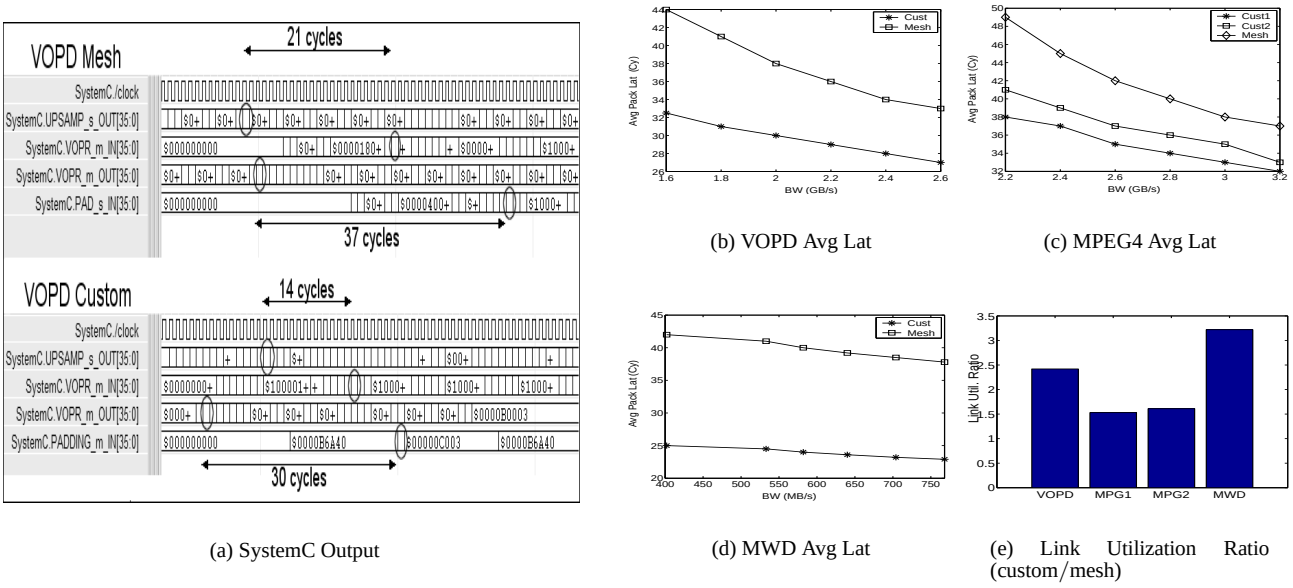
appl	type	area mm^2	rat. mesh/cust
vopd	mesh	1.26	
	cust	0.22	5.73
mpeg4-1	mesh	1.31	
	cust	0.86	1.52
mpeg4-2	mesh	1.31	
	cust	0.71	1.85
mwd	mesh	1.22	
	cust	0.10	12.2

Table 2. Power estimates

appln	topol	pow mW	rat. mesh/cust
vopd	mesh	108.74	
	cust	40.08	2.71
mpeg4-1	mesh	114.36	
	cust	110.66	1.03
mpeg4-2	mesh	114.36	
	cust	93.66	1.22
mwd	mesh	25.9	
	cust	7.72	3.35

4.3 SystemC Simulation Results

We performed cycle-accurate simulation of the SystemC models of the NoCs generated by the $\times$`pipesCompiler`. We used traffic generators to model the bursty nature of the application traffic, with average communication bandwidth matching the applications' average communication bandwidth. Snapshots of SystemC simulations of mesh and custom NoCs for some of the cores of VOPD are shown in Figure 11(a). The time between transmission of a flit and its reception, which includes the switch delay, link delay and contention delay, is marked in the figure. The variation of average packet latency (for 64B packets, 32 bit flits and 7 cycle switch delay) with link bandwidth is



(a) SystemC Output

(b) VOPD Avg Lat

(c) MPEG4 Avg Lat

(d) MWD Avg Lat

(e) Link Utilization Ratio (custom/mesh)

Figure 11. Simulation Results

shown in Figures 11(b)-11(d). application-specific NoCs have lower packet latency as the average number of switch and link traversals is lower. Moreover, the latency increases more rapidly for the mesh NoCs with decrease in bandwidth. With the custom NoCs we achieve an average of 30% savings in latency (measured at the minimum plotted BW value). Also, custom NoCs have better link utilization as seen in Figure 11(e).

5 Conclusions and Future Work

In this paper we have presented ×`pipesCompiler`, a tool that automatically instantiates application-specific NoCs. The tool bridges the design gap in building application-specific NoCs that optimally match the communication requirements of the system. The network components built are highly optimized for the particular NoC design, providing large savings in area, power and latency for example designs. In the future we plan to enhance the tool with automatic selection of network topology, thus providing a complete design flow for heterogeneous NoCs.

6 Acknowledgements

This research is supported by MARCO Gigascale Systems Research Center (GSRC) and NSF (under contract CCR-0305718).

References

[1] L.Benini, G.D.Micheli, "Networks on Chips: A New SoC Paradigm", IEEE Computers, pp. 70-78, Jan. 2002.

[2] A.Jantsch, H.Tenhunen, "Networks on Chip", Kluwer Academic Publishers, 2003.

[3] E.Rijpkema et al., "Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip",DATE 2003, pp. 350-355, Mar 2003.

[4] W.Cesario et al., "Component-Based Design Approach for Multi-Core SoCs", DAC 2002, pp.789-794, June, 2002.

[5] E.B.Van der Tol, E.G.T.Jaspers,"Mapping of MPEG-4 Decoding on a Flexible Architecture Platform", SPIE 2002, pp. 1-13, Jan, 2002.

[6] M.Dallosso et. al, "×`pipes`: a Latency Insensitive Parameterized Network-on-chip Architecture For Multi-Processor SoCs", pp. 536-539, ICCD 2003.

[7] E.G.T.Jaspers, et al.,"Chip-set for Video Display of Multimedia Information", IEEE Trans. on Consumer Electronics, Vol 45, No. 3, pp. 707-716, Aug, 1999.

[8] F.Karim et al., "An Interconnect Architecture for Network Systems on Chips", IEEE Micro, Vol.22, No.5, pp.36-45, Sep. 2002.

[9] L.P.Carloni, K.L.McMillan, A.L.Sangiovanni-Vincentelli, "Theory of latency-insensitive design", IEEE Trans. on CAD of ICs and Systems, pp.1059–1076, Vol.20, no.9, Sept. 2001.

[10] P.Guerrier, A.Greiner,"A generic architecture for on-chip packet switched interconnections", Proc. DATE, pp. 250-256, March 2000.

[11] S.Kumar et al., "A network on chip architecture and design methodology",ISVLSI 2002, pp.105–112, 2002.

[12] S.J.Lee et al.," An 800MHz Star-Connected On-Chip Network for Application to Systems on a Chip", ISSCC 2003, Feb. 2003.

[13] J.Hu, R.Marculescu,"Energy-Aware Mapping for Tile-based NOC Architectures Under Performance Constraints", ASP-DAC 2003.

[14] H.Yamauchi et al., "A 0.8 W HDTV video processor with simultaneous decoding of two MPEG2 MP@HL streams and capable of 30 frames/s reverse playback", ISSCC, Vol.1, pp. 473-474, Feb. 2002

[15] H.Zhang et al., "A 1V Heterogeneous Reconfigurable DSP IC for Wireless Baseband Digital Signal Processing", IEEE Journal of SSC, pp.1697–1704, Vol.35, no.11, Nov. 2000.

[16] H.S Wang et al., "Orion: A Power-Performance Simulator for Interconnection Networks", MICRO, Nov. 2002.

[17] R. Ho, K. Mai, and M. Horowitz, "The Future of Wires", Proceedings of the IEEE, pp. 490-504, April 2001.

[18] http://www.ocpip.org/home OCP specification