The First NPoC Design

Henrique Cota de Freitas, Philippe Olivier Alexandre Navaux
Parallel and Distributed Processing Group
Graduate Program in Computer Science, Informatics Institute
Universidade Federal do Rio Grande do Sul, Brazil
{hcfreitas, navaux}@inf.ufrgs.br

Abstract

Nowadays, multi-core and many-core processors have increased the number of research works related to on-chip interconnections. In this case, a new approach called Network-on-Chip (NoC) has shown opportunities in order to increase the performance of the next generation of many-core processors. This paper presents the first design of a Network Processor on Chip (NPoC) for NoC routers capable of improving the flexibility and the performance for a large number of communication patterns. Conclusions describe other results that point out to modifications in this proposal.

1. Introduction

This section presents two concepts related to on-chip interconnection [1][2][3]:

- Communication topologies.
- Communication patterns.

Besides traffic congestion on network, node connectivity (number of links) and diameter (the largest distance between two nodes) are features to decide the best route for transmitting packets. However, a long packet path (several routers) can increase the final transmission latency, since there are several hops (routers) at the path. Adapting topologies to communication patterns can be the best alternative to reduce this long path.

Therefore, an alternative NoC (Network-on-Chip) [4][5] should implement or adapt interconnections in accordance with specific algorithm classes based on parallel programming techniques.

Figure 1 presents some topology examples used in several multicomputers and multiprocessors (based on NoCs). Each topology can be related to one algorithm class or parallel programming technique as follows:

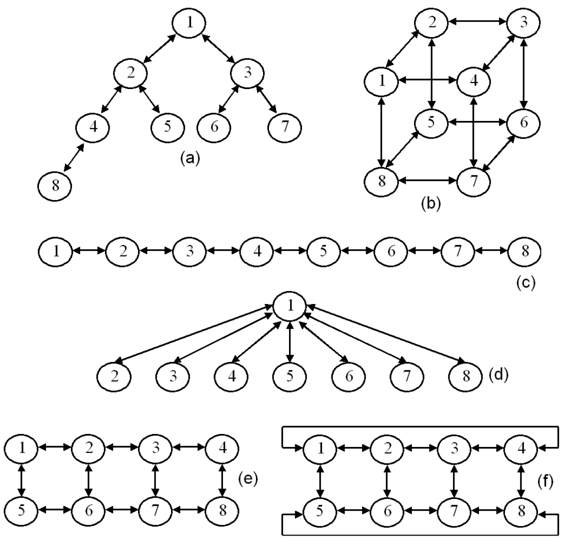

Figure 1. Topology examples: (a) tree, (b) hypercube, (c) pipeline, (d) star, (e) mesh and (f) torus

Figure 1.a: The balanced tree topology favors the execution of algorithms based on divide and conquer technique, image processing, dataflow and reduction programming.

Figure 1.b: The hypercube topology has cores arranged in n-dimensional cube. Main applications are the following: 3D scientific programs.

Figure 1.c: Linear array or pipeline topology has cores grouped in stages, where the next stage receives data from the previous stage. Each stage had one or multiple cores and some Network Processors (NPs) are examples. The NP's cores process packets in stages, and each core is responsible for executing specific threads.

Figure 1.d: The star topology is normally related to the master / slave programming technique. One node or core is the master and the remaining nodes are slaves. This is the one parallel programming technique very common.

Figure 1.e: The mesh topology architecture has nodes connected to theirs neighbors. The mesh is interesting to solve problems with bidimensional data structure, matrix operations, image processing and differential equations. The tridimensional mesh topology is used in time prediction, particle simulation and aerodynamic, for example.

Figure 1.f: Bidimensional torus topology is a variant of bidimensional mesh. The difference is based on the border nodes that are connected to opposite border nodes in the same row and column. The torus topology can be used to solve the same problems described for mesh topology.

The main goal of this paper is to present the first NPoC architecture in order to show the importance of this design in the next results of PhD thesis that focus on adaptable topologies capable of improving the performance of communication patterns.

2. NPoC Design and Verification

NPoC [6] is a RISC (Reduced Instruction Set Computing) processor with a scalar pipeline based on five stages (Figure 2). A modification regarding typical pipelines is presented in the fourth stage where the NPoC accesses input buffers and a Reconfigurable Crossbar Switch (RCS) [7]. All stages are executed in one cycle each one, the instructions are regular, and there are not floating-point instructions, since the workloads for this network processor is based on fixed-point instructions.

The four traditional pipeline stages presented in Figure 3 are the following: Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), and Write Back (WB). However, in the fourth stage, besides memory access, NPoC can access input buffers and RCS through specific instructions. The fourth stage can be described as the following:

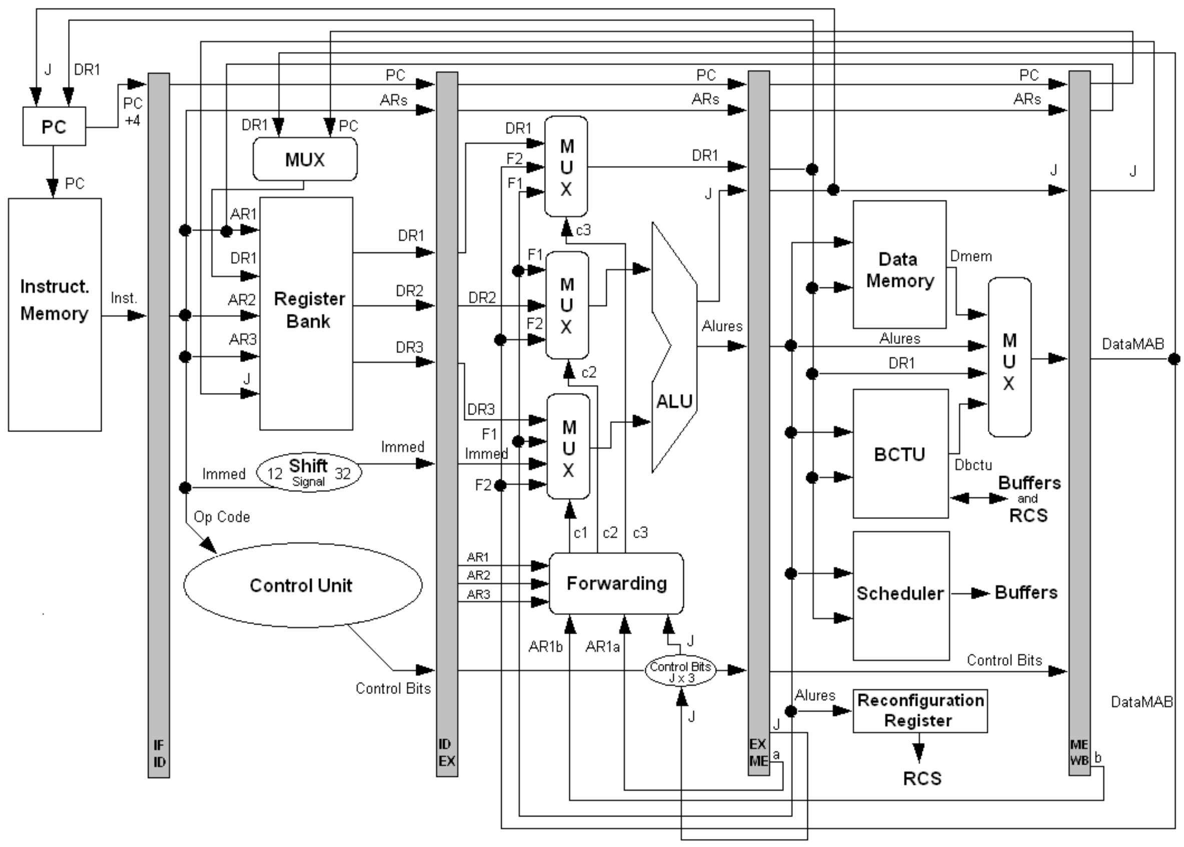

Figure 2. NPoC Architecture [6]

- MEM: Memory access (*load*/*store* instructions).
- BCTU: Buffer and Crossbar Transfer Unit. Instructions access buffer and crossbar switch through this unit.
- SCH: Scheduler. Instructions access the scheduler to manage the packet traffic.
- REC: Reconfiguration register. Instructions access the reconfiguration register to transfer data for a new topology implementation on RCS.

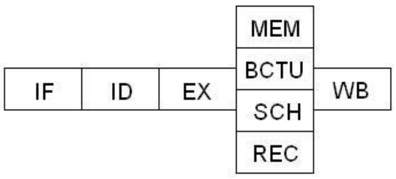

Figure 3. Pipeline stages [6]

Table 1 presents some general-purpose instructions used to elaborate a program for NPoC. They are typical instructions that use the arithmetic and logical unit (ALU), or data memory. All instructions, including network instructions, depend on ALU to finish the execution.

Table 1. General-purpose instructions [6]

Instruction	Description
add r1, r2, r3	r1 = r2 + r3
mul r1, r2, r3	r1 = r2 x r3
ori r1, r2, immed	r1 = r2 + immed
not r1, r2	r1 = r2’
load r1, r2, immed	r1 = (Address[r2 + immed]) content
store r1, r2, immed	Address[r2 + immed] = r1
jump r1, r2, immed	PC = r2 + immed, r1 = PC
jeq r1, r2, r3	PC = r1, se r2 = r3

Table 2 presents the NPoC’s network instructions. The instructions *read* and *write* access the BCTU in order to read or modify data from packets. Instructions *send*, *block* and *erase* access the scheduler to alter the packet status: ready to send, to block a packet while NPoC waits some event, and to erase some packet. The *reconf* instruction transfers data word to reconfiguration register.

Table 2. Network instructions [6]

Instruction	Description
read r1, r2, immed	r1 = (Address[r2 + immed]) content
write r1, r2, immed	Address[r2 + immed] = r1
send r1, r2, immed	Buffer (r1), packet (r2 + immed)
block r1, r2, immed	Buffer (r1), packet (r2 + immed)
erase r1, r2, immed	Buffer (r1), packet (r2 + immed)
reconf r1	Reconfiguration register = r1

NPoC processor was simulated and verified through ArchC [8] and SystemC [9] tools. The main function of NPoC is to manage RCS topologies in order to increase the performance of workloads based on communication patterns. Hence, topology examples presented in Figure 1 are shown in Figure 4 as topologies implemented onto RCS and managed by NPoC.

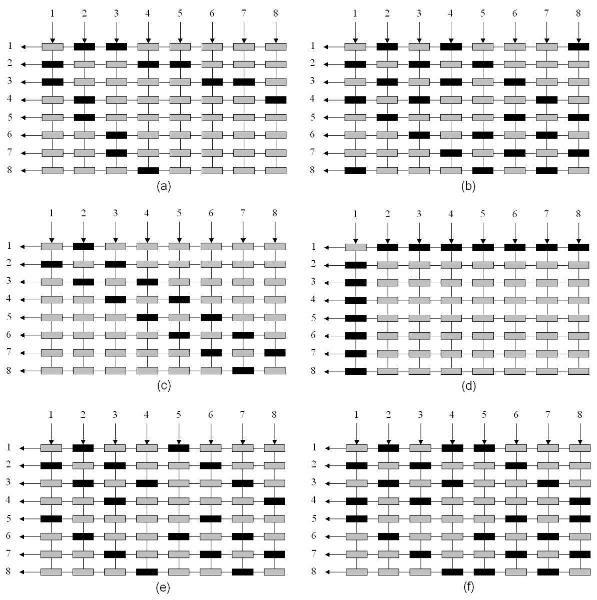

Figure 4. Mapped topologies: (a) tree, (b) hypercube, (c) pipeline, (d) star, (e) mesh and (f) torus

Figure 5 presents the algorithm elaborated to manage the topology reconfigurations. This is a simple program that checks the moment for a new topology implementation. Figure 5.a shows the high-level algorithm that verifies communication status of each input buffer. After the last sent packet, the algorithm sends new topology information (*reconfigure* instruction) to support the next demanded communication pattern.

Program description and register initial constants are presented in Figure 5.b. The assembly program for NPoC is described in Figure 5.c. The first part (before L1) represents initial constants. The second part of the program manages the topologies. Simulation results verified that this operation demands 17 cycles of clock after the last packet.

Therefore, to implement a new topology is necessary 17 cycles of NPoC program and 2 cycles to reconfigure switching nodes onto RCS. The frequency used by ArchC is 50MHz; so in order to implement a new topology it is necessary 19 cycles or 0.38µs.

```
while (1)
{
 i = 1;
 counter = 1;
 while (i < 9)
   read (reg_buffer[i], data);
   if data == 1
     counter = counter + 1;
     write (reg_buffer[i], 0);
     if counter == 8
       counter = 1;
       reconfigure topology;
       number_of_topologies ++;
       if number_of_topologies == 6
         exit;
   i = i + 1;
 }
}
```

(a)

```
$1 = 9 (constant)
$2 = i (buffer number)
$3 = 1 (constant)
$4 = counter
$5 = data
$6 = 0 (constant)
$7 = 8 (constant)
$8 = 6 (topology / constant)
$10 = label 1
$11 = label 2
$12 = topology address
$13 = number of topologies
$14 = label 4
```

(b)

```
    addi $1, $0, 9
    addi $3, $0, 1
    addi $6, $0, 0
    addi $7, $0, 8
    addi $8, $0, 6
    addi $10, $0, L1
    addi $11, $0, L2
    addi $12, $0, 100
    addi $13, $0, 106
    addi $14, $0, L4

L1: load $8, $12, 0
    addi $2, $0, 1
    addi $4, $0, 1

L3: jeq $10, $1, $2
    read $5, $2, 0
    jdi $11, $5, $3
    addi $4, $4, 1
    write $6, $2, 0
    jdi $11, $4, $7
    addi $4, $0, 1
    reconf $8
    addi $12, $12, 1
    jeq $14, $12, $13
L2: addi $2, $2, 1
    addi $13, $13, 1
    jump $9, $0, L3
L4: ......
```

(c)

Figure 5. Algorithm example: (a) high-level algorithm, (b) description, (c) NPoC assembly

In order to perform broadcast communication on buffered RCS, the total transmission time to send one thousand 4096 bits packets for each topology presented in Figure 1 is 2.56ms. Therefore, the latency to manage and implement new topology (0.38µs) is very low. It represents an overhead of 0.014%.

It is important to notice that this program changes the topology only after the last communication. However, it is possible to implement hybrid topologies if a new communication pattern demands a new interconnection before last communication between processing cores. This total latency considering hybrid topologies on demand was not evaluated, but probably, the performance will be higher than first one simulation.

3. Conclusions

This paper described the first NPoC design and verification. Simulation results show that performance overhead in order to manage the RCS is very low (0.014%.).

However, other results of PhD studies [10][11] point out to modification in the NPoC architecture. The main modification is related to area occupation. In this case, the number of pipeline stages will be reduced and the instruction format will be different in order to increase the performance and compatibility with other processors.

Besides, other future works are the following:

- New ArchC-based description.
- ArchC-based full programmable NoC.
- Performance evaluation based on parallel workloads.

References

[1] J. Duato, S. Yalamanchili, L. Ni, “Interconnection Networks”, Morgan Kaufmann, 2002.

[2] J. Dongarra, et al., “The Sourcebook of Parallel Computing”, Morgan Kaufmann, 2003.

[3] Duncan, R., “A Survey of Parallel Computer Architectures”, IEEE Survey & Tutorial Series, pp.5-16, February 1990.

[4] T. Bjerregaard and S. Mahadevan, “A Survey of Research and Practices of Network-on-Chip”, ACM Computing Surveys, Vol. 38, No 1, pp. 1-51, March 2006.

[5] L. Benini and G. D. Micheli, “Network-on-Chip Architectures and Design Methods”, IEE Computers & Digital Techniques, Vol. 152, Issue 2, pp. 261-272, 2005.

[6] H. C. Freitas, et al., "Projeto de um Processador de Rede Intra-Chip para Controle de Comunicação entre Múltiplos Cores", VII Workshop em Sistemas Computacionais de Alto Desempenho, WSCAD 2006, (held in conjunction with the 18th International Symposium on Computer Architecture and High Performance Computing), Ouro Preto, MG, Brazil, pp.3-10, 2006 (in Portuguese).

[7] H. C. Freitas, et al., “Reconfigurable Crossbar Switch Architecture for Network Processors”, IEEE International Symposium on Circuits and Systems, pp.4042-4045, 2006.

[8] A. Ghosh, et al., “System modeling with SystemC”, International Conference on ASIC, pp.18-20, 2001.

[9] S. Rigo, et al., “ArchC: A SystemC-Based Architecture Description Language”, International Symposium on Computer Architecture and High Performance Processing, pp.66-73, October 2004.

[10] H. C. Freitas., T. G. S. Santos and P. O. A. Navaux, “Design of Programmable NoC Router Architecture on FPGA for Multi-Cluster NoCs”, IET Electronics Letters, Vol. 44, No. 16, pp. 969-971, 31st July 2008.

[11] H. C. Freitas, T. G. S. Santos and P. O. A. Navaux, “NoC Architecture Design for Multi-Cluster Chips”, IEEE International Conference on Field Programmable Logic and Applications, in press, 2008.