Chronos: an Abstract NoC-based Manycore with Preserved Temporal and Spatial Traffic Distribution

Geaninne Lopes, Iaçanã Weber, César Marcon, Fernando Gehm Moraes
PUCRS – School of Technology - Av. Ipiranga 6681, Porto Alegre, Brazil
{geaninne.lopes, iacana.weber}@edu.pucrs.br, {cesar.marcon, fernando.moraes}@pucrs.br

***Abstract*—The time spent to assess the application performance through clock-cycle simulators is a bottleneck of an NoC-based manycore design; thus, requiring higher abstraction levels at early design stages. However, high-level synchronization of processing and communication in such systems is a challenge. This work develops and validates *Chronos*, an untimed abstraction of an NoC-based manycore, built with Open Virtual Platform, that seeks precise traffic modeling in such a way to preserve the temporal and spatial distributions of the physical implementation. Results show the similarity of the temporal and spatial traffic distributions compared to a reference RTL-level platform.**

***Index Terms*—NoC-based manycores, abstract modeling, traffic evaluation, OVP**

I. Introduction and Related Work

In the last decade, several works [1]–[8] proposed Virtual Platforms (VPs) to model systems at a higher abstraction level. The VP approach aims to accomplish two goals: (*i*) to estimate performance, area, and energy to the designers in the early stages of the project; (*ii*) to enable software and hardware development in parallel, reducing the time-to-market [9].

Models targeting performance estimation for early design space exploration are tuned based on data obtained from the physical level [10]. According to design constraints, high-level synthesis tools can generate Pareto curves related to the best configuration for processing, communication, and storage.

Lemaire et al. [2] introduced a flexible modeling environment built around a SystemC-TLM kernel to explore some hardware features, demonstrating its potential for silicon implementation validation and application mapping at early design stages. Helmstetter et al. [3] proposed an MPSoC VP to evaluate the hardware costs of a specific architecture, reaching guidelines to exploit these higher dynamicity protocols according to application needs. MPSoCBench is a VP for heterogeneous MPSoC modeling introduced by Duenha et al. [4] that provides energy consumption estimation. Lima et al. [7] proposed a VP based on multiagent systems to compare and select routing traffic algorithms.

Software development models seek simulation speed and not necessarily performance estimation. Madalozzo et al. [5] proposed a VP that combines different architecture description languages and simulators to improve software productivity in manycore systems, providing fast software validation and debuggability. Cataldo et al. [6] built an abstract platform on GEM5 to model NoC-based MPSoCs and evaluate synchronization mechanisms of parallel applications.

Using binary code translation techniques, tools such as OVPSim [9] allow the entire simulation of a software stack, including an operating system and user applications. These models do not require a detailed hardware description, requiring only an instruction set simulator and peripheral modeling support; the number of instructions executed during the simulation usually provides an acceptable performance estimation.

The knowledge of the traffic behavior enables, for example, to identify hotspots [11] and detect anomalies that may signalize attacks [12]. However, the traffic behavior in NoC-based manycore systems, including its temporal and spatial distribution, is an abstract modeling gap. This work aims to fill this gap allowing the software development on *Chronos*, an abstract model of an NoC-based manycore, which provides high-speed simulation to observe long-time traffic series.

II. OVP Background

OVP allows accurate temporal modeling by counting instructions executed by the processor. The execution time is computed by dividing the number of instructions executed in all simulation steps by the nominal processor speed in MIPS (Million Instructions per Second). The simulation step is called *quantum*, which is the number of instructions each processor executes in each turn [9]. The user in the OVP simulator can specify the number of instructions per quantum q. For example, the $P0$ processor starts executing q instructions, followed by $P1$ that executes the same number of instructions, and so on, until all processors have executed q instructions. Only after that, the simulation time advances, and the next quantum executes.

The quantum only covers instructions executed on processors. Peripherals may execute in two ways, out of the quantum: (*i*) before processors start their executions by using OVP API functions; (*ii*) atomic execution triggered by callbacks, such as when a processor write or read in a memory-mapped register.

Figure 1 exemplifies a sequence diagram with the execution of two peripherals (*Periph0* and *Router*) and two processors (*proc0* and *proc1*) on the platform. First, peripherals are sequentially initialized by connecting packetnet ports (connection between peripherals) and registers through the constructor function inserted in the main file. Then, the quantum of the processors starts running *proc0* that sends the message "data" to *proc1* by writing it in a memory-mapped register, which links *proc0* with Router. Next, the memory-register callback is called. The callback sends a message to *Periph0* notifying that there is a flit to be transmitted, and sends data to *proc1*. During the quantum 1, *proc1* receives this data, and after completing quantum 1, *Periph0* calls the *printDebug()* function that shows the number of transmissions in that quantum.

OVP allows the parallel execution of processor code through the quantum approach, but the execution of peripherals is

978-1-7281-7670-3/21/$31.00 ©2021 IEEE

2021 IEEE 12th Latin America Symposium on Circuits and System (LASCAS) | 978-1-7281-7670-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/LASCAS51355.2021.9459124

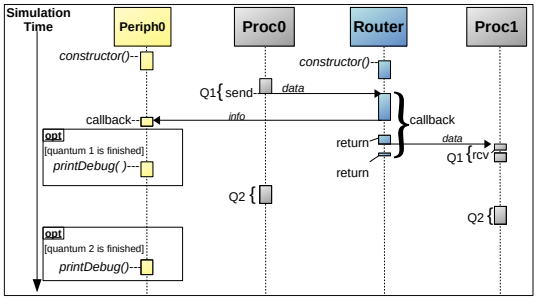


Fig. 1. Execution model in OVP, showing the communication among processors and peripherals.

atomic. Therefore, modeling NoC as a peripheral is a challenge due to the absence of mechanisms that allow NoC and processors to execute in parallel.

III. NoC-based Manycore Model

The *Chronos* behavior resembles a clock cycle-accurate NoC-based manycore encompassing a matrix of Processing Elements (PEs), each one containing a processor connected through a Network Interface (NI) to a 2D-mesh NoC with XY routing, round-robin arbitration, and input buffering. NI is responsible for exchanging packets between the NoC and local memory. Packet reception and transmission are performed in parallel to avoid deadlocks. For packet transmission, NI manages a control flow algorithm that checks whether NoC has a channel available for packet injection. NI interrupts the processor for packet reception using a packet reading routine implemented at the Operating System (OS) level.

Figure 2 displays an instance of *Chronos* that preserves the same routing and arbitration features of the RTL NoC-based manycore to generate similar traffic distributions.

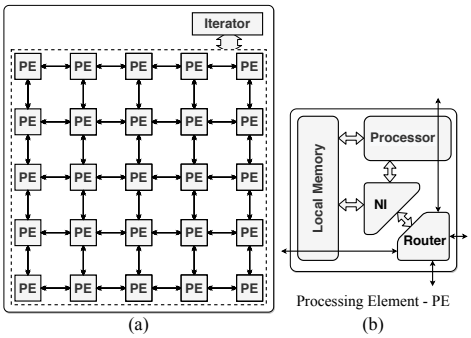


Fig. 2. (a) instance of *Chronos*, (b) main components of each PE. $Iterator$ is a component used by the abstract model.

The OVP simulation tools provide the modeling of processors and local memories. Thus, the modeling of these components is out of the scope of this work.

IV. NI and NoC Models

Before describing the NI and NoC models, it is necessary to understand the underlying problems related to the packet synchronization in a quantum-based temporal model, and the way callback functions operate:

(i) Consider a set of PEs in a 3x3 mesh NoC. If PE0 and PE8 send packets to PE7 in the same quantum, only the packet data produced by PE0 will be consumed, since PE8 runs after PE7. Consequently, only in the next quantum PE7 will consume the packet data produced by PE8. This problem leads to the packet transmission indeterminism since the quantum model orders parallel events according to the PE placement in the NoC.

(ii) Suppose PE0 and PE6 send data to PE7 in the same quantum. The callback from PE0 remains "open" until PE7 consumes the packet (see Figure 1 - the producer "opens" the callback, while the consumer "closes" it). Thus, even if PE6 is executed before PE7, there will be no consumption by PE6 due to the callback behavior.

Due to these problems, *Chronos* requires the $Iterator$ component (Figure 2) for activating the packet transmissions. The $iterator$ is a virtual component responsible for triggering each router at the end of a quantum. It evaluates all routers sequentially with flits to transmit, sending one flit to the next router or local port. This process is similar to the RTL model behavior.

A. Network Interface Modeling

NI performs the communication between processor and NoC and implements a Direct Memory Access (DMA). Figure 3 describes the NI environment, ports, and connected modules. The communication API provides functions to send and receive packets. NI uses these functions to write in memory-mapped registers the packet address to be sent or received, releasing the processor during the communication with the NoC. The following ports access these registers:

- *address*: NI configure the address register with the address of incoming packets at the processor startup. During the application execution, NI uses this register to store the packet address to be transmitted to another PE;
- *statusTX*: the processor reads a status register to verify the local port availability. If the read value is zero, the processor may start a packet transmission;
- *statusRX*: the processor notifies the NI a complete packet reception.

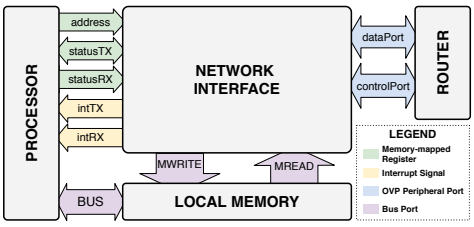


Fig. 3. The NI environment describing ports and connected modules.

B. Packet Injection

Two actions occur when a processor writes in *statusTX*: (i) NI sends to the local port buffer the first flit of the packet; (ii) the router notifies the $iterator$ that it has data to transmit. These actions characterize a non-blocking transmission; i.e., if the router local port is available, the processor programs the NI to transmit the packet at the end of the quantum and continues the application execution.

When the quantum ends, the *iterator* triggers routers to transmit flits according to the control flow signals (flits may be blocked due to an NoC congestion). Sequentially, the *iterator* searches for routers with flits to transmit; for each positive case, the *iterator* performs a communication step transmitting one flit to the next router respecting the arbitration and routing policies. The transmission stops when flits cannot advance due to congestion, or the packets were delivered to their targets. If all flits of a packet were transmitted, NI rises *intTX*, notifying a complete packet transmission. A new quantum period starts at the end of a transmission round. Processors with *intTX* asserted execute the interruption handler, releasing the *statusTX* register for another transmission.

Figure 4 exemplifies a 3x3 mesh NoC with flows $6 \rightarrow 7$, $0 \rightarrow 7$, $2 \rightarrow 4$ to understand that the abstract communication method implements the expected NoC behavior. NI7 receives the packet from R6 and blocks the R7 local port. Flow $0 \rightarrow 7$ stops at the R7 south buffer and flow $2 \rightarrow 4$ blocks at the R1 east port. Only in the next quantum, NI7 receives flow $0 \rightarrow 7$, releasing flow $2 \rightarrow 4$.

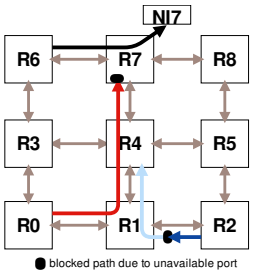


Fig. 4. Example of flows being transmitted during a simulation quantum.

C. Packet Reception

The execution of a *receive()* call blocks the processor up to the end of the quantum. The packet reception occurs only at the next quantum, if available. The NI handles the packet reception during the *iterator* execution. If NI receives an incoming flit (condition defined by the control flow signals), it writes the flit into the memory address. Upon the reception of a complete packet, NI rises the processor *intRX* signal.

The processors execute the interruption handler at the beginning of the quantum. If *intRx* is active, the processor copies the packet contents to the address defined in the *receive()* call. At the end of the packet reception the processor writes in the *statusRX* signal, enabling NI to receive a new packet.

Note that each processor may transmit and receive one packet per quantum. This model adopts a small quantum value (250 to 1,000 instructions) to avoid long simulation periods with processors waiting for communication data.

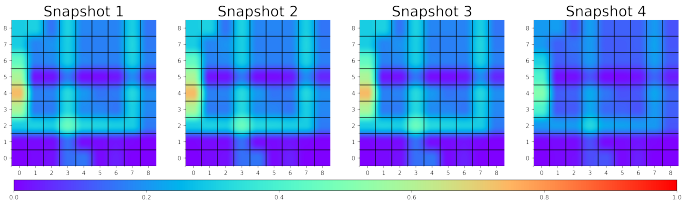
V. Results

Experiments evaluate the spatial and temporal traffic distribution aiming to compare the RTL NoC-based Manycore implementation to *Chronos*.

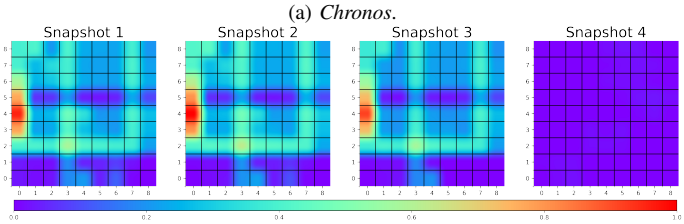
A. Spatial Traffic Distribution

The first experiment evaluates the spatial traffic distribution on both platforms. The simulation annotates the moment that each flit is transmitted in all ports of the routers. A script computes the sum of all flits from all NoC ports at simulation intervals; a snapshot represents each interval. The total simulation time considers the number of clock cycles or quantums for the RTL platform or *Chronos*, respectively. The traffic volume is also normalized according to the largest number of flits transmitted by a given router during the scenario simulation.

Figure 5 presents four intervals of the traffic distribution during the simulation of a 9x9 manycore, executing 12 applications, partitioned into 65 tasks. The communication volume in the RTL simulation is higher in the first snapshots, while the *Chronos* is spread along with the execution time. This difference is due to the timing models. Besides this difference, it is noticeable a similar spatial traffic distribution, with the same congested areas.



(a) *Chronos*.



(b) RTL model.

Fig. 5. Evaluation of the spatial traffic distribution.

This experiment shows the similarity of spatial traffic distribution between *Chronos* and RTL platform. This result shows the feasibility of adopting *Chronos* to evaluate mapping heuristics [13], routing algorithms, congestion effects due to the traffic flows, energy consumption in the communication architecture.

B. Temporal Traffic Distribution

The second experiment evaluates the temporal traffic similarity between the RTL platform and *Chronos* when executing a hotspot application on a 5x5 system and with the same task mapping. The simulations report the number of flits traversing each router port at each 10,000 clock cycles (RTL platform) and 1,000 instructions (one *Chronos* quantum).

Figure 6(a) shows the number of flits captured at the router 16 north port (the hotspot target PE). As expected, this figure displays that there is no timing relation between the RTL platform and *Chronos*. The difference comes from the packet communication modeled in *Chronos*, allowing only one packet reception/transmission per quantum; i.e., if a processor requests a packet, it will be delivered in the next quantum, stalling the processor.

Figure 6(b) groups quantums according to the relation $\frac{number\ of\ quantums}{RTL\ samples'\ number}$ into sets. As one can see, the bursts of flits are not aligned on the X-axis due to the timing models.

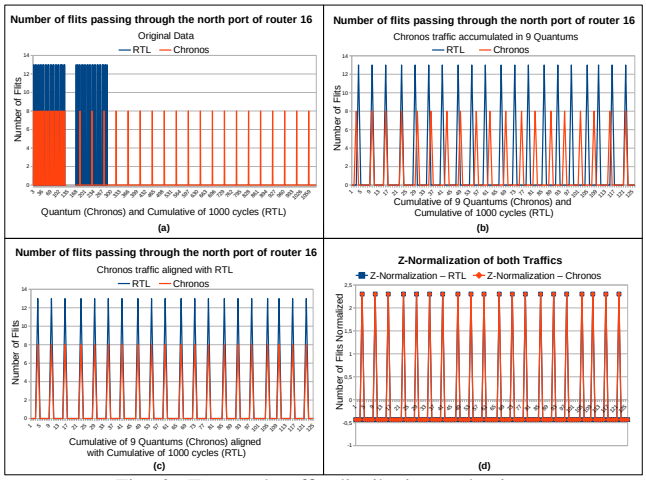


Fig. 6. Temporal traffic distribution evaluation.

Aiming to evaluate the *Chronos* precision, instead of the accuracy compared to the RTL platform, we employed the Dynamic Time Warping (DTW) algorithm [14], which aligns two temporal series. Figure 6(c) shows both packet series perfectly aligned in time. However, there is a difference in the number of transmitted flits due to two reasons: (i) the RTL platform adds control flits in the packets, while *Chronos* does not include them; (ii) the RTL platform has a manager processor that communicates with control packets with other tasks, these control packets are absent in *Chronos*.

This work adopts the Euclidean distance (ED) between series to quantify the traffic similarity on both platforms. According to [15], it is necessary to normalize the temporal series due to different offsets and amplitudes. Normalization consists of translating and resizing the series so that all have zero mean and unit standard deviation. The normalization method used in this work was z-normalization, presented in Equation (1), where *z[i]* is the value that replaces the *u[i]* element of the U series, and m and s are the mean and standard deviation of U, respectively. After normalization, the ED (Equation (2)) is used to measure the similarity degree of the series. Figure 6(d) presents the normalized data, which coincides 100%, making ED equals to 0. The closer to 0, the greater the similarity between the two series.

$$z[i] = \frac{u[i] - m}{s} \tag{1}$$

$$d_{Euc}(U,V) = \sum_{i=1}^{n}((u[i] - v[i])^2)^{\frac{1}{2}} \tag{2}$$

In this experiment, 38.5% of the router ports with application traffic presented $ED = 0$; 17.14% of the routers obtained $ED = 3.86$ due to a burst at the beginning of the simulation, when the manager processor of the RTL platform sends control packets to all other processors. The maximum, average, and median ED were 13.86, 4.64, and 3.86, respectively. It was also observed that ports without traffic from the manager processor obtained a shorter distance than the others.

This result demonstrates that *Chronos* generates similar temporal traffic distribution than the RTL platform because the NoC behavior (routing algorithm and arbitration policies) is the same in both platforms. This result paves the way to use *Chronos* in algorithms requiring long time series, such as security (anomaly detection) and aging mitigation (temperature estimation).

VI. Conclusion

This article presented *Chronos*, an abstraction of an NoC-based manycore implemented in OVP. *Chronos* provides a novel mechanism for synchronizing processing and NoC communication flows, resulting in spatial and temporal behavior similar to clock-cycle accurate implementations. The results showed that the quantum granularity does not allow *Chronos* to have higher temporal accuracy than a reference platform with clock-cycle precision. However, the traffic distribution maps and the similarity of temporal traffic distributions demonstrated that the proposed platform has a high precision degree to reach high-quality volumetric and temporal application execution comparisons. Future works move towards the *Chronos* usage related to research methods that require the manycore platform simulation for long periods, as dynamic thermal management techniques and intrusion detection.

Acknowledgment

The authors would like to thank Imperas Software and Open Virtual Platforms for their support and access to their models and simulator. This work was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) – Finance Code 001, and CNPq.

References

[1] R. Leupers *et al.*, "Virtual Manycore platforms: Moving towards 100+ processor cores," in *DATE*, 2011, pp. 715–720.
[2] R. Lemaire, S. Thuries, and F. Heitzmann, "A Flexible Modeling Environment for a NoC-based Multicore Architecture," in *HLDVT*, 2012, pp. 140–147.
[3] C. Helmstetter *et al.*, "A Dynamic Stream Link for Efficient Data Flow Control in NoC Based Heterogeneous MPSoC," in *ASP-DAC*, 2013, pp. 41–46.
[4] L. Duenha *et al.*, "MPSoCBench: A toolset for MPSoC system level evaluation," in *SAMOS*, 2014, pp. 164–171.
[5] G. Madalozzo, M. Mandelli, L. Ost, and F. Moraes, "A platform-based design framework to boost many-core software development," in *ICECS*, 2015, pp. 320–323.
[6] R. Cataldo, R. Fernandes, K. Martin, J. Sepúlveda, A. Susin, C. Marcon, and J. Diguet, "Subutai: distributed synchronization primitives in NoC interfaces for legacy parallel-applications," in *DAC*, 2018, pp. 83:1–83:6.
[7] G. Lima *et al.*, "Exploring MAS to a High Level Abstraction NoC Simulation Environment," in *ICECS*, 2018, pp. 365–368.
[8] Y. Qureshi, W. Simon, M. Zapater, D. Atienza, and K. Olcoz, "Gem5-X: A Gem5-Based System Level Simulation Framework to Optimize Many-Core Platforms," in *HPC*, 2019, pp. 1–12.
[9] Imperas, "Open Virtual Platforms - OVP," 2019. [Online]. Available: http://www.ovpworld.org/
[10] C. Lo and P. Chow, "Multi-fidelity Optimization for High-Level Synthesis Directives," in *FPL*, 2018, pp. 272–279.
[11] M. Reza, D. Zhao, and M. Bayoumi, "Power- Thermal Aware Balanced Task-Resource Co-Allocation in Heterogeneous Many CPU-GPU Cores NoC in Dark Silicon Era," in *SOCC*, 2018, pp. 260–265.
[12] G. Fernandes, J. Rodrigues, L. Carvalho, J. Al-Muhtadi, and M. Proença, "A Comprehensive Survey on Network Anomaly Detection," *Telecommunication Systems*, vol. 70, no. 3, pp. 447–489, 2019.
[13] E. Carvalho, C. A. M. Marcon, N. Calazans, and F. Moraes, "Evaluation of Static and Dynamic Task Mapping Algorithms in NoC-based MPSoCs," in *SOC*, 2009, pp. 87–90.
[14] D. Diab *et al.*, "Anomaly Detection Using Dynamic Time Warping," in *CSE and EUC*, 2019, pp. 193–198.
[15] J. Lin, R. Khade, and Y. Li, "Rotation-invariant similarity in time series using bag-of-patterns representation," *Journal of Intelligent Information Systems*, vol. 39, no. 2, pp. 287–315, 2012.