# VisualNoC: A Visualization and Evaluation Environment for Simulation and Mapping

Junshi Wang[1,2], Yang Huang[1], Masoumeh Ebrahimi[3], Letian Huang[1], Qiang Li[1], Axel Jantsch[2], Guangjun Li[1]
[1]University of Electronic Science and Technology of China, Chengdu, China, 611731
[2]TU Wien, Vienna, Austria, 1040
[3]Royal Institute of Technology, Stockholm, Sweden; University of Turku, Turku, Finland
Corresponding Author: {wangjsh, huangyang}@std.uestc.edu.cn

## ABSTRACT

Simulation is the most common approach to evaluate Network on Chip (NoC) designs and many simulators at different abstraction levels have been developed so far. However, researchers have to spend a considerable amount of time and effort to debug, analyze, and extract meaningful information from the simulator reports. In this work, we propose a full-system visualization framework, called VisualNoC, that support both network simulation and task mapping. VisualNoC operates in a cycle-accurate mode and is based on an event-based trace model which can record the behaviors of routers, processing elements and packets. The visualization interface can provide efficient debugging and analysis platform by representing the simulation process and results in a variety of ways. One of the main features of VisualNoC is providing an intuitive way of analyzing the efficiency of different mapping algorithms that helps in finding bottlenecks and optimizing the design.

## CCS Concepts

**•Computing methodologies → Simulation and mapping tool;**

## Keywords

Visualization Simulator, Many-Core System, Network-on-Chip, Mapping Algorithm, Debug, Statistics

## 1. INTRODUCTION

One application can be partitioned into several tasks, and each task can be allocated to one processing element (PE). Mapping algorithms determine the locations of tasks on different PEs based on the computational capacity, data dependency, and traffic distribution. To achieve a high throughput for the underlying task mapping algorithm, traffic congestion and hot-spot should be minimized. High-level full system simulations can provide the quantitative performance evaluation by using the abstract knowledge on the application tasks and the communication between tasks. The complexity of analysis increases as more details are concerned regarding the operations of PEs and routers. Translating the collected data to the network behavior is very time-consuming and complex.



A proper visualization of the network dynamics can greatly facilitate the study of specific effects and aid debugging the mapping algorithms. In design and optimization of mapping algorithms, the important parameters are as packet traveling time, traffic distribution, and the utilization of routers, links and buffers [1]. As the reported data increases, more time will be needed to analyze the data and extract important information. Displaying the statistical data would help developers to observe the design bottlenecks through intuitive plots and graphs in early design phases. Our visualization platform is also very useful for debugging purposes. Debugging is time-consuming and requires efficient tool supports. Debugging based on output text, waveform (e.g. VCD), and general debugging tools for programming languages (e.g. GDB) may not be efficient for the NoC design. A tool that tracks the network status and translates it into parameters familiar to the NoC designers can be helpful in debugging and design optimization.

VisualNoC provides a graphical user interface (GUI) to observe the behavior of routers and packets, and PEs (such as task assignment, task execution, and task termination). On the other hand, VisualNoC is not only a rich graphical user interface but also a technical tool to analyze the network behavior and task mapping by reporting statistical data. In principle VisualNoC can help developers in:

1. Analyzing routing and mapping algorithms in different aspects visually and statistically (Note that in this paper we focus on the mapping algorithms while routing algorithms can be analyzed in a similar way).
2. Observing the detailed operations of routers and movements of flits in an animation mode by tracing events.
3. Locating bugs related to coding, designs, and algorithms, e.g. deadlock, mis-routing, disorder mapping, and wrong arbitration.
4. Evaluating the statistics of the simulation, e.g. spatial and temporal traffic distribution and packet delay.

In Section 2, related work is reviewed. Section 3 introduces the overall architecture of VisualNoC. Section 4 describes the PE model as well as the event-based simulation

model. The debugger and analyzer are described in Section 5. To show the main properties of the developed platform, in Section 6, FF [2], CoNA [3], and WeNA [4] mapping algorithms are analyzed and compared with each other. The conclusion and future work are given in the last section.

## 2. RELATED WORK

An evaluation platform for mapping algorithms often integrates communication infrastructures and processors. The PE models are based on instruction-based PE models (i.e. run ISA of different kinds of PEs on the same pipeline architectures) or the actual model of a particular type of PEs (e.g. MIPS [5] and SPARC). To support PE models, some simulators provide detail and accurate models for memory systems [5, 6]. Although these models provide accurate results, they are costly in terms of time and resources.

Several cycle-accurate Network-on-Chip simulators have been developed so far, focusing on the communication between PEs, such as Noxim [7], GARNET [8] in GEMS [6], and Hermes NoC [9] in HeMPS [5].

To reduce scripting, some simulation environments have been advanced by the means of visualization. In [10, 11, 12, 13], GUI interfaces have been used to configure the simulation parameters. In [13, 14, 15], intuitive performance charts or tables are provided in addition to reporting information such as throughput, buffer utilization, traffic distribution, and power consumption. NoCScope [14] and NoCVision [15] can provide information in the dynamic environment to obverse the variations.

Even through GUIs can provide much useful information, they cannot still visualize the details of router operations and the movement of flits within routers. This shortage limits the debugging capacity as packets cannot be traced within routers. In addition, current GUIs do not provide information about the full-system simulation integrating the network and processors together, which puts up barriers on the analysis of mapping algorithms.

VisualNoC integrates a cycle-accurate NoC model with a communication-based PE model that offers a unique platform to analyze mapping algorithms. Moreover, VisualNoC is enriched with a GUI tool to automate the behavior of PEs, routers and packets while presenting the analysis results in intuitive charts and graphs and reporting the statistical data in tables.

## 3. OVERVIEW OF THE DEVELOPED PLATFORM

VisualNoC consists of two main elements: a GUI interface to visualize network behavior and a simulator to evaluate mapping algorithms (Figure 1).

A three-step design flow is shown in Figure 1. The first step is "Network Configuration" and "Task Generator". The user configures the network architecture parameters such as topology, the number of physical ports, virtual channel, and buffer sizes. The configuration is stored in an XML file that will be used by the mapping simulator. Concurrently the tasks are generated for example by TGFF (Task Graphs For Free [16]) and stored in an input task file. The input task file also contains the general information of PEs such as average power, static power, and average execution time of a task on a target PE. In the second step, several simulations are run in series or parallel using a "Cycle-Accurate

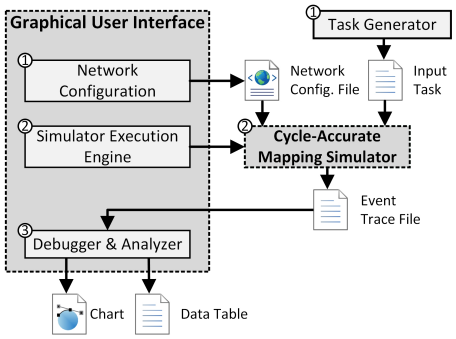



**Figure 1: Architecture of the VisualNoC platform**

Mapping Simulator", controlled by the "Simulation Execution Engine". The operations of routers, PEs, and packets are stored in an event trace file after each simulation run. In the last step, the "Debugger/Analyzer" parses the event trace file in the visualization mode. The user can easily check the correctness of the design by analyzing the operations of routers and PEs and tracing the movements of flits in the network and inside routers. Any desired operation can be easily reproduced for further analysis. The features provided by Debugger/Analyzer are very helpful to understand the design bottlenecks and locate the bugs.

## 4. SIMULATION MODELS

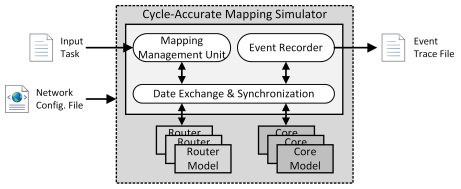



**Figure 2: Architecture of the mapping simulator**

The mapping simulator (Figure 2) is a cycle-accurate simulator, modeling both routers and PEs. The router model describes the behavior of routers such as routing and arbitration while a PE simulates a task finite state machine. The simulation platform controls the data exchanges and synchronization between the routers and PEs. The mapping algorithms are implemented in the "Mapping Management Unit" while this unit is also responsible for the allocation of tasks to PEs. The "Event Recorder" collects the events generated by the mapping simulator. The event trace file will be used by GUI for visualization.

### 4.1 Application and the PE Model

The data dependencies of an application can be presented by a directed acyclic graph (DAG) [17], as shown in Figure 3 (a). Each node in DAG denotes a task and each weighted edge denotes the communication between two connected tasks. The weight is the amount of transferred data. The task in the root is called the entry task while the tasks in leaves are called the exit tasks. The source node of an edge is called the parent task while the sink node is called the child task. Each task begins the computation only after receiving all of the necessary data. The entry tasks have no

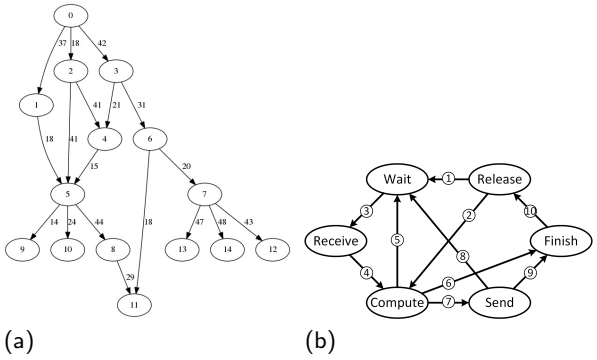



**Figure 3: Example of a task graph generated by (a) TGFF and (b) PE state machine**

parent task and thus they start running immediately after being scheduled. The parent tasks send a certain amount of data to their child tasks after the computation except the exit tasks which have no child. For example in Figure 3(a), the task 0 is the entry task and it begins the computation when the task graph is scheduled. The task 1 begins computation after it has received 37 units of data from the task 0. A task graph is called periodic if executed for several times while it is called aperiodic if run just once.

VisualNoC uses TGFF to generate task graphs which are stored in the input task file (Figure 1). This file also contains the general information of PEs such as the execution time of a task on a PE. A user can edit our template to generate a custom task graph.

The task execution process is described in a form of a state machine (Figure 3(b)). Each PE in the proposed simulator contains six states: *Release*, *Wait*, *Receive*, *Compute*, *Send*, and *Finish*. Three parameters are used to control the transitions in the state machine: the number of received packets ($P_R = \{P_R^1, P_R^2, \dots\}$), the number of sent packets ($P_S = \{P_S^1, P_S^2, \dots\}$) and the execution time ($t_{exec}$). $P_R^i$ indicates the number of received packets from the parent $i$ while $P_S^j$ represents the number of delivered packets to the child $j$. A PE is initialized in the "Release" state and switches to the "Wait" state when a task is assigned to it (Arrow 1). Once the PE has received a packet from one of its parent tasks, it switches to the "Receive" state (Arrow 3) and stays in this state until receiving all $P_R$ packets from the parent tasks. Then a transition takes place to the "Compute" state (Arrow 4). If the task is an entry task, the PE switches directly from the "Release" state to the "Compute" state (Arrow 2). The PE waits for $t_{exec}$ cycles before it switches to the "Send" state (Arrow 7). The "Send" state is skipped by the exit tasks (Arrow 5 and 6). If the task graph is periodic, the PE switches to the "Wait" state (Arrow 5 and 8) and repeats the operations until the last operation. If the task graph is aperiodic or it is under the last execution loop, the PE switches to the "Finish" state (Arrow 6 and 9). The PEs belong to the same task graph are halted in the "Finish" state until all of them switch to the "Finish" state. When the execution of the task graph is completed, the PEs return to the "Release" state (Arrow 10).

The Network Interface (NI) function is integrated into the PE model. PEs inject packets into the network by triggering events in the simulation platform. Routers generate packets as a response to these events (the number of payload flits is selected randomly). Similarly, PEs receive packets from the network by responding the events generated by routers.

## 4.2 Task Mapping and Management

The mapping simulator has a mapping management unit (MMU) where a mapping algorithm is implemented. MMU is also responsible for monitoring the latest system status and controlling the execution of applications on the whole platform. MMU reads the task graphs from the application repository and schedules them by a scheduling policy. A mapping algorithm accepts the scheduled task graph and determines the location of tasks on PEs. Then the corresponding PEs are informed about the mapping decision by receiving a triple ($P_R, P_S, t_{exec}$). A user can implement the desired mapping algorithm and the application scheduling policy using our defined interface.

## 4.3 Event-based Trace Model

**Table 1: Events in the event-based behavioral model**

| Event | Definition |
|---|---|
| PI | NI Injects a Packet into the network |
| PR | NI Receives a Packet from the network |
| FD | The router Delivers a Flit from an output virtual channel |
| FR | The router Receives a Flit from an input virtual channel |
| FS | A Flit Switches from an input virtual channel to an output virtual channel |
| CR | An input virtual Channel Requests an output virtual channel after routing calculation and virtual channel arbitration |
| CRR | The Channel Request is Released from an input virtual channel to an output virtual channel |
| CG | An input virtual Channel is Granted an output virtual channel |
| CGR | The Grant is Released from an input virtual Channel to an output virtual channel |
| CS | The State of an input virtual Channel changes. Possible values include "INIT", "ROUTING", "SW_AB", and "SW_TR". |
| PS | The State of a PE changes. Possible value include "Release", "Wait", "Receive", "Execution", "Send", and "Finish" |
| AR | Application Requests scheduling |
| AB | Application Begins execution |
| AS | Application Stops execution |

An event-based model is employed to record the network behavior and mapping operations. The behavior of routers, PEs, and flits are abstracted into a series of events listed in Table 1. Each event is described by a set of parameters, representing almost all information about that event. The parameters are such as the occurrence time of the event and the source and the destination of a packet. The PI (PR) event denotes packets' injection (removal) to (from) the network. FR, FS, and FD represent the movement of flits within a router. CR and CRR events show the results of the routing decision while CG and CGR events show the allocation results. The CS event denotes the operations of the state machine in routers. In summary, PI, PR, FR, FS, and FD events trace the activities on the data path (such as buffers and the crossbar unit) while CR, CRR, CG, CGR, and CS events record the behavior of the control units (such as the routing unit and the switch allocation unit). In addition to

these events, AR, AB, AS, and PS events are involved with the task mapping where they refer to application scheduling request, execution start, execution stop, and the PE state change, respectively.

## 4.4 Statistical Data

The event trace file is used to extract a lot of useful information and statistical data such as resource utilization, task throughput and congestion.

Most of the common statistic metrics can be extracted using the described events as follows where in the equations $sc$ and $ec$ refer to the "Start Cycle" and "End Cycle", respectively. $Count(e, sc, ec)$ is a function to count the number of times that the event $e$ has occurred from cycle $sc$ to cycle $ec$ ($ec$ is not included). Similarly, $CounterR(e, c)$ and $CounterB(e, c)$ are the number of times that the event $e$ (related to an specific router or buffer) has occurred up to the cycle $c$ (cycle $c$ is included).

(a) Packet injection rate (packets/cycle) from cycle $sc$ to cycle $ec$ can be obtained by counting the number of PI events occurring within specified time.

$$\text{PIR} = \frac{Count(PI, sc, ec)}{ec - sc} \quad (1)$$

(b) Network throughput (NetT) (packets/cycle) from cycle $sc$ to cycle $ec$ is the number of PR events occurring within specified time.

$$\text{NetT} = \frac{Count(PR, sc, ec)}{ec - sc} \quad (2)$$

(c) The number of stored flits in a router (FIR) at cycle $c$:difference between the number of FR events and the number of FS events (related to the specific router), from simulation start time to cycle $T$.

$$FIR(c) = CounterR(FR, c) - CounterR(FS, c) \quad (3)$$

(d) The number of stored flits in a buffer (FIB) at cycle $c$:the same with FIR, except that the FR and FS events is related to the specific buffer.

$$FIB(c) = CounterB(FR, c) - CounterB(FS, c) \quad (4)$$

(e) Application throughput (AppT) from cycle $sc$ to cycle $ec$ is the number of AS events occurring within specified time.

$$\text{AppT} = \frac{Count(AS, sc, ec)}{ec - sc} \quad (5)$$

(f) Application execution time (AppET) is the occurrence time of AS minus the occurrence time of AB for a specific application. Application average execution time (AppAET) from cycle $sc$ to cycle $ec$ is the average of AppETs for all applications.

$$\text{AppAET} = \frac{\sum_{i=1}^{count(AS,sc,ec)} (AS_i - AB_i)}{Count(AS, sc, ec)} \quad (6)$$

# 5. FUNCTIONS OF GRAPHICAL USER INTERFACE

**Table 2: Network and router configuration parameters**

| | |
|---|---|
| Network Configuration | Topology (2D-mesh/2D-torus/Irregular), network size, number of ports |
| Port Configuration in Network Configuration | Number of virtual channels, port direction, network interface port, size of input buffers and output buffers |
| Router Configuration | Router position in graphics axis, number of ports |
| Port Configuration in Router Configuration | Number of virtual channels, port direction, network interface port, size of input buffers and output buffers, neighboring ports (router id, port id) |

## 5.1 Network Configuration

VisualNoC provides an option to configure the network in GUI while it also accepts the network configuration in an XML file. The network configuration parameters are classified into network configuration and specified router configuration. In the network configuration, the parameters (in the first and second row of Table 2) are valid for all routers. If there are some heterogeneous routers (different from the default parameters in the network configuration), router configuration parameters (in the third and fourth row in Table 2) are used to customize the special routers. With the help of network configuration and specified router configuration, VisualNoC supports regular topologies as well as irregular topologies.

## 5.2 Execution Engine

The simulations and GUI run in parallel. The user can configure the arguments of simulations and call the simulation in GUI directly. To take advantage of the computational capacity of a physical computer and to speed up the whole process, the execution engine supports parallel simulations, each with different parameters. The execution engine can start more than one simulation in parallel and collect the simulation results at the end of each simulation and generate a table directly for further analysis.

## 5.3 Debugging

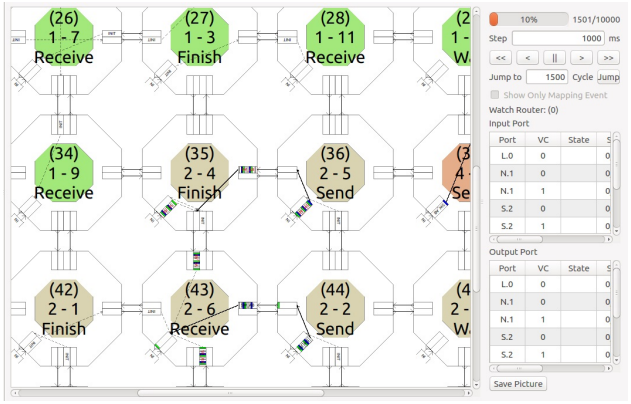



**Figure 4: Event trace reproduction panel**

To debug the design, the simulation can be replayed using a replay event trace panel (Figure 4). The control panel

controls the reproduction process, including pause, forward, backward, fast-forward and fast-backward at the desired speed. The program parses the event trace file and visualizes the corresponding activities on each virtual simulation cycle. Therefore, VisualNoC offers a cycle-accurate retrace of network behavior. Animation may take a long time from the start to the desired cycle. The replay process of VisualNoC allows jumping directly to any desired cycle. If only the PE behavior is concerned, the network event activity can be turned off to accelerate the retrace process. During visualization, extra information regarding the router under observation is dynamically reported in a panel, such as buffer utilization, and the source and destination of the first packet in the buffers.

During the retrace of the event trace file, packets move in NoC on the screen. As shown in Figure 4, both PE 36 and PE 44 send packets to PE 43. Both streams request the NI of the Router 43 at the same time which leads to the congestion at the input ports of the router 43. This port contention affects the congestion at the router 35, 36 and 44 as well. As shown in Figure 4, the application ID, the task number, and the PE state are labeled in each router. For example, the label of the router 36 indicates that the task 5 of the application 2 is currently executing in the PE and the PE is in the "Send" state. The routers of the PEs which are assigned to the same application are represented by the same color, like PE 35, 36, 42, 43, and 44 that are allocated to application 2 and thus colored the same.

## 5.4 Analysis

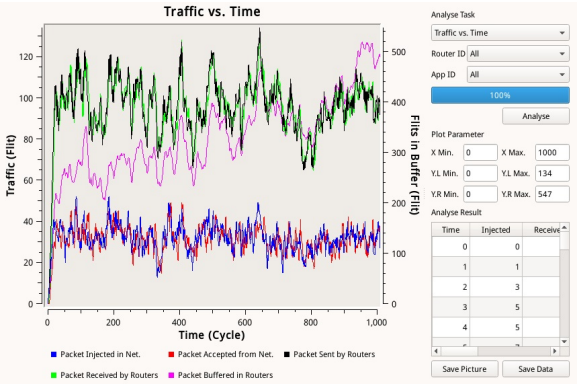



**Figure 5: Event trace analysis panel**

Event trace analysis panel displays the information provided in the event trace file. In Figure 5, the analysis results are depicted with line charts, bar charts or spectrogram charts to give a clear and direct view of statistic variables. In the right panel, the analysis tasks, statistic filters, figure types, and axis ranges can be controlled by the user. Moreover, the values of statistic variables are listed in the table. Event trace analysis accepts filters to specify the analysis range. The parameters of a filter contain specified stream, router, port and application. The analyzer only considers the events specified by the filter. For instance, if the application ID is given in the filter, the analyzer only counts the events related to this application.

The analysis tasks are listed as bellow:

1. *General Statistic* shows the global statistic variables about the application and network behavior, for example the packet injection rate, throughput, average latency, maximum latency, application throughput, and average execution time.

2. *Traffic Temporal Distribution* shows the fluctuation in injecting/receiving flits to/from the network, fluctuation in receiving/sending flits by routers, and fluctuation in buffering flits in routers during the simulation. This option accepts the specified router and application ID for filtering.

3. *Traffic Spatial Distribution* shows the distribution of flits injected into the network, flits received from the network, and flits flow through the routers. The spatial traffic distribution is illustrated with a bar chart or a color-spectrogram. Moreover, VisualNoC can show the distribution within a time window. The window can shift with a specified step length and the charts are updated automatically. This option accepts the application ID for filtering.

4. *Traffic Spatial Distribution of Ports* shows the flow of flits through the ports by reporting the number of flits at the ports. This option accepts the application ID for filtering.

5. *Distance and Latency Distribution* shows the frequency of packets with a certain distance or latency. This option accepts specified stream and application ID for filtering.

6. *Application Information* shows the states of PEs in the Gantt chart. The states of PEs are marked with different colors. The result table lists the time when applications enter and quit the network and when applications are mapped.

# 6. CASE STUDIES

In this section, an $8 \times 8$ homogeneous many-cores system is simulated. Each router has four ports in the East, West, North, and South directions and one port to the PE [18]. No virtual channel is used in this network and the XY routing algorithm is applied [19]. PEs inject packets (5 flits/packet) into the network with the maximum packet injection rate of 0.2 packet/cycle/router.

Task graphs are generated by TGFF. The number of tasks is between 4 and 16. The communication volumes between each two tasks are spread over 10 to 50 packets and the computation time is assumed to be between 60 to 140 cycles. All task graphs are a-periodic. The applications enter the platform at a rate of 0.002 application/cycle and are scheduled under the First Come First Serve (FCFS) policy. Simulations last for 10,000 cycles.

First Free (FF [2]), CoNA [3] and WeNA [4] mapping algorithms are used for the evaluation. The FF algorithm tracks the rows in the ascending order and maps the tasks into the first free PE. The CoNA algorithm tries to keep the mapped region contiguous and thus places the communicating tasks in a close neighborhood as much as possible. The WeNA algorithm improves the CoNA algorithm by sorting the tasks based on the communication volume. WeNA tries to replace the CoNA random steps by exploiting a more deterministic and reasonable allocation policy. For this purpose, WeNA

utilizes the expanding parameter and occupancy status in its mapping decision. To have a fair performance comparison between WeNA and CoNA, the first node selection policy of CoNA is also applied in WeNA as well. For all mapping algorithms, PE 0 is used as the central manager (CM) which is in charge of the resource management.

All the figures in this section are screenshots that are directly extracted from VisualNoC.

## 6.1 Simulation Reproduction

Figure 6 shows the screenshots of the PEs and routers at the random cycle of 1553 which gives an intuitive view about the mapping strategy of the FF, WeNA, and CoNA algorithms regarding App 0, 1, 2 and 3. As the simulation starts, the tasks of App 0 are mapped into PE 1, 8, 9, 10 and 11 (all the tasks are terminated before the cycle 1553 and thus not shown in Figure 6). Then App 1 (green color) and App2 (pink color) enter the platform at the cycle 500 and 1000, respectively. CoNA and WeNA map these applications close to App 0. Tasks of App 1 are mapped into a more contiguous region using the WeNA mapping algorithm than CoNA. Tasks of App 2 are mapped to the almost similar PEs under both WeNA and CoNA. Based on the quantitative values, extracted from VisualNoC, the Average Weighted Manhattan Distance (AWMD) of App 1 and App 2 under WeNA is 1.87 and 1.61 hops while these values are 2.28 and 1.79 hops under CoNA. The task graph of App 2 is shown in Figure 3(a). Based on the WeNA mapping algorithm, the tasks with more communication volume are mapped close to each other such as the tasks in communication with the task 5.

Before App 3 enters the platform, App 0 has already been completed and a fragment in the northwest corner of the network is de-allocated and became free. Because the mapping algorithms tend to start mapping from the closest PE to the central manager (i.e. PE 0 in this example), this free fragment is selected for the mapping of App 3 which results in a discontinuous mapping region. The AWMD of App 3 is 2.11 and 1.77 hops under WeNA and CoNA, respectively.

The FF mapping algorithm, shown in Figure 6(c), maps the tasks row by row that leads to a totally different task mapping than both WeNA and CoNA.

## 6.2 Analysis and Statistics of the Entire Simulation

### 6.2.1 General Statistic

**Table 3: Statistical variables from simulation**

| Parameter | FF | CoNA | WeNA |
|---|---|---|---|
| Packet Injection Rate (packets/cycle) | 0.61 | 0.615 | 0.614 |
| Packet Throughput (packets/cycle) | 0.605 | 0.61 | 0.608 |
| Average Weighted Manhattan Distance (AWMD) | 3.281 | 1.783 | 1.66 |
| Max Manhattan Distance | 10 | 5 | 6 |
| Average Latency (cycles) | 91.017 | 50.944 | 42.177 |
| Maximum Latency (cycles) | 711 | 528 | 526 |
| Total Application Request | 20 | 20 | 20 |
| Total Application Enter | 20 | 20 | 20 |
| Total Application Exit | 15 | 15 | 15 |
| Average Execution Time (cycles) | 1757.67 | 1593.53 | 1529.4 |

Table 3 shows the statistics on the mapping algorithms and also the network performance parameters. FF shows the highest latency value and execution time. This is mainly due to the highest average distance than the other two algorithms. WeNA leads to the best average distance, average latency and average execution time. The average packet latency of the FF, CoNA and WeNA mapping algorithms is 91.0, 50.9, and 42.2 cycles, respectively, and AWMD of FF, CoNA and WeNA is 3.28, 1.78 and 1.66 hops, respectively. The reason behind this observation is that WeNA has a better allocation policy by taking into account more parameters in the task mapping decision.

Due to the similar performance level of the CoNA and WeNA mapping algorithms, for the rest of the evaluation part, we focus on these algorithms to observe the differences in other aspects.

### 6.2.2 Gantt Charts

Figure 7 is the Gantt chart for the WeNA algorithm when running all applications during the entire simulation period. Different colors represent different execution phases (i.e. described in Section 4.1) while different border colors denote different applications. "Release" state has no color and no border. The color for "Receive", "Compute" and "Send" is magenta, green and red, respectively. In "Wait" and "Finish", the color is same as the border but in different opacity. The horizontal axis shows the time and the vertical axis represents the PE ID.

From the Gantt chart, we can easily observe that the utilization of PEs that are located closer to the central management unit is higher than the rest (e.g. PE 1 vs. PE 52). PE 0 is the central management unit, taking the responsibility of task scheduling and mapping. PE 1 executes applications for 8836 cycles with the utilization of 88%. On the other side, PE 56, 57 and 63 are never used and thus the utilization is 0%. The reason for this low utilization is the trend of mapping tasks closer to the central management unit.

### 6.2.3 Distance Distribution

The packet distance distribution of the WeNA and CoNA mapping algorithms are illustrated in Figure 8(a) and (e). The figures show that WeNA has a higher proportion of short distance packets compared with CoNA. Using WeNA, 3720 packets take only one hop while using CoNA 3131 packets route just for one hop. On the other hand, the worst-case packet distance is higher in WeNA than CoNA (18 packets with 6 hops vs. no packet in CoNA).

### 6.2.4 Traffic Spatial Distribution

Traffic spatial distribution graph shows the spatial distribution of packets injected or accepted by processing elements. It also shows the spatial distribution of packets sent or received by routers during the execution time of a particular application or an entire simulation run. The graph can be a bar chart or a spectrum. The red color shows the routers with the most delivered packets while the blue color indicates those with the least delivered packets.

Figure 8(d) and (h) show the spatial distribution of packets passing through routers during the entire simulation run. The hotspot under both WeNA and CoNA appears at the router (1,1) with 1840 delivered flits. The traffic on the northwest part is heavier than the other parts mainly because of the location of the central manager at PE 0.

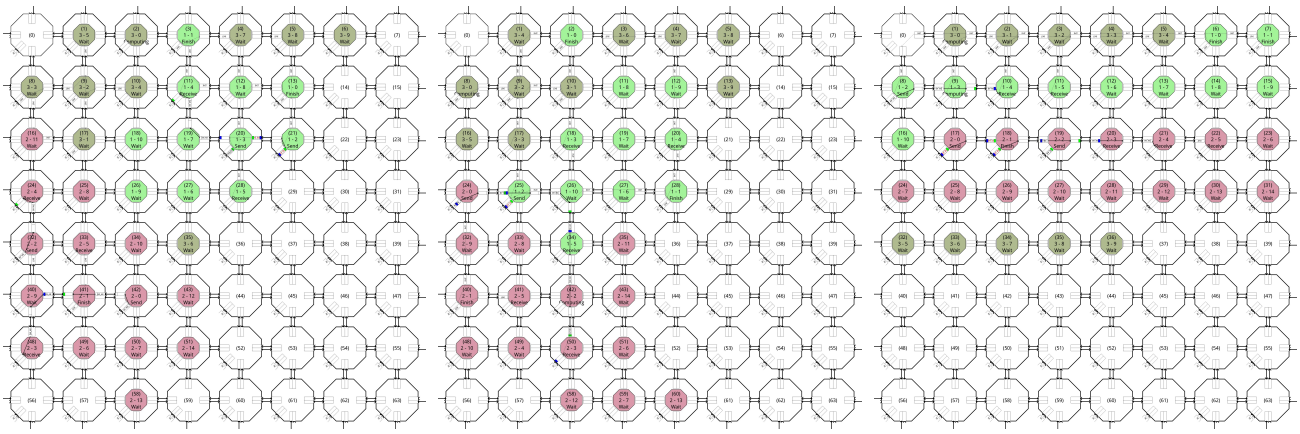



**Figure 6: Mapping reproduction of applications 1, 2, and 3 under (a) WeNA (b) CoNA, and (c) FF**

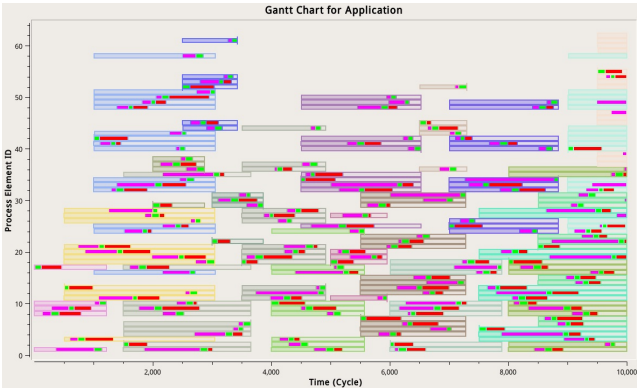



**Figure 7: Gantt chart for all applications using WeNA mapping algorithm.**

## 6.3 Analysis and Statistics of Application 2

In this section, we take App 2 as an example. The task graph of App 2 is shown in Figure 3(a).

Figure 8(b) and (f) illustrate Gantt charts for the App 2 under WeNA and CoNA. In CoNA, App 2 lasts for 2388 cycles while in WeNA it lasts 2059 cycles. The major difference between these two Gantt charts can be seen in the "Receive" status of PE 50 in Figure 8(f) (i.e. Task 3 of App2 in CoNA) which lasts for about 400 cycles. This status holds for only 200 cycles at PE 48 in Figure 8(b) (i.e. Task 3 of App2 in WeNA). This observation can guide a designer to recognize the bottleneck.

Figure 8(c) and (g) show the spatial distribution of packets passing through routers during the execution time of the App 2. When CoNA is applied, the hotspot appears in the router 26 within the region of App2 while when WeNA is applied, the hotspot is the router 19 which is far away from the area of App2.

# 7. CONCLUSION AND FUTURE WORK

Simulation is the most common approach to evaluate the mapping algorithms. But the lack of visualization tools raises the difficulties to analyze the simulation results and fix the bugs introduced by design defects and programming errors. In this work, we introduced VisualNoC, a unique visualization framework for simulation and mapping. VisualNoC is able to reproduce the detailed operations of PEs, routers and packets and report the simulation results visually and statistically. VisualNoC integrates a mapping simulator based on the communication-based PE model with a graphical user interface. The rich aspects of VisualNoC can greatly help designers on debugging and analysis.

# Acknowledgment

This paper was supported by the National Natural Science Foundation of China (NSFC) under grant No. 61176025, No. 61006027, No. 61534002, and the Oversea Academic Training Funds (OATF), UESTC. This work is also supported by VINNOVA-MarieCurie and Academy of Finland.

# 8. REFERENCES

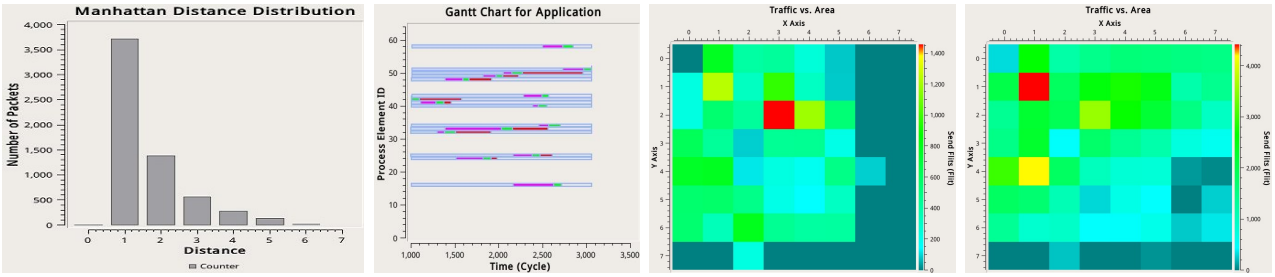
[1] H. Hsin, E. Chang, C. Lin, and A.-Y. Wu, "Ant colony optimization-based fault-aware routing in mesh-based network-on-chip systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 11, pp. 1693–1705, 2014.

[2] E. Carvalho, N. Calazans, and F. Moraes, "Heuristics for dynamic task mapping in noc-based heterogeneous mpsocs," in *Rapid System Prototyping, 2007. RSP 2007. 18th IEEE/IFIP International Workshop on.* IEEE, 2007, pp. 34–40.

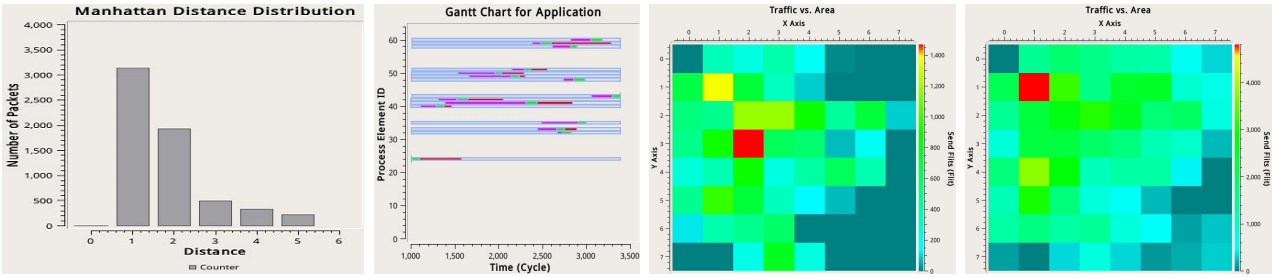
[3] M. Fattah, M. Ramirez, M. Daneshtalab, P. Liljeberg, and J. Plosila, "Cona: Dynamic application mapping for congestion reduction in many-core systems," in *Computer Design (ICCD), 2012 IEEE 30th International Conference on.* IEEE, 2012, pp. 364–370.

[4] L.-T. Huang, H. Dong, J.-S. Wang, M. Daneshtalab, and G.-J. Li, "Wena: Deterministic run-time task mapping for performance improvement in many-core embedded systems," *Embedded Systems Letters, IEEE*, vol. 7, no. 4, pp. 93–96, 2015.

[5] E. A. Carara, R. P. De Oliveira, N. L. Calazans, and F. G. Moraes, "Hemps-a framework for noc-based




(a) Packet distance distribution, WeNA (b) Gantt chart, App 2, WeNA (c) Traffic vs. area, App 2, WeNA (d) Traffic vs. area, all Apps, WeNA




(e) Packet distance distribution, CoNA (f) Gantt chart, App 2, CoNA (g) Traffic vs. area, App 2, CoNA (h) Traffic vs. area, all Apps, CoNA

**Figure 8: Analyze charts. Subfigure (a) and (e) are packet distance distribution for all applications. Subfigure (b) and (f) are Gantt chart for App 2. Subfigure(c) and (g) are traffic spatial distribution for App 2. Subfigure(d) and (e) are traffic spatial distribution for all applications.**


mpsoc generation," in *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1345–1348.

[6] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 92–99, 2005.

[7] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Noxim: an open, extensible and cycle-accurate network on chip simulator," in *Application-specific Systems, Architectures and Processors (ASAP), 2015 IEEE 26th International Conference on*. IEEE, 2015, pp. 162–163.

[8] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 33–42.

[9] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "Hermes: an infrastructure for low area overhead packet-switching networks on chip," *INTEGRATION, the VLSI journal*, vol. 38, no. 1, pp. 69–93, 2004.

[10] D. Ghosh, P. Ghosal, and S. Mohanty, "A highly parameterizable simulator for performance analysis of noc architectures," in *Information Technology (ICIT), 2014 International Conference on*, 2014, pp. 311–315.

[11] https://www.ice.rwth-aachen.de/research/tools-projects/grace/visualization/.

[12] https://www.academia.edu/2805883/ATLAS-An_Environment_for_NoC_Generation_and_Evaluation, Tech. Rep.

[13] P. Gottschling, H. Ying, and K. Hofmann, "Gsnoc ui-a comfortable graphical user interface for advanced design and evaluation of 3-dimensional scalable networks-on-chip," in *2012 International Conference on High Performance Computing and Simulation (HPCS)*, 2012, pp. 261–267.

[14] L. MoÌĹller, L. Indrusiak, and M. Glesner, "Nocscope: A graphical interface to improve networks-on-chip monitoring and design space exploration," in *4th International Design and Test Workshop (IDT)*, 2009, pp. 1–6.

[15] http://nocvision.eecs.umich.edu/.

[16] 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.

[17] O. Sinnen, *Task scheduling for parallel systems*. John Wiley & Sons, 2007, vol. 60.

[18] L. Huang, J. Wang, M. Ebrahimi, M. Daneshtalab, X. Zhang, G. Li, and A. Jantsch, "Non-blocking testing for network-on-chip," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 679–692, March 2016.

[19] F. Farahnakian, M. Ebrahimi, M. Daneshtalab, P. Liljeberg, and J. Plosila, "Q-learning based congestion-aware routing algorithm for on-chip network," in *Networked Embedded Systems for Enterprise Applications (NESEA), 2011 IEEE 2nd International Conference on*, Dec 2011, pp. 1–7.