Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Technical Communication

SimuRed: A flit-level event-driven simulator for multicomputer network performance evaluation

Fernando Pardo [*], Jose A. Boluda

Departamento de Informática, Universidad de Valencia, Avda. Vicente Andrés Estellés s/n, 46.100 Burjassot, Valencia, Spain

A R T I C L E   I N F O

Article history:
Received 17 October 2007
Received in revised form 18 September 2008
Accepted 10 February 2009
Available online 16 March 2009

Keywords:
Multicomputers
Interconnection networks
Flit-level simulators
Packet latency
Performance modelling
Simulation

A B S T R A C T

The interconnection network is one of the most important multicomputer components, since it has a great impact on global system performance. Many models and simulators have been proposed to evaluate network performance. This paper presents SimuRed, an event-driven flit-level, cycle-accurate simulator to evaluate different orthogonal network configurations. The core of the simulator has been designed to be expandable and portable to different situations. Some of the advantages of this simulator over other similar tools are its visual interface, its fast execution and its simplicity. Moreover, it is multiplatform and its source code versions (C++ and Java) are freely available under GNU open-source license. The performance of this simulator has been evaluated, including a performance impact study of injection channels and deterministic/adaptive routing for meshes and hypercubes.

© 2009 Elsevier Ltd. All rights reserved.

1. Introduction

1.1. Multicomputer network evaluation

Current high-performance systems contain from tens to thousands of processors. The interconnection network in these systems, even if they only have tens of processing nodes, is of vital importance to the global performance of the system. For this reason, many researchers have addressed this issue, which is currently one of the most active research topics in computer architecture [1]. Network simulation is one of the key stages when designing new parallel systems, since the network is one of the elements with higher impact in global machine performance.

Simulation is necessary to design new multicomputer interconnection networks and to evaluate their performance before spending money on their construction. There are two widely-used approaches to network simulation: the first one describes the hardware behaviour at component level to obtain accurate (flit-level), though time-consuming, simulations; the second approach creates abstract analytical (statistical) models of network behaviour, obtaining less accurate simulations, though with faster execution times. Some researchers have defended the statistical model as the main tool for multicomputer network evaluation [2], while most researchers have employed some kind of flit-level simulator to validate their statistical models [3–6]. Both approaches are important to performance evaluation and we will discuss their pros and cons in this paper.

Statistical network modelling has several advantages over other approaches. Probably the most important is the low execution time, even for large networks; a flit-level simulator may take hours to complete a single simulation, while the analytical model gives similar results in a few minutes. But there are other advantages: the modelling process requires a perfect

* Corresponding author. Tel.: +34 9635 43943; fax: +34 9635 44768.
E-mail addresses: Fernando.Pardo@uv.es (F. Pardo), Jose.A.Boluda@uv.es (J.A. Boluda).

0045-7906/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compeleceng.2009.02.004

knowledge of the network being modelled and the behaviour of its internal processes; thus on making a model of a network, a great deal of information about the way it works is produced.

As network analytical models have such advantages, then why should we still use flit-level simulators? There are at least two reasons: accuracy and robustness. Analytical models are usually complex and this complexity increases with the precision required. Most models are based on differential equations that may require sound programming algorithms that cannot converge in some situations. On the other hand, model debugging also becomes difficult since there are many sources of error and results are difficult to validate. In fact, most researchers have employed a flit-level simulator to validate their analytical models.

Analytical models and flit-level simulations can coexist perfectly; one can be more suitable than the other depending on the situation. Analytical models seem to be superior for general network characterization and evaluation, fast simulation, network understanding, statistical traffic analysis, stationary network behaviour, etc. Meanwhile, flit-level simulators are better suited to accurate network simulation, network debugging (routing and switching design), analytical model validation, real traffic simulation, visual network inspection, stationary and transient network behaviour, etc.

One of the main advantages of flit-level simulators is their capability for network debugging, since they enable packet movement to be checked cycle by cycle. Most flit-level simulators do not have a visual interface to check packet movement, thus debugging is still possible even though it is more complex.

Another interesting feature of flit-level simulators is their ability to monitor packet traffic at every instant to see transient network behaviour. This feature is not possible, or becomes very complex, when modelled analytically.

The last remarkable feature of flit-level simulators, which is very difficult for an analytical model, is the simulation of real traffic where the arrival of one packet may produce the generation of more packets.

We present SimuRed in this article, a flit-level simulator with special emphasis on those features that are difficult for an analytical model to achieve, these being: network debugging, transient traffic analysis and real traffic simulation. Measurements of the execution time of SimuRed have shown that this tool is usually slower than an analytical model, but it is fast enough to obtain good results in just minutes.

1.2. Flit-level computer network simulators

While many analytical models are described in literature for computer network evaluation, few flit-level simulators have been reported. There are some references to flit-level simulators used to evaluate analytical models [3–6], but detailed descriptions are not given. Other authors have published network performance evaluations using their own flit-level simulator [7–10], but again there are few details or specific literature about their simulators.

Most flit-level simulators reported in the past are not available anymore. One of the few flit-level simulators reported and still in use is SWORDFISH [11]. This simulator shares many characteristics with SimuRed and is still under development [12]. Another active flit-level simulator is the Book sim presented at [13]; this simulator does not offer a graphical interface unlike SWORDFISH and SimuRed. One of the best reported flit-level simulators is pp-mess-sim [7]. This simulator was designed to evaluate the performance and design of multicomputer networks; some of the evaluation results have been employed in the design of a real-time router architecture [8]. Apart from [7] there is no more literature about this simulator and the source code or the program itself is not on the web. The same is true in case of another good flit-level simulator called SMART [14]. Again, the source code and the program itself are difficult to find and use. There are other multicomputer architecture simulators like PEPE [15]. In these cases, there is little flexibility on the details and configuration of the network, since the goal is to study other higher-level architectural parameters.

One of the areas where flit-level simulators are applied is to evaluate network analytical models. A simple simulator for multicomputer routing networks, Pertel [16], has been used to evaluate a model for interconnecting subsystems for massively parallel computers [2]. Other researchers [3–6] have validated their analytical models using a flit-level simulator for different routing and switching techniques. In none of these cases have the authors given details (developer, language, platform, etc.) about the flit-level simulator implementation they have used. This lack of description of the validation tools, has lead some authors to complain about how difficult it is to reproduce and confirm some simulated research results [17]. Our goal in creating SimuRed was to develop a publicly available tool so that everybody can use and test it. The source code is freely available under GNU general public license and it can be downloaded from its web site at http://simured.uv.es. This will allow each researcher to evaluate, reproduce and eventually validate research results achieved using this tool.

Flit-level simulation is in between packet-level and gate-level simulations. Gate-level simulation is the lowest and the most accurate level and it is interesting when designing and refining multicomputer network routers. Before exploring this level, it is better to use analytical models or flit-level simulators to get an idea of network performance parameters. One gate-level simulator worth citing is Orion [18], which is able to evaluate network performance, though it is specially dedicated to studying network power consumption.

Regarding higher-level simulation (packet level, message level and above [19]), there are some tools where real programs are evaluated on simulated multicomputer systems. One of these tools is the BigSim simulator [20], which is a parallel application for simulating large systems like the Blue Gene. It has an internal flit-level module called BigNetSim [21], which can be changed to adapt to specific network characteristics. One of the main advantages of this tool is that it is parallelized so it can run faster in a parallel computer [22], though its performance decreases in a non-parallel system.

2. SimuRed network modelling

Network topology, switching mechanism and routing algorithm are the three main issues characterizing any multicomputer network [9]. The network topology defines node inputs and outputs and their interconnections. The switching mechanism defines the way in which information is physically sent through the network. Finally, the routing algorithm decides the path between the origin and destination nodes.

2.1. Network topology

Many topologies have been proposed for multicomputer network implementation. Nevertheless, during recent years, strictly orthogonal direct network topologies have been those most commonly employed for massively parallel multicomputers, especially the k-ary n-cube type. Some authors argue that, from the scalability and performance point of view, direct networks of the k-ary n-cube (torus) type are recommended [23]. The hypercube is a popular instance of k-ary n-cube found in systems like the SGI Origin 2000 [24]. Torus is also a very common example of k-ary n-cube found in the J-machine [25], Cray T3D [26], Cray T3E [27] and the most recent IBM Blue Gene/L [28].

SimuRed implements the popular k-ary n-cube (torus) topology. Hypercube is a particular case when $k = 2$. The definition of a mesh does not strictly fit in a k-ary n-cube topology, though any mesh with an equal number of nodes per dimension can be simulated as a torus if the *warp-around* channels are forbidden inside the routing algorithm; in fact, this is the technique used in SimuRed for simulating meshes of any dimension. Complex meshes with different node number per dimension can be still simulated imposing further restrictions on the routing algorithm.

Fig. 1 shows a portion of a typical 2-D mesh or 2-D torus topology. This a particular case SimuRed can handle since it supports any-dimensional mesh or torus. Each node of the network has a router and a processor. SimuRed is able to print this structure showing the internal elements of the router and the state of the packets in the network (see Fig. 4).

The list of topological parameters that can be specified in the SimuRed simulator follows (these parameters are directly available from the simulator Graphical User Interface):

Dimensions: It is the number of network dimensions (n). Tori usually have two or three dimensions, but this number may reach higher values for small k, like in hypercubes ($k = 2$) where a large number of nodes implies a large number of dimensions.

Nodes per dimension: It is the number of nodes for each dimension (k). The total number of nodes in the network is thus $n \times k$.

Buffer length: The buffer length is defined in flits. It can be any number starting from 1 (null buffer is not allowed). There is an input buffer and an output buffer for each channel and this length is specified for both buffers. Also this length applies for the injection and ejection channel buffers.

Virtual channels: The number of virtual channels for each physical channel can also be specified. The presence of virtual channels increases the network throughput and they are necessary for most of adaptive routing algorithms.

Injection channels: This is the number of channels from the processor to the network. Usually there is just one but when there are a large number of virtual channels, it would be better to have more than one injection channel depending on the network throughput.

Ejection channels: This is the number of channels from the network to the processor. It is usually one, but by implementing many of these channels the contention at the arrival node can be reduced.

The only factors limiting these parameters are the total available memory of the computer and the amount of patience needed to obtain the results (a simulation with millions of nodes could take days to complete). These topological components may display different behaviour depending on the following options:

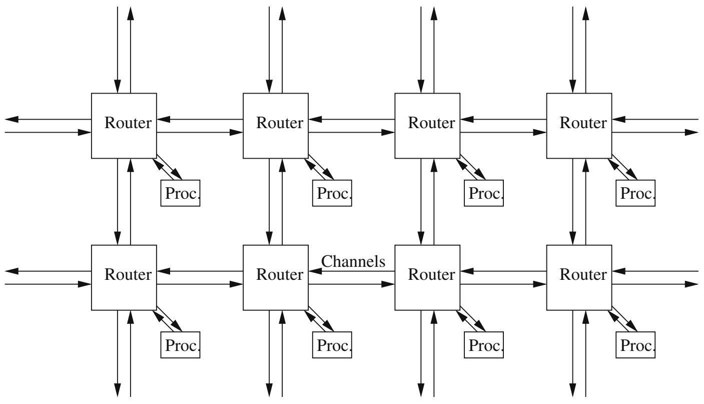

Fig. 1. Particular network model for a 2-D mesh or torus topology.

Bidirectional: This option specifies whether node interconnections are bidirectional or not. For bidirectional interconnection an input and output buffer is provided for each channel.

Buffer forwarding: It specifies whether flits may jump directly ahead of the FIFO buffer if it is empty. If not selected, flits must cross all the buffer's registers from the beginning to the end, even when they are empty.

Physical channels: When several virtual channels have been specified, this option indicates whether these virtual channels share the same physical channel or, if on the contrary, each virtual channel is in fact a physical channel in itself.

The time unit in SimuRed is the *clock cycle*, which is the time step in the simulation. Delays for the network physical components are specified in cycles. Thus, simulation accuracy is one cycle. These are the components that introduce some delay in packet movement:

Buffer: The delay in buffer is the time that a packet flit needs to go from one flit buffer to the next. Usually this is the least complex operation in any network, so this delay tends to be one.

Crossbar: This delay is the time a flit takes to pass through the crossbar switch. Depending on its complexity it may be more than one compared to the buffer delay.

Switching: This is the time to take a routing decision and inform the crossbar to switch in specified direction. Depending on the routing complexity and the crossbar implementation it may take several cycles.

Channel: This is the time required for one flit to go from the output buffer of one node to the input buffer of the next node through the physical channel.

This delay model is extensively described in [9]. All these values can be changed at the simulator user interface. Most analytical models and simulation results shown in bibliography use a unit time delay; for more accurate network analysis it is necessary to obtain accurate delay values from gate-level simulation and feed them to SimuRed.

2.1.1. Router model

The router implemented in SimuRed is an adaptation of the one described in [9]. This router is general and matches those used in other simulators and analytical models [10]. Fig. 2 shows the router scheme.

The router model of Fig. 2 has v virtual channels for each physical channel (physical option disabled). It involves n dimensions and bidirectional channels, thus it has n input channels and n output channels. In that model there is only one ejection and one injection channel, but this is a particular case since SimuRed supports any number of injection/ejection channels.

2.2. Switching mechanism

There are several switching mechanisms available for multicomputer networks. One of the most common techniques is the *wormhole switching* for its simplicity at the router level. SimuRed is especially suitable for pipelined switching techniques like the wormhole. Virtual Cut-Through can be implemented just making the input/output buffers large enough. Other pipelined techniques like packet switching may require the change of the source code a little bit. The implementation of non-pipelined techniques such as classic circuit switching requires a deep change in the program source code.

2.3. Routing algorithms

One of the most interesting properties of flit-level simulators is the ease with which new routing algorithms are implemented and tested. If the simulator also includes a visual tool, the debugging of the routing algorithm implementation be-

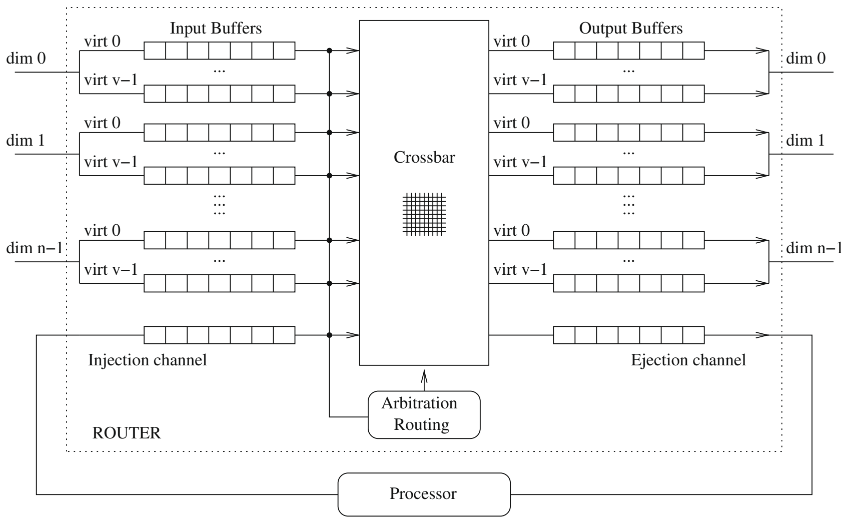

Fig. 2. General simulator router model.

comes a simple task. SimuRed has a graphical interface which allows the user to see the network nodes and traffic of the two lowest network dimensions. The simulator can also be run step by step, thus packet movement can be tracked at flit-level cycle by cycle.

Implementation of new routing algorithms is simple in SimuRed from the already existing routing examples. The code is completely object-oriented in its two versions (C++ and Java) so any new routing algorithm can be coded just by making a copy and pasting it from the existing examples and then introducing the required changes.

This is a list of currently implemented routing algorithms:

Dimensional order for meshes: This is the simplest deterministic routing algorithm for meshes of any dimension and size. It is also known as *XY* for two-dimensional meshes since it first routes the packet in *X* direction and then in *Y* direction. The path is always the same for the same source/destination pair, thus it is really deterministic. In the case that there is more than one virtual channel, the packet is routed through the first randomly selected free channel.

Dimensional order for torus: This algorithm is the same as the one described before but for *k*-ary *n*-cubes. This algorithm is an example of non deadlock-free routing, since deadlock may happen due to the torus *wrap-around* channels.

Duato's adaptive routing for meshes: This is a simple, fully-adaptive routing algorithm based on Duato's protocol [29] and dimension-order deterministic routing. It is necessary to have at least two virtual channels to achieve full adaptability; otherwise, it is exactly like the dimension-order algorithm.

Fully-adaptive routing for meshes (non deadlock-free): This is an example of full adaptability without paying attention to deadlock avoidance.

The non deadlock-free algorithms described before have been included in the simulator just for the sake of testing its ability to detect and show deadlock configurations in the network. SimuRed has not been designed to check exhaustively whether an algorithm is deadlock-free or not, but if an algorithm is not deadlock-free, a deadlock configuration will probably show up sooner or later.

When two or more packets in the crossbar demand the same output buffer, contention is produced in the crossbar. This contention is solved using an arbitration mechanism. The default arbitration implemented in SimuRed is the simple and fair *fifo* (the first packet to arrive is the first to go).

The routing algorithms presented so far use minimal paths. This is not a limitation of the simulator but of the specific routing implementation. Non minimal path algorithms can easily be implemented in SimuRed.

3. Simulator architecture

SimuRed is divided into two blocks: the core and the user interface. The core of the simulator is a set of classes and methods that define a generic interconnection network and its behaviour. This core also includes the classes and methods for packet description and behaviour. The user interface is the program layer above the core: it interfaces the user and the simulator core and provides a graphical interface to show simulation development and results, allowing data input and interactive/batch simulation specification.

In the C++ implementation, these two blocks are clearly separated in different files: full core specification is located in files `red.h` and `red.cpp`, while the user interface source code has the name of the application (`simured.cpp` for the visual version and `simured_cmd.cpp` for the command-line version). The core has been designed to be portable between platforms and operating systems: it compiles under *gcc* (Windows or Linux) and under Borland C++ compiler. Two different user interfaces are provided: one is graphical and sophisticated, it has been written for Borland C++ Builder; the other user interface works in command-line mode, but has the same functionality as the other (non visual, though) and the code is easier to understand; it serves as an example for simulator core integration in any other tool.

The differentiation between the user interface and the simulator core is not as evident in the Java version since each class has its own file. Basically, the whole core is mainly comprised of the *Red* and the *Packet* classes. The core of the Java version is exactly the same as the C++ version, the differences between the Java and C++ versions are to be found in the user interface. There is not yet a Java command-line version, but there is a reduced Java Applet version directly accessible from the SimuRed web page.

There are several advantages of using object-oriented languages for hardware modelling, among these every hardware component can easily be specified using a class, and its behaviour and interaction with other elements can be modelled by its class methods.

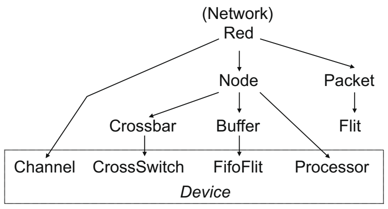

Fig. 3. Simulator classes and class hierarchy.

Fig. 3 shows the classes and class hierarchy of the SimuRed core components. At the highest level there is the *Red* class (Network). This class implements the statistical counters and the node, channel and packet lists. It has several methods that model network behaviour, but the most important is the *RunCycle()* method, which advances the simulation time by one cycle performing all the necessary changes to the packet positions.

The network is created by interconnecting the set of channels and nodes. The *Channel* and *Node* classes define the behaviour and constants of the network channels and nodes. Each node has one processor, several input/output buffers and a crossbar; these components are defined in the *Processor*, *Buffer* and *Crossbar* classes, respectively. The crossbar is further divided into several switches coded in the class *CrossSwitch* and the buffers are divided in flits defined in the class *FifoFlit*. All lowest level network components (CrossSwitches, FifoFlits, Processors and Channels) share properties with the common class *Device*.

The *Packet* forms part of the network though its behaviour and creation is dynamic and takes place outside the network. This class has its own *RunCycle()* method whose fuction is to move the corresponding packet instance one cycle. In fact, the *RunCycle()* method of the *Red* class calls the *RunCycle()* methods of all packets present in the network. This is one of the special characteristics of SimuRed: its physical components do not perform any action during simulation; it is the packet itself which moves depending on the static behaviour of the components it is going through. This makes simulations run faster than in other approaches, making it feasible to simulate large systems in a few minutes.

3.1. Graphical user interface

SimuRed offers interactive simulation: the program shows the network structure and the packet traffic at the flit-level during run time. The simulation can be stopped at any time and continued step by step. The graphical user interface also allows batch execution by specifying a set of multiple simulations. Up to two nested parameters can be specified in a multiple simulation; the inner parameter is always the delivery rate of packets, while the optional outer parameter can be the number of dimensions, nodes per dimension, buffer length or number of virtual channels. A multiple simulation generates a chart that can be viewed immediately using SimuRed's own chart presentation tool. These results are also stored in standard CSV text format for further processing using any other external tool.

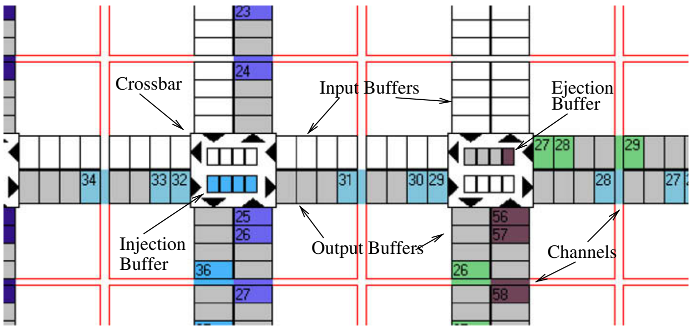

Fig. 4. Simulator snapshot showing network elements and numbered packet flits; each packet has a different colour. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

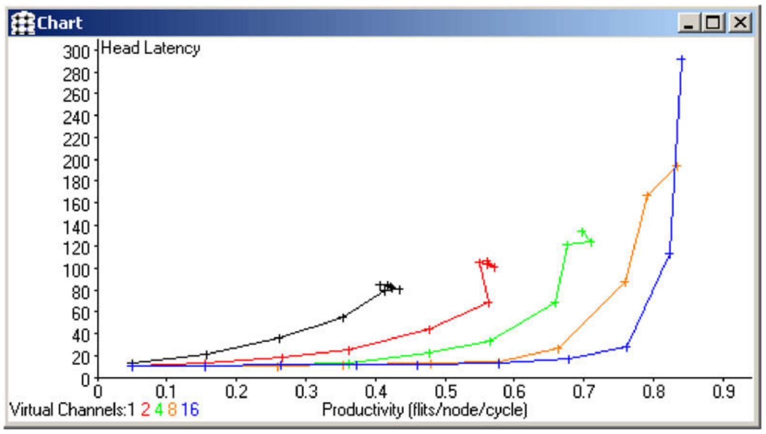

Fig. 5. Simulator chart window showing head latency for different productivities and virtual channels. (Different settings are shown using different colours in the simulator.) (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

One of the most interesting features of SimuRed is its ability to show the network state and the packet motion at any time. Fig. 4 shows a slice of the simulator network window with some of the network components and packet state. This figure shows two nodes of the network already shown in Fig. 1. Each packet has a different, random colour and each packet flit has a number starting at one for the head flit. Network visual explorations, state visualization and step by step execution are necessary features to new routing algorithm debugging. These visual features are also interesting for educational purposes.

Fig. 5 shows the simulator chart window for a specific run as an example of chart drawing and multiple batch processing. In this case, the average packet latency of the order dimension algorithm for a 4×4 mesh is represented against packet traffic (productivity expressed as flits per node per cycle) for each different number of virtual channels (1, 2, 4, 8 and 16). Each curve corresponds to a different number of virtual channels and is represented using a different colour, since the output is intended for a colour monitor.

4. Traffic generation

A single simulation involves a number of packets being injected into the network. At the arrival of each packet, the statistical counters are updated and at the end of the simulation the results are shown. The most common way to evaluate performance is to plot the mean packet latency for several packet delivery rates up to network saturation. Packet source and destination nodes are randomly selected from among all the nodes in the network. Random packet generation follows an arrival Poisson process. Other packet traffic and patterns, including real application traffic, can be simulated specifying a trace file.

Packet latency depends on the network workload at a specific moment; if the network is full of packets, new packets will probably get blocked, increasing their own latency. Network occupancy may increase over time, until it reaches a stable state in which packet latency only depends on network static parameters. This is usually the state required for most simulations; therefore, before calculating any statistical value, the stationary condition must be satisfied. The common way to ensure network stabilization is to inject a specific number of *warm-up* packets before making any other calculations. Another option is to track packet latency until its average value hardly changes, but this can lead to errors since packet latency is not uniform due to path length non-uniformity, especially for large networks.

SimuRed can keep track of statistical counters during simulation. This transient simulation allows the user to perform dynamic simulations to see how the statistics change throughout the simulation time. The main use of this option is to calculate the instant at which the network stabilises. Thus, before carrying out a simulation, it is recommended to perform a dynamic simulation at a very high injection rate; this allows one to see two important parameters for further simulations: the first parameter is the network saturation delivery rate and the other is the number of dummy warm-up packets. Other interesting purposes of this dynamic simulation is to study non-uniform traffic arrival rates, like bursty traffic or those found in real applications, especially in multimedia programs [30].

The SimuRed user interface allows the user to produce a single simulation run selecting the packet number, packet length, header length, delivery rate, and the number of warm-up packets. The user can also specify a set of simulations from a minimum arrival rate to a maximum (usually close to the saturation point) and a number of points in between. Most charts in this article have been generated this way in a single run.

4.1. Testing real packet traffic

Dynamic network testing is becoming increasingly important since it has been shown that real programs tend to send burst of packets in time and space. Uniform traffic simulations can tell little about network behaviour under these circumstances. In fact, some analytical models have already been proposed to deal with such a burst [10,31] and hot-spot traffic [32].

SimuRed can read a file with the list of packets to be delivered. This file is in fact a trace file that could have been generated from a real application. The format of the trace file is text: each line represents a new packet defined by up to five fields. These fields specify the clock cycle at which the packet is delivered or the dependent packet identification, the packet source and destination node, the packet length and the unique packet identification. The dependent packet and the unique packet identifications are good to specify that a packet must be sent only when the dependent packet arrives to its destination.

Any complex traffic pattern can be generated using a program to write a trace file, though the most interesting option is to simulate real traffic like that generated by a parallel program. In this case, the ability to define dependent packets is fundamental, since many inter-process communications are serialized due to the pure underlying algorithmic nature of programs. Dependent packets are also necessary to carry out simulations of the *ping–pong* type, which are important to some network performance indicators.

5. Simulator performance

One of the most criticised aspects of event-driven simulators is the time required to complete a run. The flit-level simulator consumption time is usually longer than the time required by an analytical model to obtain results. Nevertheless, the flit-level simulator offers accurate results and can perform more sophisticated simulations. The simulation execution time of SimuRed has been measured. This time, even using an average desktop computer, is not very long and is comparable to the time required by some analytical models.

All the experiments in this section have been performed using a Pentium IV desktop computer running at 2.54 GHz; which falls within the average of low-performance desktop computer and is below any simple lab workstation. Two- and three-dimensional meshes have been chosen to carry out the experiments. The routing mechanism has been the deterministic dimensional order for meshes, with no virtual channels, 2-flit buffer length and 32-flit packet length. The packet number was 200.000 for all experiments; this number is common for simulations of up to a few hundred nodes. The injection rate was also the same for all the experiments: 0.3 flits/node/cycle; this rate is large enough to ensure network saturation, which represents one of the worst cases for simulation speed. Simulation time measurements are summarized in Fig. 6.

The most time-consuming experiment was that performed with a 65.536 node two-dimensional network. In this case, the time required to complete the simulation was 82 min. Most of the experiments shown in literature are for systems with tens or hundreds of nodes; SimuRed takes from 1 to 10 min to perform a single simulation run for networks with even more than one thousand nodes using a modest desktop computer. These results contrast with those published in [2] where the authors compared their analytical model with a flit-level simulator; their analytical model took 13 min to complete while their flit-level simulator took 1077 min (almost 18 h!). The authors used a state-of-the-art workstation (year 2000–2002). The same experiment has been repeated with SimuRed running in a desktop computer: 2-D mesh of 63×63 nodes, with 32-flit messages and 8 runs for several network loads up to saturation. SimuRed took 50 min which is 4 times slower than the analytical model, but 20 times faster than the flit-level simulator these authors used. A total of 60,000 packets were used on each run: 30,000 warm-up packets and 30,000 more for real data analysis. The performance of a desktop computer is probably not far from a good workstation 4–6 years older, so it is the simulator, more than the platform, the guilty for the performance improvement. The reason for the higher performance of SimuRed come from the strategy used for driving the process: while in most flit-level simulators the nodes drive the program execution, in SimuRed the packets drive the program execution. In SimuRed only those elements containing a packet flit are processed, while in other simulator all elements in the network execute some code even when they are idle.

Simulation time depends on many factors. In Fig. 6 the time has been plotted against the network node number, since the network size is the parameter with the greatest impact on simulation time. Both time and node number axes are logarithmic to show an almost linear dependency between logarithms of time and node number. This linear dependency is especially noticeable in the 2-D mesh experiment. In this case we have performed several experiments and we have found an empirical formula that roughly modulates the simulation time in 2-D meshes:

$$\text{seconds} \approx \text{nodes}^{0.8} \tag{1}$$

Since the exponent in (1) is lower than 1, the simulation time scales well with the network size. In fact, this time seems to depend on the packet average path; this is the reason why the time decreases for the 3-D mesh in large networks compared to the 2-D mesh, as shown in Fig. 6.

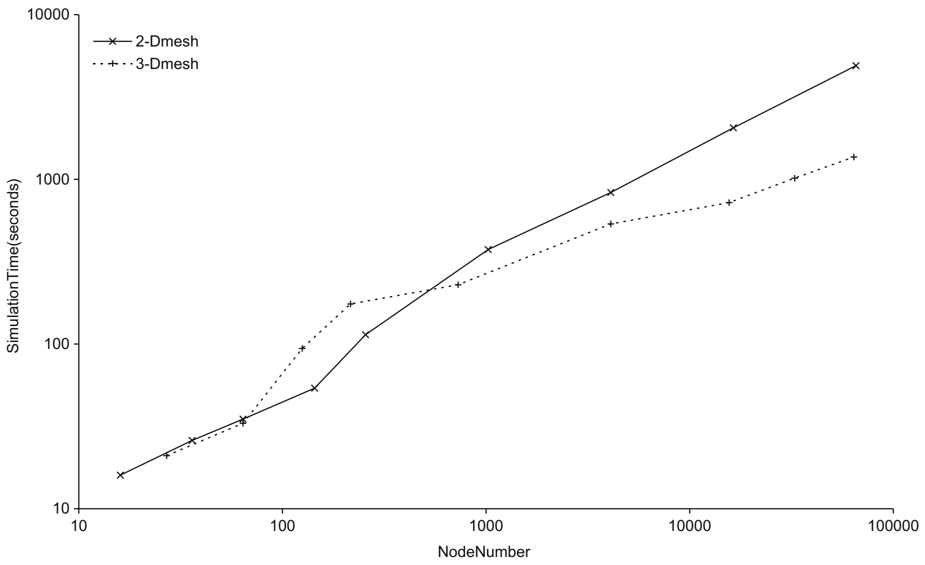

Fig. 6. Simulation time in seconds for several network node counts and network dimensions (2 and 3).

6. Network evaluation using SimuRed

SimuRed can enable many experiments and studies. In this section we analyze the impact of injection channels on stationary network performance. We also compare deterministic versus adaptive routing and the impact injection channels have on this comparison. This study was chosen since the impact of injection/ejection channels on performance is something SimuRed is capable to analyze unlike to other simulators and models. The study has been performed for two common network topologies, namely meshes and hypercubes. These networks have quite different performance behaviour depending on the routing algorithm type as shown next.

6.1. Mesh network performance

The experiment involves the packet latency measure for different numbers of injection channels using deterministic and adaptive routing in a 2-D mesh network. The experimental results have been obtained for an 8×8 2-D mesh (64 nodes), 32-flit packets, 2-flit buffers and two virtual channels. There were 4 ejection channels to avoid packet arrival bottleneck. The deterministic routing algorithm was the dimensional order routing. Duato's fully-adaptive algorithm for meshes was chosen for the adaptive experimental set up.

Fig. 7 shows the simulation results for the deterministic routing. The injection channel addition has a negative effect on deterministic routing since the packet latency increases, although the maximum delivery rate remains almost constant. Deterministic routing imposes limitations on the degree of freedom of the packet; if there are more injection channels than packet escape ways, it is very likely that packets will stay in the injection buffer for a long time before entering the network. This is the reason why adding injection channels usually increases packet latency, in fact, this extra latency is due to the packet waiting time in the injection buffer. Network maximum throughput remains constant since extra packets are stopped in injection channels and do not over-saturate the network.

Fig. 8 shows the same simulation but using adaptive routing. For this routing type, packet latency changes little with respect to the number of injection channels, but the maximum delivery rate decreases when using several injection channels. Network saturation is worse for adaptive routing than for deterministic routing since packet latency increases a lot and the maximum delivery rate decreases slightly. Adaptive routing allows a higher degree of freedom for packet routing. This freedom in routing makes it easier to inject packets in the network, even at higher rates than the network can support. In the stationary situation the network becomes saturated and the probability of packet blocking increases up to the point where it becomes difficult to reach the destination node; in this situation packet latency increases dramatically while the maximum throughput decreases. This saturation state is shown in the charts as spikes and non-linearities around high delivery rates. The good thing about adaptive routing is that it can handle several concurrent injection buffers without performance degradation; the problem comes when traffic is so high that it over-saturates the network and latency becomes unpredictable.

Fig. 9 shows a comparison between adaptive and deterministic routing for 1 and 4 injection channels in meshes. Performance of adaptive and deterministic routing is almost the same when one single injection channel is used. This similar performance of adaptive and deterministic routing in stationary 2-D mesh network analysis is not strange and has already been reported by other authors [33]. Nevertheless, there is a noticeable change in performance when more than one injection

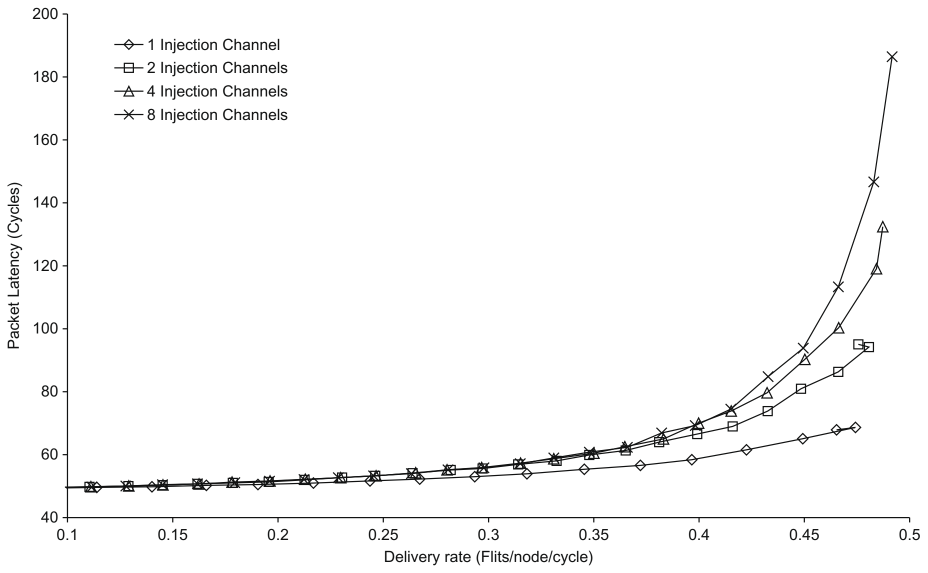

Fig. 7. Packet latency for different numbers of injection channels using deterministic routing in a mesh network.

channel is used: while adaptive routing performance remains almost unaltered for low and medium traffic, deterministic packet latency increases as shown in the figure. The cost of adaptive routing in this case is the degradation of network maximum throughput and a dramatic increase in packet latency for heavy traffic.

6.2. Hypercube network

The same experiment has been repeated for a 64-node hypercube ($n = 6$, $k = 2$). The node number is the same as the mesh, but the connectivity capabilities and cost of the hypercube are much greater than those of the mesh.

Fig. 10 shows the packet latencies for different number of injection channels and routing algorithms. The first remarkable result is the different behaviour for the number of injection channels in deterministic and adaptive routings. In fact, this behaviour is the same reported for the mesh experiment: adaptive packet latency changes little with the number of injection channels, but when packet traffic becomes very high (above the network bandwidth), the network saturates and the maximum delivery rate decreases while packet latency dramatically increases. There is a difference though; packet maximum

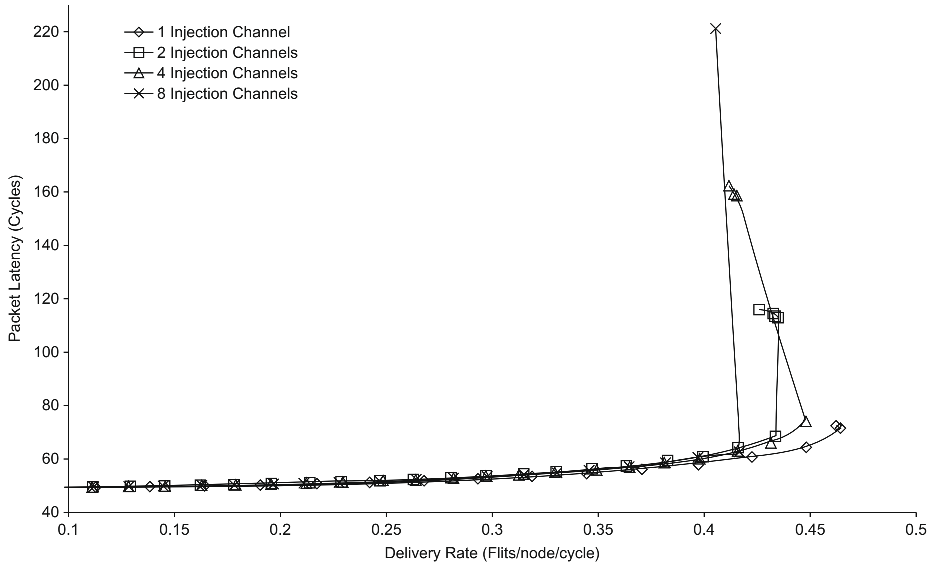

Fig. 8. Packet latency for different numbers of injection channels using adaptive routing in a mesh network.

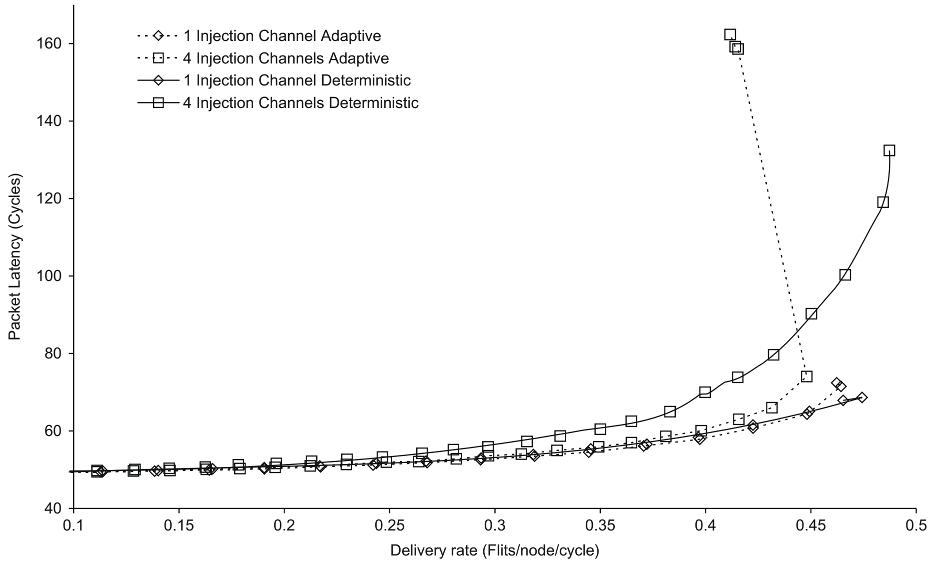

Fig. 9. Comparison between adaptive and deterministic routing for 1 and 4 injection channels in a mesh network.

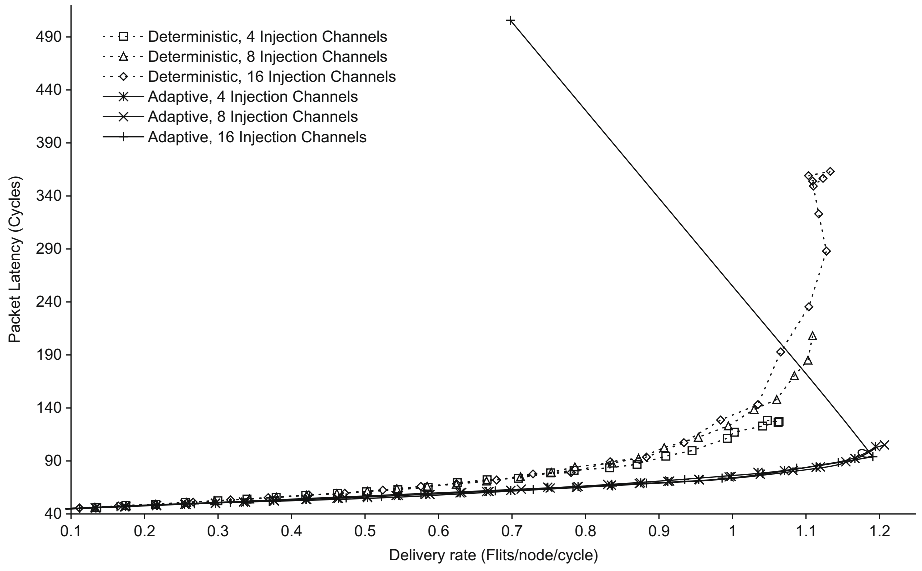

Fig. 10. Comparison between adaptive and deterministic routing for 4, 8 and 4 injection channels in a hypercube network.

delivery rate changes little with the number of injection channels; only when this number is very high the net over-saturates and the maximum delivery rate decreases. Deterministic routing performance dependency on injection channel number demonstrates the same behaviour as that shown for meshes: the higher the number of injection channels, the worse the performance, though maximum delivery rate increases a little.

The other important result shown in Fig. 10 is the noticeable difference in performance between adaptive and deterministic routing in hypercube networks. While in the mesh network the difference in performance was not very high, this difference is especially visible in the hypercube network. Not only is packet latency lower in all cases, but also adaptive routing shows larger throughput with the exception of a very large number of injection channels. These results comparing packet latency and maximum throughput in adaptive and deterministic routing are in accordance with results of similar experiments [34].

7. Conclusions

A multicomputer network simulator has been described, whose visual/interactive user interface makes it especially suited for new routing algorithm debugging, and among other purposes also in education. The simulator is simple to use and its Java or C++ implementations are easy to change allowing researchers to tailor the simulator core to fit their custom systems. SimuRed can perform static and transient network simulations and it has the ability to cope with real-traffic traces. The simulation is driven by the packet flits instead of the network physical components; this makes simulations run faster than in other approaches. In fact, SimuRed performance allows simulation of hundreds of nodes in the order of minutes, making it comparable, though always slower, to most of analytical models. SimuRed helps in the design of many interconnection networks, not only for modern parallel systems, but also for systems based on new trends as the Network on Chip (NoC) technology.

Some experiments have been carried out to study the performance impact of injection channels and deterministic/adaptive routing in two-dimensional meshes and hypercubes. These experiments have shown a small difference in performance when comparing adaptive and deterministic routing in a 2-D mesh. Nevertheless, this difference is especially noticeable in hypercubes, where adaptive routing performs better than deterministic one. These experiments have also shown that the number of injection channels can play an important role in network performance. In deterministic routing the larger the number of injection channels, the worse the performance. In adaptive routing, the number of injection channels has a little impact on performance, except when it is very high and the network over-saturates, increasing the packet latency while decreasing the maximum delivery rate.

References

[1] Ould-Khaoua M, Loucif S, Rabhi FA. On the performance of multicomputer interconnection networks. J Syst Architec 2004;50:563–74.
[2] Gregorio JA, Beivide R, Vallejo F. Modeling of interconnection subsystems for massively parallel computers. Perform Eval 2002;47:105–29.
[3] Ould-Khaoua M. A performance model for duato's fully adaptive routing algorithm in *k*-ary *n*-cubes. IEEE Trans Comput 1999;48(12):1297–304.
[4] Najaf-abadi HH, Sarbazi-azad H, Rajabzadeh P. Performance modeling of fully adaptive wormhole routing in 2-D mesh-connected multiprocessors. In: 12th annual international symposium on modeling, analysis, and simulation of computer and telecommunications systems (MASCOTS'04). Washington, DC, USA: IEEE Computer Society; 2004. p. 528–34.

[5] Chlamtac I, Ganz A, Kienzle MG. A performance model of a connection-oriented hypercube interconnection system. Perform Eval. 1996;25(2):151–67.
[6] Colajanni M, Ciciani B, Tucci S. Performance analysis of circuit-switching interconnection networks with deterministic and adaptive routing. Perform Eval. 1998;34(1):1–26.
[7] Rexford J, Feng W, Dolter J, Shin KG. PP-MESS-SIM: a flexible and extensible simulator for evaluating multicomputer networks. IEEE Trans Parall Distri Syst 1997;8–1(1):25–40.
[8] Rexford J, Hall J, Shin KG. A router architecture for real-time communication in multicomputer networks. IEEE Trans Comput 1998;47(10):1088–101.
[9] Duato J, Yalamanchili S, Ni L. Interconnection networks, an engineering approach. USA: IEEE Computer Society Press; 1997.
[10] Min G, Ould-Khaoua M. Performance modelling and evaluation of virtual channels in multicomputer networks with bursty traffic. Perform Eval. 2004;58(2+3):143–62.
[11] Nnssle M, Frning H, Brnning U. SWORDFISH: a simulator for high-performance networks. In: IASTED conference: parallel and distributed computing and systems (PDCS). Phoenix, AZ, USA; 2005.
[12] SWORDFISH web page. <http://ra.ziti.uni-heidelberg.de/index.php?page=projects&id=swordfish>.
[13] Dally WJ, Towles BP. Principles and practices of interconnection networks. Morgan Kaufman; 2003.
[14] Petrini F, Vanneschi M. SMART: a simulator of massive architectures and topologies. In: International conference on parallel and distributed systems. Barcelona, Spain; 1997.
[15] García JM, Sánchez JL, González P. PEPE: a trace-driven simulator to evaluate reconfigurable multicomputer architectures. Lect Notes Comput Sci 1996;1184:302–11.
[16] Pertel MJ. A simple simulator for multicomputer routing networks. Technical Report CaltechCSTR:1992.cs-tr-92-06. Pasadena, CA, USA: California Institute of Technology; 1992.
[17] Mudge T. Report on the panel: how can computer architecture researchers avoid becoming the society for irreproducible results? SIGARCH Comput Architect News 1996;24(1):1–5.
[18] Wang H-S, Zhu X, Peh L-S, Malik S. Orion: a power-performance simulator for interconnection networks. In: 35th annual ACM/IEEE international symposium on microarchitecture, MICRO'02. Los Alamitos, CA, USA: IEEE Computer Society Press; 2002. p. 294–305.
[19] Terzis A, Nikoloudakis K, Wang L, Zhang L. Irlsim: a general purpose packet level network simulator. In: ACM-SIAM 33rd annual simulation symposium; 2000. p. 109.
[20] Zheng G, Kakulapati G, Kalé LV. BigSim: a parallel simulator for performance prediction of extremely large parallel machines. In: 18th international parallel and distributed processing symposium (IPDPS). Santa Fe, New Mexico; 2004. p. 78b.
[21] Zheng G, Wilmarth T, Jagadishprasad P, Kalé LV. Simulation-based performance prediction for large parallel machines. Int J Parall Programm 2005;33(2–3):183–207.
[22] Wilmarth TL, Zheng G, Bohm EJ, Mehta Y, Choudhury N, Jagadishprasad P, et al. Performance prediction using simulation of large-scale interconnection networks in pose. In: 19th workshop on principles of advanced and distributed simulation, PADS'05. Washington, DC, USA: IEEE Computer Society; 2005. p. 109–18.
[23] Dally WJ. Performance analysis of *k*-ary *n*-cube interconnection networks. IEEE Trans Comput 1990;39(6):775–85.
[24] Laudon J, Lenoski D. The SGI origin: a ccNUMA highly scalable server. In: 24th international symposium on computer architecture; 1997. p. 241–51.
[25] Noakes M, Dally WJ. System design of the J-machine. In: Advanced research in VLSI. MIT Press; 1990. p. 179–92.
[26] Kessler RE, Schwarzmeier JL. CRAY T3D: a new dimension for cray research. In: CompCon; 1993. p. 176–82.
[27] Anderson E, Brooks J, Grassl C, Scott S. Performance of the cray T3E multiprocessor. In: ACM/IEEE supercomputing conference; 1997. p. 1–17.
[28] Blumrich M, Chen D, Coteus P, Gara A, Giampapa M, Heidelberger P, et al. Design and analysis of the BlueGene/L torus interconnection network. In: Technical report, IBM Research Division, T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY; 2003. p. 10598
[29] Duato J. A new theory of deadlock-free adaptive routing in wormhole networks. IEEE Trans Parall Distrib Syst 1993;4(12):1320–31.
[30] Orduna JM, Duato J. A high-performance router architecture for multimedia applications. In: Fifth international conference on massively parallel processing using optical interconnections (MPPOI'98). Silver Spring, MD: IEEE Computer Society Press; 1998. p. 142–9.
[31] Min G, Ould-Khaoua M. Prediction of communication delay in torus networks under multiple time-scale correlated traffic. Perform Eval. 2005;60(1–4):255–73.
[32] Ould-Khaoua M, Sarbazi-Azad H. An analytical model of adaptive wormhole routing in hypercubes in the presence of hot spot traffic. IEEE Trans Parall Distrib Syst 2001;12(3):283–92.
[33] Pertel MJ. A critique of adaptive routing. Technical report CaltechCSTR:1992.cs-tr-92-04. Pasadena, CA, USA: California Institute of Technology; 1992.
[34] Najaf-abadi HH, Sarbazi-Azad H. Comparative evaluation of adaptive and deterministic routing in the OTIS-hypercube. Lect Notes Comput Sci 2004;3189(ACSAC'2004):349–62.