Speeding Up Exascale Interconnection Network Simulations with the VEF3 trace Framework

Javier Cano-Cano[a,*], Francisco J. Andújar[b], Francisco J. Alfaro[a], José L. Sánchez[a]

[a] *Computing System Department, University of Castilla-La Mancha, 02071, Albacete, Spain*

[b] *Computing System Department, Universidad de Valladolid, 47011, Valladolid, Spain*

[*]Correspondence to: Instituto de Investigación en Informática de Albacete, Campus Universitario, 02071, Albacete, Spain.

Email addresses: `javier.cano@uclm.es` (Javier Cano-Cano), `fandujarm@infor.uva.es` (Francisco J. Andújar), `fco.alfaro@uclm.es` (Francisco J. Alfaro), `jose.sgarcia@uclm.es` (José L. Sánchez)

Abstract

Simulation is used to evaluate and validate the behavior and performance of computing systems, in particular the interconnection network in the context of high-performance computing. For the simulation to be performed, the simulator program must be provided with a mechanism that generates network traffic or workload. Although synthetic traffic has been widely used, communication from real applications is a better and more representative workload. With this kind of network workload, the simulations can become slower, especially when simulating Exascale systems.

In this paper, we extend the VEF trace framework, originally designed for feeding off-chip networks with MPI traffic, including new functionality related to the on-chip communications and introducing improvements to speed up the simulations. This way, the VEF framework allows to study the behavior of Exascale interconnection networks with realistic traffic and in reasonably short times.

Keywords:

Interconnection networks, Modeling and simulation tool, Exascale workloads, Performance evaluation

1. Introduction

As it is well known, the interconnection network is a crucial subsystem in high-performance computing (HPC) systems and data-centers (DCNs). In a lot of these systems, to break the Exascale barrier in a few years is expected. The world's most powerful supercomputer is the Summit [1] from DOE/SC/Oak Ridge National Laboratory, USA, which offers 201 PFLOPS of peak performance. On the other hand, the International Data Corporation (IDC) predicts that the Global Datasphere will grow from 33 Zettabytes (ZB) in 2018 to 175 ZB by 2025 [2], which will make the requirements for storage devices but also skyrocket. Although it is foreseeable that the number of systems increases, it is does not do so in the same proportion as the amount of data, and therefore the bandwidth requirements to access that data will also be greater.

In this scenario, the performance of the interconnection network is crucial, otherwise it would become the system bottleneck. Therefore, new design proposals of the main aspects characterizing the interconnection network must guarantee low latency overhead and high communication bandwidth. Obviously, sometimes it is impossible to physically implement and test those new proposals, so they must be evaluated and validated using software-based tools, such as interconnection network simulators.

Interconnection network simulators have traditionally used synthetic traffic as network workload. However, synthetic traffic is not the workload that real multiprocess applications generate. In the last decades, the use of application communication traces is becoming a common strategy to generate the network workload and evaluate the system performance.

The first step regarding application communication traces is to obtain the trace files. A profiling tool is required, as well as an adequate trace format. VEF framework [3] defines a trace format and is capable of obtaining and replaying trace files. VEF traces contain every communication an MPI application generates during its execution. A fundamental characteristic of VEF traces, which makes them different from the most trace systems, is that they contain information about the relationship between communication messages. Moreover, VEF traces can be used in any network simulator with minimal effort.

Initially, the VEF framework was aimed for MPI communications, which means that it was intended for off-chip interconnection networks. However, nodes in Exascale systems will be composed of multiple cores, which allows applying hybrid programming [4] so that the message passing model coexists with the shared memory model. In this scenario, the applications will generate two types of messages: those due to the data transfer between nodes, and those generated by the cache coherence protocols within the nodes. Therefore, the communication trace files obtained when running these applications should collect both the traffic between the nodes (off-chip network) and the traffic between the cores in each node (on-chip network, NoC).

Authors presented an extension to the VEF framework that allows to also collect the messages generated on the on-chip network [5]. At the moment, the framework can be used to obtain and manage trace files from both off-chip and on-chip networks, separately. The next step will be obtaining both types of traffic at the same time when the applications generate them.

In this paper, further details on the new characteristics and improvements of the framework are provided. Thus, it is explained how the framework has been extended to reduce the execution differences with respect to the previous one [5], and how intra NoC tile latencies and memory auto-mapping have been introduced. Moreover, more details are given on all the updates made to previous VEF trace format, the problems that led us to update that format and a comparison with the new trace format in terms of speed up. The related work has been extended and the evaluation section significantly improved: new simulation scenarios, new updates in Gem5 full-system, more results and performance metrics, and a discussion about how NoC trace files combined with MPI trace files may feed Exascale simulators.

The rest of this paper is organized as follows: Section 2 reviews the related work regarding simulation, trace files and tools. The VEF trace framework is explained in Section 3, in which we talk about the VEF trace format (Section 3.1), the profiling tool (Section 3.2) and the trace-replaying library (Section 3.3). Section 4 shows the results obtained evaluating the trace model accuracy and, finally, Section 5 presents some conclusions.

2. Related work

Simulation is commonly used to evaluate, verify and validate the behavior and the performance of computing systems. Simulation can also help to fine tune new techniques and proposals before implementing them in a real system, significantly reducing its cost, since the cost of simulating is significantly lower than the one of manufacturing a real design.

In the particular case of interconnection networks, during the last years extensible, open-source frameworks have been developed, such as NS-3 [6], OMNeT++ [7], SimGrid [8] or SST [9]. Some research groups have developed simulators based on these frameworks, such as Sauron [10], which is based on the OMNeT++ framework. Other research groups have developed their own network simulators from scratch, such as BookSim [11] or Topaz [12].

Other efforts have been focused on the development of tools to save time in the development, debugging and evaluation of network models, such as the RAAP HPC tools [13].

An essential aspect in these simulators is how to provide the network workload. Currently, the traffic generated by the execution of real applications, and recorded in trace files, is preferred to synthetic traffic.

Trace-driven simulation is not a recent topic in the HPC field, and there exist several tools for capturing the traffic generated by applications. For instance, MPI incorporates the profiling message interface (PMPI) to facilitate the profiling and tracing of the MPI applications.

There are other instrumentation tools to profile MPI applications such as VampirTrace [14], TAU [15], PARAVER [16] or Score-P [17]. These instrumentation tools are able to record every message sent from the processes and to store the information following a well-defined format.

In the context of network on chips (NoCs), there are several simulators proposed in the literature. Nostum [18] is a flexible NoC simulator focused on communication primitives along loss-less switches, and it implements the protocol stack for the link, the network and the session layers. Nirgam [19], Noxim [20] and Sicosys [21] are general-purpose interconnection network simulators for multiprocessor systems. These simulators accurately model a wide variety of router architectures and offer a lower computational cost, achieving modularity, versatility and connectivity with other systems. Unfortunately, these simulators obtain results using synthetic traffic or traditional non-self-related trace files. In fact, the main shortcoming of these simulators is that they do not simulate other components involved in a chip multi-processor (CMP) architecture, as the memory system, the processing elements or the interactions with the NoC. This means that these simulators offer lower accuracy.

Another well-known NoC simulator is Garnet2.0 [22], included in the Gem5 [23] full-system simulator. It provides different operation modes that differ in microarchitectural details of the modelled on-chip router. Although Garnet2.0 is an excellent tool due to its accuracy and inter-operability with Gem5, it has some problems of scalability, a limited standalone functionality and a reduced number of topologies. Furthermore, Gem5 is very slow because of the large number of simulated elements. Gem5 can capture memory traces using Google protobuf encoding. Multiple works have been published referring memory traces such as Elastic traces [24] and ElasticSimMATE [25].

Memory traces are very useful for memory researching, but the implementation of a cache coherence protocol is compulsory to replay these traces on an arbitrary NoC simulator. This means that the simulations will take more time to finish and the system will be more complex. More details about memory traces and VEF traces are included in Section 2.1.

Netrace [26] uses a communication library that captures the dependencies at system memory level into a trace file. However, the NoC is not simulated when the trace file is generated and, therefore, the dependencies do not include all interactions among the different components of a CMP system.

Moreover, these trace files are very limited to a particular processor and memory system configuration.

Finally, regarding the trace file generation, there are a few available tools. SynchroTrace [27] is an architecture-agnostic tool able to capture the interactions between threads in a running application, taking into account almost all the system components. Nevertheless, SynchroTrace requires a complex trace replay framework including an Event Queue Manager, Thread Scheduler, etc. Moreover, SynchroTrace works at thread level, meaning that these traces are not directly source/destination node traces.

In our opinion, previous trace files do not have the flexibility required to be replayed on an arbitrary simulator. For this reason we created our own trace file format, called VEF, and VEF-TraceLib, an open-source library written in ANSI C, responsible for trace reading, dependency handling, task execution management, and capable of feeding any network simulator [28, 3]. Some integration examples of VEF-TraceLib can be found in [29, 30, 31].

However, applications usually used to measure performance on NoCs do not use MPI as communication library. Even if these applications use MPI, to create/modify a communication-library-agnostic tool is a better approach.

We can take advantage of all existent VEF infrastructure for supporting NoC simulators. We have a system-agnostic trace format and a trace-replay library. Hence, we only need to get representative traces of NoC traffic, in order to support on-chip and off-chip communications in the same framework.

2.1. VEF traces vs. memory traces

Gem5 includes a tracing mechanism called Elastic Trace, which captures data and load/store order dependencies instrumenting a detailed out-of-order processor model called *DerivO3CPU*. Elastic Trace is capable of tracking dependency information inside the processor pipeline. Gem5 also includes the Trace CPU model which plays back Elastic Trace files. The execution time average error of Elastic Trace is between 7% and 17 % [24]. However, the Elastic Trace system average speed up is 8x. The main drawback is that to extract traces from multithread applications is not allowed. However, one of the biggest advantages is that the configuration parameters may be modified without losing more than a 17% of precision in terms of execution time.

In order to solve the multithread applications problem, ElasticSimMATE framework has been presented. The tool is compatible with both OpenMP 3.0 and POSIX thread APIs. Recording synchronization traces requires using a specific Gem5 pseudo-instruction created for this purpose. This pseudo-instruction requires to be inserted either manually or automatically by means of an instrumented runtime system. These traces can be very useful for those studies focusing on memory-system exploration, achieving an average speed up of 5x [25]. The Elastic trace files are fast, accurate and portable.

On chip network simulators are very complex, since simulate a wide variety of elements (e.g. on chip routers, links, routing protocols, power consumption models, etc.) and in a very detailed way. Therefore, the simulation time is very high. To replay Elastic trace files on an arbitrary NoC simulator, the implementation of a cache coherence protocol is required, further increasing the simulator complexity and simulation time. Furthermore, not every communication included in the memory traces will end up in a network packet. This information is useless for a NoC simulator, and will increase the execution time and the final trace size. Memory access traces such as Elastic traces are not a good option for NoC simulators, and a more specific mechanism is required to obtain the traces.

VEF3 traces do not require a complete cache coherence protocol, and moreover, the users do not need to know how these protocols work. These trace files only include communications that could end up in a message through the NoC. Therefore, VEF3 traces are a better option for NoC simulators than Elastic Traces or any other memory traces. Finally, as far as we know, the only implementation of Elastic Traces has been done in Gem5. This could mean that its integration in another simulator will be a time consuming task.

3. VEF3 trace framework

We have developed a set of tools that allow us to obtain trace files containing the communication generated by applications running in large computing systems, and to use them as workload in interconnection network simulators. Initially, the VEF framework was aimed to MPI communications, and so it was intended for off-chip networks. We have extended the framework functionality and now to capture the communication generated by several threads of the application running inside a computing multicore is also possible.

The main components of the framework are a profiler to obtain the traces, a library to manage them and the trace format. All of them have had changes up to the current version. For instance, the first trace format version (VEF) only included point-to-point MPI communications, while collective MPI communication was added in the second version (VEF2). The new VEF3 trace format includes new message dependencies for speeding up the simulations. In the following sections we offer details of the VEF framework, paying special attention to the new improvements presented in this document.

3.1. VEF3 Self-Related traces

The VEF3 trace format is based on the VEF2 trace format [28]. Although VEF traces were originally designed for MPI profiling purposes, modeling the cache hierarchy communications is also possible using the VEF2 trace format (see Section 3.1.5). Not only that: since we have developed a library [28] for replaying VEF traces (VEF-TraceLib), we can use the same library to replay both MPI traces and NoC traces. This way, we have extended the VEF trace format to support on-chip communications, preserving the compatibility with MPI VEF traces. A VEF3 trace is composed of two files, a *.vef* file (Section 3.1.1) and a *.names* file (Section 3.1.2).

3.1.1. Vef files

A VEF3 trace is a plain text file containing all the information needed to model the communications between different MPI tasks or different elements of the cache hierarchy. A trace comprises multiple records, being each line an independent record. We can distinguish three kinds of records into the traces: the trace header, the communicators, and the communication records.

1. **Trace header.** This is the first line in the file and contains basic trace information. Its format is the following:

   VEF3 *nNodes nMsgs nCOMM nCollComm nLocalCollComm noRecvDep clock*

   where:

   – **VEF3** indicates the VEF trace format version.

- *nNodes* is the total amount of memory devices (L1 caches, L2 caches, directories, etc) in NoC traces[1]. In MPI traces, this field indicates the total number of MPI tasks.
- *nMsgs* is the number of point-to-point communication records.
- *nCOMM* is the number of MPI communicators. Since the communicators are not necessary for NoC traces, this field is set to "1" and we only create a virtual communicator for all memory devices.
- *nCollComm* is the number of global collective communications, only used for MPI traces and set to "0" for NoC traces.
- *nLocalCollComm* is the number of local collective communications, only used for MPI traces and set to "0" for NoC traces.
- *noRecvDep* is a deprecated parameter maintained for compatibility.
- *clock* is the clock resolution measured in picoseconds. Since the trace time fields are measured in a full-system simulator in cycles, is necessary to calculate the cycles on an arbitrary NoC simulator. This field has been included in VEF3[2].

2. **Communicators**, or COMMs, indicate communicators in the MPI trace. They are not necessary in shared-memory communications, and for compatibility reasons, we add to each NoC trace a virtual communicator with all possible destinations. Its format is the following:

$$C0\ element_0[element_1...element_{n-1}]$$

[1]We will use "NoC traces" to refer to the trace files containing the communication the NoC supports.

[2]For MPI traces, this field is the clock resolution of a POSIX clock (1 ns, i.e. 1000 ps).

where:

  - *CO* is the COMM identifier composed of the character “C” followed by a unique natural number, in this example “0”.
  - [$element_0 ... element_{n-1}$] are the memory device identifiers. For example, cache L1 inside node 0 could have identifier 1. The identifiers assignation is responsibility of the simulator.

3. **Communication records.** They contain the communications generated by the MPI tasks or the cache hierarchy elements. Each record contains its dependency relationship with other communication records. There are two types of these records: point-to-point communications records, and collective communication records. We focus on the point-to-point communication records[3]. The format of these records is the following:

   ID src dst length Dep dTime IDdep

   where:

   - *ID* is the message identifier, unique for each message.
   - *src* is the source memory device identifier in NoC traces or it is the source task identifier in MPI traces.
   - *dst* is the destination memory device identifier in NoC traces or it is the source task identifier in MPI traces.

[3]Note that, although we do not use the collective records in NoC traces, the VEF trace format has the capability of modeling multicast operations. This is an extra advantage of the VEF trace format, since it will allow us to reproduce communications of another cache coherence protocol that performs multicast operations. For more information about collective operations, see [28].

- *length* is the size of the message measured in bytes.
- *Dep* is the type of dependency between records. Section 3.1.3 explains this in detail.
- *dTime* is the injection time or the dependency time, measured in cycles.
- *IDdep* is the record identifier on which the current record depends on.

Table 1 shows an example of a .vef file. Note that each trace section is delimited and annotated on the right side.

3.1.2. Names files

The reason to have this file separated from the .vef file is because on the original VEF format, every process is a task that can be distinguished with a natural number (ID). In on-chip communications there are several memory devices communicating (L1, L2, etc.) with each other and we cannot accurately distinguish them. For example, the ID 1 could be a L1, L2, DMA, etc., in short, could be any memory device. However, if we take a look to the .names file we can know what memory device is represented with the ID 1.

In order to preserve compatibility and be able to know the relationship between IDs and memory devices, we decided to add this file to each trace with this information. Furthermore, these files can be very useful to map memory devices in network interfaces (NI) during the trace replay process. The .names file contains the following information:

- **NODES:n:m** is the first file line. It shows the total amount of memory devices (n) and the latencies in cycles between any memory device attached to the same NoC tile (m), being n and m natural numbers.

– **id:element_n**, being *id* the memory device ID, shown in the .vef file as source (src) or destination (dst). *element* indicates the kind of memory device, being the most common possibilities: L1Cache, L2Cache, Directory and DMA. Finally, *n* indicates to which NoC tile the memory device was attached on the simulation that generated the trace.

Table 2 shows an example of a .names file. Note that this file is associated with the .vef file shown in Table 1.

Table 1: VEF3 sample trace (.vef)

VEF3	50	29	1	0	0	0	1000	// Header
C0	0	1	2	3	4	5	...	// Comms.
0	0	18	8	4	17	-1		/* Comm.
1	0	18	8	5	0	0		(send and
3	18	0	8	6	2	0		receive
4	18	0	72	6	2	1		messages)*/
5	0	18	8	2	2	3		
6	0	18	8	2	2	4		
7	0	17	8	5	2	6		
8	0	17	8	5	0	7		

Table 2: VEF3 sample trace (.names)

NODES:50:2	// Header
0:L1Cache_0	/* Identifier
1:L1Cache_1	(Memory devices
...	with their associated
17:L2Cache_1	Ids and NoC tile) */
18:L2Cache_2	
...	
47:Directory_15	
48:DMA_0	
49:DMA_1	

3.1.3. Dependency types

The field *Dep* is probably the most relevant field in the communication records. This field allows the VEF traces to be self-related, and indicates the condition that must be satisfied to process the following record in the workflow of each memory device. The dependency types are the following[4]:

0: **Independent records:** they correspond to the first messages sent by the memory devices or MPI tasks. These records do not depend on

[4]Note that dependency types have the same behavior either on MPI or NoC traces.

other records and are processed at the cycle $dTime$, being $IDdep = -1$.

1: **Send dependency:** the record can be processed when a previous message, whose identifier is specified in $IDdep$, is sent by the current device/task. The processing of this record is programmed $dTime$ cycles after the $IDdep$ message was sent.

2: **Receive dependency:** the current record can be processed after a message generated by another memory device or MPI task, whose identifier is specified in $IDdep$, is received by the current memory device/task. During the simulation, when the previous record is processed, two things happen with the current record:

   – The waited message has been received. In this case, the processing of the record is programmed $dTime$ cycles after the $IDdep$ message was received.

   – The message has not been received. In this case, the execution of the memory device/task is stopped until the message is received. Then, the processing of the current record is programmed $dTime$ cycles after the reception of the desired message.

3: **Group dependency:** the current record can be processed after a previous collective communication has finished. Only used in MPI traces.

4: **Independent records and trigger:** as the first dependency value, this record does not depend on a previous record and is also a “trigger message”. A message is a “trigger” message when the receiver has a record that depends on the reception of this message.

5: **Send and trigger:** send dependency and trigger message.

6: **Receive and trigger:** receive dependency and trigger message.

7: **Group dependency and trigger:** the current record shows a group dependency and is a trigger message. Only used in MPI traces.

Note that the dependencies from 4 to 7 were not considered in the previous version of the VEF2 trace format. These dependencies have been added in the VEF3 trace format to speed up the replay process. In the previous version, when a message is received, VEF-TraceLib checks if this reception triggers a new message. This checking was performed for all received messages, regardless of whether a new message was triggered or not. This is because, without these new dependencies, the system has no information about future messages, forcing to check all the messages that could potentially satisfy its dependency in the current simulation cycle. With these new dependencies, we significantly speed up the trace replay process. More details about this issue are given in Section 3.1.5 .

3.1.4. Example

Let us consider the VEF3 trace chunk shown in Table 1, corresponding to a 16 NoC tiles system running an application. After reading the trace header and the .names file (Table 2), we know the relationship between IDs and memory devices, the number of memory devices and the clock resolution.

Figure 1 shows the communication visual representation. Let us analyse the activity assuming that a message can be delivered in two cycles to any memory device; a memory device can process and reply a received message in the same cycle; and a memory device can deliver two simultaneous messages.

The first and second messages can be sent in cycle 17. The message 0 is independent (IDdep = -1) and is sent in the cycle indicated in the field *dTime*. Since the message 1 has a send dependency with the previous one, once the message 0 is sent, the message 1 is delivered as well.

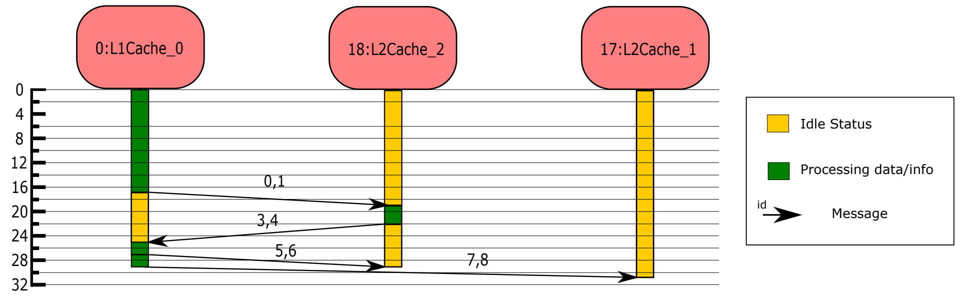

Figure 1: Communication between the three caches of the trace example.

Two cycles later, in cycle 19, the messages 0 and 1 arrive at the cache L2 in NoC tile 2 (18:L2_Cache2). The messages 3 and 4 have a receive dependency with the messages 0 and 1, respectively. Therefore, the messages 3 and 4 are sent two cycles later, because the cache needs these cycles to process the received messages. Messages 5 and 6 are in the same situation than messages 3 and 4. Hence, in cycle 25 they are delivered and received in cycle 27.

The message 7 has a send dependency with the message 6, which is sent in the cycle 25. Therefore, the message 7 is sent two cycles later. The message 8 has a send dependency with the message 7 that does not have processing cycles, so is sent in the same cycle as the message 7.

3.1.5. From VEF2 to VEF3

As stated at Section 3.1, the VEF3 trace format is an extension of VEF2 trace format, originally designed for MPI profiling. The first version of the trace format only includes the point-to-point communications, being added the collective communication in the second version. However, this trace format is not as efficient as desired and, because of that, we had to improve it.

The main problem regarding the VEF2 trace format is the strategy followed by the trace-replay library (VEF-TraceLib) to update the record dependencies after receiving a message. In VEF-TraceLib, each memory device/task has its own message window. The message window takes n communication records from the trace file, that are loaded in main memory and processed by VEF-TraceLib to determine if they have to be sent. When a communication record is used, it is removed from the message window and the library loads a new message and this process is repeated until the trace file end is reached. When a message arrives at its destination, the simulator notifies the message arrival to VEF-TraceLib. Then, the library goes through the arrival message window until: a) it finds a record that depends on the received message; b) the library checks the entire window message without finding a dependent record. In case b), a "false fail" can happen. That is, the received message triggers a new record, but this record has not yet been read from the trace file and added to the message window. In this case, the simulation will fail after a while.

Therefore, two problems appear; i) to properly adjust the size of the message window; ii) the amount of time spent cycling through the messages window increases proportionally with the message window size. Problem i) can be solved by increasing the window size as much as the computer memory allows us, but the simulation will take much more time, which is against of the trace-driven simulation goal. However, problem ii) cannot be solved because it is a consequence of the problem i). For this reason, we

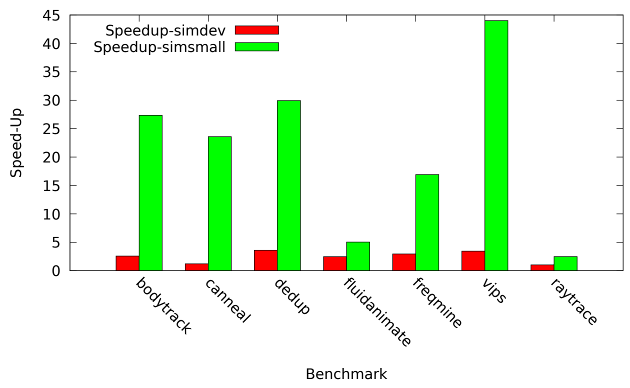

Figure 2: Speed-up of VEF3 versus VEF2 trace file format.

decided to update the trace format in order to solve these problems.

The new VEF3 trace format includes the dependencies 4 to 7, detailed in Section 3.1.3. These new dependencies provide the trace-replay library with more knowledge about the trace. Therefore, when a message is received, VEF-TraceLib only updates the message window if the received message is marked as a trigger-message, speeding up the trace-driven simulation.

We have carried out several experiments in order to determine if the VEF3 format improves the simulation time. We have extracted some traces from benchmark applications in the VEF3 trace format and we have downgraded them to the VEF2 trace format. After that, we have run simulations using both traces, measured the simulation times and checked the speed up.

We have chosen the Parsec 3.0 benchmark suite to perform our experiments. More details about the benchmarks can be found in Section 4.1.

Figure 2 shows the speed up achieved on each application. The best case is `vips` with the input `simsmall` achieving a speed up of 44.01x. The worst case is `raytrace` with the input `simdev` achieving a speed up of 1.029x.

The message window has been fixed to 500,000 messages in order to make a proper comparison between different applications and inputs. Note that in Figure 2 there is a selection of Parsec 3.0 benchmark applications. However, there are enough benchmark programs to extract conclusions about the VEF2 versus the VEF3 trace formats.

3.2. VEF3 profiler

To obtain the traces, we need a tool capable of capturing the network traffic generated by an application. In the case of NoC traces, our tool is a modified version of Gem5. We have chosen Gem5 because it is one of the most detailed full system simulators, is well documented and has a great user and developer community.

Since the message generation in shared-memory communication systems depends on the cache coherence protocol, we have implemented a sniffer interface on it. We have chosen MOESI_CMP_directory because it is a very popular cache coherence protocol on CMP systems. However, any cache coherence protocol can be easily modified to include the sniffer. Figure 3 shows a generic sniffer schema on which we base our trace sniffer. Each memory device has two tiny gray rectangles attached, representing sniffer links. The sniffer system is composed of:

– Sniffer engine: tracks every message sent or received, computes times, handles dependencies and stores the final trace into a file.

– Sniffer link: included inside each memory device, it communicates every message to the sniffer engine. Two kinds of sniffer links can be distinguished: send and receive sniffer links.

Before a message is sent or received, the sniffer analyzes the message, finding out the source and destination information, determining the dependencies with other messages and calculating how many cycles have been necessary for the system to process the information according to those dependencies.

We could also capture the messages at network level. However, using this approach, the source and destination of a message are defined in terms of NIs instead of cache devices. In other words, NIs hide this information due to the packing process. By capturing the traffic at the cache hierarchy level, we achieve more flexibility during the trace replay process, since the cache devices can be mapped to NIs as desired.

Once the application finishes its execution, the sniffer stores all the information following the VEF3 format.

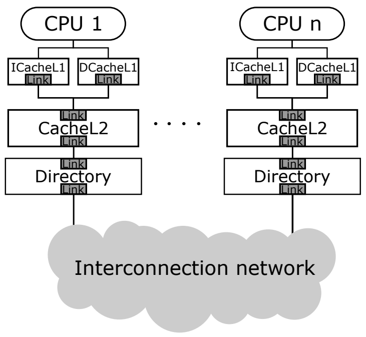

Figure 3: Generic sniffer diagram.

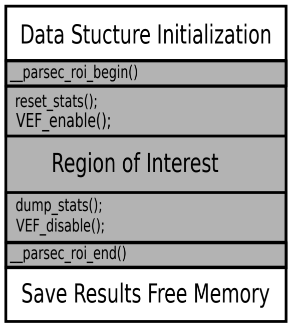

Figure 4: Execution block of modified applications.

3.2.1. Sniffer switching

During a full system simulation, many things happen before the parallelized code of the simulated application starts to be executed. If we want to obtain its communication trace, we face two possibilities; i) to capture all the traffic from the beginning of the simulation; ii) to switch on the sniffer in the most representative application region.

We have developed both options because this allows us much more flexibility. The first option is easy to be developed, but the second one is more complicated. In order to develop the second option, we took advantage of a Gem5 special instruction called **M5ops** [32], that allows us to change the simulation state inserting these instructions in the application source code. Moreover, including these special instructions into our applications, we can perform specific actions like setting a checkpoint, resetting the stats, dumping the stats into a file, etc. Taking advantage of this feature, we extend the set of *M5ops* instructions including two new ones: `VEF_enable` and `VEF_disable`, which enable and disable the sniffer, respectively.

The next step is to modify the applications in order to include the new *M5ops* instructions into the source code. As we point out in Section 4.1, we have chosen the Parsec 3.0 benchmark suite. We decided to place our new *M5ops* instructions inside the source code of the Parsec 3.0 profiling library because every benchmark application uses this library to be executed. This strategy avoids to place the new instructions in every benchmark application.

Figure 4 shows where and how the new *M5ops* instructions are placed, as well as other instructions used for managing the statistics collected until the application reaches the `VEF_disable` instruction. Each application of the Parsec benchmark suite defines a Region of Interest (ROI), that represents the computationally intensive and parallelized phase of the application, ignoring the initialization and the shutdown phase. These different phases are delimited in the source code by the Hooks library. For this reason, we have included the new *M5ops* instructions in the Hooks library: we can insert these instructions in all the applications making minimal changes in the benchmark source code.

After building the Parsec benchmark suite inside the simulated operating system on Gem5, it is possible to record the communications and to collect accurate statistics only during the ROI phase. Using both the trace and the statistics obtained, we have the possibility to evaluate our sniffer model.

3.3. VEF-TraceLib

VEF-TraceLib is the library responsible for replaying the VEF traces and it can be integrated in any interconnection network simulator. The library allows to simulate multiple VEF traces, add traces to the simulation on the fly, to allocate any task/memory devices to any NI and manage all the task/memory device activities transparently to the simulator, which only receives requests for sending messages between two different NIs. Note that, although NoC traces and MPI traces are obtained using different tools, both kind of traces can be used in the same simulation.

As we exposed in our previous work [5], using VEF-TraceLib2.4, results obtained are accurate enough, but in a few applications there are still significant differences in the network throughput. These differences are caused by the intra-messages in VEF-TraceLib2.4. An intra-message is a message between two memory devices attached to the same NI (e.g. L1 and L2 caches of the same NoC tile). The intra-messages are managed by VEF-TraceLib2.4 and the network simulator has no knowledge of these messages.

Therefore, in the simulation using VEF-TraceLib2.4 + Garnet2.0, Garnet has no information about the intra-messages, but Garnet takes into account these messages in the Gem5 simulation. For this reason, there are significant throughput differences if the number of intra-messages is high enough.

In order to improve the trace-replay accuracy and facilitate the use of the library, we have updated VEF-TraceLib2.4 to VEF-TraceLib2.5 including intra NoC tile latencies and memory devices auto-mapping.

3.3.1. Intra NoC tile latencies

VEF-TraceLib1.0 was originally developed for off-chip networks. In these systems, the MPI communications between tasks mapped in the same node do not require to use the off-chip network. For this reason, these communications must be managed by VEF-TraceLib1.0. Since we do not want to implement a complex NoC (remember that our main objective is to speed up the simulation), we implement a "perfect" network inside VEF-TraceLib1.0. This model is only composed of a fixed bandwidth configurable by the user. VEF-TraceLib1.0 simply uses this fixed bandwidth and the message size to calculate the number of cycles required to deliver this message in the NoC.

However, this "perfect" network does not model the behavior of the intra-messages in the Gem5 simulation, generating significant differences between the full-system and the trace-driven simulation. To smooth the intra-messages impact, we have implemented new configuration options that take control of the intra NoC tile latencies in the new version VEF-TraceLib2.5. Currently, the latency information can be set from multiple sources:

– "Perfect" network with a fixed bandwidth (intra NoC tile of previous library versions).

– From .names files generated by our modified Gem5 version (see Section 3.1.2). In Gem5, the link latencies cannot be configured separately; this is, every link in the system always has the same latency value.

Hence, the latency between all the intra NoC tile memory devices is set to the same value, which is indicated in the .names file.

- Setting latencies manually when configuring VEF-TraceLib2.5 at the start of the simulation. Users can manually define each latency between any given memory device. This method provides a lot of flexibility allowing almost any latency configuration.
- From a default latency value. If the intra NoC tile latencies mode is enabled, we set a default value of 1 cycle on each memory device.

These changes aim to reduce full-system versus trace-driven simulation differences. Note that our goal is to speed up simulations maintaining the accuracy with respect to the full-system execution.

3.3.2. Memory devices auto-mapping

As we briefly explained in Sections 3.1.2 and 3.2, memory devices have to be mapped into NIs. This is because in a NoC simulation memory devices are not simulated, but only the interconnection devices (switches, arbiters, buffers, etc.). VEF-TraceLib takes care of them managing their messages. However, VEF-TraceLib needs to know which memory device is connected to each NoC tile or NI.

Using the previous VEF-TraceLib versions, the final user was the responsible of mapping each memory device in the desired NI. The new library version, VEF-TraceLib2.5, brings the possibility to auto-map memory devices, replicating the same configuration used in the full-system simulation. The auto-mapper reads the .names files to know where the memory devices were mapped in the full-system simulation. The only exception are DMA devices, since these particular memory devices are always mapped into NI 0, matching with the Gem5 DMA mapping policy. Manual mapping is still available, allowing any mapping desired by the user.

3.3.3. Integration of VEF-TraceLib in Gem5

One of our objectives is to check the accuracy of the NoC trace model and for that we compare it against full-system executions (Section 4), comparing the statistics obtained for the network simulator Garnet in both scenarios. For this purpose, we have developed a trace reader inside Gem5 using VEF-TraceLib2.5. The main drawback of this proposal is that messages in Garnet cannot be sent between NIs directly. Instead, they have to follow the same shared-memory approach employed in the full-system. In order to solve this issue, we based our reader on the system used by Gem5 to generate synthetic traffic. This system is composed of the custom cache coherence protocol called **Garnet standalone** [33] and a CPU model which generates the network workload. Figure 5 shows how a message is sent from Core 0 to Core 1 using the cache hierarchy and the shared-memory philosophy.

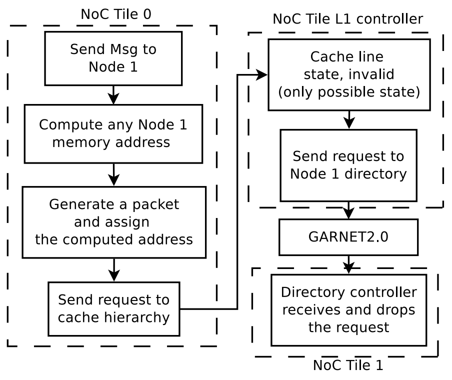
NoC Tile 0
Send Msg to Node 1
Compute any Node 1 memory address
Generate a packet and assign the computed address
Send request to cache hierarchy
NoC Tile L1 controller
Cache line state, invalid (only possible state)
Send request to Node 1 directory
GARNET2.0
Directory controller receives and drops the request
NoC Tile 1

Figure 5: Garnet standalone mechanism to send messages to NoC tiles.

Hence, we have included VEF-TraceLib2.5 inside the new CPU model. Every cycle, the CPU model asks the library if there are new messages to be sent, delivering these messages following the Garnet standalone mechanism. When a message reaches its network destination, Garnet communicates the reception to the CPU model, and VEF-TraceLib2.5 processes the received message. More detailed information about the integration of VEF-TraceLib in other simulators can be found in [3].

4. Experiments and results

The aim of this section is to highlight the acceleration in trace-based simulations due to the improvements introduced in the VEF framework. It is also important to check the accuracy of the trace-based simulations and therefore we have compared the results obtained when trace files are used in the interconnection network simulator with the ones obtained from full-system simulations.

Next, we show the obtained results, but previously we indicate the scenarios considered and the modifications in Gem5 to develop the experiments.

4.1. Simulation scenario

We have used our custom Gem5 version to generate and replay the VEF3 traces files. We have not used checkpoints to run the experiments. This is because we faced some problems to reuse checkpoints when a change in any aspect (i.e. topology) was required.

We have used a modified version of the Parsec 3.0 benchmark suite to perform our experiments. Parsec 3.0 is one of the most famous set of applications for NoC benchmarking. Moreover, this benchmark suite is userfriendly, including a manager to abstract the specific use of each application and a profiling library, called Hooks.

In all the experiments, we have configured Gem5 and Garnet2.0 using the parameters shown in Table 3 as baseline (default parameters). We have performed a series of experiments in which we have changed just one parameter and ran a collection of Parsec 3.0 benchmark programs. The sets of experiments are the following:

- Topology mesh_XY + link width 672 bits + default parameters
- Topology torus + link width 672 bits + default parameters
- Topology mesh_XY + link width 128 bits + default parameters
- Topology torus + link width 128 bits + default parameters

Table 3: Parameters used in the experiments (default parameters).

Parameter	**Value**	**Parameter**	**Value**
Ruby-clock	1GHz	Number of L1 caches	16
sys-clock	1GHz	Number of L2 caches	16
cpu-clock	1GHz	Number of directories	16
Network model	Garnet2.0	Number of nodes	1
Topology size	4x4	Number of NoC tiles per node	16
Kernel version	3.4.112	Memory model	Ruby
Routing Algorithm	DOR	CPU model	DerivO3CPU

As we pointed out in Section 3.3, VEF-TraceLib2.5 is now the responsible for the mapping strategy. Therefore, this is not longer a user concern. In the experiments, each memory device is attached to the same NoC tile it was connected in the full-system simulation. However, we have only simulated one memory device on the trace-driven simulation (it is required, as we have explained in Section 3.3.3 for sending and receiving packets through the network). This means that in trace-driven simulations, no matter if the source of a given message is an L1, L2, etc.. It will be dispatched from the L1 of the same NoC tile, just as in the full-system simulations. Hence, the L1 port is multiplexed among all memory devices in the NoC tile.

The NoC router in trace-driven simulations has at most 5 ports, 1 port for connecting the L1 and 4 ports for connecting with neighbor routers, although border and corner NoC routers on the mesh topology only have 4 and 3 ports, respectively.

Note that in these experiments we have used the *DerivO3CPU* CPU model, that allows to get the most detailed experiments which are aimed to compare the full-system execution with the trace-driven execution.

We have performed the experiments choosing the inputs *simdev* and *simsmall* because they take a reasonable simulation time and their traffic workloads are enough to study the trace model accuracy.

4.2. Custom Gem5

Compared to our previous Gem5 version [5] we have included some new features required to develop these experiments. Firstly, the torus topology has been implemented. We took Mesh_XY as baseline and managed to add the additional links, creating this new topology. The routing protocol is based on shortest path given weighted links, where vertical links have a weight of 2 and horizontal links have a weight of 1 [22].

Secondly, VEF-TraceLib2.5 includes functions for obtaining basic information about the trace execution: total amount of sent bytes, number of sent messages, throughput, etc. Moreover, these functions offer statistics about the intra-messages. In order to improve the overall statistics, we have combined the statistics offered by VEF-TraceLib2.5 and Garnet2.0 to be aware of intra NoC tile traffic.

4.3. Results

In order to check the accuracy of the trace-based simulations, we have compared the packet latency and the throughput obtained when trace files are used in the two simulation models considered (i.e. using Gem5+Garnet2.0 and TraceLib+Garnet2.0). In the case of packet latency, we have recorded the latency of each packet for the two simulation models and calculated the differences. Figure 6 shows results on the average of all differences.

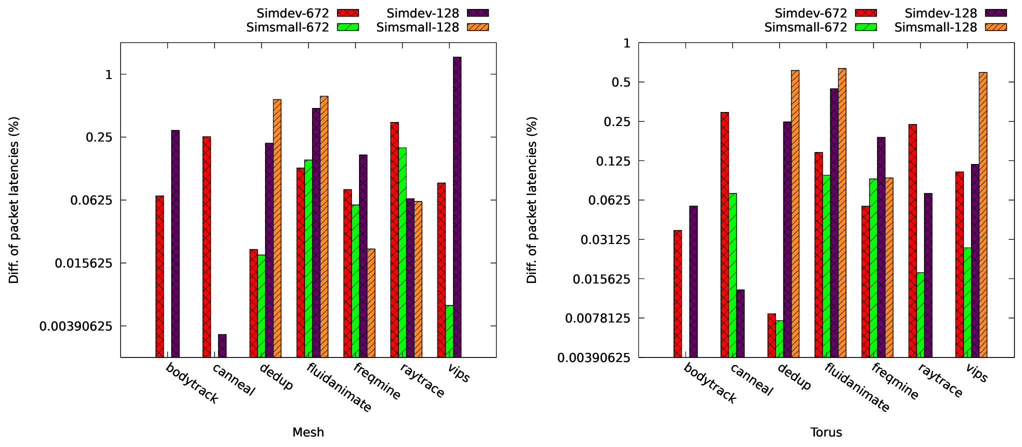

Figure 6: Latency differences between full-system and trace-driven simulation.

Throughput values for each simulation model have been obtained every 10,000 packets received by the NIs, and also the differences have been obtained. Figure 7 shows data on the average of all differences.

As it can be seen in both figures, the average differences in latency and throughput are expressed as a percentage. The absolute values of both metrics for all configurations have been included as supplementary material.

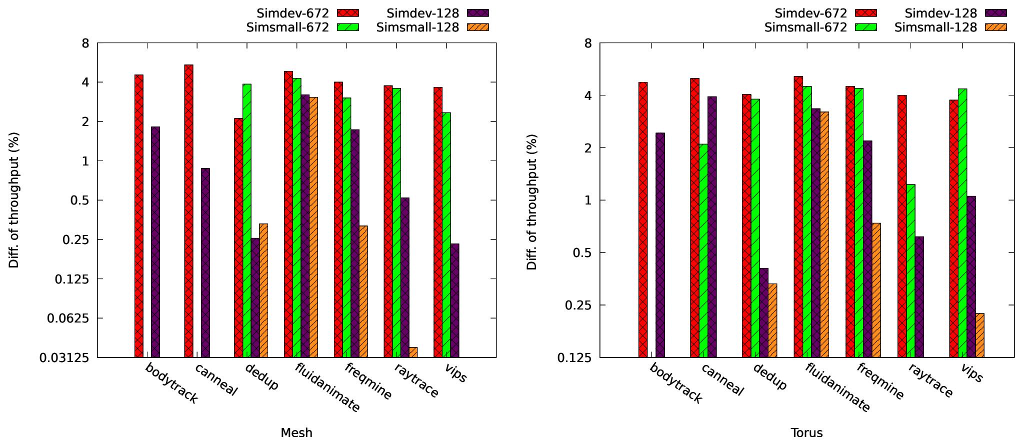

Figure 7: Throughput differences between full-system and trace-driven simulation.

In Figure 6 we can observe that only one case exceeds 1% for average difference, which is a very positive result. In the case of throughput the differences are slightly higher, but always less than 5.42%. These results are better than that obtained for Elastic traces, which are in the range 7-17%.

On the other hand, no dependence is observed on the accuracy results, neither on the aspects of the network considered (topology, size of the network and link width) nor on those related to the applications (application and input size). This means, at least from this point of view, that there are not significant differences between the two simulation models.

Therefore, the differences in terms of two important network performance metrics are negligible and so we conclude the trace-based network simulator behavior is almost equal to the full-system.

The main source of differences, either latency or throughput, is caused by intra-messages, which are messages that have to be sent to memory devices

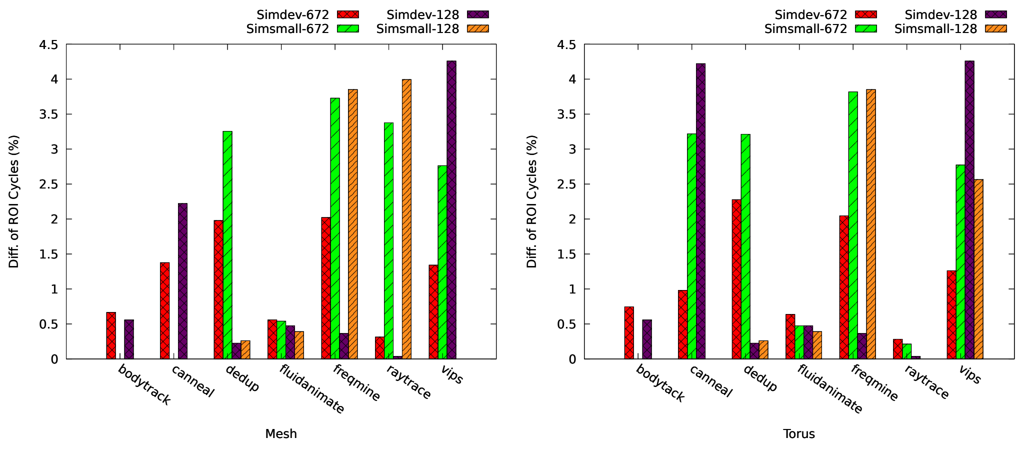

Figure 8: Difference of cycles on ROI section using the trace-driven simulation.

included in the same NoC tile. Since these messages do not require to be sent through the NoC tile NI, these intra-messages are internally managed by VEF-TraceLib2.5 when the trace files are replayed.

Intra-messages also have dependencies that must be satisfied before moving to the next message. For example, in a given time x VEF-TraceLib2.5 finds an intra-message m_5 with 4 processing cycles, and the next message is an inter-message m_6, that has a send dependency with m_5. VEF-TraceLib2.5 will guarantee that m_6 will not be sent before $x + 4$ cycles. Hence, intra-message latencies must be configured in a proper way in order to reduce these differences. In these experiments we have used the latency and the clock frequency provided by .names files and the .vef header, respectively.

Even so, there are still some differences between trace–driven and full–system models. For instance, in the full-system simulations each memory device has its own port to the router while in the trace-based simulations there is only one shared port for L1, L2 and Directory. Despite that, the differences are low, and in many cases insignificant.

In addition, for checking accuracy in terms of metrics such as latency and throughput, it is also interesting to do it considering the ROI execution. This is the computation-intensive part of the application, in which the most messages will be generated because, on the one hand, the consistency of the data in the caches must be kept, and, on the other hand, because there will be more failures in the caches.

Figure 8 shows the percentage of difference between trace-driven and full-system simulations on total cycles executing the ROI section of each application. The difference is never greater than 4.25%. The best case is raytrace using the simdev input, mesh topology and link width of 128 bits, with a difference of 0.036%. The worst case is vips using simdev, torus topology and link width of 128 bits with a difference of 4.25%.

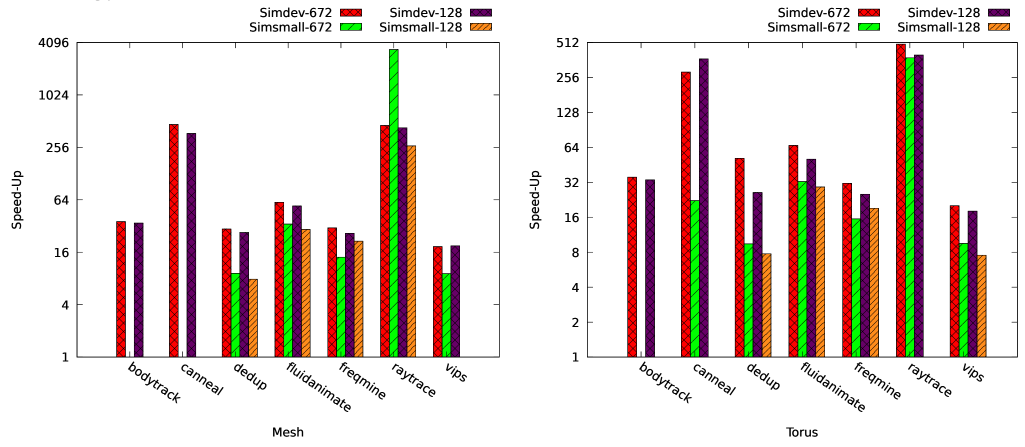

Figure 9: Simulation speed up using the trace-driven simulation.

Another main objective pursued with the improvements introduced in the VEF trace framework is the reduction of the total simulation time. Figure 9 shows the speed up achieved in each of the configurations considered. The simulation time is reduced in an important factor which ranges from 7x to 3392x. This means that some simulations that with the full-system would take days, can be completed with the trace-based system in a few hours, and even minutes.

This simulation time reduction has great relevance, and is essential when simulating Exascale systems, where the interconnection network can be composed of hundreds of thousands of nodes.

Table 4: Trace file sizes and total number of messages for the traces obtained for the mesh topology with link width of 672 bits.

Benchmark	**Input**	**Size (MB)**	**Number of messages**
Dedup	Simsmall	2440	82,138,304
Fluidanimate	Simsmall	490	17,098,687
Freqmine	Simsmall	332	11,866,998
Raytrace	Simsmall	201	7,240,623
Vips	Simsmall	3643	122,451,603
Bodytrack	Simdev	193	7,027,735
Canneal	Simdev	12	458,358
Dedup	Simdev	250	9,046,777
Fluidanimate	Simdev	141	5,118,125
Freqmine	Simdev	263	9,524,430
Raytrace	Simdev	12	437,716
Vips	Simdev	298	10,773,892

When simulating large systems using trace-based model, an important aspect to consider is the trace files size. That size can be very large and its handling can slow down the simulations. Several strategies can be followed so that this is not a problem. We comment briefly on this issue, using the trace file data used in this work.

Table 4 shows the file size in MBs and the total number of messages for each trace generated in mesh topology with link width of 672 bits. We obtain similar values for the rest of configurations considered. We consider these file sizes are not too large given the tremendous amount of messages. Note that trace files are ASCII encoded plain text. However, trace files may be compressed, which will reduce the total trace file size keeping the same number of messages. We have tested the size reduction just by compressing `dedup` using simsmall input with GNU tar 1.28 [34] getting a total file size of 592MB.

5. Conclusions

VEF trace framework provides tools to obtain and manage file traces containing the off- and on-chip communication generated for applications running in large computing systems. VEF traces can be used on any interconnection network simulator with minimal effort. A essential characteristic of VEF traces is that they contain information about the relationship between communication messages, making the simulations much more realistic.

We present several improvements related mainly to the on-chip communications, which significantly speed up the simulations and increase their accuracy. Thus, on the one hand, when using different applications and inputs from the Parsec 3.0 benchmark suite, the simulation time using VEF traces is reduced in a great factor which ranges from 7x to 3392x.

On the other hand, and to check the accuracy of the trace-based simulations, we have compared the results obtained using Gem5+Garnet2.0 and TraceLib+Garnet2.0. The results show that differences, for instance, in terms of packet latency and throughput respect to the full-system applications execution are 5.42% in the worst-case scenario.

It is also important to note that collecting on-chip communication at the cache hierarchy level offers greater possibilities in the cache device mapping process, achieving, in this way, more flexibility during the trace replay process in any interconnection network simulator. Other improvements introduced in the framework are related to the auto-mapping the memory devices into NIs, managing intra-messages and more accurate simulation statistics.

Finally, to point out that VEF trace framework allows us to capture messages from both off-chip and on-chip networks, for now separately, and our next objective is to capture both types of traffic that an application generates when is running on an Exascale system.

Acknowledgment

This work has been supported by the Junta de Comunidades de Castilla-La Mancha, European Commission (FEDER funds) and MICINN under projects SBPLY/17/180501/000498 and RTI2018-098156-B-C52 respectively. It is also co-financed by the University of Castilla-La Mancha and FEDER funds under project

2019-GRIN-27060. Javier Cano-Cano is also funded by the MINECO under FPI grant BES-2016-078800.

References

References

[1] Summit oak ridge national laboratory's next high performance supercomputer, http://tiny.cc/m8pc3y, (Accessed February 16, 2019).

[2] D. Reinsel, J. Gantz, R. John, Data age 2025, The Digitization of the World: From Edge to Core. Available at: http://tiny.cc/m5pc3y.

[3] F. J. Andújar, J. A. Villar, J. L. Sánchez, F. J. Alfaro, J. Escudero-Sahuquillo, An open-source family of tools to reproduce MPI-based workloads in interconnection network simulators, The Journal of Supercomputing (2016) 1–28.

[4] R. Rabenseifner, G. Hager, G. Jost, Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes, in: Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on, IEEE, 2009, pp. 427–436.

[5] J. Cano-Cano, F. J. Andújar, F. J. Alfaro, J. L. Sánchez, Vef3 traces: Towards a complete framework for modelling network workloads for exascale systems, in: 2018 IEEE 4th International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB), 2018, pp. 32–39.

[6] G. F. Riley, T. R. Henderson, The ns-3 network simulator, in: Modeling and tools for network simulation, Springer, 2010, pp. 15–34.

[7] A. Varga, et al., Omnet++ user manual, OMNeT++ Discrete Event Simulation System. Available at: http://www. omnetpp. org/doc/manual/usman. html.

[8] H. Casanova, A. Giersch, A. Legrand, M. Quinson, F. Suter, Versatile, scalable, and accurate simulation of distributed applications and platforms, Journal of Parallel and Distributed Computing 74 (10) (2014) 2899–2917.

[9] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, et al., The structural simulation toolkit, SIGMETRICS Performance Evaluation Review 38 (4) (2011) 37–42.

[10] P. Yebenes, J. Escudero-Sahuquillo, P. J. Garcia, F. J. Quiles, Towards modeling interconnection networks of exascale systems with omnet++, in: 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, IEEE, 2013, pp. 203–207.

[11] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, W. J. Dally, A detailed and flexible cycle-accurate network-on-chip simulator, in: 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), IEEE, 2013, pp. 86–96.

[12] P. Abad, P. Prieto, L. G. Menezo, V. Puente, J.-Á. Gregorio, et al., Topaz: An open-source interconnection network simulator for chip multiprocessors and supercomputers, in: 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip, IEEE, 2012, pp. 99–106.

[13] G. Maglione-Mathey, P. Yebenes, J. Escudero-Sahuquillo, P. J. Garcia, F. J. Quiles, Combining openfabrics software and simulation tools for modeling

infiniband-based interconnection networks, in: 2016 2nd IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB), IEEE, 2016, pp. 55–58.

[14] M. S. Müller, A. Knüpfer, M. Jurenz, M. Lieber, H. Brunst, H. Mix, W. E. Nagel, Developing scalable applications with vampir, vampirserver and vampirtrace., in: PARCO, Vol. 15, Citeseer, 2007, pp. 637–644.

[15] S. S. Shende, A. D. Malony, The TAU parallel performance system, The International Journal of High Performance Computing Applications 20 (2) (2006) 287–311.

[16] V. Pillet, J. Labarta, T. Cortes, S. Girona, Paraver: A tool to visualize and analyze parallel code, in: Proceedings of WoTUG-18: transputer and occam developments, Vol. 44, IOS Press, 1995, pp. 17–31.

[17] A. Knüpfer, C. Rössel, D. an Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, et al., Score-p: A joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir, in: Tools for High Performance Computing 2011, Springer, 2012, pp. 79–91.

[18] Z. Lu, R. Thid, M. Millberg, E. Nilsson, A. Jantsch, NNSE: Nostrum network-on-chip simulation environment, Proc. of SSoCC.

[19] L. Jain, B. Al-Hashimi, M. Gaur, V. Laxmi, A. Narayanan, NIRGAM: a simulator for NoC interconnect routing and application modeling, in: Design, Automation and Test in Europe Conference, IEEE, 2007, pp. 16–20.

[20] V. Catania, A. Mineo, S. Monteleone, M. Palesi, 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.

[21] V. Puente, J. A. Gregorio, R. Beivide, Sicosys: an integrated framework for studying interconnection network performance in multiprocessor systems, in: Parallel, Distributed and Network-based Processing, 2002. 10th Euromicro Workshop on, IEEE, 2002, pp. 15–22.

[22] N. Agarwal, T. Krishna, L.-S. Peh, 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.

[23] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al., The Gem5 simulator, ACM SIGARCH Computer Architecture News 39 (2) (2011) 1–7.

[24] R. Jagtap, S. Diestelhorst, A. Hansson, M. Jung, Exploring system performance using elastic traces: Fast, accurate and portable, in: Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), 2016 International Conference on, IEEE, 2016, pp. 96–105.

[25] A. Nocua, F. Bruguier, G. Sassatelli, A. Gamatie, Elasticsimmate: A fast and accurate gem5 trace-driven simulator for multicore systems, in: Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2017 12th International Symposium on, IEEE, 2017, pp. 1–8.

[26] J. Hestness, B. Grot, S. W. Keckler, Netrace: dependency-driven trace-based network-on-chip simulation, in: Proceedings of the Third International Workshop on Network on Chip Architectures, ACM, 2010, pp. 31–36.

[27] S. Nilakantan, K. Sangaiah, A. More, G. Salvadory, B. Taskin, M. Hempstead, Synchrotrace: synchronization-aware architecture-agnostic traces for light-weight multicore simulation, in: Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on, IEEE, 2015, pp. 278–287.

[28] F. J. Andújar, J. A. Villar, J. L. Sánchez, F. J. Alfaro, J. Escudero-Sahuquillo, VEF traces: a framework for modelling MPI traffic in interconnection network simulators, in: Cluster Computing (CLUSTER), 2015 IEEE International Conference on, IEEE, 2015, pp. 841–848.

[29] P. Yébenes, J. Escudero-Sahuquillo, P. J. García, F. J. Quiles, Straightforward solutions to reduce hol blocking in different dragonfly fully-connected interconnection patterns, The Journal of Supercomputing 72 (12) (2016) 4497–4519.

[30] F. Zahn, S. Lammel, H. Fröning, Early experiences with saving energy in direct interconnection networks, in: 2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB), IEEE, 2017, pp. 33–40.

[31] F. J. Andujar, S. Coll, M. Alonso, J.-M. Martinez, P. Lopez, F. J. Alfaro, J. L. Sanchez, R. Martinez, Analyzing topology parameters for achieving energy-efficient k-ary n-cubes, in: 2018 IEEE 4th International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB), IEEE, 2018, pp. 24–31.

[32] M5ops homepage, `http://gem5.org/M5ops`, (Accessed May 22, 2018).

[33] Garnet Standalone homepage, http://www.gem5.org/Garnet_standalone, (Accessed May 22, 2018).

[34] Tar gnu project free software foundation, https://www.gnu.org/software/tar, (Accessed February 5, 2019).