Workload Characterization for Exascale Computing Networks

José Duro, Salvador Petit, Julio Sahuquillo and María E. Gómez Departamento de Informaática de Sistemas y Computadores
Universitat Politècnica de València, Spain
Email: jodugo1@gap.upv.es

***Abstract*—Exascale computing is the next step in high performance computing provided by systems composed of millions of interconnected processing cores. In order to guide the design and implementation of such systems, multiple workload characterization studies and system performance evaluations are required.**

This paper provides a workload characterization study in the context of the European Project ExaNeSt, which focuses, among others, on developing the network technology required to implement future exascale systems. In this work, we characterize different ExaNeSt applications from the computer network perspective by analyzing the distribution of messages, the dynamic bandwidth consumption, and the spatial communication patterns among cores.

The analysis highlights three main observations; i) message sizes are, in general, below 50 kB; ii) communication patterns are usually bursty; and iii) spatial communication among cores concentrate in hot spots for most applications. Taking into account these observations, one can conclude that in order to unclog congested network links, an exascale network must be designed to briefly support higher-than-average bandwidths in the vicinity of key network cores.

***Keywords*—*Workload characterization*; *Exascale computing*; *MPI*.**

I. Introduction

Exascale computing, which aims to reach exaflop (10^{18} floating-point operations per second) computing power, is the next challenge for the supercomputing community. According to the current growing computational trend, this goal is expected to be achieved by 2020. In order to achieve such a huge computing capability, systems will require millions of interconnected computing elements that execute massive parallel applications.

Exascale networks will be composed of thousands of computing cores, so data transmission among them becomes a major design concern. In this context, new requirements rise, not only in terms of throughput, but also regarding energy demands. In such systems, the underlying network technology [1], [2] and topology are critical design choices. The focus of the European Exascale System Interconnect and Storage project (ExaNeSt) [3], [4], which is currently being developed, is to provide a feasible implementation that meets the mentioned requirements.

ExaNeSt simulation frameworks model electrical [5], [6] and optical [7] interconnection networks. To perform a realistic analysis of the entire system behavior, simulations platforms must be fed either with real applications or traces modeling the behavior of such applications. The main purpose of this paper is to present a detailed characterization of the ExaNeSt workloads in order to provide a deep knowledge of the network requirements and to guide network designers in decision taking.

To characterize the network requirements of the ExaNeSt workloads, they were profiled to collect their execution time and both point-to-point and MPI (Message Passing Interface) collective messages [8]. The profiled information is used to build traces that are analyzed to obtain insight about distribution of message sizes, temporal evolution of bandwidth utilization, and spatial communication patterns. The characterization data will be useful to select and/or develop suitable topologies and network technologies for a feasible and efficient exascale network implementation.

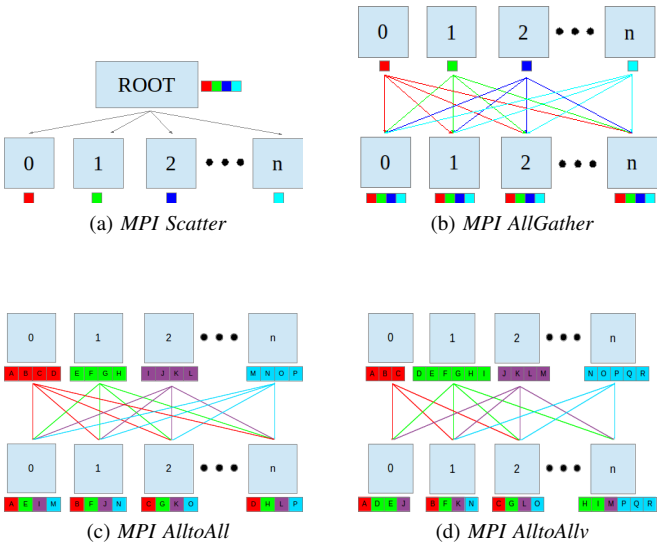
The characterization study highlights three main observations; i) communication among cores are, in general, mostly performed with messages whose size is lower than 50KB; ii) although applications present a wide range of bandwidth consumptions, communication patterns are usually bursty, and iii) our analysis shows that spatial communication among cores can concentrate in hot spots. As a result, we can conclude that to avoid performance losses because of the computer network, an exascale network should be designed to support higher-than-average bandwidths in key network cores at specific points of time.

The remainder of this paper is organized as follows. Section II motivates this work in the context of the ExaNeSt project. Section III gives a background about MPI collectives. Section IV presents an overview of the studied workloads. Sections V, VI, and VII analyze, respectively, the distribution of message sizes, the temporal evolution of bandwidth consumption, and the spatial communication patterns of the studied workloads. Finally, Section VIII presents some concluding remarks.

II. Motivation

The requirements for Exascale computing over the current decade are expected to scale the network performance. The ExaNeSt project is currently designing and building a prototype network architecture capable of reaching Exascale computing.

The aim of ExaNeSt is to develop a system that can be scaled up to the tens of millions of interconnected low-power



(a) *MPI Scatter* (b) *MPI AllGather*

(c) *MPI AlltoAll* (d) *MPI AlltoAllv*

Fig. 1: MPI collectives.

consumption cores to solve large-scale scientific and big data problems. In order to support a system of this size, ExaNeSt is confronted with the huge challenge of designing an interconnect able to meet very strict performance, resilience, and cost constraints for a range of computational challenges.

The ExaNeSt interconnect is a multi-tier interconnect which can be divided into two distinct parts. The lower tiers, which are physically fixed by means of boards and backplanes, and the higher tiers which are fully reconfigurable using custom-made FPGA-based [9] routers. This flexibility allows to build any network topology, i.e. direct, indirect or hybrid, or even the use of standard off-the-shelf commodity switches.

In order to meet the requirements of such demanding interconnect, we analyze in this work the real applications that are being used to test the entire system (compute cores, interconnection network, and storage).

III. Background on MPI Collectives

ExaNeSt applications consist of thousands of threads which have been coded with message passing interface (MPI) collectives. In order to help understanding the characterization study this section identifies the MPI collectives used by the studied workloads and describes how they work.

When profiling the ExaNeSt workloads, we found the following set of primitives: *MPI_Bcast, MPI_Scatter, MPI_Scatterv, MPI_Gather, MPI_Allgather, MPI_Reduce, MPI_Allreduce, MPI_Alltoall, MPI_Alltoallv,* and *MPI_Scan*.

- *MPI_Bcast*: this primitive allows a process (i.e. the root) to send an array chunk to all processes in a communicator (i.e. the set of cores involved in the collective).
- *MPI_Scatter*: this collective is similar to *MPI_Bcast*. The main difference is that *MPI_Scatter* sends different equally-sized chunks of an array to different processes (see Figure 1a).
- *MPI_Scatterv*: a variation of the *MPI_Scatter* collective where chunks can present different sizes.
- *MPI_Gather*: it implements the opposite behaviour of *MPI_Scatter*. Instead of spreading elements from one process (root) to many processes, *MPI_Gather* takes elements from many processes and sends them to the same single process (root).
- *MPI_Gatherv*: as *MPI_Gather*, but with chunks of different sizes.
- *MPI_Allgather*: given a set of data elements distributed across all processes, this collective sends all the elements to all the processes. On the first stage, *MPI_Allgather* forwards the elements to the root process, and then, on a second stage, this process sends the collected data to all process (see Figure 1b).
- *MPI_Reduce*: this primitive is similar to *MPI_Gather*. *MPI_Reduce* takes an array of elements on each process and returns a processed array of elements to the root process. Thus, this primitive implies a computation with the data of each element.
- *MPI_Allreduce*: as *MPI_Reduce* but the result of the computation is distributed among all the processes.
- *MPI_Barrier*: this primitive implements a barrier. Thus, a process calling it stalls until all the processes in the communicator have also called it. It is implemented as *MPI_Bcast* but with very short messages (i.e. tokens).
- *MPI_Alltoall*: it is similar to *MPI_Scatter* but in this case, all processes divide input arrays with the same size into equal chunks and send each chunk to all processes in the communicator (see Figure 1c). With this primitive, each process sends and receives the same amount of data.
- *MPI_Alltoallv*: this primitive presents two main differences with respect to *MPI_Alltoall*. On the one hand, the input arrays can have different size and, on the other hand, a process can receive differently-sized chunks from each sender (or not receive anything from a particular core) (see Figure 1d).
- *MPI_Scan*: it calculates an incremental partial reduction among participant processes. This means that each process i calculates the reduction from process 0 to itself. That is, the last process n will obtain the total reduction among all the processes.

IV. ExaNeSt Workloads and Trace Analysis Methodology

The ExaNeSt workloads consist of four main applications existing or developed by the ExaNeSt partners, namely *Lammps* [10], *RegCM* [11], *DPSNN* [12] and *Gadget* [13]. Details about how each application works can be found in each particular reference.

To provide insights on the relationship between the bandwidth requirements and the network core count, different workload sizes have been considered. More precisely, ExaNeSt partners provided 19 traces generated with real-hardware using different network statistic collection tools such as *scalasca* [14], which provide MPI primitives information in addition to computation times and message timestamps.

TABLE I: Average bandwidth requirements for each trace.

Application		Execution Time (cycles)	MB Transf (Total)	MB/s (average)
Lammps	24 Cores	43,754,790,287	24,644	1,126
	48 Cores	21,713,810,259	31,141	2,868
	96 Cores	10,887,071,229	40,934	7,520
	192 Cores	5,983,794,523	55,338	18,496
	384 Cores	152,820,600,368	71,924	941
	768 Cores	322,882,228,358	97,993	607
RegCM	24 Cores	139,543.000,917	22,976	329
	48 Cores	80,112,643,804	33,157	828
	96 Cores	49,580,028,588	47,213	1,905
	144 Cores	51,640,032,689	58,138	2,252
	192 Cores	40,411,947,679	69,131	3,421
Gadget	24 Cores	316,651,890,866	152,567	964
	48 Cores	257,497,854,652	267,104	2,075
	72 Cores	207,637,515,308	415,640	4,004
DPSNN 32x32	32 Cores	8,795,221,526,254	34,533	8
	64 Cores	4,568,949,694,490	45,690	20
	128 Cores	2,744,786,342,258	65,125	47
DPSNN 64x64	64 Cores	23,829,773,201,115	170,572	14
	128 Cores	12,287,105,198,156	199,951	33

These traces corresponding to 6 different executions (varying the core count from 24 to 768) from Lammps, 5 from RegCM (from 24 to 192 cores), 5 from DSPNN (from 32 to 128 cores in the 32x32 neural columns configuration and from 64 to 128 cores in the 64x64 neural columns configuration), and 3 from Gadget (from 24 to 72 cores). In this work we present the analysis of a representative subset of the provided traces.

The first step in the characterization study is to provide an overall overview of the traces from the network bandwidth perspective. For this purpose, we gathered two key parameters: the execution time and the overall transferred data. From them, we computed the average bandwidth consumed by each application.

We consider all the transferred data, both header and payload. The messages are split into packets of 72 bytes to be injected on the network, where 64 bytes correspond to payload and the remaining 8 bytes left are for the header. More precisely, a 1KB-message is split into 16 packets as done in some modern machines [15], incurring a 128B overhead for the headers.

Table I shows the results. As observed, the obtained values widely vary across the studied variables. The execution time presents differences ranging from around 6 up to 23839 billion cycles. Bandwidth requirements exhibit high variations regardless of the execution time. Since the traces do not provide the processor frequency, we have assumed a 2 GHz processor clock to calculate the average bandwidth in seconds. The results show that there are applications with relatively few bandwidth requirements (about 8 MBps) and applications with huge bandwidth requirements (e.g. around 18.5GBps). Note that when the number of cores exceeds 192 in Lammps, the execution time increases. The reason is that the original system used to generate these traces has less cores than parallel threads are generated in these configurations, which affects the execution time.

V. Analysis of Message Sizes

The analysis of the message size was performed to discern if message delivery could be improved by prioritizing either latency or bandwidth. For instance, if there is a high amount of short messages we could opt to prioritize latency over bandwidth; however, upon large message sizes, we should prioritize bandwidth to improve network and hence application performance. Although 19 traces have been analyzed, only the identified representative patterns are presented and discussed for illustrative purposes.

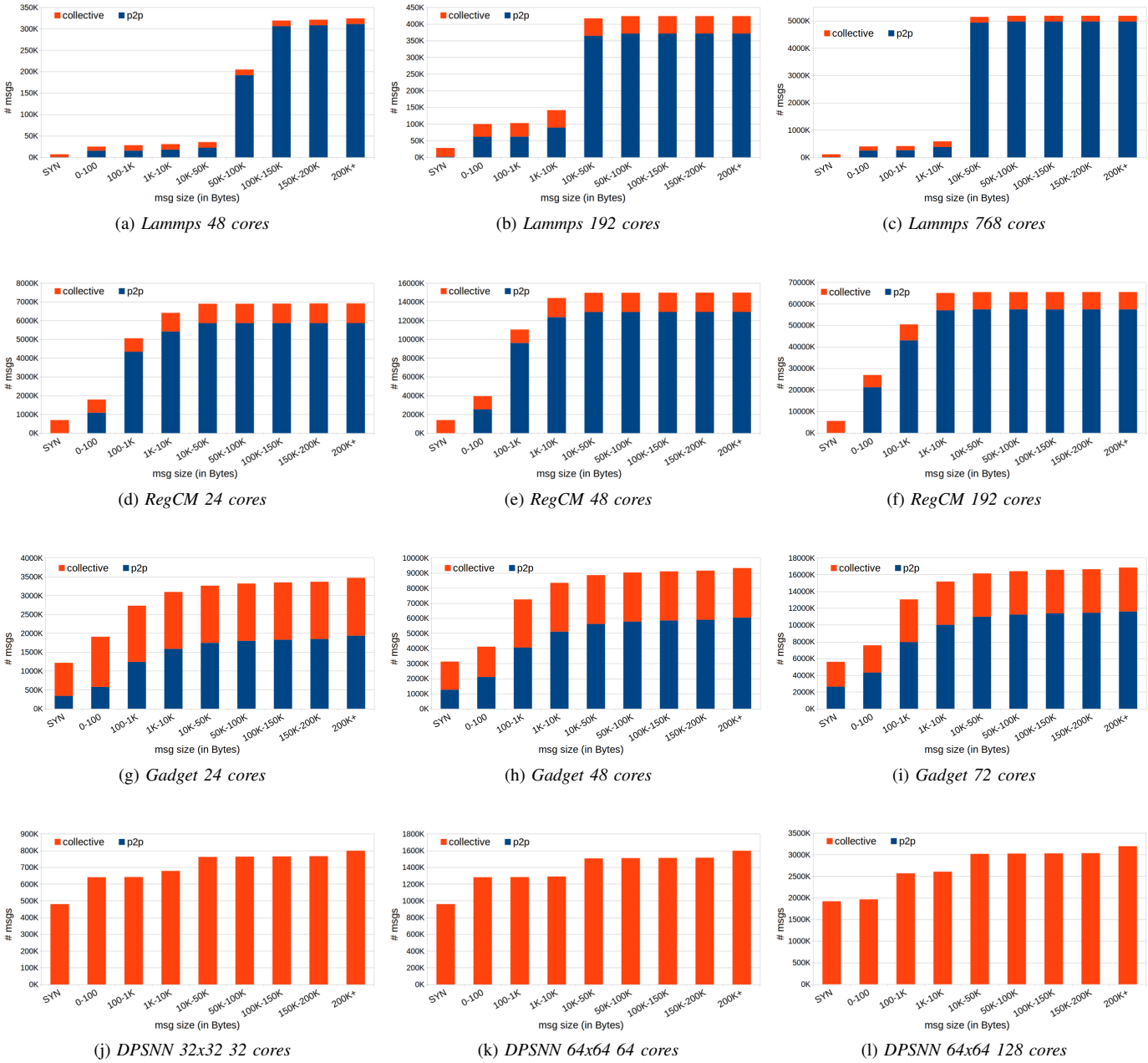
Figure 2 shows the cumulative message size distribution varying the core count across the studied traces. The Y-axis indicates the amount of messages (both due to collective and point-to-point MPI primitives) transferred and the X-axis the message size distributed in ranges. The first column (labelled as SYN) refers to the number of synchronization messages, whose main characteristic is that they do not include payload; however, they are also analyzed because, as results will show, they can generate a considerable amount of traffic in some applications.

Regarding message size distribution in Lammps (see Figures 2a-2c), it can be appreciated that most of the message sizes fall in between 10KB and 50KB for a core count larger or equal than 192. In contrast, for 48 cores, the dominating message size ranges from 50KB to 100KB. In summary, on average, the lower the core count, the larger message sizes are used.

Figures 2d-2f plot the message size distribution in the traces from the RegCM application. The message size in this application is quite homogeneous when varying the core count. The dominant size ranges from 100B to 1KB in all traces, although the 192-core trace also presents a significant amount of smaller messages (e.g from 0 to 10KB).

Gadget (Figures 2g-2i) shows a high percentage of synchronization messages regardless of the number of cores. As observed, a significant amount of messages (ranging from 30% to about 50%) present a size smaller than 1KB, although around 20% of messages are bigger.

Finally, Figures 2j-2l depicts the message size distribution in the traces from DPSNN. The first plot (Figure 2j) corresponds to the application working with 32x32 neural columns (about 1.2M neurons and 2.6G synapses), while the two remaining plots refer to the application with 64x64 neural columns (about 5M neurons and 10G synapses). On average, the traffic generated by synchronization messages represents as high as by 60% of the total amount. Unlike the previous



(a) *Lammps 48 cores* (b) *Lammps 192 cores* (c) *Lammps 768 cores*

(d) *RegCM 24 cores* (e) *RegCM 48 cores* (f) *RegCM 192 cores*

(g) *Gadget 24 cores* (h) *Gadget 48 cores* (i) *Gadget 72 cores*

(j) *DPSNN 32x32 32 cores* (k) *DPSNN 64x64 64 cores* (l) *DPSNN 64x64 128 cores*

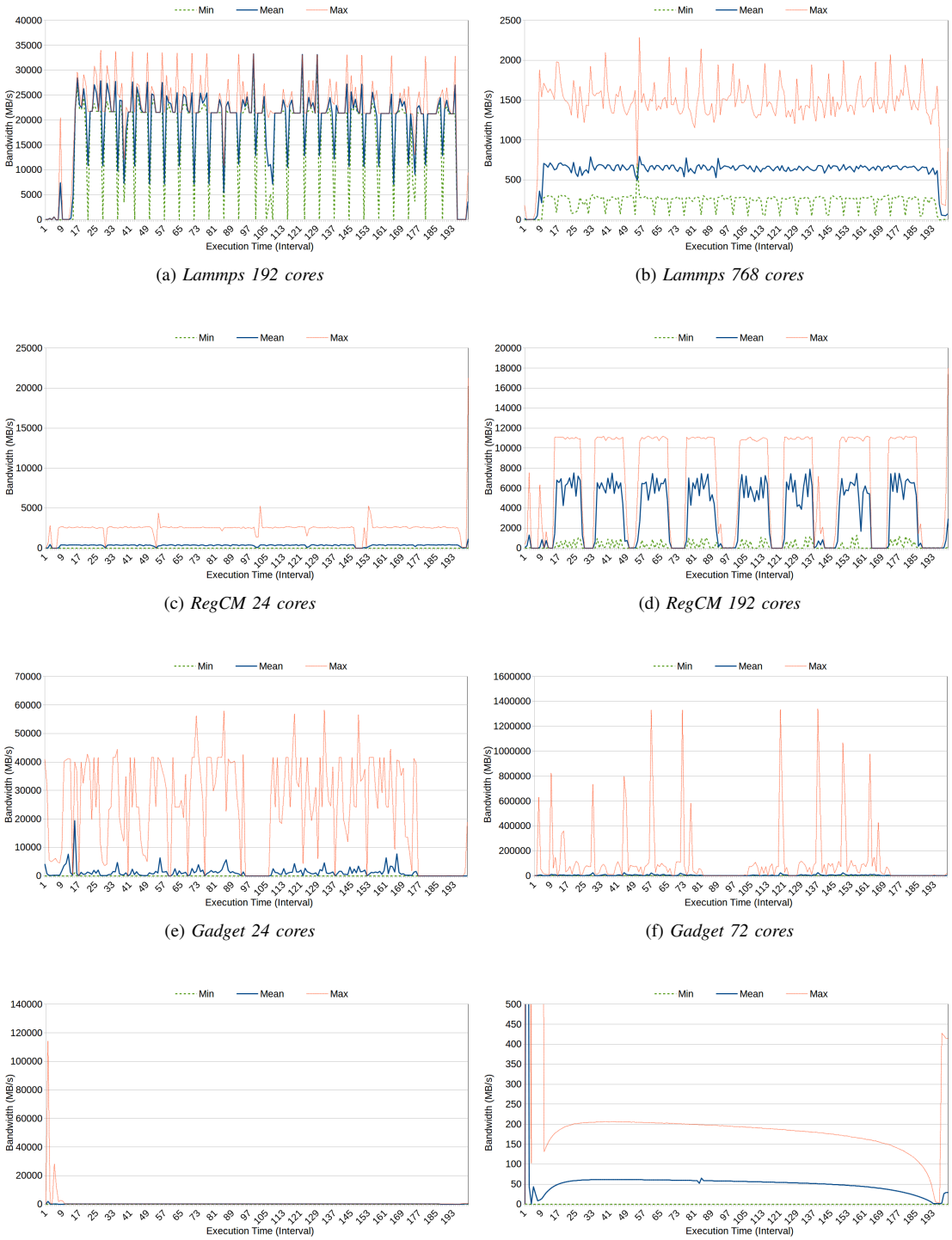
Fig. 2: Message size distribution.

analyzed traces, the traffic in this application is due to MPI collectives instead of point-to-point messages.

VI. Analysis of Temporal Evolution

In Section IV (Table I) we showed the bandwidth consumption for each studied trace averaged along the application execution time. The aim of this section is to analyze the dynamic bandwidth requirements in order to explore how requirements evolve with time.

We analyze all traces of each application and we found that the behavior of some applications changes when varying the number of cores. For illustrative purposes, we show an example of each of the multiple patterns exhibited by the studied applications when varying the core count. To ease the visual analysis and to homogenize the representation of the plots, we divided the execution time of each application in 200 intervals of the same length. Then, we divided each interval in 10M-cycle subintervals and calculated the bandwidth consumption of each subinterval. These values are averaged to obtain the bandwidth consumption of each interval, which is labeled as Mean in the figures. In addition, the maximum (Max) and minimum (Min) bandwidth consumptions among the subintervals are also plotted to easily discern bursts communication patterns.



(a) *Lammps 192 cores*

(b) *Lammps 768 cores*

(c) *RegCM 24 cores*

(d) *RegCM 192 cores*

(e) *Gadget 24 cores*

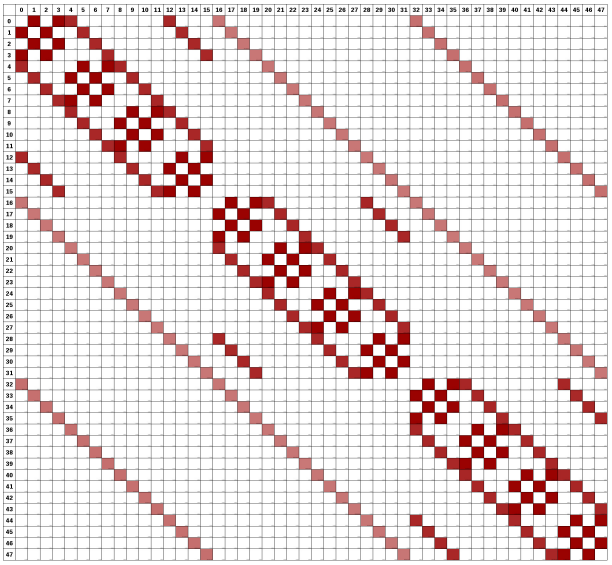
(f) *Gadget 72 cores*

(g) *DPSNN 64x64 128 cores*

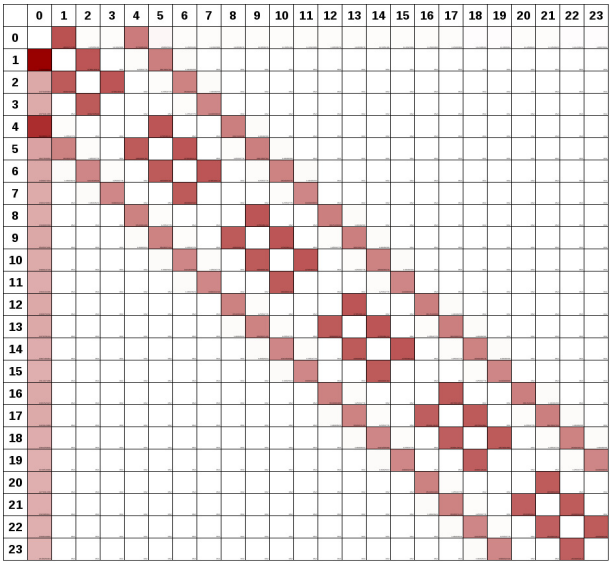
(h) *Zoomed Y axis of DPSNN 64x64 128 cores*

Fig. 3: Temporal evolution of min, mean and max bandwidth.

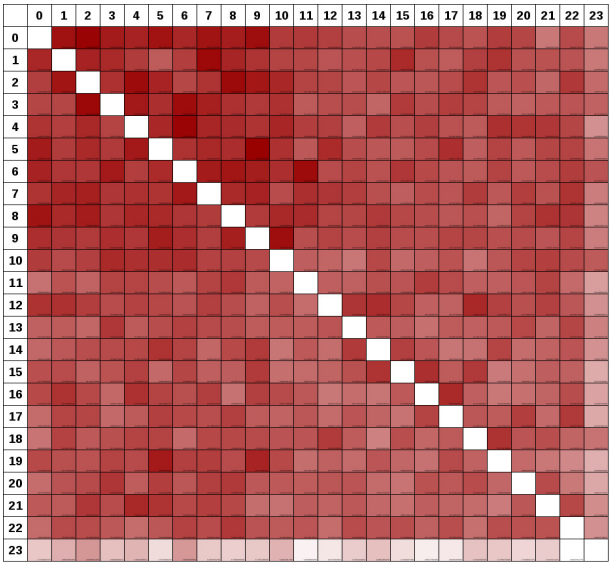
Figure 3a and Figure 3b show the bandwidth patterns of Lammps for 192 and 768 cores, respectively; which are representative of all the patterns of this application. Figure 3a shows the behavior exhibited for a core count less or equal than 192 while the other figure shows the behavior exhibited when the number of cores rises over 192. It can be appreciated that in the former case, the studied variables many times overlap each other. However, when the number of cores rises over 192 there is a clear differentiation among the three plotted variables. As expected (see Table I), the network traffic is much less important in 768 cores than in 192. Since the difference between the maximum and the average is



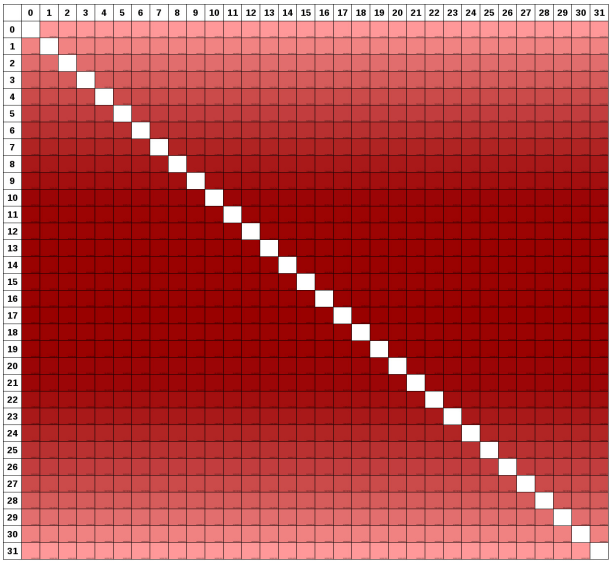
(a) *Lammps 48 cores*



(b) *RegCM 24 cores*



(c) *Gadget 24 cores*



(d) *DPSNN 32x32 32 cores*

Fig. 4: Communication matrix among cores.

relatively large, we made a further refinement by focusing on the most critical interval, that is, the interval with the highest difference.

The RegCM application exhibits similar bandwidth patterns regardless of the core count with the only exception of the 24-core trace. Figure 3c and Figure 3d show both patterns. We chose the 192-core trace which is the one presenting the highest average bandwidth.

Compared to Lammps, RegCM presents much higher bandwidth requirements; more than 8x on average and a similar factor for the maximum bandwidth. An interesting observation is that bandwidth experiences a sharp rise at the end of the execution in both exhibited behaviors. Regarding Figure 3d, RegCM shows on average less bandwidth requirements than Lammps (by 40%) and similar maximum bandwidth requirements.

Figure 3e and Figure 3f show the temporal evolution of Gadget. This application presents a huge difference between the average and the maximum, which implies a bursty communication pattern; that is, there are sub-intervals with high communication requirements and others presenting very low traffic.

DPSNN exhibits an homogeneous behavior across all the studied traces. Because of this reason, we chose the trace presenting the highest bandwidth requirements, that is, a 64x64 neural matrix with 128 cores. Figure 3g shows the results. In comparison to the previous applications, it can be observed that DPSNN has low bandwidth requirements during almost all the execution. However, in the first intervals,

DPSNN presents huge bandwidth consumptions. Because of this fact, we had to reduce the Y axis in Figure 3g to appreciate the average as shown in Figure 3h.

VII. Analysis of spatial communication

Once the applications across the execution time have been analyzed, this section studies the amount of traffic that each core sends/receives to/from each other. In other words, the spatial distribution of communication among cores (matrix source-destination).

Figure 4 shows the resulting matrix of communications for the four studied applications. The darker the color the higher the amount of transferred bytes. It can be seen that the traffic concentrates on a small percentage of cores in Lammps and RegCM, whereas it spreads among all the cores in DSPNN and Gadget. In DSPNN the traffic follows a regular pattern (darker in the central cores and lighter in the top and bottom cores), while in Gadget cores communicate all-to-all in a random way.

VIII. Conclusions

Workload characterizations are required to guide researchers in the design of new systems. In this paper, we have analyzed real traces of applications used in the European project ExaNeSt, which are being used to design and implement the interconnection network for an exascale system.

The analysis has been performed taking into account three main characteristics: the distribution of message types and sizes, the bandwidth consumed during the execution time and the spatial communication among cores.

Regarding the analysis of distribution of messages, most applications (three out of the four studied) present a higher amount of point-to-point messages, although one of the applications (DPSNN) is completely dominated by MPI collectives. In general, most messages are below 50KB regardless of the workload size.

The analysis of the bandwidth consumed during execution time indicates that applications present a wide range of average bandwidth requirements; however, most applications present bursty communications patterns that can stress the interconnection network at given points of time.

Finally, the analysis of the spatial communications matrix for the different applications shows very different spatial communication patterns among applications. For instance, in some applications the traffic is spread among all the cores whereas in some others bandwidth consumption is concentrated in hot spots. This means that, in order to support communication bursts and unclog congested network links, a suitable exascale network must provide higher-than-average bandwidth in the surroundings of key cores at specific points of time.

Acknowledgments

This work was supported by the ExaNest project, funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 671553, and by the Spanish Ministerio de Economía y Competitividad (MINECO) and Plan E funds under Grant TIN2015-66972-C5-1-R.

References

[1] A. K. Kodi, B. Neel, and W. C. Brantley, "Photonic interconnects for exascale and datacenter architectures," *IEEE Micro*, vol. 34, no. 5, pp. 18–30, 2014.

[2] S. Rumley, D. Nikolova, R. Hendry, Q. Li, D. Calhoun, and K. Bergman, "Silicon photonics for exascale systems," *Journal of Lightwave Technology*, vol. 33, no. 3, pp. 547–562, 2015.

[3] M. Katevenis, N. Chrysos, M. Marazakis, I. Mavroidis, F. Chaix, N. Kallimanis, J. Navaridas, J. Goodacre, P. Vicini, A. Biagioni *et al.*, "The exanest project: Interconnects, storage, and packaging for exascale systems," in *Digital System Design (DSD), 2016 Euromicro Conference on*. IEEE, 2016, pp. 60–67.

[4] (2018, May) ExaNeSt website. [Online]. Available: http://exanest.eu

[5] F. J. Ridruejo Perez and J. Miguel-Alonso, "Insee: An interconnection network simulation and evaluation environment," in *Proceedings of the 11th International Euro-Par Conference on Parallel Processing*, ser. Euro-Par'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 1014–1023.

[6] J. Navaridas, J. Miguel-Alonso, J. A. Pascual, and F. J. Ridruejo, "Simulating and evaluating interconnection networks with {INSEE}," *Simulation Modelling Practice and Theory*, vol. 19, no. 1, pp. 494 – 515, 2011, modeling and Performance Analysis of Networking and Collaborative Systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1569190X1000184X

[7] J. Duro, S. Petit, J. Sahuquillo, and M. E. Gómez, "Modeling a photonic network for exascale computing," in *High Performance Computing & Simulation (HPCS), 2017 International Conference on*. IEEE, 2017, pp. 511–518.

[8] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface*. MIT press, 1999, vol. 1.

[9] C. Concatto, J. A. Pascual, J. Navaridas, J. Lant, A. Attwood, M. Lujan, and J. Goodacre, "A cam-free exascalable hpc router," ser. Architecture of Computing Systems (ARCS), 2018, p. to appear.

[10] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of computational physics*, vol. 117, no. 1, pp. 1–19, 1995.

[11] F. Giorgi, E. Coppola, F. Solmon, L. Mariotti, M. Sylla, X. Bi, N. Elguindi, G. Diro, V. Nair, G. Giuliani *et al.*, "Regcm4: model description and preliminary tests over multiple cordex domains," *Climate Research*, vol. 52, pp. 7–29, 2012.

[12] P. S. Paolucci, R. Ammendola, A. Biagioni, O. Frezza, F. L. Cicero, A. Lonardo, E. Pastorelli, F. Simula, L. Tosoratto, and P. Vicini, "Distributed simulation of polychronous and plastic spiking neural networks: strong and weak scaling of a representative mini-application benchmark executed on a small-scale commodity cluster," *arXiv preprint arXiv:1310.8478*, 2013.

[13] V. Springel, "The cosmological simulation code gadget-2," *Monthly notices of the royal astronomical society*, vol. 364, no. 4, pp. 1105–1134, 2005.

[14] (2018, May) scalasca website. [Online]. Available: http://www.scalasca.org

[15] S. Derradji, T. Palfer-Sollier, J.-P. Panziera, A. Poudes, and F. W. Atos, "The bxi interconnect architecture," in *High-Performance Interconnects (HOTI), 2015 IEEE 23rd Annual Symposium on*. IEEE, 2015, pp. 18–25.