A Case Study on the Communication and Computation Behaviors of Real Applications in NoC-based MPSoCs

Zhe Wang, Weichen Liu, Jiang Xu, Bin Li†, Ravi Iyer†, Ramesh Illikkal†, Xiaowen Wu, Wai Ho Mow, Wenjing Ye
The Hong Kong University of Science and Technology, †Intel Labs, Hillsboro, OR, USA.

***Abstract*—Network-on-chip (NoC) based multiprocessor system-on-chips (MPSoCs) have been proposed as promising architectures to meet modern applications' ever-increasing demands for computing capability under limited power budget. Understanding the behaviors of MPSoC applications is the key to design MPSoCs under tight power and performance constraints. In this case study, we systematically examine the computation and communication behaviors of four real applications on MPSoCs based on three popular NoC topologies. We formally model real multiprocessor applications as task communication graphs (TCG) to accurately capture their computation and communication requirements. We publicly release a multiprocessor benchmark suite called COSMIC online, which includes the TCG models. In this work, we analyze the spatial distributions of workloads and traffics for each application, and evaluate their performance and energy efficiency on various MPSoC architectures. Our study shows that fat tree based MPSoCs are good choices for applications requiring high network throughput.**

I. Introduction

Multiprocessor system-on-chips (MPSoCs), can improve computing power per energy and lower cost per function to efficiently meet the ever-increasing computation requirements of emerging applications. The performance of an MPSoC is decided by both the performance of each processor, and the efficiency of the data communications among them. As the degree of chip integration and complexity of applications grow, the demand for on-chip communication bandwidth also increases, and often at a even faster pace. Traditional buses or ad-hoc interconnects can hardly satisfy this demand. Network-on-chip (NoC) [1], as a scalable alternative with better modularity than the conventional architectures, are becoming a promising choice for the on-chip communication architectures of MPSoCs.

A well designed NoC-based MPSoC should be able to not only provide sufficient computing power for targeted applications, but also efficiently accommodate the traffics generated. The computation and communication behaviors of MPSoC applications provide useful knowledge for designing computation and communication subsystems for MPSoCs. Synthetic models were used to capture the characteristics of real applications for architecture design and evaluation in early works. As researchers realize that the traditional synthetic approaches can hardly faithfully reflect the requirements of real applications [2], more and more designers are starting to analyze the behaviors of real applications. Barrow-Williams *et al.* examined the communication and memory sharing behaviors of real applications selected from existing benchmark suites [3]. Gratz *et al.* analyzed realistic application traffic behaviors based on a new set of metrics [4]. And many works provided application characterization methodologies [5]–[7]. These works mostly selected applications from multi-threaded benchmark suites, such as SPLASH-2 [8] and PARSEC [9]. However, there are only a small number of existing multiprocessor systems or simulators that can execute high-level programs, such as Tilera Tile64 [10] and GEM5 [11], but even for these multiprocessor platforms, the choices of network architectures are limited, and simulation runtime is long.

This work is partially supported by INTEL11EG01, GRF620911, GRF620512, and DAG11EG05S.

In this work, we conduct a case study on the computation and communication behaviors of four real applications executed on MPSoCs in different scales and three regular network topologies, *i.e.*, mesh, torus and fat tree. The applications are named *FFT-1024_complex, RS-32_28_8_dec TURBO-dec* and *MD-144_gs*. Descriptions for these four applications are provided in section III. A formal computation model, called task communication graph (TCG), is developed to capture the computation and communication requirements of applications. We systematically generate the TCG models for a set of real applications. We publicly release a multiprocessor benchmark suite called COSMIC (Communication-Observant Schedulable Memory-Inclusive Computation), which includes the application TCG models, the high-level programs of the applications, sample input datasets for each application and a tool for task scheduling and memory allocation. Using TCG models in addition to real application programs allows us to bypass operating systems (OS) and compilers to quickly explore novel multiprocessor architectures, such as MPSoCs. The COSMIC Benchmark Suite is online available at [12].

We analyze the spatial distributions of workloads and traffics for each applications, and evaluate their execution performance and energy efficiency on various MPSoC architectures based on the results obtained from cycle-level simulations. The experimental results show that, compared to *RS* and *TURBO*, *FFT* and *MD* can be better parallelized over multiple processors and benefit more from the increase of processors. On the other hand, the workloads of *RS* and *TURBO* are distributed mainly onto several neighbouring processors, and increasing the number of processors will hardly improve the performance but incur large amount of leakage overhead which reduces the energy efficiency significantly. Based on the experimental results, fat tree network based MPSoCs are good choices for applications requiring high network bandwidth and whose traffic patterns are of low locality. Moreover, the results suggest that not all applications need as many processing resources as possible to improve performance or energy efficiency, but may have some bounds that further increasing processors will not benefit the execution. The main contributions of our work includes:

- we develop the task communication graph model to faithfully capture real application characteristics and generate TCG models for real applications which are publicly released in the COSMIC benchmark suite.
- we comprehensively analyze the computation and communication behaviors as well as energy efficiency of the four multiprocessor applications executed on different sized MPSoCs based on three popular NoC topologies.

The rest of this paper is organized as follows. In Section II, we illustrate the application evaluation process. Section III listed the evaluation setup in the experiments. The experimental results and related analysis are presented in section IV. Section V concludes the work.

II. Real applications evaluation methods

An overview of the evaluation method is shown in Fig. 1. The evaluation process starts with the real multiprocessor application algorithms. We analyze the algorithms of the real applications, implement them in high-level languages, such as C and C++, or find their existing implementations, and develop different sample inputs for each application. Based on the analysis and implementation, we perform computation and communication behavior modeling to generate TCG model for each real application. To execute the modeled applications on multiprocessor platforms including MPSoC, two necessary steps are required, one is the allocation of proper memory resources for each application, and the other is the scheduling of application tasks onto different processors for execution. Since MPSoC hardware systems are performance-sensitive and subject to tight constraints, we specifically optimized these two steps to improve the performance. After this, we execute the applications on a SystemC based cycle-level MPSoC simulator to evaluate the performance and energy efficiency for each application under various MPSoC architectures.

A. Application modeling

The applications are modeled by a weighted directed acyclic graph model, called task communication graph. The TCG is defined as a tuple $G_t = (V, E)$, where V is the set of weighted vertices denoting the computational tasks, and E is the set of weighted directed edges denoting the communication channels among tasks. Each task $v_i \in V$ has a worst-case execution time t_i , which is measured in the number of clock cycles. Each edge $e_i = (v_{i,s}, v_{i,d}, w_i) \in E$ has a source task $v_{i,s}$, a destination task $v_{i,d}$ and the amount of data w_i sent

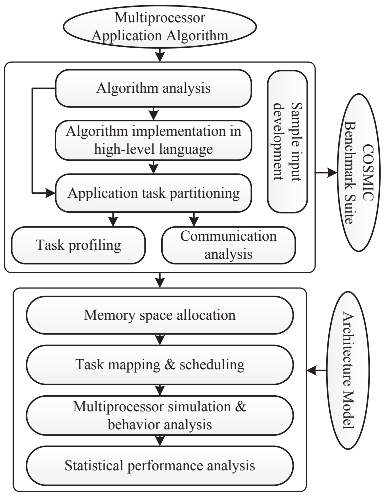

Multiprocessor Application Algorithm
Algorithm analysis
Algorithm implementation in high-level language
Application task partitioning
Sample input development
COSMIC Benchmark Suite
Task profiling
Communication analysis
Memory space allocation
Task mapping & scheduling
Multiprocessor simulation & behavior analysis
Statistical performance analysis
Architecture Model

Fig. 1. An overview of the multiprocessor application evaluation method.

from $v_{i,s}$ to $v_{i,d}$, which is measured in number of words (32 bits in this work). A TCG example is showed in Fig. 2, in which the number next to each task refers to the worst-case execution time, and the number on each edge refers to the communication volume. There are 11 tasks and 15 edges in this TCG.

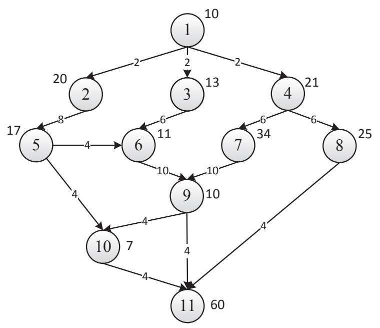


Fig. 2. An example task communication graph.

We take the application *TURBO* as example to better illustrate the application modeling process in the following subsections. In this work, we derive the TCG model based on the turbo decoding application in 3GPP standard [13], the block size of which is 40 bits and it adopts double sets of parity checking bits and the Viterbi algorithm.

1) Application partitioning: Applications are partitioned into various tasks for execution. Generally, the applications are partitioned into tasks in different manners according to specific needs of designers. We systematically analyze the algorithm of each application to help decide how to properly partition the application into multiple tasks. In this work, we partition the applications into multiple tasks in a fine-grain manner in order to facilitate potential parallelism exploration in the execution. Besides, fine-grain models could be transformed by exploiting the task clustering or grouping techniques, to merge multiple elementary tasks into larger ones based on specific needs or design requirements. In our work, not all the instructions are packaged into some tasks, instructions performing structural operations, such as those updating loop iterators and interacting with memories, will be translated as inter-task relations and reflected in the structure of the TCG. The instructions inside a task must be executed sequentially in the same processor without preemption. Besides this, for conditional bodies (*e.g.* "if-else" in C), all instructions contained inside are treated as one task to comply with the definition of a directed acyclic graph. Taking the application *TURBO* as example, the application is divided into 7 types of tasks: data input (I), floating point multiplication (M), three-input addition (A), forward state metrics calculation (F), backward state metrics calculation (B), extrinsic information calculation (E), interleaver/deinterlever (I/D).

2) Dependency graph generation: Task dependencies are mainly reflected by data communication between different tasks. So the dependency analysis are conducted in pairwise. At code level, tasks with the same operands or using pointers referring to the same memory address have potential dependencies. And for the loop entity, as TCG is acyclic, each iteration needs to be unfolded and the interactions among different iterations are extracted as the task dependencies. And for tasks wrapped as functions, the function calls and their inner sub-calls are completely analyzed to decide all possible dependencies. A part of the dependency graph for the application *TURBO* is showed in Fig. 3. In this figure, nodes represent the tasks previously mentioned.

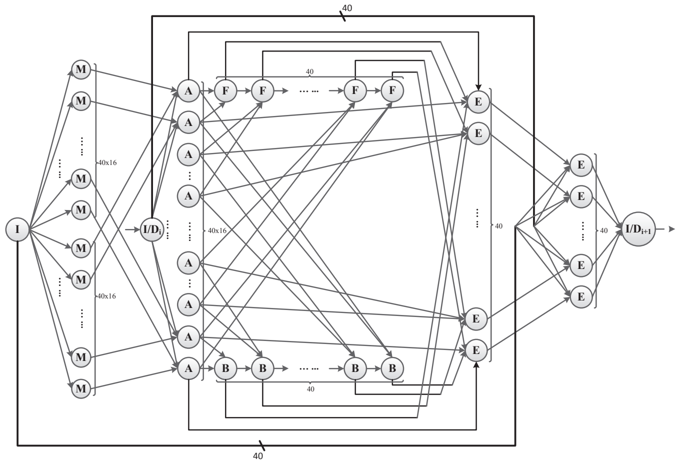


Fig. 3. Dependency graph of *TURBO-dec*.

3) Task profiling: The partitioned task programs are compiled and executed on a processor simulator called SimpleScalar [14] to obtain the worst-case execution time and output data volume of each task. The SimpleScalar processor emulates PISA architecture, which is similar to MIPS-IV ISA [15]. Performances based on SimpleScalar serve as comparison baselines for new processors, accelerators, and other processing units.

The worst-case task execution time is measured in number of clock cycles instead of absolute timing for the ease of cycle-level simulation. Since the theoretical worst-case execution time is difficult to determine, we resort to empirical approaches to approximate the worst case. Firstly the application is executed for multiple times with different input datasets and every time the execution time for each task is recorded. We choose the largest execution time for each task and add it with a 10% margin to be the final worst-case time approximation. The communication volumes between tasks are calculated mainly by recording the variables used for transferring data between task functions, including function parameters and global variables. There are two special cases here, one is dealing with user-defined data types, and the other is the use of pointers. In both of these situations, we statically calculate the sizes of the specific memory spaces defined in the source program, as it is difficult to determine the size of the memory referred to by the labels of data structure or pointers.

B. Memory space allocation and task mapping and scheduling

We assume that a virtual private memory space is assigned to each edge to store the data generated by the source task. Many applications, such as the *RS* and *TURBO*, run iteratively rather than for once in real systems, thusly insufficient memory allocation can limit the application throughput and affect the overall performance. Hence it is very beneficial to properly determine the memory requirements on each communication edge such that the performance can be maximized with minimum memory resources allocated in total.

In this work, we adopt the genetic algorithms to achieve this goal based on an existing technique proposed in [16]. And there are two objectives to be optimized for: maximizing application throughput is with higher priority, and minimizing the total memory allocated is with lower priority. Basically, we iteratively search for a satisfiable result by checking if the theoretical throughput with memory constraint is met.

For task mapping and scheduling, the basic principle is trying to distribute computation and communication workloads over the whole system evenly while the overall execution time can be minimized. The tasks are evaluated and assigned one by one in the order defined be the dependency relationships in the TCG. For each task v in V, the weight of assigning it to a processor p in processor set P is calculated in the following cost function:

$$w(v,p) = c_1 \times t(v,p) + c_2 \times q \times n(v,p), \tag{1}$$

in which $t(v,p)$ is the required time for task v to finish execution on p, defined by the time for executing previously assigned tasks on p plus the execution time of v on p, and $n(v,p)$ is the total amount of network transmission, defined by the number of packets generated by v and sent to other PBs. q is an architecture-specific scaling factor which balances the two terms which can be measured in different units, and c_1, c_2 are constant factors which can be manually adjusted to tradeoff the weight of the two parts in the overall cost.

III. Evaluation setup

Four real applications are modeled into TCGs, and we list their details in Table. I. We run the applications on different homogeneous MPSoC architectures comprising 4, 8, 16, 32, 64 processors interconnected by fat tree, mesh and torus based NoCs, specifically the processors are organized as 2x2, 2x4, 4x4, 4x8, 8x8 matrices in mesh and torus topologies. Sample

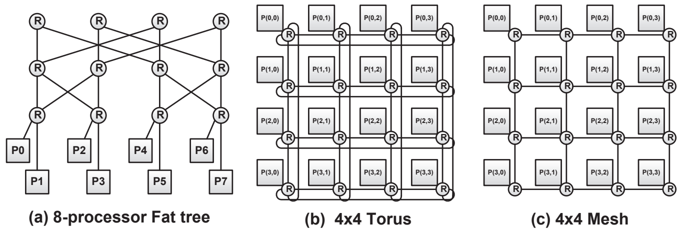


Fig. 4. MPSoCs with three different regular-topology NoCs.

diagrams for these three topologies are showed in Fig. 4. Each router is connected to the neighbouring routers or local processor with bidirectional channels. We implement the NoC based MPSoCs on a SystemC based platform, and conduct a detailed cycle-level simulation to evaluate their performance. All the experiments are conducted on the host system with Intel Xeon CPU E5-2687W processors running Linux version 2.6.18.

TABLE I
INFORMATION OF THE REALISTIC APPLICATIONS EVALUATED

Application	Description	No. tasks	No. edges
FFT-1024_complex	Fast Fourier Transform with 1024 complex number inputs	16384	25600
RS-32_28_8_dec	Reed-Solomon code decoder with codeword format RS(32,28,8)	182	392
TURBO-dec	Turbo code decoder in 3GPP protocol with block size 40-bit and rate 1/3	33257	86208
MD-144_gs	Molecular dynamic simulating the gas molecular dynamics when shotted towards surface composing of 144 solid atoms	11320	53803

The technology we target here is 45nm, and the frequency of electrical components including routers, links and processors, is set to be 1.25 GHz. For all three types of networks, packet switching and wormhole routing scheme is adopted, and a fixed packet size of 8 flits with 32 bits per flits (same as the width of each router port) is assumed. For mesh and torus based networks, XY routing algorithm is applied, and turn around routing is used for fat tree networks. The routers are working in a pipelined manner and we assume three cycles delay for flits to cross a router. Two virtual channels are implemented to avoid the head-of-line (HOL) problem. Link delay is confined in one cycle, and the bandwidth is 40 Gbps. The above configurations and timing assumptions are commonly applied in NoC studies. The power models of the processing and communication components are derived from [17]. Specifically, the average dynamic power is 37.5mW and leakage power is 10mW for each processor. For the network energy model, on average the crossbar consumes 0.06 pJ/bit, the buffer consumes 0.003 pJ/bit and the control unit consumes 1.5 pJ to make decision for each packet.

IV. EXPERIMENTAL RESULTS

We firstly show the computation and communication behaviors of four applications, which is reflected by the spatial distribution of the computation intensities and communication traffics during one execution iteration. Then we evaluate the system performance, including the network performance and overall execution performance, for each application on MPSoCs under different network topologies and scales. At last we show the energy consumption of *MD* executed on various MPSoC architectures and analyze the energy efficiency.

A. Computation and communication patterns of applications

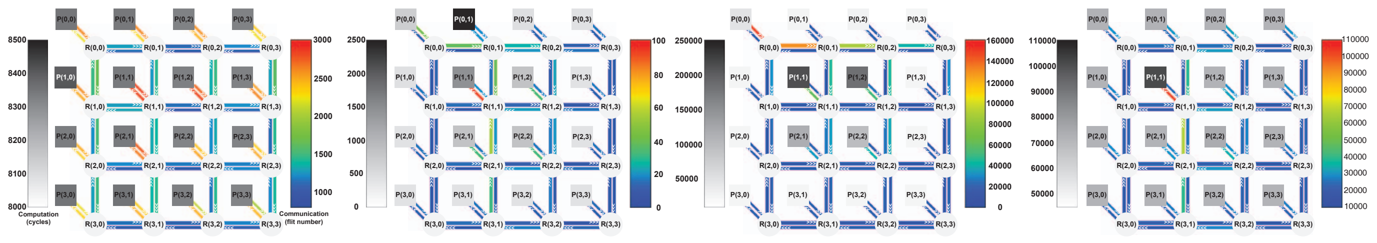
We exhibit the spatial distribution of the computation workloads and communication traffics of each application. For space limitation, only the results of 4x4 mesh based MPSoC are visualized in Fig. 5. In the figures, the color on each link refers to the number of flits going through and the grey scale for each processor (marked as P(x,y) in the figures) refers to the computation intensity measured in cycles. We can find that the workloads of *FFT* and *MD* are distributed more evenly than those of *TURBO* and *RS*. The workloads of the latter two applications are more centralized, where computation workloads are mostly processed by one processor or several processors closed to each other, for example, 67% of the workloads of *TURBO* are laid on P(1,1), P(1,2), P(2,1) and P(2,2). For the traffic patterns, outstanding hot links exist for all applications except for *FFT*, and these links are mostly connected to the processors with large amount of workloads. These distribution patterns could help decide the overloaded processors or links in MPSoCs, and make specific augmentations to avoid potential performance degradation.

We also derived the average (over three topologies) standard deviation (SD) of workloads on processors in Fig. 6 to reflect the variance degree of workload spatial distribution on each sized MPSoC architecture. The results here are relative values with respect to the average workload per processor on each sized MPSoC. A general trend is that with the growth of the number of processors, the workloads are distributed more unevenly, and this situation is more remarkable for *TURBO* and *RS*. This shows that *FFT* and *MD* can better exploit the computing resources on multi-processor systems, which indicates that the two applications are of relatively high parallelism.

B. Performance evaluation

In the experiments, each application is executed in a pipelined fashion for 20 iterations. The results for evaluation are measured from the 5th to 15th iteration, since the MPSoC is in a more stable state in this interval than those in the first and last 5 iterations in which the system may not be entirely loaded. All the performance results are the average value over these 10 iterations. We firstly evaluate the network performance for each application run on different MPSoCs in terms of *network throughput* and *end-to-end packet delay*. And then the overall performance is reflected by the *execution time of one iteration*.

Fig. 7 and Fig. 8 show the average network throughput and packet delay. And the value of each application run on 4-processor fat tree based MPSoC is normalized to 1, and the other results are relative values to these baselines


Computation (cycles)
Communication (flit number)
P(0,0) P(0,1) P(0,2) P(0,3)
R(0,0) R(0,1) R(0,2) R(0,3)
P(1,0) P(1,1) P(1,2) P(1,3)
R(1,0) R(1,1) R(1,2) R(1,3)
P(2,0) P(2,1) P(2,2) P(2,3)
R(2,0) R(2,1) R(2,2) R(2,3)
P(3,0) P(3,1) P(3,2) P(3,3)
R(3,0) R(3,1) R(3,2) R(3,3)

(a) FFT-1024_complex (b) RS-32_28_8_dec (c) TURBO-dec (d) MD-144_gs

Fig. 5. Spatial distribution of computation and communication workloads of the MPSoC applications.

correspondingly. Fig. 7 shows that the network throughput of *FFT* and *MD* are significantly improved with the growth of MPSoC size. But for the other two applications, the network throughput get limited improvement, or even slight degradation in some cases, for example the network throughput of *RS* run on 32-processor fat tree system is around 10% less than that on 16-processor fat tree system. In Fig. 8, the packet delays for most of these applications are reduced as the number of processors and network scales get larger. This is mainly due to lower traffic density in larger networks. This effect benefits *FFT* and *MD* more apparently than the other two applications. For example, the packet delay of *FFT* run on 64-processor mesh system is more than 86% less than that on 4-processor mesh system. But for the other two applications, the results show great variance.

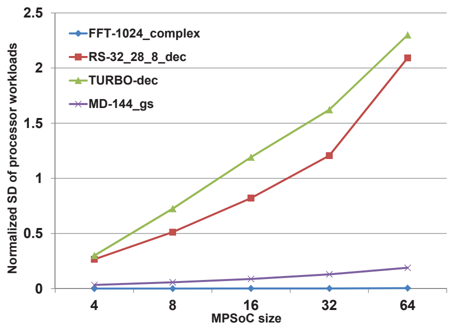


Fig. 6. Normalized standard deviation of processor workloads of each application on different sized MPSoCs.

The network performance variances exhibited by different applications can be partially explained by the different patterns of workloads spatial distribution analyzed above. For *RS* and *TURBO*, since the workloads are concentrated on a small number of processors, the associated traffics can hardly spread over the whole networks. While for *FFT* and *MD*, the workloads are distributed more evenly, thus the network resources are better utilized, which results in the better network performance than the other two.

Fig. 9 shows the normalized average execution time of one iteration for each application on MPSoCs with different network topologies. The result of each application on 4-processor fat tree based MPSoC is normalized to 1. An interesting observation is that both *FFT* and *MD*, which are of high parallelism, get better performance on the MPSoC with fat tree based network, while the other two applications perform worst on fat tree based MPSoCs. For example, *FFT* and *MD* achieve respectively on average 6.9% and 5.3% better performance on

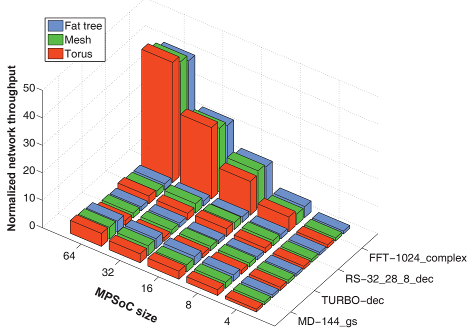


Fig. 7. Normalized average network throughput of each application run on mesh/torus/fat tree-based MPSoCs with different sizes.

MPSoCs with fat tree based networks in contrast with mesh and torus, while *RS* and *TURBO* perform on average 2.5% and 9.5% worse conversely. Fat tree based network provides larger network bandwidth but longer average end-to-end transmission distance. These features make it more suitable for the applications that can be highly parallelized onto different processors for execution and require high network bandwidth, such as *FFT* and *MD*. But the workloads of *RS* and *TURBO* are highly centralized and the traffics are of high locality, therefore the networks providing smaller end-to-end distance will fit them better, such as mesh and torus, than fat tree.

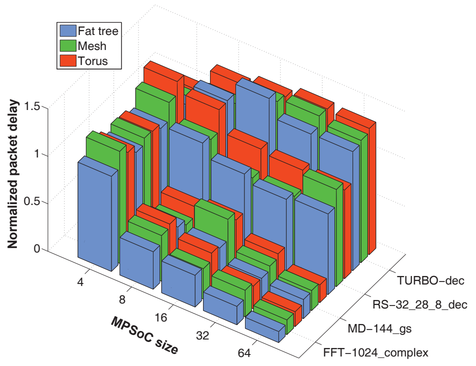


Fig. 8. Normalized average packet delay of each application run on mesh/torus/fat tree-based MPSoCs with different sizes.

On the other hand, regarding the changes of MPSoC sizes, *FFT* almost get proportional performance improvement with the processor number grow exponentially. *MD* firstly get significant performance enhancement, but the performance curve tends to get saturated with further increasing the number of processors. For *RS* and *TURBO*, the performance even degrades as the MPSoC sizes are enlarged. These results imply that applications may not need as many processor resources as possible to improve performance, and there could be such bound that further increasing processors will not bring or bring much performance benefit.

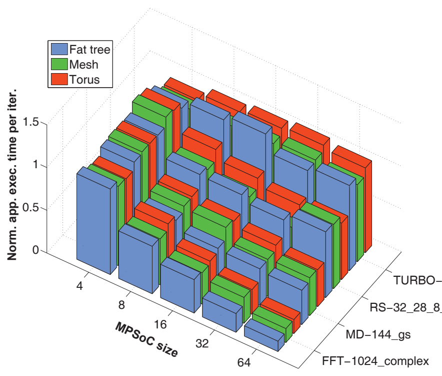


Fig. 9. Normalized average execution time per iteration of each application run on mesh/torus/fat tree-based MPSoCs with different sizes.

C. Energy evaluation

Due to space limitation, only the energy consumptions of *MD* run on different MPSoC architectures are showed in Fig. 10. In this figure, the result of 4-processor fat tree based MPSoC is normalized to 1. The energy consumption are divided into three parts: the communication energy (Comm.), processor working energy (Proc. busy) and processor idle energy (Proc. idle) which is resulted from static leakage. As the number of processors gets larger, the energy consumption of communication and idle processors increase apparently. For example, the communication energy and leakage energy on 64-processor fat tree based MPSoC take 23.9% and 48.2% of the total energy consumed. Keep increasing the amount of processing resources brings little performance improvement due to the application's intrinsic parallelism limitation, but incurs significant leakage overhead. Based on the results, we can also derive the performance-to-energy ratio index by dividing the reciprocal of the normalized execution time per iteration by the normalized energy consumption, which is especially helpful for high performance MPSoC design with tight power budgets. For *MD*, 8-processor fat tree based MPSoC achieves the highest index value of 1.65, while for larger MPSoCs, this value is smaller, for example it is 0.78 for 64-processor fat tree MPSoCs, which means the performance gain can not make up for the extra energy consumed.

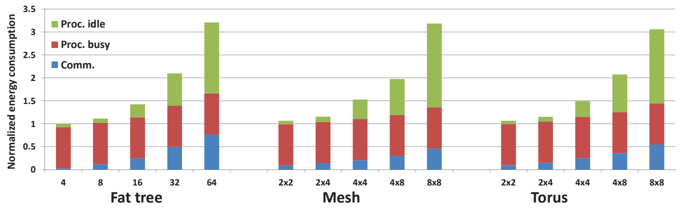


Fig. 10. Normalized energy consumption of *MD* run on mesh/torus/fat tree-based MPSoCs with different sizes.

V. Conclusion

In this work, we formally model real applications as TCGs and publicly release the COSMIC multiprocessor benchmark suite, which includes the application TCG models, the high-level programs of the applications, sample input datasets for each application and a tool for task scheduling and memory allocation. We systematically examine the computation and communication behaviors of 4 real applications on different sized NoC-MPSoCs based on three popular NoC topologies. We analyze the spatial distribution of workloads and traffics of each application, and evaluate their performance and energy efficiency on various MPSoC architectures. The experimental results show that fat tree based MPSoCs are good choices for applications requiring high network bandwidth and whose traffic patterns are of low locality, such as *FFT* and *MD*. These two applications can benefit a lot from the large amount of processing resources, and achieve relatively high energy efficiency. Moreover, the results suggest that not all applications need as many processor resources as possible to improve performance or energy efficiency, but may have some bounds that further increasing processors will not benefit significantly.

References

[1] J. Xu, W. Wolf, J. Henkel, and S. Chakradhar, "A methodology for design, modeling, and analysis of networks-on-chip," in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, May 2005, pp. 1778–1781 Vol. 2.

[2] W. Liu, J. Xu, X. Wu, Y. Ye, X. Wang, W. Zhang, M. Nikdast, and Z. Wang, "A NoC Traffic Suite Based on Real Applications," in *IEEE Computer Society Annual Symp. VLSI*, 2011.

[3] N. Barrow-Williams *et al.*, "A communication characterisation of Splash-2 and Parsec," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, oct. 2009, pp. 86 –97.

[4] P. Gratz and S. W. Keclker, "Realistic Workload Characterization and Analysis for Networks-on-Chip Design," in *The 4th Workshop on Chip Multiprocessor Memory Systems and Interconnects (CMP-MSI)*, 2010.

[5] S. Chodnekar, V. Srinivasan, A. Vaidya, A. Sivasubramaniam, and C. Das, "Towards a communication characterization methodology for parallel applications," in *High-Performance Computer Architecture, 1997., Third International Symposium on*, 1997, pp. 310–319.

[6] D. Feitelson and L. Rudolph, "Parallel application characterization for multiprocessor scheduling policy design," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, 1996, vol. 1162.

[7] P. Bogdan and R. Marculescu, "Workload characterization and its impact on multicore platform design," in *CODES+ISSS, 2010 IEEE/ACM/IFIP International Conference on*, 2010, pp. 231–240.

[8] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, "The SPLASH-2 programs: characterization and methodological considerations," in *ISCA, '95, Proceedings.*, june 1995, pp. 24 –36.

[9] C. Bienia, "BENCHMARKING MODERN MULTIPROCESSORS," Ph.D. dissertation, Princeton University, 2011.

[10] S. Bell, B. Edwards *et al.*, "Tile64 - processor: A 64-core soc with mesh interconnect," in *ISSCC 2008. Digest of Technical Papers. IEEE International*, 2008, pp. 88–598.

[11] N. Binkert, B. Beckmann *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[12] COSMIC Benchmark Suite: http://www.ece.ust.hk/∼eexu/.

[13] http://www.3gpp.org/.

[14] T. Austin *et al.*, "Simplescalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, Feb. 2002.

[15] C. Price, *MIPS IV Instruction Set: Revision 3.0.* MIPS Technologies, 1994.

[16] W. Liu *et al.*, "Satisfiability Modulo Graph Theory for Task Mapping and Scheduling on Multiprocessor Systems," *IEEE Transactions on Parallel and Distributed Systems*, 2010.

[17] Y. Wang *et al.*, "Power gating aware task scheduling in mpsoc," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 10, pp. 1801–1812, Oct 2011.