SAMi: Self-Aware Migration Approach for Congestion Reduction in NoC-based MCSoC

Amin Rezaei[1], Masoud Daneshtalab[2], Dan Zhao[3], Mehdi Modarressi[4]
[1] University of Louisiana at Lafayette (ULL), Lafayette, USA (me@aminrezaei.com)
[2] Mälardalen University (MDH) and Royal Institute of Technology (KTH), Sweden (masdan@kth.se)
[3] Old Dominion University (ODU), Norfolk, USA
[4] University of Tehran (UT), Tehran, Iran (modarressi@ut.ac.ir)

***Abstract*—Many-Core System-on-Chips (MCSoCs) require efficient task migration approach in order to reach system performance objectives such as load balancing, communication optimization, fault tolerance, and temperature control. In this paper an efficient self-aware migration approach is introduced for NoC-based MCSoCs using a centralized feedback controller in order to control the congestion over the system. The proposed approach is divided into four main steps: predicting behavior of the application, defining reliable triggers to initiate task migration, introducing cost comparison functions, and presenting a streamlined controlling mechanism to migrate tasks. The experimental results affirm that the proposed self-aware migration approach can help achieving significant throughput and system utilization while efficiently controlling system congestion.**

Keywords—MCSoC; NoC; Task Migration; Feedback Controller; Congestion; Performance

I. INTRODUCTION

Nowadays, by way of extremely high energy costs, single-core high-frequency processors are less considered and manufacturers are moving toward designing multi and many-core chips [1]. Thus, commercial Many-Core Systems-on-Chips (MCSoCs) are available based on Network-on-Chip (NoC) interconnection network [2-4]. NoC provides a regular platform for connecting system resources and makes the communication scalable and flexible compared to traditional bus or hierarchical bus architectures [5]. Moreover, according to the International Technology Roadmap for Semiconductors (ITRS) [6], NoC-based MCSoCs may integrate hundreds of Processor Elements (PEs) in near future. Such complex systems will confront an exceedingly dynamic workload. Applications, as sets of communicating tasks, will enter and leave the system at run-time. Therefore, run-time task mapping for future MCSoCs is a challenging issue.

Significant variation of application behavior places an upper-bound on improving the performance by primary task mapping. This makes some mapping strategies take behavior variation into consideration [7]. Task migration is a dynamic task re-mapping mechanism, which follows real-time variation of workload behavior. Task migration in a MCSoC is described as transferring a task from the source core where it is currently running to a destination core and then resuming its execution there in such a way that some system performance objectives are improved.

The overall performance of a NoC-based system is highly correlated to the network congestion [8]. Congestion increases the network latency severely [9] and also increases the network power consumption significantly [10]. Thus, in this paper a Self-Aware Migration (SAMi) approach for NoC-based MCSoCs is presented in order to control system congestion. The key contributions of the proposed strategy are:

- Predicting end-to-end traffic of the system based on application behavior;
- Defining triggers for initiating task migration based on average core congestion and average regional congestion of the network;
- Introducing cost functions for specifying the possible destination cores of the task migration process;
- Presenting a streamlined feedback controller for monitoring and managing the task migration process.

The rest of the paper is arranged as follows. Section II addresses backgrounds and previous works. A self-aware migration approach for NoC-based MCSoCs is proposed in Section III. Section IV depicts and discusses experimental results. Lastly, the conclusion and the proposal for future works are given in Section V.

II. BACKGROUNDS AND PREVIOUS WORKS

Since task mapping is known as an NP-hard problem, different heuristics such as Nearest Neighbor (NN) and Best Neighbor (BN) [11-12] addressed dynamic workload management in many-core systems. In these heuristics, the first node to start mapping is selected based the on assumed clustering nodes. In another approach called Incremental Approach (INC) [13], the mapping problem is divided into two phases: the region selection, and the task allocation. In the region selection phase, the algorithm starts from the closest node to the Central Manager (CM) and includes it in the region. Then, it iteratively adds nodes to the selected region trying to keep both the selected region and remaining nodes contiguous. Then, in the task allocation phase, application tasks are mapped within the selected region.

Due to significant vibration of application behavior, after initial task mapping, some dynamic task re-mapping mechanisms are required to improve the system performance at run-time. Task migration has been traditionally studied in

978-1-5090-1367-8/16/$31.00 ©2016 IEEE

distributed systems for dynamic load balancing. However, with the increasing popularity of MCSoCs in modern embedded systems, task migration has also gained research attention in this domain and has been studied for different purposes [14]. In [15] a lightweight migration mechanism for bus-based MCSoC is presented. The task migration method relies on modification of the program to define the checkpoints. When running to a checkpoint, the program checks whether there is a migration request for current task. Authors in [16] proposed a methodology using virtual channels to create temporary circuit connections that provide low latency and low power communication paths for the task migration flows. They adopted a 2D-mesh NoC, creating sub-meshes which may contain one or more cores. In [17] a task migration protocol is presented. Task may be migrated at any moment, not requiring migration checkpoints, and its context is also migrated.

However, in previous works there is a lack of centralized controlling platform for managing the migration process. Furthermore, just a few of previous works have considered the effect of the trigger initiating a migration, as an important factor of task migration mechanisms. Moreover, predicting application behavior in order to adaptively define the threshold values for task migration, is another area that has rarely been addressed in the literature. A proper migration trigger based on application behavior can improve performance while keeping the complexity of migration algorithm low. The key contribution of this paper is to present an efficient self-aware migration approach for NoC-based MCSoCs taking advantage of application behavior and using control theory.

III. Proposed Task Migration Approach

In this work, NoC-based MCSoC is represented by an architecture graph $AG(C, L)$. The AG contains a set of cores $c_i \in C$, which are connected together through unidirectional links $l_k \in L$. Each $c_i \in C$ has a set of running tasks Tc_i on it. Task migration is defined as transferring a weighted task $(t_{sj}, w_{sj}) \in Tc_s$ from source core $c_s \in C$ to destination core $c_d \in C$ and then resuming its execution. The weights are computed based on the communication demand of each task.

A. Application Behavior Prediction

A general approach for predicting the future is to capture the past behaviors. Most existing works rely on some counters for capturing past behaviors, due to the simplicity and low area overhead of such counters [18]. There are two predictors supported by SAMi: short-term and long-term predictors. The short-term predictor anticipates based on the recent information stored in $\text{counter}_{si,dj}$, where $(t_{si}, w_{si}) \in Tc_s$ and $(t_{dj}, w_{dj}) \in Tc_d$ are the tasks communicating with each other. On the contrary, the long-term predictor anticipates based on the pattern of communication experiences. The prediction function is defined as follows:

$$P_t(si, dj) = \begin{cases} A_{t-1}(si, dj), & sel(si, dj) = 0 \\ table_{si,dj}(history_{si,dj}), & sel(si, dj) = 1 \end{cases} \quad (1)$$

Where $A_t(si, dj)$ is the actual traffic and $P_t(si, dj)$ is the predicted traffic from task $(t_{si}, w_{si}) \in Tc_s$ to task $(t_{dj}, w_{dj}) \in Tc_d$ at the t-th interval. Also, $history_{si,dj}$ is the record of the communication pattern between tasks $(t_{si}, w_{si}) \in Tc_s$ and $(t_{dj}, w_{dj}) \in Tc_d$ while $table_{si,dj}$ is the long-term prediction table, in which each entry is indexed by $history_{si,dj}$ and includes a prediction rate. The prediction rate traces the amount of data transmitted when this pattern was encountered last time. If the same pattern appears again, the traced value is used as the prediction rate. The anticipation either comes from the short-term predictor or the long-term predictor, decided by a selector function sel. The selector function is designed according to the system requirements.

ALGORITHM 1. SAMi Algorithm

th_c: core threshold value
th_r: region threshold value
C: set of cores of the system
c_s: source core
c_d: destination core
R: set of regions of the system
Cr_i: set of cores within the region i
$C(c_i)$: core congestion value of core c_i
$R(r_i)$: region congestion value of region r_i
Tc_i: set of tasks and their communication weights running on core c_i
(t_{sj}, w_{sj}) : a running task j and its weight on c_s

```
while true
  set the values of th_c and th_r
  if there is a c_s ∈ C that C(c_s) > th_c then
    choose c_d ∈ C with min[ C(c_d)]
    while there is a (t_sj, w_sj) ∈ Tc_s OR C(c_s) ≤ th_c
      choose (t_sj, w_sj) ∈ Tc_s with maximum weight
      name c_s after migrate (t_sj, w_sj) as c_s(new)
      if C(c_s) − C(c_s(new)) + C(c_d) ≤ th_c then
        migrate (t_sj, w_sj) to c_d
      else
        remove (t_sj, w_sj) from Tc_s
      end
    end
  end
  if there is a r_s ∈ R that R(r_s) > th_r then
    choose c_d ∈ C − Cr_s with minimum [ C(c_d)]
    while there is c ∈ Cr_s OR R(r_s) ≤ th_r
      choose c_s ∈ Cr_s with maximum [ C(c_s)]
      while there is (t_sj, w_sj) ∈ Tc_s OR R(r_s) ≤ th_r
        choose (t_sj, w_sj) ∈ Tc_s with maximum weight
        name c_s after migrate (t_s, w_s) as c_s(new)
        if C(c_s) − C(c_s(new)) + C(c_d) ≤ th_c AND R(r_s) − R(r_s(new)) + R(r_d) ≤ th_r then
        migrate (t_sj, w_sj) to c_d
        else
        remove (t_sj, w_sj) from Tc_s
        end
        remove c_s from Cr_s
      end
    end
  end
end
```

B. Trigger Definition

We define two triggers in order to control congestion of the system: Core Congestion (CC) and Region Congestion (RC). CC considers congestion over each single core, while RC deals with total congestion of a region. For the CC trigger, the migration starts when the amount of packets received by a core in the network reaches the core congestion threshold (th_c). The value for th_c is defined adaptively based on the average traffic of the cores and prediction of the application's behavior. For the RC trigger, NoC is divided into regions. The migration

starts when sum of the traffic in a region reaches the region congestion threshold (th_r). The value for th_r is defined adaptively based on the average traffic of the regions and prediction of the application's behavior. In order to avoid unnecessary task migration in low traffic patterns, a lower-bound is considered for th_c and th_r.

C. Core Comparison Function

A cost function is needed to determine the best destination core for task migration. For the CC trigger, the most weighted task on the source core will migrate to the destination core. The destination core is the one which has the minimum congestion among all other cores while satisfying the threshold constraint after the task migration. If there is more than one candidate as the destination, one of them is randomly selected. For RC trigger, the approach is almost similar. After determining the congested region, the most weighted task on the core which is the most congested core in the region migrates to a destination which in addition to the above conditions should be outside the region boundary of the source core. Algorithm 1 shows SAMi algorithm.

D. Congestion Control Platform

Basically each controller compares the system output with a target value. After comparison, it manipulates the system actuators to minimize the error. The controller policy to tune the actuators strongly depends on the dynamic model of the target system and the system robustness against error disturbance. The dynamic model defines how the system reacts to the inputs including actuations and other inputs. The system robustness is defined as the system stability against overshooting of the output values from the target intended output.

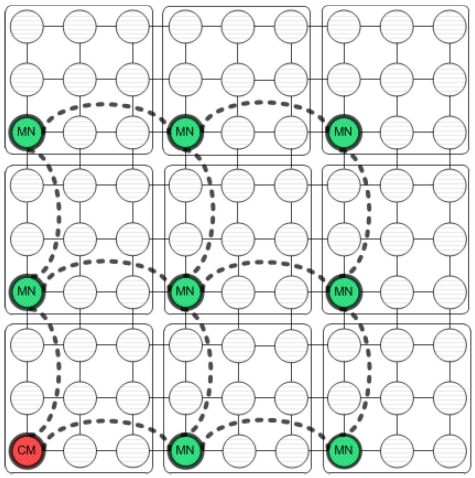

Fig. 1. A 9×9 NoC with 9 regions

In order to speed up the task migration process, there is a Manager Node (MN) in each region that is responsible for collecting information about all the cores of the region. These MNs have created a virtual mesh network based on [16] in order to control task migration process efficiently. In addition to MNs there is a Central Manager (CM) that is responsible to manage the whole network. Fig. 1. shows a NoC with 81 nodes and nine regions. Moreover, the proposed congestion control platform for a 2D mesh NoC-based MCSoCs is depicted in Fig. 2. The platform operates based on the Algorithm 1.

Congestion Meter: Each core measures the traffic dynamically by calculating the moving average of packet flow in every link of its router. Then, the congestion level of each router is transferred to its MN. MNs send these information to Core Congestion Meter (CCM). In addition, the sum of congestion level of each region is also transferred to the Region Congestion Meter (RCM) by MN of each region. CCM and RCM compute average core congestion level and average region congestion level respectively and send them to the threshold calculator.

Threshold Calculator: Threshold Calculator (TC) calculates the amounts of th_c and th_r based on the average core and region congestion and also prediction of the application behavior.

PID Controller: A Proportional Integral Derivative (PID) controller for actuator manipulation is employed. The general formula for the PID controller is as follows:

$$PID_{out}(t) = K_p e(t) + K_i \int e(t)\,dt + K_d \frac{de(t)}{dt} \qquad (2)$$

Where $PID_{out}(t)$, $e(t)$, K_p, K_i, and K_d are the controller output, error, proportional gain, integral gain, and derivative gain, respectively. The proportional part determines how fast or aggressive the controller reacts to changes in the input signal. The main function of the integral part is to ensure that the process output agrees with the set point value in steady state. The derivative part determines how the system reacts to changes in the reference value; It also enhances stability in the system [19].

Task Migration Manager: Task Migration Manager (TMM) performs the task migration based on the information from controller outputs and congestion vectors. When a core or a region marked as congested by one of PIDs' outputs, the TMM finds the best destination core for the task migration based on the congestion vectors. As it is shown in algorithm 1, the best destination is the one which has the minimum congestion among all other cores while satisfying the threshold constraint after the task migration.

E. Task Migration Overhead

In order to lower the overhead of task migration strategy, SAMi is implemented based on MCSoC Message Passing Interface [20], in which task mapping is independent of task re-mapping. By changing task mapping table, task is remapped to another core. Then task state information is transferred using special mesh network shaped by MNs, hence the migrated task can restore execution on a different core. The task migration contributes less communication overhead because task state information excludes task code; therefore, after choosing the destination, TMM transfers state information of the chosen task (e.g. most weighted one) from the source core to the destination core. The task then resumes its execution there. Hence, remapping a task of application might not affect the application execution.

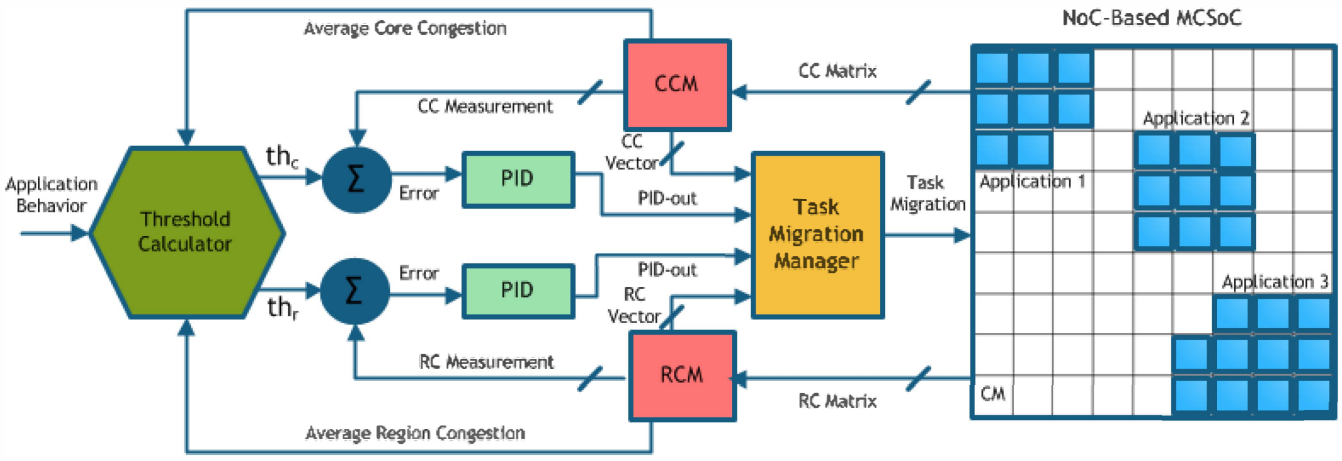

Fig. 2. Congestion control platform for NoC-based MCSoC

IV. Experimental Results

Experiments are performed on a cycle-accurate many-core platform implemented in SystemC. A pruned version of an open source simulator for mesh-based NoCs called Noxim [21] is utilized as its communication architecture. Moreover, the power model taken from [22] is integrated as library into the simulator. Some multi-threaded real-time applications from the PARSEC [23] benchmark suite as well as several sets of non-real-time applications with 4 to 35 tasks generated by TGG [24] with randomly distributed communication volumes are used in the experiments. The probabilities of selecting real-time and non-real-time applications from the application repository are 30% and 70% respectively.

Our many-core platform is enforced to support dynamic application mapping by implementing a CM residing in the $c_0 \in C$ (node $n_{0,0}$). In our NoC-based MCSoC system, a random sequence of applications enters the scheduler but this sequence is kept fixed in all experiments on account of fair comparison. Applications are scheduled based on First Come First Serve (FCFS) policy, and an application is scheduled if and only if there is enough available nodes in the system. An allocation request for the application is scheduled to send to the CM of the platform. CM selects the region method using INC [13], and maps the application based on its real-time attributes. The time critical applications are mapped contiguously and isolated using pre-analyzed patterns. The non-real-time applications are mapped onto the available cores. A successfully mapped application is allocated to the corresponding nodes, where each node emulates the behavior of its allocated task. In addition to the dynamic mapping unit, the congestion control platform (including TC, PID controllers, CCM, RCM, and TMM) is also implemented (i.e., soft coded) as a part of the CM. In most of the available dynamic application mapping techniques in the literature, one core is already dedicated to the CM. This makes the area overhead of our congestion control platform so negligible. CM receives the congestion feedback from the MNs and sends actuation commands back to them using a dedicated low overhead circuit switch mesh network.

After the predictability analysis, we compare the latency, throughput, system utilization, and power consumption of the baseline in which the task migration policy is disabled with the SAMi platform under the set of applications. We run three set of experiments shown in Table I.

TABLE I. System Specification

	NoC Size	Regions
1st Setup	12×12	9
2nd Setup	10×10	4
3rd Setup	8×8	4

A. Predictability Analysis

In order to predict traffic patterns, a sample period has to be decided for information gathering. The duration of a period considerably affects the precision of prediction. Actual accuracy of the history-based prediction depends on the similarity of the application behavior between the sample period and the period after task migration. In other words, choosing the suitable sample period for recording the communication pattern between tasks $(t_{si}, w_{si}) \epsilon Tc_s$ and $(t_{dj}, w_{dj}) \epsilon Tc_d$ is essential.

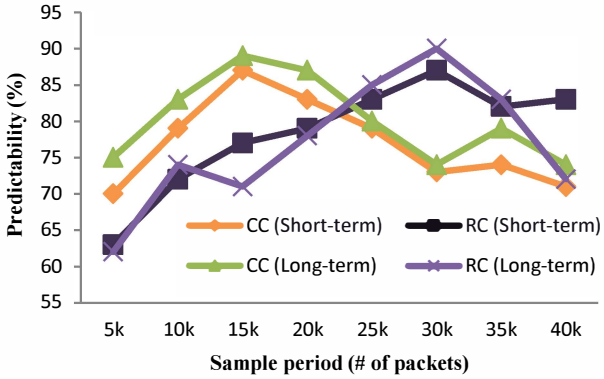

Fig. 3. Predictability analysis for the 3rd setup

Fig. 3 shows the average predictabilities of our applications categories using RC and CC triggers for the 3rd setup using both short-term and long-term predictors. The horizontal coordinate presents the sample period (in term of the number

of packets) for adapting threshold values. The best predictability of the applications is about 90% and the worst predictability is about 62%. We can see that the traffic is predictable, which provides great potential for run-time optimization of overall performance.

B. Latency

Fig. 4 shows the performance gains of using SAMi. The results indicate that SAMi reduces average packet latency in comparison with baseline NoC (e.g. down to 38% for the first experiment setup). This reduction is because of distributing congestion across the system. The average number of clock cycles that is elapsed in CM to map applications using INC [13] with number of tasks varying from 4 to 10 is presented in Table II.

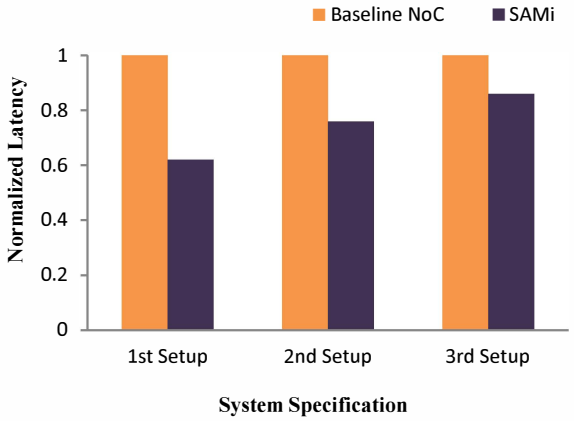

Fig. 4. Normalized latency

In addition, Table III shows the average number of clock cycles based on different task sizes for the 3rd setup. For example, the impact of migrating a 8 KB task is of 210 KCycles, which translates into a delay of 2.1 ms on the operating frequency of 100 MHz. Although, the overhead time of task migration is negligible, this delay should have considered in process' deadlines for real-time tasks.

TABLE II. TASK MAPPING OVERHEAD

Number of Tasks	4	5	6	7	8	9	10
Clock Cycles (KCycles)	66	120	151	182	207	263	297

TABLE III. TASK MIGRATION OVERHEAD

Task Size (KB)	1	2	4	8	16	32
Clock Cycles (KCycles)	45	87	147	210	462	974

C. Throughput

Fig. 5 compares the normalized throughput for the set of applications on SAMi and the baseline NoC with no congestion control platform. SAMi can significantly improve the overall system throughput for different technology nodes (e.g. up to 50% for the first experiment setup). In addition, SAMi becomes increasingly effective in terms of the throughput with advanced technology nodes.

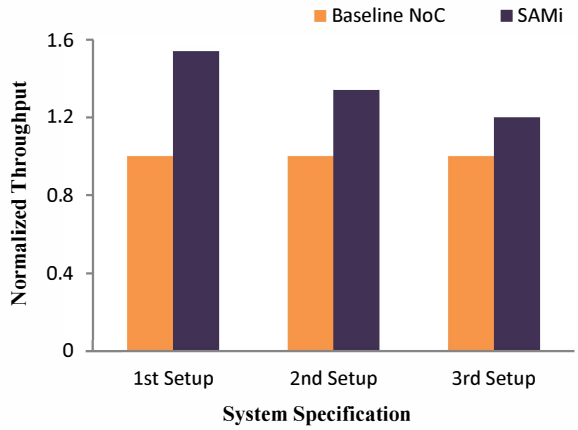

Fig. 5. Normalized throughput

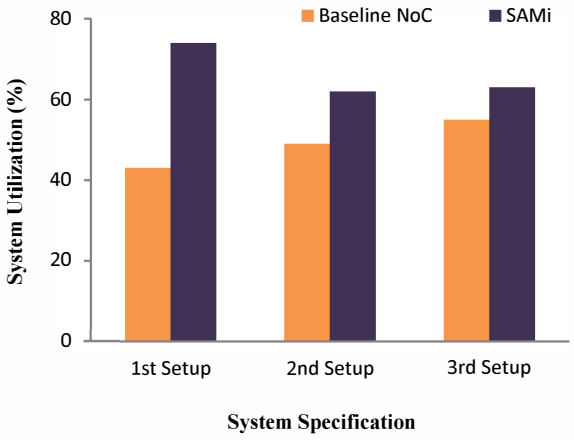

Fig. 6. System utilization

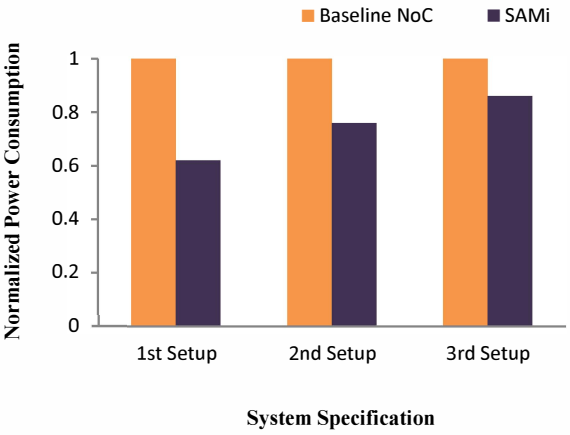

Fig. 7. Normalized power consumption

D. System Utilization

System utilization is another important factor to be analyzed between SAMi and the baseline NoC. As shown in Fig. 6, SAMi considerably increases the system utilization compared to the baseline NoC. Due to the area fragmentation that occurs because of dynamic mapping policy, SAMi cannot reach 100% utilization. (i.e. when applications does not exactly fit into the many-core system).

E. Power Consumption

Fig. 7 compares power consumption between SAMi and the baseline NoC. It can be noticed that, for all the setups, SAMi with task migration scenario consumes less power than the baseline NoC (e.g. down to 23% for the first experiment setup). Also for a larger NoC, the reduction of power consumption is larger when SAMi approach is applied.

V. Conclusion and Future Works

Congestion causes the many-core system to run sub-optimally, and increasingly inefficient with scale. Thus in this paper an efficient self-aware task migration approach, SAMi, depending on application prediction along with congestion triggers and cost functions was proposed for NoC-based MCSoCs. Then a congestion control platform based on a control loop feedback mechanism (i.e. PID controller) was introduced. Experimental results revealed that the proposed application-specific migration approach can help achieving significant throughput and system utilization as well as less latency and power consumption in comparison with the baseline NoC, while efficiently controlling system congestion. Furthermore, for a larger NoC, the reduction of both latency and power consumption is larger when SAMi approach is applied. In a simple word, SAMi has a better scalability than the baseline approach.

For future works, we are planning to minimize the task migration overhead for real-time tasks. Moreover, implementing SAMi in architectures that are specifically designed for dark silicon age (e.g. NoC-Sprinting [25] and Shift Sprinting [26]) seems to be beneficial.

References

[1] A. Rezaei, M. Daneshtalab, F. Safaei, and D. Zhao, "Hierarchical approach for hybrid wireless network-on-chip in many-core era," In *International Journal of Computers and Electrical Engineering*, Vol. 51, Issue C, pp. 225–234, 2016.

[2] "Adapteva, Inc." [Online]. Available: http://www.adapteva.com/.

[3] "Arteris, Inc." [Online]. Available: http://www.arteris.com/.

[4] "Sonics, Inc." [Online]. Available: http://sonicsinc.com/.

[5] A. Rezaei, M. Daneshtalab, M. Palesi, and D. Zhao, "Efficient congestion-aware scheme for wireless on-chip networks," In *IEEE Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP)*, pp. 742-749, 2016.

[6] ITRS. International Technology Roadmap for Semiconductors, 2011 edition.

[7] C. Wang, L. Yu, L. Liu, and T Chen, "Packet triggered prediction based task migration for network-on-chip," In *Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pp. 491-498, 2012.

[8] C. L. Chou and R. Marculescu, "Contention-aware application mapping for network-on-chip communication architectures," In *IEEE International Conference on Computer Design (ICCD)*, pp. 164-169, 2008.

[9] J. W. Brand, C. Ciordas, K. Goossens, and T. Basten, "Congestion-controlled best-effort communication for networks-on-chip," In *Proceedings of Design, Automation and Test in Europe (DATE)*, pp. 1-6, 2007.

[10] S. Ma, N. E. Jerger, and Z. Wang, "DBAR: an efficient routing algorithm to support multiple concurrent applications in networks-on-chip," In *Proceedings of International Symposium on Computer Architecture (ISCA)*, pp. 413–424, 2011.

[11] E. L. Carvalho, N. L. V. Calazans, and F. G. Moraes, "Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs," In *IEEE/IFIP International Workshop on Rapid System Prototyping (RSP)*, pp. 34-40, 2007.

[12] E. L. Carvalho, N. L. V. Calazans, and F. G. Moraes, "Dynamic task mapping for MPSoCs," In *IEEE Design & Test of Computers*, Vol. 27, Issue 5, pp. 26-35, 2010.

[13] C. L. Chou, U. Y. Ogras, and R. Marculescu, "Energy- and performance-aware incremental mapping for networks on chip with multiple voltage levels," In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 27, Issue 10, pp. 1866-1879, 2008.

[14] W. Quan and A. D. Pimentel, "A system-level simulation framework for evaluating task migration in MPSoCs," In *Proceedings of International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, Article 13, 2014.

[15] S. Bertozzi, A. Acquaviva, D. Bertozzi, and A. Poggiali, "Supporting task migration in multi-processor systems-on-chip: a feasibility study," In *Proceedings of Design, Automation and Test in Europe (DATE)*, pp. 1-6, 2006.

[16] B. Goodarzi and H. Sarbazi-Azad, "Task migration in mesh NoCs over virtual point-to-point connections," In *Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pp. 463-469, 2011.

[17] F. G. Moraes, G. A. Madalozzo, G. M. Castilhos, and E. A. Carara, "Proposal and evaluation of a task migration protocol for NoC-based MPSoCs," In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 644-647, 2012.

[18] Y. Huang, K.-K. Chou, C.-T. King, and S.-Y. Tseng, "NTPT: on the end-to-end traffic prediction in the on-chip networks," In *Proceedings of Design Automation Conference (DAC)*, pp. 449-452, 2010.

[19] S. Holmbacka, S. Lafond, and J. Lilius, "A PID-controlled power manager for energy efficient web clusters," In *Proceedings of IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC)*, pp. 721-728, 2011.

[20] F. Fu, S. Sun, X. Hu, J. Song, J. Wang, and M. Yu, "MMPI: a flexible and efficient multiprocessor message passing interface for NoC-based MPSoC," In *Proceedings of IEEE International SoC Conference (SOCC)*, pp. 359-362, 2010.

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

[22] L. Wang and K. Skadron "Dark vs. dim silicon and near-threshold computing extended results," Technical Report (UVA-CS-2013-01), Department of Computer Science, University of Virginia, 2013.

[23] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: characterization and architectural implications," In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 72-81, 2008.

[24] Task graph generator (TGG). [Online]. Available: http://sourceforge.net/projects/taskgraphgen/.

[25] J. Zhan, Y. Xie, and G. Sun,"NoC-sprinting: interconnect for fine-grained sprinting in the dark silicon era," In *Proceedings of Design Automation Conference (DAC)*, pp. 1-6, 2014.

[26] A. Rezaei, D. Zhao, M. Daneshtalab, and H. Wu, "Shift sprinting: fine-grained temperature-aware NoC-based MCSoC architecture in dark silicon age," In *Proceedings of Design Automation Conference (DAC)*, Article 155, 2016.