Job-Arrival Aware Distributed Run-Time Resource Management on Intel SCC Manycore Platform

Vasileios Tsoutsouras, Sotirios Xydis, Dimitrios Soudris

School of Electrical and Computer Engineering, National Tech. Univ. of Athens, Greece

Contact info: {billtsou, sxydis, dsoudris}@microlab.ntua.gr

Abstract—Modern computing systems are dealing with a diverse set of complex and dynamic workloads in the presence of varying job arrival rates. This diversity is raising the need for the development of sophisticated run-time mechanisms that efficiently manage system's resources. In addition, moving towards kilo-core processor architectures, centralized resource management approaches will most probably form a severe performance bottleneck, thus the study of Distributed Run-Time Resource Management (DRTRM) schemes is now gaining a lot of attention. In this paper, we propose a job-arrival aware DRTRM framework for applications with malleable characteristics, implemented on top of the Intel Single-Chip Cloud Computer (SCC) many-core platform. We show that resource allocation is highly affected not only by the internal decision mechanisms but also from the incoming application interval rate on the system. Based on this observation, we propose an effective admission control strategy utilizing Voltage and Frequency Scaling (VFS) of parts of the DRTRM which eventually retains the distributed decision making thus improving system performance in combination with significant gains in its consumed energy.¹

Index Terms—distributed resource management, run-time, many-core, Intel SCC

I. INTRODUCTION

Computing systems design is constantly becoming more and more complex to meet the increasing demands in performance, power consumption and system reliability [1]–[3]. Consequently, this has rendered the nature of applications to be dynamic, variable in workload and imposing run-time adaptivity requirements to the resource management design of the underlying hardware platform. To fulfil the aforementioned requirements, the resource allocation of the system is performed in run-time in an effort to optimally distribute the available resources to the applications and properly react to unpredictable events, such as changes in the arrival rate of applications or in available hardware resources due to system malfunction. This in turn has launched a shift in the system resource allocation paradigm traversing it from centralized to distributed runtime resource management (DRTRM) decision processes [4]–[6]. The advantages of this trend is its increased scalability, distribution of computational burden for decision making. Additionally, it enhance system reliability, by eliminating the existence if a single point of failure as in centralized approaches. For example, Al-Fareque et al. [7] proposed an agent-based framework to reduce peak temperature along with enhanced performance and reduced energy consumption. Agent's negotiations are based on a supply/demand economical model and through this they distribute power consumption on the system in a pro-active manner. Agent-based management is also utilized in [8] performing distributed task migration for thermal management. Neural

networks are used to produce prediction of peak temperature in cases of workload variation showing higher performance and reduced migration overhead compared to other centralized predictive dynamic thermal managers.

However, the nature of distributed decision making does come with an increased complexity in the resource management process. The absence of central management strategy forces all cores with managerial responsibilities, i.e. management agents, to communicate with each other via exchanged messages to resolve resource management requirements [5], [9], [10]. Furthermore, no single point has an overview of the platform leading to limited ability for the system to adjust in scenarios that the need for resources is stressed. In this paper, we examine and analyze the behavior of DRTRM under stress-marked scenarios, in which an excessive amount of applications arrive on the platform within a small time window and request admission and resource allocation. More specifically, we implemented a parametrised DRTRM framework on the Intel Single Cloud Chip Computer (Intel SCC) [1] in respect to i) application arrival rates and ii) underlying cluster topology, which enables an extensive exploration and characterization of the many-core system under dynamic/malleable workloads. Recently in [11], the authors studied the combined effects of job-arrival rate on application's parallelism degree, however their study focused on centralized resource management systems. To the best of our knowledge, this is the first study on DRTRM for manycore systems analysing and correlating the application arrival with platform partitioning decisions for dynamic application scenarios.

In addition, we show that DRTRM frameworks inherently exhibit unbalanced workload characteristics, since a subset of the available cores are dedicated only to management-related tasks, not concerning application workload execution. Although, their role is crucial, their tasks are not computationally intensive, thus forming promising candidates to enforce voltage and frequency scaling (VFS) techniques to reduce the total energy consumption. We utilize the aforementioned concept of VFS at the DRTRM level to efficiently regulate resource management requirements to tailor to system stress conditions, without compromising the distributed nature of our management framework. The proposed approach is evaluated across a variety of system configurations and workloads exhibiting a robust behaviour regarding workload regulations as well as enabling energy savings of 18% originating only from the internal components of the DRTRM.

The rest of the paper is organized as follows: Section II briefly introduces the architectural features of the targeted Intel SCC platform. Section III then describes in depth the structural components of the proposed parametrized job-arrival aware DRTRM framework on Intel SCC, i.e. workload malleability (Section III-A2) and workload arrival rate modeling (Section

III-A1). DTRM mechanisms are summarized in Section IV and admission control in IV-A. In Section V, we provide extensive experimental data inferring and characterizing the main manifested behaviours of DTRM on Intel SCC platform, while Section VI concludes the paper.

II. DESCRIPTION OF INTEL SCC PLATFORM

In this paper, we utilize the Intel Single Cloud Chip Computer [1] as the driver many-core platform. Intel SCC is a 48-core single chip platform with a mesh NoC interconnection as on-chip communication. It consists of 24 tiles with each one containing 2 processing cores of x86 instruction set. Tiles are connected through a 2D-mesh and a router inside each tile is responsible for forwarding each outgoing packet of the tile to the correct target. The tile also includes L2 cache memory for the two cores and a special memory called Message Passing Buffer which is the base to provide to the programmer a message passing interface through which fast and reliable data dispatching amongst different cores is achieved. The tiles of the platform are divided into 6 voltage islands and a Voltage Regulator Controller (VRC) included in SCC, provides the ability to regulate the operation voltage of each island separately. It is also possible to regulate the operating frequency at granularity of each tile and always in limits dictated by the voltage of the island the tile belongs to.

Each processing core runs its own lightweight Linux OS distribution, which significantly improves the functionality of each core since it is a full-operation system. Intel SCC comes with an API control library called RCCE which offers MPI primitives in order to develop applications on the entire SCC system. It also exposes to the user a safe API to work with the VFS capabilities, offering the ability to scale the frequency and voltage of the cores of an island by dictating a frequency/voltage divider pair, ranging from $< 800MHz, 1.1V >$ down to $< 100MHz, 0.7V >$.

The SCC platform is coming together with a power metering infrastructure, enabling the reporting of instant voltage and current values drawn by the many-core chip. For this paper, a custom power metering daemon program has been developed, which samples the power metering registers by periodically calling "sccBmc -c status" command. The gathered values are then numerically integrated over each time interval, i , to calculate the dissipated energy: $E = \sum V_i \times I_i \times \Delta t_i$ to acquire the sum of its consumed energy. A similar infrastructure has been used also in [12] to examine the impact of DVFS decisions on parallel MPI-based application execution. While their analysis focuses mainly on analysing single application instance mapped onto the SCC, we are considering more complex scenarios of multiple applications running in parallel focusing on the impact of DVFS on the internal structures of the DTRM.

III. JOB-ARRIVAL AWARE DTRM FRAMEWORK

In this section we analyze the proposed job-arrival aware DTRM framework implemented on top of Intel SCC many-core platform. Figure 1 shows its major components. It consists of i) the workload generator module that instantiates the incoming traffic, i.e. dynamic malleable applications trace according to differing arrival rate distributions, ii) the corresponding application queue constructed upon the arrival of a new application on the system, iii) the DTRM module that takes all the necessary actions to initiate the execution of the

Fig. 1: The proposed job-arrival aware DTRM framework.

new application and manage the system resources of the many-core and iv) the admission control module that regulates the dispatching of jobs to be mapped onto the targeted many-core. The proposed schema implements a feedback loop approach, in an effort to configure DTRM at run-time according to the rate of incoming applications and the state of the platform. In the rest of the section, we elaborate in more detail on each specific component.

A. Workload generator of dynamic malleable applications

1) *Modeling workload arrival rates:* We model applications arrival based on interval rates generated by probability density function of the Poisson random distribution. This distribution was employed, as in [11] as it sufficiently models a system with steady rate of incoming applications which at one point in time exhibits a sudden influx of applications requiring admission on the system. On this basis, we examined four different arrival rate scenarios by varying the λ coefficient of the Poisson distribution, i.e. $\lambda \in 16, 23, 48, 64$. The interval rate curve is presented in Fig. 2. X axis represents application id and Y axis is the interval between the i -th application and its following one. This form of application interval rate describes a system at which applications arrive at an almost steady rate and this rate sharply increases at a given point in time with different number of already admitted applications.

In addition to the Poisson derived intervals, a scenario was created where the intervals between two consecutive applications derive randomly from a range which was considered large enough in an effort not to stress the system. We refer to this scenario as slow and also provide its dual "fast"

Fig. 2: Interval rate of incoming application trace.

Fig. 3: Arrival times of application trace.

scenario where the range inside which the next application time is randomly chosen, was significantly diminished. Fig. 3 illustrates the application arrival times of "slow", "fast" and Poisson derived scenario with lambda coefficient equal to 48.

2) *Modeling workload malleability*: We evaluate the efficiency of DRTRM on dynamic workload trace based on a set of dynamic malleable applications [9], i.e. applications that dynamically can be resized up/down to more resources for increased speedup. We model and implement application malleability within an integer matrix multiplication code between a square matrix of size $M \times M$ and a vector of size $M \times 1$. Application resizing is enabled only at specific synchronization points between the W multiplication repetitions. Aiming to create computationally intensive applications, the core multiplication is repeated W times and this variable is considered the workload of the application. Thus, application speedup in respect to the available resources can be modeled as:

$$T_{rem} = W_{rem} * Exec_time(n, M) \quad (1)$$

where T_{rem} refers to the remaining execution time, W_{rem} refers to the remaining multiplication repetitions, n is the number of application's cores and $Exec_time$ is a function returning the execution time for one instance of the matrix multiplication for n cores. $Exec_time$ is dependent both on input dataset size as well as on the number of parallel processes mapped onto processing cores. Fig. 4 shows the execution delay of the employed workloads on the Intel SCC in respect to dataset sizes and allocated resources. In this paper, we force the DRTRM to allocate one parallel process per core, thus eliminating co-scheduling interference effects. This constraint is actually imposed by the micro-architectural features of SCC which does not support hyper-threading at the core level.

In the context of this work, each application is considered to be one distinct job. It is managed atomically, distributing its workload W to SCC worker cores and each one of them executes the multiplication of a subset of rows of the input square matrix with the input vector. The manager (Section IV) communicates to each worker the upper and lower bound of the consecutive rows of the input matrix it has to multiply. The workload is as evenly distributed as possible and the workers do not exchange any information with one another. Each worker core after completion writes in the memory of the

Fig. 4: Performance malleability model in respect to #cores and workload size for matrix size $M = 4096$.

manager core the computed results. In order to alleviate the overheads of memory block caching that limits the benefits of dynamic malleability, we carefully tiled the workload distributed to each core to fit the SCC node's cache capacity, while also enabling data pre-fetching to be performed during initialization of the application.

IV. THE DRTRM ARCHITECTURE

In DRTRM the cores alternate between a number of roles throughout their lifetime to support either managerial or workload executing responsibilities. We identify three types of cores:

- **Manager cores** that control the execution of an application by appointing workload to the working nodes, gathering the output of their execution and performing all the necessary actions to acquire more working cores for its application in an effort to speedup its execution. There is only one manager core per application and it does not execute any of the actual workload of the application.
- **Controller cores** that maintain a record of active managers in a certain core region, i.e. cluster of cores. This record is referred to as **Distributed Directory Service (DDS)**. In order for a core to acquire this information for a region of cores in the system, it must issue such a request to the respective controller cores. Controller cores serve as the manager cores of idle cores, i.e. whenever an application asks to increase its resources, the controller core can offer him one or more idle cores.
- **Initial cores** that are responsible to discover at least one processing core for a newly arrived application on the system to be executed on. There is one initial core per application, randomly assigned at run-time and it performs the search phase repeatedly until at least one has been offered for the new application. Once the new core is found, the new manager core is appointed and the initial core role completes execution.

This DRTRM scheme implicitly imposes an hierarchy to the roles that each core of the SCC platform can be assigned to. At the top level, the controller cores act as building blocks of the correct function of the DRTRM framework. At next level, the manager cores preside over application execution and subscribe their presence to the controller cores regional directory information. Initial and worker cores form the lowest level which are assigned to an application and frequently interchange their role. All these run-time roles (manager, initial, worker) are ideally appointed to cores which are idle and consequently the last level in the hierarchy is a pool of idle cores waiting to be appointed a certain task.

Algorithm 1 Controller core algorithm

```
1: procedure SIGNAL_HANDLER(sig_no, sender_id)
2:   if sig_no = DISCOVER_CNTR_CORES then
3:     region R  $\leftarrow$  Read_signal_data ()
4:     cntr_cores_list  $\leftarrow$  Discover_cntr_func (R)
5:     Send_reply (sender_id, cntr_cores_list)
6:   else if sig_no = REQUEST_DDS_INFO then
7:     region R  $\leftarrow$  Read_signal_data ()
8:     cores_list  $\leftarrow$  Request_DDS_info_func (R)
9:     Send_reply (sender_id, cores_list)
10:  else if sig_no = ADD_CORES_DDS then
11:    cores_list  $\leftarrow$  Read_signal_data ()
12:    Add_cores_DDS_func (sender_id, cores_list)
13:  else if sig_no = REMOVE_CORES_DDS then
14:    cores_list  $\leftarrow$  Read_signal_data ()
15:    Rem_cores_DDS_func (sender_id, cores_list)
16:  else if sig_no = REQUEST_FOR_CORES then
17:    req_app  $\leftarrow$  Read_signal_data ()
18:    my_offer  $\leftarrow$  Core_offer_func (req_app)
19:    Send_reply (sender_id, my_offer)
20:  else if sig_no = REPLY_TO_OFFER then
21:    offer_reply  $\leftarrow$  Read_signal_data ()
22:    my_cores_list  $\leftarrow$  Update_my_cores (offer_reply)
23:  end if
24: end procedure
25:
26: procedure CONTROLLER_CORE
27:   while TRUE do
28:     pause () /* Sleep until a new signal is received */
29:   end while
30: end procedure
```

Ultimately, the DRTRM framework aims at successfully executing incoming applications while maximizing the performance of the system. This is achieved by increasing the resources of each application in such a way that the overall performance is maximized in addition to serving as many incoming applications simultaneously as possible. At run-time, during specific intervals a negotiation round is performed between each two active application instances, $A_i, A_j, i, j \in \mathbf{A}$. We adopt the negotiation mechanism proposed in [10], i.e. a re-assignment of resources is performed between A_i, A_j whenever the following condition is evaluated to true, $Speedup(A_i, \#cores - 1) < Speedup(A_j, \#cores + 1)$, i.e. a core is reallocated to A_j if the speedup loss of A_i is lower than the speedup gain of A_j . The speedup projection is extracted by the performance malleability model depicted in Fig. 4.

The aforementioned negotiation mechanism applies to manager cores in order to re-size their applications, to initial cores in the process of instantiating a new application on the system and controllers in order to offer idle cores. For any core to communicate with each other user-defined signals are necessary. Algorithm 1 shows the signal handling in each controller. For Intel SCC, this user-defined signaling have been developed on top of the RCCE API utilizing the shared MPB memory buffers that allows data to be moved from core to core through the NoC infrastructure.

Finally, the distributed nature of DRTRM provides the ability of deriving different cluster topologies by changing the number of controller cores. To provide a consistent analysis on the impact of these differing topologies, we examined a number of controller cores up to 8². Since the combination of number of cores and cluster topologies, hereafter denoted

by $[\#CNTR_{cores}, \#Cluster]$, explodes the available design options, we examined only cases of 2, 4, 6 and 8 controller cores. Furthermore, for each number of controller cores, variations of allocated cores in each cluster are examined in order to quantify the impact of clusters' fragmentation on DRTRM efficiency. Fig. 5 shows the different cluster topologies examined with each cluster marked with a different color.

A. Adaptive Job Admission Control

The resource management framework should be able to adapt to the current situation of the platform i.e. the great number of applications asking for admission and new resources by the system. Such an adaptive behavior could be translated to a policy at the initial core level. For example, in such a case the frequency upon which initial core re-initiates its core searching cycle, should be decreased to meet the diminished available resources on the system.

Such an adaptation mechanism stumbles upon the nature of a purely distributed resource management framework. Inherently, it suffers from two major drawbacks:

- It requires a central decision to be made from a resource management point of view to assure effective adaptation to the incoming application demands. Consequently this creates a central point of information acquisition and possible failure which is intended to be avoided in distributed framework.
- It suffers from synchronization latency. Even if the designer decides to gather the necessary information to proceed to an adaptation decision, the gathering process, followed by a broadcast to the cores could create a significant latency upon the enforcement of the new policy. Eventually, when the new policy is enforced, the state of the system might be different compared to the one when the gathering process was initialized.

In order to enforce a policy in a distributed manner, controller cores should be able to detect and apply a regulation policy for job admission on their own, seamlessly to the rest of the cores. In this paper, we implement a simple yet effective regulation policy, that decelerates the admission of the jobs in cases that there are not enough available resources to serve the incoming workload. The idea behind admission deceleration is that in heavy load scenarios by delaying the internal mechanisms of controller cores, thus slowing down the resource allocation process, allowing already admitted applications to proceed with their execution uninterrupted. Consequently they free their resources faster and thus the search of available idle resources is relaxed.

The job admission control in the proposed framework is based on frequency scaling of the controller cores enabling their deceleration. Indeed, the requests served by controller cores, though numerous, are not computationally intensive, thus rendering the controller cores adequate candidates to enforce frequency scaling. Thus, the controller cores are assigned with a different operating frequency according to the size of the input queue, in respect to the worker cores. Taking into consideration the Intel SCC's VRC architecture, i.e. pre-configured voltage islands, we further optimize on the efficiency of the applied job admission solution by exploiting the hierarchical structure of the DRTRM. Specifically, we enforce a grouping of the controller cores to be all allocated to a specific voltage island. By this way, the frequency scaling

²The case of one controller core is omitted since it contradicts to the design principles of a distributed resource management

Fig. 5: Examined cluster topologies [#Controller cores, Cluster conf.]. Regions of the same color belong to the same cluster.

Fig. 6: Time gap in RTRM signal exchange

can be also combined with voltage scaling to further reduce the power consumption of the DRTRM infrastructure.

Fig. 6 shows in more detail how slowing down the operations of a controller core eventually will result in slowing down the evolution of the tasks in both initial and manager cores. It can be considered as zooming in the communication between a controller and an initial/manager core. The intention of the latter is to execute its internal Cores search task which requires information from its controller core. The left side of Fig. 6 shows the how the communication evolves in time in a typical frequency configuration. Its right side, depicts the behaviour of both cores after the frequency of the controller core has been scaled down. Execution of controller core tasks is prolonged in duration which also affects the evolution of tasks in the other core. The summation of small delays in each sub-task of the controller core, results in a time gap in the completion of tasks of the other core and this time gap, if generalized to all cores, results in stalling their operation.

TABLE I: Normalized deviation for variable input workloads

App arrival scenario	Slow	Lambda 16	Lambda 32	Lambda 48	Lambda 64	Fast
Total apps execution latency (ms)	9914.3	9720	9620.2	9818.1	10053.8	28414.5
Initial core execution latency (ms)	1039.9	1050.8	1044.1	1048.6	1050.4	2542.3

V. EXPERIMENTAL EVALUATION

In this section, we provide an extensive experimental evaluation of the proposed job-arrival aware DRTRM framework, considering the impact of differing arrival rates an distributions, the various design alternatives and the application admission control. All the experiments were conducted on the Intel SCC platform.

1) *Sensitivity to workload arrival rates:* In Table I, we quantify the impact of different arrival scenarios by summarizing the required execution time for the incoming workload. As shown, variations in the lambda parameter of Poisson distribution do not result in significant variation of execution time. The proposed DRTRM exhibits a stable behavior along such arrival patterns. The system remains in a relatively similar condition when the incoming application interval rises, the disturbance caused to the resource allocation is similar in all cases of Poisson distribution interval rate. As intended, "slow" scenario does not stress the performance of the system and conversely a steep rise of 187% in the total execution latency for the incoming applications is observed in the "fast" scenario.

We further analyze the stressed arrival scenario, denoted as "fast". In Table I, we report the value of Initial core execution latency, which is the execution time of the initial cores during the process of discovering the manager core of a new application. From another point of view, this time is an quantitative indication of the effort spent throughout the resource management process in order to introduce all applications into the system. This latency in the fast scenario is significantly increased around 130% compared to the slower

Fig. 7: DRTRM behaviour for all cluster configurations

scenarios. This leads to the conclusion that despite the shortage of available computational resources, initial cores kept on initiating a process to discover a core for their application, even though it was highly probable that such core was not available and would not be available until one of the running applications executed all of its workload and freed its resources. In the rest of this section, we perform analysis considering the "fast" scenario, that stresses the proposed DRTRM framework.

2) *Cluster topology impact on system performance:* In order to quantify the impact of differing topologies, we experimentally evaluated DRTRM's performance for all the cluster topologies presented in Fig. 5. We measured total execution time of applications and execution time spend on initial cores. Fig. 7 shows that increasing the number of controller cores results in larger latencies. This increase, on the one hand reduces the number of cores available to execute application workload and on the other hand it fragments the regional framework-related administrative information, which in turn increases the number of exchanged messages required in order to successfully gather this information. Given that the [2,1] topology configuration presents the best behavior, it is shown that for up to 48 cores the simpler the structure of the DRTRM the higher its efficiency.

3) *Performance-power gains derived from admission control:* This set of experiments focuses on evaluating the efficiency of the proposed admission policy to diminish the congestion created by the "fast" arrival scenario. All controllers are mapped to Voltage Island 0 of the SCC. Fig. 8 presents performance-energy metrics considering DRTRM configurations with 2 and 4 controllers, with the frequency of them dropped from 800 MHz to 533 MHz which also translates to a voltage drop from about 1.1 V to 0.8 V in the voltage of the island.

The results are expressed in normalized gain in respect to the case that no voltage-frequency scaling has been applied. As shown, in all cases performance and energy improvements are reported. The results show a lack of symmetry amongst the improvement in performance compared to energy. For example, in the configuration [2,1] (Fig. 5a) there is a 20% improvement in performance accompanied by a 12% reduction in the amount of energy whereas in configuration [4,2] (Fig. 5f) the respective numbers are 3% and 18%. This is explained on the premise that the measured performance does not indicate

Fig. 8: Performance-energy gains from job admission control in DRTRM.

the degree of concurrent execution of different applications on the system. This degree severely affects the time each scenario required to be completed and thus its requirements in energy. Therefore, in configuration [2,1], applications acquired more working cores, their summed execution time was small but they were executed in a more "serialized" way thus consuming more energy compared to configuration [4,2] where applications are executed in a more concurrent manner ergo possessing less cores in average and having increased total time of execution.

Fig. 9 reports experiments conducted considering all the available operating frequencies, where an interesting performance-energy trade-off is exposed. Decreasing the operating frequency of controller cores results in a decrease in the sum of application execution latency. However, this constant decrease of latency does not imply better resource allocation. Application initialization on the system is highly related to controller cores' operation and thus, when their operating frequency is reduced, this initialization is serialized. As a result when applications are instantiated on the platform, few other applications occupy resources and consequently the new application acquires the maximum cores it can handle thus minimizing its execution time.

On the contrary, this phenomenon results in increased time

Fig. 9: RTRM behaviour for all available freq divs in V.I. 0

Fig. 10: Measured values for all configurations (cluster + frequency divider)

of execution for the entire scenario since few application are executed in parallel. This is expressed through the increased energy that the scenario consumes in order to be executed and explains why as the operating frequency of the controller cores decreases, a great increase in the consumed energy is observed. Thus, for applications that require high throughput an admission control policy of lowering the controllers' frequency should be adopted, while in cases that the optimization goal is the overall energy consumption of the many-core platform controllers' operating frequencies should be set to higher values. This trend is also highlighted with the red and blue line on the figure. The red one blue one represents the curve of execution latency while the red one is the respective curve for the total energy consumed by each configuration.

4) *Exploratory analysis of the DRTRM parameters:* In order to evaluate the combined effects of the proposed DRTRM's parameters, we performed a large exploration campaign over the differing cluster topologies and operating frequencies of the controller cores. Fig. 10 summarizes the exploratory results, i.e. (a) execution latency of the initial cores and worker cores, (b) distribution of instant power consumption (c) respective total energy, (d) total number of exchanged messages, (f) size of the exchanged messages and (e) summed Manhattan distance between sender and receiver for all these messages. In all diagrams, the X-axis refers to the examined cluster configuration (Fig. 5) and the frequency of controller cores. Frequency of 800MHz implies that no admission control is performed and all the controller and worker cores run at the same frequency, while frequency 533MHz implies that admission control is active by lowering the frequency of the

controller cores while leaving workers cores to run at high frequency of 800MHz.

Regarding to system performance, Fig. 10a validates the fact that increasing the number of controller cores in SCC platform results in increased latency for both the application and initial cores and the execution. In addition, it is shown that the reduced operating frequency of the controller cores results in all cases to lower latency, showing that the proposed admission control policy enables performance optimization of 6%, in average. It is important to note that this performance gain comes together with energy consumption gains up to 12% (Fig. 10c). As shown in Fig. 10b, the proposed admission control scheme leads to better power distribution both in terms of i) robustness (the 25- and 75-percentiles in cases of 533MHz are consistently closer than in the case of 800MHz) and ii) peak power for which gains of 6% are reported. Power distribution for cluster topologies with increased number of controller cores, e.g. configurations [6, 2] and [6, 3] in Fig. 5, exhibits a more robust behaviour in their power trace given that the incoming workload is distributed in a fine manner across system resources.

As shown in Fig 10d, this increased performance is highly correlated to the number of exchanged messages. Given a DRTRM cluster topology, the proposed admission control policy results in a reduced number of total exchanged messages, which in turn validates the intended goal of control policy to regulate the evolution of operations of all involved agents of DRTRM on the platform. Regarding to the total size of exchanged messages (Fig. 10e), the trend is the same, as in their total number, but the actual values are not proportional since

the size of each message varies according to the information it carries. For example, an offer for cores from one manager to another is a few bytes long since it involves the ids of the offered cores, while the corresponding reply message is only one byte long to indicate the acceptance/rejection of the offer. Interestingly, the same trend does not apply to the summed Manhattan distance of exchanged messages (Fig. 10f) where cluster configurations of 4 controller cores exhibit elevated Manhattan distance values. This is can be attributed to the topology of these configurations which is highly fragmented and increases the probability of sender and receiver core to be located far apart on the platform.

5) *Robustness against workload scalability*: Finally, we evaluated the robustness of the proposed DRTRM configuration against scaled workloads to prove a steady behaviour³. Incoming application interval rates were set to "fast" and in each experiment their number and size of workload to be executed varied. The workload i.e. value W presented in section III-A2 was randomly generated using a random number generation function whose output derives from a Poisson distribution with mean value equal to W_{λ} . Four different scenarios were tested for four different values of this W_{λ} coefficient. The greater the lambda coefficient is, the more heavy workload input is produced. This variation in workload was combined with an ascending number of incoming applications ranging from 16 to 128.

Fig. 11 presents the total execution time of each examined input workload combination. Inspection of the results shows an escalation in execution latency as application workload increased. This escalation can be observed both for ascending number of incoming applications and different value of lambda coefficient of Poisson distribution. As stated, increase in both these values contribute in creating more workload intensive input applications and vice versa. Additionally, in all cases deviation from the respective mean value is very small. This qualifies DRTRM as quite robust especially taking into account the noise (expressed through variations in measured latency) injected in the results due to the software stack on top which DRTRM scheme is executed (Linux OS of each core of SCC platform). The highest measured deviation from the respective mean value was measured in the most workload intensive configuration of all, which resulted from Poisson distribution with W_{λ} equal to 64 and comprised of 128 incoming applications.

VI. CONCLUSION

In this paper we presented a job-arrival aware distributed run-time resource management framework focused on the execution of applications with malleable features on Intel SCC platform. We analyzed in detail the structure and internal mechanisms of the developed distributed management framework and we proposed an operating frequency scaling strategy that regulates job admission without degenerating the distributed nature of the resource management. We showed that the performance and power efficiency of the distributed resource management mechanisms is highly dependent on both the applications' arrival patterns and management policies/mechanisms. Extensive experimental analysis is provided to quantitatively analyze the features of the proposed framework, showing that both performance, power and energy gains are delivered.

³The DRTRM configuration [2,1] at 533 MHz is considered

Fig. 11: Mean value and standard deviation of examined application workloads

REFERENCES

- [1] Jason Howard, Saurabh Dighe, Yatin Hoskote, Sriram Vangal, David Finan, Gregory Ruhl, David Jenkins, Howard Wilson, Nitin Borkar, Gerhard Schrom, et al. A 48-core ia-32 message-passing processor with dvfs in 45nm cmos. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 108–109. IEEE, 2010.
- [2] George Chrysos. Intel® xeon phi coprocessor-the architecture. *Intel Whitepaper*, 2014.
- [3] Shane Bell, Bruce Edwards, John Amann, Rich Conlin, Kevin Joyce, Vince Leung, John MacKay, Mike Reif, Liewei Bao, John Brown, et al. Tile64-processor: A 64-core soc with mesh interconnect. In *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pages 88–598. IEEE, 2008.
- [4] Adrian Schüpbach, Simon Peter, Andrew Baumann, Timothy Roscoe, Paul Barham, Tim Harris, and Rebecca Isaacs. Embracing diversity in the barrelfish manycore operating system. In *Proceedings of the Workshop on Managed Many-Core Systems*, page 27, 2008.
- [5] Juan A Colmenares, Sarah Bird, Henry Cook, Paul Pearce, David Zhu, John Shalf, Steven Hofmeyr, Krste Asanovic, and John Kubiatowicz. Resource management in the tessellation manycore os. *HotPar'10, Berkeley, CA*, 2010.
- [6] David Wentzlaff, Charles Gruenwald III, Nathan Beckmann, Kevin Modzelewski, Adam Belay, Lamia Youseff, Jason Miller, and Anant Agarwal. A unified operating system for clouds and manycore: fos. *Proceedings of the 1*, 2009.
- [7] Thomas Ebi, M Faruque, and Jörg Henkel. Tape: Thermal-aware agent-based power econom multi/many-core architectures. In *Computer-Aided Design-Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, pages 302–309. IEEE, 2009.
- [8] Yang Ge, Parth Malani, and Qinru Qiu. Distributed task migration for thermal management in many-core systems. In *Proceedings of the 47th Design Automation Conference*, pages 579–584. ACM, 2010.
- [9] Sebastian Kobbe, Lars Bauer, Daniel Lohmann, Wolfgang Schröder-Preikschat, and Jörg Henkel. Distrm: distributed resource management for on-chip many-core systems. In *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software code-sign and system synthesis*, pages 119–128. ACM, 2011.
- [10] Iraklis Anagnostopoulos, Vasileios Tsoutsouras, Alexandros Bartzas, and Dimitrios Soudris. Distributed run-time resource management for malleable applications on many-core platforms. In *Proceedings of the 50th Annual Design Automation Conference*, page 168. ACM, 2013.
- [11] Bharathwaj Raghunathan and Siddharth Garg. Job arrival rate aware scheduling for asymmetric multi-core servers in the dark silicon era. In *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis, CODES '14*, pages 14:1–14:9, 2014.
- [12] Andrea Bartolini, M Sadri, J Furst, Ayse Kivilcim Coskun, and Luca Benini. Quantifying the impact of frequency scaling on the energy efficiency of the single-chip cloud computer. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*, pages 181–186. IEEE, 2012.