Application Interference Analysis: Towards Energy-efficient Workload Management on Heterogeneous Micro-Server Architectures

Markus Hähnel, Frehiwot Melak Arega, Waltenegus Dargie
Chair of Computer Networks,
Technische Universität Dresden, Germany
Email:{markus.haehnel1, frehiwot_melak.arega,
waltenegus.dargie}@tu-dresden.de

Robert Khasanov, Jeronimo Castrillon
Chair for Compiler Construction,
Technische Universität Dresden, Germany
Email:{robert.khasanov,
jeronimo.castrillon}@tu-dresden.de

***Abstract*—The ever increasing demand for Internet traffic, storage and processing requires an ever increasing amount of hardware resources. In addition to this, infrastructure providers over-provision system architectures to serve users at peak times without performance delays. Over-provisioning leads to underutilization and thus to unnecessary power consumption. Therefore, there is a need for workload management strategies to map and schedule different services simultaneously in an energy-efficient manner without compromising performance, specially for heterogeneous micro-server architectures. This requires statistical models of how services *interfere* with each other, thereby affecting both performance and energy consumption. Indeed, the performance-energy behavior when mixing workloads is not well understood. This paper presents an interference analysis for heterogeneous workloads (i.e., CPU- and memory-intensive) on a big.LITTLE MPSoC architecture. We employ state-of-the-art tools to generate multiple single-application mappings and characterize the interference among two different services. We observed a performance degradation factor between 1.1 and 2.5. For some configurations, executing on different clusters resulted in reduced energy consumption with no performance penalty. This kind of detailed analysis give us first insights towards more general models for future workload management systems.**

***Index Terms*—Energy-efficient computing, MPSoCs, heterogeneous, mapping, scheduling, interference**

I. Introduction

A wide range of power management strategies have been proposed at various levels of abstraction to make Internet servers energy-efficient. These include dynamic voltage and frequency scaling at core level [1], efficient resource allocation and dynamic task migration by a scheduler [2]; server virtualization and workload consolidation through the live migration of virtual machines [3], and dynamic load-balancing at the application level [4]. Complementary to the runtime adaptation strategies, effort is also being made both by the academia and the industry to develop energy-efficient and energy-proportional processor architectures including (1) the efficient integration of multicore and heterogeneous processors, (2) fast and efficient simultaneous multi-threading, (3) non-uniform cache architecture, and (4) advanced branch prediction strategies, among others.

A further improvement in the energy-awareness of Internet servers can be achieved through a closer examination of the resource requirements and execution characteristics of individual services and their mutual interference as they share resources at runtime. This is because (1) not all resources of a server (CPU, memory, IO bandwidth, etc.) may be utilized by the running services with a comparable proportion, and (2) contention for a particular resource (e.g. last-level cache [LLC]) may result in execution latency for all the contending services as well as inefficient energy consumption. Knowledge of the execution characteristics can be useful to schedule complementary services on the same server and to avoid the co-location of contending services.

Nowadays, heterogeneous architectures have become common on the server side [5]. Energy-efficient application mapping onto such architectures have been broadly studied in the embedded domain [6], but the effect on micro-servers is not studied enough. Therefore, it is necessary to develop strategies which take service characteristics into account. In this paper we *borrow* state-of-the-art mapping algorithms and use them to quantitively investigate the resulting interference between contending threads belonging to different services with dynamic workload. We do this on a heterogeneous multiprocessor architecture using a CPU-intensive application and a memory benchmark. We expect our analysis to enable sophisticated energy-aware thread-to-core mapping algorithms, that take into account the resource utilization characteristics of the building blocks of executing services.

The rest of this paper is organized as follows: In Section II, we summarize related work; in Section III, we introduce our concept; in Section IV, we describe our experimental setup; in Section V, we present and discuss results and share the insight we obtained. Finally, in Section VI, we give concluding remarks and outline future work.

II. Related work

A large amount of research has been conducted by adapting embedded domain's techniques and principles for server-side computing and are therefore related to our work. This includes

Dreamcloud [7] that develops workload management strategies for complex embedded systems for cloud-based use cases, and ANTAREX [8] that uses domain specific languages to express self-adaptivity and enable runtime auto-tuning.

In this paper we employ dataflow applications and built on top of years of research on mapping these applications to embedded heterogeneous multi-core architectures, e.g., for performance [9], for energy efficient computing [10] and for multiple objectives [11] (see also survey in [6]). Some of these works use formal models of computation that make it possible to provide design-time guarantees for multi-application scenarios, e.g., synchronous dataflow in [12] or Kahn Process Networks (KPN) in [13], [14].

There are several approaches to model the interference of co-located applications. The authors in [15] propose a model for Virtual Machines (VMs) with a CPU-dependent and a fixed CPU-independent component. The latter results mainly from contention in shared caches and the memory bandwidth [16]. The authors in [17] observe that the runtime consists of calculation time and the access times of the cache levels and memory. When services are consolidated, both the number of LLC-misses and the memory access time increase. Based on performance counters, such as LLC misses and references, they defined interference sensitive and interference intensive applications. A more detailed framework for estimating the *worst-case execution time* (WCET) and the *worst-case response time* (WCRT) caused by contention in the caches and the memory bus was proposed by the authors in [18]. There are several approaches to minimize such contention. The authors in [19] propose cache partitions as new hardware feature to isolate different services and a software control unit between cores to manage the intra-task communication. A closer intra-task cache analysis can be found in [20] and [21] where a messaging graph is built for each service to determine the best task mapping. Furthermore, the WCET can be improved drastically by inserting appropriated locks [22]. Other shared resources such as disk I/O have been also studied in [23].

In contrast to the aforementioned works, we assume a more dynamic server scenario by randomly varying workload instead of using only static benchmarks. Thanks to the consideration of distribution functions for the workload and its interference, we seek a more general statistical characterization. Further, we integrate the compilation process into the realization of a new thread-to-core mapping, which optimizes the executable service with respect to the currently assigned resources. Regarding interference analysis, we investigate different levels of contention including CPU time of a single core, cache and globally shared resources. Moreover, we consider the energy saving not only by switching off unutilized machines, but rather by workload consolidation.

III. Concept

Assuming that a data center provides a limited number of services but serves a sizeable number of users, its workload can be classified into a delay-sensitive query-based workload and delay-tolerant batch workload. The former is typically generated by users whereas the latter consists of a collection of background jobs. The workload type depends on the services the data center provides. On the other hand, the magnitude of a query-based workload at any given time cannot be known a priori except in a probabilistic sense whereas the magnitude of a batch workload can be approximated before its execution.

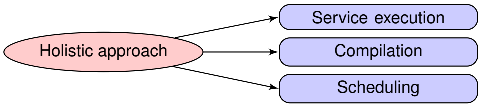

Fig. 1: Energy-efficiency is achieved by taking into account high-level service execution, compilation and low-level thread/task scheduling.

The execution of batch workloads can be optimized to achieve execution (performance) and energy efficiency by combining knowledge of workload statistics, service execution characteristics, and resource consumption characteristics. This task requires a holistic approach involving service execution, compilation and low-level thread/task scheduling (c.f. Fig. 1). The service execution layer estimates the magnitude of the batch workload and sets constraints on the quality of service execution (such as on job completion time). The compiler examines the anatomy of executable services, infers their resource consumption profile, and maps executable subtasks to physical cores based on an analysis of available parallelism. The scheduler monitors the resource consumption characteristics of executing services, identifies and quantifies interference effects, and reschedules or re-maps subtasks to avoid both underutilization and overloading conditions.

The service execution layer also aims to achieve a higher-level efficiency by matching the supply and demand of resources in the data center. This can be done by dynamically consolidating the executing and incoming workloads on a few number of servers and by switching off the rest. However, dynamic workload consolidation may create and exacerbate resource contention at a local level and may degrade the service execution quality. At the same time, local contention can be deliberately tolerated if its effect is significantly smaller than the overall gain which can be achieved through service consolidation. Hence, a trade-off between using as less as possible resources and the contention has to be conducted in advance. We need to quantify the contention of services running on a shared resource in order to foresee and deal with its effects. This can be done by quantifying either performance degradation or the extra power consumption incurred as a result of contention. Furthermore, this time can be correlated with the resource consumption characteristic of the executing workloads in order to determine the cause of contention and to minimize its effect.

As a first step towards implementing the conceptual approach proposed in this section, we experimentally investigate how the effect of contention can be reflected on the statistical footprint of job completion time and the power consumption on a heterogeneous architecture.

IV. Experimental setup

In order to experimentally investigate the impact of (1) processor architecture, (2) the resource consumption char-

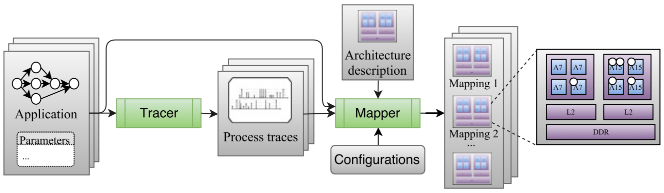

Fig. 2: Dataflow application mapping flow in MAPS.

acteristic and (3) the mapping of service elements onto physical resources on the performance of services and the energy-efficiency of service execution, we set up an experiment environment consisting of (a) the Odroid-XU4 heterogeneous processor platform[1], (b) a mapper, and (c) a CPU-bound (JPEG transcoder) service and a memory-bound benchmark.

Processor architecture. The heterogeneous platform consists of an octa-core "big.LITTLE" architecture, with a cluster of four "LITTLE" (ARM Cortex-A7) cores and a cluster of four "big" (ARM Cortex-A15) cores. Each LITTLE core has an exclusive 32 kB L1 cache and shares a 512 kB L2 cache with the other LITTLE cores. Similarly, each big core has exclusive 32 kB L1 cache and shares 2 MB L2 cache with the other big cores. The most significant contention between LITTLE and big cores occurs on the main memory. The higher performance of the big cores is due to a higher peak operation frequency as well as a more complex micro-architecture (3-way out-of-order execution pipeline as opposed to the single in-order execution pipeline of the Cortex-A7).

Mapping. The *mapper* takes a software implementation of a service and decides on the assignment to physical resources and the ordering of execution of application elements (i.e., data, communication and computation) to achieve a given optimization goal such as performance or energy-efficiency or both. There are many possible parallel programming models, but for our experiment we focus on *dataflow programming models*, that is, a graph-based model, where vertices represent processes and edges represent FIFO channels through which processes communicate with each other by exchanging data. Such models expose parallelism by allowing processes to be executed simultaneously, independently on each other. We chose "C for Process Networks" (CPN) [24], a KPN-based programming language [25]. Along with an application, parameters are supplied to the mapper; these parameters may refer to the workload (such as size of the workload). For our experiment, we employ MAPS [9], [26], a design-time mapper and adapt it to produce multiple mappings depending on the application parameters. Fig. 2 shows the MAPS flow for generating multiple mapping configurations for dataflow applications onto heterogeneous multicore architectures.

As its input, the mapper takes a CPN application, a description of the target architecture along with the corresponding performance model, and a configuration that includes user requirements (mapping constraints and real time constraints). Process traces are generated from the source code by the *tracer* and contain information about the control paths, followed by every process and the channel access events (read and write). We configured MAPS to use the performance-oriented heuristic *Group-Based Mapping* (GBM) which iteratively assigns resources to the application by analysing its dynamic critical path [27]. To produce mappings with different trade-offs between execution time and resource utilization, MAPS was configured to generate mapping variants that utilize different subsets of architectural components. To reduce the overall exploration time we reduced the number of subsets by removing symmetric resource configurations [28]. Resultant mapping variants are then used to generate executables.

Applications. In the literature, benchmarks are often analyzed with a fixed workload configuration. We, in turn, conduct our analysis two parallel application that allow a variable load: a dataflow implementation of the JPEG encoder/decoder (as described in [9]) and the RandomAccess memory benchmark [29]. The latter tests the speed at which a machine can update the elements of a table spread across global system memory (measured in Giga-updates-per-second). The benchmark takes as its input three parameters, namely, the length of the global table, the number of update sets, and the number of updates in each set. We fixed the first and the last parameter to 2^{26} ($\approx$250 MB) and 1024, respectively, and varied the update sets from 1 to 10 000.

Measurement setup. In order to investigate the impact and level of interference when two services share computing resources, we generate probabilistic workloads. To this end, we divide time into slots of equal durations (10 s). At the beginning of each slot, both the CPU- and memory-bound services receive jobs. The job sizes (the image size for JPEG and the count of update sets for RandomAccess) are sampled from an exponential probability distribution. Therefore, each application is intermittently affected by the other one which runs at the beginning but might finish earlier. Finally, we measure the power consumption of the Odroid-XU4 board using YOKOGAWA WT210 power analyzer with a sample rate of approximately 10 Hz.

We study three different scenarios: (1) all variants executed alone to serve as reference, (2) two JPEG instances are executed, and (3) each JPEG mapping is co-located with the memory benchmark. (2) and (3) are divided into running on (a) the same cores (e.g., configuration 8), (b) different cores but same cluster (e.g., configuration 12), and (c) different clusters (e. g., configuration 15). The resulting 16 experiments with the particular mappings generated by MAPS are illustrated in Fig. 3. Every measurement run for 1 h to achieve a statistical significant amount of interval samples of approx. 360.

V. Evaluation

In this section we examine the interference of the configurations in Fig. 3. The type of interference effects we wish to investigate can be broadly categorized as intra-core (core level), intra-cluster (cluster level), and inter-cluster (board level) contention. The intra-core contention manifests itself mainly in the form of context switches (shared CPU cycles) and contention for the L1 cache. The intra-cluster contention

[1] http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825

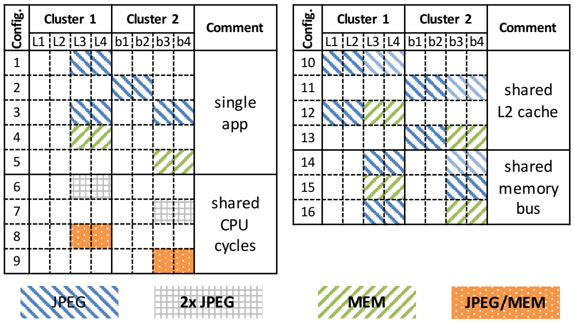

Fig. 3: All configurations of measurements.

manifests itself mainly in the form of contention for the L2 cache. Finally, the inter-cluster contention manifests itself mainly for contention for the shared memory bus. We focus on the impact of interference on job completion time and overall power consumption.

A. Job completion time

Fig. 4 shows the completion time for the JPEG transcoder when executing with another JPEG instance. The first three curves on the top Fig. 4a serve as a reference (config. 1–3). The curves indicate that the job completion time mirrors the workload statistics, i.e., the CDFs can be approximated as exponential distributions which reflects the exponential distribution of the workload. Moreover, the difference in the performance of the three configurations (2 LITTLE, 2 big, 2 LITTLE and 2 big cores) was comparably small as long as the transcoder was executed in isolation. This clearly indicates that using 2 LITTLE and 2 big cores does not give a significant performance gain compared to using only 2 big cores. On the other hand, as the transcoder shares a single core with another service (Fig. 4a, config. 6–7), the job completion time significantly increased. Interestingly, this feature does not discriminate between LITTLE and big cores. When the two services executed on separate cores, whether the arrangement is intra- or inter-cluster, the job completion time reduces considerably compared to the intra-core job completion times.

Fig. 5 shows the CDFs of the completion time when JPEG executed in parallel with the memory benchmark. The intra-cluster (Fig. 5b) interference effect on the LITTLE cores remained the same as in JPEG-JPEG, but changed slightly on the big cores. Similarly, the difference between using big or LITTLE cores becomes larger for the intra-cluster configurations (12–13) when compared with JPEG-JPEG, suggesting an elevated contention on the L2 cache. On the other hand, the distribution of the inter-cluster configurations (15–16) (Fig. 5c) remained more or less the same as for the JPEG-JPEG configuration.

Fig. 6 shows the CDFs of the completion time for the memory benchmark when it executed in parallel with JPEG. The first two lines in the top of Fig. 6a correspond to configurations 4–5. In the other plots, the memory benchmark shared the same cores with the transcoder (8–9). As expected, regardless of the types of core shared, the job completion time deteriorated. But the plot suggests that it is preferable sharing two big cores rather than executing the memory benchmark in isolation on two LITTLE cores. Otherwise, the memory benchmark seems to be unaffected by the execution of the transcoder in the intra-core and inter-core configuration, for the distributions of its completion time remain more or less the same.

B. Power consumption

Fig. 7 displays the CDFs of the overall power when two instances of JPEG execute in parallel. The Fig. 7a serves as a reference, for it describes the power consumption of config. 1–3. As an indication of the energy-proportionality of the processor, all curves are similar and there is no conspicuous penalty for availing more cores for the exclusive use of the service. But as we have already seen above, over-provisioning did not produce any appreciable gain in job completion time. The Fig. 7b displays the cost of intra-core contention in both cores, but the effect is considerable in the big cores. In contrast, the effect of intra-core and inter-core contention on the power consumption is minimum. This can be seen, for example, in the Fig. 7c, which displays the power consumption characteristic of the intra-cluster configuration (10–11).

Fig. 8 displays the CDFs of the overall power consumption when JPEG was executing in parallel with the memory benchmark. As in the previous case, here as well, the intra-core contention (Fig. 8b, config. 8–9) on the big cores produced the worst power consumption. For all the other cases, the power consumption does not seem to favour any particular configuration for the execution of the memory benchmark suggesting that the decision to favour a particular configuration should be made based on the performance aspect (job completion time, for our case).

C. Summary

Fig. 9 shows a comparison of the average energy overhead ($\frac{E_{\text{shared}}-(E_1+E_2)}{E_1+E_2}$, idle excluded) and the average performance degradation factor ($\frac{t_{\text{shared}}}{t_{\text{single}}}$) for the JPEG-JPEG as well as the JPEG and memory benchmark cases. In both scenarios we expect a performance degradation factor of 2 when running on the same cores. The overhead of context switching, cache misses and alike results in a degradation factor of around 2.5. When running on different cores the effective execution time is the same. Thus, the expected factor is 1, but is affected by sharing at the L2 cache. As observed, JPEG is sensitive to the type of the second application, with a factor of 1.1 in the JPEG-JPEG case and 1.4 in the JPEG and memory benchmark case. The latter is due to much more L2 cache misses for JPEG. The memory benchmark itself with an average performance degradation of 1.05 is not significantly affected by the JPEG application. Last but not least we could not measure any significant contention in performance when running on different clusters.

What energy is concerned, we observe more savings the less contention occurs. Due to job execution time overhead there is an energy overhead of 15 to 41 % when running on the same cores. When running on different cores on the same cluster

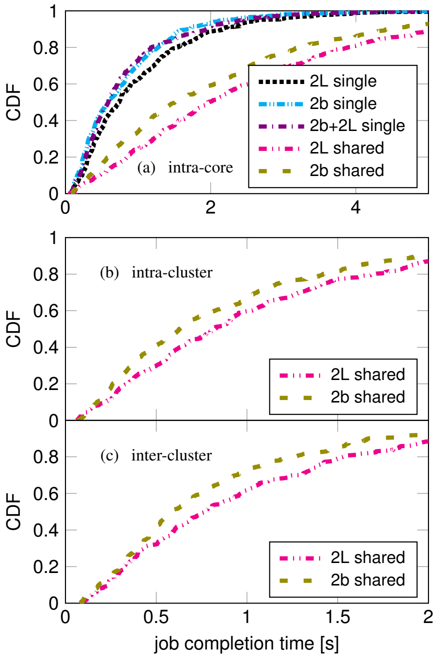

Fig. 4: CDFs of job completion time for JPEG (with JPEG).

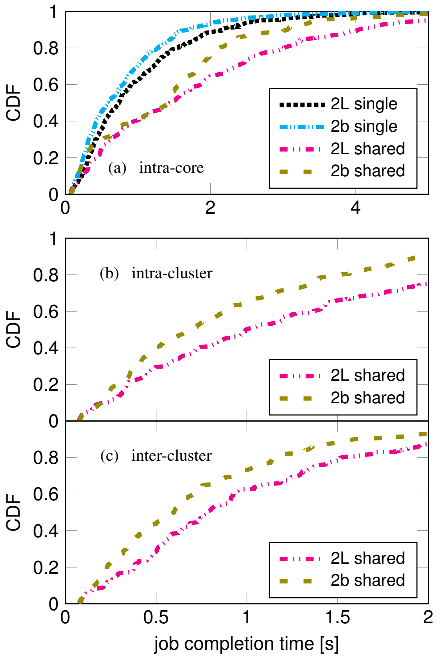

Fig. 5: CDFs of job completion time for JPEG (with the memory benchmark).

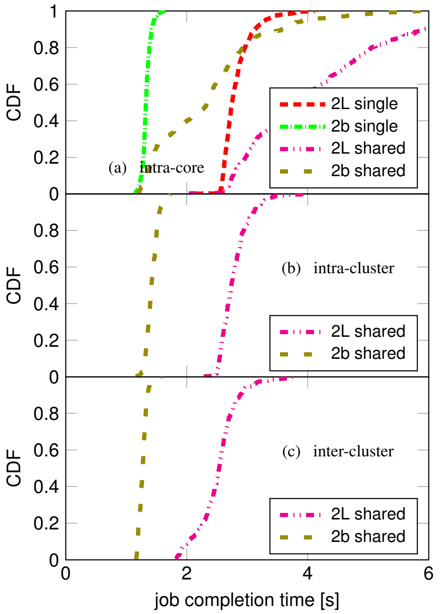

Fig. 6: CDFs of job completion time for the memory benchmark (with JPEG).

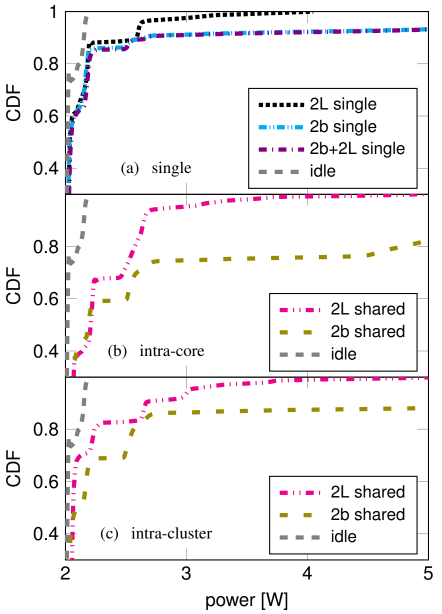

Fig. 7: CDFs of power consumption for JPEG with JPEG.

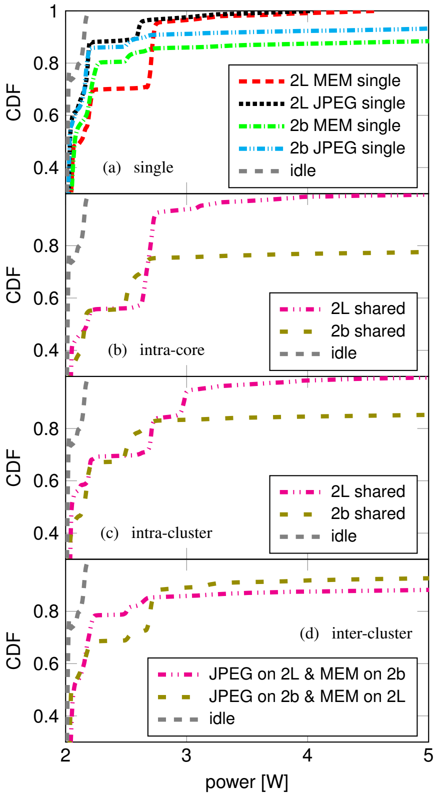

Fig. 8: CDFs of power consumption for JPEG with the memory benchmark.

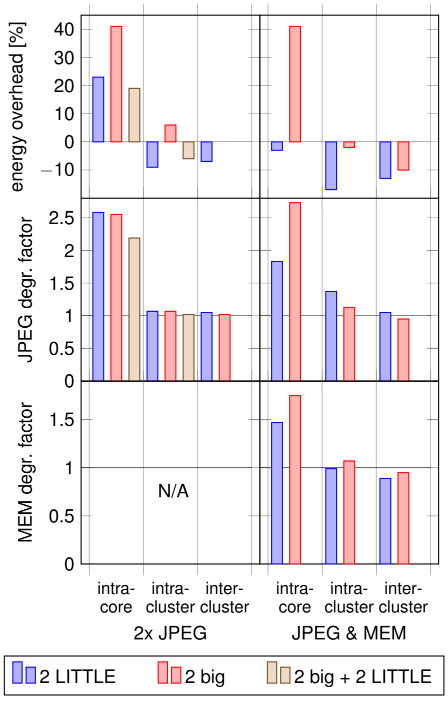

Fig. 9: Average energy overhead and average performance degradation factor.

there is almost no energy overhead or even a small energy saving of −2 to −17 % with the memory benchmark. This can be explained by shared resources such as the L2 cache which have to be powered only once even when more cores are active. Surprisingly there is also an energy reduction when running on different clusters although more subsystems such as second caches have to be powered on. Probably since performance is not degraded thanks to almost no contention, the hardware can return to a deep sleep state very soon. Hence, energy can be saved by consolidating services on a subset of machines but on the machine itself it is justified to distribute the services to independent resources in order to avoid the energy overhead due to increased job completion time caused by contention.

Based on these results, we can derive some insights for the future workload management strategy. The services and their types might be used as an input and the workload management has to decide how to co-locate them on the available resources. Particularly, running a memory intensive service on LITTLE cores and a CPU intensive service on big cores leads to energy-efficient co-location. If there are too many services, the scheduler should prefer the co-location on LITTLE cores.

VI. Conclusion

In this paper, we have conducted an interferences analysis of CPU- and memory-intensive applications running on a heterogeneous micro-server architecture. We analyzed the job completion time and the power consumption, focusing on different levels of contention, namely, intra-core, intra-cluster, and inter-cluster. With only a single application, we could not observe any appreciable gain in performance in case of over-provisioning of the hardware. At the core level, we observed a significant performance degradation factor of up to 2.5 due to contention. At the cluster level, we observed a strong dependency of performance degradation on the types of applications which were co-located. When a CPU intensive application is co-located with a memory intensive application, the latter is the one the performance of which degrades worse. In the inter-cluster configuration, the performance degradation was not considerable. In this case, the overall power consumption was smaller than the sum of the power consumption of the individual executing jobs. We also observed that the effect of consolidation was strongly dependent on the type of co-located applications due to the contention for cache. Therefore, the resource manager need new mapping algorithms which take the instantaneous load and the internal thread communication via caches into account.

In future, our aim is to predict the job completion time and the power consumption in co-location scenarios based on the results of our experiment. This will enable us to consolidate services in accordance with their execution, compilation and scheduling characteristics.

Acknowledgment

This work was supported by the German Research Foundation (DFG) within the Collaborative Research Center HAEC and the cfaed Cluster of Excellence. We also thank Silexica for making their software tool suite available to us.

References

[1] V. Hanumaiah and S. Vrudhula, "Energy-efficient operation of multicore processors by dvfs, task migration, and active cooling," *IEEE Transactions on Computers*, 2014.

[2] A. Merkel, J. Stoess, and F. Bellosa, "Resource-conscious scheduling for energy efficiency on multicore processors," *EuroSys*, 2010.

[3] A. Strunk and W. Dargie, "Does live migration of virtual machines cost energy?" *AINA*, 2013.

[4] D. Kliazovich, P. Bouvry, and S. U. Khan, "Dens: Data center energy-efficient network-aware scheduling," *CLUSTER*, 2013.

[5] R. T. Kaushik, M. Bhandarkar, and K. Nahrstedt, "Evaluation and analysis of greenhdfs: A self-adaptive, energy-conserving variant of the hadoop distributed file system," *CloudCom*, 2010.

[6] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," *DAC*, 2013.

[7] *The dream cloud project*, http://www.dreamcloud-project.org.

[8] C. Silvano *et al.*, "The antarex approach to autotuning and adaptivity for energy efficient hpc systems," *Computing Frontiers*, 2016.

[9] J. Castrillon and R. Leupers, *Programming Heterogeneous MPSoCs: Tool Flows to Close the Software Productivity Gap*. Springer, 2014.

[10] C. A. M. Marcon, E. I. Moreno, N. L. V. Calazans, and F. G. Moraes, "Comparison of network-on-chip mapping algorithms targeting low energy consumption," *IET Computers Digital Techniques*, 2008.

[11] G. Ascia, V. Catania, and M. Palesi, "Multi-objective mapping for mesh-based noc architectures," *CODES+ISSS*, 2004.

[12] A. Kumar, S. Fernando, Y. Ha, B. Mesman, and H. Corporaal, "Multiprocessor systems synthesis for multiple use-cases of multiple applications on fpga," *TODEAS*, 2008.

[13] L. Schor *et al.*, "Scenario-based design flow for mapping streaming applications onto on-chip many-core systems," *CASES*, Tampere, Finland: ACM, 2012.

[14] J. Castrillon, R. Leupers, and G. Ascheid, "Maps: Mapping concurrent dataflow applications to heterogeneous MPSoCs," *Industrial Informatics*, 2013.

[15] S. P. Srinivasan and U. Bellur, "Watttime: novel system power model and completion time model for dvfs-enabled servers," *ICPADS*, 2015.

[16] A. Roytman, A. Kansal, S. Govindan, J. Liu, and S. Nath, "Pacman: performance aware virtual machine consolidation," *ICAC*, 2013.

[17] S. G. Kim, H. Eom, and H. Y. Yeom, "Virtual machine consolidation based on interference modeling," *The Journal of Supercomputing*, 2013.

[18] S. Chattopadhyay, A. Roychoudhury, and T. Mitra, "Modeling shared cache and bus in multi-cores for timing analysis," *SCOPES*, 2010.

[19] M. Slijepcevic, L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla, "Time-analysable non-partitioned shared caches for real-time multicore systems," *DAC*, 2014.

[20] H. Ding, Y. Liang, and T. Mitra, "Shared cache aware task mapping for wcrt minimization," *ASP-DAC*, 2013.

[21] Y. Li, V. Suhendra, Y. Liang, T. Mitra, and A. Roychoudhury, "Timing analysis of concurrent programs running on shared cache multi-cores," *RTSS*, 2009.

[22] W. Zheng and H. Wu, "Wcet-aware dynamic instruction cache locking," *LCTES*, 2014.

[23] S. Verboven, K. Vanmechelen, and J. Broeckhove, "Black box scheduling for resource intensive virtual machine workloads with interference models," *Future Generation Computer Systems*, 2013.

[24] W. Sheng *et al.*, "A compiler infrastructure for embedded heterogeneous mpsocs," *Parallel Computing*, 2014.

[25] G. Kahn, "The semantics of a simple language for parallel programming," *Information processing*, 1974.

[26] Silexica, *Slx tool suite*, http://silexica.com/products, [Online; accessed 27-April-2016].

[27] J. Castrillon, A. Tretter, R. Leupers, and G. Ascheid, "Communication-aware mapping of kpn applications onto heterogeneous mpsocs," *DAC*, 2012.

[28] A. Goens and J. Castrillon, "Analysis of process traces for mapping dynamic kpn applications to mpsocs," *IESS*, 2015.

[29] S. Plimpton, R. Brightwell, C. Vaughan, and K. Underwood, "A simple synchronous distributed-memory algorithm for the hpcc randomaccess benchmark," *CLUSTER*, 2006.