Co-scheduling on Upcoming Many-Core Architectures

Simon Pickartz
Institute for Automation of Complex Power Systems
RWTH Aachen University
Aachen, Germany
spickartz@eonerc.rwth-aachen.de

Jens Breitbart
Bosch Chassis Systems Control
Stuttgart, Germany
jens.breitbart@de.bosch.com

Stefan Lankes
Institute for Automation of Complex Power Systems
RWTH Aachen University
Aachen, Germany
slankes@eonerc.rwth-aachen.de

ABSTRACT

Co-scheduling is known to optimize the utilization of supercomputers. By choosing applications with distict resource demands, the application throughput can be increased avoiding an underutilization of the available nodes. This is especially true for traditional multi-core architecture where a subset of the available cores are already able to saturate the main memory bandwidth.

In this paper, we apply this concept to upcoming many-core architectures by taking the example of the Intel KNL. Therefore, we take a memory-bound and a compute-bound kernel from the NAS Parallel Benchmarks as example applications. Furthermore, we examine the effect of different memory assignment strategies that are enabled by the two-layered memory hierarchy of the KNL.

Keywords

HPC; Co-Scheduling; Many-core architectures; Energy efficiency; Memory Hierarchy

1. INTRODUCTION

Supercomputers will considerably change on their path to exascale systems. The growth in size will be accompanied by an increasing complexity of the compute nodes in terms ascending core counts and multi-level memory hierarchies. These changes put high demands on the application developers. They do not only have to ensure scalability on the inter-node level for up to thousands of nodes, but also need to hand-tune their code for the full exploitation of resources on the intra-node level. This trend is especially challenging for legacy codes which are rarely adapted to the peculiarities of every upcoming generation of supercomputers.

Co-scheduling is one approach for an optimization of the system's utilization when running non-highly tuned codes. As opposed to an exclusive allocation of nodes to applications, the goal is to find suchlike with diverse resource demands to share a set of cluster nodes. Thereby, the individual job experiences a slow down compared to its exclusive execution but the overall runtime may be reduced compared to their serialized execution.

Intel's recently introduced many-core architecture Knights Landing (KNL) presages potential characteristics of upcoming supercomputers. It is a single chip possessing up to 72 cores exposing four Hardware Thread Contexts (HTCs) each. In contrast to common Xeon CPU cores, they are based on Intel's Silvermont micro-architecture which was originally designed for mobile processors such as Intel's Atom. However, the large amount of cores per chip and their 512 bit-wide Single Instruction Multiple Data (SIMD) units reveals a strong potential for highly parallel and vectorized codes. Furthermore, the KNL provides an additional level in the memory hierarchy by the introduction of on-die high bandwidth memory. This is put close to the processor and provides a bandwidth superior to the normal DRAM at the expense of slightly increased latency at lower bandwidth utilizations.

In this paper we investigate the viability of co-scheduling on many-core architectures by taking the example of the Intel KNL. Therefore, we use two kernels of the NAS Parallel Benchmarks (NPBs) suite for the conduction of both performance and energy measurements. Thereby, we want to estimate if our considerations made in previous works [2] apply to upcoming many-core architectures as well. In contrast to these studies, the additional level in the memory hierarchy of the KNL enables further co-scheduling scenarios, e. g., two applications may be executed concurrently using the different memory levels.

This paper is structured as follows: the next section presents the Intel KNL while discussing the different cluster modes and memory configurations. Section 3 gives an introduction to the NPBs and provides information on the runtime characteristics of the two kernels EP and CG which have been used for the evaluation of our work. After presenting a comprehensive evaluation in terms of performance results and energy consumption in Section 4, we discuss related work in Section 5 before concluding the paper in Section 6.

2. INTEL KNIGHTS LANDING

The Intel KNL is the second generation many-core processor of Intel's Xeon Phi product line. In contrast to its predecessor, the KNL is a self-boot processor that features binary compatibility to the mainline Xeon Instruction Set Architecture (ISA) [9]. The processor is divided into 36 tiles connected by a 2D-mesh which implements a YX-routing. Each tile posses two cores based on Intel's Silvermont micro-

COSH-VisorHPC 2017 Jan 24, 2017, Stockholm, Sweden

ISBN 978-3-00-055564-0
DOI: 10.14459/2017md1344415

architecture which have been adapted especially for High Performance Computing (HPC), e. g., each core provides four HTCs. There are two memory controllers on either side of the chip providing three channels respectively for memory bandwidths of up to 90 GiB/s. Apart from the well-known DDR4-DRAM, the KNL additionally provides up to 16 GiB of Multi-Channel DRAM (MCDRAM) which is 3D-stacked on-package memory. This memory type targets at high bandwidths of around 400 GiB/s, however in contrast to DDR4 it shows slightly higher latencies. The MCDRAM is directly connected to the mesh by using 8 memory controllers.

For the evaluation we used the Intel Xeon Phi 7210 possessing 64 cores and a total of 256 hyperthreads. The cores are clocked at 1.3 GHz. Besides the 16 GiB of MCDRAM the system is equipped with 96 GiB of DDR4 memory. We disabled the turbo mode for the avoidance of fluctuations in the measurements.

2.1 Cluster Modes

The 2D interconnect is used for the implementation of cache coherency among the cores. It can be configured in three different so-called cluster modes on the BIOS level. The all-to-all mode is the most general and allows for unbalanced memory distributions across the two DDR4 memory controllers [5]. Therefore, it does not implement an affinity between the tiles, directories, and the memory by uniformly hashing all memory addresses across the distributed directories.

In contrast, the quadrant mode divides the chip into four virtual quadrants which realize an affinity between the distributed directories and the memory. In this mode, the memory addresses are hashed to the directories which belong to the same quadrant as the memory. However, this is transparent to the software which sees a one-socket system exposing 64 cores.

Finally, the Sub-NUMA Clustering (SNC) exposes the quadrants as individual NUMA domains to the software. Hence, NUMA-aware applications can profit from lower latencies by reducing memory accesses to remote quadrants. For all measurements presented in this paper we activated the SNC mode.

2.2 Memory Configurations

The 3D-stacked MCDRAM can be configured in three different ways. Just as with the cluster modes, the configuration can only be performed by changing the BIOS settings accordingly. In the so-called flat mode, the memory extends the physical address space of the system and is explicitly exposed as NUMA domain to the software. Thus, in the quadrant cluster mode one additional NUMA domain shows up while there are four in the SNC mode.

Alternatively, the MCDRAM can operate in the cache mode in which it acts as last-level cache for the DDR4 memory in transparency to the software. Finally, in the hybrid mode, a portion of the MCDRAM acts as cache while the remainder can be used in flat mode. The measurements presented in this paper where conducted by using the stacked memory in flat mode.

3. NAS PARALLEL BENCHMARKS

The NPBs [1] is a set of computing kernels intended for the performance evaluation of supercomputers. They represent common kernels of computational fluid dynamics applications and offer different problem classes. Therefore, the benchmarks are well-suited for the evaluation of a wide range of cluster sizes.

For the evaluation of our work we chose the two kernels EP and CG. Furthermore, we chose Class C as problem size depicting a reasonable size for a workstation test system like the one used for our work. The CG kernel computes the approximation to the smallest eigenvalue of a large sparse matrix [8]. This benchmark is characterized by irregular memory accesses and communication and therefore likely to be limited in performance by the available memory bandwidth. EP computes statistics from Gaussian pseudo random numbers. As the name suggests, there are neither dependencies between the individual work items nor does the benchmark dependent on the available main memory bandwidth.

As we aim at applying co-scheduling to many-core systems, the combination of these two kernels is a reasonable choice. They perfectly meet the requirement of distinct resource demands and should complement each other in a co-scheduling scenario.

4. EVALUATION

This section presents an evaluation of the two kernels EP and CG from the NPBs suite. We start with an analysis of their scalability and power consumption using DDR4 memory and MCDRAM respectively. In the second part we discuss the performance and energy results obtained from co-scheduling the two kernels in different configurations. Each meter point was captured by executing the respective applications for 15 min in a loop and averaging the results afterwards. For a reduction of the captured meter points we chose a step width of 4 threads.

4.1 Application Scalability

The scalability results of the CG and the EP kernel are presented for both memory types in Figure 1. The speedup is computed based on the best sequential execution. This was for both kernels a single thread running on DDR4 memory since this provides slightly better latencies than the MCDRAM. The EP kernel should not be sensitive to certain pinning strategies and we therefore performed a compact pinning on the core level, i. e., we started to fill up the physical cores one by another before using the HTCs. In contrast, we used a scatter pinning on the core level for the CG kernel due to its memory-bound characteristics.

At least for small thread counts, the speedup curves of the EP kernel do not differ when using either of the memory types (cf. Figure 1b) since the kernel is compute-bound. However, for rising thread counts we can observe a continuously growing overhead of up to around 10 % when using traditional DDR4 memory (—△—). This might stem from the higher contention on the memory controllers and the mesh which can be handled more efficiently by the 8 controllers for the high-bandwidth memory in contrast to the two DDR4 memory controllers.

The CG kernel stronger benefits from the high-bandwidth memory. Already using 8 threads, its execution on MCDRAM starts to outperform the results obtained on the DRAM. For thread counts greater equal to 40 threads, we observe a performance benefit of 10 % to 20 %. However, for both memory types the kernel does not make efficient use of additional HTCs. This can be explained by the fact

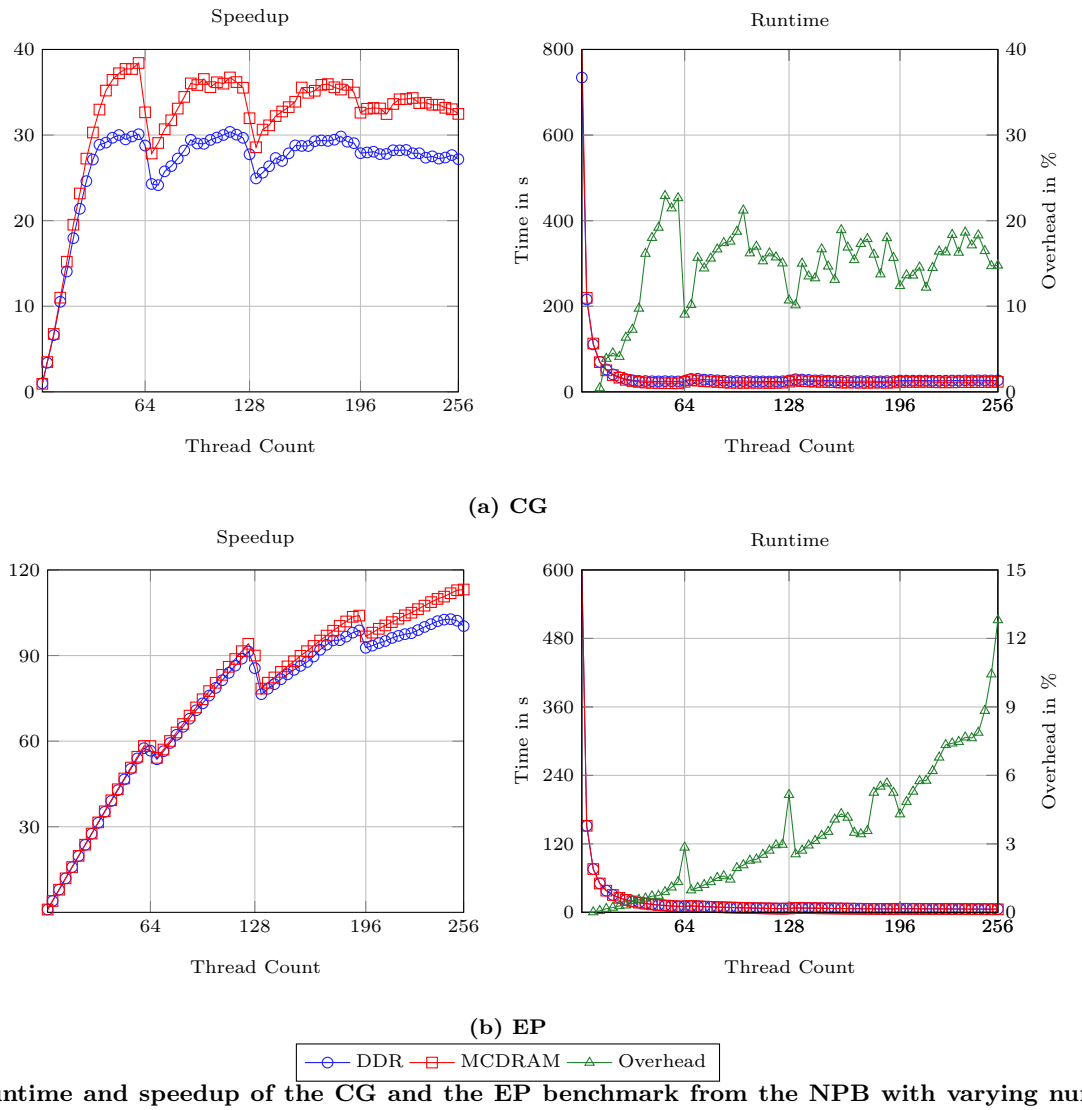


Figure 1: Runtime and speedup of the CG and the EP benchmark from the NPB with varying number of threads. The CG benchmark has been pinned in accordance with the scatter strategy while compact pinning on a core granularity was used for EP.

that the KNL employs out-of-order cores. In contrast, Ramachandran et al. could observe a constant performance increase for the CG kernel on its predecessor which was still based on an in-order architecture [6].

Figure 2 shows the power consumption of the two computing kernels which was captured by both using the Running Average Power Limit (RAPL) counter [3, 4] and a node-external Power Distribution Unit (PDU). The latter is a MEGWARE[1] Clustsafe unit which gauges the complete system power consumption including the power supply. Unfortunately, the RAPL counter available on the KNL only provide energy information of the memory and the whole CPU package but do not allow for a measurement of the energy consumed by the cores only.

The EP kernel exhibits a nearly constant growth of the power consumption except for the spikes at the HTC boundaries which correspond to the performance results discussed above. This is also true for the energy efficiency measured in mega-operations per watt (—— / - - -). The measurements reveal that the usage of the on-package memory result in a slight decrease of the overall power consumption (- - - / ——) which matches with the values obtained for the package and the memory controllers. The latter decreases by around 4 W to 6 W for larger thread counts while the power consumption of the package only increases by 1 W to 3 W.

The power consumption of the CG kernel increases as well for rising thread counts, however logarithmically in this case. Although we see a performance optimum at around 60 threads, the optimal power efficiency is already reached at around 36 and 40 threads when using DDR4 memory and MCDRAM respectively. As the energy efficiency for the CG kernel is improved by up to 20 % when using MCDRAM, we should obtain best co-scheduling performance when running it on the MCDRAM. In contrast, EP is not sensitive to the assigned memory type and therefore we might see performance benefits on its execution using DDR4 memory in a co-scheudling scenario.

[1] http://www.megware.com/.

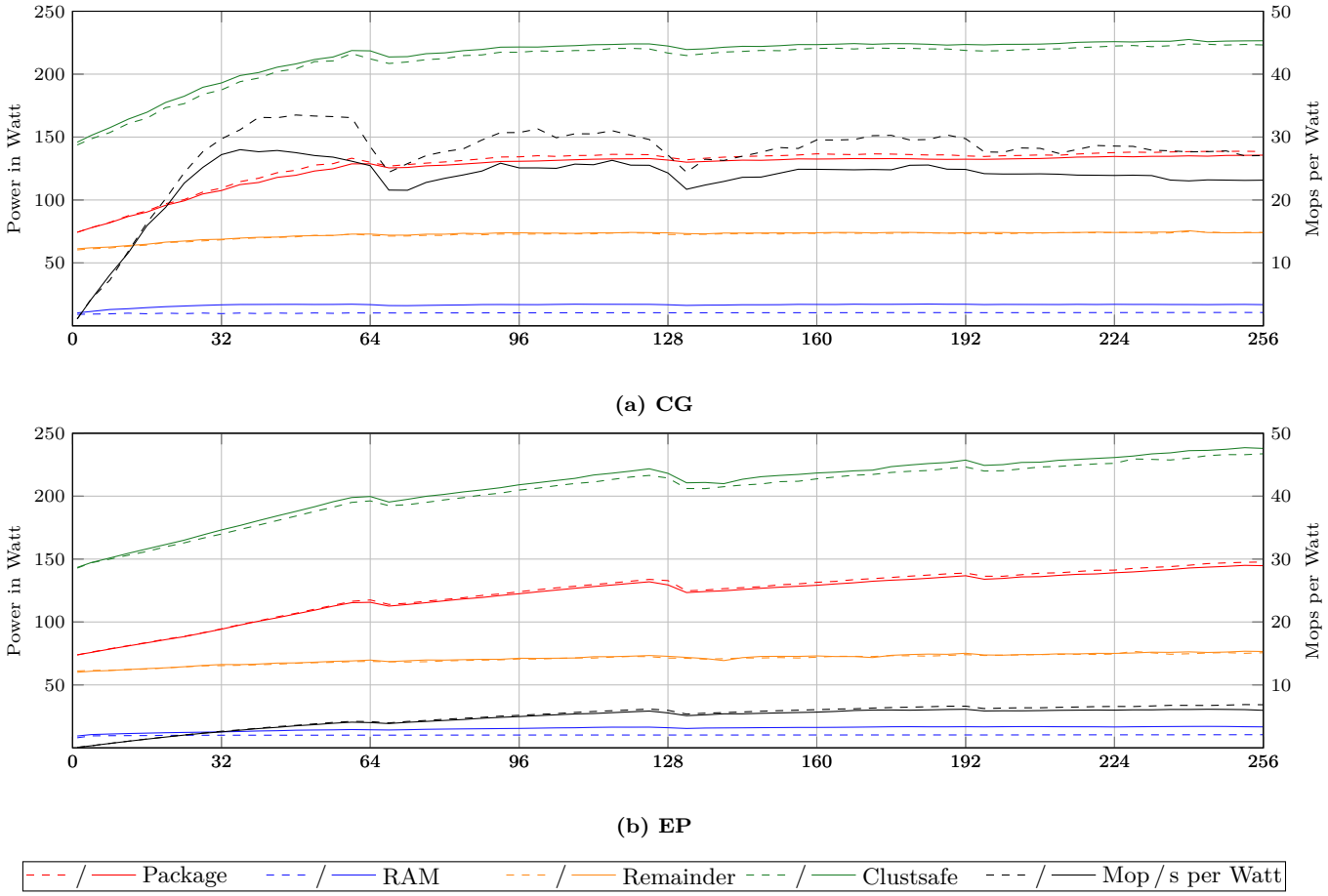


Figure 2: Power consumption (left y-axis) and power efficiency (right y-axis) of CG and EP using different thread counts respectively. The solid lines represent application runs using the DDR4 memory while the dashed lines show the results when using the on-package MCDRAM.

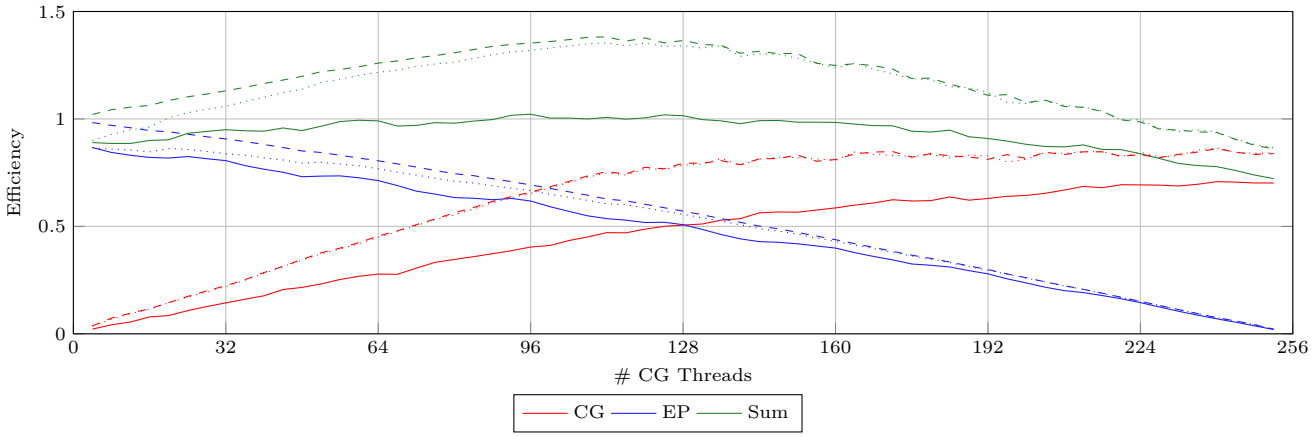
4.2 Co-scheduling Applications

An analysis of co-scheduling the CG and the EP kernel is presented in Figure 3 with respect to the applications' efficiency and power consumption. The former is computed based on the most efficient exclusive application run, i. e., using 60 threads for CG and 256 threads for EP, while assigning the MCDRAM respectively.

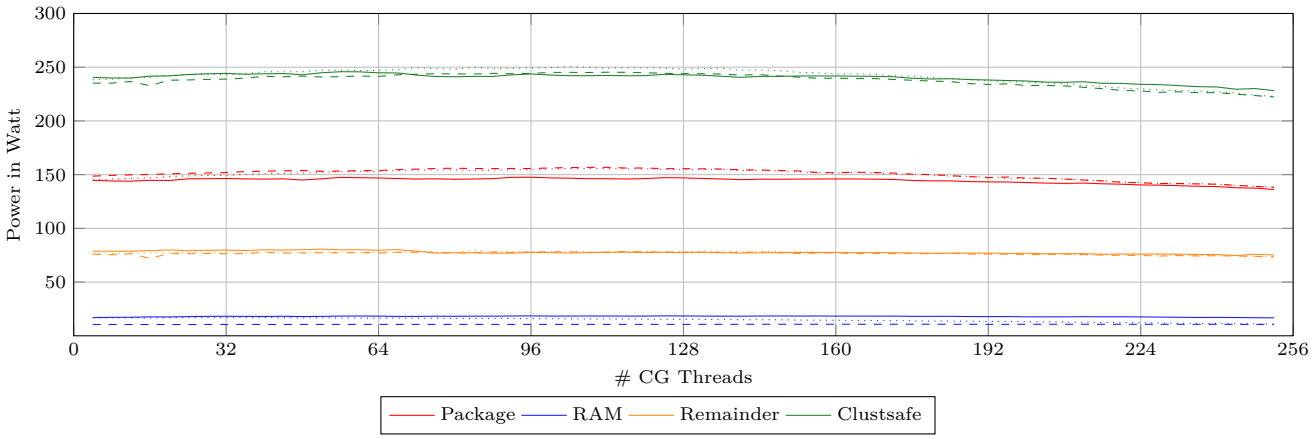
The solid lines represent application runs in which both kernels only allocate memory from the DDR4 memory. In this case, we do not expect larger benefits of co-scheduling both applications as the CG kernel strongly profits from the exection on MCDRAM (cf. Section 4.1). All the more surprising that we can observe a total efficiency greater than one for certain configurations.

In contrast, the dashed lines correspond to applications runs using only the MCDRAM for both kernels. In this scenario we can observe an increase of the overall efficiency by up to 38 % when assigning 112 to CG and the remaining cores to EP. Interestingly, around twice as much CG threads are necessary for a saturation of the efficiency in the co-scheduling case (instead of only 60 threads in the exclusive case). We expect this to be the result of a high contention within the on-die mesh caused by EP threads. There is only little influence on the power consumption when switching the memory types. The power consumption of the package increases when allocating the memory purely on the MCDRAM while that of the RAM decreases likewise. This is expected behavior since the former belongs to the package power domain. However, both CG and EP experience with 26.4 s and 8.5 s respectively a significant reduction of their performance compared to the exclusive cases. Since EP achieves almost the same efficiency on DDR4 memory and MCDRAM, i. e., it is only about 3 % faster on the MCDRAM when using 144 threads, one could assume that the efficiency of the co-scheduling scenario can be further increased when assigning distinct memory resources to each application.

The results obtained when running CG exclusively on the MCDRAM while EP uses DDR4 memory are represented by the dotted curves. However, in contrast to our expectations this strategy does not allow for an improvement of the overall efficiency. We expect that this is again caused by contentions within the on-die mesh as discussed before. Although this scenario realizes an exclusive assignment of the available resources, i. e., the two different memory types, the interconnect is still a shared resource and becomes the bottleneck. However, further research is necessary for a validation of these assumptions, e. g., a more sophisticated pinning of the threads could reduce the average path length to the memory controllers. The deviation from the MC-



(a)



(b)

Figure 3: Co-scheduling scenario of CG and EP: (a) the application efficiency which is computed based on the most efficient exclusive application run and (b) the power consumption for each application run. The solid, dashed, and dotted lines represent runs using DDR4 memory, MCDRAM, and DDR4 for EP and MCDRAM for CG respectively. The x-axis shows the number of threads used by the CG benchmark, the HTCs used by the EP benchmark can be computed via 256 minus the number of CG threads.

DRAM-only case for a high number of EP threads can be explained by its sensitivity to the memory type for high thread counts (cf. Section 4.1).

5. RELATED WORK

Simultaneous scheduling of different applications is common in the area of server and desktops systems. The hardware is designed with multiple HTCs for this type of simultaneous scheduling. However, most compute centers do not offer support for co-scheduling and rather assign the nodes explicitly to HPC applications.

A more common approach for the handling of underutilized nodes is frequency scaling. In doing so, the frequency is dynamically adapted which implicitly results in a reduction of the power consumption. Such an approach is obviously not able to increase the throughput but rather targets at an improved energy efficiency of the HPC systems. Wang et al. [10] discuss a scheduling heuristic reducing the overall system power consumption via Dynamic Voltage Frequency Scaling (DVFS). The Adagio [7] tool analyzes the time spent in blocking MPI function calls and decreases the CPU frequency accordingly. Thereby, the energy efficiency of the HPC system can be improved.

Energy efficient scheduling algorithms are also developed for task scheduling where a task graph representing a program is allocated and ordered on multiple processors. DVFS has been employed for a reduction of the energy consumption of the generated schedules, hence running the processors at heterogeneous speeds. Sinnen et al. discuss task scheduling algorithms [11] and show their potential to improve the energy efficiency on current CPUs.

6. CONCLUSION

In this paper we analyzed the benefit of co-scheduling on upcoming many-core architectures by taking the example of the Intel KNL. Furthermore, we examined the influence of different memory assignment strategies that are enabled by the additional layer in the memory hierarchy provided by the KNL. Our results show that co-scheduling can be valuable for manycore architectures as well, i. e., we could increase the

overall efficiency by up to 38 %. Against our expectations, we could not further improve the performance by dedicating the MCDRAM exclusively to the memory-bound application while serving the other from DDR4 memory. We assume that a high contention in the on-die mesh is the reason for the observed behavior.

For future work we plan a comprehensive analysis of the effects from different pinning strategies to confirm our assumptions. Therefore, we will use a set of micro-benchmarks to determine key figures such as latencies and maximum per-tile bandwidth.

7. ACKNOWLEDGMENTS

This research and development was supported by the Federal Ministry of Education and Research (BMBF) under Grant 01IH13004B (Project FAST). Furthermore, we want to thank MEGWARE who provided us with a Clustsafe to perform the energy measurements.

8. REFERENCES

[1] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS Parallel Benchmarks. *Int. Journal of High Performance Computing Applications*, Sept. 1991.

[2] J. Breitbart, J. Weidendorfer, and C. Trinitis. Case Study on Co-scheduling for HPC Applications. In *2015 44th International Conference on Parallel Processing Workshops*, pages 277–285, Sept 2015.

[3] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. RAPL: Memory power estimation and capping. In *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, pages 189–194, Aug 2010.

[4] M. Hähnel, B. Döbel, M. Völp, and H. Härtig. Measuring energy consumption for short code paths using RAPL. *ACM SIGMETRICS Performance Evaluation Review*, 40(3):13–17, 2012.

[5] J. Jeffers, J. Reinders, and A. Sodani. *Intel Xeon Phi Processor High Performance Programming Knights Landing Edition*. Morgan Kaufmann Publishers Inc, 2016.

[6] A. Ramachandran, J. Vienne, R. V. D. Wijngaart, L. Koesterke, and I. Sharapov. Performance Evaluation of NAS Parallel Benchmarks on Intel Xeon Phi. In *Proceedings of the 2013 42Nd International Conference on Parallel Processing*, pages 736–743. IEEE Computer Society, 2013.

[7] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch. Adagio: Making DVS Practical for Complex HPC Applications. In *Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, pages 460–469. ACM, 2009.

[8] S. Saini, J. Chang, R. Hood, and H. Jin. A Scalability Study of Columbia using the NAS Parallel Benchmarks. Technical report, Aug 2006.

[9] A. Sodani. Knights landing (KNL): 2nd Generation Intel Xeon Phi processor. In *2015 IEEE Hot Chips 27 Symposium (HCS)*, pages 1–24, Aug 2015.

[10] L. Wang, G. von Laszewski, J. Dayal, and F. Wang. Towards Energy Aware Scheduling for Precedence Constrained Parallel Tasks in a Cluster with DVFS. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 368–377. IEEE Computer Society, 2010.

[11] A. Zaliwski, O. Sinnen, and S. Lankes. Evaluating DVFS scheduling algorithms on real hardware. In *5th International Workshop on Power-aware Algorithms, Systems, and Architectures (PASA 2016), held in conjunction with 45th International Conference on Parallel Processing (ICPP 2016)*, pages 273–280, Aug 2016.