Deploying and Monitoring Hadoop MapReduce Analytics on Single-chip Cloud Computer

Andreas-Lazaros Georgiadis, Sotirios Xydis, Dimitrios Soudris

Microprocessor and Microsystems Laboratory
Electrical and Computer Engineering Department
National Technical University of Athens
Info: sxydis@microlab.ntua.gr

ABSTRACT

Modern data analytics applications exhibit scale-out characteristics, requiring large amount of computational power. Recent research has shown that modern manycore architectures forms a promising platform solution for this emerging type of workloads. In this paper, we present a framework for the deployment, monitoring and automated exploration of Hadoop MapReduce clusters implementing data analytics applications onto the Intel SCC manycore platform. We provide an in-depth analysis on the performance and energy characteristics of Hadoop MapReduce workloads on the Intel SCC, i.e. on a real-silicon manycore which highly differentiates from typical server and accelerator architectures. Through extensive experimentation, we show that there is a trade-off between the number of worker nodes and the per-node available I/O bandwidth and that intelligently scaling the frequency of data-nodes yields in energy savings with minimal impact on performance.

CCS Concepts

•Computer systems organization → Multicore architectures;
Cloud computing;

Keywords

Hadoop MapReduce; Intel Single-cloud Chip (SCC); Big Data Analytics; Workload Characterization

1. INTRODUCTION

Nowadays, more and more large business enterprises are getting involved in analyzing massive amount of data - so called *big data* - with the goal of converting big data to "big value". Typical data analysis workloads, i.e. business intelligence, machine learning, bio-informatics etc., are characterized by massive working sets, high degrees of parallelism and usually stringent performance and power constraints, exposing scale-out characteristics [1] that differ significantly from traditional desktop, or high performance parallel workloads. The business potential of big data and cloud services in turn forms a driving force for innovative application's and computer system's design. For example, Intel has recently launched

the Single-chip Cloud Computer (SCC), i.e. experimental silicon implementing a manycore platform [2] for addressing the micro-architectural challenges of such emerging workloads.

The MapReduce programming paradigm [3] has been emerged to to address scalability issues of big data analytics, farming out requests to a cluster of nodes that first perform filtering and transformation of the data (map) and then aggregate the results (reduce). Currently, Hadoop's MapReduce forms the defacto software framework for writing analytic applications processing vast amounts of data in-parallel on computer clusters in a reliable, fault-tolerant manner. Several research papers have analyzed the deployment of Hadoop analytics on cluster infrastructures [4, 5]. Only a limited number of publications have studied the deployment of Hadoop analytics on the emerging single-chip manycore platforms, considering mainly on GPU-based acceleration [6, 7]. Although GPU-based acceleration has proven very efficient in terms of performance, it focuses on typical kernel acceleration schemes, i.e. critical application kernels are offloaded to the GPU for acceleration, which in most cases are orthogonal to the behavior exposed by the Hadoop's MapReduce framework itself. Except from a limited number of studies utilizing simulation models [1], up to today there is a lack of a thorough performance/power analysis regarding to the deployment of Hadoop clusters implementing data analytic workloads onto real-silicon manycore architectures.

In this paper, we address the aforementioned research gaps by developing a framework that enables the deployment, monitoring and automated exploration of Hadoop clusters onto the Intel SCC. We provide detailed description of the tools developed so as to map Hadoop analytics on the Intel SCC in respect to the physical limitations that the platform introduces. In addition, we present a runtime monitoring framework that captures per-core, i.e. CPU utilization, and system metrics, i.e. network traffic, power consumption and board temperature. The proposed software infrastructure is subsequently utilized so as to characterize representative Hadoop's MapReduce applications on Intel SCC, in terms of performance, scalability and power consumption.

More in detail, we explore the performance and power characteristics of Hadoop MapReduce workloads over diverse cluster deployments, topologies that utilize different amount of SCC resources and frequency islands configurations. We draw conclusions regarding the scalability of those applications in terms of the number of cores they employ, exposing the design points for which network congestion or I/O bandwidth saturation become bottlenecks, resulting in suboptimal performance and utilization of the on-die resources. Furthermore, performance and power consumption tradeoffs are investigated so as to spot possible power and energy saving opportunities, exploiting the advanced island-based power delivery scheme of Intel SCC.

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

PARMA-DITAM '16, January 18 2016, Prague, Czech Republic

© 2016 ACM. ISBN 978-1-4503-4052-6/16/01...\$15.00

DOI: <http://dx.doi.org/10.1145/2872421.2872423>

2. RELATED RESEARCH

Today's volume servers are designed with processors that are essentially general-purpose. These conventional processors combine a handful of aggressively speculative and high clock frequency cores supplemented by a large on-chip cache. Several research activities have examined workload characteristics of scale-out [1] and traditional server workloads [8] demonstrating several interesting correlations to micro-architectural decisions, e.g. the inefficiency of large last-level caches (LLCs), found in conventional server architectures. Recently, tiled processors have emerged as competition to volume processors in the scale-out server space [9]. Recognizing the importance of per-server throughput, these processors use a large number of relatively simple cores [10], each with a slice of the shared LLC, interconnected via a packet-based mesh interconnect. Intel SCC [2] manycore platform provide experimental silicon prototypes implementing several of the aforementioned architectural decisions. In this paper we explore the behavior of such real-life scale-out workloads, implemented over the Hadoop's MapReduce framework, when they are mapped on the Intel SCC manycore platform, as opposed to the traditional cluster topology organization.

Regarding to Intel SCC's workload characterization, up to now research efforts are focused on performance analysis of typical parallel workloads. In [11], the authors examine several HPC kernels under various power settings and mapping decisions focusing on the impact of data exchange between cores on the overall system performance. In [12], Furst and Coskun developed a system monitoring software and benchmarks specifically targeted at investigation the impact of message passing on the performance and the power consumption of the Intel SCC, while Bartolini et al. [13] evaluate the impact of frequency scaling on the performance and power of many-core systems with MPI. In our paper, we differentiate from previous research by expanding Intel SCC's workload analysis towards characterizing performance and power consumption of scale-out workloads, which expose significantly different runtime behavior, compared to traditional parallel benchmarks.

3. INTEL SCC ARCHITECTURE

In this paper, we utilize the Intel SCC [2] 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 two IA P54C processing cores. 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. It has four on-die memory controllers capable of addressing a total of up to 32 GB of external memory. Each includes L2 cache memory for the two cores and a special memory called Message Passing Buffer. The tiles of the platform are divided into 6 voltage islands and a Voltage Regulator Controller included in SCC, provides the ability to regulate the operation voltage of each island separately. Each processing core runs its own lightweight Linux OS distribution which exposes a safe API for scaling the frequency and voltage of the cores of an island by dictating a frequency/voltage divider pair, ranging from $< 800MHz, 1.1V >$ down to $< 100MHz, 0.7V >$. Intel SCC is commonly connected by a PCI-express cable to a PC acting as a Management Console.

4. HADOOP BASICS: HDFS & MAPREDUCE

Hadoop Distributed File System (HDFS) has a master/slave architecture [14]. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage

storage attached to the nodes that they run on. The DataNodes are responsible for serving read and write requests from the file systems clients. The DataNodes also perform several operations, e.g. block creation, deletion etc. upon instruction from the NameNode.

Hadoop's Mapreduce [15] framework operates on HDFS objects. A MapReduce job usually splits the input dataset into independent chunks which are then processed in parallel by the map tasks. MapReduce operates exclusively on $< key, value >$ pairs, i.e. the framework views the input to the job as a set of $< key, value >$ pairs and produces a set of $< key, value >$ pairs as the output of the job. The framework calls the `map()` method for each key/value pair in the InputSplit for that task. The Hadoop MapReduce framework spawns one map task for each InputSplit generated by the InputFormat for the job. Thus, the number of maps is driven by the total size of the inputs, that is, the total number of blocks of the input files. During the Reduce stage, the `reduce()` method is called for each $< key, (listofvalues) >$ pair in the grouped inputs. The Reducer function reduces a set of intermediate values which share a key to a smaller set of values. Each Reducer implements three major operations, i.e. shuffle, sort and reduce, on the data fetched from the the mappers.

5. HADOOP CLUSTER DEPLOYMENT ON THE INTEL SCC

This section focuses on the deployment of Hadoop clusters on the Intel SCC. We discuss the prerequisite configurations for enabling the installation of a Hadoop cluster on Intel SCC, a lightweight failover mechanism we have developed and the differing Hadoop topologies considered.

5.1 Prerequisite Intel SCC & Hadoop configurations

Gentoo Linux for the Intel SCC: Since the Intel SCC Linux provides only a restricted application development API that does not cover the requirements of a Hadoop Cluster installation, we utilized a Gentoo image, specifically developed for the Intel SCC [16]. A Gentoo Linux image is mounted to each of the SCC cores.

Intel SCC Network Configuration: We have performed several modifications to the network configuration of the Intel SCC and the MCPC. First, we enabled the Intel SCC cores access the public Internet through the MCPC. The MCPC is configured as the default gateway and as a Network Address Translation (NAT) router for the Intel SCC cores, so that they can have access to the Gentoo Linux repositories needed for Hadoop installation. The *eth0* is configured as the primary WAN connection, while the Intel SCC cores are connected via *eth1* (LAN). We modified the routing table of the MCPC, by adding static routes, so that it can access the internal virtual network interfaces of the Intel SCC cores (*mb0*) directly. The purpose of this was to manipulate the way that the file blocks are stored in the HDFS namespace. This way, the HDFS file blocks are stored using the IP addresses of the *mb0* interfaces in the HDFS namespace and data transfer between DataNodes and TaskTrackers has been enabled.

Hadoop Java Heap Configuration: Heap size forms a very severe limitation due to the lack of sufficient main memory space for each core. Since 32 GB of memory are connected to the Intel SCC die through the memory controllers, only 640 MB of main memory is available for each core, thus Hadoop Cluster node. After stability testing of several MapReduce jobs for various values of maximum Java Heap Size, we concluded that 128MB is appropriate for the Hadoop Daemons and the Child JVMs, so that the Hadoop's Java processes will neither be killed by the OS (or make the core

Figure 1: Hadoop cluster topologies on Intel SCC.

freeze), nor be terminated due to out-of-memory error exceptions. This limited heap size constraints the deployment strategy to eliminate cluster configurations in which DataNodes and TaskTrackers run on the same cluster nodes. In our setup, we have configured DataNodes and TaskTrackers to run on different cores and have explicitly divided the on-die cluster to Hadoop Racks, so that the rack locality-aware scheduling mechanism of MapReduce is activated.

Hadoop MapReduce Configuration: In addition, the typical MapReduce framework configuration determines that the Reduce phase of a MapReduce job is triggered after only the 5% of the Map phase has completed. That is, reduce tasks are scheduled for cluster nodes where child JVMs are already executing Map tasks. With so little main memory available per core, this was evidently impossible. As a result, we have configured the Reduce phase to start after the 100% of the Map phase has completed successfully. For the same reason, the TaskTracker nodes are configured to run only one Map or Reduce task at a time. Moreover, we significantly reduced the file block size from the default value of 64 MB to 4 MB. Since a Map task is scheduled for each InputSplit, this decision reduces the I/O load for the DataNode cores.

5.2 Node Failover Watchdog

The presence of high I/O load and very low free main memory space causes some cores to freeze and become unreachable rather frequently. That is, at least one core freezing during the execution of a MapReduce job is the norm and not the exception on the Intel SCC. Configuring a replication factor greater than 1 looked a reasonable solution at first, since this is the out-of-the-box mechanism Hadoop provides so as to ensure data is always available. However, a replication factor greater than 1, forces the DataNodes replicate under-replicated file blocks, thus significantly increasing their I/O load, which is the reason that causes them to fail in the first place. Thus, configuring a replication factor greater than 1 causes the DataNode cores to freeze one after the other, rendering the completion of the MapReduce job impossible.

In order to tackle this problem, we followed a completely different approach. We implemented a node-failover watchdog script, which *pings* the Intel SCC cores periodically so as to ensure that all of them are up and running. In case a core is observed to be unreachable, i.e. *ping* receives no response packet, the SCC Linux is booted on the core immediately, the Hadoop Runtime Environment is set up and the corresponding Hadoop Daemon is started. This sequence of actions is also triggered if a core is reachable by ping, but the Hadoop Daemon it is supposed to run has been killed by the

OS. This way, we have overcome the complication of cores freezing frequently and have ensured the forward progress of MapReduce jobs despite the presence of this situation. Map or Reduce tasks may be terminated with exceptions during a core (especially a DataNode because of data being unavailable) is rebooted, but the retry mechanism of MapReduce guarantees that those tasks will be completed successfully once they are re-executed and the Hadoop Cluster is in a stable state.

5.3 Hadoop Cluster Topologies on the Intel SCC

We considered four Hadoop cluster topologies on Intel SCC, by scaling the number of allocated resources. Specifically, we considered the following configurations: i) a 16-node cluster topology with one master node (where the NameNode and the JobTracker are executed) and 15 slave nodes, which break down to 7 DataNodes and 8 TaskTracker, ii) a 24-node cluster with one master node and 23 slave nodes (7 DataNodes and 16 TaskTrackers), iii) a 32-node cluster with one master node and 31 slave nodes (15 DataNodes and 16 TaskTrackers), iv) a 48-node cluster with one master node and 47 slave nodes (15 DataNodes and 32 TaskTrackers).

Figure 1 depicts the two layouts of the examined Hadoop Cluster nodes on the Intel SCC die, i.e. the 16-node and 48-node cluster configuration. In the 16- and 24-node topologies as well as the 32- and the 48-node cluster topologies the DataNodes run on the same Intel SCC cores, thus they share the same HDFS namespace and differ only in the number of TaskTracker nodes they employ, which execute the Map and Reduce tasks of MapReduce jobs. Regarding to the placement of the cluster nodes on the Intel SCC die, we located the DataNodes at the edge of the Intel SCC die. Since DataNodes are responsible for storing file blocks in their local filesystem, they are expected to have heavier I/O load during the execution of a MapReduce job. Thus, placing the DataNodes closer to the memory controllers of the Intel SCC die, reduces the latency of their frequent accesses to the NFS.

6. MONITORING INFRASTRUCTURE FOR THE INTEL SCC

This section presents and analyzes the Runtime Monitoring Framework we have developed for the Intel SCC. We utilize a relational MySQL database to store the runtime metrics that are collected from the platform and each core, respectively.

Core and network monitoring: We set up a Ganglia monitoring cluster [17] that consists of the Intel SCC cores and the MCPC.

We have utilized the *gmond* daemon of Ganglia, which is responsible for collecting and transmitting the per-core runtime metrics. In order to collect per-core metrics, such as CPU utilization and Network Traffic, we have configured Ganglia with a UDP unicast topology. The Intel SCC cores are configured as deaf nodes, that is they do not listen to any unicast or multicast channel for cluster state information. The MCPC is configured as a mute node, so that it aggregates all the metrics collected from the Intel SCC cores.

The *gmond* instance that runs on the MCPC listens to one UDP unicast channel, which receives the runtime metrics of the Intel SCC cores. We have configured two collection groups. The first collection group concerns CPU-related runtime metrics. This collection group contains four metrics, which are i) *cpu_user*: the percentage of CPU utilization that occurred while executing at the user level, ii) *cpu_system*: the percentage of CPU utilization that occurred while executing at the system level, iii) *cpu_wio*: the percentage of time that the CPU was idle during which the system had an outstanding I/O request and iv) *cpu_idle*: the percentage of time that the CPU was idle during which the system did not have any outstanding I/O request. The second collection group concerns runtime metrics that capture the NoC traffic from and to each Intel SCC core. This collection group contains two metrics, which are *bytes_in* and *bytes_out* and capture the traffic in bytes/second that was received and sent by the core, respectively.

Power and temperature monitoring: In order to calculate the instant power consumption and the overall energy of the chip, we use short time intervals, because of the big variations that characterize the intensity of the electric current that is drawn by the Intel SCC board. Metrics that concern the voltage-supply of the Intel SCC board as well as the current intensity it draws are stored in a separate power file and concern the current time interval. Two scripts have been developed to extract the run-time metrics. The first script repeatedly executes the *sccBmc* command, it parses the output it provides and stores the voltage, current, power and calculates energy consumption through a trapezoid integral. The second script mines the power file and calculates the total energy consumption for the specific time interval. Finally, it queries the Ganglia Cluster so as to obtain the CPU utilization and the network traffic of each core, parses the XML that is returned by Ganglia and stores the extracted metrics in the monitoring database.

7. EXPERIMENTAL RESULTS

Experimental analysis is based on Hadoop MapReduce applications introduced in the Cloudsuite [18] and DCBench [19] benchmark suits. Both Cloudsuite and DCBench employ machine learning algorithms implementations that are included in Apache Mahout [15], a scalable machine learning library for Hadoop. Cloudsuite consists of several emerging scale-out applications, while DCBench contains representative workloads that are found in modern datacenters. We focus our experiments on data analytics workloads, i.e. we examine the Wordcount, K-Means Clustering and Frequent Pattern Growth from DCBench and the the Bayesian Classification algorithm from CloudSuite, respectively.

Runtime footprint analysis: Fig. 2 shows the monitored and annotated runtime behavior, i.e. the average CPU utilization for DataNode and TaskTracker as well as the power trace of the Intel SCC platform, for the Bayes Classification application on the 48-node HDFS cluster topology. The DataNodes and the TaskTrackers configured to operate at 800 MHz. The depicted case is quite representative, since similar behavior has been recorded for the Map and Reduced phases on the rest of the examined applications. From Fig. 2(a), we can observe that DataNode exhibit very low CPU utilization. This observation is consistent across the

Figure 2: Bayes Classifier monitoring data with phase annotation, i.e. average CPU utilization (red: CPU user utilization, yellow: CPU system utilization, blue: outstanding I/O, green: CPU idle) for DataNode and TaskTracker and power trace of the Intel SCC platform.

all the examined benchmarks, and as shown later in this section enables the down scaling of the core operating frequencies with minimal performance impairment. The CPU utilization plots of the TaskTracker nodes also denote the period at the beginning of each MapReduce job, when the mahout job is distributed and expanded at those cores. This period is characterized by a high percentage of CPU cycles due to outstanding I/O. The overview utilization figure also depicts that the CPU idle period that is observed in the beginning of each MapReduce job is attributed to the execution of the setup Map task in one TaskTracker node. Also, as shown in Figure 2(c), Intel SCC’s power trace is highly sensitive not only to CPU utilization but also to the on-chip network traffic generated by the outstanding I/O operations.

Performance-energy workload characterization: Figure 3 depicts our analysis concerning the behavior of the examined analytic workloads executed on top of various Intel SCC Hadoop cluster topologies. Both the DataNodes and the TaskTrackers of each cluster topology have been configured to operate at the maximum frequency of 800 MHz. The idle nodes of each topology (if any) operate at the minimum frequency of 100 MHz (no power gating available in Intel SCC) and with deactivated the *gmond* daemon. We focus on analysing the performance in terms of execution time and the energy consumption of the examined applications. *As shown, there is no a one-size-fit-all solution and the best cluster topology is highly dependent on the workload characteristics.* **Wordcount** benefits from a increased number of TaskTracker nodes, both in execution time and in energy consumption. However, the energy consumption gain is not proportional to the decrease of the execution time, since the power that is drawn by the Intel SCC is higher as the number of nodes of the Hadoop cluster increases. Decreasing the TaskTracker node count, more Map tasks are issued to each TaskTracker node and since the Hadoop’s HashPartitioner distributes evenly the intermediate $\langle key, value \rangle$ pairs among the reducers, a slight increase in the duration of the Reduce phase is also observed. In addition, for a given number of TaskTracker nodes, both the execution time and the energy consumption of the

application deteriorate when the number of DataNodes is increased from 7 to 15. (24-Node Topology versus 32-Node Topology). This is to the fact that the DataNodes do not execute any computation regarding the MapReduce job and as a consequence, the application is charged with non utilized power consumption.. In contrast to WordCount, in **Bayes Classifier** experimental results in Figure 3(b) clearly indicate that the 48-Node cluster forms a non-optimal topology regarding to energy consumption. Bayes classifier completes at approximately the same time when it is executed on the 24-Node and the 48-Node cluster, but because of the lower power consumption of the 24-Node cluster, it consumes 22% less energy. Bayes classifier exhibits periods with a high percentage of idle CPU cycles due to outstanding I/O. This CPU idleness is reduced significantly as the number of cores decreases, since the requested I/O bandwidth and thus the I/O saturation diminish. As a consequence, the reduced parallelism that is imposed by the smaller number of TaskTracker nodes is mitigated by the fact that less CPU cycles are wasted for outstanding I/O requests, resulting in the same execution time and reduced energy consumption for the 24-Node cluster topology compared to the 48-Node cluster topology. However, this behavior would be most probably overturned if the total size of the input category based splits is increased, due to the fact that the CPU-intensive part of the application would be expanded and the impact of the reduced parallelism would be more intense. **Frequent Pattern Growth** and **k-Means clustering**¹ exhibit similar behavior, since both consider 1 InputSplit of the dataset. For the specific InputSplit, the applications do not fully utilize the parallelism that is offered by the underlying HDFS cluster, i.e. increasing the number of nodes does not reduce the execution time of the application, but increases it, since the idle CPU cycles period because of outstanding I/O is prolonged due to the fact that the increased number of nodes increases the I/O bandwidth congestion, leading in a higher percentage of wasted CPU cycles. One significant difference between these two applications, regards to the Reduce phase of all three MapReduce jobs of the Frequent Pattern Growth application. The numGroups parameter of the application is set to 1000, thus all Reduce tasks of the three MapReduce jobs receive intermediate $< key, value >$ pairs. As a consequence, the load of the Reduce tasks increases for cluster topologies with less TaskTracker nodes. However, this increased load is mitigated by the absence of a high percentage of idle CPU cycles because of outstanding I/O during the period when the mahout job is distributed and expanded in all TaskTracker nodes of the cluster, resulting in lower execution time and energy consumption for cluster topologies that employ less Intel SCC cores.

Impact of frequency scaling: Finally, we analyze the impacts of frequency scaling on execution latency and energy consumption of the MapReduce workloads. Considering the 48-node cluster topology, we have configured the frequency of DataNodes and the TaskTrackers to either 200 MHz, 533 MHz or 800 MHz. Each examined frequency setting represents one combination of those values. Figure 4 depicts the contour graph with the average latency and energy across the examined workloads. As shown, energy consumption appears to be driven primarily by the frequency of the TaskTracker nodes. The frequency of the DataNodes seems to have a negligible impact, since increasing the DataNodes frequency seems to slightly reduce the execution time and the energy consumption of the application for a given frequency for the TaskTracker nodes. The DN200-TT800 setting forms a very promising configuration, showing that scaling down the frequency of the DataNodes yields energy consumption savings without performance degradation. Like in all

¹We consider 5 clusters. Execution time and the energy consumption are normalized over the number of iterations.

Figure 4: Impact of frequency scaling on performance-energy.

other application the DN200-TT800 setting is optimal if both performance and energy consumption are taken into account.

8. CONCLUSION

This paper focused on the study of the runtime behavior of emerging data analytics workloads, implemented with the Hadoop MapReduce framework, deployed on the Intel SCC cores. We presented the developed framework that enabled the deployment and monitoring of Hadoop clusters on the Intel SCC in respect to the platform's physical limitations. Extensive experimentation regarding to the analysis of Hadoop MapReduce workloads over diverse cluster topologies has been performed, showing interesting performance-energy trade-offs in respect to the resource count of the active computation nodes and the on-chip network bandwidth, as well as the frequency allocation decisions of the data-nodes.

9. REFERENCES

- [1] Michael Ferdman, Almutaz Adileh, Onur Kocerberber, Stavros Volos, Mohammad Alisafae, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 37–48, New York, NY, USA, 2012. ACM.
- [2] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Paillet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. Van Der Wijngaart, and T. Mattson. A 48-core ia-32 message-passing processor with dvfs in 45nm cmos. In *Solid-State Circuits Conference Digest of Technical Papers*

Figure 3: Performance-Energy characterization of MapReduce workloads on Intel SCC Hadoop cluster topologies.

- (ISSCC), 2010 IEEE International, pages 108–109, Feb 2010.
- [3] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.
- [4] Jens Dittrich and Jorge-Arnulfo Quiané-Ruiz. Efficient big data processing in hadoop mapreduce. *Proc. VLDB Endow.*, 5(12):2014–2015, August 2012.
- [5] Jacob Leverich and Christos Kozyrakis. On the energy (in)efficiency of hadoop clusters. *SIGOPS Oper. Syst. Rev.*, 44(1):61–65, March 2010.
- [6] Bingsheng He, Wenbin Fang, Qiong Luo, Naga K. Govindaraju, and Tuyong Wang. Mars: A mapreduce framework on graphics processors. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 260–269, New York, NY, USA, 2008. ACM.
- [7] Miao Xin and Hao Li. An implementation of gpu accelerated mapreduce: Using hadoop with opencl for data- and compute-intensive jobs. In *Service Sciences (IJCSS), 2012 International Joint Conference on*, pages 6–11, May 2012.
- [8] Nikos Hardavellas, Ippokratis Pandis, Ryan Johnson, Naju Mancheril, Anastassia Ailamaki, and Babak Falsafi. Database servers on chip multiprocessors: Limitations and opportunities. In *CIDR 2007, Third Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*, pages 79–87, 2007.
- [9] B. Wheeler. Tiler sees opening in clouds. *Microprocessor Report*, 25(7):13–16, March 2011.
- [10] Kevin Lim, Parthasarathy Ranganathan, Jichuan Chang, Chandrakant Patel, Trevor Mudge, and Steven Reinhardt. Understanding and designing new server architectures for emerging warehouse-computing environments. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 315–326, Washington, DC, USA, 2008. IEEE Computer Society.
- [11] Philipp Gschwandtner, Thomas Fahringer, and Radu Prodan. Performance analysis and benchmarking of the intel scc. In *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, CLUSTER '11, pages 139–149, Washington, DC, USA, 2011.
- [12] John-Nicholas Furst and Ayse K. Coskun. Performance and power analysis of rcce message passing on the intel single chip cloud computer. In *Proceedings of the 4th Many-core Applications Research Community (MARC) Symposium*, MARC '11, 2011.
- [13] Andrea Bartolini, MohammadSadeq Sadri, John-Nicholas Furst, Ayse Kivilcim Coskun, and Luca Benini. Quantifying the impact of frequency scaling on the energy efficiency of the single-chip cloud computer. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '12, pages 181–186, San Jose, CA, USA, 2012. EDA Consortium.
- [14] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.
- [15] Hadoop: Open source implementation of mapreduce. <http://hadoop.apache.org/>.
- [16] Peter Troger Jan-Arne Sobania. Gentoo linux on intel scc. operating systems and middleware group, it systems engineering, university potsdam, 2010 <https://www.dcl.hpi.uni-potsdam.de/research/scc/gentoo.htm>.
- [17] Ganglia monitoring system: <http://ganglia.sourceforge.net>.
- [18] Cloudsuite, a benchmark suite for scale-out applications. <http://parsa.epfl.ch/cloudsuite/cloudsuite.html>.
- [19] Dcbench, a benchmark suite for data center workloads. ict, chinese academy of sciences. <http://prof.ict.ac.cn/dcbench/>.