Quantifying the Impact of Frequency Scaling on the Energy Efficiency of the Single-Chip Cloud Computer

Andrea Bartolini*, MohammadSadegh Sadri*, John-Nicholas Furst†, Ayse Kivilcim Coskun† and Luca Benini*
*University of Bologna, DEIS - Bologna, Italy. Email: {a.bartolini, mohammadsadegh.sadr2, luca.benini}@unibo.it
†Boston University, ECE Department - Boston, MA, USA. Email: {jnfurst, acoskun}@bu.edu

***Abstract*—Dynamic frequency and voltage scaling (DVFS) techniques have been widely used for meeting energy constraints. Single-chip many-core systems bring new challenges owing to the large number of operating points and the shift to message passing interface (MPI) from shared memory communication. DVFS, however, has been mostly studied on single-chip systems with one or few cores, without considering the impact of the communication among cores. This paper evaluates the impact of frequency scaling on the performance and power of many-core systems with MPI. We conduct experiments on the Single-Chip Cloud Computer (SCC), an experimental many-core processor developed by Intel. The paper first introduces the run-time monitoring infrastructure and the application suite we have designed for an in-depth evaluation of the SCC. We provide an extensive analysis quantifying the effects of frequency perturbations on performance and energy efficiency. Experimental results show that run-time communication patterns lead to significant differences in power/performance tradeoffs in many-core systems with MPI.**

I. Introduction

Advancements in the process technology enables integrating dozens of cores on a single chip today. Such *many-core* systems bring new challenges in runtime system management as they offer a vast amount of operating points, covering various speed settings of cores and workload allocation scenarios. In addition, many-core systems are expected to leverage message passing interface (MPI) for inter-core communication, as opposed to the shared memory communication available in modern commercial multi-core systems. MPI is a standardized and portable message-passing system designed to function on a wide variety of parallel computers. MPI has been widely used in computing clusters for communication across nodes; however, many open research problems exist for single-chip many-core systems with MPI.

In tandem with the increase in the number of cores integrated on a single-chip, energy efficiency has become a critical constraint in computing. Today's processors typically include a set of built-in power management features. Dynamic voltage and frequency scaling (DVFS) is one of the most commonly utilized techniques for reducing power consumption of chips. Recent research has developed efficient DVFS techniques based on characterizing on-chip/off-chip workloads [4], identifying application phases with a high number of stall cycles [9], or using machine learning techniques to adapt to changing workload [6], [10]. The common goal in these approaches is reducing the performance overhead of operating at lower frequencies, as DVFS may incur severe slow-downs. These techniques improve the energy efficiency for the current single-core or multi-core systems with a few number of cores; however, they do not capture the unique performance-power tradeoffs in many-core systems with MPI.

978-3-9810801-8-6/DATE12/

The goal of this paper is to analyze the impact of core frequency perturbations on the performance of many-core systems with MPI. To enable this analysis, we utilize the *Single-chip Cloud Computer (SCC)* [8], which is a 48-core experimental processor created by Intel Labs. The SCC incorporates features such as a network-on-chip (NoC), DVFS capabilities, and support for MPI. The chip resembles a cloud of computers integrated into a single chip, as each core is capable of booting an OS instance. While infrastructures to measure real-time performance, power, and temperature exist for commercial systems [14], the unique features of the SCC require developing a novel framework for runtime monitoring of the system. This paper is the first to provide a comprehensive measurement infrastructure for the SCC. We then leverage this infrastructure to quantify frequency setting-performance correlations under a diverse set of workloads. The paper makes the following specific contributions:

- We develop an infrastructure to accurately track performance, power, and temperature of the SCC at runtime with very low performance overhead. Our setup collects performance counter data from each core, tracks main memory accesses, logs messages passed among cores, and measures power and temperature.
- To analyze the impact of frequency changes on the performance for many-core systems with MPI, we develop a benchmark suite for the SCC. The benchmarks in the suite cover a wide range of workload scenarios such as applications with different levels of communication distances and intensity or applications that stress different levels of the memory hierarchy.
- We conduct a large set of experiments using the monitoring infrastructure and the benchmark suite to measure the impact of frequency changes on performance and power. Our analysis demonstrates that the communication patterns have a significant impact on the performance loss. In fact, under frequency perturbations, differences in message traffic can create execution time overhead and energy loss variations from 10% up to 200%.

The rest of the paper starts with an overview of the related work. Section III provides the details of our mea-

surement infrastructure. Section IV discusses the application suite developed for the experiments. In Section V we demonstrate the results of our analysis and Section VI concludes the paper.

II. Related Work

DVFS is one of the most commonly used power management knobs in current processors for regulating energy consumption. Earlier solutions focus on single-core and embedded architectures [7], [4], [6]. More recent methods target specifically multi-core systems [13], [16], [17], [12]. Kim et al. [13] investigate how different DVFS granularities (i.e., chip-wide vs. per-core DVFS) in chip multiprocessors (CMP) impact the energy savings and the overhead of power management. They show that memory-bounded phases allow operating at low frequencies with limited performance loss. As applications often include phases of asynchronous memory events across the cores, per-core DVFS brings substantial advantages in energy savings [13].

As the number of cores integrated on a chip increases, per-core DVFS leads to the complexity of selecting the optimal voltage and frequency levels for all the cores among a vast number of operating points. To overcome this issue, de-centralized techniques have been proposed [16], [17]. These techniques utilize a hierarchical structure where a central controller allocates power budgets to each local controller. Each local controller then selects the optimal frequency assignment for a small set of cores based on the provided power budget.

Kai at al. [16] introduce a novel layer in the controller structure to perform group-level partitioning. Their technique controls a set of cores that are running the threads of a parallel application. For parallel applications, the authors show that a frequency selection policy that considers the threads as independent tasks leads to sub-optimal performance. Group-level partitioning allocates the frequency/power quotas to improve the performance of critical threads within a parallel application. Thread criticality can be identified in a shared memory system using a weighted cache miss index [3]. Using thread criticality in management avoids favoring high-IPC threads for assigning high frequencies, which can potentially result in imbalanced execution [1]. While some of these techniques are scalable to many-core systems, they mainly focus on shared memory architectures and do not consider how DVFS affects performance in MPI-based many-core systems [8].

At the cluster-level, recent studies [15], [11] show that performance models for MPI applications can be analytically derived, and these models are effective in identifying aggregating patterns that minimize the cluster's energy consumption. When the MPI protocol is implemented within cores of the same chip, however, the performance/energy tradeoffs change significantly owing to the substantial decrease in the communication latency between different MPI nodes in the NoC [19], [18]. Therefore, compared to multi-node MPI-based systems, scaling the frequency of cores in a single-chip many-core system with MPI has higher impact due to the stronger coupling of communication characteristics and performance.

This work differentiates from prior work in DVFS and many-core system management as we explicitly target single-chip many-core systems with MPI. We run frequency perturbation experiments on the Intel SCC, analyzing the impact on power and performance. In addition to providing an infrastructure for detailed monitoring of the SCC, our paper models how frequency map of cores, message traffic, and other architectural events correlate with application performance. We expect our results to guide energy-efficient policy design for DVFS on many-core systems with MPI.

III. Performance, Power, and Temperature Measurement Infrastructure for the SCC

Analyzing the impact of frequency scaling on the energy efficiency of the SCC requires monitoring performance, power, and temperature of the system at runtime. The SCC includes unique hardware and software features compared to off-the-shelf multi-core processors; thus, a novel infrastructure is needed to enable accurate and low-cost runtime monitoring. This section discusses the relevant features in the SCC architecture and provides the details of the novel monitoring framework we have developed.

Hardware and Software Architecture of the SCC:

The SCC has 24 dual-core tiles arranged in a 6x4 mesh. Each core is a P54C CPU and runs an instance of Linux 2.6.38 kernel. Each instance of Linux executes independently and the cores communicate through a network interface. Frequency setting of the tiles can be scaled individually, whereas the voltage can be scaled for groups of four tiles. Each core has private L1 and L2 caches. Intra-core cache coherence is managed through a software protocol as opposed to commonly used hardware MESI/MOESI protocols. Each tile has a message passing buffer (MPB), which facilitates the message exchange among cores. The entire system is controlled by a board management microcontroller (BMC) that initializes or shuts down critical system functions. The SCC is connected by a PCI-Express cable to a PC acting as the Management Console (MCPC).

Each tile in the SCC contains two ring-oscillator based thermal sensors: one located in proximity of the router and the other located close to the bottom core's L1 cache. The BMC also includes a power sensor capable of measuring the full SCC chip power consumption. This sensor can be directly accessed from the SCC cores through an emulated register in the FPGA.

Each P54C core has two performance counters. These counters can be programmed to track various architectural events (such as number of instructions or cache misses) at periodic intervals. Performance counters can be accessed from the specific core they are located at by reading the dedicated registers. SCC system also includes reconfigurable extensions. The network-on-chip (NoC) is connected to an FPGA through a router. This FPGA chip can be used for adding useful features that are not available in the SCC. Currently the FPGA synthesizes 48 atomic counters, one global time stamp counter (GTSC), and a set of power measurement registers. All of these registers are memory-mapped in the address space of each core.

SCC software includes RCCE, which is a lightweight message passing library developed by Intel and optimized for SCC [19]. It uses the hardware MPB to send and receive messages. In this way, it avoids using the network layer abstraction and the TCP/IP protocol overhead for exchanging messages among different physical cores. At the lower layer, the library implements two message passing primitives *RCCE_put* and *RCCE_get*. These primitives move the data from a local buffer to the MPB of another core and move the data back from a remote MPB to local memory, respectively.

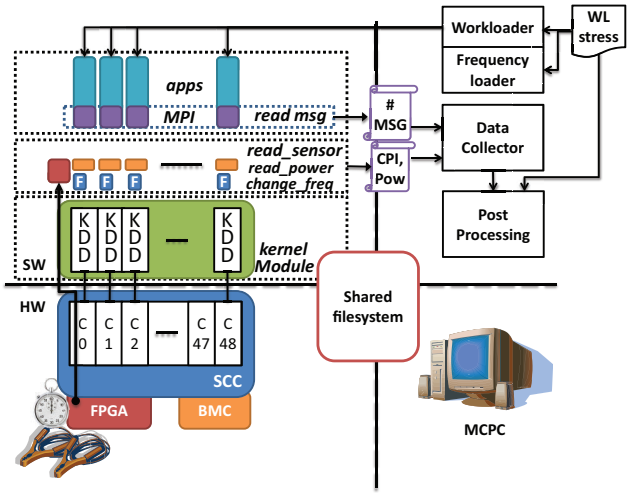

Fig. 1. SCC measurement framework. The figure demonstrates the SW components built for the SCC and the MCPC.

Figure 1 demonstrates the entire system setup including the SCC and the MCPC, and also the monitoring framework we have developed. On the SCC, we implemented the utilities to track performance counters, collect and calibrate thermal sensor data, collect power measurements, and log the message traffic. On the MCPC, we implemented the software to load the desired benchmarks and experimental configurations to the SCC and the software to analyze the collected data.

Software Modules Developed for the SCC:

- *Monitor KDD*: We developed a kernel module with two kernel timers to sample the performance counters and the temperature sensors. The module exports the collected data into the user space. In comparison to instrumenting the application code, our kernel module has the main advantage of decoupling the core activity logging from the application execution. In addition, the kernel timer ensures low overhead for sampling the counters. We use a sampling interval of 100ms in our experiments.
  Temperature sensors have a dedicated kernel timer that executes only in one core of each tile. We implemented an alpha filter inside this timer to remove the sensor noise. The thermal sensors are originally uncalibrated in the SCC. Thus, the thermal sensor readings need to be processed by software to provide meaningful data. By measuring the core power consumption, the ambient temperature, and the sensor output over a large variety of workload, we extracted the calibration constant for each sensor using a least square fit [2].
- *read_sensor*: We wrote a user-space program that gathers the performance counters and the tile temperature readings from the KDD Monitor and saves them into a log file. It executes every 100ms and has very low overhead: 54us@ 533MHz and 75us@ 166Mhz for collecting each sample. The trace collection can be triggered and stopped by sending the signal SIGUSR1 to the read_sensor process. The read_sensor program also collects the GTSC counter values at the beginning and at the end of its execution. The GTSC counter provides a global time reference for all the cores and it is not sensitive to frequency scaling. The OS timers on the SCC have known accuracy issues in presence of frequency changes. Thus, we use the GTSC value to measure the benchmark execution time.
- *read_power*: We designed another user-space program, read_power, to gather the power meter measurements for the cores, the routers, and the memory controllers. This program also collects the motherboard temperature. These power and temperature values are collected by accessing the dedicated memory mapped register in the FPGA at every 1s interval. Note that the power meter only provides the power reading for the SCC chip, and power measurements at the tile or core-level are not available.
- *change_freq*: We designed a user-space program to change the frequency of cores. This program executes on the core the frequency change is applied. The new frequency value is passed as a parameter and written in the frequency control register of the specific tile. This control register directly changes the tile clock divider.
- *Message Logger*: We modified the lower level *RCCE_put* and *RCCE_get* routines in the RCCE library to log the number of messages sent and the source/destination of each message. At the end of each parallel thread the library generates a log containing the communication matrix. Each element in the matrix $\{m_{i,j}\}$ corresponds to the number of messages that $core_i$ has sent to $core_j$. In addition, we instrumented the RCCE library to trigger the *read_sensor* daemon to start logging the performance counters at the beginning of each the parallel thread and save the trace at the end of it.

Software Modules Developed for the MCPC:

- *Stress files*: These files contain the frequency vector and the benchmark sequence for the tests. For each benchmark, the stress file provides the name, number of threads, and the cores to allocate the benchmark. The *app-loader* and the *frequency loader* load the files on the SCC to start the experiments.
- *App-loader*: We wrote a set of python scripts that run on the MCPC. These scripts load the stress configuration files and start the RCCE benchmarks in SCC.
- *Frequency loader*: This script first loads the stress file that contains the frequency setting for each core in

the SCC. Second, it executes the *change_freq* daemon remotely in each SCC core to apply the new frequency setting.

- *Post-processing SW*: We designed a SW module for processing the collected data. This script interfaces with the *app-loader* and the *frequency loader* to receive the experimental setup. For each benchmark, the script collects the logs and parses them to extract useful statistics. The module contains a front-end component written in Python and a back-end part written in Matlab, allowing the implementation of complex analysis functions. The post-processing SW enables extracting empirical models that correlate frequency changes with performance, energy, and temperature through mining a vast amount of data.

In this paper, we use the monitoring infrastructure described above for analyzing the impact of frequency changes on energy efficiency on the SCC. The framework can also be leveraged for enabling runtime management policies on the SCC computer.

IV. Application Space

We utilize a set of benchmarks to assess the performance of the SCC under a variety of operating conditions. In addition to expanding the benchmarks provided by Intel, we design several microbenchmarks to stress different parts of the system. We select the following application and synthetic benchmarks as they provide a heterogeneous set of performance data to use during analysis and creation of our model.

Intel benchmarks:

- *Share*: Tests the off-chip shared memory access.
- *Shift*: Passes messages around a logical ring of cores.
- *Stencil*: Solves a simple PDE with a basic stencil code.
- *Pingpong*: Bounces messages between a pair of cores.
- *NPB*: NAS Parallel Benchmarks, LU and BT.

Custom-designed microbenchmarks:

- *Bcast*: Broadcasts messages from one core to all other cores.
- *DRAM*: Executes an ALU operation on a circular buffer in memory. At each iteration a read-write is performed for one entry of the circular buffer. The dimension of circular buffer is 4MB.
- *L1 and L2*: Have the same principle as the DRAM benchmark. Circular buffer size for L1 is 16KB and buffer size for L2 is 32KB.

Table I categorizes the Intel benchmarks based on IPC, L1 instruction misses, number of messages, and execution time. All parameters are normalized with respect to the number of instructions executed to enable a fair comparison. Each benchmark in this categorization runs on two neighbor cores on the SCC (only 2 cores active). We observe that Share does not have messages and is an example of a memory-bounded application. Shift represents a message intensive application and Stencil represents a high-IPC application. Finally, Pingpong is a low-IPC application with a large number of L1 cache misses. Note that Stencil, Shift, Share, and Pingpong benchmarks all rely on the blocking send / receive calls from the RCCE API.

TABLE I
Benchmark Categorization

Benchmark	L1CM	Time	Msgs	IPC
Share	High	High	Low	Low
Shift	High	Low	High	Medium
Stencil	Low	Low	Low	High
Pingpong	High	Medium	Medium	Low

We designed the Broadcast (Bcast) benchmark based on Pingpong, which sends messages among cores and test latencies. Instead of having a source and a single destination as in Pingpong, Bcast sends messages from a single core to multiple cores. DRAM, L1 and L2 benchmarks are custom-designed applications that do not use the RCCE library. We use these memory benchmarks to compare the execution time of memory-intensive (DRAM) or cache-intensive (L1 and L2) applications in our analysis and to analyze the tradeoffs under frequency scaling. The benchmarks are run with the following configurations to cover a wide range of workload scenarios.

- *Intel benchmarks*: Stencil, Shift, Share, and Pingpong benchmarks run on *pairs* of cores in the following configurations:
  - *0-hop*: Cores on the same tile. (e.g., cores 0-1)
  - *1-hop*: Cores on neighboring tiles (e.g., cores 0-2)
  - *2-hops*: Cores on tiles that are 2-hops distance away (e.g., cores 0-4)
  - *3-hops*: Cores on tiles that are 3-hops distance away (e.g., cores 0-6)
  - *8-hops*: Cores on corners (e.g., cores 0-47)

  We run either a single pair (2 cores active) or concurrently run 24 pairs for each benchmark (48 cores active) in the experiments with Intel benchmarks.
- *NPB*: The LU benchmark runs on 32 cores and the BT benchmarks runs on 36 cores. These choices are due to restrictions with LU and BT software preventing execution on all 48 cores.
- *Bcast*: The Bcast benchmark is run with 1 core sending messages to N cores ($1 \leq N \leq 47$).
- *DRAM, L1, L2*: Our custom-designed memory benchmarks are single-threaded benchmarks. We run 48 instances on 48 cores.

V. Experimental Evaluation

This section provides the results of our analysis. First we carry out a test to highlight how different benchmarks behave under frequency perturbation. For this experiment, we execute each of the Intel benchmarks on two cores of the SCC. One of the cores ($core_A$) is always Core0 (corner core) while the second one ($core_B$) moves step by step towards the opposite corner from 1-hop distance to 8-hop distance in the SCC floorplan. Then for each of these configurations we perturb frequency of the tiles of the running cores to generate the following frequency patterns: $\{tile_A, tile_B\}$: $\{f_{min}, f_{min}\}$, $\{f_{max}, f_{min}\}$,$\{f_{min}, f_{max}\}$,$\{f_{max}, f_{max}\}$. In our experiments, f_{max} is 533 MHz and f_{min} is 166 MHz. These

choices for f_{max} and f_{min} were made considering the stability of the SCC system.

During each run, we probe the (1) execution time overhead, (2) the full chip power saving, (3) the energy saving, (4) instructions per second (IPS), (5) message density, and (6) memory access density. For the first three metrics, the baseline has the $\{f_{max}, f_{max}\}$ setting and $core_A$ is adjacent to $core_B$. The message density is computed as the number of messages sent and received by a given core divided by the total number of instructions, whereas the memory access density is computed as a ratio of the non-cacheable memory read performance counter over the total number of instructions[1].

In Figure 2 we show the results of the stress patterns for nearest and farthest position of $core_B$ (denoted with "near" and "far"). Bcast is an asymmetric benchmark, meaning the communication direction is always from a source core to a destination, and has a high message density. In contrast to the other benchmarks, the performance loss when only one core has lower frequency while running Bcast is significantly lower when $core_B$ (the destination core) is slowed down. This is not the case for the other benchmarks, as other benchmarks include bidirectional communication among cores. In addition, Bcast strongly benefits from running both cores at the same frequency, as the execution time overhead and the energy are lower compared to running cores at different frequencies. Note that as we do not scale the voltage of the cores, we do not observe energy savings when both the cores run at the minimum frequency.

Pingpong and Share show similar trends even though they are significantly different applications. Their execution times have lower sensitivity to frequency changes compared to other benchmarks. For Share, this effect can be explained looking at IPS, which is lower compared to the other benchmarks. Also, the memory read access statistics show that Share is memory-bound.

Shift has high message density and, similar to Bcast, its execution time strongly depends on the core frequency. Stencil, on the other hand, has low memory access density and high IPS. Stencil's throughput decreases significantly as we scale down the frequency of one core. In addition, similar to Share, Stencil's execution time increases when running on cores far from each other. This increase is mainly due to the usage of the shared memory buffers allocated off-chip (for Share) or in the MPB (for Stencil). For Stencil, increasing the distance reduces the throughput (IPS) considerably. The slow-down saturates when just one core runs at low frequency. In this case, scaling down the other core does not affect the execution time as Stencil uses barrier synchronization.

We observe that all the benchmarks benefit from having the core frequencies equalized. In fact, for most of the benchmarks we see significant energy savings when moving from only one core operating at low frequency to both cores operating at lower frequency. An unbalanced frequency configuration can lead up to 2x energy efficiency loss.

This analysis highlights the importance of predicting the impact of a generic frequency perturbation on the execution time of a parallel benchmark. In addition, it suggests that message density, IPS, and frequency can be utilized to estimate the changes in execution time. Following this intuition, we generate a data set by running each of the Intel benchmarks separately in all the cores of the SCC (i.e., each benchmark with 24 pairs on 48 cores). We also run Bcast (1-to-47 cores), our memory microbencmarks, and the NPB benchmarks. We run all the applications at a wide range of frequency settings, including all cores at max/min frequency and checkerboard patterns. The collected data have five columns: execution time overhead compared to 1-hop case when all the cores are running at maximum frequency; a vector of core frequency settings; IPS; the message density; and the weighted average of the frequency settings of the communicating cores[2].

We then use half of this data-set to train four neural networks (NNs). For all the four NNs, *execution time* is the target output to be predicted. The first NN takes only the frequency vector as input. The second takes frequency vector and IPS. The third NN takes frequency vector, IPS, and the message densities. Finally, the fourth NN takes frequency, IPS, the message densities and the weighted average frequency of the communicating cores as inputs. We choose a two layer NN topology with *tansig* and *purelin* activation functions for the first and second layer[5]. We check the performance of the network against a linear multi-variable model (demonstrated as Lin) under two validation data-sets: The first data-set (Mix) contains the other half of the original data-set whereas the second one (NPB Only) contains only the NPB benchmarks. This second data-set with NPB benchmarks is useful to evaluate the prediction performance of the network for real-life-like workloads.

TABLE II
ACCURACY OF THE PERFORMANCE PREDICTORS.

	R-SQUARE	
INPUT	Mix data-set	NPB Only
NN Freq.	0.42	0.01
NN Freq., IPS	0.59	0.01
NN Freq,IPS,MSG	0.83	0.09
NN Freq,IPS,MSG,MSGFREQ	0.87	0.80
Lin	0.44	0.18

Table II shows the residuals for the four NN and the linear fit computed on the two validation data-sets. Low residual values correspond to lower accuracy predictions whereas a value of 1 in the residual means a fully accurate prediction. The table shows that the message density information improves the prediction performance significantly; however, it is not sufficient for highly parallel and more realistic benchmarks (i.e., NPB data-set). We see that it is important to consider not only the overall number of messages received but also the frequency of the sender

[1]Note that SCC does not include a performance counter to track the L2 miss rate. We tested the memory read performance counter with microbenchmarks and verified that there is a strong correlation with off-chip memory access.

[2]The weighted average for the i^{th} core is the sum of the core frequencies weighted by the number of messages sent by each core to the i^{th} core.

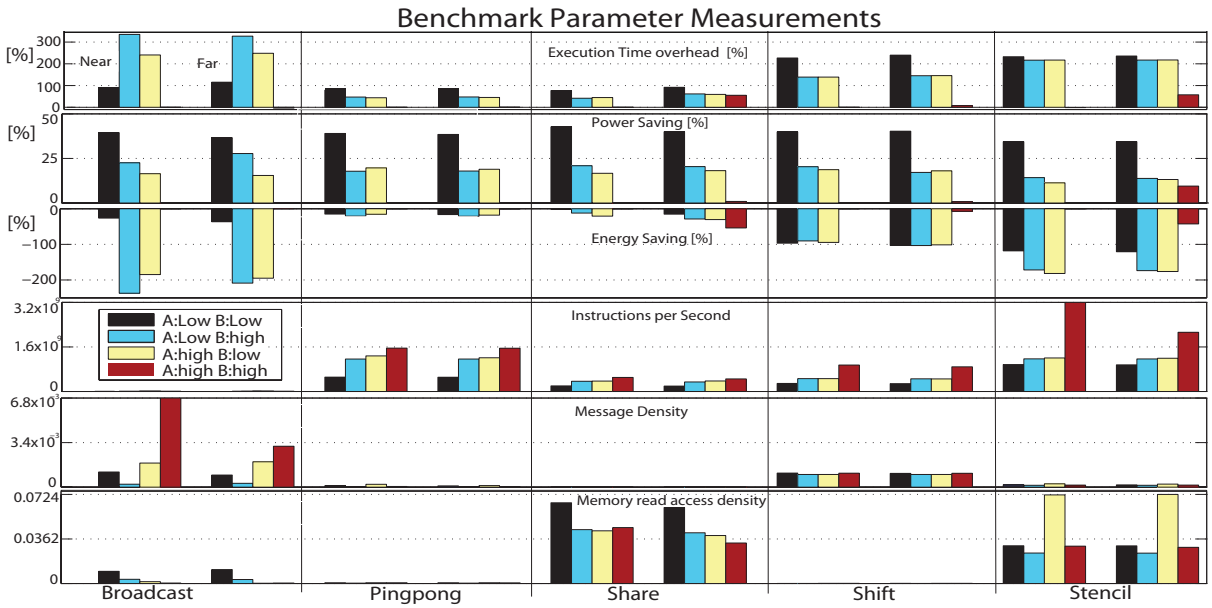

Fig. 2. Sensitivity of the Intel SCC benchmarks to frequency scaling.

core. This analysis verifies that the number of messages and the sender core's performance cannot be neglected while predicting overall application performance. As a result, the communication patterns and message densities should be included in DVFS performance optimization on many-core systems with MPI.

VI. Conclusion

In this paper, we have introduced an infrastructure to accurately track performance, power, and temperature of the SCC at runtime with very low performance overhead. We then used this framework to analyze the impact of frequency changes on the performance for many-core systems with MPI. We designed a large set of experiments using the monitoring infrastructure. Our analysis demonstrates that the communication patterns have a significant impact on performance and energy efficiency, varying from 0.1x to 2x, compared to the baseline case without frequency perturbations. We also show that accurate performance prediction for many-core systems with MPI needs to take core performance and message traffic into account. We believe our infrastructure and results will provide guidelines for designing runtime management policies for many-core systems with MPI.

Acknowledgements

This work was in part supported by the EU FP7 Projects Pro3D (GA n. 248776) and Therminator (GA n. 248603). J. N. Furst has been funded by the Undergraduate Research Opportunities Program (UROP) at Boston University.

References

[1] A. Alameldeen and D. Wood. Ipc considered harmful for multiprocessor workloads. *Micro, IEEE*, 26(4):8–17, july-aug. 2006.

[2] A. Bartolini et al. A system level approach to multi-core thermal sensors calibration. In *PATMOS'11*, pages 22–31, 2011.

[3] A. Bhattacharjee and M. Martonosi. Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors. *SIGARCH Comput. Archit. News*, 37:290–301, June 2009.

[4] K. Choi, R. Soma, and M. Pedram. Dynamic voltage and frequency scaling based on workload decomposition. In *ISLPED '04*, pages 174–179, 2004.

[5] H. Demuth, M. Beale, H. Demuth, and M. Beale. Neural network toolbox for use with matlab, 1993.

[6] G. Dhiman and T. S. Rosing. Dynamic voltage frequency scaling for multi-tasking systems using online learning. In *ISLPED '07*, pages 207–212, New York, NY, USA, 2007. ACM.

[7] K. Flautner and T. Mudge. Vertigo: automatic performance-setting for linux. In *OSDI '02*, pages 105–116.

[8] J. Howard et al. A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling. *Solid-State Circuits, IEEE Journal of*, 46(1):173–183, jan. 2011.

[9] C. Isci, G. Contreras, and M. Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *MICRO 39*, pages 359–370, Washington, DC, USA, 2006. IEEE Computer Society.

[10] H. Jung and M. Pedram. Supervised learning based power management for multicore processors. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 29:1395–1408, September 2010.

[11] N. Kappiah, V. W. Freeh, and D. K. Lowenthal. Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs. In *SC '05*, pages 33–. IEEE Computer Society, 2005.

[12] G. Keramidas, V. Spiliopoulos, and S. Kaxiras. Interval-based models for run-time dvfs orchestration in superscalar processors. In *CF '10*, pages 287–296, New York, NY, USA, 2010. ACM.

[13] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 123–134, feb. 2008.

[14] Y. Kim, J. Choi, A. Sivasubramaniam, J. Srebric, Q. Wang, and J. Lee. Modeling and managing thermal profiles of rack-mounted servers with thermostat. In *Proceedings of IEEE 13th International Symposium on High Performance Computer Architecture*, pages 205–215, feb. 2007.

[15] D. Li et al. Power-aware mpi task aggregation prediction for high-end computing systems. In *IPDPS '10*, pages 1–12, april 2010.

[16] K. Ma, X. Li, M. Chen, and X. Wang. Scalable power control for many-core architectures running multi-threaded applications. *SIGARCH Comput. Archit. News*, 39:449–460, June 2011.

[17] A. Mishra, S. Srikantaiah, M. Kandemir, and C. Das. Cpm in cmps: Coordinated power management in chip-multiprocessors. In *SC '10*, pages 1–12, nov. 2010.

[18] P. Salihundam et al. A 2 tb/s 6, $times$, 4 mesh network for a single-chip cloud computer with dvfs in 45 nm cmos. *Solid-State Circuits, IEEE Journal of*, 46(4):757–766, april 2011.

[19] R. F. van der Wijngaart, T. G. Mattson, and W. Haas. Light-weight communications on intel's single-chip cloud computer processor. *SIGOPS Oper. Syst. Rev.*, 45:73–83, February 2011.