Journal of
Low Power Electronics
Vol. 10, 1–19, 2014

Message Passing-Aware Power Management on Many-Core Systems

Andrea Bartolini[1,*], Can Hankendi[2], Ayse Kivilcim Coskun[2], and Luca Benini[1]

[1] *DEI, University of Bologna 40136, Bologna, Italy*
[2] *ECE Department, Boston University, Boston, MA, 02215, US*

(Received: 1 July 2014; Accepted: 15 October 2014)

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 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 voltage and frequency scaling on the performance and power of many-core systems with message passing (MP) based communication, and proposes a power management policy that leverages the communication pattern information to efficiently traverse the search space for finding the optimal voltage and frequency operating point. We conduct experiments on a 48-core Intel Single-Chip Cloud Computer (SCC), as our target many-core platform. The paper first introduces the runtime monitoring infrastructure and the application suite we have designed for an in-depth evaluation of the SCC. We then quantify the effects of frequency perturbations on performance and energy efficiency. Experimental results show that runtime communication patterns lead to significant differences in power/performance tradeoffs in many-core systems with MP-based communication. We show that the proposed power management policy achieves up to the 70% energy-delay-product (EDP) improvements compared to existing DVFS policies, while meeting the performance constraints.

Keywords: Message Passing, Dynamic Voltage and Frequency Scaling, Many-Core Systems.

1. INTRODUCTION

Building complex, high-performance cores is limited by the tight power and temperature constraints; thus, the current processor design trends have shifted towards integrating a number of smaller, lower power processing cores connected by an on-chip network. The number of cores on a single chip increases rapidly every year toward *many-core* systems. The development of run-time management techniques has been a key element in system design for dynamically optimizing power and performance tradeoffs depending on the application characteristics to achieve energy-efficient operation.[1–5]

Many-core systems bring additional challenges in run-time system management, as they offer a vast amount of operating points, such as various combinations of voltage and frequency settings across many cores. In addition, many-core systems are expected to leverage message passing (MP) based communication for inter-core communication, as opposed to the traditional shared memory communication available on commercial multi-core systems. As MP provides efficient methods to handle concurrency (i.e., synchronization) on many-node systems, MP-based communication has been widely used in computing clusters.[6] However, there are still many open research problems for single-chip many-core systems that utilize MP for communication.

A common runtime energy efficiency control knob in modern processors is dynamic voltage and frequency scaling (DVFS). Recent research has developed efficient DVFS techniques based on characterizing on-chip/off-chip workloads,[2] identifying application phases with a high number of stall cycles,[3] or using machine learning techniques to adapt to changing workload phases.[4,5] As DVFS may incur severe performance degradation, the common goal of these approaches is reducing the negative performance impact of operating at lower frequencies. Although these techniques improve the energy efficiency

*Author to whom correspondence should be addressed.
Email: a.bartolini@unibo.it

for the current single-core or multi-core systems with a small number of cores, they do not address the unique performance-power tradeoffs in many-core systems with MP. For instance, on a many-core system with MP, a local DVFS change in one or few cores may severely impact the performance of the entire parallel application running on the system, due to the potential effects of DVFS on communication time.[7,8]

In this paper, we propose a DVFS policy that takes the communication characteristics into account to improve the energy efficiency on many-core systems with MP-based communication. In order to efficiently utilize DVFS on many-core systems with MP-based communication, it is essential to capture the communication characteristics of the applications. Therefore, we first develop a measurement infrastructure that can monitor the communication characteristics of MP-based applications. We next analyze and optimize the impact of the core frequency on the performance of many-core systems with MP. We then use our observations to create a DVFS policy targeted towards parallel applications running on a many-core system with MP. We conduct all experiments on the Intel *Single-Chip Cloud Computer* (*SCC*) as a representative many-core system, which consists of 48 cores with MP capabilities.[9] SCC incorporates a network-on-chip (NoC), DVFS capabilities, and support for MP-based communication.[a] The chip resembles a cloud of computers integrated into a single chip, as each core is capable of booting an OS instance. While infrastructure to measure real-time performance, power, and temperature exists for commercial systems,[10] the unique features of the SCC require developing a framework for runtime monitoring of the system. We provide the details of our comprehensive measurement infrastructure for the SCC, expanding the one presented in our recent work[11] with voltage scaling features. We leverage this infrastructure to quantify the correlations among frequency settings, voltage settings, performance and energy for a diverse set of workloads. Finally, we compare the proposed DVFS policy with commonly used policies and we present the benefits of utilizing the communication characteristics while making DVFS decisions. The paper makes the following specific contributions:

- We revise the benchmark suite presented in our recent paper[11] with a new set of programmable benchmarks that represent MP-based parallel applications. This includes corner case applications, as well as NAS Parallel Benchmarks.[12] This allows us to create a training dataset to evaluate the performance of different strategies for learning energy consumption models.
- We show that both non-linear and linear model templates fail to accurately model and predict the effect of DVFS on the performance of the target applications, which is a significantly limiting factor for model-predictive based power management solutions.
- To evaluate the impact of both voltage and frequency decisions on the performance and power of real applications, we conduct a large set of experiments using the monitoring infrastructure and the benchmark suite presented in Ref. [11]. Our analysis demonstrates that the communication patterns significantly impact the achievable energy savings. We show that applying DVFS policies without considering the communication characteristics of the applications can lead to an energy increase up to 130%, whereas considering the communication patterns while making DVFS decisions can provide up to 50% energy savings.
- We propose and implement a novel power management strategy that exploits the benefits of keeping the communicating cores at the same frequency levels. Our policy iteratively searches the voltage and frequency setting space to reach an optimum operating point. Our results highlight that MP-agnostic power management strategies do not always save energy and can lead to 1.9× energy-delay product (EDP) increase for CPU-bound applications.
- We show that MP-aware policies need to account for not only direct MP communication, but also indirect communication.[b] Our results show that considering both direct and indirect communication patterns significantly improves the energy and EDP savings and always outperforms the performance of the MP-agnostic policy, leading up to 80% of EDP reduction for applications with 4 threads and up to 70% for applications with higher levels of parallelism (i.e., 16 threads).

The rest of the paper starts with an overview of related work. Section 3 provides the details of our measurement infrastructure and discusses the application suite developed for the experiments. Section 4 presents a set of preliminary tests conducted to analyze the sensitivity of the MP applications to frequency scaling. In Section 5, we present a novel MP-aware energy saving policy. Section 6 demonstrates the efficacy of the proposed policy when compare to standard ones. Section 7 concludes the paper.

2. RELATED WORK

Most of the commercial processors today support several voltage-frequency settings, and DVFS is among the most commonly used power management knobs for regulating power consumption. Most DVFS solutions focus on single-core and embedded systems.[1,2,4] More recent methods specifically target multi-core systems.[13–16] Kim et al. investigate how different DVFS granularities such as chip-wide versus per-core DVFS in multi-core systems

[a] Standard SCC Message Passing library namely RCCE supports only blocking send and receives.

[b] Consider a case, where *core A* sends messages to *core B* and *core B* sends messages to *core C*. We call the communication in between the *core A* and *core B* "direct" and the one in between "*core A* and *core C* "indirect."

impact the energy savings and the overhead of power management. As applications often include phases of asynchronous memory events across the cores, per-core DVFS brings substantial advantages in energy savings.[13] In addition, applying more aggressive DVFS during application phases with a high number of stall cycles is an attractive approach for reducing energy at limited performance cost.[3]

As the number of cores integrated on a chip increases, per-core DVFS leads to high complexity, as the optimal voltage and frequency levels for all the cores need to be selected among a vast number of operating points. De-centralized techniques have been proposed to limit the overhead of per-core DVFS.[14, 15] These techniques utilize a hierarchical structure, where a central controller allocates power budgets to local controllers. Each local controller then selects the (locally optimal) frequency assignment for a small set of cores based on the provided power budget.

Kai et al. introduce a novel layer in the controller structure to perform group-level partitioning of threads.[14] 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 and 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.[17] Using thread criticality during management avoids favoring high-IPC threads for assigning high frequencies, which can potentially result in unbalanced execution.[18] 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 the performance on MP-based many-core systems.[9]

Performance of the MP-based systems has been traditionally studied at the cluster level. Recent studies show that performance models for MP applications can be analytically derived, and these models are effective in identifying groups of threads to be aggregated in the same shared-memory node to minimize the computing cluster's energy consumption.[19, 20] Springer et al. present a methodology based on a combination of performance prediction, profiling and benchmark re-execution to find the optimal frequency and task mapping scheduling.[7] The results show that typically less then 15 executions are needed to find a valid schedule. Rountree et al. instead present a framework, where the target MPI-application is first profiled for each combination of available DVFS power point and results are combined on an LP problem to find the optimal scheduling and mapping.[8] Even if this solution provides significant energy savings, the initial profiling overhead is not negligible and does not scale with a larger number of nodes and operating points.

When the MP protocol is implemented within the cores of the same chip, the performance/energy tradeoffs change significantly owing to the substantial decrease in the communication latency among different MP nodes in the NoC.[21, 22] Therefore, compared to the multi-node MP-based clusters, per-core frequency scaling decisions on an MP-based single-chip many-core system have higher impact on the application runtime, due to the strong coupling of communication characteristics and performance.

Some recent work has used Intel SCC as an experimental platform for developing power management techniques for many-core systems. Ioannou et al. present a power management scheme that is composed of a set of local controllers and a supervisor.[23] The local controller identifies and predicts the MP phases by means of a Super-Maximal Repeat phase Predictor that stores MP call sequences and finds the current one. Iteratively at each phase repetition the local controller computes a new frequency setting based on the previous one and the program phase performance overhead. Whereas MP-phase prediction allows the policy to adapt to the workload phases, the information of communicating cores is neglected and the final frequency value is chosen by the supervisor to minimize the voltage on a per-voltage island base. Gammel et al. investigate the power behavior of scientific Partitioned Global Address Space (PGAS) application kernels on the SCC platform.[24] They show that various PGAS primitives need to be considered in the power management strategies. David et al. show a power management strategy on SCC for parallel workloads that uses queues to buffer the communication in between threads.[25] The power manager uses the information on data arrival and queues state to select the tile frequency at runtime. These results are tightly coupled with the programming model and communication abstractions and thus cannot be directly applied to the MP case. Li et al. combines DVFS and dynamic concurrency throttling (DCT) to reduce the energy consumption of hybrid MPI/OpenMP applications by identifying the slacks due to inter and intra-node interactions on a large multi-core cluster.[26] The proposed algorithms heavily relies on code profiling, therefore hard to generalize for a wider set of applications. Chen et al. propose network monitoring techniques for guiding DVFS policies to reduce energy consumption.[27] Although the proposed technique is based on monitoring the stress on the network components, communication aspect has not been considered. In a recent work, Bogdan et al. propose an optimal control algorithm for power management in MPSoCs with multiple voltage/frequency islands.[28] The proposed algorithm models the power optimization problem as a fractal-state equations rather than a linear model to take into account the NoC aspects, such as queue utilization in a network.

In this work we focus on single-chip many-core systems with MP, and demonstrate that the communication patterns of applications running on such systems strongly influence the performance and energy tradeoffs of DVFS. We leverage this observation to devise an intelligent search algorithm for finding the optimal voltage-frequency settings

for each core on many-core systems, while limiting the search space and overhead. We conduct our experiments on the SCC as the representative many-core system and design an MP-aware DVFS policy based our analysis. The experimental evaluation is conducted on blocking message passing, as it is supported in SCC and is commonly used in large scale HPC application.

3. PERFORMANCE, POWER AND TEMPERATURE MEASUREMENT INFRASTRUCTURE

Analyzing the impact of voltage and frequency scaling on the energy efficiency of the SCC requires
(1) monitoring the performance and power of the system at runtime and
(2) a software framework that connects the monitoring results with DVFS actions.

The SCC includes unique hardware and software features compared to off-the-shelf multi-core processors; thus, a novel infrastructure is required to enable accurate and low-cost runtime monitoring. This section discusses the relevant features in the SCC architecture and provides the details of our novel monitoring framework. While the implementation described in this section is specific to the Intel SCC, we believe that the core of the components of the infrastructure would be highly relevant to other many-core system platforms. In other words, the lessons learned from designing the measurement infrastructure on the SCC would provide guidance to researchers, who seek to build similar measurement infrastructures.

3.1. Hardware and Software Architecture

SCC has 24 dual-core tiles arranged in a 6×4 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 NoC. 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 exchanges among the cores. The entire system is controlled by a board management microcontroller (BMC) that initializes or shuts down critical system functions. SCC is connected through a PCI-Express cable to a PC acting as the Management Console (MCPC).

Each P54C core has two performance counters, which can be programmed to track various architectural events, such as number of instructions or cache misses at periodic intervals. Performance counters can be accessed by reading the dedicated registers that are available on each SCC core. SCC system also includes reconfigurable extensions. The NoC is connected to an FPGA through a router. This FPGA chip can be used for adding useful features that are not available in 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. BMC includes a power sensor that is capable of measuring the full SCC chip power consumption. This power sensor can be directly accessed from the SCC cores through an emulated register in the FPGA.

SCC software includes RCCE library, which is a lightweight message passing library developed by Intel and optimized for the SCC.[21] 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. RCCE provides message passing functions, which implements a subset of MPI[29] primitives on the SCC hardware. This paper is not intended to compare RCCE with MPI standard, but to explore the potential for MP-aware power management strategies. At the lower layer, the RCCE 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.

Figure 1 demonstrates the full system setup including the SCC and the MCPC, and also the monitoring framework we have developed. On the SCC-side, we implemented utilities to track performance counters, collect power measurements, and log the message traffic. On the MCPC-side, we developed custom softwares to load the desired benchmarks and experimental configurations to SCC and to analyze the collected data.

3.2. Software Modules Developed for Runtime Monitoring and Analysis

- *Monitor KDD*: We developed a kernel module with two kernel timers to sample the performance counters. 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 100 ms in our experiments.
- *read_sensor*: We wrote a user-space program that gathers the performance counters from the KDD Monitor and saves them into a log file. It executes in every 100 ms with a negligible overhead (i.e., 54 μs@533 MHz and 75 μs@166 MHz 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

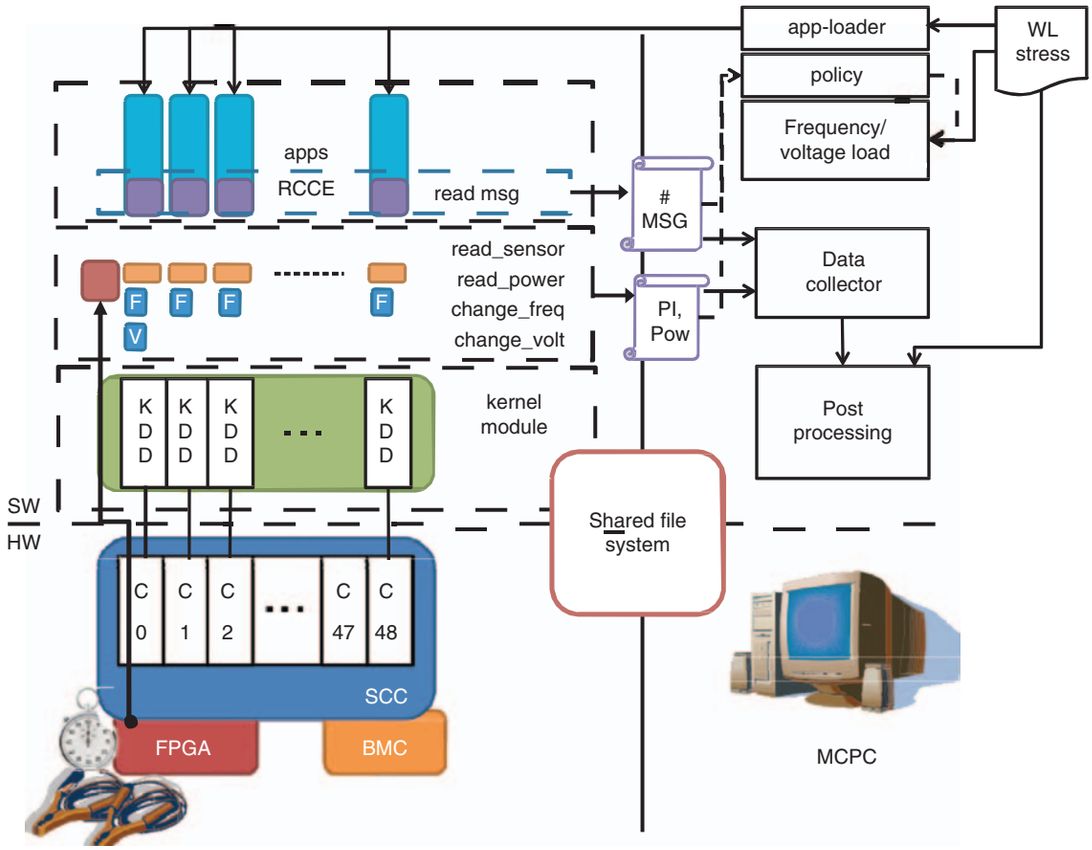

Fig. 1. SCC measurement framework. The figure demonstrates the SW components built for the SCC and the MCPC.

and it is not sensitive to frequency scaling. OS timers on the SCC have known accuracy issues in the presence of frequency changes. Thus, we use the GTSC value to measure the benchmark execution time.

• *read_power*: We designed a user-space program, *read_power*, to gather the power meter measurements for the cores, the routers, and the memory controllers. These power values are collected by accessing the dedicated memory mapped register in the FPGA in every second. Note that the power meter only provides the power reading for the whole 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 the cores. This program executes on the core, where 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, which directly changes the tile clock divider.

• *change_volt*: We designed a user-space program to change the supply voltage of the different voltage islands. The program executes on one core in two modes. The first mode finds the minimum voltage that can be applied to each voltage island. This mode is used after applying a new set of frequency values to the system. For each island, our program gathers the frequency of the different tiles, computes the maximum frequency of these tiles, and applies the minimum voltage required to sustain this frequency. The second mode of the program is used for determining and applying the required minimum voltage value before increasing the frequency of a tile.

• *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, 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 parallel thread and to save the trace at the end of the thread.

3.3. Software Modules Developed for MCPC

• *Stress files*: These files contain the frequency vector and the benchmark sequence for the experiments. For each benchmark, the stress file provides the name of the benchmark, the number of threads, and the specific cores to allocate the benchmark. The *app-loader* and the *voltage/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 to load the stress configuration files and to start the execution of RCCE benchmarks on SCC.
• *Policy*: This script implements the energy-aware DVFS policy. The script iteratively executes the target application while changing the frequency and voltage settings with the goal of increasing the energy efficiency. This is done by triggering the *App-loader* and controlling the *Voltage/Frequency loader*. The specific policy we implemented is discussed in Section 5.
• *Voltage/Frequency loader*: This script first loads the stress file that contains the frequency setting for each core in the SCC. The stress file can be defined offline or generated at run-time by the *Policy*. Next, it executes the *change_freq* daemon remotely on each SCC core to apply the new frequency setting and the *change_volt* to minimize the energy consumption.
• *Post-processing SW*: We designed a software module for processing the collected data. This module interfaces with the *app-loader* and the *frequency loader* to receive the experimental configuration. It also collects the logs and parses them to extract useful statistics. The post-processing software contains a front-end component written in Python and a back-end part written in Matlab, which allows the implementation of complex analysis functions. The post-processing software also enables extracting empirical models that correlate frequency changes with performance, energy, and temperature through mining a vast amount of data, while enabling the performance evaluation of the *Policy*.

4. DVFS ANALYSIS FOR PARALLEL WORKLOADS RUNNING ON MANY-CORE SYSTEMS

As many-core systems have distinct characteristics when compared to multi-core systems, it is essential to choose and design applications that can exploit the many-core characteristics. Most of the multi-core benchmark suites, such as PARSEC, SPLASH, are designed to evaluate shared-memory architectures. Therefore, these application suites are not suitable for assessing many-core systems with MP. Furthermore, previous work already showed that the communication density of these benchmark suites is very limited to evaluate an MP-system.[30] Thus, we use both the existing many-core benchmarks and also custom-designed micro benchmarks that can be programmed to generate variety of communication densities on a many-core system, which provides us a broader application space. In this section, we provide details and analysis of the benchmarks used in this work.

4.1. Application Space

We utilize a set of benchmarks to assess the performance of the SCC under various operating conditions. These benchmarks are derived from the ones presented in our recent paper.[11] In addition, we design programmable custom micro-benchmarks to stress different parts of the system. We select the following applications and synthetic benchmarks to evaluate various DVFS policies under various conditions:

Intel Many-Core 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.

Programmable Custom-Designed Microbenchmarks:
• bcast: Broadcasts messages from one core to all other cores.
• PairMP: This synthetic benchmark is derived from the pingpong benchmark and allows us to generate various message traffic densities across two different cores.

Table I categorizes the Intel benchmarks based on IPC, L1 instruction misses, number of messages, and execution time. We normalize all measurements with respect to the number of instructions executed. Each benchmark in Table I runs on two neighbor cores on the SCC (i.e., only 2 cores active). We observe that share does not exhibit any communication, therefore it is an example of a memory-bound application. shift represents a message intensive application and stencil represents a high-IPC application. Finally, pingpong is a low-IPC application that generates a high number of L1 cache misses. Note that stencil, shift, share and pingpong benchmarks rely on the blocking send/receive calls from the RCCE library.

We design the bcast benchmark based on pingpong, which sends messages across cores, which enables us to evaluate the message passing latencies. Instead of having a source and a single destination as in pingpong, bcast sends messages from a single core to multiple cores. PairMP benchmark is a custom-designed application that combines computationally intensive phases together with message exchange phases. The messages are sent between the active cores. This micro-benchmark can be configured to have each $core_i$ sending N_i number of messages with different sizes/densities (MD_i) in each of the iterations (I_i) of an arithmetic loop on the dataset with the dimension of DD_i. By configuring these parameters, it is possible to

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
BT	Medium	Medium	Low	Medium
LU	Medium	Medium	Medium	Medium

generate a large variety of message, memory and computation patterns. We can modulate the communication-to-computation ratio by tuning N_i and I_i, while tuning the DD_i parameter allows us to change the memory access locality, which modulating the IPC of the application. If I_i is selected to be equal to zero, we can have a benchmark that only exchanges messages and we can maximize its message density by increasing MD_i parameter.

The programmable custom micro-benchmarks allow us to test a wide range of communication patterns and communication intensity scenarios. By leveraging both the Intel benchmarks, NAS and the custom benchmarks, our goal is to achieve an assessment over real-life workload scenarios, as well as the significant corner cases. In the rest of this section, we analyze the performance of the target applications under various execution conditions. We utilize the measurement framework presented in the previous section (i.e., Section 3) to obtain the results.

4.2. Sensitivity on On-Chip Network Traffic

We analyze the sensitivity of the MP benchmarks to the on-chip network traffic. We evaluate the sensitivity by configuring the PairMP to maximize the message exchange rate across two cores. We map the PairMP benchmark threads in two central tiles with one link and two routers in between and we measure the execution time. We then iterate on the experiments by generating additional traffic through the routers and link through executing additional PairMP applications around them. Since the routers of SCC use a static x–y routing, allocating additional PairMP applications on different tiles between the target routers and link allows us to increase the message traffic. Figure 2(a) shows the number of additional PairMP traffic generator application pairs and the target PairMP benchmarks (T). We perform the experiment by increasing the number of PairMP traffic generator applications and each of experiment is repeated N times, where $N = 7$.

Figure 2(b) shows the number of additional traffic generator applications on the x-axis, and the average, maximum and minimum execution time of the target PairMP application for the different traffic congestions on the y-axis. We notice that even though the mean execution time for the target application is perturbed by the additional traffic, this perturbation is within the error range of the execution time measurements. This shows that message exchange in SCC using the RCCE library does not saturate the link and router bandwidth and thus multiple MP applications can be scheduled on distinct cores of SCC without perturbing each other. Thus, we can neglect the cross-interference across applications that are running on the same chip. In the following analyses, we execute only a single parallel application at any time.

4.3. Sensitivity on DVFS Scaling

In this section, we perform experiments to investigate how different benchmarks behave under voltage and frequency

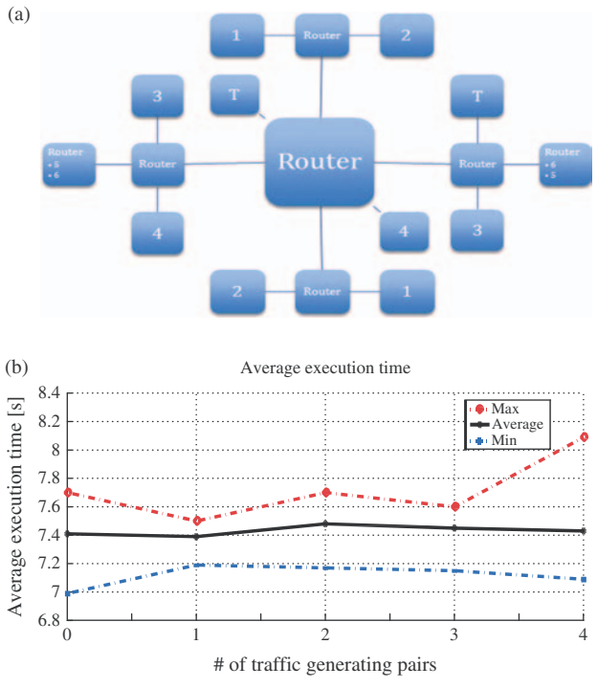

Fig. 2. (a) Network topology for the traffic generation, (b) MP application sensitivity on NoC router/link congestion.

perturbation. For this experiment, we execute each of the Intel many-core benchmarks on two cores of the SCC. One of the cores ($core_A$) is always Core 0 (i.e., 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 800 MHz and $f_{\min}$ is 166 MHz. We execute two runs of the entire test, one without voltage scaling and the other with voltage scaling.

In both the 1-hop and 8-hop settings, two cores are in different voltage islands, thus the voltage can be scaled independently for both of the cores ($core_A$, $core_B$).

In Figure 3, we report the execution time overhead (rows 1 and 4), the full chip power savings (rows 2 and 5) and the energy saving for both the frequency scaling and the voltage and frequency scaling (rows 3 and 6). In addition, we probe the instructions per second (IPS) (row 7), message density (row 8) and memory access density (row 9). For the first three metrics, the baseline has the $\{f_{\max}, f_{\max}\}$ setting and $core_A$ tile is adjacent to $core_B$ tile. 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 the ratio of the non-cacheable

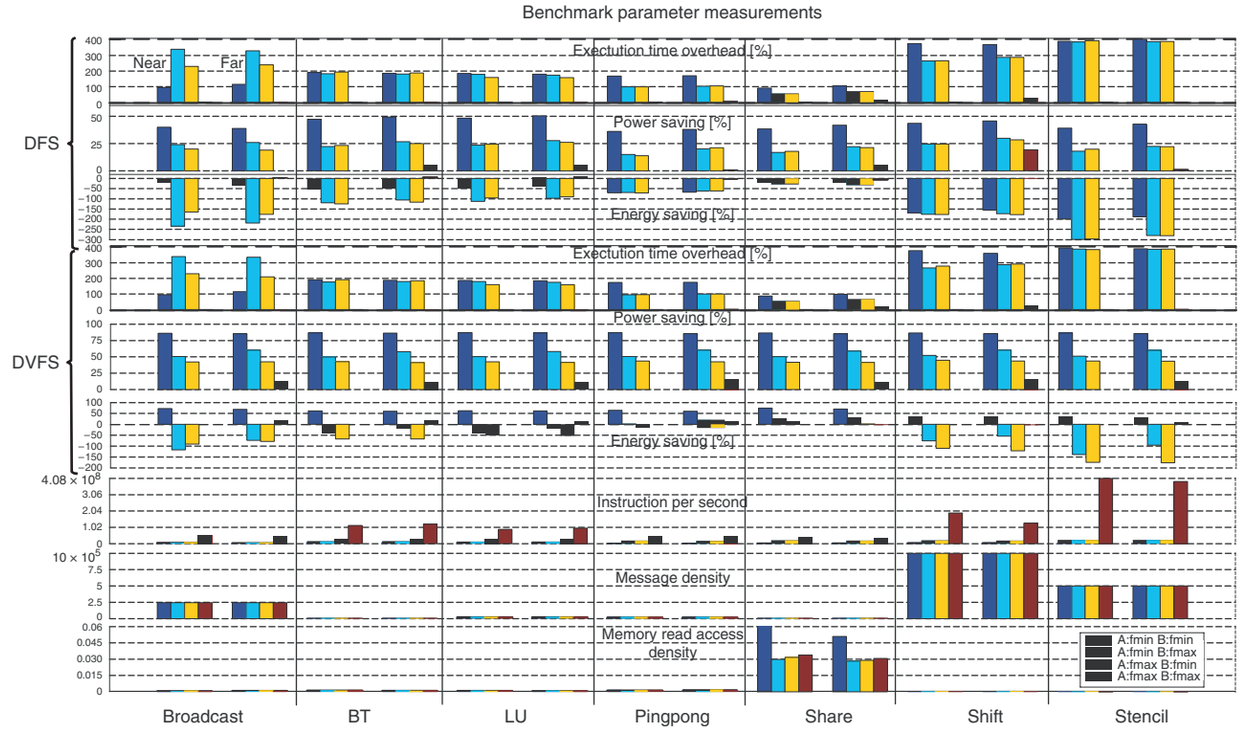

Fig. 3. Sensitivity of Intel many-core benchmarks to voltage and frequency scaling.

memory read performance counter over the total number of instructions.[c]

In Figure 3, we show the results of the stress patterns for nearest and farthest position of $core_B$ (denoted as "near" and "far"). On the top, we report the metrics computed for the frequency scaling test (DFS), whereas on the center plots we report the results for the voltage and frequency scaling case (DVFS). bcast is an asymmetric benchmark, meaning that 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 for bcast when only one core has lower frequency 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 bi-directional 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.

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 by its low IPS. Also, the memory read access statistics show that share is memory-bound. On the other hand, shift has high message density and its execution time strongly depends on the core frequency, which is similar to bcast stencil has low memory access density and high IPS. Therefore, stencil's throughput decreases significantly, as we scale down the frequency of one core. In addition, stencil's execution time increases when running on cores far from each other, which is similar to share. 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. BT and LU exhibit similar behavior, even though they are characterized by a significantly lower IPS that translates into a lower execution time overhead when running at lower frequency.

Furthermore, DFS always increases the energy consumption compared to DVFS, as the execution time overhead is higher than the power saving. We can also note that all the benchmarks benefit from matching the core frequencies. 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 low frequency. An unbalanced frequency configuration can lead up to 2× energy efficiency loss for the DFS case. The same consideration holds for the DVFS case. We notice that operating only one core at low-voltage and frequency

[c]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.

often translates to a energy loss up to 150%, whereas equally scaling the voltage and frequency of the cores leads to a significant power saving up to 30% for stencil. From the same plots, we also notice that the higher energy savings are achieved by the applications that have lower IPS, as the impact of DVFS on m. In addition, employing DVFS causes variation on the power consumption of $core_A$ and $core_B$ at lower frequencies. As this effect was not present in the DFS case, it suggests that even if the tiles are configured with the same voltage setting the two cores might have a mismatch in the power consumption. This can be due to either a mismatch in the voltage regulator performance or process variation that becomes more significant at lower voltages. By looking at the energy saving plots for the DVFS case, we notice the energy savings is a non-linear and non-convex function on the cores voltage and frequency scaling. Indeed, if only one core is scaled the energy increases, whereas if both the cores are scaled together energy consumption is significantly reduced.

These analyses highlight the importance of predicting the impact of a generic frequency perturbation on the execution time of a parallel benchmark. In addition, our observations suggest that the message density, IPS, and frequency selections are the major factors determining the execution time.

4.4. Modeling of Many-Core Applications with MP

In this section we present the results of our modeling analysis to verify the feasibility of capturing the relationship between the energy savings, the voltage-frequency settings and the application characteristics by an empirical model. If this model exists and has good accuracy, it is possible to predict the energy saving for given a frequency pattern, which then can be exploited by an optimization step to find the optimal DVFS settings for a given application. On the contrary, if this empirical model cannot be learned, the energy saving policy can only be reactive (i.e., based on a feedback loop) and optimized through heuristics.

For this purpose, we randomly generated 310 instances of the PairMP benchmark and executed on the 1-hop configuration. Then, for each application instance, we perturbed the frequency of the tiles of the running cores to generate the following frequency patterns: $\{\text{tile}_A, tile_B\}$: $\{f_{\min}, f_{\min}\}$, $\{f_{\max}, f_{\min}\}$, $\{f_{\min}, f_{\max}\}$, $\{f_{\max}, f_{\max}\}$. In our experiments, $f_{\max}$ is 800 MHz and $f_{\min}$ is 166 MHz.

Figure 4 shows the IPC (Instruction per Cycle), message density (# messages/# instruction retired) and memory access density (# of memory access/# instruction retired) space for each PairMP instance, when compared to the Intel many-core benchmarks. This indicates that we obtained a good coverage of all the Intel many-core benchmarks.

Figure 5(a) shows for each dataset instance on the x-axis the execution time slowdown and on the y-axis it shows the energy savings. Different colors/markers refer to different $\{tile_A, tile_B\}$ frequency and voltage settings. Figure 5(a) shows that for given a frequency/voltage setting, the energy savings are linearly proportional to the execution time slowdown. This allows us to simplify the task of modeling the energy saving by translating it into modeling the execution time slowdown. Figure 5(b) shows the sum of the frequency of the two cores (frequency accumulation) (x-axis) and the power savings (y-axis) for each instance of the dataset. Figure 5(b) shows, the power consumption is dominated by the DVFS level and it is robust to different application characteristics, as for each frequency level the values are clustered. As highlighted in previous section, we see that the power consumption of the two cores ($core_A$, $core_B$) is not symmetric, and $Tile_A$ is less efficient than $Tile_B$ at low voltages.

We combine these results in a dataset composed of $310 \times 4 \times 2$ instances. Each instance (i) of the dataset is a tuple composed of $(y^i, x_0^i, \ldots, x_k^i)$, where y is the

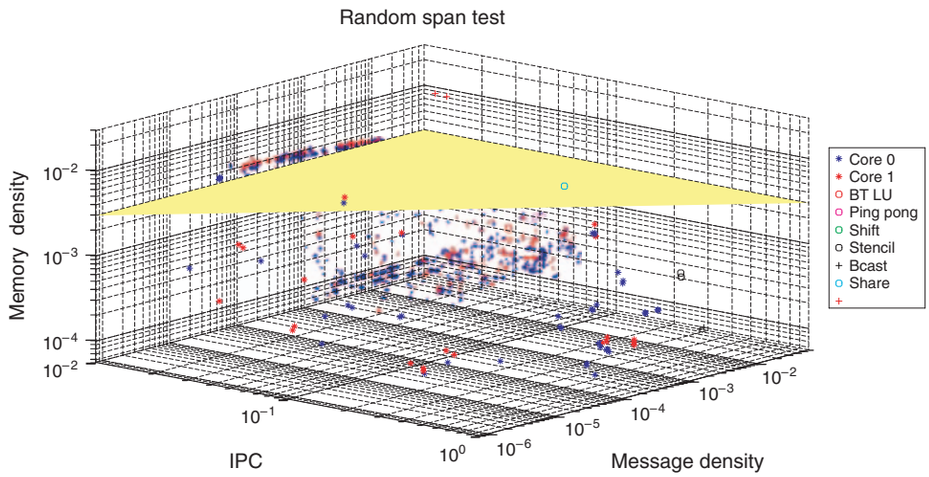

Fig. 4. PairMP-based dataset versus Intel many-core benchmarks—CPI, message density and memory access density—log space.

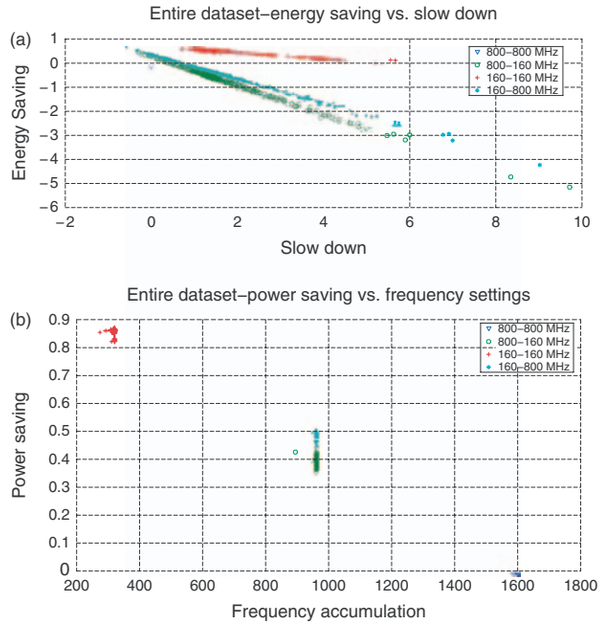

Fig. 5. Energy savings versus execution time slowdown/power consumption versus frequency.

execution time slowdown with respect to the case when both the cores have maximum frequency and x^k is the k-th application parameter. We use this dataset to perform a modeling exploration with the goal of identifying an execution time model that is capable of estimating the performance loss given the workload/application parameters and the frequency scaling factor for the communicating cores. We post-process a set of x_k application parameters/metrics from the original traces to be used as the input to the model learning phase. These parameters/metrics are:

- Cpi_{T}, Cpi_C are respectively the clock per instruction of the target core and the communicating core;
- Cpi_{AVG} is the average of the CPI computed in between the target core and the communicating core;
- Cpi_{DIFF} is the module of the difference in between the CPI of the target core and the communicating core;
- $MSG_{\mathrm{SZ}\times\mathrm{IST}}$ is total number of bytes received by the target core divided by the total number of instruction retired by the target core;
- $MEM_{\mathrm{ACC}\times\mathrm{IST}}$ is the total memory access over the total instruction retired by the target core;
- $MSG_{\mathrm{SENT}\times\mathrm{RCV}}$ is the ratio between the messages sent and the messages received by the target core;
- $Freq_{\mathrm{SF,T}}$, $Freq_{\mathrm{SF,C}}$ is respectively the frequency scaling factor for the target core and the communicating core.

We then select three different model templates, namely
(1) a linear model (LIN),
(2) an artificial neural network (ANN), and
(3) an analytical model (AM).

All the three models are in the form of $\hat{y} = f(\bar{x})$, where $\bar{x}$ is a set of the metrics defined above and the $\hat{y}$ is the predicted execution time slowdown of the target core (y).

The linear model is a linear combination of the input $\bar{x}$ parameter $\hat{y} = a_0 + \sum_{i=1}^{N} a_i * x_i$ where N is the number of input parameters. The coefficients a_i are computed by solving a linear least square problem. The artificial neural network is composed by one hidden layer, with a *tansig*, *sigmoid*, *tansig* activation functions respectively for the input layer, hidden layer and output layer. The input layer has $2N$ neurons, the hidden layer has N neurons, while the output layer has only one neuron. We split the dataset into a training and validation one that respectively are the 80% and 20% of the original dataset obtained by random sampling. We select the optimal set of input parameters by performing a feature selection pre-processing step in Matlab and use the relative error computed in the training dataset as the performance metric. We show the final input parameter generated as output of the feature selection step in Table II. The analytical model is chosen in between different model templates following the intuition that the slowdown of MP benchmark execution time will be a composition of the independent slowdown induced by the frequency scaling on the target core and the communicating core. The model template is described in Eq. (1)

$$\hat{y} = y_{\mathrm{T}} \cdot y_C \tag{1}$$

$$y_{\mathrm{T}} = 1 - \left(\frac{a_1}{CPI_{\mathrm{T}}}\right)^{a_2} - \left(\frac{a_1/Freq_{\mathrm{SF,T}}}{CPI_{\mathrm{T}}}\right)^{a_2} \tag{2}$$

$$y_C = 1 - \left(\frac{a_3}{CPI_C}\right)^{a_4} - \left(\frac{a_3/Freq_{\mathrm{SF,C}}}{CPI_C}\right)^{a_4} \tag{3}$$

Both the y_{T} and y_C are equal to one when no DVFS is applied ($Freq_{\mathrm{SF,T}}$, $Freq_{\mathrm{SF,C}} = 1$). Then it produces a slowdown inversely proportional to the CPI of the frequency scaled thread. We then compute the model parameters a_i by solving a non linear least-square problem in Matlab using the Levenberg-Marquardt algorithm.

Table II shows the model accuracy in predicting the execution time slowdown for the validation dataset. The model accuracy is evaluated as the average relative error in between the predicted execution time and the real one. Our first observation is that the non-linear models (ANN, AM) achieve a better fitting than the linear model (LIN). Even if the NN has the best accuracy, its relative error is significant, which is almost 15%. As we will discuss in next

Table II. Model fitting results on the validation dataset.

Model type	Average relative error (%)	Input metrics
Linear	30.5	$Freq_{\mathrm{SF,T}}$, $MEM_{\mathrm{ACC}\times\mathrm{IST}}$, Cpi_{AVG}, $Freq_{\mathrm{SF,C}}$
Analytic	26.7	$Freq_{\mathrm{SF,T}}$, Cpi_C, $Freq_{\mathrm{SF,C}}$, Cpi_{T}
Neural network	14.9	Cpi_C, Cpi_{AVG}, Cpi_{DIFF}, $MSG_{\mathrm{SZ}\times\mathrm{IST}}$, $MEM_{\mathrm{ACC}\times\mathrm{IST}}$, $MSGSENT \times RCV$, $Freq_{\mathrm{SF,T}}$, $Freq_{\mathrm{SF,C}}$

section, this low accuracy prevents the usage of predictive resource management policies. In addition, all the models depend on both the cores (i.e., target and communicating) frequency ($Freq_{SF,T}$, $Freq_{SF,C}$). This demonstrates that the communicating cores' operating points have a large impact on the final execution time. In the next section, we explain our proposed reactive MP-aware power management algorithm that takes advantage of this information to find the optimal energy frequency scaling for the active cores.

5. PROPOSED POWER MANAGEMENT POLICY FOR MANY-CORE SYSTEMS WITH MP

The previous section highlights the difficulties in gathering an accurate predictive model that estimates the energy savings and performance overhead given the characteristics of an MP parallel application and the cores' frequencies. This clearly limits the applicability of a predictive and model-based power management strategies to parallel benchmarks based on MP. On the other hand, application domains for which the same application is executed repetitively on the same device can take advantages from iterative power management strategies that tunes at every execution of the same application, by adjusting the frequency of each core with the goal of minimizing the energy consumption of the application, while ensuring target performance goal.[7] SCC HW can be exploited to evaluate the efficacy of feedback loops based on the direct measurement of power consumption and execution time of each application run. Based on these information, the frequencies of the cores are modulated to minimize the full system energy consumption.

A parallel application with N threads that is mapped on N cores placed in M frequency islands ($N \geq M$) can be configured to L different frequency levels. If we consider a fixed mapping, the search space has the dimension of $M \times L$. For parallel applications composed by independent threads, the total energy consumption decreases linearly with the energy consumption reduction of each single thread. In other words, the minimum energy can be found by independently scaling the frequency of each frequency island and applying the minimum voltage allowed in each voltage island. These properties do not hold for MP parallel applications, for which the energy consumption is shown to be a non-linear and non-convex function of the tiles voltage and frequency, as depicted in Section 4.3. Therefore, the minimum energy point cannot be found by scaling the cores individually, which lead to energy loss in most of the cases.

In this section, we present a novel MP-aware power management technique that extracts the communication map of the parallel benchmark and reduces the DVFS search space by forcing the communicating threads to scale the frequency simultaneously. We evaluate the benefits of this solution against the techniques that neglect the communication information.

If specific information on the application is not available, the simplest policy is scaling the frequency of the multicore, first individually in each tiles, then in larger groups of tiles with increasing dimensions. This basic policy (i.e., *baseline*) is presented in Section 5.1. Due to the large number of voltage and frequency islands in SCC, the dimensions of the search space is too large to be exhaustively searched by this algorithm in finite time and this may lead to sub-optimal solutions. A smarter approach is to randomly select the search direction. This policy is presented in Section 5.2 and we call it *random policy*. Finally, in Sections 5.3 and 5.4 we take advantage of the MP middleware to extract the message exchange map and using it to force the communicating cores to scale their frequency homogeneously. We present two different implementations for the message passing aware policy: (1) one considers the communicating cores to be the one with direct message exchange only (Section 5.4) and (2) the other that considers as communicating cores also the ones that are indirectly connected (Section 5.5).

Figure 6 depicts the general flow of the presented energy management policies highlighting their common parts.

- *Configuration*: Configuration module is composed of a set of configuration files that define the application to be executed and its running parameters (i.e., the number of threads), the mapping of the threads to the active cores and the initial frequency map (active tiles at f_{max} and idle tiles at f_{min}, where $f_{max} = 800$ MHz and $f_{min} = 160$ MHz).[d]
- *Initialization*: First the *freq/volt. loader* applies the frequency pattern defined in the config file.

To avoid voltage/frequency inconsistencies, at each frequency change first the voltage is set to the maximum level for each voltage island and then the frequencies are changed. Consequently for each voltage island, the voltage is lowered to support the maximum frequency of the tiles in the voltage island. Then, the *app. loader* launches the application specified in the config files. When finished the *data collector* accesses the logging traces of each active core (i.e., the power traces, the CPI traces, execution time data, number of messages sent and received) and computes different metrics according to the policy requirements (i.e., energy consumption, EDP, MP communication matrix, application execution time). These metrics are then saved as the *reference* one and will be used later by the policy to make decisions.

- *Policy initialization*: The policy initialization, as will be described later for the different policies, selects the first group of tiles to start scaling the frequency and creates a *new frequency* config file accordingly.
- *Main loop*: This loop executes until the number of iteration is below the maximum value (*IMAX*).

Internally the loop first applies, by means of the *freq/volt. loader*, the *new frequency* configuration then in

[d] Active tiles/cores execute an application thread and the idle tile/cores are on but do not run any application.

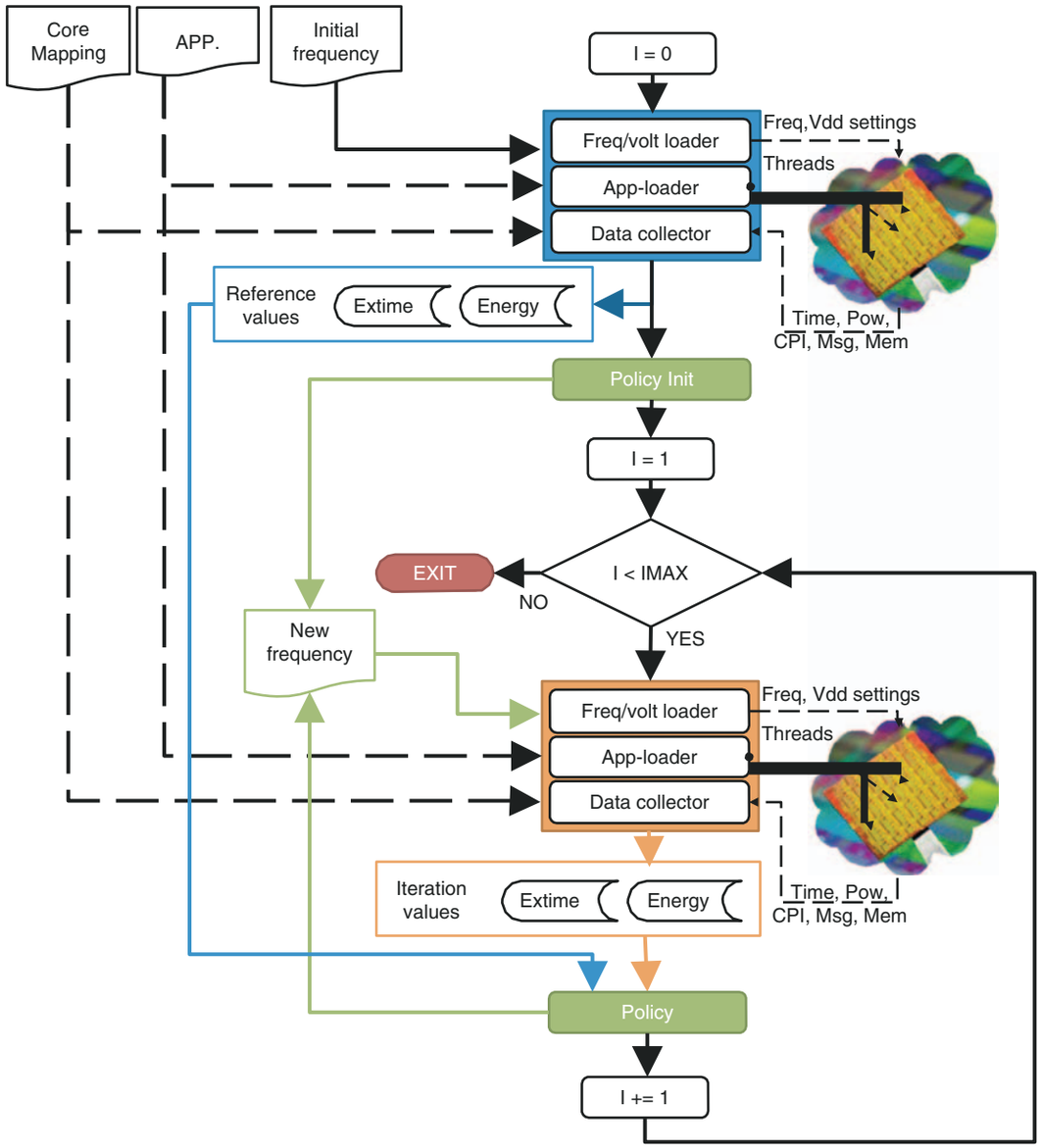

Fig. 6. Flow graph of the proposed energy management policies.

sequence executes the application through the *app. loader*, collects the results with the *data collector* and computes the new performance metrics (i.e., application execution time and energy consumption) for the current iteration I. These current metrics are then used by the *policy* together with the *reference* one to decide the *new frequency* configuration for the next iteration. Once the *IMAX* iterations is reached, the best run in terms of energy savings is selected for the following re-execution of the application.

5.1. Baseline Policy

If no information on the internal synchronization and communication of a parallel application is available, the simplest policy is to sequentially scale the frequency of an increasing set of active tiles. As the number of core increases, the effectiveness of the policy is reduced by the dimension of the search space. Figure 7 depicts the working principles of the *policy* by describing the initialization phase (policy_init) and the main loop (*policy*) components of the policy.

During the initialization phase, as depicted in Figure 7, the iteration counter I is set to zero and the policy computes the reference energy $En(0)$ and stores its value. Then the list of active cores is extracted and stored as a list of active tiles (A_{TILE}) from the *core mapping* file.[e]

[e]Frequency can be scaled only at tile's granularity on our experimental platform.

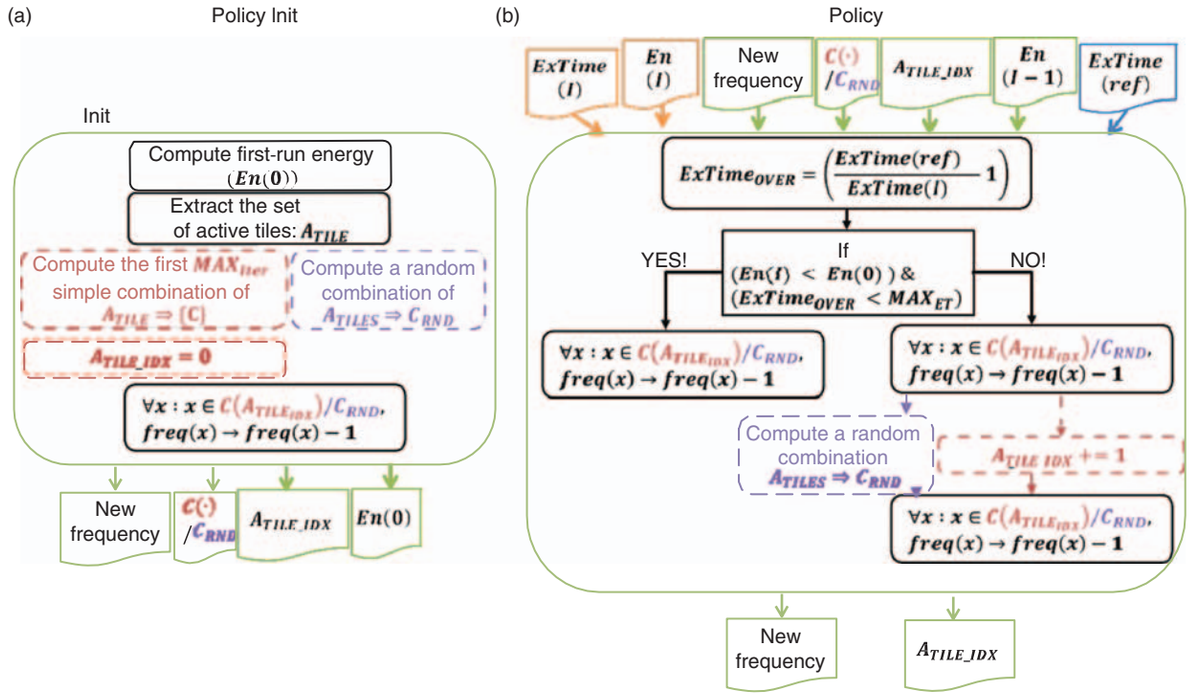

Fig. 7. Baseline and random flow graph.

The baseline policy (i.e., dashed red line in Fig. 7) uses this information to compute a list of simple combination of the A_{TILE} elements with an increasing number of elements ($C(\cdot)$). The number of combination tuples is equal to the number of maximum iteration specified in the policy (*IMAX*). The initialization phase will then select the first combination present in C and for each tile present in it will scale down the frequency of one step. This is implemented by writing the new frequency values in the *new frequency* config file. The policy will use an internal index A_{TILEIDX} to refer to the last combination used. The initialization policy will then store as output the combination set $C(\cdot)$, the last combination index A_{TILEIDX}, the energy consumption of the reference execution $En(0)$ and the *new frequency* configuration file.

As shown in Figure 7(b), inside the main loop during the I-th iteration, the policy receives the application execution time for the current iteration ($ExTime(I)$) and for the reference one ($ExTime(ref)$), the *new frequency*, the combination set ($C(\cdot)$), last combination index A_{TILEIDX} and the energy consumption of the previous execution $En(I-1)$ as inputs. Starting from the application execution time the policy computes the performance overhead $ExTime_{\mathrm{OVER}}$ with respect to the reference application execution time. This value measures the slowdown of the application due to the frequency scaling configuration that has been applied during the last iteration (I). The slowdown is compared with respect to the maximum frequency run. If this value is lower than a maximum tolerated slowdown for the application execution time ($\mathrm{MAX_{ET}}$) and there has been an energy saving w.r.t. previous iteration $En(I) < En(I-1)$, the frequency of the tiles present in the previous combination tuple ($C(A_{\mathrm{TILEIDX}})$) are scaled down by another step. The new frequency configuration is saved as a *new frequency* configuration for the next iteration.

If the latest frequency configuration have produced an execution time overhead ($ExTime_{\mathrm{OVER}}$) that is bigger than the maximum allowed one ($\mathrm{MAX_{ET}}$) or an energy loss, the policy will roll back the frequency configuration by increasing the frequency of the last combination tuple ($C(A_{\mathrm{TILEIDX}})$), increasing the last combination index A_{TILEIDX} and select a new tile combination tuple $C(A_{\mathrm{TILEIDX}})$. Then for each tiles in the tuple, the frequency is decreased by one step and the new values are updated in the *new frequency* configuration file.

5.2. Random Policy

As previously introduced, while the number of active tiles increases the efficacy of the baseline policy to explore the optimization search space decreases, as it moves linearly in between all the possible combinations. To have a better comparison for our MP-aware proposed solution, we implement a second baseline policy that randomly generates the active tile combination tuple to scale their frequency. The main building blocks of this policy are reported in Figure 7 with the purple dashed line. As Figure 7 shows, the main difference with the previous policy is in the initialization part, during which the combination of the active tiles are randomly computed C_{RND}.

The random combination is composed by a random number of elements, each one containing a random active tile index. Then only unique values are used. These values are then stored for future use. This computation is repeated along the *false* path of the policy main loop, when after the previous frequency configuration is restored the policy computes a new random tile combination C_{RND} and scale down the frequency for each tile present in the set.

5.3. Proposed MP-Aware Policy

The two baseline policies presented in the previous subsections neglect the parallel nature of the MP applications and the interdependency of the threads generated by its communication exchange as described in Section 4.3. For this reason, we present a novel policy that exploits the communication patterns during the frequency selection. Figure 6 describes the main components of our policy.

Figure 8(a) shows that during the initialization phase the MP-aware policy computes the reference energy $En(0)$, and the list of active tiles (A_{TILE}). From each message exchange log of each active tile, the policy computes the communication matrix M_{MSG} in which each elements $m_{i,j}$ contains the number of messages sent by the core i to the core j. The policy then initializes the internal index A_{TILEIDX} to point to the last active tile selected by the policy. The policy initialization phase concludes by accessing the first row of the communication matrix M_{MSG} and generating the new frequency configuration by scaling down of one step the frequency of tiles corresponding to each non zero element in the selected row M_{MSG} ($i = A_{\text{TILEIDX}}$, $j = x$). In the main loop, at the I-th iteration the MP-aware policy as shown in Figure 8(b) uses the precomputed M_{MSG} matrix and the current active tile index A_{TILEIDX} to update the frequency accordingly to the condition on the energy consumption $En(I) < En(I1)$ and execution time overhead $ExTime_{\text{OVER}} < \text{MAX}_{\text{ET}}$ as described in Section 5.1.

5.4. Proposed Fully Connected MP-Aware Policy (MP-FC)

The presented MP-aware policy considers only the direct communication relation and neglects the interference of threads that are indirectly communicating. Indeed it may be the case that $core_A$ sends messages to $core_B$ and $core_B$ sends messages to $core_C$. We call the communication in between the $core_A$ and $core_C$ indirect. The presented MP-aware policy simultaneously scales the frequency of $cores_{A,B}$ and in a different iteration the frequency of $cores_{B,C}$. This is sub-optimal as the $core_C$ is indirectly influenced by the $core_A$ frequency scaling. To account for this situation, we propose a variant of the MP-aware policy called fully connected (*MP-FC policy*) that takes advantage of the properties of the adjacency matrices to derive from the connection matrix the fully connected tiles. Indeed, the i-th power of M^i_{MSG} represent the i-th indirect level of communication. By summing the first A_{TILE} powers of the communicating matrix, we can derive a new communicating matrix that has the $m_{i,j}$ element different from zero where there is a direct or indirect link in

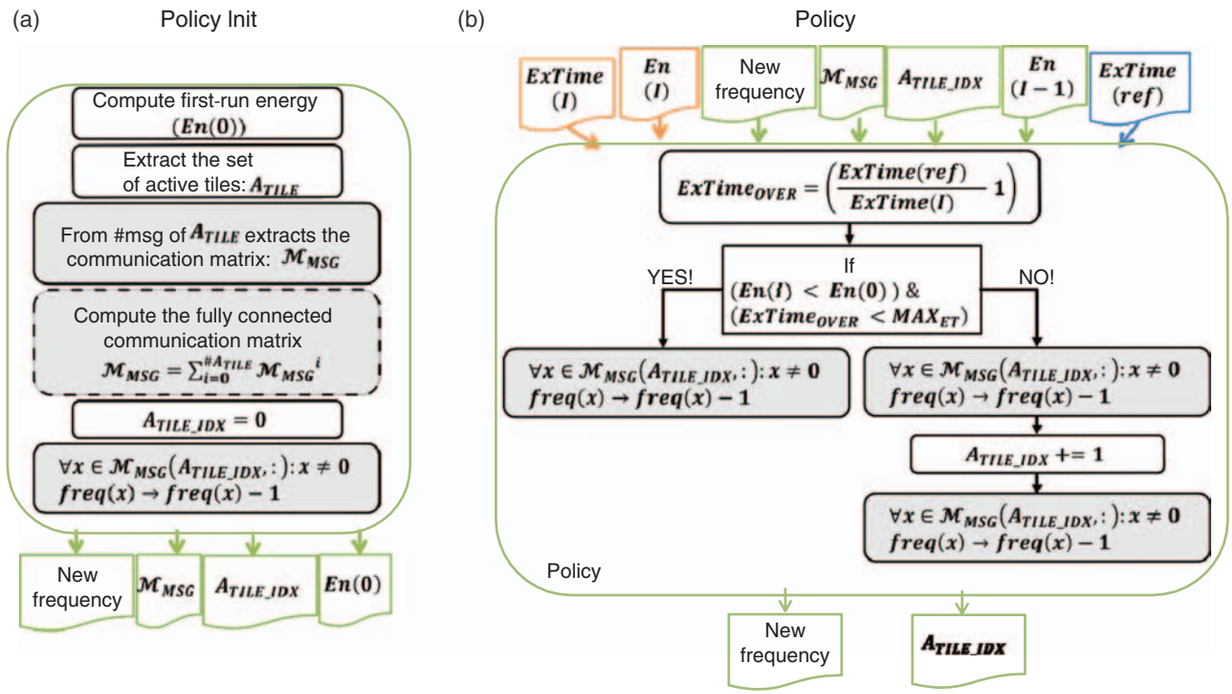

Fig. 8. MP-aware and fully connected MP-aware policy flow graph.

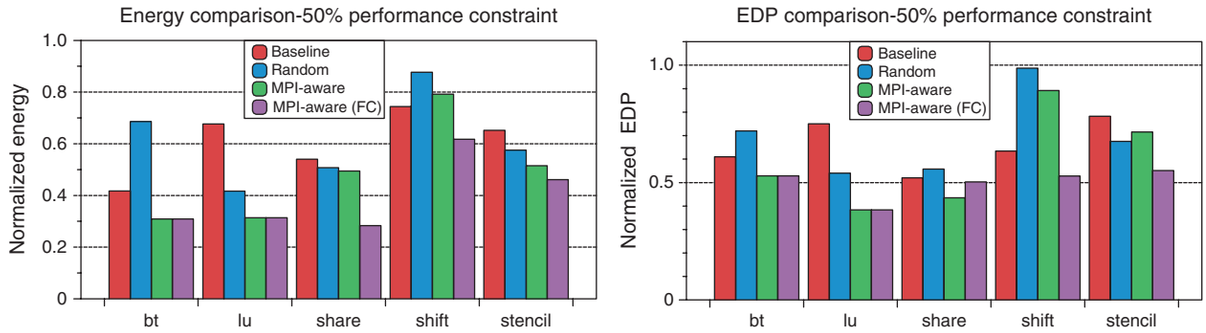

Fig. 9. Energy and EDP comparison of various policies with 50% maximum performance degradation constraints, running on 4 cores on different voltage and frequency island.

between the *i*-th and *j*-th core. This step is described with a dashed line in Figure 8(a). The rest of the policy behaves identical to the MP-aware policy.

6. EXPERIMENTAL EVALUATION

In this section, we evaluate the proposed MP-aware techniques (i.e., MP-aware and MP-aware fully connected (MP-FC)) against the baseline and random DVFS policies that are explained in Section 5. We run all the experiments on the Intel SCC platform and compare the energy and energy-delay products (EDP) of various policies for all the Intel many-core benchmarks presented in Section 4, namely BT, LU, share, shift, stencil. We iterate the policy for a maximum of ten iterations *IMAX* and the result shows the average of the different performance metrics on these ten iterations.

First, we evaluate the policies under various performance constraints (i.e., maximum performance degradation-MAX_{ET}—of 20% and 50%). Second set of experiments evaluate the policies, when threads are mapped onto the cores that are on the same voltage island. Finally, we evaluate our technique for a higher level of parallelism, in which we run the workloads with 16 threads. In all experiments, we normalize the energy and EDP with respect to the reference case. In this reference case, we run the workloads at the maximum frequency setting, while keeping the inactive cores at the minimum one. We compare all cases with respect to a baseline experiment, in which we run the applications with 4 threads running on different voltage islands under 50% performance constraint.

6.1. Reference Case

Figure 9 compares the policy for the reference case, where all the applications are parallelized on 4 threads and allocated to different voltage islands. We report the average energy savings (left plot) and the EDP savings (right plot) for each application and policy. As Figure 9 shows, all the policies lead to an energy and EDP saving and the degree of savings depend on the application characteristics. The baseline and the random policies behave similarly due to the fact that the reduced number of active tiles does not stress the scalability limits of the baseline policy. Same observation applies to the MP-aware policy, which outperforms the baseline and random policy and achieves up to 10% energy and EDP savings on average. Furthermore, MP-aware policy and the fully connected (MP-FC) policy exhibit similar behavior for BT and LU. This behavior can be explained by the fact that all the threads are directly connected in BT and LU. However, this is not the case for share, shift and stencil. Figure 9 also shows that the purely MP-aware policy is less effective then the baseline policy in some cases.

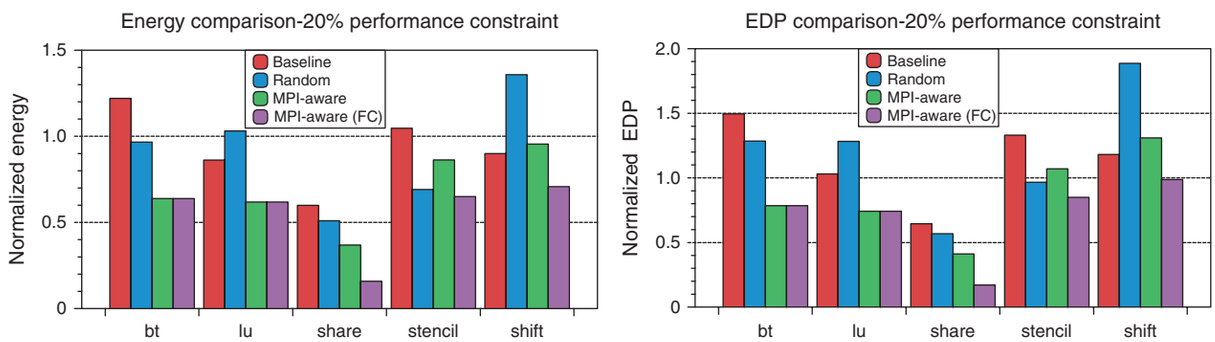

Fig. 10. Energy and EDP comparison of various policies with 20% maximum performance degradation constraints.

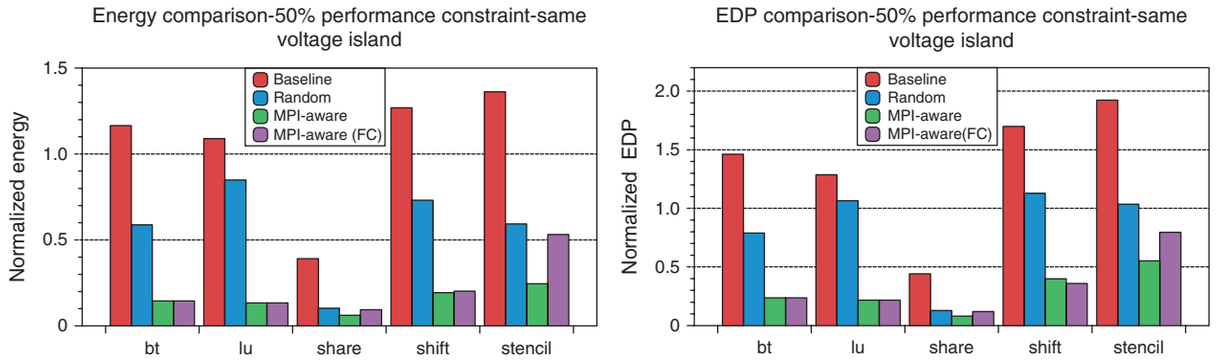

Fig. 11. Energy and EDP comparison of various policies with 50% maximum performance degradation constraints, when threads are scheduled on the same voltage island.

6.2. Low Performance Constraints

As the energy savings and EDP improvements vary under performance constraints, we evaluate the policies under a lower performance constraints. We evaluate the maximum performance degradation (MAX_{ET}) of 20% in Figure 10. When comparing them with the results of (MAX_{ET}) of 50% (see Fig. 9), we notice that all the policies lead to lower energy and EDP savings under lower performance constraints.

For both the baseline and the random policies, the average energy is increased up to 1.2× and 1.4× for BT and stencil respectively. This energy increase is due to the fact that these policies provide intermediate voltage and frequency patterns, which lead an increase of overall energy consumption that is not compensated by the energy savings achieved by the optimum configurations. In terms of EDP, the baseline and random policy provides improvement only for share that is memory bound. For the other applications, these policies lead almost up to 2x performance degradation for stencil, which is a CPU bound application. For the same performance constraint, the proposed MP-aware policies consistently lead to energy savings. EDP plot also shows that the pure MP-aware policy does not bring any benefits for the applications that exhibit complex commutation patterns (i.e., share, shift, stencil), therefore increase the EDP for shift and stencil, which are not memory bound. On the other hand, the fully connected MP-aware policy outperforms the pure MP-aware policy and consistently leads to increased energy and EDP savings. Energy consumption and the EDP is reduced by 20% for share, whereas energy is reduced by 30% and EDP by 10% for stencil.

6.3. Mapping on the Same Voltage Island

Mapping the threads on separate voltage islands enables us to set v–f independent pairs for each individual thread. However, in a real-life scenario, there might be constraints that prevent mapping the threads on separate voltage islands, such as security or availability of the cores. Furthermore, the underlying hardware might not support changing the frequency and voltage settings of the individual cores. Therefore, in the following experiments, we evaluate our technique on a configuration, where we schedule the threads on the same voltage islands.

Figure 11 shows the experiments, for which the four threads are scheduled on the same voltage island and on two tiles/frequency island. As Figure 11 shows, the baseline policy does not provide any energy or EDP savings. This is due to the fact that the frequency scaling on a single tile prevents the voltage island to run at lower voltages. Furthermore, the random policy causes performance loss. On the contrary, both of the MP-aware policies exhibit

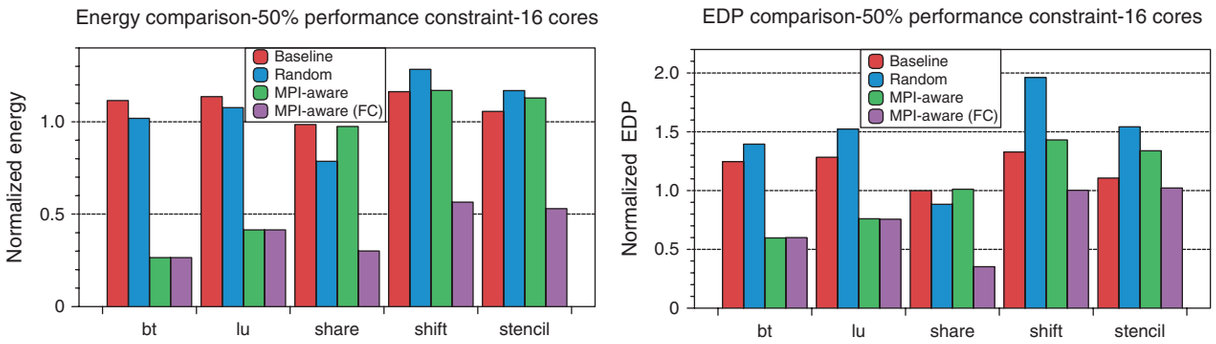

Fig. 12. Energy and EDP comparison of various policies with 50% maximum performance degradation constraints, running on 16 cores.

benefits from this mapping and provide similar energy and EDP benefits. This is due to the fact that the each tile have more direct connections by grouping all the cores in a lower number of tiles. This leads to similar trends for both the energy and EDP savings. In this case, the average energy consumption for both the MP-aware policies and for all the applications is decreased by 25% and the EDP saving is higher than 50% on average.

6.4. Higher Levels of Parallelism

In these set of experiments, we evaluate the impact of the proposed techniques for a higher level of parallelism, where we run the workloads with 16 threads running on 16 cores. For higher number of threads, baseline and random policy perform worse than the reference case, except for share that is memory bound. As baseline and random policies scale down the frequency setting of each core separately, reaching to the minimum energy/EDP for higher number of cores requires significant amount of iterations to converge to a minimum energy/EDP setting. However, MP-awareness allows us to reach to a minimum energy/EDP point much faster by scaling down the frequency settings simultaneously for the cores that are communicating. The fully connected MP-aware policy provides up to 65% lower energy for the benchmarks that have indirect communications. For these cases, the pure MP-aware policy does not always lead to energy and EDP savings, whereas the MP-aware (FC) consistently provides energy savings (between 50% and 70%) and significantly reduces the EDP for the applications that are not CPU bound and provides up to 30% EDP savings for strongly memory bound applications.

7. CONCLUSION

In this paper, we explored the dynamic voltage and frequency scaling implications on a many-core system running MP applications using blocking communication. We first analyzed the performance and main characteristics of MP applications, when subject to a voltage and frequency changes. We demonstrate that the energy savings through DVFS is significantly higher for MP applications, when the communicating cores are scaled simultaneously, which implies that designing communication-aware policies is essential for achieving energy efficiency on many-core systems. We show that the achievable energy savings at a given DFVS setting cannot be easily modeled with standard learning strategies, which prevents the use of advanced predictive power management techniques. To overcome this limitation, we present novel MP-aware power management policies that extract the communications patterns and use this information to guide the voltage and frequency scaling decisions. We show that the MP-aware policies consistently lead to energy-delay-product reductions while achieving the same performance, when compared to power management strategies that are not communication-aware. When threads are grouped in the same voltage island, MP-awareness leads to an average EDP savings of 50% for lower levels of parallelism (4-thread execution), and up to 70% EDP savings for applications with higher levels parallelism (16-thread execution).

Acknowledgments: This work was supported, in parts, by the EU FP7 ERC Project MULTITHERMAN (GA n. 291125) and the EU FP7 Project Phidias (GA n. 318013).

References

1. K. Flautner and T. Mudge, Vertigo: Automatic performance-setting for linux, *Proceedings of the 5th symposium on Operating systems design and implementation, OSDI'02*, ACM, New York, NY, USA **(2002)**, pp. 105–116.
2. K. Choi, R. Soma, and M. Pedram, Dynamic voltage and frequency scaling based on workload decomposition, *Proceedings of the 2004 International Symposium on Low Power Electronics and Design, ISLPED'04*, ACM, New York, NY, USA **(2004)**, pp. 174–179.
3. C. Isci, G. Contreras, and M. Martonosi, Live, runtime phase monitoring and prediction on real systems with application to dynamic power management, *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 39*, IEEE Computer Society, Washington, DC, USA **(2006)**, pp. 359–370.
4. G. Dhiman and T. S. Rosing, Dynamic voltage frequency scaling for multi-tasking systems using online learning, *Proceedings of the 2007 International Symposium on Low Power Electronics and Design, ISLPED'07*, ACM, New York, NY, USA **(2007)**, pp. 207–212.
5. H. Jung and M. Pedram, Supervised learning based power management for multicore processors. *Trans. Comp.-Aided Des. Integ. Cir. Sys.* 29, 1395 **(2010)**.
6. H. Lu, S. Dwarkadas, A. L. Cox, and W. Zwaenepoel, Message passing versus distributed shared memory on networks of workstations, *Proceedings of the IEEE/ACM SC95 Conference, Supercomputing 1995*, IEEE **(1995)**, p. 37.
7. R. Springer, D. K. Lowenthal, B. Rountree, and V. W. Freeh, Minimizing execution time in mpi programs on an energy- constrained, power-scalable cluster, *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP'06*, ACM, New York, NY, USA **(2006)**, pp. 230–238.
8. B. Rountree, D. Lowenthal, S. Funk, V. W. Freeh, B. De Supinski, and M. Schulz, Bounding energy consumption in large-scale mpi programs, *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing 2007, SC'07* **(2007)**, pp. 1–9.
9. J. Howard, S. Dighe, S. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, and R. Van Der Wijngaart, A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling. *IEEE Journal of Solid-State Circuits* 46, 173 **(2011)**.
10. J. Choi, Y. Kim, A. Sivasubramaniam, J. Srebric, Q. Wang, and J. Lee, Modeling and managing thermal profiles of rack- mounted servers with thermostat, *IEEE 13th International Symposium on High Performance Computer Architecture 2007, HPCA 2007* **(2007)**, pp. 205–215.
11. A. Bartolini, M. Sadri, J. Furst, A. Coskun, and L. Benini, Quantifying the impact of frequency scaling on the energy efficiency of the single-chip cloud computer, *Design, Automation Test in Europe Conference Exhibition (DATE), 2012* **(2012)**, pp. 181–186.
12. 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, et al., The nas parallel benchmarks. *International Journal of High Performance Computing Applications* 5, 63 **(1991)**.

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

14. 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 **(2011)**.

15. A. Mishra, S. Srikantaiah, M. Kandemir, and C. Das, Cpm in cmps: Coordinated power management in chip-multiprocessors, *2010 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* **(2010)**, pp. 1–12.

16. G. Keramidas, V. Spiliopoulos, and S. Kaxiras, Interval-based models for run-time dvfs orchestration in superscalar processors, *Proceedings of the 7th ACM International Conference on Computing Frontiers, CF'10*, ACM, New York, NY, USA **(2010)**, pp. 287–296.

17. A. Bhattacharjee and M. Martonosi, Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors. *SIGARCH Comput. Archit. News* 37, 290 **(2009)**.

18. A. Alameldeen and D. Wood, Ipc considered harmful for multiprocessor workloads. *Micro, IEEE* 26, 8 **(2006)**.

19. D. Li, D. Nikolopoulos, K. Cameron, B. De Supinski, and M. Schulz, Power-aware mpi task aggregation prediction for high-end computing systems, *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)* **(2010)**, pp. 1–12.

20. N. Kappiah, V. W. Freeh, and D. Lowenthal, Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs, *Proceedings of the ACM/IEEE SC 2005 Conference Supercomputing 2005* **(2005)**, p. 33.

21. 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 **(2011)**.

22. P. Salihundam, S. Jain, T. Jacob, S. Kumar, V. Erraguntla, Y. Hoskote, S. Vangal, G. Ruhl, and N. Borkar, A 2 tbs 6 4 mesh network for a single-chip cloud computer with dvfs in 45 nm cmos. *IEEE Journal of Solid-State Circuits* 46, 757 **(2011)**.

23. N. Ioannou, M. Kauschke, M. Gries, and M. Cintra, Phase-based application-driven hierarchical power management on the single-chip cloud computer, *2011 International Conference on Parallel Architectures and Compilation Techniques (PACT)* **(2011)**, pp. 131–142.

24. M. Gamell, I. Rodero, M. Parashar, and R. Muralidhar, Exploring cross-layer power management for pgas applications on the scc platform, *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing, HPDC'12*, ACM, New York, NY, USA **(2012)**, pp. 235–246.

25. R. David, P. Bogdan, and R. Marculescu, Dynamic power management for multicores: Case study using the intel scc, *2012 IEEE/IFIP 20th International Conference on VLSI and System-on-Chip (VLSI-SoC)* **(2012)**, pp. 147–152.

26. D. Li, B. R. de Supinski, M. Schulz, K. Cameron, and D. S. Nikolopoulos, Hybrid mpi/openmp power-aware computing, *2010 IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, IEEE **(2010)**, pp. 1–12.

27. X. Chen, Z. Xu, H. Kim, P. Gratz, J. Hu, M. Kishinevsky, and U. Ogras, In-network monitoring and control policy for dvfs of cmp networks-on-chip and last level caches. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 18, 47:1 **(2013)**.

28. P. Bogdan, R. Marculescu, and S. Jain, Dynamic power management for multidomain system-on-chip platforms: An optimal control approach. *ACM TODAES* 18, 46:1 **(2013)**.

29. I. A. Ureña, M. Riepen, M. Konow, and M. Gerndt, Invasive mpi on intel's single-chip cloud computer, *Proceedings of the 25th International Conference on Architecture of Computing Systems*, Springer-Verlag **(2012)**, pp. 74–85.

30. J. Demme and S. Sethumadhavan, Rapid identification of architectural bottlenecks via precise event counting, *Proceedings of the 38th Annual International Symposium on Computer Architecture, ISCA'11* **(2011)**, pp. 353–364.

Andrea Bartolini

Andrea Bartolini *received a Ph.D. degree in Electrical Engineering from the University of Bologna, Italy, in 2011. He is currently a postdoc researcher in the Department of Electrical, Electronic and Information Engineering Guglielmo Marconi (DEI) at the University of Bologna. He also holds a postdoc position in the Integrated Systems Laboratory at ETH Zurich. His research interests concern dynamic resource management of embedded systems and multi-core systems with special emphasis on software-level thermal and power-aware techniques. His research interest also includes ultra-low power design strategies for bio-sensors nodes operating in near-threshold.*

Can Hankendi

Can Hankendi *received his M.S. degree in Electrical Engineering from University of Southern California, Los Angeles, in 2010. He is currently a Ph.D. candidate in Electrical and Computer Engineering Department at Boston University. His research interests are resource and power management techniques on multi-core servers specializing in adaptive resource and power management techniques for multi-threaded workloads.*

Ayse Kivilcim Coskun

Ayse Kivilcim Coskun *(M06) received the M.S. and Ph.D. degrees in Computer Science and Engineering from the University of California, San Diego. She is currently an Assistant Professor at the Department of Electrical and Computer Engineering, Boston University (BU), Boston, MA. She was with Sun Microsystems (now Oracle), San Diego, prior to her current position at BU. Her current research interests include energy-efficient computing, multicore systems, 3-D stack architectures, computer architecture, and embedded systems and software. Dr. Coskun received the Best Paper Award at IFIP/IEEE VLSI-SoC Conference in 2009 and in HPEC Workshop in 2011. She currently serves on the program committees of many design automation conferences including DAC, DATE, GLSVLSI, and VLSI-SoC. She has served as a guest editor in ACM TODAES journal and currently is an associate editor of IEEE Embedded Systems Letters. She is a member of the IEEE and ACM.*

Luca Benini

Luca Benini *is Full Professor at the Department of Electrical Engineering and Computer Science (DEIS) of the University of Bologna. He also holds a visiting faculty position at the Ecole Polytechnique Federale de Lausanne (EPFL) and he is currently serving as Chief Architect for the Platform 2012 project in STmicroelectronics, Grenoble. He received a Ph.D. degree in electrical engineering from Stanford University in 1997. Dr. Benini's research interests are in energy-efficient system design and Multi-Core SoC design. He is also active in the area of energy-efficient smart sensors and sensor networks for biomedical and ambient intelligence applications. He has published more than 600 papers in peer-reviewed international journals and conferences, four books and several book chapters. He is a member of the Academia Europaea.*