High-Performance Energy-Efficient Multi-core Embedded Computing

Arslan Munir, *Student Member, IEEE,* Sanjay Ranka, *Fellow, IEEE, AAAS,* and Ann Gordon-Ross, *Member, IEEE*

Abstract—With Moore's law supplying billions of transistors on-chip, embedded systems are undergoing a transition from single-core to multi-core to exploit this high transistor density for high performance. Embedded systems differ from traditional high-performance supercomputers in that power is a first-order constraint for embedded systems whereas performance is the major benchmark for supercomputers. The increase in on-chip transistor density exacerbates power/thermal issues in embedded systems, which necessitates novel hardware/software power/thermal management techniques to meet the ever-increasing high-performance embedded computing demands in an energy-efficient manner. This paper outlines typical requirements of embedded applications and discusses state-of-the-art hardware/software high-performance energy-efficient embedded computing (HPEEC) techniques that help meeting these requirements. We also discuss modern multi-core processors that leverage these HPEEC techniques to deliver high performance per watt. Finally, we present design challenges and future research directions for HPEEC system development.

Index Terms—High-performance computing (HPC), multi-core, energy-efficient computing, green computing, low-power, embedded systems

1 Introduction

EMBEDDED system design is traditionally power-centric but there has been a recent shift towards high-performance embedded computing (HPEC) due to the proliferation of compute-intensive embedded applications. For example, the signal processing for a 3G mobile handset requires 35-40 Giga operations per second (GOPS) for a 14.4 Mbps channel and 210-290 GOPS for a 100 Mbps orthogonal frequency-division multiplexing (OFDM) channel. Considering the limited energy of a mobile handset battery, these performance levels must be met with a power dissipation budget of approximately 1W, which translates to a performance efficiency of 25 mW/GOP or 25 pJ/operation for the 3G receiver and 3-5 pJ/operation for the OFDM receiver [1][2]. These demanding and competing power-performance requirements make modern embedded system design challenging.

The high-performance energy-efficient embedded computing (HPEEC) domain addresses the unique design challenges of high-performance and low-power/energy (can be termed as *green*, however, green may refer to a bigger notion of environmental impact) embedded computing. These design challenges are competing because high-performance typically requires maximum processor speeds with enormous energy consumption, whereas low-power typically requires nominal or low processor speeds that offer modest performance. HPEEC requires thorough consideration of the thermal design power (TDP) and processor frequency relationship while selecting an appropriate processor for an embedded application. For example, decreasing the processor frequency by a fraction of the maximum operating frequency (e.g., reducing from 3.16 GHz to 3.0 GHz) can cause 10% performance degradation but can decrease power consumption by 30-40% [3]. To meet HPEEC power-performance requirements, embedded system design has transitioned from a single-core to a multi-core paradigm that favors multiple low-power cores running at low processors speeds rather than a single high-speed power-hungry core.

Chip multiprocessors (CMPs) provide a scalable HPEEC platform as performance can be increased by increasing the number of cores as long as the increase in the number of cores offsets the clock frequency reduction by maintaining a given performance level with less power [4]. Multi-processor systems-on-chip (MPSoCs), which are multi-processor version of systems-on-chip (SoCs), are another alternative HPEEC platform, which provide an unlimited combination of homogeneous and heterogeneous cores. Though both CMPs and MPSoCs are HPEEC platforms, MPSoCs differ from CMPs in that MPSoCs provide custom architectures (including specialized instruction sets) tailored for meeting peculiar requirements of specific embedded applications (e.g., real-time, throughput-intensive, reliability-constrained). Both CMPs and MPSoCs rely on HPEEC hardware/software techniques for delivering high performance per watt and meeting diverse application requirements.

- *Arslan Munir and Ann Gordon-Ross are with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, 32611 USA. Ann Gordon-Ross is also affiliated with the NSF Center for High-Performance Reconfigurable Computing (CHREC) at the University of Florida. Sanjay Ranka is with the Department of Computer and Information Science and Engineering at the University of Florida. e-mail: {amunir@ufl.edu, ranka@cise.ufl.edu, ann@ece.ufl.edu}*

TABLE 1: Top Green500 and Top500 supercomputers as of June 2011 [5][6]

Supercomputer	Green500 Rank	Top500 Rank	Cores	Power Efficiency	Peak Performance	Peak Power
Blue Gene/Q Prototype 2	1	109	8192	2097.19 MFLOPS/W	104857.6 GFLOPS	40.95 kW
Blue Gene/Q Prototype 1	2	165	8192	1684.20 MFLOPS/W	104857.6 GFLOPS	38.80 kW
DEGIMA Cluster, Intel i5	3	430	7920	1375.88 MFLOPS/W	111150 GFLOPS	34.24 kW
TSUBAME 2.0	4	5	73278	958.35 MFLOPS/W	2287630 GFLOPS	1243.80 kW
iDataPlex DX360M3, Xeon 2.4	5	54	3072	891.88 MFLOPS/W	293274 GFLOPS	160.00 kW

Even though literature discusses high-performance computing (HPC) for supercomputers [7][8][9][10], there exists little discussion on HPEEC [11]. The distinction between HPC for supercomputers and HPEEC is important because performance is the most significant metric for supercomputers with less emphasis given to energy-efficiency, whereas energy-efficiency is a primary concern for HPEEC. For example, each of the 10 most powerful contemporary supercomputers has a peak power requirement of up to 10 MW, which is equivalent to the power needs of a city with a population of 40,000 [5][11]. To acknowledge the increasing significance of energy-efficient computing, the Green500 list ranks supercomputers using the FLOPS per watt performance metric [6]. Table 1 lists the top 5 green supercomputers along with their top 500 supercomputer ranking. The table shows that the top performing supercomputers are not necessarily energy-efficient [5][6]. Table 1 indicates that most of the top green supercomputers consist of low-power embedded processor clusters aiming at achieving high performance per watt and high performance per unit area [12].

Fig. 1 gives an overview of the HPEEC domain, which spans architectural approaches to middleware and software approaches. In this paper we focus on high-performance and energy-efficient techniques that are applicable to embedded systems (CMPs, SoCs, or MPSoCs) to meet particular application requirements. Although the main focus of the paper is on embedded systems, many of the energy and performance issues are equally applicable to supercomputers since state-of-the-art supercomputers leverage embedded processors/chips (e.g., Jaguar supercomputer comprising of 224162 processor cores leverages AMD Opteron six-core CMPs [5]). However, we summarize several differences between supercomputing applications and embedded applications as follows:

1) Supercomputing applications tend to be highly data parallel where the goal is to decompose a task with a large dataset across many processing units where each subtask operates on a portion of the dataset. On the other hand, embedded applications tend to consist of many tasks where each task is executed on a single processing unit and may have arrival and deadline constraints.
2) Supercomputing applications tend to focus on leveraging a large number of processors whereas the scale of embedded applications is generally much smaller.
3) Supercomputing applications' main optimization objective is performance (although energy is increasingly becoming a very important secondary metric), while performance and energy are equally important objectives for embedded applications. Also, reliability and fault tolerance play a more important role in embedded applications as compared to supercomputing applications.

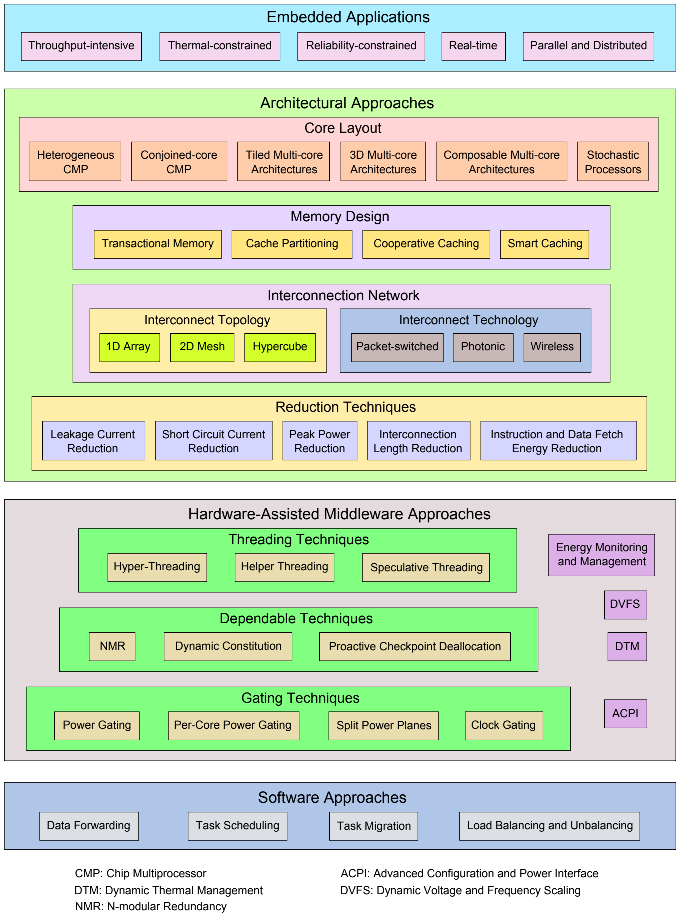

Fig. 1: High-performance energy-efficient embedded computing (HPEEC) domain.

The HPEEC domain benefits from architectural innovations in processor core layouts (e.g., heterogeneous CMP, tiled multi-core architectures), memory design (e.g., transactional memory, cache partitioning), and interconnection networks (e.g., packet-switched, photonic, wireless). The HPEEC platforms provide hardware support for functionalities that can be controlled by middleware such as dynamic voltage and frequency scaling (DVFS), hyper-threading, helper threading, energy monitoring and management, dynamic thermal management (DTM), and various power-gating techniques. The HPEEC domain benefits from software approaches such as task scheduling, task migration, and load balancing. Many of the HPEEC techniques at different levels (e.g., architectural, middleware, and software) are complementary in nature and work in conjunction with one another to better meet application requirements. To the best of our knowledge, this is the first paper targeting HPEEC that provides a comprehensive classification of various HPEEC techniques in relation to meeting diverse embedded application requirements.

2 Embedded Applications

The proliferation of embedded systems in various domains (e.g., consumer electronics, automotive, industrial automation, networking, medical, defense, space, etc.) due to technological advancements has given rise to a plethora of embedded applications. Thus, embedded systems require HPEEC hardware/software techniques to meet the ever increasing processing demands of the embedded applications. Since economic pressures have a large influence on embedded system development, many embedded applications require embedded systems to be reliable and robust, easy to use, able to connected with other devices, and low cost. Since many embedded application requirements are competing, tradeoffs must be made between these requirements, such as size versus flexibility, robustness versus richness of functionality, and power consumption versus performance. Therefore, embedded system vendors market domain-specific platforms that are specialized for a particular domain and offer appropriate tradeoffs to better meet that domain's typical application requirements [13].

Different embedded applications have different characteristics. Although a complete characterization of embedded applications with respect to applications' characteristics is outside the scope of this paper, Fig. 2 provides a concise classification of embedded applications based on their characteristics. We discuss below some of these application characteristics in context of their associated embedded domains.

Throughput-intensive: *Throughput-intensive* embedded applications are applications that require high processing throughput. Networking and multimedia applications, which constitute a large fraction of embedded applications [13], are typically throughput-intensive due to ever increasing quality of service (QoS) demands. An embedded system containing an embedded processor requires a network stack and network protocols to connect with other devices. Connecting an embedded device or a widget to a network enables remote device management including automatic application upgrades. On a large scale, networked embedded systems can enable HPEC for solving complex large problems traditionally handled only by supercomputers (e.g., climate research, weather forecasting, molecular modeling, physical simulations, and data mining). However, connecting hundreds to thousands of embedded systems for HPC requires sophisticated and scalable interconnection technologies (e.g., packet-switched, wireless interconnects). Examples of networking applications include server I/O devices, network infrastructure equipment, consumer electronics (mobile phones, media players), and various home appliances (e.g., home automation including networked TVs, VCRs, stereos, refrigerators, etc.). Multimedia applications, such as video streaming, require very high throughput of the order of several GOPs. A broadcast video with a specification of 30 frames per second with 720×480 pixels per frame requires approximately 400,000 blocks (group of pixels) to be processed per second. A telemedicine application requires processing of 5 million blocks per second [14].

Thermal-constrained: An embedded application is *thermal-constrained* if an increase in temperature above a threshold could lead to incorrect results or even the embedded system failure. Depending on the target market, embedded applications typically operate above 45 °C (e.g., telecommunication embedded equipment temperature exceeds 55 °C) in contrast to traditional computer systems, which normally operate below 38°C [15]. Meeting embedded application thermal constraints is challenging due to typically harsh and high-temperature operating environments. Limited space and energy budgets exacerbate these thermal challenges since active cooling systems (fans-based) are typically infeasible in most embedded systems, resulting in only passive and fanless thermal solutions.

Reliability-constrained: Embedded systems with high *reliability* constraints are typically required to operate for many years without errors and/or must recover from errors since many reliability-constrained embedded systems are deployed in harsh environments where post-deployment removal and maintenance is infeasible. Hence, hardware and software for reliability-constrained embedded systems must be developed and tested more carefully than traditional computer systems. Safety critical embedded systems (e.g., automotive airbags, space missions, aircraft flight controllers) have very high reliability requirements (e.g., the reliability requirement for a flight-control embedded system on a commercial airliner is 10^{-10} failures per hour where a failure could lead to aircraft loss [16]).

Real-time: In addition to correct functional operation, *real-time* embedded applications have additional stringent timing constraints, which impose real-time operational deadlines on the embedded system's response time. Although real-time operation does not strictly imply high-performance, real-time embedded

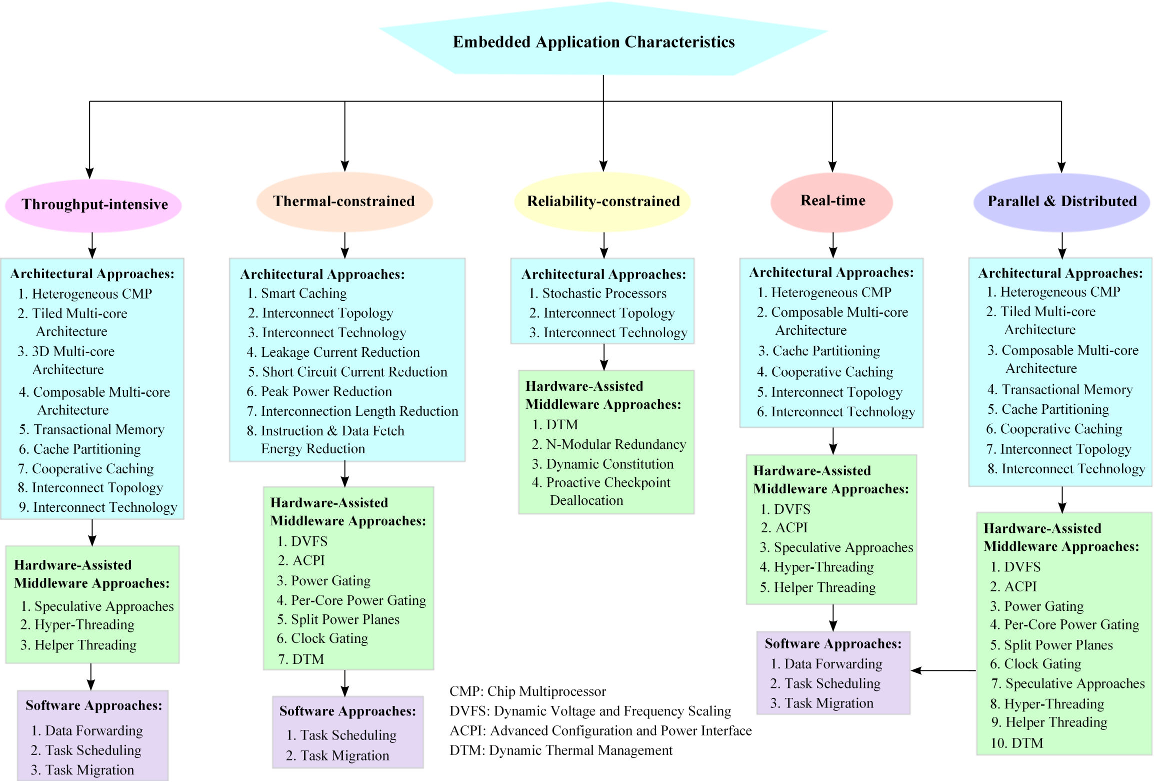

Fig. 2: Classification of high-performance energy-efficient embedded computing (HPEEC) techniques based on embedded application characteristics.

systems require high performance only to the point that the deadline is met, at which time high performance is no longer needed. Hence, real-time embedded systems require *predictable* high-performance. Real-time operating systems (RTOSs) provide guarantees for meeting the stringent deadline requirements for embedded applications.

Parallel and Distributed: *Parallel* and *distributed* embedded applications leverage distributed embedded devices to cooperate and aggregate their functionalities or resources. Wireless sensor network (WSN) applications use sensor nodes to gather sensed information (statistics and data) and use distributed fault-detection algorithms. Mobile agent (autonomous software agent)-based distributed embedded applications allow the process state to be saved and transported to another new embedded system where the process resumes execution from the suspended point (e.g., virtual migration). Many embedded applications exhibit varying degrees (low to high levels) of *parallelism*, such as instruction level parallelism (ILP) and thread-level parallelism (TLP). Innovative architectural and software HPEEC techniques are required to exploit an embedded application's available parallelism to achieve high-performance with low power consumption.

Various HPEEC techniques at different levels (e.g., architecture, middleware, and software) can be used to enable an embedded platform to meet the embedded application requirements. Fig. 2 classifies embedded application characteristics and the HPEEC techniques available at architecture, middleware, and software levels that can be leveraged by the embedded platforms executing these applications to meet the application requirements (we describe the details of these techniques in later sections of the paper). For example, throughput-intensive applications can leverage architectural innovations (e.g., tiled multi-core architectures, high-bandwidth interconnects), hardware-assisted middleware techniques (e.g., speculative approaches, DVFS, hyper-threading), and software techniques (e.g., data forwarding, task scheduling, and task migration). We point out that HPEEC techniques are not orthogonal and many of these techniques can be applied in conjunction with one another to more closely meet application requirements. Furthermore, HPEEC techniques that benefit one application requirement (e.g., reliability) may also benefit other application requirements (e.g., throughput, real-time deadlines). For example, the interconnection network not only determines the fault-tolerance characteristics of embedded systems but also affects the attainable throughput and response time.

3 Architectural Approaches

Novel HPEEC architectural approaches play a dominant role in meeting varying application requirements. These architectural approaches can be broadly categorized into four categories: core layout, memory design, interconnection networks, and reduction techniques. In this section, we describe these HPEEC architectural approaches.

3.1 Core Layout

In this subsection, we discuss various core layout techniques encompassing chip and processor design since high-performance cannot be achieved only from semiconductor technology advancements. There exist various core layout considerations during chip and processor design such as whether to use homogeneous (cores of the same type) or heterogeneous cores (cores of varying types), whether to position the cores in a 2D or 3D layout on the chip, whether to design independent processor cores with switches that can turn on/off processor cores, or to have a reconfigurable integrated circuit that can be configured to form processor cores of different granularity. In this subsection, we describe a few core layout techniques including heterogeneous CMP, conjoined-core CMP, tiled multi-core architectures, 3D multi-core architectures, composable multi-core architectures, multi-component architectures, and stochastic processors. We also discuss the power/energy issues associated with these architectural approaches.

Heterogeneous CMP: Heterogeneous CMPs consist of multiple cores of varying size, performance, and complexity on a single die. Since the amount of ILP or TLP varies for different workloads, building a CMP with some large cores with high single-thread performance and some small cores with greater throughput per die area provides an attractive approach for chip design. Research indicates that the best heterogeneous CMPs contain cores customized to a subset of application characteristics (since no single core can be well suited for all embedded applications) resulting in *non-monotonic* cores (i.e., cores cannot be strictly ordered in terms of performance or complexity for all the applications) [17]. To achieve high performance, applications are mapped to the heterogeneous cores such that the assigned core best meets an application's resource requirements. Heterogeneous CMPs can provide performance gains as high as 40% but at the expense of additional customization cost [18].

Conjoined-core CMP: Conjoined-core CMPs are multiprocessors that allow topologically feasible resource sharing (e.g., floating-point units (FPUs), instruction and data caches) between adjacent cores to reduce die area with minimal impact on performance and improve the overall computational efficiency. Since conjoined-core CMPs are topology oriented, the layout must be co-designed with the architecture otherwise the architectural specifications for resource sharing may not be topologically possible or may incur higher communication costs. In general, the shared resources should be large enough so that the cost of the additional wiring required for sharing may not exceed the area benefits achieved by sharing. Static scheduling is the simplest way to organize resource sharing in conjoined-core CMPs where cores share resources in different non-overlapping cycles (e.g., one core may use the shared resource during even cycles and the other core may use the shared resource during odd cycles, or one core may share the resource for the first five cycles, another core for the next five cycles, and so on). Results indicate that conjoined-core CMPs can reduce area requirements by 50% and maintain performance within 9-12% of conventional cores without conjoining [19].

Tiled Multi-core Architectures: Tiled multi-core architectures exploit massive on-chip resources by combining each processor core with a switch to create a modular element called a *tile*, which can be replicated to create a multi-core embedded system with any number of tiles. Tiled multi-core architectures contain a high-performance interconnection network that constrains interconnection wire length to no longer than the tile width and a switch (communication router) interconnects neighboring switches. Examples of tiled multi-core architectures include the Raw processor, Intel's Tera-Scale research processor, Tilera TILE64, TILE*Pro*64, and TILE-Gx processor family [20].

3D Multi-core Architectures: A 3D multi-core architecture is an integrated circuit that orchestrates architectural units (e.g., processor cores and memories) across cores in a 3D layout. The architecture provides HPEEC by decreasing the interconnection lengths across the chip, which results in reduced communication latency. Research reveals that 3D multi-core processors can achieve 47% performance gain and 20% power reduction on average over 2D multi-core processors [21]. The 3D multi-core architectures' disadvantages include high power density that exacerbates thermal challenges as well as increased interconnect capacitance due to electrical coupling between different layers [22].

Composable Multi-core Architectures: The *composable* multi-core architecture is an integrated circuit that allows the number of processors and each processor's granularity to be configured based on application requirements (i.e., large powerful processors for applications (tasks) with more ILP and small less powerful processors for tasks with more TLP). The architecture consists of an array of composable lightweight processors (CLPs) that can be aggregated to form large powerful processors to achieve high performance depending upon the task granularity. Examples of composable multi-core architectures include TRIPS and TFlex [20].

Stochastic Processors: Stochastic processors are processors used for fault-tolerant computing that are scalable with respect to performance requirements and power constraints while producing outputs that are *stochastically correct* in the worst case. Stochastic processors maintain scalability by exposing multiple functionally equivalent units to the application layer that differ in their architecture and exhibit different reliability levels. Applications select appropriate functional units for a program or program phase based on the program and/or program phase's reliability requirements. Stochastic processors can provide significant power reduction and throughput improvement especially for stochastic applications (applications with *a priori* knowledge of reliability requirements, such as multimedia applications, where computational errors are considered an additional noise source). Results indicate that stochastic processors can achieve 20-60% power savings in the the motion estimation block of H.264 video encoding application [23].

3.2 Memory Design

The cache miss rate, fetch latency, and data transfer bandwidth are some of the main factors impacting the performance and energy consumption of embedded systems. The memory subsystem encompasses the main memory and cache hierarchy and must take into consideration issues such as consistency, sharing, contention, size, and power dissipation. In this subsection, we discuss HPEEC memory design techniques, which include transactional memory, cache partitioning, cooperative caching, and smart caching.

Transactional Memory: Transactional memory incorporates the definition of a *transaction* (a sequence of instructions executed by a single process with the following properties: atomicity, consistency, and isolation) in parallel programming to achieve lock-free synchronization efficiency by coordinating concurrent threads. A computation within a transaction executes atomically and *commits* on successful completion, making the transaction's changes visible to other processes, or *aborts*, causing the transaction's changes to be discarded. A transaction ensures that concurrent reads and writes to shared data do not produce inconsistent or incorrect results. The isolation property of a transaction ensures that a transaction produces the same result as if no other transactions were running concurrently [24]. In transactional memories, regions of code in parallel programming can be defined as a transaction. Transactional memory benefits from hardware support that ranges from complete execution of transactions in hardware to hardware-accelerated software implementations of transactional memory [20].

Cache Partitioning: One of the major challenges in using multi-core embedded systems for real-time applications is timing unpredictability due to core contention for on-chip shared resources (e.g., level two (L2) or level three (L3) caches, interconnect networks). Worst-case execution time (WCET) estimation techniques for single-core embedded systems are not directly applicable to multi-core embedded systems because a task running on one core may evict useful L2 cache contents of another task running on another core. Cache partitioning is a cache space isolation technique that exclusively allocates different portions of shared caches to different cores to avoid cache contention for hard real-time tasks, thus ensuring a more predictable runtime. Cache partitioning-aware scheduling techniques allow each task to use a fixed number of cache partitions ensuring that a cache partition is occupied by at most one scheduled task at any time [25]. Cache partitioning can enhance performance by assigning larger portions of shared caches to cores with higher workloads as compared to the cores with lighter workloads.

Cooperative Caching: Cooperative caching is a hardware technique that creates a globally-managed shared cache using the cooperation of private caches. Cooperative caching allows remote L2 caches to hold and serve data that would not fit in the local L2 cache of a core and therefore improves average access latency by minimizing off-chip accesses [26]. Cooperative caching provides three performance enhancing mechanisms: cooperative caching facilitates cache-to-cache transfers of unmodified data to minimize off-chip accesses, cooperative caching replaces replicated data blocks to make room for unique on-chip data blocks called *singlets*, and cooperative caching allows eviction of singlets from a local L2 cache to be placed in another L2 cache. Cooperative caching implementation requires placement of cooperation-related information in private caches and the extension of cache coherence protocols to support data migration across private caches for capacity sharing. Results indicate that for an 8-core CMP with 1MB L2 cache per core, cooperative caching improves the performance of multi-threaded commercial workloads by 5-11% and 4-38% as compared to shared L2 cache and private L2 caches, respectively [27].

Smart Caching: Smart caching focuses on energy-efficient computing and leverages cache set (way) prediction and low-power cache design techniques [14]. Instead of waiting for the tag array comparison, way prediction predicts the matching way prior to the cache access. Way prediction enables faster *average* cache access time and reduces power consumption because only the predicted way is accessed if the prediction is correct. However, if the prediction is incorrect, the remaining ways are accessed during the subsequent clock cycle(s), resulting in a longer cache access time and increased energy consumption as compared to a cache without way prediction. The *drowsy cache* is a low-power cache design technique that reduces leakage power by periodically setting the unused cache line's SRAM cells to a drowsy, low-power mode. A drowsy cache is advantageous over turning off cache lines completely because the drowsy mode preserves the cache line's data whereas turning off the cache line loses the data. However, drowsy mode requires transitioning the drowsy cache line to a high-power mode before accessing cache line's data. Research reveals that 80-90%

of cache lines can be put in drowsy mode with less than a 1% performance degradation and result in a cache static and dynamic energy reduction of 50-75% [28].

3.3 Interconnection Network

As the number of on-chip cores increases, a scalable and high-bandwidth interconnection network to connect on-chip resources becomes crucial. Interconnection networks can be *static* or *dynamic*. Static interconnection networks consist of point-to-point communication links between computing nodes and are also referred to as *direct* networks (e.g., bus, ring, hypercube). Dynamic interconnection networks consist of switches (routers) and links and are also referred to as *indirect* networks (e.g., packet-switched networks). This subsection discusses prominent interconnect topologies (e.g., bus, 2D mesh, hypercube) and interconnect technologies (e.g., packet-switched, photonic, wireless).

Interconnect Topology: One of the most critical interconnection network parameters is the network topology, which determines the on-chip network cost and performance. The interconnect topology governs the number of hops or routers a message must traverse as well as the interconnection length. Therefore, the interconnect topology determines the communication latency and energy dissipation (since message traversal across links and through routers dissipates energy). Furthermore, the interconnect topology determines the number of alternate paths between computing nodes, which affects reliability (since messages can route around faulty paths) as well as the ability to evenly distribute network traffic across multiple paths, which affects the effective on-chip network bandwidth and performance. The interconnect topology cost is dictated by the *node degree* (the number of links at each computing node) and length of the interconnecting wires. Examples of on-chip interconnection network topologies include buses (linear 1D array or ring), 2D mesh, and hypercube. In bus topology, the processor cores share a common bus for exchanging data. Buses are the most prevalent interconnect network in multi-core embedded systems due to the bus's low cost and ease of implementation. Buses provide lower costs than other interconnect topologies because of a lower node degree: the node degree for a bus interconnect is two, for a 2D mesh is four, and for a hypercube is $\log p$ where p is the total number of computing nodes. However, buses do not scale well as the number of cores in the CMP increases. The 2D mesh interconnect topology provides short channel lengths and low router complexity, however, the 2D mesh diameter is proportional to the perimeter of the mesh, which can lead to energy inefficiency and high network latency (e.g., the diameter of 10×10 mesh is 18 hops) [29]. The hypercube topology is a special case of a d-dimensional mesh (a d-dimensional mesh has a node degree of $2d$) when $d = \log p$.

Packet-switched Interconnect: Packet-switched interconnection networks replace buses and crossbar interconnects as scalability and high-bandwidth demand increases for multi-core embedded systems. Packet-switched networks connect a router to each computing node and routers are connected to each other via short-length interconnect wires. Packet-switched interconnection networks multiplex multiple packet flows over the interconnect wires to provide highly scalable bandwidth [20]. Tilera's TILE architectures leverage the packet-switched interconnection network.

Photonic Interconnect: As the number of on-chip cores in a CMP increases, *global on-chip communication* plays a prominent role in overall performance. While local interconnects scale with the number of transistors, the global wires do not because the global wires span across the entire chip to connect distant logic gates and the global wires' bandwidth requirements increases as the number of cores increases. A photonic interconnection network — consisting of a photonic source, optical modulators (rates exceed 12.5 Gbps), and symmetrical optical waveguides — can deliver higher bandwidth and lower latencies with considerably lower power consumption than an electronic signaling based interconnect network. In photonic interconnects, once a photonic path is established using optical waveguides, data can be transmitted end-to-end without repeaters, regenerators, or buffers as opposed to the electronic interconnects that requires buffering, regeneration, and retransmission of messages multiple times from source to destination [30]. The photonic interconnection network is divided into zones each with a drop point such that the clock signal is optically routed to the drop point where the optical clock signal is converted to the electrical signal. Analysis reveals that power dissipation in an optical clock distribution is lower than an electrical clock distribution [22].

The photonic interconnection networks can benefit several classes of embedded applications, including real-time and throughput-intensive applications (especially applications with limited data reuse such as streaming applications) (Fig. 2). However, even though photonic interconnection networks provide many benefits, these networks have several drawbacks such as delays associated with the rise and fall times of optical emitters and detectors, losses in the optical waveguides, signal noise due to waveguides coupling, limited buffering, and signal processing [22].

Wireless Interconnect: Wireless interconnect is an emerging technology that promises to provide high bandwidth, low latency, and low energy dissipation by eliminating lengthy wired interconnects. Carbon nanotubes (CNT) are a good candidate for wireless antennas due to a CNT's high aspect ratio (virtually a one-dimensional wire), high conductance (low losses), and high current carrying capacity (109 A/cm^2, which is much higher than silver and copper) [22]. Wireless interconnect can deliver high bandwidth by providing multiple channels and using time-division, code-division, frequency-division, or some hybrid of these multiplexing techniques. Experiments indicate that a wireless interconnect can reduce the communication latency by 20-45% as compared to a 2D-mesh interconnect while consuming a comparable amount of power [29]. A wireless interconnect's performance advantage increases as the number of on-chip cores increases. For example, a wireless interconnect can provide a performance gain of 217%, 279%, 600% over a 2D-mesh interconnect when the number of on-chip cores is equal to 128, 256, and 512, respectively [31].

3.4 Reduction Techniques

Due to an embedded system's constrained resources, embedded system architectural design must consider power dissipation reduction techniques. Power reduction techniques can be applied at various design levels: the *complementary metal-oxide-semiconductor (CMOS)-level* targets leakage and short circuit current reduction, the *processor-level* targets instruction/data supply energy reduction as well as power-efficient management of other processor components (e.g., execution units, reorder buffers, etc.), and the *interconnection network-level* targets minimizing interconnection length using an appropriate network layout. In this subsection, we present several power reduction techniques including leakage current reduction, short circuit current reduction, peak power reduction, and interconnection length reduction.

Leakage Current Reduction: As advances in the chip fabrication process reduces the feature size, the CMOS leakage current and associated leakage power has increased. Leakage current reduction techniques include back biasing, silicon on insulator technologies, multi-threshold MOS transistors, and power gating [14].

Short Circuit Current Reduction: Short circuit current flows in a CMOS gate when both nMOSFET and pMOSFET are on, which causes a large amount of current to flow through transistors and can result in increased power dissipation or even transistor burn-out. The short circuit effect is exacerbated as the clock period approaches the transistor switching period due to increasing clock frequencies. The short circuit current can be reduced using low-level design techniques that aim to reduce the time during which both nMOSFET and pMOSFET are on [14].

Peak Power Reduction: Peak power reduction not only increases power supply efficiency but also reduces packaging, cooling, and power supply cost as these costs are proportional to the peak power dissipation rather than the average power dissipation. *Adaptive processors* can reduce peak power by centrally managing architectural component configurations (e.g., instruction and data caches, integer and floating point instruction queues, reorder buffers, load-store execution units, integer and floating point registers, register renaming, etc.) to ensure that not of all these components are maximally configured simultaneously. Adaptive processors incur minimal performance loss and high peak power reduction by restricting maximum configuration to a single resource or a few resources (but not all) at a time. Research reveals that adaptive processors reduce peak power consumption by 25% with only a 5% performance degradation [32].

Interconnection Length Reduction: The interconnecting wire length increases as the number of on-chip devices increases, resulting in both increased power dissipation and delay. An energy-efficient design requires reduced interconnection wire lengths for high switching activity signals and use of placement and routing optimization algorithms for reduced delay and power consumption [14]. Chip design techniques (e.g., 3D multi-core architectures) and various interconnect topologies (e.g., 2D-mesh, hypercube) help in reducing interconnection wire lengths.

Instruction and Data Fetch Energy Reduction: Hardwired ASICs typically provide 50x more efficient computing as compared to general purpose programmable processors, however, architecture-level energy consumption analysis can help in energy-efficient design of programmable processors [1]. Previous work indicates that the programmable processors spend approximately 70% of the total energy consumption fetching instructions (42%) and data (28%) to the arithmetic units, whereas performing the arithmetic consumes a small fraction of the total energy (around 6%). Moreover, the instruction cache consumes the majority of the instruction fetch energy (67%) [1]. Research indicates that reducing instruction and data fetch energy can reduce the energy-efficiency gap between ASICs and programmable processors to 3x. Specifically, instruction fetch techniques that avoid accessing power-hungry caches are required for energy-efficient programmable processors (e.g., the Stanford efficient low-power microprocessor (ELM) fetches instructions from a set of distributed instruction registers rather than the cache) [1].

4 Hardware - Assisted Middleware Approaches

Various HPEEC techniques (Fig. 1) are implemented as middleware and/or part of an embedded OS to meet application requirements. The HPEEC middleware techniques are assisted and/or partly implemented in hardware to provide the requested functionalities (e.g., power gating support in hardware enables middleware to power gate processor cores). HPEEC hardware-assisted middleware techniques include dynamic voltage and frequency scaling (DVFS), advanced configuration and power interface (ACPI), threading techniques (hyper-threading, helper threading, and speculative threading), energy monitoring and management, dynamic thermal management (DTM), dependable HPEEC techniques (N-modular redundancy, dynamic constitution, and proactive checkpoint deallocation), and various low-power gating techniques (power gating, per-core power gating, split power planes, and clock gating).

4.1 Dynamic Voltage and Frequency Scaling

DVFS is a dynamic power management (DPM) technique in which the performance and power dissipation is regulated by adjusting the processor's voltage and frequency. The one-to-one correspondence between processor's voltage and frequency in CMOS circuits imposes a strict constraint on dynamic voltage scaling (DVS) techniques to ensure that the voltage adjustments do not violate application timing (deadline) constraints (especially for real-time applications). Multi-core embedded systems leverage two DVFS techniques: *global DVFS* scales the voltages and frequencies of all the cores simultaneously and *local DVFS* scales the voltage and frequency on a per-core basis [14]. Experiments indicate that local DVFS can improve performance (throughput) by 2.5x on average and can provide an 18% higher throughput than global DVFS on average [33][34].

DVFS-based optimizations can be employed for real-time applications to conform with tasks' deadlines in an energy-efficient manner. For example, if a task deadline is impending, DVFS can be adjusted to operate at the highest frequency to meet the task deadline whereas if the task deadline is not close, then DVFS can be adjusted to lower voltage and frequency settings to conserve energy while still meeting the task deadline.

Although DVFS is regarded as one of the most efficient energy saving technique, the associated overhead of performing DVFS needs to be considered. DVFS requires a programmable DC-DC converter and a programmable clock generator (mostly phase lock loop (PLL)-based) that incurs time and energy overhead whenever the processor changes its voltage and frequency setting. This overhead dictates the minimum duration of time that the target system should stay in a particular voltage-frequency state for the DVS to produce a positive energy gain [35].

4.2 Advanced Configuration and Power Interface

Though DPM techniques can be implemented in hardware as part of the electronic circuit, hardware implementation complicates the modification and reconfiguration of power management policies. The advanced configuration and power interface (ACPI) specification is a platform-independent software-based power management interface that attempts to unify existing DPM techniques (e.g., DVFS, power and clock gating) and put these techniques under the OS control [36]. ACPI defines various states for an ACPI-compliant embedded system, but the processor power states (C-states) and the processor performance states (P-states) are most relevant to HPEEC. ACPI defines four C-states: C0 (the *operating state* where the processor executes instructions normally), C1 (the *halt state* where the processor stops executing instructions but can return to C0 instantaneously), C2 (the *stop-clock state* where the processor and cache maintains state but can take longer to return to C0), and C3 (the *sleep state* where the processor goes to sleep, does not maintain the processor and cache state, and takes longest as compared to other C-states to return to C0). ACPI defines n P-states (P1, P2, $\ldots$, Pn) where $n \leq 16$, corresponding to the processor C0 state. Each P-state designates a specific DVFS setting such that P0 is the highest performance state while P1 to Pn are successively lower performance states. ACPI specification is implemented in various manufactured chips (e.g., Intel names P-states as SpeedStep while AMD as Cool'n'Quiet).

4.3 Gating Techniques

To enable low-power operation and meet an application's constrained energy budget, various hardware-supported low power gating techniques can be controlled by the middleware. These gating techniques can switch off a component's supply voltage or clock signal to save power during otherwise idle periods. In this subsection, we discuss gating techniques such as power gating, per-core power gating, split power planes, and clock gating.

Power Gating: Power gating is a power management technique that reduces leakage power by switching off the supply voltage to idle logic elements after detecting no activity for a certain period of time. Power gating can be applied to idle functional units, cores, and cache banks [14].

Per-Core Power Gating: Per-core power gating is a fine-grained power gating technique that individually switches off idle cores. In conjunction with DVFS, per-core power gating provides more flexibility in optimizing performance and power dissipation of multi-core processors running applications with varying degrees of parallelism. Per-core power gating increases single-thread performance on a single active core by increasing the active core's supply voltage while power gating the other idle cores, which provides additional power- and thermal-headroom for the active core. Experiments indicate that per-core power gating in conjunction with DVFS can increase the throughput of a multi-core processor (with 16 cores) by 16% on average for different workloads exhibiting a range of parallelism while maintaining the power and thermal constraints [37].

Split Power Planes: Split power planes is a low-power technique that allows different power planes to coexist on the same chip and minimizes both static and dynamic power dissipation by removing power from idle portions of the chip. Each power plane has separate pins, a separate (or isolated) power supply, and independent power distribution routing. For example, Freescale's MPC8536E PowerQUIC III processor has two power planes: one plane for the processor core (e500) and L2 cache arrays, and a second plane for the remainder of the chip's components [38].

Clock Gating: Clock gating is a low-power technique that allows gating off the clock signal to registers, latches, clock regenerators, or entire subsystems (e.g., cache banks). Clock gating can yield significant power savings by gating off the functional units (e.g., adders, multipliers, and shifters) not required by the currently executing instruction, as determined by the instruction decode unit. Clock gating can also be applied internally for each functional unit to further reduce power consumption by disabling the functional unit's upper bits for small operand values that do not require the functional unit's full bit width. The granularity at which clock gating can be applied is limited by the overhead associated with the clock enable signal generation [14].

4.4 Threading Techniques

Different threading techniques target high performance by either enabling a single processor to execute multiple threads or by speculatively executing multiple threads. Prominent high-performance threading techniques include hyper-threading, helper threading, and speculative threading. We point out that helper and speculative threading are performance-centric and may lead to increased power consumption in case of misspeculation where speculative processing needs to be discarded. Therefore, helper and speculative threading should be used with caution in energy critical embedded systems. Below we describe a brief description of these threading techniques.

Hyper-Threading: Hyper-threading leverages simultaneous multithreading to enable a single processor to appear as two logical processors and allows instructions from both of the logical processors to execute simultaneously on the shared resources [26]. Hyper-threading enables the OS to schedule multiple threads to the processor so that different threads can use the idle execution units. The architecture state, consisting of general-purpose registers, interrupt controller registers, control registers, and some machine state registers, is duplicated for each logical processor. However, hyper-threading does not offer the same performance as a multiprocessor with two physical processors.

Helper Threading: Helper threading leverages special execution modes to provide faster execution by reducing cache miss rates and miss latency [26]. Helper threading accelerates performance of single-threaded applications using speculative pre-execution. This pre-execution is most beneficial for irregular applications where data prefetching is ineffective due to challenging data addresses prediction. The helper threads run ahead of the main thread and reduce cache miss rates and miss latencies by pre-executing regions of the code that are likely to incur many cache misses. Helper threading can be particularly useful for applications with multiple control paths where helper threads pre-execute all possible paths and prefetch the data references for all paths instead of waiting until the correct path is determined. Once the correct execution path is determined, all the helper threads executing incorrect paths are aborted.

Speculative Threading Speculative threading approaches provide high performance by removing unnecessary serialization in programs. We discuss two speculative approaches: speculative multi-threading and speculative synchronization.

Speculative multi-threading divides a sequential program into multiple contiguous program segments called tasks and execute these tasks in parallel on multiple cores. The architecture provides hardware support for detecting dependencies in a sequential program and rolling back the program state on misspeculations. Speculative multi-threaded architectures exploit high transistor density by having multiple cores and relieves programmers from parallel programming, as is required for conventional CMPs. Speculative multi-threaded architectures provide instruction windows much larger than conventional uniprocessors by combining the instruction windows of multiple cores to exploit distant TLP as opposed to the nearby ILP exploited by conventional uniprocessors [20].

Speculative synchronization removes unnecessary serialization by applying thread-level speculation to parallel applications and preventing speculative threads from blocking at barriers, busy locks, and unset flags. Hardware monitors detect conflicting accesses and roll back the speculative threads to the synchronization point prior to the access violation. Speculative synchronization guarantees forward execution using a safe thread that ensures that the worst case performance of the order of conventional synchronization (i.e., threads not using any speculation) when speculative threads fail to make progress.

4.5 Energy Monitoring and Management

Profiling the power consumption of various components (e.g., processor cores, caches) for different embedded applications at a fine granularity identifies how, when, and where power is consumed by the embedded system and the applications. Power profiling is important for energy-efficient HPEEC system design. Energy monitoring software can monitor, track, and analyze performance and power consumption for different components at the function-level or block-level granularity. PowerPack is an energy monitoring tool that uses a combination of hardware (e.g., sensors and digital meters) and software (e.g., drivers, benchmarks, and analysis tools). PowerPack profiles power and energy, as well as power dynamics, of DVFS in CMP-based cluster systems for different parallel applications at the component and code segment granularity [39].

Power management middleware dynamically adapts the application behavior in response to fluctuations in workload and power budget. PowerDial is a power management middleware that transforms static application configuration parameters into dynamic control variables stored in the address space of the executing application [40]. These control variables are accessible via a set of dynamic knobs to change the running application's configuration dynamically to tradeoff computation accuracy (as far as the applications minimum accuracy requirements are satisfied) and resource requirements, which translates to power savings. Experiments indicate that PowerDial can reduce power consumption by 75%.

Green is a power management middleware that enables application programmers to exploit approximation opportunities to meet performance demands while meeting quality of service (QoS) guarantees [41]. Green provides a framework that enables application programmers to approximate expensive functions and loops. Green operates in two phases: the *calibration phase* and the *operation phase*. In the calibration phase, Green creates a QoS loss model for the approximated functions to quantify the approximation impact (loss in accuracy). The operational phase uses this QoS loss model to make approximation decisions based on programmer-specified QoS constraints. Experiments indicate that Green can improve the performance and energy consumption by 21% and 14%, respectively, with only a 0.27% QoS degradation.

4.6 Dynamic Thermal Management

Temperature has become an important constraint in HPEEC embedded systems because high temperature increases cooling costs, degrades reliability, and reduces performance. Furthermore, an embedded application's distinct and time-varying thermal profile necessitates dynamic thermal management (DTM) approaches. DTM for multi-core embedded systems is more challenging than for the single-core embedded systems because a core's configuration and workload has a significant impact on the temperature of neighboring cores due to lateral heat transfer between adjacent cores. The goal of DTM techniques is to maximize performance while keeping temperature below a defined threshold.

Temperature Determination for DTM: DTM requires efficient chip thermal profiling, which can be done using *sensor-based*, *thermal model-based*, or *performance counters-based* methods. Sensor-based methods leverage physical sensors to monitor the temperature in real-time. DTM typically uses one of the two sensor placement techniques: *global sensor placement* monitors global chip hotspots and *local sensor placement* places sensors in each processor component to monitor local processor components. Thermal model-based methods use thermal models that exploit the duality between electrical and thermal phenomena by leveraging lumped-RC (resistor/capacitor) models. Thermal models can either be low-level or high-level. *Low-level thermal models* estimate temperature accurately and report the steady state as well as provide transient temperature estimation, however, are computationally expensive. *High-level thermal models* leverage a simplified lumped-RC model that can only estimate the steady state temperature, however, are computationally less expensive than the low-level thermal methods. Performance counters-based methods estimate the temperature of different on-chip functional units using temperature values read from specific processor counter registers. These counter readings can be used to estimate the access rate and timing information of various on-chip functional units.

Techniques Assisting DTM: DVFS is one of the major technique that helps DTM in maintaining a chip's thermal balance and alleviates a core's thermal emergency by reducing the core voltage and frequency. DVFS can be *global* or *local*. Global DVFS provides less control and efficiency as a single core's hotspot could result in unnecessary stalling or scaling of all the remaining cores. Local DVFS control each core's voltage and frequency individually to alleviate thermal emergency of the affected cores, however, introduces design complexity. A *hybrid* local-global thermal management approach has the potential to provide better performance than local DVFS while maintaining the simplicity of global DVFS. The hybrid approach applies global DVFS across all the cores but specializes the architectural parameters (e.g., instruction window size, issue width, fetch throttling/gating) of each core locally. Research reveals that the hybrid approach achieves a 5% better throughput than the local DVFS [34]. Although DVFS can help DTM to maintain thermal balance, there exists other techniques to assist DTM, e.g., Zhou et al. [42] suggested that adjusting micro-architectural parameters such as instruction window size and issue width have relatively lower overhead than DVFS-based approaches.

4.7 Dependable Techniques

To achieve performance-efficiency while meeting an application's reliability requirements defines the dependable HPEEC (DHPEEC) domain, which ranges from redundancy techniques to dependable processor design. DHPEEC platforms are critical for space exploration, space science, and defense applications with ever increasing demands for high data bandwidth, processing capability, and reliability. We describe several hardware-assisted middleware techniques leveraged by DHPEEC including N-modular redundancy, dynamic constitution, and proactive checkpoint deallocation.

N-modular Redundancy: The process variation, technology scaling (deep submicron and nanoscale devices), and computational energy approaching thermal equilibrium leads to high error rates in CMPs, which necessitates redundancy to meet reliability requirements. Core-level N-modular redundancy (NMR) runs N program copies on N different cores and can meet high reliability goals for multi-core processors. Each core performs the same computation and the results are voted (compared) for consistency. Voting can either be time-based or event-based. Based on the voting result, program execution continues or rolls back to a checkpoint (a previously stored, valid architectural state). A multi-core NMR framework can provide either *static* or *dynamic* redundancy. Static redundancy uses a set of statically configured cores whereas dynamic redundancy assigns redundant cores during runtime based on the application's reliability requirements and environmental stimuli [43]. Static redundancy incurs high area requirement and power consumption due to the large number of cores required to meet an application's reliability requirements, whereas dynamic redundancy provides better performance, power, and reliability tradeoffs.

The dependable multiprocessor (DM) is an example of a DHPEEC platform which leverages NMR. The DM design includes a fault-tolerant embedded message passing interface (FEMPI) (a lightweight fault-tolerant version of the Message Passing Interface (MPI) standard) for providing fault-tolerance to parallel embedded applications [44]. Furthermore, DM can leverage HPEC platforms such as the Tile*Pro*64 [45].

Dynamic Constitution: *Dynamic constitution*, an extension of dynamic redundancy, permits an arbitrary core on a chip to be a part of an NMR group, which increases dependability as compared to the static NMR configuration by scheduling around cores with permanent faults. For example, if an NMR group is statically constituted and the number of cores with permanent faults drops below the threshold to meet the application's reliability requirements, the remaining non-faulty cores in the NMR group are rendered useless. Dynamic constitution can also be helpful in alleviating thermal constraints by preventing NMR hotspots [46].

Proactive Checkpoint Deallocation: *Proactive checkpoint deallocation* is a high-performance extension for NMR that permits cores participating in voting to continue execution instead of waiting on the voting logic results. After a voting logic decision, only the cores with correct results are allowed to continue further execution.

5 Software Approaches

The performance and power efficiency of an embedded platform not only depends upon the built-in hardware techniques but also depends upon the software's ability to effectively leverage the hardware support. Software-based HPEEC techniques assist DPM by signaling the hardware of the resource requirements of an application phase. Software approaches enable high performance by scheduling and migrating tasks statically or dynamically to meet application requirements. HPEEC software-based techniques include data forwarding, task scheduling, task migration, and load balancing.

5.1 Data Forwarding

Data forwarding benefits HPEEC by hiding memory latency, which is more challenging in multiprocessor systems as compared to uniprocessor systems because uniprocessor caches can hide memory latency by exploiting spatial and temporal locality whereas coherent multiprocessors have sharing misses in addition to the non-sharing misses present in uniprocessor systems. In a shared memory architecture, processors that cache the same data address are referred as *sharing processors*. Data forwarding integrates fine-grained message passing capabilities in a shared memory architecture and hides the memory latency associated with sharing accesses by sending the data values to the sharing processors as soon as the data values are produced [47]. Data forwarding can be performed by the compiler where the compiler inserts *write and forward* assembly instructions in place of ordinary write instructions. Compiler-assisted data forwarding uses an extra register to indicate the processors that should receive the forwarded data. Another data forwarding technique referred as programmer-assisted data forwarding requires a programmer to insert a *post-store* operation that causes a copy of an updated data value to be sent to all the sharing processors. Experiments indicate that remote writes together with prefetching improve performance by 10-48% relative to the base system (no data forwarding and prefetching) whereas remote writes improve performance by 3-28% relative to the base system with prefetching [26].

5.2 Load Distribution

A multi-core embedded system's performance is dictated by the workload distribution across the cores, which in turn dictates the execution time and power/thermal profile of each core. Load distribution techniques focus on load balancing between the executing cores via task scheduling and task migration.

Task Scheduling: The task scheduling problem can be defined as determining an optimal assignment of tasks to cores that minimizes the power consumption while maintaining the chip temperature below the DTM enforced ceiling temperature with minimal or no performance degradation given the total energy budget. Task scheduling applies for both DPM and DTM and plays a pivotal role in extending battery life for portable embedded systems, alleviating thermal emergencies, and enabling long-term savings from reduced cooling costs. Task scheduling can be applied in conjunction with DVFS to meet real-time task deadlines as a higher processing speed results in faster task execution and shorter scheduling lengths, but at the expense of greater power consumption. Conversely, the decrease in processor frequency reduces power consumption but increases the scheduling length, which may increase the overall energy consumption. Since the task scheduling overhead increases as the number of cores increases, hardware-assisted task scheduling techniques are the focus of emerging research (e.g., thread scheduling in graphics processing units (GPUs) is hardware-assisted). Experiments indicate that hardware-assisted task scheduling can improve the scheduling time by 8.1% for CMPs [48].

Task Migration: In a multi-threaded environment, threads periodically and/or aperiodically enter and leave cores. Thread migration is a DPM and DTM technique that allows a scheduled thread to execute, preempt, or migrate to another core based on the thread's thermal and/or power profile. The OS or thread scheduler can dynamically migrate threads running on cores with limited resources to the cores with more resources as resources become available. Depending on the executing workloads, there can be a substantial temperature variation across cores on the same chip. Thread migration-based DTM periodically moves threads away from the hot cores to the cold cores based on this temperature differential to maintain the cores' thermal balance. A thread migration technique must take into account the overhead incurred due to thread migration communication costs and address space updates. Temperature determination techniques (e.g., performance counter-based, sensor-based) assist thread management techniques in making migration decisions.

Thread migration techniques can be characterized as *rotation-based*, *temperature-based*, or *power-based* [49]. The rotation-based technique migrates a thread from core (i) to core $((i+1) \mod N)$ where N denotes the total number of processor cores. The temperature-based technique orders cores based on the cores' temperature and the thread on core (i) is swapped with the thread on core $(N-i-1)$ (i.e., the thread on the hottest core is swapped with the thread on the coldest core, the thread on the second hottest core is swapped with the thread on the second coldest core, and so on). The power-based technique orders cores based on the cores'temperature in ascending order and orders threads based on the threads' power consumption in descending order. The power-based technique then schedules thread (i) to core (i) (e.g., the most power-hungry thread is scheduled to the coldest core).

Thread migration can be applied in conjunction with DVFS to enhance performance. Research indicates that thread migration alone can improve performance by 2x on average whereas thread migration in conjunction with DVFS can improve performance by 2.6x on average [33].

Load Balancing and Unbalancing: Load balancing techniques distribute a workload equally across all the cores in a multi-core embedded system. Load unbalancing can be caused by either *extrinsic* or *intrinsic* factors. Extrinsic factors are associated with the OS and hardware topology. For example, the OS can schedule daemon processes during the execution of a parallel application and an asymmetric hardware topology can result in varying communication latencies for different processes. Intrinsic factors include imbalanced parallel algorithms, imbalanced data distribution, and changes in the input data set. An unbalanced task assignment can lead to a performance degradation because cores executing light workloads may have to wait/stall for other cores executing heavier workloads to reach a synchronization point. Load balancing relies on efficient task scheduling techniques as well as balanced parallel algorithms. Cache partitioning can assist load balancing by assigning more cache partitions to the cores executing heavier workloads to decrease the cache miss rate and increase the core's execution speed, and thus reduce the stall time for the cores executing light workloads [50].

Although load balancing provides a mechanism to achieve high performance in embedded systems, load balancing may lead to high power consumption if not applied judiciously because load balancing focuses on utilizing all the cores even for a small number of tasks. A load unbalancing strategy that considers workload characteristics (i.e., *periodic* or *aperiodic*) can achieve better performance and lower power consumption as compared to a load balancing or a load unbalancing strategy that ignores workload characteristics. A workload-aware load unbalancing strategy assigns repeatedly executed periodic tasks to a minimum number of cores and distributes aperiodic tasks that are not likely to be executed repeatedly to a maximum number of cores. We point out that the critical performance metric for periodic tasks is deadline satisfaction rather than faster execution (a longer waiting time is not a problem as long as the deadline is met), whereas the critical performance metric for aperiodic tasks is response time rather than deadline satisfaction. The periodic tasks not distributed over all the cores leave more idle cores for scheduling aperiodic tasks, which shortens the response time of aperiodic tasks. Results on an ARM11MPCore chip demonstrate that the workload-aware load unbalancing strategy reduces power consumption and the mean waiting time of aperiodic tasks by 26% and 82%, respectively, as compared to a load balancing strategy. The workload-aware load unbalancing strategy reduces the mean waiting time of aperiodic tasks by 92% with similar power efficiency as compared to a workload unaware load unbalancing strategy [51].

TABLE 2: High-performance energy-efficient multi-core processors

Processor	Cores	Speed	Power	Performance
ARM11 MPCore	1 - 4	620 MHz	600 mW	2600 DMIPS
ARM Cortex A-9 MPCore	1 - 4	800 MHz - 2 GHz	250 mW per CPU	4,000 - 10,000 DMIPS
MPC8572E PowerQUICC III	2	1.2 GHz - 1.5 GHz	17.3 W @ 1.5 GHz	6897 MIPS @ 1.5 GHz
Tilera TILE*Pro*64	64 tiles	700 MHz - 866 MHz	19 - 23 W @ 700 MHz	443 GOPS
Tilera TILE-Gx	16/36/64/100 tiles	1 GHz - 1.5 GHz	10 - 55 W	750 GOPS
AMD Opteron 6100	8/12	1.7 - 2.3 GHz	65 - 105 W	—
Intel Xeon Processor LV 5148	2	2.33 GHz	40 W	—
Intel Sandy Bridge	4	3.8 GHz	35 - 45 W	121.6 GFLOPS
AMD Phenom II X6 1090T	6	3.6 GHz	125 W	—
NVIDIA GeForce GTX 460	336 CUDA cores	1.3 GHz	160 W	748.8 GFLOPS
NVIDIA GeForce 9800 GX2	256 CUDA cores	1.5 GHz	197 W	1152 GFLOPS
NVIDIA GeForce GTX 295	480 CUDA cores	1.242 GHz	289 W	748.8 GFLOPS
NVIDIA Tesla C2050/C2070	448 CUDA cores	1.15 GHz	238 W	1.03 TFLOPS
AMD FireStream 9270	800 stream cores	750 MHz	160 W	1.2 TFLOPS
ATI Radeon HD 4870 X2	1600 stream cores	750 MHz	423 W	2.4 TFLOPS

6 High-Performance Energy-Efficient Multi-core Processors

Silicon and chip vendors have developed various high-performance multi-core processors that leverage the various HPEEC techniques discussed in this paper. Although providing an exhaustive list of all the prevalent high-performance multi-core processors that can be used in embedded applications is outside of the scope of this paper, we discuss some prominent multi-core processors (summarized in Table 2) and focus on their HPEEC features.[1]

Tilera TILE*Pro*64 and TILE-Gx: Tilera revolutionizes high-performance multi-core embedded computing by leveraging a tiled multi-core architecture (e.g., the TILE*Pro*64 and TILE-Gx processor family [52][53]). The TILE*Pro*64 and TILE-Gx processor family feature an 8x8 grid and an array of 16 to 100 tiles (cores), respectively, where each tile consists of a 32-bit very long instruction word (VLIW) processor, three deep pipelines delivering up to 3 instructions per cycle (IPC), integrated L1 and L2 cache, and a non-blocking switch that integrates the tile into a power-efficient interconnect mesh. The TILE*Pro*64 and TILE-Gx processors offer 5.6 MB and 32 MB of on-chip cache, respectively, and implement Tilera's dynamic distributed cache (DDC) technology that provides a 2x improvement on average in cache coherence performance over traditional cache technologies using a cache coherence protocol. Each tile can independently run a complete OS or multiple tiles can be grouped together to run a multi-processing OS like SMP Linux. The TILE*Pro*64 and TILE-Gx processor family employs DPM to put idle tiles into a low-power sleep mode. The TILE*Pro*64 and TILE-Gx family of processor can support a wide range of computing applications including advanced networking, wireless infrastructure, telecom, digital multimedia, and cloud computing.

Intel Xeon Processor: Intel leverages Hafnium Hi-K and metal gates in next generation Xeon processors to achieve higher clock speeds and better performance per watt. The Xeon processors also implement *hyper-threading* and *wide dynamic execution* technologies for high performance. The wider execution pipelines enable each core to simultaneously fetch, dispatch, execute, and retire up to four instructions per cycle [54]. The Intel Xeon 5500 processor family features 15 power states and a fast transition between these power states (less than 2 microseconds) [3]. The Xeon processors are based on Intel Core 2 Duo micro-architecture where the two cores share a common L2 cache to provide faster inter-core communication. The shared L2 cache can be dynamically resized depending on individual core's needs. Intel's *deep power down* technology enables both cores and the L2 cache to be powered down when the processor is idle [55]. Intel's *dynamic power coordination* technology allows software-based DPM to alter each core's sleep state to tradeoff between power dissipation and performance. The processor incorporates digital temperature sensors on each core to monitor thermal behavior using Intel's *advanced thermal manager* technology [56]. The Dual-core Intel Xeon processor LV 5148 — a low-power embedded processor — enables *micro-gating* of processor circuitry to disable the processor's inactive portions with finer granularity [57]. Typical applications for the Intel Xeon processor include medical imaging, gaming, industrial control and automation systems, mobile devices, military, and aerospace.

Graphics Processing Units: A graphics processing unit (GPU) is a massively parallel processor capable of executing a large number of threads concurrently, and accelerates and offloads graphics rendering from the CPU. GPUs feature high memory bandwidth that is typically 10x faster than contemporary CPUs. NVIDIA and AMD/ATI are the two main GPU vendors. GPUs are suitable for high-definition (HD) videos, photos,

[1] Additional discussion for multi-core processors is given in appendix which is provided by the authors as a supplementary material available at http://ieeexplore.ieee.org.

3D movies, high-resolution graphics, and gaming. Apart from high graphics performance, GPUs enable general-purpose computing on graphics processing units (GPGPU), which is a computing technique that leverages GPUs to perform compute-intensive operations traditionally handled by CPUs. GPGPUs are realized by adding programmable stages and higher precision arithmetic to the rendering pipelines, which enables stream processors to process non-graphics data. For example, NVIDIA Tesla personal supercomputer consisting of 3 or 4 Tesla C1060 computing processors [58] offers up to 4 TFLOPS of compute capability with 4 GB of dedicated memory per GPU [59].

NVIDIA's PowerMizer technology — available on all NVIDIA GPUs — is a DPM technique that adapts the GPU to suit an application's requirements [60]. Digital watchdogs monitor GPU utilization and turn off idle processor engines. NVIDIA's Parallel DataCache technology accelerates algorithms, such as ray-tracing, physics solvers, and sparse matrix multiplication, where data addresses are not known *a priori* [61]. ATI's PowerPlay technology is a DPM solution that monitors GPU activity and adjusts GPU power between low, medium, and high states via DVFS based on workload charateristics. For example, PowerPlay puts the GPU in a low-power state when receiving and composing emails, and switches the GPU to a high-power state for compute-intensive gaming applications. PowerPlay incorporates on-chip sensors to monitor the GPU's temperature and triggers thermal actions accordingly. The PowerPlay technology is available on the ATI Radeon HD 3800 and 4800 series graphics processors, the ATI Mobility Radeon graphics processors, and the Radeon Express motherboard chipsets.

7 Conclusions, Challenges, and Future Research Directions

HPEEC is an active and expanding research domain with applications ranging from consumer electronics to supercomputers. The introduction of HPEEC into supercomputing has boosted the significance of the HPEEC domain as power is becoming a concern for modern supercomputing considering the long-term operation and cooling costs. Modern supercomputers are a combination of custom-design and embedded processors, such as Opteron, Xeon, and coprocessors such as NVIDIA Tesla general-purpose graphics processing units (GPGPUs), AMD graphics processing units (GPUs), etc. For example, the Tianhe-1A supercomputer (the world's second fastest supercomputer as of June 2011 and located at the National Supercomputing Center in Tianjinin, China [5]) leverages Intel Xeon processors as well as NVIDIA Tesla GPGPUs. An increasing growth and expansion of HPEEC is envisioned in the foreseeable future as supercomputers rely more and more on HPEEC.

This paper gives an overarching survey of HPEEC techniques that enable meeting diverse embedded application requirements. We discuss state-of-the-art multi-core processors that leverage these HPEEC techniques. Despite remarkable advancements, the HPEEC domain still faces various arduous challenges, which require further research to leverage the full-scale benefits of HPEEC techniques. Although power is still a first-order constraint in HPEEC platforms, we discuss several additional challenges facing the HPEEC domain (summarized in Table 3) along with future research directions.

Heterogeneous CMPs provide performance efficiency, but present additional design challenges as design space increases considering various types of cores and the flexibility of changing each core's architectural parameters (e.g., issue width, instruction window size, fetch gating) for an arbitrary permutations of workloads. Furthermore, for a given die size, there exists a fundamental tradeoff between number and type of cores and appropriate cache sizes for these cores. Efficient distribution of available cache size across the cache hierarchies (private and shared) to provide high performance is challenging [17].

Synchronization between multiple threads running on multiple cores introduces performance challenges. Threads use semaphores or locks to control access to shared data, which degrades performance due to the busy waiting of threads. Furthermore, threads use synchronization barriers (a defined point in the code where all threads must reach before further execution), which decreases performance due to idle-waiting of faster threads for slower threads.

Although different threads can work independently on private data, shared memory becomes a bottleneck due to large number of shared-data accesses to different cache partitions. Furthermore, threads can communicate via shared memory, which requires cache state transitions to transfer data between threads. Threads must stall until cache state transitions occur, as there is likely insufficient speculative or out-of-order work available for these threads. Moreover, designing a common interface to the shared cache, clock distribution, and cache coherence provides additional design challenges [26].

Cache coherence is required to provide a consistent memory view in shared-memory multi-core processors with various cache hierarchies. Embedded systems conventionally rely on software-managed cache coherency, which does not scale well with the number of cores and thereby necessitates hardware-assisted cache coherence. Hardware-software codesign of cache coherence protocol defines challenging tradeoffs between performance, power, and time-to-market [62].

Cache thrashing — an additional HPEEC challenge — is a phenomenon where threads continually evict each others working set from the cache, which increases the miss rate and latency for all threads. Although

TABLE 3: High-performance energy-efficient embedded computing (HPEEC) challenges.

Challenge	Description
Complex design space	Large design space due to various core types (homogeneous, heterogeneous) and each core's tunable parameters (e.g., instruction window size, issue width, fetch gating)
High on-chip bandwidth	Increased communication due to increasing number of cores requires high-bandwidth on-chip interconnects
Synchronization	Synchronization primitives (e.g., locks, barriers) results in programs serialization degrading performance
Shared memory bottleneck	Threads running on different cores make large number of accesses to various shared memory data partitions
Cache coherence	Heterogeneous cores with different cache line sizes require cache coherence protocols redesign and synchronization primitives (e.g., semaphores, locks) increase cache coherence traffic
Cache thrashing	Threads working concurrently evict each others data out of the shared cache to bring their own data

direct-mapped caches present an attractive choice for multi-core embedded systems due to a direct-mapped cache's power efficiency as compared to associative caches, direct-mapped caches are more predisposed to thrashing as compared to set associative caches. Cache thrashing can be minimized by providing larger and more associative caches, however, these opportunities are constrained by strict power requirements for embedded systems. Victim caches employed alongside direct-mapped caches help to alleviating cache thrashing by providing associativity for localized cache conflict regions [63].

Various new avenues are emerging in HPEEC such as energy-efficient data centers, grid and cluster embedded computing and dependable HPEEC. Various vendors are developing energy-efficient high-performance architectures for data centers by leveraging a huge volume of low-power mobile processors (e.g., SeaMicro's SM10000 servers family integrates 512 low-power X86 1.66 GHz, 64-bit, Intel Atom cores [64]). Advances are being made in grid and cluster embedded computing, e.g., AMAX's ClusterMax SuperG GPGPU clusters consisting of NVIDIA Tesla 20-series GPU computing platforms feature 57,344 GPU cores and offer 131.84 TFLOPS of single precision performance and 65.92 TFLOPS of double precision performance [65]. Though grid embedded computing has revolutionized HPEEC, but requires further investigation in associated task scheduling policies due to the unique dynamics of grid embedded computing. Different heterogeneous embedded processors can be added to or removed from the grid dynamically, which requires intelligent dynamic task scheduling policies to map tasks to the best available computing nodes. The task scheduling policies must consider the impact of dynamic changes in available computing resources on time and energy requirements of tasks.

As the number of on-chip cores increases to satisfy performance demands, communicating data between these cores in an energy-efficient manner becomes challenging and requires scalable, high-bandwidth interconnection networks. Although wireless interconnects provide a power-efficient high-performance alternative to wired interconnects, associated research challenges include partitioning of wired and wireless interconnect domains, directional antenna design, and lightweight medium access control (MAC) protocols. Since many supercomputing applications leverage multiple many-core chips (CMOS technology and power dissipation limit restricts the number of processor cores on a single chip), design of high-bandwidth and low-power interconnection networks between these many-core chips is also an emerging research avenue. Although photonic network designs have been proposed in literature as a prospective low-power and high-bandwidth solution to interconnect many-core CMPs [66][67], the domain of scalable interconnection networks (inter-chip and intra-chip) requires further research.

Dynamic optimization techniques that can autonomously adapt embedded systems according to changing application requirements and environmental stimuli present an interesting research avenue. The task scheduling techniques in real-time embedded systems are typically based on tasks' worst-case execution times, which can produce slack time whenever a task finishes execution before the task's deadline. Therefore, dynamic task scheduling techniques that leverage this slack time information at runtime to reduce energy consumption are crucial for HPEEC systems and require further research.

To keep up with the Moore's law, innovative transistor technologies are needed that can permit high transistor density on-chip facilitating chip miniaturization while allowing operation at higher speeds with lower power consumption as compared to the contemporary CMOS transistor technology. Miniaturized embedded multi-core processor/memory design and fabrication using new transistor technologies (e.g., multiple gate field-effect transistors (MuGFETs), FinFETs, Intel's tri-gate) is an interesting HPEEC lithography research avenue [68].

Finally, advanced power monitoring and analysis tools are required for HPEEC platforms to monitor power at a fine granularity (i.e., the functional unit-level in relation to an application's code segments) and profile architectural components with respect to power consumption for different code segments. Specifically, power measurement and analysis tools for GPUs are required considering the proliferation of GPUs in the HPEEC domain [69].

ACKNOWLEDGMENTS

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the National Science Foundation (NSF) (CNS-0953447 and CNS-0905308). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSERC and the NSF.

References

[1] W. Dally, J. Balfour, D. Black-Shaffer, J. Chen, R. Harting, V. Parikh, J. Park, and D. Sheffield, "Efficient Embedded Computing," *IEEE Computer*, vol. 41, no. 7, pp. 27–32, July 2008.

[2] J. Balfour, "Efficient Embedded Computing," *Ph.D. Thesis, EE Dept., Stanford Univ.*, May 2010.

[3] P. Gepner, D. Fraser, M. Kowalik, and R. Tylman, "New Multi-Core Intel Xeon Processors Help Design Energy Efficient Solution for High Performance Computing," in *Proc. of IMCSIT*, Mragowo, Poland, October 2009.

[4] P. Crowley, M. Franklin, J. Buhler, and R. Chamberlain, "Impact of CMP Design on High-Performance Embedded Computing," in *Proc. of HPEC Workshop*, Lexington, Massachusetts, September 2006.

[5] Top500, "Top 500 Supercomputer Sites," June 2011. [Online]. Available: http://www.top500.org/

[6] Green500, "Ranking the World's Most Energy-Efficient Supercomputers," June 2011. [Online]. Available: http://www.green500.org/

[7] K. Hwang, "Advanced Parallel Processing with Supercomputer Architectures," *Proceedings of the IEEE*, vol. 75, no. 10, pp. 1348–1379, October 1987.

[8] A. Klietz, A. Malevsky, and K. Chin-Purcell, "Mix-and-match High Performance Computing," *IEEE Potentials*, vol. 13, no. 3, pp. 6–10, August/September 1994.

[9] W. Pulleyblank, "How to Build a Supercomputer," *IEEE Review*, vol. 50, no. 1, pp. 48–52, January 2004.

[10] S. Bokhari and J. Saltz, "Exploring the Performance of Massively Multithreaded Architectures," *Concurrency and Computation: Practice & Experience*, vol. 22, no. 5, pp. 588–616, April 2010.

[11] W.-c. Feng and K. Cameron, "The Green500 List: Encouraging Sustainable Supercomputing," *IEEE Computer*, vol. 40, no. 12, pp. 38–44, December 2007.

[12] I. Ahmad and S. Ranka, *Handbook of Energy-Aware And Green Computing*. Taylor and Francis Group, CRC Press, 2011.

[13] D. Milojicic, "Trend Wars: Embedded Systems," *IEEE Concurrency*, vol. 8, no. 4, pp. 80–90, October-December 2000.

[14] G. Kornaros, *Multi-core Embedded Systems*. Taylor and Francis Group, CRC Press, 2010.

[15] C. Gonzales and H. Wang, "White Paper: Thermal Design Considerations for Embedded Applications," June 2011. [Online]. Available: http://download.intel.com/design/intarch/papers/321055.pdf

[16] J. C. Knight, *Software Challenges in Aviation Systems*. Springer Berlin/Heidelberg, 2002.

[17] R. Kumar, D. Tullsen, P. Ranganathan, N. Jouppi, and K. Farkas, "Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance," in *Proc. of IEEE ISCA*, Munich, Germany, June 2004.

[18] R. Kumar, D. Tullsen, and N. Jouppi, "Core Architecture Optimization for Heterogeneous Chip Multiprocessors," in *Proc. of ACM International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Seattle, Washington, September 2006.

[19] R. Kumar, N. Jouppi, and D. Tullsen, "Conjoined-core Chip Multiprocessing," in *Proc. of IEEE/ACM MICRO-37*, Portland, Oregon, December 2004.

[20] S. Keckler, K. Olukotun, and H. Hofstee, *Multicore Processors and Systems*. Springer, 2009.

[21] K. Puttaswamy and G. Loh, "Thermal Herding: Microarchitecture Techniques for Controlling Hotspots in High-Performance 3D-Integrated Processors," in *Proc. of IEEE HPCA*, Phoenix, Arizona, February 2007.

[22] P. Pande, A. Ganguly, B. Belzer, A. Nojeh, and A. Ivanov, "Novel Interconnect Infrastructures for Massive Multicore Chips - An Overview," in *Proc. of IEEE ISCAS*, Seattle, Washington, May 2008.

[23] S. Narayanan, J. Sartori, R. Kumar, and D. Jones, "Scalable Stochastic Processors," in *Proc. of IEEE/ACM DATE*, Dresden, Germany, March 2010.

[24] M. Hill, "Transactional Memory," in *Synthesis Lectures on Computer Architecture*, June 2010. [Online]. Available: http://www.morganclaypool.com/toc/cac/1/1

[25] N. Guan, M. Stigge, W. Yi, and G. Yu, "Cache-Aware Scheduling and Analysis for Multicores," in *Proc. of ACM EMSOFT*, Grenoble, France, October 2009.

[26] S. Fide, *Architectural Optimizations in Multi-Core Processors*. VDM Verlag, 2008.

[27] J. Chang and G. Sohi, "Cooperative Caching for Chip Multiprocessors," in *Proc. of ACM ISCA*, Boston, Massachusetts, May 2006.

[28] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy Caches: Simple Techniques for Reducing Leakage Power," in *Proc. of IEEE/ACM ISCA*, Anchorage, Alaska, May 2002.

[29] S.-B. Lee, S.-W. Tam, I. Pefkianakis, S. L. Lu, M. Chang, C. Guo, G. Reinman, C. Peng, M. Naik, L. Zhang, and J. Cong, "A Scalable Micro Wireless Interconnect Structure for CMPs," in *Proc. of ACM MobiCom*, Beijing, China, September 2009.

[30] A. Shacham, K. Bergman, and L. Carloni, "Photonic Networks-on-Chip for Future Generations of Chip Multiprocessors," *IEEE Trans. on Computers*, vol. 57, no. 9, pp. 1246–1260, September 2008.

[31] P. Pande, A. Ganguly, K. Chang, and C. Teuscher, "Hybrid Wireless Network on Chip: A New Paradigm in Multi-Core Design," in *Proc. of IEEE NoCArc*, New York, New York, December 2009.

[32] V. Kontorinis, A. Shayan, D. Tullsen, and R. Kumar, "Reducing Peak Power with a Table-Driven Adaptive Processor Core," in *Proc. of IEEE/ACM MICRO-42*, New York, New York, December 2009.

[33] J. Donald and M. Martonosi, "Techniques for Multicore Thermal Management: Classification and New Exploration," in *Proc. of IEEE ISCA*, Boston, Massachusetts, June 2006.

[34] R. Jayaseelan and T. Mitra, "A Hybrid Local-Global Approach for Multi-Core Thermal Management," in *Proc. of IEEE/ACM ICCAD*, San Jose, California, November 2009.

[35] J. Park, D. Shin, N. Chang, and M. Pedram, "Accurate Modeling and Calculation of Delay and Energy Overheads of Dynamic Voltage Scaling in Modern High-Performance Microprocessors," in *Proc. of ACM/IEEE ISLPED*, Austin, Texas, August 2010.

[36] ACPI, "Advanced Configuration and Power Interface," June 2011. [Online]. Available: http://www.acpi.info/

[37] J. Lee and N. Kim, "Optimizing Throughput of Power- and Thermal-Constrained Multicore Processors Using DVFS and Per-Core Power-Gating," in *Proc. of IEEE/ACM DAC*, San Francisco, California, July 2009.

[38] Freescale, "Green Embedded Computing and the MPC8536E PowerQUICC III Processor," 2009. [Online]. Available: http://www.freescale.com/files/32bit/doc/white_paper/MPC8536EWP.pdf

[39] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. Cameron, "PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications," *IEEE Trans. on Parallel and Distributed Systems*, vol. 21, no. 5, pp. 658–671, May 2010.

[40] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Power-Aware Computing with Dynamic Knobs," *MIT Technical Report: Computer Science and Artificial Intelligence Laboratory (MIT-CSAIL-TR-2010-027)*, May 2010.

[41] W. Baek and T. Chilimbi, "Green: A Framework for Supporting Energy-Conscious Programming using Controlled Approximation," in *Proc. of ACM SIGPLAN PLDI*, Toronto, Ontario, Canada, June 2010.

[42] X. Zhou, J. Yang, M. Chrobak, and Y. Zhang, "Performance-aware Thermal Management via Task Scheduling," *ACM Trans. on Architecture and Code Optimization (TACO)*, vol. 7, no. 1, pp. 5:1–5:31, April 2010.

[43] A. Jacobs, A. George, and G. Cieslewski, "Reconfigurable Fault Tolerance: A Framework for Environmentally Adaptive Fault Mitigation in Space," in *Proc. of IEEE FPL*, Prague, Czech Republic, August-September 2009.

[44] J. Samson, J. Ramos, A. George, M. Patel, and R. Some, "Technology Validation: NMP ST8 Dependable Multiprocessor Project," in *Proc. of IEEE Aerospace Conference*, Big Sky, Montana, March 2006.

[45] CHREC, "NSF Center for High-Performance Reconfigurable Computing," June 2011. [Online]. Available: http://www.chrec.org/

[46] J. Sloan and R. Kumar, "Towards Scalable Reliability Frameworks for Error Prone CMPs," in *Proc. of ACM CASES*, Grenoble, France, October 2009.

[47] D. Poulsen and P.-C. Yew, "Data Prefetching and Data Forwarding in Shared Memory Multiprocessors," in *Proc. of IEEE ICPP*, North Carolina State University, North Carolina, August 1994.

[48] L. Yan, W. Hu, T. Chen, and Z. Huang, "Hardware Assistant Scheduling for Synergistic Core Tasks on Embedded Heterogeneous Multi-core System," *Journal of Information & Computational Science*, vol. 5, no. 6, pp. 2369–2373, 2008.

[49] P. Chaparro, J. Gonzalez, G. Magklis, Q. Cai, and A. Gonzalez, "Understanding the Thermal Implications of Multicore Architectures," *IEEE Trans. on Parallel and Distributed Systems*, vol. 18, no. 8, pp. 1055–1065, August 2007.

[50] G. Suo and X.-j. Yang, "Balancing Parallel Applications on Multicore Processors Based on Cache Partitioning," in *Proc. of IEEE ISPA*, Chendu and JiuZhai Valley, China, August 2009.

[51] H. Jeon, W. Lee, and S. Chung, "Load Unbalancing Strategy for Multi-Core Embedded Processors," *IEEE Trans. on Computers*, vol. 59, no. 10, pp. 1434–1440, October 2010.

[52] TILERA, "Manycore without Boundaries: TILEPro64 Processor," June 2011. [Online]. Available: http://www.tilera.com/products/processors/TILEPRO64

[53] ——, "Manycore without Boundaries: TILE-Gx Processor Family," June 2011. [Online]. Available: http://www.tilera.com/products/processors/TILE-Gx_Family

[54] Intel, "High-Performance Energy-Efficient Processors for Embedded Market Segments," June 2011. [Online]. Available: http://www.intel.com/design/embedded/downloads/315336.pdf

[55] ——, "Intel Core 2 Duo Processor Maximizing Dual-Core Performance Efficiency," June 2011. [Online]. Available: ftp://download.intel.com/products/processor/core2duo/mobile_prod_brief.pdf

[56] ——, "Dual-Core Intel Xeon Processors LV and ULV for Embedded Computing," June 2011. [Online]. Available: ftp://download.intel.com/design/intarch/prodbref/31578602.pdf

[57] ——, "Intel Xeon Processor LV 5148," June 2011. [Online]. Available: http://ark.intel.com/Product.aspx?id=27223

[58] NVIDIA, "NVIDIA Tesla C1060 Computing Processor," June 2011. [Online]. Available: http://www.nvidia.com/object/product_tesla_c1060_us.html

[59] ——, "NVIDIA Tesla Personal Supercomputer," June 2011. [Online]. Available: http://www.nvidia.com/docs/IO/43395/NV_DS_Tesla_PSC_US_Mar09_LowRes.pdf

[60] ——, "NVIDIA PowerMizer Technology," June 2011. [Online]. Available: http://www.nvidia.com/object/feature_powermizer.html

[61] ——, "NVIDIA Tesla C2050/C2070 GPU Computing Processor," June 2011. [Online]. Available: http://www.nvidia.com/object/product_tesla_C2050_C2070_us.html

[62] T. Berg, "Maintaining I/O Data Coherence in Embedded Multicore Systems," *IEEE MICRO*, vol. 29, no. 3, pp. 10–19, May/June 2009.

[63] G. Bournoutian and A. Orailoglu, "Miss Reduction in Embedded Processors through Dynamic, Power-Friendly Cache Design," in *Proc. of IEEE/ACM DAC*, Anaheim, California, June 2008.

[64] SeaMicro, "The SM10000 Family," June 2011. [Online]. Available: http://www.seamicro.com/

[65] AMAX, "High Performance Computing: ClusterMax SuperG Tesla GPGPU HPC Solutions," June 2011. [Online]. Available: http://www.amax.com/hpc/productdetail.asp?product_id=superg

[66] P. Koka, M. McCracken, H. Schwetman, X. Zheng, R. Ho, and A. Krishnamoorthy, "Silicon-photonic Network Architectures for Scalable, Power-efficient Multi-chip systems," in *Proc. of ACM/IEEE ISCA*, Saint-Malo, France, June 2010.

[67] M. Asghari and A. Krishnamoorthy, "Silicon Photonics: Energy-efficient Communication," *Nature Photonics*, pp. 268–270, May 2011.

[68] C.-W. Lee, S.-R.-N. Yun, C.-G. Yu, J.-T. Park, and J.-P. Colinge, "Device Design Guidelines for Nano-scale MuGFETs," *Elsevier Solid-State Electronics*, vol. 51, no. 3, pp. 505–510, March 2007.

[69] S. Collange, D. Defour, and A. Tisserand, "Power Consumption of GPUs from a Software Perspective," in *Proc. of ACM ICCS*, Baton Rouge, Louisiana, May 2009.

Arslan Munir received his B.S. in Electrical Engineering from the University of Engineering and Technology (UET), Lahore, Pakistan, in 2004, and his M.A.Sc. degree in Electrical and Computer Engineering (ECE) from the University of British Columbia (UBC), Vancouver, Canada, in 2007. He is currently working towards his Ph.D. degree in ECE at the University of Florida (UF), Gainesville, Florida, USA. From 2007 to 2008, he worked as a software development engineer at Mentor Graphics in the Embedded Systems Division. He was the recipient of many academic awards including the Gold Medals for the best performance in Electrical Engineering and academic Roll of Honor. He received a Best Paper award at the IARIA International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM) in 2010. His current research interests include embedded systems, low-power design, computer architecture, multi-core platforms, parallel computing, dynamic optimizations, fault-tolerance, and computer networks.

Sanjay Ranka is a Professor in the Department of Computer and Information Science and Engineering at the University of Florida, Gainesville, Florida, USA. His current research interests are energy-efficient computing, high-performance computing, data mining and informatics. Most recently he was the Chief Technology Officer at Paramark where he developed real-time optimization software for optimizing marketing campaigns. Sanjay has also held positions as a tenured faculty positions at Syracuse University and as a researcher/visitor at IBM T.J. Watson Research Labs and Hitachi America Limited.

Sanjay earned his Ph.D. (Computer Science) from the University of Minnesota and a B. Tech. in Computer Science from IIT, Kanpur, India. He has coauthored two books: Elements of Neural Networks (MIT Press) and Hypercube Algorithms (Springer Verlag), 75 journal articles and 125 refereed conference articles. His recent work has received a student best paper award at ACM-BCB 2010, best paper runner up award at KDD-2009, a nomination for the Robbins Prize for the best paper in the journal of Physics in Medicine and Biology for 2008, and a best paper award at ICN 2007.

He is a fellow of the IEEE and AAAS, and a member of IFIP Committee on System Modeling and Optimization. He is the associate Editor-in-Chief of the Journal of Parallel and Distributed Computing and an associate editor for IEEE Transactions on Parallel and Distributed Computing, IEEE Transactions on Computers, Sustainable Computing: Systems and Informatics, Knowledge and Information Systems, and International Journal of Computing.

Ann Gordon-Ross received her B.S and Ph.D. degrees in Computer Science and Engineering from the University of California, Riverside (USA) in 2000 and 2007, respectively. She is currently an Assistant Professor of Electrical and Computer Engineering at the University of Florida (USA) and is a member of the NSF Center for High Performance Reconfigurable Computing (CHREC) at the University of Florida. She is also the faculty advisor for the Women in Electrical and Computer Engineering (WECE) and the Phi Sigma Rho National Society for Women in Engineering and Engineering Technology. She received her CAREER award from the National Science Foundation in 2010 and Best Paper awards at the Great Lakes Symposium on VLSI (GLSVLSI) in 2010 and the IARIA International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM) in 2010. Her research interests include embedded systems, computer architecture, low-power design, reconfigurable computing, dynamic optimizations, hardware design, real-time systems, and multi-core platforms.