ReDEEM: A Heterogeneous Distributed Microarchitecture for Energy-Efficient Reliability

***Abstract*—Diminishing energy-efficiency returns and decreasing transistor reliability are casting shadows on semiconductor scaling. Prior research has been addressing processors' energy-efficiency and transistor reliability as orthogonal problems. However, as embedded processors get more powerful and find their way into more diverse applications, both high reliability and energy-efficiency become critical. In this work, we propose ReDEEM, a novel approach to design energy-efficient and reliable miroarchitectures. Our proposed solution composes processor pipelines at runtime from redundant but heterogeneous pipeline components. Our pipeline components are loosely coupled and the control logic is decentralized so as to enable fault isolation and thereby eliminate single points of failure. We equip the microarchitecture with the ability to adapt dynamically to varying application phases by constructing energy-efficient pipelines best suited for each phase. In addition, pipeline components have power management capabilities that allow for greater energy efficiency and flexibility. Our experimental evaluation shows that our solution offers up to 60% in energy savings and can operate about 1.8x longer, when subjected to the same fault rate as a state-of-the-art reliable microarchitecture.**

I. Introduction

Diminishing returns from transistor scaling have been impacting microprocessor design trends [4]. The exponential increases in leakage power resulting from scaling have been dwarfing the performance benefits of deploying smaller transistors. Hence, in order to operate within their limited power budgets, modern processor designs can not utilize all transistors available simultaneously. In addition, the smaller transistors become, the less reliable they get, shortening the expected lifetimes of our designs. The increased power dissipation that we are experiencing due to scaling exacerbates the reliability problem further: transistors that dissipate more power tend to break down sooner.

Today, embedded electronic systems are becoming increasingly widespread. A majority of them are deployed in applications that require them to operate reliably for years while expending as little energy as possible. Automotive electronics, medical implants, satellite electronics and sensor systems in large public infrastructure (buildings, bridges, *etc.*) are all examples of systems with high requirements for both reliability and energy-efficiency. In the future, we can only expect that the demand for more computing power, better reliability and high energy-efficiency will increase, as embedded system deployments pervade our lives further and further. The processors we design for future embedded systems need to meet these demands while addressing the fragility and inefficiency of future technology nodes.

A few techniques have been proposed to design highly reliable processors. These solutions break the tight coupling within the microarchitectural components in a processor pipeline and provide redundancy at a much finer granularity [7], [18], [19]. However, the overheads introduced by these approaches significantly limit their applicability in energy-constrained environments. A few other researchers have been crafting solutions that tune the energy consumption of their processors to the demands of the applications they execute [9], [12]; however, they don't directly address reliability.

In this work, we specifically address this challenge by proposing a microarchitecture that provides both reliability and energy efficiency. We achieve this goal by leveraging a distributed control subsystem in the processor and heterogeneity in the implementation of its redundant components. The distributed control system and the redundant components enhance the reliability of our system by eliminating single-points-of-failure, while its heterogeneity enables our system to dynamically fine-tune its energy consumption to the demands of the applications it executes.

Contributions: In this work, we propose ReDEEM (**Re**liable, **D**istributed, **E**nergy-**E**fficient **M**icroarchitecture) – a novel microarchitecture for incorporating redundancy, heterogeneity and runtime energy management in a highly decoupled design, with the intent of delivering energy-efficient reliability. ReDEEM employs a distributed design where microarchitectural components are distributed and connected via a flexible interconnect. ReDEEM incorporates multiple instances of each component, which we refer to as "execution resources". While execution resources can embody any microarchitectural component, in this work execution resources correspond to pipeline stages (*e.g.*: fetch unit, decode unit, *etc.*). Our approach is new in that we make available multiple variants of an execution resource for each pipeline stage. Each variant differs in performance and energy characteristics. We generate the diversity in our variants by exploiting the performance and energy diversity in standard cell libraries. In addition, each execution resource is augmented with i) a fault manager that can periodically monitor and detect hardware failures and ii) an energy manager that controls the power state, operating frequency and voltage of the execution resource. ReDEEM composes execution pipelines at runtime from its pool of fault-free execution resources. By composing pipelines with

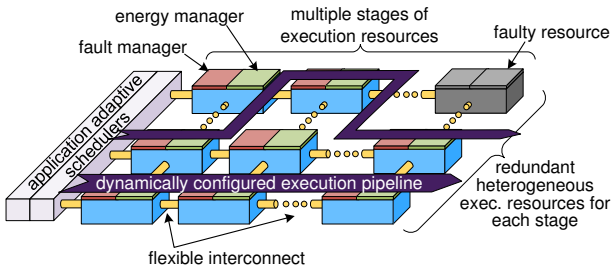


Fig. 1: **ReDEEM overview.** Several heterogeneous pipeline stages (*e.g.* fetch, decode, *etc.*) are connected via a flexible interconnect. Fault managers independently detect and disable faulty execution resources. Application-adaptive schedulers dynamically set up pipelines that meet the performance goals of each application, while minimizing energy and also routing around faulty resources. In collaboration with the scheduling units, energy managers control the sleep state and operating frequency of their execution resource.

execution resources that have certain performance and energy characteristics, ReDEEM can tune the characteristics of its execution pipelines. It can thus adapt efficiently to the needs of the workloads it executes.

II. System Overview

The high-level overview for our microarchitecture is shown in Figure 1. ReDEEM is a distributed microarchitecture with decentralized control that decouples and replicates a pipeline's scheduling and control logic within a single core. ReDEEM is inspired by previously-proposed distributed microarchitectures, such as StageNet [7], CCA [19] and Viper [18]. ReDEEM constructs execution pipelines from a distributed pool of execution resources, which are typically made of pipeline stages (fetch, decode, *etc.*). Its distributed scheduling units organize available execution resources into dynamic pipelines to execute an application's instructions. Each execution resource is augmented with a fault manager that can detect failures and disable the execution resource. Since only the active, fault-free resources advertise their availability to the scheduling units, ReDEEM's pipeline-construction logic avoids the faulty components naturally. In the absence of faults, ReDEEM can construct multiple pipelines to execute several applications in parallel. This approach is particularly useful for multi-threaded and multi-programmed workloads. As faults accumulate, ReDEEM keeps functioning as long as there are enough resources to construct at least one execution pipeline.

Unlike previous distributed microarchitectures, the execution resources for each of ReDEEM's pipeline stages are heterogeneous. Our novel approach for deploying this fine-grained heterogeneity into a core utilizes microarchitecturally identical execution resources for each pipeline stage, synthesized for different performance and power goals. We provide diversity in the implementation of our execution resources by targeting multiple transistor types and voltage/frequency (VF) operating points. For instance, fast transistor circuits operating at higher frequency and lower threshold voltage can be used for high-performance execution resources, while slower circuits that consume less power can be used for low-power execution resources. ReDEEM's pipeline construction mechanism can select any execution resource among the set of functionally equivalent resources for a given pipeline stage. This flexibility enables ReDEEM to construct execution pipelines that are tailored to an application's needs. Previous works on heterogeneous multicores and networks-on-chip (NoCs) have leveraged this type of performance-power heterogeneity to implement coarse-grained, energy-efficient systems that adapt to workload characteristics [5], [3], reporting up to 56% better energy efficiency than state-of-the-art dynamic voltage and frequency scaling (DVFS) approaches. While this work focuses on heterogeneity achieved through diversified synthesis of architecturally identical hardware blocks, it would be straightforward to also deploy architectural heterogeneity within the ReDEEM substrate. For instance, a fast adder design can be used in a high-performance execution resource whereas a slower one can be used in a lower-performance resource. A ReDEEM processor with architectural diversity alone would in fact be simpler as it would not require some of the energy management and resource selection features we propose; it would allow full flexibility in composing pipelines and simply rely on our performance monitors to gather information on the specific performance of each resource.

In addition to this implementation diversity, ReDEEM's execution resources incorporate a low-overhead energy manager to further extract static energy savings and enforce common-frequency pipelines. When an energy manager detects that its execution resource is idle, it sends it to sleep. The state of the execution resource is communicated to the scheduling units, which can then avoid including sleeping resources in the pipelines they configure. If a sleeping resource must be included in a pipeline, then the energy manager and scheduling unit bring it back up while hiding the wake-up latency in the pipeline configuration and execution latency. Moreover, in presence of faults, a pipeline may need to be composed of execution resources optimized for a mix of different operating frequencies. In this case, the energy managers are responsible for adjusting frequency appropriately so as to obtain a common operating frequency for the entire pipeline.

III. Microarchitecture design

The major microarchitectural components in ReDEEM are the execution resources, scheduling units and the interconnect fabric, which are organized as in Figure 2. In this Section, we describe each major component in detail.

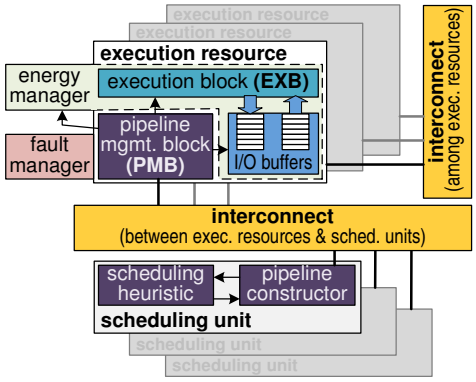


Fig. 2: **Organization of major components.** Execution resources and scheduling units communicate via fault-tolerant interconnects. Execution resources are enhanced with energy and fault managers to tolerate faults and offer energy-efficient execution.

A. Execution resources

Each execution resource in ReDEEM is organized as shown in top half of Figure 2. The bulk of an execution resource comprises two subunits: a pipeline management block (PMB) and an execution block (EXB). The PMB interfaces with the scheduling units and other resources to establish dynamic pipelines, control the data transfers to/from other resources and manage the state of the resource itself.

An execution resource that has just been included in a pipeline initially awaits data from the resources in earlier pipeline stages. Once the data becomes available in its input buffer, the EXB processes it and makes it available in the output buffer, so that it can be transferred to the next pipeline stage. Under the control of the PMB, an energy manager manages the voltage/frequency (VF) state of the EXB and the I/O buffers. The details of the energy management mechanism

are discussed in Section IV-C. A fault manager monitors the execution resource for permanent faults and disables it entirely if it is found to be faulty. The detection of permanent faults is a well-researched topic [10] and is not addressed in this work. Note that the PMB of a disabled execution resource stops interacting with the scheduling units and other resources.

B. Scheduling units

ReDEEM incorporates multiple scheduling units, each responsible for constructing and managing a pipeline to execute a sequential block of instructions. The organization of a scheduling unit is illustrated in the bottom half of Figure 2. Periodic messages from execution resources inform a scheduling unit about the state and capability of the execution resources. The pipeline construction algorithm implemented in a scheduling unit recruits execution resources to form a complete execution pipeline. The decision to include certain execution resources in a pipeline is influenced by a scheduling heuristic that attempts to optimize for energy efficiency. The heuristic relies on past application behavior to predict future performance and energy consumption. To this end, the scheduling unit collects performance statistics (*e.g.*, stall cycles and number of cache misses) from the execution resources in its pipeline and earlier scheduling units. The pipeline formation process is discussed in detail in Section IV-C.

C. Interconnect fabric

ReDEEM deploys a robust interconnect fabric to connect execution resources and schedulers. We segment this fabric into two parts, with different requirements: i) for high reliability, the connection among execution resources needs to offer multiple paths between execution resources for consecutive pipeline stages and ii) for fast pipeline formation and management, a low latency connection is required among scheduling units and between execution resources and scheduling units. A mesh network topology for the former and a crossbar for the latter, as utilized in [18], can meet these requirements. However, ReDEEM's execution mechanism is independent of the chosen network topology.

IV. Energy efficiency and reliability

A. Diversifying execution resources with heterogeneity

Modern circuit synthesis flows optimize designs for performance and power goals through the selection of various parameters, such as transistor size and threshold voltage [22]. We leverage this property to generate redundant heterogeneous execution resources for each stage. Chakraborty and Roy [5] observe significant energy savings achievable through this static optimization approach over a dynamic voltage and frequency adjustment. They report up to 86% better energy efficiency over DVFS for an arithmetic and logic unit (ALU) synthesized at 45nm.

A finite set of VF operating points are chosen as synthesis targets for all execution resources. While a large set of VF operating points offers better diversity, it comes at the cost of extra complexity in the voltage and frequency management circuitry. For each pipeline stage, multiple execution resources are synthesized with different VF targets. Each execution resource is equipped with an energy manager that can dynamically scale its operating frequency down to any of the lower frequencies in the chosen set. To reduce design complexity, this heterogeneity is introduced only to a resource's EXB and I/O buffers; the PMB remains homogeneous. In addition, due to the negative effects of higher operating frequencies on transistor lifetimes, the heterogeneity we introduce also impacts ReDEEM's reliability characteristics (see Section VI).

B. Power-gating idle resources

Redundant execution resources that are not part of an active pipeline leak static energy without doing useful work. In ReDEEM, the energy managers of the execution resources, along with the scheduling mechanisms, enable the power-gating of idle execution resources. Specifically, the EXB, I/O buffers and related portions of the clock distribution networks are enhanced with power-gating capability.

Initially, an execution resource is in an idle state where the EXB is awake and the PMB advertises its availability to the scheduling units. If the execution resource is not included in a pipeline after a fixed number of tries, the PMB makes the decision to send its EXB to sleep. A sleeping execution resource must remain inactive for about 10 to 30 cycles to recover the energy costs associated with power-gating. To this end, the execution resource transitions into an inactive state for a period of time that accounts for the power-gating latency and the time required to recover the energy costs of power-gating. While in this state, the EXB is asleep and the PMB suspends the advertisement of the resource to the scheduling units. At the end of this inactive period, the execution resource transitions into a low-leakage waiting state, where the PMB notifies the scheduling units of its availability but the EXB remains asleep. If the execution resource is selected to be included in a pipeline during this period, the EXB is awakened and execution resumes. For execution resources deep in the pipeline, the wakeup time can be overlapped with the time a block of instructions takes to execute in the preceding stages of the pipeline. Note that while selecting execution resources to include in a pipeline, a scheduling unit always gives lower priority to sleeping resources.

C. Dynamic execution pipelines

ReDEEM executes instructions in groups. We adopt a sequential block grouping of instructions, similar to "bundles" in Viper [18]. Since the sizes of these blocks determine how big the input and output buffers at each execution resource need to be, we limit a block's size to 16 instructions. When an application is first loaded for execution, the system assigns its first block to one of the available scheduling units. The scheduling unit collects advertisements from available execution resources and forms the highest-performance execution pipeline that is feasible. Upon finishing its task, each execution resource in the pipeline sends a completion message to the scheduling unit. The completion message includes information about the application's performance and any other relevant information. The execution resource that fetches instructions, for instance, informs the scheduling unit of the next address to be fetched for a new block of instructions.

In addition to building and managing a pipeline for the current block of instructions, a scheduling unit must also

assign another scheduling unit to manage the execution of the next block of instructions. This assignment occurs once the scheduling unit obtains the new block address from its fetch stage. Similar to execution resources, scheduling units advertise their availability to other scheduling units. Unlike execution resources though, they do not differ in performance or capability. Therefore, the first available scheduling unit can be assigned immediately. Scheduling units are unavailable for a new assignment for the entire duration of the execution of their assigned block of instructions.

A newly assigned scheduling unit can either decide to build a new execution pipeline or inherit the same pipeline from the older scheduling unit. The former option is chosen upon detection of a new fault in the pipeline that could have been inherited, or if the scheduling unit detects that the application is changing its activity profile – we call each portion where an application performs activities exhibiting a similar set of performance characteristics a "phase". When composing a new pipeline, a scheduling unit attempts to select resources that best match the predicted performance of the application – as indicated by the performance monitors of the prior scheduling unit. The scheduling unit always prefers to compose a from execution resources designed to operate at the same frequency. When the system is fault-free, it is always possible to attain this goal. However, as faults accumulate, a scheduling unit may be forced to construct a pipeline from execution resources synthesized for different operating frequencies. In such cases, some of the execution resources will be dynamically scaled down to operate at a common frequency in the pipeline.

In addition, a scheduling unit takes into account the sleep state and proximity of the execution resources. The number of hops between execution resources for two consecutive pipeline stages impacts the overall execution performance, while the longer an execution resource stays in a sleep state, the more energy savings can be obtained. Finally, by considering only execution resources that have advertised their availability, the scheduling unit naturally avoids faulty execution resources and those that are still recovering their power-gating costs. If enough execution resources are available, each of ReDEEM's scheduling units can build and manage a pipeline independent of other active pipelines. Thus, ReDEEM can support multiple active pipelines that execute instruction blocks from multiple threads concurrently. Many scheduling algorithms are well suited for ReDEEM, such as those summarized in [23], but we have left the specific design decisions for future work.

Figure 3 illustrates an interaction scenario between scheduling units and execution resources during the lifetime of a dynamic pipeline. Initially, scheduling unit-1 is assigned to execute block-1. Idle execution resources advertise their availability to all scheduling units. Based on its configuration heuristic, scheduling unit-1 composes a pipeline and starts the execution of block-1. Note that scheduling unit-1 assigns the new block-2 to scheduling unit-2 after receiving the block's address from the fetch resource in its pipeline. It also transfers the performance statistics for block-1 to scheduling unit-2 when block-1 finishes executing. Execution resource-1 is included in the pipeline, and thus performs its task as soon as the data from earlier stages becomes available. Execution resource-2, on the other hand, is not included in a pipeline and decides to go to sleep. It first transitions to an inactive state where it suspends its advertisements. It stays in this state long enough to recover the power-gating costs and then transitions to an idle state where the EXB is asleep and the PMB resumes sending advertisements. When it is finally included in a pipeline, it proceeds to wake itself up and resume execution.

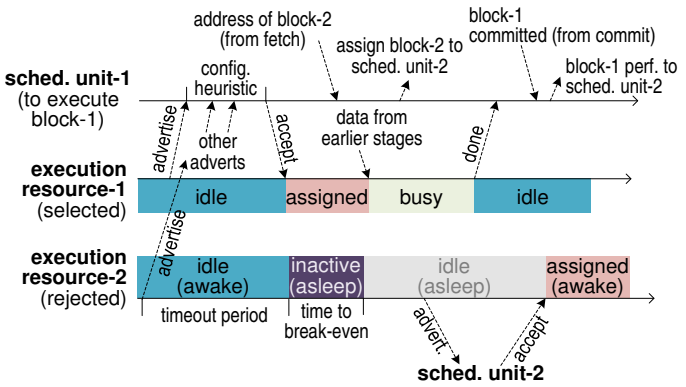


Fig. 3: **Interaction between execution resources and assigned scheduling units.** Execution resources and scheduling units collaborate to ensure energy-efficient (and fault-tolerant) execution using an advertisement-based process for available resources.

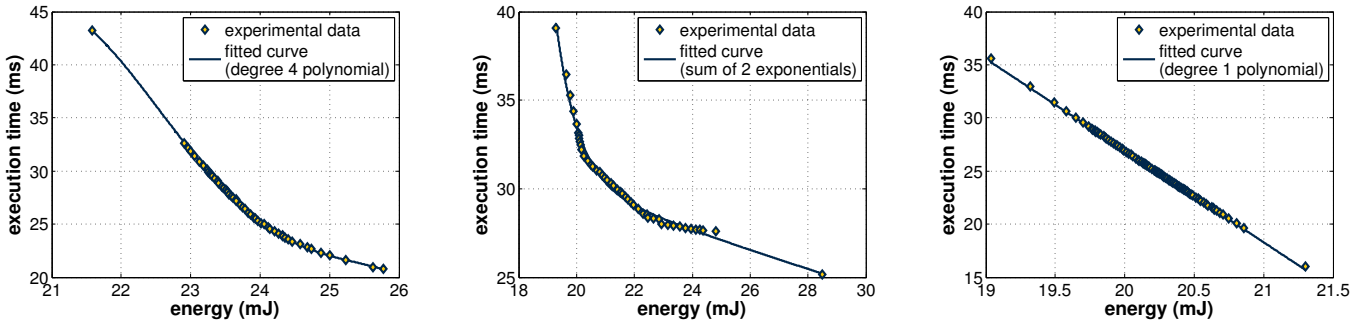
V. Energy Efficiency Experiments

We performed experiments to investigate the range of performance/energy operating points that ReDEEM can achieve in the absence of faults. We consider an ideal scheduler that can always select the pipeline, for the next execution block, that leads to one of the pareto-optimal performance/energy points for the application. To this end, we built a detailed timing simulation model for a 5-stage decentralized microarchitecture using gem5 [2] and a power model using McPAT [14]. Each stage in our design had 4 redundant heteregeneous execution resources modeled with the frequency and device type combinations shown in Table I, all at the 22nm technology node, for a total of 20 resources in the system. We used SimPoint profiling [20] to select 10,000,000 instructions each, from several SPEC CPU2006 benchmarks and we allowed our model to only consider a new pipeline formation every 1,000 instructions (equivalent to executing a few hundreds of blocks). Because we operate in a fault-free environment, the choice, when forming a new pipeline, is among four different options, one for each of the operating points in Table I.

Our implementation of the ideal scheduling algorithm operates offline: it considers the performance/energy statistics collected during the execution of each window of 1,000 instructions for all four distinct pipelines; it then chooses one pipeline for each window of instructions so as to obtain a pareto-optimal execution. Figure 4 plots our results from this evaluation for several SPEC benchmarks. In order to limit the amount of computation required for our evaluation, we rely on a similar approach as in [15].

Name	Frequency (MHz)	Device Type
2000-hp	2,000	high-performance
1500-hp	1,500	high-performance
1200-hp	1,200	high-performance
900-lstp	900	low standby power

TABLE I: **Target frequencies and device types**. We modeled 4 types of heterogeneous exeuction resources for our studies.



(a) bzip2 (b) gobmk (c) libquantum

Fig. 4: **Fault-free performance/energy pareto frontiers for SPEC CPU2006 benchmarks**. The figures plot pareto-optimal performance/energy points for a number of SPEC CPU2006 benchmarks running on ReDEEM, in the absence of faults. ReDEEM makes a wide range of operating points possible that, in the best case (libquantum), trade off as high as a 32% energy savings for a 1.55x slowdown.

For the 11 SPEC SPU2006 programs we investigated, we observe several operating points enabling a range of energy savings with corresponding sacrifices in raw performance. On average, we observe an 18% range in energy savings and a 2x range in performance. The *gobmk* benchmark exhibits a best-case scenario of operating points with up to 32% energy reduction and up to 1.55x slowdown while *libquantum* exhibits the worst-case scenario of only up to 11% energy savings for at most 2.22x slowdown. The range of available operating points is expected to increase as the number and variety of execution units increase. In addition, a non-ideal hardware scheduling algorithm may generate operating points that are not on the pareto frontier. Finally, in Table II, we present the total energy savings, averaged across all benchmarks, obtained by ReDEEM, over four equivalent homogeneous configurations. For ReDEEM, we report the lowest-energy execution that takes the same amount of time as the given homogeneous execution. Leakage energy in the homogeneous systems accounts for most of the savings we report.

homogeneous technology	**total energy (mJ)**		**savings(%)**
	homogeneous	**ReDEEM**	
2000-hp	68.02	28.03	58.79
1500-hp	84.67	33.50	60.43
1200-hp	101.54	39.05	61.54
900-lstp	23.32	22.86	1.98

TABLE II: **Energy savings.** Compared to a homogeneous decentralized microarchitecture of similar capability, our solution offers significant energy savings for fault-free single-threaded execution.

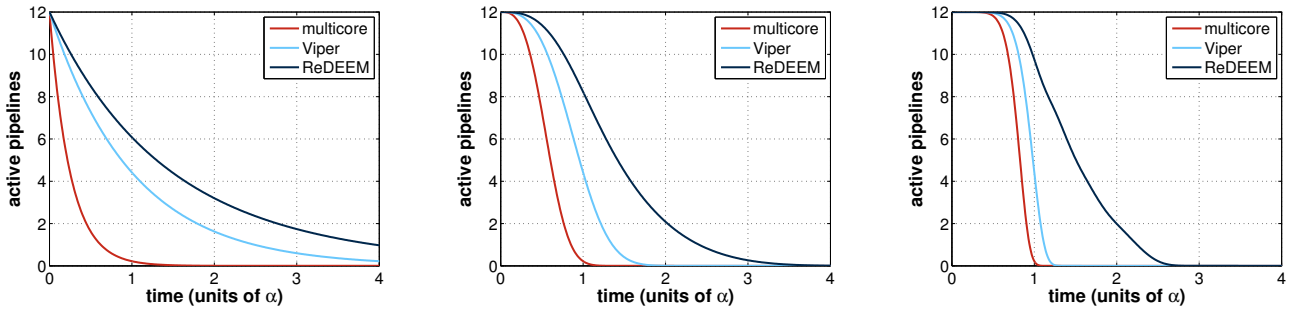
VI. Reliability Analysis

In this section, we study the reliability of ReDEEM compared to a previous decentralized microarchitecture (Viper [18]) and a homogeneous multicore system. Based on area estimates from Viper, we assume that a decentralized reliable design and a multicore one of the same area can sustain an equal number of pipelines during fault free operation. We therefore model ReDEEM and Viper to comprise 5 stages of 12 redundant execution resources while our multicore comprises 12, one-way, 5-stage pipelined cores. Both the multicore and Viper are assumed to operate at 2GHz, while the operating frequencies of ReDEEM's redundant resources are uniformly distributed among those shown in Table I. We build technology-independent, comparative statistical models of wearout-induced failure rates for all execution resources based on the Weibull distribution [21]. For all systems, we define random variables to estimate the number of active pipelines that can be available at any given time. While deriving these *Availability* random variables, we keep the Weibull shape parameter (β) constant for all execution resources and vary the scale parameter (α) inversely with frequency to model the effect of operating frequency on lifetime.

In Figure 5, we present the expected values of Availability for the three systems, each with the equivalent area of a 12-core multicore system. Since the Weibull shape parameter (β) is technology specific and usually reported for one type of failure mechanism, we demonstrate our results over a range of values for β. A value of $\beta = 1$ corresponds to a constant failure rate, which is typically used to model random failures during the expected lifetime of a device. The wearout-induced failures we are interested in are characterized by values of $\beta > 1$. Kauerauf, *et al.* [11] report values of β as high as 8 for time-dependent dielectric breakdown of SiO_2. For all values of β we consider, as time progresses, faults accumulate resulting in a decrease in the number of active pipelines that can be sustained. We observe that ReDEEM offers better availability than both a multicore system and a homogeneous decentralized microarchitecture. For $\beta = 3$, ReDEEM sustains at least one active pipeline for a duration 2.8x that of a multicore system and 1.8x that of Viper. The improvement relative to the multicore is due to the tight coupling in a conventional core that causes an entire core to be decommissioned even when only one pipeline stage fails. The improvement relative to Viper is due to the reduced lifetime of its execution resources, which operate at a uniformly high frequency.

VII. Related Work

Distributed microarchitectures. Researchers have proposed several microarchitectures incorporating execution resources that can dynamically be composed to execute applications. CCA [19], StageNet [7], Viper [18] utilize a distributed design for improved reliability. CoreGenesis [8] enhances a reliable microarchitecture [7] to offer improved efficiency via adaptive pipeline widths. However, it lacks design- and run-time fine-grained energy management techniques for tackling energy consumption. Researchers have also proposed decoupled multicores such as MorphCore [12] and Core Fusion [9], which adapt to exploit the ILP and TLP of their workloads by pooling resources from multiple cores to compose pipelines. To the



(a) $\beta = 1$ (constant failure rate) (b) $\beta = 3$ (gently increasing failure rate) (c) $\beta = 8$ (rapidly increasing failure rate)

Fig. 5: **Expected Availability**. We show the expected number of active pipelines as a function of time for a 12-core multicore system, a homogeneous, distributed microarchitecture with 12 execution resources per stage and ReDEEM with 12 heterogeneous execution resources per stage. We observe that ReDEEM always extends the lifetime of a processor in all the failure models shown.

best of our knowledge, ReDEEM is the first reliable distributed microarchitecture that offers fine-grained performance/energy heterogeneity and dynamic energy management.

Heterogeneous systems and DVFS. Heterogeneous multicore systems that offer multiple architecturally-different cores, with varied performance/energy characteristics, have been proposed [13], [6]. A lot of prior research is dedicated to coarse-grained DVFS techniques. Zhuravlev, *et al.* [23] have summarized most of the research on coarse-grained heterogeneity and DVFS. Finer-grained heterogeneous microarchitecture designs [16], [1] and scheduling algorithms [17], [15] have also been proposed. Recently, researchers have proposed systems that are composed of multiple microarchitecturally homogeneous designs synthesized for different voltage and frequency (VF) domains [5], [3]. These systems were shown to exhibit significantly greater energy efficiency than DVFS techniques.

VIII. Conclusions

We proposed an approach for designing reliable processors that are energy efficient. Our solution introduces fine-grained heterogeneity, application-adaptive scheduling, and fine-grained power-gating into a decentralized microarchitecture. We were able to show that our system offers several performance/energy operating points per application, enabling energy-efficient executions for given performance targets. In addition, we used probabilistic analysis to show that our system is more reliable than a conventional multicore and a previously proposed decentralized microarchitecture.

References

[1] D. Albonesi, R. Balasubramonian, S. Dropsbo, S. Dwarkadas, E. Friedman, M. Huang, V. Kursun, G. Magklis, M. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. Cook, and S. Schuster. Dynamically tuning processor resources with adaptive processing. *Computer*, 36(12), 2003.

[2] N. Binkert, B. Beckmann, G. Black, S. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. Hill, and D. Wood. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2), 2011.

[3] H. Bokhari, H. Javaid, M. Shafique, J. Henkel, and S. Parameswaran. darknoc: Designing energy-efficient network-on-chip with multi-vt cells for dark silicon. In *Proc. DAC*, 2014.

[4] S. Borkar and A. Chien. The future of microprocessors. *Communications of the ACM*, 54(5), 2011.

[5] K. Chakraborty and S. Roy. Architecturally homogeneous power-performance heterogeneous multicore systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(4), 2013.

[6] P. Greenhalgh. Big.LITTLE processing with ARM Cortex-15 & Cortex-a7, 2011.

[7] S. Gupta, S. Feng, A. Ansari, J. Blome, and S. Mahlke. The StageNet fabric for constructing resilient multicore systems. In *Proc. MICRO*, 2008.

[8] S. Gupta, S. Feng, A. Ansari, and S. Mahlke. Erasing core boundaries for robust and configurable performance. In *Proc. MICRO*, 2010.

[9] E. Ipek, M. Kirman, N. Kirman, and J. Martinez. Core fusion: Accommodating software diversity in chip multiprocessors. In *Proc. ISCA*, 2007.

[10] R. Kalayappan and S. Sarangi. A survey of checker architectures. *ACM Comput. Surv.*, 45(4), 2013.

[11] T. Kauerauf, R. Degraeve, E. Cartier, C. Soens, and G. Groeseneken. Low weibull slope of breakdown distributions in high-k layers. *Electron Device Letters, IEEE*, 23(4), 2002.

[12] K. Khubaib, M. Suleman, M. Hashemi, C. Wilkerson, and Y. Patt. Morphcore: An energy-efficient microarchitecture for high performance ilp and high throughput tlp. In *Proc. MICRO*, 2012.

[13] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen. Single-isa heterogeneous multi-core architectures: the potential for processor power reduction. In *Proc. MICRO*, 2003.

[14] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proc. MICRO*, 2009.

[15] A. Lukefahr, S. Padmanabha, R. Das, R. Dreslinski, Jr., T. F. Wenisch, and S. Mahlke. Heterogeneous microarchitectures trump voltage scaling for low-power cores. In *Proc. PACT*, 2014.

[16] A. Lukefahr, S. Padmanabha, R. Das, F. Sleiman, R. Dreslinski, T. Wenisch, and S. Mahlke. Composite cores: Pushing heterogeneity into a core. In *Proc. MICRO*, 2012.

[17] S. Padmanabha, A. Lukefahr, R. Das, and S. Mahlke. Trace based phase prediction for tightly-coupled heterogeneous cores. In *Proc. MICRO*, 2013.

[18] A. Pellegrini, J. Greathouse, and V. Bertacco. Viper: Virtual pipelines for enhanced reliability. In *Proc. ISCA*, 2012.

[19] B. Romanescu and D. Sorin. Core cannibalization architecture: Improving lifetime chip performance for multicore processors in the presence of hard faults. In *Proc. PACT*, 2008.

[20] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. 2002.

[21] R. L. Smith. Weibull regression models for reliability data. *Reliability Engineering and System Safety*, 34(1), 1991.

[22] N. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison Wesley, 4th edition, 2011.

[23] S. Zhuravlev, J. Saez, S. Blagodurov, A. Fedorova, and M. Prieto. Survey of energy-cognizant scheduling techniques. *IEEE Trans. on Parallel and Distributed Systems*, 24(7), 2013.