Extreme-Scale Computer Architecture: Energy Efficiency from the Ground Up[‡]

Josep Torrellas
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu/∼torrellas

***Abstract*—As we move to integration levels of 1,000-core processor chips, it is clear that energy and power consumption are the most formidable obstacles. To construct such a chip, we need to rethink the whole compute stack from the ground up for energy efficiency — and attain Extreme-Scale Computing. First of all, we want to operate at low voltage, since this is the point of maximum energy efficiency. Unfortunately, in such an environment, we have to tackle substantial process variation. Hence, it is important to design efficient voltage regulation, so that each region of the chip can operate at the most efficient voltage and frequency point. At the architecture level, we require simple cores organized in a hierarchy of clusters. Moreover, we also need techniques to reduce the leakage of on-chip memories and to lower the voltage guardbands of logic. Finally, data movement should be minimized, through both hardware and software techniques. With a systematic approach that cuts across multiple layers of the computing stack, we can deliver the required energy efficiencies.**

I. Introduction

As semiconductor devices continue to shrink, it is clear that we are about to witness stunning levels of integration on a chip. Sometime around the end of this decade, as we reach 7 nm, we will be able to integrate, for example, 1,000 sizable cores and substantial memory on a single die. There are many unknowns as to what kind of architecture such a chip should have to make it general purpose. What is clear, however, is that the main challenge will be to make it highly energy efficient. Energy and power consumption have emerged as the true limiters to developing more capable architectures.

Given the energy efficiency challenge, researchers have coined the term *Extreme Scale Computer Architecture* to refer to computer organizations that, loosely speaking, are 100-1,000 times more capable than current systems for the same power consumption and physical footprint. For example, these organizations should deliver a datacenter with a current physical footprint that provides exascale performace (10^{18} operations per second) for 20 MW; a departmental server that provides petascale performance (10^{15} operations per second) for 20 KW; and a portable device that provides *sustained* terascale performance (10^{12} operations per second) for 20 W. Extreme-scale computing is concerned with technologies that are applicable to all machine sizes — not just high-end systems. Overall, to accomplish these levels of energy efficiency, it is no exaggeration to say that these systems need to be designed for energy efficiency from the ground up.

There are certain aspects of these architectures that are clear. One is that they need to have efficient support for concurrency, since only massive parallelism will deliver this performance. In addition, they should minimize data transfers — since moving data around is a major source of energy consumption. Finally, they will have to rely on new technologies that will come online in the next few years. These technologies include low supply voltage (V_{dd}) operation, efficient on-chip V_{dd} regulation, 3D die stacking, resistive memories, and photonic interconnects, to name a few.

Perhaps less obvious is that all the layers of the computing stack will have to be designed for energy efficiency, and that new energy-efficiency techniques that cut across multiple layers will have to be designed. In this paper, we outline some of challenges that appear at different layers of the computing stack, and some of the techniques that can be used to address them. Specifically, after a brief background section, we consider the chip substrate at the level of devices and circuits, the architecture layer, data movement issues, and the programming layer.

II. Background

For several decades, the processor industry has seen a steady growth in CPU performance, driven by Moore's Law [1] and Classical (or Dennard) scaling [2]. Under classical scaling, the power density remains constant across semiconductor generations. Specifically, consider the dynamic power (P_{dyn}) consumed by a certain number of transistors that fit in a chip area *A*. The dynamic power is proportional to $C \times V_{dd}^2 \times f$, where *C* is the capacitance of the devices and *f* is the frequency of operation. Hence, the power density is proportional to $C \times V_{dd}^2 \times f/A$. As one moves to the next generation, the linear dimension of a device gets multiplied by a factor close to 0.7. The same is the case for V_{dd} and *C*, while the *f* gets multiplied by $1/0.7$. Moreover, the area of the transistors is now $0.7^2 \times A$. If we compute the new power density, we have $0.7C \times (0.7V_{dd})^2 \times f/(0.7^3 \times A)$. Consequently, the power density remains constant.

Unfortunately, as the feature size decreased below 130 nm over a decade ago, classical scaling ceased to apply for two reasons. First, V_{dd} could not be decreased as fast as before. In fact, in recent years, it has stagnated around 1 V, mostly due to the fact that, as V_{dd} gets smaller and closer to the threshold voltage (V_{th}) of the transistor, the transistor's switching speed decreases fast. The second reason is that static power became significant. The overall result is that, under real scaling, the power density of a set of transistors increases rapidly with each generation — making it progressively harder to feed the needed power and extract the resulting heat.

In addition, there are further concerns at both ends of the computing spectrum. At the high end, data centers are affected by large energy bills while, at the low end, handheld devices

[‡] This work was supported in part by the National Science Foundation under grant CCF-1012759, DARPA under PERFECT Contract Number HR0011-12-2-0019, and DOE ASCR under Award Numbers DE-FC02-10ER2599 and DE-SC0008717.

978-3-9815370-2-4/DATE14/©2014 EDAA

are limited by the energy that can be stored and supplied by batteries. Overall, all of these trends motivate the emergence of research on extreme-scale computing.

III. Energy-Efficient Chip Substrate

To realize extreme-scale computing systems, devices and circuits need to be designed to operate at low V_{dd}. This is because V_{dd} reduction is the best lever available to increase the energy efficiency of computing. V_{dd} reduction induces a quadratic reduction in dynamic energy, and a larger-than-linear reduction in static energy. There is evidence that the most energy-efficient operating point corresponds to a V_{dd} value somewhat higher than the threshold voltage (V_{th}) of the transistor [3]. This corresponds to a regime that has been called *Near-Threshold Computing (NTC)* [4], [5], [6]. Roughly speaking, for current technology, this corresponds to a V_{dd} of about 500 mV rather than the conventional 1 V.

While there are several unclear aspects with NTC, it can potentially decrease the power consumption by more than 40 times [4], [5]. This is a substantial reduction, and implies that many more cores can now be placed on a given power-constrained chip. Unfortunately, there are well-known drawbacks of NTC. They include a lower switching speed (possibly 10 times lower), and a large increase in process variation — the result of V_{dd} being close to V_{th}. It is possible that researchers will find ways of delivering NTC devices of acceptable speed. However, the issue of dealing with high process variation is especially challenging.

A. The Effects of Process Variation

Process variation is the deviation of the values of device parameters (such as a transistor's V_{th}, channel length, or channel width) from their nominal specification. Such variation causes variation in the switching speed and the static power consumption of nominally-similar devices in a chip. At the architectural level, this effect translates into cores and on-chip memories that are slower and consume more static power than they would otherwise do.

To see why, consider Figure 1. Chart (a) shows a hypothetical distribution of the latencies of dynamic logic paths in a pipeline stage. The X axis shows the latency, while the Y axis shows the number of paths with such latency. Without process variation (taller curve), the pipeline stage can be clocked at a frequency $1/\tau_{NOM}$. With variation (shorter curve), some paths become faster, while others slower. The pipeline stage's frequency is determined by the slower paths, and is now only $1/\tau_{VAR}$.

Figure 1(b) shows the effect of process variation on the static power (P_{STA}). The X axis of the figure shows the V_{th} of different transistors, and the Y axis the transistors' P_{STA}. The P_{STA} of a transistor is related to its V_{th} exponentially with $P_{STA} \propto e^{-V_{th}}$. Due to this exponential relationship, the static power saved by high-V_{th} transistors is less than the extra static power consumed by low-V_{th} transistors. Hence, integrating over all of the transistors in the core or memory module, total P_{STA} goes up with variation.

Process variation has a systematic component that exhibits spatial correlation. This means that nearby transistors will typically have similar speed and power consumption properties.

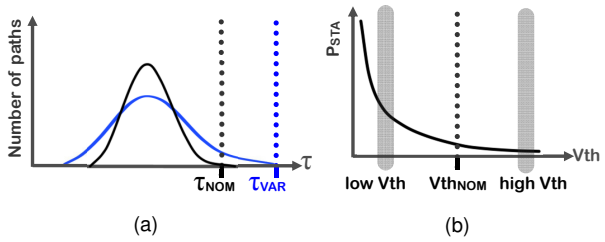

Fig. 1. Effect of process variation on the speed (a) and static power consumption (b) of architecture structures.

Hence, due to variation within a chip, some regions of the chip will be slower than others, and some will be more leaky than others. If we need to set a single V_{dd} and frequency for the whole chip, we need to set them according to the slowest and leakiest neighborhoods of the chip. This conservative design is too wasteful for extreme-scale computing.

B. Multiple Voltage Domains

NTC chips will be large and heavily affected by process variation. To tolerate process variation within a chip, the most appealing idea is to have multiple V_{dd} and frequency domains. A domain encloses a region with similar values of variation parameters. In this environment, we want to set a domain with slow transistors to higher V_{dd}, to make timing. On the other hand, we want to set a domain with fast, leaky transistors to lower V_{dd}, to save energy. For this reason, extreme scale NTC chips are likely to have multiple, possibly many, V_{dd} and frequency domains. How these domains are selected and set requires considering many tradeoffs [7].

However, current designs for V_{dd} domains are energy inefficient [8]. First, on-chip Switching Voltage Regulators (SVR) that provide the V_{dd} for a domain have a high power loss, often in the 10-15% range. Wasting so much power in an efficiency-first environment is hardly acceptable. In addition, small V_{dd} domains are more susceptible to variations in the load offered to the power grid, due to lacking as much averaging effects as a whole-chip V_{dd} domain. These variations in the load induce V_{dd} droops that need to protected against with larger V_{dd} guardbands [9] — also hardly acceptable in an efficiency-first environment. Finally, conventional SVRs take a lot of area and, therefore, including several of them on chip is unappealing. If, as a result, only a few are included in a large NTC chip, the variation inside the V_{dd} domain itself may negate some of the benefits of setting up the domain in the first place.

C. What is Needed

To address these limitations, several techniques are needed. First, an extreme-scale chip needs to be designed with devices whose parameters are optimized for low-V_{dd} operation [10]. For example, simply utilizing conventional device designs can result in slow devices.

Importantly, voltage regulators need to be designed for high energy efficiency and modest area. One possible approach is to organize them in a hierarchical manner [11]. The first level of the hierarchy is composed of one or a handful of SVRs, potentially placed on a stacked die, with devices optimized for the SVR inductances. The second level is composed of

many on-chip low-drop-out (LDO) voltage regulators. Each LDO is connected to one of the first-level SVRs and provides the V_{dd} for a core or a small number of cores. LDOs have high energy efficiency if the ratio of their output voltage (V_o) to their input voltage (V_i) is close to 1. Thanks to systematic process variation, the LDOs in a region of the chip need to provide a similar V_o to the different cores of the region. Since these LDOs take their V_i from the same first-level SVR and their V_o is similar, their efficiency can be close 95%. In addition, their area is negligible: their hardware reuses the hardware of a power-gating circuit. Such circuit is likely to be already present in the chip to power-gate the core. Finally, level converters between the resulting V_{dd} domains can be designed efficiently, by combining them with latches [12].

To minimize energy waste, the chip should have extensive power gating support. This is important at NTC because leakage accounts for the larger fraction of energy consumption. Ideally, power gating should be done at fine granularities, such as groups of cache lines, or groups of functional units. Fine granularities lead to high potential savings, but complicate circuit design. New algorithms need to be designed, possibly at the runtime system level, to control power gating from the software.

Finally, the architectural-level variation parameters of the chip should be made visible to the runtime or operating system. This includes, for each of the core clusters, the V_{dd} and frequencies supported, as well as the static power consumed. The software can then use this information for better assignment of clusters or cores to jobs.

IV. A Streamlined Architecture

A. Simple Organization

For highest energy efficiency, an exteme-scale architecture should be mostly composed of many simple, throughput-oriented cores, and rely on highly-parallel execution. NTC substantially reduces the power consumption, which can then be leveraged by increasing the number of cores that execute in parallel — as long as the application can exploit the parallelism. Such cores should avoid speculation and complex hardware structures as much as possible.

Cores should be organized in clusters. Such organization is energy-efficient because process variation has spatial correlation and, therefore, nearby cores and memories have similar variation parameter values — which can be exploited by the scheduler.

To further improve energy efficiency, a cluster typically contains a heterogeneous group of compute engines. For example, it can contain one wide superscalar core (also called latency core) to run sequential or critical sections fast. The power delivery system should be configured such that this core can run at high V_{dd} in a turbo-boost manner. Moreover, some of the cores may have special instructions implemented in hardware. These may include special synchronization or transcendental operations.

B. Minimizing Energy in On-Chip Memories

A large NTC chip can easily contain hundreds of Mbytes of on-chip memory. To improve memory reliability and energy efficiency, it is likely that SRAM cells will be redesigned for NTC [13]. In addition, to reduce leakage, such memory will likely operate at higher V_{dd} than the logic. However, even accounting for this fact, the on-chip memories will incur substantial energy losses due to leakage. To reduce this waste, the chip may support power gating of sections of the memory hierarchy — e.g., individual on-chip memory modules, or individual ways of a memory module, or groups of lines in a memory module. In principle, this approach is appealing because a large fraction of such a large memory is likely to contain unneeded data at any given time. Unfortunately, this approach is too coarse-grained to make a significant impact on the total power consumed: to power gate a memory module, we need to be sure that none of the data in the module will be used soon. This situation will likely be rare in the general case. Instead, what we need is a fine-grained approach were we power-on only the individual on-chip memory lines that contain data that will be accessed very soon.

To come close to this ideal scenario, we can use eDRAM rather than SRAM for the last levels of the cache hierarchy — either on- or off-chip. EDRAM has the advantage that it consumes much less leakage power than SRAMs. This saves substantial energy. However, eDRAM needs to be refreshed. Fortunately, refresh is done at the fine-grained level of a cache line, and we can design intelligent refresh schemes [14], [15].

One approach to intelligent refresh is to try to identify the lines that contain data that is likely to be used in the near future by the processors, and only refresh such lines in the eDRAM cache. The other lines are not refreshed and marked as invalid — after being written back to the next level of the hierarchy if they were dirty. To identify such lines we can dynamically use the history of line accesses [14] or programmer hints.

Another approach to intelligent refresh is to refresh different parts of the eDRAM modules at different frequencies, exploiting the different retention times of different cells. This approach relies on profiling the retention times of different on-chip eDRAM modules or regions. For example, one can exploit the spatial correlation of the retention times of the eDRAM cells [15]. With this technique and similar ones, we may refresh most of the eDRAM with long refresh periods, and only a few small sections with the conventional, short refresh periods.

C. Minimizing Energy in the On-Chip Network

The on-chip interconnection network in a large chip is another significant source of energy consumption. Given the importance of communication and the relative abundance of chip area, a good strategy is to have wide links and routers, and power gate the parts of the hardware that are not in use at a given time. Hence, good techniques to monitor and predict network utilization are important.

One characteristic of on-chip networks is that they are especially vulnerable to process variation. This is because the network connects distant parts of the chip. As a result, it has to work in the areas of the chip that have the slowest transistors, and in those areas with the leakeast transistors.

To address this problem, we can divide the network into multiple V_{dd} domains — each one including a few routers.

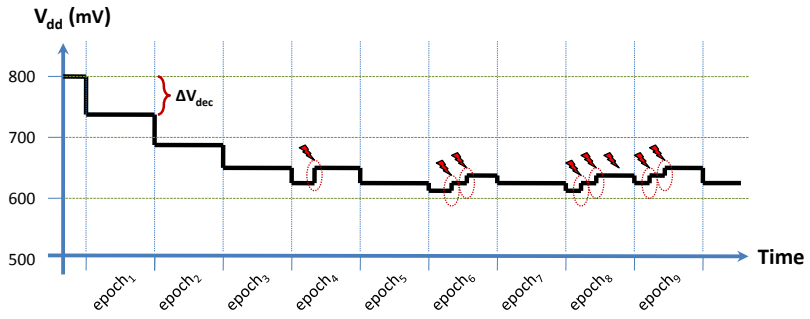

Fig. 2. Changes to the V_{dd} of a domain over time.

Due to the systematic component of process variation, the routers in the same domain are likely to have similar values of process variation parameters. Then, a controller can gradually reduce the V_{dd} of each domain dynamically, while monitoring for timing errors in the messages being transmitted. Such errors are being detected and handled with already-existing mechanisms in the network. When the controller observes an error rate in a domain that is higher than a certain threshold, the controller increases the V_{dd} of that domain slightly. In addition, the controller periodically decreases the V_{dd} of all the domains slightly, to account for changes in workloads and temperatures. Overall, with this approach, the V_{dd} of each domain converges to the lowest value that is still safe (without changing the frequency).

Figure 2 shows an example of the way the V_{dd} of a domain converges to a low, safe V_{dd}. In the figure, time is measured in 50-μs epochs, and pointers represent errors. We can see that the reduction in V_{dd} at each step (ΔV_{dec}) gets progressively smaller as time goes by. Moreover, when errors are detected, V_{dd} is increased. With this support, each domain converges to a different low V_{dd}, saving substantial energy in the process. We call this scheme Tangle [16].

V. Reducing Data Movement

As technology scales, data movement contributes with an increasingly larger fraction of the energy consumption in the chip [17]. Consequently, we need to devise approaches to minimize the amount of data transferred. In this section, we discuss a few mutually-compatible ways to do it.

One approach is to organize the chip in a hierarchy of clusters of cores with memories. Then, the system software can map a program's threads to a cluster and allocate their data in the memories of the cluster. This hardware organization and co-location for locality reduces the amount of data movement needed.

Another technique consists of using a single address space in the chip and directly manage in software the movement of data used by the application in the cache hierarchy. Many of the applications that will run on an extreme-scale 1,000-core chip are likely to have relatively simple control and data structures — e.g., performing many of their computation in regular loops with analyzable array accesses. As a result, it is conceivable that a smart compiler performing extensive program analysis [18], possibly with help from the programmer, will be able to manage (and minimize) the movement of data in the on-chip memory hierarchy.

In this case, the architecture supports simple instructions to manage the caches, rather than providing hardware cache coherence transparent to the programmer. Writes do not invalidate other cached copies of the data, and reads return the closest valid copy of the data. While the machine is now certainly harder to program, it may eliminate some data movement inefficiencies associated with the hardware cache coherence — such as false sharing, or moving whole lines when only a fraction of the data in the line is used. In addition, by providing a single address space, we eliminate the need to copy data on communication, as in message-passing models.

A third way of reducing the amount of data transfers is to use Processing in Memory (PIM) [19]. The idea is to add simple processing engines close to or imbedded in the main memory of the machine, and use them to perform some operations on the nearby data in memory — hence avoiding the round trip from the main processor to the memory.

While PIM has been studied for at least 20 years, we may now see it become a reality. Specifically, companies such as Micron and Samsung are building 3-D integrated circuits that stack one or more dies of memory with a die of logic. For example, Micron's Hybrid Memory Cube (HMC) [20] is a memory chip that contains a die of logic sitting below a stack of 4 or 8 DRAM dies, connected using through-silicon-vias (TSVs). Currently, the logic die only includes advanced memory controller functions plus self-test and error detection, correction, and repair. However, it is easy to imagine how to augment the capabilities of the logic die to support Intelligent Memory Operations [21]. These can consist of preprocessing the data as it is read from the DRAM stack into the processor chip. They can also involve performing operations in place on the DRAM data.

Finally, another means of reducing data transfers is to support in hardware efficient communication and synchronization primitives, such as those that avoid spinning in the network. These may include dynamic hierarchical hardware barriers, or efficient point-to-point synchronization between two cores using hardware full-empty bits [22].

VI. Programming Extreme-Scale Machines

The system software in extreme-scale machines has to be aware of the process variation profile of the chip. This includes knowing, for each cluster, the V_{dd} and frequency it can support and the leakage power it dissipates. With this information, the system software can make scheduling decisions that maximize energy efficiency. Similarly, the system software

should monitor different aspects of hardware components, such as their usage, the energy consumed, and the temperature reached. With this information, it can make decisions on what components to power gate, or what V_{dd} and frequency setting to use — possibly with help from application hints.

Application software is likely to be harder to write for extreme-scale architectures than for conventional machines. This is because, to save energy in data transfers, the programmer has to carefully manage locality and minimize communication. Moreover, the use of low V_{dd} in NTC requires more concurrency to attain the same performance.

An important concern is how users will program these extreme-scale architectures. In pratice, there are different types of programmers based on their expertise. Some are expert programmers, in which case they will be able to map applications to the best clusters, set the V_{dd} and frequency of the clusters, and manage the data in the cache hierarchy well. They will obtain good energy efficiency.

However, many programmers will likely be relatively inexperienced. Hence, they need a high-level programming model that is simple to program and allows them to express locality. One such model is Hierarchical Tiled Arrays (HTA) [23], which allows the computation to be expressed in recursive blocks or tiles. Another possible model is Concurrent Collections [24], which expresses the program in a dataflow-like manner. These are high-level models, and the compiler still has to translate them into efficient machine code. For this, the compiler may have to rely on program auto-tuning to find the best code mapping in these complicated machines.

VII. Conclusion

Attaining the 100-1,000 improvement in energy efficiency required for extreme-scale computing involves rethinking the whole computing stack from the ground up for energy efficiency. In this paper, we have outlined some of the techniques for energy efficiency that can be used at different levels of the computing stack. Specifically, we have discussed the need to operate at low voltage, provide efficient voltage regulation, and support simple cores organized in clusters. Memories and networks can be optimized by reducing leakage and minimizing the guardbands of logic. Finally, data movement can be minimized by managing the data in the cache hierarchy, processing in memory, and utilizing efficient synchronization. A major issue that remains in these machines is the challenge of programmability.

References

[1] G. E. Moore, "Cramming More Components onto Integrated Circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, 1965.

[2] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[3] H. Kaul, M. Anders, S. Mathew, S. Hsu, A. Agarwal, R. Krishnamurthy, and S. Borkar, "A 320mV 56W 411GOPS/Watt Ultra-Low Voltage Motion Estimation Accelerator in 65nm CMOS," in *International Solid-State Circuits Conference*, February 2008.

[4] L. Chang, D. J. Frank, R. K. Montoye, S. J. Koester, B. L. Ji, P. W. Coteus, R. H. Dennard, and W. Haensch, "Practical Strategies for Power-Efficient Computing Technologies," *Proceedings of the IEEE*, February 2010.

[5] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits," *Proceedings of the IEEE*, February 2010.

[6] D. Markovic, C. C. Wang, L. P. Alarcon, T.-T. Liu, and J. M. Rabaey, "Ultralow-Power Design in Near-Threshold Region," *Proceedings of the IEEE*, February 2010.

[7] C. Silvano, G. Palermo, S. Xydis, and I. Stamelakos, "Voltage Island Management in Near Threshold Manycore Architectures to Mitigate Dark Silicon," in *Conference on Design, Automation and Test in Europe*, March 2014.

[8] U. R. Karpuzcu, A. Sinkar, N. S. Kim, and J. Torrellas, "EnergySmart: Toward Energy-Efficient Manycores for Near-Threshold Computing," in *International Symposium on High Performance Computer Architecture*, February 2013.

[9] N. James, P. Restle, J. Friedrich, B. Huott, and B. McCredie, "Comparison of Split Versus Connected-Core Supplies in the POWER6 Microprocessor," in *International Solid-State Circuits Conference*, February 2007.

[10] H. Wang and N. S. Kim, "Improving Platform Energy-Chip Area Trade-off in Near-Threshold Computing Environment," in *International Conference on Computer Aided Design*, November 2013.

[11] H. R. Ghasemi, A. Sinkar, M. Schulte, and N. S. Kim, "Cost-Effective Power Delivery to Support Per-Core Voltage Domains for Power-Constrained Processors," in *Design Automation Conference*, June 2012.

[12] F. Ishihara, F. Sheikh, and B. Nikolić, "Level Conversion for Dual-Supply Systems," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 12, no. 2, pp. 185–195, Feb. 2004.

[13] T. Gemmeke, M. M. Sabry, J. Stuijt, P. Raghavan, F. Catthoor, and D. Atienza, "Resolving the Memory Bottleneck for Single Supply Near-Threshold Computing," in *Conference on Design, Automation and Test in Europe*, March 2014.

[14] A. Agrawal, P. Jain, A. Ansari, and J. Torrellas, "Refrint: Intelligent Refresh to Minimize Power in On-Chip Multiprocessor Cache Hierarchies," in *International Symposium on High Performance Computer Architecture*, February 2013.

[15] A. Agrawal, A. Ansari, and J. Torrellas, "Mosaic: Exploiting the Spatial Locality of Process Variation to Reduce Refresh Energy in On-Chip eDRAM Modules," in *International Symposium on High Performance Computer Architecture*, February 2014.

[16] A. Ansari, A. Mishra, J. Xu, and J. Torrellas, "Tangle: Route-Oriented Dynamic Voltage Minimization for Variation-Afflicted, Energy-Efficient On-Chip Networks," in *International Symposium on High Performance Computer Architecture*, February 2014.

[17] P. Kogge *et al.*, "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems," in *DARPA-IPTO Sponsored Study*, September 2008.

[18] P. Feautrier, "Some Efficient Solutions to the Affine Scheduling Problem – Part I One-Dimensional Time," 1996.

[19] P. Kogge, "The EXECUBE Approach to Massively Parallel Processing," in *International Conference on Parallel Processing*, August 1994.

[20] Micron Technology, Inc., "Hybrid Memory Cube," 2011, http://www.micron.com/products/hybrid-memory-cube.

[21] B. Fraguela, P. Feautrier, J. Renau, D. Padua, and J. Torrellas, "Programming the FlexRAM Parallel Intelligent Memory System," in *International Symposium on Principles and Practice of Parallel Programming*, June 2003.

[22] B. J. Smith, "Architecture and Applications of the HEP Multiprocessor Computer System," in *Real-Time Signal Processing IV*, 1982, pp. 241–248.

[23] G. Bikshandi, J. Guo, D. Hoeflinger, G. Almasi, B. B. Fraguela, M. J. Garzarán, D. Padua, and C. von Praun, "Programming for Parallelism and Locality with Hierarchically Tiled Arrays," in *International Symposium on Principles and Practice of Parallel Programming*, 2006.

[24] Z. Budimlic, A. Chandramowlishwaran, K. Knobe, G. Lowney, V. Sarkar, and L. Treggiari, "Multi-core Implementations of the Concurrent Collections Programming Model," in *Workshop on Compilers for Parallel Computers*, 2009.