Chapter 18
Approximate Ultra-Low Voltage Many-Core Processor Design

Nam Sung Kim and Ulya R. Karpuzcu

18.1 Pushing Voltage Overscaling to Its Limits at Ultra-Low Operating Voltages

Divergence from Dennardian scaling [9] rendered modern computing platforms power-(budget)-limited [15, 23]. One promising way to cram more computations into the available power budget is to reduce the operating voltage *Vdd*, as power consumption reduces super-linearly with *Vdd*. The question is by how much. If *Vdd* remains slightly above the threshold voltage *Vth*, power consumption can decrease by more than an order of magnitude [10].

Power consumption decreases with the proximity of the near-threshold *Vdd* to *Vth*, however, so does the operating frequency, *f*. Therefore, operation at as ultra-low voltages as near-threshold, *near-threshold computing (NTC)*, is only meaningful if such *f* degradation is tolerable. In fact, more parallelism by distributing computation to more compute engines (in the form of general purpose cores or dedicated accelerators) can help mask the negative impact of the degraded *f* from throughput performance. At near-threshold voltages, by construction, more compute engines can fit into a given power budget. As power savings from near-threshold voltage (NTV) operation exceed the power cost of more compute engines participating in computation [4], parallelism becomes a limiting factor, as opposed to the power budget.

N. S. Kim
Department of Electrical and Computer Engineering, University of Illinois, Urbana Champaign, IL, USA
e-mail: nskim@illinois.edu

U. R. Karpuzcu (✉)
Department of Electrical and Computer Engineering, University of Minnesota, Twin Cities, MN, USA
e-mail: ukarpuzc@umn.edu

S. Reda, M. Shafique (eds.), *Approximate Circuits*,
https://doi.org/10.1007/978-3-319-99322-5_18

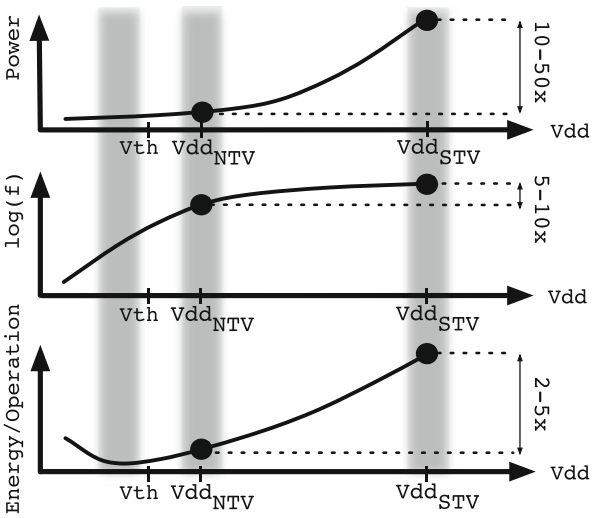

Fig. 18.1 Evolution of the operating point with *Vdd* [10, 16]

Figure 18.1 depicts power, frequency *f*, and energy per operation as a function of *Vdd*. STV (super-threshold voltage) corresponds to conventional, super-threshold voltage computing. At NTV, energy per operation improves by about 2–5× over STV—at the expense of a 5–10× *f* degradation [10, 16]. The corresponding power reduction by 10–50× enables more compute engines to fit into a given power budget. Minimum power and energy/operation points fall into the sub-threshold region (*Vdd*<*Vth*), where *f* degrades significantly. Highest *f* of operation resides in the super-threshold region (STV), but comes at the cost of notably higher power and energy/operation. Near-threshold region (NTV), on the other hand, facilitates a sweet spot with power savings closer to sub-threshold, but *f* closer to STV. Away from NTV, higher *Vdd* leads to substantially higher power, and lower *Vdd*, to substantially lower *f*. Therefore, *Vdd* should remain as close to *Vth* as possible.

To a first order, execution time is proportional to *work per parallel task* and the inverse of clock *f*, where the *problem size* distributed over the total number of cores engaged in computation (*N*) determines *work per parallel task* (Eq. (18.1)):

$$\begin{aligned} \text{Execution Time} &\propto \frac{\textit{Work per parallel task}}{f} \\ &\propto \frac{\textit{Problem Size/N}}{f} \end{aligned} \tag{18.1}$$

Therefore, for a fixed *problem size*, in order to offset the NTC-induced *f* degradation of 5–10× from Fig. 18.1 such that the execution time remains intact, at least 5–10× additional cores are required, bringing about a power cost of 5–10×. Per-core NTC power savings of 10–50× can easily counterbalance the power cost of 5–10× more cores contributing to computation. The question rather is whether the fixed problem size can render enough work to keep 5–10× more cores busy. Even in

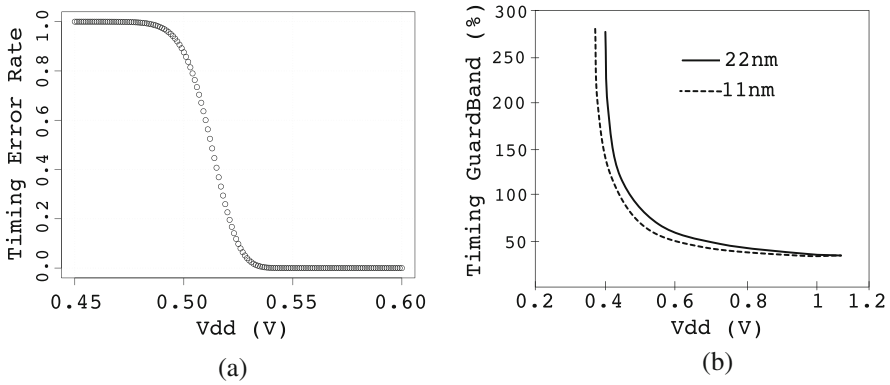

Fig. 18.2 (**a**) Variation-induced timing error rate [20], (**b**) timing guardband [4] vs. *Vdd*

the presence of abundant parallelism, however, the corresponding expansion in the chip area (to accommodate 5–10× additional cores) is likely to further exacerbate NTC's already intensified vulnerability to errors, particularly the critical class of (parametric) variation-induced errors.

With each technology generation, manufacturing imperfections exacerbate vulnerability to parametric variations, deviation of transistor parameters from nominal specifications. Already at STV variation results in not only slower cores, but also ample speed differences among the cores. At lower *Vdd*, transistor delay becomes more sensitive to variation. Therefore, NTC accentuates variation-induced slowdown and speed differences. As a result, the likelihood of variation-induced timing errors increases as *Vdd* reaches *Vth* (Fig. 18.2a). Timing errors emerge if variations slow down logic to prevent operation at the designated clock *f*. A common STV design practice to eliminate timing errors is operating the system at a lower *f* than sustainable were there no variation. This slowdown to guarantee error-free execution constitutes the *timing guardband*. Figure 18.2b depicts the timing guardband as a function of *Vdd*, considering two technology nodes. Unfortunately, the guardband excessively grows as *Vdd* reaches *Vth*, where the nominal *f*–the sustainable *f* were there no variation—is already low. Thus, relying on worst-case guardbanding is not feasible at NTV. A further difficulty stems from the diminishing efficacy of state-of-the-art STV variation mitigation techniques when adapted at NTV [11, 19].

18.2 Embracing Errors at Ultra-Low Operating Voltages

Emerging R(ecognition), M(ining), and S(ynthesis) applications rely on probabilistic, often iterative algorithms to process massive, yet noisy and redundant data. Usually the solution space has many more elements than one, deeming a

range of application outputs valid as opposed to a single "golden" output [5]. Therefore, RMS applications can tolerate errors emanating from data-intensive program phases as opposed to control [7]. Thread decomposition of most parallel RMS algorithms already conforms to decoupled data and control. Still, embracing errors necessitates

1. enforcing errors to be contained where they can be tolerated (i.e., within data-intensive program phases): where they manifest as degradation in output accuracy, i.e., quality of computing, Q, and not as catastrophic termination; and
2. controlling or configuring the output quality Q explicitly to prevent excessive loss in computation accuracy.

It is not uncommon for application-specific input parameters such as time step granularity or resolution to dictate output quality. Such inputs usually associate with the problem size, as well [6, 8]. To utilize more cores, the application can scale in two distinct ways, depending on whether the problem size changes with the number of cores (weak scaling) or not (strong scaling) [14]. Compensation for a typical NTC-induced f degradation of $\approx$5–10$\times$ (Fig. 18.1) can easily exceed strong scaling limits. For a compute-bound application, the execution time incurred by a problem size of interest is often unacceptably large. Accordingly, only problem sizes taking no longer than the maximum tolerable execution time become practical. The maximum tolerable execution time induces a time budget (e.g., as determined by machine utilization policies in a shared cluster) which remains mostly constant. For these applications, rather than accelerating the execution of a problem of fixed size, solving a problem of larger size within this constant time budget matters [22]. RMS applications are largely compute-bound [2], thus conform to weak scaling, where the problem size tends to scale with the number of cores engaged in computation.

A larger problem size cannot only facilitate a lower operating *Vdd* by engaging more cores in computation, but also result in a higher tolerance to errors where the vulnerability to errors increases due to the lower *Vdd*. By careful configuration, the degraded Q in the presence of errors can remain higher than its counterpart under a smaller problem size. In this case, increasing the problem size translates into configuring the application to generate an output of higher quality Q.

Figure 18.3 demonstrates, for two representative RMS benchmarks from the PARSEC suite [3], *hotspot* and *canneal*, respectively, how output quality Q changes with the problem size under three different scenarios. Both axes are normalized. Under *Default*, all parallel tasks assigned to computation actually contribute to computation: Q increases with problem size monotonically, although the sensitivity to problem size varies. By construction, the application can tolerate higher error rates at larger problem sizes. *hotspot* exhibits a higher sensitivity to the problem size. Thus, the same Δ increase in problem size would lead to a larger Q increase, which in turn enables operation at higher error rates. To understand how the picture changes under the onset of errors, the next two scenarios mimic a close-to-worst-case manifestation of errors by dropping a quarter (*Drop 1/4*), and a half (*Drop 1/2*) of the parallel tasks assigned to computation. The bottomline is that even *Drop 1/2* does not render an excessive Q degradation.

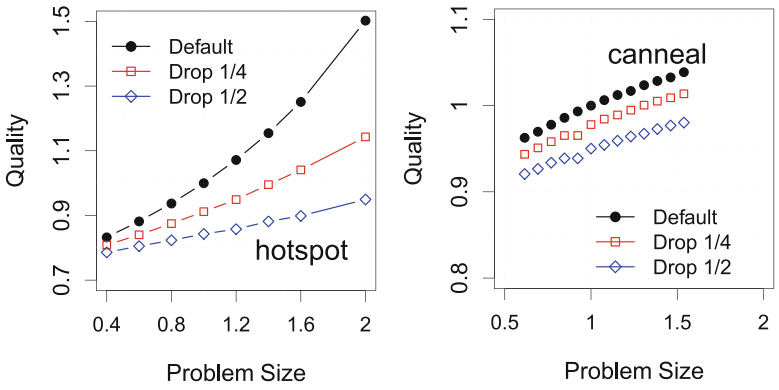

Fig. 18.3 Impact of problem size on quality of computing [20]

Hotspot is a thermal simulator that iteratively solves the heat transfer differential equations for a given chip floorplan. The number of iterations is the input parameter dictating the problem size. The output is the temperature at each point of a grid super-imposed on the floorplan. *Canneal*, on the other hand, implements simulated annealing to minimize the routing cost of a given chip design. At each temperature step, *swaps per temperature step* times, each thread attempts to swap two randomly picked blocks on chip to arrive at a lower-cost design. The thread searches for an optimal solution by attempting swaps, and only swaps leading to a lower routing cost are accepted:

```
while ( temperature steps < max temperature steps ){
   while ( swaps < swaps per temperature step){
      swap()
      ...
      swaps++
   }
   temperature steps++
}
```

Total number of temperature steps, *max temperature steps*, and *swaps per temperature step* represent input parameters. The product thereof governs the problem size. At the same time, both of the input parameters dictate how much effort the application puts to search for local optima. Since more effort is expected to result in a higher quality solution, the input parameters also affect the quality of computing. In this case quality corresponds to the relative routing cost. Figure 18.3 provides the resulting Q vs. problem size fronts as *swaps per temperature step* increases. Changes in the problem size as a function of such input parameters do not result in a different problem, but make the same problem result in a different solution accuracy. Therefore, problem size can act as a quality knob.

At NTV, the optimal core count, N, strongly depends on the proximity of the near-threshold *Vdd* to *Vth*. An effective way to have the execution time at NTV converge to the execution time at STV therefore becomes modulating N along with

Table 18.1 Modulating problem size along with number of cores and f to have NTV and STV execution times converge [20]

Goal: Execution $\text{Time}_{\text{NTV}} \rightarrow$ Execution Time_{STV}				
			Operating frequency	
Problem size (PS)	Number of cores	Quality	Default	Over-scaled
$\text{PS}_{\text{NTV}} = \text{PS}_{\text{STV}}$	$N_{\text{NTV}} > N_{\text{STV}}$	$Q_{\text{NTV}} \leq Q_{\text{STV}}$	$f_{\text{NTV}} \leq f_{\text{NTV,Safe}}$	$f_{\text{NTV}} > f_{\text{NTV,Safe}}$
$\text{PS}_{\text{NTV}} < \text{PS}_{\text{STV}}$	No restriction			
$\text{PS}_{\text{NTV}} > \text{PS}_{\text{STV}}$	$N_{\text{NTV}} > N_{\text{STV}}$			

f_{STV} corresponds to the nominal f at STV; $f_{\text{NTV,Safe}}$, to the highest possible f at NTV to exclude timing errors; f_{NTV}, to the modulated f at NTV. $f_{\text{STV}} > f_{\text{NTV}} \geq f_{\text{NTV,Safe}}$ applies

the near-threshold *Vdd* and f [20]:

$$\underbrace{\frac{Problem\ Size_{\text{NTV}}}{f_{\text{NTV}} \times N_{\text{NTV}}}}_{\propto \text{Execution Time}_{\text{NTV}}} \longrightarrow \underbrace{\frac{Problem\ Size_{\text{STV}}}{f_{\text{STV}} \times N_{\text{STV}}}}_{\propto \text{Execution Time}_{\text{STV}}} \tag{18.2}$$

Table 18.1 summarizes different ways to modulate the problem size, number of cores, and frequency of operation to have NTV and STV execution times converge.

The first option strictly follows strong scaling semantics by keeping the problem size (PS) intact. Due to $\text{PS}_{\text{NTV}} = \text{PS}_{\text{STV}}$, the NTV core count N_{NTV} should increase by at least $f_{\text{STV}}/f_{\text{NTV}}$ to retain the STV execution time. The next option is compressing the problem size to reach the STV execution time despite lower f_{NTV}. As long as per-core work $\propto \text{PS}/N$ reduces proportional to $f_{\text{NTV}}/f_{\text{STV}}$, NTV execution time can reach the STV execution time. However, the compressed problem size translates into a degradation in output quality Q. Hence, a threshold on Q may impose a limit on the compressed problem size. Even under limited compression, per-core work can reduce by $f_{\text{NTV}}/f_{\text{STV}}$, by carefully modulating N. The final option is expanding the problem size. In this case, reaching the STV execution time despite f_{NTV} is only possible if N increases more than the problem size does, such that per-core work $\propto \text{PS}/N$ reduces. However, in order to come close to the STV execution time, per-core work should reduce by $f_{\text{NTV}}/f_{\text{STV}}$. This translates into N increasing by $f_{\text{STV}}/f_{\text{NTV}} \times \text{PS}_{\text{NTV}}/\text{PS}_{\text{STV}}$. Such an increase in N may not always be feasible. In this case, operation at a higher f_{NTV} than the near-threshold *Vdd* can safely accommodate ($f_{\text{NTV,Safe}}$) can help. A higher f_{NTV} than safe would raise the likelihood of timing errors, yet the expanded problem size can make up for the corresponding Q degradation by careful modulation. Depending on how f_{NTV} is set, each option from Table 18.1 comes in two flavors:

- *Default* with $f_{\text{NTV}} \leq f_{\text{NTV,Safe}}$ excludes quality degradation due to errors by imposing a safe operating f. However, problem size compression may still result in degraded output quality.
- *Over-scaled* with $f_{\text{NTV}} > f_{\text{NTV,Safe}}$, on the other hand, by imposing a higher operating f than safe, may incur quality degradation due to the potential onset

of timing errors. This *voltage overscaling* equivalent at NTV is especially feasible under problem size expansion, where the (expanded) problem size can be configured to reduce quality degradation in the presence of errors.

Default variants under problem size compression (which exclude timing errors) may lead to similar quality degradation (due to the compressed problem size) as *Over-scaled* variants under constant or expanded problem size (which embrace errors). Only under problem size compression can N_{NTV} remain less than N_{STV}.

Over-scaled variants represent a more typical use case for expanded problem sizes. However, since the problem size gets modulated along with the number of cores, even for *Default* variants under problem size expansion (where the operating f is not increased) not necessarily more work would be the case for already slow NTV cores—as long as N increases more than the problem size does.

18.3 Decoupling Data and Control Flows

All cores engaged in (data-intensive) computation run at the same frequency f to ensure that parallel tasks make similar progress. This typically leads to faster overall execution, and eliminates any synchronization overhead that would be incurred if cores operated at different frequencies. An expanded problem size activates more cores to operate at a lower f. As the number of active cores expands, cores suffering from substantial variation-induced slowdown become more likely to participate in computation. These cores can limit the overall operating f. A compressed problem size, on the other hand, activates less cores to operate at a higher f. Due to the higher operating f and the increased likelihood of including more resilient and faster cores, a compressed problem size would not necessarily incur severe quality degradation.

Timing errors should stay where they can be tolerated. One option is executing error-tolerant data-intensive program phases on error-prone **D**ata **C**ores, and reserving more reliable **C**ontrol **C**ores for control [1]. CCs and DCs work in master–slave mode, where each CC coordinates computation on a designated set of DCs.

Control Cores should be protected from any type of error to prevent catastrophic failures or hangs. To this end, these cores can be designed to encompass robust transistors and circuits and/or operate at a higher *Vdd*. A variation-afflicted can reserve the fastest (or most resilient) cores for CCs. CCs periodically check whether DCs are done with the assigned computation. CCs are in charge of housekeeping; once DCs finish computation, the master CCs merge or reduce results from different DCs. To detect potential crashes or hangs of DCs, CCs keep watchdogs on a per DC basis. To prevent error propagation from DCs, CCs never rely on data produced by DCs for control. CCs communicate with DCs over a dedicated memory location, both to find out whether slave DCs are done with computation and to collect DC results. DCs can only read, but not modify the data produced by master CCs.

To facilitate effective coordination with CCs, DCs feature fast reset and restart hardware. A DC has access to a private read–write memory where the DC can write,

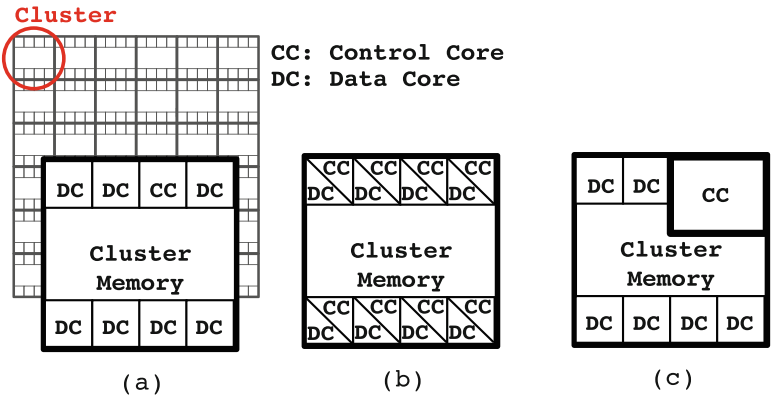

Fig. 18.4 Architectural design space for decoupled data and control flow [1]

in addition to a read-only memory where shared data managed by master CCs reside. To avoid error propagation, DCs cannot write to the private space of CCs or of other DCs. Instead, a dedicated memory location serves intra-DC communication.

Figure 18.4 demonstrates a proof-of-concept chip design, clustered to enhance scalability. A few cores with per-core private memories and a shared cluster memory constitute each cluster. Various options exist to differentiate control cores from data cores.

Spatio-temporal Heterogeneity (Fig. 18.4a): Each cluster accommodates identical cores. There is no difference in the design of CCs and DCs. CCs are distinguished from DCs *spatio-temporally*: CCs correspond to the fastest, most reliable cores (i.e., the cores of minimum slowdown under variation). This option is simpler from the hardware perspective. Still, the organization is very flexible in that number of CCs can be configured—although the example from Fig. 18.4a depicts one CC per cluster. Since variations govern the f of CCs, the range of CC frequencies may differ across chips.

Temporal Heterogeneity, (Fig. 18.4b): CCs and DCs are identical by design. Instead of explicitly assigning cores to operate as CCs or DCs, each core is time-multiplexed between CC and DC functionality. This option provides a better use of hardware resources, yet complicates the design due to the support required for different memory protection domains.

Design-driven Heterogeneity, (Fig. 18.4c): CCs and DCs per cluster represent different types of cores by design. In this case DCs and CCs specialize. However, the number of CCs may easily become a bottleneck. Depending on the application, a higher or a lower CC to DC ratio may be favorable. The organization from Fig. 18.4c assumes one CC per cluster. CCs are expected to consume more area than DCs due to the control-intensive specialization and the demand for enhanced reliability.

18.4 Modeling Errors

The most challenging task in designing systems to embrace errors is modeling errors themselves. Tools such as VARIUS-NTV [18] can help extract Vdd_{MIN}, the minimum near-threshold *Vdd* each memory block can support under parametric variation. To have blocks stay functional at NTV, any designated near-threshold operating *Vdd* should remain higher than such Vdd_{MIN}. VARIUS-NTV can also determine safe operating frequencies, and estimate timing error rates as a function of the operating frequency at the designated near-threshold *Vdd*.

VARIUS-NTV can quantify the variation-induced slowdown and the resulting timing error rates across the cores of the hypothetical NTV chip from Fig. 18.4a, as follows: first extract the minimum *Vdd*, Vdd_{MIN}, each cluster can support to remain functional at NTV. Below such Vdd_{MIN}, memory blocks may not be able to hold or change state. Figure 18.5a shows the distribution of per-cluster Vdd_{MIN} for one representative chip (out of 100): Per-cluster Vdd_{MIN} is defined as the maximum Vdd_{MIN} across all memory blocks within a cluster. The data is shown as a histogram. Per-cluster Vdd_{MIN} values vary in a significant range, and the maximum per cluster Vdd_{MIN} becomes the chip-wide Vdd_{NTV}.

Tasks get assigned to cores at the granularity of clusters. Each cluster constitutes a f domain. The slowest core within each cluster determines the operating f of the cluster. Figure 18.5b depicts variation-induced timing error rate per cycle, *Perr*, when operating at Vdd_{NTV}, as a function of f. For each cluster, the figure demonstrates the error rate curve of the slowest (i.e., most error prone) core within the cluster. The y axis is on log-scale. *Perr* values rapidly increase to reach 1, as f increases beyond 0.5 GHz. Even at acceptably low *Perr* of [1e-16,1e-12], the majority of the cores cannot operate at the NTV f_{NOM} of 1 GHz—the sustainable f were there no variation. *Perr* in the range of [1e-16,1e-12] induce a timing error

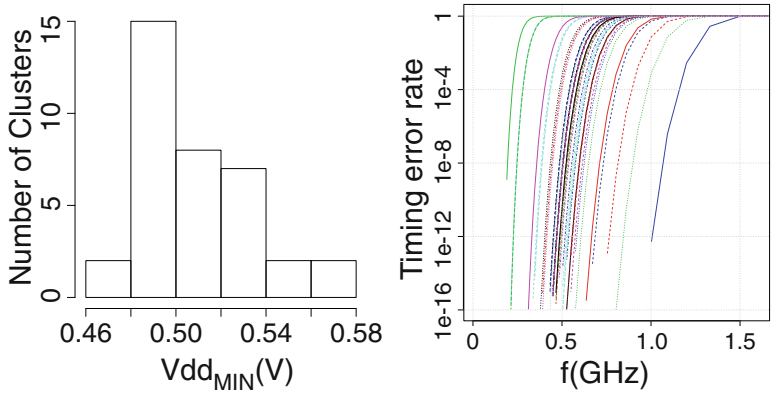

Fig. 18.5 Impact of parametric variation [20]

every [1e16,1e12] cycles. According to Fig. 18.5b, at the lower end of this *Perr* range, the maximum f the slowest core in each cluster can support exhibits 0.14–0.72× slowdown over the NTV f_{NOM}.

VARIUS-NTV does not capture how the errors would actually manifest. The exhaustive set of potential manifestations of variation-induced errors can be summarized as (1) no termination (due to crashes or hangs); (2) termination with excessive Q degradation; and (3) termination with acceptable Q degradation. The execution model relies on CCs to detect (1), e.g., by deploying watchdog timers. The application layer would perceive (1) mainly as dropped computation, as captured by *Drop* scenarios in Fig. 18.3. (2) points to degradation of data-intensive phases to lead to unacceptable accuracy. CCs can capture (2) by enforcing preset limits on maximum Q degradation (e.g., as defined by the user). Threads not conforming to such limits can be treated exactly as threads leading to (1), as mimicked by *Drop*.

On the other hand, (3) does not require CCs intervention. The above analysis assumes that (3) cannot lead to higher Q degradation than (1), purely relying on inherent algorithmic error tolerance of RMS applications. This assumption can be validated by statistical error injection, to corrupt the end result in various ways: all bits/higher order bits only/lower order bits only stuck-at-1(0)/randomly flipped/inverted, instead of ignoring the end result of infected threads. Then, it can be concluded that *Drop* can capture close-to-worst case error manifestation *under the decoupled execution model*. *Drop* conservatively ignores potential masking of errors at various levels of the system stack: Any timing error of probability *Perr* reaches the application layer to corrupt the end output each thread generates. The end result of any infected thread is ignored altogether.

18.5 Concluding Remarks

In strict sense, *weak scaling* implies constant per-core work. While applications strictly conforming to weak scaling would benefit most from the techniques covered in this chapter, problem size can still be useful as an effective quality knob even if per-core work increases slightly with problem size.

The fundamental limitation, however, stems from the modeling of errors. Without loss of generality, this chapter focused on variation-induced timing errors, as a running example. *Voltage overscaling* or *timing speculation* variants [12], be it implemented at STV or NTV, can always lead to a sudden bursty onset of errors, due to aggressive timing optimization practices [21]. The remedy is designing these systems from the ground-up for timing speculation [13]. Even then, capturing actual manifestation of errors at higher levels of the system stack is challenging. Attempting to characterize the worst-case as covered in Sect. 18.4 may help.

At ultra-low voltages, due to the reduced operating voltage aging-induced errors become less of a concern. However, soft errors as induced by alpha particle radiation become more prominent [17]. Due to their transient and random nature, detecting and tolerating soft errors in logic is especially challenging. The techniques from this chapter can still help if, for example, soft errors were confined to data cores

(Sect. 18.3) by hardening control cores against soft errors by design. Still, the question of how exactly such errors would manifest themselves at higher levels of the system stack at ultra-low voltages remains open.

References

1. Akturk I, Kim NS, Karpuzcu UR (2015) Decoupled control and data processing for approximate near-threshold voltage computing. IEEE Micro 35(4):70–78
2. Bhadauria M, Weaver VM, McKee SA (2009) Understanding PARSEC performance on contemporary CMPs. In: Proceedings of the IEEE International Symposium on Workload Characterization (IISWC), Washington, DC, USA, pp 98–107
3. Bienia C (January 2011) Benchmarking modern multiprocessors. Ph.D. Thesis, Princeton University
4. Chang L et al (February 2010) Practical strategies for power-efficient computing technologies. Proc IEEE 98(2):215–236
5. Chippa VK, Mohapatra D, Raghunathan A, Roy K, Chakradhar ST (2010) Scalable effort hardware design: exploiting algorithmic resilience for energy efficiency. In: ACM/EDAC/IEEE design automation conference
6. Chippa V, Raghunathan A, Roy K, Chakradhar S (2011) Dynamic effort scaling: managing the quality-efficiency tradeoff. In: ACM/EDAC/IEEE design automation conference
7. Cho H, Leem L, Mitra S (April 2012) ERSA: Error Resilient System Architecture for probabilistic applications. In: IEEE Trans Comput Aided Des Integr Circuits Syst 31(4):546–558
8. de Kruijf M, Nomura S, Sankaralingam K (2011) Relax: an architectural framework for software recovery of hardware faults. In: IEEE/ACM International Symposium on Computer Architecture (ISCA)
9. Dennard RH, Gaensslen FH, Rideout VL, Bassous E, LeBlanc AR (October 1974) Design of ion-implanted MOSFET's with very small physical dimensions. IEEE J Solid State Circuits 9(5):256–268
10. Dreslinski RG, Wieckowski M, Blaauw D, Sylvester D, Mudge T (February 2010) Near-threshold computing: reclaiming Moore's law through energy efficient integrated circuits. Proc IEEE 98(2):253–266
11. Dreslinski RG, Giridhar B, Pinckney N, Blaauw D, Sylvester D, Mudge T (2012) Reevaluating fast dual-voltage power rail switching circuitry. In: Annual Workshop of Duplicating, Deconstructing and Debunking (WDDD) in conjunction with ISCA, vol. 39
12. Esmaeilzadeh H, Sampson A, Ceze L, Burger D (2012) In: ACM international conference on architectural support for programming languages and operating systems
13. Greskamp B et al (2009) Blueshift: designing processors for timing speculation from the ground up. In: IEEE international symposium on high performance computer architecture
14. Gustafson JL (1988) Reevaluating Amdahl's law. Commun ACM 31(5):532–533
15. Horowitz M (2014) Computing's energy problem (and what we can do about it). In: Keynote at IEEE international conference on solid state circuits
16. Jain S et al (2012) A 280mV-to-1.2V wide-operating-range IA-32 processor in 32nm CMOS. In: IEEE international solid-state circuits conference, San Francisco, CA, pp 66–68
17. Kaul H, Anders M, Hsu S, Agarwal A, Krishnamurthy R, Borkar S (2012) Near-threshold voltage (NTV) design – opportunities and challenges. In: ACM/EDAC/IEEE design automation conference
18. Karpuzcu UR, Kolluru KB, Kim NS, Torrellas J (2012) VARIUS-NTV: a microarchitectural model to capture the increased sensitivity of manycores to process variations at near-threshold voltages. In: IEEE/IFIP international conference on dependable systems and networks

19. Karpuzcu UR, Sinkar A, Kim NS, Torrellas J (2013) EnergySmart: toward energy-efficient manycores for near-threshold computing. IEEE international symposium on High Performance Computer Architecture (HPCA), Shenzhen, pp 542–553
20. Karpuzcu UR, Akturk I, Kim NS (2014) Accordion: toward soft near-threshold voltage computing. IEEE international symposium on High Performance Computer Architecture (HPCA), Orlando, FL, pp 72–83
21. Patel J (2008) CMOS process variations: a critical operation point hypothesis. https://web.stanford.edu/class/ee380/Abstracts/080402-jhpatel.pdf
22. Snyder L (1986) Type architectures, shared memory, and the corollary of modest potential. In: Traub JF, Grosz BJ, Lampson BW, Nilsson NJ (eds.) Annual review of computer science, vol 1. Annual Reviews Inc., Palo Alto, pp 289–317
23. Taylor M (2012) Is dark silicon useful? Harnessing the four horsemen of the coming dark silicon apocalypse. In: ACM/EDAC/IEEE design automation conference