Accordion: Toward Soft Near-Threshold Voltage Computing *

Ulya R. Karpuzcu, Ismail Akturk
University of Minnesota, Twin Cities
{ukarpuzc,aktur002}@umn.edu

Nam Sung Kim
University of Wisconsin, Madison
nskim3@wisc.edu

Abstract

While more cores can find place in the unit chip area every technology generation, excessive growth in power density prevents simultaneous utilization of all. Due to the lower operating voltage, Near-Threshold Voltage Computing (NTC) promises to fit more cores in a given power envelope. Yet NTC prospects for energy efficiency disappear without mitigating (i) the performance degradation due to the lower operating frequency; (ii) the intensified vulnerability to parametric variation. To compensate for the first barrier, we need to raise the degree of parallelism – the number of cores engaged in computation. NTC-prompted power savings dominate the power cost of increasing the core count. Hence, limited parallelism in the application domain constitutes the critical barrier to engaging more cores in computation. To avoid the second barrier, the system should tolerate variation-induced errors. Unfortunately, engaging more cores in computation exacerbates vulnerability to variation further.

To overcome NTC barriers, we introduce Accordion, a novel, light-weight framework, which exploits weak scaling along with inherent fault tolerance of emerging R(ecognition), M(ining), S(ynthesis) applications. The key observation is that the problem size not only dictates the number of cores engaged in computation, but also the application output quality. Consequently, Accordion designates the problem size as the main knob to trade off the degree of parallelism (i.e. the number of cores engaged in computation), with the degree of vulnerability to variation (i.e. the corruption in application output quality due to variation-induced errors). Parametric variation renders ample reliability differences between the cores. Since RMS applications can tolerate faults emanating from data-intensive program phases as opposed to control, variation-afflicted Accordion hardware executes fault-tolerant data-intensive phases on error-prone cores, and reserves reliable cores for control.

1. Motivation

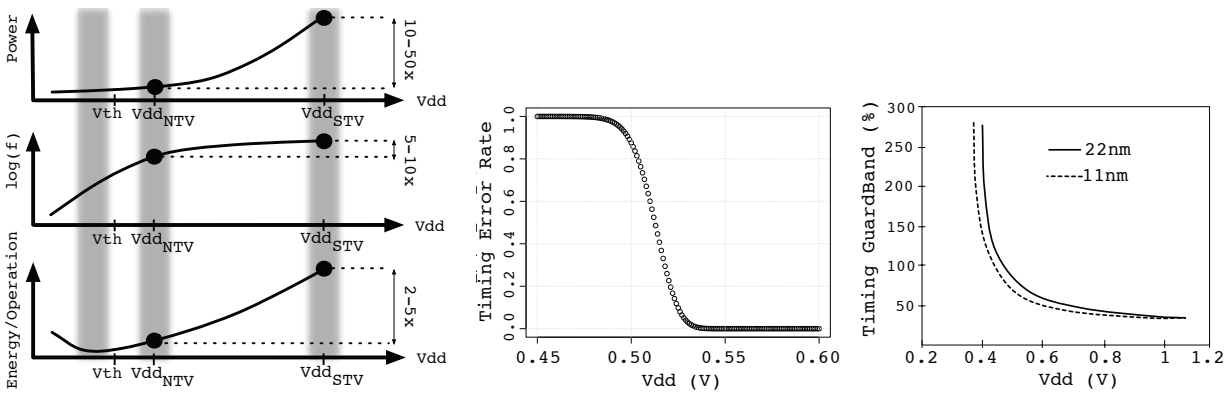
Contemporary scaling deviates from Dennard's prospects [12] by the escalating chip power density over technology generations. Still, more transistors can find place in the unit chip area each generation. However, available chip power budget cannot keep up with the growing power density due to system cooling limitations. As a result, the number of cores that we can integrate on a fixed-area chip surpasses the number of cores that we can utilize simultaneously [32]. One promising way to cram more cores into the available power budget is reducing the operating voltage `Vdd`. If `Vdd` remains slightly above the threshold voltage `Vth`, power consumption can decrease by more than an order of magnitude [13]. This unconventional regime of operation, Near-Threshold Voltage Computing (NTC), enables more cores to operate simultaneously.

Power savings increase with the proximity of the near-threshold `Vdd` to `Vth`. Unfortunately, as `Vdd` reaches `Vth`, two barriers confront us: (i) degradation in operating frequency `f` and (ii) weakened resilience to parametric variation. The first barrier restricts NTC's compatibility with latency-critical applications. Nevertheless, we can avoid degradation in throughput by boosting the degree of parallelism – the number of cores engaged in computation. Since power savings from operation at near-threshold voltages (NTV) exceed the power cost of more cores participating in computation [5], not the power budget, but the limited parallelism in the application domain constitutes the critical barrier to engaging more cores in computation.

Even if the application domain featured perfect parallel scaling, the next barrier to NTC, weakened resilience to parametric variation, would preclude vigorous `Vdd` reductions. Each technology generation, manufacturing imperfections amplify parametric variation, i.e. deviation of actual transistor parameters from design specifications. Already at conventional, super-threshold voltages (STV), variation results in slower cores, ample speed differences between the cores, and consequently, higher likelihood of errors. At lower `Vdd` transistor speed becomes more sensitive to variation. Accordingly, NTC accentuates variation-induced slowdown, speed differences, and errors. We cannot rely on worst-case timing guardbands since nominal `f` at NTV is already low. At the same time, state-of-the-art STV techniques for variation mitigation lose effectiveness when adapted at NTV [14, 21]. In the absence of NTC-specific variation mitigation techniques, a safe `Vdd` to guarantee error-free execution can barely reach the near-threshold region. To avoid the second barrier, the system should tolerate variation-induced errors.

To unlock NTC's energy-efficiency potential, `Vdd` should remain as close to `Vth` as possible, ergo we need to surmount the two critical barriers. To avoid the first barrier, more cores should contribute to computation. The corresponding expansion in the chip area, however, is likely to further exacerbate the already intensified vulnerability to variation, the second barrier. Thus, *complementary* mitigation techniques are necessary. This paper introduces *Accordion*, a novel, light-weight approach, which exploits the following characteristics of emerging R(ecognition), M(ining), and S(ynthesis) applica-

*This work was supported in part by generous grants from AMD, NSF (CCF-0953603 and CCF-1016262), MSIP GFP/(CISS-2011-00311816), DARPA (HR0011-12-2-0019). Nam Sung Kim has a financial interest in AMD.



(a) Power, f(requency), energy/operation [13, 18]. (b) Timing error rate. (c) Timing guardband [5].

Figure 1: Evolution of the operating point (a), variation-induced timing error rate (b) and timing guardband (c) with Vdd.

tions [7]:

I. Compliance with Gustafson-Barsis Law [16], a.k.a. weak scaling, where the problem size naturally expands (shrinks) as the application scales to utilize more (less) cores. RMS applications are largely compute-intensive [2]. Often, compute-intensive applications cannot accommodate problem sizes of interest due to an unacceptably long execution time. Accordingly, only problem sizes taking no longer than the maximum tolerable execution time become practical. The maximum tolerable execution time – as determined by machine utilization policies in a shared cluster, e.g. – induces a time budget which remains mainly constant. Hence, rather than accelerating the execution of a problem of fixed size, solving a problem of larger size within this constant time budget matters [31].

II. Intrinsic fault tolerance due to processing of massive, yet noisy and redundant data by probabilistic, often iterative, algorithms. Usually the solution space has many more elements than one, deeming a range of application outputs valid as opposed to a single "golden" output [8].

The key observation is that the problem size not only dictates the number of cores engaged in computation, but also the application output quality. Consequently, Accordion designates the problem size as the main knob to trade off the degree of parallelism (i.e. the number of cores engaged in computation), with the degree of vulnerability to variation (i.e. the corruption in application output quality due to variation-induced errors). Parametric variation renders substantial reliability differences between cores. Since RMS applications can tolerate faults emanating from data-intensive program phases as opposed to control-intensive [10], variation-afflicted Accordion hardware executes fault-tolerant data-intensive phases on error-prone cores, and reserves reliable cores for control. The main contributions are as follows:

- Accordion, a novel, light-weight framework to overcome NTC barriers by devising the problem size as the main knob to control the degree of parallelism along with the degree of vulnerability to variation.
- Architectural design space exploration for the Accordion hardware to decouple the processing of fault-tolerant data-intensive program phases from fault-sensitive control-intensive program phases of RMS applications.

The rest of the paper is organized as follows: Section 2 provides the background; Section 3 introduces the Accordion framework; Section 4 explores the architectural design space; Sections 5 and 6 evaluate Accordion; Section 7 summarizes our findings; and Section 8 covers related work.

2. Background

2.1. NTC Basics

For the current technology generation, NTC corresponds to $\texttt{Vdd} \approx 0.5$V, conventional STC, to 1V. Figure 1a depicts power, frequency f, and energy per operation as a function of Vdd. Energy per operation improves by about 2-5× over STC – at the expense of a 5-10× f degradation [13, 18]. The corresponding power reduction by 10-50× enables more cores 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. Super-threshold region (STV) accommodates the maximum f 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.

2.2. NTC Barrier: Performance Degradation

To unlock NTC's energy efficiency potential, Vdd should remain as close to Vth as possible. The lower the degree of parallelism – the number of cores engaged in computation, the bigger would the difference be between Vdd and Vth. Lower degrees of parallelism enable fewer cores to operate, and a few number of cores may not suffice to compensate for a higher f degradation as incurred by a lower Vdd.

$$\texttt{Execution Time} \propto \frac{\frac{\texttt{Work per parallel task}}{\texttt{f}}}{\frac{\texttt{Problem Size/N}}{\texttt{f}}} \quad (1)$$

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` (Equation 1). Accordingly, for a fixed `problem size`, in order to offset the NTC-induced `f` degradation of 5-10× from Figure 1a 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 becomes whether the fixed problem size can render enough work to keep 5-10× more cores busy. Thus, not the power budget, but the limited parallelism in the application domain constitutes the critical barrier to engaging more cores in computation. Even if the application domain featured abundant parallelism, the corresponding expansion in the chip area is likely to further exacerbate NTC's already intensified vulnerability to variation.

2.3. NTC Barrier: Vulnerability to Parametric Variation

Each technology generation, manufacturing imperfections exacerbate vulnerability to parametric variation, deviation of transistor parameters from nominal specifications. Already at STV variation results in not only slower cores, but also ample speed differences between the cores. At lower `Vdd` transistor speed becomes more sensitive to variation. Therefore, NTC accentuates variation-induced slowdown and speed differences. Accordingly, the likelihood of variation-induced timing errors increases as `Vdd` reaches `Vth` (Figure 1b). Timing errors emerge if variation slows 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 speed than sustainable were there no variation. This slowdown to guarantee error-free execution constitutes the *timing guardband*. Figure 1c depicts the timing guardband as a function of `Vdd`, considering contemporary (22nm) and near-future (11nm) 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 does not represent a practical design option at NTV. A further difficulty stems from the diminishing efficacy of state-of-the-art STV variation mitigation techniques when adapted at NTV [14, 21]. We need NTC-specific variation mitigation techniques, otherwise a safe `Vdd` to guarantee error-free execution can barely reach the near-threshold region.

2.4. Essential Characteristics of RMS Applications

In overcoming NTC barriers, Accordion leverages two intrinsic characteristics of RMS applications: (i) compliance with weak scaling, (ii) inherent fault tolerance.

Compliance with Weak Scaling (Gustafson-Barsis Law): 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) [16]. Amdahl's Law hits strong scaling limits [28]. At the same time, compensation for a typical NTC-induced `f` degradation 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 (as determined by machine utilization policies in a shared cluster, e.g.) 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 [31]. RMS applications are largely compute-bound [2], thus conform to weak scaling: the problem size tends to scale with the number of cores engaged in computation.

Inherent Fault Tolerance: RMS 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 [8]. Therefore, RMS applications can tolerate faults emanating from data-intensive program phases as opposed to control [10].

3. Basics of Accordion Operation

3.1. Main Idea

RMS applications can mask errors in data-intensive program phases as opposed to control. However, to be able to embrace errors

- errors should be confined to fault-tolerant data-intensive phases such that they manifest as degradation in accuracy, i.e. quality (`Q`) of computing;
- the degradation in output quality should remain within acceptable boundaries.

To meet the first condition, variation-afflicted Accordion hardware executes fault-tolerant data-intensive phases on error-prone cores, and reserves reliable cores for control. In other words, Accordion enforces variation-induced errors to be contained where they can be tolerated: within data-intensive program phases. Thread decomposition of most RMS algorithms already conforms to the view of decoupled data and control, and can accordingly be exploited to distinguish data-intensive parallel tasks from control-intensive.

The second condition, on the other hand, cannot be satisfied without the capability to control or configure the output quality explicitly. The system is expected to delimit `Q` degradation. An application-specific set of input parameters such as time step granularity or resolution can be devised to serve the purpose. Such input parameters to control `Q` usually associate with the problem size [9, 11]: by expanding the problem size, we can configure the application to generate an output of higher `Q`. Therefore, a larger problem size can not only facilitate a lower operating `Vdd` by engaging more cores to computation, but also a higher tolerance to variation-induced errors where the

Goal: $\texttt{Execution Time}_{NTV} \rightarrow \texttt{Execution Time}_{STV}$

Mode	Problem Size	Number of Cores	Quality	Operating frequency	
				Safe	Speculative
Still	$\texttt{Problem Size}_{NTV} = \texttt{Problem Size}_{STV}$	$N_{NTV} > N_{STV}$	$Q_{NTV} \leq Q_{STV}$	$f_{NTV} \leq f_{NTV,Safe}$	$f_{NTV} > f_{NTV,Safe}$
Compress	$\texttt{Problem Size}_{NTV} < \texttt{Problem Size}_{STV}$	No restriction			
Expand	$\texttt{Problem Size}_{NTV} > \texttt{Problem Size}_{STV}$	$N_{NTV} > N_{STV}$			

Table 1: Basic Accordion modes of operation. f_{STV} corresponds to the nominal STV f; $f_{NTV,Safe}$, to the highest possible NTV f to exclude timing errors; f_{NTV}, to the Accordion f at NTV. $f_{STV} > f_{NTV} \geq f_{NTV,Safe}$ applies.

vulnerability to variation increases due to the lower Vdd. The degraded Q in the presence of variation would remain higher than its counterpart under a smaller problem size.

Figure 2 demonstrates, for two representative RMS benchmarks from optimization (canneal) and physics simulation (hotspot) domains, how Q changes with the problem size under three different scenarios. Under Default, all parallel tasks assigned to computation actually contribute to computation: Q increases with problem size monotonically, although the sensitivity to problem size may vary. Accordingly, larger problem sizes can accommodate higher error rates. hotspot exhibits a higher sensitivity to the problem size. Thus, the same Δ increase in problem size would lead to a larger Q improvement, enabling operation at higher error rates.

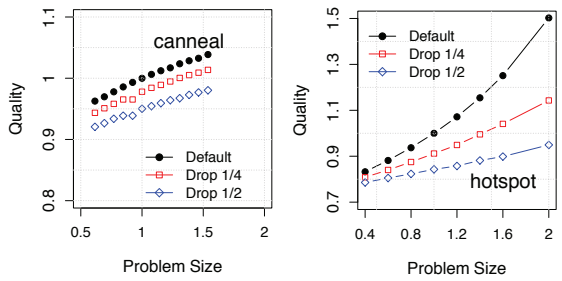


Figure 2: Impact of problem size on quality of computing.

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. Even Drop 1/2 does not render an excessive Q degradation.

canneal 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. Total number of temperature steps, max temperature steps, and swaps per temperature step represent input parameters. The product thereof governs the problem size:

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

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 accuracy of computing. Thus, changes in the problem size as a function of such input parameters would not result in a different problem, but the same problem to render a different solution accuracy.

Based on the observation that the problem size not only dictates the number of cores engaged in computation, but also the corruption in application output due to variation-induced errors, Accordion designates the problem size as the main knob to trade off the degree of parallelism with the degree of vulnerability to variation.

3.2. Accordion Modes

Under contemporary scaling, system cooling limitations render a practically constant power budget for a fixed-area many-core. Factoring in the plateaued out f over technology generations, performance improvements can only come from increasing the core count engaged in computation, N. As captured by Equation 2, however, the constant power (budget) cannot accommodate more cores unless energy efficiency, i.e. performance per Watt, increases:

$$\texttt{Power} \times \texttt{Energy Efficiency} = \texttt{Performance} \propto f \times N \quad (2)$$

NTC enhances energy efficiency by reducing Vdd to reach Vth, where power decreases more than performance even if N increases to compensate for the f degradation. The optimal core count engaged in computation strongly depends on the proximity of Vdd to Vth. Accordingly, Accordion modulates N along with the near-threshold Vdd and f to have the performance, i.e. the execution time at NTV converge to the execution time at STV without hurting energy efficiency:

$$\underbrace{\frac{\texttt{Problem Size}_{NTV}}{f_{NTV} \times N_{NTV}}}_{\propto \texttt{Execution Time}_{NTV}} \xrightarrow{\texttt{Accordion}} \underbrace{\frac{\texttt{Problem Size}_{STV}}{f_{STV} \times N_{STV}}}_{\propto \texttt{Execution Time}_{STV}}$$

Accordion supports various modes of operation, each characterized by a distinct way to bring the NTV execution time as close as possible to the STV execution time. Depending on how the problem size *accords* with the number of cores, Accordion differentiates between three basic modes of operation: Still, Compress, Expand (Table 1).

Still Mode strictly follows strong scaling semantics by keep-

ing the problem size intact. Due to $\texttt{Problem Size}_{NTV}$= $\texttt{Problem Size}_{STV}$, the NTV core count N_{NTV} should increase by at least f_{STV}/f_{NTV} to retain the STV execution time.

Compress Mode gets its name from a compressed problem size, which helps the low f_{NTV} accommodate the STV execution time. As long as per core work $\propto$ Problem Size/N reduces proportional to f_{NTV}/f_{STV} , Accordion can conserve the STV execution time at NTV. 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, Accordion can reduce per core work by f_{NTV}/f_{STV} by carefully modulating N.

Expand Mode gets its name from an expanded problem size. For the low f_{NTV} to accommodate the STV execution time, N should increase more than the problem size does such that per core work $\propto$ Problem Size/N reduces. However, in order to come close to the STV execution time, per core work should reduce by f_{NTV}/f_{STV}. This translates into N increasing by $f_{STV}/f_{NTV} \times \texttt{Problem Size}_{NTV}/\texttt{Problem Size}_{STV}$. 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 ($f_{NTV,Safe}$) can help. A higher f_{NTV} than safe would raise the likelihood of variation-induced timing errors, yet the expanded problem size can make up for the corresponding Q degradation.

3.3. Putting It All Together

The basic Accordion modes of Table 1 come in two flavors, depending on how f_{NTV} is set:

- **Safe ($f_{NTV} \leq f_{NTV,Safe}$)** excludes quality degradation due to variation-induced errors by imposing a safe operating f. However, Compress may still result in degraded output quality due to the compressed problem size.
- **(Timing) Speculative ($f_{NTV} > f_{NTV,Safe}$)**, on the other hand, by imposing a higher operating f than safe, suffers from quality degradation due to the inevitable onset of variation-induced timing errors. Speculative is especially feasible under Expand, where the expanded problem size enhances Q by construction, hence can be configured to reduce quality degradation in the presence of errors.

Safe variants of Compress which exclude variation-induced errors may lead to similar quality degradation (due to the compressed problem size) as Speculative variants of Still or Expand which embrace errors. Compress is also the only mode where N_{NTV} can remain less than N_{STV}. This can specifically be useful in heavily loaded multi-programmed environments.

Speculative variants represent a more typical use case for Expand. However, since Accordion modulates the problem size along with the number of cores, even under Safe variants of Expand where the operating f is not increased, Accordion would not necessarily create more work for already slow NTV cores – as long as N increases more than the problem size does.

4. Architectural Design Space Exploration

Accordion runs all cores engaged in (data-intensive) computation at the same 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 speeds. 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. The lower number of active cores gives Accordion more freedom in picking the cores. 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 Q degradation.

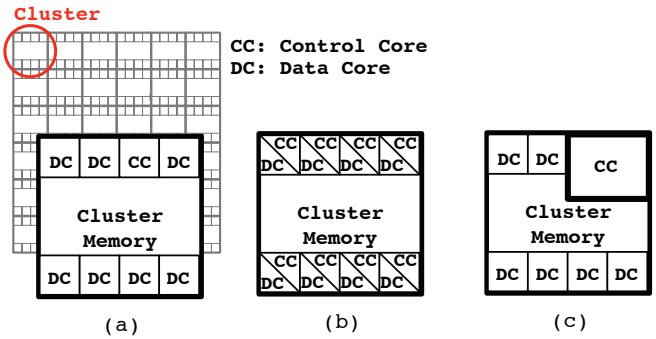


Figure 3: Accordion design space.

4.1. Decoupling Processing of Data from Control

Being tailored to work at high error rates, Accordion enables operation at higher frequencies than $f_{NTV,Safe}$. This gives rise to Speculative variants of Accordion modes to embrace variation-induced timing errors. As a safety net Accordion can still rely on checkpoint-recovery, yet of significantly reduced complexity due to the anticipated decrease in the frequency of checkpointing and recovery.

Accordion enforces variation-induced errors to be contained where they can be tolerated. The architecture executes fault-tolerant data-intensive program phases on error-prone **D**ata **C**ores, and reserves more reliable **C**ontrol **C**ores for control. CCs and DCs work in master-slave mode, where each CC coordinates computation on a designated set of DCs.

Control Core (CC) Semantics: 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. At the same time, we can impose a higher operating Vdd, or a variation-afflicted Accordion chip 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.

Data Core (DC) Semantics: To facilitate effective coordination with Control Cores, Data Cores feature fast reset and restart hardware. A DC has access to a private read-write memory where the DC can write (either organized as a scratch-pad or to communicate with other cores), 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.

4.2. Accordion Design Space

Figure 3 demonstrates an Accordion chip, 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.

Homogeneous Clusters (Figure 3a, b): Each cluster accommodates a set of identical cores. There is no difference in the design of CCs and DCs. The first option, as shown in Figure 3a, differentiates CCs from DCs *spatio-temporally*: CCs correspond to the fastest, most reliable cores in the variation-afflicted Accordion chip. This option is simpler from the hardware perspective, yet the semantics from Section 4.1 should all be programmed. Still, the organization is very flexible in that the number of CCs can be configured, although the example organization from Figure 3a depicts one CC per cluster. Since variation governs the `f` of CCs, the range of CC frequencies may differ across chips. The second option, as captured in Figure 3b, on the other hand, features *temporal* differentiation of CCs from DCs. 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.

Heterogeneous Clusters (Figure 3c): CCs and DCs per cluster represent different types of cores by design. In this case DCs and CCs specialize; the semantics from Section 4.1 can be directly implemented in hardware. 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 Figure 3c 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.

5. Evaluation Setup

5.1. Technology and Architecture

To quantify the impact of Accordion operation, we deploy a hypothetical NTV chip of 288 cores organized in 36 clusters at 11nm. This chip mimics the Accordion organization from Figure 3a (Section 4.2). Each cluster has 8 cores and a shared per-cluster memory block. Each core is a single-issue engine where memory accesses can be overlapped with computation. A bus inside a cluster and a 2D-torus across clusters constitute the network.

System Parameters	
Technology node: 11nm # cores: 288 # clusters: 36 (8 cores/cluster)	P_{MAX}= 100W T_{MIN}= 80°C Chip area ≈ 20mm x 20mm
Variation Parameters	
Correlation range: $\phi = 0.1$ Total $(\sigma/\mu)_{Vth} = 15\%$	Sample size: 100 chips Total $(\sigma/\mu)_{Leff} = 7.5\%$
Technology Parameters	
Vdd_{NOM}= 0.55V Vth_{NOM}= 0.33V	f_{NOM}= 1.0GHz $f_{network}$= 0.8GHz
Architectural Parameters	
Core-private mem: 64KB WT, 4-way, 2ns access, 64B line Network: bus inside cluster and 2D-torus across clusters	Cluster mem: 2MB WB, 16-way, 10ns access, 64B line Avg. mem round-trip access time (without contention): ≈80ns

Table 2: Technology and architecture parameters.

The nominal values of `Vdd` and `f` are 0.55V and 1.0GHz (which approximately correspond to 1V and 3.3GHz for STC). Table 2 covers the technology and architecture parameters. Technology parameters are derived from ITRS [1] and fine-tuned considering industry projections for 11nm. To evaluate performance we use ESESC [19] with the power analysis relying on McPAT [22] scaled to 11nm. The power budget is fixed to 100W.

We experiment with VARIUS-NTV [20] to extract Vdd_{MIN}, the minimum near-threshold `Vdd` each memory block can support. To have blocks stay functional at NTV, any designated near-threshold operating `Vdd` should remain higher than such Vdd_{MIN}. We further use VARIUS-NTV to determine safe operating frequencies, and to estimate timing error rates as a function of the operating frequency at the designated near-threshold `Vdd`.

5.2. Benchmarks

We experiment with select benchmarks from PARSEC [3] and Rodinia [6] suites to cover emerging application domains. Table 3 captures benchmark characteristics. For each benchmark, we identify at least one application input parameter (Accordion input) governing both, the problem size and the accuracy. We rely on application-specific accuracy metrics to measure quality of computing. Applications usually generate multiple numeric values as output, most of the time in vector or matrix format. Accordingly, the quality metric should capture the deviation in each numeric output value to derive the cumulative deviation from the baseline execution. To this end, we rely on the `distortion` metric introduced by Misailovic et al. [26]: Distortion is defined as the average, across all output values, of `relative error` per output value. The relative quality of an execution outcome thus becomes `1-distortion`. Distortion is a generic metric, yet its calculation is application

Benchmark	Application domain	Quality metric	Accordion input	Dependence on Accordion input	
				Problem Size	Quality
canneal (PARSEC)	Optimization	Relative routing cost	Swaps per temperature step	linear	linear
			Number of temperature steps	linear	linear
ferret (PARSEC)	Similarity search	Based on number of common images	Size factor	complex	complex
bodytrack (PARSEC)	Computer vision	SSD based	Number of annealing layers	complex	complex
x264 (PARSEC)	Multimedia	SSIM based	Quantizer	complex	linear
hotspot (Rodinia)	Physics simulation	SSD based	Number of iterations	linear	linear
srad (Rodinia)	Image processing	PSNR based	Number of iterations	linear	linear

Table 3: RMS benchmarks deployed from PARSEC [3] and Rodinia [6] suites.

dependent. The application dependence mainly comes from how `relative error` is calculated.

`canneal` relies on simulated annealing to minimize the chip routing cost. At each temperature step, `swaps_per_temp` times, each thread attempts to swap two randomly picked blocks on chip. Both, the total number of temperature steps, and the number of swaps per temperature step (`swaps_per_temp`) represent Accordion inputs. In the following, we use the latter without loss of generality. Relative routing cost captures quality.

`ferret` conducts content-base similarity search in an image database. To extract similarity, images are partitioned and processed in regions. The number of regions dictates work per thread, and depends on the minimum region size calculated by `number of pixels`×`size_factor` (Accordion input). A pre-set number (`n`) of similar images found, on a per query basis, constitutes the output. `relative error` (per query) becomes `1-[common image count]/n`, where common image count is the number of output images common with the baseline outcome. The configurations we experiment with involve multiple queries.

`bodytrack` relies on an annealed particle filter to track human movement through a scene. Number of annealing layers (Accordion input) affects both, filtering accuracy, and the problem size. The output is a vector of tracked configurations. Distortion calculation relies on SSD (sum of squared distance).

`x264` implements the H.264 standard for video encoding. `QP`, the quantizer to control the degree of compression on encoding is the Accordion input. A smaller `QP` leads to less compression, hence a higher accuracy. Distortion calculation is based on SSIM (structural similarity index), a metric shown to match human perception of changes in accuracy better than PSNR (peak signal to noise ratio).

`hotspot` is a thermal simulator to iteratively solve heat transfer differential equations for a given floorplan. The number of iterations represents the Accordion input. The output is the temperature at each point of a grid super-imposed on the floorplan. Distortion calculation relies on SSD, sum of squared difference of temperatures across grid cells.

`srad` implements an algorithm to effectively remove correlated noise from imaging applications, based on an iterative PDE (partial differential equation) solver. We designate the number of iterations as the Accordion input. Distortion calculation is based on PSNR.

Table 3 also tabulates, for each benchmark, how sensitive problem size and `Q` are to changes in Accordion inputs. We identify two broad types of dependencies: Linear, and complex to capture mainly super-linear behavior. Different combinations apply across different benchmarks, however, except `x264`, problem size and `Q` tend to assume the same type of dependency on the designated Accordion input, on a a per benchmark basis. The type of the dependencies determine how well different benchmarks respond to Accordion operation.

6. Evaluation

To understand the basics of Accordion operation, we start with how variation-induced timing error rates evolve as a function of operating `f`. We next analyze how the quality of computing changes with the problem size. We conclude this section with a detailed characterization of the operating points under `Safe` and `Speculative` modes of Accordion.

6.1. Impact of Parametric Variation

To quantify the variation-induced slowdown and the resulting timing error rates across the cores of the hypothetical NTV chip, we first need to determine the near-threshold operating `Vdd`, $\texttt{Vdd}_{\texttt{NTV}}$. To this end, we deploy VARIUS-NTV to extract the minimum `Vdd`, $\texttt{Vdd}_{\texttt{MIN}}$, each cluster can support to remain functional at NTV. If we let the system operate below such $\texttt{Vdd}_{\texttt{MIN}}$, memory blocks may not be able to hold or change state. Figure 5a shows the distribution of per-cluster $\texttt{Vdd}_{\texttt{MIN}}$ for one representative chip (out of the 100 we experiment with): Per-cluster $\texttt{Vdd}_{\texttt{MIN}}$ is defined as the maximum $\texttt{Vdd}_{\texttt{MIN}}$ across all memory blocks within a cluster. The data is shown as a histogram. Per-cluster $\texttt{Vdd}_{\texttt{MIN}}$ values vary in a significant 0.46-0.58V range. We designate the maximum per cluster $\texttt{Vdd}_{\texttt{MIN}}$ as the chip-wide $\texttt{Vdd}_{\texttt{NTV}}$.

We assign tasks to cores at the granularity of clusters. Each cluster constitutes an `f` domain. The slowest core within each cluster determines the operating `f` of the cluster. Figure 5b depicts variation-induced timing error rate per cycle, `Perr`, when operating at $\texttt{Vdd}_{\texttt{NTV}}$, 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, rendering 36 curves for 36 clusters. The y axis is on log-scale. `Perr` values rapidly increase to reach 1, as `f` increases beyond 0.5GHz. Even at acceptably low `Perr` of [1e-16,1e-12], the majority of the cores cannot operate at the NTV $\texttt{f}_{\texttt{NOM}}$ of 1GHz – the sustainable `f` were there no variation. `Perr` in the range of

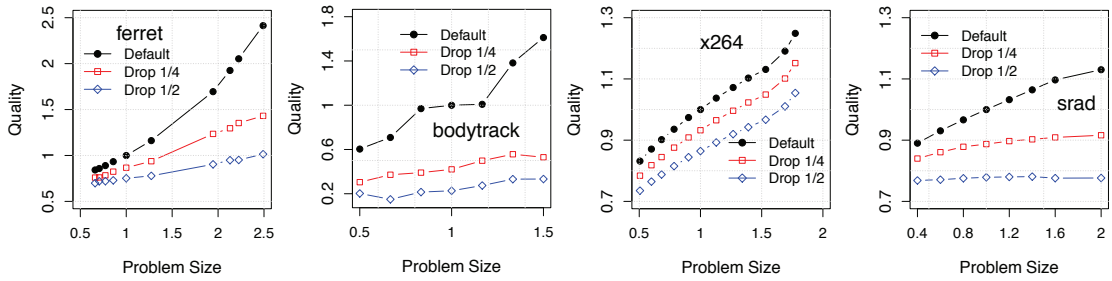


Figure 4: Impact of problem size on application output quality for various RMS applications.

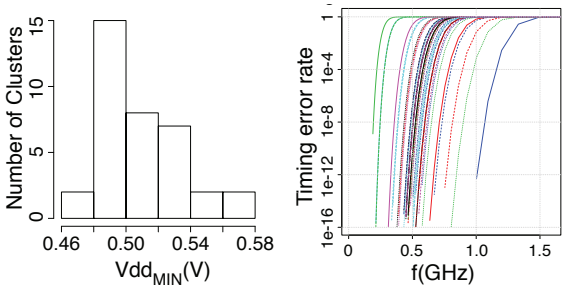


Figure 5: Impact of parametric variation.

[1e-16,1e-12] induce a timing error every [1e16,1e12] cycles. According to Figure 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}.

6.2. Impact of Problem Size on Quality of Computing

To demonstrate how the quality of computing changes as a function of the problem size, we monotonically increase the Accordion inputs from Table 3. We track the corresponding changes in the problem size and the quality. Figures 2 (for `canneal` and `hotspot`) and 4 (for the rest of the benchmarks) provide the resulting quality (y-axis) vs. problem size (x-axis) fronts. The axes are normalized to the quality and problem size for the default Accordion input values under simsmall for PARSEC, and as provided, for Rodinia benchmarks. We profile the benchmarks under 64 threads with the exception of srad (profiled under 32 threads).

Each distinct Accordion input value corresponds to one point on the fronts. For each value considered, we first calculate the quality (Table 3). Per Section 5.2, quality represents a relative entity. Accordingly, we calculate the quality with respect to a "hyper-accurate" execution outcome (which characterizes the maximum achievable accuracy). We normalize these quality values to the quality for the default values of Accordion inputs to generate the y-axis of Figures 2 and 4.

We conduct three different experiments: The first set correspond to `Default` execution, where we let all parallel tasks assigned to computation actually contribute to computation. Tracking the `Default` fronts, we observe that `Q` increases with problem size monotonically, although its sensitivity to problem size varies across different benchmarks. Accordingly, larger problem sizes can accommodate higher error rates.

Next two scenarios mimic a close-to-worst-case manifestation of errors by preventing a quarter (`Drop 1/4`), and a half (`Drop 1/2`) of the parallel tasks assigned to computation from actually contributing to computation – the tasks are uniformly dropped [1]. `Drop` conservatively ignores potential masking of faults at various levels of the system stack: Any timing fault of probability `Perr` reaches the application layer to corrupt the end output each thread generates. The end result of any infected thread is ignored all together.

Under the onset of errors, `Q` still increases monotonically with the problem size. With the exception of `bodytrack`, `Q` degradation does not become excessive even if half of the threads are dropped. For `bodytrack`, the monotonicity seems to be broken for the second data point on `Drop 1/2`, and seventh data point on `Drop 1/4` (in the increasing direction of problem size). These slight drops are due to non-determinism in the executions, however, the excessive `Q` degradation under `Drop` suggests a higher sensitivity of `Q` to errors.

For `hotspot` and `ferret`, `Q` exhibits a higher sensitivity to the problem size than for `canneal` and `srad`. Thus, the same Δ increase in problem size would lead to a larger `Q` improvement, enabling operation at higher `Perr`. Higher `Perr` associates with a higher operating `f`, hence the same execution time can be achieved at a lower core count. This further implies power savings: Power is more sensitive to core count than `f`, since core count increases both static and dynamic power where `f` only affects the dynamic component. At the same time, the share of static power is higher at NTV.

The `Q` difference between `Default`, `Drop 1/2` and `Drop 1/4` tends to increase as the problem size increases with the exception of `canneal` and `x264`. This is because we experiment with a fixed thread count. As the problem size increases, so does (dropped) work per thread. For `canneal` and `x264`, on the other hand, the dropped work does not seem to be as critical for `Q`. Due to the probabilistic nature of `canneal` we may

[1] `Drop` is enforced for `canneal`, by preventing `swap()`; for `bodytrack`, at `ParticleFilterPthread:Exec()` to prevent row and column filtering, and at `TrackingModelPthread:Exec()` to prevent particle weight calculation; for `x264`, at `x26_slice_write()` to prohibit the encoding of a macro block; for `hotspot` by preventing solution of the temperature equation and update of the corresponding cell temperature; for `srad` by preventing calculation of directional derivatives, ICOV, diffusion coefficients, along with divergence and image update in each iteration.

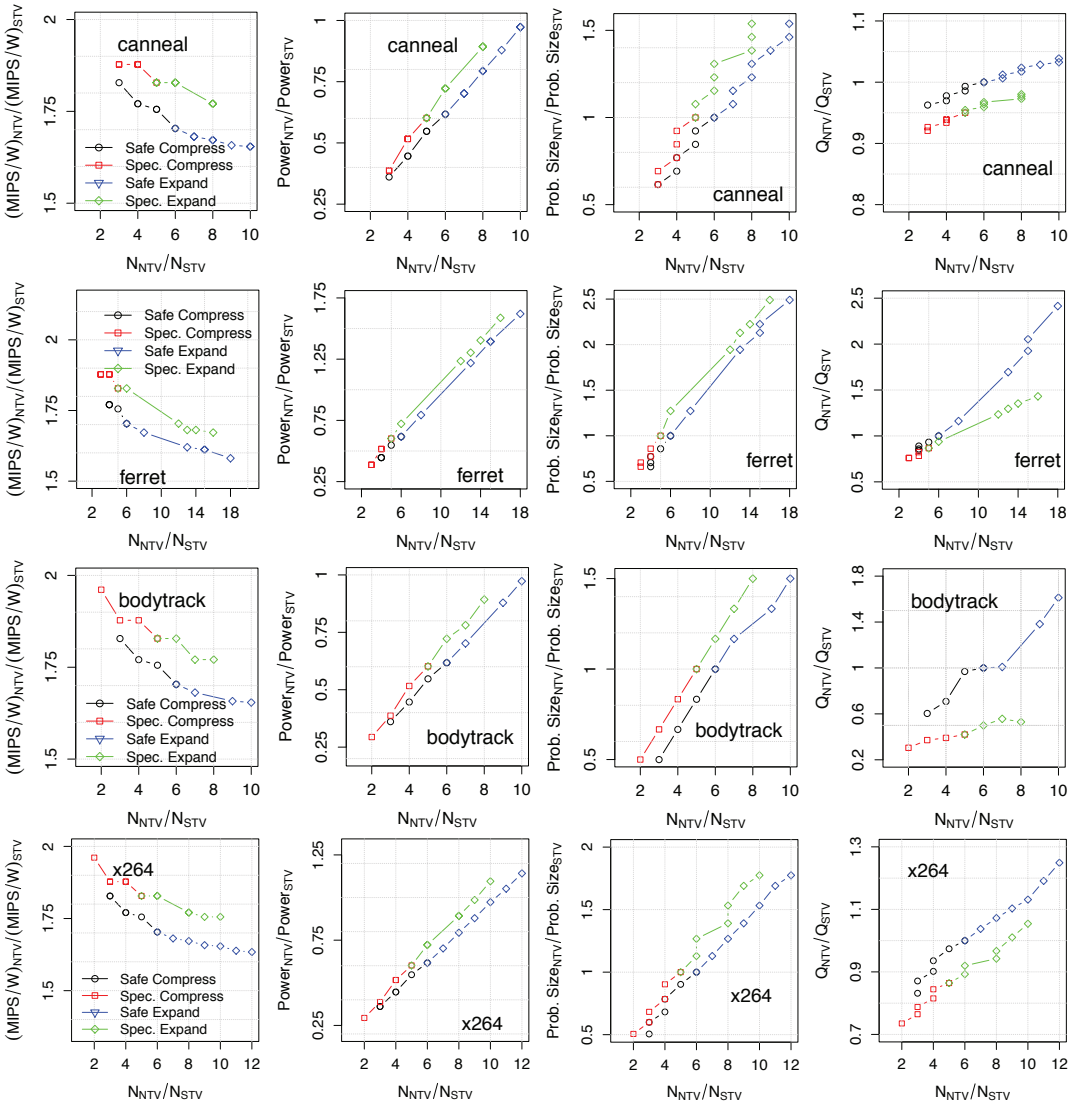


Figure 6: Iso-execution time fronts for canneal, ferret, bodytrack and x264.

still achieve an accurate-enough solution even if a significant fraction of per thread work (i.e. swaps) are dropped.

Error Model Semantics: `Drop` can capture close-to-worst case error manifestation *under the decoupled execution model Accordion assumes*. The exhaustive set of potential manifestations of variation-induced errors can be summarized as (i) no termination (due to crashes or hangs); (ii) termination with excessive `Q` degradation; (iii) termination with acceptable `Q` degradation. Accordion execution model relies on CCs to detect (i), e.g. by deploying watchdog timers. The application layer would perceive (i) mainly as `Drop`. (ii) points to degradation of data-intensive phases to lead to unacceptable accuracy. We envision CCs to capture (ii) by enforcing preset limits on maximum `Q` degradation (as defined by the application developer, e.g.). Threads not conforming to such limits can be treated exactly as threads leading to (i), as mimicked by `Drop`.

On the other hand, (iii) does not require CCs intervention. In the above analysis we assume that (iii) cannot lead to higher `Q` degradation than (i), purely relying on inherent algorithmic fault tolerance of RMS applications. To validate the accuracy of this assumption, instead of ignoring the end result of infected threads, we corrupted the end result in various ways: all bits/higher order bits only/lower order bits only stuck-at-1(0)/randomly flipped/inverted. We injected errors to the end result each thread generates to exclude any system-level masking of errors. We observed that corruption under these error modes generally does not fall below the corruption under `Drop`, unless these error modes lead to excessive corruption to have the corresponding executions binned under (ii) rather than (iii). For `canneal`, for example, we injected these errors into the decision variables which control whether a `swap` should be conducted or not. None of the error modes rendered a lower accuracy than `Drop`. Only when we tried to invert the decision

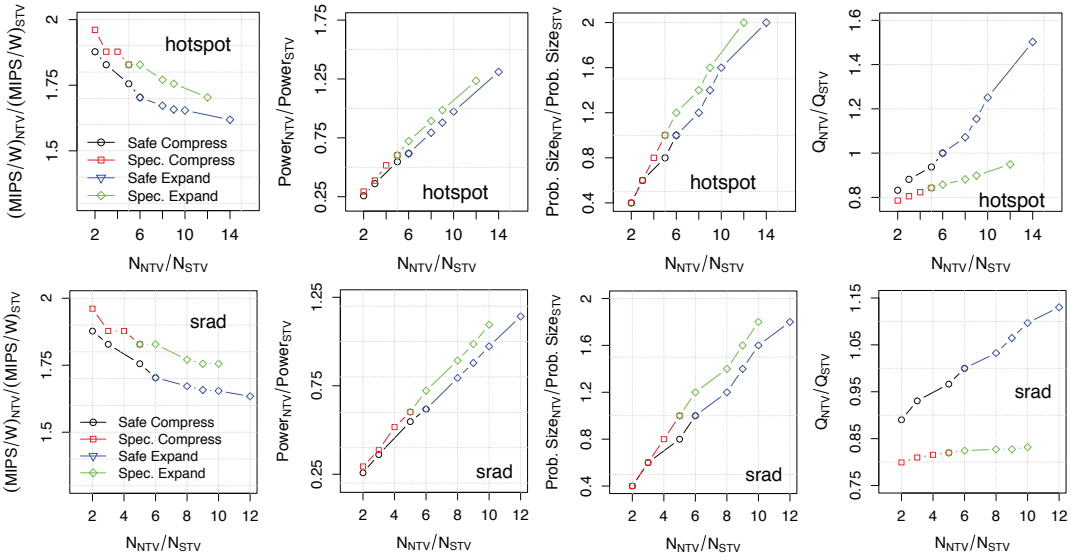


Figure 7: Iso-execution time fronts (cont.) for hotspot and srad.

semantics, such that a `swap` is accepted where it shouldn't be and vice versa, we observed a `Q` degradation of 77% (a quarter of threads uniformly infected) and 69% (half of the threads uniformly infected) over the nominal, where `Drop` renders 98% and 96%, respectively.

6.3. Accordion Operation

To characterize Accordion operation, we analyze how the operating point and the core count should evolve, as a function of the problem size, in order to have $\texttt{Execution Time}_{NTV}$ converge to $\texttt{Execution Time}_{STV}$. Accordingly, we extract the *iso-execution time* pareto front for each benchmark. Each point on the pareto front characterizes a distinct problem size, and indicates how N_{NTV} and f_{NTV} should be set to have $\texttt{Execution Time}_{NTV}$ converge to $\texttt{Execution Time}_{STV}$. In order visualize the multi-dimensional iso-execution time pareto front, we project it to two-dimensions. This results in four graphs per benchmark, as captured by each row of Figures 6&7. The y axis depicts energy efficiency in MIPS/W(att) in the first column; power, problem size, and quality, in the following columns, all normalized to the STV baseline [2]. The x axis, on the other hand, reflects N_{NTV} normalized to N_{STV}. N_{STV} is the maximum number of cores that can fit into the power budget at the STV Vdd_{NOM}. $\texttt{Execution Time}_{STV}$ is obtained at the default problem size, N_{STV}, and the STV f_{NOM}. Accordion targets an execution time no larger than $\texttt{Execution Time}_{STV}$. We impose the same power budget at NTV and STV.

In generating the pareto front, if a particular problem size demands a core count of N_{NTV} to achieve iso-execution time operation, we pick the most energy-efficient N_{NTV} cores from the variation afflicted chip. These cores can deliver more performance per Watt. At the same time, N_{NTV} dictates how f_{NTV} should be set, since we let all cores assigned to a parallel task operate at the same `f` (Section 4). To prevent the onset on timing errors under `Safe`, the slowest core among N_{NTV} cores determines f_{NTV}. On the other hand, we favor STV operation by neglecting the impact of variation at STV. Hence, f_{STV} assumes a larger value than actual.

The first column from Figures 6&7 shows the energy efficiency of the pareto front across the benchmarks considered. `Speculative` and `Safe` give rise to two different pareto fronts. Pareto points under `Expand` and `Compress` modes are distinctly marked on each. Each pareto point corresponds to a different problem size. The problem size increases monotonically as we move from left to right on the pareto front. Hence, the default problem size, which corresponds to `Still`, is to be located at the intersection of `Compress` and `Expand`. The y coordinate of each pareto point captures the energy efficiency, the x coordinate, the core count required to achieve $\texttt{Execution Time}_{STV}$. The following columns report (normalized) power, problem size, and quality of the very same pareto points by changing the y coordinates while the x coordinates remain intact.

`Safe` Modes: Examining the first columns from Figures 6&7, we observe that MIPS/W improvement over STV across all benchmarks remains less than 2×. MIPS/W corresponds to the inverse of energy/operation. This improvement remains much less than the expected 2-5× improvement from Figure 1a. Figures 6&7 report *iso-execution-time* energy-efficiency, while Figure1a assumes a significantly slowed down (i.e. *degraded-execution-time*) NTV execution. At the same time, Figure1a characterizes the expected potential from NTV operation under ideal conditions, hence the ranges reported rather correspond to maxima.

`Compress` and `Expand` iso-execution-time pareto fronts under `Safe` mode are labeled as `Safe Compress` snd `Safe Expand` in Figures 6&7. Points on these pareto fronts all corre-

[2] A different STV baseline applies to each application mainly due to the difference in quality vs. problem size characteristics. The "hyper-accurate" execution outcome used for quality calculations is also application-specific.

spond to the same (STV) execution time, and each characterize a distinct value of the problem size (increasing from left to right). Safe Compress and Safe Expand fronts intersect at the pareto point corresponding to Still. We observe that points on the Safe Compress pareto front can achieve the same execution time at lower core counts when compared to the points on the Safe Expand pareto front, mainly due to the compressed problem size.

Under Safe Expand the core count N should increase more than the problem size does, such that the low NTV f can achieve the same execution time as at STV. A higher N not only increases power (second column of Figures 6 &7), but also decreases the operating f. This is because we let the set of cores assigned to an application operate at the f of the slowest core in the set, and as N increases chances get higher for slower cores to get engaged in computation. The end result is a degrading MIPS/W with increasing N (first column of Figures 6&7). This phenomenon generally applies as the core count increases, both, under Safe Expand and Safe Compress execution.

As N increases, Safe Expand may hit a core count where the f of the slowest core no more suffices to satisfy the STV execution time. This is not the case for Safe Compress: Since the problem is smaller, we can accommodate a lower f to achieve the STV execution time, hence we are less restricted by the operating f of a slow core as we increase the number of cores.

Due to the lower core count, operation under Safe Compress generally assumes a higher f_{NTV} than under Safe Expand. However, although f_{NTV} is higher, Safe Compress consumes less power than Safe Expand (second column of Figures 6&7). This is mainly because the same execution time can be achieved at a lower core count. Power is more sensitive to core count than f, since core count increases both static and dynamic power where f only affects the dynamic component. At the same time, the share of static power is higher at NTV.

Both, under Safe Expand and Safe Compress, a higher problem size demands more cores to achieve the STV execution time (third column of Figures 6 and 7). The quality trends closely track problem size trends on the pareto fronts (fourth column of Figures 6&7).

Speculative Modes: To characterize Accordion operation under the onset of timing errors, we repeat the experiments by permitting f_{NTV} to exceed $f_{NTV,Safe}$. Compress and Expand iso-execution-time pareto fronts under Speculative mode are labeled as Spec. Compress and Spec. Expand in Figures 6&7. This time, we are no more restricted by the f of the slowest core in the set of cores assigned to computation. We experiment with Drop 1/4, where a quarter of the threads get dropped under errors. For benchmarks where the Q degradation under Drop 1/4 remained negligible, we report the more conservative execution under Drop 1/2.

In this case, the error rate is dictated by the execution time per infected thread, since practically we observe an error at the end of the execution of each infected thread. If an infected thread takes e cycles to execute, the error rate per cycle, Perr would be $1/e$. We consult Figure 5b for each such $1/e$ to estimate the operating f. This time, for a given set of cores, f_{NTV} is determined by the speculative operating f of the slowest core. We observe 8-41% f increase across chip due to operation at a higher error rate. Generally, the same trends for Safe apply. When compared to Safe, the higher f_{NTV} increases the power slightly, but degrades Q. At the same time, due to the higher f_{NTV}, a lower N suffices to achieve the STV execution time, rendering a higher MIPS/W.

Putting It All Together: Not all of the points on the fronts reflect feasible operation. Depending on the mode, the execution may become limited by Q, N, or power. Examining the fourth columns, except bodytrack, Q degradation does not exceed 73.51% (x264) even under Speculative Compress. Still, tight preset limits on Q degradation may render some of these operating points infeasible. The most Q-limited mode is Speculative Compress, where on top of the already decreased base problem size (hence Q), we introduce errors. Both area and power constraints may limit N. Expand is mainly N-limited, since increasing N is the only way to achieve iso-execution-time under degraded f_{NTV}. When compared to Safe, for Speculative Expand the pressure on N is partially released by the higher speculative f_{NTV}. An N-limited mode directly translates into power-limited, since power consumption is more sensitive to N than to f. Accordingly, Expand is the most power-limited mode. Already for ferret, x264, hotspot and srad we observe that power budget would have to be exceeded to achieve iso-execution-time operation under Expand, for the largest problem sizes considered (Figures 6&7, 2nd column).

7. Discussion

Throughout the paper, *weak scaling* refers to the scaling of the problem size with the core count, which does not always translate into fixed per thread work, as weak scaling in strict sense would imply. Applications strictly conforming to weak scaling would benefit most from Accordion operation. However, for the select RMS benchmarks we deployed, per thread work tends to increase with problem size. We are extending our study to strict weak scaling, considering novel application domains such as bitcoin mining [33]. We also did not explore how to orchestrate Accordion operation dynamically at runtime. While the number of cores assigned to computation can be changed midst-execution, the problem size may not be. For this study, we let resource allocation and the assigned operating point apply for the entire duration of execution. However, both, phases of the application, and the hardware resources may experience changes in resiliency within the course of execution. How such fine grain temporal changes in resiliency can be captured represents another open question.

8. Related Work

Near-Threshold Voltage Computing: To the best of our knowledge, no previous NTC proposal suggested weak scaling

to mask limited application parallelism, or exploited inherent fault tolerance of RMS applications to mask the impact of variation: Booster [25] and EnergySmart [21] constitute two designs tailored for variation mitigation at NTV. In a Booster chip each core can switch between two independent `Vdd` rails, to run at two distinct `f`. An on-chip governor determines how long each core spends on each rail as a function of the core's variation profile. The goal is to attain the same effective `f` across all the cores such that applications do not perceive variation-induced speed differences between the cores. EnergySmart, on the other hand, tries to attain the most energy-efficient variation-aware task schedule considering a single-`Vdd`-rail NTV chip. EnergySmart supports multiple distinct frequencies across chip, however, as opposed to Booster, at the granularity of clusters of cores to enhance scalability.

Soft Computing: Related work in soft computing exists focusing on energy vs. reliability trade-off at low voltages, however, usually does not cover as low voltages as near-threshold: Algorithmic fault tolerance was exploited for improved performance [24], energy efficiency [17, 4, 27] or lower hardware design complexity [8]. At the core of these efforts lies quantification of the degradation in application output quality. Fault injection is rigorously deployed to this end, where the output quality of RMS applications is shown to be insensitive to errors in the data flow, as opposed to control [35, 23, 34]. Similar to Accordion, ERSA [10] leverages this characteristic by dividing multicore compute power into unreliable cores in charge of data, and reliable cores in charge of control, however, not in the context of NTC. In [11], a software recovery approach is explored based on the observation that RMS applications can tolerate even discard of computations. Recently, approximate data structures [30, 29] along with the required hardware support [15] have also been explored.

9. Conclusion

Accordion overcomes NTC barriers by leveraging weak scaling and implicit fault tolerance of emerging R(ecognition), M(ining), S(ynthesis) applications. The key observation is that the problem size not only dictates the number of cores engaged in computation, but also the application output quality. Consequently, Accordion designates the problem size as the main knob to trade off the degree of parallelism (i.e. the number of cores engaged in computation), with the degree of vulnerability to variation (i.e. the corruption in application output quality due to variation-induced errors). Parametric variation renders ample differences between the reliability of cores. Since RMS applications can tolerate faults emanating from data-intensive program phases as opposed to control, variation-afflicted Accordion hardware executes fault-tolerant data-intensive phases on error-prone cores, and reserves reliable cores for control. Across a representative subset of RMS applications, Accordion can achieve the STV execution time while operating 1.61-1.87$\times$ more energy efficiently.

References

[1] International Technology Roadmap for Semiconductors, 2011 Edition. [Online]. Available: http://www.itrs.net.

[2] M. Bhadauria, V. M. Weaver, and S. A. McKee, "Understanding PARSEC Performance on Contemporary CMPs," *IISWC*, 2009.

[3] C. Bienia *et al.*, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," Princeton University, Tech. Rep. TR-811-08, January 2008.

[4] M. A. Breuer, "Multi-media Applications and Imprecise Computation," *Euromicro Symposium on Digital Systems Design*, August 2005.

[5] L. Chang *et al.*, "Practical Strategies for Power-Efficient Computing Technologies," *Proceedings of the IEEE*, vol. 98, no. 2, February 2010.

[6] S. Che *et al.*, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *IISWC*, 2009.

[7] Y.-K. Chen *et al.*, "Convergence of Recognition, Mining, and Synthesis Workloads and Its Implications," *Proceedings of the IEEE*, vol. 96, no. 5, May 2008.

[8] V. Chippa *et al.*, "Scalable Effort Hardware Design: Exploiting Algorithmic Resilience for Energy Efficiency," *DAC*, 2010.

[9] ——, "Dynamic Effort Scaling: Managing the Quality-Efficiency Tradeoff," in *DAC*, 2011.

[10] H. Cho *et al.*, "ERSA: Error Resilient System Architecture for Probabilistic Applications," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 4, 2012.

[11] M. de Kruijf *et al.*, "Relax: An Architectural Framework for Software Recovery of Hardware Faults," in *ISCA*, 2010.

[12] R. Dennard *et al.*, "Design of Ion-Implanted MOSFETs with Very Small Physical Dimensions," in *JSSC*, October 1974.

[13] R. G. Dreslinski *et al.*, "Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits," *Proceedings of the IEEE*, vol. 98, no. 2, February 2010.

[14] ——, "Reevaluating Fast Dual-Voltage Power Rail Switching Circuitry," in *WDDD*, June 2012.

[15] H. Esmaeilzadeh *et al.*, "Architecture Support for Disciplined Approximate Programming," in *ASPLOS*, 2012.

[16] J. L. Gustafson, "Reevaluating Amdahl's Law," *Communications of ACM*, vol. 31, no. 5, 1988.

[17] R. Hegde and N. R. Shanbhag, "Energy-efficient Signal Processing via Algorithmic Noise-tolerance," in *ISLPED*, 1999.

[18] S. Jain *et al.*, "A 280mV-to-1.2V Wide-Operating-Range IA-32 Processor in 32nm CMOS," in *ISSCC*, 2012.

[19] E. K. Ardestani and J. Renau, "ESESC: A Fast Multicore Simulator Using Time-Based Sampling," in *HPCA*, 2013.

[20] U. R. Karpuzcu *et al.*, "VARIUS-NTV: A Microarchitectural Model to Capture the Increased Sensitivity of Manycores to Process Variations at Near-Threshold Voltages," in *DSN*, 2012.

[21] ——, "EnergySmart: Toward Energy-Efficient Manycores for Near-Threshold Computing," in *HPCA*, 2013.

[22] S. Li *et al.*, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *MICRO*, 2009.

[23] X. Li and D. Yeung, "Application-Level Correctness and its Impact on Fault Tolerance," in *HPCA*, 2007.

[24] K. J. Lin *et al.*, "Imprecise Results: Utilizing Partial Computations In Real-Time Systems," in *Real-Time Systems Symposium*, 1987.

[25] T. Miller *et al.*, "Booster: Reactive Core Acceleration for Mitigating the Effects of Process Variation and Application Imbalance in Low-voltage Chips," in *HPCA*, 2012.

[26] S. Misailovic *et al.*, "Quality of Service Profiling," in *International Conference on Software Engineering*, 2010.

[27] S. Narayanan *et al.*, "Scalable Stochastic Processors," in *DATE*, 2010.

[28] N. Pinckney *et al.*, "Assessing the Performance Limits of Parallelized Near-Threshold Computing," in *DAC*, 2012.

[29] M. Rinnard, "Parallel Synchronization-Free Approximate Data Structure Construction," *Workshop on Hot Topics in Parallelism*, 2013.

[30] A. Sampson *et al.*, "EnerJ: Approximate Data Types for Safe and General Low-power Computation," in *PLDI*, 2011.

[31] L. Snyder, "Type Architectures, Shared Memory, and the Corollary of Modest Potential," *Annual Review of Computer Science*, vol. 1, 1986.

[32] M. Taylor, "Is Dark Silicon Useful? Harnessing the Four Horsemen of the Coming Dark Silicon Apocalypse," in *DAC*, June 2012.

[33] M. B. Taylor, "Bitcoin and the Age of Bespoke Silicon," in *CASES*, 2013.

[34] D. Thaker *et al.*, "Characterization of Error-Tolerant Applications when Protecting Control Data," in *IISWC*, 2006.

[35] V. Wong and M. Horowitz, "Soft Error Resilience of Probabilistic Inference Applications," in *SELSE*, 2006.