Exploring the Interplay of Yield, Area, and Performance in Processor Caches

Hyunjin Lee, Sangyeun Cho, and Bruce R. Childers
Department of Computer Science
University of Pittsburgh
{abraham,cho,childers}@cs.pitt.edu

Abstract

The deployment of future deep submicron technology calls for a careful review of existing cache organizations and design practices in terms of yield and performance. This paper presents a cache design flow that enables processor architects to consider yield, area, and performance (YAP) together in a unified framework. Since there is a complex, changing trade-off between these metrics depending on the technology, the cache organization, and the yield enhancement scheme employed, such a design flow becomes invaluable to processor architects when they assess a design and explore the design space quickly at an early stage. We develop a complete set of tools supporting the proposed design flow, from injecting defects into a wafer to evaluating program performance of individual processors in the wafer. A case study is presented to demonstrate the effectiveness of the proposed design flow and developed tools.

1 Introduction

At feature sizes below 65nm, more frequent hard faults due to random defects and process variations pose a serious burden to processor design and yield management [16]. As device size scales downward, a smaller defect size makes it easier to introduce faults that cause critical failures. Process variations also become problematic due to limitations in lithography and process control. The primary effect is on device length and threshold voltage, which can adversely impact timing and leakage. Amplified process variations require that operational margins widen, making it difficult to get functioning chips with worst case design.

This work was supported in part by NSF grants: CNS-0702236, CNS-0551492, CNS-0509115, CNS-0305198.

This problem is even more pronounced in the memory hierarchy of a processor chip. As the march continues toward large on-chip memories, more of the total transistor budget is devoted to memory. For example, in Intel's Montecito processor [13], the memory arrays for L2 and L3 caches account for well over 60% of the total chip area. There is a greater likelihood of defects and variations in memory as these structures grow in size, because memory transistors are some of the smallest and most timing sensitive. Traditional techniques for masking memory faults use redundancy, where functional spare elements (*e.g.*, columns and/or rows) take the place of defective ones. However, this approach will have to devote a large, valuable chip area to the spares in future nanometer-scale technology [1], leading to an adverse interplay with yield.

In response to the problems with redundancy in nanometer technology, new yield-enhancing techniques based on *graceful degradation* are gaining attention [1, 10, 18]. In graceful degradation, all memory capacity is exposed to the microarchitecture (*i.e.*, no spares). Graceful degradation can disable failed components and possibly reconfigure operational ones to serve as substitutes. For caches, faulty components, such as cache blocks, sets, and ways, can be disabled without changing the processor functionality, possibly with a performance penalty. Programs that are cache intensive may suffer a larger penalty than programs having few memory accesses. This approach has been applied to recent processors [6, 13] and will become more commonplace, even in embedded and SoC designs.

An advantage to traditional redundancy approaches is that cache microarchitecture and yield can be treated independently, especially at an early stage during architecture specification. Typically, a processor architect does not have to worry about yield at such an early point. However, with graceful degradation, there is a strong interdependence between cache microarchitecture and yield. In this case, the cache microarchitec-

ture and the yield management scheme must be considered *simultaneously* to make appropriate design decisions. For example, one graceful degradation mechanism might have good defect coverage but a high performance penalty on a benchmark workload. Another scheme might have a negligible performance impact but cover only certain faults in a few cache resources. A processor architect needs to evaluate the choices to select the most appropriate one, given the expected workload and defect and process variation characteristics of the target technology.

In this paper, we propose a novel methodology, models and metric that can effectively evaluate the tradeoff between yield and performance of the caches with different fault masking mechanisms. The methodology relates the occurrence of defects at the circuit level to faults at the cache organization level. The occurrence of a process variation or defect in the circuit can cause many fault manifestations, including the failure of cache cells, blocks, ways, and banks. Based on the failures, microarchitecture schemes that disable faulty components may be modeled and workloads simulated to evaluate the performance impact. Our methodology introduces a new metric, called *yield-area-performance* (YAP), for comparing alternative yield management schemes. YAP can be used to evaluate different graceful degradation approaches, as well as traditional redundancy schemes. It can be used to quantify yield at a particular performance goal. Importantly, our methodology lets a processor architect evaluate the trade-offs quickly and early in the design process.

This paper makes several contributions. First, it presents a new high-level yield management evaluation methodology. Second, it describes a model that can relate circuit-level defects and process variation in caches to faults at the organizational level. Third, it gives a novel metric (YAP) for evaluating different graceful degradation and redundancy approaches. Finally, the paper presents a case study using our methodology and the YAP metric. We find that a graceful degradation approach based on disabling faulty cache blocks can achieve a higher yield than redundancy at a slight performance cost.

2 Related Work

There is much previous work on defect modeling and yield analysis [3, 12]. Physical defects have been a major factor undermining yield. Maly and Deszczka [11] developed a random defect model to compute defect distributions. Their model was later extended to accommodate a submicron domain [14]. Dekker *et al.* [7] developed an efficient SRAM fault model and a set of linear march-based test algorithms. Wang *et al.* [21] categorized various defect models and established a defect injection procedure. They considered only cell-level faults and did not analyze defects affecting wires such as a bitline or a wordline.

Process variations have become a major reliability threat in modern deep sub-micron (DSM) technologies, which are further classified into inter- and intra-die components [5, 16]. With aggressive technology scaling, the random and correlated components of intra-die variations exceed those of inter-die variations [8]. Agarwal *et al.* [1] analyzed the impact of process variations on cache yield and developed a direct-mapped cache structure to mask off faulty memory cells. Their work, however, considered only process variations (and not physical defects).

There are previous works studying the impact of redundancy techniques on yield. Thomas and Anthony [20] analyzed a set of redundancy schemes in terms of yield and performance. They, however, assumed only a simple defect model with a defect density and a probability of a faulty row. Shivakumar *et al.* [17] developed a metric called *performance averaged yield*. Unfortunately, their model assumes performance loss due to defects in logic blocks, not from memory arrays.

There are a few studies that examined the performance of a fault-degradable cache approach. Sohi [18] looked at the performance impact of line deletion in a unified cache. Pour and Hill [15] extended Sohi's work by introducing a more systematic way to evaluate the impact of defects. Unfortunately, these earlier studies simulate a rather simple unified cache configuration using ATUM traces and use the miss rate as the only metric. More recently, Lee *et al.* [10] analyzed the performance impact of defects more rigorously using a generic fault degradable cache model methodology. Our work in this paper focuses on studying the interplay between cache yield, cache area, and program performance in an integrated framework and builds on these previous performance models.

3 Design Flow

The goal of the proposed design flow is to provide a processor architect with an integrated framework to evaluate a cache design in terms of yield, area, and performance.[1] We note that the proposed design flow can be easily extended to include other traditional metrics such as power consumption and leakage, by integrating existing models into the framework, for instance [19].

[1] Yield usually refers to the proportion of dies on a wafer that perform properly up to a design specification. In this paper, we extend the use of "yield" to a cache within a processor chip.

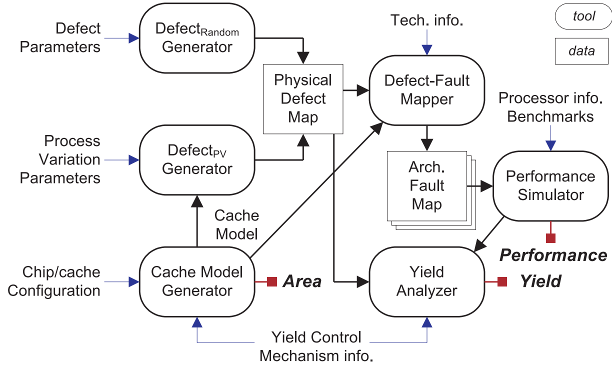

Figure 1. Design flow overview.

There are three obstacles in current methodologies that have to be addressed. First, during early design exploration, a processor architect needs to understand how defects and process variations affect the cache to select a good yield management scheme. There are complex trade-offs between yield, area, and performance, especially when yield control mechanisms with different area and defect coverage are considered [10,20]. Second, the defects and process variations manifest themselves at the physical level and need to be related to the cache microarchitecture under consideration. However, the physical design is unknown during early design exploration. Finally, because a particular yield management scheme can affect several layers from the application level to the physical level, efforts from independent design groups have to be integrated.

Our design flow addresses these problems in the following way. From a "cache microarchitecture specification," the design flow automatically derives a "physical cache model." The physical cache model approximates the anticipated implementation of the given cache specification. With the derived physical model, defects and process variations are automatically placed in the physical model according to a "defect model." The defect model is based on the characteristics of the target technology. The defects and process variations at the circuit level are then mapped to the organizational level to determine how they affect the cache (*e.g.*, which cache blocks failed). Because some yield management schemes do not guarantee program performance, our methodology simulates – at the architecture behavioral level – a cache with a set of given faults. To ensure statistically valid coverage of many possible fault manifestations, the methodology uses extensive sampling at the wafer level to derive and evaluate different defect and process variation scenarios.

To use the design flow, several inputs have to be specified, as shown in Figure 1, including (1) technology information (wafer size, cell size, and process variation parameters); (2) defect distribution parameters; (3) chip and cache configuration (cache microarchitectural and organizational parameters and the geometry of components); (4) yield control mechanism information; and (5) processor architecture parameters and benchmark programs (used in performance evaluation). The processor architect would typically specify the cache organization, program workload and processor architecture. Information about the target technology can be provided by circuit and process engineers. The output of the design flow includes yield, area, and program performance. Obtaining the output from the input is fully automated.

3.1 Cache specification

To accurately translate physical defects into architectural faults, a detailed cache specification is necessary. The most important cache design parameters are architectural parameters, organizational (*i.e.*, banking) parameters, and physical layout information such as array geometry and SRAM cell layout. Our design flow provides an interface to describe a cache's geometry inside a chip.

Architectural parameters are a 3-tuple: A (associativity), B (block size), and C (cache size). We consider three organizational parameters: N_{dwl}, N_{dbl}, and N_{spd}, which determine the internal sub-banking by specifying how wordlines and bitlines are partitioned [19]. They also define cache block to sub-bank mapping. Cache geometry is modeled as a hierarchy of non-overlapping rectangles, from wafer and chip down to cache and cache components. Cache components include memory arrays and control logic. Table 1 summarizes our cache parameters. To simplify the presentation, we consider only the data array (*i.e.*, not tag array) in the following discussion without losing generality.

3.2 Defect model

For a target technology, we consider two major sources of defects: *random defects* and *process variations*. *Random defects* including physical defects and abnormally thin lines that cause incorrect logical results. *Process variations*, which may have many parameters, can be abstracted by varying V_{th}. The random defects and process variations are specified according to a user-defined distribution, such as a Gaussian distribution.

Based on the target technology, defects are placed into the physical cache model with a three step process. First, random defects, characterized by their location (on a wafer) and size, are generated. The number of

Table 1. Cache specification parameters.

Parameter	Description
Architectural parameters	
A	Set associativity
B	Block size (in bytes)
C	Cache size (in bytes)
Organizational parameters	
N_{dwl}	# of wordline segments
N_{dbl}	# of bitline segments
N_{spd}	# of sets mapped to a single wordline
Layout- and defect-related parameters	
cx, cy	SRAM cell dimension (in *nm*)
px, py	Coord. of a given physical defect ϕ (in *nm*)
Derived parameters	
W_b	Block width; $8 \cdot cx \cdot B$ (in *nm*)
W_{sb}	Sub-bank width; $A \cdot W_b \cdot N_{spd}/N_{dwl}$ (in *nm*)
W_{ssb}	Set width in a sub-bank ($N_{spd} \geq 1$); W_{sb}/N_{spd}
px_{sb}	Rel. X-coord. of ϕ in a sub-bank; $px \bmod W_{sb}$
N_{set}	# of sets; $C/(A \cdot B)$
N_{sbl}	# of sets per bitline; N_{set}/N_{dbl}
N_{pr}	Index of memory row having ϕ; $\lfloor py/cy \rfloor$
N_{xsb}	X-index of sub-bank having ϕ; $\lfloor px/W_{sb} \rfloor$

generated defects depends on the defect density, an input to the defect generator. In the second step, uninteresting defects (out of the target cache area) are discarded. The coordinates of the remaining defects are re-computed relative to the origin of a specific cache component. Lastly, each defect is decomposed into a set of sub-defects if applicable, each of which places a physical fault on a wordline, a bitline, or a memory cell. After this final step, there remain a set of physical defects $\{\phi_i = (px_i, py_i)\}$ that can lead to a cache fault. We consider the effect of process variation to SRAM cell operations, similar to [1]. How we generate faults due to process variation is detailed in Section 3.3.3.

3.3 Mapping defects to faults

A defect or process variation in the physical model is mapped to the cache microarchitecture based on the affected component. In this mapping, our methodology considers three component types: bitline, wordline, and memory cell.

3.3.1 Bitline

A defect in a bitline (*e.g.*, an open bitline) causes faulty cache blocks. Depending on the cache organization, all the blocks in a way can be lost as in Figure 2(a) or only a portion of them as shown in Figure 2(b) and (c). We consider two cases depending on the value

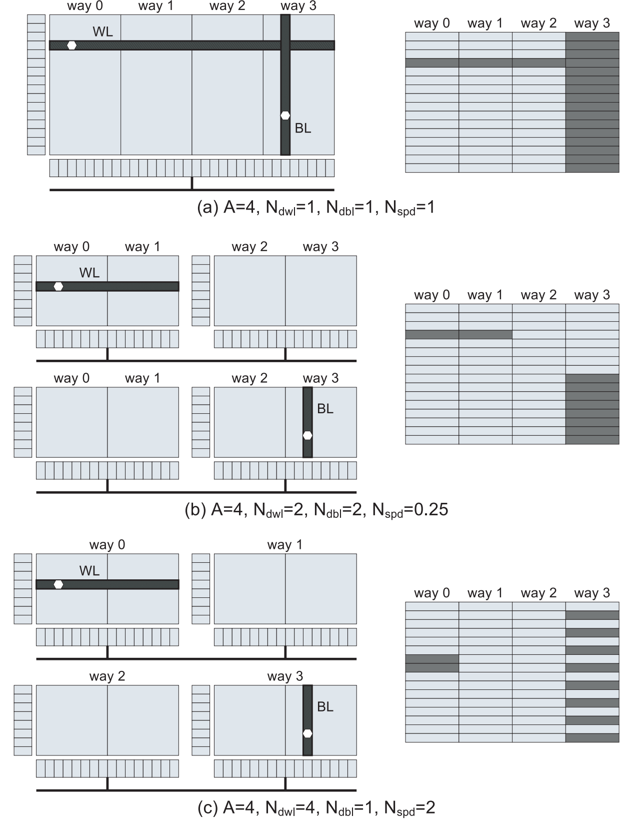

Figure 2. Defective wordline or bitline (left) results in various architectural faults (right).

of N_{spd}. When $N_{spd} \geq 1$, total $(N_{set}/(N_{spd} \cdot N_{dbl}))$ blocks are affected. Within a selected cache way (*way #* below), one cache block in every N_{spd} consecutive cache blocks is faulty. In other words, given a defect $\phi = (px, py)$, all the sets with their number equal to $((\textit{initial set}\ \# + n \cdot N_{spd} \bmod (N_{set}/N_{dbl}))$, $n \geq 0$, are faulty. *way #* and *initial set #* are computed as:

$$\begin{aligned} \textit{way}\ \# &= N_{xsb} \cdot W_{ssb}/W_{sb} + \frac{px_{sb} \bmod W_{ssb}}{W_b} \\ \textit{initial set}\ \# &= N_{pr} \cdot N_{spd} + px_{sb}/W_{ssb} \end{aligned}$$

When $N_{spd} < 1$, total (N_{set}/N_{dbl}) consecutive cache blocks become faulty. They are identified with:

$$\begin{aligned} \textit{way}\ \# &= \lfloor px/(W_b \cdot N_{spd}) \rfloor \\ \textit{set}\ \#, \textit{start} &= \lfloor \frac{N_{pr} \cdot N_{spd}}{N_{sbl}} \rfloor \cdot N_{sbl} \end{aligned}$$

3.3.2 Wordline

Like a defective bitline, a defective wordline can cause unavailable cache blocks in a number of different ways. All blocks in a set can be lost as in Figure 2(a), a

few cache blocks in a set may become unavailable as in Figure 2(b), or cache blocks in multiple cache sets can become faulty as in Figure 2(c). When $N_{spd} \geq 1$, faulty cache blocks are clustered in a rectangle (N_{spd} by A/N_{dwl}) whose origin can be computed as follows:

$$\begin{aligned} \textit{set \#, start} &= N_{pr} \cdot N_{spd} \\ \textit{block \#, start} &= N_{xsb} \cdot A/N_{dwl} \end{aligned}$$

When $N_{spd} < 1$, (A/N_{dwl}) consecutive blocks become faulty within a single set. The initial block is identified with the following:

$$\begin{aligned} \textit{set \#} &= \lfloor N_{pr} \cdot N_{spd} \rfloor \\ \textit{block \#, start} &= \lfloor \frac{\lfloor px/(W_b \cdot N_{spd}) \rfloor}{A/N_{dwl}} \rfloor \cdot \frac{A}{N_{dwl}} \end{aligned}$$

3.3.3 Memory cell

A faulty cache block due to a defective cell is determined by the following equations. Again, we consider two cases based on the value of N_{spd}. When $N_{spd} \geq 1$:

$$\begin{aligned} \textit{set \#} &= N_{pr} \cdot N_{spd} + px_{sb}/W_{ssb} \\ \textit{block \#} &= N_{xsb} \cdot W_{ssb}/W_b + (px_{sb} \; mod \, W_{ssb})/W_b \end{aligned}$$

When $N_{spd} < 1$:

$$\begin{aligned} \textit{set \#} &= \lfloor N_{pr} \cdot N_{spd} \rfloor \\ \textit{block \#} &= \lfloor px/(W_b \cdot N_{spd}) \rfloor \end{aligned}$$

Besides the effect of random defects, we consider the impact of process variations on memory cell reliability. To do so, we consider both inter-die variation ($\Delta V_{th-inter}$, common to devices in a die) and intra-die variation ($\Delta V_{th-intra}$) using a Gaussian distribution whose standard deviation is given by the BPTM model [4]. We then assign ($\Delta V_{th-inter} + \Delta V_{th-intra}$) to each memory cell. Finally, we determine that a memory cell in consideration has a fault if its ($V_{th} + \Delta V_{th}$) is not within a user-defined range. In essence, whether a memory cell has a fault due to process variations or not is decided probabilistically, similar to [1].

3.4 Yield, area, and performance

The yield of a cache memory depends on its area and organization as well as defect distribution and process variation effects. A cache memory with a defect which hinders its normal operation is considered a failed component, unless a yield enhancement technique (*e.g.*, disabling) masks the effect of the defect. The area of a cache depends on its baseline design (specified by design parameters) and additional support function such as redundancy. The performance of a cache memory (*e.g.*, hit rate) is primarily determined by its architectural parameters. Note that these metrics interplay. For example, a larger cache design may lead to lower yield. If a disabling scheme is used, the resulting yield may be high, but the average performance obtainable from the cache may be low. Therefore, these metrics should be made available together to a designer early so that right design decisions can be made then.

Our design flow always reports these metrics together in a 3-tuple (Y (yield), A (area), P (performance)). To conveniently compare multiple design points with a single number, one can combine the three terms in various ways. Because better cache would have larger Y and P, and smaller A, we can evaluate cache design with a single number $aY^l \cdot bP^m \cdot cA^{-n}$; larger one shows better design. The choice of a, b, c, l, m, and n will depend on the emphasis of analysis. In this work, we set $(a, b, c) = (1, 1, 1)$.

There is a trade off between chip frequency and yield, where chip yield can be improved by lowering the frequency. We assume the frequency is constant in our current models. The interplay of chip frequency and process variation on cache yield is left as future work.

3.5 Validation

The defect model was validated with statistical rigor by directly comparing input parameters and the resulting defect distributions. The defect-to-fault mapping model validation entailed much effort and was tackled in a hierarchical manner. First, we validated our SRAM cell layout model with varying defect locations and sizes within and on the periphery of it. Then we validated at the component level with a set of synthesized defect patterns to exercise all possible fault types. We used inspection-based methods assisted with a visualization tool. Finally, a large number of defect patterns and cache models were used to produce results, which were then statistically correlated with the input parameters. For cache configuration synthesis and architecture-level performance modeling, we use previously validated tools [2, 10, 19].

4 Case Study

To investigate the utility of our methodology, we did a case study to select a yield-effective cache design with YAP among a set of candidate designs. In this section, we first explain how to specify the design space. Next, we describe an evaluation of candidate designs.

Table 2. Case study specification

Parameter	Description
Architecture parameters	
Processor	ARM
Caches	Split instruction & data
Cache arch.	16-way, 16kB, 32B block
Cache org.	$N_{dwl} = 1$, $N_{dbl} = 4$, $N_{spd} = 0.125$
Technology parameters	
Size	$70nm$, $6.89mm^2$ chip size (SoC)
SRAM cell	$0.75\mu m$ width, $1\mu m$ height
Wafer	$300mm$
Yield management schemes	
No redundancy	No spare cache rows (baseline)
Redundancy	12.5% & 25% spare rows
Disabling	Graceful degradation for failed rows
ECC	SECDED
Defect and process variation parameters	
Location	Uniform distribution
Defect radius	Gaussian dist. ($\mu = \sigma = 70nm$)
Variations ($\sigma_{V_{th}}$)	$10mV$, $20mV$, $30mV$, $40mV$
Defect densities	1, 10, 100
Die sampling	Uniform distribution
Sample size	100 die from 10 wafers (1000 die)
Experimental workload	
Simulator	SimpleScalar sim-outorder [2]
Programs	*basicmath*, *cjpeg*, *rsynth*, *dijkstra*, *rijndaelenc*, *fft*

4.1 Specification

The case study is a system-on-a-chip (SoC) with an ARM processor that has instruction and data caches. To simplify the study, we assume that defects occur only in the caches.

The case study considers four yield management schemes for the cache: no redundancy (baseline), 12.5% and 25% row redundancy, and block disabling. Row redundancy uses spare memory rows to cover defective ones. The number of spares is the percentage 12.5% or 25% of total cache blocks. Block disabling uses graceful degradation to permanently disable defective blocks. Program addresses that map to defective blocks are not cached [10]. Because error correction codes (ECC) can mitigate the effect of process variations [1], we consider the designs with and without ECC.

The defect model uses a uniform distribution to select defect locations and a Gaussian distribution for defect size. Process variation has a single parameter: $\sigma_{V_{th}}$. The model is configured with a tuple (*defect density*, $\sigma_{V_{th}}$). The study uses the configurations (1, $30mV$), (10, $30mV$), (100, $30mV$), (100, $20mV$), and (100, $40mV$).[2] The study uses 1000 total die samples.

To measure program performance, we use the SimpleScalar tool set [2] to simulate an ARM-like in-order pipelined processor. The workload is six programs from MiBench [9] with a range of cache miss rates.

4.2 Evaluation

Using the design flow, we evaluated the yield management schemes. The results are in Tables 3 and 4 (in page 8). In the tables, yield is the ratio of operational caches to total caches. Area is the cache area relative to the baseline. Performance is an average of the benchmarks relative to the baseline. In this case study, we use four different YAP metrics, labeled M1–M4, denoting $Y \cdot A^{-1} \cdot P$, $Y \cdot A^{-1} \cdot P^2$, $Y^2 \cdot A^{-1} \cdot P$, and $Y \cdot A^{-2} \cdot P$, respectively. There are ten designs with two cases for block disabling. 95% block disabling discards all caches that have more than a 5% performance degradation. Similarly, 99% block disabling discards caches above a 1% degradation. Performance for all cases is 100% due to rounding, except for 95% disabling with defect density of 100.

The tables show that as defect density and process variation increase, yield suffers. No redundancy has the worst yield and shows the need for aggressive yield management. 12.5% and 25% row redundancy improve yield with a 8.1% and 16.3% area overhead.

Most interestingly, block disabling has a very good yield for a small 2% (no ECC) and 7% (with ECC) area cost. For example, in the (10, $30mV$) configuration from Table 3, 95% block disabling has a 98% yield, while 25% redundancy has a 82% yield. For one defect per cache from Table 3 and process variations $10mV$ and $20mV$ from Table 4, block disabling achieves yields close to 100% (the numbers in the table are rounded up from yields as high as 99.9% for 95% disabling with ECC). Yield is higher for disabling because it selects caches that meet the degradation threshold without attempting to cover the faults. This effect is most pronounced at high defect densities and process variations (*i.e.*, (100, $40mV$)).

Another interesting result is how much ECC helps yield under varying process variation as shown in Table 4. For example, no redundancy with ECC has an 85% yield in the (100, $40mV$) case versus 30% for no redundancy without ECC. ECC is also beneficial to disabling. In Table 4, 95% disabling with ECC has a yield that is near 100% for all process variations.

The YAP metrics capture the trade-off between better yield and increased area/decreased performance. The tables show that no redundancy has a reduced YAP as defect density and process variation increase.

[2] We scaled the defect densities for intuitive presentation and comparison. The defect density 1 corresponds to $0.5/mm^2$.

ECC and redundancy generally improve YAP. In some cases, the yield gain is offset by the area cost. The (10, $30mV$) configuration in Table 3 has a 92 M1 for no redundancy with ECC and a 74 M1 for 12.5% redundancy without ECC. Another example is (100, $20mV$) in Table 4. 25% redundancy has a better yield than no redundancy, but its M4 is lower (76 vs. 82). Block disabling has the overall highest YAP because yield is improved for a modest performance and area cost. For example, in Table 3, 99% block disabling in the (100, $30mV$) case has a 90 M3. In this configuration, disabling does much better than redundancy, even with ECC. Overall, the disabling scheme achieves the best yield and YAP in all cases. These results show the benefit of graceful degradation: It is sensitive to workload and does not blindly discard caches that have many faults but good performance.

To judge the computational demands of our tools, we measured the speed of the flow by continuously running simulations for 15 hours. With 10 cache designs, 100 samples from 10 wafers, and 5 defect-process variation configurations, the design space has 50,000 caches. Using four 3.4GHz Intel Xeon-based Linux machines, the flow evaluated all caches with no redundancy and redundancy schemes. For caches with disabling, the flow could finish roughly 10% of all caches, resulting in a maximum error of as low as 0.2% at the 95% confidence interval. This rate is fast enough to evaluate a large design space overnight.

The case study demonstrates how our flow and tools can be used to evaluate yield, area and performance of many cache designs. It also highlights the importance that graceful degradation schemes will play in yield management for future cache designs and technologies.

5 Concluding Remarks

Traditional cache optimizations have focused on the performance, area, and power aspect of the resulting design. Guaranteeing fault tolerance and enhancing chip yield have been largely a separate effort made by low-level circuit quality engineers, layout designers, and process engineers. As chips built with a future DSM technology are more susceptible, and thus, subject to manufacturing defects, process variations, and aging phenomena, it becomes imperative to consider reliability and yield together at an early design time.

This paper presented a new cache defect, yield, and performance model, laying a solid foundation for an integrated cache design flow in the era of nanometer-scale technologies. We also introduced a metric called YAP (yield-area-performance). Using the proposed design flow and the YAP metric, a cache designer can directly evaluate a cache design or compare different designs in terms of yield, area, and performance at an early design stage. The case study in this paper demonstrates the capability of the proposed design flow as well as its efficiency as a practical design tool. We are currently extending our design flow with a detailed cache power and leakage model.

References

[1] A. Agarwal *et al.* "A Process-Tolerant Cache Architecture for Improved Yield in Nanoscale Technologies," *IEEE TVLSI*, 13(1):27–38, Jan. 2005.

[2] T. Austin *et al.* "SimpleScalar: An Infrastructure for Computer System Modeling," *IEEE Computer*, 35(2), Feb. 2002.

[3] C. N. Berglund. "A Unified Yield Model Incorporating Both Defect and Parametric Effects," *IEEE TSM*, 9(3):447–454, Aug. 1996.

[4] Berkeley Predictive Technology Model. `http://www-device.eecs.berkeley.edu/~ptm/`.

[5] S. Borkar *et al.* "Parameter Variations and Impact on Circuits and Microarchitecture," *DAC*, June 2003.

[6] D. C Bossen, J. M. Tendler, and K. Reick. "Power4 System Design for High Reliability," *IEEE Micro*, 22(1), Jan. 2002.

[7] R. Dekker *et al.* "Fault Modeling and Test Algorithm Development for Static Random Access Memories," *ITC*, pp. 343–352, Sep. 1988.

[8] S. G. Duvall. "Statistical Circuit Modeling and Optimization," *Int'l Workshop on Stat. Metrology*, 18(3), Aug. 2000.

[9] M. R. Guthaus *et al.* "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," *WWC*, Dec. 2001.

[10] H. Lee, S. Cho, and B. Childers. "On the Impact of Defects in Cache Memory," *ISVLSI*, pp. 409–415, May 2007.

[11] W. Maly and J. Deszczka. "Yield Estimation Model for VLSI Artwork Evaluation," *Electronics Letters*, 19(6):226–227, Mar. 1983.

[12] L. S. Milor. "Yield Modeling Based on In-Line Scanner Defect Sizing and a Circuit's Critical Area," *IEEE TSM*, 12(1), Feb. 1999.

[13] S. Naffziger *et al.* "The Implementation of a 2-core Multi-Threaded Itanium-Family Processor," *ISSCC*, Feb. 2005.

[14] W. A. Pleskacz and W. Maly. "Improved Yield Model for Submicron Domain," *DFT*, pp. 2–10, Oct. 1997.

[15] A. F. Pour and M. D. Hill. "Performance Implications of Tolerating Cache Faults," *IEEE TC*, 42(3), Mar. 1993.

[16] SEMATECH. *Critical Reliability Challenges for ITRS, Technology Transfer #03024377A-TR*, Mar. 2003.

[17] P. Shivakumar *et al.* "Exploiting Microarchitectural Redundancy For Defect Tolerance," *ICCD*, Oct. 2003.

[18] G. S. Sohi. "Cache Memory Organization to Enhance the Yield of High-Performance VLSI Processors," *IEEE TC*, 38(4), Apr. 1989.

[19] D. Tarjan, S. Thoziyoor, and N. P. Jouppi. "CACTI 4.0," *HP Lab. Tech. Rep. HPL-2006-86*, June 2006.

[20] T. Thomas and B. Anthony. "Area, Performance, and Yield Implications of Redundancy in On-Chip Caches," *ICCD*, Feb. 1999.

[21] X. Wang *et al.* "Estimating the Manufacturing Yield of Compiler-Based Embedded SRAMs," *IEEE TSM*, 18(3):412–421, Aug. 2005.

Table 3. Yield, Area, Performance, and four YAP metrics (M1–M4) in %, given defect density = 1, 10, 100 and $\sigma_{V_{th}}$ = 30mV. The YAP metrics M1–M4 are Y$\cdot$A$^{-1}\cdot$P, Y$\cdot$A$^{-1}\cdot$P^2, Y$^2\cdot$A$^{-1}\cdot$P, and Y$\cdot$A$^{-2}\cdot$P, respectively. Numbers in bold face represent the best YAP values

	$(1, 30mV)$							$(10, 30mV)$							$(100, 30mV)$						
Design	Y	A	P	M1	M2	M3	M4	Y	A	P	M1	M2	M3	M4	Y	A	P	M1	M2	M3	M4
No redundancy	81	100	100	81	81	66	81	80	100	100	80	80	64	80	68	100	100	68	68	46	68
No redundancy ECC	97	104	100	93	93	90	90	95	104	100	92	92	87	88	85	104	100	81	81	69	78
12.5% redundancy	81	108	100	75	75	61	69	80	108	100	74	74	60	69	77	108	100	72	72	55	66
25% redundancy	83	116	100	71	71	59	61	82	116	100	71	71	58	61	82	116	100	71	71	58	61
12.5% redun. ECC	97	113	100	86	86	84	77	96	113	100	85	85	81	76	91	113	100	81	81	74	72
25% redun. ECC	99	121	100	82	82	81	68	97	121	100	80	80	78	66	92	121	100	76	76	69	63
95% disabling	99	102	100	**97**	**97**	**96**	**95**	98	102	100	**96**	**96**	**94**	**94**	97	102	98	93	92	91	**92**
99% disabling	99	102	100	**97**	**97**	**96**	**95**	98	102	100	**96**	**96**	**94**	**94**	96	102	100	**94**	**94**	90	**92**
95% disabling ECC	99	106	100	94	94	93	88	99	106	100	93	93	92	88	100	106	100	**94**	**94**	**93**	88
99% disabling ECC	99	106	100	94	94	93	88	99	106	100	93	93	92	88	99	106	100	93	93	**93**	88

Table 4. Yield, Area, Performance, and four YAP metrics (M1–M4) in %, given defect density = 100 and $\sigma_{V_{th}}$ = 20mV, 30mV, 40mV. The YAP metrics M1–M4 are Y$\cdot$A$^{-1}\cdot$P, Y$\cdot$A$^{-1}\cdot$P^2, Y$^2\cdot$A$^{-1}\cdot$P, and Y$\cdot$A$^{-2}\cdot$P, respectively. Numbers in bold face represent the best YAP values

	$(100, 20mV)$							$(100, 30mV)$							$(100, 40mV)$						
Design	Y	A	P	M1	M2	M3	M4	Y	A	P	M1	M2	M3	M4	Y	A	P	M1	M2	M3	M4
No redundancy	82	100	100	82	82	66	82	68	100	100	68	68	46	68	30	100	100	30	30	9	30
No redundancy ECC	86	104	100	82	82	70	79	85	104	100	81	81	69	78	85	104	100	81	81	69	78
12.5% redundancy	89	108	100	82	82	73	76	77	108	100	72	72	55	66	53	108	100	49	49	26	45
25% redundancy	90	116	100	78	78	70	76	82	116	100	71	71	58	61	52	116	100	45	45	24	39
12.5% redun. ECC	90	113	100	80	80	73	71	91	113	100	81	81	74	72	91	113	100	81	81	73	72
25% redun. ECC	92	121	100	76	76	70	63	92	121	100	76	76	69	63	92	121	100	76	76	70	63
95% disabling	99	102	97	94	91	93	92	97	102	98	93	92	91	**92**	97	102	97	92	88	89	**90**
99% disabling	98	102	100	**96**	**96**	**94**	**94**	96	102	100	**94**	**94**	90	**92**	94	102	100	92	92	87	**90**
95% disabling ECC	100	106	100	94	94	**94**	89	100	106	100	**94**	**94**	**93**	88	99	106	100	**94**	**94**	**93**	88
99% disabling ECC	100	106	100	94	94	**94**	89	99	106	100	93	93	**93**	88	99	106	100	93	93	**93**	88