Summary from the First Cross-Layer Reliability Meeting: March 26 and 27, 2009 — Santa Clara, CA

The first workshop was organized around the six main questions from the Cross-Layer Reliability proposal. These questions are:

1. How do we organize, manage, and analyze layering for cooperative fault mitigation?
2. How do we best accommodate repair?
3. What is the right level of filtering at each level of the hierarchy?
4. Can we establish a useful theory and collection of design patterns for lightweight checking?
5. What would a theory and framework for expressing and reasoning about differential reliability look like?
6. Can a scalable theory and architectures that will allow adaptation to various upset rates and system reliability targets be developed?

We broke into two different sets of working groups for each question on the first day so that these questions could be discussed at length with the community leaders. On the second day, presentations regarding the progress made on these questions was to be presented.

Time was also allocated for discussion of issues and additional questions that did not fit into the original six. While this yielded interesting discussion of many issues at different levels that the study should consider (See Section 2) and many discussions about the boundaries of these questions, it did not yield completely new questions.

1 Overview of Working Group Discussions

What follows are digested highlights from the working groups for each question.

Question 1:

How do we organize, manage, and analyze layering for cooperative fault mitigation?

The working groups that met on this question felt there were several advantages for a multi-layer solution, including a chance to globally optimize computing systems and to enhance repair, adaptation and flexibility of computing systems. A number of disadvantages were also raised, including whether optimizing across layers implies cooperation across corporations, whether testing and validation is harder in a cross-layer computing systems, and whether existing layer contracts would need to be renegotiated.

Methods for managing layers were discussed during both working groups. The working groups felt that the layer interfaces would need to be formalized. In particular, determining how and what data is passed is necessary and may eventually lead to a standard protocol for passing data between layers. There was also discussion on how errors and exceptions should be handled, as well as whether part of the new contract between layers would include a FIT rate and the timeliness of fault analysis, detection and recovery.

Throughout the meeting there were discussions about the possibility of coercing errors into a few defined error types. This has the advantage of simplifying the interface to a higher-level layer. Some participants note that the design for manufacturing community was already trying to coerce variability problems into permanent faults.

Question 2:

How do we best accommodate repair?

The repair and recovery working groups were primarily concerned with faults that do not resolve over time and correcting corrupted computations. The working groups discussed what the appropriate level of granularity for repair is, as repairing at a higher level of computation is often simpler but more expensive than a lower level of computation. Furthermore, the working groups discussed design for repair methodologies that would allow for graceful degradation of the system over time.

Within the repair discussion the question of how to interface between layers also came up. The groups discussed an API that could communicate to other layers what level of repair is happening. The groups also felt that these interfaces would enable solutions across layers with lower level devices communicating what repair features were possible and their costs and the higher level devices determining which repair technique to use. These groups also felt that the API would need to accommodate application-specific needs and be OS-independent.

The recovery discussion within these groups focused on how recovery was dependent on both the fault rate and the application.

Question 3:

What is the right level of filtering at each level of the hierarchy?

The working groups on these topics focused on how to filter messages between layers.

The first group focused on how each layer could detect and recover from errors. To some degree this group felt that a layer could manage these processes without interaction from the other layers, as long as FIT or BER information and error logging was part of the layer contract. The first group also felt that a multi-layer approach should not degrade the ability to fix an error locally or quickly.

The second group focused at the software level. There are existing systems where software can directly handle faults at the user level, such as with database re-try, which could be more strongly leveraged.

Both groups noted the need for filtering to be adaptable to the user or application. The user or system may need to turn off filtering to reduce system overhead. Applications will have different needs in terms of resilience and configurability and should have some control over error detection and mitigation.

Question 4:

Can we establish a useful theory and collection of design patterns for lightweight checking?

These groups felt that finding and detecting the problem was the most important part of lightweight checking and perhaps the biggest challenge overall.

It is first useful to identify the different failure cases where detection is useful:

- transient/SEU
- permanent errors
- intermittent errors
- aging slowdown

Hard errors are potentially simplest to detect since they persist, meaning the system does not need to catch the error on a single cycle. This way they admit to less frequent (less continuous) checking.

The groups created a taxonomy of detection strategies and identified their strengths, weaknesses, and applicability to the failure cases:

- Duplication – potentially the gold standard. This works for all failure cases. It costs 100% overhead. Alternatives need to at least beat this. More generally, this is only the gold standard when the error rate is low enough that double errors are rare; the generalization for larger error rate cases is replication.
- Self-Check – expensive in general case; good in some cases (*e.g.* residue codes for arithmetic)
- Time-Redundant— good for transients and intermittent cases; if we can insert diversity of hardware used, perhaps this is usable for permanent errors. The slowdown for duplicated computation may be less than a factor of two in modern processors, but it still has near 100% energy overhead.
- Application Specific – good when can find them. How general? Is there a universal set?
  - algorithm-based fault-tolerance
  - signal processing, FFT
  - image processing
  - communication protocols
  - formal methods
  - result checking
  - graph algorithms
- Circuit-Level Prediction – probably not for transients
- High-Level Prediction
- Razor – works for aging slowdown case if change in timing sufficiently incremental
- Interleaved self-test – for hard errors and potentially for persistent wear cases

What other opportunities do we have to reduce the cost of checking?

- Complete and continuous checks vs. statistical checks
  - Statistical can be cheaper, but in what cases can we afford to miss errors?
  - Alternately, can we guarantee errors propagate to a point where can be detect?
  - How do we characterize the level of coverage/completeness?
- Importance of bits (differential reliability need) – some bits are more important than others – provide stronger protection for those
  - How do we assess the importance of a bit?

    * analysis?
    * dynamic measurement/experimentation – *e.g.* probability of error propagation?

Checking exists in a larger system context including, typically, checkpointing and rollback. Ultimately, we care about minimizing the total cost of the system, so we need to think about how tradeoffs that reduce checking costs may move costs into other areas.

- Latency of detection – checking may be cheaper when we allow more latency to detect
- Cost of checkpointing – higher if latency to detect is high
- Lost work in rollback – higher if latency to detect is high
- Diagnostic capability – lighter-weight checks may give less information on source of error
- Recovery – perhaps harder with less information about impact of error?
- Increase designer burden – if designer must identify something unique for each case; duplication doesn't require much additional thought; what of this can be automated?

What is the right framework/model for making these tradeoffs? These are used in an *ad hoc* way today. Can there be a more systematic approach to identifying opportunities and evaluating tradeoffs? A good model of goals, techniques, and tradeoffs may enable automated optimization.

Duplication/replication is an obvious option. So, if we are to show improvement, it should be much better than the duplication/replication scheme. Reducing checking costs by an order-of-magnitude from duplication would mean a checking overhead around 10%. For a scheme where the target reliability can only be achieved with n-replication (for $n > 2$), need to reduce to $n \times 10\%$ to achieve order-of-magnitude improvement.

Question 5:

What would a theory and framework for expressing and reasoning about differential reliability look like?

These groups felt that by using differential reliability the system designers could reduce area, delay and energy costs for a given reliability by not forcing the entire system to respond to problems from the weakest link.

What are our opportunities to tune the reliability of some computation?

- Physical
  - Energy (e.g. Vdd)
  - Clock Frequency
  - Margining
- Logical
  - Replication/Voting
  - ECC

Some examples of places where we might be able to tolerate lower reliability computations:

- Lightweight Checked Computation – the checked computation is guarded by a more reliable check
- Computation protected by ECC
- Convergent Numerical computation – correctness depends on exit condition; errors in update impact performance
- Control vs. Data plane in signal processing – data plane errors have less impact on result

The group discussed what gives rise to the opportunity for differential redundancy? In some cases it may come form importance of the data to the application. In others, it may be part of the way the internals of a computation are optimized. This raised questions about metrics (See Section 2.2).

How do we capture differential reliability requirements?

- Reflect in Programming Language – perhaps a noise-specification for data type (*e.g.*, PSNR Ratio). Maybe this should be coarse-grained for simplicity (*e.g.* Control variables might be typed reliable, while Data plane variables would be typed statistical with a noise-level specification).
- High-level reliability specification – *e.g.*, Verilog annotation? Option to synthesis?

The group noted that today this is exploited by hand using rules of thumb. With proper metrics, this could be a promising area for automation—both tools to analyze and tools to optimize.

Question 6:

Can a scalable theory and architectures that will allow adaptation to various upset rates and system reliability targets be developed?

These groups felt that adaptable solutions would be useful for systems that would be designed in one environment and used for many different environments. These groups felt that adaptation was needed to allow systems to naturally be able to handle different application demands, environments, fault/power/temperature conditions, bandwidth and aging. Parametrization and reconfiguration within the OS and hardware would be useful to allow systems options for adapting. Unlike many other groups, these teams felt that this area was struggling for innovation.

2 Overall Meeting Notes

2.1 Do no harm

Participants cautioned that software failures have traditionally accounted for more downtime than hardware failures. We should remember this as we add layers of software to address the hardware failure, making sure the complexity added in software and handling does not add more visible failures than it solves.

Some of the discussions about multi-layer approaches and software interaction became muddled as people conflated "software" with "application writer". It is important to more clearly identify that there are multiple layers in the "software" space (*e.g.*, firmware, OS, middleware, virtual machines) before the application. Even in the case where the application does assist with fault detection and recovery, the compiler has an opportunity to insert information and handling without burdening the application developer.

2.2 Metrics, characterization, and composability

We agreed we don't have an obvious set of metrics for capturing the characteristics of a component, stating the requirements for a system, and for composing them together. Ideally, we want some small vector of numbers to represent the reliability of a component and then a way of composing those together to compute the reliability of a system. FIT rates is the old way (and the starting-point strawman), but there is concern this doesn't cover the space adequately and is not properly composable. Some participants suggested the need for useful metrics that don't require FITs/bit as input.

This is clearly an important and central issue necessary to define the problems and metric success. Consequently, we are forming a focus group to look at this issue and make recommendations at the next meeting.

2.3 Benchmarks

Several members recognized the need for benchmarks that would help quantify the effect of resilience with performance and power.

2.4 Roadmap stake in the ground

As has often happened, the academics lament the lack of an absolute characterization of the problem. The industry generally does not provide such data. It is possible the industry is uncertain about many of these issues as well. Many agreed a roadmap of upset rates would be useful. Industry participants suggested academics do homework and make a proposal that industry could give a thumbs up/down on. Suggestions included looking/extrapolating Eugene Norman's data. Others note that there is public data that may be useful here and that several radiation conferences publish data that is not in arbitrary units. Having a roadmap (even if it's not completely accurate) would help motivates students.

As a result, we are formulating a roadmap focus group to review the literature and phenomena and try to suggest a roadmap.

2.5 Constituencies

The workshop and post-workshop discussions included lively debate of overhead targets. Commercial industry floated the idea of a goal of no more than 10% overhead per technology generation (doubling in transistor capacity). This goal aims to keep the useful capacity from Moore's Law continuing to grow at an exponential rate (*i.e.* $1.8\times$ per generation rather than the ideal $2\times$). This raised concerns that this may not be the right goal for system's that must operate in particularly harsh environments (*e.g.* space) or provide high reliability for life-critical applications (*e.g.*, airplanes, medical equipment). These applications already face extreme reliability challenges and may be willing to accept larger overheads. This discussion underscored the need for adaptable and scalable solutions (Question 6) and made it clear that we were not ready to speak with one voice about the nature of the challenge ahead.

Out of this discussion, we concluded that an important next step would be to assemble constituency groups and have them each identify their challenge goal. At the very least, this will help us understand the different needs and perspectives. Hopefully, this further allow us to identify the common needs or sub-problems among these groups. We are presently forming five constituencies groups: (1) commercial and consumer electronics, (2) space and avionics, (3) life critical systems, (4) infrastructure, and (5) large-scale systems.

Another concern raised was that goals stated only in terms of incremental expense could rule out interesting solutions that came at coarser granularity. For example, a solution that required 2× overhead but which allowed the equivalent of 7 generations of additional scaling (128× ideal) would provide the effective net benefit of achieving at least 1.8× scaling benefit per generation. Discussion accepted that these should be useful solutions to pursue. This can be seen as one way of decomposing the problem—first show the solution is in the desired scaling envelope, then figure out how to decompose the granularity of the design to get partial benefits along the way.

Anyone interested in learning more or contributing to the study should visit <`http://www.relxlayer.org`>. The web page is organized as a wiki allowing community contribution.

Acknowledgment: This study is supported by the Computing Research Association's (CRA) Computing Community Consortium (CCC) Visioning program.