REFUELING: PREVENTING WIRE DEGRADATION DUE TO ELECTROMIGRATION

ELECTROMIGRATION IS A MAJOR SOURCE OF WIRE AND VIA FAILURE. REFUELING UNDOES EM FOR BIDIRECTIONAL WIRES AND POWER/GROUND GRIDS—SOME OF A CHIP'S MOST VULNERABLE WIRES. REFUELING EXPLOITS EM'S SELF-HEALING EFFECT BY BALANCING THE AMOUNT OF CURRENT FLOWING IN BOTH DIRECTIONS OF A WIRE. IT CAN SIGNIFICANTLY EXTEND A WIRE'S LIFETIME WHILE REDUCING THE CHIP AREA DEVOTED TO WIRES.

Jaume Abella
Xavier Vera
Osman S. Unsal
Oguz Ergin
Intel Barcelona Research Center

Antonio González
Intel Barcelona Research Center and Universitat Politècnica de Catalunya

James W. Tschanz
Circuit Research Labs, Intel Corporation

Reliability is a key issue in microprocessor design because users expect failure-free operation throughout the product's lifetime. However, failure rates will likely grow as transistors and wires shrink and the supply voltage scales slowly, leading to higher current densities and temperatures. As a result, transistor and wire degradation will accelerate and shorten the product's lifetime.

Electromigration, an undesirable consequence of driving current through wires, is a major source of wire and via failure.[1] EM mainly occurs in bidirectional wires (for example, data buses between caches or buses between cores in multicore processors) and power/ground grids. The conventional solution to EM is to use wider wires and vias because they have lower current densities so degrade more slowly (see the "Related Work in Electromigration" sidebar). However, the area overhead in the metal layer impacts interconnect density, increasing constraints on processor design and leaving significant performance on the table. Additionally, this solution adds some guard bands in the cycle time to tolerate EM degradation to some extent, which decreases operating frequency.

We've observed that driving similar amounts of current in both directions undoes the EM. *Refueling* is a microarchitectural technique that exploits this observation. It consists of injecting current whenever the amount of current in one direction is higher than the amount in the opposite direction. We use this technique to extend the EM lifetimes of bidirectional wires and power/ground grids by several orders of magnitude without requiring wider wires. In addition, because our technique reduces EM degradation, it can reduce guard bands, and so increase the operating frequency or reduce the affected blocks' latency. This type of ad hoc technique helps increase a chip's overall reliability by mitigating one of the most significant sources of failure for each block. Refueling might also solve the EM problem in some critical wires for future technologies. To the best of our knowledge, refueling is the first microarchitectural approach to make up for EM in bidirectional buses and power/ground grids in the field.

0272-1732/08/$20.00 © 2008 IEEE Published by the IEEE Computer Society

Related Work in Electromigration

Conventional solutions for electromigration in wires and vias rely on increasing wire width, which has a significant overhead in terms of area and replicating and resizing vias. Some researchers propose analyzing circuits to widen only the most vulnerable wires.[1] Despite some promising results, this technique is useless for circuits in which all the wires behave the same (such as bidirectional buses) because they all must be widened.

Some researchers propose reducing the activity in some wires to reduce EM, while not repairing the wires.[2] This technique's benefits are far from those of refueling, depend on the workload, and are applicable only if multiple instances of such wires are in place, which is rarely the case (an example is the bus between the data cache and the unified cache, DL0/UL1).

EM is considered inevitable in circuits, and chip manufacturers use some design rules to deal with it, especially in power/ground grids.[3,4] These rules redistribute the current when possible. They then measure the current that wires must drive and widen the wires as needed to satisfy the lifetime requirements.

Our previous work analyzed EM physics from a microarchitecture perspective, identified wires that might experience some degree of EM, and evaluated potential benefits of avoiding EM in future technologies.[5] However, the survey provided no solutions.

References

1. X. Xuan, "Analysis and Design of Reliable Mixed-Signal CMOS Circuits," PhD thesis, Georgia Inst. of Technology, Dept. of Electrical and Computer Engineering, 2004.
2. A. Dasgupta and R. Karri, "Electromigration Reliability Enhancement Via Bus Activity Distribution," *Proc. 33rd Ann. Conf. Design Automation* (DAC 96), ACM Press, 1996, pp. 353-356.
3. J. Lienig and G. Jerke, "Embedded Tutorial: Electromigration-Aware Physical Design of Integrated Circuits," *Proc. 18th Int'l Conf. VLSI Design* (VLSID 05), IEEE Press, 2005, pp. 77-82.
4. J. Tao et al., "Modeling and Characterization of Electromigration Failures under Bidirectional Current Stress," *IEEE Trans. Electron Devices*, vol. 43, no. 5, May 1996, pp. 800-808.
5. J. Abella and X. Vera, "Electromigration for Microarchitects," to appear in *ACM Computing Surveys*.

Electromigration

Metal wires are imperfect because of missing atoms (vacancies), impurities, boundaries between crystals of metal with different orientations (grain boundaries), and so on. These imperfections cause electrons flowing through the wires to collide with metal atoms, which are pulled in the direction of the current flow, causing EM.[2] High temperatures (which make metal atoms looser) and higher current densities (which increase the dragging force) increase metal atoms' likelihood to move, so voids and hillocks are more likely. Voids appear in those parts of the wire from which metal atoms are pulled, and eventually those parts of the wire can break. Meanwhile, metal atoms pile up in other parts of the wire to form hillocks, which can create shorts with neighbor wires. Black's law[2,3] describes the mean time to failure (MTTF) due to EM as:

$$\mathrm{MTTF} = \mathrm{A} \cdot j^{-\mathrm{n}} \cdot e^{\frac{Q}{\mathrm{k} \cdot T}} \qquad (1)$$

where A is a technology-dependent constant that we must determine empirically, j is the interconnect's current density, n is a constant depending on the metal used for the interconnect (1.1 for copper[1]), Q is the activation energy for EM, k is Boltzmann's constant, and T is temperature in Kelvin.

We compute j as follows[3]:

$$j = \frac{C \cdot V_{\mathrm{DD}}}{W \cdot H} \cdot f \cdot p \qquad (2)$$

where C is capacitance, W and H are the wire's width and height, V_{DD} is the supply voltage, f is the clock frequency, and p is the switching probability.

Equation 2 clearly shows that increasing wire and via widths would reduce the current density, which would reduce EM and extend the wire's lifetime. Replicating vias or increasing their size has a small area overhead. However, increasing the wire stripes' width has a significant area overhead because of the extra area needed for these wider metal stripes. This increase in area significantly impacts the design because it reduces the connection density, letting us place fewer wires and possibly leading to less efficient designs.

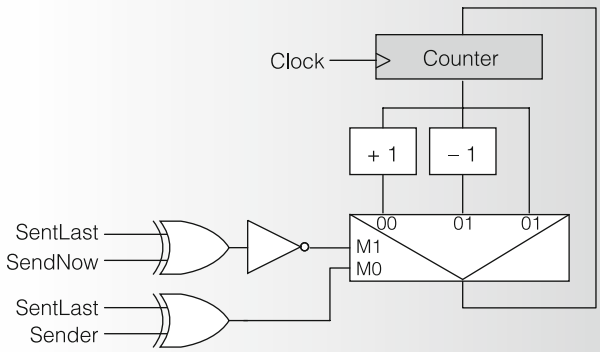
However, there is a self-healing effect that can undo EM. This effect occurs when current flows in both directions of a wire.[4] The parts of the wire that are prone to form voids when current flows in one direction are prone to form hillocks when it flows in the opposite direction. Thus, if the same amount of current flows in each direction, EM self-heals. This effect could extend wire lifetime by several orders of magnitude[5]—depending on the metal used, a lifetime can be more than 1,000 or 10,000 times longer.

As an example, take the low end of that range and assume that a current that is perfectly balanced in both directions extends a wire's lifetime by 1,000 times. Using Equations 1 and 2 and assuming that n is 1.1 (copper's value), we would have to

```
// UPDATE COUNTER
If (SentLast==0 and SendNow==1 and AtoB) then
    counter = counter + 1
If (SentLast==1 and SendNow==0 and BtoA) then
    counter = counter + 1
If (SentLast==1 and SendNow==0 and AtoB) then
    counter = counter - 1
If (SentLast==0 and SendNow==1 and BtoA) then
    counter = counter - 1
Endif
SentLast = SendNow

// UNDO ELECTROMIGRATION IF NEEDED
If (|counter| > EMthreshold) then
    If (counter > 0) then
        send "1" from B to A
        send "0" from A to B
    Else
        send "1" from A to B
        send "0" from B to A
    Endif
Endif
```

(a)



(b)

Figure 1. Refueling in bidirectional buses: algorithm for updating the counter and undoing electromigration (a), and the circuit to monitor EM (b).

increase the wire's width at least by a factor of 534 to achieve this same 1,000× lifetime extension. Obviously, no one would increase a wire width that much, because other sources of failure would become the limiting ones if nothing was done to solve them. Still, the example helps illustrate self-healing's potential benefits.

Refueling bidirectional buses

In a bidirectional bus, both ends (for example, A and B) can send signals. EM can cause them to degrade depending on signal patterns. For instance, the value in the bus might be "0." A might send a "1" to B, thus charging some current through A. Then B might send a "0" to A, thus discharging through B the current sent from A. Thus, the current charged through one end (A) can be discharged through the other end (B), causing some EM.

We monitor the amount of current flowing in each direction for bidirectional buses, and whenever the imbalance is above a given threshold we inject current in the proper direction (refuel) to undo EM. By avoiding the widening of bidirectional wires and vias while extending their lifetimes several orders of magnitude, refueling reduces the cost of bidirectional buses.

Mechanism

Based on the observation that we can approximate the EM caused by each current flow using Equation 1, we determine the amount of current flow required to make up for EM produced by past flows. We can do this using a single counter for each wire. We assume that similar currents are charged from both ends, A and B, whenever the bus state is 0 and any of the bus ends sends a 1 unless otherwise stated. Such amounts of current depend on the interface with the bidirectional bus at each end (that is, the number and size of transistors to feed). Because bidirectional wires mostly communicate symmetrical blocks (such as different cache levels or two cores), both interfaces can be similar. Thus, both current amounts are typically similar. Different amounts would require minor modifications to our algorithm.

Figure 1a shows the algorithm to update the counter and undo EM when required. We refer to the last value sent through the wire (0 or 1) and the current value as *SentLast* and *SendNow*. The algorithm increments the counter when current is charged through A or discharged through B. Similarly, it decrements the counter when current is discharged through A or charged

```
1. Send from A to B
2. Send from B to A
3. Send from A to B
```

```
// Send from A to B is:
for (each wire)
  If (counter > 0) then
    Send "0"
  Else
    Send "1"
  Endif
endfor
```

```
// Send from B to A is:
for (each wire)
  If (counter > 0) then
    Send "1"
  Else
    Send "0"
  Endif
endfor
```

Figure 2. Algorithm to refuel buses with multiple wires.

through B. We account for the amount of current sent twice—when charging and discharging—which gives the counter double the desired value. We solve this by doubling the EMthreshold value because we compare the counter with EMthreshold. If the counter is higher than EMthreshold, we inject current in the proper direction just once to keep the imbalance below the threshold and update the counter. To do this, we make the wire unavailable during refueling (as if it were serving a normal access) after any ongoing access finishes. For instance, if we must send current from A to B, we send 1 from A to B, and then we send 0 from B to A. Thus, current is charged through A and discharged through B. We update the counter's value accordingly. We perform refueling only once to keep the imbalance below the threshold instead of trying to achieve perfectly balanced refueling as often as needed. This is because future activity might balance the current discharged through each end.

Because we perform refueling soon after the imbalanced activity occurs (typically a few thousands of cycles later), the temperature of both the regular and the refueling activities is the same, leading to near-optimal refueling. Thus, the mechanism doesn't need to weight the amount of EM caused by the regular activity and the amount of EM recovered by the refueling activity with the actual temperature. We can achieve maximum lifetime benefits even if the degradation level remains near EMthreshold, because the amount of degradation occurring during some microseconds is negligible for a processor, which typically has a lifetime of some years.

Figure 1b shows the hardware implementation of the algorithm to update the counter. The number of bits required to implement the counter depends on EMthreshold. As we show later, when we use lower EMthreshold values, we need fewer bits to implement the counter.

Refueling multiple wires

Refueling one wire only needs two transitions, as the second part of the algorithm in Figure 1a shows. When we refuel one wire of a bus, we must make the whole bus unavailable for normal operation. We can exploit this unavailability to refuel all wires of the bus, even if they haven't reached the EMthreshold value.

Because different wires of a given bus might have different refueling requirements because of different activity patterns, we must do both types of refueling—that is, from end A to end B and vice versa. Figure 2 shows this process. Depending on the previous activity in the wires, we refuel some wires twice, which isn't detrimental because we refuel them in the proper direction. This is the case when one wire's counter is above 0 and the last signal through the wire (*SentLast*) is 1. In this scenario, we send 0 from A to B, which refuels the wire; then we send 1 from B to A; and finally, we send 0 from A to B, which further refuels the wire.

Evaluation

We evaluated an example of a core in which we apply refueling to both the data bus between the data (DL0) cache and the unified (UL1) cache, and the on-chip bus between the UL1 cache and memory. Although we evaluated these two concrete buses, our technique is applicable to any bidirectional bus in the chip. For instance, it has great potential for communication buses between cores in a multicore processor.

We collected the results presented here from an IA32 execution-driven simulator resembling Intel's Core microarchitecture. Our workload consisted of more than 500

traces (each consisting of 10 million consecutive IA32 instructions), which we obtained from a wide variety of programs. We split the UL1 cache latency (13 cycles) into three delays:

- cycles to send the request to the UL1 cache,
- cycles to access the cache itself, and
- cycles to send the reply back.

We assumed that the bus from DL0 to UL1 takes 4 cycles and accessing the UL1 takes 5 cycles. This assumption might be a bit pessimistic in terms of bus latency, but because our technique makes the bus unavailable during refueling, the penalty is larger. We considered that the on-chip bus to communicate between UL1 and main memory has the same latency as the bus between DL0 and UL1, because most of the memory latency is spent in the off-chip bus and the DRAM access. Thus, whenever we refuel a bus, it's unavailable for normal processing during 12 cycles because of the three transmissions (see Figure 2) of 4 cycles each (bus latency).

Performance evaluation. Table 1 presents performance results for different values of EMthreshold. The slowdown is practically negligible, especially for a threshold of 1,000—a 0.35 percent drop in instructions per cycle (IPC). Because this threshold requires 11 bits per counter [−1,000, +1,000], we consider a more practical design that uses a lower threshold (5 bits for an EMthreshold of 15). The slowdown for an EMthreshold of 15 averages 0.78 percent IPC. On average, this scenario requires one refueling every 1,000 cycles for each bus. As expected, the more refuelings, the higher the IPC loss.

Lifetime evaluation. We further studied refueling's benefits in terms of lifetime with respect to both the worst-case situation (any access produces imbalance always in the same direction), and the real case without refueling. As we stated previously, lifetime with refueling should be around 1,000 times longer than it is in the worst case. Table 2 shows the results for DL0/UL1 and UL1/memory buses normalized with respect to the worst case. Although real case bus lifetimes average approximately 28 times better than the worst case, this is far from the lifetimes achieved with refueling, which are 1,000 times longer[5] than the worst case (and 36 times longer than the real case). In particular, refueling lets us reduce buses' area by 4×, and, based on Equations 1 and 2, simultaneously extend their lifetime by 8×. Even if the worst-case and real-case buses are already affordable, refueling can reduce the delay guard band due to EM, leading to higher operating frequencies or lower latencies for the affected components.

Table 1. Slowdown and number of refuelings.

EMthreshold	Slowdown refueling both DL0/UL1 and UL1/memory (%)	Refuelings per million cycles	
		DL0/UL1	UL1/memory
1,000	0.35	954	798
15	0.78	1,570	1,341

Power and area implications. The overhead for updating counters is low. Updating the counters for each wire of the bus takes much less power than the power dissipated on the wire itself. Additionally, refueling occurs only 5.5 percent of the times that we use the DL0/UL1 bus (1 out of 18 misses in DL0) and 4.8 percent of the times that we use the UL1/memory bus (1 out of 21 misses in UL1). Hence, the energy consumed is negligible.

In terms of area, the circuit in Figure 1b requires few transistors (around 100). Hence, a 64-bit bus only requires 7,000 transistors. Furthermore, these transistors can be very small because they aren't in the critical path and the circuit itself is rather small. To illustrate that such an area overhead is negligible, we compare it against

Table 2. Lifetime of buses for different scenarios (normalized).

Scenario	Lifetime DL0/UL1 bus	Lifetime UL1/memory bus
Worst case	1	1
Real case	28	27
Refueling	1,000	1,000

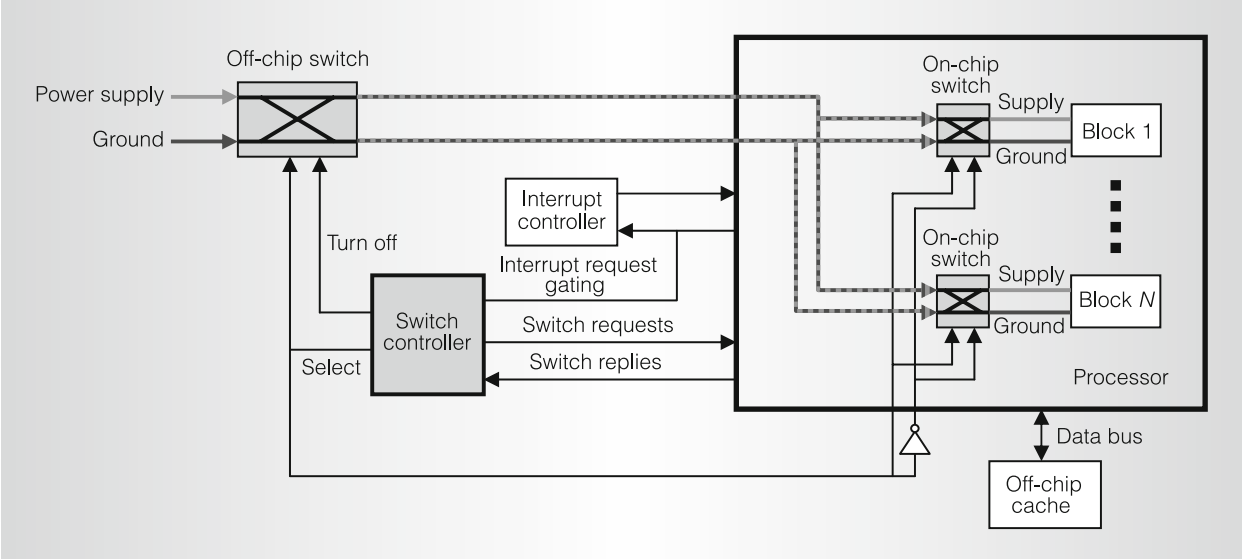


Figure 3. Mechanism to switch power/ground grids.

the area of a 32-Kbyte cache, which requires more than 2 million transistors just for the data bit cells (300 times more transistors).

Thus, with little hardware overhead and negligible performance degradation, we can monitor bus activity and refuel wires to undo EM, thus saving significant area in the metal layers.

Refueling power/ground grids

Power/ground grids involve numerous wires that occupy a vast area of some metal layers.[6] Based on the observation that both grids drive similar amounts of current in opposite directions, we switch both grids to drive current in different directions alternatively.

Mechanism

Power/ground grids are typically designed as interleaved identical stripes of metal whose width decreases as they move to lower metal layers.[6] Hence, at any metal layer, power/ground wires are identical and distributed in the same way. We propose a mechanism to refuel power/ground wires by switching grids. Because both grids are identical, driving current in one direction (power supply) or the opposite direction (ground) through metal stripes and vias doesn't impact the circuits below. Our only concern is how to perform this switch, because we must guarantee that blocks are always properly connected to power and ground, that processor state can't be lost, and that any device communicating with the processor won't notice the switch. By switching power/ground grids, we make both grids drive similar amounts of current in the opposite direction, so the refueling of such grids is near-optimal in both wires and vias.

Figure 3 illustrates our switching technique. Implementing our mechanism to refuel the grids requires some extra hardware:

- an off-chip controller to decide when to switch both grids,
- off-chip and on-chip switches,
- off-chip storage space to save the processor state if switching the grids is allowed during normal operation (the storage space can be inside the microcontroller but we can use any off-chip cache instead), and
- a mechanism to manage the interrupt controller so it doesn't release interrupts when the processor is unavailable.

The switch controller decides when to switch. It includes a register (time counter) implemented in nonvolatile memory that

tracks how long each grid provides the power supply. When a user turns the computer on, the switch controller selects (using the *select* signal) the least-used grid to drive power, while the other grid drives ground.

The blocks in Figure 3 can represent single transistors or full cores. Our mechanism refuels any power/ground wire and via between the off-chip and on-chip switches. The finer the granularity, the greater the number of power/ground wires and vias refueled. The proper granularity is strongly related to the devices used to implement the on-chip switches. How much power and area overhead is acceptable for a given design will depend on EM degradation's significance at the current technology node.

Figure 4 shows our proposed on-chip switch. Its four transistors have the same functionality as those used to gate structures for power savings.[7] We carefully size gate transistors in on-chip switches to provide enough current regardless of the circuit's activity. One way to size the transistors is to use the sum of the widths of all the transistors that could simultaneously switch.[7] That is, we use the sum of the widths of the *n*-channel metal oxide semiconductor transistors to size both of our on-chip switch's NMOS transistors, and the sum of the widths of its *p*-channel MOS transistors to size both PMOS transistors. If we consider that most of the area is devoted to memory-like structures (first- and second-level caches, branch predictors, and so on) and that few of these transistors can switch simultaneously, we can see that the overhead is small. However, we must still estimate it for each block in the chip. For instance, the worst case for a 1-Mbyte UL1 cache might be 2^9 bits changing among the 2^{20}—that is, 0.05 percent. Hence, the transistors' relative overhead is only significant for blocks in which a significant fraction of the transistors can switch simultaneously. If such blocks represent a significant fraction of the entire chip area, the overhead might be significant. Therefore, we must apply refueling only if we can't widen the power/ground grid wires to fit the EM requirements.

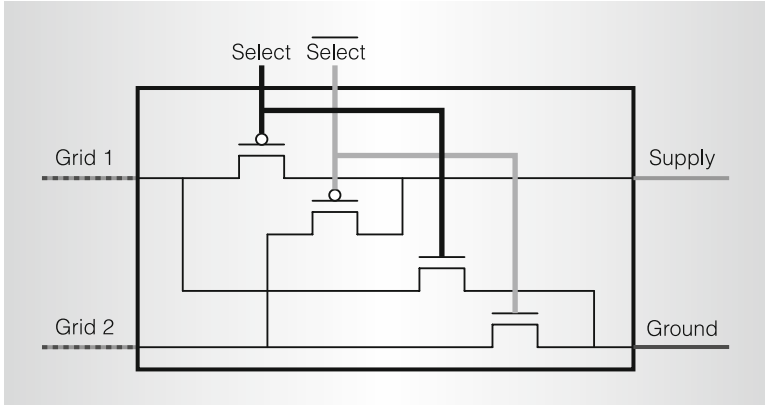


Figure 4. Implementation of an on-chip switch.

The overhead of the *select* wires to switch power and ground inside the chip is also low. Such wires aren't critical because they won't switch during operation. Using narrow wires is thus sufficient, even if they take microseconds to propagate and stabilize the select signal. Additionally, EM isn't a concern for these wires because they drive current only at switching. In addition, the wires are unidirectional buses that benefit from EM's self-healing effect.

Because users might rarely turn off their computers, we need a mechanism to switch supply and ground during operation. We can perform this online switching whenever the time counter exceeds a given threshold, which can be several minutes or hours. This process involves several actions:

1. The switch controller notifies the processor that it's going to switch using the *switch requests* signal.
2. The processor stops accepting interrupts and doesn't start any other direct memory access (DMA) transfer.
3. When all in-service interrupts and ongoing DMA transfers have finished, all cores flush their pipelines.
4. If caches aren't write-through, the processor copies dirty data back to off-chip memory.
5. The processor copies the cores' (registers') architectural state to off-chip storage (for example, off-chip cache).
6. The processor notifies the switch controller that it's ready to be turned off using the *switch replies* signal.

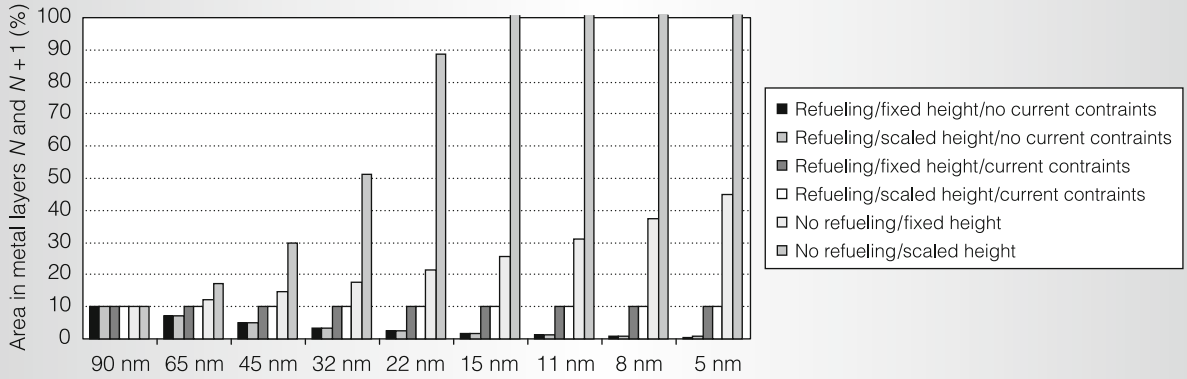


Figure 5. Area devoted to the power/ground grids.

7. The switch controller sets the *interrupt request* (IRQ) *gating* signal to ensure that the interrupt controller won't release interrupts when the processor is off.
8. The switch controller turns off the processor, inverts the *select* signal to switch the grids, and turns on the processor. It waits for some preset time after each operation to let the processor stabilize completely.
9. The switch controller notifies the processor through the *switch requests* signal that it can restore the cores' state, and waits until the processor signals that it's ready to resume execution.
10. The switch controller disables the IRQ gating signal and notifies the processor that it can resume execution.

During switching the system is frozen, so if switching takes too long the user will experience a glitch. Because switching grids doesn't involve any mechanical components and so shouldn't take long, we don't expect this to be an issue. The time required for switching depends on the technology, so it's design-dependent (probably some milliseconds). To reduce glitches, the system can perform switching at computer turn-on time, during hibernation, at speed-step changes of operation frequency, and so on.

Benefits of refueling power/ground grids

Because switching occurs at a coarse granularity (minutes or hours) and its impact is in the order of milliseconds, the performance impact is negligible. However, the benefits of refueling power/ground grids are huge in terms of EM lifetime and area savings.

To illustrate the significant improvements in terms of area and lifetime, we use Choi and colleagues' example, in which metal layers N and $N + 1$ have 3-μm wide wires for power and ground with a period of 60 μm.[6] If we move to the next technology generation (for example, from 90-nanometer to 65-nm), they must be 3.6 μm wide for the same wire height, or 5.2 μm for scaled wire height to keep lifetime constant based on expected switching capacity increase per area unit.[8] Assuming the gap between wires is at least as wide as the wires, the base case uses 10 percent of these metal layers' area. When moving to the next technology generation, the area devoted to the power/ground grids in these metal layers is 12 percent if the height doesn't scale and 17 percent if it does. On the other hand, if we use refueling, the area devoted to power/ground grids in these metal layers reduces to 7 percent or remains constant at 10 percent if needed due to current constraints.

Figure 5 illustrates the evaluation space. The first two columns show the results of using refueling and an unconstrained wire size—that is, the wire can shrink because there are no current constraints and lithography allows scaling. If the wire height is fixed, we scale the width for all technology generations; otherwise, the wire width increases after 8-nanometer technology but its area is below 1 percent even for 5-nm

technology. The third and fourth columns represent the same situations in the first two columns, but in this case we can't decrease the wire width due to current constraints (in addition to the EM constraints). Hence, the wire width remains at the initial 10 percent of the area. The last two columns show the results of using no refueling. In both cases, we must increase the wire width, which can make the design unfeasible because the wires need more than 100 percent of the area (15 nm for scaled height). Even when we can increase wire width, we must devote much more area to the power/ground grids, reducing the area available for other wires. As Figure 5 shows, area savings for 22 nm are as significant as 37× (88.5 versus 2.4 percent). In addition, based on Equations 1 and 2, refueling extends lifetime by 19× simultaneously for those wires and vias.

Mechanisms based on refueling can increase the reliability of other wires affected by EM, besides those we have discussed in this article, and thus increase overall chip reliability. We can address other sources of failure using similar ad hoc techniques to reduce the number of expected in-the-field failures per processor. MICRO

Acknowledgments

We thank the anonymous reviewers for their insightful comments. The Spanish Ministry of Education and Science under grants TIN2004-03702 and TIN2007-61763, the EU European Funds for Regional Development (FEDER), and the Generalitat de Catalunya under grant 2005SGR00950 partially supported this work.

References

1. J. Srinivasan et al., ''The Impact of Technology Scaling on Lifetime Reliability,'' *Proc. Int'l Conf. Dependable Systems and Networks* (DSN 04), IEEE CS Press, 2004, pp. 177-186.
2. J.R. Black, ''Electromigration Failure Modes in Aluminum Metallization for Semiconductor Devices,'' *Proc. IEEE*, vol. 57, no. 9, 1969, pp. 1587-1594.
3. F.M. D'Heurle, ''Electromigration and Failure in Electronics: An Introduction,'' *Proc. IEEE*, vol. 59, no. 10, 1971, pp. 1409-1418.
4. I.A. Blech and E.S. Meieran, ''Direct Transmission Electron Microscope Observation of Electrotransport in Aluminum Thin Films,'' *Applied Physics Letters*, vol. 11, no. 8, Oct. 1967, p. 147.
5. B.K. Liew, N.W. Cheung, and C. Hu, ''Projecting Interconnect Electromigration Lifetime for Arbitrary Current Waveforms,'' *IEEE Trans. Electron Devices*, vol. 37, no. 5, May 1990, pp. 1343-1351.
6. J. Choi et al., ''Modeling of Realistic On-Chip Power Grid Using the FDTD Method,'' *Proc. IEEE Int'l Symp. Electromagnetic Compatibility* (EMC 02), IEEE Press, 2002, pp. 238-243.
7. M. Powell et al., ''Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories,'' *Proc. Int'l Symp. Low Power Electronics and Design* (ISLPED 00), ACM Press, 2000, pp. 90-95.
8. J. Abella and X. Vera, ''Electromigration for Microarchitects,'' to appear in *ACM Computing Surveys*.

Jaume Abella is a senior researcher at the Intel Barcelona Research Center. His main research interests are hardware reliability and low-power designs. He received a PhD in computer science from the Universitat Politècnica de Catalunya, Barcelona, Spain.

Xavier Vera is a senior researcher at the Intel Barcelona Research Center. His research interests include reliable and variation-aware microarchitectures, and the design and analysis of high-performance and low-power computer architectures. He received an MS in computer science from the Universitat Politècnica de Catalunya and a PhD in computer science from Mälardalens Högskola at Västerås.

Osman S. Unsal is a group manager at the Barcelona Supercomputing Center. His research interests include computer architecture, parallel processing, and reliability. He received a PhD in electrical and computer engineering from the University of Massachusetts, Amherst.

Oguz Ergin is an assistant professor in the Department of Computer Engineering of TOBB University of Economics and Tech-

nology, Ankara, Turkey. He was a senior research scientist at Intel Barcelona Research Center before he joined his current university. His research interests include computer architecture and VLSI design. He received a PhD in computer science from the State University of New York at Binghamton.

Antonio González is a professor in the Computer Architecture Department at the Universitat Politècnica de Catalunya. He is the founding director of the Intel Barcelona Research Center, which focuses on new microarchitecture paradigms and code-generation techniques for future microprocessors. He received a PhD in computer architecture from the Universitat Politècnica de Catalunya.

James W. Tschanz is a researcher at Intel's Circuit Research Lab. His research interests include low-power and variation-tolerant circuits. He received an MS in electrical engineering from the University of Illinois at Urbana-Champaign.

Direct questions and comments about this article to Jaume Abella, Intel Barcelona Research Center, Intel Labs-UPC, Jordi Girona, 29, 3rd floor, 08034 Barcelona, Spain; jaume.abella@intel.com.