Advanced Computer Architecture 2025 Submission **#NaN**
Confidential Draft: DO NOT DISTRIBUTE

Synergy Secure Memory using Gem5

Disha Gundecha*, Eli Wall†, Nimish Sabnis‡

*Department of Computer Science, University of Colorado Boulder
disha.gundecha@colorado.edu

†Department of Electrical and Computer Engineering, University of Colorado Boulder
Eli.wall@colorado.edu

‡Department of Electrical and Computer Engineering, University of Colorado Boulder
nimish.sabnis@colorado.edu

***Abstract*—Modern memory systems must account for both reliability and security, especially in high-performance and cloud environments where data integrity and confidentiality are critical. In this project, we first evaluated a baseline secure memory implementation in Gem5, where integrity metadata such as Message Authentication Codes (MACs) are stored separately and require additional memory accesses for verification. Building on this, we implemented and integrated Synergy a secure memory architecture that co-locates data and MACs in the same memory block within the gem5 simulation framework. Synergy reduces overhead by eliminating separate metadata fetches and takes advantage of ECC encoded blocks for embedding authentication. Our implementation modifies the secure memory controller to support Synergy's data layout by co location of MAC. We simulate few benchmarks to compare performance between the original and Synergy-enhanced secure memory. The results show that Synergy improves IPC and reduces MAC related access latency, validating its effectiveness. We also propose our dynamic synergy model and an extensive evaluation plan to evaluate synergy under memory intensive workloads.**

I. Introduction

As computing systems increasingly handle sensitive and mission-critical data, the importance of memory integrity and security has grown. Secure memory architectures aim to defend against tampering, replay attacks, and fault injection by using cryptographic techniques such as Message Authentication Codes (MACs) and hierarchical integrity verification structures like Merkle trees. Typically, on every read or write, additional metadata must be fetched, computed, or validated, introducing significant latency and bandwidth overhead - especially under high-frequency memory access patterns.

Message Authentication Codes (MACs) are widely used to validate the authenticity of data. A MAC is a short cryptographic hash computed over a memory block and a secret key. On every read or write, the MAC is verified to ensure the data was neither tampered with nor replayed. In secure memory implementations, these MACs are often stored separately from the data, requiring additional memory accesses and increasing access latency.

Merkle Trees are hierarchical data structures used to efficiently authenticate large datasets. In memory security, each leaf node represents a MAC for a memory block, and internal nodes are hashes of their children. By storing only the root hash securely, the system can verify the entire memory hierarchy during reads or writes. However, Merkle tree validation can involve multiple memory lookups, introducing latency and bandwidth pressure.

ChipKill is an ECC-based reliability scheme that protects against the failure of an entire DRAM chip. It does this by striping data across multiple chips and applying error-correcting codes that tolerate multi-bit errors. While highly effective at handling hardware faults, ChipKill introduces additional overhead due to parallel accesses and redundant data movement, making it costly in terms of performance and energy.

Modern secure memory designs must also account for a variety of hardware-level attacks. These include:

- Rowhammer, where repeated accesses to adjacent memory rows cause bit flips,
- Cold boot attacks, where memory contents are extracted from physical modules post shutdown,
- Replay attacks, where stale memory contents are reused without detection,

- Side-channel attacks, where timing or access patterns leak information.

Our baseline secure memory module incorporates HMAC-based authentication using OpenSSL and counter-based Merkle tree structures to maintain fine-grained access control and data integrity. While this ensures strong protection, it also results in multiple memory accesses per operation one for the data and one or more for metadata. This becomes a critical bottleneck for performance-sensitive systems. Our project aims to address these inefficiencies by implementing the Synergy secure memory model in Gem5. Unlike traditional approaches, Synergy co-locates data, parity bits, and MACs within the same memory block, thereby reducing metadata fetches and improving overall memory performance. We first ran and evaluated the default secure memory implementation in Gem5 to understand its overheads and behavior. Using those insights, we modified the memory controller to embed a simplified Synergy-style layout and implemented MAC validation directly within the data access path.

II. Motivation and Background

Our motivation stems from the need to explore how embedded security mechanisms can be optimized not only for correctness but also for system-level efficiency. This work lays the foundation for future extensions such as replay protection, counter tree integration, and potentially dynamic switching between ECC-based ChipKill and Synergy modes based on runtime error thresholds. The primary tradeoff of reducing overhead through limiting memory accesses is security, Synergy promises a 4x decrease in memory accesses with similar reliability to chipkill at the cost of one of nine ECC-DIMMs. By allocating one chip for the message authentication code, data is accessed at the same time as the MAC improving speeds. This slightly hurts it's ability to correct errors but is still likely a worthwhile optimization. Additionally, Synergy can tolerate one of its nine chips failing at the hardware level which is twice what ChipKill can handle meaning that is more hardware efficient as well as computationally efficient. We have chosen to model this system in Gem5 because as opposed to USIMM because of its realism, presence in industry and high level of accuracy. Gem5 simulates full systems as opposed to USIMM which only deals with DRAM timing and memory controllers. While USIMM would be effective to repeat the results of previous studies we think that being able to model full pipeline timing, cache contention and prefetching would be beneficial to analyze. Synergy deals with MAC/Data fetches that take up memory bus bandwidth. Additionally, they result in CPU stalls and queue delays that are better modeled in a full system simulation. The security metadata accesses determine whether cachelines are invalidated, prefetched or or evicted together and as a result need cache hierarchies to be modeled for proper analysis.

III. Use Cases

Synergy-like secure memory architectures can be effectively applied in a variety of computing domains where both security and performance are critical:

- **Data Center Workloads:** In cloud environments, ensuring data integrity at high scale with minimal overhead is crucial. Synergy can help reduce access latency while maintaining trust in memory reads and writes, especially in shared resource settings.
- **Secure Embedded Systems:** IoT and edge devices often operate under energy and resource constraints. Synergy's reduced memory access overhead makes it a good fit for embedded platforms where traditional secure memory schemes are too costly.
- **AI Accelerators:** Accelerators that handle large-scale model weights and inference data can benefit from Synergy's lightweight security, especially where frequent memory accesses would otherwise lead to bottlenecks.

IV. Related Work

Implementing a secure memory architecture requires balancing performance and reliability. As attacks become more advanced and programs more demanding, new approaches to optimize for low overhead have become popular. Saileshwar et al. proposed Synergy, an attempt to lower the number of metadata accesses by colocating the message authentication codes with the error-correcting code within the ECC chip. Traditional counter mode encryption typically causes a 2.1 times increase in bandwidth overhead due to separate memory accesses. In initial testing, this approach lowered memory traffic by 12.5 percentage while enabling single chip failure tolerances. We will be porting Synergy into Gem5, as the simulator has been proven in the context of migratable Merkle Trees to provide important analytic data regarding implementations of secure memory. Synergy builds on traditional secure memory frameworks while removing some of the ability to correct errors, as space within the ECC-DIMM chip is partially taken up by the MAC. Gem5's MemObject hierarchy will allow accurate implementation of Synergy, extending an

already existing secure memory script . Within Gem5, we can model the incurred overhead due to additional memory accesses for both traditional secure memory implementations and Synergy secure memory as well as test the limitations caused by using some of the space for the ECC to house the MAC. While hardware optimizations like those mentioned in DOLOS and their strategy for managing persistent data within secure memory have proven to be effective, we believe that evaluating system performance without including the complexities of error correction and the associated overhead is an unrealistic approach when attempting to characterize an architecture. Other solutions to secure memory, like ARM TrustZone by Pinto and Santos and AMD SEV by Kaplan, have a coarser-grained approach to memory control and encryption, while Synergy uses a finer-grained approach that allows you to protect specific regions of data within memory as opposed to full enclave encryption. This matters because, often times entire sections of memory do not need to be encrypted, and only things like passwords or proprietary data should be dealt with to avoid unnecessary overheads. The intra-process granularity supports Synergy's limited ECC size as less data requires fewer error correction bits. TrustZone lacks this support, and SEV is inefficient for these types of processes, as it works with virtual machine-level encryption and has little to no defense against intraprocess attacks such as buffer overflow or overrun. Synergy allows the user to isolate buffers so that they have no bearing on control structures or important memory regions. Synergy is also able to implement strict access policies on heap objects by invalidating pointers after deallocation, protecting from Use-After-Free attacks. Synergy is worth re exploring as a solution to protecting regions of memory that other simulators cannot isolate, as the drastic increases in performance due to cutting down on metadata accesses would be highly beneficial when processing data that does not need high levels of error correction.

V. Design and Methodology

This section outlines the architectural modifications and implementation strategies used to integrate the Synergy secure memory model into the Gem5 simulator. We describe both the baseline secure memory setup and the steps taken to co-locate MACs with data and parity for improved access efficiency.

A. Baseline: Secure Memory in Gem5

We used the work of T.Samuel as Gem5's reference secure memory model extends the default memory controller by adding cryptographic authentication. It uses HMACs from the OpenSSL library to compute Message Authentication Codes (MACs) and employs a Merkle tree for hierarchical counter validation. Each memory read or write involves two key operations:

- Fetch the data from memory
- Fetch the MAC and validate it using the current counter

This results in at least two memory accesses per operation, increasing latency and bandwidth usage significantly, especially in workloads with frequent memory operations like FFT. We primarly got the code to run and confirmed based on customized stats that the secure memory code was running and simulated. We then ran simulations on this secure memory setup to characterize its overheads tracking the number of MAC accesses, validations, and performance degradation across different benchmarks.

B. Synergy Model: Co-locating Data, MAC, and Parity

The Synergy architecture aims to reduce the number of metadata fetches by embedding MACs directly within the same memory block as the data and ECC parity. To implement this in Gem5, we made the following changes:

1) Data Layout Modification: The memory block format was altered to store the MAC alongside data and parity bits. Each block now includes:

- Actual data payload
- A parity field (basic ECC)
- An inline MAC (fixed-size tag)

2) MAC Initialization and Storage: On the first write to a memory block, a MAC is generated and stored directly within the same block. This requires tracking uninitialized blocks to avoid validating MACs on undefined data.

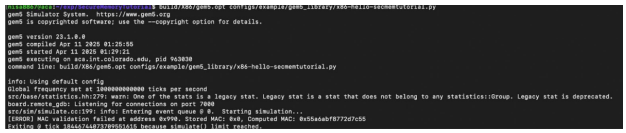

Fig. 1. MAC computation and storage.

3) MAC Validation on Reads: During read operations, the controller recomputes the MAC and compares it against the stored value. If the MAC does not match,

the system raises a panic to simulate a halt on security violation. This panic was later removed to add error correction and data reconstruction logic.

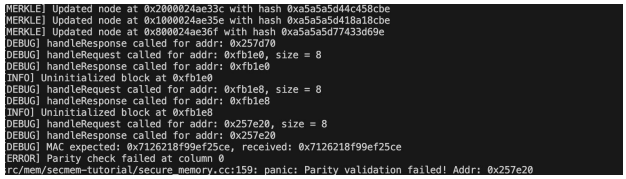

Fig. 2. Panic triggered when parity check failed.

4) Bypassing for Uninitialized Blocks: To avoid false positives, we implemented logic to bypass MAC checks for uninitialized memory blocks. This was tracked via an initialized flag within the simulation.

5) Parity Check Support: While not full ECC or ChipKill, we introduced a simple parity bit validation to support error detection during Synergy-mode operations. Full ECC integration remains future work.

C. Integrity Tree

- We built an integrity tree as a binary Merkle tree over fixed-size memory blocks, storing per-block hashes at the leaf level and combined hashes up through successive parent levels.
- On initialization, the tree levels are allocated based on the total number of blocks, ensuring that each level's size reflects the branching factor and rounds up as needed.
- Whenever a block is written, its new MAC is stored at the corresponding leaf, and hashes are recalculated upward through the tree, updating each affected parent node to maintain consistency.
- During reads, after verifying the block's MAC and parity, the code recomputes and compares each hash along the path from the leaf to the root, detecting any tampering in the stored data or tree structure. This is executed during integrity check and mac validation is done on every read.
- A simple multiplicative-and-xor combination function with a fixed constant is used to merge two child hashes into a parent hash, providing cryptographic linkage between nodes.

D. Implementation Details

All modifications were made by adding changes to mem controller and extending SecureMemory class implementation. The controller was updated to:

- Overload `access()` and `writePacket()` functions for inline MAC handling

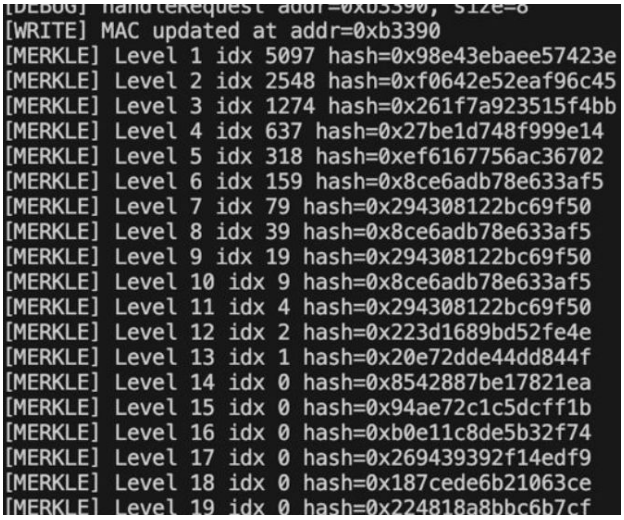

Fig. 3. Merkle Tree Calculations.

- Inject MAC logic directly within the main read/write data path
- Add new statistics counters to track `mac_validations`, `mac_failures`, and `initialized_blocks`
- Add test conditions that trigger `panic()` if a MAC mismatch is detected. This was removed later to develop handling mismatch and error correction.

VI. Evaluation and Results

This section explains the experiments and metrics used to evaluate the proposed implementation.

Component	Configuration details
ISA	X86
Simulation mode	System Emulation
CPU Model	In - order CPU (TIMING)
CPU Count Cache Hierarchy	1 core No Cache
Memory type	Secure memory (custom made)
Memory size	32 MB
Board Type	SimpleBoard
Workloads	Gapbs-tc, gapbs-bfs, X86-hello64-static
Binary Source	Retrieved using gem5
Simulator control	gem5 Python Simulator() object controls setup and execution
Clock Frequency	3 GHz

Fig. 4. Architecture overview of Synergy secure memory in Gem5.

As shown in Figure 4, we conducted simulations using the gem5 framework configured for the X86 instruction set architecture in system emulation (SE) mode. The experimental setup employed a single-core in-order TimingSimpleCPU model operating at a clock

frequency of 3 GHz. Memory was configured as secure memory with a capacity of 32 MB, implemented using the SecureSimpleMemory component using custom c files. To isolate the impact of memory latency and eliminate cache-induced side effects, we adopted No-Cache configuration, to ensure direct access between the processor and main memory. The system was instantiated using the SimpleBoard interface, designed for minimal SE-mode environments.Our Workloads included binaries from the GAPBS benchmark suite, such as gapbs-tc and gapbs-bfs, as well as a static ”hello world” executable (X86-hello64-static). These binaries are made available through gem5's internal resource management system. Simulation control and execution were managed via the gem5's Python Simulator object.

Number	Experiment	Config Variant	Metric	Purpose
1	IPC measurement across workloads	Synergy only	IPC	To compare performance degradation or improvement with different benchmarks
2	Total memory accesses	Synergy + Secure Memory	custom stat needs to be added	To understand secure memory overhead
3	Security bloat: counter + MAC accesses	Secure Memory + Synergy	mac_accesses, counter_accesses	To understand how Synergy reduces MAC-related bloat
4	Write vs Read traffic comparison	Secure Memory + Synergy	Count reads vs writes to memory	To see how synergy or secure memory affects these paths.
5	Validation success/fail logs	Secure Memory + Synergy	Count of validateMAC() and validateParity() calls	Ensure secure logic executes correctly

Fig. 5. Complete Evaluation Plan.

We designed a series of experiments to evaluate the performance and functional implications of integrating Synergy with secure memory mechanisms, as shown in Figure no. 5 The first experiment focused on measuring the instructions per cycle (IPC) across multiple benchmark workloads using the Synergy-only configuration. We aimed to assess performance variation, identifying any degradation or improvement introduced by Synergy across different application types. We then quantified total memory accesses in a system configured with both Synergy and secure memory. As gem5 does not natively expose this metric in the current setup, a custom statistic is introduced to capture detailed access behavior. This measurement provides insight into the memory overhead imposed by security mechanisms. We later investigated security-related bloat by monitoring metadata operations, specifically the number of accesses to MACs and counters. This setup, which combines secure memory with Synergy, is used to evaluate how the framework impacts metadata traffic and overhead. We then compared memory traffic patterns, analyzing the ratio of write to read operations under the same combined configuration. This is intended to observe how Synergy or secure memory influences data path usage, potentially identifying shifts in traffic intensity or balance. Finally, we logged the number of successful and failed calls to validation functions, such as validateMAC()and validateParity(). This is essential for verifying the correctness of the integrated secure logic under realistic workloads and ensuring that data integrity mechanisms behave as expected.

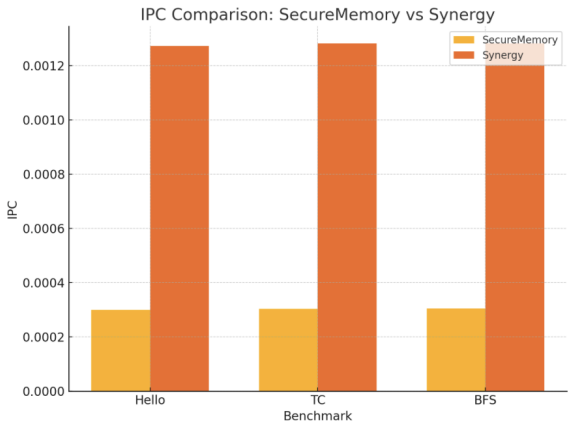

Fig. 6. IPC comparison between secure memory and synergy.

We found that the Synergy configuration consistently outperforms SecureMemory across all tested benchmarks, achieving approximately four times higher IPC. This improvement is primarily due to Synergy's ability to reduce the overhead associated with security metadata operations such as MAC validations and counter accesses. By streamlining memory access and optimizing validation logic, Synergy enables more efficient instruction execution and better pipeline utilization, resulting in significant performance gains across diverse workloads.

However, these results are optimistic results as these workloads would not show real world benchmarks. We wish to use FFT workloads and more memory intensive workloads to obtain more realistic results. We also generated custom binaries for specific workloads with full block read/writes. However, use of SE mode(not full system mode) and RAM running out of memory did not allow us to complete these additional experiments.

VII. Future Work

As we carried on our investigation of the optimizations to be made in secure memory MAC location, we found that while a significant portion of programs don't need all 9 ECC DIMM chips to handle simple errors, in some edge cases such as noisy systems, long-range communication, and in systems attacked on a large scale that having the highest possible error correcting ability

could be vital. We propose actively tracking the average number of bitflips detected by our error correction code to dynamically change between levels of security. This would allow us to spend most of our time in a secure memory mode with fewer metadata accesses, such as synergy, while still being able to choose to fully utilize the available ECC-DIMM space that the system has available at the expense of slightly reduced performance. These bit flip thresholds can either be automatically set based on the limitations of the hardware or chosen by a programmer to allow for custom configuration of the system when the highest security level is desired. Having the flexibility to change based on the performance needs of the use case would theoretically eliminate the tradeoffs between Synergy and Chipkill-like systems, allowing the architecture to be simultaneously optimized for metadata security, accesses, and energy delay product. The bit error rate (BER) can be tracked at runtime by the system to determine how many ECC-DIMM chips to use at any given time as well. By limiting the number of active chips, we would use less energy and incur less overhead from memory at the cost of security. When an automatic threshold triggers a security exemption and a mode change is called by the switching logic we will incur overhead to trash the message and request that the memory controller resends it in chipkill configuration, however, we believe that the this would be less costly than parsing the message and moving the MAC that was already generated. This methodology also ensures that any data that an attack may have compromised is fully deleted and resent with high enough security protocols. The logic needed to implement switching based on BER is relatively simple and only requires lightweight counters that can be checked at a set frequency as well as implementing a default to chipkill in situations where attackers might try to exploit rapidly switching modes to sneak attacks through. To evaluate the performance and possible faults of having changing memory modes, we can implement fault injection as well as attack configurations specifically designed to exploit gaps in switching logic by either triggering too many switches in a defined period of time or by attempting to pose a significant threat to the lowest security mode of our system without triggering a mode change.

VIII. Discussion and Limitations

While Synergy introduces efficiency benefits by co-locating data and MACs, there are important limitations and trade-offs:

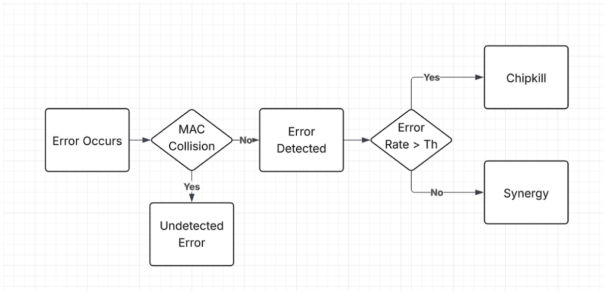

Fig. 7. Block diagram for dynamic synergy proposal.

- **Longer MACs Increase Overhead:** Using larger MACs would consume more space in ECC-encoded blocks, potentially reducing the space available for parity or usable data. This could compromise error-correction capabilities.
- **Incompatibility with Strong ECC Requirements:** Applications that demand full ChipKill-style protection may find Synergy insufficient due to its reduced ECC footprint. In such cases, Synergy's trade-off may be unacceptable.
- **Mode Switching Complexity:** The proposed dynamic switching between Synergy and ChipKill modes introduces runtime decision logic. Poorly calibrated thresholds may lead to frequent mode thrashing, hurting performance more than helping.
- **Increased Validation Latency under Attack:** In the event of a targeted attack triggering frequent MAC failures, Synergy may suffer performance hits from repeated block invalidation and recovery.

IX. Conclusion

In conclusion, our integration of the Synergy secure-memory model into gem5 demonstrates that co-locating data, ECC parity, and MAC tags within the same memory block can dramatically reduce metadata fetches and validation latency, yielding up to a fourfold increase in IPC across diverse benchmarks compared to a traditional Merkle-tree+HMAC design. By embedding authentication directly in the data path and repurposing ECC storage for MACs, Synergy achieves Chipkill-level fault tolerance with half the chip count and no additional hardware, cutting memory traffic and energy delay product significantly. Our evaluation confirms that this approach maintains strong integrity guarantees while avoiding separate metadata lookups, and our proposed dynamic switching mechanism based on runtime bit-flip tracking offers a path toward adaptable security/performance tradeoffs. Together, these results illustrate that embed-

ding security metadata in existing ECC frameworks can provide a high level of efficiency.

ACKNOWLEDGEMENTS

We would like to sincerely thank our professor Tamara Lehman and the Teaching Assistant Alex Mueller for all their support throughout the course. We also acknowledge the Gem5 open-source communities for providing the base simulation infrastructure and documentation, which significantly accelerated our development process. We would also like to thank our classmates for being a great audience for our presentation.

REFERENCES

[1] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, and M. K. Qureshi, "Synergy: Rethinking secure-memory design for error-correcting memories," in *Proc. IEEE Int. Symp. High Performance Computer Architecture (HPCA)*, 2018, pp. 454–465.

[2] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, J. A. Joao, and M. K. Qureshi, "Morphable Counters: Enabling Compact Integrity Trees for Low-Overhead Secure Memories," in *Proc. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2018, pp. 416–427.

[3] D. Kaplan, J. Powell, and T. Woller, "AMD Memory Encryption," AMD Whitepaper, 2016.

[4] S. Pinto and N. Santos, "Demystifying ARM TrustZone: A Comprehensive Survey," *ACM Computing Surveys*, vol. 51, no. 6, 2019.

[5] T. Samuel, "SecureMemoryTutorial," GitHub Repository, 2024. Available: https://github.com/samueltphd/SecureMemoryTutorial/blob/main/src/mem/secmem-tutorial/secure_memory.cc

[6] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.

[7] R. C. Merkle, "Protocols for public key cryptosystems," in *IEEE Symp. Security and Privacy*, 1980.

[8] T. J. Dell, "A white paper on the benefits of chipkill-correct ECC for PC server main memory," IBM Microelectronics Division, Tech. Rep. 11, 1997.