A Tunable, Software-based DRAM Error Detection and Correction Library for HPC

David Fiala (NCSU), Kurt Ferreira (SNL),
Frank Mueller (NCSU), Christian Engelmann (ORNL)

NC STATE UNIVERSITY
Department of Computer Science

Sandia National Laboratories

OAK RIDGE National Laboratory

Motivation

- Silent Data Corruption (SDC) → undetected soft errors that result in corruption in storage (Processor, Cache, Disks, RAM, etc)
- SDC faults may manifest themselves as bit-flips in memory
  - Some bit-flips are not correctable or even detected even with hardware ECC protection
  - Exacerbating this situation, when SDC goes undetected, applications continue to run while reporting **invalid** results
    - This is a severe problem for today's large-scale simulations
- Server class hardware supports ECC; one common form provides single error correct, double error detect (SECDEC)
- Non-server class hardware provides no protection
- **Today there is no generic way to protect applications without ECC**
- **Even with ECC, hardware SECDEC protection fails you when 3 or more bit flips occur**
- SDC events are expected to grow dramatically as chip density, heat generation, and core counts increase in larger HPC systems

LIBSDC: A software-based solution

- **Idea:** Provide SDC protection in software by tracking accesses to memory regions and ensuring their integrity before an application uses that region's data
- For each region of memory choose one or both:
  - *Hashes*: Detect memory corruption via hash mismatches
  - *ECC/Hamming Codes*: Correct some SDCs, even if hardware ECC failed to detect them
- Application-independent and transparent
  No code changes required for applications
- Using the MMU provides a granularity of a single page for a region

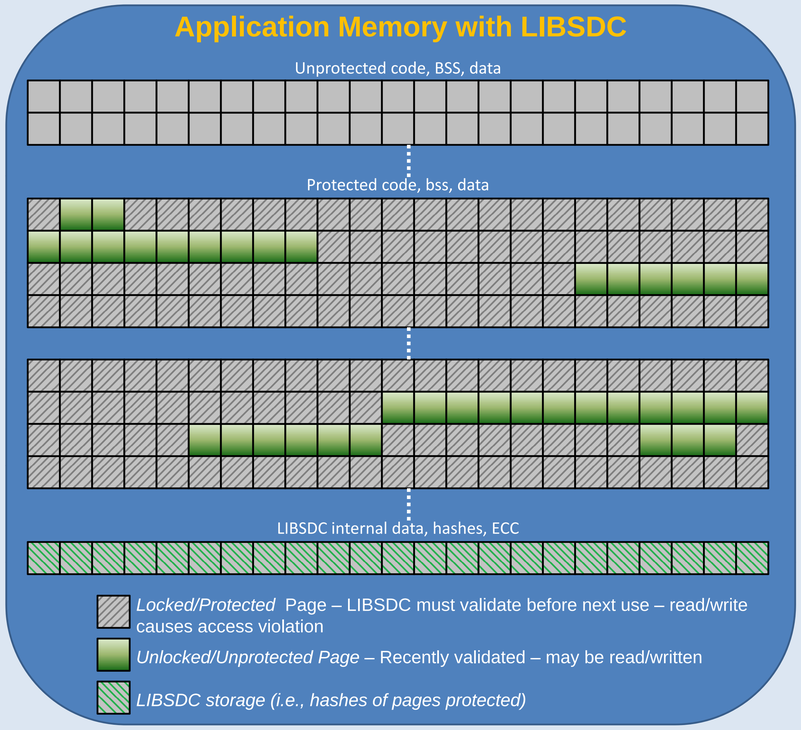
Method	SHA1 Hashing (4KB pages)	72/64 Hamming Codes (ECC)
Storage Overhead	0.49%	12.5%

Tuning

- ***Max-unlocked:*** Adjust the maximum number of pages to be allowed "unlocked" at a time. Ideally set at the number of pages in an application's working-set during runtime
- ***Hash or ECC:*** Choose if you desire SDC detection and/or correction
- **Memory to protect:** Choose or combine:
  - Application's heap, bss, data, and/or code
  - Other linked libraries (optionally include or exclude)

Implementation

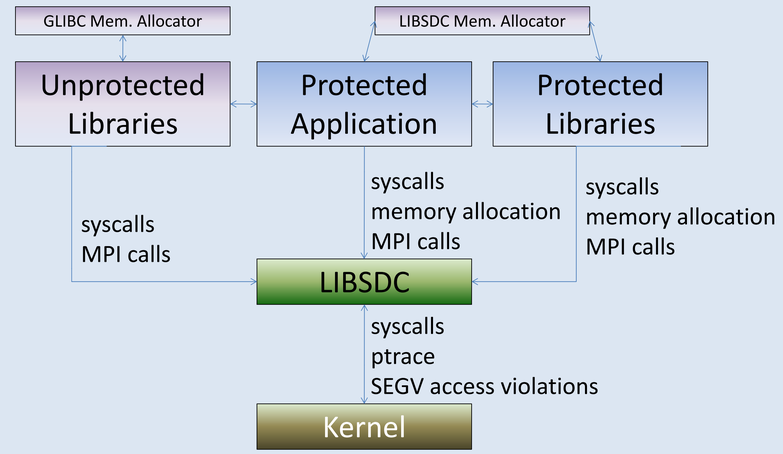
- Page tracking is accomplished with **mprotect** (removing read/write)
  - Each new page access triggers an access violation which allows LIBSDC to monitor application activity (SEGV handler)
  - Swap out unlocked pages upon reach max-unlocked
- Permission changes break many libraries
  - Syscalls will fail if passed protected pointers
    - *ptrace* is used to intercept all syscalls and unprotect pointers within syscall parameters
  - MPI implementations will fail with protected pointers, too
    - LIBSDC's MPI profiling layer wrappers unprotect passed buffers
  - Separate memory allocators prevent unprotected libraries from sharing virtual addresses in the same page as protected data



Memory Verification

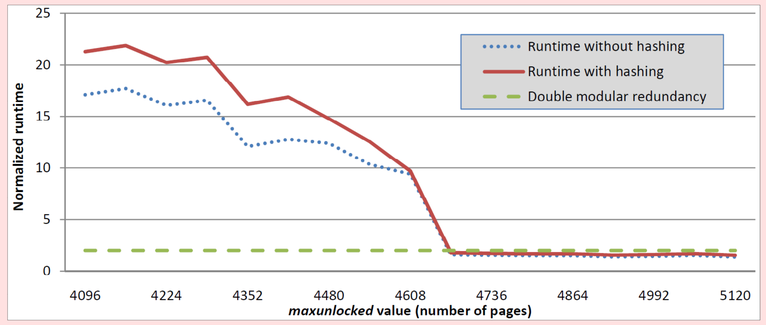
```
On page request (inital read or write):
  If page is locked:
    Perform hash of page
    Compare current hash with previously stored known-good hash
    If any inconsistency found:
      Notify the presence of SDC and report location
      Terminate application / Rollback to previous checkpoint
    Mark page as unlocked (mprotect)

On page lock:
  Calculate new hash of entire page
  Storage hash in separate location
  Mark page as locked (mprotect)
```



HPCCG Results

- HPCCG – A Sandia Natl. Labs kernel conjugate gradient solver from the Mantevo Miniapps
- 256 processes on a 768x8x8 matrix
  - AMD Opteron 6128 (Magny Core) – 16 cores per node
  - 32GB RAM per node
  - 40Gbit/s Infiniband



- Ideal max-unlocked around 4096-5120 to match working-set size
- **On average, about 15% of overhead spent on page hashing**
- **53% overhead vs baseline when tuned with optimal max-unlocked**

Future Work

- Recent related work has shown (Ferreira, SC11) page hashing on GPUs can greatly reduce the overhead spent hashing on the CPU
- Replace LIBSDC's FIFO policy of unlocked pages with a smarter frequency-based algorithm
- Investigate using kernel page tables/invalid bit to reduce the overhead incurred from frequent use of mprotect (TLB flushes may be responsible for much overhead)

This work was supported in part by NSF grants CNS-1058779, CNS-0958311, DOE grant DE-FG02-08ER25837, a subcontract from Sandia National Laboratory. and by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. De-AC05-00OR22725.