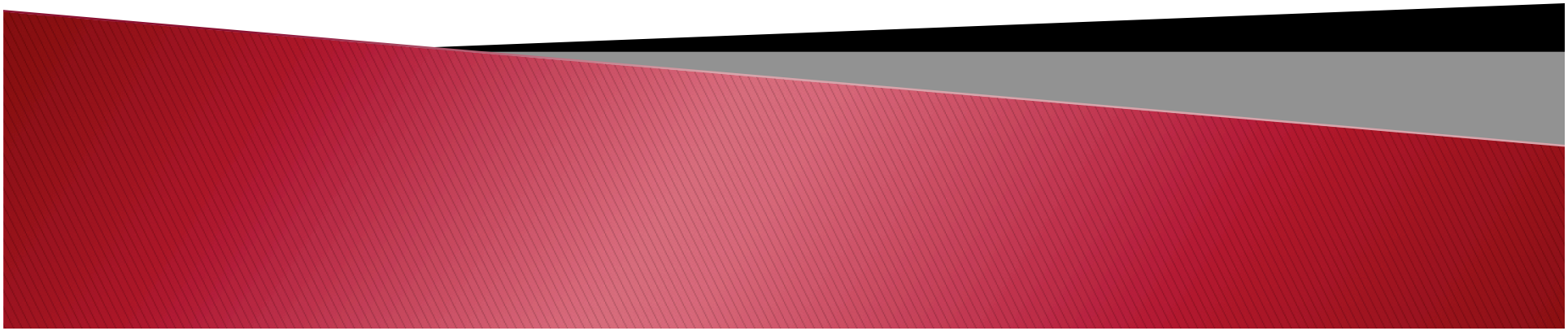
UNM SCHOOL of ENGINEERING
Department of Computer Science

Evaluating the Feasibility of Using Memory Content Similarity to Improve System Resilience

Scott Levy, Kurt B. Ferreira, Patrick G. Bridges,
Aidan P. Thompson and Christian Trott



Resilience Matters

- In large-scale systems, errors are proportional to socket count
- Current systems have 10s of 1000s of sockets; the first exascale system will have 100s of 1000s of sockets
- Traditional resilience strategies (e.g., coordinated checkpoint/restart) mean that less and less time is spent doing actual work
- Either: (a) develop more efficient ways to recover from failure; or (b) reduce the rate at which errors lead to failure

DRAM Errors

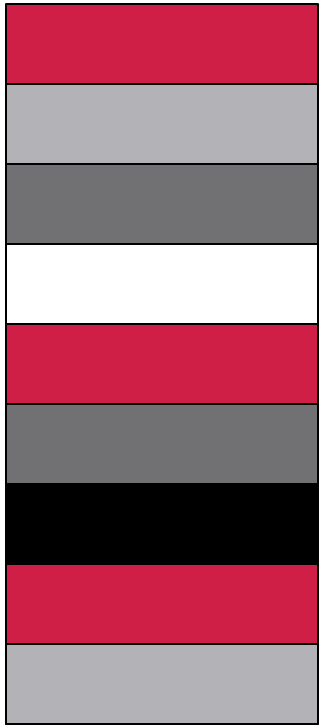
- One of the most common sources of node failure is memory errors
- On an x86 system, a DRAM ECC error results in a machine check exception (MCE)
- The consequence of an MCE varies by operating system
- In modern Linux, several mitigation strategies are employed
- If none of these strategies work, the kernel terminates the application(s) that have the faulted page mapped into their address space

Page Categories

- DUPLICATE : pages with at least on non-zero byte whose contents match at least one other page
- ZERO : all-zero pages
- SIMILAR : pages that : (a) are neither duplicate nor zero; and (b) can be paired with a another page such that the difference between the two can be compactly represented
- UNIQUE: all other pages

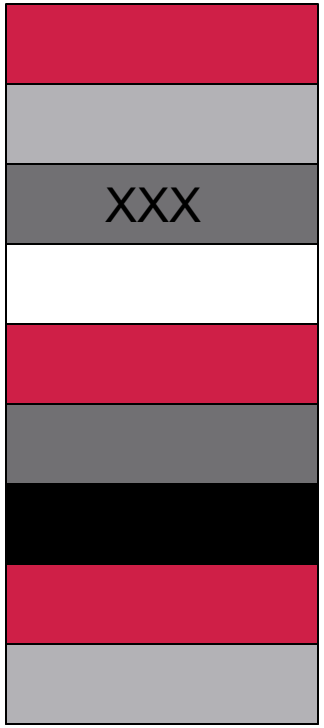
For the data presented here, "compactly represented" means a `cxbsdiff` patch that is smaller than 1024 bytes

Repairing DRAM ECC Errors



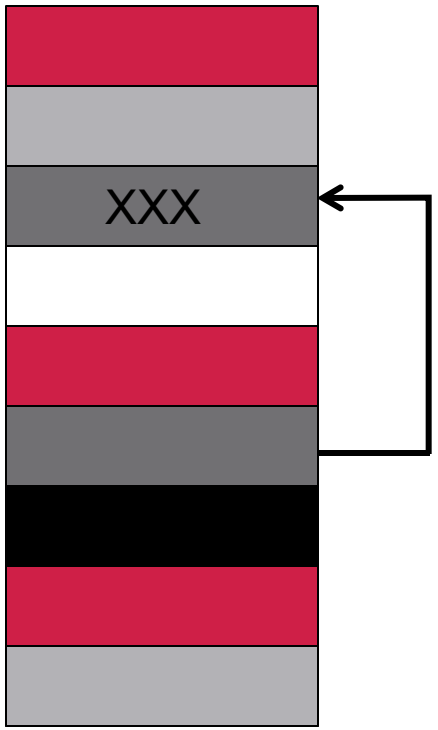
Begin with uncorrupted block of memory in which we've colored memory to reflect similarity.

Repairing DRAM ECC Errors



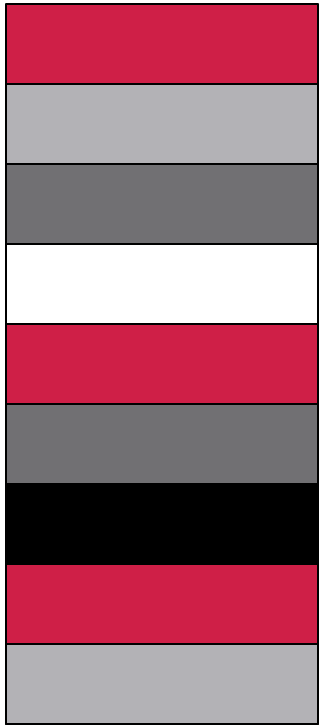
DRAM ECC error is detected indicating that the contents of memory are no longer valid.

Repairing DRAM ECC Errors



Using the contents of a similar page in memory we can reconstruct the contents of the corrupted page.

Repairing DRAM ECC Errors



After the faulted page has been repaired, the application can continue without requiring a restart.

Evaluation

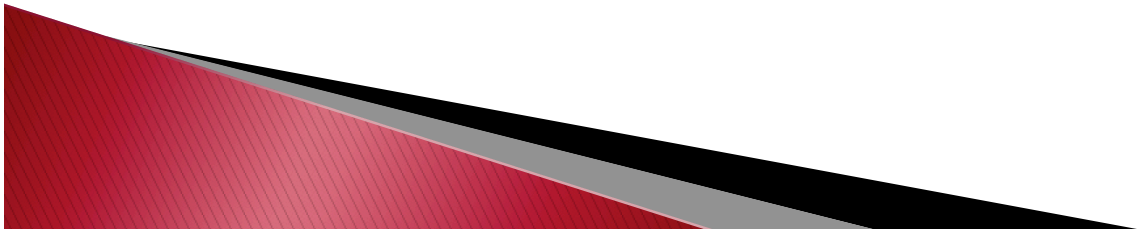
- Objective is to evaluate feasibility
- Approximate the cost and benefit of exploiting memory content similarity
- BENEFIT :
  - number of non-unique pages
- COSTS:
  - storage overhead (e.g., patch data)
  - runtime overhead

HPC Application Suite

ASC Sequoia Marquee Performance Codes	AMG	Parallel algebraic multigrid solver
	IRS	Implicit Radiation Solver; radiation transport
DOE Production Applications	CTH	Multi-material, large deformation, strong shock wave, solid mechanics code
	LAMMPS	Molecular dynamics simulator
Mantevo Mini-Applications	HPCCG	Mimics finite element generation, assembly and solution for an unstructured grid problem
	phdMesh	Mimics the contact search applications in an explicit finite element application
Miscellaneous Applications	SAMRAI	Enables application of structured adaptive mesh refinement to large-scale multi-physics problems
	Sweep3D	Neutron transport problem

Data Collection

- Collected memory snapshots using `libmemstate`, a library built on the MPI Profiling layer and linked against our target applications
- Periodically, it wakes up and, based on the contents of `/proc/self/maps`, copies the application's memory to persistent storage
- Analysis is performed off-line



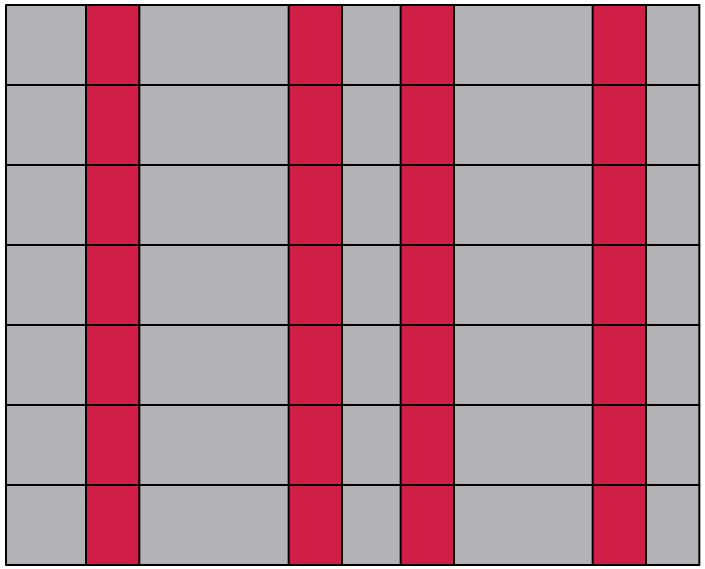
Data Collection (cont'd)

- Naively, identifying similar and duplicate pages is a $O(n^2)$ operation
- Hashing can reduce the cost of identifying duplicate pages
- Similar pages are trickier; we borrow from the Difference Engine

Identifying Similarity (cont'd)

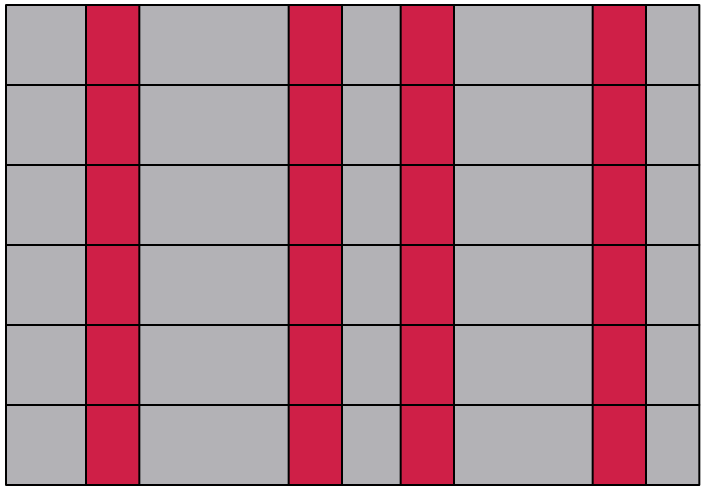
Suppose the memory of an application consists of these seven pages

Identifying Similarity (cont'd)



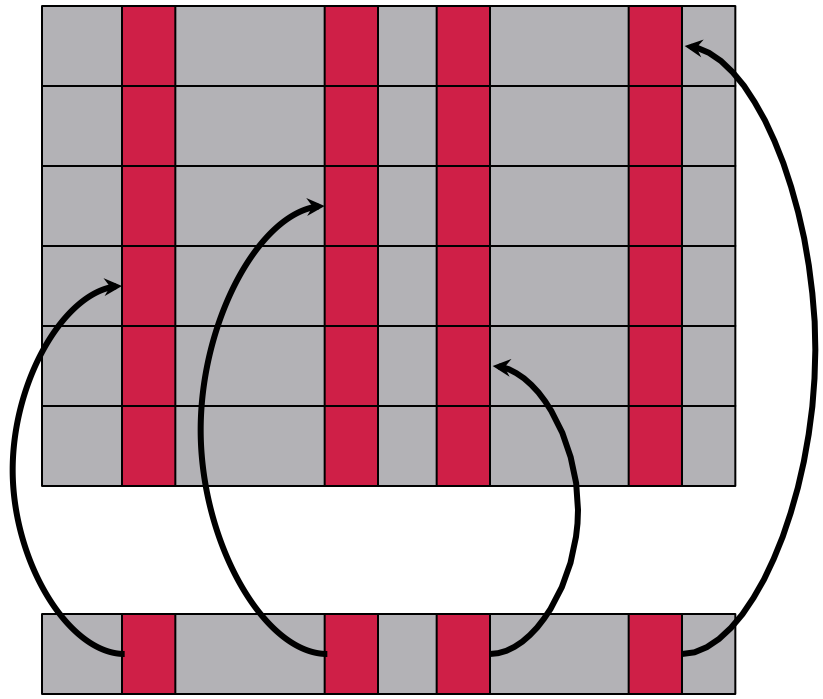
During initialization, we choose four random offsets. The 128 byte blocks at each offset is a "signature" of the page

Identifying Similarity (cont'd)



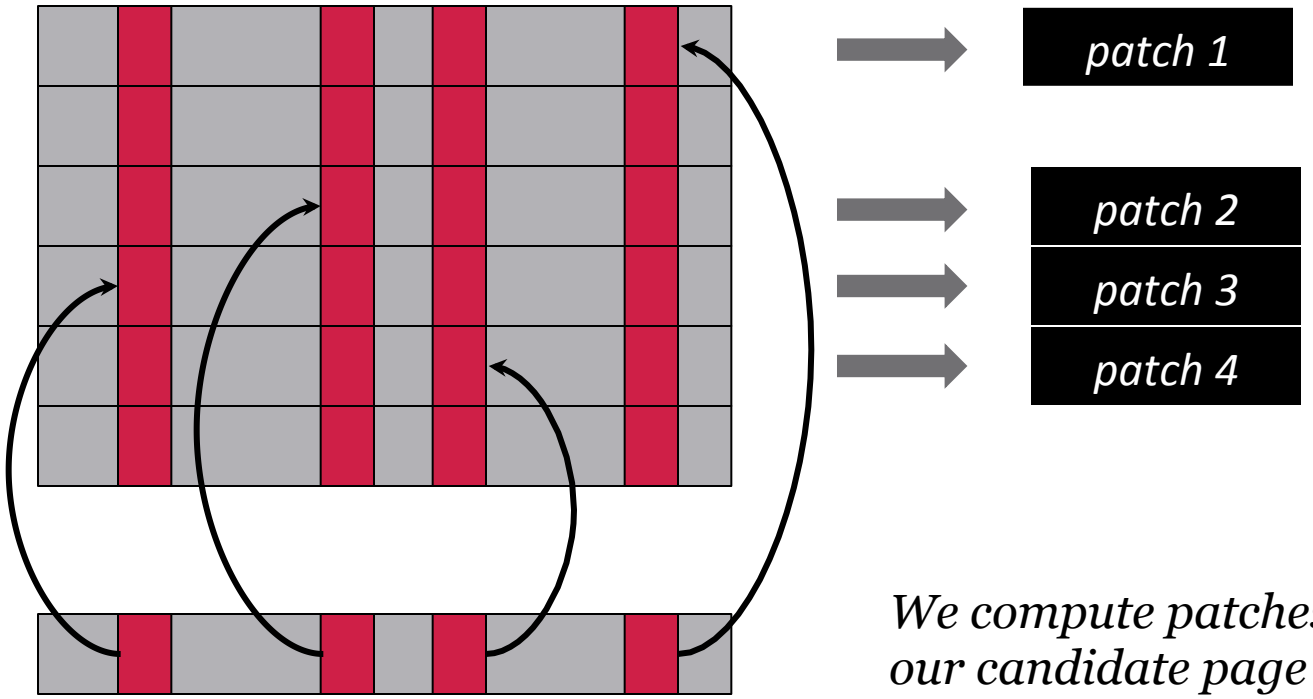
Let's suppose we are trying to determine whether this page is similar to any other page in the application's memory similar

Identifying Similarity (cont'd)



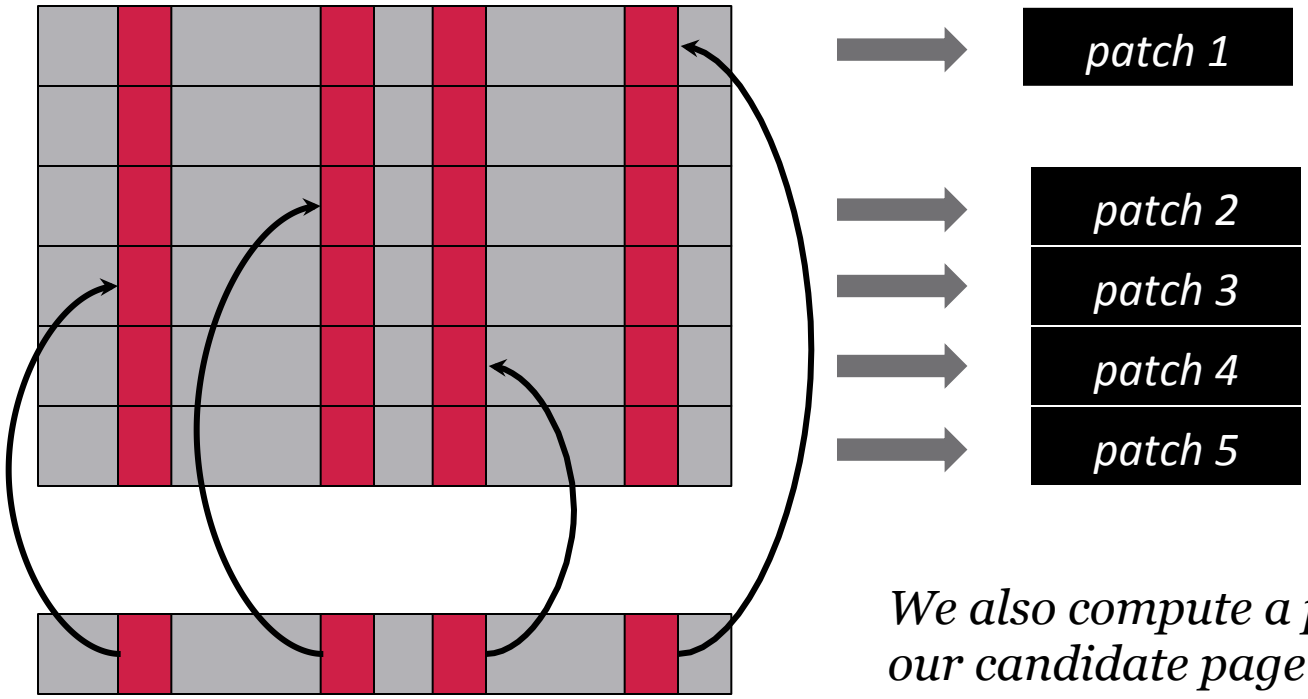
Using hash values, we find up to four pages that match one or more of the candidate page's signatures

Identifying Similarity (cont'd)



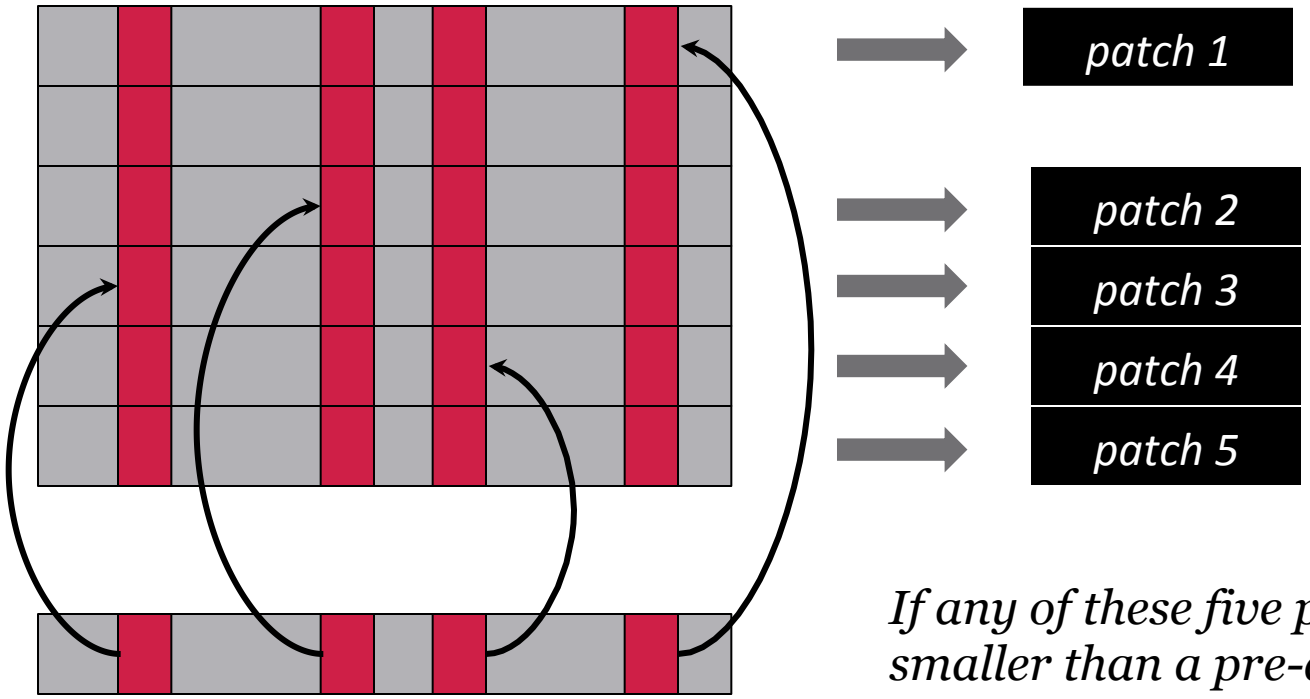
We compute patches between our candidate page and each of the four pages identified by signature

Identifying Similarity (cont'd)



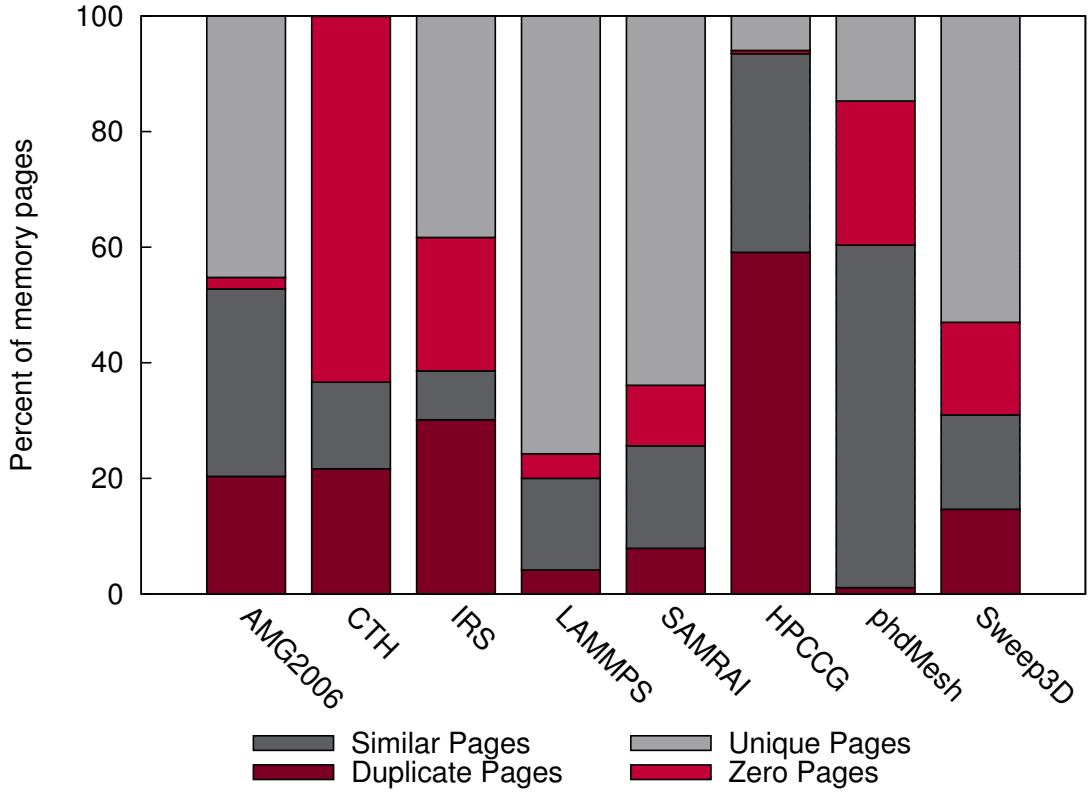
We also compute a patch between our candidate page and the page at the next lowest virtual address

Identifying Similarity (cont'd)



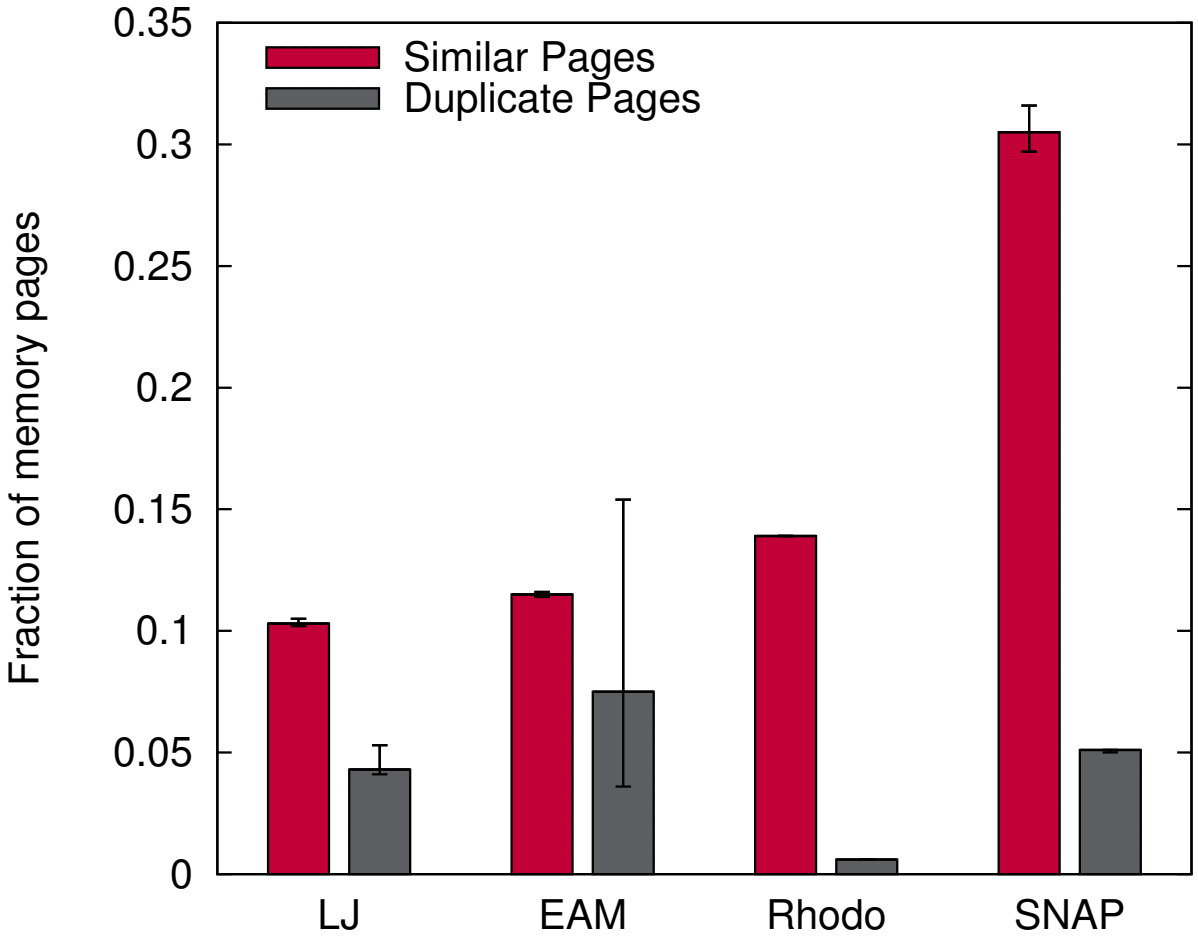
If any of these five patches are smaller than a pre-determined threshold (e.g., 1024 bytes) we classify our candidate page as similar

Prevalence of Similarity

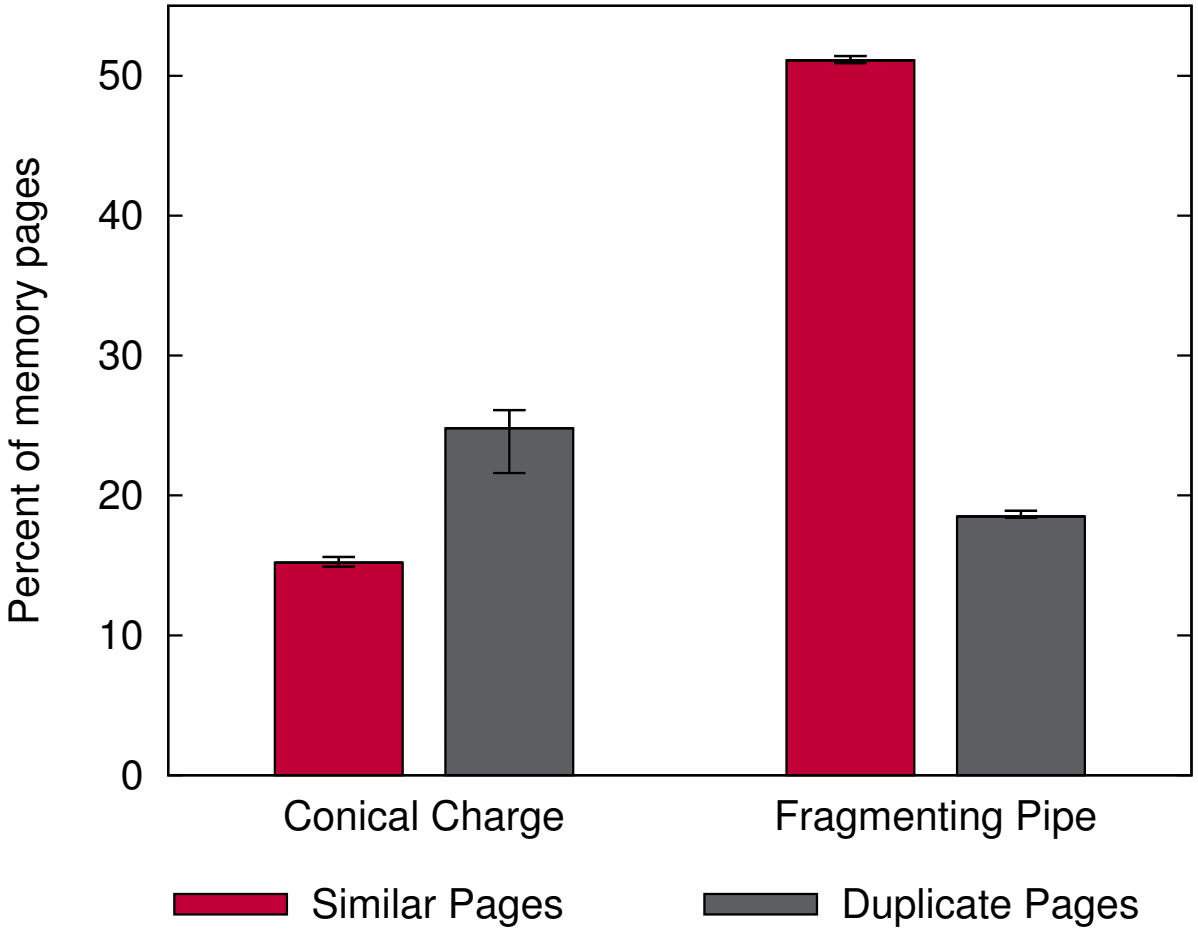


- For 5 out of 8 applications less than half of their memory is composed of unique pages

Inputs matter (LAMMPS)



Inputs matter (CTH)

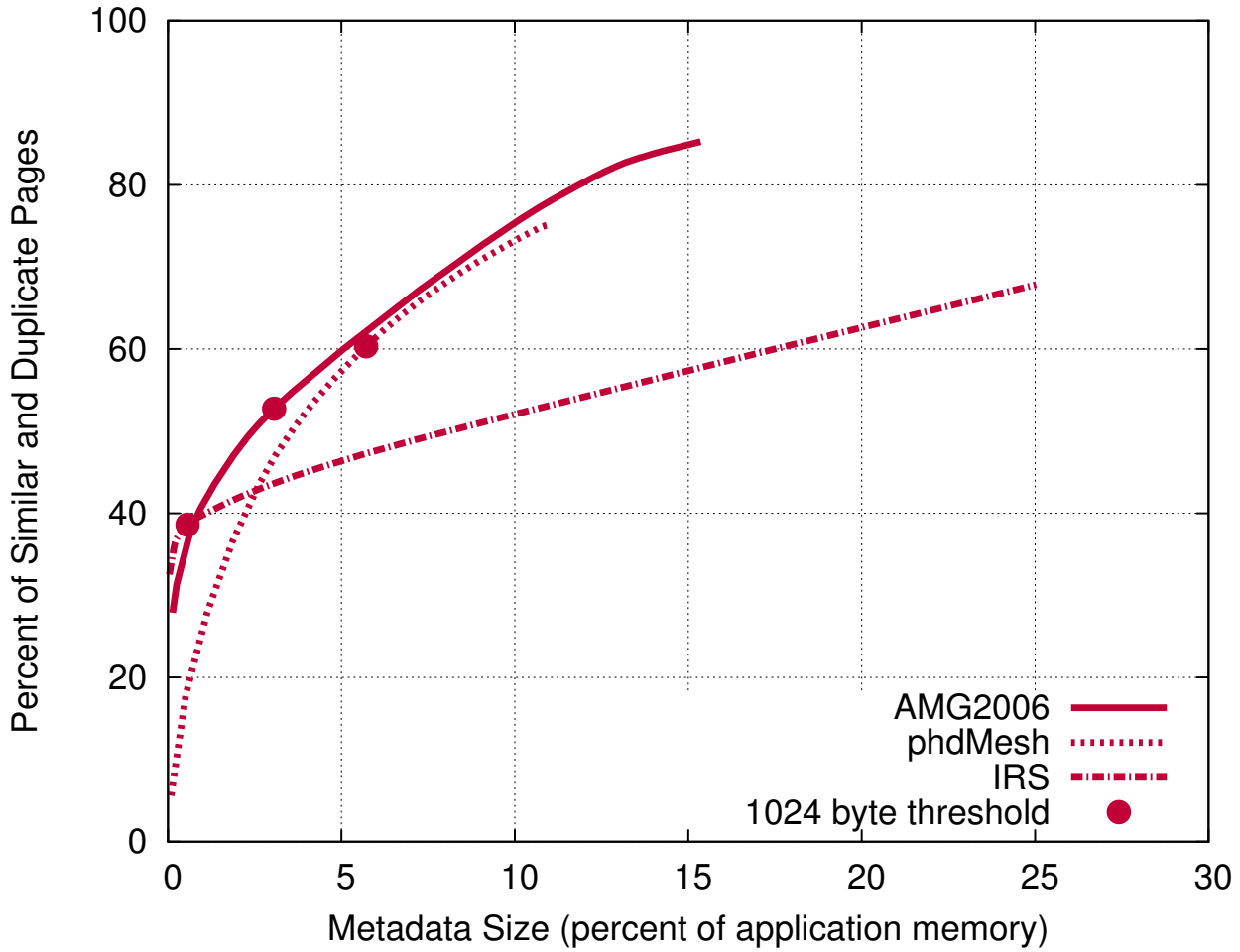


Metadata Overhead

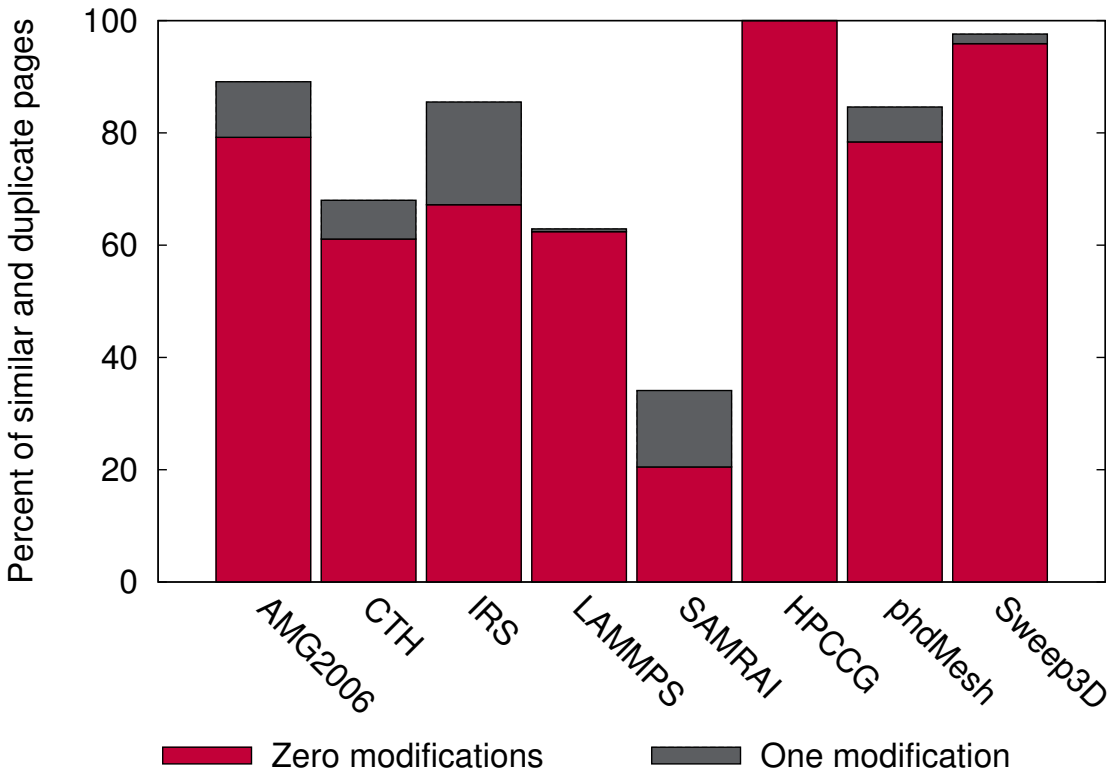
Application	Metadata Size (as % of application memory)
AMG2006	3.99 %
CTH	0.31 %
IRS	0.57 %
LAMMPS	1.34 %
SAMRAI	1.36 %
HPCCG	1.15 %
phdMesh	5.81 %
Sweep3D	0.46 %

- The patch data for 6 out of 8 applications occupies less than 1.5% of application memory

Metadata Overhead (cont'd)



Modification Behavior



- In the memory of 5 out of 8 applications, more than 84% of the duplicate & similar pages change either once or not at all

Conclusion

- The results are promising
- For many applications, we can potentially protect a significant fraction of the memory footprint
- Based on our initial examination, the overhead of storing and maintaining metadata appear reasonable
- More detail in: *An Examination of Content Similarity within the Memory of HPC Applications*, SAND2013-005

Questions?

slevy@cs.unm.edu