PRESAGE : Selective Low-Overhead Error Amplification for Easy Detection

Vishal Chandra Sharma, Arnab Das, Ian Briggs, Ganesh Gopalakrishnan
School of Computing, University of Utah, Salt Lake City, UT
Sriram Krishnamoorthy
Pacific Northwest National Laboratory, Richland, WA

Detection versus Amplification

- Transient errors are likely to affect long-running applications
- Detecting these errors with precision is basic to error-recovery
- Unfortunately,
  - Precise detectors come with high overheads
  - False positive rates reflect heavily on the introduced overhead especially when positive rates and soft error rates are comparable
  - Tuning for low false positive rates usually increases omissions
    - No net gain (unreliability persists, accompanied by slowdown)

- IDEA: Amplify errors to magnify their footprint

How to achieve:
- Focus on critical sub-computations amenable to "error chaining"
- Propagate any errors in to all similar future computations
- Net gain:
  - Error detection becomes cheaper (e.g. address invariants)
  - The common case (error-free computation) runs faster

Our Contribution

- Demonstrate that we can meaningfully amplify errors
- Amplification is robust under compiler transformations
- Process can take advantage of certain ISAs

Error amplification , when selectively applied to well-matched problems, may lead to sound detector based solutions.

We develop and release the tool PRESAGE based on LLVM

Recent Results

Deeper understanding of how PRESAGE fares under
* Portable compiler technologies (e.g. LLVM)
* More extensive measurements to confirm efficacy
* Measurement of efficacy under two ISAs
  - Intel and ARM
* Better measurement of
  - crashes versus normal-looking finishes
  - SDC detection efficacy

How ISA May Help

Serendipitous advantage for some ISAs.
- The ARM ISA has the ability to not only calculate relative addresses, but preserve it to help with successive relative address calculations.
- Observing this, we modified the ARM code generator to take advantage of this ISA feature and beat down the PRESAGE overhead to virtually zero in many cases.

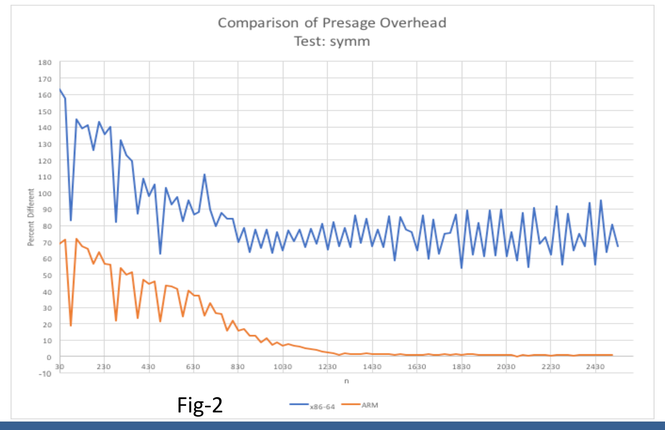


Fig-2

Validation with Transformation

High level code transformations (e.g. Polyhedral) are essential to enhance the overall efficiency HPC programs

- We studied whether such transformations nullify advantages
- Applied polyhedral transformations (using PLUTO) to HPC applications

- **Verified that the advantages are preserved under transformations**

- Crash percentage tend to improve under them

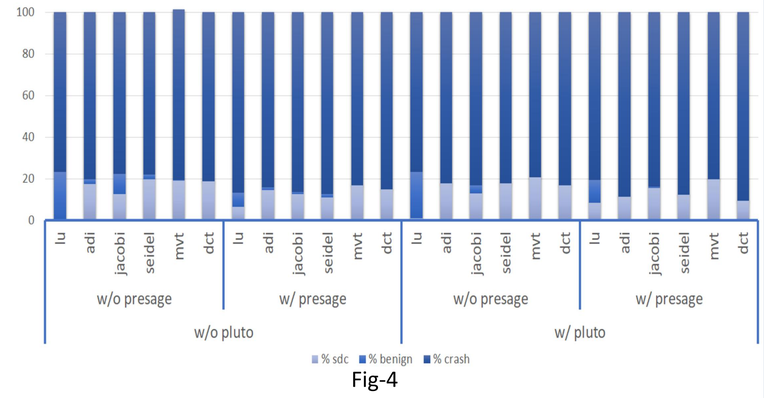


Fig-4

Bar Graph: Application-studies with and without polyhedral transforms as well as with and without PRESAGE transforms.

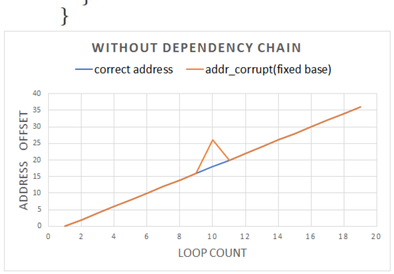
- In both cases, higher number of crashes are observed with PRESAGE transformations (confirming error amplification)

Focus

Structured Address Generation :
Address(A[i]) = BaseAddress(A) + size*offset
Relativized Address Generation :
Address(A[i]) = Address(A[i-j]) + j*size

```
L0: void foo1(double* a, unsigned n){
L1:   double* addr=a;
L2:   for(int i=1;i<n;i++){
L3:     int id=2*i-2;
L4:     addr=&a[id];
L5:     *addr=i;
      }
    }
```

```
L0: void foo2(double *a, int n){
L1:   double* addr=a;
L2:   int pid=0;
L3:   for(int i=1;i<n;i++){
L4:     int id=2*i-2;
L5:     int rid=id-pid;
L6:     addr[rid]=i;
L7:     pid=id;
L8:     addr=&addr[rid];
      }
    }
```



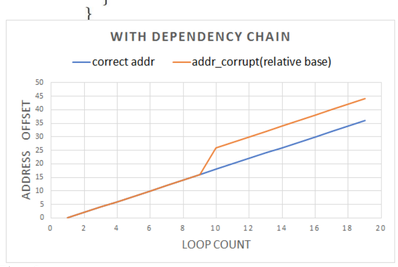


Fig-1

Snowballs the transient error to all future addresses.
Increased chance of errors leading to visible program crashes

PRESAGE Efficacy

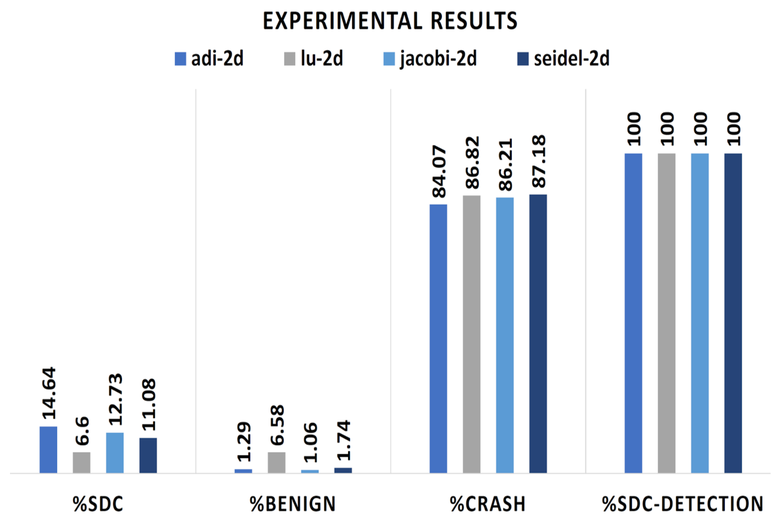


Fig-3

LLVM Transformation

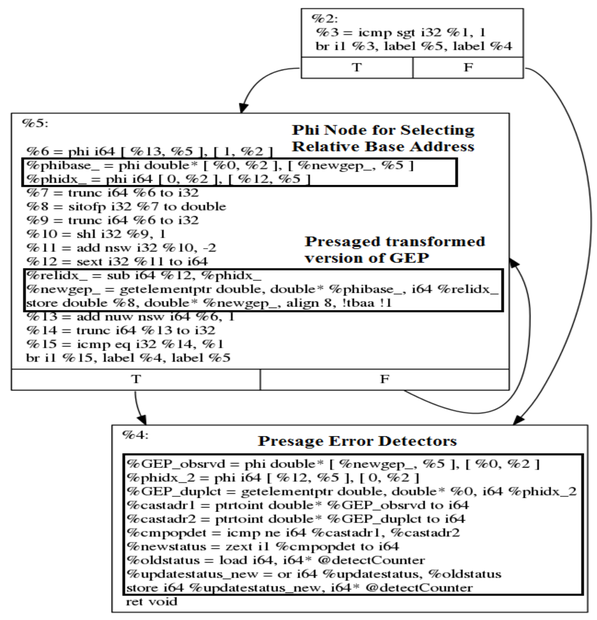


Fig-5

Benefits of the PRESAGE Approach

- 100% Silent Data Corruption detection FOR the chosen aspect to protect
  - i.e. structured address computation
- This advantage is preserved even under polyhedral transformation.
  - This makes the methodology work with important memory optimizations

- Benchmarks like adi, seidel have negligible overheads in range of 0.3% to 3.2%.
- Benchmarks like jacobi-2d lead to worst case overheads ~40% (high register pressure and increased inter-block dependency chains)
- Verified that some ISA can help minimize PRESAGE overheads by exploiting special instructions that allow easy reuse of relativized addresses
- Developed using LLVM compiler infrastructure for portability across compilers/ISA

Limitations, and Future Work

PRESAGE efficiently amplifies transient errors during structured array generation to visible program crashes
* At present it covers only structured address generation
* Such focus was a significant enabler
* Remains to be seen whether other aspects of system resilience are amenable to similar error-amplification (future work)

Currently supports one dimensional array structures.
* Future work includes covering more complex array/struct usages

We now better understand variance in SDC and benign errors across different kernels.
ISA-specific benefits are variable
* We are working on identifying loss of efficiency and mitigating the same

Acknowledgement

This work was supported in part by the U.S. Department of Energy's (DOE) Office of Science, Office of Advanced Scientific Computing Research, under award 66905. Pacific Northwest National Laboratory is operated by Battelle for DOE under Contract DE-AC05-76RL01830.
The Utah authors were supported in part under the same DOE project with award number 55800790.

References

1. Vishal Chandra Sharma, Ganesh Gopalakrishnan, Sriram Krishnamoorthy, "PRESAGE:Protecting Structred Address Generation against Soft Errors" in Proceedings – 23rd IEEE International Conference on High Performance Computing, HiPC 2016
2. U. Bondhugula, J. Ramanujam, and P. Sadayappan, 2007. Pluto: A Practical and Fully Automatic Polyhedral Parallelizer and Locality Optimizer. Technical Report OSU-CISRC-10/07- TR70. The Ohio State University.
3. Sheng Di and Frank Capello, 2015. Adaptive Impact Driven Detection of Silent Data Corruption for HPC Applications. IEEE Transactions on Parallel and Distributed Systems(12/2015 2015).
4. Sanket Tavarageri, Sriram Krishnamoorthy, and P. Sadayappan. 2014. Compiler-assisted detection of transient memory errors. In ACM SIGPLAN Conference on Programming Language Design and Implementation , PLDI'14, Edinburgh, United Kingdom – June 09-11, 2014