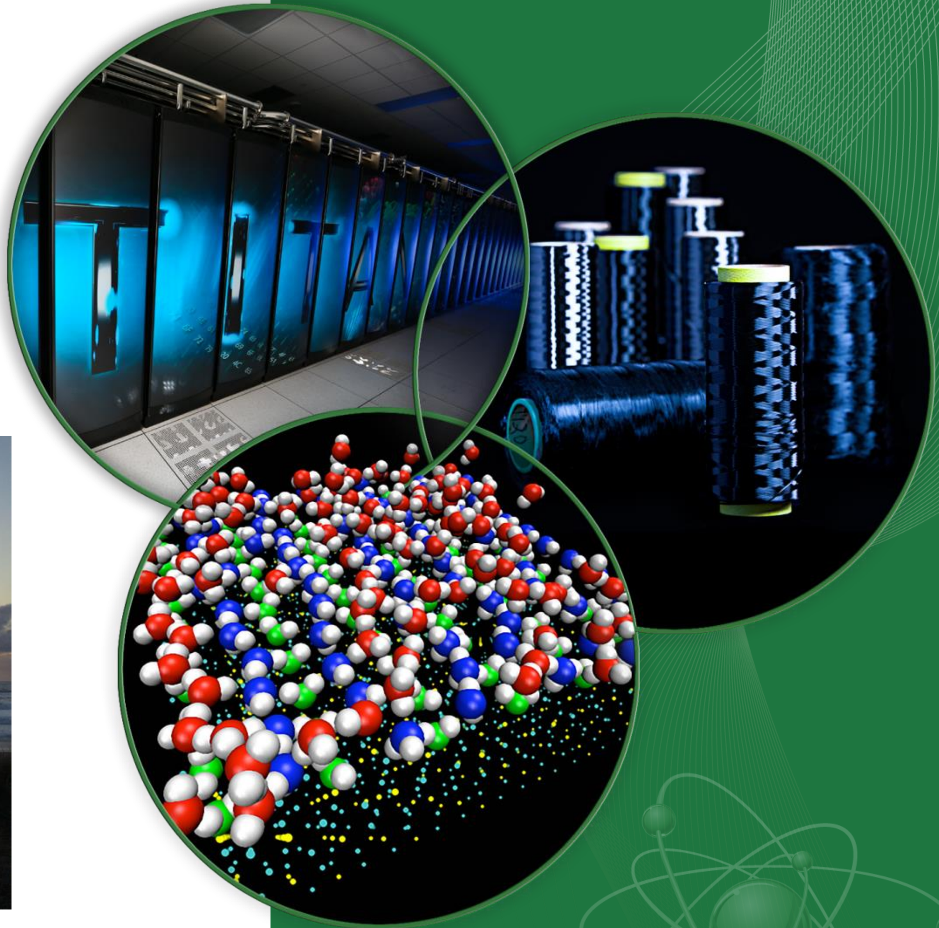
Quantitatively Modeling Application Resilience with the Data Vulnerability Factor

Jeffrey S. Vetter

Li Yu, Sparsh Mittal, Dong Li,
Jeremy Meredith

Presented to
2015 Salishan Conference on High-Speed Computing
Gleneden Beach, Oregon

30 Apr 2015



OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE DEPARTMENT OF ENERGY

Georgia Tech College of Computing
Computational Science and Engineering

ORNL is managed by UT-Battelle
for the US Department of Energy

http://ft.ornl.gov ♦ vetter@computer.org

OAK RIDGE National Laboratory

Overview

- We need methodologies and tools to balance the competing demands of resiliency, power, performance, cost, etc.
  - Application scientists need tools to manage limited resources
    - End-to-End design for application resilience
    - ABFT, C/R, etc
  - Architects need tools to design next generation systems
    - How many application data structures need double chipkill memory protection? At what cost?
- We propose a new metric: the data vulnerability factor (DVF)
  - Prototyped DVF using Aspen performance modeling language
  - Must classify memory access patterns
  - Demonstrate use of DVF on several algorithms
- Initial results appear promising but more work remains

GPU Users want/demand no-ECC!

An Investigation of the Effects of Error Correcting Code on GPU-accelerated Molecular Dynamics Simulations

Ross C. Walker
San Diego Supercomputer Center
Department of Chemistry and Biochemistry
UC San Diego
La Jolla, CA 92093
ross@rosswalker.co.uk

Robin M. Betz
San Diego Supercomputer Center
La Jolla, CA 92093
rbetz@ucsd.edu

ABSTRACT

Molecular dynamics (MD) simulations rely on the accurate evaluation and integration of Newton's equations of motion to propagate the positions of atoms in proteins during a simulation. As such, one can expect them to be sensitive to any form of numerical error that may occur during a simulation. Increasingly graphics processing units (GPUs) are

Keywords

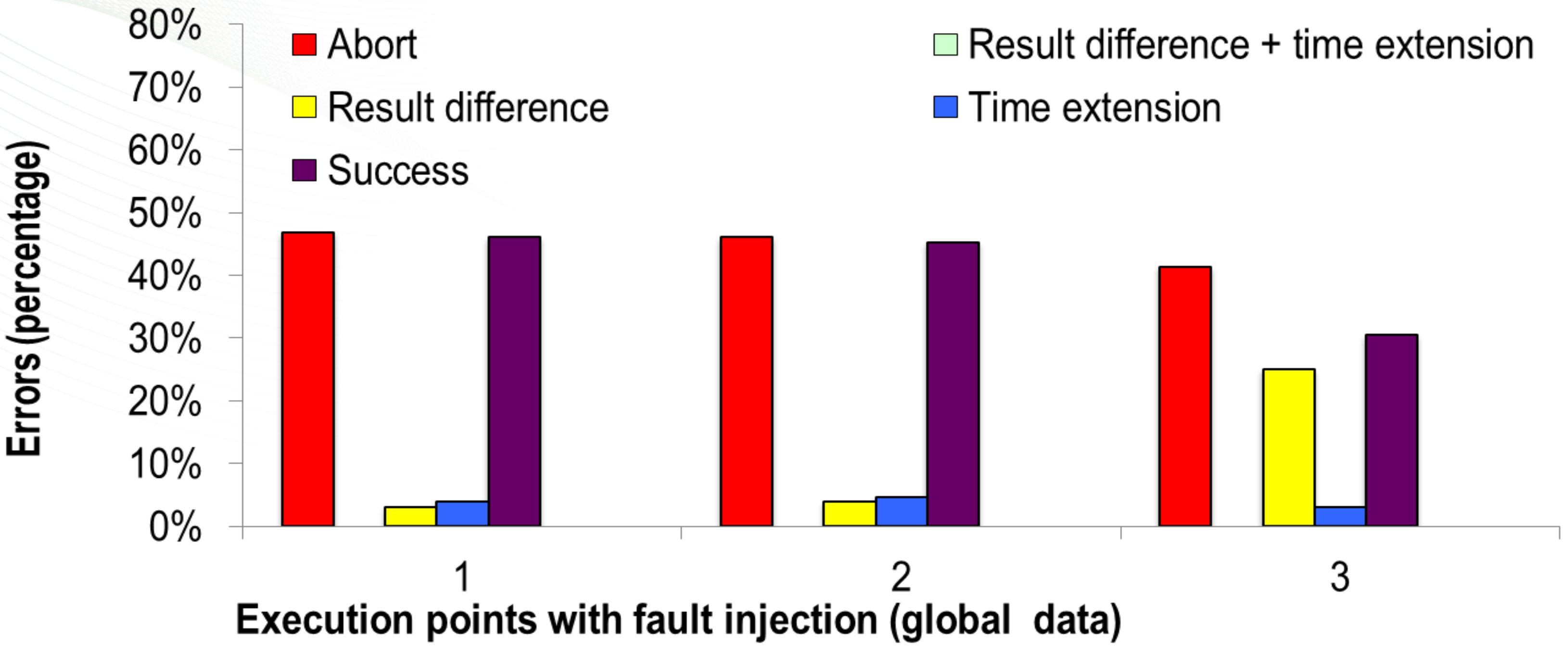
XSEDE 2013, GPU-acceleration, ECC error

1. INTRODUCTION

The field of computational sciences uses the power of modern computers to gain insight into scientific systems. Re-

R.C. Walker and R.M. Betz, "An investigation of the effects of error correcting code on GPU-accelerated molecular dynamics simulations," Proc. Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery, 2013, pp. 8,

BIFIT Results (S3D)

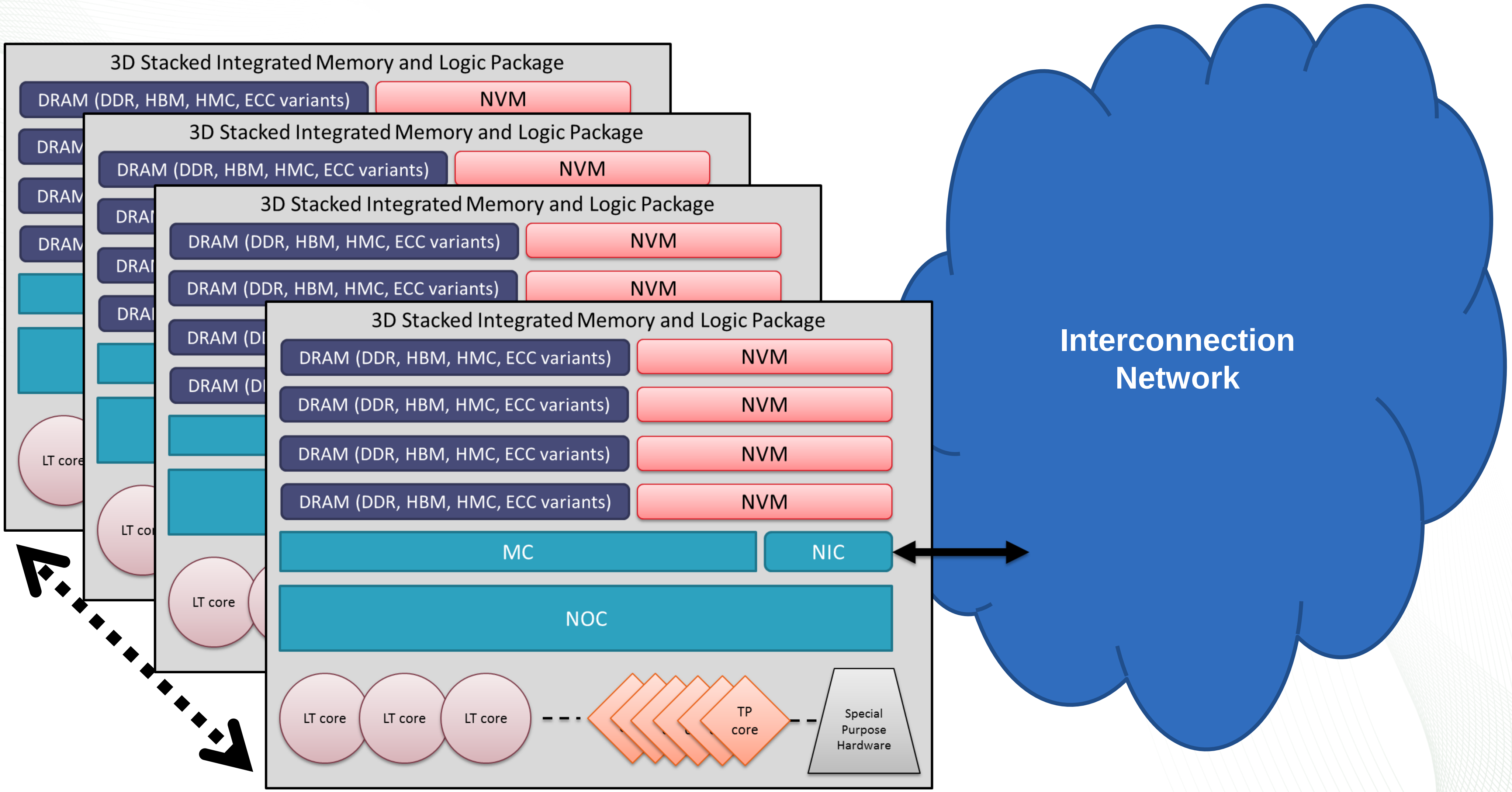


- 1000s of executions to cover statistically significant sample
- Limited capability to change architecture, algorithm

Observation: the global data objects with fault injected are responsible for most of the abort errors throughout the application execution

D. Li, J.S. Vetter, and W. Yu, "Classifying Soft Error Vulnerabilities in Extreme-Scale Scientific Applications Using a Binary Instrumentation Tool," in *SC12: ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis. Salt Lake City, 2012*

Notional Future Architecture



Current status

- Applications scientists can (need to) provide valuable input about resiliency requirements
  - Application usage scenarios: ensembles, MC
  - Employ ABFT, C/R, etc

- Multimode memory systems will be the norm in coming years
  - ECC (none, double chipkill), Persistence, Performance

- Current methods (i.e., fault-injection) can be useful but are often too expensive and inflexible

A new methodology:

Data Vulnerability Metric

Data Vulnerability Factor: Why a new metric and methodology?

- Analytical model of resiliency that includes important features of architecture and application
  - Fast
  - Flexible
- Balance multiple design dimensions
  - Application requirements
  - Architecture (memory capacity and type)
- Focus on main memory initially
- Prioritize vulnerabilities of application data

L. Yu, D. Li *et al.*, *"Quantitatively modeling application resilience with the data vulnerability factor (Best Student Paper Finalist)," in SC14: International Conference for High Performance Computing, Networking, Storage and Analysis. New Orleans, Louisiana: IEEE Press, 2014, pp. 695-706, 10.1109/sc.2014.62.*

DVF Defined

Data Structure Vulnerability $\rightarrow$ $DVF_d = \boxed{N_{error}} * \boxed{N_{ha}}$

Application Vulnerability $\rightarrow$ $DVF_a = \sum_{i=1}^{n} DVF_{d_i}$

Larger DVF indicates higher vulnerability, and vice versa

$N_{error} = FIT * T * S_d$

We focus on a specific hardware component, the main memory, in this work

$N_{ha} \leftarrow$ Hardware Access Pattern

Implementing DVF

- Extend Aspen performance modeling language
- Specify memory access patterns
- Combine error rates with memory regions and performance
- Assign DVF to each application memory region, Sum for application

Brief Introduction to Aspen

Prediction Techniques Ranked

	Speed	Ease	Flexibility	Accuracy	Scalability
Ad-hoc Analytical Models	1	3	2	4	1
Structured Analytical Models	1	2	1	4	1
Aspen	1	1	1	4	1
Simulation – Functional	3	2	2	3	3
Simulation – Cycle Accurate	4	2	2	2	4
Hardware Emulation (FPGA)	3	3	3	2	3
Similar hardware measurement	2	1	4	2	2
Node Prototype	2	1	4	1	4
Prototype at Scale	2	1	4	1	2
Final System	-	-	-	-	-

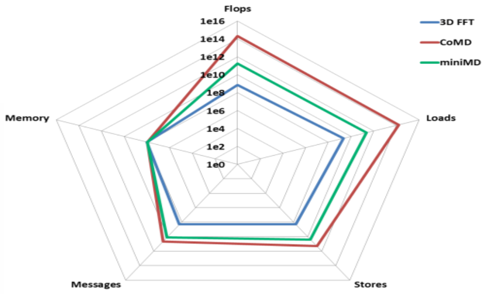
Aspen Design Flow

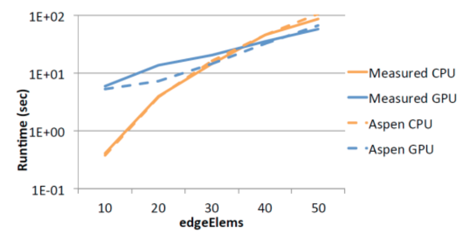
Source code

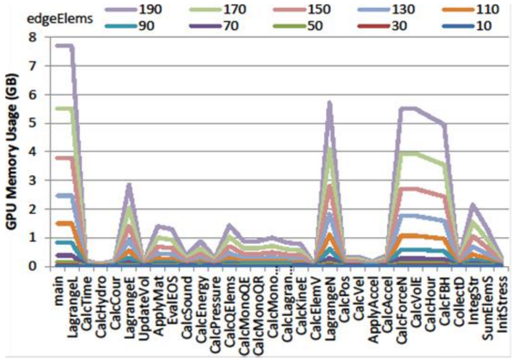
```
static inline
void CalcMonotonicQGradientsForElems(Index_t p_nodelist[T_NUMELEM8],
        Real_t p_x[T_NUMNODE], Real_t p_y[T_NUMNODE], Real_t p_z[T_NUMNODE],
        Real_t p_xd[T_NUMNODE], Real_t p_yd[T_NUMNODE],Real_t p_zd[T_NUMNODE],
        Real_t p_volo[T_NUMELEM], Real_t p_vnew[T_NUMELEM],
        Real_t p_delx_zeta[T_NUMELEM], Real_t p_delv_zeta[T_NUMELEM],
        Real_t p_delx_xi[T_NUMELEM], Real_t p_delv_xi[T_NUMELEM],
        Real_t p_delx_eta[T_NUMELEM], Real_t p_delv_eta[T_NUMELEM])
{
   Index_t i;
   Index_t numElem = m_numElem;
#pragma acc parallel loop independent present(p_vnew, p_nodelist, p_x, p_y, p_z, p_xd, \
p_yd, p_zd, p_volo, p_delx_xi, p_delx_eta, p_delx_zeta, p_delv_xi, p_delv_eta,\
p_delv_zeta)
   for (i = 0 ; i < numElem ; ++i ) {
      const Real_t ptiny = 1.e-36 ;
      Real_t ax,ay,az ;
      Real_t dxv,dyv,dzv ;

      const Index_t *elemToNode = &p_nodelist[8*i];
      Index_t n0 = elemToNode[0] ;
      Index_t n1 = elemToNode[1] ;
      Index_t n2 = elemToNode[2] ;
      Index_t n3 = elemToNode[3] ;
      Index_t n4 = elemToNode[4] ;
      Index_t n5 = elemToNode[5] ;
      Index_t n6 = elemToNode[6] ;
      Index_t n7 = elemToNode[7] ;

      Real_t x0 = p_x[n0] ;
```

Aspen code

```
kernel CalcMonotonicQGradients {
  execute [numElems]
  {
    loads [8 * indexWordSize] from nodelist
    // Load and cache position and velocity.
    loads/caching [8 * wordSize] from x
    loads/caching [8 * wordSize] from y
    loads/caching [8 * wordSize] from z

    loads/caching [8 * wordSize] from xvel
    loads/caching [8 * wordSize] from yvel
    loads/caching [8 * wordSize] from zvel

    loads [wordSize] from volo
    loads [wordSize] from vnew
    // dx, dy, etc.
    flops [90] as dp, simd
    // delvk delxk
    flops [9 + 8 + 3 + 30 + 5] as dp, simd
    stores [wordSize] to delv_xeta
    // delxi delvi
    flops [9 + 8 + 3 + 30 + 5] as dp, simd
    stores [wordSize] to delx_xi
    // delxj and delvj
    flops [9 + 8 + 3 + 30 + 5] as dp, simd
    stores [wordSize] to delv_eta
  }
}
```







Creation

- Manual for future applications
- Static analysis via compilers
- Historical
- Empirical

Representation in Aspen
- **Modular**
- **Sharable**
- **Composable**
- **Reflects prog structure**

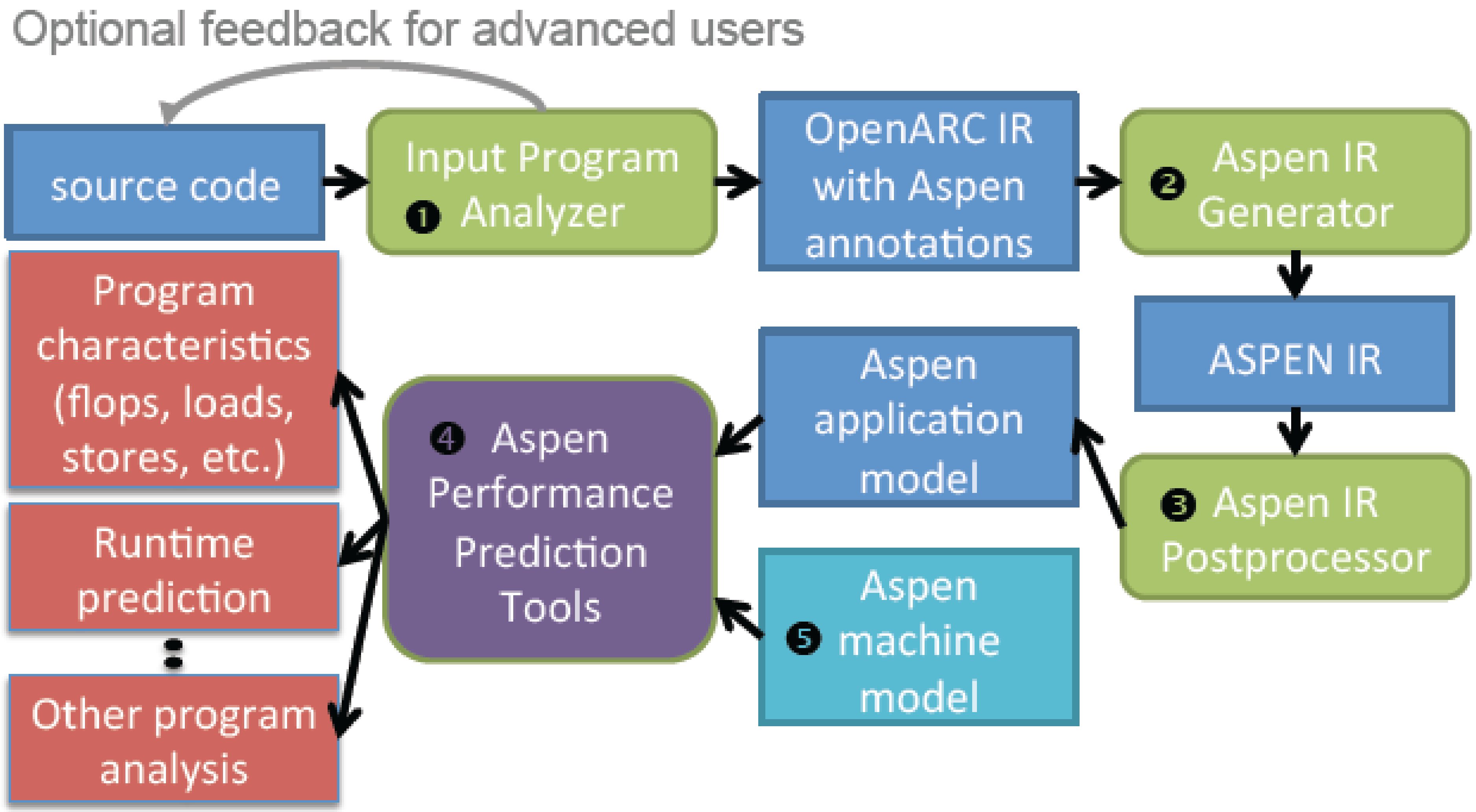
Existing models for MD, UHPC CP 1, Lulesh, 3D FFT, CoMD, VPFFT, …

Use

- Interactive tools for graphs, queries
- Design space optimization
- Drive simulators
- Feedback to runtime systems

K. Spafford and J.S. Vetter, "Aspen: A Domain Specific Language for Performance Modeling," in *SC12: ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis, 2012*

Creating Aspen Models



S. Lee, J.S. Meredith, and J.S. Vetter, “COMPASS: A Framework for Automated Performance Modeling and Prediction,” in *ACM International Conference on Supercomputing (ICS). Newport Beach, California: ACM, 2015, 10.1145/2751205.2751220.*

Simple MM example generated from COMPASS

Original Source

```
int N = 1024;
void matmul(float *a, float *b, float *c){ int i, j, k ;
#pragma acc kernels loop gang copyout(a[0:(N*N)]) \
copyin(b[0:(N*N)],c[0:(N*N)])
  for (i=0; i<N; i++){
#pragma acc loop worker
    for (j=0; j<N; j++) { float sum = 0.0 ;
      for (k=0; k<N; k++) {sum+=b[i*N+k]*c[k*N+j];}
      a[i*N+j] = sum; }
  } //end of i loop
} //end of matmul()
int main() {
  int i; float *A = (float*) malloc(N*N*sizeof(float));
  float *B = (float*) malloc(N*N*sizeof(float));
  float *C = (float*) malloc(N*N*sizeof(float));
  for (i = 0; i <  N*N; i++)
  {  A[i] = 0.0F; B[i] = (float) i; C[i] = 1.0F; }
#pragma aspen modelregion label(MM)
  matmul(A,B,C);
  free(A); free(B); free(C); return 0;
} //end of main()
```

Compiler-generated Aspen

```
model MM {
  param floatS = 4; param N = 1024
  data A as Array((N*N), floatS)
  data B as Array((N*N), floatS)
  data C as Array((N*N), floatS)
  kernel matmul {
    execute matmul2_intracommIN
    {  intracomm [floatS*(N*N)] to C as copyin
      intracomm [floatS*(N*N)] to B as copyin }
    map matmul2 [N] {
      map matmul3 [N] {
        iterate [N] {
          execute matmul5
          {  loads [floatS] from B as stride(1)
            loads [floatS] from C; flops [2] as sp, simd }
        } //end of iterate
        execute matmul6 { stores [floatS] to A as stride(1) }
      } // end of map matmul3
    } //end of map matmul2
    execute matmul2_intracommOUT
    { intracomm [floatS*(N*N)] to A as copyout }
  } //end of kernel matmul
  kernel main { matmul() }
} //end of model MM
```

OAK RIDGE National Laboratory

LULESH in Aspen

branch: master ▾ aspen / models / lulesh / **lulesh.aspen**

jsmeredith on Sep 20, 2013 adding models

1 contributor

336 lines (288 sloc) | 9.213 kb

Raw | Blame | History

```
//
// lulesh.aspen
//
// An ASPEN application model for the LULESH 1.01 challenge problem.  Based
// on the CUDA version of the source code found at:
//  https://computation.llnl.gov/casc/ShockHydro/
//
param nTimeSteps = 1495

// Information about domain
param edgeElems = 45
param edgeNodes = edgeElems + 1

param numElems = edgeElems^3
param numNodes = edgeNodes^3

// Double precision
param wordSize = 8

// Element data
data mNodeList as Array(numElems, wordSize)
data mMatElemList as Array(numElems, wordSize)
data mNodeList as Array(8 * numElems, wordSize) //  8 nodes per element
data mlxim as Array(numElems, wordSize)
data mlxip as Array(numElems, wordSize)
data mletam as Array(numElems, wordSize)
data mletap as Array(numElems, wordSize)
data mzetam as Array(numElems, wordSize)
data mzetap as Array(numElems, wordSize)
data melemBC as Array(numElems, wordSize)
data mE as Array(numElems, wordSize)
data mP as Array(numElems, wordSize)
```

```
kernel CalcMonotonicQGradients {
  execute [numElems]
  {
    loads [8 * indexWordSize] from nodelist
    // Load and cache position and velocity.
    loads/caching [8 * wordSize] from x
    loads/caching [8 * wordSize] from y
    loads/caching [8 * wordSize] from z

    loads/caching [8 * wordSize] from xvel
    loads/caching [8 * wordSize] from yvel
    loads/caching [8 * wordSize] from zvel

    loads [wordSize] from volo
    loads [wordSize] from vnew
    // dx, dy, etc.
    flops [90] as dp, simd
    // delvk delxk
    flops [9 + 8 + 3 + 30 + 5] as dp, simd
    stores [wordSize] to delv_xeta
    // delxi delvi
    flops [9 + 8 + 3 + 30 + 5] as dp, simd
    stores [wordSize] to delx_xi
    // delxj and delvj
    flops [9 + 8 + 3 + 30 + 5] as dp, simd
    stores [wordSize] to delv_eta
  }
}
```

LULESH – runtime optimizations

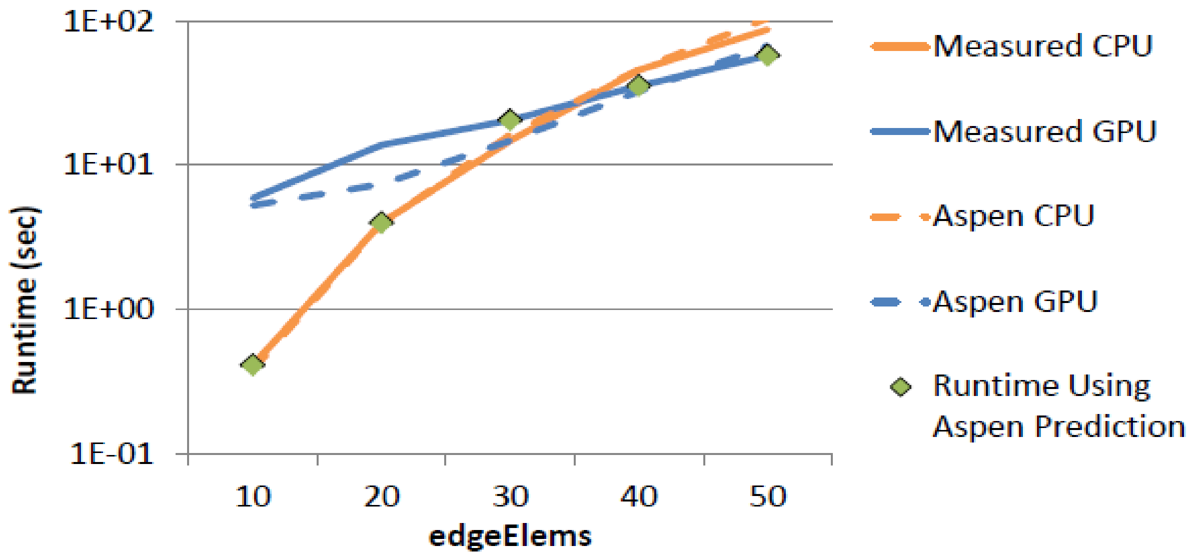


Fig. 7: Measured and predicted runtime of the entire LULESH program on CPU and GPU, including measured runtimes using the automatically predicted optimal target device at each size.

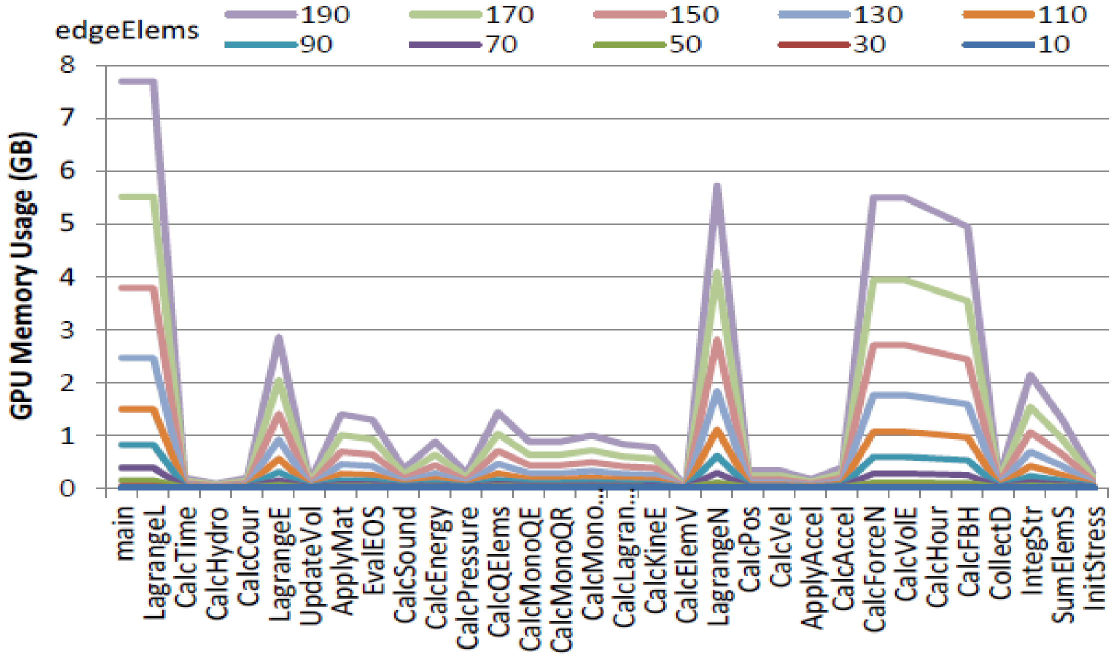
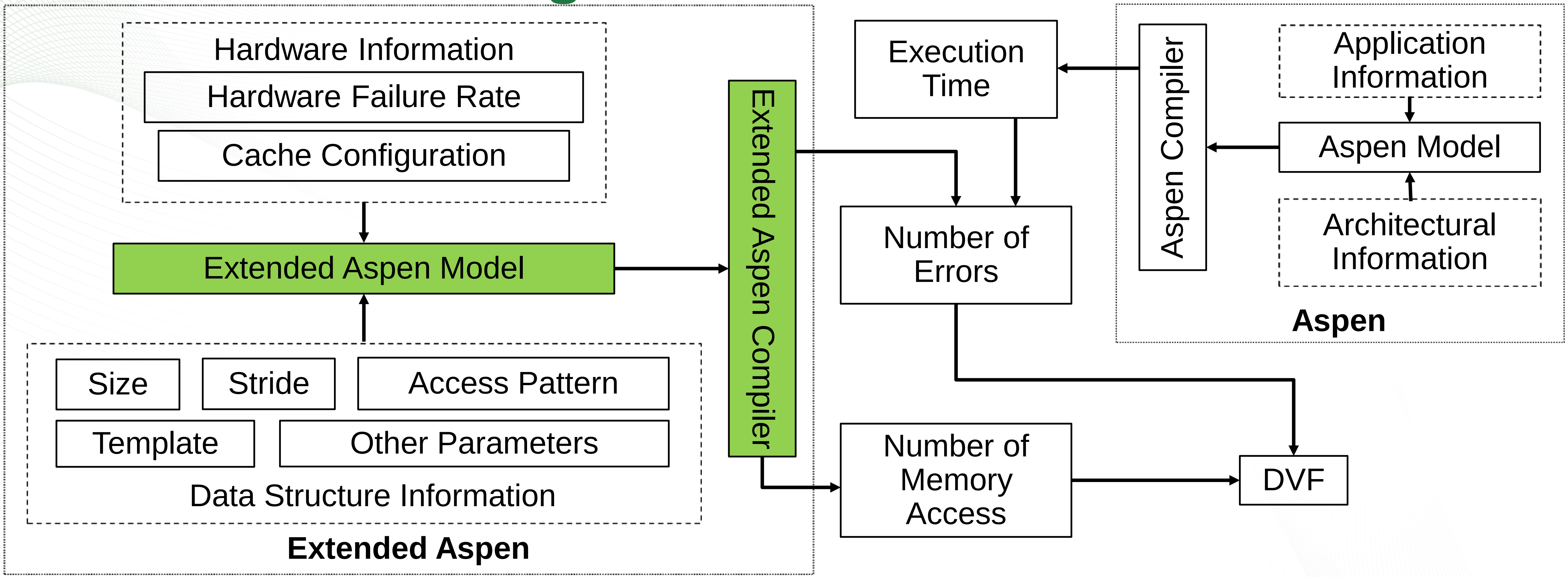


Fig. 8: GPU Memory Usage of each Function in LULESH, where the memory usage of a function is inclusive; value for a parent function includes data accessed by its child functions in the call graph.

OAK RIDGE National Laboratory

Extending Aspen for DVF

Resilience Modeling Workflow



- Aspen Extension
  - Grammar & Syntax for hardware vulnerability and targeted data structures
  - Compiler

Counting Main Memory Accesses

- Challenges
  - We need to consider the caching effects
    - Data in higher levels of memory is ‘protected’
- Goals
  - We must maintain the successful paradigm of Aspen
    - No detailed application source code
    - Very limited architecture information – use simple cache model
    - Fast exploration on various options
  - We have to connect data semantics and memory accesses
- Counting number of memory accesses based on probability analysis

Memory Access Patterns Classification

- Streaming access pattern
  - E.g., vector multiplication
- Random access pattern
  - E.g., N-body simulation and Monte Carlo simulation
- Template-based access pattern
  - E.g., structured multi-grid
- Data reuse pattern
  - E.g., conjugate gradient method

An Example of Aspen Program for DVF

```
procedure VM(A,B,C)
    for i ← 1, 1000 do
        C[i] ← C[i] + A[i*4] * B[i*8]
    end for
end procedure
```

Pseudocode

```
kernel vecmul {
    execute mainblock2 [1]
    {
    flops [2*(n^3)] as sp, fmad, simd
    access {1000} from {matA} as stream(4,16)
    access {4000} from {matB} as stream(4,32)
    access {8000} from {matC} as stream(4,4)
    }
}
```

Extended Aspen Statements

Extended Parser

```
Resilience Statements:
    Footprint Sizes:
        Int: 16,000
    Data Structures:
        Ident: matA
    Access Pattern: Stream
        Int: 4
        Int: 16
Resilience Statements:
    Footprint Sizes:
        Int: 16,000
    Data Structures:
        Ident: matA
    Access Pattern: Stream
        Int: 4
        Int: 16
Resilience Statements:
    Footprint Sizes:
        Int: 16,000
    Data Structures:
        Ident: matA
    Access Pattern: Stream
        Int: 4
        Int: 16
```

Syntax Tree

Extended Complier

Data structure A:
Number of errors: 30,400
Number of memory accesses: 51
DVF: 105504e+06
…

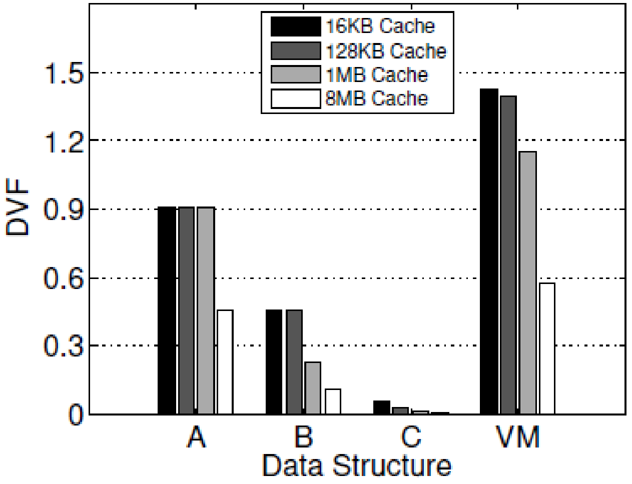
Resilience Modeling Results

Evaluation

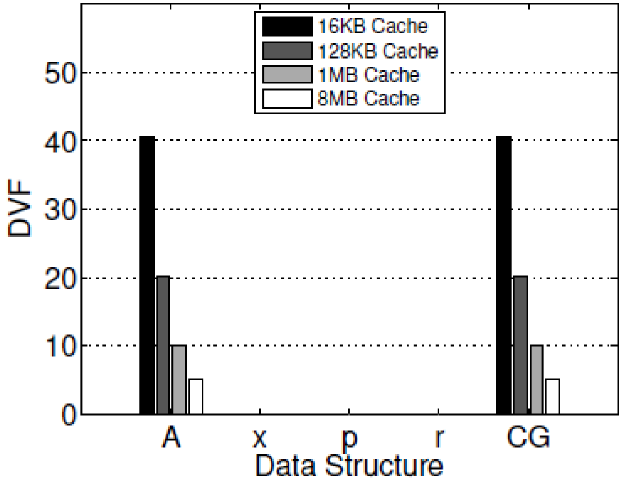
Six Computational Kernels

Algorithm Name	Computational Method Class	Major Data Structures	**Memory Access Patterns**	Example Benchmarks
Vector Multiplication (VM)	Dense linear algebra	A, B, and C	**Streaming**	Homemade code
Conjugate Gradient (CG)	Sparse linear algebra	A, x, p and r	**Template + Reuse + Streaming**	NPB CG
Barnes-Hut simulation (NB)	N-body method	T and P	**Random**	Online code
Multi-grid (MG)	Structured grids	R	**Template-based**	NPB MG
1D FFT (FT)	Spectral methods	A	**Template-based**	NPB FT
Monte Carlo simulation (MC)	Monte Carlo	G and E	**Random**	XSBench

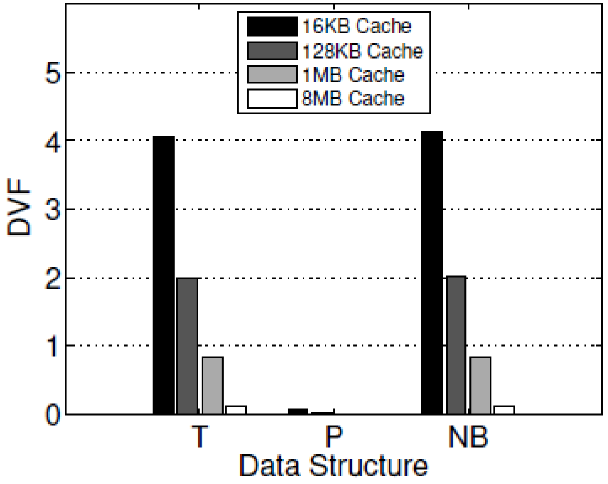
DVF Results



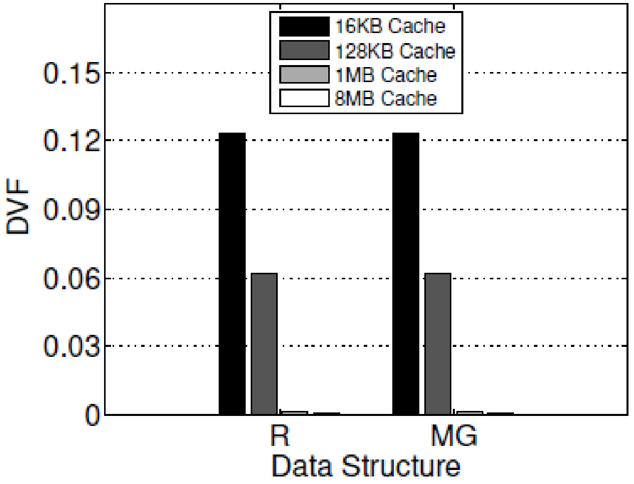
(a) Vector Multiplication



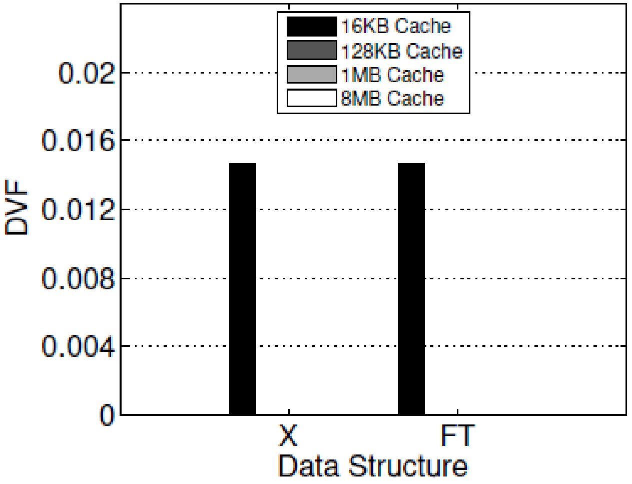
(b) Conjugate Gradient



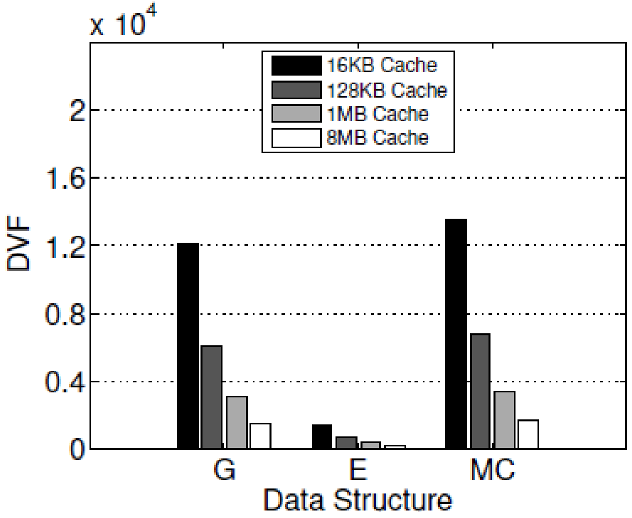
(c) Nbody (Barnes-hut)



(d) Multi-grid



(e) 1D FFT

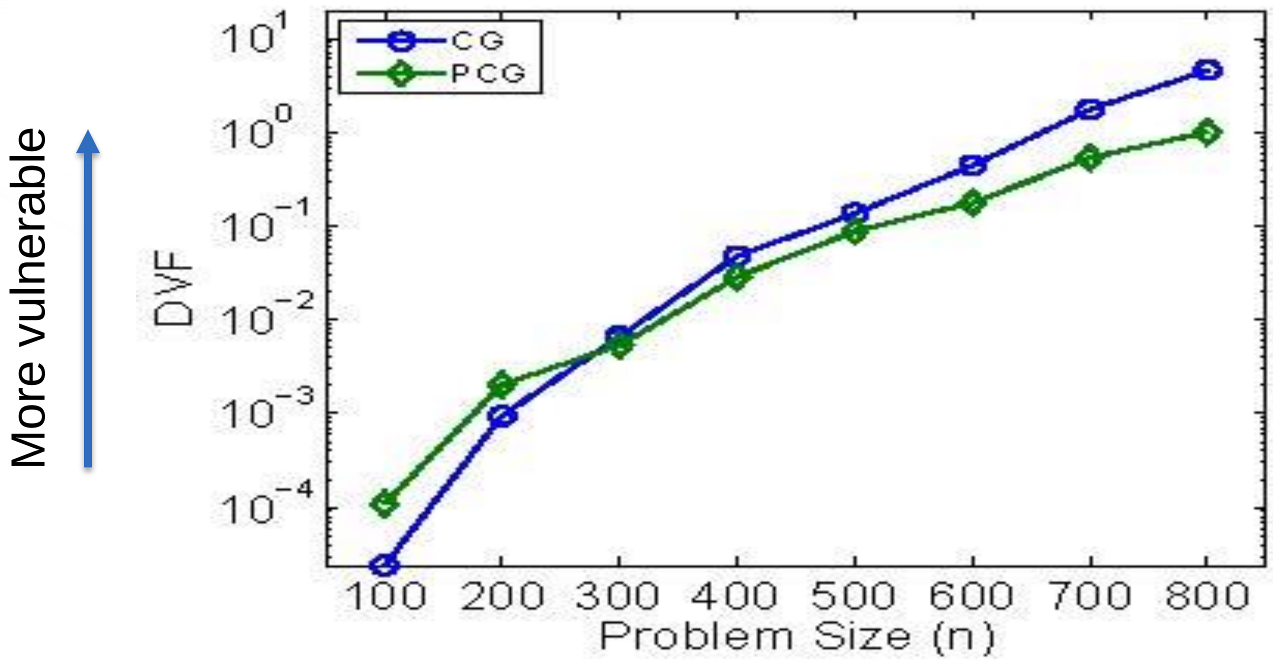


(f) Monte Carlo

Use Case 1: Quantifying the Impact of Algorithm Optimization

- Conjugate Gradient (CG)
  - Providing numeric solutions to linear equations
  - Having mainly four data structures
- Preconditioned Conjugate Gradient (PCG)
  - One of the optimized versions of CG
  - Adding extra data structures
  - Faster convergence

Use Case 1: Quantifying the Impact of Algorithm Optimization



– In PCG, the performance improvement and larger working set size have contradicting contributions to DVF
– We can achieve joint optimization of performance and resilience

OAK RIDGE National Laboratory

DVF Possibilities

- DVF is applicable to other hardware components
  - E.g., Cache hierarchy
  - E.g., Register file
  - E.g., Network interface card
- DVF can benefit the designs of a variety of resilience mechanisms
  - E.g., Checkpointing
  - E.g., Algorithm-based fault tolerance methods (ABFT)
- DVF makes model integration easier
  - Exploring the tradeoff between performance, resilience and power

Conclusions

- We introduce a novel resilience metric, DVF, to help with design of future architectures and applications
- We extended Aspen - a domain specific language - for resilience modeling
- Our method is applied to scientific applications from six computational domains
- Our resilience modeling can be applied to various optimization problems

Acknowledgements

- Contributors and Sponsors
  - Future Technologies Group: http://ft.ornl.gov
  - US Department of Energy Office of Science
    - DOE Vancouver Project: https://ft.ornl.gov/trac/vancouver
    - DOE Blackcomb Project: https://ft.ornl.gov/trac/blackcomb
    - DOE ExMatEx Codesign Center: http://codesign.lanl.gov
    - DOE Cesar Codesign Center: http://cesar.mcs.anl.gov/
    - DOE Exascale Efforts: http://science.energy.gov/ascr/research/computer-science/
  - Scalable Heterogeneous Computing Benchmark team: http://bit.ly/shocmarx
  - US National Science Foundation Keeneland Project: http://keeneland.gatech.edu
  - US DARPA
  - NVIDIA CUDA Center of Excellence

OAK RIDGE National Laboratory