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
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
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 92039  
ross@rosswalker.co.uk  

Robin M. Betz  
San Diego Supercomputer Center  
La Jolla, CA 92039  
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 used for MD simulations, as they can provide significant speedups over traditional central processing units (CPUs). However, the use of GPUs can also introduce new sources of error, such as error correcting code (ECC). This paper investigates the effects of using ECC on GPU-accelerated MD simulations, with the goal of understanding the trade-offs between accuracy and performance.

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

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

DVF Defined

Data Structure Vulnerability → $DVF_d = N_{error} \times N_{ha}$

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

Larger DVF indicates higher vulnerability, and vice versa

$N_{error} = FIT \times T \times S_d$

$N_{ha} \leftarrow$ Hardware Access Pattern

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

<table>
<thead>
<tr>
<th>Technique</th>
<th>Speed</th>
<th>Ease</th>
<th>Flexibility</th>
<th>Accuracy</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ad-hoc Analytical Models</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Structured Analytical Models</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td><strong>Aspen</strong></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Simulation – Functional</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Simulation – Cycle Accurate</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Hardware Emulation (FPGA)</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Similar hardware measurement</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Node Prototype</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Prototype at Scale</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Final System</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Aspen Design Flow

Source code

Aspen code

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

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

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

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

Creating Aspen Models

Simple MM example generated from COMPASS

**Original Source**

```c
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++){
        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;
        }
    }
}
```

**Compiler-generated Aspen**

```c
model MM {
    param float S = 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
```
LULESH in Aspen

```c
147 kernel CalcMonotonicQGradients {
148     execute [numElements]
149     {
150         loads [8 * indexWordSize] from nodelist
151         // Load and cache position and velocity.
152         loads/caching [8 * wordSize] from x
153         loads/caching [8 * wordSize] from y
154         loads/caching [8 * wordSize] from z
155         loads/caching [8 * wordSize] from xvel
156         loads/caching [8 * wordSize] from yvel
157         loads/caching [8 * wordSize] from zvel
158         loads [wordSize] from volo
159         loads [wordSize] from vunew
160         // dx, dy, etc.
161         #flops [98] as dp, simd
162         // delvk delvk
163         #flops [9 + 8 + 3 + 30 + 5] as dp, simd
164         stores [wordSize] to delv_xeta
165         // delxi delvi
166         #flops [9 + 8 + 3 + 30 + 5] as dp, simd
167         stores [wordSize] to delx_xi
168         // delxj and delvj
169         #flops [9 + 8 + 3 + 30 + 5] as dp, simd
170         stores [wordSize] to delv_eta
171     }
172 }
```
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.
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

**Pseudocode**

```pseudocode
procedure VM(A,B,C)
    for i ← 1, 1000 do
    end for
end procedure
```

**Extended Aspen Statements**

```aspen
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)
    }
}
```

**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

**Extended Parser**

**Extended Complier**

**Resilience Modeling Results**
- Data structure A:
  - Number of errors: 30,400
  - Number of memory accesses: 51
  - DVF: 105504e+06

**Syntax Tree**
Evaluation
## Six Computational Kernels

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>Computational Method Class</th>
<th>Major Data Structures</th>
<th>Memory Access Patterns</th>
<th>Example Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector Multiplication (VM)</td>
<td>Dense linear algebra</td>
<td>A, B, and C</td>
<td>Streaming</td>
<td>Homemade code</td>
</tr>
<tr>
<td>Conjugate Gradient (CG)</td>
<td>Sparse linear algebra</td>
<td>A, x, p and r</td>
<td>Template + Reuse + Streaming</td>
<td>NPB CG</td>
</tr>
<tr>
<td>Barnes-Hut simulation (NB)</td>
<td>N-body method</td>
<td>T and P</td>
<td>Random</td>
<td>Online code</td>
</tr>
<tr>
<td>Multi-grid (MG)</td>
<td>Structured grids</td>
<td>R</td>
<td>Template-based</td>
<td>NPB MG</td>
</tr>
<tr>
<td>1D FFT (FT)</td>
<td>Spectral methods</td>
<td>A</td>
<td>Template-based</td>
<td>NPB FT</td>
</tr>
<tr>
<td>Monte Carlo simulation (MC)</td>
<td>Monte Carlo</td>
<td>G and E</td>
<td>Random</td>
<td>XSBench</td>
</tr>
</tbody>
</table>
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
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
  – US Department of Energy Office of Science
    • DOE Vancouver Project: https://ft.orl.gov/trac/vancouver
    • DOE Blackcomb Project: https://ft.orl.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/
  – US National Science Foundation Keeneland Project: http://keeneland.gatech.edu
  – US DARPA
  – NVIDIA CUDA Center of Excellence