Managing Application Resilience A Programming Language Approach

Saurabh Hukerikar, ***Pedro C. Diniz*** and Robert F. Lucas
{saurabh, pedro, rflucas@isi.edu}
USC Information Sciences Institute

Presentation at NIST
Gaithersburg, Maryland
July 21, 2015

Research partially funded by the US Army Research Office under Contract #63295-CS-ACI

USC Viterbi School of Engineering

The Exascale Challenge

ISI Information Sciences Institute

- Technology Trends:
  - Future systems will be based on VLSI technology **less reliable** than that used today (smaller devices/margins)
  - Expected **increase** in Soft Error Rate

- Programming Model Impact:
  - Faults will become the norm, not the exception

These changes require the development of a new range of resilient applications that can react flexibly to runtime errors

USC

USC Viterbi School of Engineering

Real Problem?

ISI Information Sciences Institute

- 2006 Univ. of Virginia builds "Big MAC" out of 1,100 Apple G5
  - Claimed 3rd fastest system in the world
    - Reality: system would not stay up during the day; only at night
    - Reason: PC motherboards don't use ECC memory. Cosmic rays from the Sun caused too many single-bit memory faults
    - After few months decommissioned and sold for parts on eBay
- Other cases:
    - LANL @ 5,000' altitude gets more cosmic rays than at sea level.
      - Supercomputers there see increased gates flip spontaneously in memory and processor chips;
    - SNL has seen bits flip inside data as it flowed from node to node in Red (was a HW problem not cosmic rays)
      - Was very difficult to detect and track down this rare, transient fault

USC

USC Viterbi School of Engineering

Large Scale Studies of DRAM Failures: High-Performance System

ISI Information Sciences Institute

- System: Jaguar @ Oak Ridge Natl. Lab.
  - 11 months study
- Conclusions:
  - Both transient and permanent faults in the field, but that permanent faults constitute at least 70% of all DRAM failures.
  - Multi-bit faults such as row, column and bank faults constitute almost 50% of the DRAM failures in Jaguar's memory system.
  - ChipKill error-correcting codes (ECC) are extremely effective, reducing the node failure rate from uncorrected DRAM errors by 42x compared to single-error correct/double-error detect (SEC-DED) ECC.

References

V. Sridharan and D. Liberty, "A Study of DRAM Failures in the Field,Proc. of 2012 the ACM/IEEE Supercomputng Conf., 2012

USC

USC Viterbi School of Engineering

Real Problem @ Exascale?*

ISI Information Sciences Institute

Transient error bit flips in memory are the first place we are experiencing continuous failure. Thankfully, ECC memory continuously fixes the single bit errors, but not double bit errors

- Double bit Error Rate
  - Jaguar has a lot of memory (362 TB)
  - It endures a Constant Stream of Single Bit Errors (ECC corrects these)
  - It has a double bit error about every 24 hours (beats DRAM FIT rate)
  - ChipKill allows the system to run through double bit errors but DRAM can not correct double bit errors
- What is double bit Error Rate of Exascale system?
  - Exascale system target is 128 PB of memory (354 times Jaguar)
  - Translates into a double bit error about every 4 minutes
  - Frequent enough to need something better than ChipKill at Exascale

USC

* Data from Al Geist @ Oak Ridge Natl. Lab., 2013

USC Viterbi School of Engineering

System Scale: Trends & Projections

ISI Information Sciences Institute

	2009	2012	2016	2020
System Peak Performance	2 Petaflops	20 Petaflops	~200 Petaflops	1 Exaflops
System Memory	0.3 PB	1.5 PB	~5 PB	~30 PB
System Node Count	8,000	18,000	~50,000	~100,000
Total Core Count	300,000	1,500,000	~ 50 million	~ 1 billion
Mean Time To Interrupt (MTTI)	1 day	20 hours	40 – 50 minutes	20 minutes
Power	7MW	8.2MW	~15MW	20MW

Petascale systems today already experience[1]:

- ~20 faults/hour
- 1 double-bit DRAM error every 24 hours
- Constant stream of single bit memory errors

[1] Al Geist, "What is the monster in the closet?" Talk at Workshop on Architectures I: Exascale and Beyond: Gaps in Research, Gaps in our Thinking

USC

USC Viterbi School of Engineering

Resilience: Hardware Challenges

ISI Information Sciences Institute

- Number of components both memory and processors will increase by an order of magnitude which will increase hard and soft errors
- Smaller circuit sizes, running at lower voltages to reduce power consumption, increases the probability of switches flipping spontaneously due to thermal and voltage variations as well as radiation, increasing transient errors
- Power management cycling significantly decreases the components lifetimes due to thermal and mechanical stresses
- Resistance to add additional HW detection and recovery logic right on the chips to detect silent errors as it will increase power consumption by 15% and increase the chip costs.
- Heterogeneous systems make error detection and recovery even harder, for example, detecting and recovering from an error in a GPU can involve hundreds of threads simultaneously on the GPU and hundreds of cycles in drain pipelines to begin recovery.

USC

Resilience: Software Challenges

USC Viterbi School of Engineering

ISI Information Sciences Institute

- Existing fault tolerance techniques (global checkpoint/global restart) will be impractical at Exascale.
- There is no standard fault model, nor standard fault test suite or metrics to stress resilience solutions and compare them fairly.
- Errors, fault root causes, and propagation are not well understood. Hard to solve something that isn't understood
- Current programming models and languages do not offer a paradigm for resilient programming. A failure of a single task often leads to the killing of the entire MPI application.
- Current applications (and most System software) is not fault tolerant nor fault aware and is not designed to confine errors/faults, to avoid or limit their propagation, and to recover from them in a holistic fashion. (perfect opportunity for co-design)

USC

USC Viterbi School of Engineering

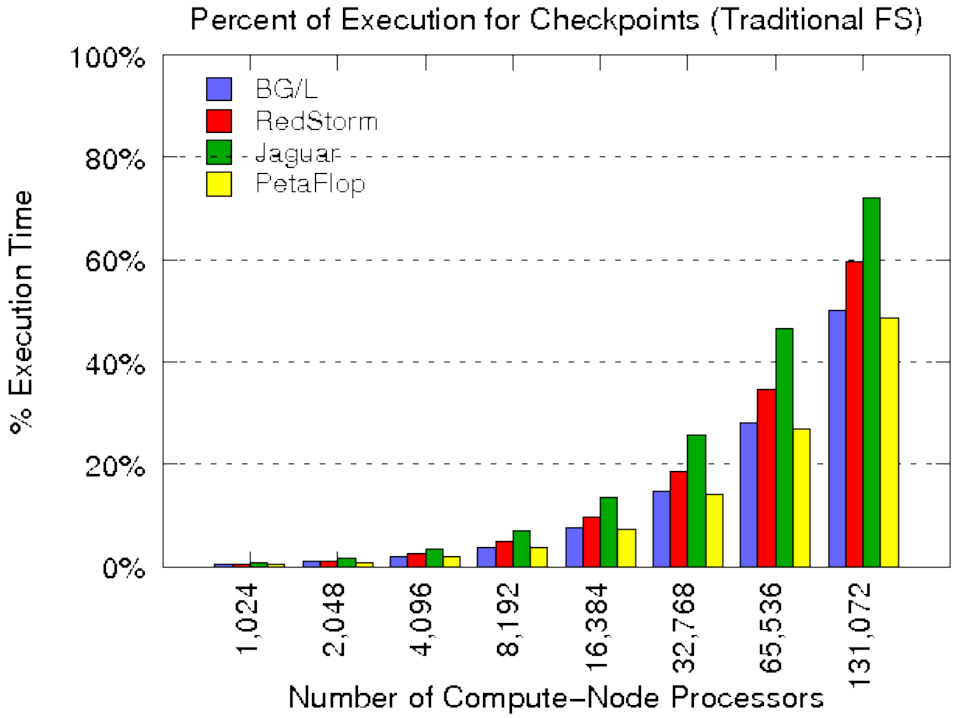
What to Do? Mitigation Strategies

ISI Information Sciences Institute

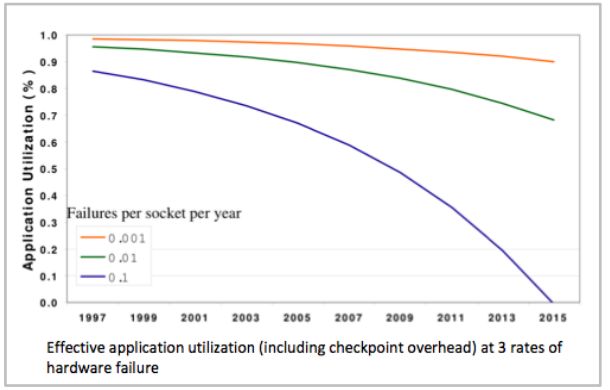
- Hardware Approaches:
  - Hardware TMR (Triple Modular Redundancy)
  - More ECC and/or ChipKill
- Software Approaches:
  - Check-Pointing and Restart (C/R) to disk/memory
  - FT-MPI (redundancy and/or protection of messages)
  - Software TMR – replication of pointers/code
  - Algorithm Based Fault Tolerance (ABFT)
- Holistic Approaches:
  - Use high-level approaches to convey Algorithm-level Knowledge
  - Checkpoint size reduction (programmer assisted variable check-pointing)

USC

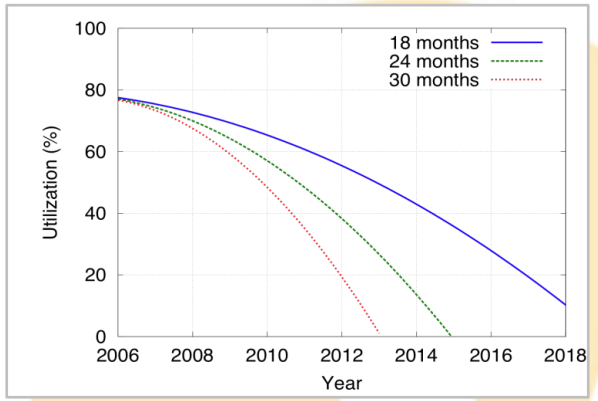
Check-Pointing and Restart



Reference:
Oldfield et al., *Modeling the Impact of Checkpoints on Next-Generation Systems*. MSST, 2007

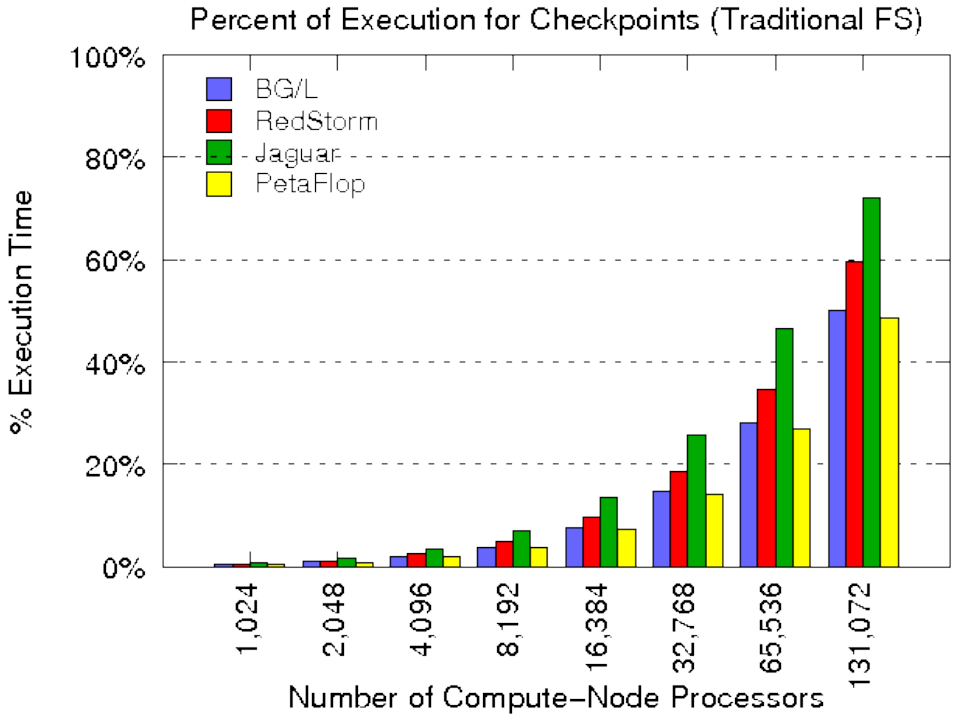


(Courtesy of Lucy Nowell & Sonia Sachs, at DoE)

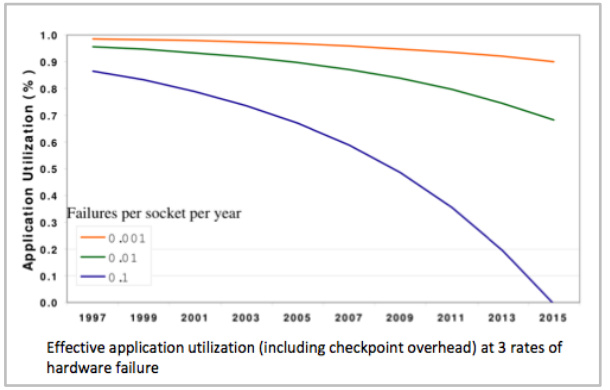


Reference:
Schroeder and Gibson, *Understanding Failures in Petascale Computers*. Journal of Physics, 2007
(assuming that the number of cores per socket grows by a factor of 2 every 18, 24 and 30 months)

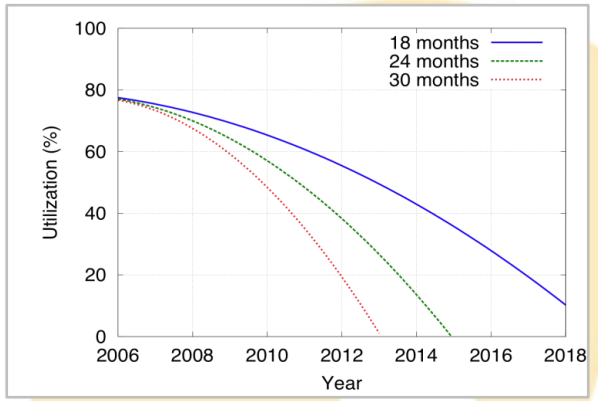
Check-Pointing and Restart



Reference:
Oldfield et al., *Modeling the Impact of Checkpoints on Next-Generation Systems*. MSST, 2007



(Courtesy of Lucy Nowell & Sonia Sachs, at DoE)



Reference:
Schroeder and Gibson, *Understanding Failures in Petascale Computers*. Journal of Physics, 2007
(assuming that the number of cores per socket grows by a factor of 2 every 18, 24 and 30 months)

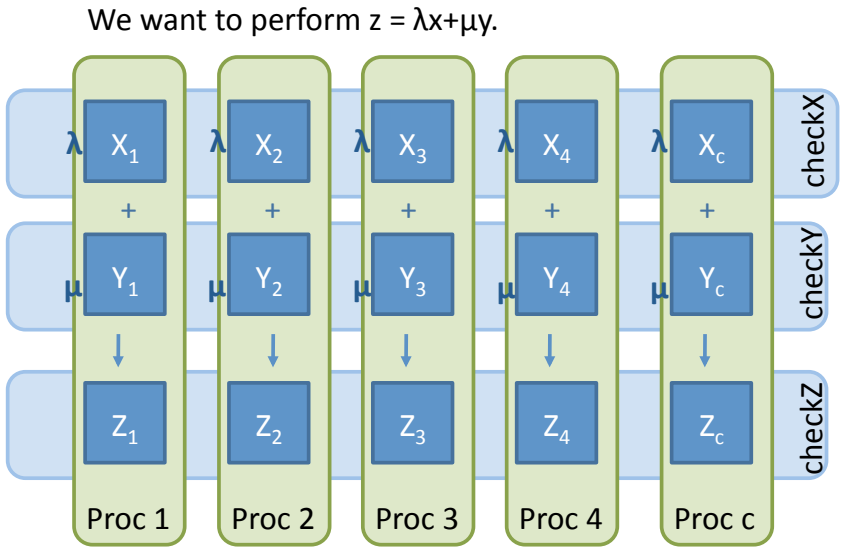
Not a Scalable Approach

USC Viterbi School of Engineering

ABFT: Linear Algebra Computations

ISI Information Sciences Institute

- Idea: Exploit Linearity and Associativity of Algebraic Operations
  - Compute CheckSum of Rows and Columns
  - Check the CheckSum of Results Equals Sum of CheckSums
- Extend to More Complex Linear Operations
  - LU, Cholesky and QR Factorization
- Can Detect and Correct Single Errors
  - Linear Space and Time Complexity
- Works for Limited Computations...



References:

K. Huang, J. Abraham, "*Algorithm-Based Fault Tolerance for Matrix Operations*," IEEE Trans. on Comp. (Spec. Issue Reliable & Fault-Tolerant Comp.), C-33, 1984, pp. 518-528

Aurelien Bouteiller, Thomas Herault, George Bosilca, Peng Du, and Jack Dongarra, "Algorithm-based Fault Tolerance for Dense Matrix Factorizations, Multiple Failures and Accuracy, ACM TECS, 2013

USC

USC Viterbi School of Engineering

Naturally Resilient Algorithms

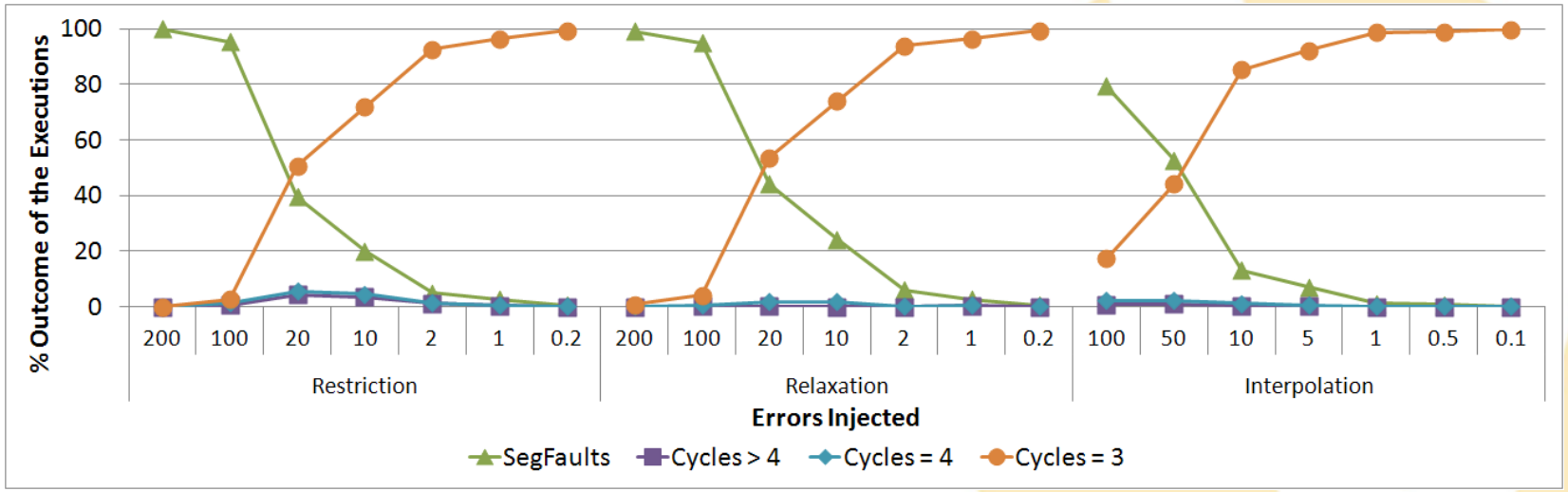
Ex: Adaptive Multi-Grid Solver (AMG)

ISI Information Sciences Institute

- Work at Lawrence Livermore Natl. Lab., Calif.
- Extensive fault injection campaign
  - 3 main AMG phases
  - 7 fault injection rates for each phase
  - 10,000 executions for each phase and fault injection rate

	Restriction	Interpolation	Relaxation
Rate 1	200 f, 4273 e/s	100 f, 5128 e/s	200 f, 1666 e/s
Rate 2	100 f, 2136 e/s	50 f, 2564 e/s	100 f, 833 e/s
Rate 3	20 f, 427 e/s	10 f, 513 e/s	20 f, 167 e/s
Rate 4	10 f, 214 e/s	5 f, 256 e/s	10 f, 83 e/s
Rate 5	2 f, 43 e/s	1 f, 51 e/s	2 f, 17 e/s
Rate 6	1 f, 21 e/s	0.5 f, 25 e/s	1 f, 9 e/s
Rate 7	0.2 f, 4 e/s	0.1 f, 5 e/s	0.2 f, 2 e/s

- Vulnerability profile of each phase:



USC

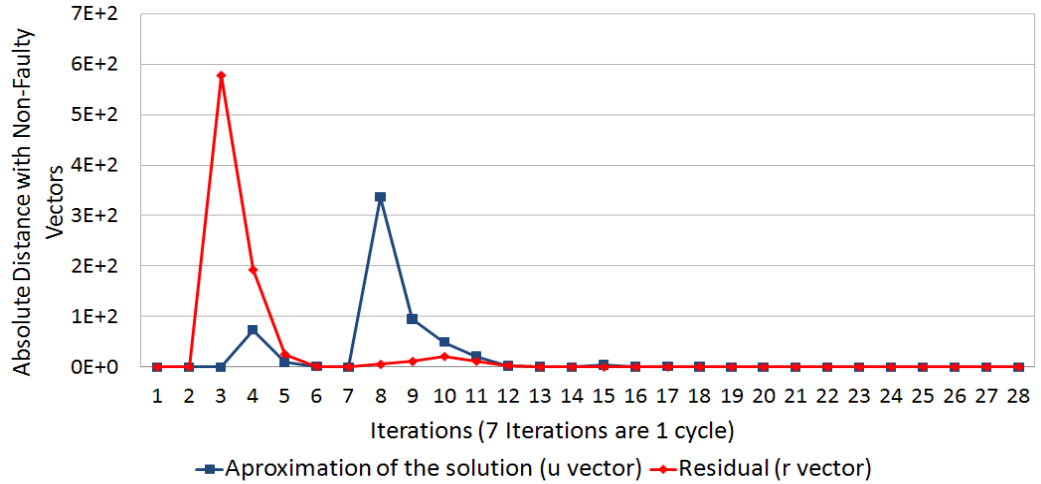
USC Viterbi School of Engineering

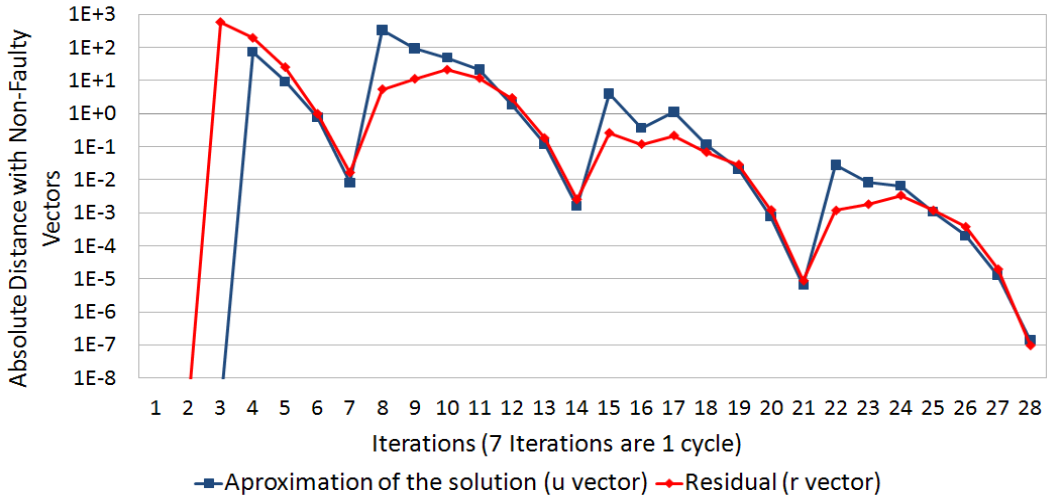
Naturally Resilient Algorithms

Ex: Adaptive Multi-Grid Solver (AMG)

Information Sciences Institute

- Numeric errors smoothed out
  - The error spreads among the variables when execution returns to the finest levels.
  - The correction on the coarsest grids is more significant than the subsequent contamination
  - AMG's coarsening smoothes out the effects of large local errors
  - Other errors induce segmentation faults





USC

Reference:

Marc Casas-Guix, Bronis R. de Supinski, Greg Bronevetsky, Martin Schulz: "Fault Resilience of the Algebraic Multi-Grid Solver", Proc. of the Intl Conf. on Supercomputing (ICS) 2012: 91-100

USC Viterbi School of Engineering

Software TMR: Pointer (index) Replication

ISI Information Sciences Institute

- Source code #pragma recognized by ROSE Compiler (LLNL, Calif.)

```
int main (int argc, char** argv) {
   #pragma critical 3
   int* A_j = NULL;
   int A_i[100];
   int y_data[100], x_data[100], A_data[100];
   int jj = 0, i = 0;

   //A_j  = (int *) malloc (sizeof(int)  * 100);


   for (jj = A_i[i]; jj < A_i[i + 1]; ++jj) {
      y_data[i] += A_data[jj] * x_data[A_j[jj]];
   }

   free(A_j);
}
```

```
#include "new_critical_var.h"

int main(int argc,char  **argv)
{

#pragma critical 3
  int *A_j = (int *)((int *)((void *)0));
  int *A_j0 = (int *)((int *)((void *)0));
  int *A_j1 = (int *)((int *)((void *)0));
  int A_i[100UL];
  int y_data[100UL];
  int x_data[100UL];
  int A_data[100UL];
  int jj = 0;
  int i = 0;
//A_j  = (int *) malloc (sizeof(int)  * 100);
  for (jj = A_i[i]; jj < A_i[i + 1]; ++jj) {
    y_data[i] += (A_data[jj] * x_data[((int *)(triplication(A_j,A_j0,A_j1)))[jj]]);
  }
  free(((int *)(triplication(A_j,A_j0,A_j1))));
  return 0;
}
```

```
void* triplication(p1, p2, p3) {
   if (p1 == p2 ll p1 == p3) {
      return p1;
   } else if (p2 == p3) {
      return p2;
   } else {
      exit(1);
   }
}
```

USC

USC Viterbi School of Engineering

Software TMR: Pointer (index) Replication

ISI Information Sciences Institute

- Source code #pragma recognized by ROSE Compiler (LLNL, Calif.)

```
int main (int argc, char** argv) {
  #pragma critical 3
  int* A_j = NULL;
  int A_i[100];
  int y_data[100], x_data[100], A_data[100];
  int jj = 0, i = 0;

  //A_j = (int *) malloc (sizeof(int) * 100);


  for (jj = A_i[i]; jj < A_i[ + 1]; ++jj) {
     y_data[i] += A_data[jj] * x_data[A_j[jj]];
  }

  free(A_j);
}
```

number of replicas of pointer variable in the subsequent program text line

```
#include "new_critical_var.h"

int main(int argc,char **argv){

#pragma critical 3
  int *A_j = (int *)((int *)((void *)0));
  int *A_j0 = (int *)((int *)((void *)0));
  int *A_j1 = (int *)((int *)((void *)0));
  int A_i[100UL];
  int y_data[100UL];
  int x_data[100UL];
  int A_data[100UL];
  int jj = 0;
  int i = 0;
//A_j = (int *) malloc (sizeof(int) * 100);
  for (jj = A_i[i]; jj < A_i[i + 1]; ++jj) {
    y_data[i] += (A_data[jj] * x_data[((int *)(triplication(A_j,A_j0,A_j1)))[jj]]);
  }
  free(((int *)(triplication(A_j,A_j0,A_j1))));
  return 0;
}
```

```
void* triplication(p1, p2, p3) {
    if (p1 == p2 || p1 == p3) {
        return p1;
    } else if (p2 == p3) {
        return p2;
    } else {
        exit(1);
    }
}
```

USC

Observation: Not All Faults Need to be Fatal

- HPCS Random Access Benchmark explicitly tolerates a finite number of errors in both memory and arithmetic.
- Iterative solvers like Algebraic Multi-Grid (AMG) can hill climb over errors in intermediate state.
- Post-processing such as iterative refinement after solving a linear system can correct for small errors, as a function of condition number.
- Users can add their own fault detection and correction techniques.

USC Viterbi School of Engineering

Focus of Our Work

ISI Information Sciences Institute

Provide Linguistic Mechanisms to Convey Criticality of Errors

- Emphasis on keeping things simple
- Existing libraries and application code base need not be entirely rewritten
- Engage all software layers including the application
  - Compiler infrastructure,
  - Operating system
- Fault Model: SECDED Failures
- Complementary:
  - Check-Pointing & Restart: Increase Checkpoint Interval
  - Redundant Computation: Potential Energy Savings

USC

USC Viterbi School of Engineering

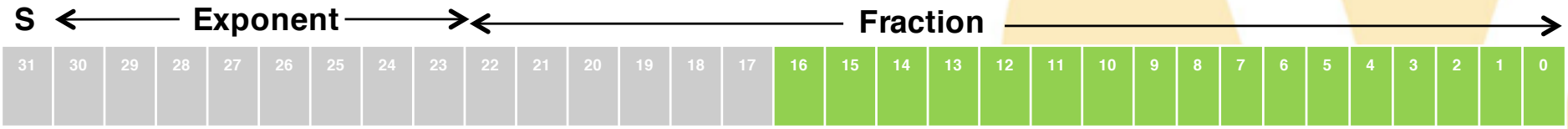
Programming Model Extensions: Type Qualifiers

ISI Information Sciences Institute

```
tolerant int rgb[XDIM][YDIM];
```

```
tolerant<MAX.VALUE=...> unsigned int counter;
```


```
tolerant<precision.6f> double low_precision;
```



USC

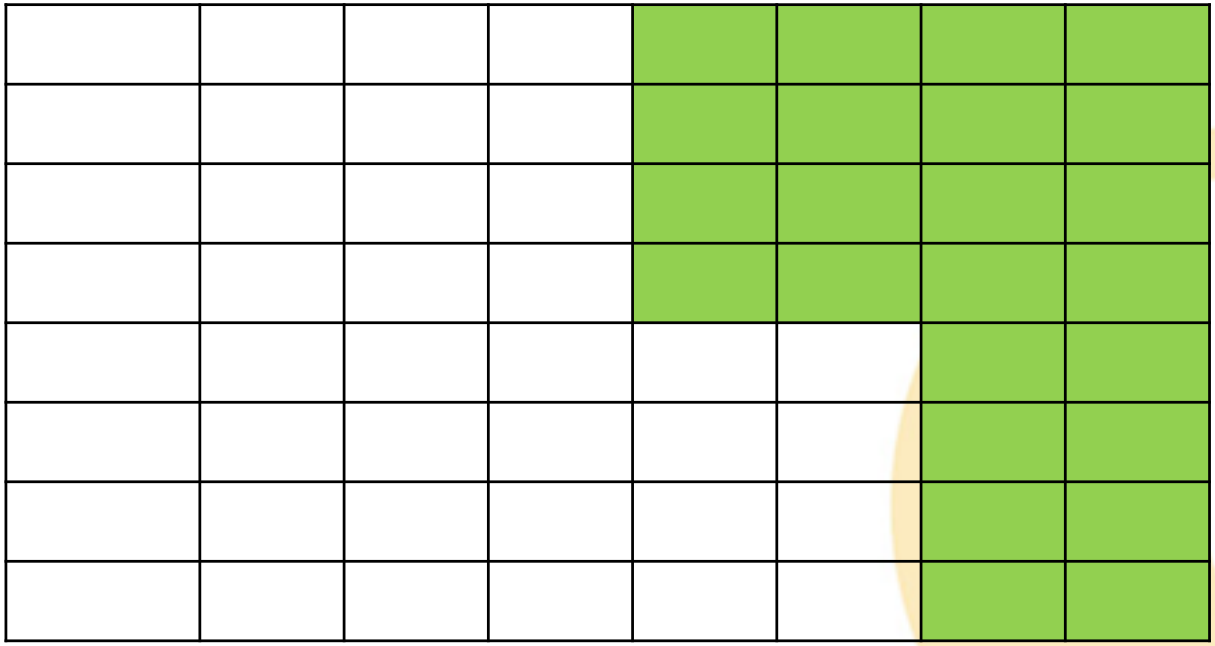
USC Viterbi School of Engineering

Dynamic Memory Allocation

ISI Information Sciences Institute

<type>* <var> = (<cast>) tolerant_malloc(sizeof(<type>));

<type>* <var> = (<cast>) tolerant_malloc(NUM * sizeof(<type><MAX.VALUE=..>));



USC

USC Viterbi School of Engineering

Simple Experiments

ISI Information Sciences Institute

- Methodology:
  - "Inject" errors in memory
  - Check if storage has been labeled as tolerant
  - If so ignore; else continue (and possibly crash later)

- Simple Codes:
  - HPCC Random Access
  - Molecular Dynamics
  - Algebraic Multi-Grid Solver
  - Graph – Breath-First-Search

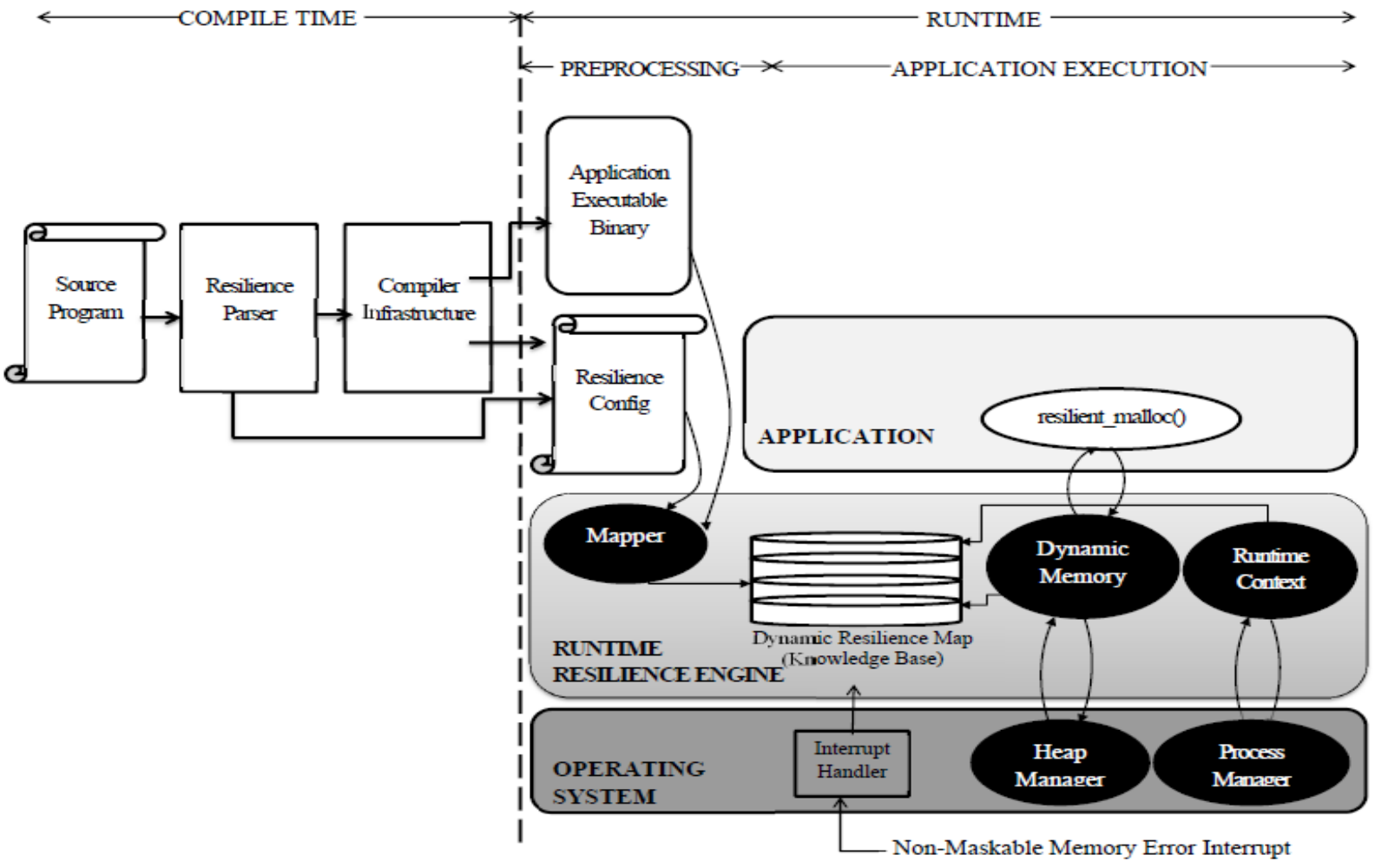
- Simple "Amelioration":
  - Ignore in many cases (AMG will naturally converge...)
  - Remove offending data in other cases (MD code)

USC

USC Viterbi School of Engineering

System Workflow

ISI Information Sciences Institute



USC

USC Viterbi School of Engineering

Experimental Results

ISI Information Sciences Institute

Fault Injection Rate (minutes)	% Execution Runs to Completion			
	Random Access	**Molecular Dynamics Simulation**	**Algebraic Multi-Grid Linear Solver**	**Graph Breadth-First-Search Traversal**
15	99.5 %	66.2 %	96.1 %	61.4 %
10	99.2 %	61.3 %	92.7 %	32.8 %
5	99.1 %	36.3 %	86.6 %	19.8 %
2	97.4 %	7.1 %	81.2 %	2.3 %
1	96.1 %	1.2 %	63.3 %	0.8 %

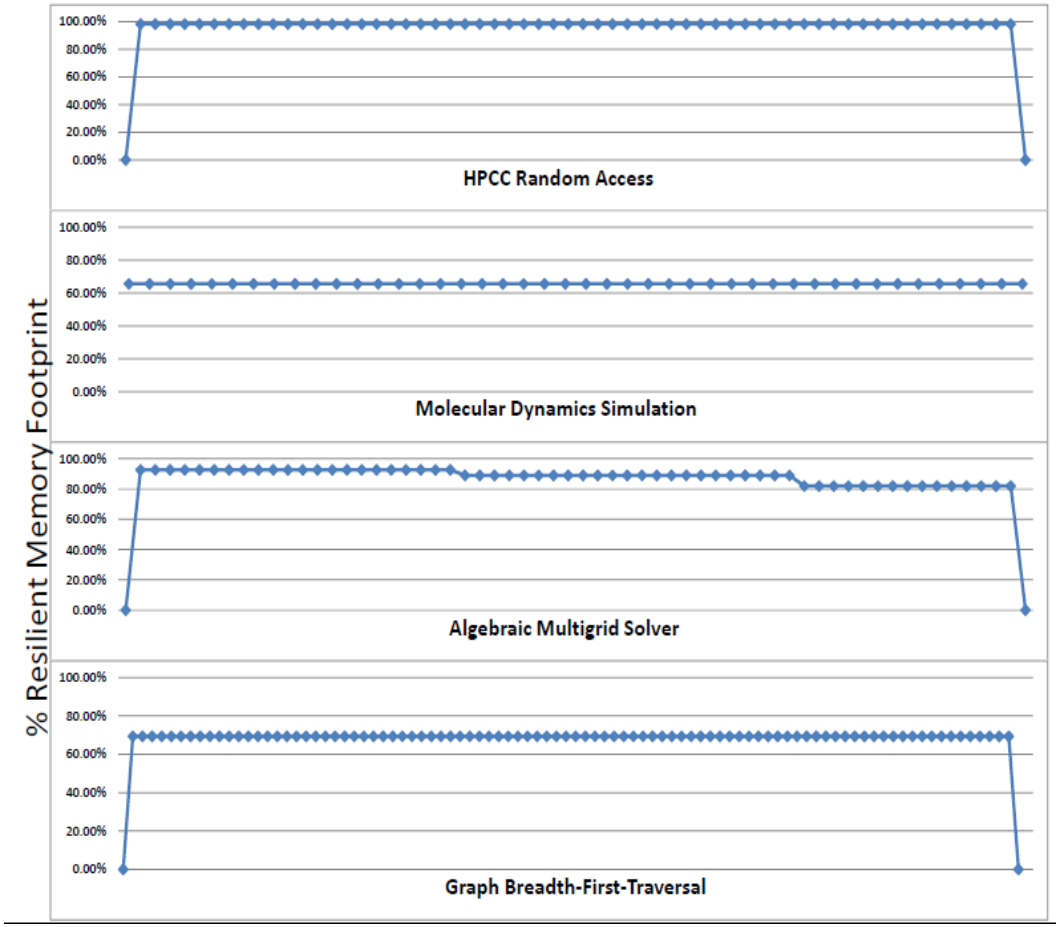
Fault injection: Multi-bit faults, non-recoverable by ECC

USC

USC Viterbi School of Engineering

Vulnerability Analysis: Execution Lifetime

Information Sciences Institute



Safely Ignore Errors: Programmer knows best

If molecule outside box – ignore it - removed

Just keep computing (ex. NaN) will converge – albeit slower

Just crashed – nothing you can do about it for the time being

USC

USC Viterbi School of Engineering

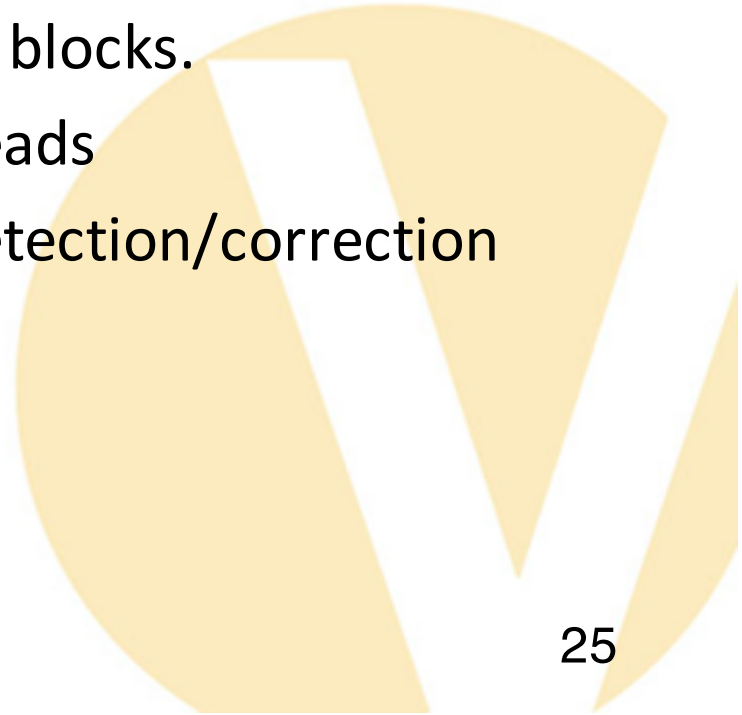
How about Silent Data Corruption?

ISI Information Sciences Institute

Application level fault detection approach that is based on adaptive redundant multithreading

(*simple*) Idea:

1. Programmer define structured code blocks.
2. Blocks are executed by multiple threads
3. Their outputs compared for error detection/correction



USC

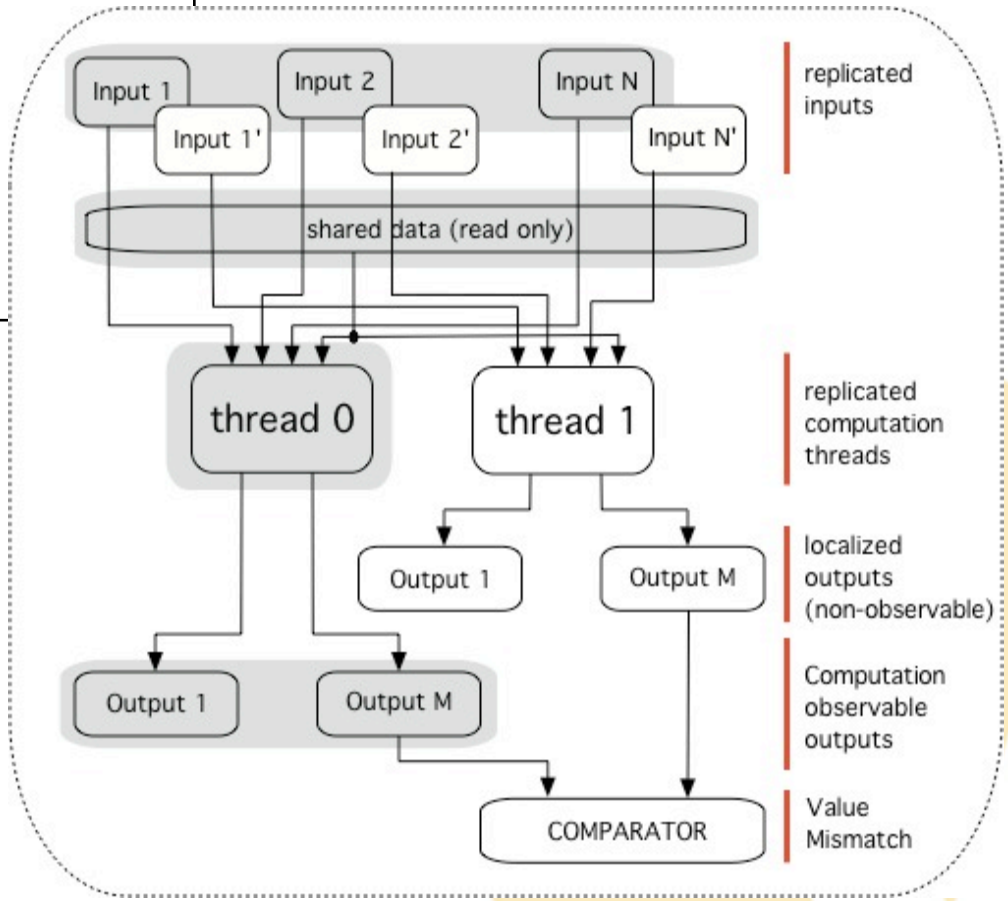
Detection and Recovery: Robust Code Blocks

```
#pragma robust private ( variable list ... )
   shared (variable list ... )
     compare (variable list ... ) {
      /* ... code ... */
 }
```

- How to use this information?
  - Dual Threading for detection of invalid executions
  - Triple Treading uses “compare” variables for voting
  - Can be Adaptive
    - Dual threading for simple detection
    - Roll-back and proceed with Triple-Threading

Robust Directive

```
#pragma robust private ( variable list ... )
   shared (variable list ... )
     compare (variable list ... ) {
      /* ... code ... */
 }
```

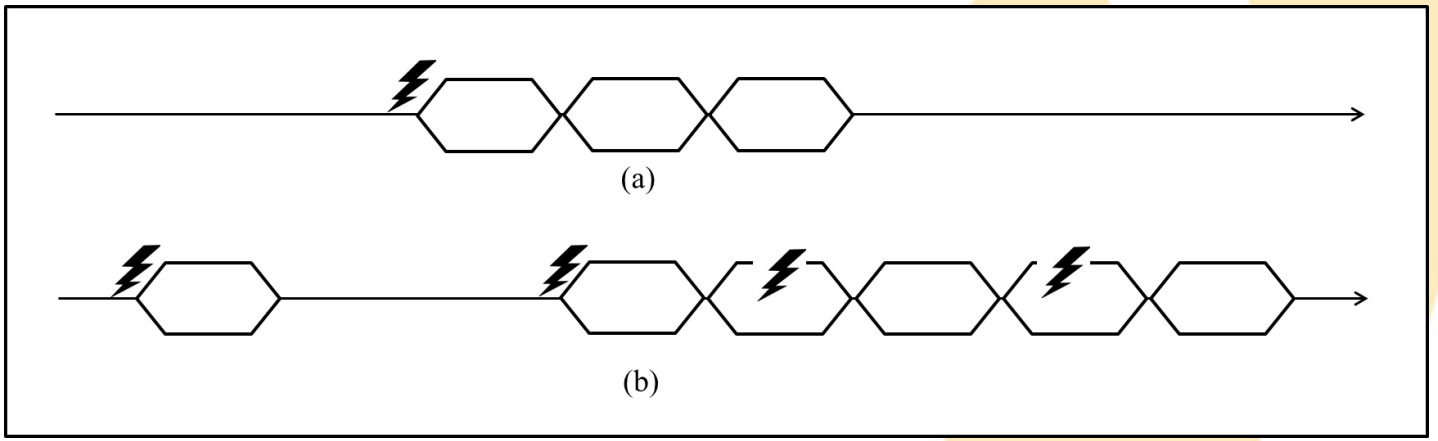


USC Viterbi School of Engineering

Adaptive Strategy

ISI Information Sciences Institute

- Monitoring Events – ***corrected errors***
- Time Since Last Event (TSLE) & Time Between Events (TBE)
  - On Event Activate Redundant Threading
  - Deactivate if event-free execution extends beyond TBE
- More "aggressive" by increasing TBE
  - Trade-Off: Energy vs. Robustness

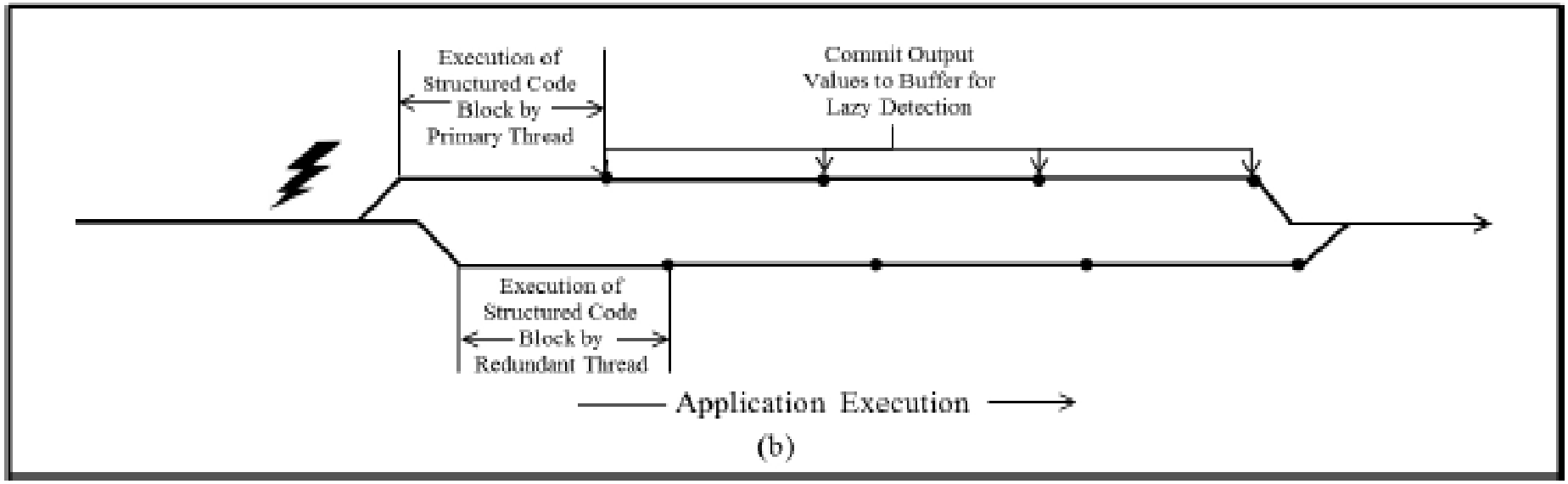


USC

USC Viterbi School of Engineering

Lazy Fault Detection

ISI Information Sciences Institute



- Redundant Computation and Detection "off-critical path"
- Intermediate Results Committed to a Ring Buffer
- Dedicated "lazy" detection thread which does value comparison

USC

USC Viterbi School of Engineering

Experimental Evaluation

ISI Information Sciences Institute

- **Applications**
  - Double Precision Matrix-Matrix Multiply
  - Sparse Matrix Vector Multiply
  - Conjugate Gradient
  - Self-Stabilizing CG (Variant of CG)

- **Experiments**
  - Single compute node on USC's Center for High Performance Computing (HPCC)
  - Dynamic Fault Injection Tool: Generates fault "events"
  - Time to solution for 10000 application runs
  - Fault Event Rate = 1, 2, 3, 4, 5 per run
  - Fault distribution randomized

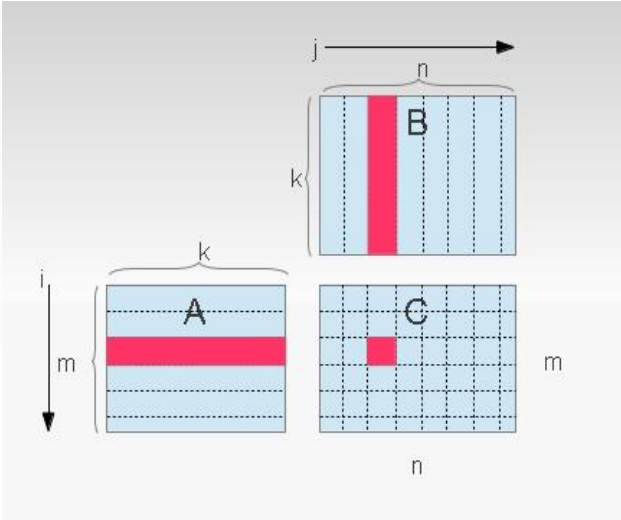
USC

USC Viterbi School of Engineering

Example: DGEMM

Information Sciences Institute

```
for (i = 0; i < N; i++) {
  for (j = 0; j < N; j++) {
    for (k = 0; k < N; k++) {
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
  }
}
```




```
for (i = 0; i < N; i++) {
  for (j = 0; j < N; j++) {
    #pragma robust private (k) shared (i, j) compare (C[i][j]){
      for (k = 0; k < N; k++) {
        C[i][j] = C[i][j] + A[i][k] * B[k][j];
      }
    }
  }
}
```

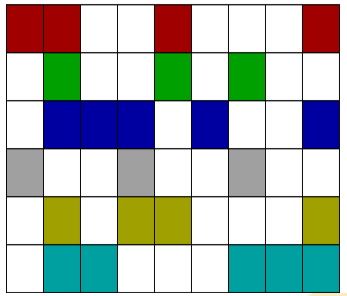
USC

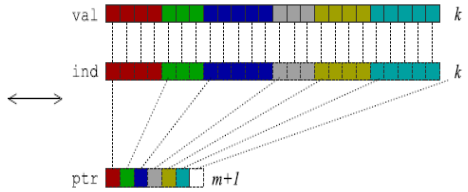
USC Viterbi School of Engineering

Example: SpMV

Information Sciences Institute

```
for ( i = 0 ; i < N; i ++)
{
  for ( j = row_ptr [ i ] ; j < row_ptr [i + 1]; j++)
        y [ i ] += a [j] x [ col_ind [j] ] ;
}
```





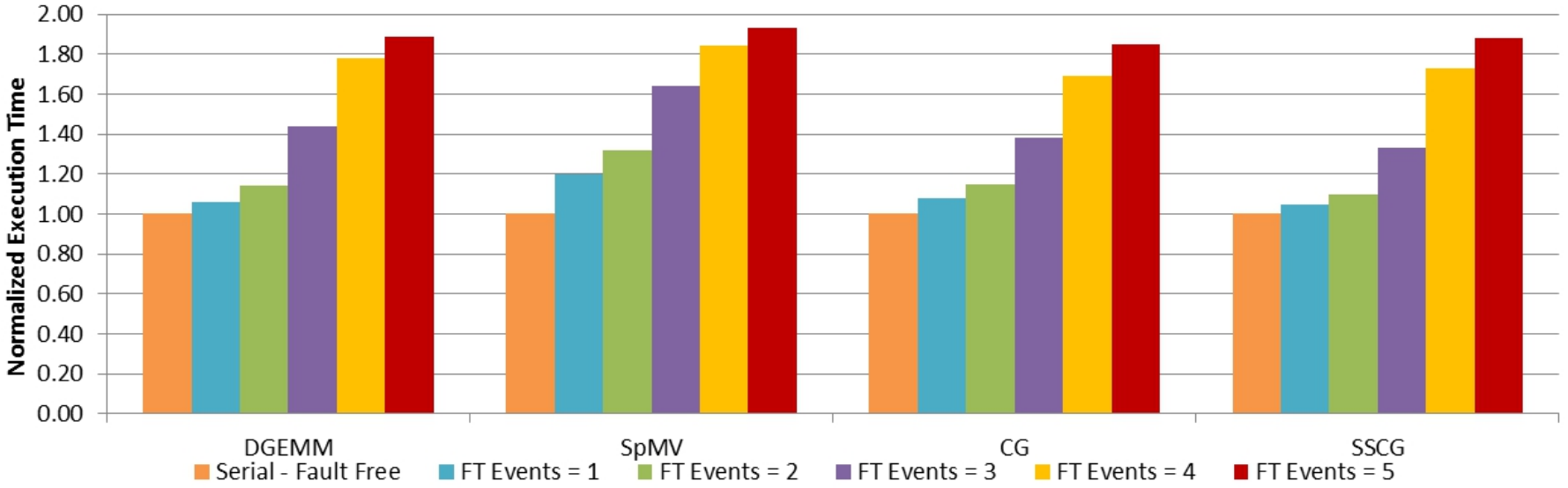

```
for ( i = 0 ; i < N; i ++)
{
  #pragma robust private (j) shared (row_ptr, col_ind, a) compare (y[i])
  {
        for ( j = row_ptr [ i ]; j < row_ptr [i + 1]; j++)
              y [ i ] += a [j] x [ col_ind [j] ];
  }
}
```

USC

USC Viterbi School of Engineering

Numeric Results

ISI Information Sciences Institute



Average execution times are 1.78x, 1.81x, 1.7x and 1.74x respectively (in comparison to the serial fault-free case)

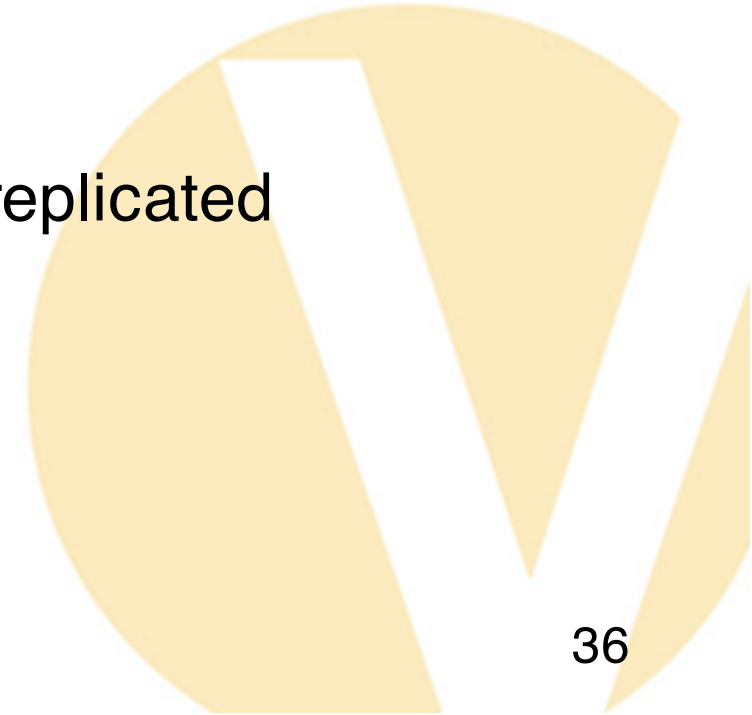
USC

USC Viterbi School of Engineering

Thread Placement Policy

ISI Information Sciences Institute

- Where to place the redundant thread?
  - **Trailing Thread:**
    - Same processor core.
    - Data locality advantages for shared data.
  - **Separate Cores:**
    - Minimal impact on scheduling
    - Requires shared data to be replicated

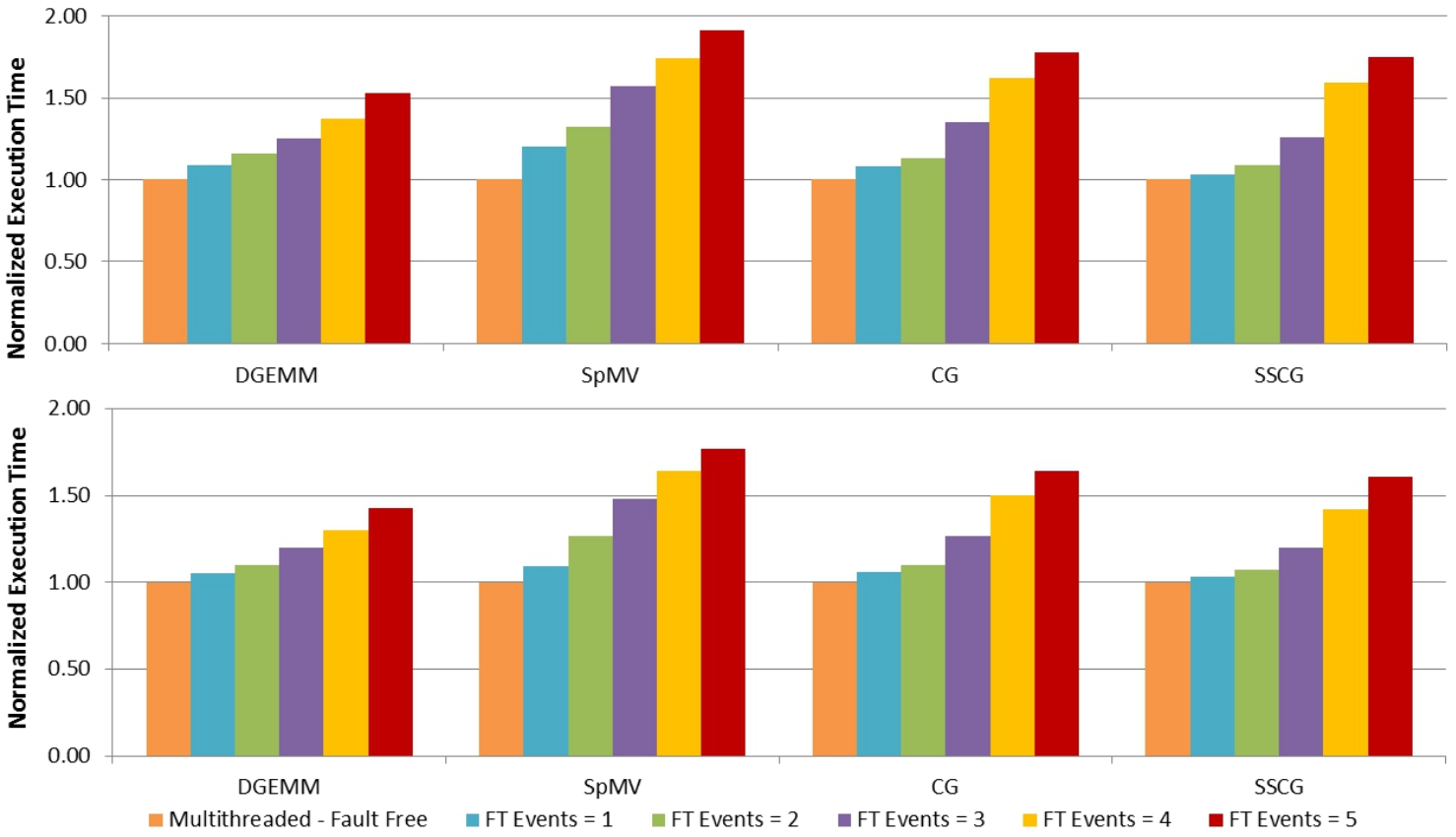


USC

USC Viterbi School of Engineering

Thread Placement Policy

ISI Information Sciences Institute



Trailing Threads

Separate Cores

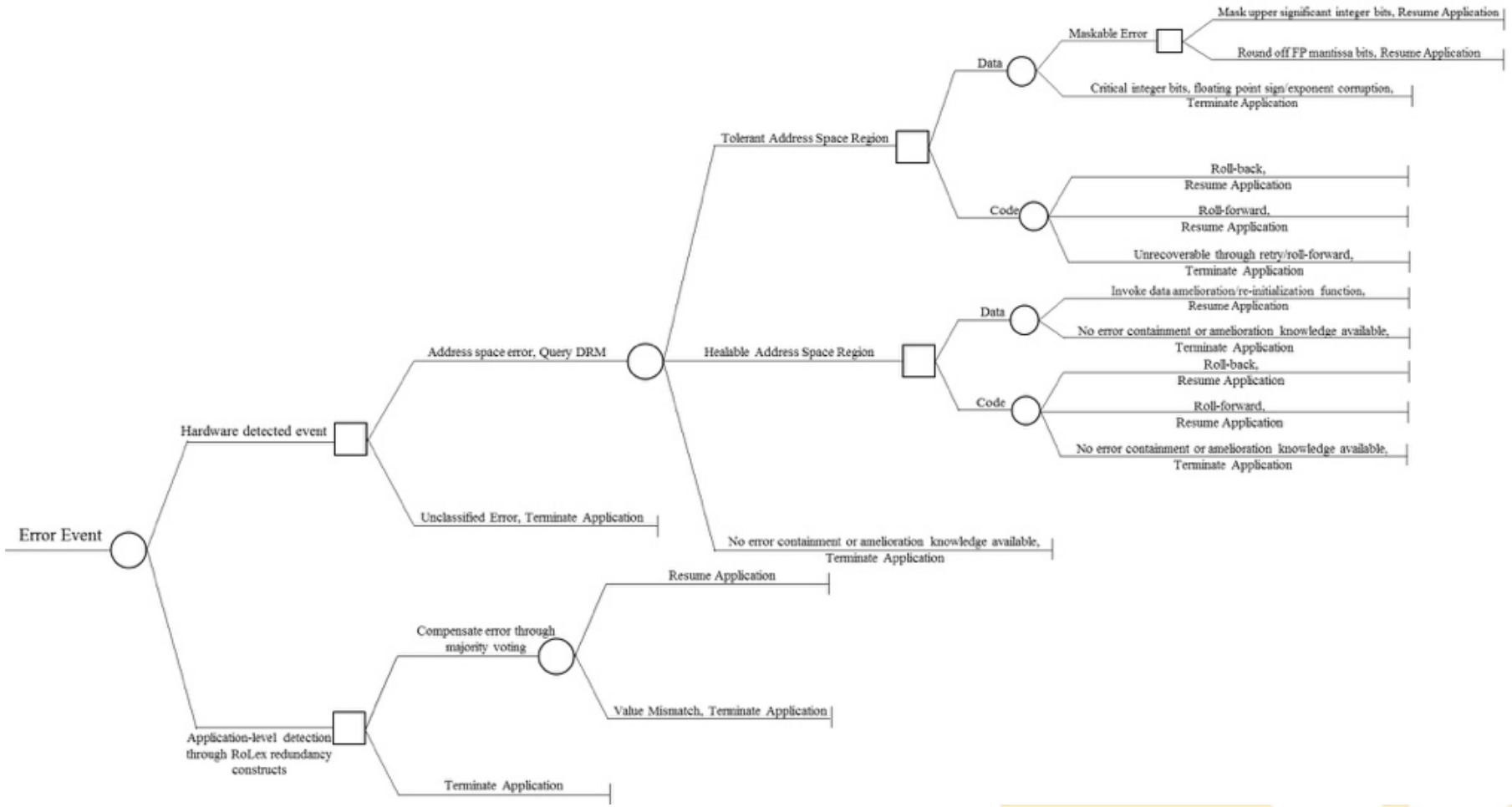
Separate Cores approach has marginally lower overhead

USC

USC Viterbi School of Engineering

Many Response to Errors

ISI Information Sciences Institute

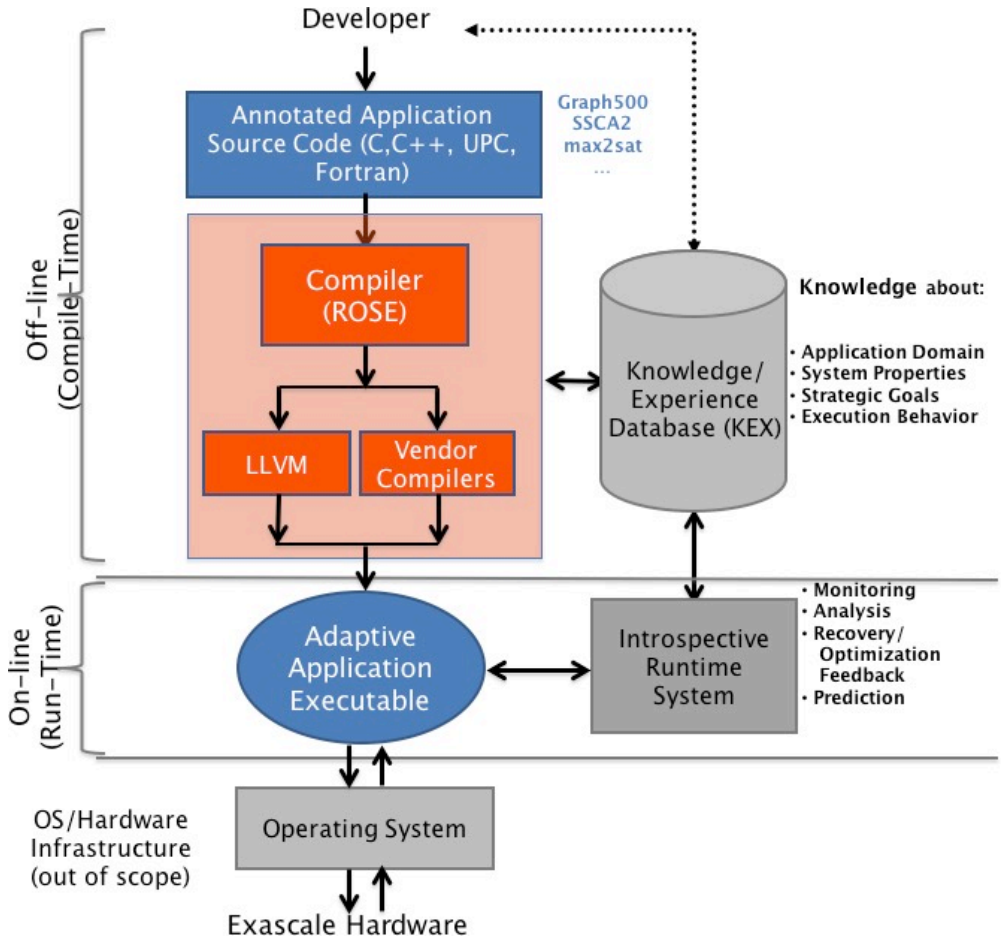


USC

USC Viterbi School of Engineering

Vision: Introspection

ISI Information Sciences Institute



- **Assertion Language:** We developed a new assertion language that allows the expression of user knowledge about dynamic program state, tolerance requirements, and the fault-tolerant control of redundant program execution.
- **ROSE**: The ROSE compiler infrastructure used to perform an optimized translation from constructs of the assertion language into the knowledge base and rules of the introspection framework and its inference engine.
- **Introspection Framework for Resilience (IFR)**: The IFR monitors the execution of the application and recognizes special events such as uncorrected faults detected by ECC or violations of a correctness predicate. It reasons about each event and generates an appropriate action based on the results of its analysis, the application's resilience requirements, and the computation status.
- **Knowledge/Experience Database (KEX):** The KEX stores knowledge about the application as well as behavioral data extracted from actual executions and IFR analysis. This information can be used both by the software developer and the compiler to adapt their strategies and enhance the resilience of subsequent versions of the application.

USC

USC Viterbi School of Engineering

Future Trends

ISI Information Sciences Institute

- Will this problem be relevant in 5, 10 years?
  - Use NVRAMs, Resistive RAMs to C/R in Memory
- Will see better ECC-based Techniques
  - ChipKill Techniques are very Effective Today…
- Communication Libraries Will be more Robust
  - Does not seem to be a Problem Today
  - Efforts FT-MPI…
- Scheduling and Dynamically Adapt to Failing Nodes

USC

USC Viterbi School of Engineering

Conclusion & Perspectives

ISI Information Sciences Institute

- The need for Reliability is not New:
  - Hardware / Software TMR in Space-borne & Safety-critical systems.
  - HPC Systems have a 'different' constraint – all simultaneously available and scalability issues.

- Plenty of system and application knowledge about acceptable solutions – but no widely accepted mechanisms to covey it.

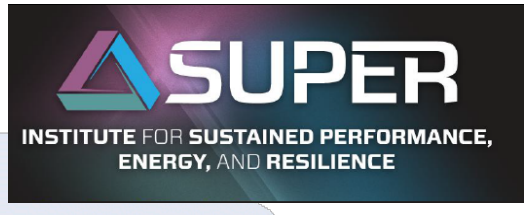
- Programmers Need to be Aware the Machine is no longer 'perfect' – change of paradigm.

USC

USC Viterbi School of Engineering

Acknowledgements

ISI Information Sciences Institute



Partial support for this work was provided by the US Army Research Office under Contract #63295-CS-ACI and Award W911NF-13-1-0219.
Partial support for this work was provided by the US Department of Energy (DoE) Office of Science, Advanced Scientific Computing Research through the SciDAC-3 SUPER Research Institute (Contract Number DE-SC0006844)

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04- 94AL85000

USC