Adaptive Software-based Fault Tolerance for Space Multicore Processing

Adam Jacobs, Grzegorz Cieslewski, Alan D. George
NSF Center for High-Performance Reconfigurable Computing (CHREC) [1]
ECE Department, University of Florida
{jacobs, cieslewski, george}@chrec.org

Increasing demand for high-performance computing in space, coupled with limitations of device-level methods for SEU mitigation, are driving innovations in advanced space computing with system- and application-level fault tolerance. As devices increasingly feature multicore architectures, the space community must adapt and incorporate these devices into future missions. These multicore devices are an increasingly attractive option for processing in space-based systems due to their inherent advantages in performance, scalability, energy efficiency, size, and cost, but with them come challenges in attaining optimal performability. This presentation will highlight research activities at the University of Florida from two recent projects on this path, the NASA Dependable Multiprocessor (DM) developed at Florida and Honeywell, and the hybrid fault tolerance (HFT) framework of CHREC.

The NASA Dependable Multiprocessor project features a multitude of system- and application-level techniques for fault tolerance to protect the system from SEU-induced errors, much of which is applicable to the needs of space multicore processing. The DM system consists of primary and secondary RadHard system controllers and a suite of COTS-based, data-processing boards featuring PPC, AltiVec, and FPGA processors, all connected through Gigabit Ethernet, similar to many traditional supercomputing clusters. Fault tolerance in DM can adapt to environmental radiation conditions, with an array of disparate and flexible modes, including SIFT at the highest level via high-availability middleware with manager and agent processes running on RadHard and COTS microprocessor technologies, respectively, along with a variety of modes for fault tolerance operating underneath, many available in either spatial or temporal form. The high-availability middleware manages the health and status of multiple concurrent jobs, taking corrective action when necessary. Application-level communication between nodes is facilitated through the use of Fault-Tolerant Embedded MPI (FEMPI), allowing for the recovery of a parallel job without the need to completely restart the application. Application-level techniques for fault tolerance, such as replication, algorithm-based fault tolerance, and checkpoint/rollback are also featured and examined with a range of applications including LU decomposition, 2D-FFT, synthetic aperture radar, and hyperspectral imaging.

The hybrid fault tolerance or HFT framework is a new component in an on-going research project of CHREC entitled F6-09, Reconfigurable and Hybrid Fault Tolerance. One of the tasks in current work on HFT that is applicable to the needs of space multicore processing is a new method of protecting microprocessor cores from SEU-induced errors via automated source-to-source (S2S) translation with high productivity. Replication embedded in the application program instructs the processor to perform redundant calculations. These calculations can then be compared and/or voted upon to detect and/or correct errors

[1] This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422.

automatically. Through the use of S2S translation, we present a method of performing this replication through a high-level language (in this case, C). A translator would take an input program source code and output a fault-tolerant version of the same program (with very little or no user intervention) that could then be compiled with any valid compiler. Additionally, we are exploring methods for software-based fault injection to examine the reliability of various microprocessor devices and the efficiency of newly proposed FT methods for them. Our simple, portable fault injector (SPFI) allows us to emulate SEUs by injecting errors directly into processor registers of each processor core. The injector software can work with any system that supports the GNU debugger, making the tool highly portable. This approach allows us to quickly inject faults, test behavior, and estimate error rates expected without the need for expensive radiation testing at each step.

Adaptive Software-based Fault Tolerance for Space Multicore Processing

CHREC
NSF Center for High-Performance Reconfigurable Computing

Space Multicore Workshop

UF UNIVERSITY of FLORIDA
Virginia Tech
VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
BYU
BRIGHAM YOUNG UNIVERSITY
THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON DC

Adam Jacobs
Grzegorz Cieslewski
Research Students
University of Florida

Dr. Alan D. George
Professor of ECE
University of Florida

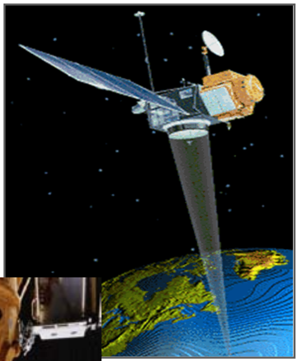
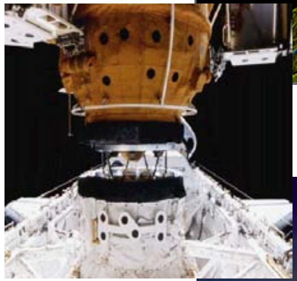
July 19-21, 2009

Outline

- Overview
  - Taxonomy of fault tolerance
- Dependable Multiprocessor
  - Hardware Infrastructure
  - Software Infrastructure
  - Possible DM Applications
- Hybrid Fault Tolerance (HFT)
  - Goals, motivations, and challenges
  - Source-to-Source Translation
- Simple, Portable, Fault Injector (SPFI)
- Conclusions

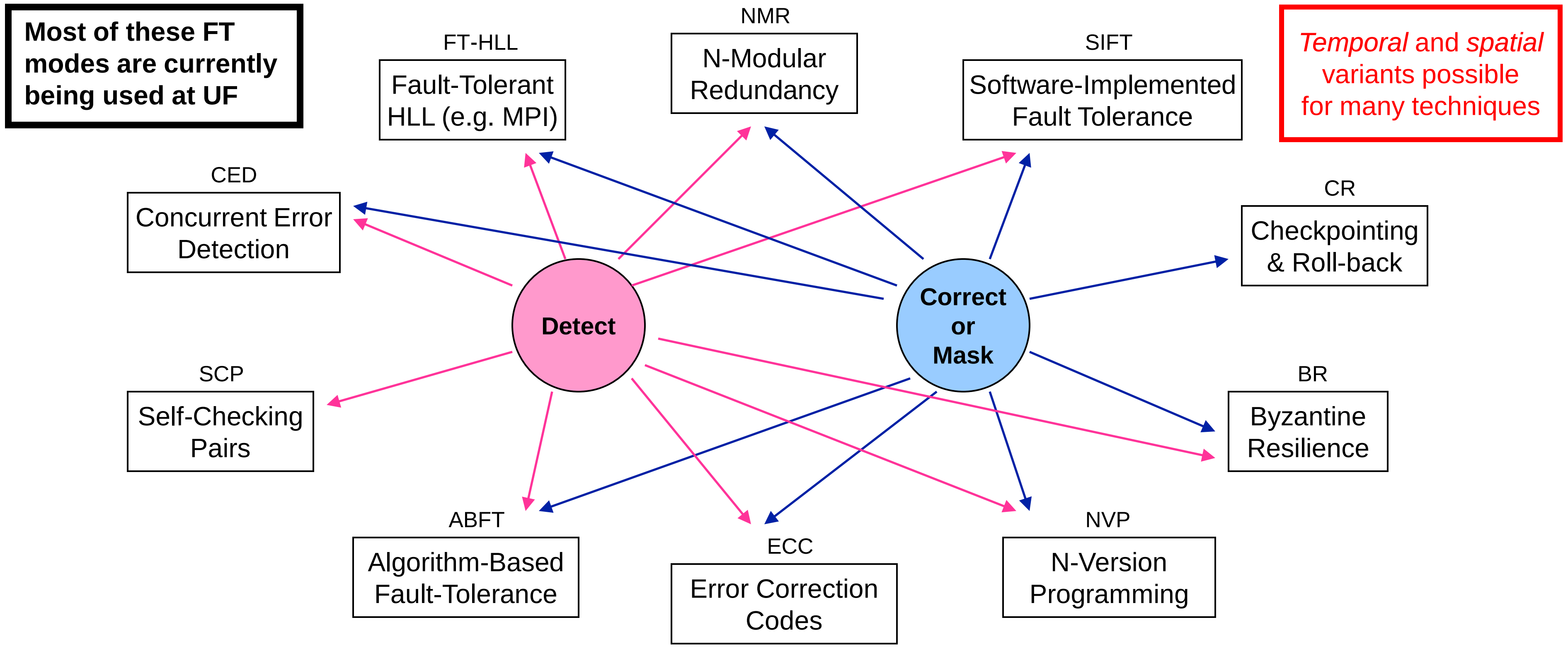
Overview

- What is advanced space computing?
  - New concepts, methods, and technologies to enable and deploy high-performance computing in space – for an increasing variety of missions and applications
- Why is advanced space computing vital?
  - On-board data processing
    - Downlink bandwidth to Earth is extremely limited
    - Sensor data rates, resolutions, and modes are dramatically increasing
    - Remote data processing from Earth is no longer viable
    - Must process sensor data where it is captured, then downlink results
  - On-board autonomous processing & control
    - Remote control from Earth is often not viable
    - Propagation delays and bandwidth limits are insurmountable
    - Space vehicles and space-delivered vehicles require autonomy
    - Autonomy requires high-speed computing for decision-making
- Why is it difficult to achieve?
  - Cannot simply strap a Cray to a rocket!
    - Hazardous radiation environment in space
    - Platforms with limited power, weight, size, cooling, etc.
    - Traditional space processing technologies (RadHard) are severely limited
  - Potential for long mission times with diverse set of needs
    - Need powerful yet adaptive technologies
    - Must ensure high levels of reliability and availability

Taxonomy of Fault Tolerance

- First, let us define various possible modes/methods of providing fault tolerance (FT)
  - Many other options beyond simply throwing triple-modular redundancy (TMR) at the problem
  - Software FT vs. hardware FT concepts largely similar, differences only at implementation level
  - Radiation-hardening not listed, falls under "prevention" as opposed to detection or correction

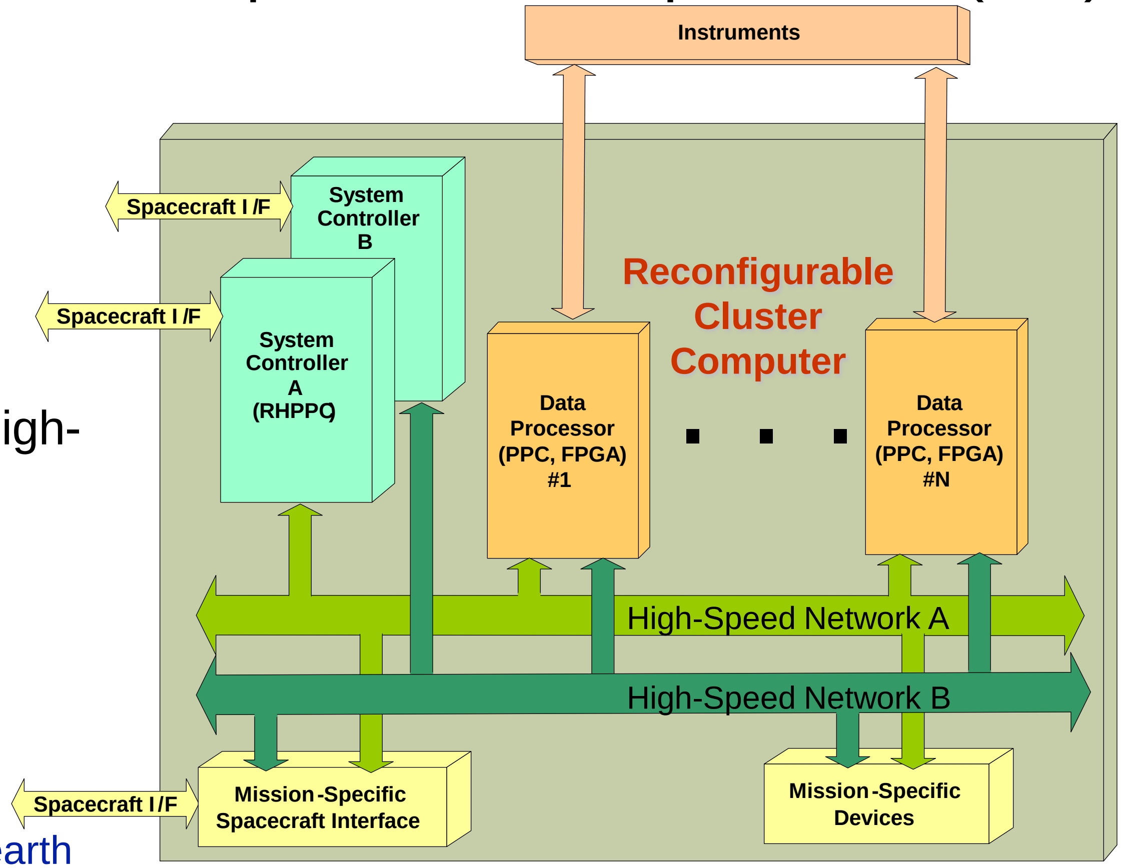


Most of these FT modes are currently being used at UF
FT-HLL
Fault-Tolerant HLL (e.g. MPI)
NMR
N-Modular Redundancy
SIFT
Software-Implemented Fault Tolerance
Temporal and *spatial* variants possible for many techniques
CED
Concurrent Error Detection
CR
Checkpointing & Roll-back
Detect
Correct or Mask
SCP
Self-Checking Pairs
BR
Byzantine Resilience
ABFT
Algorithm-Based Fault-Tolerance
ECC
Error Correction Codes
NVP
N-Version Programming

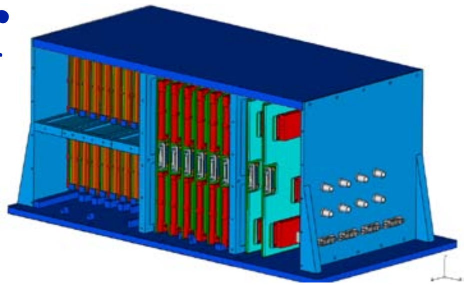
NASA/Honeywell/UF Project

- 1st Space Supercomputer
  - Funded by NASA NMP
  - In-situ sensor processing
  - Autonomous control
  - Speedups of 100× to 1000×
  - First fault-tolerant, parallel, reconfigurable computer for space
- Infrastructure for fault-tolerant, high-speed computing in space
  - Robust system services
  - Fault-tolerant MPI services
  - Application services
  - FPGA services
  - Standard design framework
  - Transparent API to resources for earth & space scientists

NASA Dependable Multiprocessor (DM)


Instruments
Spacecraft I/F
System Controller B
Spacecraft I/F
System Controller A (RHPPC)
Reconfigurable Cluster Computer
Data Processor (PPC, FPGA) #1
. . .
Data Processor (PPC, FPGA) #N
High-Speed Network A
High-Speed Network B
Spacecraft I/F
Mission-Specific Spacecraft Interface
Mission-Specific Devices

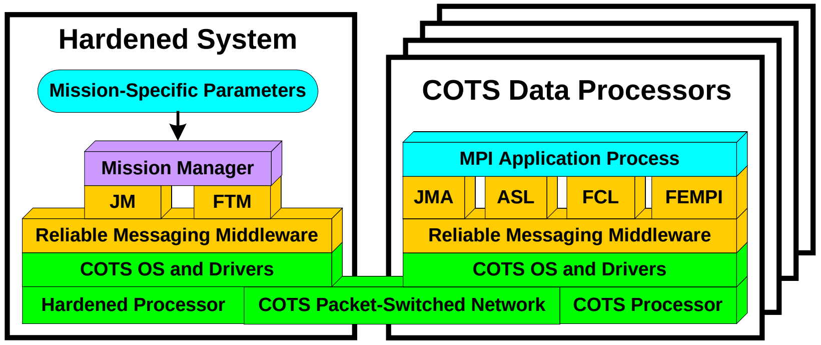
Dependable Multiprocessor



- DM System Architecture
  - Dual system controllers
    - Redundant radiation-hardened PPC boards
    - Monitor data processors' health and communicate with spacecraft
  - Data processing engines
    - High-performance, low-power COTS SBCs running Linux
    - PowerPC with AltiVec capabilities
    - Optional FPGA co-processor for additional performance
    - Scalable to 20 data processing nodes
  - Redundant Interconnect
    - Dual GigE connections
    - Automatically switch networks when error is detected

- DM Middleware (DMM)
  - FT System Services
    - Manages status and health of multiple concurrent jobs
  - FT Embedded MPI (FEMPI)
    - Lightweight subset of MPI
    - Allows fault recovery without restarting an entire parallel application
  - Application & FPGA Services
    - Commonly used libraries such as ATLAS, FFTW, GSL
    - Simplified, generic API for FPGA usage
  - High-Availability Middleware
    - Framework used to enable health monitoring of cluster

DMM Components



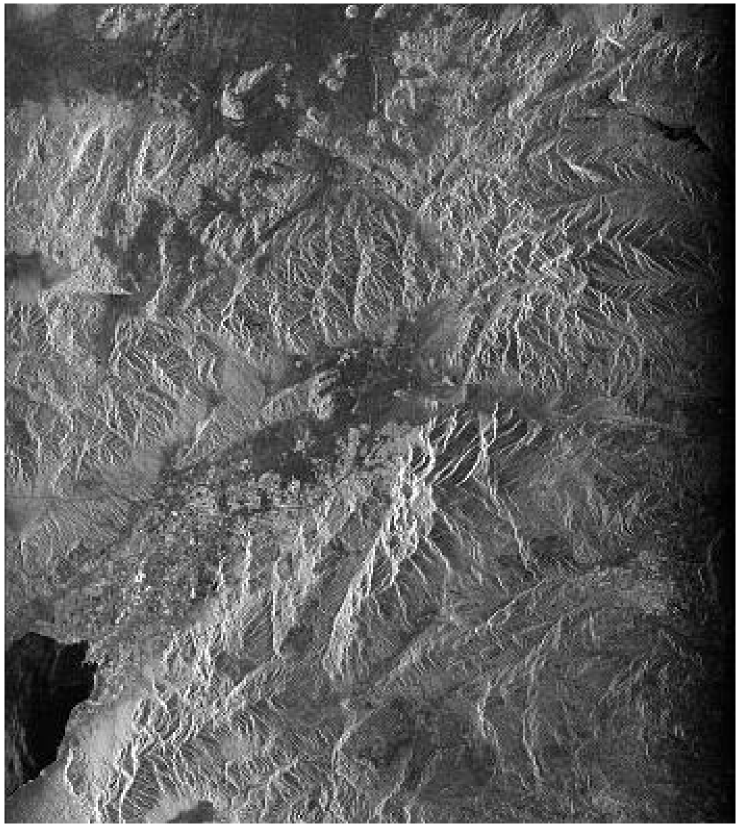
JM – Job Manager
JMA – Job Manager Agent
FTM – Fault Tolerance Manager

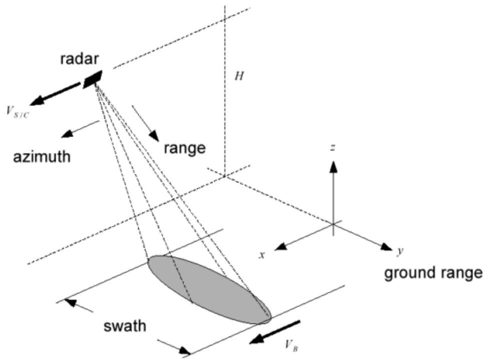
FEMPI – Fault-Tolerant Embedded MPI
ASL – Application Services Library
FCL – FPGA Coprocessor Library

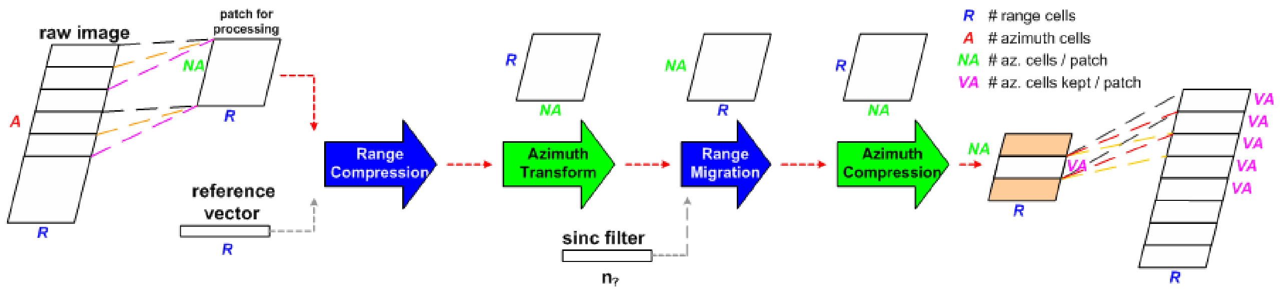
- Mission Manager (MM)
  - Controls high-level job deployment
  - Facilitates replication of lower-level jobs
    - Spatial or temporal replication
    - Automatically compares and validates outputs
  - Monitors real-time deadlines
  - Enables roll-forward / roll-back when faults occur
- Job Manager (JM)
  - Controls low-level job deployment and scheduling across system
- FT Manager (FTM)
  - Manages low-level system faults (node crash, job crash)
- JM Agent (JMA)
  - Deploys and monitors programs on given node
  - Provides application "heartbeat" to system controller
- Mass Data Store (MDS)
  - Provides reliable centralized data services
  - Enables reliable checkpointing

Space Applications

- Synthetic Aperture Radar (SAR)
  - Used to form high-resolution images of Earth's surface from moving platform in space
  - Patch-based processing with significant amount of overlap between patch boundaries
- Parallelizable on multiple levels of granularity, possible without need for *any* inter-processor communication (one patch per node)
- 2-dimensional data set, can range in size from several hundred Megabytes to Gigabytes
- Data set *not* significantly reduced through course of application
- Highly amenable to ABFT

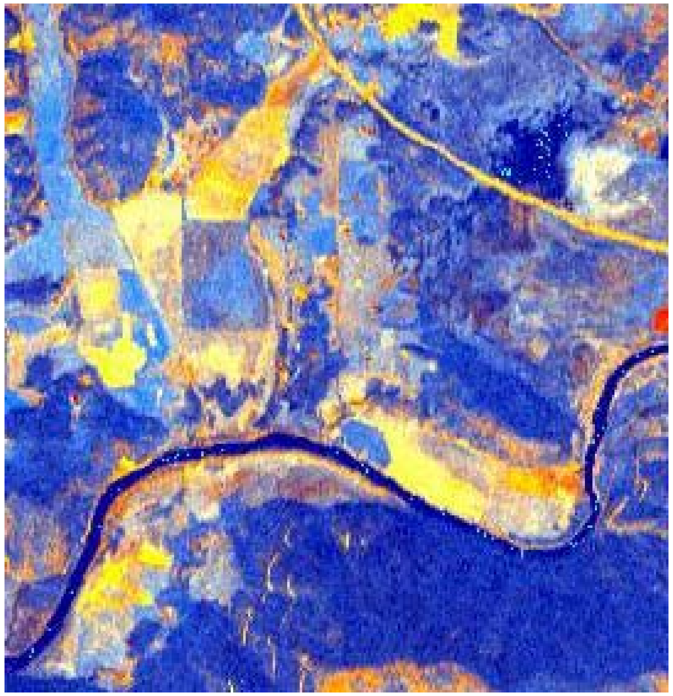


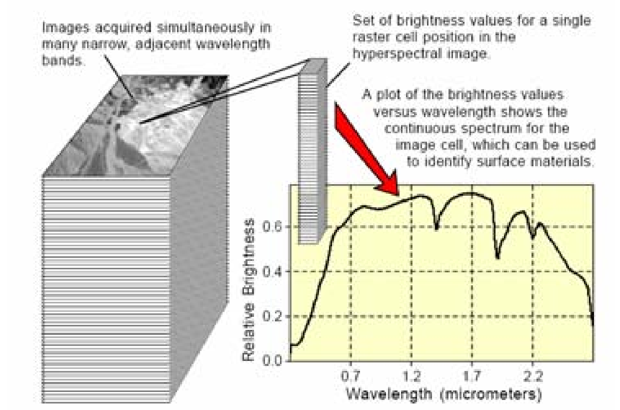


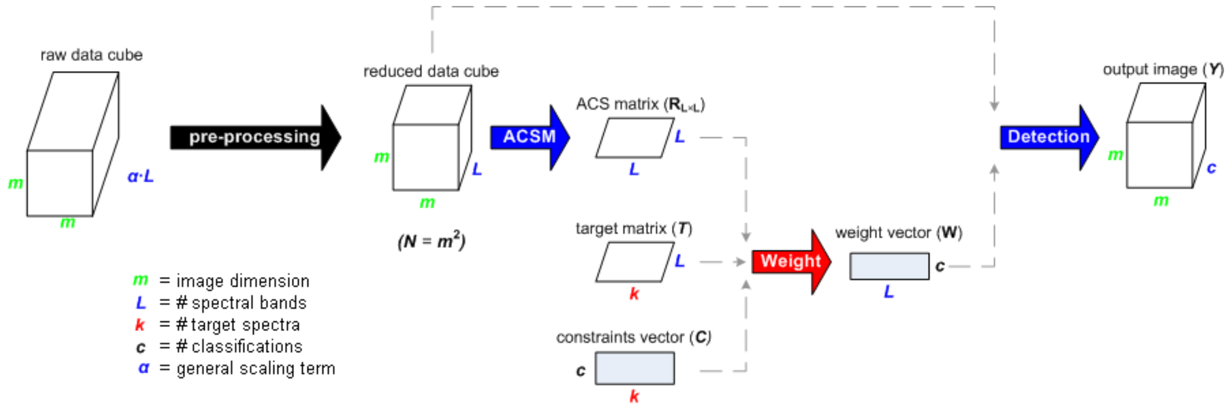


Space Applications

- Hyperspectral Imaging (HSI)
  - Uses traditional beamforming techniques to perform coarse-grained classification on hyperspectral images
  - Adjustable to enable real-time processing
- Mostly embarrassingly parallel, exception being weight computation (shown in red below)
- 3-dimensional data set
- Data set reduced through course of application
- Auto-correlation sample matrix (ACSM) calculation and beamforming (detection) amenable to ABFT
  - Suggest NMR for weight computation (weight)



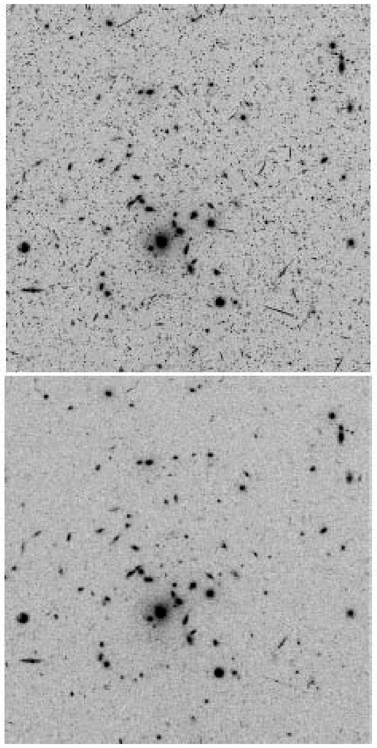

Images acquired simultaneously in many narrow, adjacent wavelength bands.
Set of brightness values for a single raster cell position in the hyperspectral image.
A plot of the brightness values versus wavelength shows the continuous spectrum for the image cell, which can be used to identify surface materials.
Relative Brightness
0.0
0.2
0.4
0.6
0.7
1.2
1.7
2.2
Wavelength (micrometers)



m = image dimension
L = # spectral bands
k = # target spectra
c = # classifications
α = general scaling term

Space Applications

- Cosmic Ray Elimination
  - Uses image processing techniques to remove artifacts caused by cosmic rays
  - Image shows pre- and post-processed versions of a Hubble Telescope observation
- Images are highly parallelizable, with minimal communication necessary
- Main computation: median filtering
  - Fault-tolerant median filter developed
- Other portions of algorithm replicated by hand or S2S translator



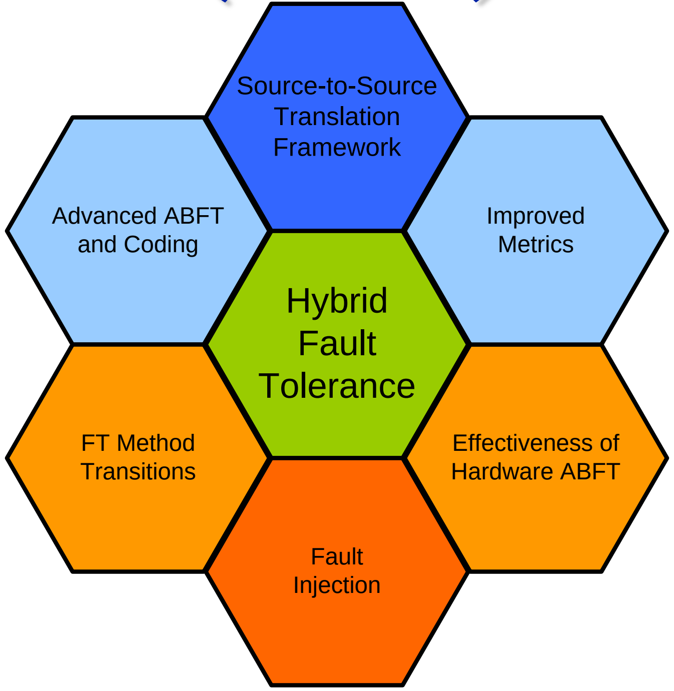
- Other aerospace-related application kernels
  - Space-Time Adaptive Processing (STAP)
  - Ground Moving Target Indicator (GMTI)
  - Airborne LIDAR
  - Digital Down Conversion (DDC)
  - PDF Estimation

Application to Multicore Systems

- Original DM system designed around traditional single-core processors
  - However, there are no limitations on architectures for data processor nodes
    - Some nodes have special processing units (FPGAs, Altivec, etc.)
  - Multicore data processors allow for decreased area footprint and high performance
- Two possible use cases for multicore systems
  - Use one core for communication with other processors
    - “Master” core has a JMA to communicate health status to the system controller
    - “Slave” cores communicate health status to “Master”
  - Every core is able to communicate when needed
    - Every core has a JMA for communicating with the system controller
- Necessary modifications
  - Update Job Manager to enable efficient scheduling on a multicore system
  - FT Manager must recognize locality
    - An error on multiple cores may represent a single fault in a processor

Hybrid Fault Tolerance (HFT)

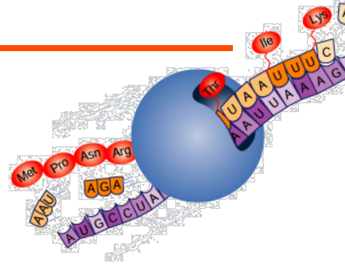
- Motivations for HFT
  - One FT technique is not always suitable for each application phase
    - Mixing many techniques improves performance
  - Soft/hard-core processors used in embedded need protection from SEUs
    - Radiation hardened parts are not always the best solution
    - Temporal techniques (e.g. repetition) can be as effective as spatial



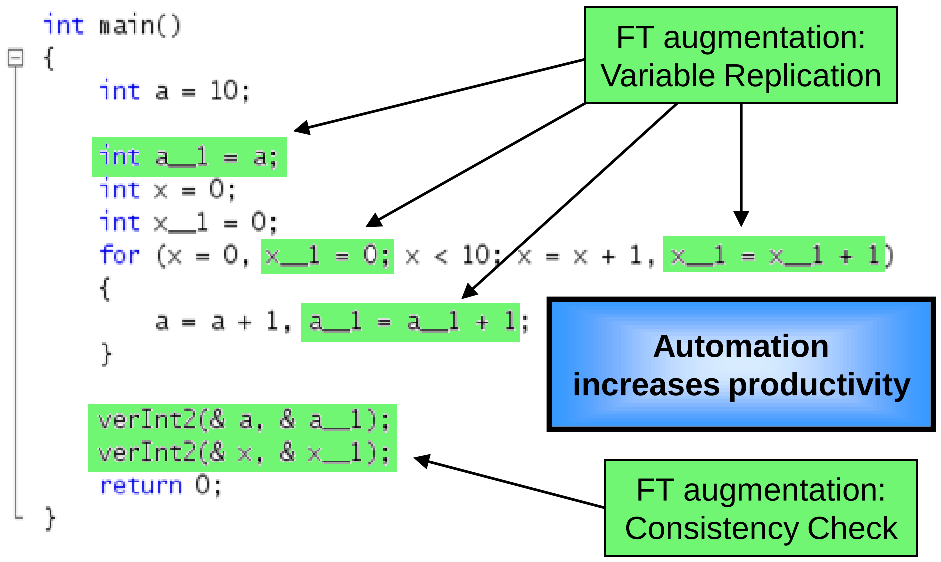
- Source-to-Source Translation Framework
  - Reliability of a program can be significantly improved by transforming the source code before compilation
    - Decreased overhead
    - More control over FT techniques

- Fault Injection
  - Using particle accelerators characterize FT application during development is expensive, lacks coverage and is time consuming
  - Portable tools are needed to accelerate this process by emulating faults through software fault injection

HFT through S2S Translation



- Most science applications are inherently non-fault-tolerant
  - Requires SIFT framework to handle failures due to SEU
  - Most of those failures can be mitigated by properly modifying the source code of the application



FT augmentation: Variable Replication
Automation increases productivity
FT augmentation: Consistency Check

```
int main()
{
    int a = 10;

    int a__1 = a;
    int x = 0;
    int x__1 = 0;
    for (x = 0, x__1 = 0; x < 10; x = x + 1, x__1 = x__1 + 1)
    {
        a = a + 1, a__1 = a__1 + 1;
    }

    verInt2(& a, & a__1);
    verInt2(& x, & x__1);
    return 0;
}
```

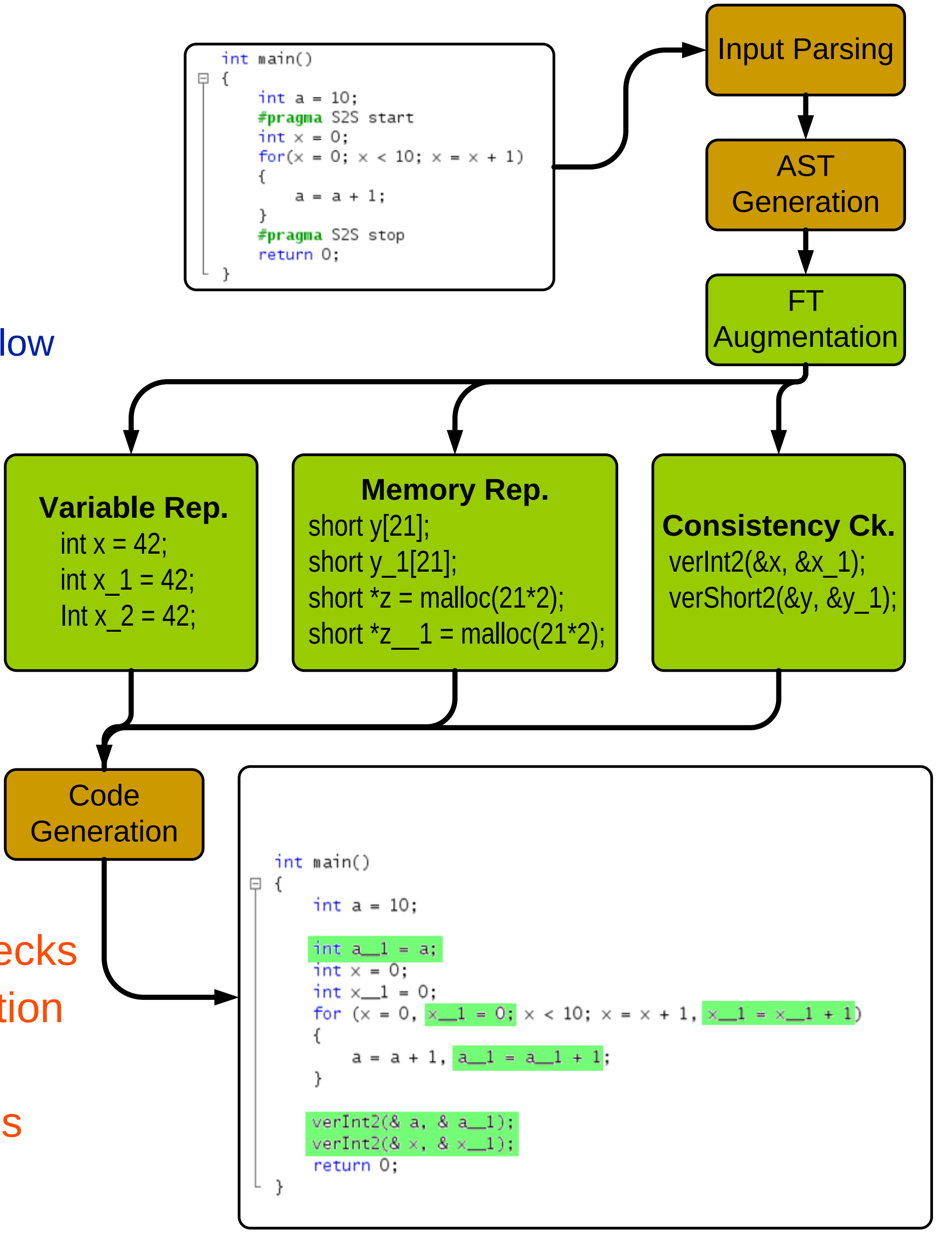
- Source-to-Source Translator
  - FT S2S translator takes HLL source code as input, outputs FT-augmented HLL code
    - Compiler-independent
- Goals of S2S Tool
  - Accepts C source files as inputs
  - Generates fault tolerant C source code
  - Use fine- and coarse-grain NMR approach to improve reliability and dependability
  - Provides means of control flow checking (CFC) through software
  - Minimizes number of undetected errors
    - Error detection is very important do ensure dependability of the output
    - Error correction useful for performance (error recovery) but not always necessary
  - Can be used on embedded and soft-core processors as well as on standard platforms

CHREC NSF Center for High-Performance Reconfigurable Computing

UF UNIVERSITY of FLORIDA
Virginia Tech
BYU BRIGHAM YOUNG UNIVERSITY
THE GEORGE WASHINGTON UNIVERSITY

S2S Tool

- Software Structure
  - Code Parsing and AST Generation
  - Multiple AST transformations
    - Stacking multiple transformations will allow for customizable levels of FT
  - Code Generation
- Tool is currently in development
  - Written in Java for portability
  - Uses popular ANTLR 3.1 parser/lexer generator for grammar generation
- Envisioned Transformation options
  - Variable and function replication
  - Memory duplication or memory encoding
  - Memory and Variable Consistency Checks
  - Post- and pre-branch condition evaluation
  - Block protection
  - Advanced ABFT and coding techniques



```
int main()
{
    int a = 10;
    #pragma S2S start
    int x = 0;
    for(x = 0; x < 10; x = x + 1)
    {
        a = a + 1;
    }
    #pragma S2S stop
    return 0;
}
```

Variable Rep.
```
int x = 42;
int x_1 = 42;
Int x_2 = 42;
```

Memory Rep.
```
short y[21];
short y_1[21];
short *z = malloc(21*2);
short *z__1 = malloc(21*2);
```

Consistency Ck.
```
verInt2(&x, &x_1);
verShort2(&y, &y_1);
```

```
int main()
{
    int a = 10;

    int a__1 = a;
    int x = 0;
    int x__1 = 0;
    for (x = 0, x__1 = 0; x < 10; x = x + 1, x__1 = x__1 + 1)
    {
        a = a + 1, a__1 = a__1 + 1;
    }

    verInt2(& a, & a__1);
    verInt2(& x, & x__1);
    return 0;
}
```

Fault Injection Results

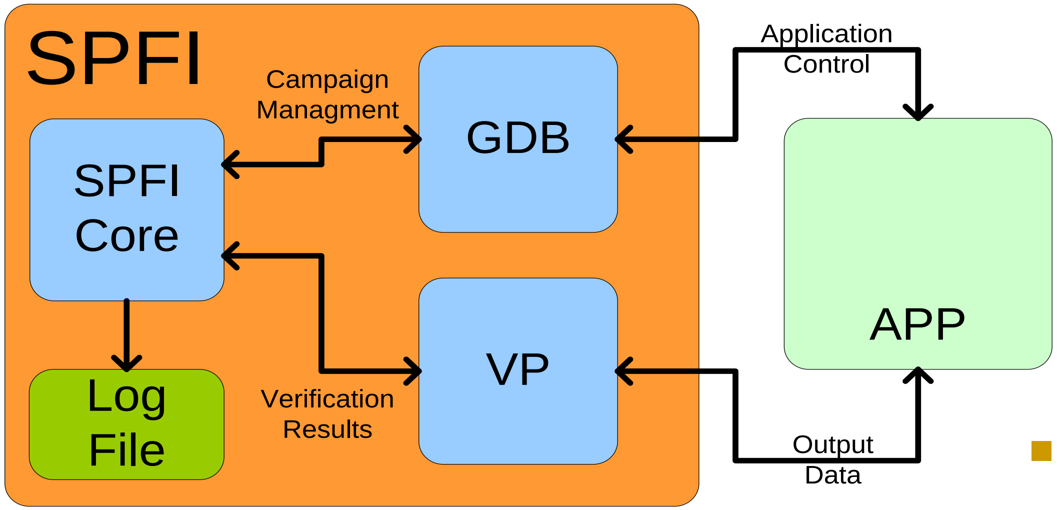
- Preliminary case studies
  - Two algorithms which showcase computational density and complex control flow patterns
  - 10000 faults injected for each test
  - LU Decomposition (SCP)
    - Overhead of 130% for 500x500 size matrix
    - Only 2.65% errors are undetected
  - Matrix Multiply (partial TMR)
    - Correction is not supported for all constructs due to temporary tool limitations
    - Overhead of 162% for 400x400 size matrix
    - Only 4.07% errors are undetected
  - Adaptation for ABFT in future

- Testing Methodology
  - Faults are randomly distributed over runtime of the program
  - Output of each run is compared to established gold standard
  - Targeted registers
    - Control Register
    - Floating Point Registers (F0-F31)
    - Integer Registers (R0-R31)

Case Study	Number of Injections	Execution Overhead	Undetected Errors
LU Decomposition with no FT	10000	--	15.24%
LU Decomposition with SCP	10000	~130%	2.65%
Matrix Multiply with no FT	10000	--	10.24%
Matrix Multiply with partial TMR	10000	~162%	4.07%

SPFI – Simple Portable Fault Injector

- Motivations
  - Alternatives to expensive radiation testing are needed
  - Current tools are rigid and tied to specific platforms
- SPFI
  - New fault injector designed with simplicity and portability in mind
  - Employs GDB to provide portable solution for uP fault injection



- SPFI System Components
  - SPFI Core
    - Logging Engine
    - Campaign Manger
    - GDB Controller
    - Fault injection
  - VP – Verification Program
    - User plug-in to verify the results of the injections
  - GDB – GNU Debugger
    - Memory and register manipulation
    - Breakpoint management
    - Application control
  - APP
    - User provide application subject to fault injection
- Portable
  - Any system that supports standard GDB

SPFI - Injection Modes

- Injection Modes
  - Breakpoint based
    - Breakpoints are set in predetermined locations in the program
    - Injections occurs when breakpoint is reached
    - Repeatable results
    - Breakpoints can be set anywhere in writable program memory (even shared objects)
  - Timer based
    - Breakpoint is issued after predetermined amount of time has elapsed
    - Closely resembles how SEU's occur
    - Every test is different even if the parameters are the same due to granularity of the timing mechanisms and communication latencies
    - Knowledge where the error is injected is available through GDB
- Injections Targets
  - Registers – Integer, Floating Point, Control, Altivec – More likely target for SEU's as registers lack protection
  - Memory – Anywhere within processes space – Less likely target for SEU's as memory is usually protected by hardware FT schemes

Application to Multicore Systems

- Hybrid Fault Tolerance
  - Multitude of FT methods are available
  - Both spatial and temporal methods can be easily used
  - Different cores could use different methods to compute same result
- Source-to-Source Translation
  - If standard ANCI C is used on the platform S2S tool could provide rapid way for providing FT for single and multcore applications
    - Adaptations would be required in order to handle special communication functions
- Fault Injection
  - Many multicore systems use GDB derivatives as a method of debugging
  - SPFI could be adapted to provide FI capabilities
    - Multicore applications
    - Ability to select which or how many cores are targeted

Conclusions

- Fault tolerance for space should be more than RadHard components & spatial TMR designs
  - Fixed worst-case designs extremely limited in perf/Watt
  - Instead, many FT methods & modes can be exploited
  - Adaptive systems that react to environmental changes
  - COTS featured inside critical performance path
  - RadHard for FT management, outside critical perf. path
- UF active on many space-related FT issues
  - NASA Dependable Multiprocessor, CHREC HFT F6-09
  - Modes: SIFT, ABFT, S2S, FEMPI, CR, CED, etc.
  - Devices: PPC/AV, FPGA, FPOA, Tilera, ElementCXi, etc.
  - Space apps: HSI, SAR, LIDAR, GMTI, CRE, et al.