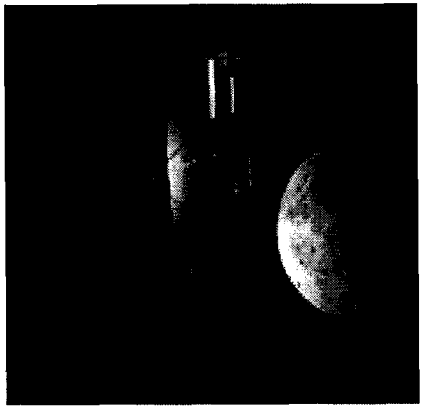
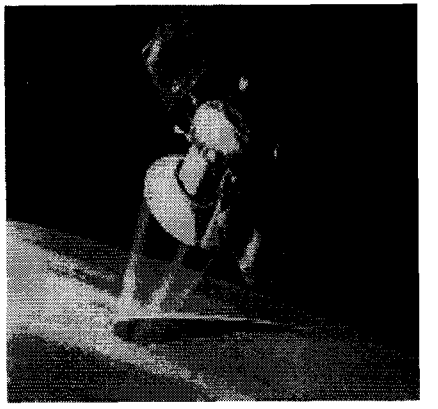
Application-Based Fault Tolerance for Spaceborne Applications

(A Selective History of the Remote Exploration and Experimentation (REE) Project)

Autonomous Robotic Vehicles



Deep Space Exploration



High Data Rate Instruments

Daniel S. Katz
(and many others; see references)

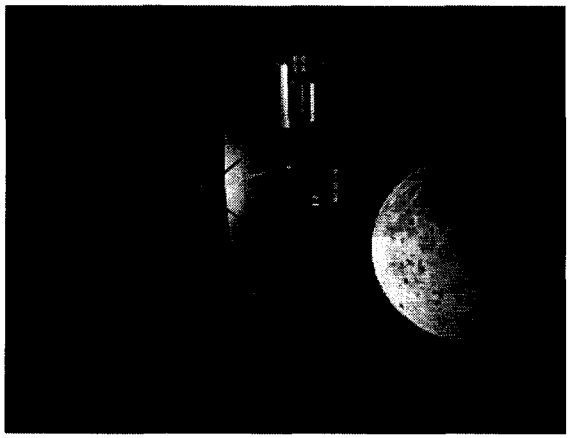
Jet Propulsion Laboratory
California Institute of Technology

Daniel.S.Katz@jpl.nasa.gov

REE Vision

Vision:

Move Earth-based Scalable Supercomputing Technology into Space



Background

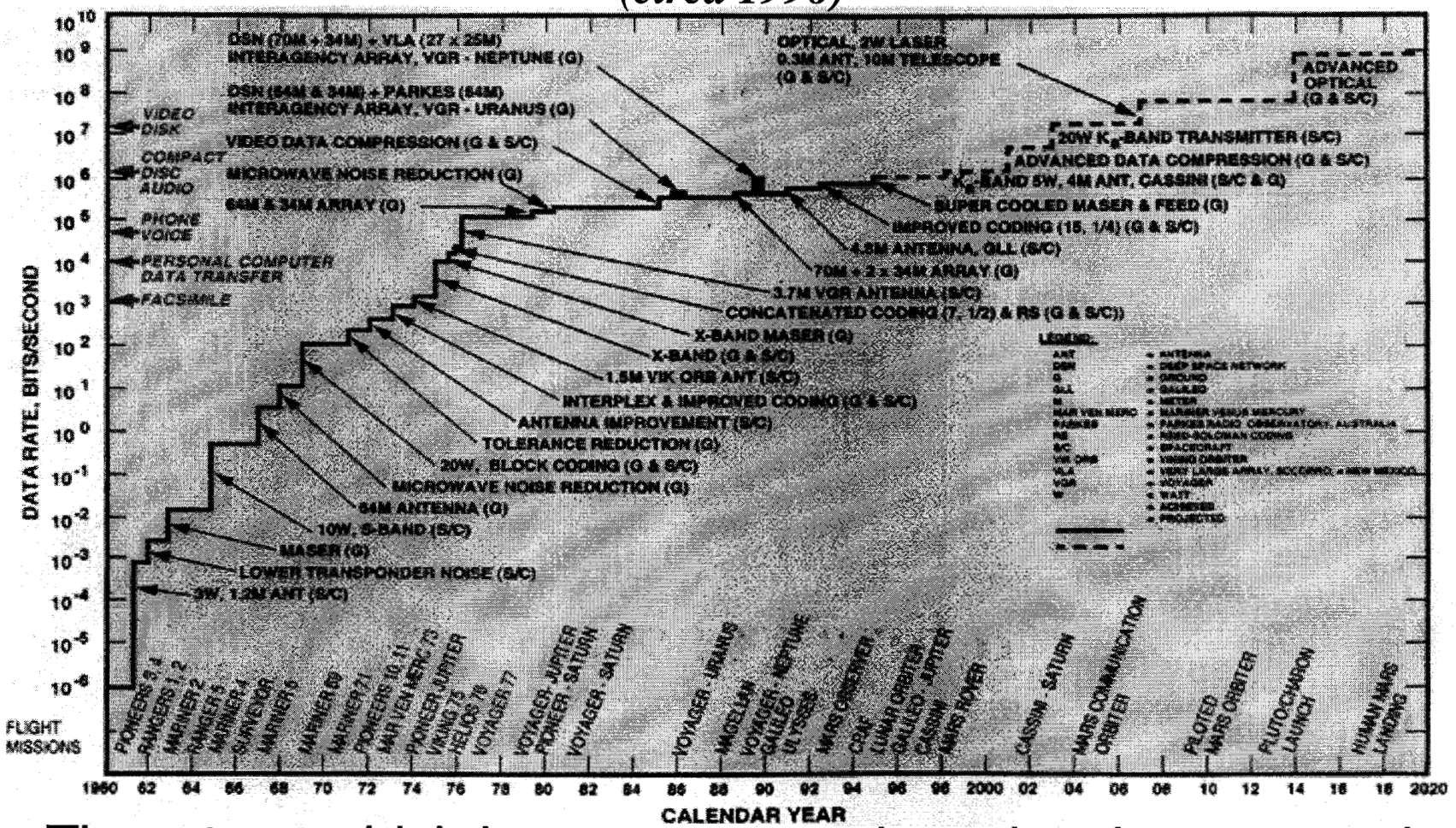
- **Funded by Office of Space Science (Code S) as part of NASA's High Performance Computing and Communications Program**
- **Started in FY1996**
- **Funding zeroed in FY2002**

1996 Intended Outcome: REE Impact on NASA and DOD Missions by FY03

Faster -	**Fly State-of-the-Art Commercial Computing Technologies within 18 month of availability on the ground**
Better -	**Onboard computer operating at > 300MOPS/watt scalable to mission requirements *(> 100x Mars Pathfinder power performance)***
Cheaper -	**No high cost radiation hardened processors or special purpose architectures**

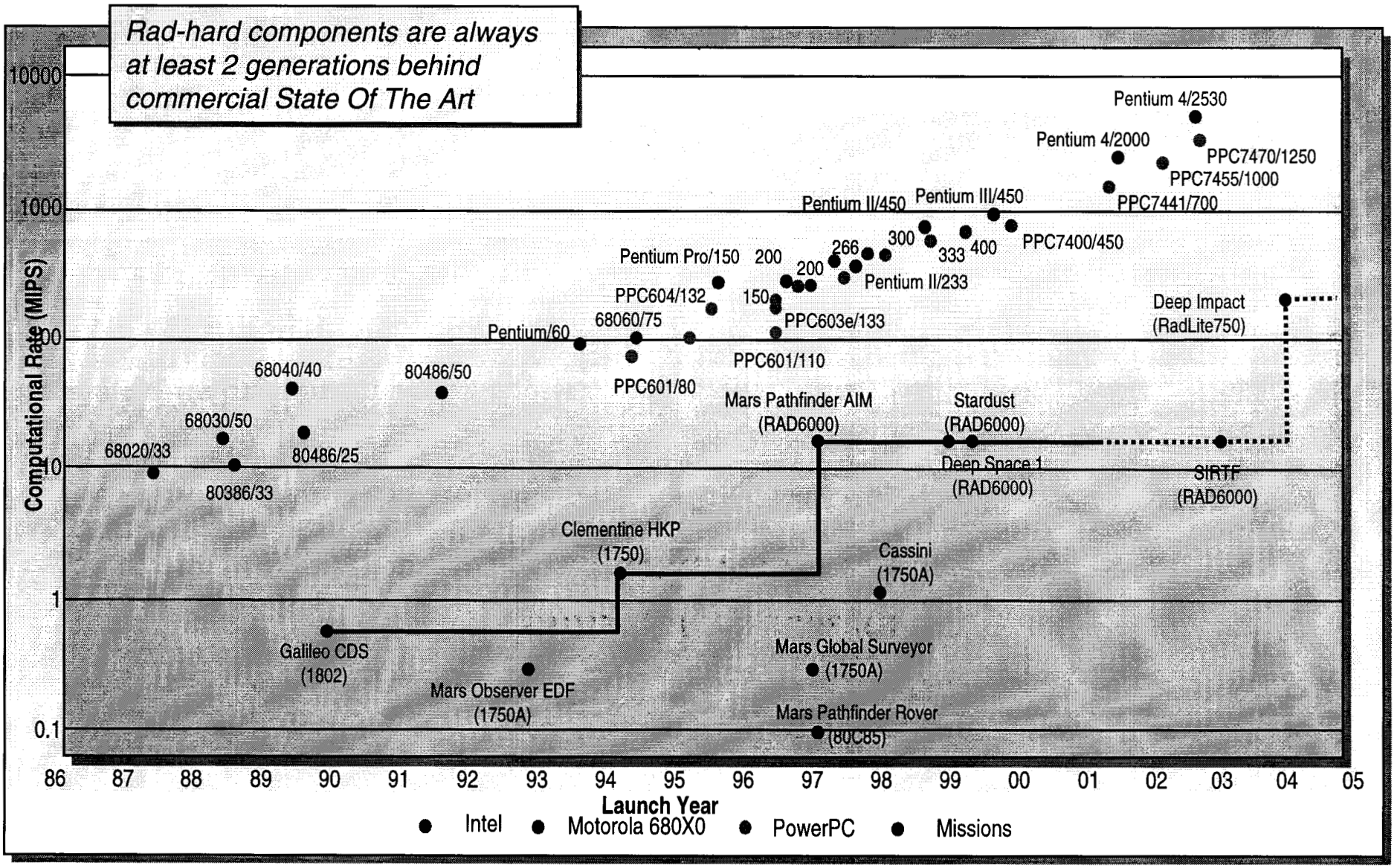
Trends in Space to Earth Bandwidth

"Equivalent Data Rate at Jupiter Distance"
(circa 1996)



The rate at which instruments produce data is now growing much faster than rate at which data can be sent to Earth

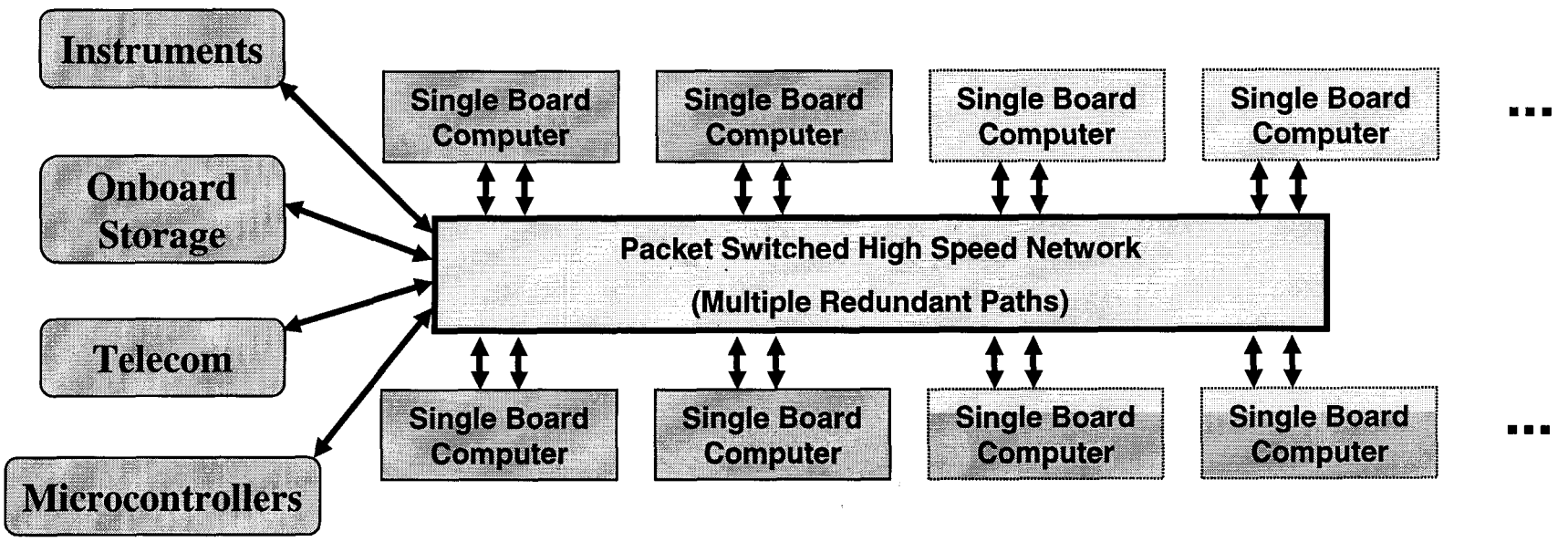
Space Flight Avionics & Microcomputer Processor History



REE Baseline Architecture

Homogeneous, scalable, parallel, distributed memory embedded computer system

- Replicated assets as the basis for fault tolerance, graceful degradation
- Fault containment boundaries
- Maximum leverage of commercial offerings
- Scalable to mission computing requirements, mass & power limitations
- Towards a "Spacecraft as a Network of Devices" concept



Understanding COTS Systems in a Radiation Environment

Preliminary Fault Model Completed for PowerPC 750 Based Onboard Computer

Model takes gate, register fault rates measured by testing and predicts total board-level faults induced by radiation in a specific environment

Ground-based radiation testing completed for PowerPC 750

- **TID > 50 KRads**

Predicted Ave Hours/Fault with 100 Mil Al shielding, nominal solar activity

Environment	Deep Space	600km-98° Earth Orbit	Surface of Mars	600km-28° Earth Orbit
Per Node	5	3	50	5

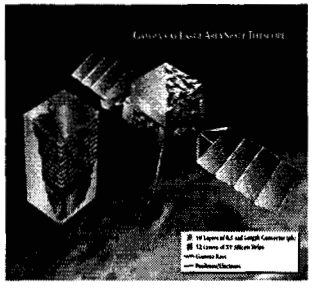
We have measured Total Dose - high enough for most NASA missions and predicted Fault Rate - low enough to be handled by software

Prototyping NASA On-Board Applications

We asked scientists: "What would you do with 100x more onboard computing power?"

Next Generation Space Telescope - John Mather/GSFC

- Correcting the cosmic ray impacts on the CCDs
- Autonomous control and optimization of the adaptive optics



Gamma-ray Large Area Space Telescope

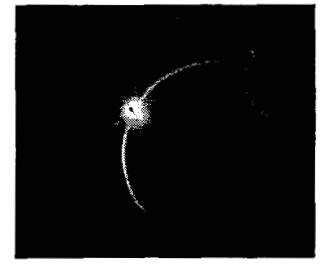
- Peter Michelson/Stanford
- Toby Burnett/U Washington
  - Onboard cosmic ray rejection
  - Real time gamma ray burst identification

Orbiting Thermal Imaging Spectrometer - Alan Gillespie/U Washington

- Onboard atmospheric corrections, radiance calculations

Mars Rover Science - R. Steve Saunders/JPL

- Autonomous optimal terrain navigation
- Autonomous field geology



Solar Terrestrial Probe Program - Steve Curtis/GSFC

- Constellation/formation flying missions to probe the Sun-Earth Connection
- Onboard plasma moment calculations, multi-instrument cross correlations, autonomous operations

Science Applications on REE Testbed

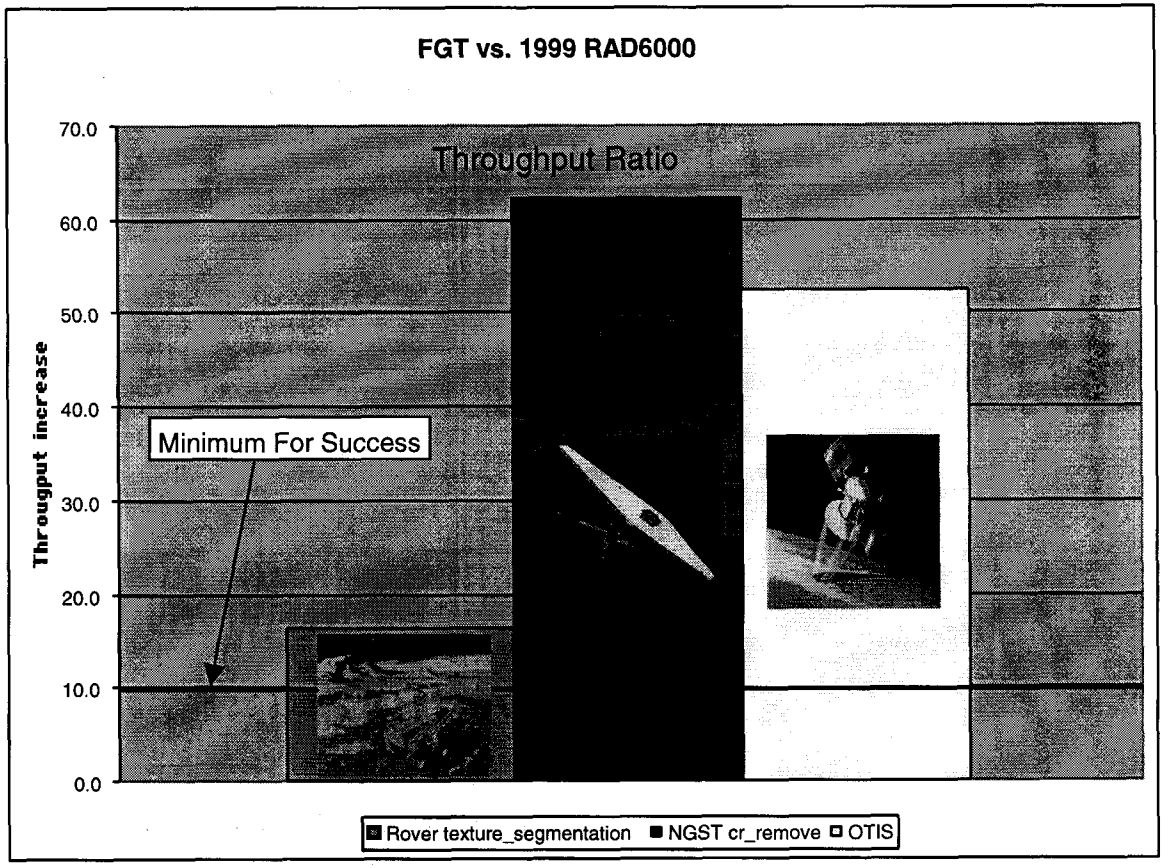
Milestone Metrics: 3 applications, 10x throughput improvement (per processor) over 1999 RAD6000, sqrt(n) scalability, 50% ideal speedup

Applications:
- Rover Image Texture Analysis
- NGST Image Cosmic Ray Removal
- Orbiting Thermal Imaging Spectrometer processing

Throughput
- Exceeded metric in all cases

Scaling, Speedup
- Exceeded metric in all cases, approaching ideal limits

Demonstrated that:
- COTS in space offers enormous throughput improvement over Rad-hard Onboard science applications can run efficiently on multiple processors (demo to 40 procs)

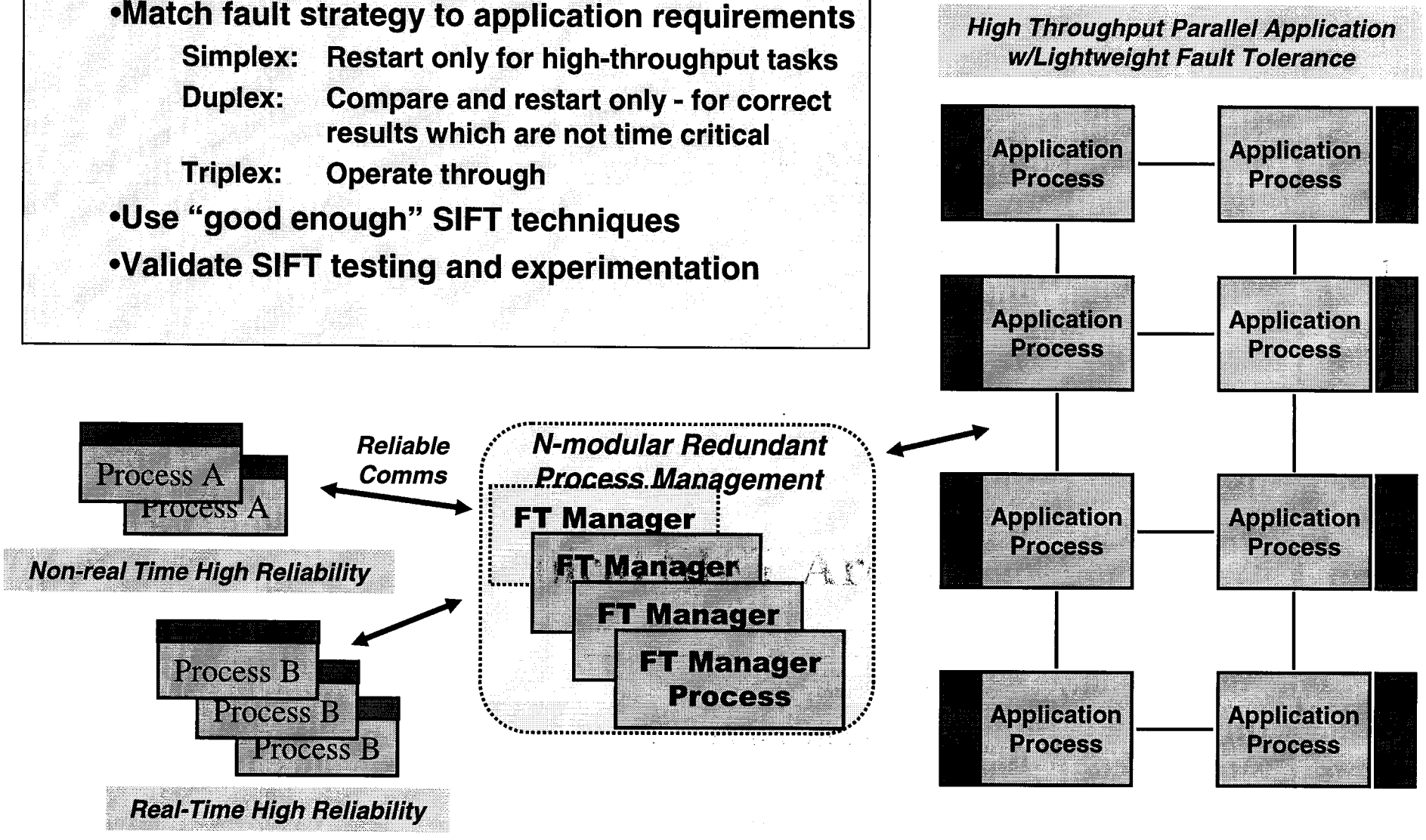


REE Testbed: 20 dual-processor PPC750 cards, connected via Myrinet with a custom FPGA communications processor, contained by two VME racks

Hierarchical SIFT Architecture

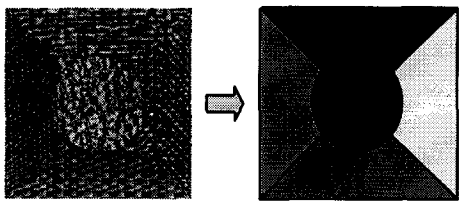
Approach:

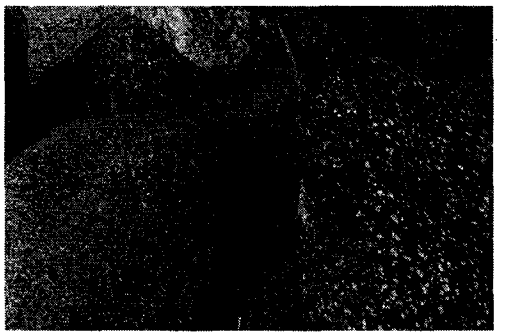
- **Match fault strategy to application requirements**
  - **Simplex:** Restart only for high-throughput tasks
  - **Duplex:** Compare and restart only - for correct results which are not time critical
  - **Triplex:** Operate through
- **Use "good enough" SIFT techniques**
- **Validate SIFT testing and experimentation**

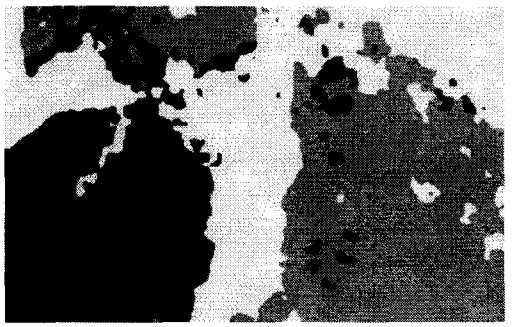


Fault Injection: Texture Application

Goal: Segment image into homogeneous regions
Motivation: Texture contains valuable geological information





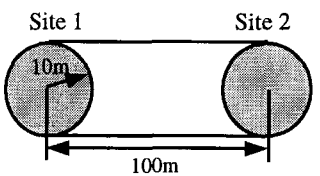


Igneous **Metamorphic**

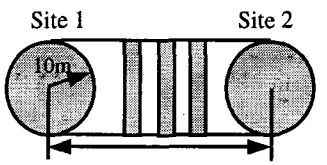
Both rocks have the same mineralogical content (a spectrometer would not see anything different), but have undergone distinct geological processing

Science Benefits:

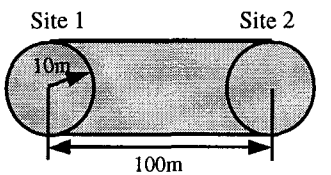
- **Rover can examine region it traverses**
  - **Signal if something unusual is found**
  - **Keep running averages**



Convenient data returned



Prioritization of data for novelty and pre-specified target classes

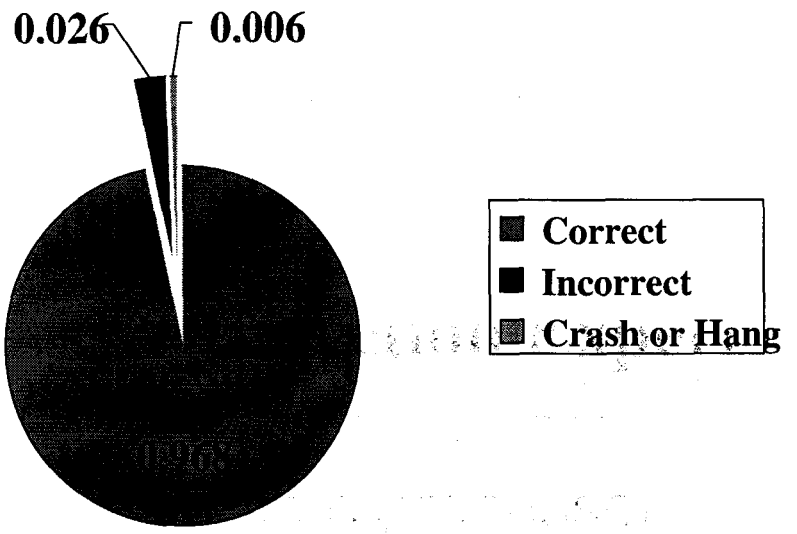


Prioritization of data based on geological analysis of region

Texture Fault Injection Experiments

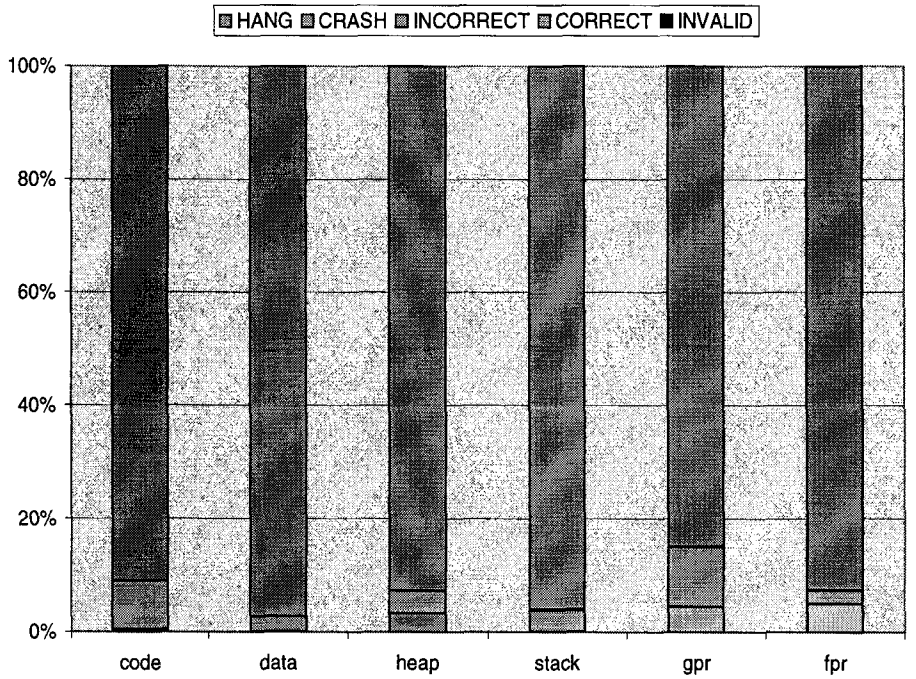
One fault randomly injected into processor registers and memory during each execution of the application;
No fault tolerance or fault avoidance coding techniques

- **Probability of no error due to single fault - 96.8%**
- **Probability of crash/hang due to single fault - 0.6%**
- **Probability of incorrect result due to single fault - 2.6%**

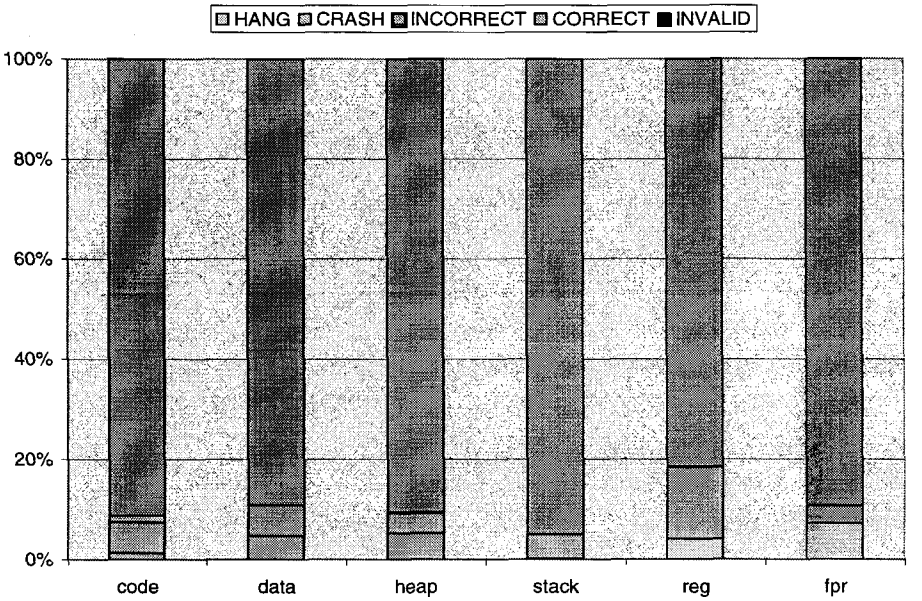


Bottom line: For this application, most faults have no effect

Fault Injection into OTIS Application



Faults Injected into Master Process

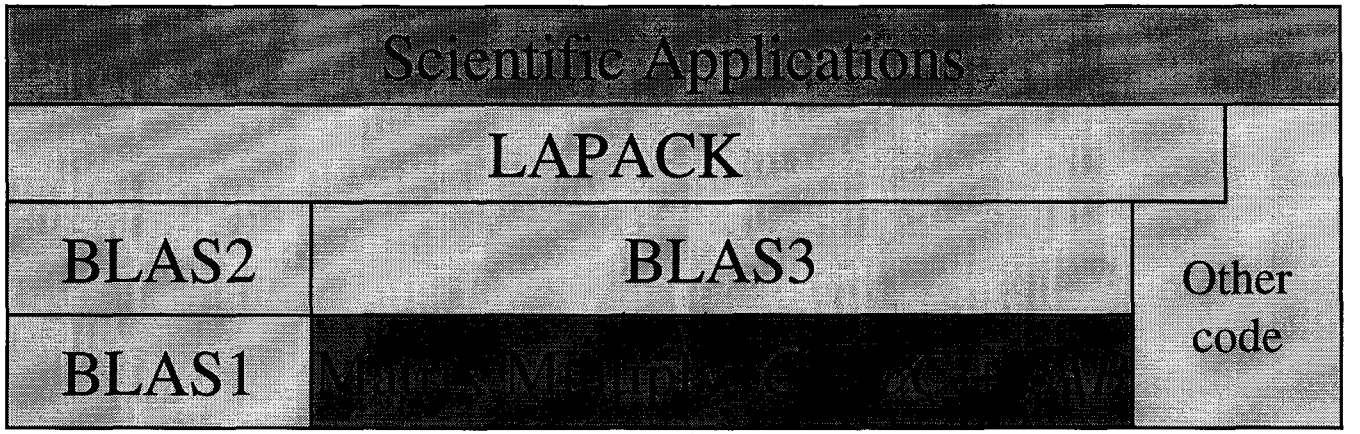


Faults Injected into Slave Process

OTIS is a master/slave application

Overall result: **80-90% of faults have no effect**
5-15% of faults cause hang or crash
<2% of faults cause data error

Science Applications: Built on Matrix Multiplication?



(overly simplified)

We want to make scientific applications fault tolerant

Scientific applications often make heavy use of linear algebra, such as LAPACK

Linear algebra makes use of BLAS routines

LAPACK was designed to use BLAS3 for most of the computations

Modern implementations of BLAS3 are built purely on Matrix Multiplication, Kågström et al. (1998), Whaley and Dongarra (1998)

Algorithm-Based Fault Tolerance (ABFT)

Algorithm-Based Fault Tolerance (ABFT) - Huang and Abraham (1984):

- Calculate *C = A B*, where *A*, *B*, and *C* are augmented as:

$$A^* = \left(\frac{A}{v^T A}\right),\ B^* = (B \mid Bw),\ C^* = \left(\frac{C}{v^T C} \middle| \frac{Cw}{v^T Cw}\right)$$

- In the absence of errors:

$$C^* = \left(\frac{C}{v^T C} \middle| \frac{Cw}{v^T Cw}\right) = \left(\frac{A}{v^T A}\right)(B \mid Bw) = \left(\frac{AB}{v^T AB} \middle| \frac{ABw}{v^T ABw}\right) = A^* B^*$$

- Checking $v^T C = v^T AB$ and $Cw = ABw$ allows one to detect and correct errors in C
- Input arrays are augmented

  Libraries which use ABFT must either copy data to larger arrays or demand larger arrays be used outside

Result checking (RC) - Wasserman and Blum (1997), Prata and Silva (1999), Turmon et al. (2000):

- Check $Cw = ABw$ as a post condition
- If incorrect, repeat calculation
- Initial arrays are not changed

  Library can be written as a black box, with same used interface as non-RC library

REE ABFT Work at JPL

JPL in-house team, led by M. Turmon (1998-2001):

- Determined how to set numerical tolerance to separate errors caused by a fault from those inherent in finite-precision numerical calculations
- Developed a general mechanism for obtaining bounds for new operations based on the operator-specific postcondition
- Used Matlab-based testing to validate efficiency/accuracy tradeoffs of threshold tests

  Detect 99.9% of numerically significant faults in mat. mult., QR, LU, SVD, mat. inv., and FFT, with zero false alarm

  (A numerically significant fault is one causing a relative error of at least one part in 10^{10} in the affected element.)
- Wrote a full set of ABFT-RC ScaLAPACK wrappers for above lin. alg. operations, with

  C and Fortran interfaces

  Customization of system actions in the event of fault detection, e.g. retry or immediate abort

  "Plug-in" routines - direct replacements for their ScaLAPACK counterparts which are accessed simply by recompiling with new libraries

  "Expert" version - exposes some internal structures to allow greater application speed and memory allocation control
- Wrote ABFT-RC Plapack wrappers for some lin. alg. operations
- Wrote ABFT-RC FFTW wrappers

REE ABFT Work at U. Texas

R. Van de Geijn et al. (2000-2001) extended Abraham's ABFT work to detect errors introduced into A and B

Right-sided check ↓ Left-sided check ↓

Showed check of $Cw = ABw$ is not sufficient, also need to check $v^T C = v^T AB$

- RC (Prata and Silva, 1999) checks only $Cw = ABw$
- Evaluated success of error detection using each check, as well as overhead of doing each check

Adopted techniques of Turmon et al. (2000) to differentiate between errors due to corruption and errors due to limited machine precision.

Adopted roll-back mechanism from RC, but at a new level

- This lowers the overhead of correcting each fault

Created a practical implementation that adds fault tolerance to a high-performance library

- FLARE: Formal Linear Algebra Recovery Environment

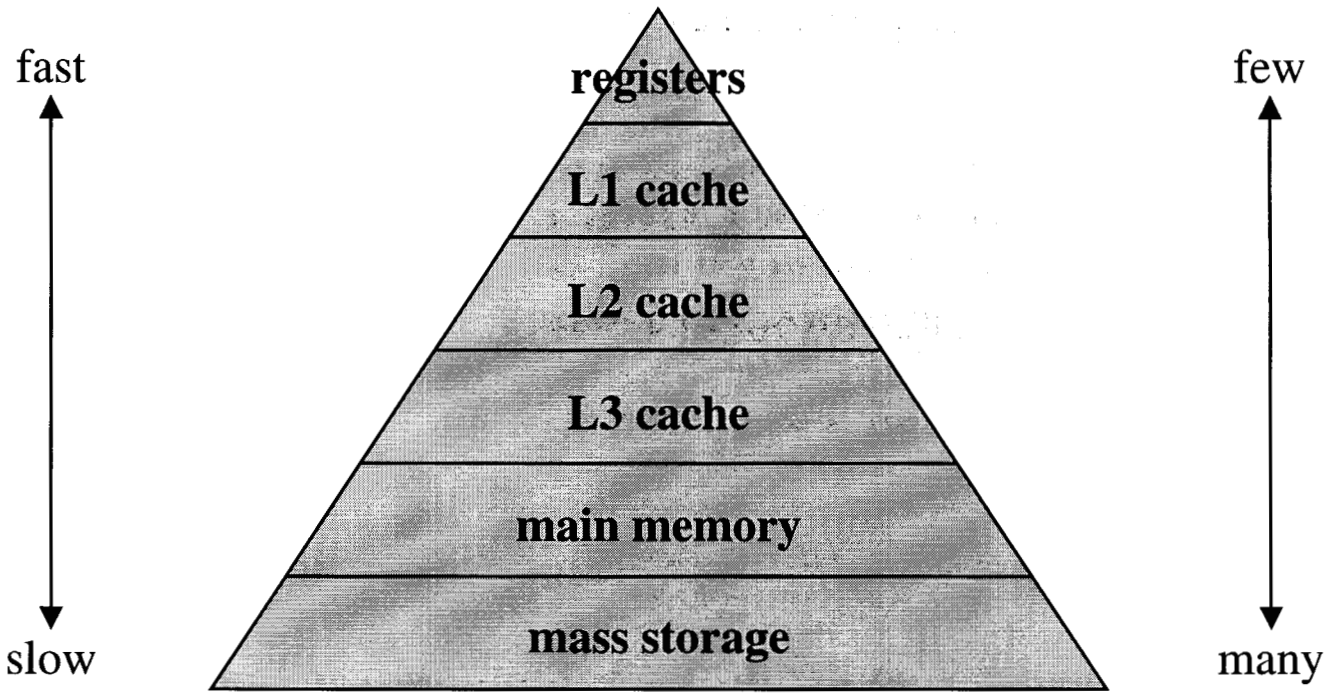
Implementation of Left and Right-sided Detection and Correction

Where should Error Detection and Correction be applied?

- **Most matrix multiplication schemes are generally divide the matrices at multiple levels**
- **High level**
  - **Compute the entire multiply, then perform the check**
  - **Costs less to detect an error**
    - $O(n^2)$ vs. $O(n^3)$ as n is larger
- **Low level**
  - **Compute a very small part of the multiply, then perform the check**
  - **Uses less memory**
  - **Costs less to correct an error**
  - **Increases frequency with which errors can be detected**
- **Left-sided D/C is lower-level than right-sided D/C**
- **At a given level, left-sided D/C is more efficient than right-sided D/C**
- **Take advantage of the division of the matrices at multiple levels, which is done primarily because of the memory hierarchy of modern processors, for fault tolerance**

Computer Architecture

Modern computer architectures use hierarchical memory:

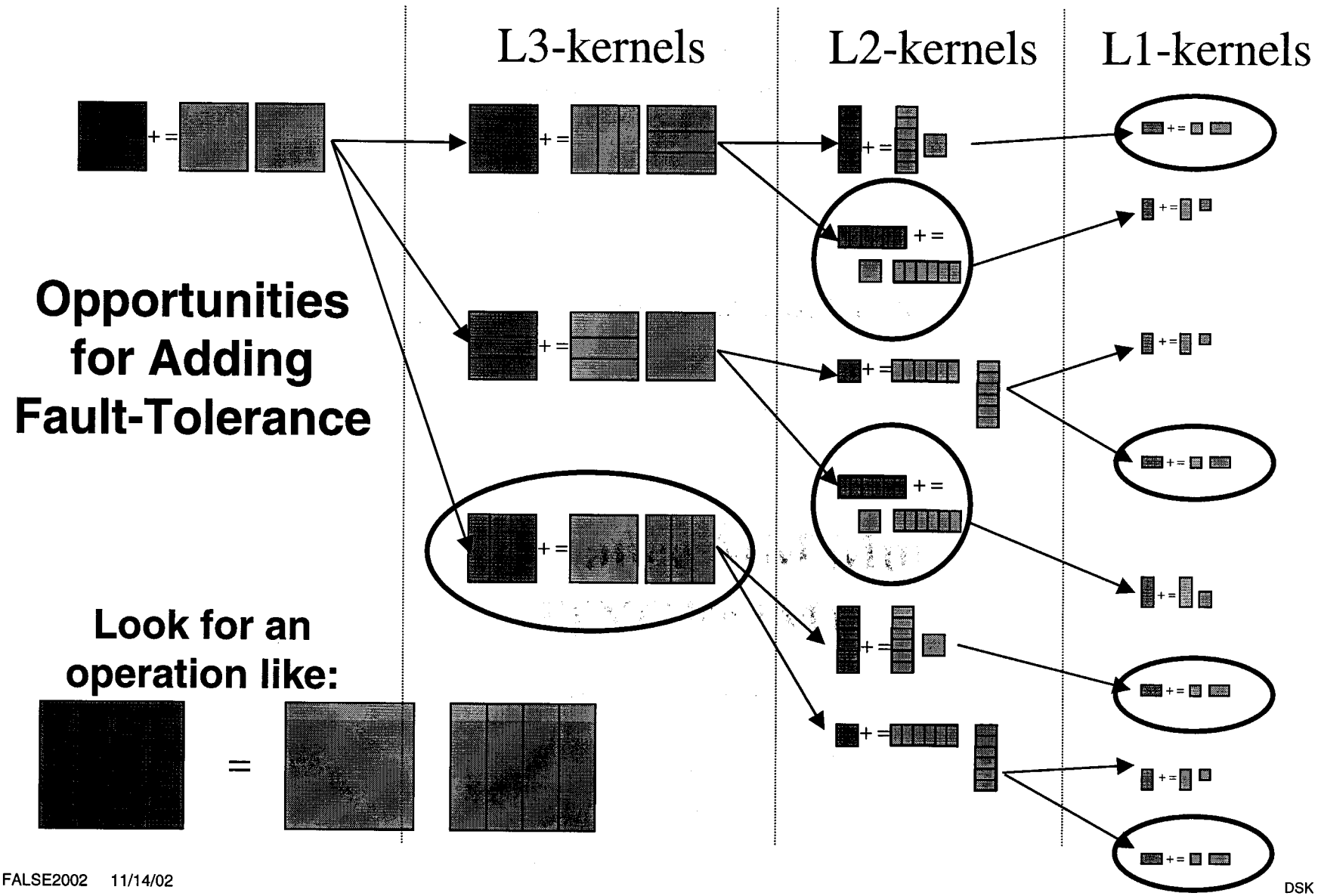


At each level:

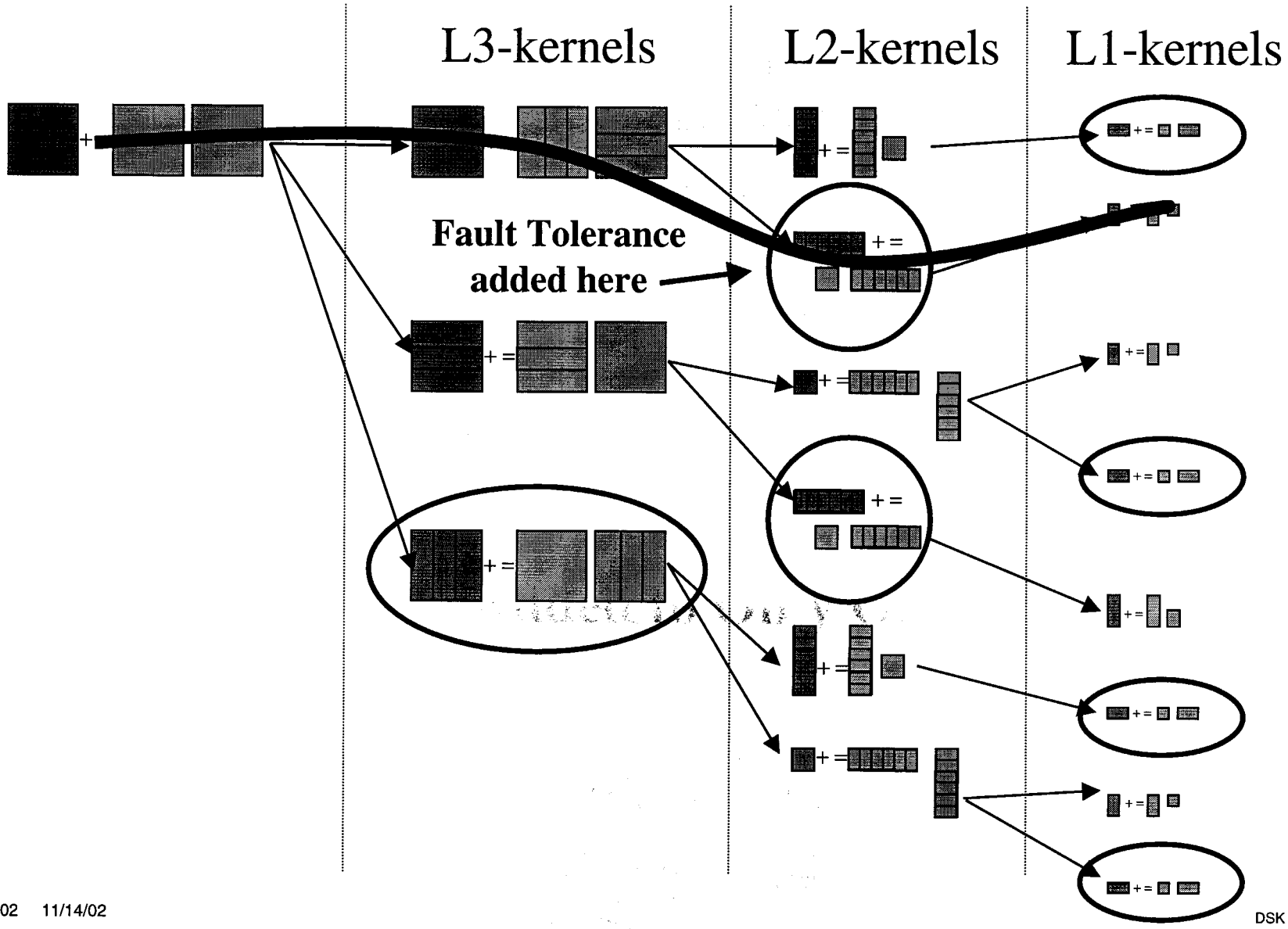
- **less storage than next level**
- **faster access than next level**

Other computers may have different layers, but the principles are the same

A High-Performance MM Multiplication (ITXGEMM)



FT Scheme Added to Only One Path



Test Environment

Test computer:

- Intel Pentium ® III processor
- 650 MHz
- 16 KBytes L1 cache
- 256 KBytes L2 cache
- IEEE double-precision floating point arithmetic

$\tau = u \approx 2.2 \times 10^{-16}$

- Worse case
- Will detect some round-off errors

  If a possible error is detected, the appropriate caches are invalidated, the kernel is done again, and the check is recalculated

  - If the same error is detected, assume it is round-off error
  - If the same error is not detected, assume there was a transient error, and repeat until same error is detected twice successively

For fault injection, the L1 kernel includes instructions to randomly alter a random bit in a random word of A or B in L1 cache

- Only 1 fault is injected for each C = A B

Test Results (Practice vs. Theory)

The error detection mechanisms performed as predicted

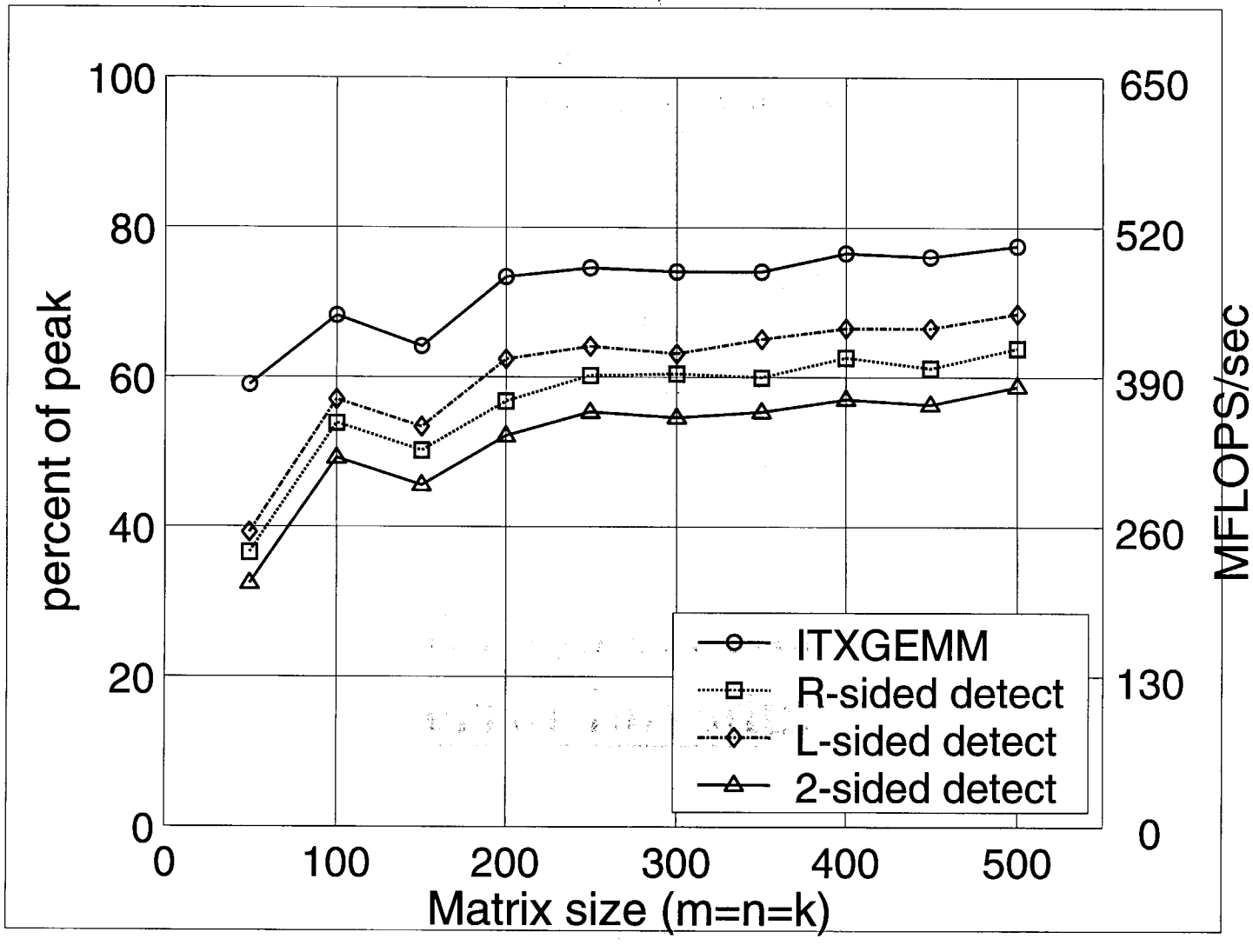
All significant errors introduced into A were detected by both the left-sided and two-sided detection methods

All significant errors introduced into B were detected by both the right-sided and two-sided detection methods

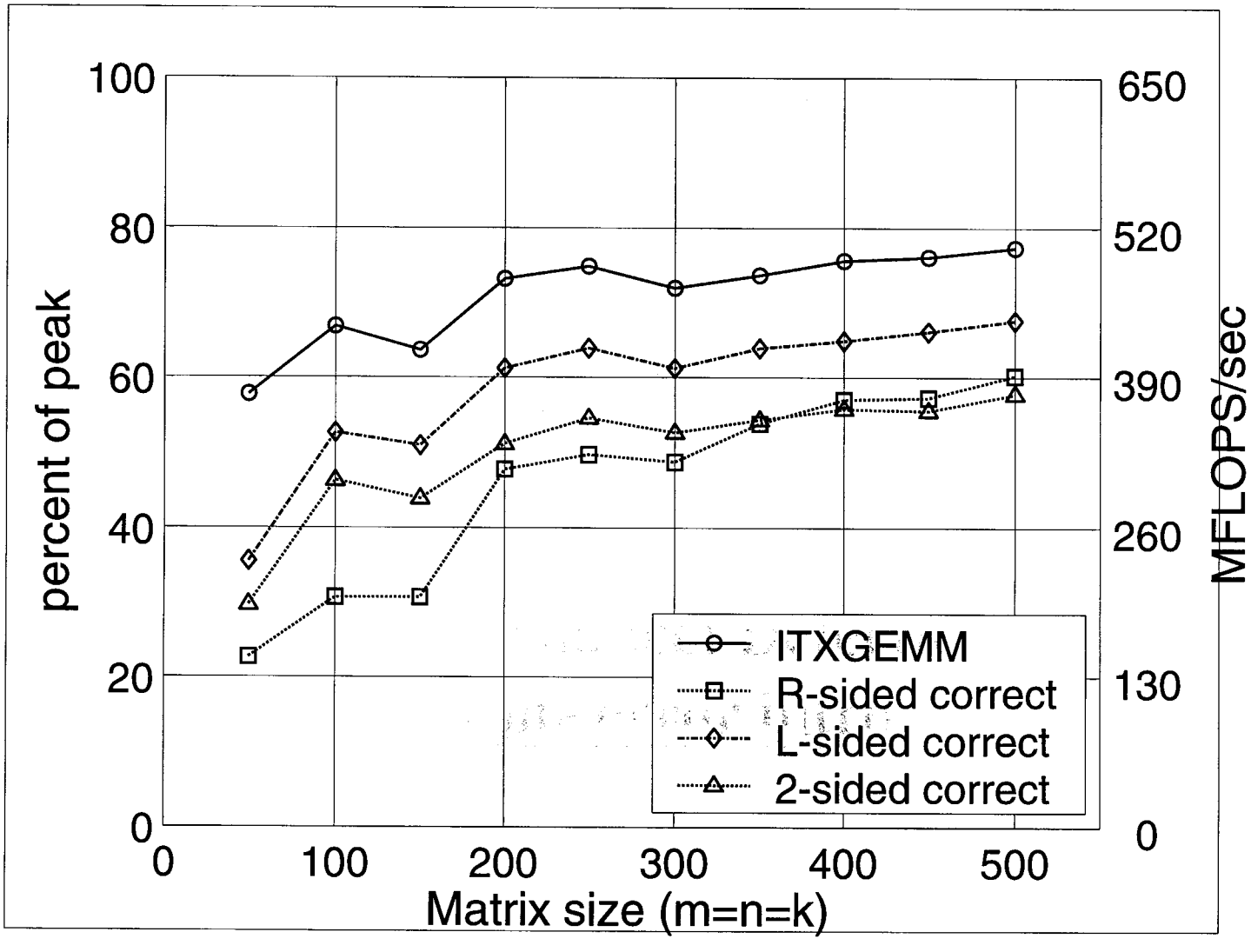
Both left- and right-sided methods detected significant errors introduced in either A or B, except:

- When A had columns with elements that summed to zero, the left-sided detection mechanism had trouble detecting errors in B
  - When all columns of A summed to zero, left-sided detection could not detect any errors in B
- When B had rows with elements that summed to zero, the right-sided detection mechanism had trouble detecting errors in A
  - When all rows of B summed to zero, right-sided detection could not detect any errors in A

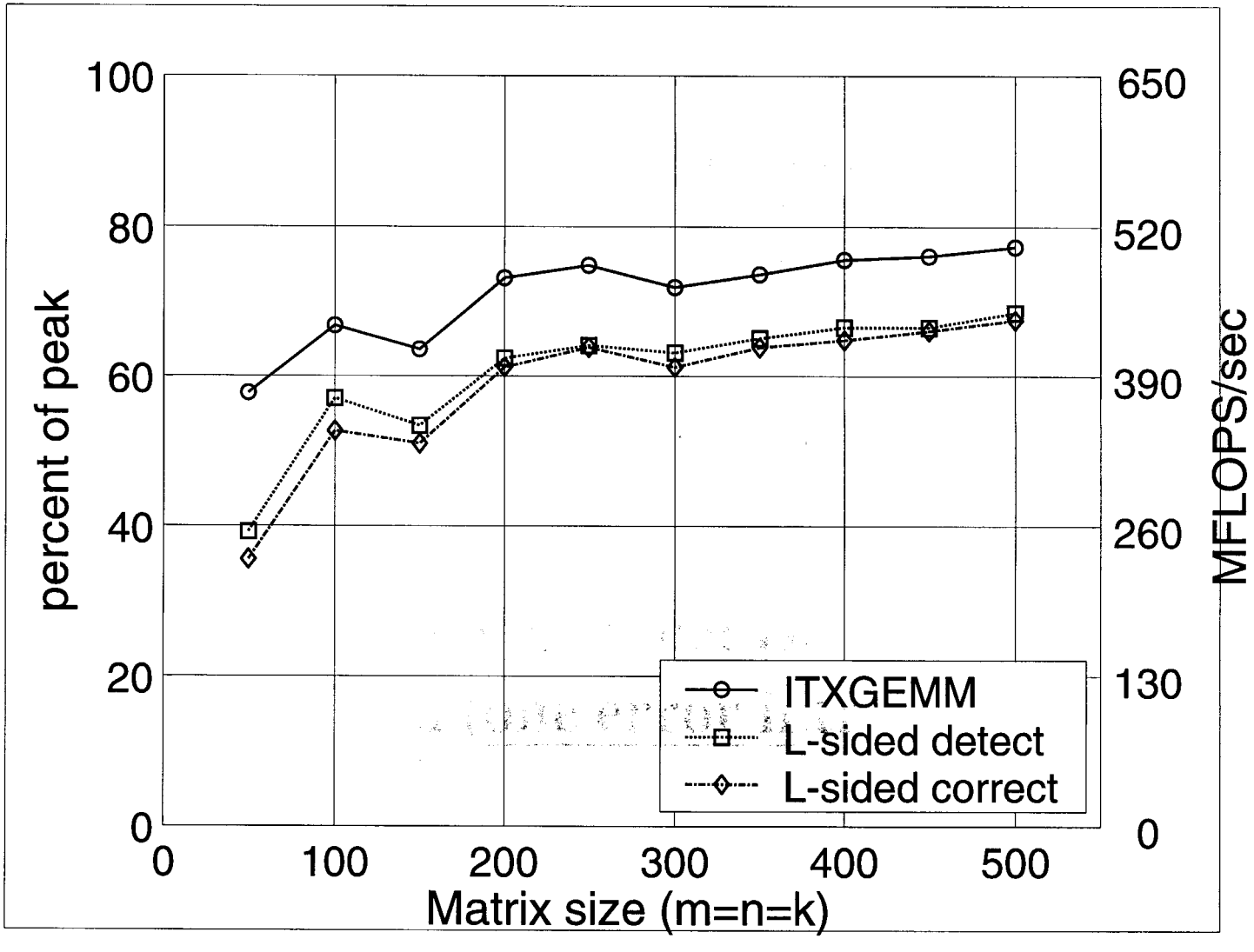
FLARE Performance: Detection Only (no error introduced)



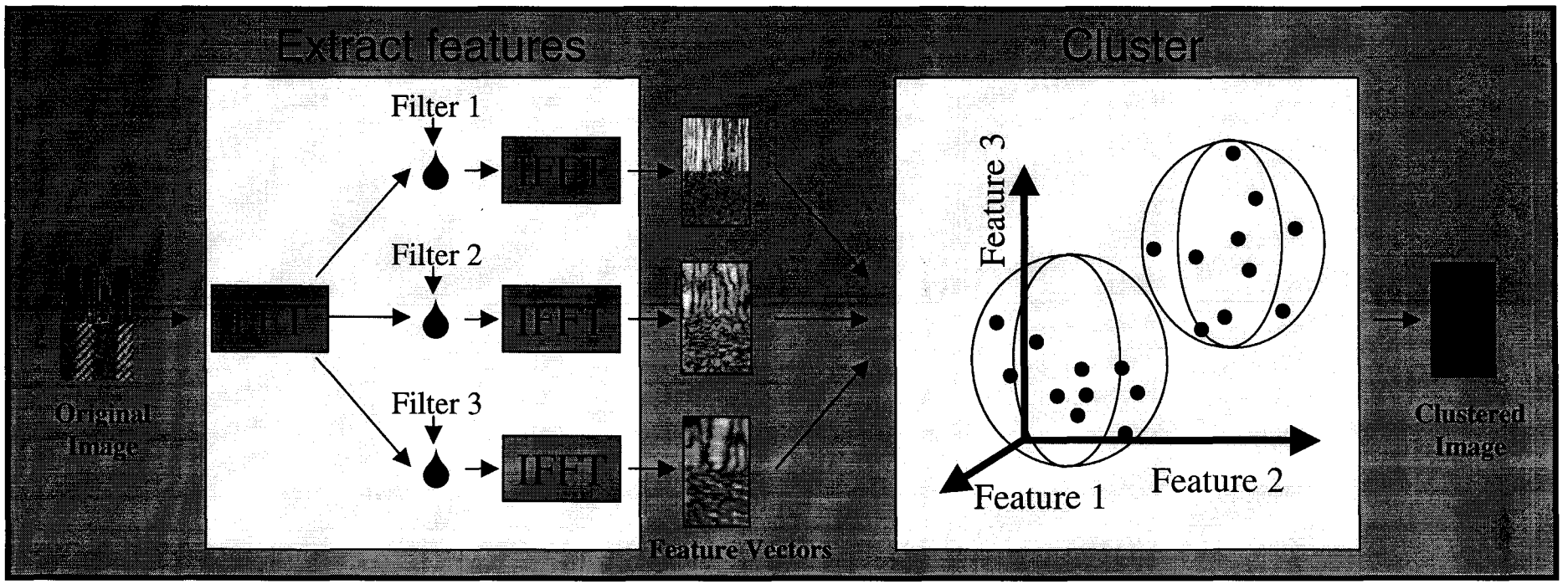
FLARE Performance: Detection and Correction (one error introduced)



FLARE Performance: Detection vs. Correction (one error introduced)



Example: ABFT applied to Rover Texture Analysis



- FFT and IFFT (protected by ABFT) take 60% of run time for runs with 12 filters
- Cluster (protected by reasonableness checks) takes 20%, I/O (reliable at a system level) takes 10%, other code takes 10%
- An REE node on Mars is expected to see a fault every 50 hours
- We believe ~ 3% of faults could cause a error that changes data, so the average time between these events is 1700 hours
- Assuming FFT and IFFT ABFT have 98% coverage, reasonableness checks on cluster have 50% coverage, and reliable I/O is reliable, makes the average time between errors that change data > 8000 hours (11 months)

Lessons Learned

Fault rates are much lower than expected

- 1 per 3-4 hours per processor in low Earth Orbit
- 1 per 100 hours per processor on the surface of Mars

A low percentage of faults transition into errors for science data processing applications

- Depends on application
- Perhaps 1 out of 10 faults become errors on average

Application fault tolerance techniques have high coverage

- > 95% for code sections covered

The rest of the system has a small cross section to faults

- Faults trigger errors in proportion to execution time
- The OS, though untested, consumes only a small percentage of the execution time

Conclusion:

A COTS-based payload data processing system is feasible today

References

Beahan, J., L. Edmonds, R. Ferraro, A. Johnston, D. S. Katz, and R. R. Some, "Detailed Radiation Fault Modeling of the Remote Exploration and Experimentation (REE) First Generation Testbed Architecture," 2000 IEEE Aerospace Conference, 2000

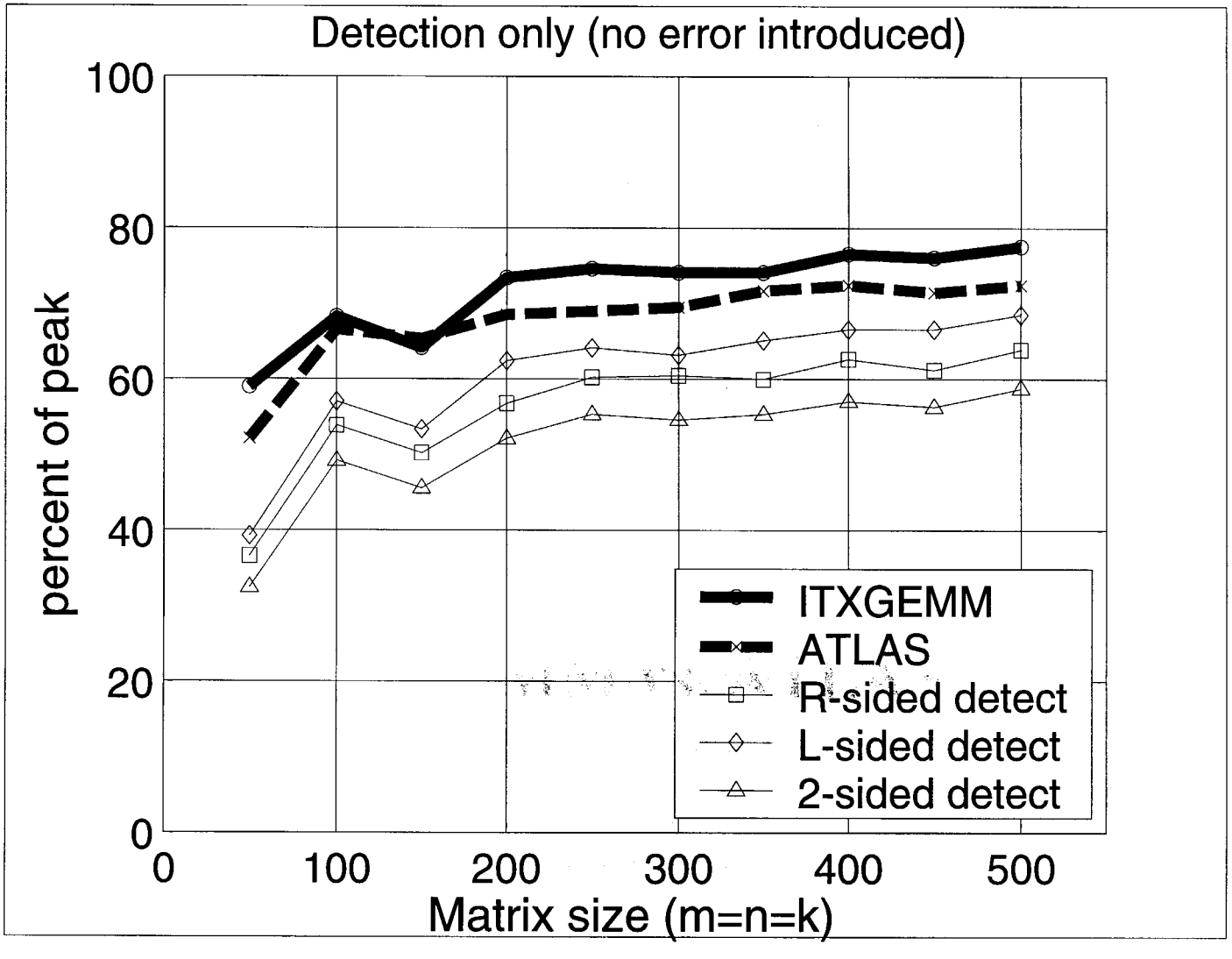
Castaño, R. L., T. Mann, and E. Mjolsness, "Texture Analysis for Mars Rover Images," Applications of Digital Image Processing XXII, SPIE Vol. 3808, 1999

Chen, F., L. Craymer, J. Deifik, A. J. Fogel, D. S. Katz, A. G. Silliman, Jr, R. R. Some, S. A. Upchurch, and K. Whisnant, "Demonstration of the Remote Exploration and Experimentation (REE) Fault-Tolerant Parallel-Processing Supercomputer for Spacecraft Onboard Scientific Data Processing," International Conference on Dependable Systems and Networks, 2000.

Gunnels, J. A., D. S. Katz, E. S. Quintana-Ortí, and R. A. van de Geijn, "Fault-Tolerant High-Performance Matrix Multiplication: Theory and Practice," International Conference on Dependable Systems and Networks, 2001

Gustafson, W. and A. Gillespie, "Onboard Processing of Orbital Hyperspectral Thermal Infrared Images," IEEE Aerospace Conference, 2002

Some, R. R., W. S. Kim, G. Khanoyan, L. Callum, A. Agrawal, J. J. Beahan, A. Shamilian, and A. Nikora, "Fault Injection Experiment Results in Spaceborne Parallel Application Programs," IEEE Aerospace Conference, 2002

Turmon, M., R. Granat, and D. S. Katz, "Software-Implemented Fault Detection for High-Performance Space Applications," International Conference on Dependable Systems and Networks, 2000

ITXGEMM vs. ATLAS