

# Software Fault Detection Using Dynamic Translation

George A. Reis<sup>1</sup> · David I. August<sup>1</sup>  
Robert Cohn<sup>2</sup> · Shubhendu S. Mukherjee<sup>2</sup>

<sup>1</sup>Liberty Research Group  
Princeton University

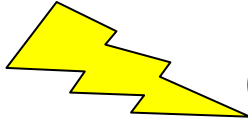
<sup>2</sup>FACT and VSSAD Groups  
Intel Massachusetts

February 3, 2006

<http://liberty.princeton.edu>

# Transient faults

- Hardware faults
  - Different than design or manufacturing faults
  - Cannot test for fault before hardware use
  - Hardware is not permanently damaged
- Caused by external energetic particle striking chip
- Randomly change one bit of state element or computation


$$\begin{array}{r} 0x8675309 \\ * \quad \underline{\quad 0x42} \\ 0x32AA36852 \end{array}$$

# Severity of transient faults

- IBM historically adds 20-30% additional logic for mainframe processors for fault tolerance [Slegel 1999]
- In 2000, Sun server systems deployed to America Online, eBay, and others crashed due to cosmic rays [Baumann 2002]
- In 2003, Fujitsu released SPARC64 with 80% of 200,000 latches covered by transient fault protection [Ando 2003]
- “it was found that a single soft fail ... was causing an entire interleaved system farm (hundreds of computers) to crash.” [SER: History, Trends, and Challenges 2004]
- Los Alamos National Lab ASC Q 2048-node supercomputer was crashing regularly from soft faults due to cosmic radiation. [Michalak 2005]
- Processors are becoming more susceptible
  - lower voltage thresholds
  - increased transistor count
  - faster clock speeds



# Goals

---

- Develop transparent (to user) way to increase reliability, specifically targeting soft errors, without any hardware requirements.
- This can be used to increase the reliability of currently deployed systems.



# Mitigation of transient faults

---

- Levels to add reliability
  - Circuit, Logic, Microarchitectural, Architectural, Application
- Hardware techniques
  - Lockstepping processors [HP NonStop]
  - Redundant multithreading (RMT) [Reinhardt & Mukherjee, 2000]
- Software techniques
  - NMR, TMR
  - Source-to-source [Rebaudengo et al. 2001]
  - SWIFT [Reis et al. 2005], EDDI, CFCSS [Oh et al. 2002]



# Dynamic Software Translation

- Software techniques
  - Can be applied **today** to existing applications on **existing hardware**
- Binary translation
  - Can be applied **without recompilation**
    - legacy binaries with no source code
    - compilation of included libraries
- Dynamic binary translation
  - Can easily handle:
    - Variable-length instructions
    - Mixed code and data
    - Statically unknown indirect jump targets
    - Dynamically generated code
    - Dynamically loaded libraries
  - Can attach to **running application** (and later detach)



## Store Protection

---

If a tree falls in the forest,  
but nobody is around to hear it,  
does it make a sound?

If a fault affects some data,  
but does not change the output,  
does it make a error?

Only store operations affect output,  
so validate data before stores.



# Our implementation

---

- Create single-threaded, software-only version of RMT
  - Add redundant instruction
  - Add verification before memory accesses
  - Add duplication of loaded values
- Use PIN's dynamic instrumentation infrastructure
- Implemented for x86
  - Only 8 registers available
    - register pressure is big issue
  - Implicit register operands (PUSH, SAL)
    - add more constraints to register allocation
    - EFLAGS is frequently used





# PIN Reliability Transform

FORMAT: OP DEST1 DEST2 = SRC1 SRC2

Another version of  
EFLAGS

Compare before  
memory instruction

```

CMP EFLAGS'' = EAX EAX'
JNZ EIP
  = 0xFAULTDETECT EIP EFLAGS''
  
```

Duplicate non-  
memory instruction

```

MOV (DS:EAX) = 0x00000000
INC EDX EFLAGS = EDX
INC EDX' EFLAGS' = EDX'
ADD EAX EFLAGS = EAX 0x04
ADD EAX' EFLAGS' = EAX' 0x04
CMP EFLAGS = EDX 0x4a
CMP EFLAGS' = EDX' 0x4a
JBE EIP
  = 0xABCDABCD EIP EFLAGS
  
```

Duplicate version  
of EFLAGS



# PIN Reliability Transform - Loads

Compare before  
memory instruction

```
CMP EFLAGS'' = ESP ESP'  
JNZ EIP  
    = 0xFAULTDETECT EIP EFLAGS''
```

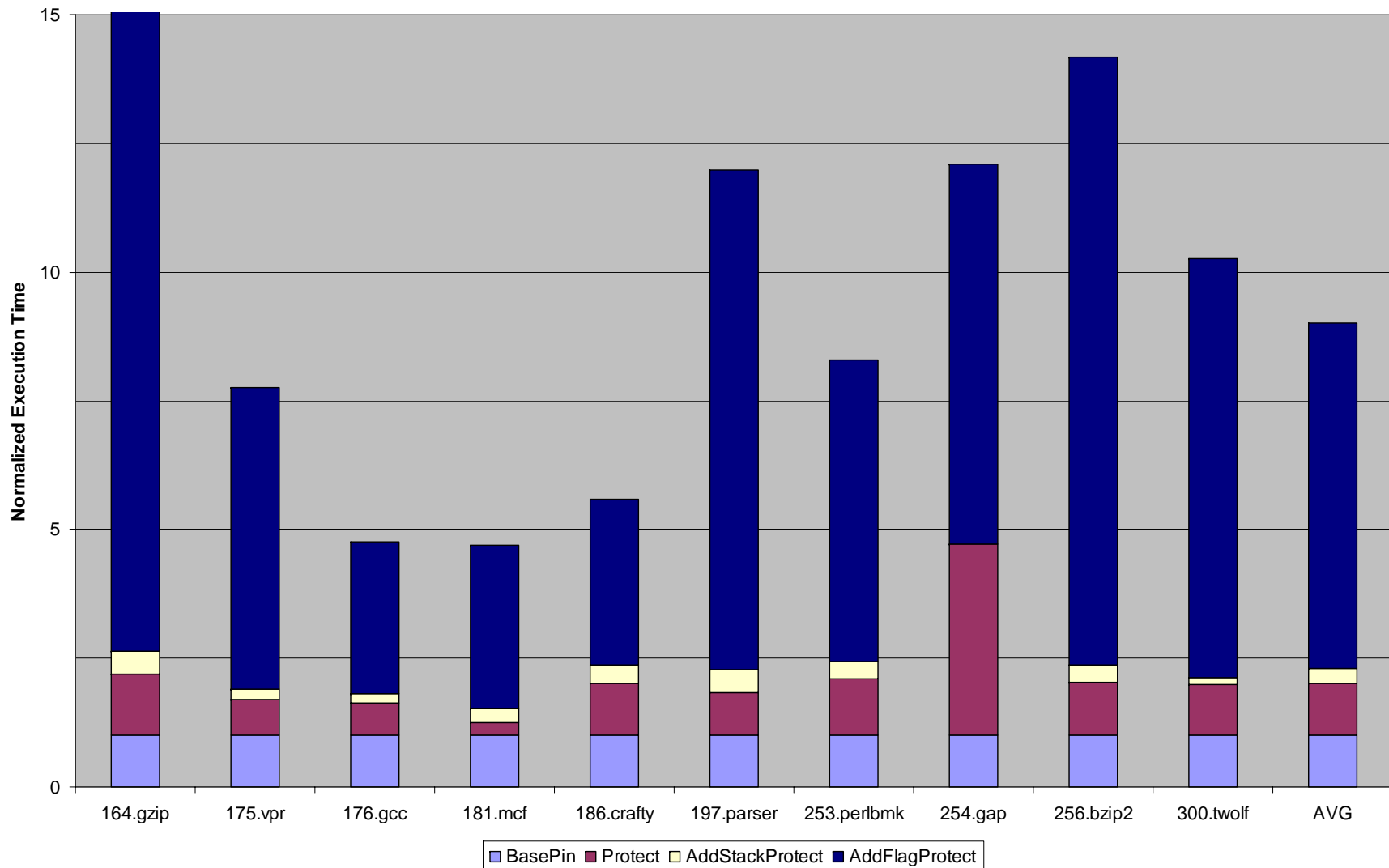
Duplicate loaded  
value

```
MOV EAX = (ESP, 0xfffffd64)  
MOV EAX' = EAX  
. . .
```



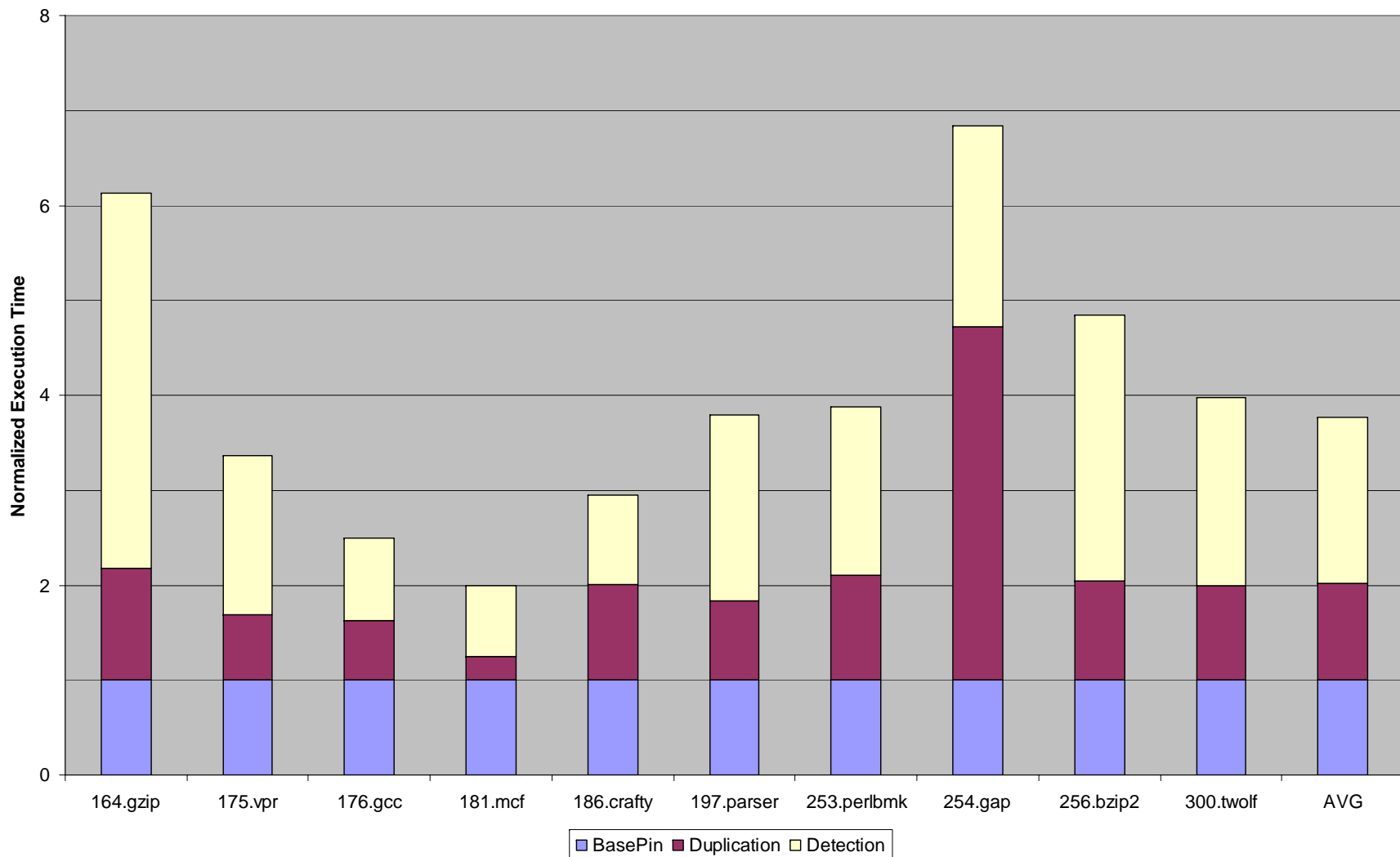


# Performance: duplication with register breakdown





# Performance: detection vs. duplication



## Just the beginning...

---

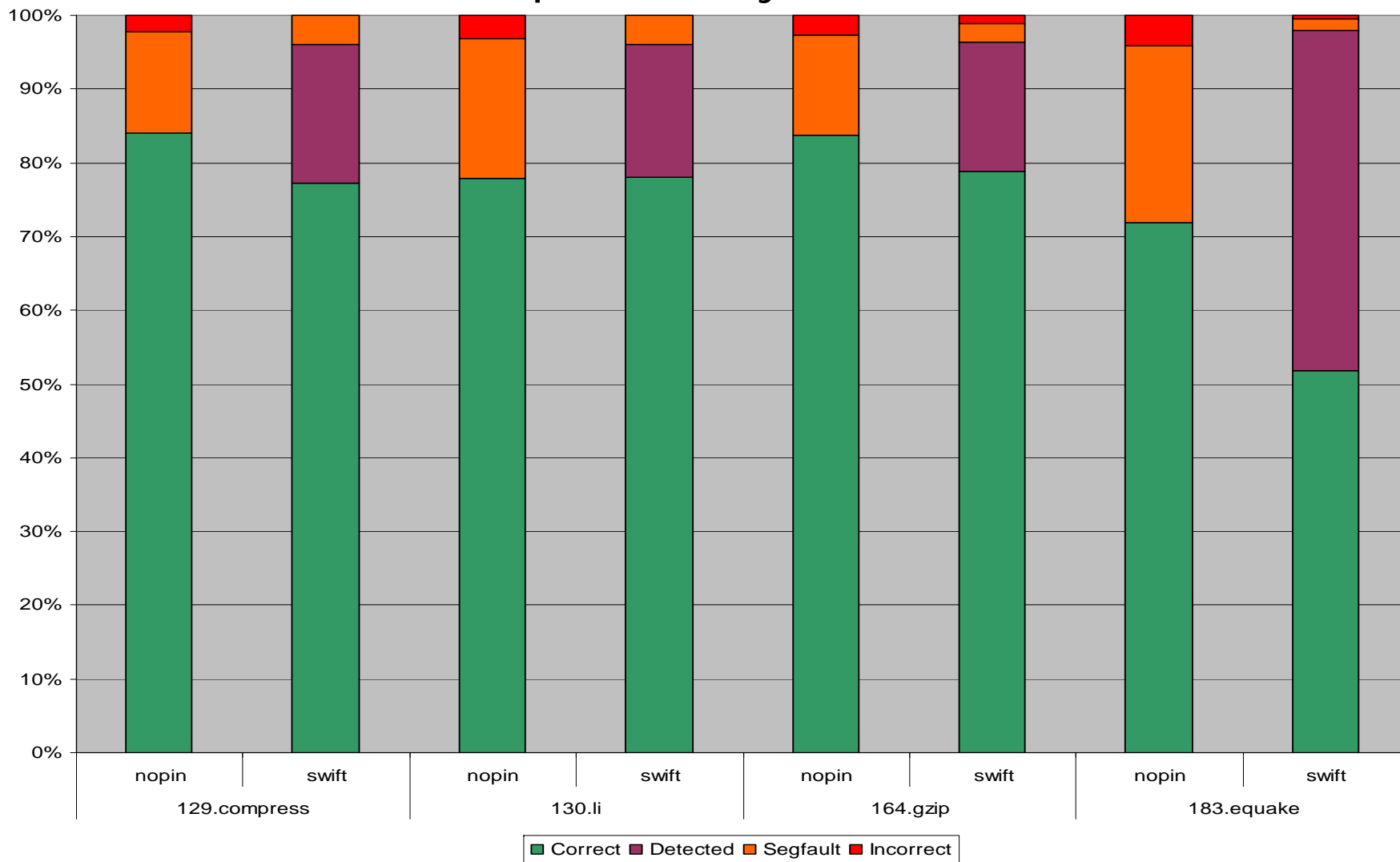
- Register allocation
  - Can greatly reduce overhead via more sophisticated algorithm
  - Increase reliability by protecting MMX, FP registers
- Persistence Pin [Janapa Reddi WBIA-2005]
  - Cache (on disk) the instrumented code
  - Eliminate most of dynamic translation cost
- Running on x86-64
  - More available registers will decrease overhead due to spill/fill
- Fault injection to determine error coverage





# Just the beginning... Error coverage

preliminary results



## Just the beginning... Software-modulated Fault Tolerance

- Dynamic translation can make different decisions for different code regions, and can change over time
  - Programs
  - Functions
  - Individual store dependence chains
- Programs have varying level of importance
- Programs have varying level of natural fault resistance
- Output corrupting faults have varying severity



*original jpegenc output*



*faulty jpegenc output*



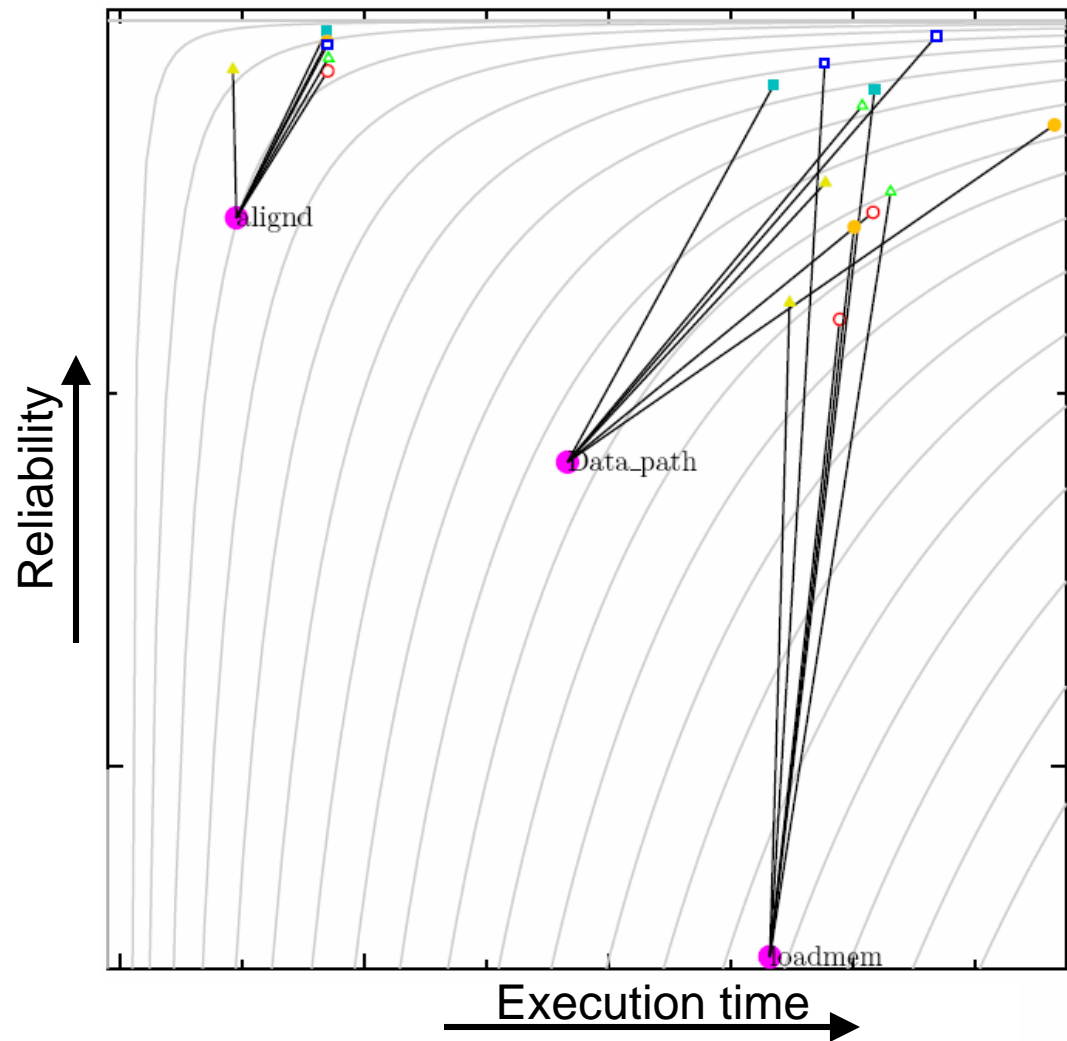
*faulty? jpegenc output*





# Just the beginning... Software-modulated Fault Tolerance

- Software flexibility allows tradeoff between performance and reliability
- Tune redundancy based on reliability and performance response
- Example: changes in reliability and execution time for different function of 124.m88ksim





Questions?

