Chaotic-Map Method for Detection and Diagnosis of CPU-GPU Hybrid Computing Systems

Nagi Rao

Oak Ridge National Laboratory

Discussion Presentation

June 5, 2015

Innovative Computing Laboratory

University of Tennessee, Knoxville

Research Sponsored by

ASCR Applied Mathematics Program, U.S. Department of Energy

UT-BATTELLE

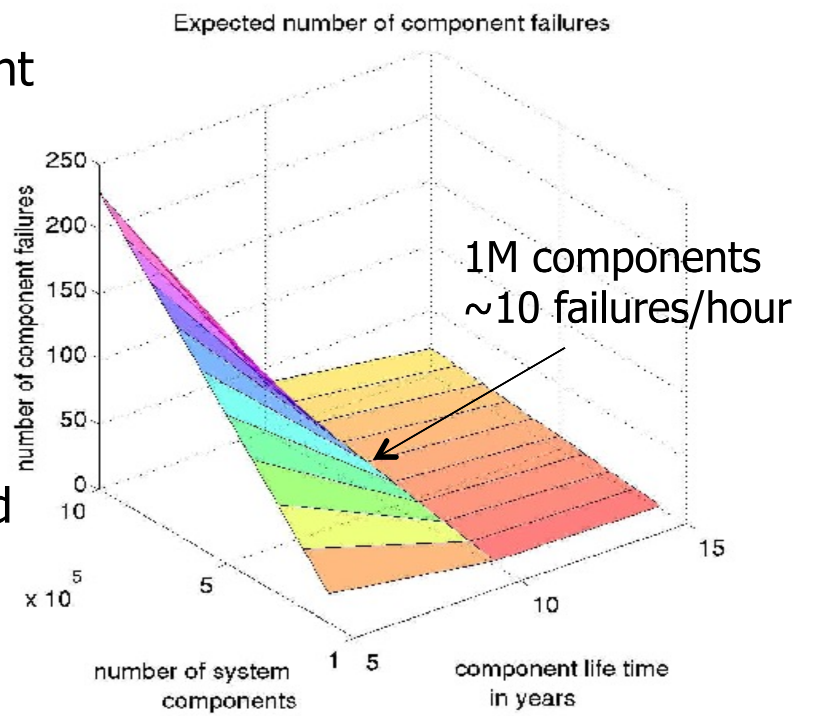
Oak Ridge National Laboratory
U. S. Department of Energy

Outline

1. Background
2. Chaotic map method
3. Diagnosis of hybrid systems
4. Codes and experimental results

Inherent Failures in Exascale Computing Systems

- Exascale computing systems are expected to have millions of processor cores and other components.
  - components with expected life-span of ten years
    - ~100k hours/component = 10 failures/hour among 1M components
  - codes that run for a few hours likely experience failures of several components.
- Failure rates limit the effectiveness of current check-point/recovery methods:
  - Recovery times could be hours for Exascale systems
  - transient silent errors may lead to erroneous computations
- Failures will be integral part of Exascale computations – must be explicitly accounted
  - code outputs must be quantified with confidence estimates
    - specific to system failure profile
    - justifiable by measurements



Related Areas: Resilient Computations

- Foundational works:
  - von Neumann studied (in 1950s) mathematical aspects of achieving reliable computations over systems with unreliable components
  - subsequent reliability improvements in computing systems, perhaps, led to such studies not being extensively continued
  - Several fault detection problems in digital systems are known to be NP-hard
- Deployed systems: computing systems in satellites
  - deployed over past decades - enhanced with Software-Implemented Hardware Fault Tolerance (SIHFT) methods to counteract errors due to radiation in space environments.

But, Exascale computations present new challenges:

- sheer size and system complexity makes dynamic profiling of the failures and robustness complicated
- computation becomes inherently probabilistic:
  - for most applications, 100% guarantee of robustness against failures in not possible
  - requires confidence measures for code outputs – running to completion is not sufficient

Undecidability of Resilient Computations and Proofs

Addressed computational aspects of resilient computations under broad class of faults

Resilient computations present significant computational challenges:

(a) asserting resiliency of computations is non-computable

(b) mathematical proofs of resilience of algorithms are undecidable

These problems are not solvable in general form by computations and mathematical proofs alone: but,

- resilient computations can be designed for specific classes
- additional fault detection methods could make some problems computable

In general, these results motivate: deeper investigations of fault classes and resilient computations customized for them with complementary information

Reference: Resilience 2014 paper

Chaotic Poincare maps

Poincare Map: $M:\Re^d \to \Re^d$

$$X_{i+1} = M\left(X_i\right)$$

Trajectory

$$X_0, X_1, X_2, \ldots$$

Examples:

logistic map: $X \in [0,1]$

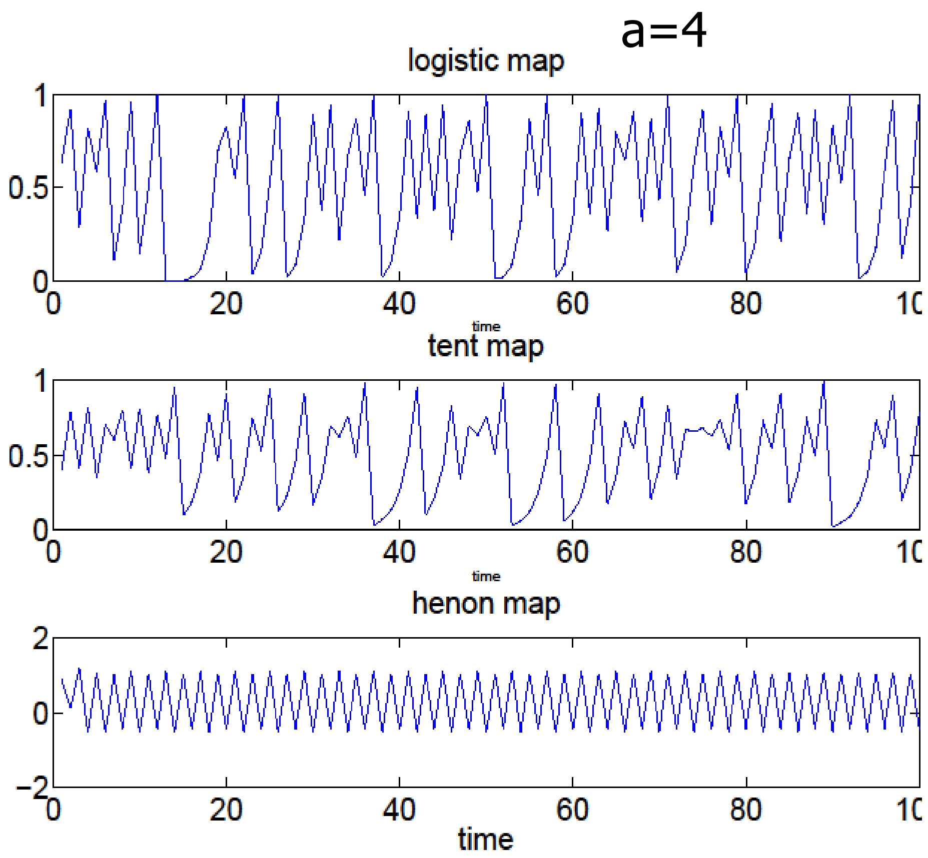
$$M_{L_a}\left(X\right) = aX\left(1-X\right)$$

tent map: $X \in [0,1]$

$$M_T\left(X\right) = \begin{cases} 2X & if \quad X \le 1/2 \\ 2\left(1-X\right) & if \quad X > 1/2 \end{cases}$$

Hennon map

$$M_H\left(X,Y\right) = \left(a - X^2 + bY, X\right)$$



Simple computations generate seemingly complex trajectories

Chaotic maps amplify state errors and spread across bit-space

Chaotic trajectory: $X_0, X_1, X_2, \ldots$ is chaotic if

(i) it is not asymptotically periodic, and

(ii) Lyapunov exponent is positive

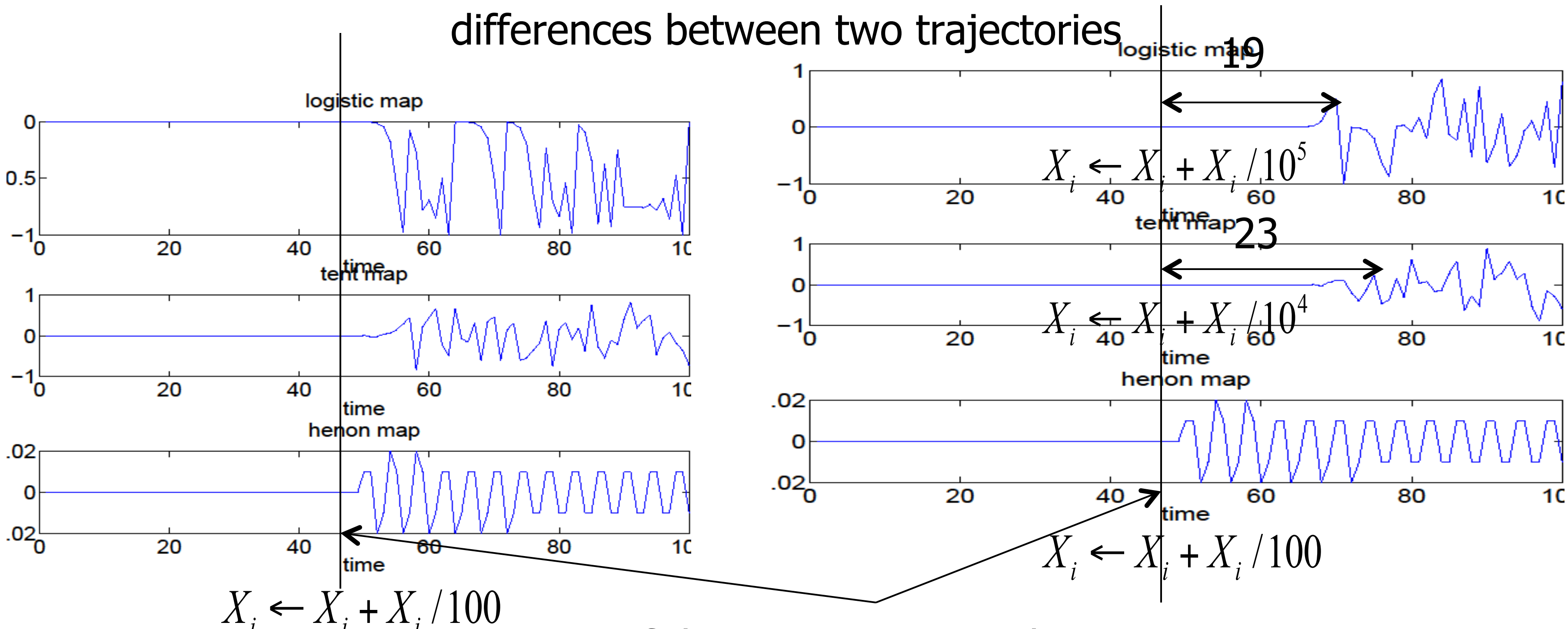
$$\Lambda_M = \ln\left|\frac{dM}{dX}\right| > 0$$

Key Properties:

(i) Extreme sensitivity: small differences in states rapidly diverge

(ii) Wide Fourier spectrum: few iterates cover bit-space

differences between two trajectories



one of the states corrupted at t=50

Poincare maps for fault detection

Poincare maps computed in parallel at different computing units: fault at one will lead to quick divergence of the outputs, depending on:

- **Type of faults**: Wide range of faults in
  - arithmetic and logical operations
  - registers and memory

  but are limited to those in operations used by M(.)

- **Poincare map properties**: Computation of M(.)
  - sensitive to errors
    - in constituent operations, and
    - mechanisms used in storing and updating the states
  - rate of divergence and its detectability depends on the Lyapunov exponent
    - generally, larger Lyapunov exponent values lead to quicker divergence
    - for tent map, $\mathrm{L}_M = \ln 2 > 0$ except at *X=1/2*

Side Note: Codes with known outputs are routinely used for diagnosis of computing systems – Poincare maps are among the least complex

Chaotic-Identity Map

Poincare map amplifies errors in operations used in its own computation

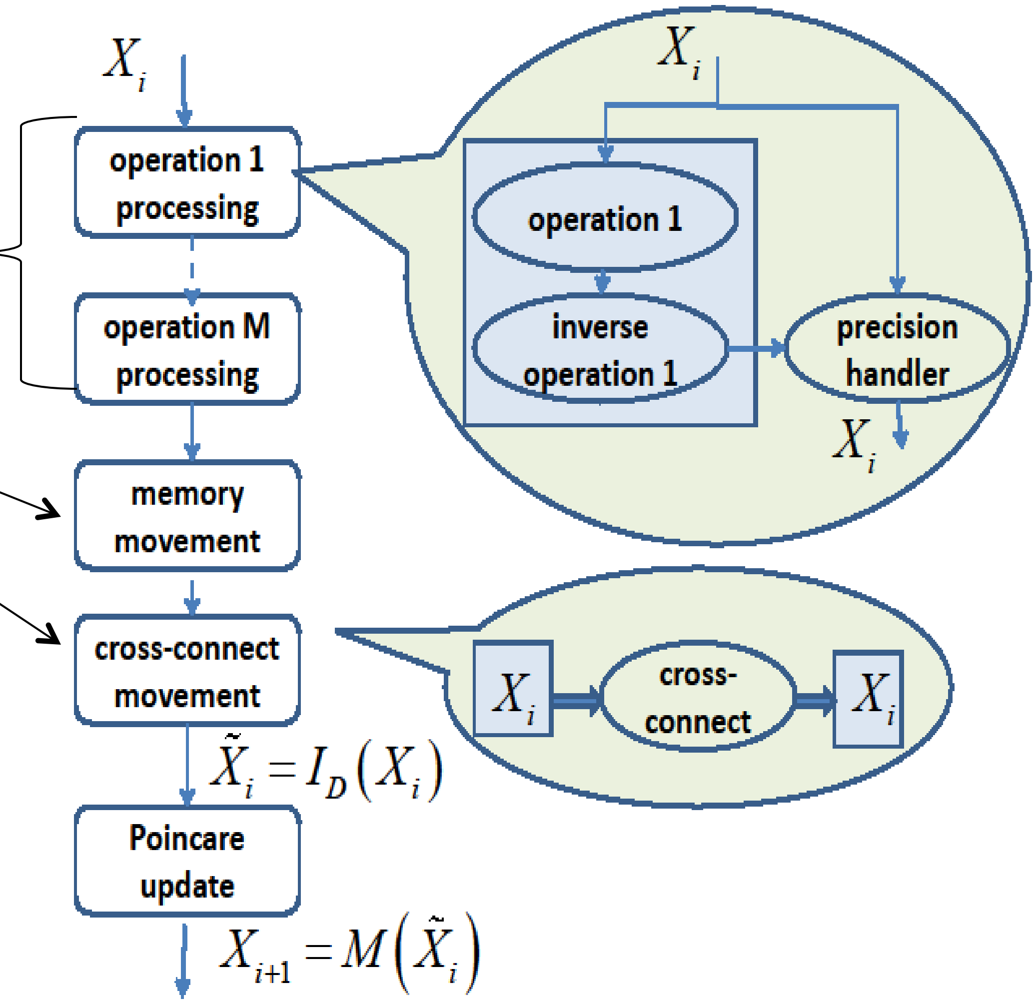
Chaotic-Identity Map:

$$\tilde{X}_i \leftarrow I_D(X_i)$$
$$X_{i+1} \leftarrow M(\tilde{X}_i)$$

Execution routed through

- computing operations
- memory locations
- interconnect links

to capture errors in them



Output $I_D(X_i)$ is identical to X_i if there are no faults

It catches errors in specified operations – instructions, sub-routines, libraries

Chaotic-Computing Map: Identity computations replaced by other operations

Summary: Proof-of-Principle Detection Codes

Initial codes developed and tested on these systems

i. Single-Host System Diagnosis
   - Multiple Cores: pthreads - **delivered to OLCF**
     - 4-core Intel Xeon 2.67GHz; 16-core 16-core AMD Opteron; 32-core Intel Xeon 2.7GHz; 48-core AMD Opteron 2.29GHz
   - GPU Accelerators: CUDA C - **delivered to OLCF**
     - Single-GPU: Quadro 600, Tesla T10, Tesla C1060, Tesla K20X
     - Multiple-GPU: 8 Tesla T10

ii. Multi-Host Hybrid Systems Diagnosis
   - Multi-host, mutli-cores system: MPI+pthreads
   - Multi-host, single GPU system: MPI+ CUDA C
   - Multi-host, multi-core, single GPU: MPI+pthreads+ CUDA C

Systems Used in Tests:

Lens:

77-node linux cluster: 16-core/node 2.3 GHz AMD Opteron; 32 nodes with NVIDIA Tesla C1060

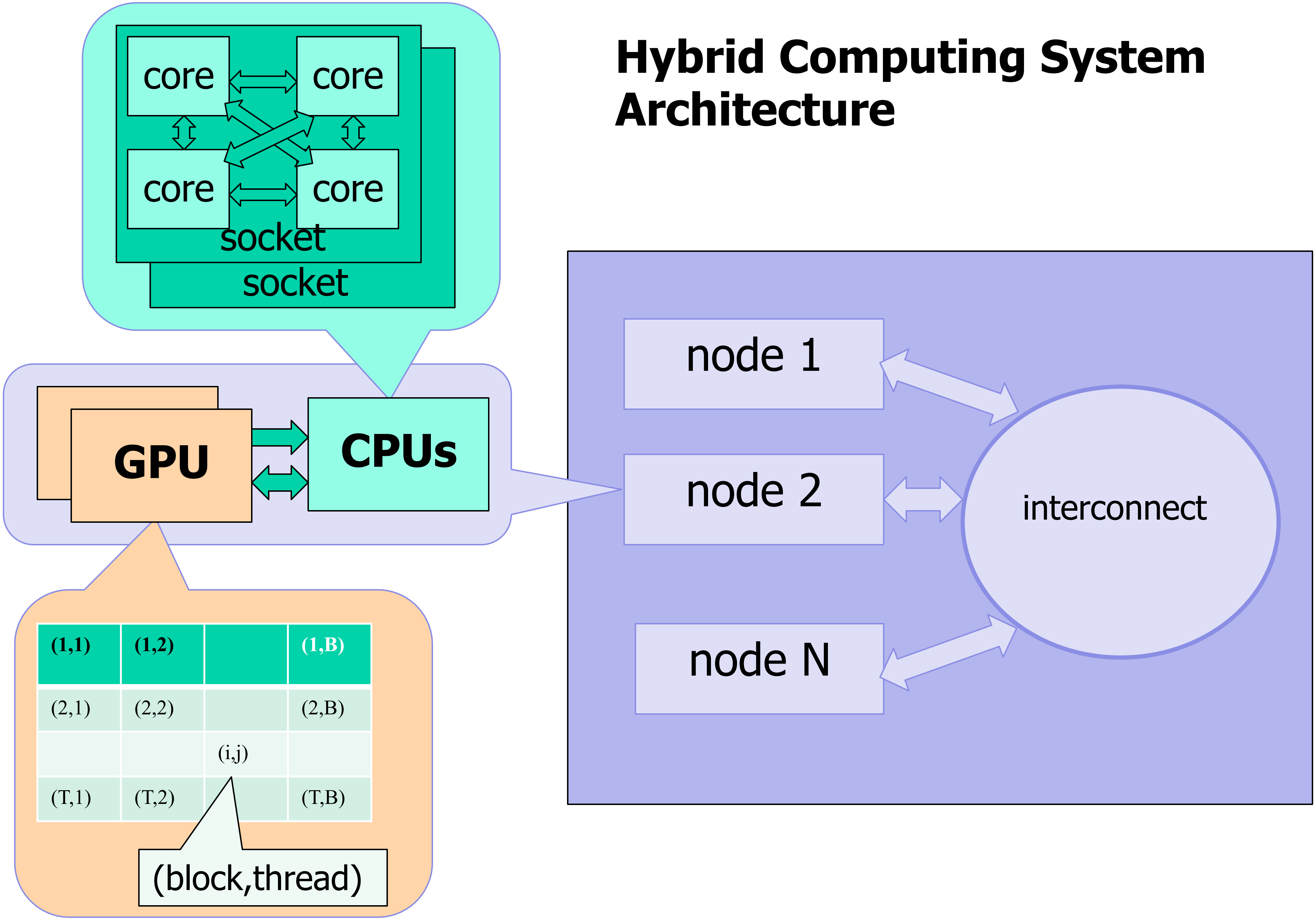
Titan:

OLCF supercomputer: 18,688 nodes: 16-core/node AMD Opteron 22.2GHz; unconventional NVIDIA Kepler Tesla K20X

Chester:

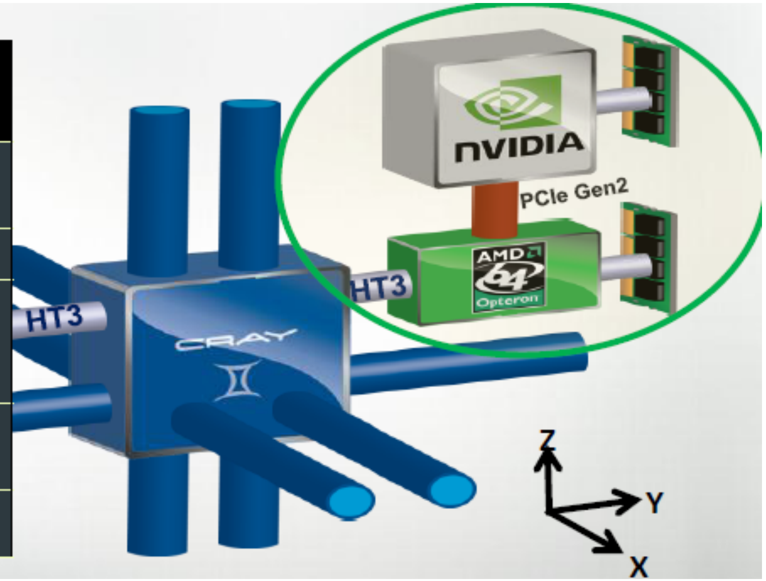
"test" version of Titan: 95 nodes

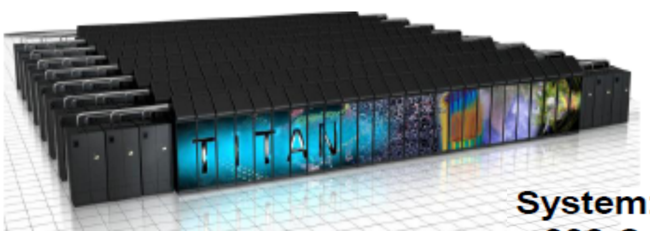
Hybrid Computing System Architecture



Titan: Cray XK7

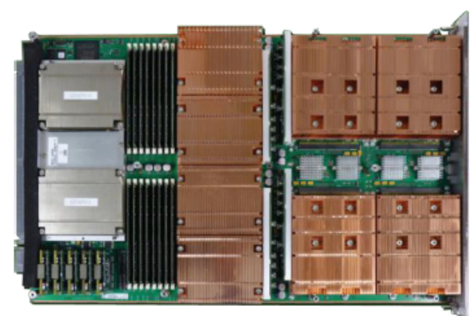
XK7 Compute Node Characteristics
AMD Opteron 6274 16 core processor @ 141 GF
Tesla K20x @ 1311 GF
Host Memory 32GB 1600 MHz DDR3
Tesla K20x Memory 6GB GDDR5
Gemini High Speed Interconnect





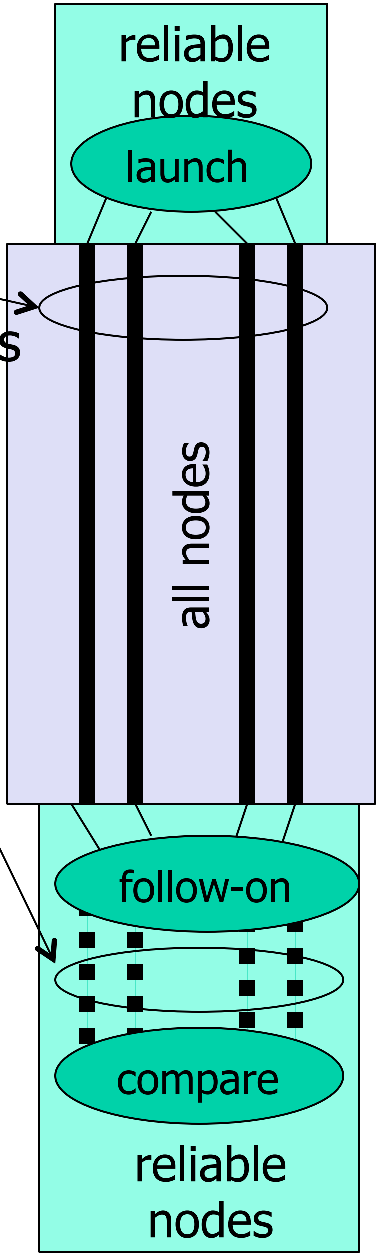
System:
200 Cabinets
18,688 Nodes
27 PF
710 TB

Cabinet:
24 Boards
96 Nodes
139 TF
3.6 TB



Board:
4 Compute Nodes
5.8 TF
152 GB

Overall Detection Approach



Chaotic-Map Method:

- Compute chaotic maps in parallel on “all” nodes and paths
- Compute follow-on maps on “reliable” nodes

Implementations: system specific

- Multi-core systems: ptherads
- GPUs: CUDA C block-threads
- Multi-node CPU+GPU systems:
  - threads+CUDA+MPI

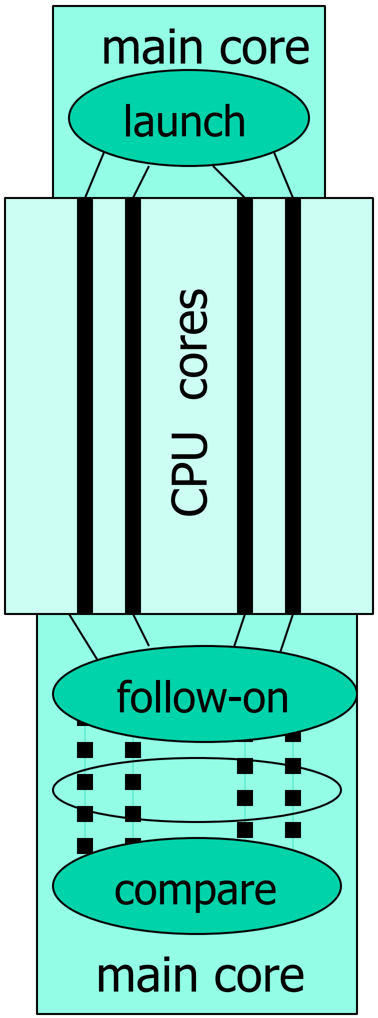
- **Detection**: “errors” amplified by chaotic maps:
  - in-situ
  - follow-on computations

- Diagnosis: may require additional codes

Implementation: Single Nodes

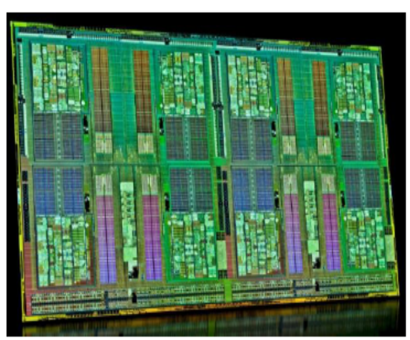
Multi-Core Node:

•pthreads: chaotic map trajectory on every core



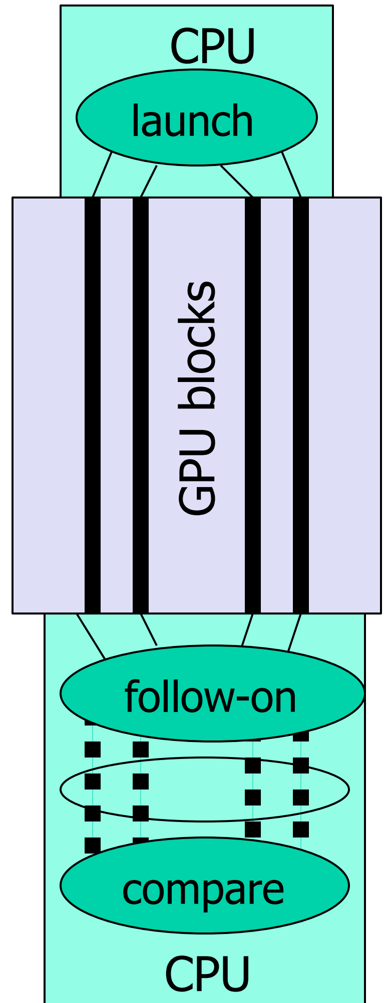
AMD Opteron 6274

- 16 cores
- 141 GFLOPs peak



GPU Accelerator:

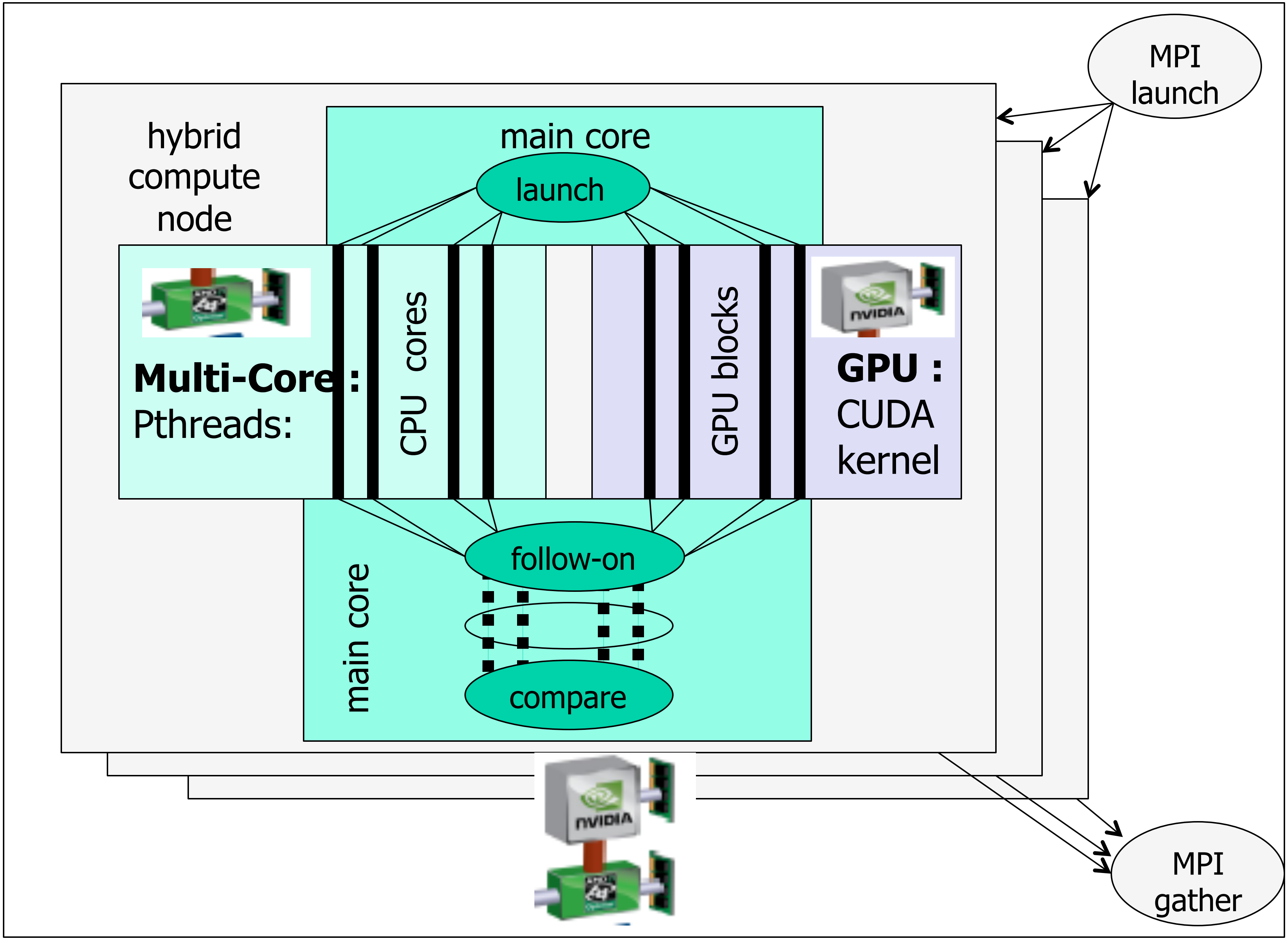
•CUDA C kernel: chaotic map threads on every block



NVIDIA Tesla K20x

- 14 Streaming Multiprocessors
- 2,688 CUDA cores
- 1.31 TFLOPs peak (DP)
- 6 GB GDDR5 memory
- HPL: >2.0 GFLOPs per Watt (Titan full system measured power)

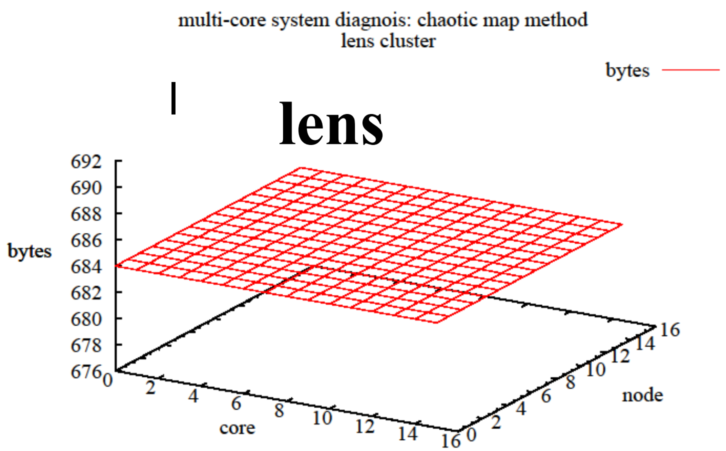
Implementation: Hybrid Systems



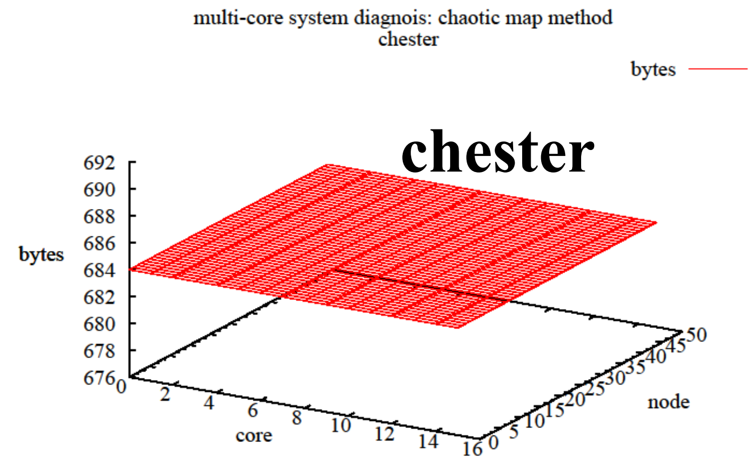
CPU Multi-Core Results Summary

All CPU chaotic-map output results match:

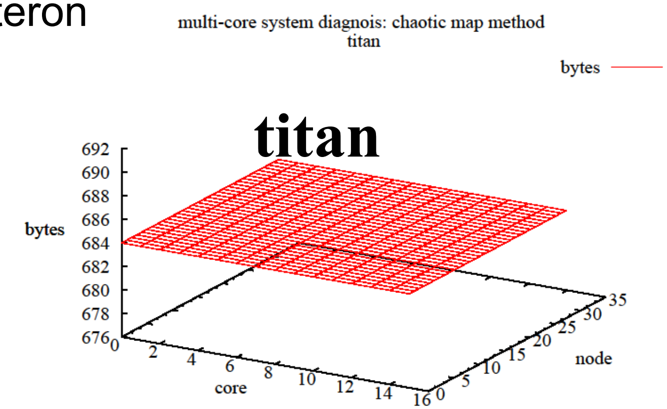
- Match to the bit on AMD Opteron and Intel cores
- Floating point operations are IEEE 754 compliant



16-core/node
2.3 GHz AMD Opteron



16-core/node
AMD Opteron 22.2GHz



16-core/node
AMD Opteron 22.2GHz

GPU Computations:

Different GPU blocks of same GPU producing different answers in some cases:

- Observed when integer and fractional variables are mixed on GPU blocks
- Observed on multiple GPUs, and repeatable
- Implications are not entirely understood – potentially destabilize certain non-linear computations

Example run: titan

Tesla K20X

```
I have no name!@nid06983:/tmp/work/nrao> ./diag_gpu_titan
Device Name: Tesla K20X
[deviceProp.major.deviceProp.minor] = [3.5]
multi-processor count = 14
warp_size = 32
cudaGetDevice()=0

 CPU:      Number of cores detected=16

 GPU:      Number of threads=100;      Number of blocks=50
 chaotic map:      x=0.200000;      l=4.000000;      n=10000

GPU: Chaotic Map
        block_x[0]=    0.682320  <-> 3F2EAC8E
        block_x[1]=    1.682320  <-> 3F2EAC8E
        block_x[2]=    2.682321  <-> 3F2EAC90
        block_x[3]=    3.682321  <-> 3F2EAC90

        block_x[13]=13.682321  <-> 3F2EAC90
        block_x[14]=14.682321  <-> 3F2EAC90
        block_x[15]=15.682321  <-> 3F2EAC90
        block_x[16]=16.682320  <-> 3F2EAC80
        block_x[17]=17.682320  <-> 3F2EAC80
```

Output: fractional part is Chaotic-map state
-not identical across the blocks of same GPU

may “appear” same under C printf but different

GPU Computations: follow-on chaotic map trajectory

Example run: titan

I have no name!@nid06983:/tmp/work/nrao> ./diag_gpu_titan

GPU: Chaotic Map	Follow-on Chaotic Map	Follow-on linear Map
block_x[0]=0.682320 <-> 3F2EAC8E	block_x[0]=0.682320 <-> 0.860477	block_x[0]=0.682320 <-> 0.000016
block_x[1]=1.682320 <-> 3F2EAC8E	block_x[1]=1.682320 <-> 0.860477	block_x[1]=1.682320 <-> 0.000016
block_x[2]=2.682321 <-> 3F2EAC90	block_x[2]=2.682321 <-> 0.000000	block_x[2]=2.682321 <-> 0.000016
block_x[3]=3.682321 <-> 3F2EAC90	block_x[3]=3.682321 <-> 0.000000	block_x[3]=3.682321 <-> 0.000016
block_x[4]=4.682321 <-> 3F2EAC90	block_x[4]=4.682321 <-> 0.000000	block_x[4]=4.682321 <-> 0.000016
block_x[5]=5.682321 <-> 3F2EAC90	block_x[5]=5.682321 <-> 0.000000	block_x[5]=5.682321 <-> 0.000016
block_x[6]=6.682321 <-> 3F2EAC90	block_x[6]=6.682321 <-> 0.000000	block_x[6]=6.682321 <-> 0.000016
block_x[7]=7.682321 <-> 3F2EAC90	block_x[7]=7.682321 <-> 0.000000	block_x[7]=7.682321 <-> 0.000016
block_x[8]=8.682321 <-> 3F2EAC90	block_x[8]=8.682321 <-> 0.000000	block_x[8]=8.682321 <-> 0.000016
block_x[9]=9.682321 <-> 3F2EAC90	block_x[9]=9.682321 <-> 0.000000	block_x[9]=9.682321 <-> 0.000016
block_x[10]=10.682321 <-> 3F2EAC90	block_x[10]=10.682321 <-> 0.000000	block_x[10]=10.682321 <-> 0.000016
block_x[11]=11.682321 <-> 3F2EAC90	block_x[11]=11.682321 <-> 0.000000	block_x[11]=11.682321 <-> 0.000016
block_x[12]=12.682321 <-> 3F2EAC90	block_x[12]=12.682321 <-> 0.000000	block_x[12]=12.682321 <-> 0.000016
block_x[13]=13.682321 <-> 3F2EAC90	block_x[13]=13.682321 <-> 0.000000	block_x[13]=13.682321 <-> 0.000016
block_x[14]=14.682321 <-> 3F2EAC90	block_x[14]=14.682321 <-> 0.000000	block_x[14]=14.682321 <-> 0.000016
block_x[15]=15.682321 <-> 3F2EAC90	block_x[15]=15.682321 <-> 0.000000	block_x[15]=15.682321 <-> 0.000016
block_x[16]=16.682320 <-> 3F2EAC80	block_x[16]=16.682320 <-> 0.671719	block_x[16]=16.682320 <-> 0.000016
block_x[17]=17.682320 <-> 3F2EAC80	block_x[17]=17.682320 <-> 0.671719	block_x[17]=17.682320 <-> 0.000016
block_x[18]=18.682320 <-> 3F2EAC80	block_x[18]=18.682320 <-> 0.671719	block_x[18]=18.682320 <-> 0.000016
block_x[19]=19.682320 <-> 3F2EAC80	block_x[19]=19.682320 <-> 0.671719	block_x[19]=19.682320 <-> 0.000016
block_x[20]=20.682320 <-> 3F2EAC80	block_x[20]=20.682320 <-> 0.671719	block_x[20]=20.682320 <-> 0.000016
block_x[21]=21.682320 <-> 3F2EAC80	block_x[21]=21.682320 <-> 0.671719	block_x[21]=21.682320 <-> 0.000016
block_x[22]=22.682320 <-> 3F2EAC80	block_x[22]=22.682320 <-> 0.671719	block_x[22]=22.682320 <-> 0.000016

Follow-on chaotic maps diverge significantly

follow-on linear maps May “absorb” the differences

CPU:
n_iter:10000: x_0:0.200000 l:4.000000, x_n:0.682320
logistic_map:0.682320 <-> 0.860477
linear_map 0.682320 <-> 0.000016
x_n=3F2EAC8E

Operational "Artifacts" Discovered

Execution of diagnosis codes led to the discovery of "operational artifacts"

GPU-emulations and incorrect executions: code delays

- Unless explicitly tested for presence of GPUs, codes may be
  - executed in "emulated mode": long execution times
  - incorrectly executed: incorrect results
- Resolved by explicitly checking for "physical" GPUs

Data transfers errors when MPI is used to launch CUDA kernels

- Outputs from certain blocks has zero fractional part:
  - Happens randomly but always the GPU block number matches the node number
- Implications are not entirely understood – potentially destabilize certain non-linear computations

Simulation Results

We simulate three types of errors:

i. ALU errors corrupt state by a multiplier
   - bit flip to 1 in ALU registers

ii. memory errors clamp state to a fixed value
   - stuck-at fault in RAM

iii. cross-connect errors modify state by a multiplier.
   - link transmission error

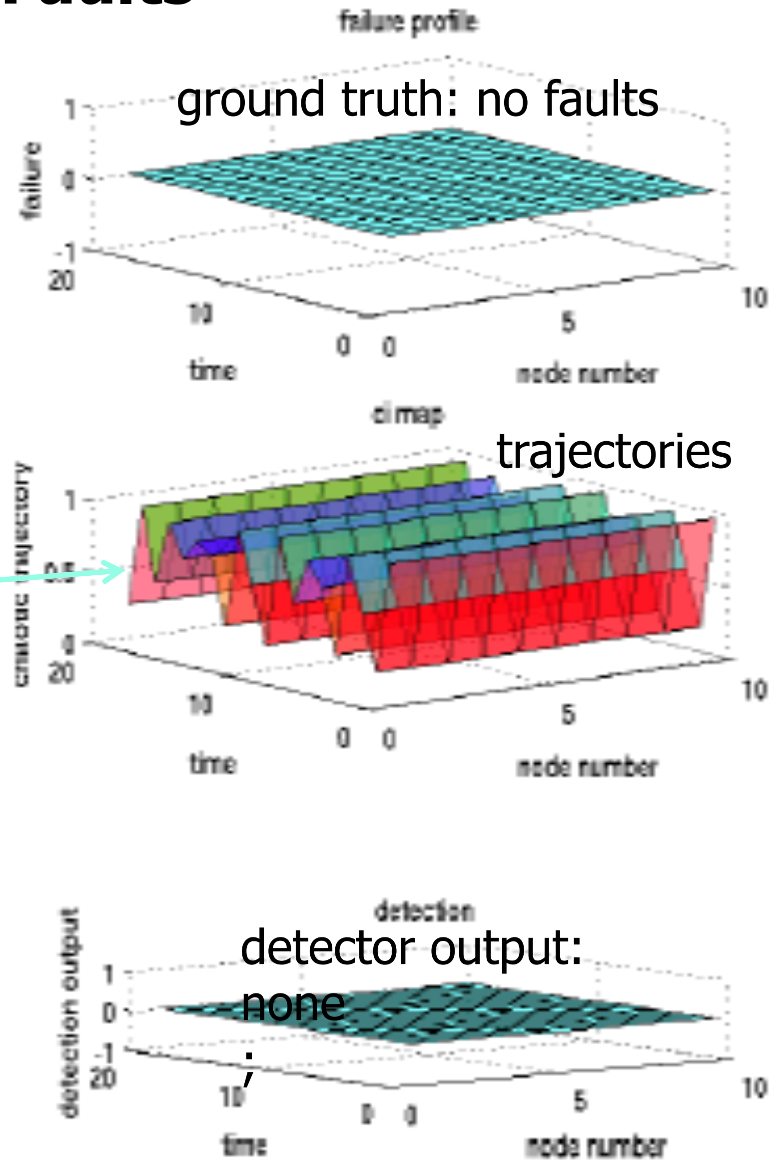
Nodes transition to a faulty mode with probability *p*, and once transitioned

- errors type (i) and (ii) are permanent,
- error type (iii) lasts only for a single time step

Simulation Results: No Faults

Case of no faults:
10-node pipeline of depth *k = 10*

- none are detected
- all chaotic time traces are identical across nodes

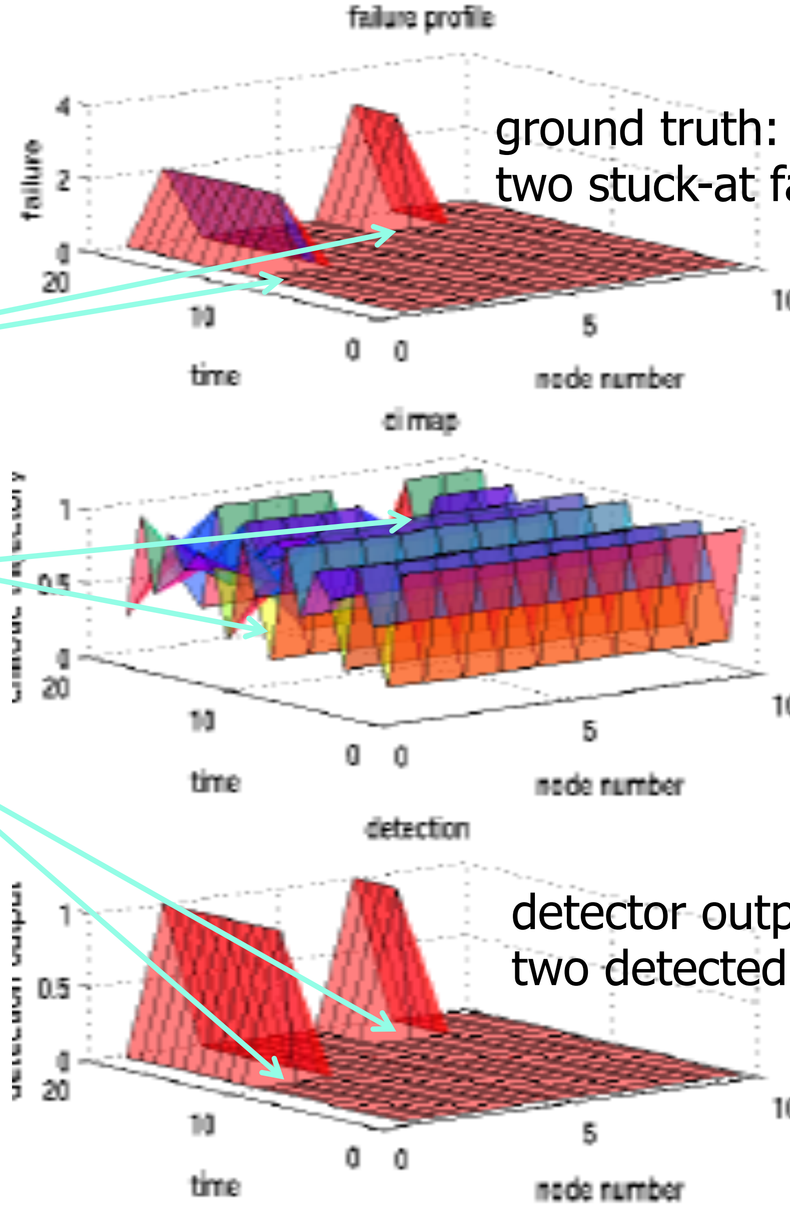


(a) no failures

Simulation Results

Stuck-at faults:
- full pipeline, spanning all 10 nodes
- trajectories disrupted by faulty nodes

- detection within one time step



ground truth:
two stuck-at faults

detector output:
two detected

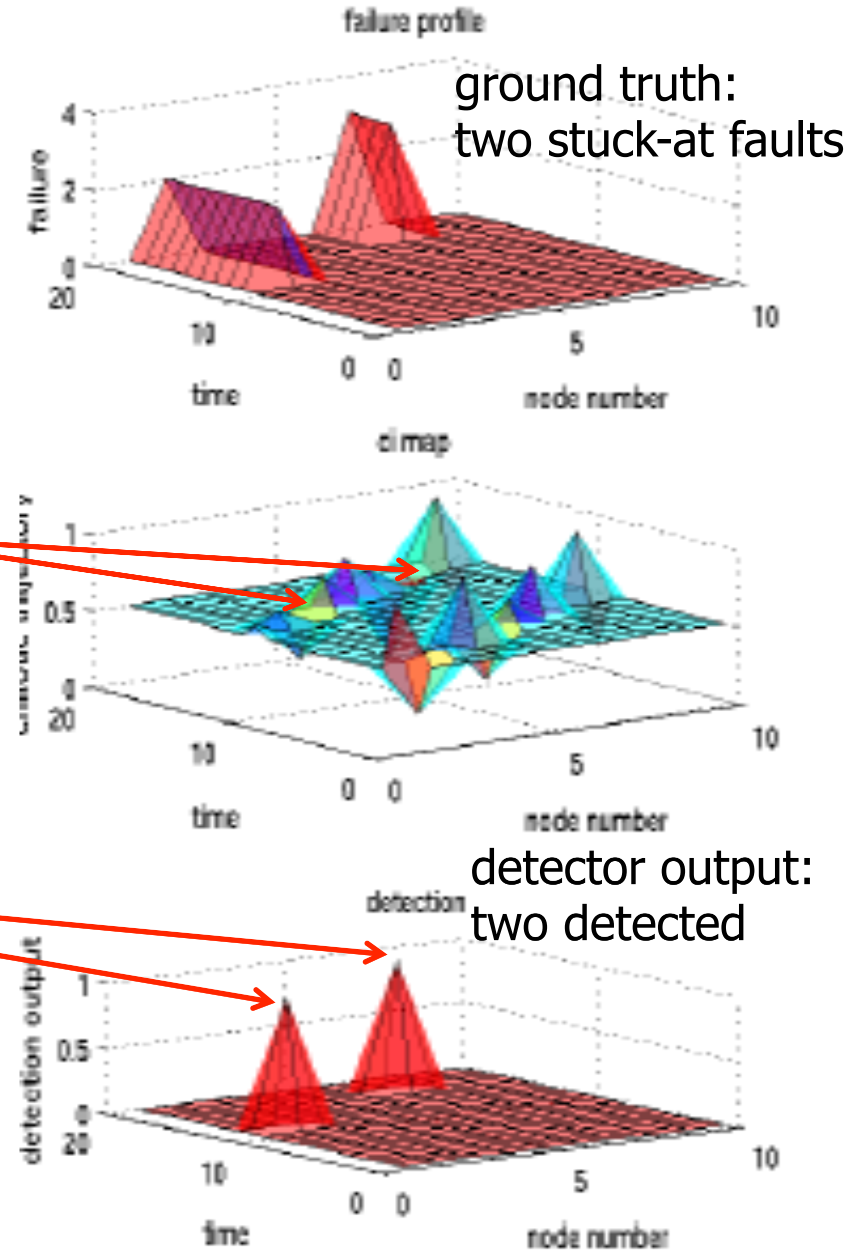
(b) full pipeline for stuck-at failures

Simulation Results

Pipeline of single chain
- executed by one node at time
- chain "sweeps" across nodes in time

Both faults are detected:
- detection delayed until the chain reaches faulty node

The total computational cost:
- 1/10 of the case (b)
- detection achieved, albeit delayed by few time steps



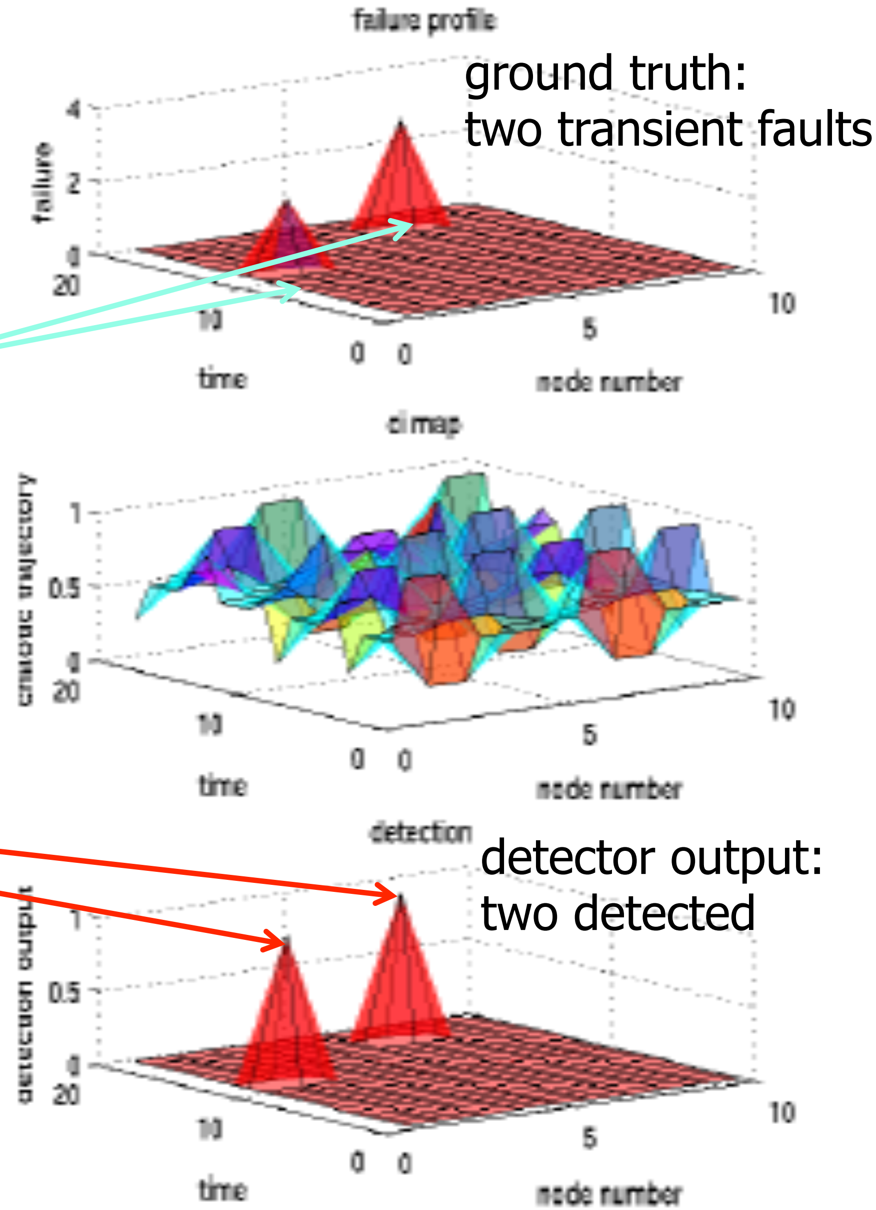
c) sparse pipeline for stuck-at failures

Simulation Results

Transient fault in interconnect payload lasted for one time unit

Full pipeline spanning all nodes will detect such failure

Pipeline of two chains with periodicity of 5 nodes is able to detect



(d) transient failure

Simulation System

Simulations on 48-core Linux workstation: 2.23GHz AMD Opteron processors

Computation on a single processor core and delay of 10 micro seconds to simulate the latency of interconnect.

- $N = 500{,}000$ nodes: runtimes under 2 seconds for
  - logistic map and a pair of reciprocal operations (5 operations for CI-map).

First-order approximation: for CI-map

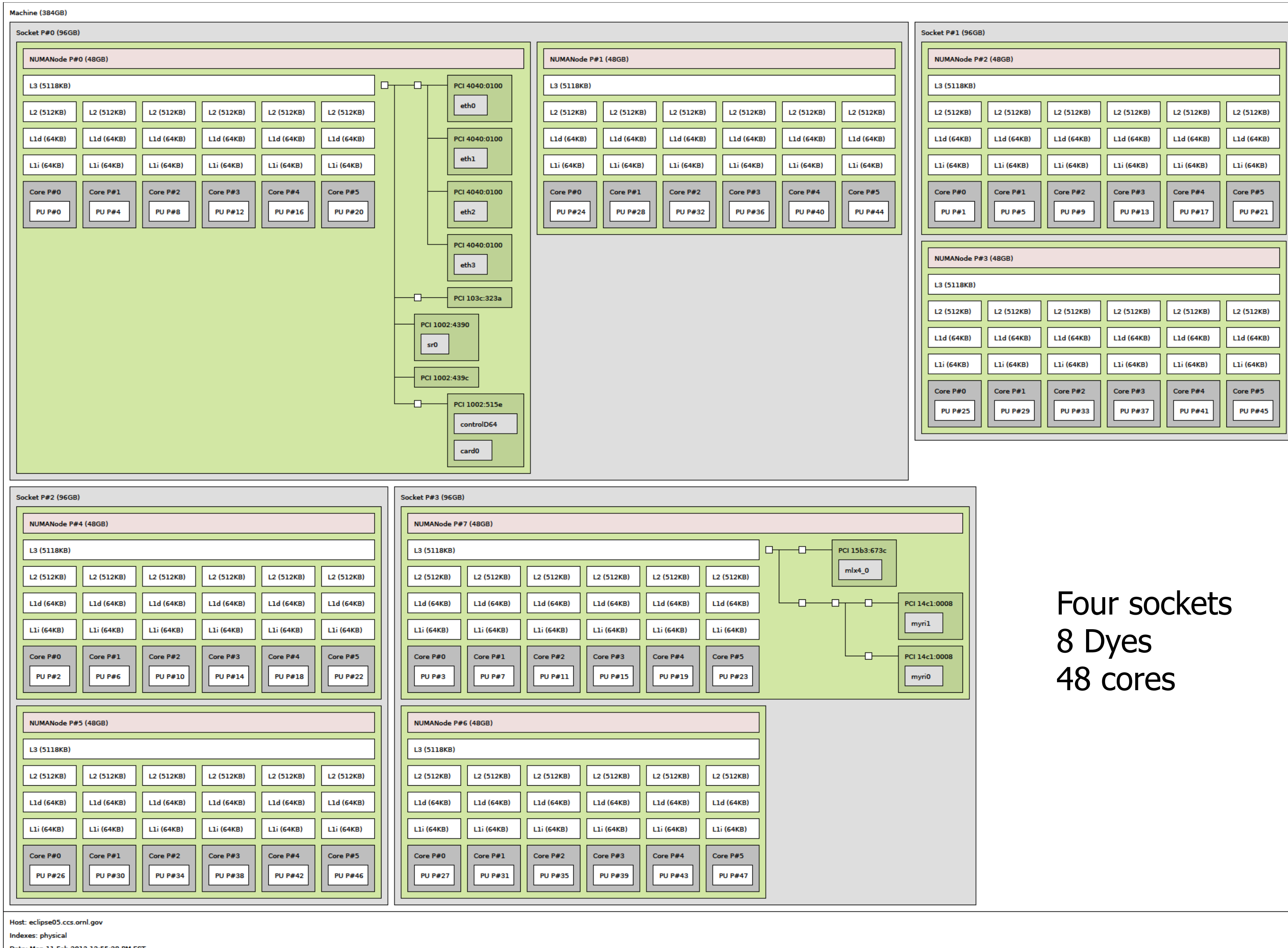
- 10 operations each with 10 micro seconds execution time, and
- interconnect with 10 microsecond latency

pipeline execution time is 11 seconds for $N=100{,}000$

All chains of PCC^2 -map are computed in parallel

- execution time scales linearly in N
- under 2 minutes for million computing nodes

HP Proliant 48-core Linux workstation: 2.23GHz AMD Opteron processors



Four sockets
8 Dyes
48 cores

Diagnosis output

HP Proliant 48-core Linux workstation:
2.23GHz AMD Opteron processors

```
System times:
        user time:  9998.000000 useconds
        kernel time :  23996.000000 useconds
Diagnosis Summary:
        Core 0:  output:  0.000000
        Core 1:  output:  0.492877
        Core 2:  output:  0.076975
        Core 3:  output:  0.932237
        Core 4:  output:  0.932237
        Core 5:  output:  0.932237
        ...
        Core 41:  output:  0.932237
        Core 42:  output:  0.932237
        Core 43:  output:  0.932237
        Core 44:  output:  0.932237
        Core 45:  output:  0.932237
        Core 46:  output:  0.932237
        Core 47:  output:  0.932237
```

simulated errors

no errors

System Profiling and Application Tracing

System Diagnosis and Profiling:

- executed at the beginning for an initial system profile
  - repeated periodically or triggered by failure events.
- typically, all system resources are devoted for initial profiling

- our method:
  - execute diagnosis modules customized to static and silent failures in processing nodes, memory units and interconnects
  - generate robustness estimates from outputs of diagnosis modules.

Application Tracing:

- diagnosis modules are strategically inserted into application codes
  - during compilation or preprocessing
- confidence measures are estimated for their outputs.

Basic idea: execution paths of these tracer codes “follow” along the same components as the application codes:

- processing nodes, memory elements and interconnect links,

Require “new” detection, profiling and tracing theory and algorithms:

Failure detection: schedule application around, replace nodes

Failure likelihood: set application fault tolerance, estimate confidence

Our Approach

Our approach: synthesis of methods from fault diagnosis, chaotic Poincare maps, and statistical estimation:

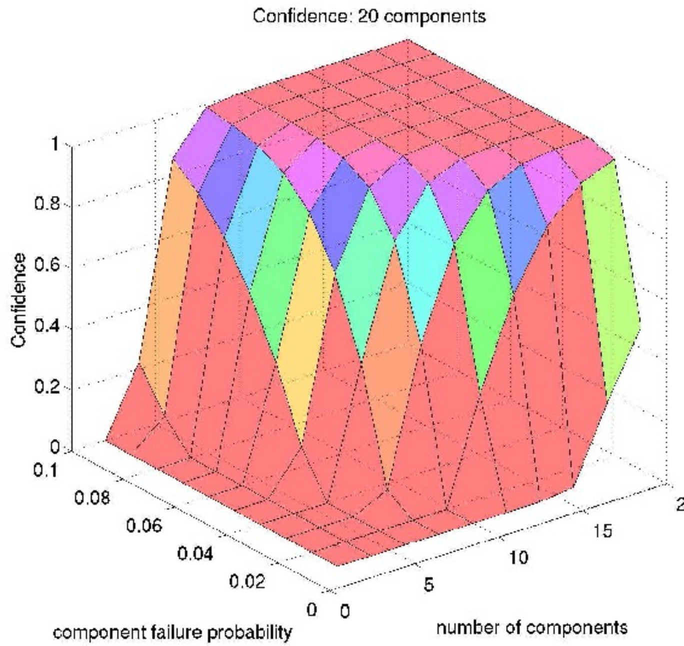
a) **Diagnosis methods:** identify computation errors due to component failures, in arithmetic and logic unit (ALU), memory and cross-connect, by strategically guiding the execution paths:
   i. system diagnosis pipelines
   ii. application traces

b) **Poincare maps** amplify effects of component failures making them quickly detectable,

c) **Statistical estimation** methods process data from execution traces to generate
   i. system robustness profiles
   ii. confidence estimates for applications

Confidence Estimates

Outputs of CI-maps are used to generate confidence measures for executions, particularly if no failures are detected



$I_D(.);M(.)$ executed at rate R_P
- once every $1/R_P$ seconds

P_{1/R_P} probability of node failure during $1/R_P$ sec

Under statistical independence
probability of failure during N_P executions

$$1-\left(1-P_{1/R_P}\right)^{N_P}$$

Confidence: $C(\alpha, N_P)$
that node failure probability is less than α

If no failures are detected in N_P executions

$$C(\alpha, N_P) = P\left\{P_{1/R_P} < \alpha\right\} > 1-2^{-2\left[1-(1-\alpha)^{N_P}\right]^2 N_P}$$

Confidence Estimate for Triplicated Application

Application triplicated with majority vote at each step:

- error-free under single faults
- makes error if there are two or more faults within "unit" time T_U

Application executed for duration T with application tracing detecting $\hat{N}_T$ faults:

if two or more faults detected within "unit" time: check-point

if single are no fault detected in all unit times:

confidence that application is error-free

$$C(T,\alpha) = 1 - P\left\{N_{T_U} > 1\right\} > 1 - \left(\frac{\hat{N}_T}{T} + \alpha\right)$$

with probability $\delta = 1 - ae^{-b\alpha^2 T^2}$

under statistically independent component failures

Qualitatively, confidence

- improves with lower number of faults detected
- improves with longer tracing period:
  - longer T means higher δ

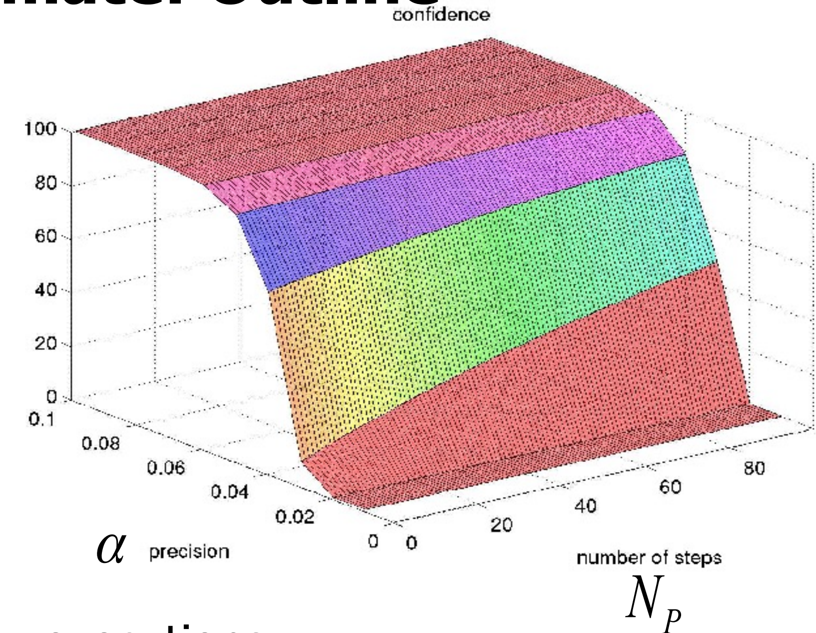
Note: zero errors do not imply 100% confidence

Derivation of Confidence Estimate: Outline

By Hoeffding's Inequality we have

$$P\left\{\left|1-\left(1-P_{1/R_P}\right)^{N_P}\right|>\in\right\}<2e^{-2\in^2 N_P}$$

$$P\left\{P_{1/R_P}<\alpha\right\}>1-2e^{-2\left[1-(1-\alpha)^{N_P}\right]^2 N_P}$$



General Confidence Estimate:

If failures are detected in $\hat{P}_E$ fraction of N_P executions

General confidence estimate:

$$C\left(\alpha,N_P\right)=P\left\{P_{1/R_P}<\alpha\right\}>1-2e^{-2\left[1-(1-\alpha)^{N_P}-\hat{P}_E\right]^2 N_P}$$

Derivation: By Hoeffding's Inequality we have

$$P\left\{\left|\left(1-P_{1/R_P}\right)^{N_P}-\hat{P}_E\right|>\in\right\}<2e^{-2\in^2 N_P}$$

$$P\left\{\left|P_{1/R_P}-\hat{P}_E\right|<\beta\right\}>1-2e^{-2\left[1-(1-\beta)^{N_P}\right]^2 N_P}$$

Confidence Estimate for Replicated Application: General Case

Application replicated $2\gamma + 1$ times with majority vote at component level:

- error-free under γ faults or fewer faults
- makes error if there are $\gamma + 1$ or more faults within "unit" time

Application executed for duration T with application tracing detecting $\hat{N}_T$ faults

if two or more faults detected within "unit" time: check-point

if single are no fault detected in all unit times:

confidence that application is error-free

$$C(T, \gamma, \in) = 1 - P\{N_U > \gamma\} > 1 - \left(\frac{\hat{N}_T}{T\gamma} + \frac{\alpha}{\gamma} \right)$$

with probability $\delta = 1 - ae^{-b\alpha^2 T^2}$

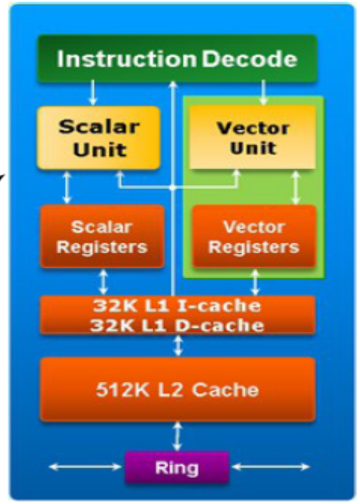
under statistical independence of component failures

Qualitatively, confidence

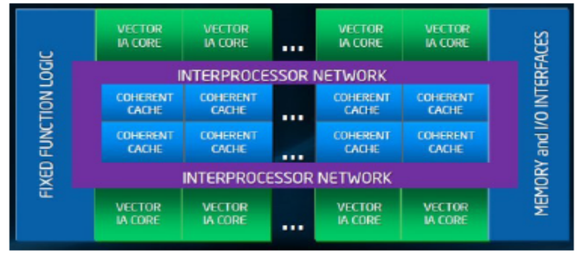
- improves with lower number of faults detected
- improves with longer tracing period
- also, imrpoves with replication level

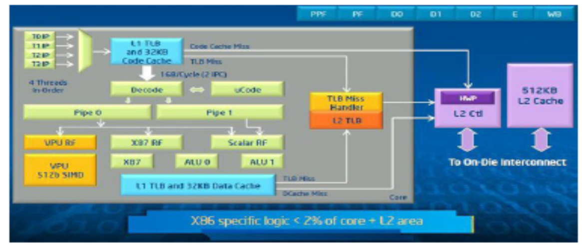
Xeon Phi and GPU Architectures

Architecture and core definition



Xeon Phi core



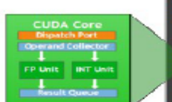


Xeon Phi core

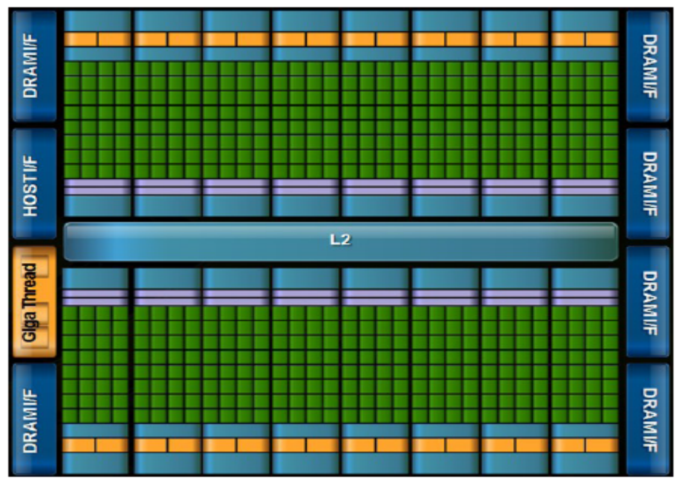
1 to 1.3 GHz

1 SPU
- 1 double op/cycle
- In-order architecture
- x86 + mic extensions
- 4 hardware threads

1 VPU
- 32 float op/cycle
- 16 double op/cycle
- Supports fused mult-add
- Supports transcendentals
- 4 clock latency
- 4 hardware threads



CUDA core



nVidia Kepler SMX

735 to 745 MHz

192 SP CUDA cores
- 2 double op/cycle
- Supports fused mult-add

64 DPUnits
- 2 double op/cycle
- Supports fused mult-add

32 SFU units
- 1 double op/cycle
- Supports transcendentals

Execution Path – Xeon Phi

- Different compiler switches exercise different parts of hardware
  - $ icc -mmic diag_multicore_light.c (default)

    Core 0: output: 0.940222 : 3E2F8EBE
    …
    Core 227: output: 0.940222 : 3E2F8EBE
  - $ icc -mmic diag_multicore_light.c –no-vec

    Core 0: output: 0.940222 : 3E2F8EBE
    …
    Core 227: output: 0.940222 : 3E2F8EBE
  - $ icc -mmic diag_multicore_light.c -fimf-precision=high

    Core 0: output: 0.940222 : 3E2F8EBE
    …
    Core 227: output: 0.940222 : 3E2F8EBE
  - $ icc -mmic diag_multicore_light.c -fimf-arch-consistency=true

    Core 0: output: 0.936652 : 5E46ED57
    …
    Core 227: output: 0.936652 : 5E46ED57
  - $ icc -mmic diag_multicore_light.c -fp-model strict

    Core 0: output: 0.932237 : 938210F1
    …
    Core 227: output: 0.932237 : 938210F1
  - $ icc -mmic diag_multicore_light.c -fp-model precise -fp-model source

    Core 0: output: 0.932237 : 938210F1
    …
    Core 227: output: 0.932237 : 938210F1

Agreement with xeon and opteron

Conclusions

Our approach

(i) utilizes light-weight computations based on chaotic and identity maps to detect certain classes of errors in computations, and

(ii) implementation for diagnosis of multi-core processors, GPUs, and hybrid systems

- tested on three hybrid systems:
    - 4 multi-core processors
    - 4 GPUs

This approach is suitable for exascale systems:

(a) low computational requirements

(b) linear scaling of the execution time

both for system profiling and application tracing

Future Work:

- These results are only a very first step
    - Implementations for high-performance machines and clusters
    - Incorporation of failure classes and application footprints
- More analysis and simulations needed
    - understand and quantify classes of errors detected by a given set of Poincare and identity maps

References

Conference Papers

- N. S. V. Rao, Fault detection in multi-core processors using chaotic maps, 3rd Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS 2013), 2013.
- N. S. V. Rao, Resiliency in Exascale systems and computations using chaotic-identity maps, Workshop on Resiliency in High Performance Computing in Clusters, Clouds and Grids (Resilience 2012), 2012, extended abstract, invited talk.
- N. S. V. Rao, Chaotic-identity maps for robustness estimation of Exascale computations, 2nd Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS 2012), 2012.

Whitepapers

- N. S. V. Rao, Fault detection and profiling algorithms for exascale computing Systems, https://collab.mcs.anl.gov/display/examath/Submitted+Papers
- N. S. V. Rao, Confidence estimation for exascale computations, https://collab.mcs.anl.gov/display/examath/Submitted+Papers

Publications related to the topic

Fault diagnosis

- N. S. V. Rao and S. Toida, On polynomial-time testable combinational circuits, IEEE Transactions on Computers, vol. 43, no. 11, 1994, pp. 1298-1308.
- N. S. V. Rao, Expected-value analysis of two single fault diagnosis algorithms, IEEE Transactions on Computers, vol. 42, no. 3, 1993, pp. 272-280.
- N. S. V. Rao, Computational complexity issues in operative diagnosis of graph-based systems, IEEE Transactions on Computers, vol. 42, no. 4, 1993, pp. 447-457.

Chaotic Maps

- N. S. V. Rao, J. Gao, L. O. Chua, On dynamics of transport protocols in wide-area Internet connections, in Complex Dynamics in Communication Networks, L. Kocarev and G. Vattay (editors), 2005, pp. 69- 102.
- J. Gao, N. S. V. Rao, J. Hu, J. Ai, Quasi-periodic route to chaos in the dynamics of Internet transport protocols, Physical Review Letters, 2005.

Statistical Estimation

- N. S. V. Rao, Measurement-based statistical fusion methods for distributed sensor networks, in Distributed Sensor Networks, 2nd Edition, R. R. Brooks and S. S. Iyengar (editors), 2011, Chapman and Hall Publishers.

Thank you