

International Conference on Parallel Architectures and
Compilation Techniques (PACT)

Minneapolis, MN, Sep 21th, 2012

Probabilistic Diagnosis of Performance Faults in Large-Scale Parallel Applications

Ignacio Laguna,
Saurabh Bagchi

Dong H. Ahn, Bronis R. de Supinski,
Todd Gamblin



Lawrence Livermore National Laboratory

Slide 1/24



Developing Resilient HPC Applications is Challenging

MTTF of hours in Future Exascale Supercomputers

Faults come from:

Hardware
Software
Network

Software bugs from:

Application
Libraries
OS & Runtime system

Multiple manifestations:

Hangs, crashes
Silent data corruption
Applications is slower than usual



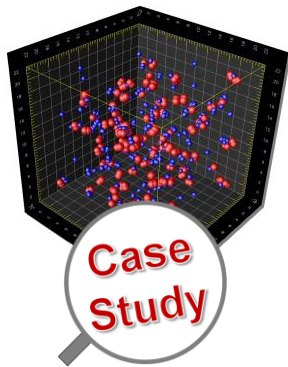
Lawrence Livermore National Laboratory

Slide 2/24



Some Faults Manifest Only at Large Scale

Molecular dynamics
simulation code (ddcMD)



Fault Characteristics

- Application hangs with 8,000 MPI tasks
- Only fails in Blue Gene/L
- Manifestation was intermittent
- Large amount of time spent on fixing the problem
- *Out technique isolated the origin of the problem in a few seconds*



Lawrence Livermore National Laboratory

Slide 3/24

PURDUE
UNIVERSITY

Why Do We Need New Debugging Tools?

Current Tools



- Old (breakpoint) technology (>30 years old)
- Manual process to find bugs
- Poor scalability

Future Tools



- Automatic problem determination
- Less human intervention in determining failure root cause
- Scalable (~millions of processes)



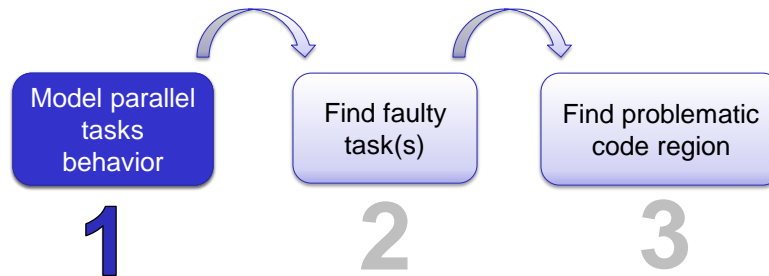
Lawrence Livermore National Laboratory

Slide 4/24

PURDUE
UNIVERSITY

Approach Overview

- We focus on pinpointing the origin of performance faults:
 - Application hangs
 - Execution is slower than usual
- Could have multiple causes:
 - Deadlocks, slower code regions, communication problems, etc.



Summarizing Execution History

- HPC applications generate a large amount of traces
- Use a probabilistic model to summarize traces
- We model control flow behavior of MPI tasks
 - Allow us to find the least progressed task



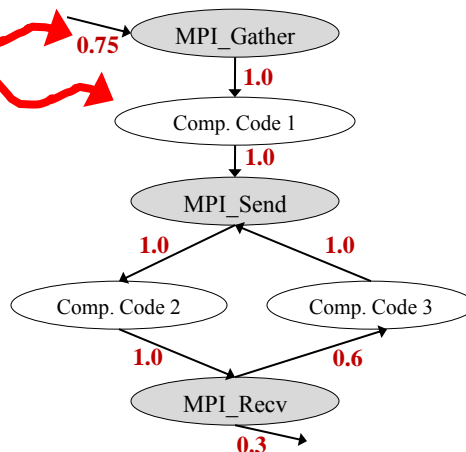
Each MPI Tasks is Modeled as a Markov Model

Sample code

```
foo() {
  MPI_gather( )
  // Computation code
  for (...) {
    // Computation code
    MPI_Send( )
    // Computation code
    MPI_Recv( )
    // Computation code
  }
}
```

MPI calls wrappers:
 - Gather call stack
 - Create states in the model

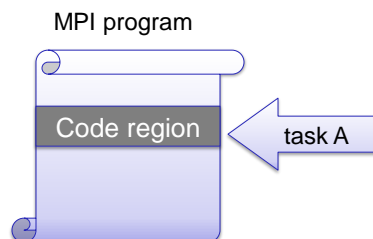
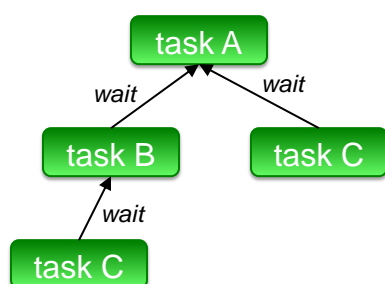
Markov Model



Approach Overview



The Progress Dependence Graph



- Facilitates finding the origin of performance faults
- Allows programmer to focus on the origin of the problem:
 - The *least progressed task*



What Tasks are Progress Dependent On Other Tasks?

Point-to-Point Operations

Task X:

```
// computation code...
MPI_Recv(..., taskY, ...)
// ...
```

- Task X depends on task Y
- Dependency can be obtained from MPI calls parameters and request handlers

Collective Operations

Task X:

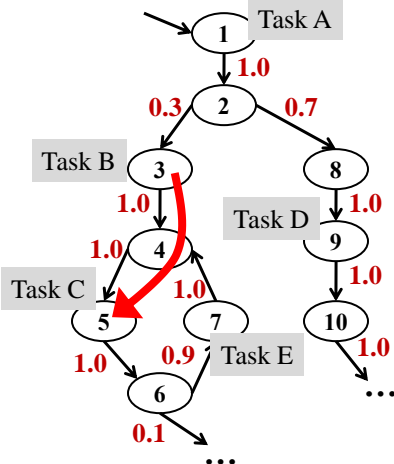
```
// computation code ...
MPI_Reduce(...)
// ...
```

- Multiple implementations (e.g., binomial trees)
- A task can reach MPI_Reduce and continue
- Task X could block waiting for another task (less progressed)



Probabilistic Inference of Progress-Dependence Graph

Sample Markov Model



Progress dependence between tasks B and C?

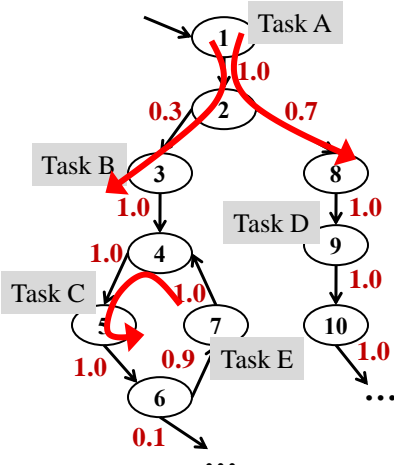
$Probability(3 \rightarrow 5) = 1.0$
 $Probability(5 \rightarrow 3) = 0$

Task C is likely waiting for task B
 (A task in 3 always reaches 5)

C has progressed further than B

Resolving Conflicting Probability Values

Sample Markov Model



Dependence between tasks B and D?

$Probability(3 \rightarrow 9) = 0$
 $Probability(9 \rightarrow 3) = 0$

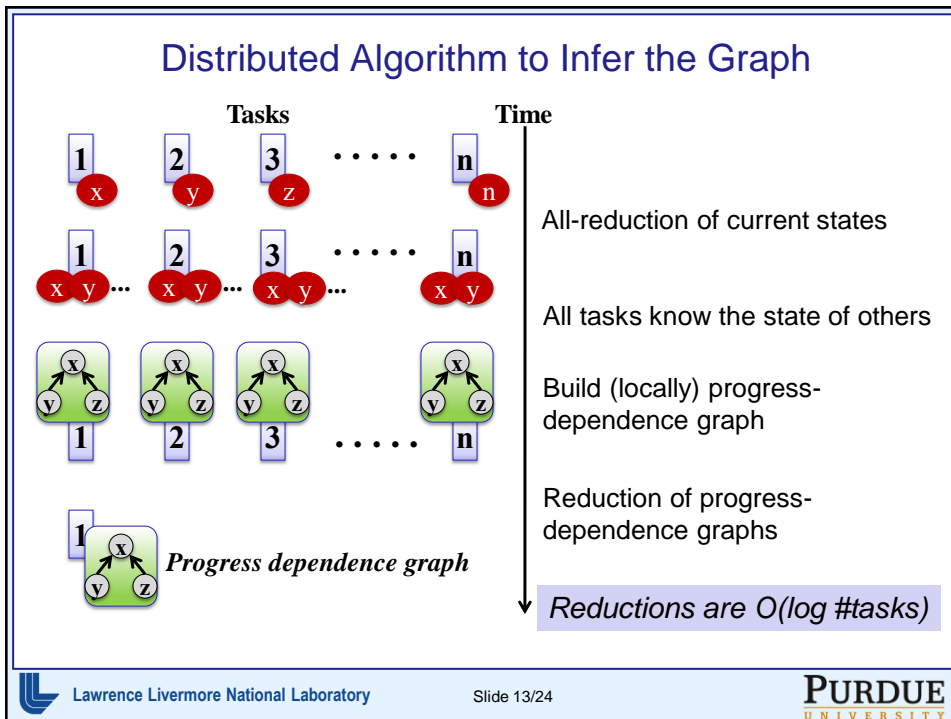
The dependency is null

Dependence between tasks C and E?

$Probability(7 \rightarrow 5) = 1.0$
 $Probability(5 \rightarrow 7) = 0.9$

Heuristic: Trust the highest probability

C is likely waiting for E



Reduction Operations: *Graph Dependencies Unions*

Examples of reduction operations
 $X \rightarrow Y$: *X is progress dependent on Y*

Task A	Task B	Result
$X \rightarrow Y$	$X \rightarrow Y$	$X \rightarrow Y$ (Same dependence)
$X \rightarrow Y$	Null	$X \rightarrow Y$ (First dominates)
$X \rightarrow Y$	$Y \rightarrow X$	Undefined (or Null)
Null	Null	Null

Lawrence Livermore National Laboratory Slide 14/24 PURDUE UNIVERSITY

Case Study

Bug Progress Dependence Graph

Hang with ~8,000 MPI tasks in BlueGene/L

```

graph BT
    T1([1-2047, 3073-3135, ...]) --> T2([0, 2048, 3072])
    T3([6841-7995]) --> T4([6840])
    T2 --> T5([3136])
    T4 --> T2
  
```

[3136] Least-progressed task

- Our tool finds that task 3136 is the origin of the hang
 - *How did it reach its current state?*

Lawrence Livermore National Laboratory Slide 15/24 PURDUE UNIVERSITY

Approach Overview

```

graph LR
    A[1 Model parallel tasks behavior] --> B[2 Find faulty task(s)]
    B --> C[3 Find problematic code region]
  
```

- 1 Model parallel tasks behavior
- 2 Find faulty task(s)
- 3 Find problematic code region

Lawrence Livermore National Laboratory Slide 16/24 PURDUE UNIVERSITY

Finding the Faulty Code Region: *Program Slicing*

Progress dependence graph

```

    graph BT
      Task3[Task 3] --> Task2[Task 2]
      Task4[Task 4] --> Task2
      Task2 --> Task1[Task 1]
      
```

Dyninst
Slicing Tool

```

done = 1;
for (...) {
  if (event) {
    flag = 1;
  }
}
if (flag == 1) {
  MPI_Recv();
  ...
}
...
if (done == 1) {
  MPI_Barrier();
}
    
```

Task 1 State

Task 2 State

Lawrence Livermore National Laboratory

Slide 17/24

Case Study

Code to Handle Buffered I/O in DDCMD

Writer:

```

Send() → Signals
Recv() ← Data
    
```

Non-Writer

```

Recv() ← Signal
Send() → Data
    
```


```

dataWritten = 0
for (...) {
  Probe(..., &flag,...)
  if (flag == 1) {
    Recv()
    Send()
    dataWritten = 1
  }
  Send()
  Recv()
  // Write data
}
if (dataWritten == 0) {
  Recv()
  Send()
}
Reduce()
Barrier()
    
```

Check if another writer asks for data

Lawrence Livermore National Laboratory

Slide 18/24



Case Study

Slice With Origin of the Bug


```

dataWritten = 0
for (...) {
  Probe(..., &flag,...)
  if (flag == 1) {
    Recv()
    Send()
    dataWritten = 1
  }
  Send()
  Recv()
  // Write data
}
if (dataWritten == 0) {
  Recv()
  Send()
}
Reduce()
Barrier()

```




Dual condition occurs in BlueGene/L

- A task is a writer and a non-writer


MPI_Probe checks for *source, tag and comm* of a message

- Another writer intercepted wrong message

Programmer used unique MPI tags to isolate different I/O groups






Least-progressed task State




Lawrence Livermore National Laboratory

Slide 19/24




Fault Injections Methodology

- Faults injected in Two Sequoia benchmarks:
 - AMG-2006
 - LAMMPS
- We injected a hang in random MPI tasks:
 - 20 user function calls, 5 MPI calls
 - Only injected in executed functions
 - Functions are selected randomly



Lawrence Livermore National Laboratory

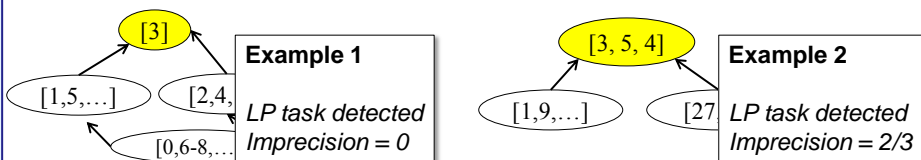
Slide 20/24



Accurate Detection of Least-Prog. Tasks

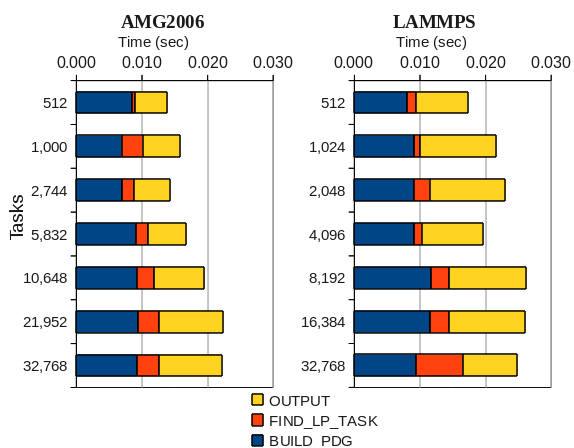
- Least-progressed task detection recall:
 - Cases when LP task is detected correctly
- Imprecision:
 - % of extra tasks in LP tasks set

Example Runs: 64 tasks, fault injected in task 3



- Overall results:
 - Average LP task detection recall is 88%
 - 86% of injections has imprecision of zero

Performance Results: Least-Prog. Task Detection Takes a Fraction Of A Second



Performance Results: *Slowdown Is Small For a Variety of Benchmarks*

Benchmark	Slowdown	Memory-usage Increase
LAMMPS	1.59	6.11
AMG2006	1.46	10.36
BT	1.08	3.75
SP	1.67	5.14
CG	1.14	2.21
FT	1.05	1.01
LU	1.39	5.37
MG	1.04	1.04

- Tested slowdown with NAS Parallel and Sequoia benchmarks
 - Maximum slowdown of ~1.67
- Slowdown depends on number of MPI calls from different contexts



Conclusions

- Our debugging approach diagnose faults in HPC applications
- *Novelties:*
 - *Compression of historic control-flow behavior*
 - *Probabilistic inference of the least-progressed tasks*
 - *Guided application of program slicing*
- Distributed debugging method is scalable
 - Takes fraction of a second with 32,000 BlueGene/P tasks
- Successful evaluation with hard-to-detect bug and representative benchmarks



Thank you!



Lawrence Livermore National Laboratory

Slide 25/24



Backup Slides



Lawrence Livermore National Laboratory

Slide 26/24



