CORVETTE: Program Correctness, Verification, and Testing for Exascale

PI: Koushik Sen, UC Berkeley
coPI: James W. Demmel, UC Berkeley
coPI: Costin Iancu, LBL

Students and Post-docs:
Cindy Rubio Gonzalez, Anh Cuong Nguyen, Hong Diep Nguyen
Chang-Seo Park, Xuehai Qian

Collaborators:
William Kahan (UC Berkeley), David Bailey (LBL), Wim Lavrijsen (LBL), David Hough (Oracle),

Vision: Reduce Programming to Debugging ratio

Vision: Reduce Programming to Debugging ratio

Domain

Hybrid Parallelism + Floating Point + Modularity

Advantages:

- Performance
- Scalability
- Abstraction
- Modularity

Disadvantages:

- non-deterministic bugs
- non-reproducible results
- redundant syncs
- redundant precision

Vision: Reduce Programming to Debugging ratio

Domain

Hybrid Parallelism + Floating Point + Modularity

Advantages:

- Performance
- Scalability
- Abstraction
- Modularity

Disadvantages:

- non-deterministic bugs
- non-reproducible results
- redundant syncs
- redundant precision

Goals:

Correctness/Performance tools to help programmers with development

1. efficient
2. scalable
3. reproducible
4. precise
5. coverage

Vision: Reduce Programming to Debugging ratio

Domain

Hybrid Parallelism + Floating Point + Modularity

Advantages:

- Performance
- Scalability
- Abstraction
- Modularity

Disadvantages:

- non-deterministic bugs
- non-reproducible results
- redundant syncs
- redundant precision

Goals:

Correctness/Performance tools to help programmers with development

1. efficient
2. scalable
3. reproducible
4. precise
5. coverage

Our Approach:

1. Dynamic analysis
2. Symbolic execution
3. New Algorithms

Vision: Reduce Programming to Debugging ratio

Domain

Hybrid Parallelism + Floating Point + Modularity

Advantages:

- Performance
- Scalability
- Abstraction
- Modularity

Disadvantages:

- non-deterministic bugs
- non-reproducible results
- redundant syncs
- redundant precision

Goals:

Correctness/Performance tools to help programmers with development

1. efficient
2. scalable
3. reproducible
4. precise
5. coverage

Our Approach:

1. Dynamic analysis
2. Symbolic execution
3. New Algorithms

Tools:

1. UP-Thrille: data races
2. Precimonious: FP precision tuning
3. ReproBLAS: reproducible num. algorithms

Vision: Reduce Programming to Debugging ratio

Domain

Hybrid Parallelism + Floating Point + Modularity

Advantages:

- Performance
- Scalability
- Abstraction
- Modularity

Disadvantages:

- non-deterministic bugs
- non-reproducible results
- redundant syncs
- redundant precision

Goals:

Correctness/Performance tools to help programmers with development

1. efficient
2. scalable
3. reproducible
4. precise
5. coverage

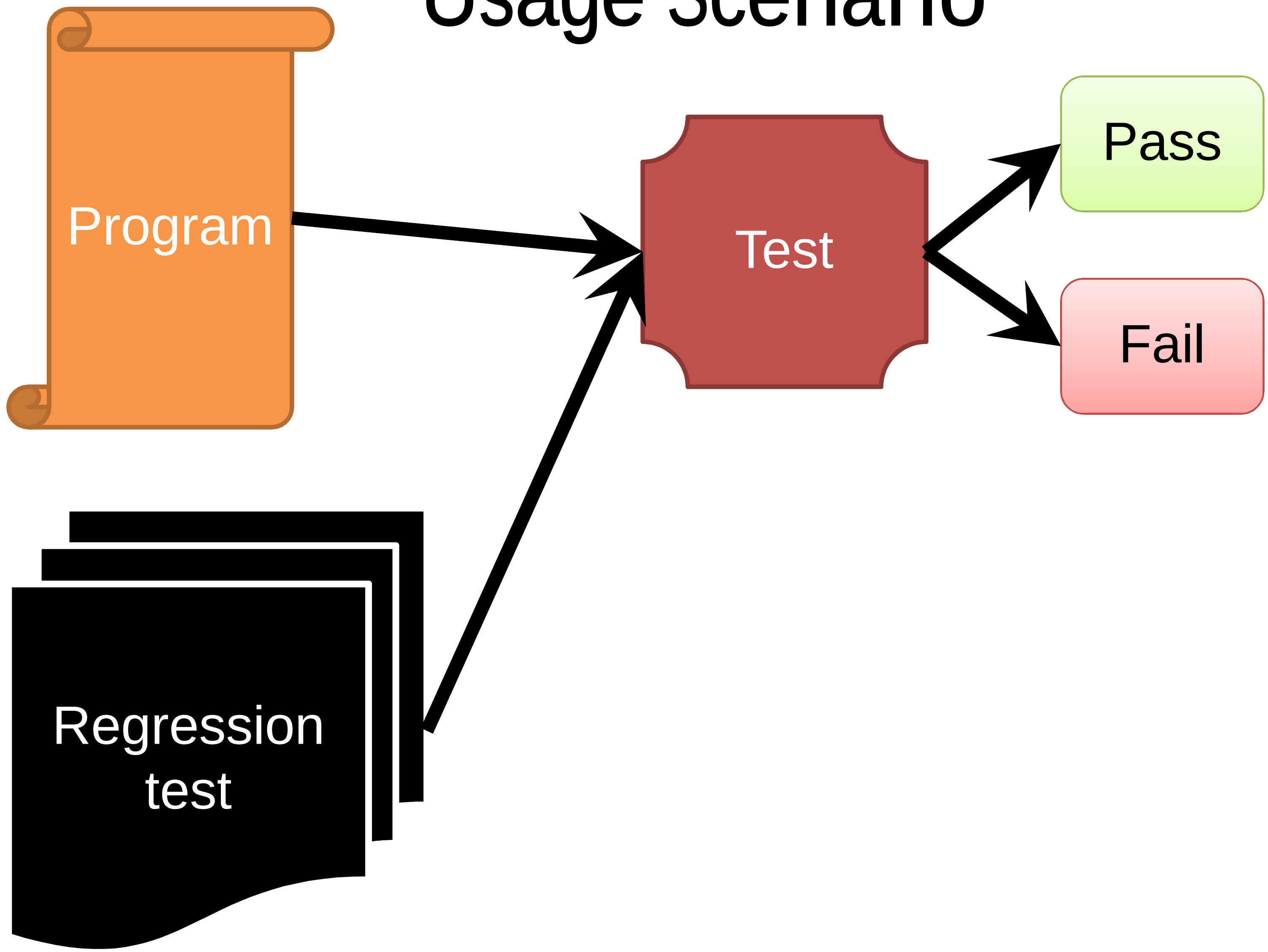
Our Approach:

1. Dynamic analysis
2. Symbolic execution
3. New Algorithms

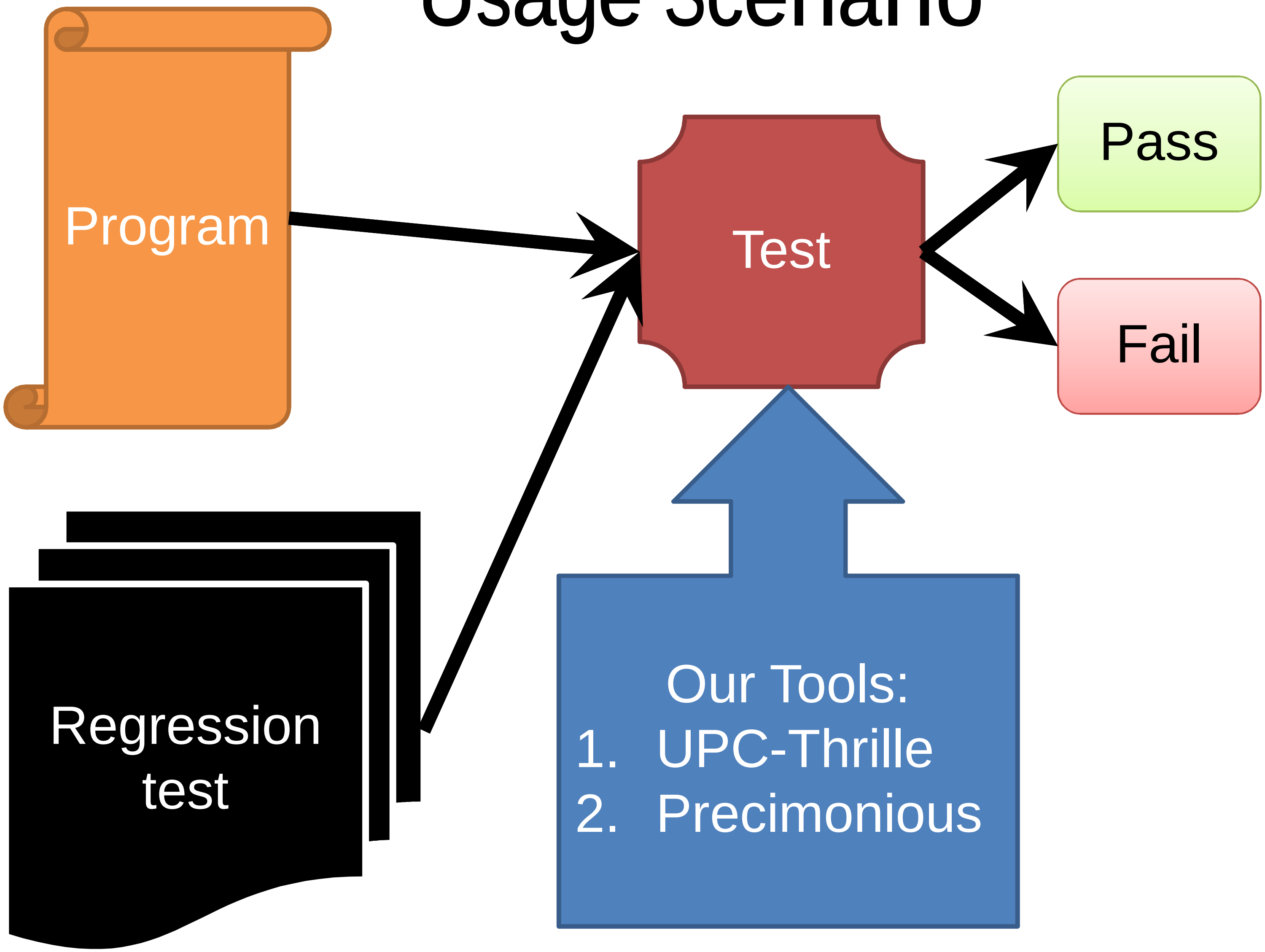
Tools:

1. UP-Thrille: data races
2. Precimonious: FP precision tuning
3. ReproBLAS: reproducible num. algorithms
4. LLVM Shadow Execution: dynamic analysis
5. Sync reduction: remove redundant syncs

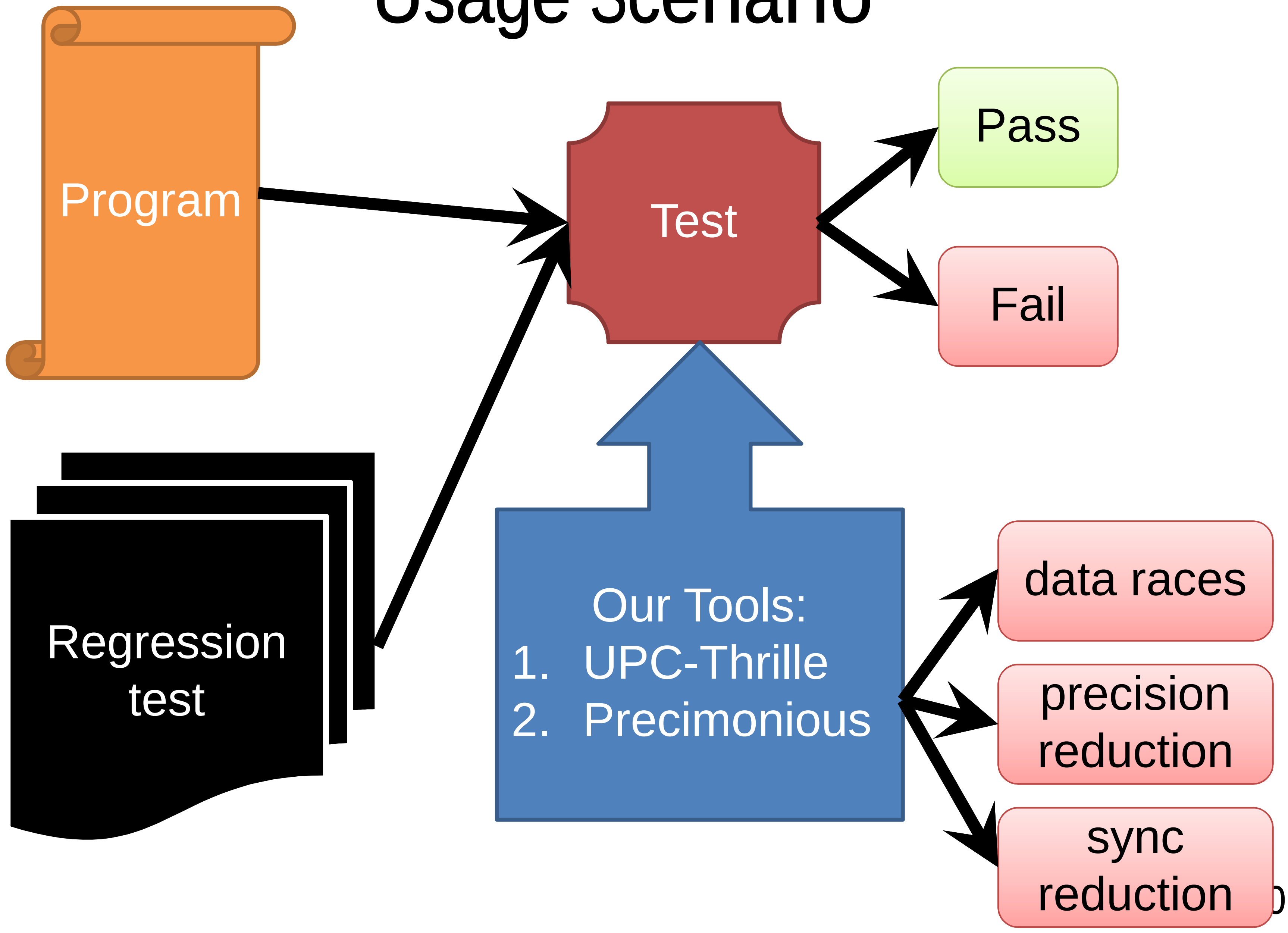
Usage Scenario



Usage Scenario



Usage Scenario



Challenges: Comparison with State-of-the-art

- Existing tools are slow: 10X-100X
- Handles either shared memory or message-passing
- Does not scale with number of nodes
- Cannot handle hybrid applications
  - Scientists run frameworks not mini-apps
  - Composition of multi-language (C, C++, Fortran), multi-paradigm (GASnet, MPI, OpenMP, Pthreads) solvers/libraries (Scalapack, MKL, BoxLib)
- Precision tuning is conservative
- Reproducible algorithms do not scale to large number of nodes

Finding Non-Deterministic Bugs

Case Study: Scalable Data Race Detection for Partitioned Global Address Space Programs

Motivation

- High performance computing is rapidly evolving
  - Exascale: $O(10^6)$ nodes, $O(10^3)$ cores per node
  - Programs with side-effects through one-sided communication
  - Unstructured parallelism, dynamic tasking, shared memory
  - Non-blocking, highly asynchronous behavior
- Many correctness challenges
  - Hard to diagnose correctness and performance bugs (data races, atomicity violations, deadlocks)
  - Non-determinism leads to non-reproducible results
- Current tools offer limited support
  - Limited functionality (e.g. communication only DAMPI)
  - Do not scale (e.g. Intel ThreadChecker)
- We need tools for the “next” generation languages

Design Requirements

- Efficiency and low overhead
  - Currently 10X-1000X
- Complete memory coverage
  - Currently either shared memory OR communication only
- Precise: report only the “bugs”
- Reproducible: identical behavior across executions
- Scalable in program size (LoCs), input size and concurrency

Data Races in PGAS

- Thrille: data race detector for UPC using Active Testing
  - Communication only
  - Precise and reproducible
  - Scalable with cores due to distributed analysis

- Active Testing
  - Phase 1: Dynamic analysis to find potential concurrency bug patterns
    - Such as data races, deadlocks, atomicity violations
  - Phase 2: "Direct" testing (or model checking) based on the bug patterns obtained from phase 1
    - Confirm bugs
- How do we achieve low overhead and scalability with input and cores for a complete analysis ?

Data Race Detection Implementation

- For each load/store or communication operation
  Examine the address –> instrumentation overhead
  Record the address –> data management overhead

- For each synchronization operation
  Exchange information about all L/S and comms
  Analyze for conflicts

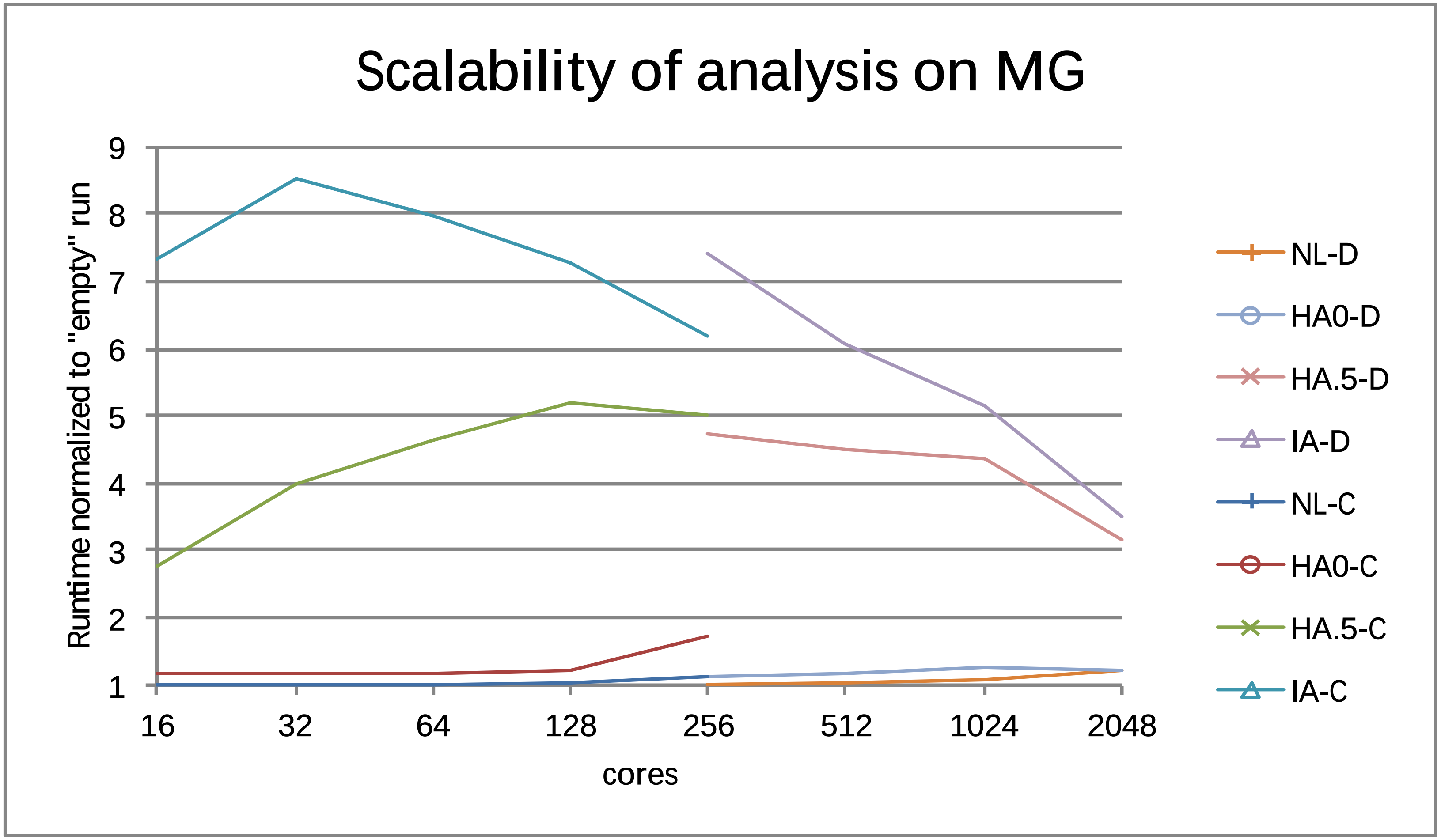
- Instrumentation overhead is reduced by
  – Hybrid Sampling (instruction + function level sampling)
  – Further pruning using program analysis

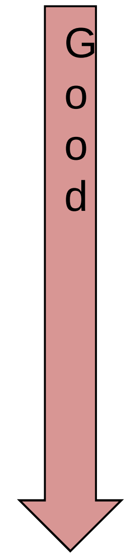
- Data management overhead is reduced with better data structures

Sampling Strategies

- Instruction sampling – sample every instruction with decaying probability
  - Introduces up to 40X slowdown for our benchmarks
- State-of-the-art function level sampling (LiteRace) does NOT work
  - (Marino et al. LiteRace: Effective Sampling for Lightweight Data-Race Detection. PLDI, 2009.)
- Novel hierarchical sampling approach provides best performance
  - Introduces up to 10x slowdown for our benchmarks
- Sampling needs to be supplemented with other pruning
  - Runtime alias based pruning (or other static analyses)

Overall Scalability





< 50% slowdown up to 2K cores

Commercial tools : 1000X slowdown on 16 cores

Bugs Found

Bench	LoC	Run time (s)	Races	Overhead (%)				
				NL	HA.5	IA	FA0	I
guppie	271	19.070	2(2)+0(0)	54.9	54.2	53.7	DNF	74.9
psearch	803	0.697	3(1)+2(2)	2.48	10.8	666	8.01	6490
BT 3.3	9698	189.48	7(0)+3(1)	0.574	1.16	77.6	DNF	-
CG 2.4	1654	39.573	0(0)+1(1)	1.09	27.6	57.6	DNF	2579
EP 2.4	678	54.453	0(0)+0(0)	-0.618	0.805	2.09	4.74	111
FT 2.4	2289	62.663	2(2)+0(0)	0.601	30.1	121	DNF	2744
IS 2.4	1426	5.130	0(0)+0(0)	0.376	119	159	DNF	1201
LU 3.3	6348	155.997	0(0)+24(2)	-0.425	-	75.7	DNF	-
MG 2.4	2229	18.687	2(2)+4(0)	0.336	176	632	DNF	2020
SP 3.3	5740	247.937	10(0)+3(1)	0.160	0.861	29.1	DNF	-

Races: A(B) + C(D), where A represents the number of races detected by the original UPC-Thrille tool (NL) with B of them confirmed, and C represents the additional number of races detected with our extensions (HA.5) with D of them confirmed through phase 2

KEY FOR VARIANTS
NL: no instrumentation on local accesses (SC'11) / H: hierarchical sampling / I: instruction-level sampling only / F: function-level sampling only
A: indicates the use of the persistent alias heuristic
 # (0 or .5): Back-off factor for function-level sampling (0 means only first invocation of functions sampled)

< 50% slowdown up to 2K cores

Highlights

- Conference Publication
  - Our paper entitled *“Scaling Data Race Detection for Partitioned Global Address Space Programs”* was presented at the International Supercomputing Conference (ICS’13) in Oregon, June 2013.
- Software Release
  - Publicly released UPC-Thrille under the BSD license: http://upc.lbl.gov/thrille.shtml

Finding Redundancy in Precision

Case Study: Tuning Precision of

Floating-point Programs

Floating-Point Precision Tuning

- Reasoning about FP programs is difficult
  - Large variety of numerical programs
  - Most programmers are not experts in FP
  - Even experts on scientific computing may not be expert in FP

- Common practice
  - Use highest available precision
  - Disadvantages: more expensive in terms of running time, memory and energy consumption

Precimonious

- "Parsimonious with precision"
- Common Practice: Use widest precision available (usually IEEE double or long double)
  - Pros: Easy, reliable
  - Cons: Maximizes time, memory, energy used
- Precimonious automatically decides which variables/operations can be in lower precision (single) and still get an acceptable answer

Example (D.H. Bailey):

$$\sum_{k=0}^{n-1} \sqrt{h^2 + (g(x_{k+1}) - g(x_k))^2}$$

```
long double g(long double x) {
  int k, n = 5;
  long double t1 = x;
  long double d1 = 1.0L;

  for(k = 1; k <= n; k++) {
     ...
  }
  return t1;
}

int main() {
  int i, n = 1000000;
  long double h, t1, t2, dppi;
  long double s1;
  ...
  for(i = 1; i <= n; i++) {
    t2 = g(i * h);
    s1 = s1 + sqrt(h*h + (t2 - t1)*(t2 - t1));
    t1 = t2;
  }
  // final answer stored in variable s1
  return 0;
}
```

Original Program

?

Tuned Program

Example (D.H. Bailey): $\sum_{k=0}^{n-1} \sqrt{h^2 + (g(x_{k+1}) - g(x_k))^2}$

```
long double g(long double x) {
  int k, n = 5;
  long double t1 = x;
  long double d1 = 1.0L;

  for(k = 1; k <= n; k++) {
     ...
  }
  return t1;
}

int main() {
  int i, n = 1000000;
  long double h, t1, t2, dppi;
  long double s1;
  ...
  for(i = 1; i <= n; i++) {
    t2 = g(i * h);
    s1 = s1 + sqrt(h*h + (t2 - t1)*(t2 - t1));
    t1 = t2;
  }
  // final answer stored in variable s1
  return 0;
}
```

Original Program

```
double g(double x) {
  int k, n = 5;
  double t1 = x;
  float d1 = 1.0f;

  for(k = 1; k <= n; k++) {
     ...
  }
  return t
}

int main() {
  int i, n = 1000000;
  double h, t1, t2, dppi;
  long double s1;
  ...
  for(i = 1; i <= n; i++) {
    t2 = g(i * h);
    s1 = s1 + sqrt(h*h + (t2 - t1)*(t2 - t1));
    t1 = t2;
  }
  // final answer stored in variable s1
  return 0;
}
```

Same answer as all long double
10% faster
3 more correct digits than double

Tuned Program

Challenges for Precision Tuning

- Searching efficiently over variable types and function implementations
  - Naïve approach → exponential time
    - 19,683 configurations for arc length program (3^9)
    - 11 hours 5 minutes
  - Global minimum vs. a local minimum
- Evaluating type configurations
  - Less precision does not always result in performance improvement
  - Run time, memory usage, energy consumption, etc.
- Determining accuracy constraints
  - How accurate must the final result be?
  - What error threshold to use?

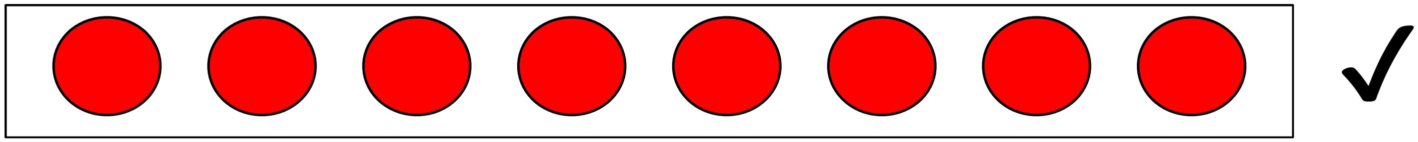
Automated:
116 configs.
4 min 47 sec

Specified by the user

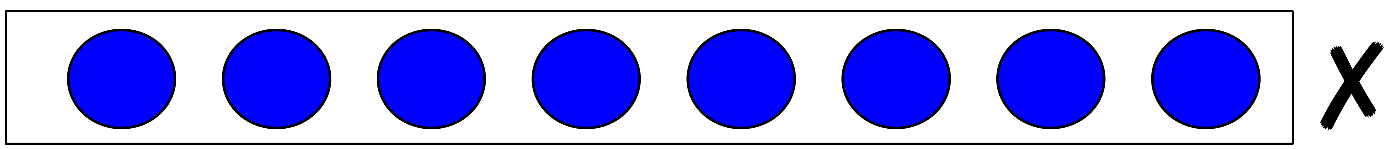
LCCSEARCH Algorithm

based on Delta Debugging [Zeller et al]

double precision

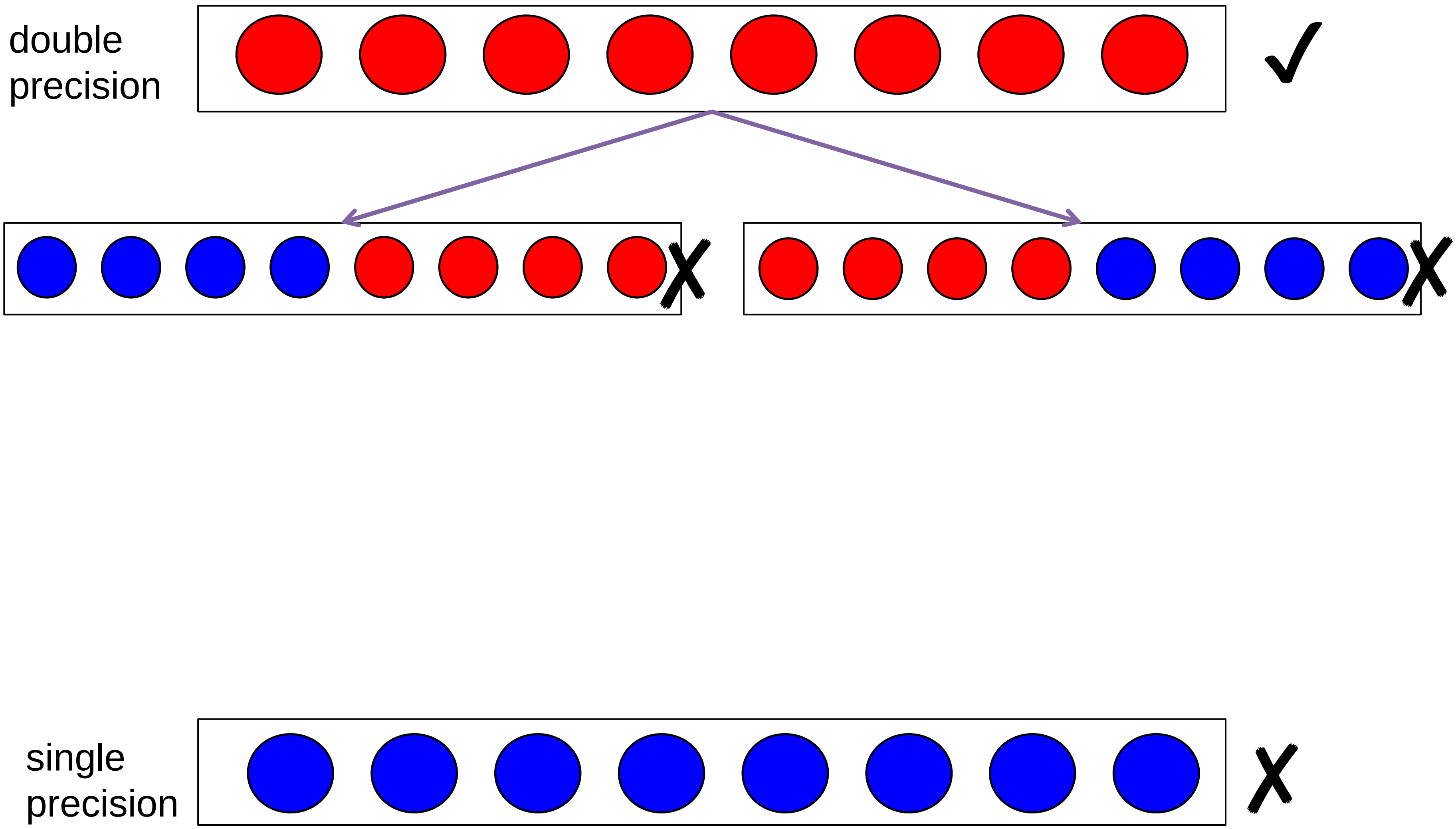


single precision



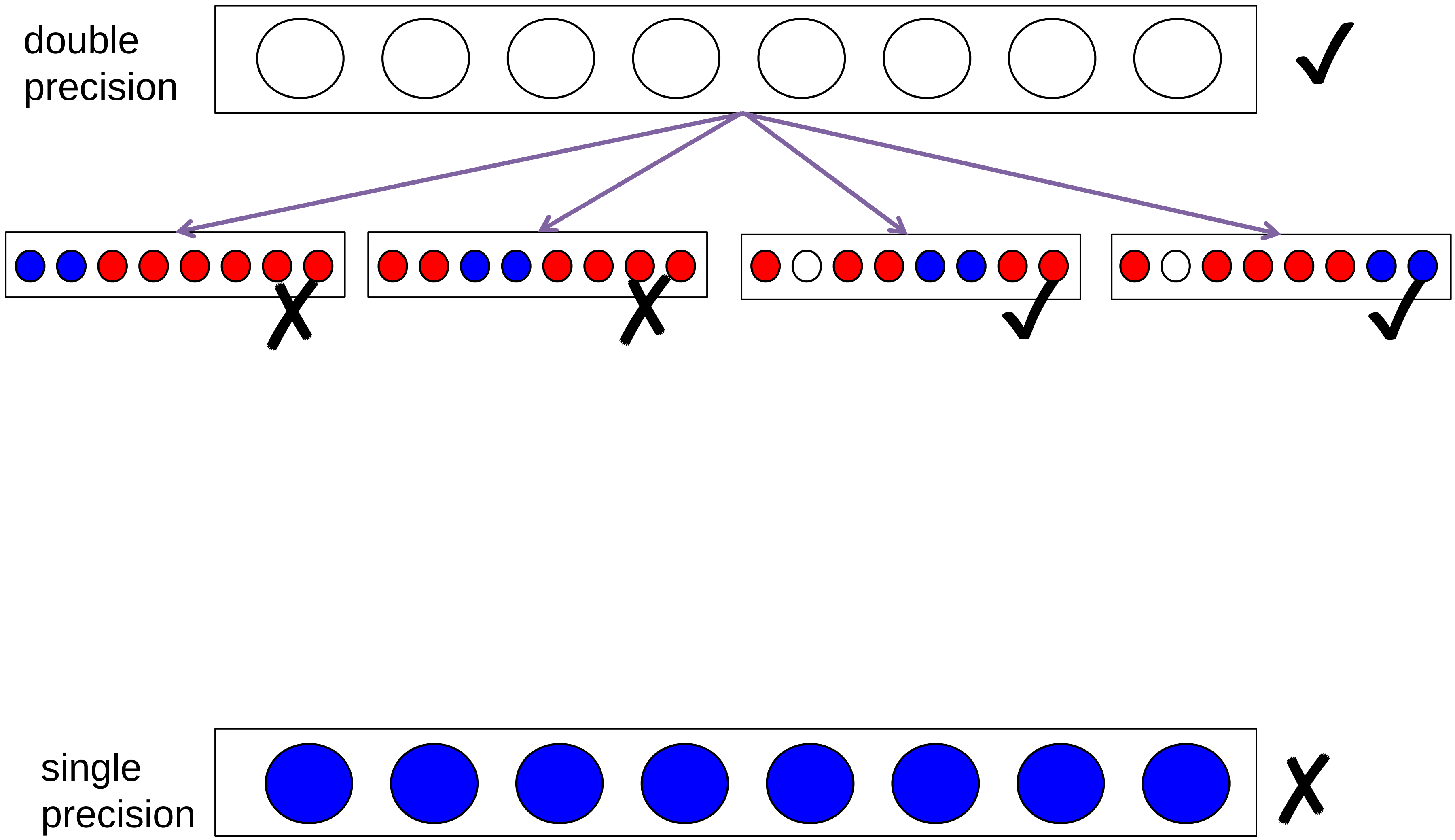
LCCSEARCH Algorithm

based on Delta Debugging [Zeller et al]



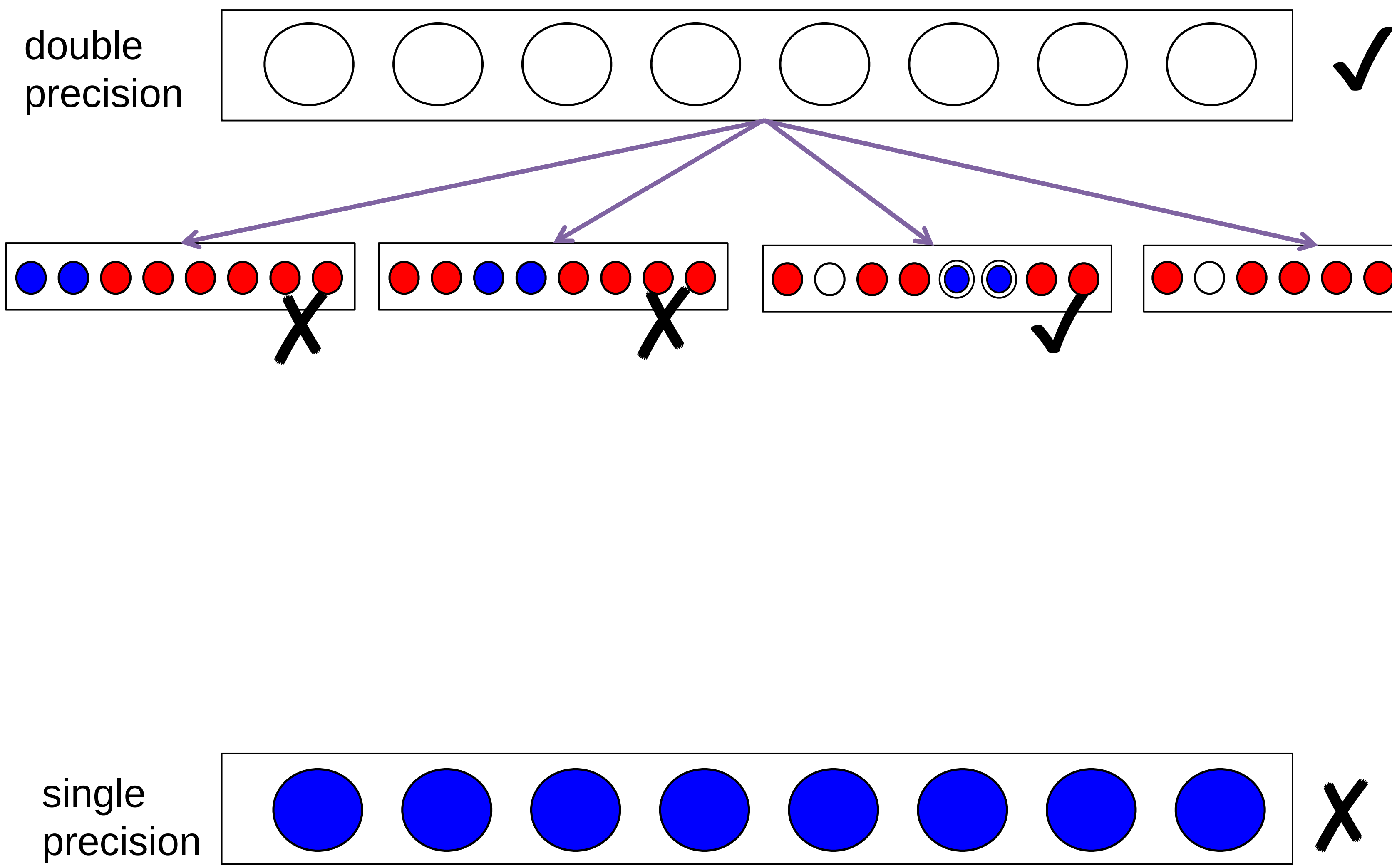
LCCSEARCH Algorithm

based on Delta Debugging [Zeller et al]



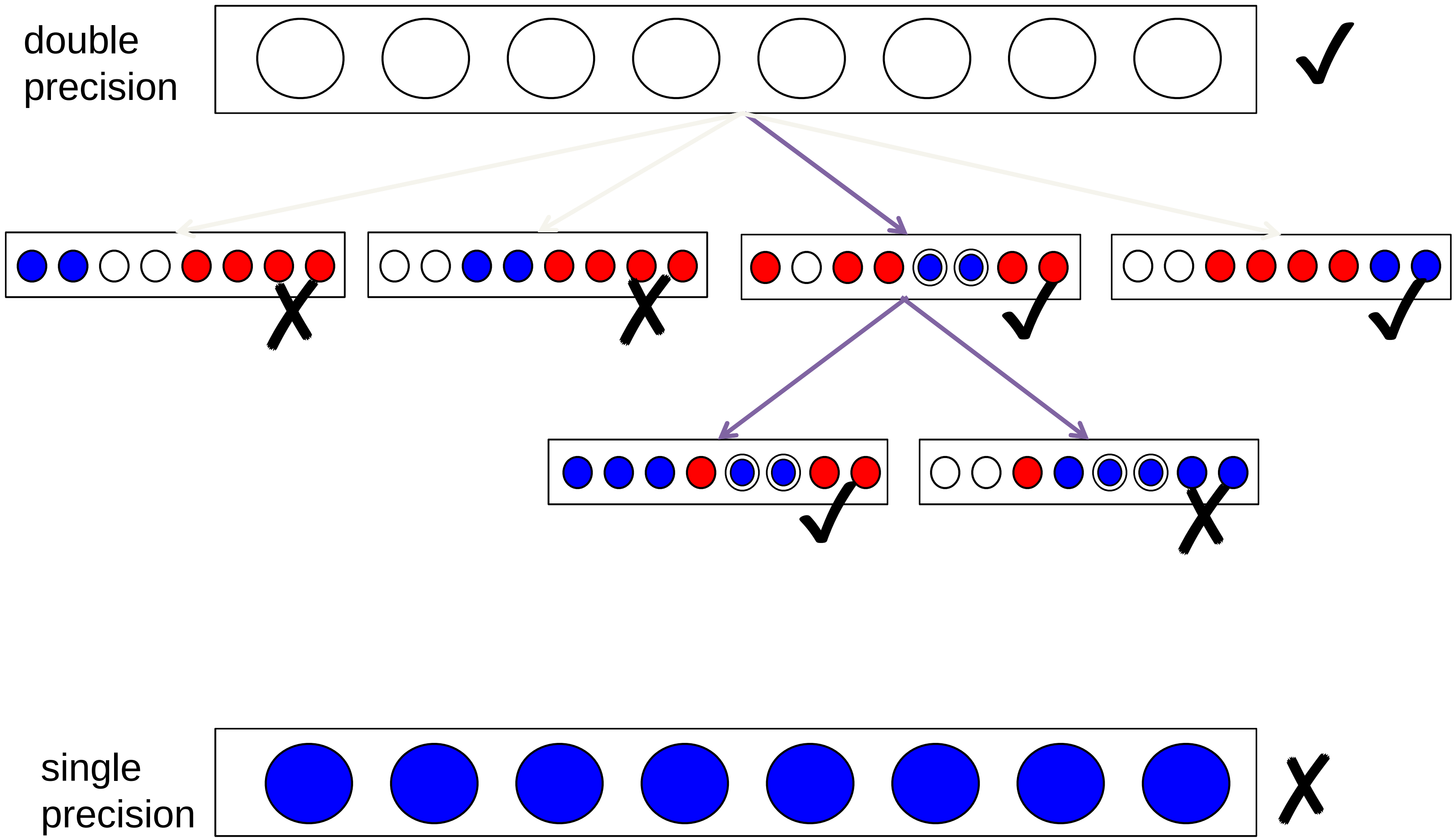
LCCSEARCH Algorithm

based on Delta Debugging [Zeller et al]



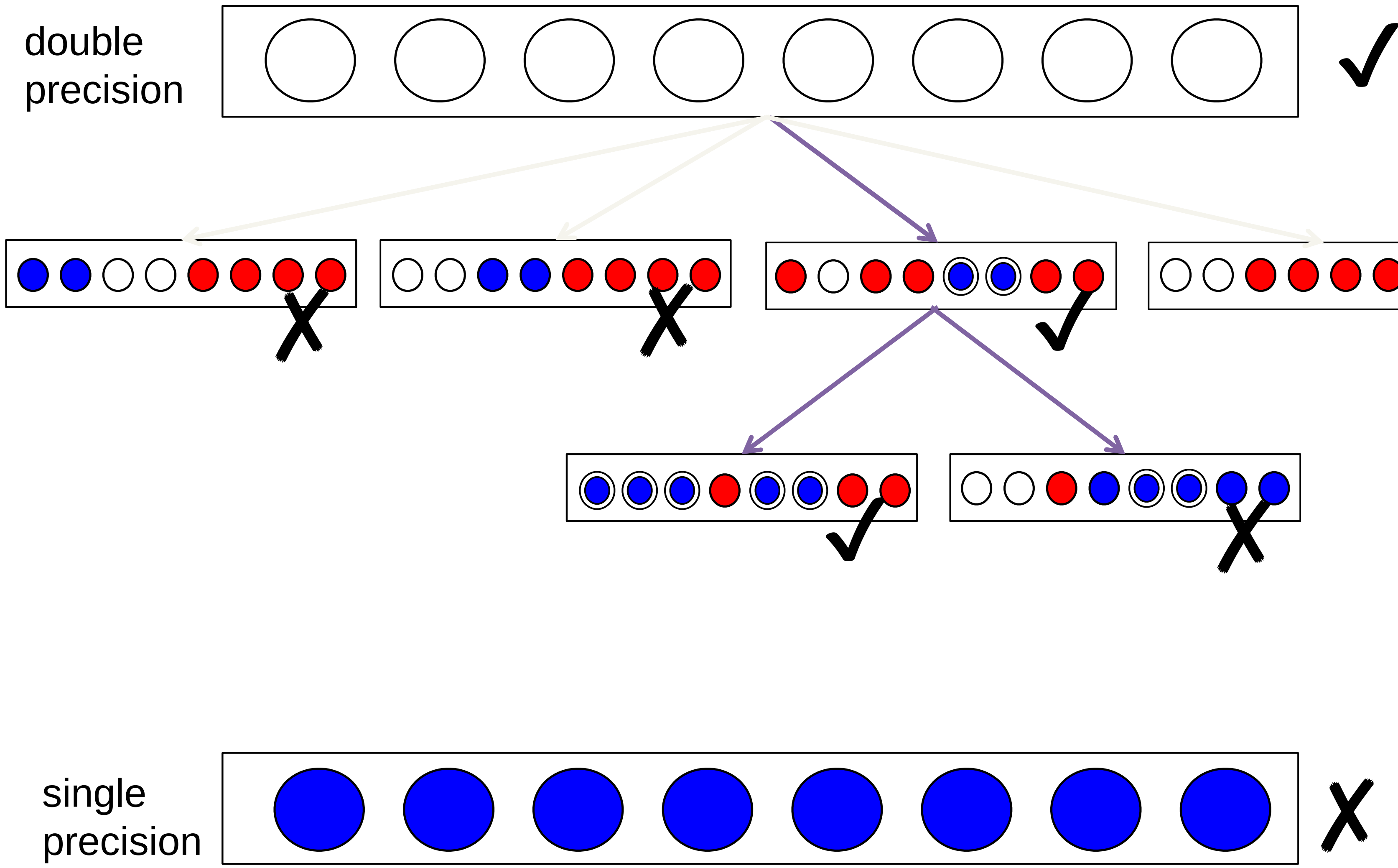
LCCSEARCH Algorithm

based on Delta Debugging [Zeller et al]



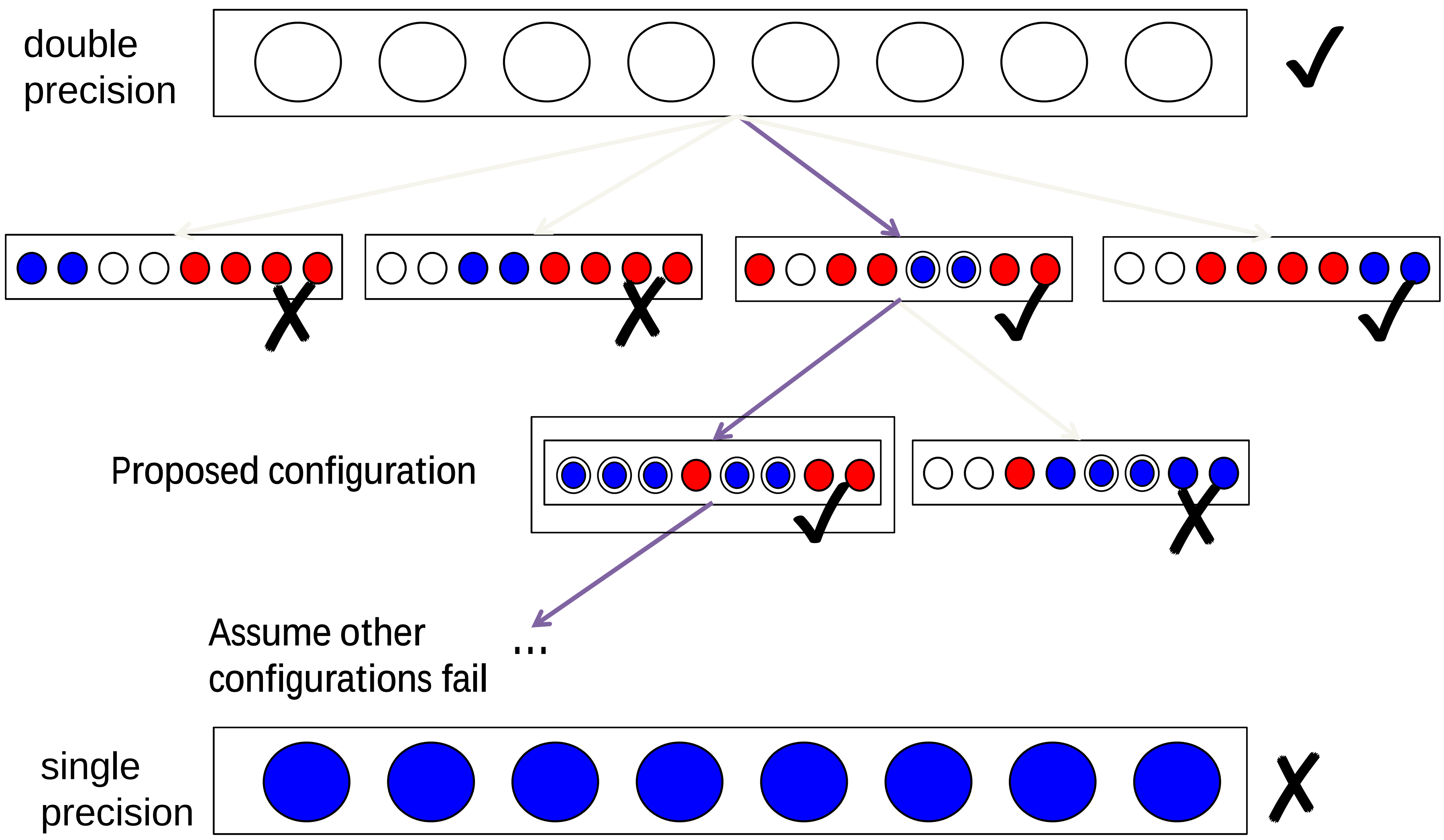
LCCSEARCH Algorithm

based on Delta Debugging [Zeller et al]



LCCSEARCH Algorithm

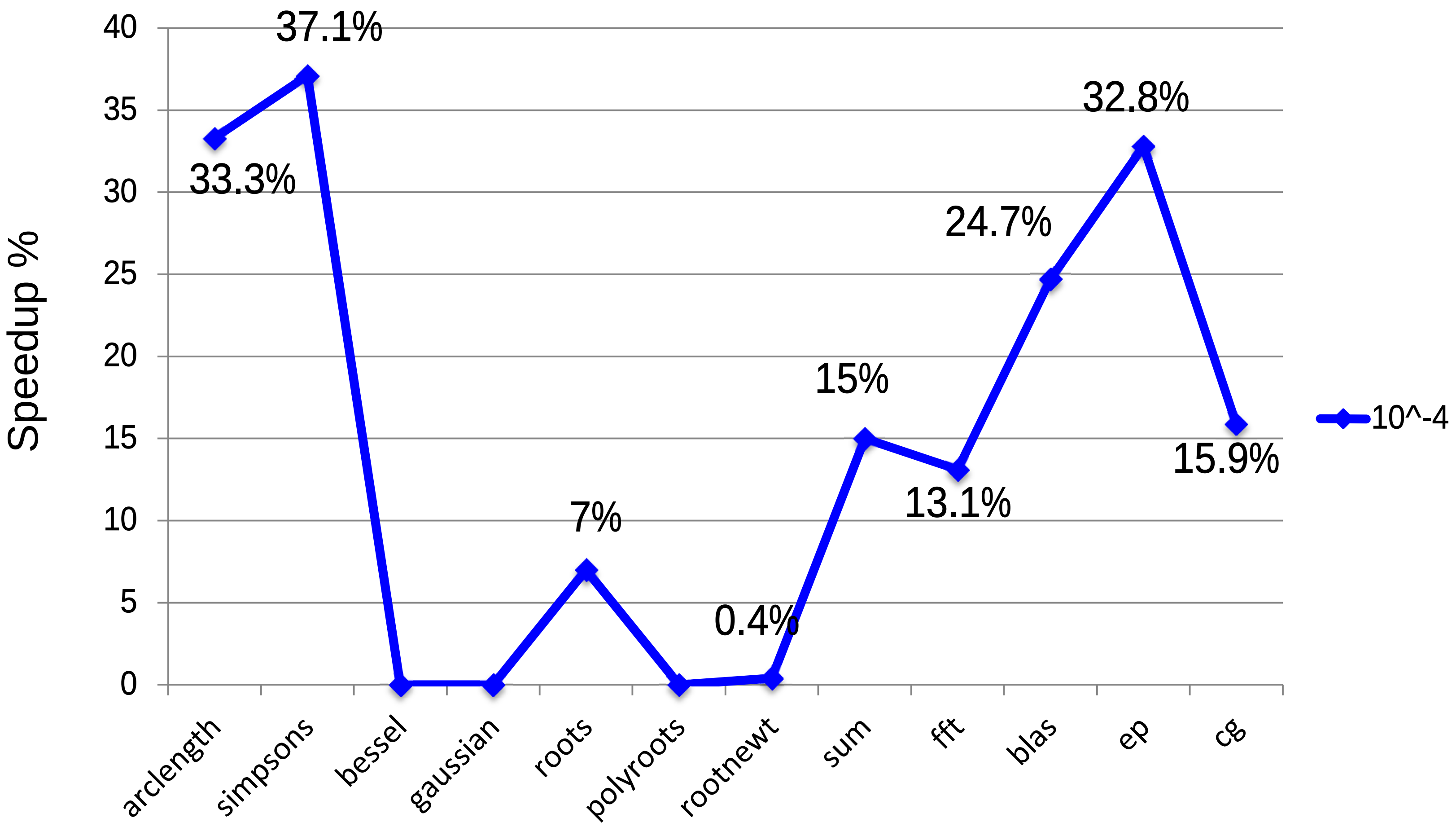
based on Delta Debugging [Zeller et al]



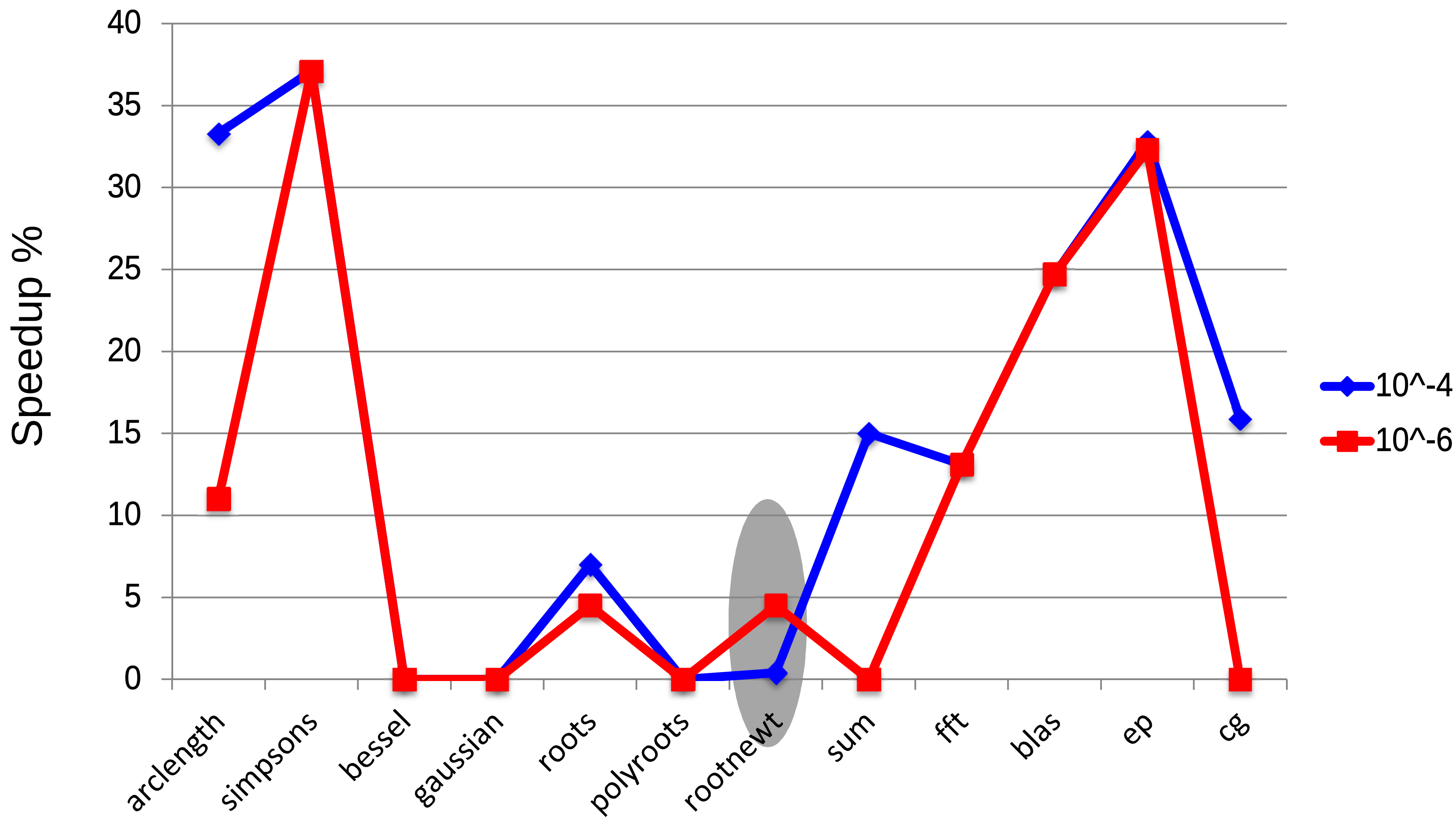
Experimental Setup

- Benchmarks
  - 8 GSL programs
  - 2 NAS Parallel Benchmarks: *ep* and *cg*
  - 2 other numerical programs
- Test inputs
  - Inputs Class A for *ep* and *cg* programs
  - 1000 random floating-point inputs for the rest
- Error thresholds
  - Multiple error thresholds: 10^{-10}, 10^{-8}, 10^{-6}, and 10^{-4}
  - User can evaluate trade-off between accuracy and speedup

Speedup for Various Error Thresholds

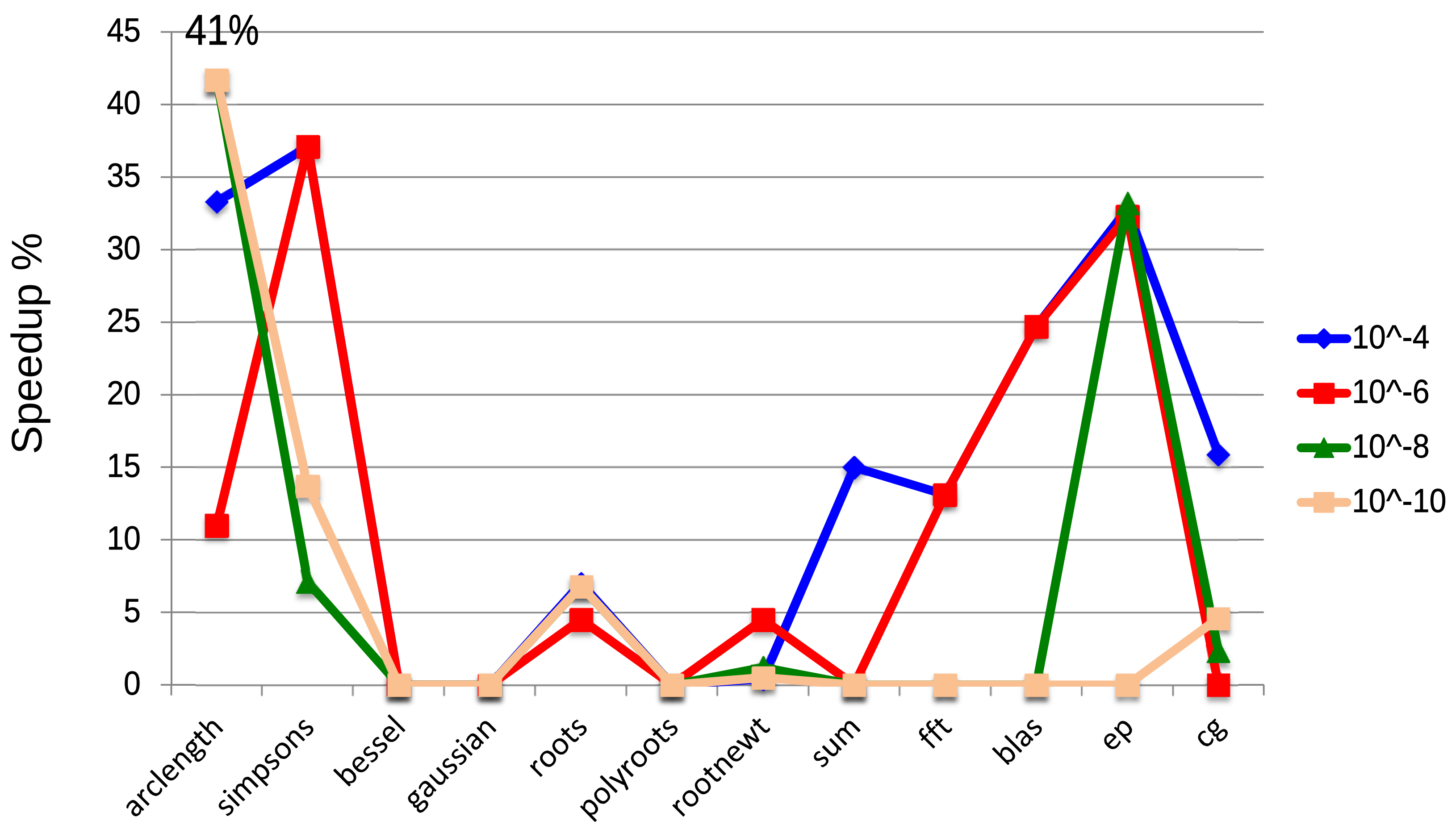


Speedup for Various Error Thresholds



Error threshold 10^{-6} → slightly larger speedup than error threshold10^{-4}
(4.5% vs. 0.4% for rootnewt program)

Speedup for Various Error Thresholds



Scalability Limitation

Largest benchmark: 52 FP variables
Configurations explored: 1,435 configurations
Analysis running time: 1hr 26min

Too many runs for larger programs!

Shadow Execution

- Motivation: Precimonious uses input after conversion to LLVM in order to modify, track changes in code: What else can we do with this infrastructure?
- Shadow Execution: Track execution dynamically
  - Compare to results computed in different precisions
  - Track sources of inaccuracy: “Blame analysis”
    - Reduce search space for Precimonious
    - Up to 5x fewer configurations to search

Highlights

- Conference Publication
  - PRECIMONIOUS[11] was accepted for conference publication at the prestigious *International Conference for High Performance Computing, Networking, Storage and Analysis* (SC'13).
- Cindy Rubio Gonzalez will join UC Davis as an Assistant Professor in Fall 2014
- We have released PRECIMONIOUS under the BSD license. The tool is available at https://github.com/corvette-berkeley/precimonious .

Reproducibility of Floating-point Programs

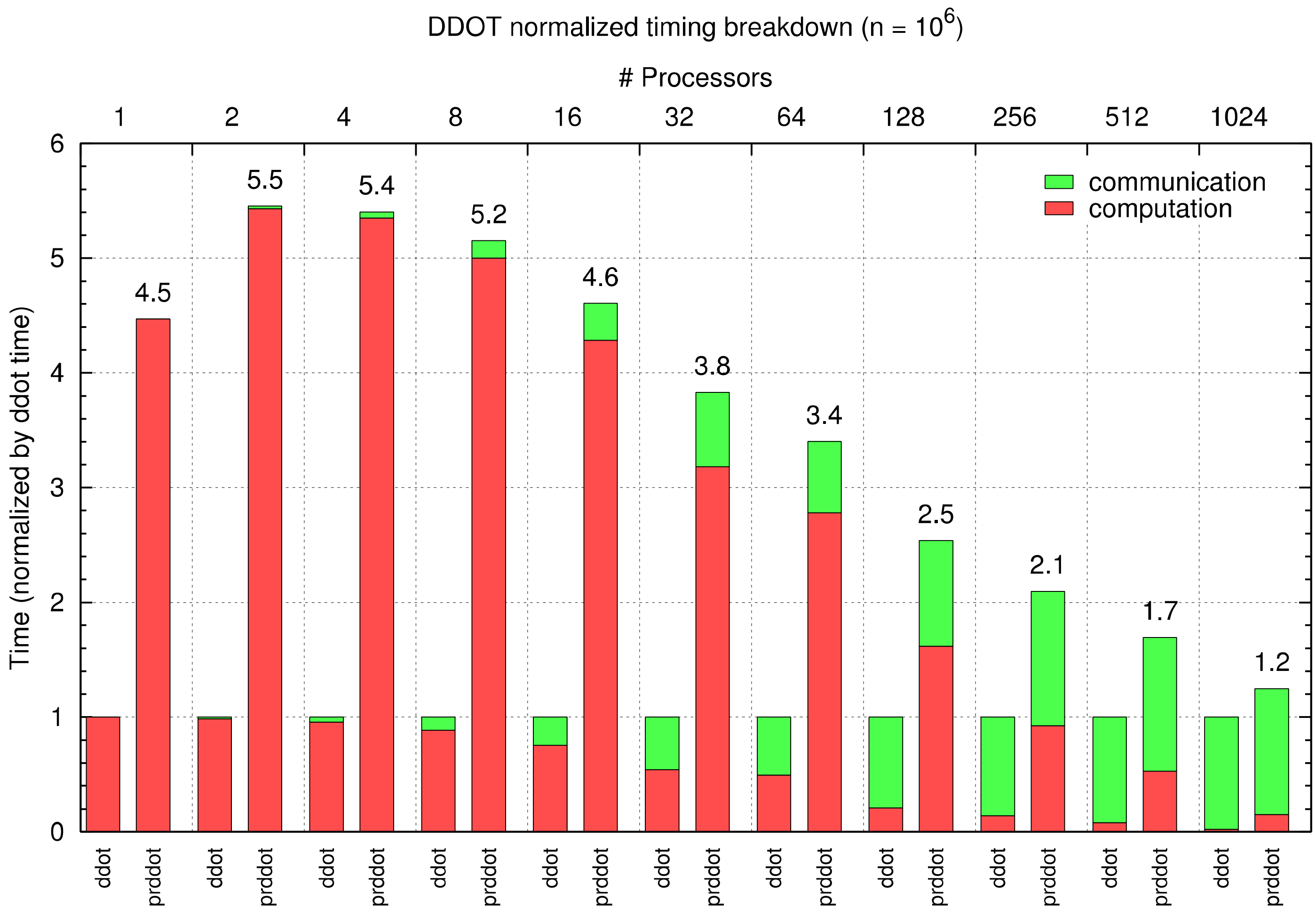
Motivation for Reproducibility

- Reproducibility = bitwise identical results when running code more than once
- No longer guaranteed because of parallelism, nondeterminism, and nonassociativity of floating point addition/multiplication:
  – $fl(1 + (1e20 - 1e20)) = 1 \neq 0 = fl((1 + 1e20) - 1e20)$
- Demanded by many users, for debugging, correctness, contractual obligations
  – BOFs at recent Supercomputing conferences
  – Intel, Mathworks, other companies responding to demand with new (deterministic) products
- At large scale, nondeterminism unavoidable – What to do?

Reproducible BLAS: ReproBLAS

- Based on *Indexed Floating Point*: roundoff is deterministic, independent of summation order
  - Can choose same or higher accuracy than usual FP
  - Only one (nondeterministic) reduction required
- ReproBLAS for BLAS1 released (mBSD license)
  - bebop.cs.berkeley.edu/reproblas
  - Sequential and MPI versions
  - {s|d|c|z}{asum,sum,nrm2,dot}
  - Multithreaded, higher level BLAS under construction
- Integrated into CLAMR (DOE Mini-App)

Performance Results
DDOT for $n=10^6$ on Hopper



Highlights

- Conference Publications
  - Our papers entitled *"Fast Reproducible Floating-Point Summation"* [7] and *"Numerical Accuracy and Reproducibility at ExaScale"* [8] were presented at the 21st IEEE Symposium on Computer Arithmetic in Austin, Texas. Our paper *"Parallel Reproducible Summation"* [9] has been recently accepted for publication in the IEEE Transactions on Computers, Special Section on Computer Arithmetic.
- Software Release:
  - ReproBLAS released at http://bebop.cs.berkeley.edu/reproblas

Vision: Reduce Programming to Debugging ratio

Domain
Hybrid Parallelism + Floating Point + Modularity

Advantages:
- Performance
- Scalability
- Abstraction
- Modularity

Disadvantages:
- non-deterministic bugs
- non-reproducible results
- redundant syncs
- redundant precision

Goals:
Correctness/Performance tools to help programmers with development
1. efficient
2. scalable
3. reproducible
4. precise
5. coverage

Our Approach:
1. Dynamic analysis
2. Symbolic execution
3. New Algorithms

Tools:
1. UP-Thrille: data races
2. Precimonious: FP precision tuning
3. ReproBLAS: reproducible num. algorithms
4. LLVM Shadow Execution: dynamic analysis
5. Sync reduction: remove redundant syncs