Automatic Detection of MPI Application Structure with Event Flow Graphs

Karl Fürlinger[1]

joint work with
Xavier Aguilar[2] and Erwin Laure[2]

[1] Ludwig-Maximilian-University (LMU)
Munich, Germany

[2] KTH Royal Institute of Technology
Stockholm, Sweden

Tracing and Profiling

- **Trace**

  A B A D C D B C B C D D …

  – Full temporal order of events is preserved
  – A lot of data to store, process, analyze

- **Profile (summary)**

  100x A
  42x B
  33x C
  17x D

  – Temporal order is not preserved
  – Far less data

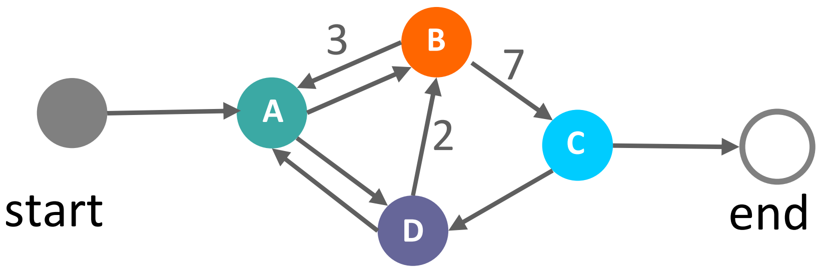
- Implementation in IPM[1]
  – Keep data in a **hash table**
  – Keys: **event** (-signatures)
  – Values: **statistics** (#calls, duration, …)

key	#calls	duration
B	42	23.1
A	100	12.0

[1]Integrated Performance Monitor
http://ipm-hpc.sourceforge.net/

Something in Between Profiling and Tracing…

- Event Flow Graphs (EFGs)
  - Keep a **history** of the **previous event** that happened
  - Keep track of pairs of events (**prev.**, **curr.**) instead of single events



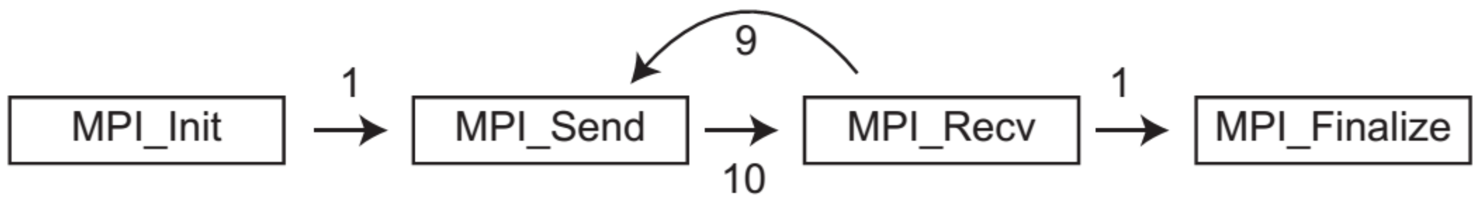
- Similar to a control flow graph, but
  - records tansitions that have actually happened in an execution
  - records how many times these transitions have happend

- Implementation in IPM:
  - Keep an **additional hash table**
  - Keys: pairs of events (prev., curr.)
  - Values: statistics (#transitions, duration, …)

key	#trans.	duration
● → A	1	0.02
A → D	7	1.05

Example Event Flow Graph (1)

```
int main() {
  MPI_Init(...);
  for(i=1;i<=10;i++) {
    MPI_Send(...);
    MPI_Recv(...);
  }
  MPI_Finalize();
}
```

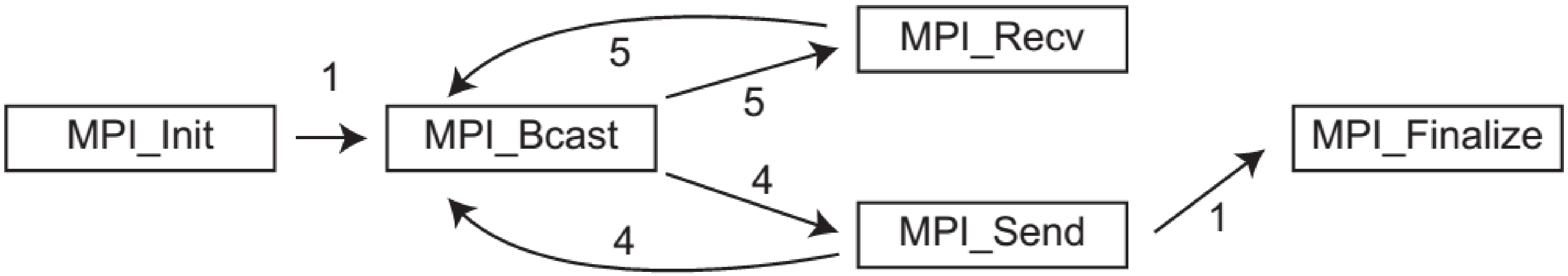


- In this case, the EFG is a perfect representation of the trace.

Example Event Flow Graph (2)

```
int main() {
  MPI_Init(...);
  for(i=1;i<=10;i++) {
    MPI_Bcast(...);
    if(i%2) /* odd */
      MPI_Recv(...);
    else /* even */
      MPI_Send(...);
  }
  MPI_Finalize();
}
```

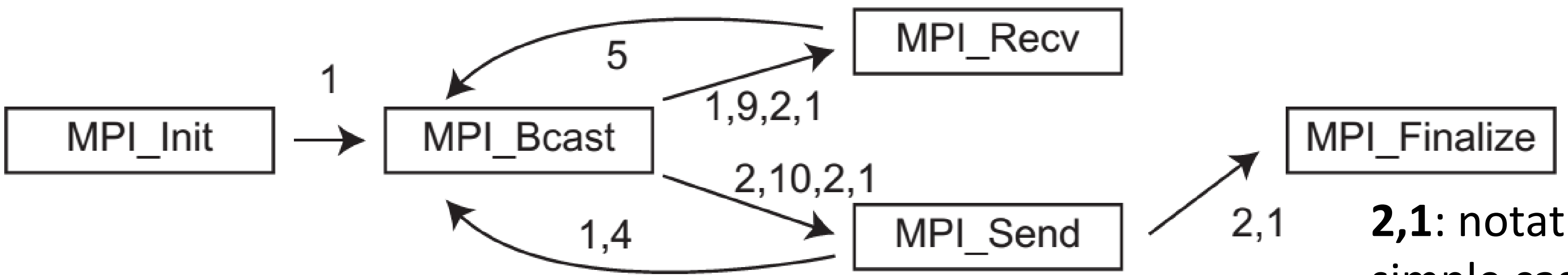
- In this case, the trace cannot be uniquely reconstructed from the EFG.



Temporal Event Flow Graphs

- **Temporal EFG (t-EFG):**
  - A modified version of an EFG that **guarantees** trace recovery
- Ideas
  - At each node, keep track of which outgoing edge to take next
  - Represent this information in a compact way
- t-EFG for the previous example:
  - Edge label describes a **partition** of the iteration space

1,9,2,1: first, last, stride, blocksize



2,1: notation for simple case

Using t-EFGs for Trace Compression

- Runtime data collection is still efficient
  - Around 2% overhead in terms of execution time
  - See: [EuroPar '14]: Xavier Aguilar, et al. **MPI Trace Compression using Event Flow Graphs**

- Compression results for some benchmarks [EuroPar '14] (sequence of events only)

Benchmark	# Ranks	Comp. Factor
AMG	96	1.76x
GTC	64	46.60x
MILC	96	39.03x
SNAP	96	119.23x
MiniDFT	40	4.33x
MiniFE	144	19.93x
MiniGhost	96	4.85x

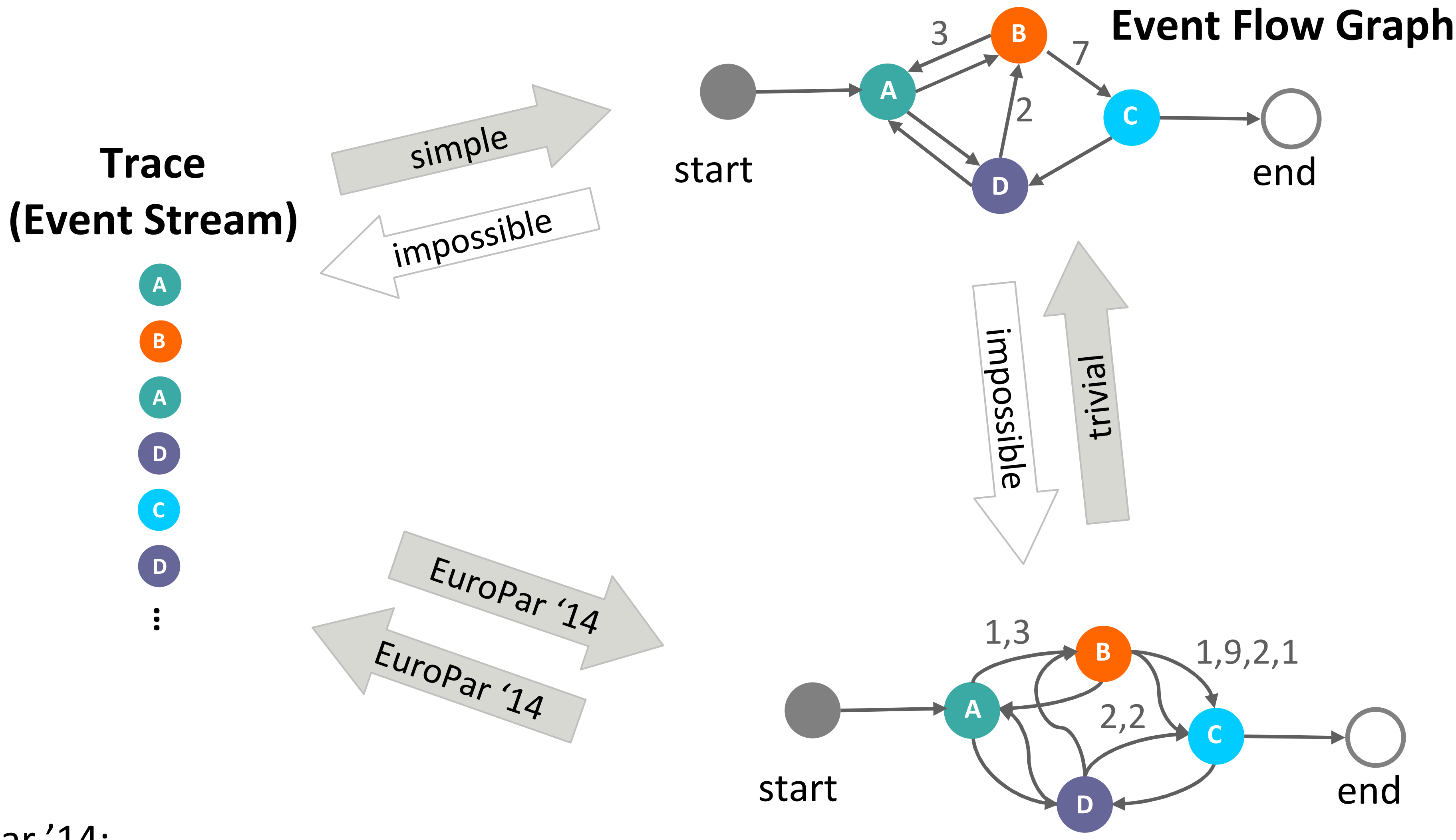
Up to 120x Compression!

EFG Graph Statistics

- Compression ratio depends on the structure of the graphs
  - Simple graphs with few nodes and edges correspond to high compression ratios

Benchmark	Avg. Compr. Ratio	Avg. Num of Nodes	Avg. Num of Edges	Avg. Node Cardinality
AMG	1.76	9,384.94	10,586.47	4.59
MiniDFT	4.33	690.30	1,980.38	27.29
SNAP	119.23	28	1,120.26	14,149.22
GTC	46.60	114.5	121.20	109.10

Overview (1)

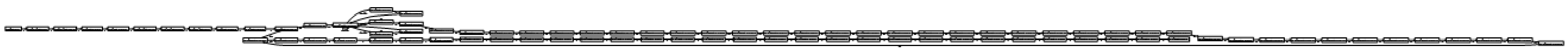


EuroPar '14:
Xavier Aguilar, Karl Fürlinger, and Erwin Laure.
MPI Trace Compression using Event Flow Graphs

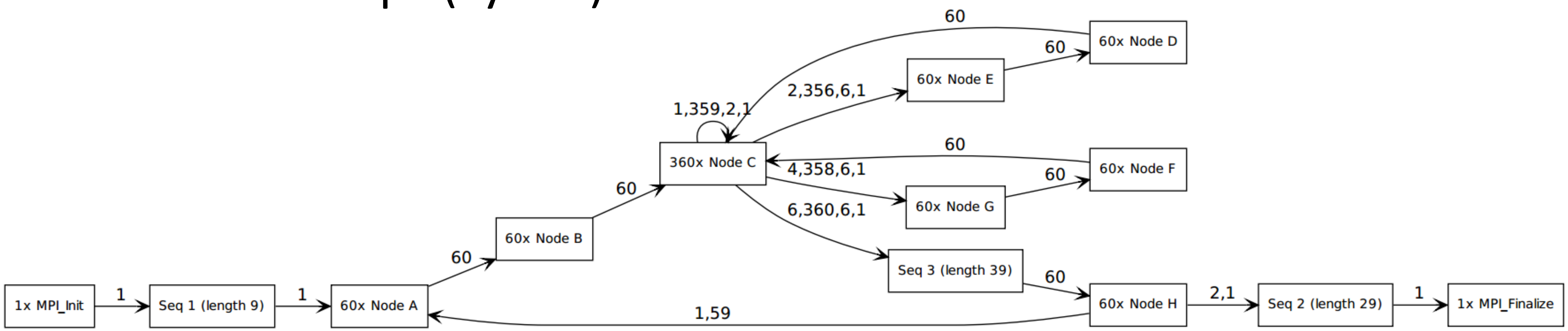
Temporal Event Flow Graph

Analyzing Event Flow Graphs

- MiniGhost example application
  - 3160 events in the trace
  - 87 nodes, 90 edges in the EFG

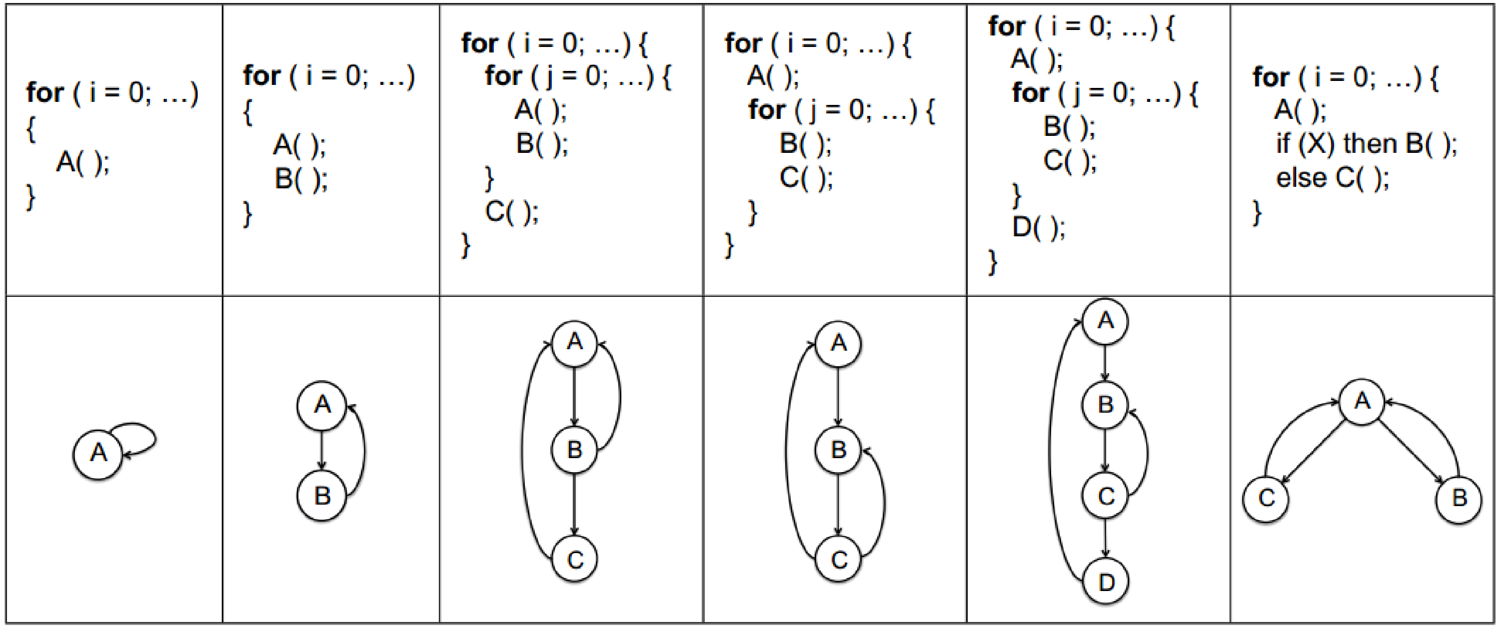


- Compressing sequences (chains)
  - 13 nodes, 16 edges
  - Nested loops (cycles) visible



Detecting Application Structure Automatically

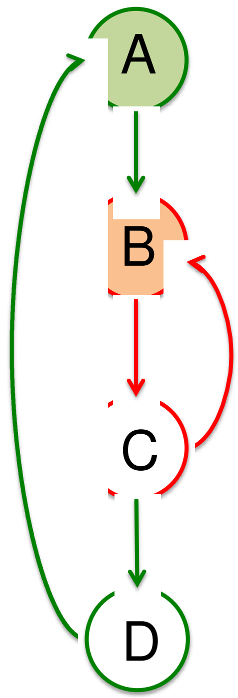
- Application Structure
  - Structure:= loops and their nesting
  - Folklore: “big outer loop hypothesis”: most scientific applications are dominated by a **big outer time-stepping loop**
- Detecting Structure
  - If a loop contains MPI calls, the loop will show up as a **cycle** in the Event Flow Graph

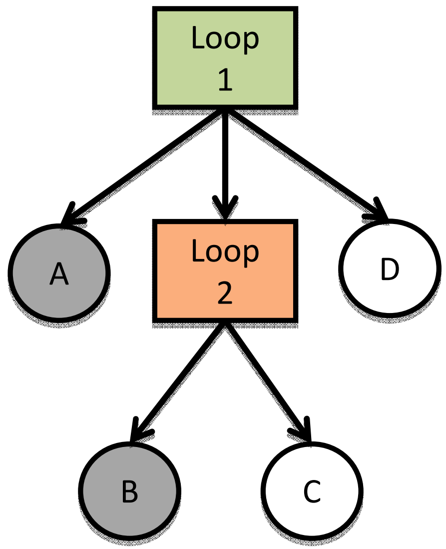


Finding Cycles in the Graph

- Detecting cycles in flow graphs is a common requirement for (de-)compilers
  - Many algorithms exist
  - We used an efficient DFS-based algorithm by T. Wei et al., "*A New Algorithm for Identifying Loops in Decompilation*", 2007

```
for ( i = 0; …) {
  A( );
  for ( j = 0; …) {
     B( );
     C( );
  }
  D( );
}
```



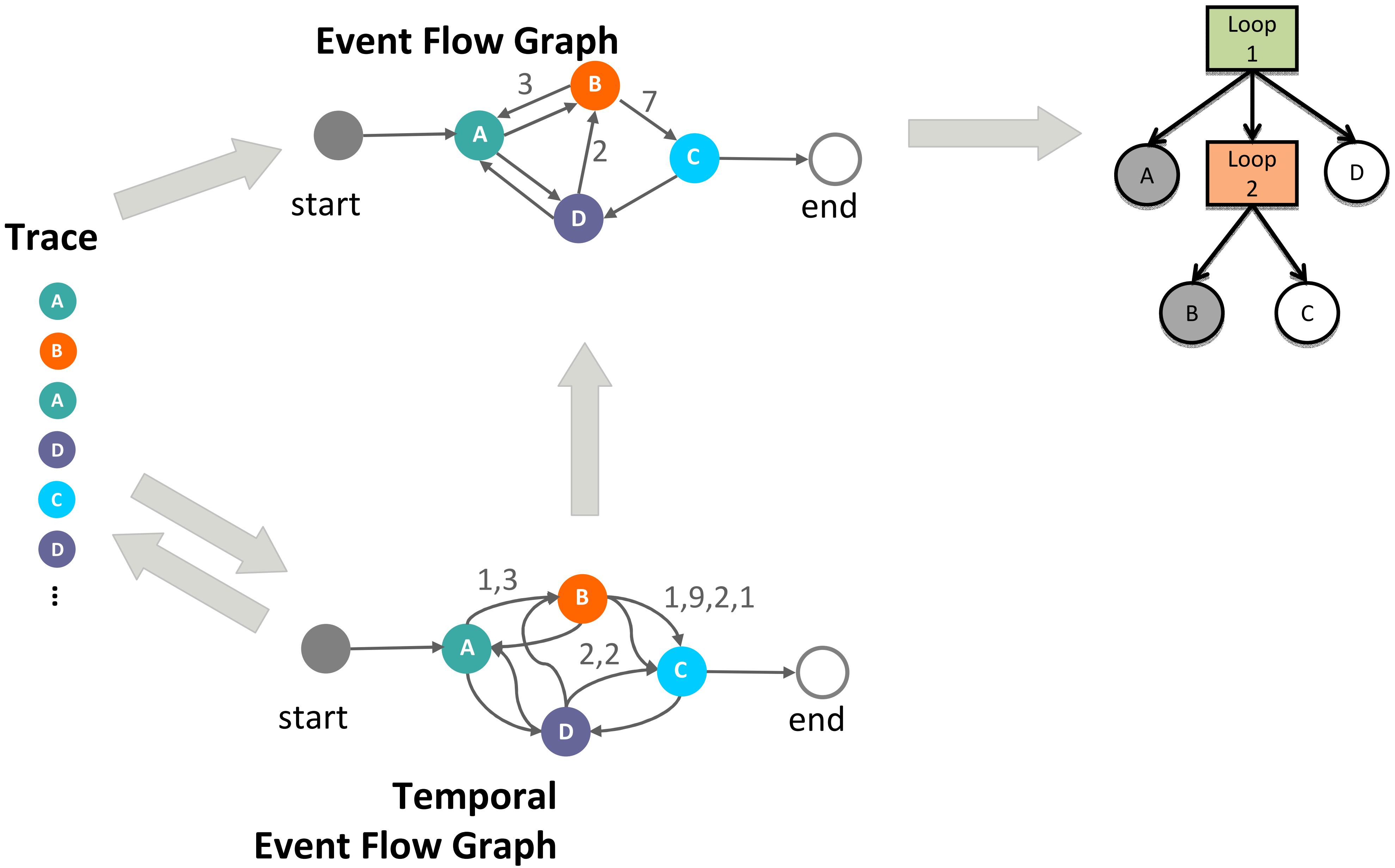


Loop Detection Results

			Outermost Loop(s)		
Benchmark	**# Ranks**	**Total Runtime (sec)**	**Count**	**Time in all**	**Time in dominant**
MiniGhost	96	282.17	1	98.8%	98.8%
MiniFE	144	133.50	13	78.1%	77.7%
BT	144	370.59	7	99.4%	99.0%
LZ	128	347.53	3	99.2%	98.9%

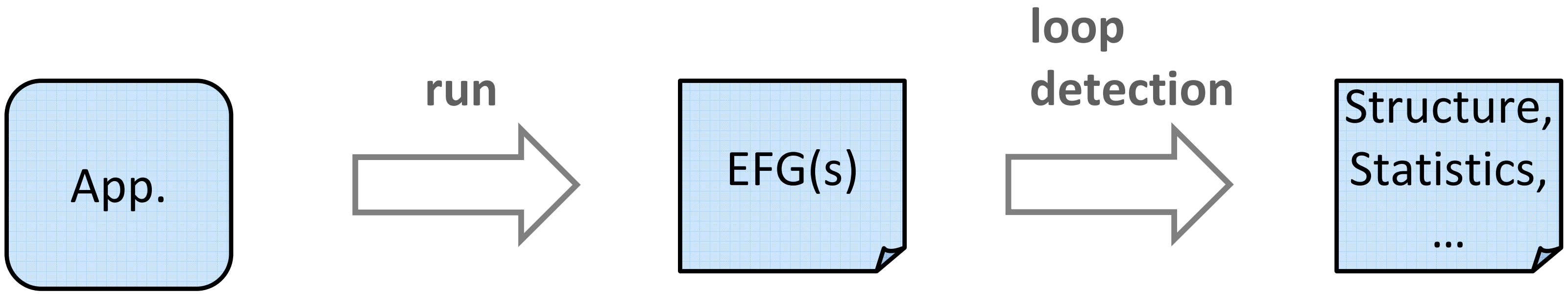
- "Big outer loop hypothesis" largely holds for these (and other) example benchmarks

Overview (2)

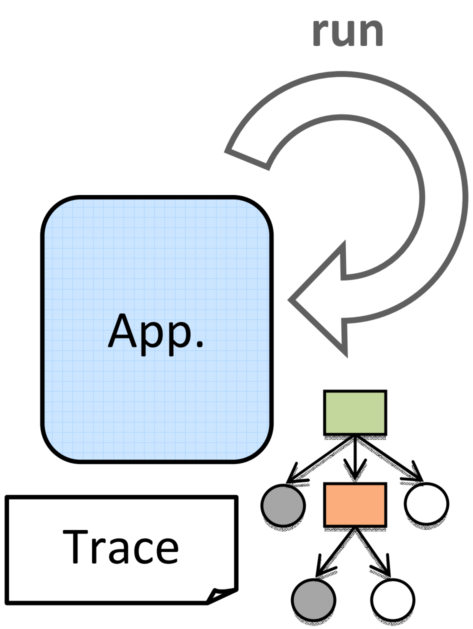

Event Flow Graph
3
7
2
A
B
C
D
start
end
Loop 1
Loop 2
A
D
B
C
Trace
A
B
A
D
C
D
⋮
1,3
1,9,2,1
2,2
A
B
C
D
start
end
Temporal Event Flow Graph

Online Structure Detection

- So far: post-mortem operation



- Now: Online operation

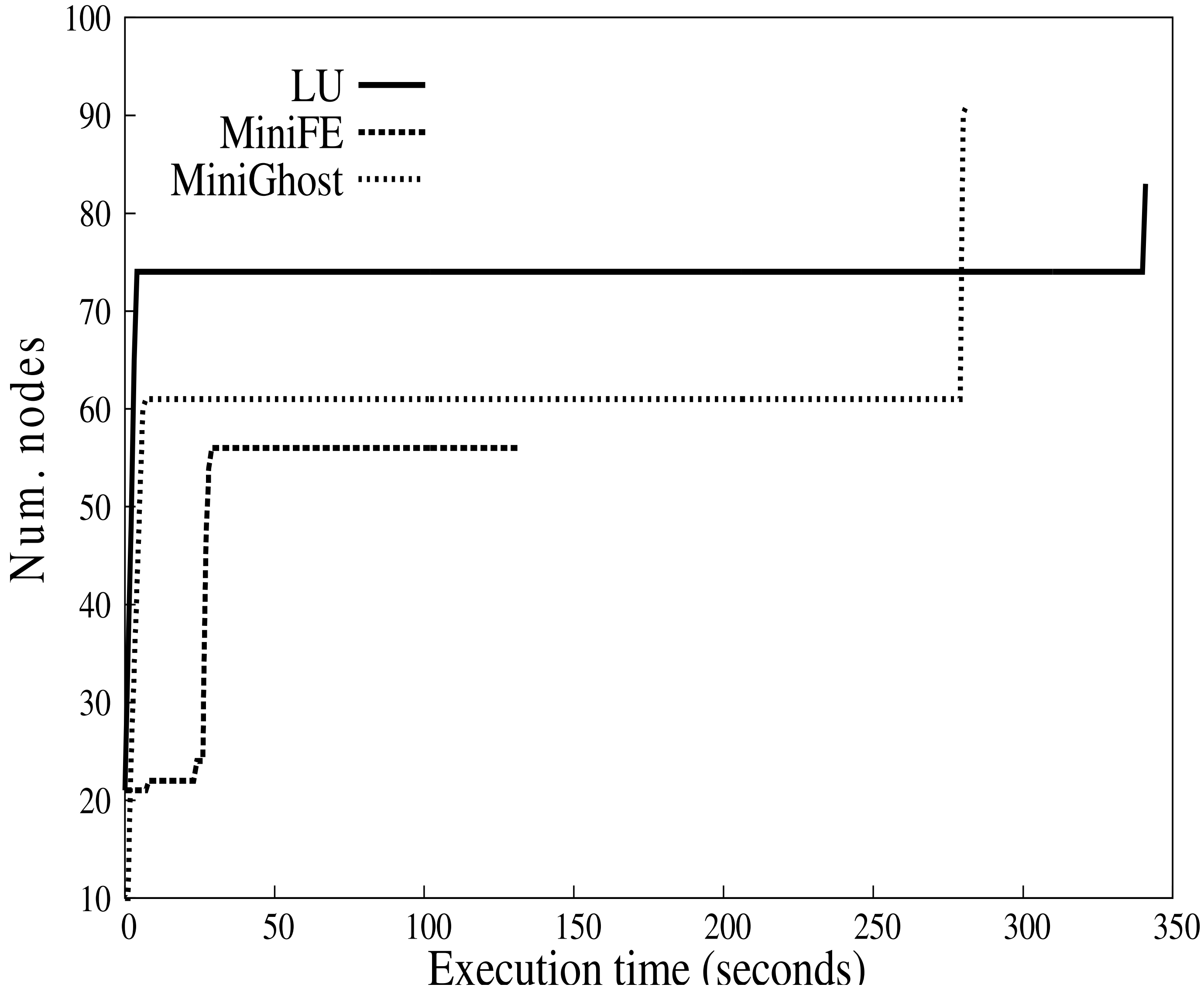


- Steady state?
  - No → do nothing
  - Yes → perform loop detection
- At main loop header?
  - No → do nothing
  - Yes → collect trace for N iterations ("smart data collection")

Detecting and Exploiting Structure Online

- Application structure can be detected online, while the application runs
  - Reduce redundant data, change data granularity, etc
- The event flow graph becomes stable once the application enters its iterative phase
- Our mechanism checks the number of nodes in the graph to detect application stability to trigger the loop detection mechanism

EFG Stability



Smart Data Collection – Experiments

- Six applications representing typical scientific codes
  - MiniGhost
  - MiniFE
  - MiniMD
  - GTC
  - LU
  - BT

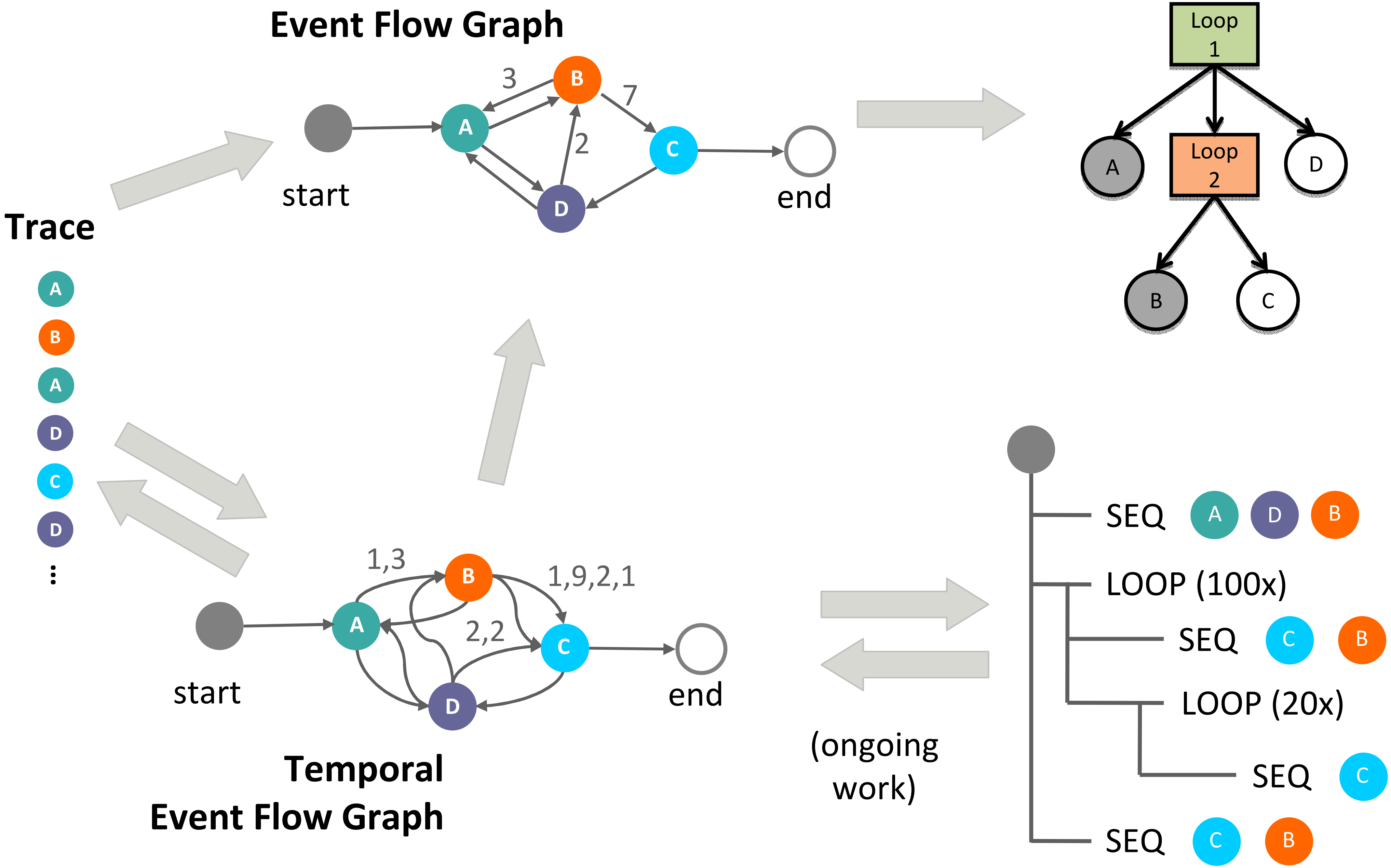
- Cray XE6 with 2 twelve-core AMD MagnyCours at 2.1 GHz
  - 32 GB DDR3 memory per node
  - Nodes interconnected with Cray Gemini network

Smart Data Collection – Trace Size

Metric	Mini-Ghost	MiniFE	GTC	MiniMD	BT	LU
Trace size	26 MB	77 MB	48 MB	555 MB	717 MB	7.7 GB
10 iterations trace	4.4 MB	4.1 MB	1.3 MB	788 KB	29 MB	267 MB
% reduced	83%	94.7%	97.3%	99.8%	96%	96.53%

- Detect the application structure on-line to keep tracing information of only 10 iterations of the main loop
- If the application is regular, a few iterations will represent the overall performance behaviour
- Performance results (statistics) still representative

Overview (3)

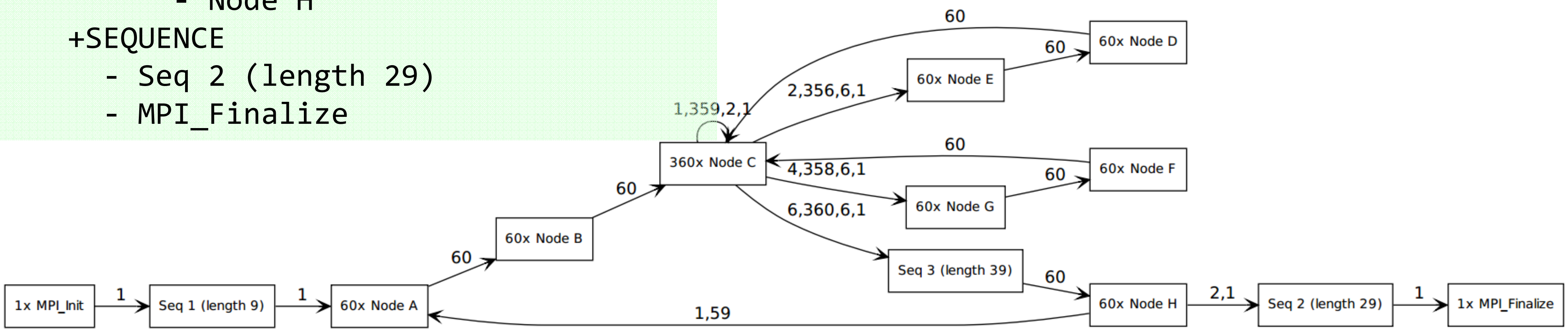

Event Flow Graph
3
7
2
A
B
C
D
start
end
Loop
1
Loop
2
A
D
B
C
Trace
A
B
A
D
C
D
1,3
1,9,2,1
2,2
start
end
Temporal
Event Flow Graph
(ongoing
work)
SEQ
A
D
B
LOOP (100x)
SEQ
C
B
LOOP (20x)
SEQ
C
SEQ
C
B

Example: MiniGhost

```
+ROOT
   +SEQUENCE
     - MPI_Init
     - Seq 1 (length 9)
   +LOOP (60x)
       +SEQUENCE
         - Node A, Node B
       +LOOP (6x)
          +SEQUENCE [3,3,0,1]
            - Node C, Node G, Node F
          +SEQUENCE [1,1,0,1]
            - Node C, Node E, Node D
          +SEQUENCE [0,2,2,1 | 4,5,0,2]
            - Node C
       +SEQUENCE
         - Seq 3 (length 39)
         - Node H
   +SEQUENCE
     - Seq 2 (length 29)
     - MPI_Finalize
```

predicate guards the activation of the node

- Compact and clear representation of what the application does
- Code generation straightforward



Conclusions

- Event flow graphs together with graph cycle detection algorithms are able to detect MPI application structure

- No source instrumentation needed
  - Graphs captured through the PMPI interface

- Some use cases:
  - Map performance data to program structure
  - Reduce amount of data collected while application runs

- Converting t-EFGs to trees onging work
  - Exciting possibilities: analysis, modeling, code generation, …