

*Master thesis*  
Master in Research and Innovation

**Inferring programs structure from  
an execution trace**

*Author*

Juan Francisco Martínez Vera

*Supervisor*

Jesús Labarta Mancho

Facultat d'Informàtica de Barcelona (FIB)  
Universitat Politècnica de Catalunya (UPC)



# Outline for section 1

## 1 Context

- Performance Analysis

## 2 Motivations

## 3 Objectives

## 4 State of the Art

## 5 Proposal

- Application structure by classification

## 6 Implementation

- Workflow

## 7 Best features analysis

## 8 Validation

- Lulesh 2.0
- CG

## 9 Scalability

## 10 Conclusions

## 11 Future work

# Performance Analysis

- Aid to detect **bottlenecks**
  - That prevents from better performance
- Requires high skilled analyst...
- ... so developers are not used to work with them
  - But derive this work to actual specialist
  - Analyst are used to **work with codes they are not familiar with.**
  - e.g. POP project: Provides **performance optimisation and productivity** services for academic and industrial code(s).



# Outline for section 2

## 1 Context

- Performance Analysis

## 2 Motivations

## 3 Objectives

## 4 State of the Art

## 5 Proposal

- Application structure by classification

## 6 Implementation

- Workflow

## 7 Best features analysis

## 8 Validation

- Lulesh 2.0
- CG

## 9 Scalability

## 10 Conclusions

## 11 Future work

# Motivations

## About improve understandability

Providing application structure will lead to better understandability about what the application is doing.

# Motivations

## About improve understandability

Providing application structure will lead to better understandability about what the application is doing.

## About improve reports

Having the structure of the application the communication with developer can be improved

# Outline for section 3

- 1 Context
  - Performance Analysis
- 2 Motivations
- 3 Objectives
- 4 State of the Art
- 5 Proposal
  - Application structure by classification
- 6 Implementation
  - Workflow
- 7 Best features analysis
- 8 Validation
  - Lulesh 2.0
  - CG
- 9 Scalability
- 10 Conclusions
- 11 Future work

# Objective

The objective for this thesis is...

**Expose the structure** of the application, by means of a post-mortem trace analysis.

# Outline for section 4

- 1 Context
  - Performance Analysis
- 2 Motivations
- 3 Objectives
- 4 State of the Art
- 5 Proposal
  - Application structure by classification
- 6 Implementation
  - Workflow
- 7 Best features analysis
- 8 Validation
  - Lulesh 2.0
  - CG
- 9 Scalability
- 10 Conclusions
- 11 Future work

## State of the Art

First step is always to explore what is over there...

- Some previous research have been driven in this field.
- Since traces are ordered sequences...
- ... Natural choice has been **sequential pattern mining** in general.

## State of the Art

First step is always to explore what is over there...

- Some previous research have been driven in this field.
- Since traces are ordered sequences...
- ... Natural choice has been **sequential pattern mining** in general.

Just **pick and develop one of the proposals?**

## State of the Art

First step is always to explore what is over there...

- Some previous research have been driven in this field.
- Since traces are ordered sequences...
- ... Natural choice has been **sequential pattern mining** in general.

Just **pick and develop one of the proposals?**

**Looking to the trend** of increasing trace sizes...

- This sort of algorithms, **used to present high complexity**.
- From  $O(n^2)$  to  $O(2^n)$ .

## State of the Art

First step is always to explore what is over there...

- Some previous research have been driven in this field.
- Since traces are ordered sequences...
- ... Natural choice has been **sequential pattern mining** in general.

Just pick and develop one of the proposals?

Looking to the trend of increasing trace sizes...

- This sort of algorithms, **used to present high complexity**.
- From  $O(n^2)$  to  $O(2^n)$ .

Finally...

The decision has been to explore a new approach!

# Outline for section 5

## 1 Context

- Performance Analysis

## 2 Motivations

## 3 Objectives

## 4 State of the Art

## 5 Proposal

- Application structure by classification

## 6 Implementation

- Workflow

## 7 Best features analysis

## 8 Validation

- Lulesh 2.0
- CG

## 9 Scalability

## 10 Conclusions

## 11 Future work

## Application structure by classification

### The key idea

Use communications as **proxies** for the observation of iterations and clustering them **by its behaviour**.

# Application structure by classification

## The key idea

Use communications as **proxies** for the observation of iterations and clustering them **by its behaviour**.

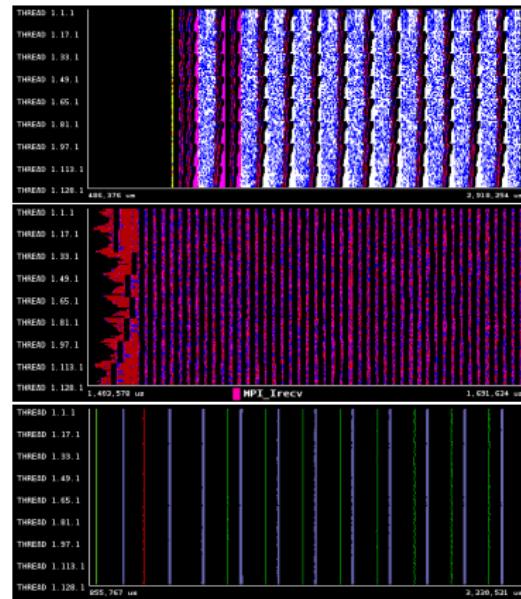
## Why?

HPC applications idiosyncracy

- Big outer loop
- Repetitive and stable executions
- Communications (MPI) lies on loops that drives the execution

So...

- **Similar behaviour for all iterations** in a given loop
- Loops by **monitoring the communications**.



# Application structure by classification

What will define **behaviour**?

Selected features must be able to

- Join MPIs from the same loop
- Separate MPIs from different loops

# Application structure by classification

What will define **behaviour**?

Selected features must be able to

- Join MPIs from the same loop
- Separate MPIs from different loops

As a starting point ...

- ① **Number of repetitions:** Two different mpi calls in same loop will be executed the same number of times.
- ② **Mean time between repetitions:** Two different loops will, presumably, execute different work.

# Outline for section 6

## 1 Context

- Performance Analysis

## 2 Motivations

## 3 Objectives

## 4 State of the Art

## 5 Proposal

- Application structure by classification

## 6 Implementation

### • Workflow

## 7 Best features analysis

## 8 Validation

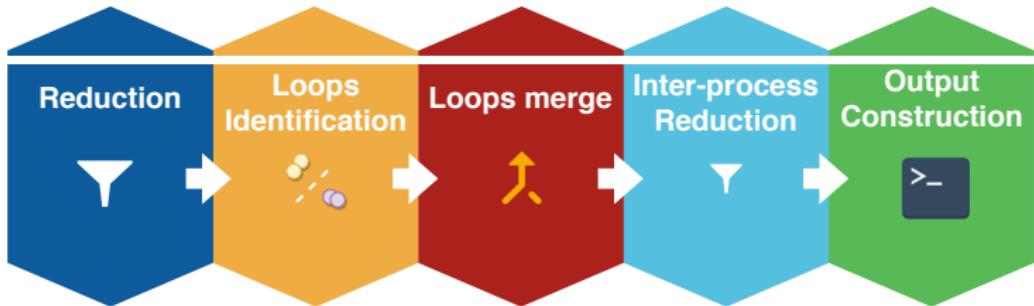
- Lulesh 2.0
- CG

## 9 Scalability

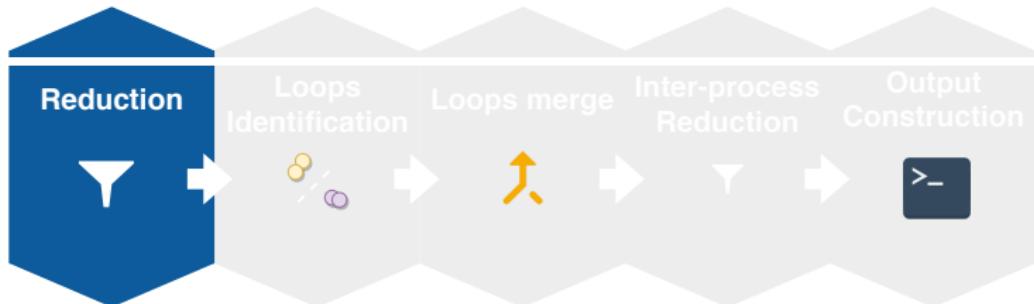
## 10 Conclusions

## 11 Future work

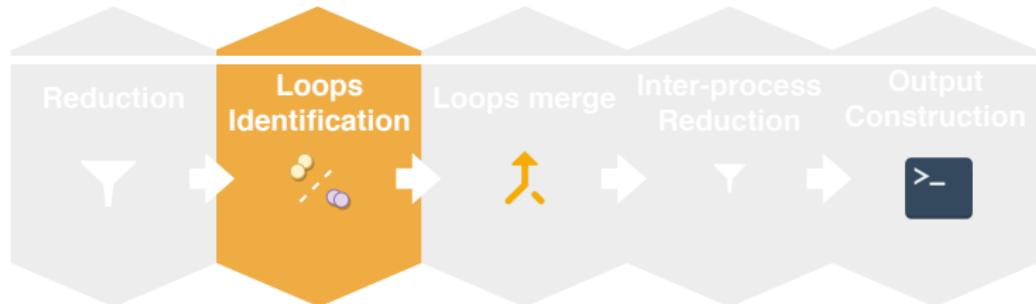
# Workflow



# Workflow

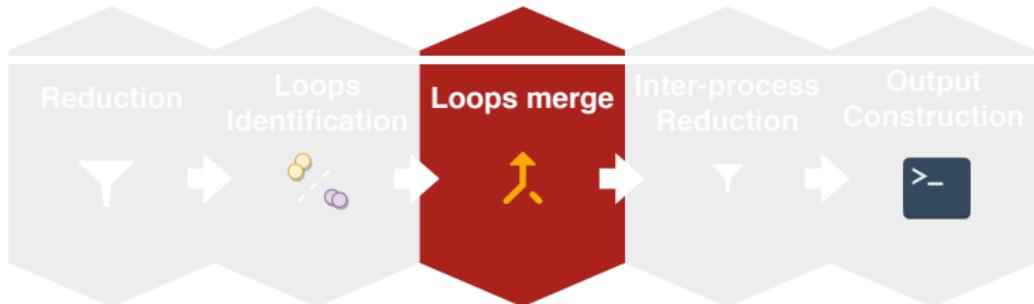


- Parsing and reducing information from trace



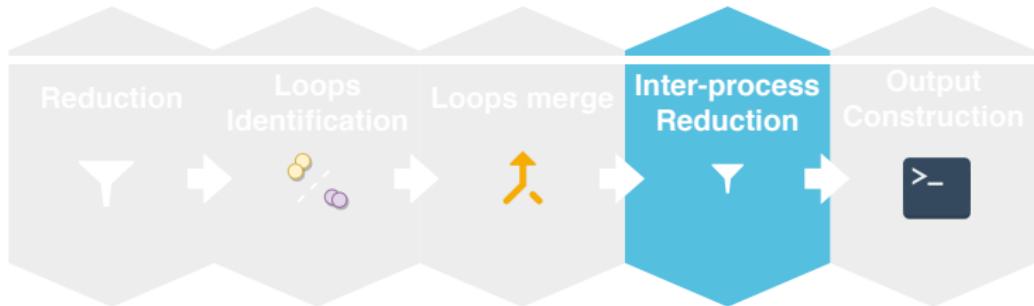
- Identify loops by MPI call sites classification

# Workflow



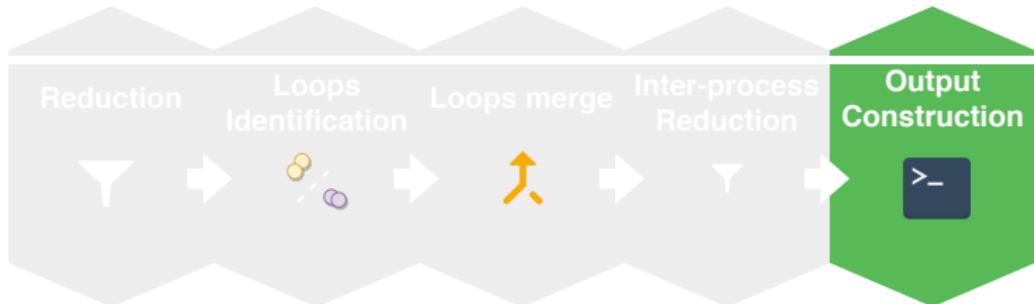
- Find out the hierarchical relations between loops

# Workflow

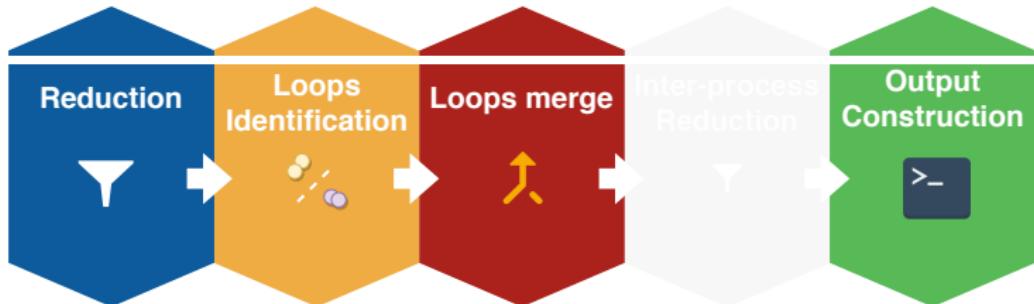


- Collapse same MPI call site from different processes

# Workflow



- Construction of the output



- Parsing and reducing information from trace
- Identify loops by MPI call sites classification
- Find out the hierarchical relations between loops
- Construction of the output

# Workflow



Trace reduction step

**Input** Tracefile, i.e. Sequence of timestamped events ordered by time.  
**Output** Set of unique MPI calls sites with attached information.

**Algorithm 6.1:** ()

```
for 1 to 50
    do { for 1 to 2
          do { MPI_A   ○
                MPI_B   ◊
    for 1 to 100
        do { for 1 to 2
              do { for 1 to 2
                    do { MPI_C
                          MPI_D
                MPI_E }
```

# Workflow

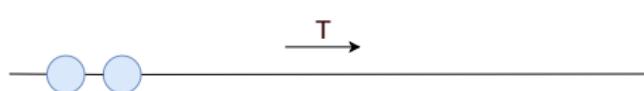


## Trace reduction step

**Input** Tracefile, i.e. Sequence of timestamped events ordered by time.  
**Output** Set of unique MPI calls sites with attached information.

### Algorithm 6.2: ()

```
for 1 to 50
    do { for 1 to 2
          do { MPI_A
                MPI_B
            }
        }
for 1 to 100
    do { for 1 to 2
          do { for 1 to 2
                do { MPI_C
                      MPI_D
                  }
                MPI_E
            }
        }
```





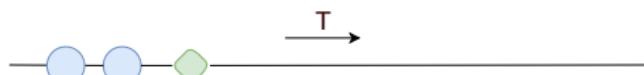
# Workflow

Trace reduction step

**Input** Tracefile, i.e. Sequence of timestamped events ordered by time.  
**Output** Set of unique MPI calls sites with attached information.

**Algorithm 6.3:** ()

```
for 1 to 50
    do { for 1 to 2
          do { MPI_A   ○
                MPI_B   □
          }
    }
for 1 to 100
    do { for 1 to 2
          do { for 1 to 2
                do { MPI_C
                      MPI_D
                }
          }
          MPI_E
    }
```



# Workflow

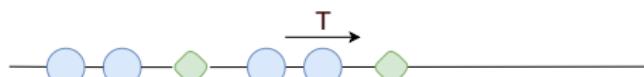


Trace reduction step

**Input** Tracefile, i.e. Sequence of timestamped events ordered by time.  
**Output** Set of unique MPI calls sites with attached information.

**Algorithm 6.4:** ()

```
for 1 to 50
    do { for 1 to 2
          do { MPI_A   ○
                MPI_B   ◊
            }
        }
for 1 to 100
    do { for 1 to 2
          do { for 1 to 2
                do { MPI_C
                      MPI_D
                  }
                MPI_E
            }
        }
```



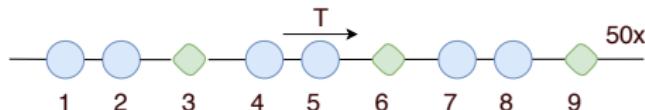
# Workflow

## Trace reduction step

**Input** Tracefile, i.e. Sequence of timestamped events ordered by time.  
**Output** Set of unique MPI calls sites with attached information.

### Algorithm 6.5: ()

```
for 1 to 50
    do { for 1 to 2
        do { MPI_A   ○
              MPI_B   ◊
    for 1 to 100
        do { for 1 to 2
            do { for 1 to 2
                do { MPI_C
                      MPI_D
              MPI_E }
```



# Workflow

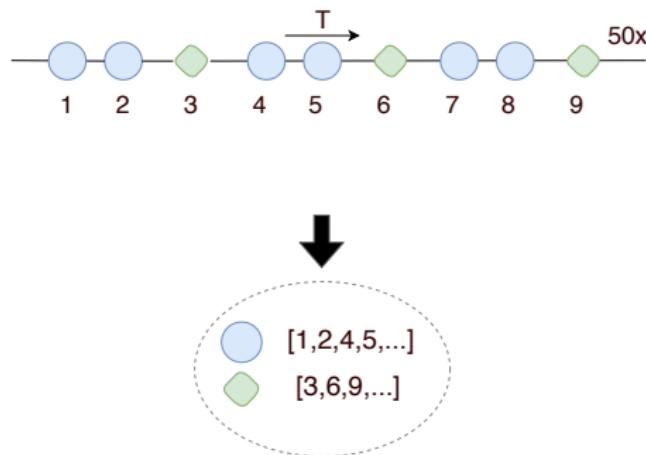


Trace reduction step

**Input** Tracefile, i.e. Sequence of timestamped events ordered by time.  
**Output** Set of unique MPI calls sites with attached information.

**Algorithm 6.6:** ()

```
for 1 to 50
  do { for 1 to 2
        do { MPI_A
              MPI_B
            }
      for 1 to 100
        do { for 1 to 2
              do { for 1 to 2
                    do { MPI_C
                          MPI_D
                        }
                  MPI_E
                }
              }
            }
```

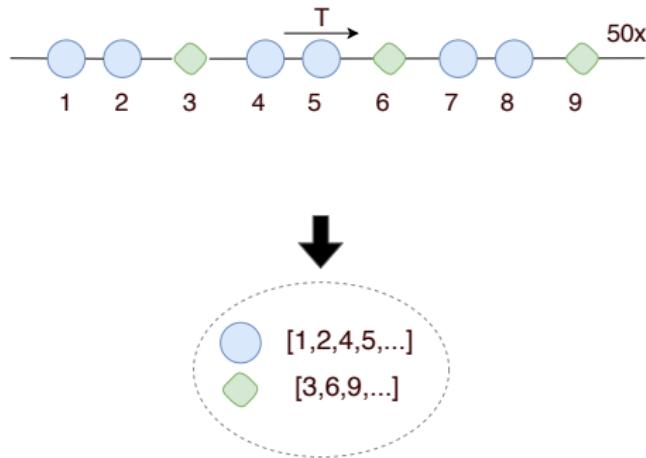




# Workflow

## Trace reduction step

- Is a sort of Map & Reduce
- Every MPI call is identified by its **signature**: Ordered sequence of pairs (*file, line*) that define the call path, i.e. The dynamic position
- Additionally **less representative MPI calls are filtered** (10%).

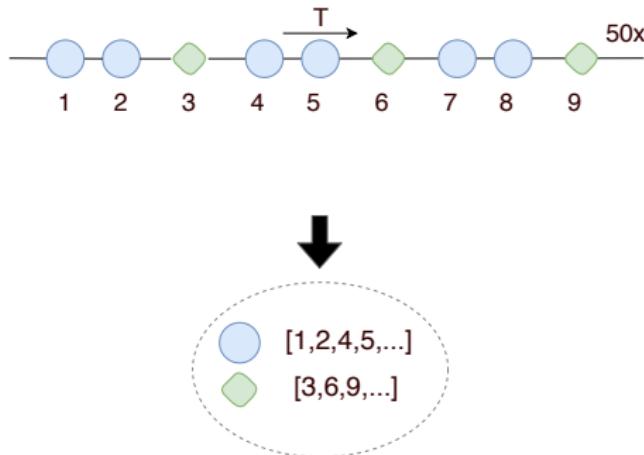




# Workflow

## Trace reduction step

- Is a sort of Map & Reduce
- Every MPI call is identified by its **signature**: Ordered sequence of pairs (*file, line*) that define the call path, i.e. The dynamic position
- Additionally **less representative MPI calls are filtered** (10%).



## Keynote

Trace reduction is **the key step for scalability** because repetitiveness of HPC applications.



# Workflow

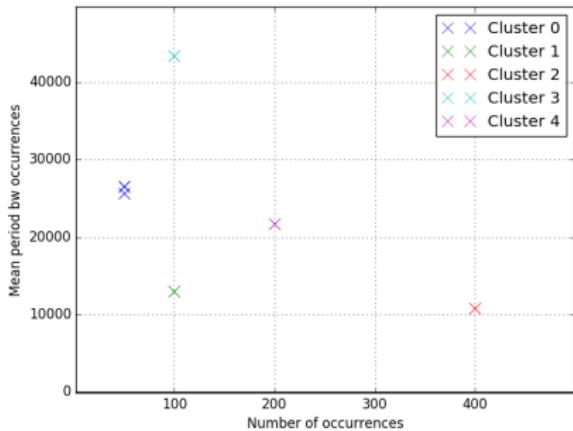
## Loops identification

**Input** Set of unique MPI calls sites with attached information.

**Output** Set of sets of MPI calls → Loops

### Algorithm 6.7: ()

```
for 1 to 50
  do { for 1 to 2
        do { MPI_A
              MPI_B
            for 1 to 100
              do { for 1 to 2
                    do { for 1 to 2
                          do { MPI_C
                                MPI_D
                              MPI_E
                            }
```





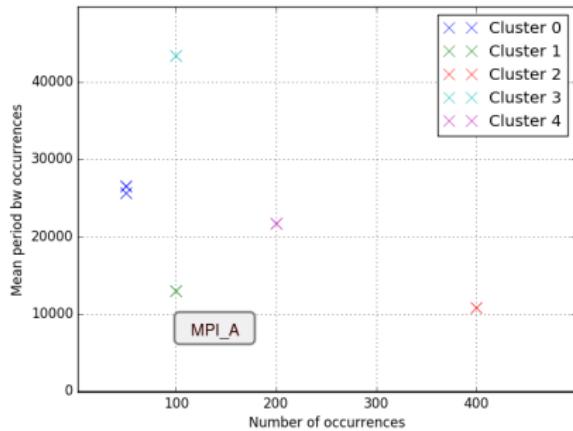
# Workflow

## Loops identification

**Input** Set of unique MPI calls sites with attached information.  
**Output** Set of sets of MPI calls → Loops

### Algorithm 6.8: ()

```
for 1 to 50
  do { for 1 to 2
        do { MPI_A
              MPI_B
            for 1 to 100
              do { for 1 to 2
                    do { for 1 to 2
                          do { MPI_C
                                MPI_D
                              MPI_E
                            }
```





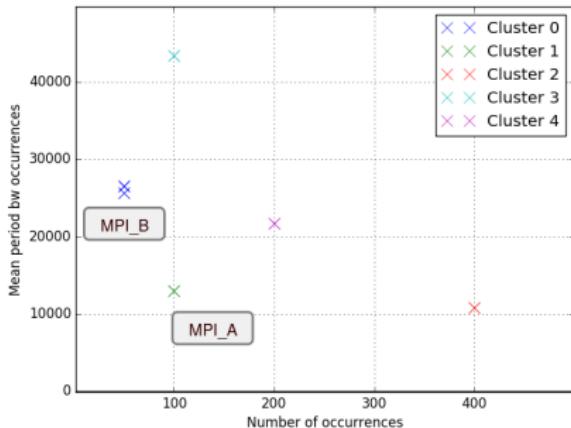
# Workflow

## Loops identification

**Input** Set of unique MPI calls sites with attached information.  
**Output** Set of sets of MPI calls → Loops

### Algorithm 6.9: ()

```
for 1 to 50
  do { for 1 to 2
        do { MPI_A
              MPI_B
            for 1 to 100
              do { for 1 to 2
                    do { for 1 to 2
                          do { MPI_C
                                MPI_D
                              MPI_E
```



# Workflow



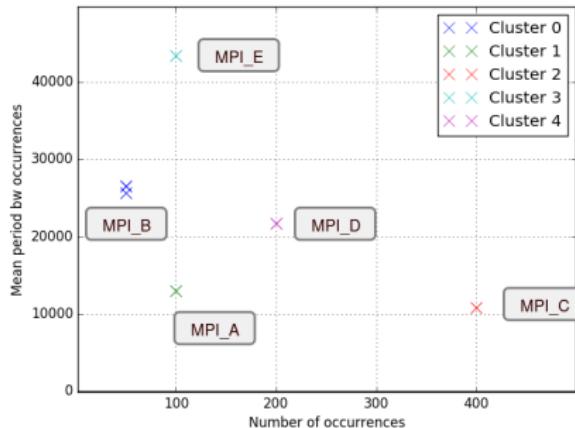
## Loops identification

**Input** Set of unique MPI calls sites with attached information.

**Output** Set of sets of MPI calls → Loops

### Algorithm 6.10: ()

```
for 1 to 50
  do { for 1 to 2
        do { MPI_A
              MPI_B
            for 1 to 100
              do { for 1 to 2
                    do { for 1 to 2
                          do { MPI_C
                                MPI_D
                              MPI_E
                            }
```

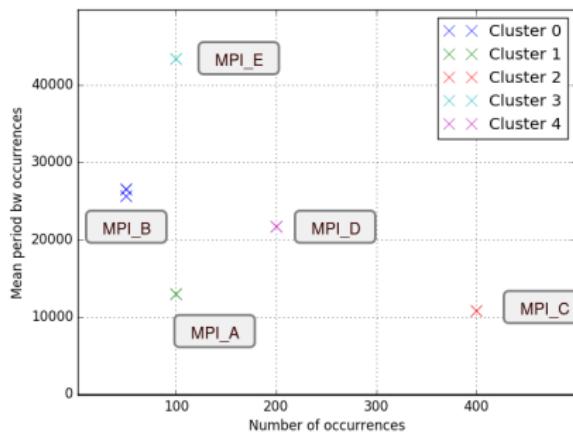




# Workflow

## Loops identification

- Need for some classification algorithm
- **DBSCAN** as clustering algorithm (preferrfed to K-means)
  - $\epsilon$  empirically set to 0.2 (in general)
  - minPts set to 1
- Resulting clusters **will be considered loops.**
  - Number of repetitions → Number of iterations.
  - Mean time between repetitions → Mean iterations time.





# Workflow

## Loops merge

**Input** Set of loops.

**Output** Set of top level loops with its related nested loops.



# Workflow

## Loops merge

**Input** Set of loops.

**Output** Set of top level loops with its related nested loops.

## Intuition

- Isolated loops are just **pieces of the overall puzzle**.
- By discover its hierarchical relations **the structure of the application will be discovered...**
- As well as the different phases



# Workflow

## Loops merge

**Input** Set of loops.

**Output** Set of top level loops with its related nested loops.

## Intuition

- Isolated loops are just **pieces of the overall puzzle**.
- By discover its hierarchical relations **the structure of the application will be discovered...**
- As well as the different phases

We can assume

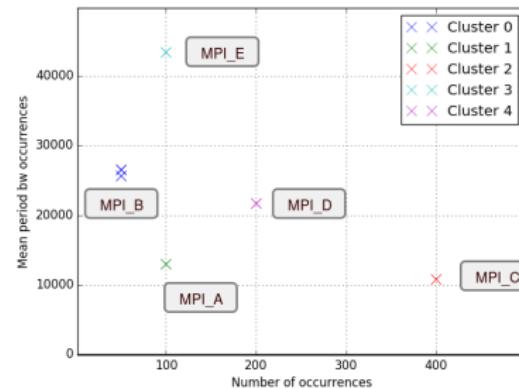
- Outer loop will have **less iterations** than nested one.
- Outer loop will spend **more time** per iteration.

# Workflow



## Loops merge

```
for 1 to 50
  do { for 1 to 2
        do { MPI_A
              MPI_B
            }
      for 1 to 100
        do { for 1 to 2
              do { for 1 to 2
                    do { MPI_C
                          MPI_D
                        }
                  MPI_E
                }
              }
            }
```

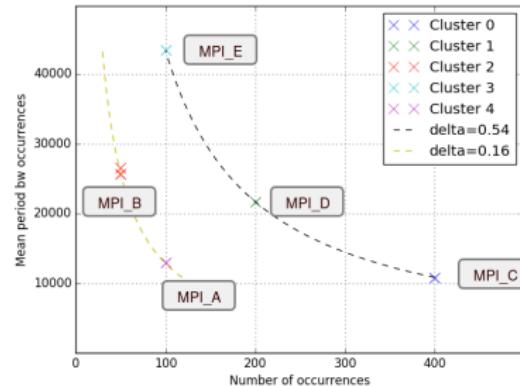


# Workflow



## Loops merge

```
for 1 to 50
  do { for 1 to 2
        do { MPI_A
              MPI_B
            }
      for 1 to 100
        do { for 1 to 2
              do { for 1 to 2
                    do { MPI_C
                          MPI_D
                        }
                  MPI_E
                }
              }
            }
```



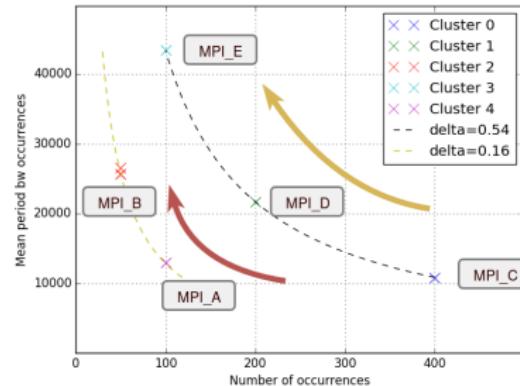
- Classify loops by its delta
  - All lies on same  $f(x) = \frac{\delta * T_{exe}}{x}$  being  $0 < \delta < 1$
- And then merge them in a less to more iterations fashion
  - Always checking iterations interleaving

# Workflow



## Loops merge

```
for 1 to 50
  do { for 1 to 2
        do { MPI_A
              MPI_B
            }
      for 1 to 100
        do { for 1 to 2
              do { for 1 to 2
                    do { MPI_C
                          MPI_D
                        }
                  MPI_E
                }
              }
            }
```



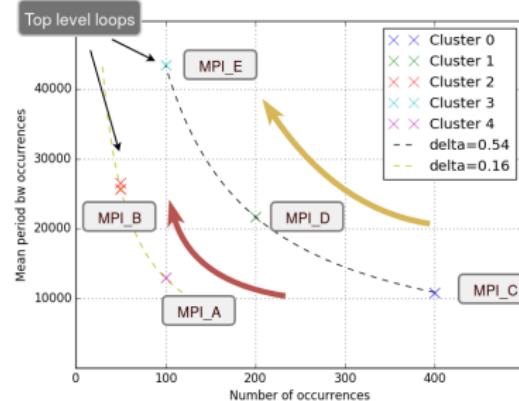
- Classify loops by its delta
  - All lies on same  $f(x) = \frac{\delta * T_{exe}}{x}$  being  $0 < \delta < 1$
- And then merge them in a less to more iterations fashion
  - Always checking iterations interleaving

# Workflow



## Loops merge

```
for 1 to 50
  do { for 1 to 2
        do { MPI_A
              MPI_B
            for 1 to 100
              do { for 1 to 2
                    do { MPI_C
                          MPI_D
                        MPI_E
```



- Classify loops by its delta
  - All lies on same  $f(x) = \frac{\delta * T_{exe}}{x}$  being  $0 < \delta < 1$
- And then merge them in a less to more iterations fashion
  - Always checking iterations interleaving

## Keynote

This mechanism is used for phases detection.



# Workflow

## Pseudocode construction

**Input** Set of top level loops with rank conditional structures.

**Output** Pseudocode representing the actual application structure.



# Workflow

## Pseudocode construction

**Input** Set of top level loops with rank conditional structures.

**Output** Pseudocode representing the actual application structure.

- Straightforward construction
- Just a refinement of the data is needed
  - ① Extracting common call path levels from code block (loops and conditional blocks)
  - ② Removing **contiguous repetitive information** what has not been removed in previous step

# Workflow



## Pseudocode construction

FILE	LINE	PSEUDOCODE	E(TIME)	E(SIZE)	E(IPC)
	0	main()	-	-	-
	17	: FOR 1 TO 50	-	-	-
test-13.c	17	: MPI_Barrier(CommId:1)	26.43us	-	0.3
	17	: FOR 1 TO 2.0	-	-	-
	17	: IF rank in [1]	-	-	-
test-13.c	23	: MPI_Send(i:0)	22.8us	4.0B	0.53
	17	: IF rank in [0]	-	-	-
test-13.c	25	: MPI_Recv()	27.09us	0.12B	0.54
	17	: END LOOP	-	-	-
<b>test-13.c</b>	<b>28</b>	<b>: [- computation -]</b>	<b>10.18ms</b>	<b>-</b>	<b>0.06</b>
test-13.c	28	: MPI_Barrier(CommId:1)	106.13us	-	1.43
	28	: END LOOP	-	-	-
	0	main()	-	-	-
	1	: FOR 1 TO 100	-	-	-
	1	: FOR 1 TO 2.0	-	-	-
	1	: FOR 1 TO 2.0	-	-	-
	1	: IF rank in [1]	-	-	-
test-13.c	40	: MPI_Send(i:0)	23.0us	4.0B	0.5
	1	: IF rank in [0]	-	-	-
test-13.c	42	: MPI_Recv()	25.1us	0.05B	0.51
	1	: END LOOP	-	-	-
test-13.c	45	: MPI_Barrier(CommId:1)	23.93us	-	0.84
	1	: END LOOP	-	-	-
<b>test-13.c</b>	<b>48</b>	<b>: [- computation -]</b>	<b>50.2ms</b>	<b>-</b>	<b>0.07</b>
test-13.c	48	: MPI_Barrier(CommId:1)	124.99us	-	1.62
	48	: END LOOP	-	-	-

# Workflow



## Pseudocode construction

FILE	LINE	PSEUDOCODE	E(TIME)	E(SIZE)	E(IPC)
test-13.c	0	main()	-	-	-
test-13.c	17	: FOR 1 TO 50 : : MPI_Barrier(CommId:1)	26.43us	-	0.3
test-13.c	23	: FOR 1 TO 2.0 : : IF rank in [1]	-	-	-
test-13.c	25	: : : MPI_Send(i:0) : : IF rank in [0]	22.8us	4.0B	0.53
test-13.c	25	: : : MPI_Recv() : END LOOP	-	-	-
test-13.c	28	: [- computation -] : MPI_Barrier(CommId:1)	10.18ms	-	0.06
test-13.c	28	: END LOOP	106.13us	-	1.43
test-13.c	0	main()	-	-	-
test-13.c	1	: FOR 1 TO 100	-	-	-
test-13.c	1	: : FOR 1 TO 2.0	-	-	-
test-13.c	1	: : : FOR 1 TO 2.0	-	-	-
test-13.c	1	: : : : IF rank in [1]	-	-	-
test-13.c	40	: : : : MPI_Send(i:0) : : : : IF rank in [0]	23.0us	4.0B	0.5
test-13.c	42	: : : : MPI_Recv() : : END LOOP	25.1us	0.05B	0.51
test-13.c	45	: : MPI_Barrier(CommId:1) : END LOOP	23.93us	-	0.84
test-13.c	48	: [- computation -]	50.2ms	-	0.07
test-13.c	48	: MPI_Barrier(CommId:1) : END LOOP	124.99us	-	1.62

# Workflow



## Pseudocode construction

FILE	LINE	PSEUDOCODE	E(TIME)	E(SIZE)	E(IPC)
	0	main()	--	--	--
test-13.c	17	: FOR 1 TO 50 : : MPI_Barrier(CommId:1)	26.43us	--	0.3
		: FOR 1 TO 2.0	--	--	--
		: : IF rank in [1]	--	--	--
test-13.c	23	: : : MPI_Send(i:0)	22.8us	4.0B	0.53
		: : : IF rank in [0]	--	--	--
test-13.c	25	: : : MPI_Recv()	27.09us	0.12B	0.54
		: END LOOP	--	--	--
test-13.c	28	: [- computation -]	10.18ms	--	0.06
test-13.c	28	: : MPI_Barrier(CommId:1)	106.13us	--	1.43
		: END LOOP	--	--	--
	0	main()	--	--	--
		: FOR 1 TO 100	--	--	--
		: : FOR 1 TO 2.0	--	--	--
		: : : FOR 1 TO 2.0	--	--	--
		: : : : IF rank in [1]	--	--	--
test-13.c	40	: : : : : MPI_Send(i:0)	23.0us	4.0B	0.5
		: : : : : IF rank in [0]	--	--	--
test-13.c	42	: : : : : MPI_Recv()	25.1us	0.05B	0.51
		: : : END LOOP	--	--	--
test-13.c	45	: : : MPI_Barrier(CommId:1)	23.93us	--	0.84
		: : END LOOP	--	--	--
test-13.c	48	: [- computation -]	50.2ms	--	0.07
test-13.c	48	: : MPI_Barrier(CommId:1)	124.99us	--	1.62
		: END LOOP	--	--	--

# Workflow



## Pseudocode construction

FILE	LINE	PSEUDOCODE	E(TIME)	E(SIZE)	E(IPC)
		0 main()	-	-	-
test-13.c	17	: FOR 1 TO 50	-	-	-
		: : MPI_Barrier(CommId:1)	26.43us	-	0.3
		: : FOR 1 TO 2.0	-	-	-
		: : : IF rank in [1]	-	-	-
test-13.c	23	: : : MPI_Send(i:0)	22.8us	4.0B	0.53
		: : : IF rank in [0]	-	-	-
test-13.c	25	: : : MPI_Recv()	27.09us	0.12B	0.54
		: : END LOOP	-	-	-
<b>test-13.c</b>	<b>28</b>	<b> : [- computation -]</b>	<b>10.18ms</b>	<b>-</b>	<b>0.06</b>
test-13.c	28	: : MPI_Barrier(CommId:1)	106.13us	-	1.43
		: : END LOOP	-	-	-
		0 main()	-	-	-
		: FOR 1 TO 100	-	-	-
		: : FOR 1 TO 2.0	-	-	-
		: : : FOR 1 TO 2.0	-	-	-
		: : : : IF rank in [1]	-	-	-
test-13.c	40	: : : : MPI_Send(i:0)	23.0us	4.0B	0.5
		: : : : IF rank in [0]	-	-	-
test-13.c	42	: : : : MPI_Recv()	25.1us	0.05B	0.51
		: : : END LOOP	-	-	-
test-13.c	45	: : : MPI_Barrier(CommId:1)	23.93us	-	0.84
		: : END LOOP	-	-	-
<b>test-13.c</b>	<b>48</b>	<b> : [- computation -]</b>	<b>50.2ms</b>	<b>-</b>	<b>0.07</b>
test-13.c	48	: : MPI_Barrier(CommId:1)	124.99us	-	1.62
		: : END LOOP	-	-	-

# Workflow



## Pseudocode construction

FILE	LINE	PSEUDOCODE	E(TIME)	E(SIZE)	E(IPC)
	0	main()	-	-	-
test-13.c	17	: FOR 1 TO 50   : MPI_BARRIER(CommId:1)   : FOR 1 TO 2.0	26.43us	-	0.3
test-13.c	23	: : IF rank in [1]   : : MPI_Send(i:0)	22.8us	4.0B	0.53
test-13.c	25	: : IF rank in [0]   : : MPI_Recv()	27.09us	0.12B	0.54
		: END_LOOP	-	-	-
test-13.c	28	: [~ computation ~]	10.18ms	-	0.06
test-13.c	28	: MPI_BARRIER(CommId:1)   : END_LOOP	106.13us	-	1.43
	0	main()	-	-	-
		: FOR 1 TO 100	-	-	-
		: FOR 1 TO 2.0	-	-	-
		: : FOR 1 TO 2.0	-	-	-
		: : IF rank in [1]	-	-	-
test-13.c	40	: : : MPI_Send(i:0)	23.0us	4.0B	0.5
test-13.c	42	: : : IF rank in [0]   : : : MPI_Recv()	25.1us	0.05B	0.51
test-13.c	45	: : : END_LOOP	-	-	-
test-13.c	45	: : MPI_BARRIER(CommId:1)	23.93us	-	0.84
		: END_LOOP	-	-	-
test-13.c	48	: [~ computation ~]	150.2ms	-	0.07
test-13.c	48	: MPI_BARRIER(CommId:1)   : END_LOOP	124.99us	-	1.62

```
(struct_detection)> help

Documented commands (type help <topic>):
=====
clustering help paraver pseudocode q quit

(struct_detection)> help pseudocode
pseudocode commands:
wo-cs: Do not show call paths
w-burst-info: Show CPU burst information
w-burst-threshold: Show burst above threshold
default: Default information
ranks: Show pseudocode just for rank
(struct_detection)> █
```

# Outline for section 7

## 1 Context

- Performance Analysis

## 2 Motivations

## 3 Objectives

## 4 State of the Art

## 5 Proposal

- Application structure by classification

## 6 Implementation

- Workflow

## 7 Best features analysis

## 8 Validation

- Lulesh 2.0
- CG

## 9 Scalability

## 10 Conclusions

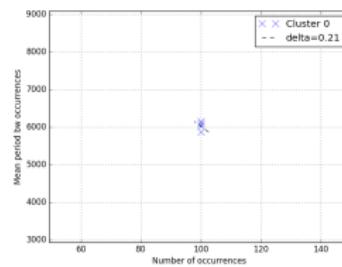
## 11 Future work

# Best features analysis

## Motivation

**Clustering aliasing** → Same behavior

```
for 1to100
  do {MPI_A()
       MPI_B())
for 1to100
  do {MPI_C()
       MPI_D())
```



# Best features analysis

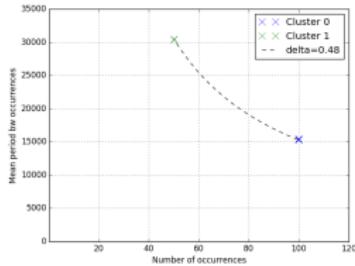
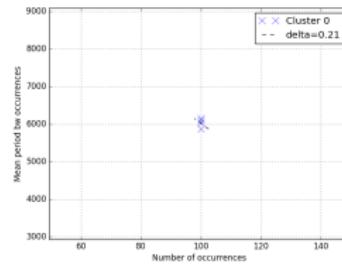
## Motivation

**Clustering aliasing** → Same behavior

```
for 1to100
  do {MPI_A()
       MPI_B()}
for 1to100
  do {MPI_C()
       MPI_D()}
```

**Clustering split** → Data conditions

```
for i = 1 to 10
  do {MPI_B()
       if isPair(i)
         then MPI_A()
       MPI_B()}
```



# Best features analysis

## Motivation

Both **solved**, but better to **avoid them**, so...

- **Validate** our first intuition
- **Explore** alternative/additional features
  - IPC, %Loads, %(Cond, Uncond, Total) branches <sup>1</sup>

---

<sup>1</sup>Strictly limited by available hardware

# Best features analysis

## Motivation

Both **solved**, but better to **avoid them**, so...

- **Validate** our first intuition
- **Explore** alternative/additional features
  - IPC, %Loads, %(Cond, Uncond, Total) branches <sup>1</sup>

Two steps:

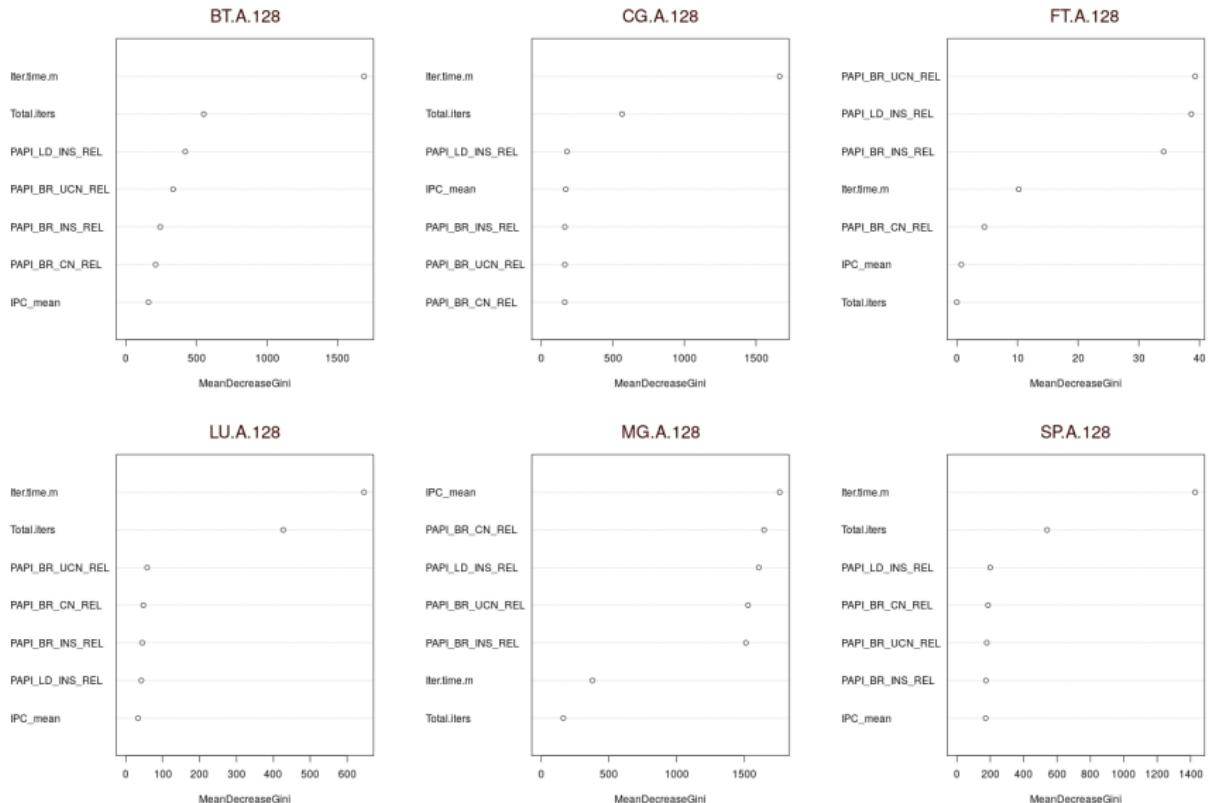
- ① Data acquisition
  - Used tracing library (Extrae) is monitoring calls to shared libraries
  - Loops monitors injection by **Mercurium**
- ② Analyze results
  - Principal Components Analysis
  - Random Forest Variable Importance

---

<sup>1</sup>Strictly limited by available hardware

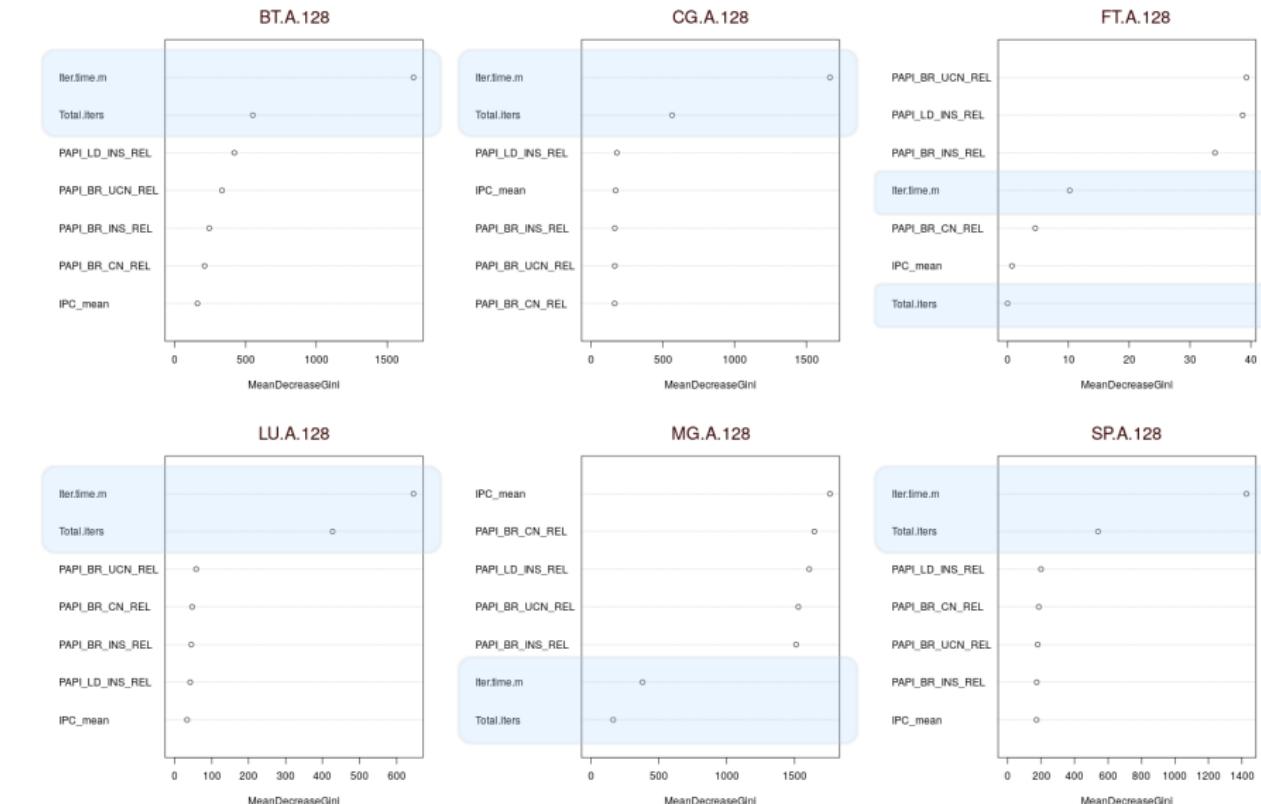
# Best features analysis

## Analysis of results: Variable Importance



# Best features analysis

## Analysis of results: Variable Importance



# Outline for section 8

## 1 Context

- Performance Analysis

## 2 Motivations

## 3 Objectives

## 4 State of the Art

## 5 Proposal

- Application structure by classification

## 6 Implementation

- Workflow

## 7 Best features analysis

## 8 Validation

- Lulesh 2.0
- CG

## 9 Scalability

## 10 Conclusions

## 11 Future work

# Validation

## Lulesh 2.0

Execution:

- 6014 code lines (C++)
- 128 parallel processes
- $\approx 160MB$  tracefile

Analysis:

- Filter all  $\delta < 10\%$
- CPU burst *time*  $> 6ms$

## Validation

## Lulesh 2.0

## Execution:

- 6014 code lines (C++)
  - 128 parallel processes
  - $\approx 160MB$  tracefile

## Analysis:

- Filter all  $\delta < 10\%$
  - CPU burst time  $> 6ms$

### Results:

- 30 iterations loop

# Validation

## Lulesh 2.0

### Execution:

- 6014 code lines (C++)
- 128 parallel processes
- $\approx 160MB$  tracefile

### Analysis:

- Filter all  $\delta < 10\%$
- CPU burst time  $> 6ms$

### Results:

- 30 iterations loop
- 5 long CPU burst

FILE	ILINE	PSEUDOCODE	E(TIME)	E(RISE)	E(IPC)
lulesh.cc	10011	DoIteration()	[~]	[~]	[~]
lulesh.cc	127741	; FOR 1 TO 30 [id=0-0]	[~]	[~]	[~]
lulesh.cc	127741	[- computation ~]	<b>156.63ms</b>	<b>[~]</b>	<b>[2.56]</b>
lulesh.cc	127741	; solve w/ Arf's explicit scheme	[~]	[~]	[~]
lulesh.cc	127741	; LagrangeLapFrcg()	[~]	[~]	[~]
lulesh.cc	12811	; LagrangeWall()	[~]	[~]	[~]
lulesh.cc	12823	; ConnSyncForNodes()	[~]	[~]	[~]
lulesh-comm.cc	12371	; i : i : ConnRecv()	[~]	[~]	[~]
lulesh-comm.cc	12381	; i : i : MPI_Comm_rank(0)	10.27ms	[~]	[0.28]
lulesh-comm.cc	12381	; i : i : MPI_Irecv(0 25)	10.51us	[22.53KB]	[0.55]
lulesh-comm.cc	12371	; i : i : MPI_Irecv(0 5)	18.67us	[22.52KB]	[0.96]
lulesh-comm.cc	12381	; i : i : MPI_Irecv(0 1)	8.43us	[22.52KB]	[1.08]
lulesh-comm.cc	12381	; i : i : MPI_Irecv(0 3)	8.43us	[144.0B]	[1.08]
lulesh-comm.cc	12201	; i : i : MPI_Irecv(0 30)	8.59us	[1744.0B]	[1.25]
lulesh-comm.cc	22101	; i : i : MPI_Irecv(0 25)	8.29us	[1744.0B]	[1.25]
lulesh-comm.cc	22101	; i : i : MPI_Irecv(0 11)	9.12us	[24.0B]	[0.9]
lulesh.cc	127741	[- computation ~]	<b>185.2ms</b>	<b>[~]</b>	<b>[2.48]</b>
lulesh-comm.cc	127741	; MPI_Comm_rank(0)	[~]	[~]	[~]
lulesh-comm.cc	14361	; i : i : MPI_Isend(0 25)	14.95us	[22.52KB]	[1.39]
lulesh-comm.cc	14361	; i : i : MPI_Isend(0 5)	10.07us	[22.52KB]	[1.65]
lulesh-comm.cc	12381	; i : i : MPI_Wait(0 1)	10.27us	[22.52KB]	[0.35]
lulesh-comm.cc	5891	; i : i : MPI_Isend(0 8)	14.79us	[1744.0B]	[1.68]
lulesh-comm.cc	6061	; i : i : MPI_Isend(0 30)	12.45us	[44.0B]	[1.86]
lulesh-comm.cc	12381	; i : i : MPI_Wait(0 1)	11.11us	[1744.0B]	[1.46]
lulesh-comm.cc	8371	; i : i : MPI_Isend(0 31)	9.48us	[24.0B]	[0.84]
lulesh-comm.cc	8421	; i : i : MPI_Waitall(0 1)	541.3us	[~]	[0.43]
lulesh-comm.cc	11011	; ConnSyncForNodes()	[~]	[~]	[~]
lulesh-comm.cc	18891	; i : i : MPI_Come_rank(0)	9.32us	[~]	[0.49]
lulesh-comm.cc	9111	; i : i : MPI_Wait(0 1)	9.25us	[~]	[0.4]
lulesh-comm.cc	12381	; i : i : MPI_Wait(0 1)	9.25us	[~]	[0.48]
lulesh-comm.cc	9801	; i : i : MPI_Wait(0 1)	8.52us	[~]	[2.35]
lulesh-comm.cc	10391	; i : i : MPI_Wait(0 1)	8.41us	[~]	[2.72]
lulesh-comm.cc	10391	; i : i : MPI_Wait(0 1)	8.41us	[~]	[2.64]
lulesh-comm.cc	10671	; i : i : MPI_Wait(0 1)	8.31us	[~]	[3.98]
lulesh-comm.cc	12551	; i : i : MPI_Wait(0 1)	58.9us	[~]	[1.44]
lulesh-comm.cc	12551	; i : i : MPI_Wait(0 1)	1.77us	[~]	[~]
lulesh-comm.cc	12551	; i : i : MPI_Come_rank(0)	7.77us	[~]	[0.73]
lulesh-comm.cc	11191	; i : i : MPI_Irecv(0 23)	9.03us	[45.0KB]	[0.78]
lulesh-comm.cc	12381	; i : i : MPI_Wait(0 1)	7.77us	[~]	[1.25]
lulesh-comm.cc	11551	; i : i : MPI_Irecv(0 11)	7.34us	[45.0KB]	[0.95]
lulesh-comm.cc	19211	; i : i : MPI_Irecv(0 4)	7.46us	[14.5KB]	[3.17]
lulesh-comm.cc	12381	; i : i : MPI_Wait(0 1)	7.46us	[14.5KB]	[1.33]
lulesh-comm.cc	22101	; i : i : MPI_Irecv(0 3)	7.45us	[14.5KB]	[1.13]
lulesh-comm.cc	3451	; i : i : MPI_Irecv(0 31)	7.5us	[48.0B]	[0.95]
lulesh-comm.cc	127741	[- computation ~]	<b>16.15ms</b>	<b>[~]</b>	<b>[2.0]</b>
lulesh-comm.cc	14831	; i : i : MPI_Waitall(0 1)	110.26us	[~]	[0.35]
lulesh-comm.cc	12921	; i : i : ConnSyncForNodes()	[~]	[~]	[~]
lulesh-comm.cc	133101	; i : i : MPI_Come_rank(0 1)	32.14us	[~]	[0.73]
lulesh-comm.cc	133101	; i : i : MPI_Wait(0 1)	12.7us	[~]	[0.79]
lulesh-comm.cc	133661	; i : i : MPI_Wait(0 1)	73.95us	[~]	[2.46]
lulesh-comm.cc	14021	; i : i : MPI_Wait(0 1)	23.24us	[~]	[2.3]
lulesh-comm.cc	14021	; i : i : MPI_Wait(0 1)	15.95us	[~]	[1.79]
lulesh-comm.cc	14751	; i : i : MPI_Wait(0 1)	7.83us	[~]	[1.83]
lulesh-comm.cc	14891	; i : i : MPI_Wait(0 1)	7.75us	[~]	[2.08]
lulesh-comm.cc	126561	; i : i : MPI_Wait(0 1)	333.07us	[~]	[11.73]
lulesh-comm.cc	126561	; i : i : ConnSyncForNodes()	[~]	[~]	[~]
lulesh-comm.cc	126561	; i : i : CalQForElement()	[~]	[~]	[~]
lulesh.cc	127741	[- computation ~]	<b>29.32ms</b>	<b>[~]</b>	<b>[12.25]</b>
lulesh-comm.cc	127741	; MPI_Irecv(0 23)	[~]	[~]	[~]
lulesh-comm.cc	127741	; i : i : MPI_Irecv(0 5)	15.26us	[12.0KB]	[0.43]
lulesh-comm.cc	127741	; i : i : MPI_Irecv(0 1)	9.23us	[12.0KB]	[1.03]
lulesh-comm.cc	127741	; i : i : MPI_Wait(0 1)	8.79us	[12.0KB]	[1.17]
lulesh.cc	127741	[- computation ~]	<b>15.35ms</b>	<b>[~]</b>	<b>[1.91]</b>
lulesh-comm.cc	127741	; MPI_Come_rank(0 1)	[~]	[~]	[~]
lulesh-comm.cc	127741	; i : i : MPI_Wait(0 1)	11.97us	[~]	[12.92]
lulesh-comm.cc	127741	; i : i : MPI_Wait(0 1)	11.14us	[12.0KB]	[1.08]
lulesh-comm.cc	4771	; i : i : MPI_Isend(0 15)	10.28us	[12.0KB]	[1.89]
lulesh-comm.cc	5181	; i : i : MPI_Isend(0 15)	10.57us	[12.0KB]	[1.26]
lulesh-comm.cc	12381	; i : i : MPI_Wait(0 1)	13.91us	[~]	[0.43]
lulesh-comm.cc	120141	; i : i : ConnSyncForNodes()	[~]	[~]	[~]
lulesh-comm.cc	127341	; i : i : MPI_Come_rank(0 1)	9.51us	[~]	[0.45]
lulesh-comm.cc	127341	; i : i : MPI_Wait(0 1)	8.89us	[~]	[0.51]
lulesh-comm.cc	12791	; i : i : MPI_Wait(0 1)	8.78us	[~]	[2.04]
lulesh-comm.cc	128241	; i : i : MPI_Wait(0 1)	8.53us	[~]	[2.13]
lulesh.cc	127741	[- computation ~]	<b>1.91ms</b>	<b>[~]</b>	<b>[~]</b>

## Validation

Lulesh 2.0

## Execution:

- 6014 code lines (C++)
  - 128 parallel processes
  - $\approx 160MB$  tracefile

### **Analysis:**

- Filter all  $\delta < 10\%$
  - CPU burst time  $> 6ms$

### Results:

- 30 iterations loop
  - 5 long CPU burst
  - 1 Data conditioned MPI

## Validation

## Lulesh 2.0

## Execution:

- 6014 code lines (C++)
  - 128 parallel processes
  - $\approx 160MB$  tracefile

## Analysis:

- Filter all  $\delta < 10\%$
  - CPU burst time  $> 6ms$

## Results:

- 30 iterations loop
  - 5 long CPU burst
  - 1 Data conditioned MPI
  - 56 no conditioned MPI

# Validation

## Lulesh 2.0

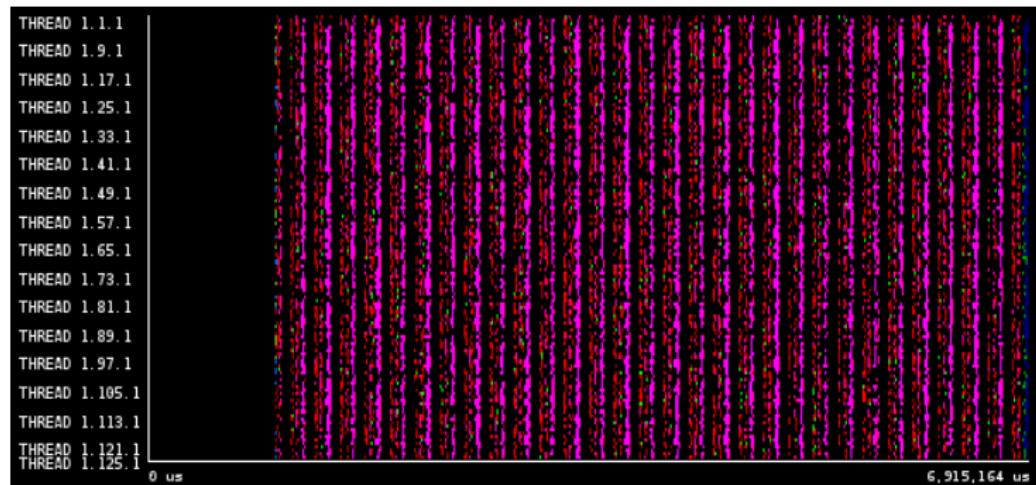


Figure: Lulesh 2.0 128 MPI ranks – 30 iterations

# Validation

## Lulesh 2.0

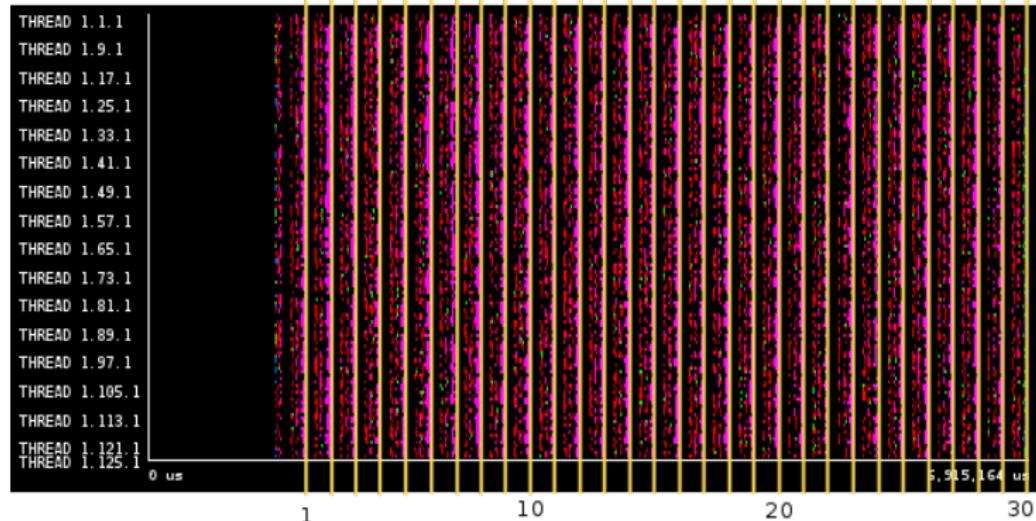
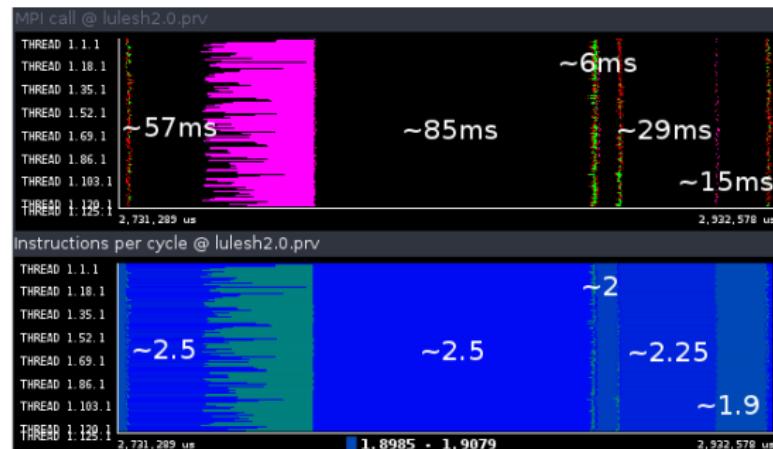


Figure: Lulesh 2.0 128 MPI ranks – 30 iterations

# Validation

## Lulesh 2.0

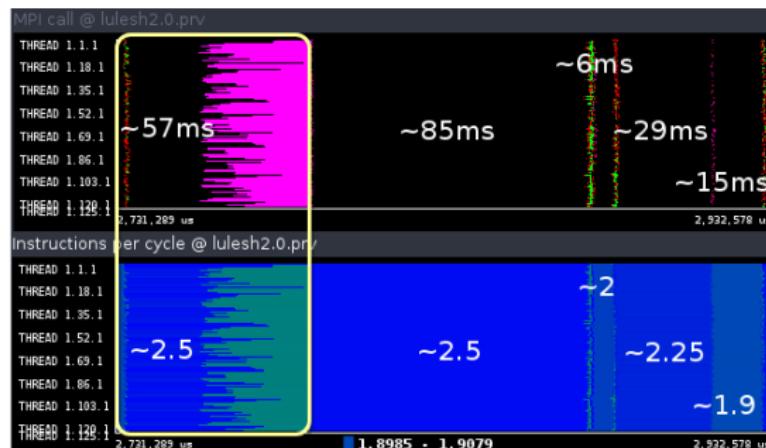
FILE	LINE	PSEUDOCODE	E(TIME)	E(SIZE)	E(IPC)
lulesh.cc	2773 : : TimeIncrement()	-	-	-	-
lulesh.cc	214 : : : [~ computation ~]	56.63ms	-	2.56	-
lulesh.cc	214 : : : 96.6% => MPI_Allreduce(CommId:1)	35.32us	8.0B/8.0B	1.57	-
lulesh.cc	1158 : : : : CommSend()	-	-	-	-
lulesh-comm.cc	401 : : : : : [~ computation ~]	85.2ms	-	2.48	-
lulesh-comm.cc	401 : : : : : MPI_Comm_rank(0:)	20.08us	-	1.59	-
lulesh.cc	1289 : : : : CommSend()	-	-	-	-
lulesh-comm.cc	401 : : : : : [~ computation ~]	6.15ms	-	2.0	-
lulesh-comm.cc	401 : : : : : MPI_Comm_rank(0:)	14.58us	-	1.63	-
lulesh.cc	1992 : : : : CommRecv()	-	-	-	-
lulesh-comm.cc	101 : : : : : [~ computation ~]	29.32ms	-	2.25	-
lulesh-comm.cc	101 : : : : : MPI_Comm_rank(0:)	18.76us	-	1.39	-
lulesh.cc	2010 : : : : CommSend()	-	-	-	-
lulesh-comm.cc	401 : : : : : [~ computation ~]	15.35ms	-	1.91	-
lulesh-comm.cc	401 : : : : : MPI_Comm_rank(0:)	15.24us	-	1.52	-



# Validation

## Lulesh 2.0

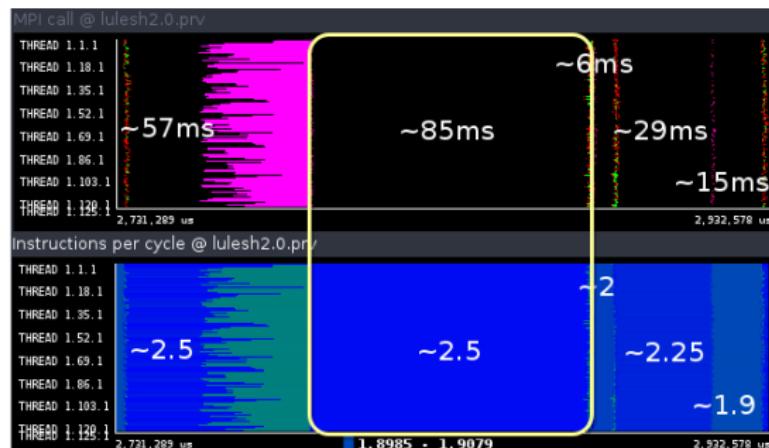
FILE	LINE	PSEUDOCODE	E(TIME)	E(SIZE)	E(IPC)
lulesh.cc	2773  : : TimeIncrement()	-	-	-	-
lulesh.cc	214  : : : [~ computation ~]	56.63ms	-	2.56	-
lulesh.cc	214  : : : 96.6% => MPI_Allreduce(CommId:1)	35.32us	8.0B/8.0B	1.57	-
lulesh.cc	1158  : : : : CommSend()	-	-	-	-
lulesh-comm.cc	401  : : : : : [~ computation ~]	85.2ms	-	2.48	-
lulesh-comm.cc	401  : : : : : MPI_Comm_rank(0:)	20.08us	-	1.59	-
lulesh.cc	1289  : : : : CommSend()	-	-	-	-
lulesh-comm.cc	401  : : : : : [~ computation ~]	6.15ms	-	2.0	-
lulesh-comm.cc	401  : : : : : MPI_Comm_rank(0:)	14.58us	-	1.63	-
lulesh.cc	1992  : : : : CommRecv()	-	-	-	-
lulesh-comm.cc	101  : : : : : [~ computation ~]	29.32ms	-	2.25	-
lulesh-comm.cc	101  : : : : : MPI_Comm_rank(0:)	18.76us	-	1.39	-
lulesh.cc	2010  : : : : CommSend()	-	-	-	-
lulesh-comm.cc	401  : : : : : [~ computation ~]	15.35ms	-	1.91	-
lulesh-comm.cc	401  : : : : : MPI_Comm_rank(0:)	15.24us	-	1.52	-



# Validation

## Lulesh 2.0

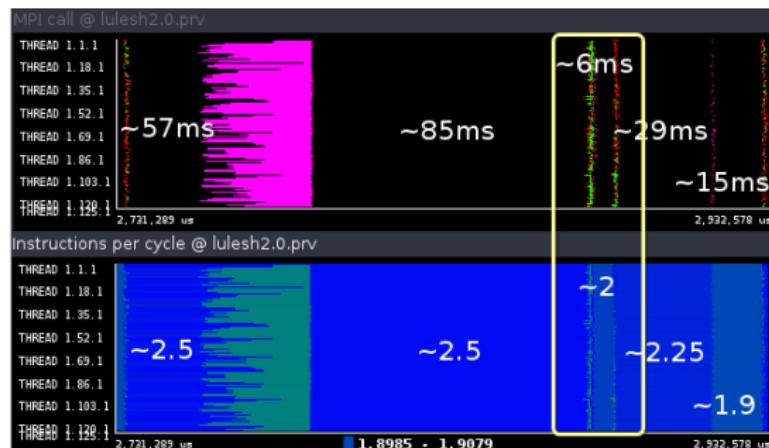
FILE	LINE	PSEUDOCODE	E(TIME)	E(SIZE)	E(IPC)
lulesh.cc	2773 : : TimeIncrement()	-	-	-	-
lulesh.cc	214 : : : [~ computation ~]	56.63ms	-	2.56	
lulesh.cc	214 : : : 96.6% => MPI_Allreduce(CommId:1)	35.32us	8.0B/8.0B	1.57	
lulesh.cc	1158 : : : : CommSend()	-	-	-	
lulesh-comm.cc	401 : : : : : [~ computation ~]	85.2ms	-	2.48	
lulesh-comm.cc	401 : : : : : MPI_Comm_rank(0:)	20.08us	-	1.59	
lulesh.cc	1289 : : : CommSend()	-	-	-	
lulesh-comm.cc	401 : : : : : [~ computation ~]	6.15ms	-	2.0	
lulesh-comm.cc	401 : : : : : MPI_Comm_rank(0:)	14.58us	-	1.63	
lulesh.cc	1992 : : : : CommRecv()	-	-	-	
lulesh-comm.cc	101 : : : : : [~ computation ~]	29.32ms	-	2.25	
lulesh-comm.cc	101 : : : : : MPI_Comm_rank(0:)	18.76us	-	1.39	
lulesh.cc	2010 : : : : CommSend()	-	-	-	
lulesh-comm.cc	401 : : : : : [~ computation ~]	15.35ms	-	1.91	
lulesh-comm.cc	401 : : : : : MPI_Comm_rank(0:)	15.24us	-	1.52	



# Validation

## Lulesh 2.0

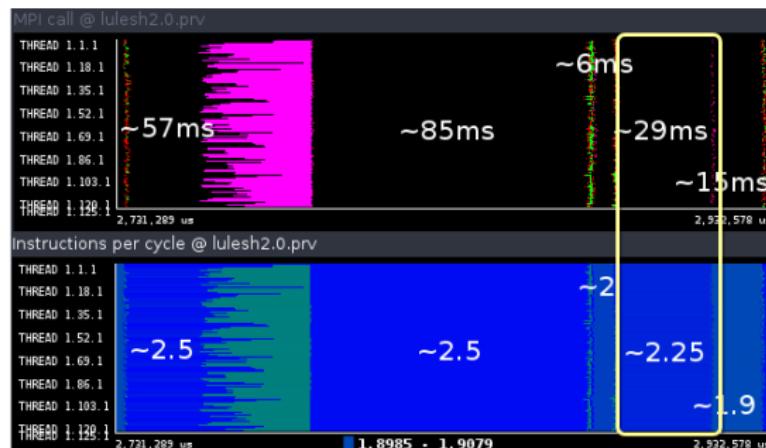
FILE	LINE	PSEUDOCODE	E(TIME)	E(SIZE)	E(IPC)
lulesh.cc	2773  : : TimeIncrement()	-	-	-	-
lulesh.cc	214  : : : [~ computation ~]	56.63ms	-	2.56	-
lulesh.cc	214  : : : 96.6% => MPI_Allreduce(CommId:1)	35.32us	8.0B/8.0B	1.57	-
lulesh.cc	1158  : : : : CommSend()	-	-	-	-
lulesh-comm.cc	401  : : : : : [~ computation ~]	85.2ms	-	2.48	-
lulesh-comm.cc	401  : : : : : MPI_Comm_rank(0:)	20.08us	-	1.59	-
lulesh.cc	1289  : : : : CommSend()	-	-	-	-
lulesh-comm.cc	401  : : : : : [~ computation ~]	6.15ms	-	2.0	-
lulesh-comm.cc	401  : : : : : MPI_Comm_rank(0:)	14.58us	-	1.63	-
lulesh.cc	1992  : : : : CommRecv()	-	-	-	-
lulesh-comm.cc	101  : : : : : [~ computation ~]	29.32ms	-	2.25	-
lulesh-comm.cc	101  : : : : : MPI_Comm_rank(0:)	18.76us	-	1.39	-
lulesh.cc	2010  : : : : CommSend()	-	-	-	-
lulesh-comm.cc	401  : : : : : [~ computation ~]	15.35ms	-	1.91	-
lulesh-comm.cc	401  : : : : : MPI_Comm_rank(0:)	15.24us	-	1.52	-



# Validation

## Lulesh 2.0

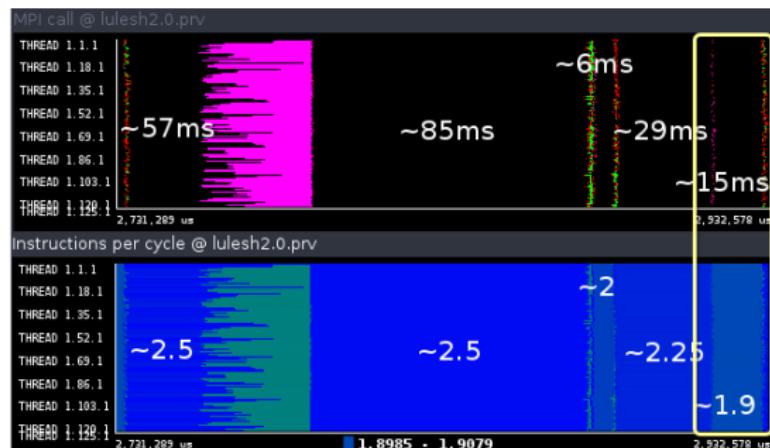
FILE	LINE	PSEUDOCODE	E(TIME)	E(SIZE)	E(IPC)
lulesh.cc	2773  : : TimeIncrement()	-	-	-	-
lulesh.cc	214  : : : [~ computation ~]	56.63ms	-	2.56	-
lulesh.cc	214  : : : 96.6% => MPI_Allreduce(CommId:1)	35.32us	8.0B/8.0B	1.57	-
lulesh.cc	1158  : : : : CommSend()	-	-	-	-
lulesh-comm.cc	401  : : : : : [~ computation ~]	85.2ms	-	2.48	-
lulesh-comm.cc	401  : : : : : MPI_Comm_rank(0:)	20.08us	-	1.59	-
lulesh.cc	1289  : : : CommSend()	-	-	-	-
lulesh-comm.cc	401  : : : : : [~ computation ~]	6.15ms	-	2.0	-
lulesh-comm.cc	401  : : : : : MPI_Comm_rank(0:)	14.58us	-	1.63	-
lulesh.cc	1992  : : : : CommRecv()	-	-	-	-
lulesh-comm.cc	101  : : : : : [~ computation ~]	29.32ms	-	2.25	-
lulesh-comm.cc	101  : : : : : MPI_Comm_rank(0:)	18.76us	-	1.39	-
lulesh.cc	2010  : : : : CommSend()	-	-	-	-
lulesh-comm.cc	401  : : : : : [~ computation ~]	15.35ms	-	1.91	-
lulesh-comm.cc	401  : : : : : MPI_Comm_rank(0:)	15.24us	-	1.52	-



# Validation

## Lulesh 2.0

FILE	LINE	PSEUDOCODE	E(TIME)	E(SIZE)	E(IPC)
lulesh.cc	2773  : : TimeIncrement()	-	-	-	-
lulesh.cc	214  : : : [~ computation ~]	56.63ms	-	2.56	-
lulesh.cc	214  : : : 96.6% => MPI_Allreduce(CommId:1)	35.32us	8.0B/8.0B	1.57	-
lulesh.cc	1158  : : : : CommSend()	-	-	-	-
lulesh-comm.cc	401  : : : : : [~ computation ~]	85.2ms	-	2.48	-
lulesh-comm.cc	401  : : : : : MPI_Comm_rank(0:)	20.08us	-	1.59	-
lulesh.cc	1289  : : : CommSend()	-	-	-	-
lulesh-comm.cc	401  : : : : : [~ computation ~]	6.15ms	-	2.0	-
lulesh-comm.cc	401  : : : : : MPI_Comm_rank(0:)	14.58us	-	1.63	-
lulesh.cc	1992  : : : CommRecv()	-	-	-	-
lulesh-comm.cc	101  : : : : : [~ computation ~]	29.32ms	-	2.25	-
lulesh-comm.cc	101  : : : : : MPI_Comm_rank(0:)	18.76us	-	1.39	-
lulesh.cc	2010  : : : CommSend()	-	-	-	-
lulesh-comm.cc	401  : : : : : [~ computation ~]	15.35ms	-	1.91	-
lulesh-comm.cc	401  : : : : : MPI_Comm_rank(0:)	15.24us	-	1.52	-



# Validation

## CG Class A

Execution:

- $\approx 2000$  code lines (Fortran)
- 32 parallel processes
- $\approx 150MB$  tracefile

Analysis:

- Filter all  $\delta < 10\%$
- CPU burst *time*  $> 100\mu s$

# Validation

## CG Class A

### Execution:

- $\approx 2000$  code lines (Fortran)
- 32 parallel processes
- $\approx 150MB$  tracefile

### Analysis:

- Filter all  $\delta < 10\%$
- CPU burst time  $> 100\mu s$

### Results:

- 15 iterations loop

	0 MAIN_()	-	-	-	-
cg.f   475 : FOR 1 TO 15	-	-	-	-	-
cg.f   475 : : config_grid()	-	-	-	-	-
cg.f   1089 : : : FOR 1 TO 3.0	-	-	-	-	-
cg.f   1096 : : : MPI_Irecv(0:1,2,4)	7.23us	8.0B	1.76		
cg.f   1096 : : : MPI_Send(0:1,2,4)	6.81us	8.0B	1.34		
cg.f   1097 : : : MPI_Wait(0:)	46.79us	-	1.32		
cg.f   1097 : : END LOOP	-	-	-		
cg.f   1097 : : : FOR 1 TO 25.0	-	-	-		
cg.f   1097 : : : : FOR 1 TO 3.0	-	-	-		
cg.f   1137 : : : : [- computation -]	115.51us	-	-	2.32	
cg.f   1137 : : : : MPI_Irecv(0:1,2,4)	7.31us	13.67KB	1.27		
cg.f   1144 : : : : MPI_Send(0:1,2,4)	10.52us	13.67KB	1.42		
cg.f   1145 : : : : MPI_Wait(0:)	12.07us	-	1.3		
cg.f   1145 : : : END LOOP	-	-	-		
cg.f   1163 : : : : MPI_Irecv(0:0)	6.7us	13.67KB	2.48		
cg.f   1171 : : : : MPI_Send(0:0)	7.04us	13.67KB	1.45		
cg.f   1172 : : : : MPI_Wait(0:)	6.45us	-	1.28		
cg.f   1172 : : : : FOR 1 TO 3.0	-	-	-		
cg.f   1207 : : : : MPI_Irecv(0:1,2,4)	6.55us	8.0B	1.83		
cg.f   1214 : : : : MPI_Send(0:1,2,4)	6.98us	8.0B	1.52		
cg.f   1216 : : : : MPI_Wait(0:)	13.15us	-	1.4		
cg.f   1216 : : : END LOOP	-	-	-		
cg.f   1216 : : : : FOR 1 TO 3.0	-	-	-		
cg.f   1262 : : : : MPI_Irecv(0:1,2,4)	-	8.0B	2.75		
cg.f   1269 : : : : MPI_Send(0:1,2,4)	-	8.0B	1.51		
cg.f   1270 : : : : MPI_Wait(0:)	-	-	1.39		
cg.f   1270 : : : END LOOP	-	-	-		
cg.f   1270 : : : : END LOOP	-	-	-		
cg.f   1270 : : : : FOR 1 TO 3.0	-	-	-		
cg.f   1320 : : : : [- computation -]	115.65us	-	-	2.31	
cg.f   1320 : : : : MPI_Irecv(0:1,2,4)	6.94us	13.67KB	1.13		
cg.f   1327 : : : : MPI_Send(0:1,2,4)	10.55us	13.67KB	1.43		
cg.f   1328 : : : : MPI_Wait(0:)	9.14us	-	1.25		
cg.f   1328 : : : : END LOOP	-	-	-		
cg.f   1347 : : : : MPI_Irecv(0:0)	6.76us	13.67KB	2.47		
cg.f   1355 : : : : MPI_Send(0:0)	7.13us	13.67KB	1.44		
cg.f   1356 : : : : MPI_Wait(0:)	6.59us	-	1.19		
cg.f   1356 : : : : FOR 1 TO 3.0	-	-	-		
cg.f   1384 : : : : MPI_Irecv(0:1,2,4)	6.57us	8.0B	2.37		
cg.f   1391 : : : : MPI_Send(0:1,2,4)	6.63us	8.0B	1.51		
cg.f   1392 : : : : MPI_Wait(0:)	25.38us	-	1.38		
cg.f   1392 : : : : END LOOP	-	-	-		
cg.f   1392 : : : : : FOR 1 TO 3.0	-	-	-		
cg.f   499 : : : : MPI_Irecv(0:1,2,4)	6.03us	16.0B	3.09		
cg.f   506 : : : : MPI_Send(0:1,2,4)	6.1us	16.0B	1.53		
cg.f   507 : : : : MPI_Wait(0:)	6.48us	-	1.35		
cg.f   507 : : : : END LOOP	-	-	-		
cg.f   507 : : : : : END LOOP	-	-	-		

15x

# Validation

## CG Class A

### Execution:

- $\approx 2000$  code lines (Fortran)
- 32 parallel processes
- $\approx 150MB$  tracefile

### Analysis:

- Filter all  $\delta < 10\%$
- CPU burst time  $> 100\mu s$

### Results:

- 15 iterations loop
- 25 iterations subloop

	0 MAIN_()	1 -	2 -	3 -	4 -	5 -	6 -	7 -	8 -	9 -	10 -	11 -	12 -	13 -	14 -	15 -	16 -	17 -	18 -	19 -	20 -	21 -	22 -	23 -	24 -	25 -	26 -	27 -	28 -	29 -	30 -	31 -	32 -	33 -	34 -	35 -	36 -	37 -	38 -	39 -	40 -	41 -	42 -	43 -	44 -	45 -	46 -	47 -	48 -	49 -	50 -	51 -	52 -	53 -	54 -	55 -	56 -	57 -	58 -	59 -	60 -	61 -	62 -	63 -	64 -	65 -	66 -	67 -	68 -	69 -	70 -	71 -	72 -	73 -	74 -	75 -	76 -	77 -	78 -	79 -	80 -	81 -	82 -	83 -	84 -	85 -	86 -	87 -	88 -	89 -	90 -	91 -	92 -	93 -	94 -	95 -	96 -	97 -	98 -	99 -	100 -
--	-----------	-----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------

# Validation

## CG Class A

### Execution:

- $\approx 2000$  code lines (Fortran)
- 32 parallel processes
- $\approx 150MB$  tracefile

### Analysis:

- Filter all  $\delta < 10\%$
- CPU burst time  $> 100\mu s$

### Results:

- 15 iterations loop
- 25 iterations subloop
- Other 3 iterations subloops

	0 MAIN_()	-	-	-	-
cg.f	475 : FOR 1 TO 15	-	-	-	-
cg.f	475 : : : : : config_grid()	-	-	-	-
cg.f	1089 : : : : : MPI_Irecv(0:1,2,4)	7.23us	8.0B	1.76	
cg.E	1096 : : : : : MPI_Send(0:1,2,4)	6.81us	8.0B	1.34	
cg.F	1097 : : : : : MPI_Wait(0:)	16.79us	-	1.32	
cg.F	1100 : : : : : END LOOP	-	-	-	
cg.F	1101 : : : : : FOR 1 TO 25,0	-	-	-	
cg.F	1101 : : : : : : : : FOR 1 TO 3,0	-	-	-	
cg.F	1101 : : : : : : : : : [- computation -]	-	-	-	
cg.F	1137 : : : : : : : : : MPI_Irecv(0:1,2,4)	13.67KB	-	2.32	
cg.F	1137 : : : : : : : : : MPI_Send(0:1,2,4)	13.67KB	-	1.27	
cg.F	1144 : : : : : : : : : MPI_Wait(0:)	-	-	1.42	
cg.F	1145 : : : : : : : : : END LOOP	-	-	1.3	
cg.F	1163 : : : : : : : : : MPI_Irecv(0:0)	16.7us	13.67KB	2.48	
cg.E	1171 : : : : : : : : : MPI_Send(0:0)	17.04us	13.67KB	1.45	
cg.F	1172 : : : : : : : : : MPI_Wait(0:)	6.45us	-	1.28	
cg.F	1172 : : : : : : : : : FOR 1 TO 3,0	-	-	-	
cg.F	1207 : : : : : : : : : MPI_Irecv(0:1,2,4)	8.0B	-	1.83	
cg.E	1214 : : : : : : : : : MPI_Send(0:1,2,4)	8.0B	-	1.52	
cg.F	1216 : : : : : : : : : MPI_Wait(0:)	-	-	1.4	
cg.F	1216 : : : : : : : : : END LOOP	-	-	-	
cg.F	1217 : : : : : : : : : FOR 1 TO 3,0	-	-	-	
cg.F	1262 : : : : : : : : : MPI_Irecv(0:1,2,4)	8.0B	-	2.75	
cg.E	1269 : : : : : : : : : MPI_Send(0:1,2,4)	8.0B	-	1.51	
cg.F	1270 : : : : : : : : : MPI_Wait(0:)	-	-	1.39	
cg.F	1270 : : : : : : : : : END LOOP	-	-	-	
cg.F	1271 : : : : : : : : : FOR 1 TO 3,0	-	-	-	
cg.F	1320 : : : : : : : : : [- computation -]	115.65us	-	2.31	
cg.F	1320 : : : : : : : : : MPI_Irecv(0:1,2,4)	6.94us	13.67KB	1.13	
cg.F	1327 : : : : : : : : : MPI_Send(0:1,2,4)	10.55us	13.67KB	1.43	
cg.F	1328 : : : : : : : : : MPI_Wait(0:)	9.14us	-	1.25	
cg.F	1328 : : : : : : : : : END LOOP	-	-	-	
cg.F	1347 : : : : : : : : : MPI_Irecv(0:0)	6.76us	13.67KB	2.47	
cg.E	1355 : : : : : : : : : MPI_Send(0:0)	7.13us	13.67KB	1.44	
cg.F	1356 : : : : : : : : : MPI_Wait(0:)	6.59us	-	1.19	
cg.F	1356 : : : : : : : : : FOR 1 TO 3,0	-	-	-	
cg.F	1384 : : : : : : : : : MPI_Irecv(0:1,2,4)	6.57us	8.0B	2.37	
cg.E	1391 : : : : : : : : : MPI_Send(0:1,2,4)	6.63us	8.0B	1.51	
cg.F	1392 : : : : : : : : : MPI_Wait(0:)	25.38us	-	1.38	
cg.F	1392 : : : : : : : : : END LOOP	-	-	-	
cg.F	1392 : : : : : : : : : FOR 1 TO 3,0	-	-	-	
cg.F	1499 : : : : : : : : : MPI_Irecv(0:1,2,4)	6.03us	16.0B	3.09	
cg.E	506 : : : : : : : : : MPI_Send(0:1,2,4)	6.1us	16.0B	1.53	
cg.F	507 : : : : : : : : : MPI_Wait(0:)	6.48us	-	1.35	
cg.F	507 : : : : : : : : : END LOOP	-	-	-	
cg.F	507 : : : : : : : : : END LOOP	-	-	-	

# Validation

## CG Class A

### Execution:

- $\approx 2000$  code lines (Fortran)
- 32 parallel processes
- $\approx 150MB$  tracefile

### Analysis:

- Filter all  $\delta < 10\%$
- CPU burst time  $> 100\mu s$

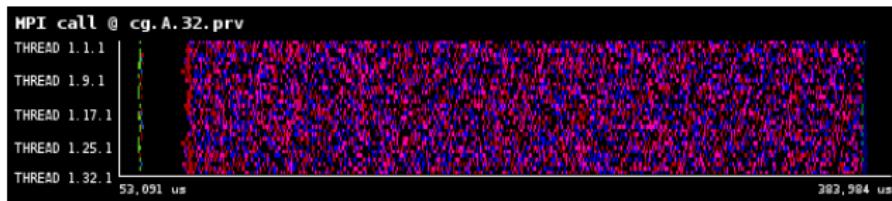
### Results:

- 15 iterations loop
- 25 iterations subloop
- Other 3 iterations subloops
- 2 long CPU burst above 100 $\mu s$

	0 MAIN_()	-	-	-	-
cg.f	475 : FOR 1 TO 15	-	-	-	-
cg.f	475 : : : : : conjgrad()	-	-	-	-
cg.f	475 : : : : : FOR 1 TO 15	-	-	-	-
cg.f	1089 : : : : : MPI_Irecv(0:1,2,4)	7.23us	8.0B	1.76	
cg.f	1096 : : : : : MPI_Send(0:1,2,4)	6.81us	8.0B	1.34	
cg.f	1097 : : : : : MPI_Wait(0:)	146.79us	-	1.32	
cg.f	1104 : : : : : END LOOP	-	-	-	
cg.f	1104 : : : : : FOR 1 TO 25,0	-	-	-	-
cg.f	1104 : : : : : : : : FOR 1 TO 3,0	-	-	-	-
cg.f	1137 : : : : : [- computation -]	115.51us	-	2.32	
cg.f	1137 : : : : : MPI_Irecv(0:1,2,4)	7.31us	13.67KB	1.27	
cg.f	1144 : : : : : MPI_Send(0:1,2,4)	10.52us	13.67KB	1.42	
cg.f	1145 : : : : : MPI_Wait(0:)	12.07us	-	1.3	
cg.f	1145 : : : : : END LOOP	-	-	-	
cg.f	1163 : : : : : MPI_Irecv(0:0)	6.7us	13.67KB	2.48	
cg.f	1171 : : : : : MPI_Send(0:0)	7.04us	13.67KB	1.45	
cg.f	1172 : : : : : MPI_Wait(0:)	6.45us	-	1.28	
cg.f	1172 : : : : : FOR 1 TO 3,0	-	-	-	-
cg.f	1207 : : : : : MPI_Irecv(0:1,2,4)	6.55us	8.0B	1.83	
cg.f	1214 : : : : : MPI_Send(0:1,2,4)	6.98us	8.0B	1.52	
cg.f	1216 : : : : : MPI_Wait(0:)	13.15us	-	1.4	
cg.f	1216 : : : : : END LOOP	-	-	-	
cg.f	1216 : : : : : FOR 1 TO 3,0	-	-	-	-
cg.f	1262 : : : : : MPI_Irecv(0:1,2,4)	6.58us	8.0B	2.75	
cg.f	1269 : : : : : MPI_Send(0:1,2,4)	6.62us	8.0B	1.51	
cg.f	1270 : : : : : MPI_Wait(0:)	9.8us	-	1.39	
cg.f	1270 : : : : : END LOOP	-	-	-	
cg.f	1270 : : : : : END LOOP	-	-	-	-
cg.f	1270 : : : : : FOR 1 TO 3,0	-	-	-	-
cg.f	1320 : : : : : [- computation -]	115.65us	-	2.31	
cg.f	1320 : : : : : MPI_Irecv(0:1,2,4)	6.94us	13.67KB	1.13	
cg.f	1327 : : : : : MPI_Send(0:1,2,4)	10.55us	13.67KB	1.43	
cg.f	1328 : : : : : MPI_Wait(0:)	9.14us	-	1.25	
cg.f	1328 : : : : : END LOOP	-	-	-	
cg.f	1347 : : : : : MPI_Irecv(0:0)	6.76us	13.67KB	2.47	
cg.f	1355 : : : : : MPI_Send(0:0)	7.13us	13.67KB	1.44	
cg.f	1356 : : : : : MPI_Wait(0:)	6.59us	-	1.19	
cg.f	1356 : : : : : FOR 1 TO 3,0	-	-	-	-
cg.f	1384 : : : : : MPI_Irecv(0:1,2,4)	6.57us	8.0B	2.37	
cg.f	1391 : : : : : MPI_Send(0:1,2,4)	6.63us	8.0B	1.51	
cg.f	1392 : : : : : MPI_Wait(0:)	25.38us	-	1.38	
cg.f	1392 : : : : : END LOOP	-	-	-	
cg.f	1392 : : : : : FOR 1 TO 3,0	-	-	-	-
cg.f	1499 : : : : : MPI_Irecv(0:1,2,4)	6.03us	16.0B	3.09	
cg.f	1506 : : : : : MPI_Send(0:1,2,4)	6.1us	16.0B	1.53	
cg.f	1507 : : : : : MPI_Wait(0:)	6.48us	-	1.35	
cg.f	1507 : : : : : END LOOP	-	-	-	
cg.f	1507 : : : : : END LOOP	-	-	-	-

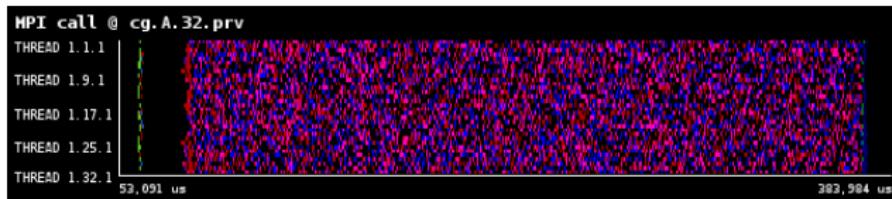
# Validation

## CG Class A

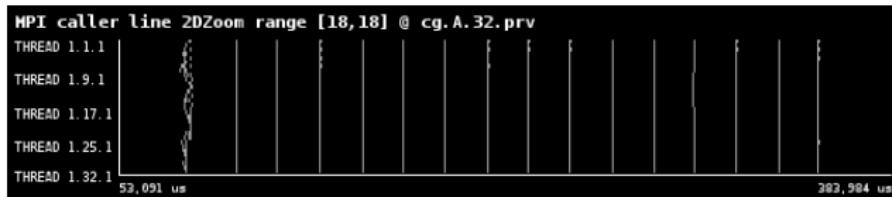


# Validation

## CG Class A

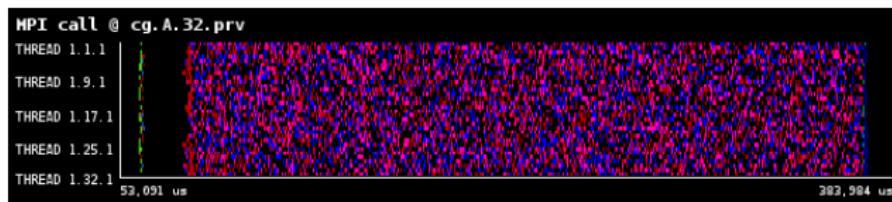


Showing just **MPI\_Irecv** on **cg.f** : 1089

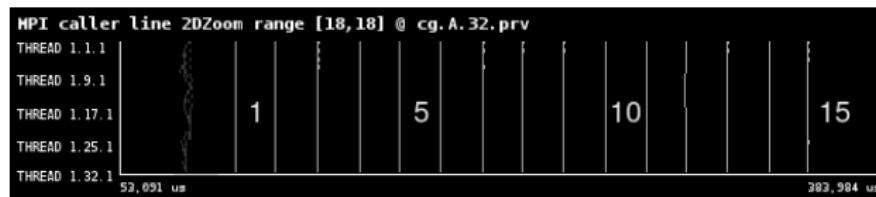


## Validation

CG Class A

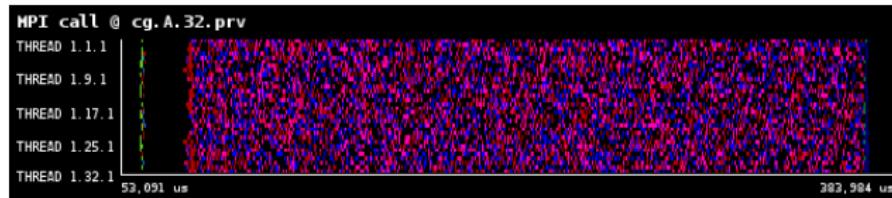


Showing just MPI\_Recv on cg.f : 1089

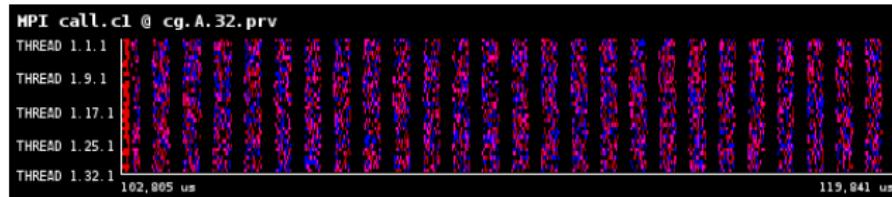
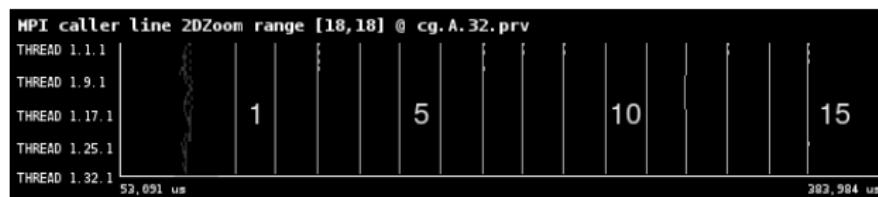


## Validation

CG Class A

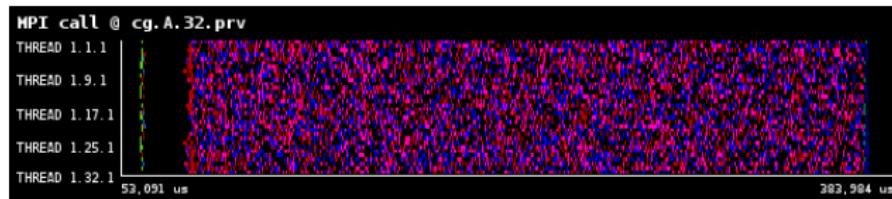


Showing just MPI\_Irecv on cg.f : 1089

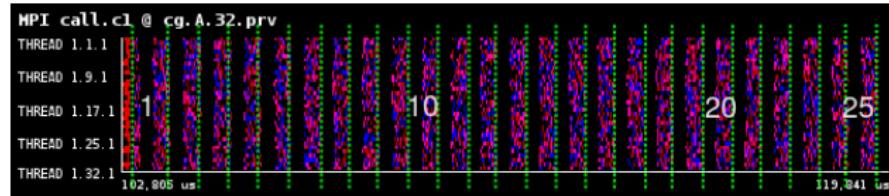


## Validation

CG Class A



Showing just MPI\_Irecv on cg.f : 1089



# Outline for section 9

## 1 Context

- Performance Analysis

## 2 Motivations

## 3 Objectives

## 4 State of the Art

## 5 Proposal

- Application structure by classification

## 6 Implementation

- Workflow

## 7 Best features analysis

## 8 Validation

- Lulesh 2.0
- CG

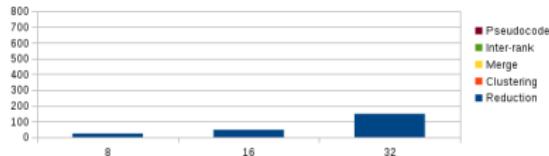
## 9 Scalability

## 10 Conclusions

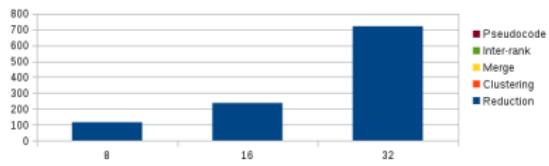
## 11 Future work

# Scalability

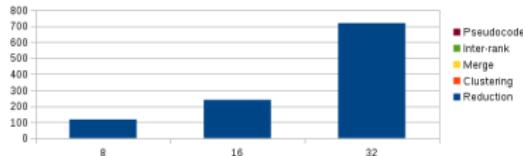
CG (A) phases breakdown



CG (B) phases breakdown

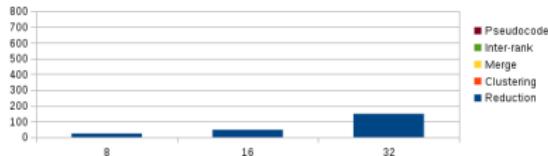


CG (C) phases breakdown

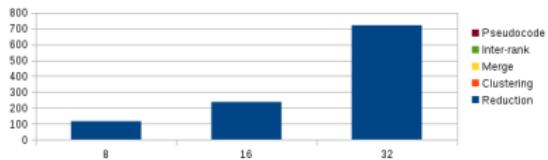


# Scalability

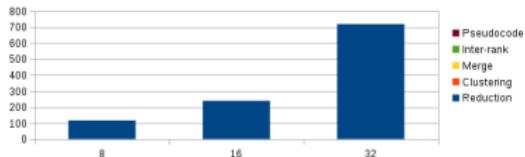
CG (A) phases breakdown



CG (B) phases breakdown



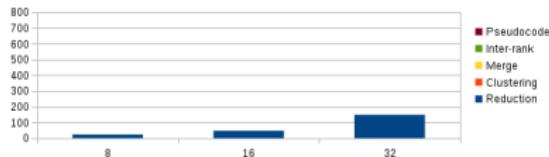
CG (C) phases breakdown



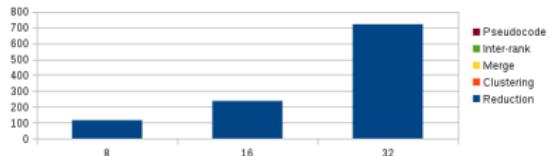
Reduction step is clearly dominating the execution time.

# Scalability

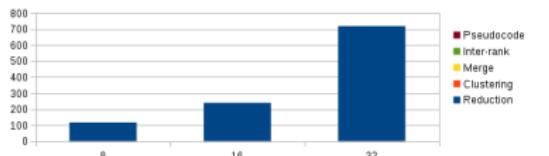
CG (A) phases breakdown



CG (B) phases breakdown

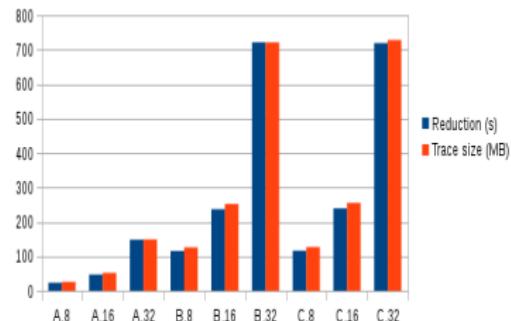


CG (C) phases breakdown



Reduction step is clearly dominating the execution time.

CG Reduction time and Trace size



Reduction step time is highly correlated with trace size.

# Outline for section 10

## 1 Context

- Performance Analysis

## 2 Motivations

## 3 Objectives

## 4 State of the Art

## 5 Proposal

- Application structure by classification

## 6 Implementation

- Workflow

## 7 Best features analysis

## 8 Validation

- Lulesh 2.0
- CG

## 9 Scalability

## 10 Conclusions

## 11 Future work

## Conclusions

- New approach for application structure detection has been explored.

## Conclusions

- New approach for application structure detection has been explored.
- Have demonstrated that it works well with HPC applications.

## Conclusions

- New approach for application structure detection has been explored.
- Have demonstrated that it works well with HPC applications.
- Number of iterations and iterations mean time has been identified as valuable features for loops classification.

## Conclusions

- New approach for application structure detection has been explored.
- Have demonstrated that it works well with HPC applications.
- Number of iterations and iterations mean time has been identified as valuable features for loops classification.
- Execution time is clearly dominated by Reduction step that presents linear complexity with the trace size.

# Outline for section 11

- 1 Context
  - Performance Analysis
- 2 Motivations
- 3 Objectives
- 4 State of the Art
- 5 Proposal
  - Application structure by classification
- 6 Implementation
  - Workflow
- 7 Best features analysis
- 8 Validation
  - Lulesh 2.0
  - CG
- 9 Scalability
- 10 Conclusions
- 11 Future work

## Future work

- Improve memory usage by timestamps compression.

## Future work

- Improve memory usage by timestamps compression.
- More research for select the best features set for loops classification that avoids cluster aliasing/split.

## Future work

- Improve memory usage by timestamps compression.
- More research for select the best features set for loops classification that avoids cluster aliasing/split.
- Improve Reduce phase times by parallelizing the reduction using for example well-known infrastructures like Hadoop.

## Future work

- Improve memory usage by timestamps compression.
- More research for select the best features set for loops classification that avoids cluster aliasing/split.
- Improve Reduce phase times by parallelizing the reduction using for example well-known infrastructures like Hadoop.
- Explore the possibility to use sampling techniques to get more detailed structure of an application.

*Master thesis*  
Master in Research and Innovation

**Inferring programs structure from  
an execution trace**

*Author*

Juan Francisco Martínez Vera

*Supervisor*

Jesús Labarta Mancho

Facultat d'Informàtica de Barcelona (FIB)  
Universitat Politècnica de Catalunya (UPC)



## Refinement

Even if it **works pretty well** in general.

Some problems **have been identified**.

- ① **Cluster aliasing** when two different loops behaves similarly enough over our defined space
- ② **Hidden superloop** when not detected superloop prevents from a good structure detection.
- ③ **Cluster split** when MPI calls belonging to the same loop behaves in a different way.

All three **have been solved by looking to timestamps**.

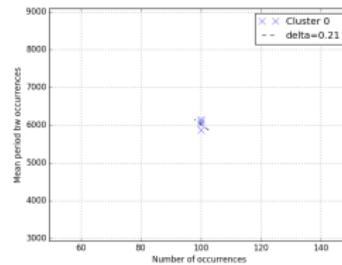
# Proposal modifications

## Cluster aliasing

### Keynote

Aliasing can be detected and solved **looking at timestamps**.

```
for 1to100
    do {MPI_A()
        MPI_B()}
for 1to100
    do {MPI_C()
        MPI_D()}
```



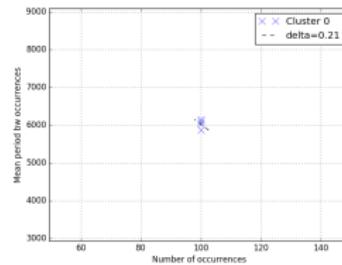
# Proposal modifications

## Cluster aliasing

### Keynote

Aliasing can be detected and solved **looking at timestamps**.

```
for 1to100
    do {MPI_A()
        MPI_B()}
for 1to100
    do {MPI_C()
        MPI_D()}
```



MPI_A	1	3
MPI_B	2	4
MPI_C	5	7
MPI_D	6	8

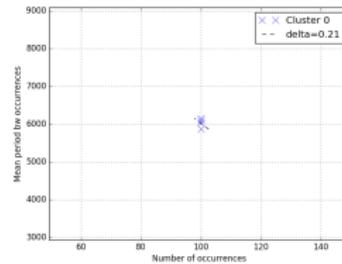
# Proposal modifications

## Cluster aliasing

### Keynote

Aliasing can be detected and solved **looking at timestamps**.

```
for 1to100
    do {MPI_A()
        MPI_B()}
for 1to100
    do {MPI_C()
        MPI_D()}
```



MPI_A	1	3
MPI_B	2	4
MPI_C	5	7
MPI_D	6	8

MPI_A	1	3
MPI_B	2	4
MPI_C		5 7
MPI_D		6 8

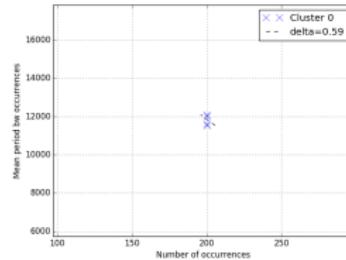
# Proposal modifications

## Hidden superloop

### Keynote

Similarly than with aliasing, **looking at timestamps**.

```
for 1to100
    do { for 1to2
        do { MPI_A()
            for 1to2
                do { MPI_B()
```



# Proposal modifications

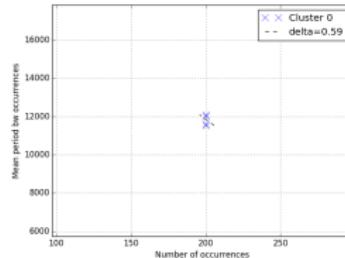
## Hidden superloop

### Keynote

Similarly than with aliasing, **looking at timestamps**.

```
for 1to100
    do { for 1to2
        do { MPI_A()
            for 1to2
                do { MPI_B()
```

MPI_A	1	2	5	6
MPI_B	3	4	7	8



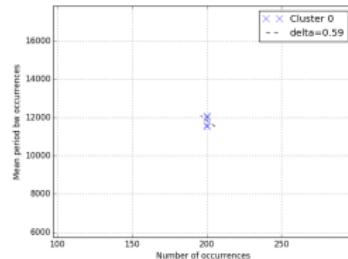
# Proposal modifications

## Hidden superloop

### Keynote

Similarly than with aliasing, **looking at timestamps**.

```
for 1to100
    do {for 1to2
        do {MPI_A()
            for 1to2
                do {MPI_B()}
```



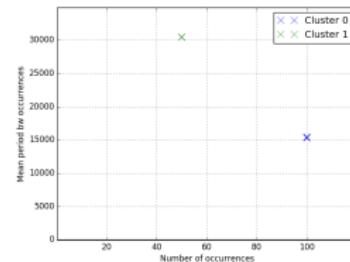
MPI_A	1	2	5	6
MPI_B	3	4	7	8

	MPI_A	1	2	5	6
MPI_B		3	4	7	8

# Proposal modifications

## Cluster split

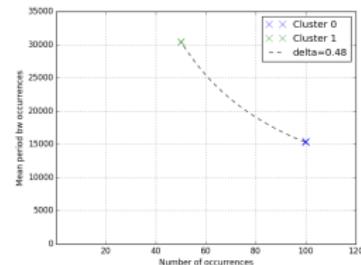
```
for i = 1 to 10
    do { MPI_B()
          if isPair(i)
              then MPI_A()
          MPI_B()}
```



# Proposal modifications

## Cluster split

```
for i = 1 to 10
    do { MPI_B()
          if isPair(i)
              then MPI_A()
          MPI_B()}
```



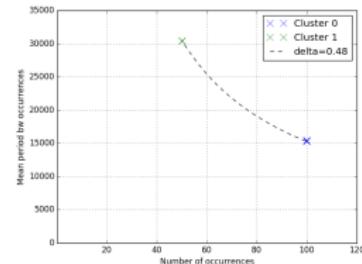
Whether MPI call is conditioned

- #repetitions and mean time bw. reps. change (in same proportion).

# Proposal modifications

## Cluster split

```
for i = 1 to 10
    do {MPI_B()
        if isPair(i)
            then MPI_A()
        MPI_B()}
```



Whether MPI call is conditioned

- #repetitions and mean time bw. reps. change (in same proportion).

By “reverse loop merging” and times checking

FILE	LINE	PSEUDOCODE	E(TIME)	E(SIZE)	E(IPC)
test-9.c	0	main()	-	-	-
	: FOR 1 TO 100		-	-	-
	: : IF rank in [1]		-	-	-
	20	: : : MPI_Send(1:0)	5.01us	4.0B	1.1
	: : : IF rank in [0]		-	-	-
	22	: : : MPI_Recv()	5.3us	-	1.03
test-9.c	24	: 50.0% => MPI_Barrier(CommId:1)	5.99us	-	0.97
	: : IF rank in [1]		-	-	-
	26	: : : MPI_Send(1:0)	4.9us	4.0B	1.14
	: : : IF rank in [0]		-	-	-
	28	: : : MPI_Recv()	5.23us	-	1.1
	: END LOOP		-	-	-

# Data analysis

## Data acquisition

Desired data...

- PCA<sup>2</sup>: Set of observations with set of features, i.e. Set of loops with **iteration-level aggregated features**.
- Variable Importance Same data as in production traces but labeled, i.e. textbf{With} information to what loop every MPI call belongs to.

How to obtain it

- ① The needed information is not in trace so ...
- ② **Manually inject monitors** to the source code
  - Very tough and error prone for huge source codes
- ③ Alternatively use source-to-source compiler to **do it automatically**
  - Mercurium

---

<sup>2</sup>Principal component analysis

# Data analysis

## Data acquisition

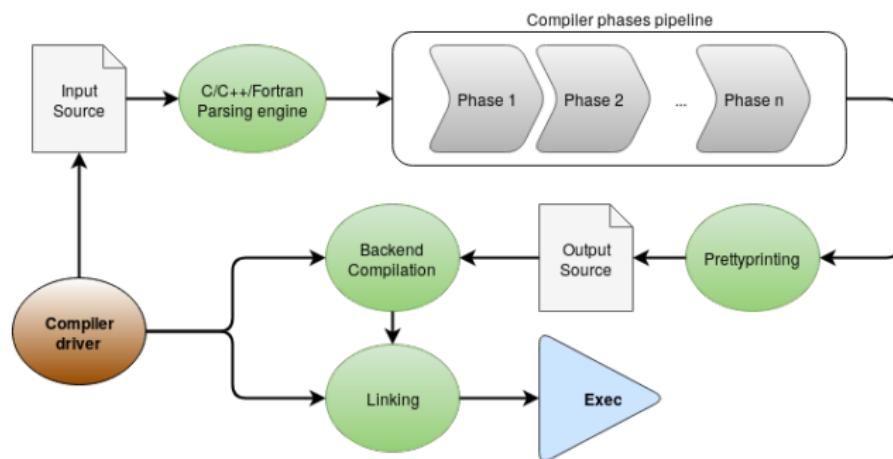


Figure: Mercurium internals overview

# Data analysis

## Data acquisition

### Algorithm 11.1: PCA()

*MLoopInit(loop<sub>line</sub>, loop<sub>file</sub>)*

**for**  $i \in I$

**do** {  $MIterInit(chance)$   
...  
 $MIterFini()$

*MLoopFini()*

- MLoopInit and MLoopFini fire loops boundaries
- MIterInit and MiterFini fire iteration boundaries with hwc information o trace

# Data analysis

## Data acquisition

### Algorithm 11.3: PCA()

```
MLoopInit(loopline, loopfile)
for i ∈ I
    do { MIterInit(chance)
          ...
          MIterFini()
      }
MLoopFini()
```

### Algorithm 11.4: VARIMP()

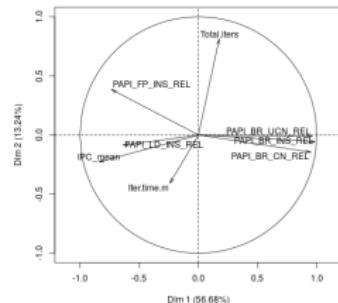
```
MLoopInit(loopline, loopfile)
for i ∈ I
    do { ...
          MBefCall()
          MPI_Call()
          MPIAftCall()
          ...
      }
MLoopFini()
```

- MLoopInit and MLoopFini fire loops boundaries
- MIterInit and MiterFini fire iteration boundaries with hwc information o trace

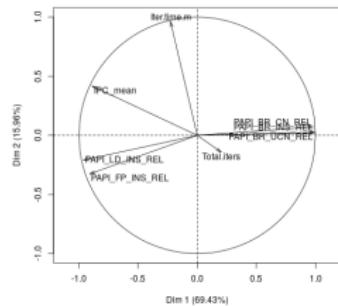
- MLoopInit and MLoopFini keep track entry/exit loops
- MPIBefCall and MPIAftCall fire loop id

# Data analysis

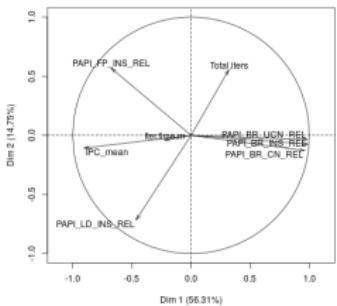
## Analysis of results: PCA



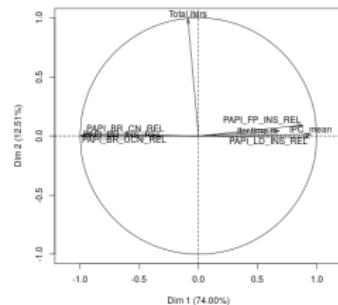
BT



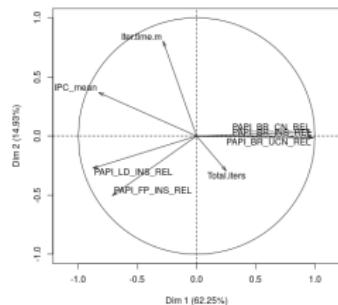
MG



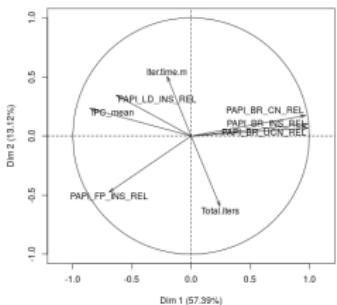
SP



CG



FT



LU

# Validation

## Lulesh 2.0

File	Line	Function	Time	Size	Count
lulesh.cc	214	: : : [~ computation ~]	<b>56.63ms</b>	-	<b>12.56</b>
lulesh.cc	214	: : : 96.6% -> MPI_Allreduce(CommId:1)	35.32us	8.0B/8.0B	1.57
lulesh.cc	2774	: : : LagrangeLeapFrog()	-	-	-
lulesh.cc	2648	: : : LagrangeNodal()	-	-	-
lulesh.cc	1263	: : : CalcForceForNodes()	-	-	-
lulesh.cc	1137	: : : CommRecv()	-	-	-
lulesh-comm.cc	101	: : : MPI_Comm_rank(0:)	10.27us	-	10.28
lulesh-comm.cc	119	: : : MPI_Irecv(0:25)	10.51us	22.52KB	10.55
lulesh-comm.cc	137	: : : MPI_Irecv(0:5)	8.67us	22.52KB	10.96
lulesh-comm.cc	155	: : : MPI_Irecv(0:1)	8.43us	22.52KB	11.08
lulesh-comm.cc	192	: : : MPI_Irecv(0:6)	8.8us	744.0B	11.08
lulesh-comm.cc	201	: : : MPI_Irecv(0:30)	8.54us	744.0B	11.25
lulesh-comm.cc	210	: : : MPI_Irecv(0:26)	8.29us	744.0B	11.25
lulesh-comm.cc	345	: : : MPI_Irecv(0:31)	8.32us	24.0B	0.9
lulesh.cc	1158	: : : CommSend()	-	-	-
lulesh-comm.cc	401	: : : : [~ computation ~]	<b>85.2ms</b>	-	<b>12.48</b>
lulesh-comm.cc	401	: : : : MPI_Comm_rank(0:)	120.08us	-	11.59

Table: Total MPI calls

	MPI_Isend	MPI_Irecv	MPI_Wait	MPI_Waitall	MPI_Allreduce	MPI_Comm_rank
Rank 0	300	510	510	90	29	270

Table: MPI calls counts by communication phases

	MPI_Isend	MPI_Irecv	MPI_Wait	MPI_Waitall	MPI_Allreduce	MPI_Comm_rank
Comm phase 1	0	7	0	0	1	1
Comm phase 2	7	7	1	0	0	3
Comm phase 3	0	0	7	1	0	2
Comm phase 4	0	3	0	0	0	1
Comm phase 5	3	0	3	1	0	2

# Validation

## Lulesh 2.0

Lulesh-conn.cc						
440	: : :	[- computation -]	~	85.2ns	~	12.48
403	: : :	MPI_Isend(0:1)	~	12.5us	~	11.9
403	: : :	MPI_Irecv(0:1)	~	12.5us	~	11.9
436	: : :	MPI_Isend(0:25)	14.95us	22.52KB	1.9	
477	: : :	MPI_Isend(0:51)	10.07us	22.52KB	1.65	
232	: : :	MPI_Isend(0:1)	12.5us	22.52KB	1.23	
589	: : :	MPI_Isend(0:6)	14.79us	1744.0B	1.68	
606	: : :	MPI_Isend(0:30)	12.45us	1744.0B	1.86	
623	: : :	MPI_Isend(0:1)	12.5us	1744.0B	1.56	
837	: : :	MPI_Isend(0:31)	9.48us	124.0B	0.84	
843	: : :	MPI_Waitall(0:1)	541.3us	~	0.43	
132	: : :	MPI_Wait(0:1)	~	~	~	
889	: : :	MPI_Comm_rank(0:1)	9.32us	~	0.49	
913	: : :	MPI_Wait(0:1)	9.32us	~	0.6	
940	: : :	MPI_Wait(0:1)	9.32us	~	0.58	
980	: : :	MPI_Wait(0:1)	9.32us	~	2.35	
1039	: : :	MPI_Wait(0:1)	8.41us	~	11.73	
1204	: : :	MPI_Wait(0:1)	9.32us	~	1.44	
1307	: : :	MPI_Wait(0:1)	9.32us	~	1.98	
1325	: : :	MPI_Wait(0:1)	58.99us	~	11.44	
1325	: : :	MPI_Wait(0:1)	~	~	~	
101	: : :	MPI_Comm_rank(0:1)	17.77us	~	0.73	
119	: : :	MPI_Irecv(0:25)	19.53us	45.05KB	0.79	
120	: : :	MPI_Irecv(0:25)	17.77us	45.05KB	1.12	
155	: : :	MPI_Irecv(0:11)	17.74us	45.05KB	0.95	
192	: : :	MPI_Irecv(0:6)	17.6us	11.45KB	11.12	
200	: : :	MPI_Irecv(0:25)	17.74us	11.45KB	1.13	
210	: : :	MPI_Irecv(0:25)	17.42us	11.45KB	1.13	
345	: : :	MPI_Irecv(0:31)	17.50us	148.0B	0.95	
345	: : :	MPI_Irecv(0:31)	~	~	~	
403	: : :	[- computation -]	16.15us	~	2.0	
403	: : :	MPI_Comm_rank(0:1)	14.58us	~	1.63	

Table: Total MPI calls

	MPI_Isend	MPI_Irecv	MPI_Wait	MPI_Waitall	MPI_Allreduce	MPI_Comm_rank
Rank 0	300	510	510	90	29	270

Table: MPI calls counts by communication phases

	MPI_Isend	MPI_Irecv	MPI_Wait	MPI_Waitall	MPI_Allreduce	MPI_Comm_rank
Comm phase 1	0	7	0	0	1	1
Comm phase 2	2	7	7	1	0	3
Comm phase 3	0	0	7	1	0	2
Comm phase 4	0	3	0	0	0	1
Comm phase 5	3	0	3	1	0	2

# Validation

## Lulesh 2.0

lulesh-comm.cc	401  : : : : [~ computation ~]	6.15ms	-	2.0
lulesh-comm.cc	401  : : : : MPI_Comm_rank(0:)	14.58us	-	1.63
lulesh-comm.cc	843  : : : : MPI_Waitall(0:)	10.26us	-	0.35
lulesh.cc	1292  : : : CommSyncPosVel()	-	-	-
lulesh-comm.cc	1310  : : : : MPI_Comm_rank(0:)	22.14us	-	0.73
lulesh-comm.cc	1332  : : : : MPI_Wait(0:)	274.73us	-	0.79
lulesh-comm.cc	1366  : : : : MPI_Wait(0:)	72.95us	-	2.46
lulesh-comm.cc	1402  : : : : MPI_Wait(0:)	23.24us	-	2.3
lulesh-comm.cc	1461  : : : : MPI_Wait(0:)	198.36us	-	1.79
lulesh-comm.cc	1475  : : : : MPI_Wait(0:)	7.83us	-	1.83
lulesh-comm.cc	1489  : : : : MPI_Wait(0:)	7.75us	-	2.08
lulesh-comm.cc	1674  : : : : MPI_Wait(0:)	333.07us	-	1.73
lulesh.cc	2656  : : : LagrangeElements()	-	-	-
lulesh.cc	2461  : : : CalcQForElems()	-	-	-
lulesh.cc	1992  : : : CommRecv()	-	-	-
lulesh-comm.cc	101  : : : : [~ computation ~]	29.32ms	-	2.25
lulesh-comm.cc	101  : : : : MPI_Comm_rank(0:)	140.76us	-	2.25

Table: Total MPI calls

	MPI_Isend	MPI_Irecv	MPI_Wait	MPI_Waitall	MPI_Allreduce	MPI_Comm_rank
Rank 0	300	510	510	90	29	270

Table: MPI calls counts by communication phases

	MPI_Isend	MPI_Irecv	MPI_Wait	MPI_Waitall	MPI_Allreduce	MPI_Comm_rank
Comm phase 1	0	7	0	0	1	1
Comm phase 2	2	7	7	1	0	3
Comm phase 3	0	0	7	1	0	2
Comm phase 4	0	3	0	0	0	1
Comm phase 5	3	0	3	1	0	2

# Validation

## Lulesh 2.0

lulesh-comm.cc	101 : : : : : [~ computation ~]	29.32ms	-	2.25				
lulesh-comm.cc	101 : : : : : MPI_Comm_rank(0:)	18.76us	-	1.39				
lulesh-comm.cc	119 : : : : : MPI_Irecv(0:25)	12.61us	21.09KB	0.43				
lulesh-comm.cc	137 : : : : : MPI_Irecv(0:5)	9.23us	21.09KB	1.01				
lulesh-comm.cc	155 : : : : : MPI_Irecv(0:1)	8.79us	21.09KB	1.17				
lulesh.cc	2010 : : : : : CommSend()	-	-	-				
lulesh-comm.cc	401 : : : : : [- computation ~]	15.35ms	-	1.91				
lulesh-comm.cc	401 : : : : : MPI_Comm_rank(0:)	15.24us	-	1.52				
lulesh-comm.cc	436 : : : : : MPI_Isend(0:25)	13.14us	21.09KB	1.9				
lulesh-comm.cc	477 : : : : : MPI_Isend(0:5)	10.28us	21.09KB	1.89				
lulesh-comm.cc	518 : : : : : MPI_Isend(0:1)	10.57us	21.09KB	1.26				
lulesh-comm.cc	843 : : : : : MPI_Waitall(0:)	291.68us	-	0.45				
lulesh.cc	2014 : : : : : CommMonoQ()	-	-	-				
lulesh-comm.cc	1734 : : : : : MPI_Comm_rank(0:)	9.51us	-	0.45				
lulesh-comm.cc	1757 : : : : : MPI_Wait(0:)	8.88us	-	0.81				
lulesh-comm.cc	1793 : : : : : MPI_Wait(0:)	8.78us	-	2.04				
lulesh-comm.cc	1824 : : : : : MPI_Wait(0:)	8.53us	-	2.13				
	: END LOOP	-	-	-				

Table: Total MPI calls

	MPI_Isend	MPI_Irecv	MPI_Wait	MPI_Waitall	MPI_Allreduce	MPI_Comm_rank
Rank 0	300	510	510	90	29	270

Table: MPI calls counts by communication phases

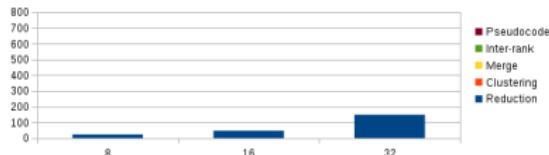
	MPI_Isend	MPI_Irecv	MPI_Wait	MPI_Waitall	MPI_Allreduce	MPI_Comm_rank
Comm phase 1	0	7	0	0	1	1
Comm phase 2	7	7	1	0	0	3
Comm phase 3	0	0	1	0	0	2
Comm phase 4	0	3	0	0	0	1
Comm phase 5	3	0	3	1	0	2

# Outline for section 12

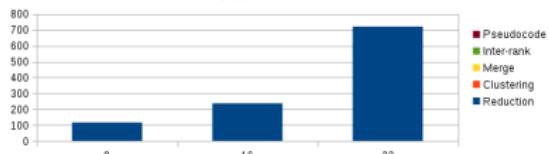
## 12 Scalability

# Scalability

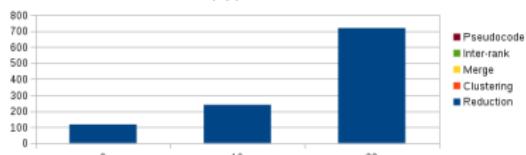
CG (A) phases breakdown



CG (B) phases breakdown

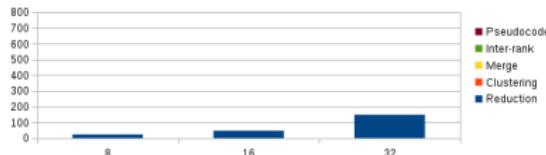


CG (C) phases breakdown

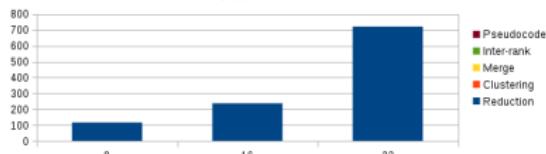


# Scalability

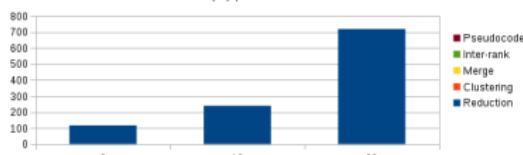
CG (A) phases breakdown



CG (B) phases breakdown



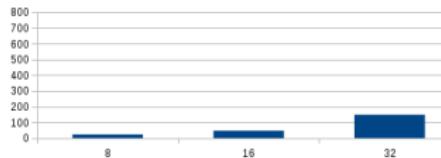
CG (C) phases breakdown



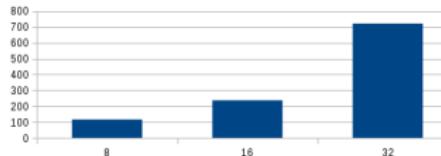
Reduction step is clearly dominating the execution time.

# Scalability

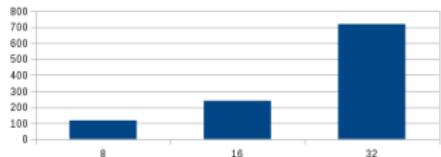
CG (A) phases breakdown



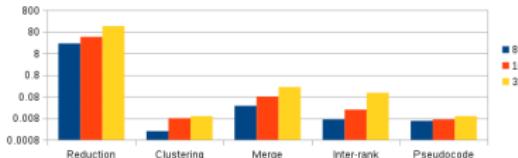
CG (B) phases breakdown



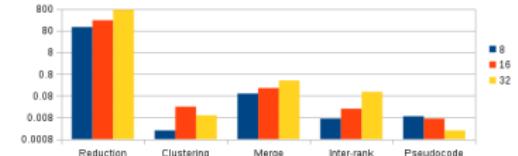
CG (C) phases breakdown



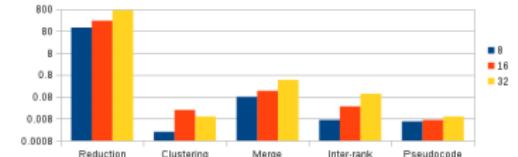
CG problem size A



CG problem size B



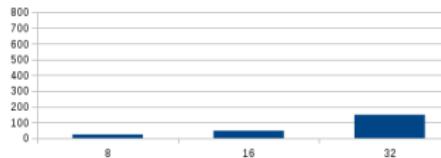
CG problem size C



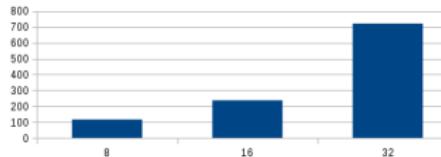
Reduction step is clearly dominating the execution time.

# Scalability

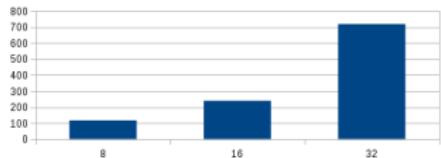
CG (A) phases breakdown



CG (B) phases breakdown

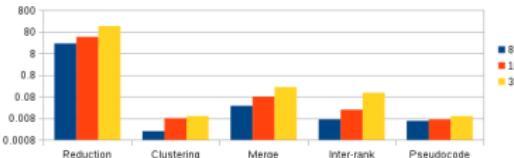


CG (C) phases breakdown

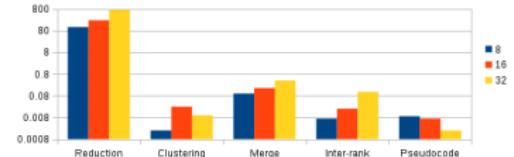


Reduction step is clearly dominating the execution time.

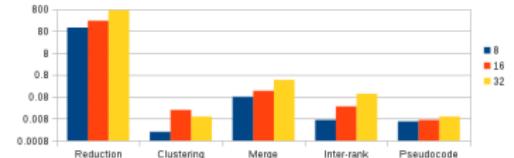
CG problem size A



CG problem size B

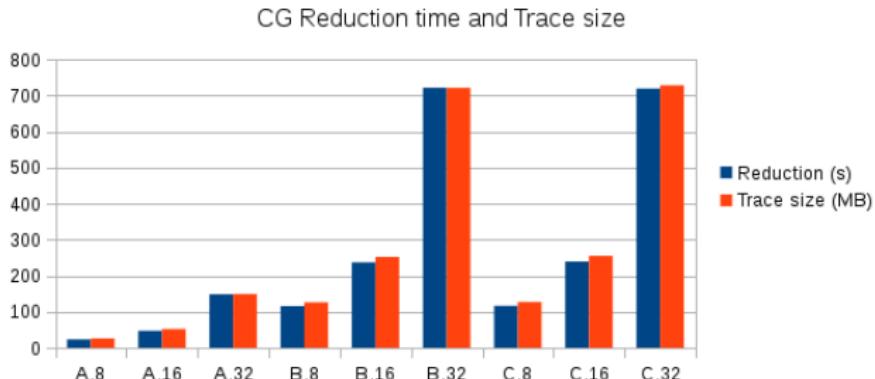


CG problem size C



The rest maintains its time even if increasing problem size.

# Scalability



Keynote

Reduction time is highly correlated with trace size