Master thesis

Master in Research and Innovation

Inferring programs structure from an execution trace

Juan Francisco Martínez Vera

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

April 18, 2018

Presentation outline I

1. Context
   - High Performance Computing
   - Performance Analysis tools
2. State of the Art
   - Syntactic structure
3. Proposal
   - Application structure by classification
   - Workflow
4. Further considerations
   - Data analysis
   - Proposal modifiations
5. Results
   - Specific capabilities
   - Real application
6. Conclusions

Outline for section 1

High Performance Computing

- Becomes the third support of science with mathematics and theory
- Tremendous improvement in all transformation hierarchy layers

High Performance Computing

- Becomes the third support of science with mathematics and theory
- Tremendous improvement in all transformation hierarchy layers

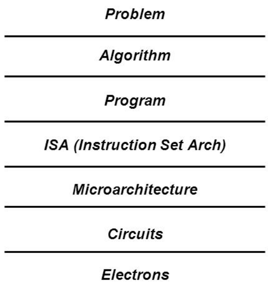
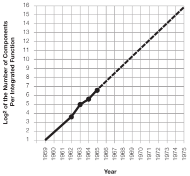


Figure: Transformation hierarchy

High Performance Computing

- Becomes the third support of science with mathematics and theory
- Tremendous improvement in all transformation hierarchy layers



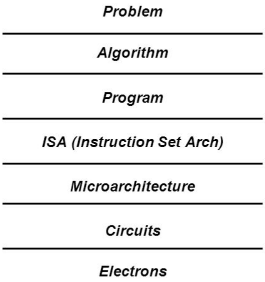
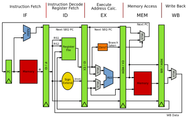
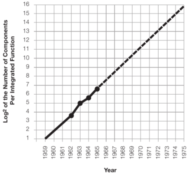


Figure: Transformation hierarchy

High Performance Computing

- Becomes the third support of science with mathematics and theory
- Tremendous improvement in all transformation hierarchy layers





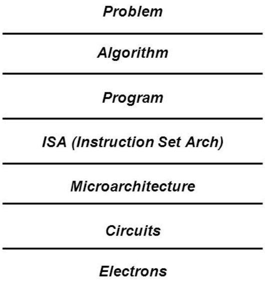
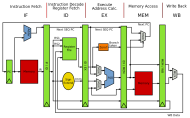
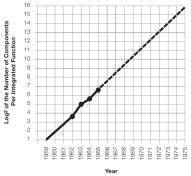


Figure: Transformation hierarchy

High Performance Computing

- Becomes the third support of science with mathematics and theory
- Tremendous improvement in all transformation hierarchy layers





Problem

Algorithm

Program

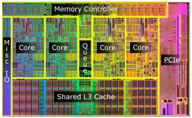
ISA (Instruction Set Arch)

Microarchitecture

Circuits

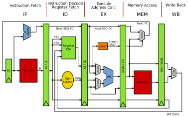
Electrons

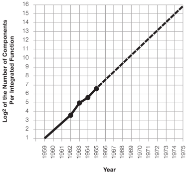
Figure: Transformation hierarchy



High Performance Computing

- Becomes the third support of science with mathematics and theory
- Tremendous improvement in all transformation hierarchy layers





Problem

Algorithm

Program

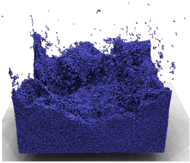
ISA (Instruction Set Arch)

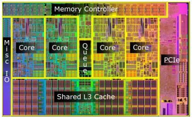
Microarchitecture

Circuits

Electrons

Figure: Transformation hierarchy





Performance Analysis tools (i)

- Focused on Program layer
- Aid to detect bottlenecks
- Is a cyclic process
    - Less iterations better
    - Depens on quality of hypothesis
    - That is strongly related with possibilities tools provides

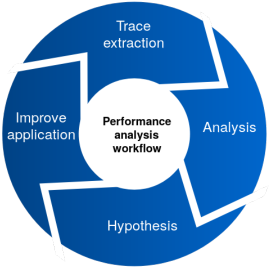


Figure: Performance analysis workflow

Performance Analysis tools (ii)

- Have been demonstrate to be valuable on detection of bottlenecks
- Demands high skilled analysts
- So developers use to delegate this work
  - **Analyst have to work with codes they are not familiar with**

Figure: Performance optimisation and Productivity project

Performance Analysis tools (ii)

- Have been demonstrate to be valuable on detection of bottlenecks
- Demands high skilled analysts
- So developers use to delegate this work
  - **Analyst have to work with codes they are not familiar with**

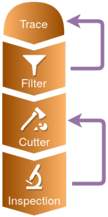
Figure: Performance optimisation and Productivity project

Motivation 1

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

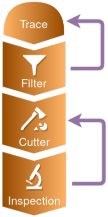
Performance Analysis tools (iii)

- When dealing with big traces:
  - Visualizers responsiveness times **becomes prohibitive**
- Analysis phases is then break down into:
  1. Filter
  2. Cutter
  3. Inspection



Performance Analysis tools (iii)

- When dealing with big traces:
  - Visualizers responsiveness times **becomes prohibitive**
- Analysis phases is then break down into:
  1. Filter
  2. Cutter
  3. Inspection



Motivation 2

Having the structure of the application will aid the process of identify regions of interest

Outline for section 2

State of the Art (i)

- On-line analysis
  - In [Noeth et al., 2009] propose structure derection for on-line trace compression
    - Rely on **detecting and aggregating patterns**
  - In [Aguilar et al., 2016] propose structure detection for on-line trace compression and visual performance analysis
    - Rely on **directed graph construction** and DFS-analysis for cycles hierarchical analysis.
- Off-line analysis
  - In [Safyallah and Sartipi, 2006] propose improving reverse engineering by structure detection analysis
    - Rely on **sequential pattern mining** techniques.
  - In [López-Cueva et al., 2012] propose to aid huge SoC traces
    - Rely on **frequent periodic pattern mining**.
  - In [Trahay et al., 2015] propose to select automatically the performance hotspots
    - Rely on **growing sequential pattern mining**.

State of the Art (ii)

- On-line analysis
  - [Noeth et al., 2009] By avoiding $O(n^2)$ they **limits pattern recognition**
  - In [Aguilar et al., 2016] **Loss temporality** when detecting hierarchical structure
- Off-line analysis
  - [Safyallah and Sartipi, 2006] [López-Cueva et al., 2012] [Trahay et al., 2015] relies on pattern mining techniques that presents **high complexity**.

State of the Art (ii)

- On-line analysis
  - [Noeth et al., 2009] By avoiding $O(n^2)$ they **limits pattern recognition**
  - In [Aguilar et al., 2016] **Loss temporality** when detecting hierarchical structure
- Off-line analysis
  - [Safyallah and Sartipi, 2006] [López-Cueva et al., 2012] [Trahay et al., 2015] relies on pattern mining techniques that presents **high complexity**.

Motivation 3

Been scalable by finding out an alternative technique

Outline for section 3

Application structure by classification(i)

HPC applications idiosincracy

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

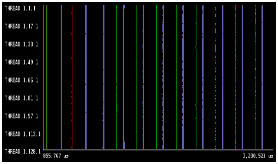


Figure: FT 128 ranks

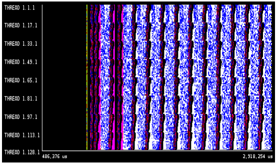


Figure: LU 128 ranks

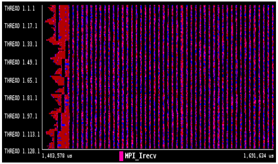


Figure: CG 128 ranks

Application structure by classification (ii)

- Instead of following the trend, different proposal
- Taking into account the characteristics of our target
- Loops can be discovered by monitoring the communications
- Stable executions implies same behaviour for all iterations in a given loop
- So...

Application structure by classification (ii)

- Instead of following the trend, different proposal
- Taking into account the characteristics of our target
- Loops can be discovered by monitoring the communications
- Stable executions implies same behaviour for all iterations in a given loop
- So...

The key idea

Communications are used as proxies for the observation of iterations. Clustering them by its behaviour the applications structures can be betrayed.

Application structure by classification (iii)

Selected features must be able to

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

Application structure by classification (iii)

Selected features must be able to

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

As a starting point ...

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

Workflow

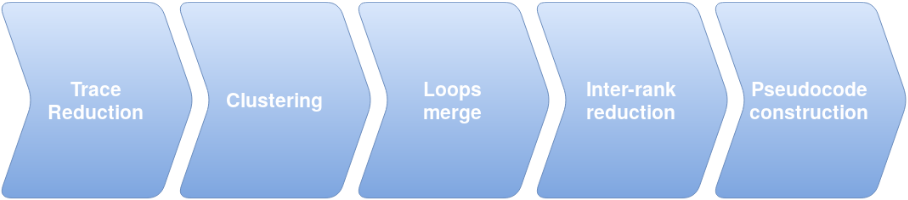


Figure: Structure detection workflow

Workflow

Trace reduction step (i)

Input Tracefile, i.e. Sequence of timestamped events ordered by time.

Output Set of unique MPI calls with attached information.

Workflow

Trace reduction step (i)

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

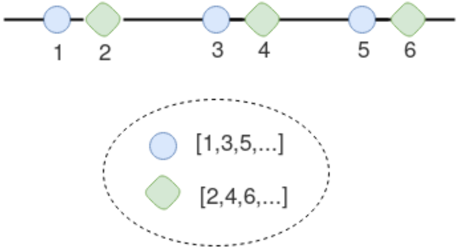
- By **reduction** and **aggregation & derivation**
- Is a sort of Map & Reduce
- Every MPI call is identified by its **signature**
- Being the signature: Ordered sequence of pairs (*file*, *line*) that define the call path, i.e. The dynamic position

Workflow

Trace reduction step (i)

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

- By **reduction** and **aggregation & derivation**
- Is a sort of Map & Reduce
- Every MPI call is identified by its **signature**
- Being the signature: Ordered sequence of pairs (*file*, *line*) that define the call path, i.e. The dynamic position



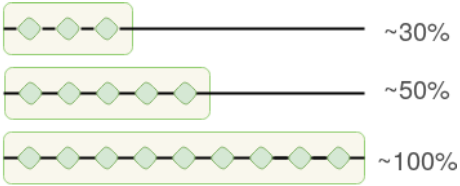
Workflow

Trace reduction step (ii)

Additionally **filter less representative MPI calls**.

- Allows to decrease even more the clustering complexity.
- Focus only on the important data.
- The criteria is whether a given threshold of “explained time” is surpased.

$$\delta(call) = \frac{it(call) * imt(call)}{T_{exe}} \tag{1}$$



Workflow

Trace reduction step (iii)

The stored information for every unique MPI call is:

- **Number of repetitions**
- **Mean time time between repetitions**
- Entire call path
- Previous burst performance information
- All timestamps
- Calculed delta

Workflow

Trace reduction step (iii)

The stored information for every unique MPI call is:

- **Number of repetitions**
- **Mean time time between repetitions**
- Entire call path
- Previous burst performance information
- All timestamps
- Calculed delta

Keynote

HPC applications are strongly repetitive over time so number of unique MPI calls will remain despite the increasing problem size.

Workflow

Clustering

Input Set of unique MPI calls with attached information.
Output Set of sets of MPI calls.

Workflow

Clustering

Input Set of unique MPI calls with attached information.
Output Set of sets of MPI calls.

- **DBSCAN** as clustering algorithm (prefered to K-means)
  - ϵ empirically set to 0.2 (in general)
  - minPts set to 1

Workflow

Clustering

Input Set of unique MPI calls with attached information.
Output Set of sets of MPI calls.

- **DBSCAN** as clustering algorithm (prefered to K-means)
  - ϵ empirically set to 0.2 (in general)
  - minPts set to 1
- In a bidimentional space defined by
  - Number of repetitions
  - Mean time between repetitions

Workflow

Clustering

Input Set of unique MPI calls with attached information.
Output Set of sets of MPI calls.

- **DBSCAN** as clustering algorithm (prefered to K-means)
  - ϵ empirically set to 0.2 (in general)
  - minPts set to 1
- In a bidimentional space defined by
  - Number of repetitions
  - Mean time between repetitions
- Resulting clusters **will be considered loops**.
  - Since MPI calls acts as proxies
  - Number of repetitions $\rightarrow$ Number of iterations.
  - Mean time between repetitions $\rightarrow$ Mean iterations time.

Workflow

Loops merge (i)

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

Workflow

Loops merge (i)

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 betrayed**.

Workflow

Loops merge (i)

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 betrayed**.

Some clues

- Outer loop will have **more iterations** then nested one.
- Outer loop will spend **more time** for per iteration.
- Outer loop will **explain the same amount of time** as inner loop.

Workflow

Loops merge (ii)

comment: Short initialization

for 1 to 10

$$\textbf{do} \begin{cases} \textbf{for } 1 \text{ to } 2 \\ \quad \textbf{do} \begin{cases} \textbf{for } 1 \text{ to } 2 \\ \quad \textbf{do} \left\{ someComms() \right. \\ \text{MPI\_Call} \end{cases} \\ \text{MPI\_Call} \end{cases}$$

comment: Body of execution

for 1 to 100

$$\textbf{do} \begin{cases} \textbf{for } 1 \text{ to } 2 \\ \quad \textbf{do} \begin{cases} \textbf{for } 1 \text{ to } 2 \\ \quad \textbf{do} \left\{ someComms() \right. \\ \text{MPI\_Call} \end{cases} \\ \text{MPI\_Call} \end{cases}$$

Workflow

Loops merge (ii)

comment: Short initialization

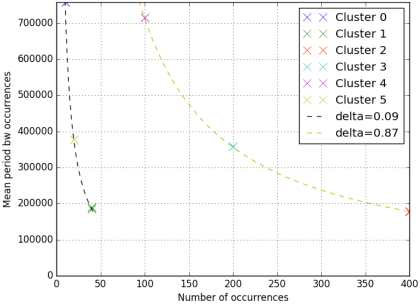
for 1 to 10

```
do {
    for 1 to 2
    do {
        for 1 to 2
        do { someComms()
        MPI_Call
    }
    MPI_Call
}
```

comment: Body of execution

for 1 to 100

```
do {
    for 1 to 2
    do {
        for 1 to 2
        do { someComms()
        MPI_Call
    }
    MPI_Call
}
```



Workflow

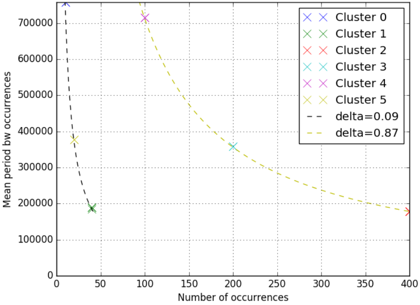
Loops merge (ii)

comment: Short initialization

for 1 to 10
do
- **for** 1 to 2
  do
  - **for** 1 to 2
    do $\{someComms()$
  - MPI_Call
- MPI_Call

comment: Body of execution

for 1 to 100
do
- **for** 1 to 2
  do
  - **for** 1 to 2
    do $\{someComms()$
  - MPI_Call
- MPI_Call



Keynote

Different phases can be detected by this way and used for the loops merging.

Workflow

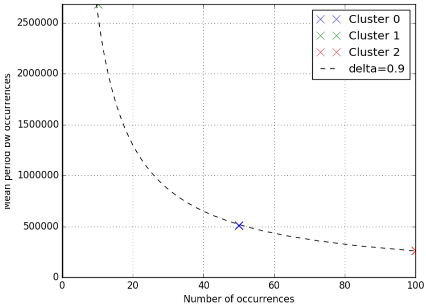
Loops merge (iii)

$$\textbf{for } 1 \text{ to } 10 \\ \quad \textbf{do } \begin{cases} \textbf{for } 1 \text{ to } 5 \\ \quad \textbf{do } \left\{ someComms() \right. \\ \textbf{for } 1 \text{ to } 10 \\ \quad \textbf{do } \left\{ someComms() \right. \\ \text{MPI\_Call} \end{cases}$$

Workflow

Loops merge (iii)

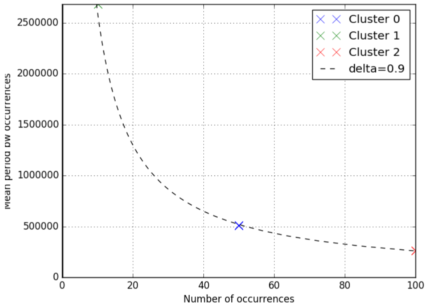
$$\textbf{for } 1 \text{ to } 10$$
$$\textbf{do} \begin{cases} \textbf{for } 1 \text{ to } 5 \\ \quad \textbf{do} \ \{ someComms() \\ \textbf{for } 1 \text{ to } 10 \\ \quad \textbf{do} \ \{ someComms() \\ \text{MPI\_Call} \end{cases}$$



Workflow

Loops merge (iii)

$$\textbf{for } 1 \text{ to } 10 \quad \textbf{do} \begin{cases} \textbf{for } 1 \text{ to } 5 \\ \quad \textbf{do} \ \{ someComms() \\ \textbf{for } 1 \text{ to } 10 \\ \quad \textbf{do} \ \{ someComms() \\ \text{MPI\_Call} \end{cases}$$



Warning

Not all loops that fulfills with nested loops conditions, are nested loops. Extra check is needed.

Workflow

Loops merge (iv)

- Classify MPI clusters/Loops ($\upsilon \in \Upsilon$) per "how much of execution represents" ($\delta \in \Delta$)[1].
- For every phase (δ) sort loops (υ) by number of iterations.
- Perform the loop merge **from high to low iterations count**.
- Before every merge, **check out** whether the hierarchical relationship is true.

$$
\begin{array}{l}
\Delta \leftarrow deltaClassification(\Upsilon) \\
\textbf{for all } \delta \in \Delta \\
\quad \textbf{do} \begin{cases}
\textbf{comment:} \text{ Sort by it}(\upsilon) \text{ desc} \\
sort(\upsilon \in \delta) \\
\textbf{for } i \in [0, |\delta| - 1) \\
\quad \textbf{do} \begin{cases}
\textbf{for } j \in [i+1, |\delta|) \\
\quad \textbf{do} \begin{cases}
\textbf{if } isSubloop(\upsilon_i, \upsilon j) \\
\quad \textbf{then } \upsilon_i \mapsto \upsilon_j
\end{cases}
\end{cases}
\end{cases}
\end{array}
$$

[1] Do not confuse with $\delta()$ function.

Workflow

Inter-rank reduction (i)

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

[2] Single Program Multiple Data

Workflow

Inter-rank reduction (i)

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

- Two calls with **same call paths still coexist** if belongs to different MPI ranks.
  - **Conservative** reduction step! If assuming SPMD[2] applications.
- Divergences between MPI ranks are understood as **conditional structures in code**.
- This step is about:
  1. MPI Calls/Subloops ordenation
  2. Reduction
  3. Arrangement in conditional blocks

[2] Single Program Multiple Data

Workflow

Inter-rank reduction (ii)

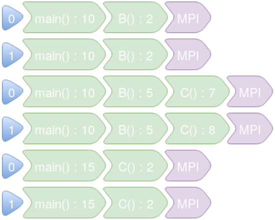


Figure: Ordenation

Workflow

Inter-rank reduction (ii)

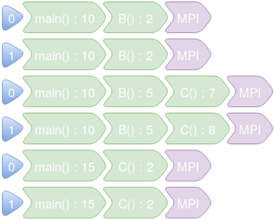


Figure: Ordenation

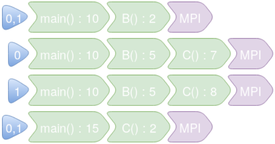


Figure: Reduction

Workflow

Inter-rank reduction (ii)

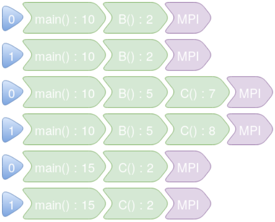


Figure: Ordenation

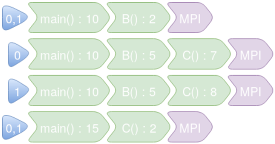


Figure: Reduction

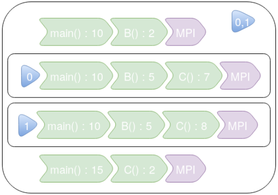


Figure: Conditional blocks

Workflow

Pseudocode construction (i)

Input Set of top level loops with rank conditional structures.
Output Pseudocode representing the actual application structure.

Workflow

Pseudocode construction (i)

Input Set of top level loops with rank conditional structures.

Output Pseudocode representing the actual application structure.

- Straightforward construction
- For bettern understanding **repetitive information from call paths is removed**.
  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

$$\textbf{for } 1 \text{ to } N \quad \textbf{do} \begin{cases} a:1 \rightarrow b:1 \rightarrow mpi \\ a:1 \rightarrow b:1 \rightarrow mpi \\ a:1 \rightarrow b:2 \rightarrow mpi \\ a:1 \rightarrow b:2 \rightarrow mpi \end{cases}$$

Figure: Raw pseudocode

Workflow

Pseudocode construction (i)

Input Set of top level loops with rank conditional structures.

Output Pseudocode representing the actual application structure.

- Straightforward construction
- For bettern understanding **repetitive information from call paths is removed**.
  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

$$\textbf{for } 1 \text{ to } N \quad \textbf{do} \begin{cases} a:1 \to b:1 \to mpi \\ a:1 \to b:1 \to mpi \\ a:1 \to b:2 \to mpi \\ a:1 \to b:2 \to mpi \end{cases}$$

Figure: Raw pseudocode

$$a:1 \to \quad \textbf{for } 1 \text{ to } N \quad \textbf{do} \begin{cases} b:1 \to mpi \\ b:1 \to mpi \\ b:2 \to mpi \\ b:2 \to mpi \end{cases}$$

Figure: After extract common call path

Workflow

Pseudocode construction (i)

Input Set of top level loops with rank conditional structures.

Output Pseudocode representing the actual application structure.

- Straightforward construction
- For bettern understanding **repetitive information from call paths is removed**.
  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

$$\textbf{for } 1 \text{ to } N \quad \textbf{do} \begin{cases} a:1 \rightarrow b:1 \rightarrow mpi \\ a:1 \rightarrow b:1 \rightarrow mpi \\ a:1 \rightarrow b:2 \rightarrow mpi \\ a:1 \rightarrow b:2 \rightarrow mpi \end{cases}$$

Figure: Raw pseudocode

$$a:1 \rightarrow \quad \textbf{for } 1 \text{ to } N \quad \textbf{do} \begin{cases} b:1 \rightarrow mpi \\ b:1 \rightarrow mpi \\ b:2 \rightarrow mpi \\ b:2 \rightarrow mpi \end{cases}$$

Figure: After extract common call path

$$\textbf{for } 1 \text{ to } N \quad \textbf{do} \begin{cases} b:1 \\ \rightarrow mpi \\ \rightarrow mpi \\ b:2 \\ \rightarrow mpi \\ \rightarrow mpi \end{cases}$$

Figure: After extract contiguous call path

Workflow

Pseudocode construction (ii)

FILE	LINE	PSEUDOCODE	E(TIME)	E(SIZE)	E(IPC)
	0	main()	-	-	-
		: FOR 1 TO 10 [id=2.0]	-	-	-
		: : FOR 1 TO 2.0 [id=0.0]	-	-	-
test-2.c	42	: : : CommSend()	-	-	-
		: : : : IF rank in [1]	-	-	-
test-2.c	14	: : : : : MPI_Send(1:0)	6.37us	4.0B	0.69
test-2.c	16	: : : : : MPI_Recv(1:1)	8.47us	4.0B	0.33
test-2.c	44	: : : CommRecv()	-	-	-
		: : : : IF rank in [0]	-	-	-
test-2.c	25	: : : : : MPI_Recv()	88.21us	-	0.7
test-2.c	26	: : : : : MPI_Send(0:1)	6.82us	4.0B	0.77
		: : END LOOP	-	-	-
		: : FOR 1 TO 5.0 [id=1.0]	-	-	-
test-2.c	49	: : : CommSend()	-	-	-
		: : : : IF rank in [1]	-	-	-
test-2.c	14	: : : : : MPI_Send(1:0)	6.08us	4.0B	0.82
test-2.c	16	: : : : : MPI_Recv(1:1)	8.54us	4.0B	0.3
test-2.c	51	: : : CommRecv()	-	-	-
		: : : : IF rank in [0]	-	-	-
test-2.c	25	: : : : : MPI_Recv()	168.88us	-	0.86
test-2.c	26	: : : : : MPI_Send(0:1)	6.27us	4.0B	0.79
		: : END LOOP	-	-	-
test-2.c	53	: : MPI_Barrier(CommId:1.0)	97.72us	-	0.44
		: END LOOP	-	-	-

Figure: Example console output

Workflow

Pseudocode construction (ii)

FILE	LINE	PSEUDOCODE	E(TIME)	E(SIZE)	E(IPC)
	0	main()	-	-	-
		: FOR 1 TO 10 [id=2.0]	-	-	-
		: : FOR 1 TO 2.0 [id=0.0]	-	-	-
test-2.c	42	: : : CommSend()	-	-	-
		: : : : IF rank in [1]	-	-	-
test-2.c	14	: : : : : MPI_Send(1:0)	6.37us	4.0B	0.69
test-2.c	16	: : : : : MPI_Recv(1:1)	8.47us	4.0B	0.33
test-2.c	44	: : : CommRecv()	-	-	-
		: : : : IF rank in [0]	-	-	-
test-2.c	25	: : : : : MPI_Recv()	88.21us	-	0.7
test-2.c	26	: : : : : MPI_Send(0:1)	6.82us	4.0B	0.77
		: : END LOOP	-	-	-
		: : FOR 1 TO 5.0 [id=1.0]	-	-	-
test-2.c	49	: : : CommSend()	-	-	-
		: : : : IF rank in [1]	-	-	-
test-2.c	14	: : : : : MPI_Send(1:0)	6.08us	4.0B	0.82
test-2.c	16	: : : : : MPI_Recv(1:1)	8.54us	4.0B	0.3
test-2.c	51	: : : CommRecv()	-	-	-
		: : : : IF rank in [0]	-	-	-
test-2.c	25	: : : : : MPI_Recv()	168.88us	-	0.86
test-2.c	26	: : : : : MPI_Send(0:1)	6.27us	4.0B	0.79
		: : END LOOP	-	-	-
test-2.c	53	: : MPI_Barrier(CommId:1.0)	97.72us	-	0.44
		: END LOOP	-	-	-

Figure: Example console output

- Additionally an **interactive shell** is provided to the user allowing...
  1. Show cpu burst metrics over a given thresshold.
  2. Filter by MPI rank.
  3. Show clustering plot.
  4. ...

Outline for section 4

Further considerations

Until now were not aware **the problems can arise from clustering step**. But there are some:

1. **Cluster aliasing** when two different loops behaves similarly enough over our defined space
2. **Cluster split** when MPI calls belonging to the same loop behaves in a different way.

Further considerations

Until now were not aware **the problems can arise from clustering step**. But there are some:

1. **Cluster aliasing** when two different loops behaves similarly enough over our defined space
2. **Cluster split** when MPI calls belonging to the same loop behaves in a different way.

$$\textbf{for } 1 \text{ to } 2 \quad \textbf{do} \begin{cases} \textbf{for } 1 \text{ to } 2 \\ \quad \textbf{do} \begin{cases} MPI\_A() \\ MPI\_B() \end{cases} \\ \textbf{for } 1 \text{ to } 2 \\ \quad \textbf{do} \begin{cases} MPI\_C() \\ MPI\_D() \end{cases} \end{cases}$$

Figure: Cluster aliasing example

$$\textbf{for } i = 1 \text{ to } 10 \quad \textbf{do} \begin{cases} \textbf{if } isPair(i) \\ \quad \textbf{then } MPI\_A() \\ MPI\_B() \\ MPI\_C() \end{cases}$$

Figure: Cluster split example

Outline for section 5

Hola manola

Hola manola

Outline for section 6

Hola manola

Aguilar, X., Fürlinger, K., and Laure, E. (2016).
Event flow graphs for mpi performance monitoring and analysis.
In *Tools for High Performance Computing 2015*, pages 103–115. Springer.

López-Cueva, P., Bertaux, A., Termier, A., Méhaut, J. F., and Santana, M. (2012).
Periodic pattern mining of embedded multimedia application traces.
In *Lecture Notes in Electrical Engineering*, volume 181 LNEE, pages 29–37.

Noeth, M., Ratn, P., Mueller, F., Schulz, M., and de Supinski, B. R. (2009).
Scalatrace: Scalable compression and replay of communication traces for high-performance computing.
Journal of Parallel and Distributed Computing, 69(8):696–710.

Patt, Y. N. (2017).
Computer architecture principles and tradeoffs.
Seminar lecture.

Safyallah, H. and Sartipi, K. (2006).
Dynamic Analysis of Software Systems using Execution Pattern Mining.
The 14th IEEE International Conference on Program Comprehension (ICPC '06), pages 84–88.

Trahay, F., Brunet, E., Bouksiaa, M. M., and Liao, J. (2015).
Selecting points of interest in traces using patterns of events.
In *Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference on*, pages 70–77. IEEE.