Enhancing the Analysis of Large Multimedia Applications Execution Traces with FrameMiner

C. Kamdem,L. Fopa,N. Ibrahim,A. Termier,M.-C. Rousset,T. Washio

christiane.kamdem-kengne@imag.fr,*leon-constantin.fopa@imag.fr*

December 10^{th}, 2012
PTDM@ICDM'12

Context

- Embedded systems
  - MPSoc: System on Chips with multiple processors

Context

- Embedded systems
  - × MPSoc: System on Chips with multiple processors
- Multimedia applications
  - × Video decoding
  - × Challenging issue using MPSoc

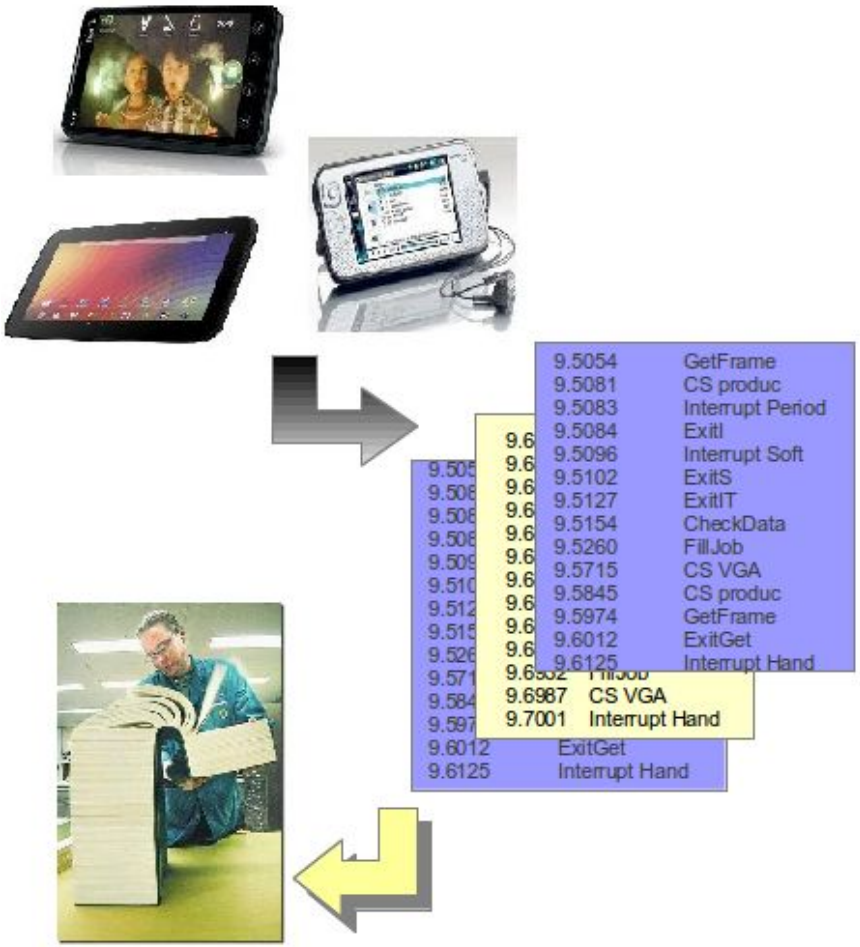
Context

- Embedded systems
  - × MPSoc: System on Chips with multiple processors
- Multimedia applications
  - × Video decoding
  - × Challenging issue using MPSoc
- Traditional debugging techniques are not optimal

Context

- Embedded systems
  - × MPSoc: System on Chips with multiple processors
- Multimedia applications
  - × Video decoding
  - × Challenging issue using MPSoc
- Traditional debugging techniques are not optimal
  $\Longrightarrow$ Execution traces analysis

Context



- Huge amount of traces

  $\Longrightarrow$ Example: 7GB of traces for less than 5mn of video decoding

State of the art - Traces Analysis

- ✓ **Visualization** [Stein,2003],[Vite,2010]

State of the art - Traces Analysis

- ✓ **Visualization** [Stein,2003],[Vite,2010]
  - Information overload

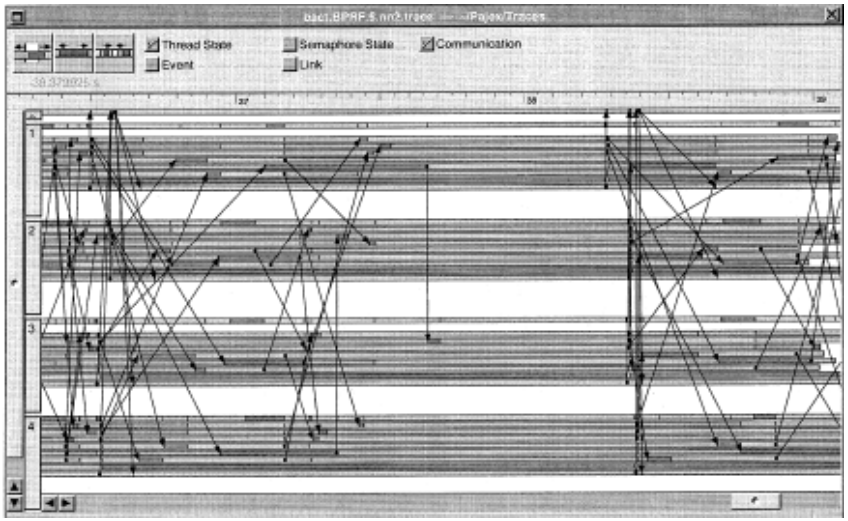


Figure: Paje, An interactive visualization tool

State of the art - Traces Analysis

✓ **Visualization** [Stein,2003],[Vite,2010]

- Information overload

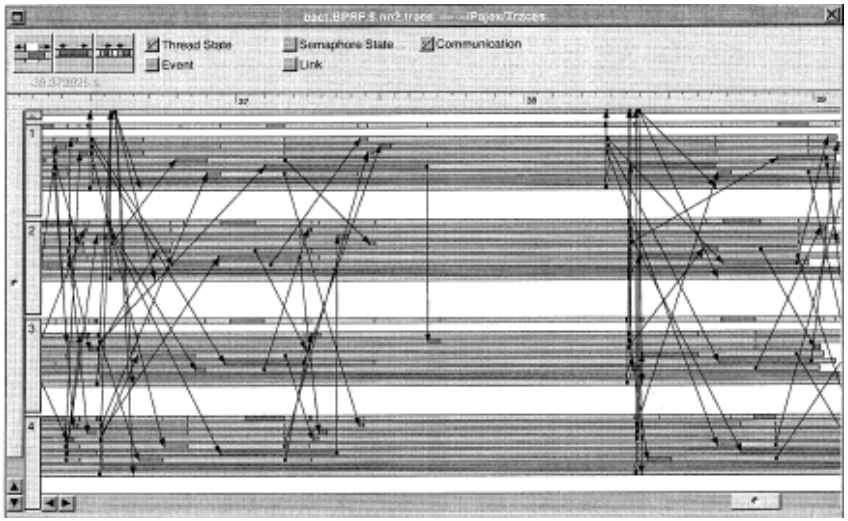


Figure: Paje, An interactive visualization tool

$\Longrightarrow$ Reduce size of execution traces

State of the art - Traces Analysis

- ✓ **Visualization** [Stein,2003],[Vite,2010]
  - Information overload

    $\Longrightarrow$ Reduce size of execution traces

- ✓ **Reduction** [Dugerdil,2007],[chan et al.,2003]

State of the art - Traces Analysis

- ✓ **Visualization** [Stein,2003],[Vite,2010]
  - Information overload

    $\Longrightarrow$ Reduce size of execution traces

- ✓ **Reduction** [Dugerdil,2007],[chan et al.,2003]
  - Not always representative of the entire trace

State of the art - Traces Analysis

- ✓ **Visualization** [Stein,2003],[Vite,2010]
  - Information overload

    $\Longrightarrow$ Reduce size of execution traces

- ✓ **Reduction** [Dugerdil,2007],[chan et al.,2003]
  - Not always representative of the entire trace

    $\Longrightarrow$ Abstract execution traces

Our goal: Abstraction

- Abstract series of low level events: blocks

9.5054	GetFrame
9.5073	ExitGet
9.5081	CS produc
9.5083	Interrupt Period
9.5084	ExitI
9.5096	Interrupt Soft
9.5102	ExitS
9.5127	ExitIT
9.5154	CheckData
9.5260	FillJob
9.5715	CS VGA
9.5845	CS produc
9.5974	GetFrame
9.6012	ExitGet
9.6125	Interrupt Hand
9.6155	ExitI
9.6234	ExitIT
9.6315	Interrupt Soft
9.6405	Interrupt Period
9.6483	ExitI
9.6514	Interrupt Soft
9.6622	ExitS
9.6715	ExitIT
9.6811	CS produc
9.6898	CheckData
9.6932	FillJob
9.6987	CS VGA
9.7001	Interrupt Hand

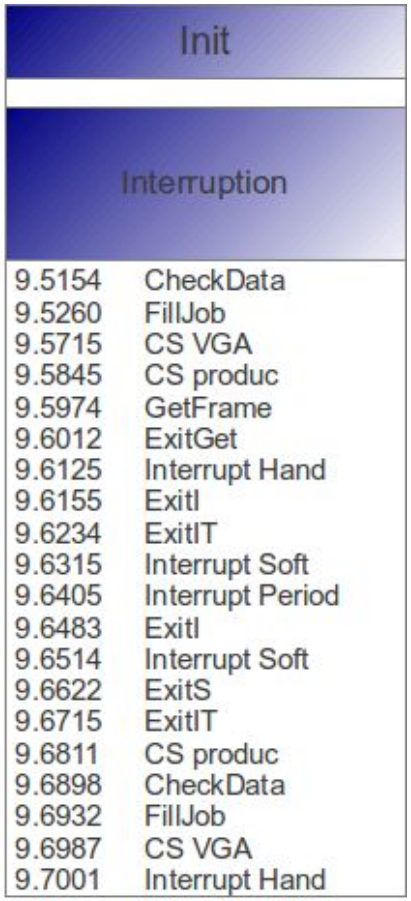
Our goal: Abstraction

- Abstract series of low level events: blocks

Init	
9.5081	CS produc
9.5083	Interrupt Period
9.5084	ExitI
9.5096	Interrupt Soft
9.5102	ExitS
9.5127	ExitIT
9.5154	CheckData
9.5260	FillJob
9.5715	CS VGA
9.5845	CS produc
9.5974	GetFrame
9.6012	ExitGet
9.6125	Interrupt Hand
9.6155	ExitI
9.6234	ExitIT
9.6315	Interrupt Soft
9.6405	Interrupt Period
9.6483	ExitI
9.6514	Interrupt Soft
9.6622	ExitS
9.6715	ExitIT
9.6811	CS produc
9.6898	CheckData
9.6932	FillJob
9.6987	CS VGA
9.7001	Interrupt Hand

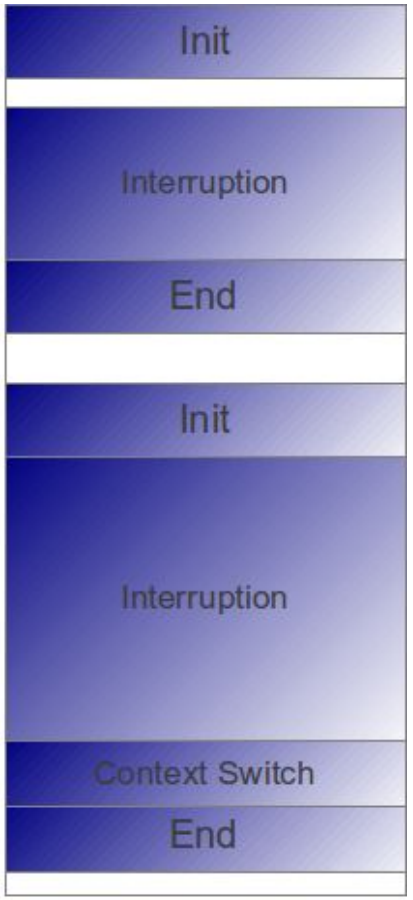
Our goal: Abstraction

- Abstract series of low level events: blocks



Our goal: Abstraction

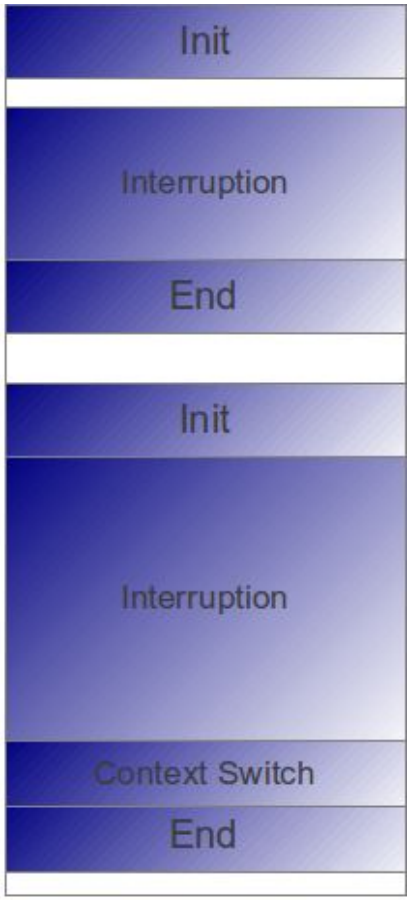
- Abstract series of low level events: blocks



Our goal: Abstraction

- Abstract series of low level events: blocks

  $\Longrightarrow$blocks are automatically discovered

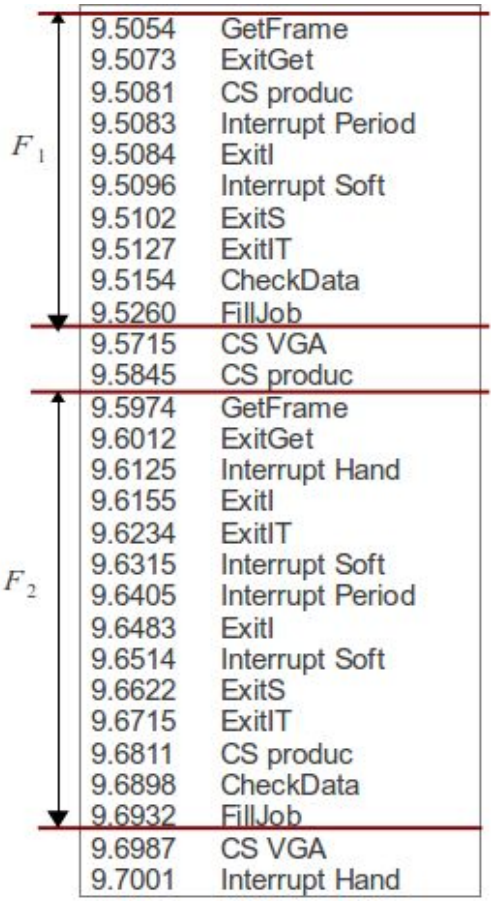


Our goal: Abstraction

- Abstract series of low level events: blocks

  $\Longrightarrow$blocks are automatically discovered

- A frame is identified by two events: *start* and *end* events

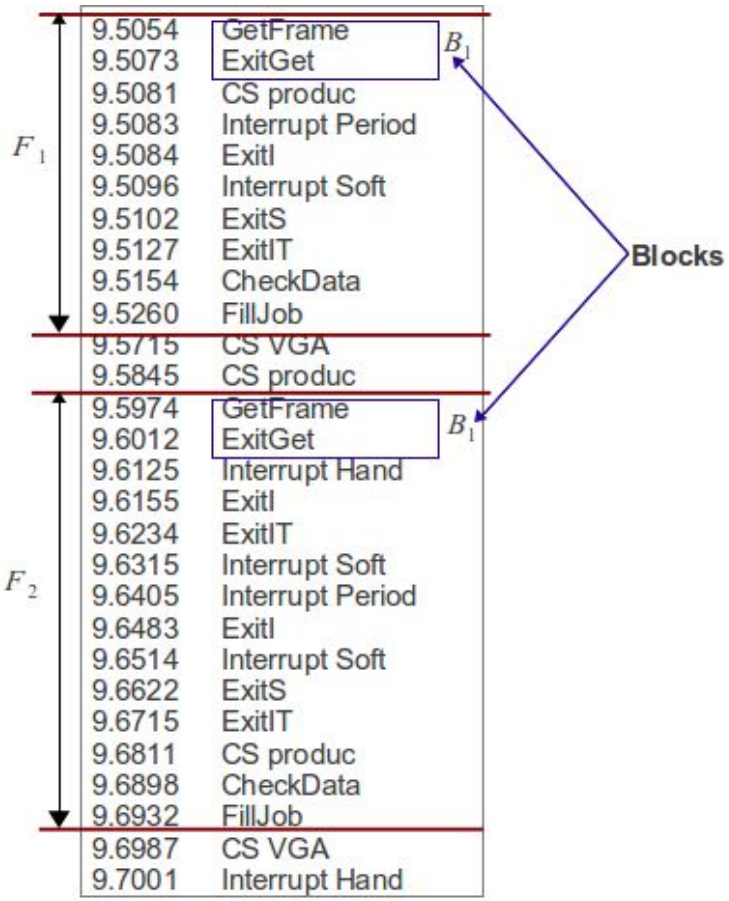


Our goal: Abstraction

- Abstract series of low level events: blocks

  $\Longrightarrow$blocks are automatically discovered

- A frame identified by two events: *start* and *end* events

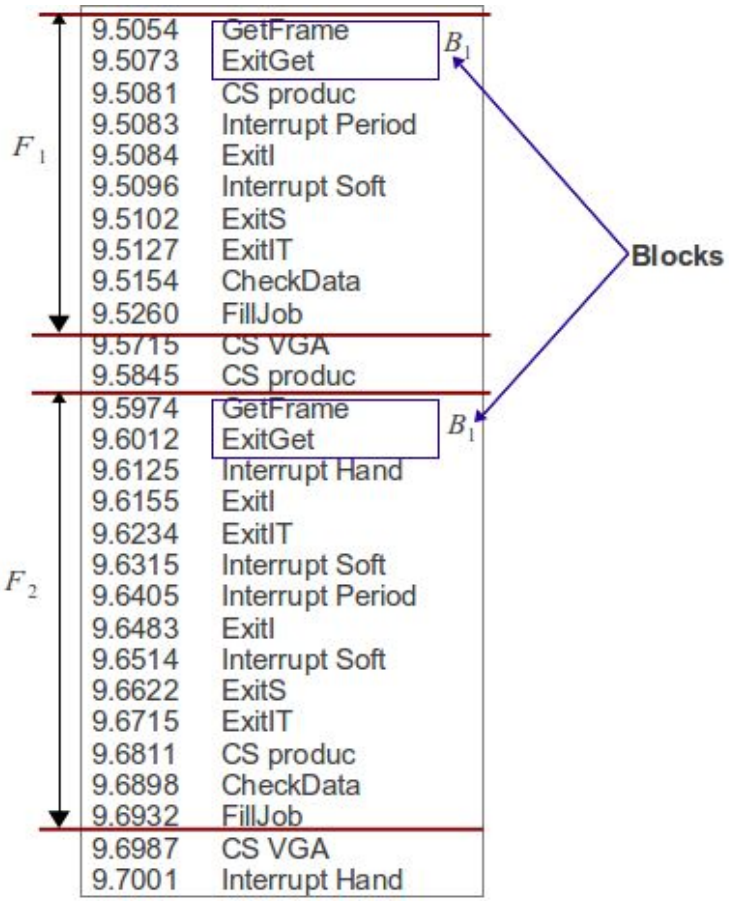


Our goal: Abstraction

- Abstract series of low level events: blocks

  $\Longrightarrow$blocks are automatically discovered

- A frame identified by two events: *start* and *end* events

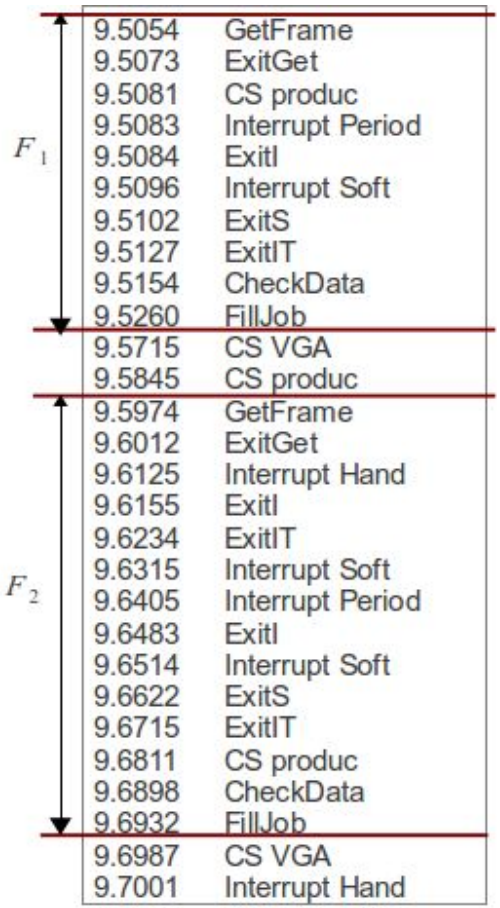


Our problem: Rewrite each frame into a short description with a small set of blocks

Definitions
example

Let $S = \{B_1, B_2, B_3, B_4\}$ with

$B_1 = \langle GetFrame, exitGet \rangle$
$B_2 = \langle InterrupPeriod, exitI, InterruptSoft, exitS, exitT \rangle$
$B_3 = \langle CheckData, FillJob \rangle$
$B_4 = \langle InterruptSoft, exitS, exitT, CSProduc \rangle$



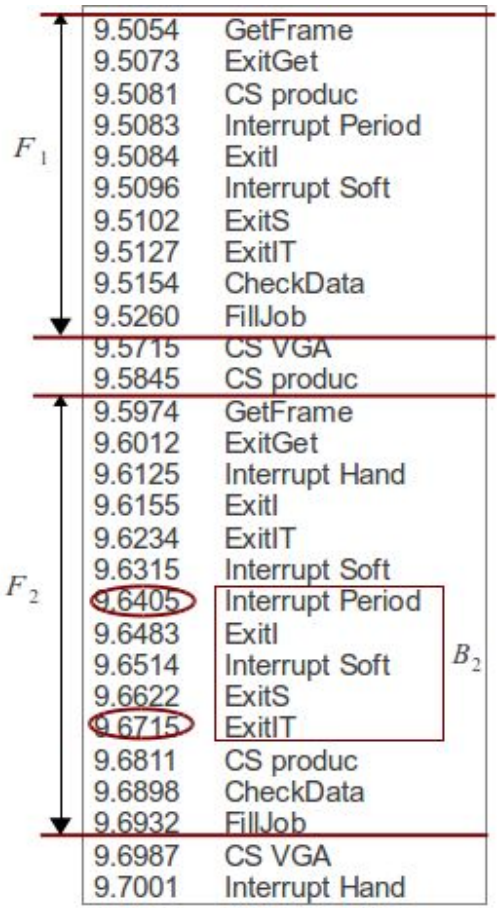
Definitions

1. local coverage
2. coverage
3. coverage rank
4. k-golden set

Definitions
example

Let $S = \{B_1, B_2, B_3, B_4\}$ with

$B_1 = \langle GetFrame, exitGet \rangle$
$B_2 = \langle InterrupPeriod, exitI, InterruptSoft, exitS, exitT \rangle$
$B_3 = \langle CheckData, FillJob \rangle$
$B_4 = \langle InterruptSoft, exitS, exitT, CSProduc \rangle$



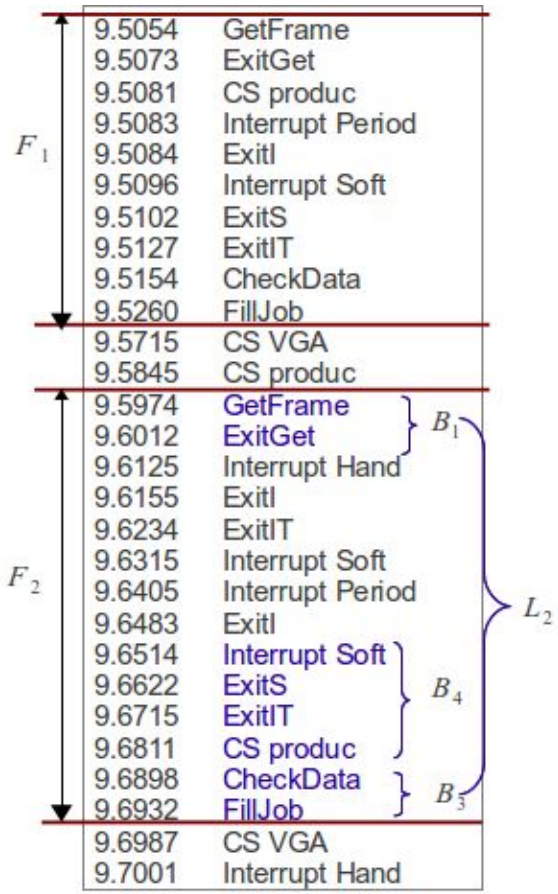
Definitions

1. local coverage
2. coverage
3. coverage rank
4. k-golden set

Definitions
example

Let $S = \{B_1, B_2, B_3, B_4\}$ with

$B_1 = \langle GetFrame, exitGet \rangle$
$B_2 = \langle InterrupPeriod, exitI, InterruptSoft, exitS, exitT \rangle$
$B_3 = \langle CheckData, FillJob \rangle$
$B_4 = \langle InterruptSoft, exitS, exitT, CSProduc \rangle$



Definitions

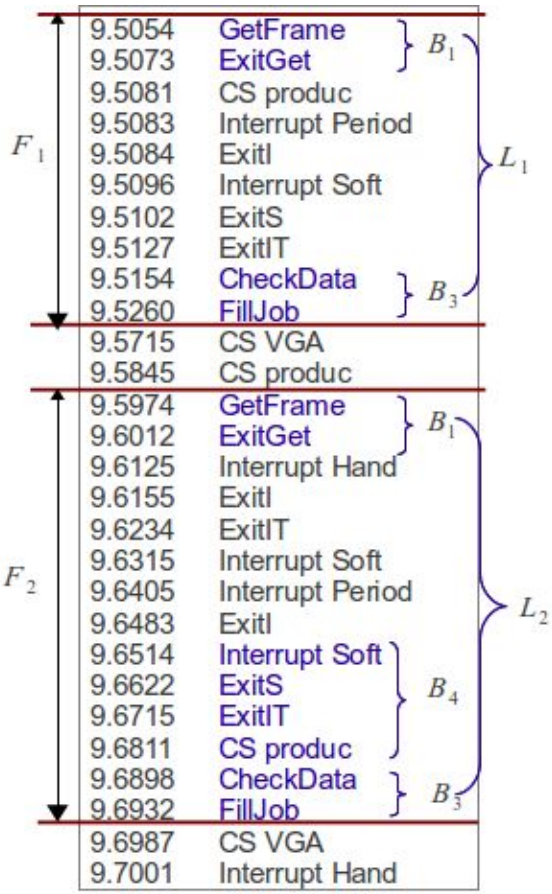
1. local coverage

   $L_2 = \langle B_1, B_4, B_3 \rangle$ *is a local coverage of* F_2
2. coverage
3. coverage rank
4. k-golden set

Definitions
example

Let $S = \{B_1, B_2, B_3, B_4\}$ with

$B_1 = \langle GetFrame, exitGet \rangle$
$B_2 = \langle InterrupPeriod, exitI, InterruptSoft, exitS, exitT \rangle$
$B_3 = \langle CheckData, FillJob \rangle$
$B_4 = \langle InterruptSoft, exitS, exitT, CSProduc \rangle$



Definitions

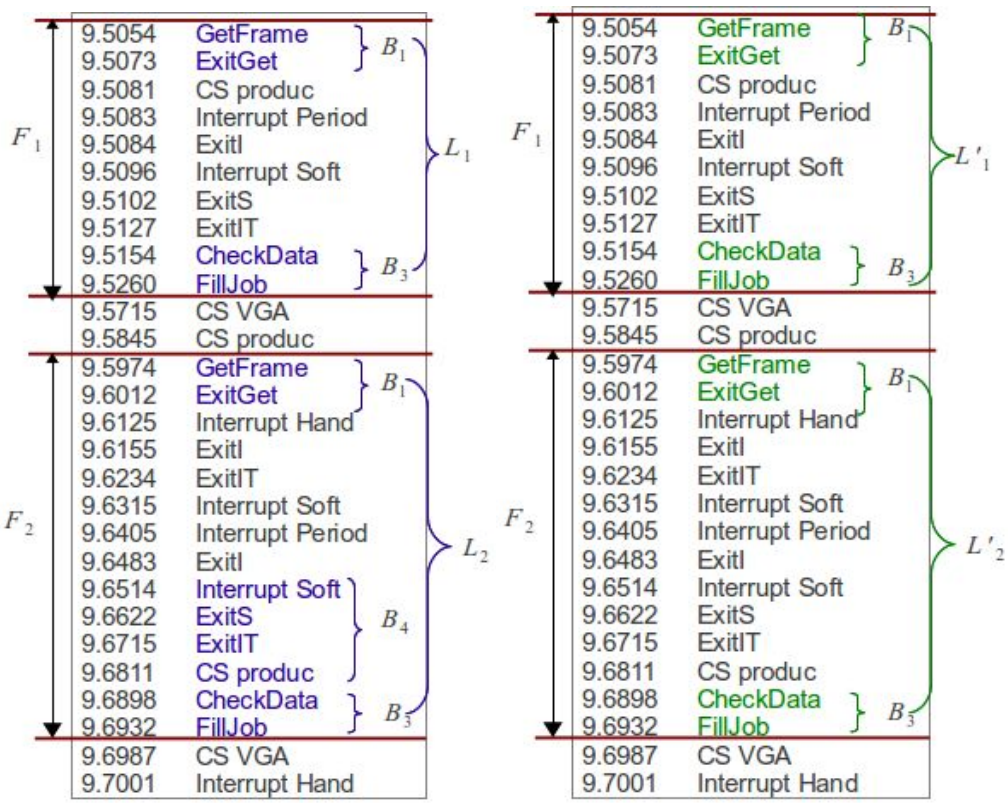
1. local coverage
2. coverage
   A coverage of $\mathcal{F} = \{F_1, F_2\}$ *using* S *is* $C = \{L_1, L_2\}$ *with* $L_1 = \langle B_1, B_3 \rangle$ *a local coverage* F_1

   The covering degree is 0.5
3. coverage rank
4. k-golden set

Definitions
example

Let $S = \{B_1, B_2, B_3, B_4\}$ with

$B_1 = \langle GetFrame, exitGet \rangle$
$B_2 = \langle InterrupPeriod, exitI, InterruptSoft, exitS, exitT \rangle$
$B_3 = \langle CheckData, FillJob \rangle$
$B_4 = \langle InterruptSoft, exitS, exitT, CSProduc \rangle$



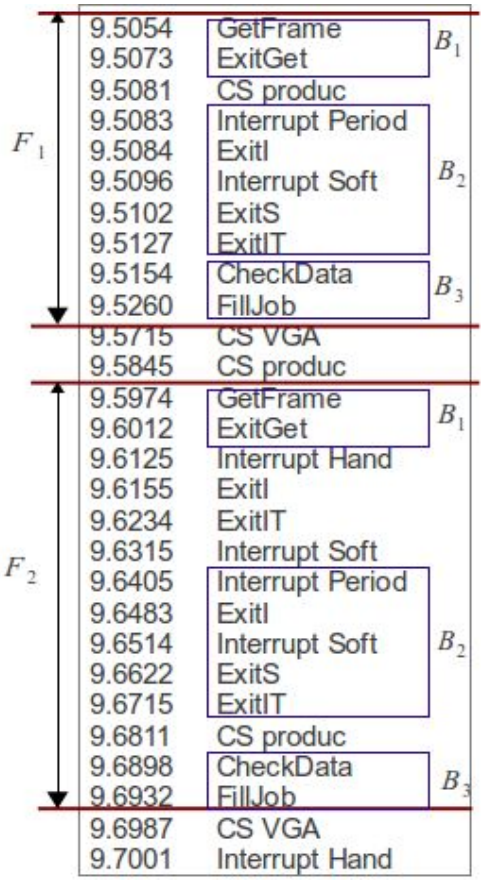
▸ Definitions

1. local coverage
2. coverage
3. coverage rank
   $coverRank(S_1, \mathcal{F}) = Max(0.25, 0.5, 0.33)$
   $with\ S_1 = \{B_1, B_3, B_4\}$
4. k-golden set

Definitions
example

Let $S = \{B_1, B_2, B_3, B_4\}$ with

$B_1 = \langle GetFrame, exitGet \rangle$
$B_2 = \langle InterrupPeriod, exitI, InterruptSoft, exitS, exitT \rangle$
$B_3 = \langle CheckData, FillJob \rangle$
$B_4 = \langle InterruptSoft, exitS, exitT, CSProduc \rangle$



Definitions

1. local coverage
2. coverage
3. coverage rank
4. k-golden set

   $S_2 = \{B_1, B_2, B_3\}$ *is the* 3-*goldenset*

   with $coverRank(S_2, \mathcal{F}) = 0.64$

Problem statement

submodular function maximization problem

Find a *k-golden* set of blocks that provides the best coverage

- ✓ Well known NP-Hard problem: $max\{f(S) : |S| \leq k\}$ [Nemhauser et al.,1978][Kulik,2009]
- ✓ Studied in particular for resource allocation
- ✓ Proved that the lower bound of any local optimum **f**, found by a greedy algorithm $\geq$ 63%

Problem statement

submodular function maximization problem

Find a *k-golden* set of blocks that provides the best coverage

- ✓ Well known NP-Hard problem: $max\{f(S) : |S| \leq k\}$ [Nemhauser et al.,1978][Kulik,2009]
- ✓ Studied in particular for resource allocation
- ✓ Proved that the lower bound of any local optimum **f**, found by a greedy algorithm $\geq$ 63%

- set of blocks obtained by applying *profSpan* [Zou et al.,2010]

Problem statement

submodular function maximization problem

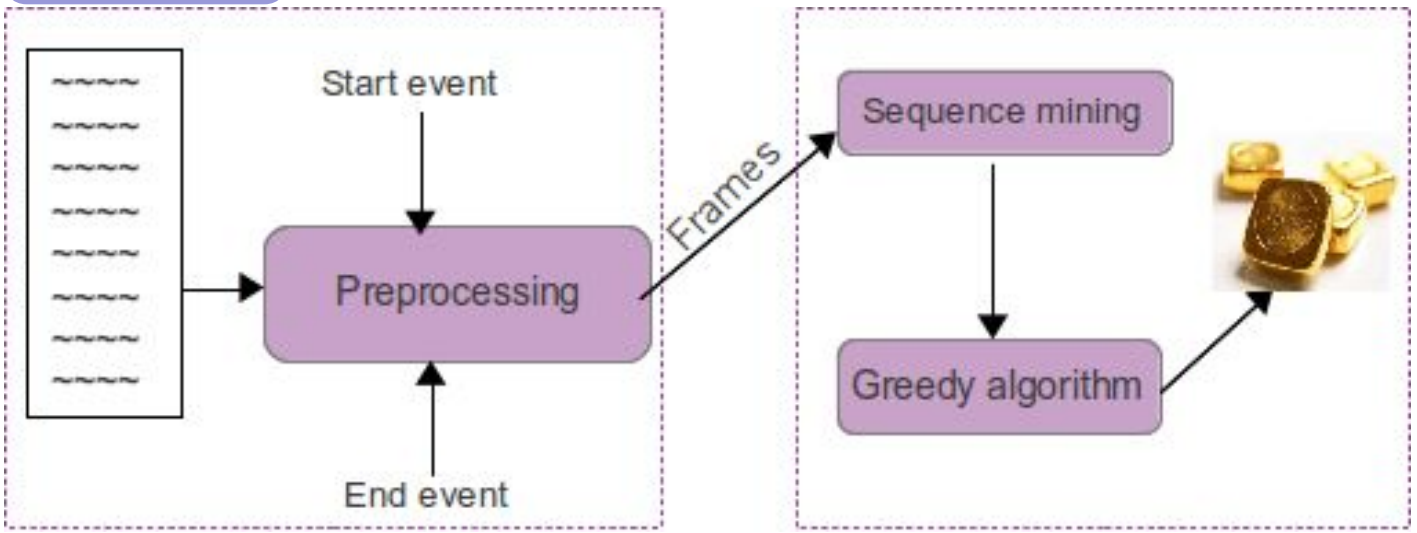
Find a *k-golden* set of blocks that provides the best coverage

- ✓ Well known NP-Hard problem: $max\{f(S) : |S| \leq k\}$ [Nemhauser et al.,1978][Kulik,2009]
- ✓ Studied in particular for resource allocation
- ✓ Proved that the lower bound of any local optimum **f**, found by a greedy algorithm $\geq$ 63%

- set of blocks obtained by applying *profSpan* [Zou et al.,2010]
- **f** is the function of ▸ *Coverage Rank*

FrameMiner Process

▸ Process Detailed

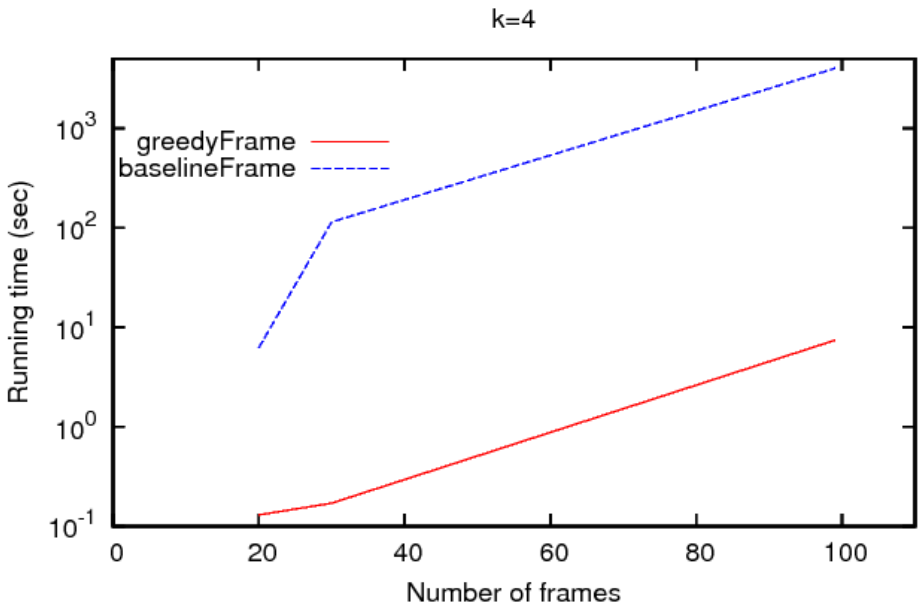


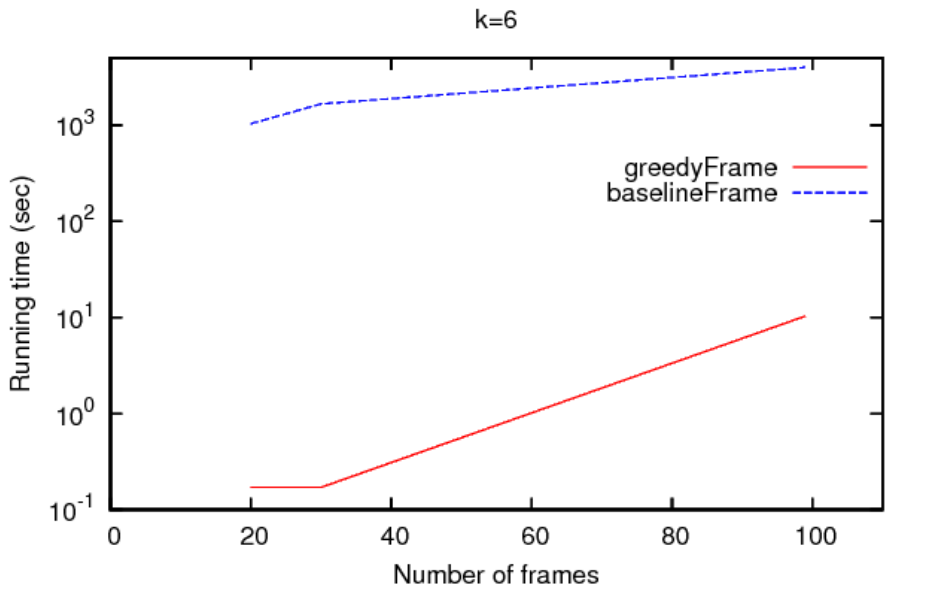
Experimentation settings

- Real trace of embedded MP4 video decoding
  - 240 frames and 123575 events
- Intel Xeon X4760 processor: 2.66GHZ and 64GB of RAM
- Exhaustive approach: *baselineFrame*
- *Profspan*: support threshold of 20%

Comparative assessment

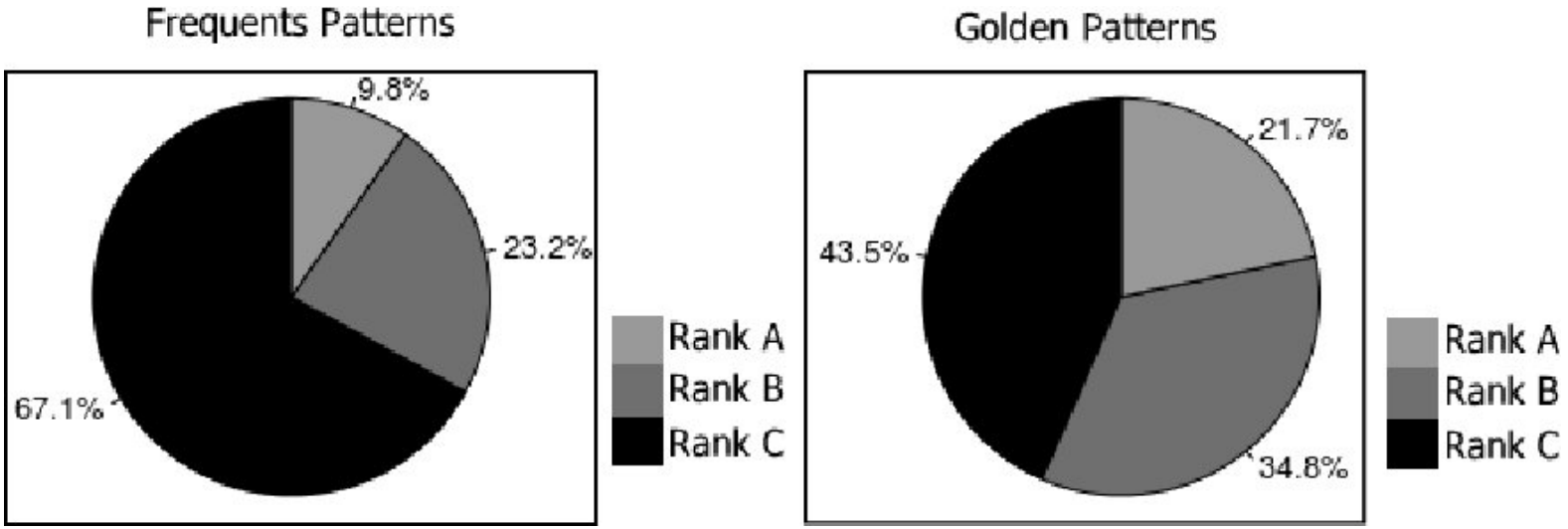
- 90% of the optimal solution with an execution three orders of magnitude faster.





Subjective assessment

Rank A: Very interesting; **Rank B**: Interesting; **Rank C**: Uninteresting

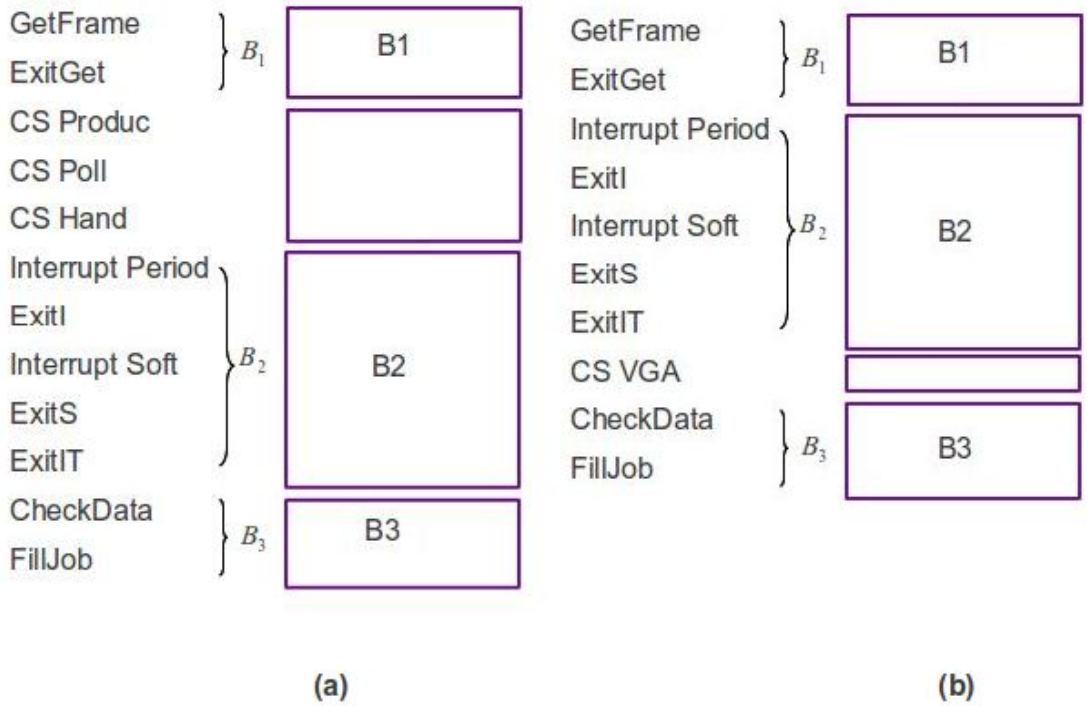


Reduction

- 70% of reduction in the information to handle

Reduction

- 70% of reduction in the information to handle
  $\Longrightarrow$ The programmer can now focus on blocks, instead of individual events



Conclusion

- An original approach to abstract an execution trace

  $\Longrightarrow$ A great simplification of trace exploration

- An efficient greedy algorithm that automatically find interesting blocks

  $\Longrightarrow$ A significant reduction of the execution trace volume

Further directions

1. Automatic labelling
2. Multicore traces
3. Beyond *Coverage rank* function

Questions

Thank you for your kind attention