Using SMT Solving for the Lookup of Infeasible Paths in Binary Programs

University of Toulouse

Workshop WCET, Lünd, 2015

by

Jordy Ruiz and Hugues Cassé

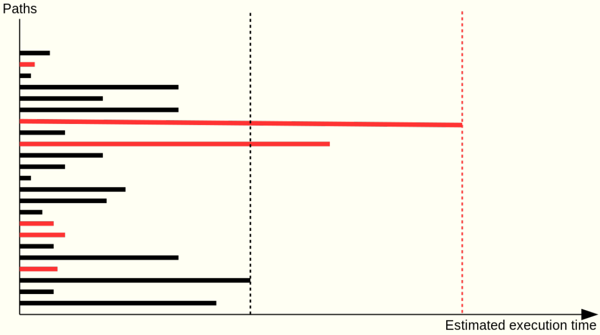
July 7, 2015

Contents

Context

Improving the WCET estimation

- Find a safe, *tight* timing bound
- Infeasible paths are the main source of overstimation in WCET computation



- Identifying infeasible paths refines the WCET estimation

Working on binary programs

Working directly on binaries is harder

- low expressivity of machine instructions
- larger size of program
- loosely typed registers
- obscure structure of data in memory
- ...

Working on binary programs

Working directly on binaries is harder but is more adapted:

- low expressivity of machine instructions
- larger size of program
- loosely typed registers
- obscure structure of data in memory
- ...

- not mapping properties from source to binaries
- independent of compiler
- available source libraries are not required
- easy injection in WCET computation

Working on binary programs

Working directly on binaries is harder but is more adapted:

- low expressivity of machine instructions
- larger size of program
- loosely typed registers
- obscure structure of data in memory
- ...

- short-circuit condition evaluation

```
if (x && a)
        /* ... */
if (x && b)
        /* ... */
```

Working on binary programs

Working directly on binaries is harder but is more adapted:

- low expressivity of machine instructions
- larger size of program
- loosely typed registers
- obscure structure of data in memory
- ...

- short-circuit condition evaluation

```
if (x)
    if (a)
        /* ... */
if (x)
    if (b)
        /* ... */
```

Analysing the semantics of a binary program

Semantic instructions

Architecture dependent?

Semantic instructions

Architecture dependent? No!

ARM machine instructions

ADD r3, r3, #1

OTAWA *semantic instructions*

seti t1, 1
add r3, r3, t1

Semantic instructions

Architecture dependent? No!

ARM machine instructions	OTAWA *semantic instructions*
ADD r3, r3, #1	seti t1, 1 add r3, r3, t1

STMDB sp!, {r4, lr}	seti t2, 4
	seti t1, 8
	sub t1, r13, t1
	set t3, t1
	store r4, t1, *int64*
	add t1, t1, t2
	store r14, t1, *int64*
	add t1, t1, t2
	set r13, t3

Abstract interpretation

- Maintain an abstract program state for each path
- Top-to-bottom analysis
- Inline calls
- Program states are represented by a *conjunction* of predicates

$$\gamma(\bigwedge \phi_i) = \{x \in S \mid \bigwedge \phi_i(s)\}$$

Updating the abstract program state

- $r_{13} = SP_0 + 0$
- $r_3 = r_1$

(initial state)

SP_0 is the initial value of the stack pointer

Updating the abstract program state

ADD r3, r3, #1

- $r_{13} = SP_0 + 0$
- $r_3 = r_1$

Updating the abstract program state

ADD r3, r3, #1
seti t1, 1

- $r_{13} = SP_0 + 0$
- $r_3 = r_1$
- $t_1 = 1$

Updating the abstract program state

ADD r3, r3, #1

seti t1, 1

add r3, r3, t1

- $r_{13} = SP_0 + 0$
- $r_3 - t_1 = r_1$
- $t_1 = 1$

Updating the abstract program state

ADD r3, r3, #1

    seti t1, 1
    add r3, r3, t1

- $r_{13} = SP_0 + 0$
- $r_3 - 1 = r_1$
- $t_1 = 1$

Updating the abstract program state

ADD r3, r3, #1

    seti t1, 1

    add r3, r3, t1

STMDB sp!, {r4, lr}

- $r_{13} = SP_0 + 0$
- $r_3 - 1 = r_1$
- ~~$t_1 = 1$~~

Updating the abstract program state

ADD r3, r3, #1

    seti t1, 1

    add r3, r3, t1

STMDB sp!, {r4, lr}

    seti t2, 4

    seti t1, 8

- $r_{13} = SP_0 + 0$
- $r_3 - 1 = r_1$
- $t_2 = 4$
- $t_1 = 8$

Updating the abstract program state

ADD r3, r3, #1

    seti t1, 1
    add r3, r3, t1

STMDB sp!, {r4, lr}

    seti t2, 4
    seti t1, 8
    sub t1, r13, t1

- $r_{13} = SP_0 + 0$
- $r_3 - 1 = r_1$
- $t_2 = 4$
- $t_1 = SP_0 - 8$

Updating the abstract program state

ADD r3, r3, #1

    seti t1, 1
    add r3, r3, t1

STMDB sp!, {r4, lr}

    seti t2, 4
    seti t1, 8
    sub t1, r13, t1
    set t3, t1

- $r_{13} = SP_0 + 0$
- $r_3 - 1 = r_1$
- $t_2 = 4$
- $t_1 = SP_0 - 8$
- $t_3 = SP_0 - 8$

Updating the abstract program state

```
ADD r3, r3, #1
      seti t1, 1
      add r3, r3, t1
STMDB sp!, {r4, lr}
      seti t2, 4
      seti t1, 8
      sub t1, r13, t1
      set t3, t1
      store r4, t1, int64
```

- $r_{13} = SP_0 + 0$
- $r_3 - 1 = r_1$
- $t_2 = 4$
- $t_1 = SP_0 - 8$
- $t_3 = SP_0 - 8$
- $[SP_0 - 8] = r_4$

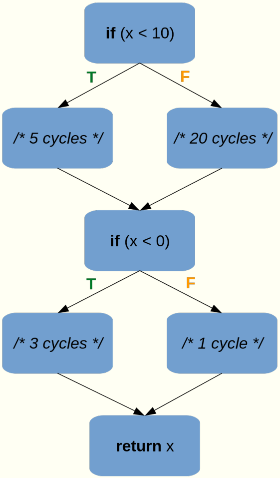
Updating the abstract program state

ADD r3, r3, #1
    seti t1, 1
    add r3, r3, t1
STMDB sp!, {r4, lr}
    seti t2, 4
    seti t1, 8
    sub t1, r13, t1
    set t3, t1
    store r4, t1, *int64*
    add t1, t1, t2
    store r14, t1, *int64*
    set r13, t3

- $r_{13} = SP_0 - 8$
- $r_3 - 1 = r_1$
- ~~$t_2 = 4$~~
- ~~$t_1 = SP_0 - 4$~~
- ~~$t_3 = SP_0 - 8$~~
- $[SP_0 - 8] = r_4$
- $[SP_0 - 4] = r_{14}$

Finding infeasible paths

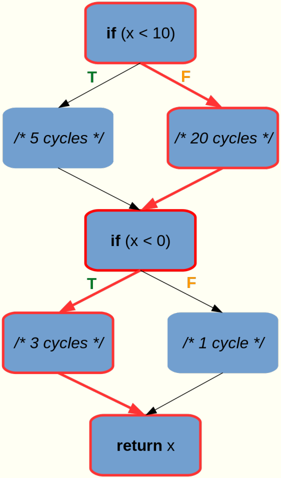
Example of infeasible path



Accounting for all 4 paths,
WCET = 23 cycles + ...

But:
$\neg(x < 10) \wedge (x < 0) \models \bot$

Without the infeasible path, WCET = 21 cycles + ...



Predicates

Example of a simple abstract program state:

- $r_0 > 8$

Predicates

Example of a simple abstract program state:

- $r_0 > 8$
- $[0\text{x}8008] = 0$

Predicates

Example of a simple abstract program state:

- $r_0 > 8$
- $[0\mathrm{x}8008] = 0$
- $r_{13} = SP_0 - 24$

SP_0 remains constant throughout the program

Predicates

Example of a simple abstract program state:

- $r_0 > 8$
- $[0\mathbf{x}8008] = 0$
- $r_{13} = SP_0 - 24$
- $r_0 = r_1$

SP_0 remains constant throughout the program

Predicates

Example of a simple abstract program state:

- $r_0 > 8$
- $[0\mathrm{x}8008] = 0$
- $r_{13} = SP_0 - 24$
- $r_0 = r_1$
- $r_1 = 0$

SP_0 remains constant throughout the program

Predicates

Example of a simple abstract program state:

- $r_0 > 8$
- $[0\text{x}8008] = 0$
- $r_{13} = SP_0 - 24$
- $r_0 = r_1$
- $r_1 = 0$

This program state is unsatisfiable!
("UNSAT")

$\Rightarrow$ The current path is infeasible

- Example:
  $0 \to 1 \to 3 \to 4 \to 5 \to 6 \to 8 \to 9$

Labelled predicates

Label predicates by the basic block(s) that generated them:

- $r_0 > 8^{(1,5)}$
- $[0\text{x}8008] = 0^{(3)}$
- $r_{13} = SP_0 - 24^{(4)}$
- $r_0 = [SP_0 - 16]^{(9)}$
- $[SP_0 - 16] = 0^{(9)}$

- Full infeasible path:
  $0 \to 1 \to 3 \to 4 \to 5 \to 6 \to 8 \to 9$

Labelled predicates

Label predicates by the basic block(s) that generated them:

- $r_0 > 8^{(1,5)}$
- $[0\mathrm{x}8008] = 0^{(3)}$
- $r_{13} = SP_0 - 24^{(4)}$
- $r_0 = [SP_0 - 16]^{(9)}$
- $[SP_0 - 16] = 0^{(9)}$

- Full infeasible path:
  $0 \to 1 \to 3 \to 4 \to 5 \to 6 \to 8 \to 9$
- Minimized infeasible path:
  $1 \to 5 \to 9$

SMT solving

Satisfiability **M**odulo **T**heories solver:

- a SAT solver enhanced with multiple theories:
  - Rational/Integer/Booleans
  - Arrays
  - BitVectors
  - ...

  $\Rightarrow$ We use Quantifier-Free Linear Integer Arithmetic

SMT solving

Satisfiability **M**odulo **T**heories solver:

- a SAT solver enhanced with multiple theories:
  - Rational/Integer/Booleans
  - Arrays
  - BitVectors
  - ...

  $\Rightarrow$ We use Quantifier-Free Linear Integer Arithmetic
- receives a list of assertions then seeks a model (satisfiability check)

UNSAT cores

Some SMT solvers feature UNSAT cores:

- Triggered when a system is proven unsatisfiable
- Gives a minimal set of assertions that preserves unsatisfiability

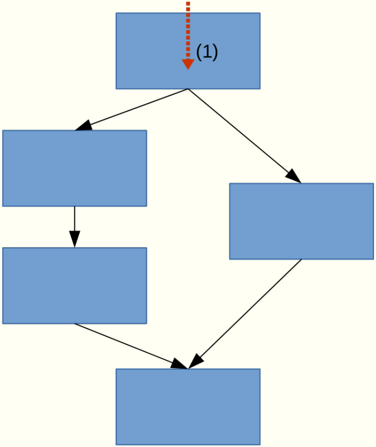
UNSAT cores

Some SMT solvers feature UNSAT cores:

- Triggered when a system is proven unsatisfiable
- Gives a minimal set of assertions that preserves unsatisfiability
- Example:
  - $r_0 > 8$
  - $[0\text{x}8008] = 0$
  - $r_{13} = SP_0 - 24$
  - $r_0 = [SP_0 - 16]$
  - $[SP_0 - 16] = 0$

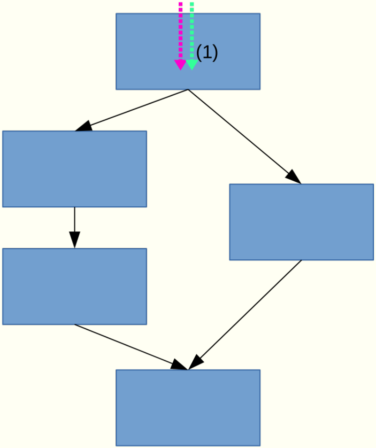
Macro analysis

- Working List algorithm: *"only process a Basic Block if all paths leading to it have been processed"*



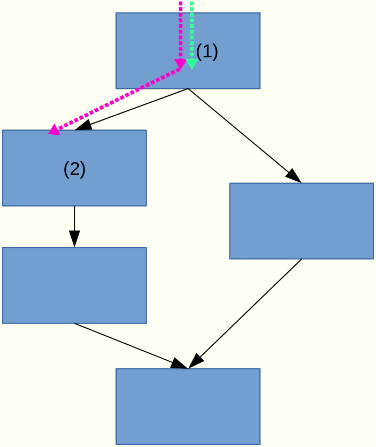
Macro analysis

- Working List algorithm: *“only process a Basic Block if all paths leading to it have been processed”*



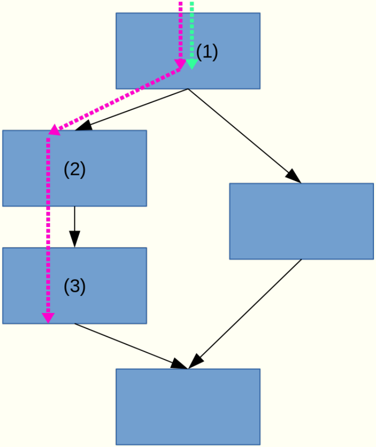
Macro analysis

- Working List algorithm: *“only process a Basic Block if all paths leading to it have been processed”*



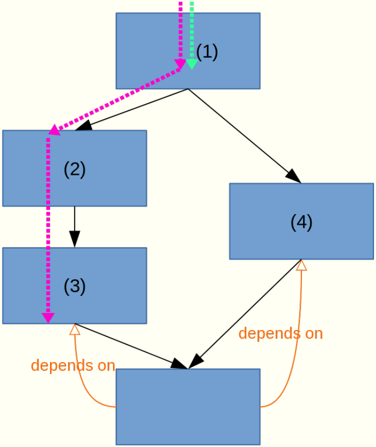
Macro analysis

- Working List algorithm: *“only process a Basic Block if all paths leading to it have been processed”*



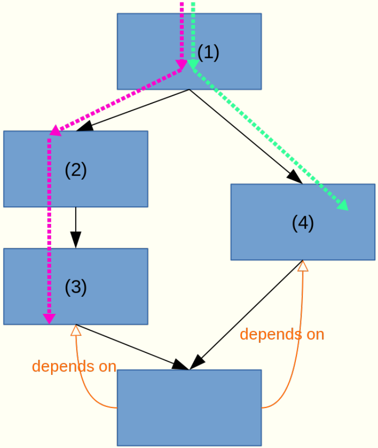
Macro analysis

- Working List algorithm: *"only process a Basic Block if all paths leading to it have been processed"*



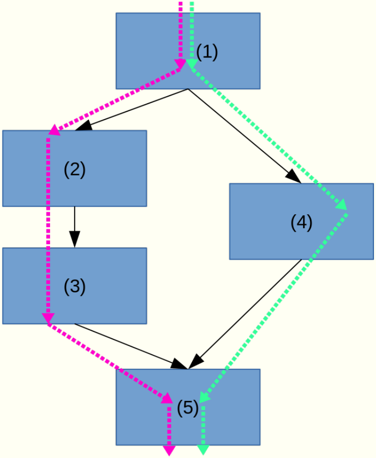
Macro analysis

- Working List algorithm: *"only process a Basic Block if all paths leading to it have been processed"*



Macro analysis

- Working List algorithm: *“only process a Basic Block if all paths leading to it have been processed”*

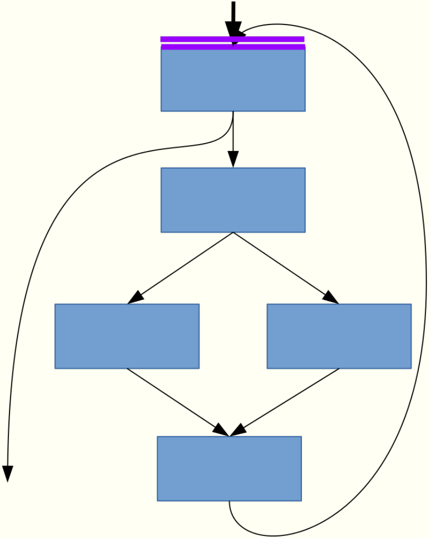


Macro analysis

- Working List algorithm: *“only process a Basic Block if all paths leading to it have been processed”*
- Loops:
  - Iterate and merge with previous state until fixpoint is reached
  - When a fixpoint is reached, enable SMT checks to find infeasible paths valid at every iteration

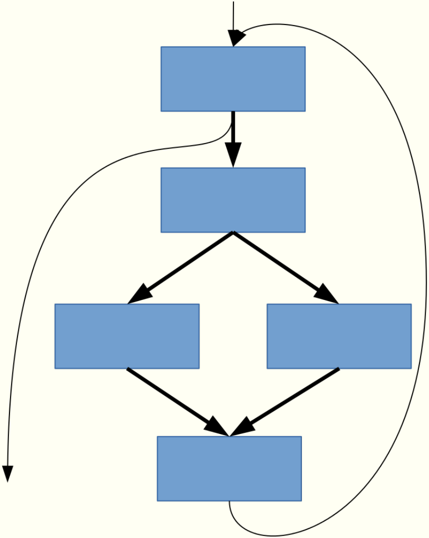
Processing loops

- To start things off, merge all incoming states



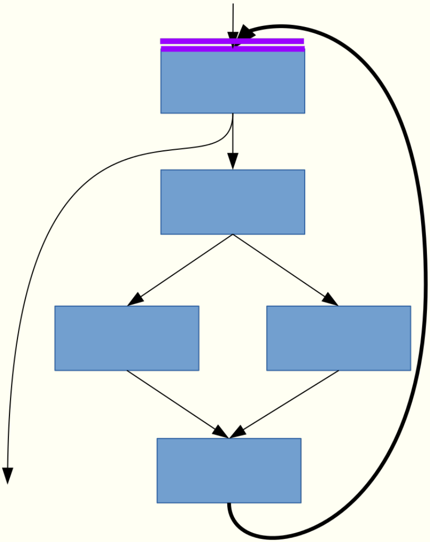
Processing loops

- To start things off, merge all incoming states
- Parse the loop body normally



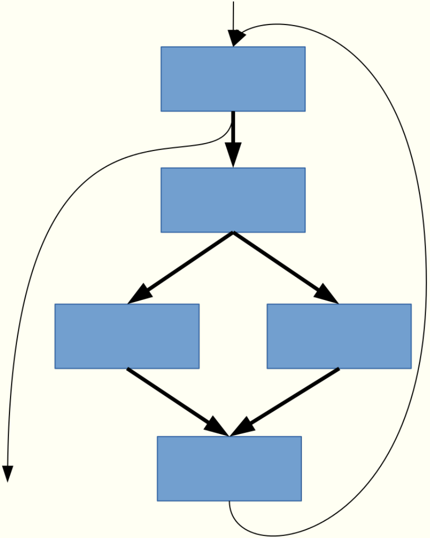
Processing loops

- To start things off, merge all incoming states
- Parse the loop body normally
- Then merge with the previous state



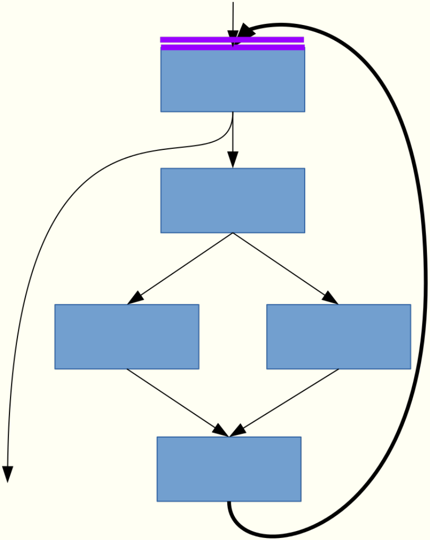
Processing loops

- To start things off, merge all incoming states
- Parse the loop body normally
- Then merge with the previous state
- Reapeat until fixpoint is reached



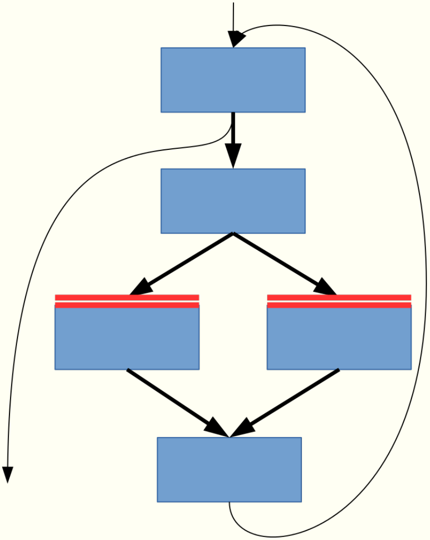
Processing loops

- To start things off, merge all incoming states
- Parse the loop body normally
- Then merge with the previous state
- Reapeat until fixpoint is reached



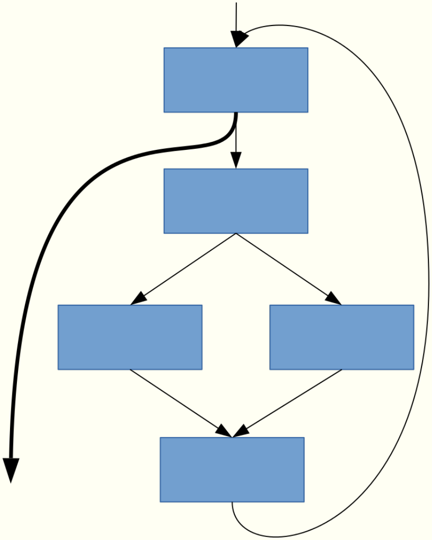
Processing loops

- To start things off, merge all incoming states
- Parse the loop body normally
- Then merge with the previous state
- Reapeat until fixpoint is reached
- Do SMT checks



Processing loops

- To start things off, merge all incoming states
- Parse the loop body normally
- Then merge with the previous state
- Reapeat until fixpoint is reached
- Do SMT checks
- Exit the loop



Merging

A very rough merging algorithm: predicate set intersection

- $r_{13} = SP - 4$
- $r_0 = [SP - 8]$
- $r_1 = 0$
- $r_2 = 0$
- $[0\text{x}8008] = 16$

- $r_{13} = SP - 4$
- $r_0 = [SP - 8]$
- $r_1 = 1$
- $r_2 > 0$

becomes:

Merging

A very rough merging algorithm: predicate set intersection

- $r_{13} = SP - 4$
- $r_0 = [SP - 8]$
- $r_1 = 0$
- $r_2 = 0$
- $[0x8008] = 16$

- $r_{13} = SP - 4$
- $r_0 = [SP - 8]$
- $r_1 = 1$
- $r_2 > 0$

becomes:

- $r_{13} = SP - 4$
- $r_0 = [SP - 8]$

Mälardalen benchmarks

Benchmark	BB (#)	Time (s)	IPs found with minimization	without minimization
Small benchmarks (no merging required)				
ndes	57	0.267	0	0
expint	70	0.748	14	34
edn	75	0.537	2	2
prime	118	4.368	22	43
compress	122	1.801	10	19
select	136	45.598	4	8
qsortexam	155	28.201	9	12
adpcm	323	0.074	3	3
Large benchmarks (merging required)				
ud	153	17.477	6	23
minver	449	188.339	4	16
statemate	453	193.849	16	22
ludcmp	632	143.088	11	510
nsichneu	754	250.385	4586	8620
qurt	2777	773.805	3	3
lms	3098	915.434	259	2376
fft1	6123	2223.125	25	815

Conclusion

Closing Thoughts

- Working directly on the binaries

Closing Thoughts

- Working directly on the binaries
- Outputting short infeasible paths
  - Reduces complexity of analyses that exploit infeasible paths

Closing Thoughts

- + Working directly on the binaries
- + Outputting short infeasible paths
  - Reduces complexity of analyses that exploit infeasible paths
- — Brutal state merging

Closing Thoughts

- + Working directly on the binaries
- + Outputting short infeasible paths
  - Reduces complexity of analyses that exploit infeasible paths
- – Brutal state merging
- – Time calculation explosion

Future works

- Program slicing

Future works

- Program slicing
- Looking for loop invariants

Future works

- Program slicing
- Looking for loop invariants
- Experiment with other SMT theories than QF-LIA

Future works

- Program slicing
- Looking for loop invariants
- Experiment with other SMT theories than QF-LIA
- Experimentations to estimate the impact on the WCET

Questions?

Mälardalen benchmarks

			Inf. paths found with minimization			w/o minimization
Benchmark	BB (#)	Time (s)	1 edge	Minimized	Non-minimized	Non-minimized
SMALL BENCHMARKS (NO MERGING REQUIRED)						
ndes	57	0.267	0	0	0	0
expint	70	0.748	4	5	5	34
edn	75	0.537	2	0	0	2
prime	118	4.368	2	8	12	43
compress	122	1.801	2	8	0	19
select	136	45.598	0	4	0	8
qsortexam	155	28.201	2	4	2	12
adpcm	323	0.074	3	0	0	3
LARGE BENCHMARKS (MERGING REQUIRED)						
ud	153	17.477	3	3	0	23
minver	449	188.339	4	0	0	16
statemate	453	193.849	0	16	0	22
ludcmp	632	143.088	5	6	0	510
nsichneu	754	250.385	0	1352	3234	8620
qurt	2777	773.805	3	0	0	3
lms	3098	915.434	26	22	211	2376
fft1	6123	2223.125	0	25	0	815