Compiler Help for Binary Manipulation Tools

Tugrul Ince, Jeffrey K. Hollingsworth

University of Maryland, College Park

28 August 2012, Rhodes, Greece

Parsing Binary Files

- Binary analysis is common for
  - Performance modeling
  - Computer security
  - Maintenance
  - Binary modification

- Parsing: first step in most binary analyses
  - Not straight-forward
  - Time consuming

Difficulties in Parsing

- Distinguishing code and data
- Disassembly is tricky
  - Finding instruction boundaries
    - Variable-length instruction set architectures
  - Identifying functions
- Building Control Flow Graphs
  - Identify Basic Block boundaries
    - Basic Block: Sequence of instructions with a single entry and single exit
  - Identify edges between basic blocks

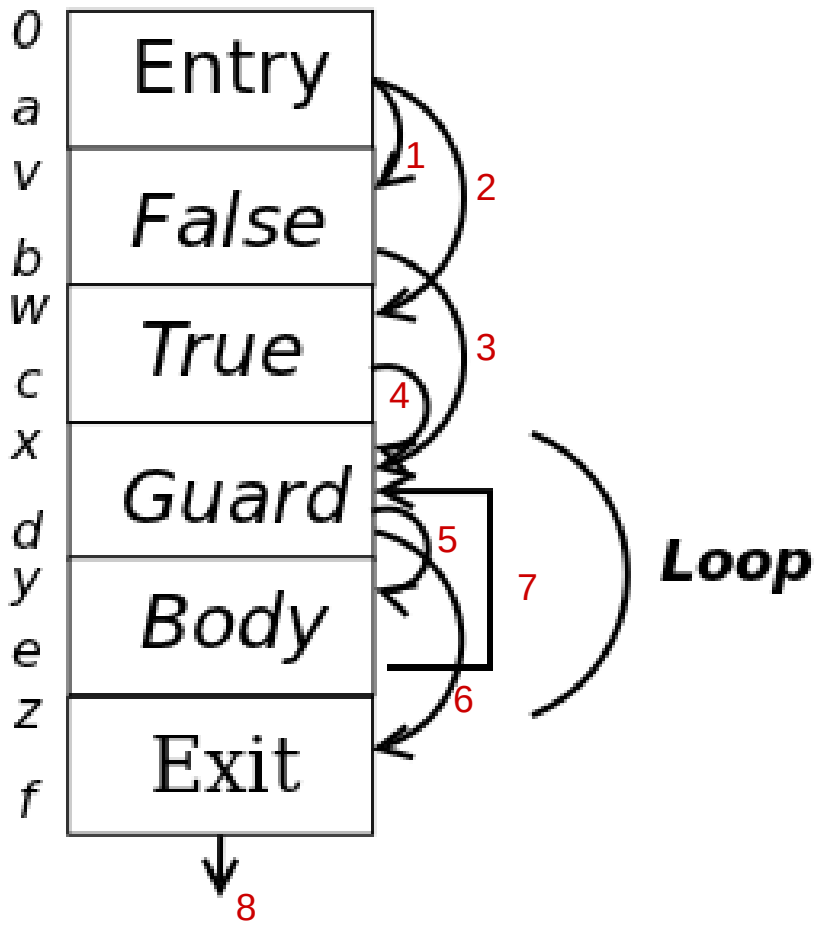
Objective

- Improve parsing speed
- Store more data in binary files
  - Basic block locations
  - Edge information (source, target, type)
- Binary analysis tools read this extra information
  - Create abstractions for:
    - Basic blocks
    - Edges
    - Finally Control Flow Graphs (CFGs)

Compiler Assistance for Parsing

- Developed new compilation mechanism
  - Wrappers for gnu compiler suite (gcc/g++)
  - Transparent to the end user
- Augments binary files with tables
  - Basic Block Table
  - Edge Table
- Analyze intermediate assembly files
  - Generate information about basic blocks and edges
  - Store in a section that is not loaded at runtime

Basic Block and Edge Tables



First Instruction	Last Instruction
0	a
v	b
w	c
x	d
y	e
z	f

Basic Block Table

	Source	Target	Edge Type
1	0	v	Fall-through
2	0	w	Conditional
3	v	x	Unconditional
4	w	x	Fall-through
5	x	y	Fall-through
6	x	z	Conditional
7	y	x	Unconditional
8	z	N/A	Return

Edge Table

Assembly Modification

a) Original Assembly Code

```
  ...
  type  foo, @function
foo:
  ...
  ...
  .size foo, foo_end - foo
```

b) Augmented Assembly Code

```
  ...
  type  foo, @function
foo:
.foo_<file_name>:
  ...
  ...
  .size foo, . - foo
  .section .edge_info
.BLK_foo_BLK__<file_name>
  ...
  ...
  .size .BLK_foo_BLK__<file_name>, \
     . - .BLK_foo_BLK__<file_name>
```

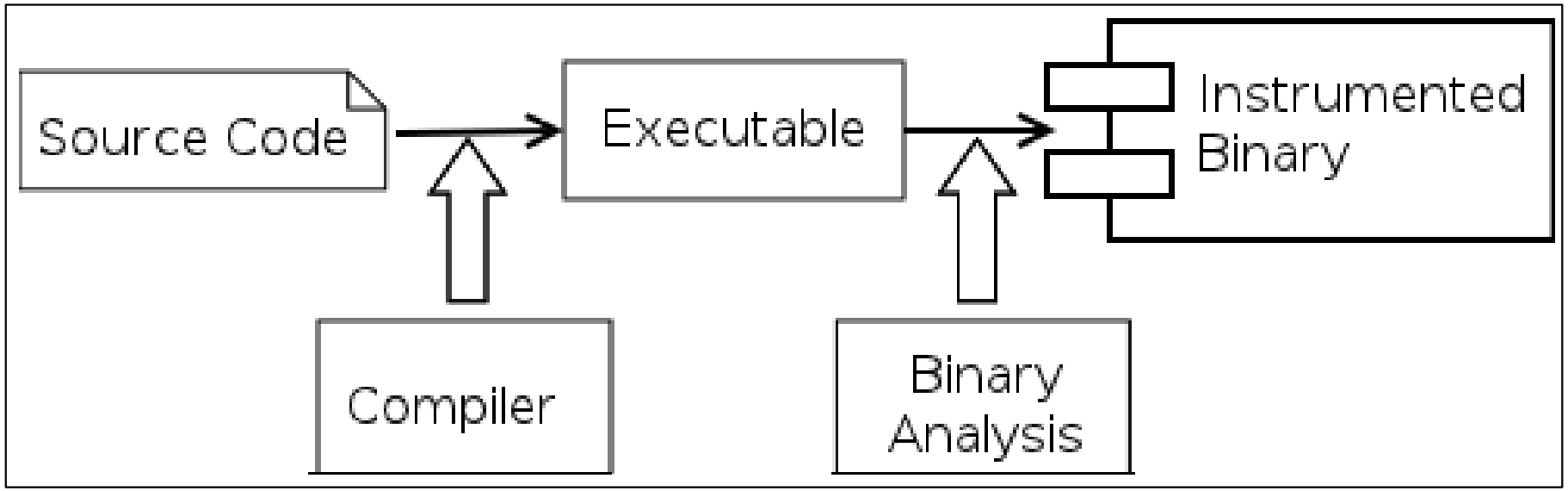
- Function Model
  - Block of code
  - "type … @function"
  - ".size …"
- Modifications
  - Add Basic Block and Edge Tables
  - Add shadow symbol

Merge Duplicate Functions

- Weak functions are merged by linker
  - Functions included multiple times
  - Binary code might slightly differ
  - Only one weak function survives

- Tables cannot be merged
  - Need to uniquely match functions and tables
  - Use shadow symbol in function to extract file name
  - Use file name and function name to identify tables

Decoupled Compilation and Analysis



- Binary analysis tools operate on executables directly
  - No interaction with the compiler

Basic Block and CFG Reconstruction

- Parsing a function involves:
  - Finding the shadow symbol stored in the function
    - File name is extracted
  - Locating Basic Block and Edge Tables with the function name and file name pair
  - Reading in the tables
  - Adding function start address to offsets
  - Creating basic block and edge abstractions

- No need to parse individual instructions

Dyninst: Dynamic and Static Analysis

- Dyninst is used for:
  - Profiling and debugging
  - Performance measurement
  - Malware analysis
- Dyninst provides binary analysis and modification
  - Runtime instrumentation
  - Binary rewriting
- Platform-independent

Dyninst: Dynamic and Static Analysis

- Static analysis capabilities
  - Control flow graph (CFG) generation
  - Iterate over instructions
  - Modify symbols, add sections, etc.

- Dynamic instrumentation capabilities well-known
  - Add/remove function calls
  - Link with new shared libraries
  - Add new code to almost anywhere in original code

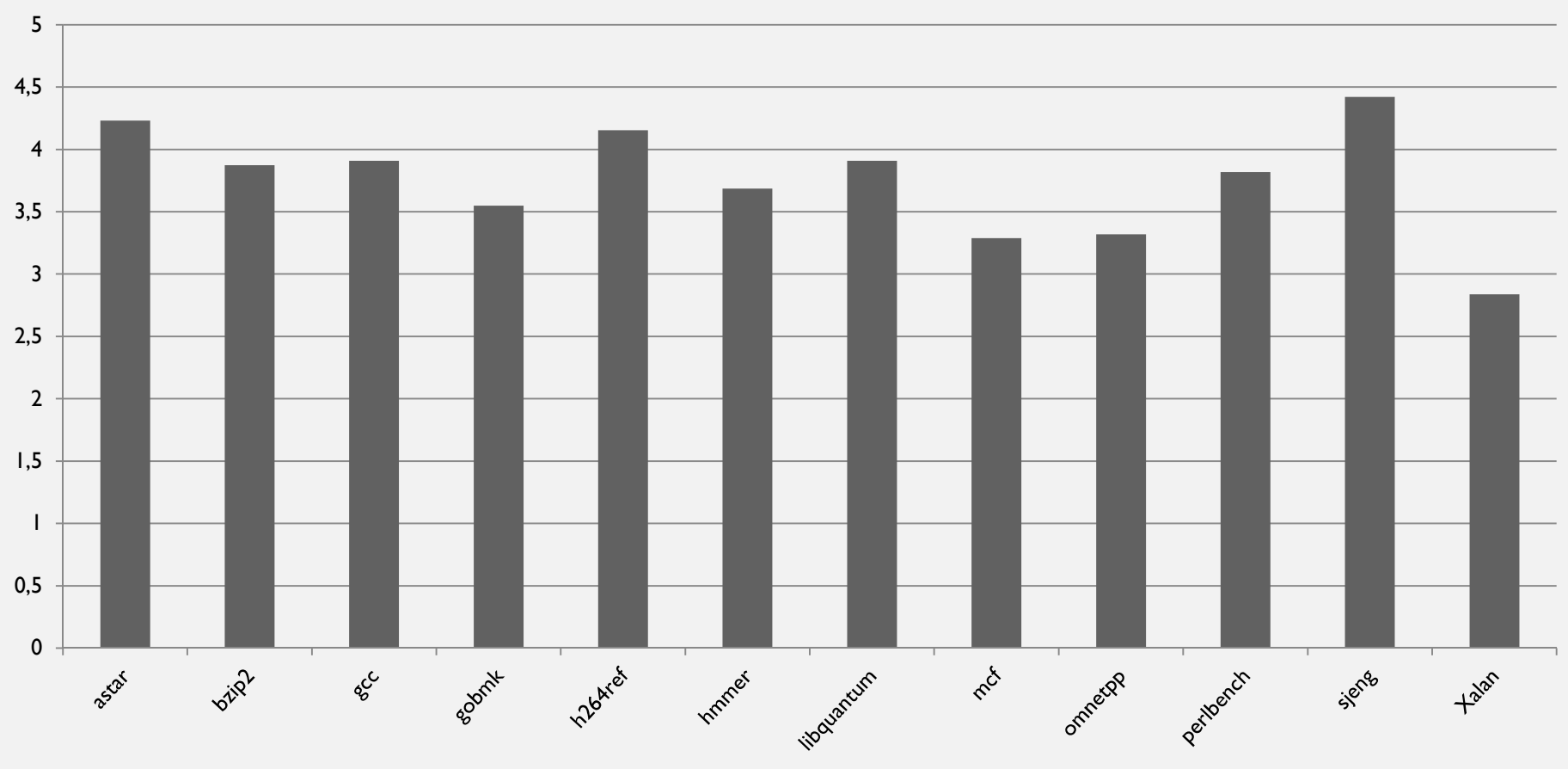
Evaluation

- Benchmarks
  - SPEC CINT2006
  - PETSc snes package
  - Firefox (v. 9.0.1)

- Methodology
  - Executed running time experiments 5 times
  - Reporting mean

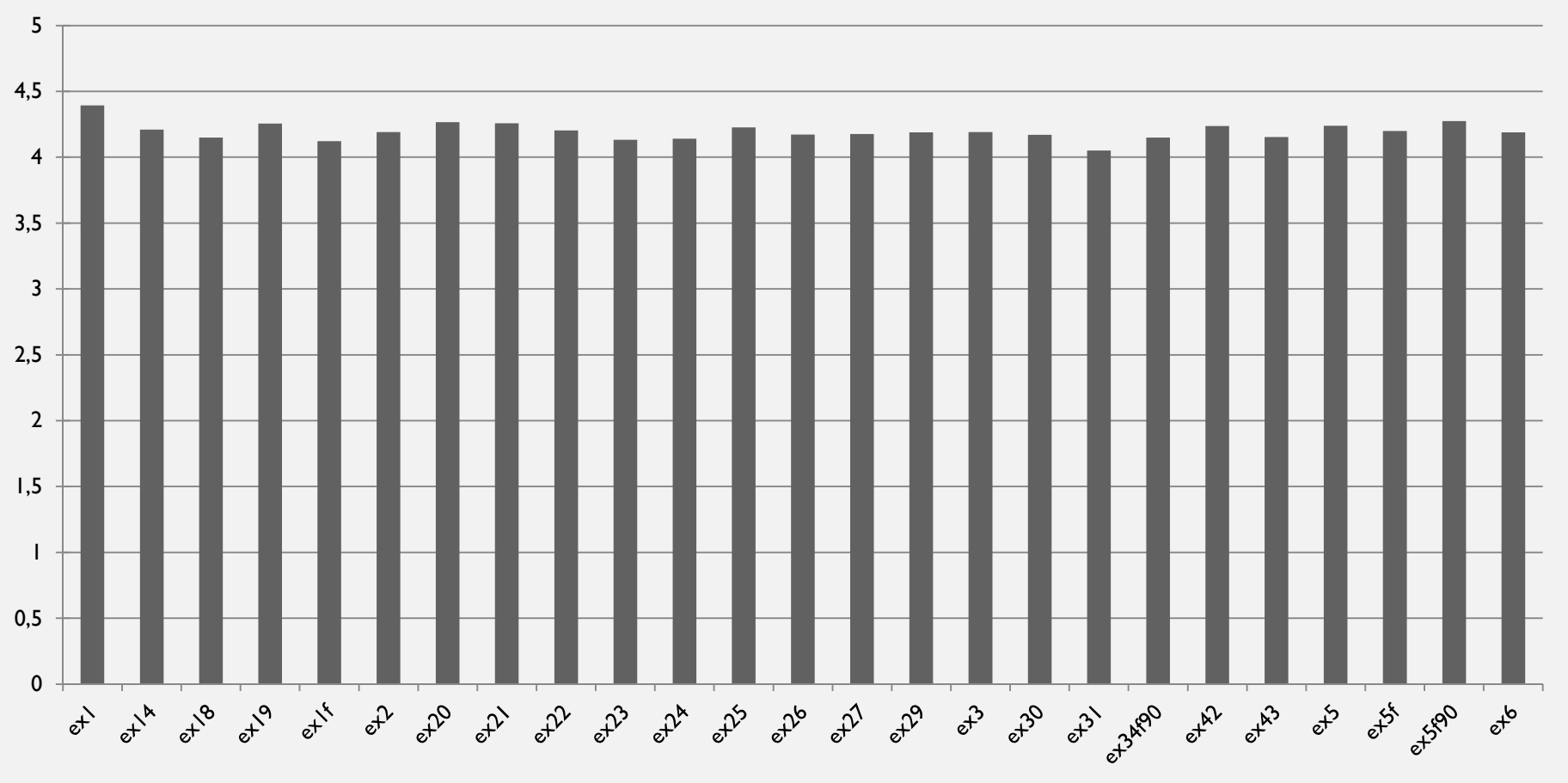
Parsing Speedup

SPEC CINT2006



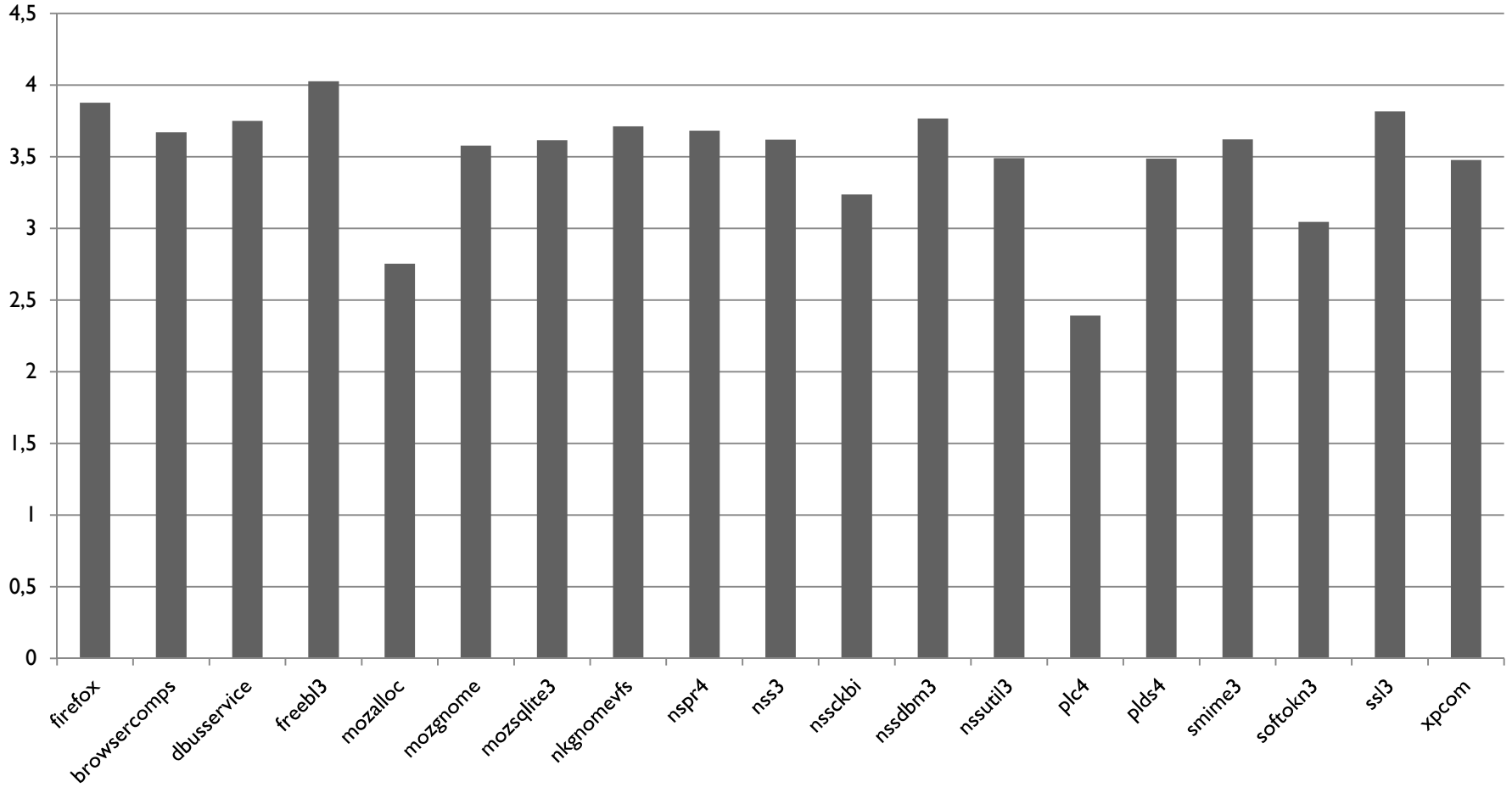
Parsing Speedup

PETSc snes Package



Parsing Speedup

Firefox v. 9.0.1



Build Time Metrics

	File Size		Compilation Time
	Without Debug	With Debug	
SPEC CINT2006	2.21x	1.38x	1.25x
PETSc	1.50x	1.09x	1.32x
Firefox	1.17x	1.21x	1.13x
OVERALL	1.63x	1.23x	1.23x

- File size increases on disk
  - Not reflected to memory footprint
- Small increase in compilation time
  - One time cost
  - Not reflected to running time performance

Runtime Metrics

	Memory Footprint	Running Time
SPEC CINT2006	1.00x	0.97x
PETSc	1.00x	0.95x
Firefox	1.00x	0.94x
OVERALL	1.00x	0.95x

- Virtually no change in runtime metrics
  - Memory requirement is almost constant
  - Change in running time is within noise

Limitations / Future Work

- Hand-written assembly
  - When branches use offsets in assembly
- 2n more symbols (n: number of functions)
- Compilation takes 23% more time
  - Integrate compilation mechanism into gcc
- File size increases
  - Compress tables – about 78% compression ratio

Conclusion

- Developed a new compilation mechanism
  - Creates Basic Block and Edge Tables
  - Transparent to end user
- Dramatically improved binary parsing speed
  - On average 73% decrease in parsing time
  - No memory or runtime overhead