Compiler-Assisted Binary Parsing

Tugrul Ince
tugrul@cs.umd.edu

PD Week 2012
26 – 27 March 2012

Dyninst

University of Maryland

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

Objective

- Improve parsing speed and accuracy
- Store more data in binary files
  - Basic block locations
  - Edge information (source, target, type)
- Binary analysis tools read this extra information
  - Create basic block, edge, and finally CFG abstractions

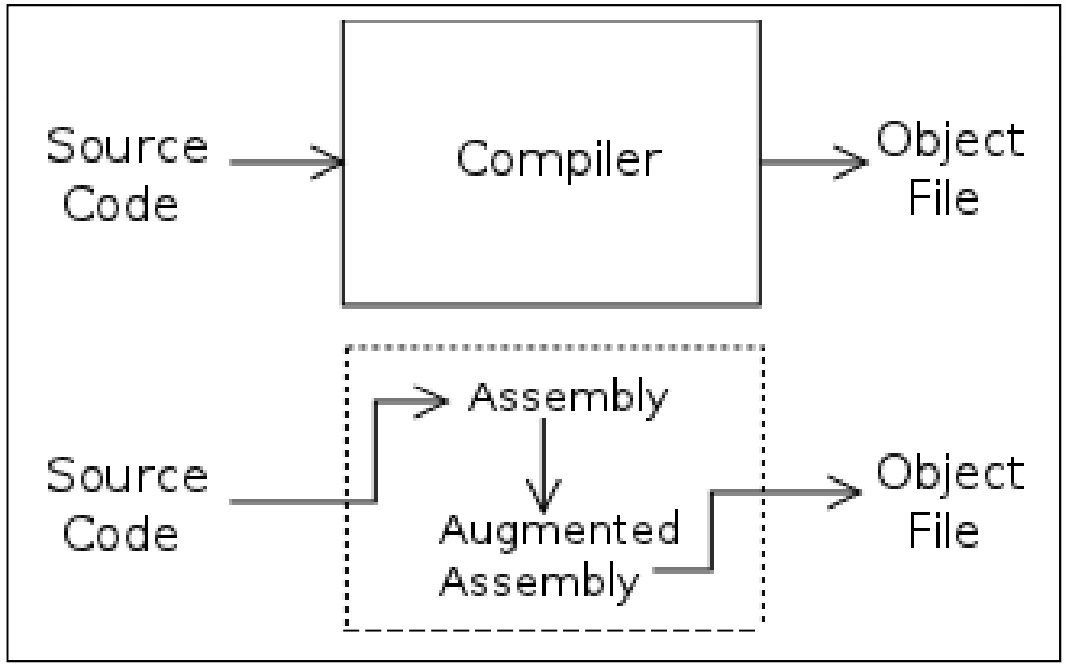
Difficulties in Parsing

- Distinguishing code and data
- Disassembly is tricky
  - Identifying functions
  - Finding instruction boundaries
    - Variable-length instruction set architectures
- Building Control Flow Graphs
  - Identify Basic Block boundaries
  - Identify edges between basic blocks

Compiler Assistance for Parsing

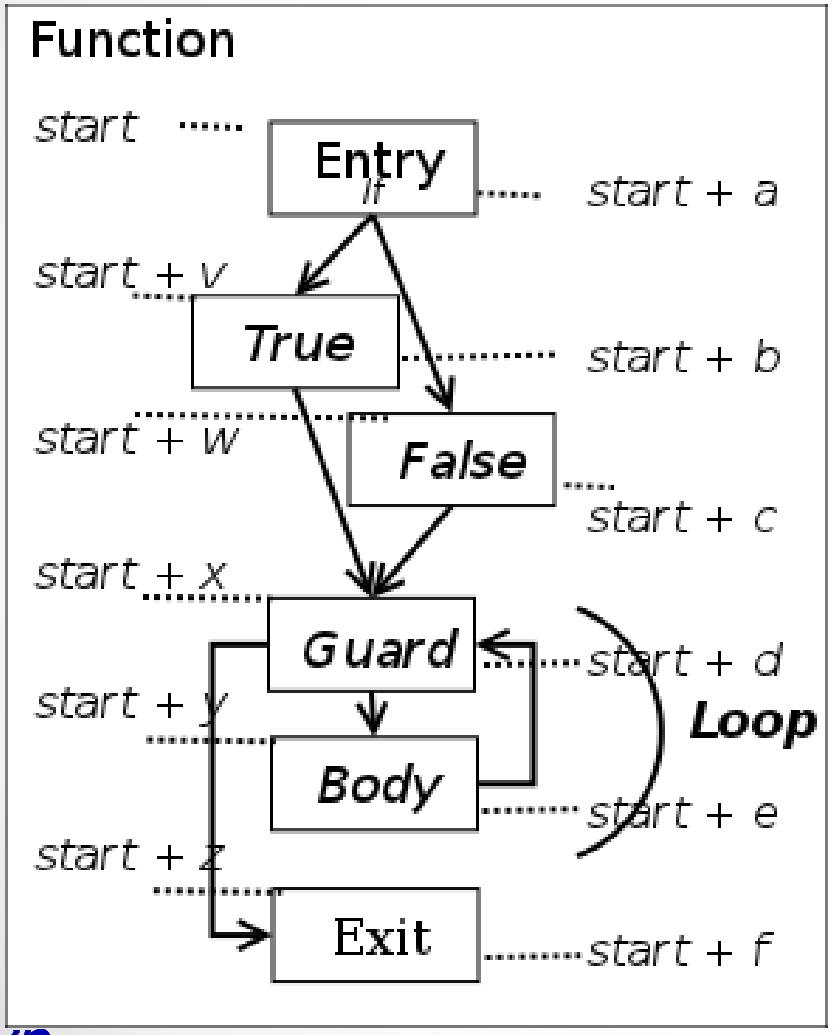
- Developed new compilation mechanism
  - Wrappers for GNU compiler suite (gcc/g++)
  - Transparent to the end user
- Support most standard flags
  - Pass flags to underlying system compiler
  - Intercept output flags (-c, -S, -o, etc.)
- Augments binary files with tables
  - Basic Block Table
  - Edge Table

Compiler Infrastructure



- Analyze intermediate assembly files
  - Generate information about basic blocks and edges
  - Store in a section that is not loaded at runtime

Basic Block - 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
0	v	Fall-through
0	w	Conditional
v	x	Unconditional
w	x	Fall-through
x	y	Fall-through
x	z	Conditional
y	x	Unconditional
z	N/A	Return

Edges Table

Dyninst

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

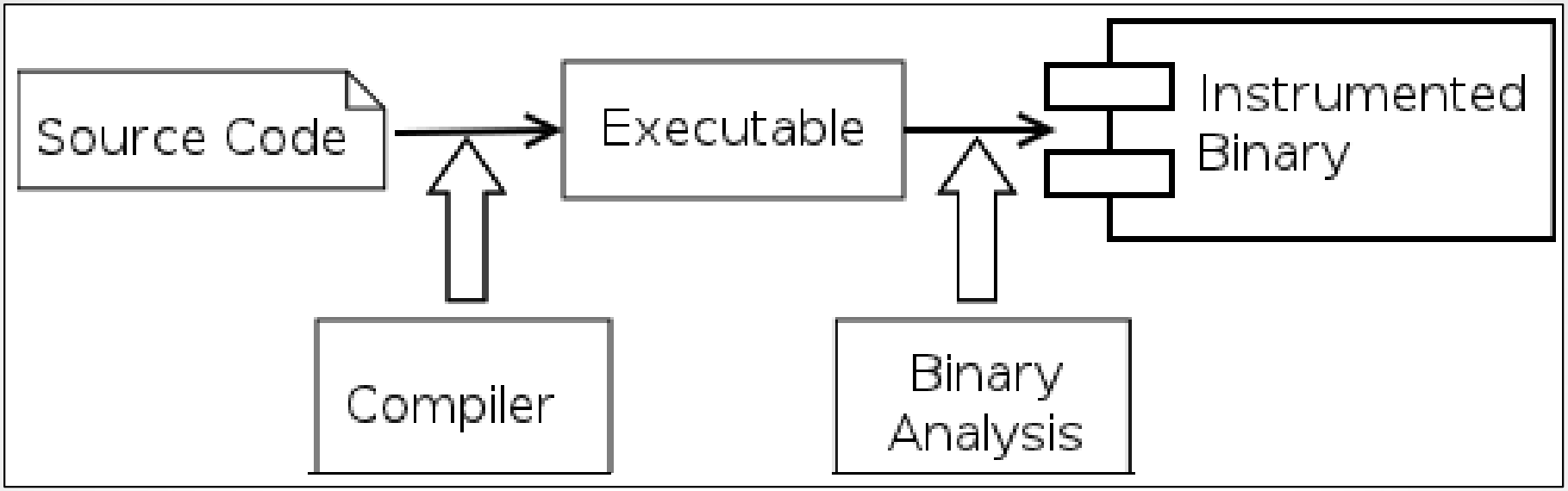
Dyninst

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

Reconstruction



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

Reconstruction

- Parsing a functions 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

Evaluation

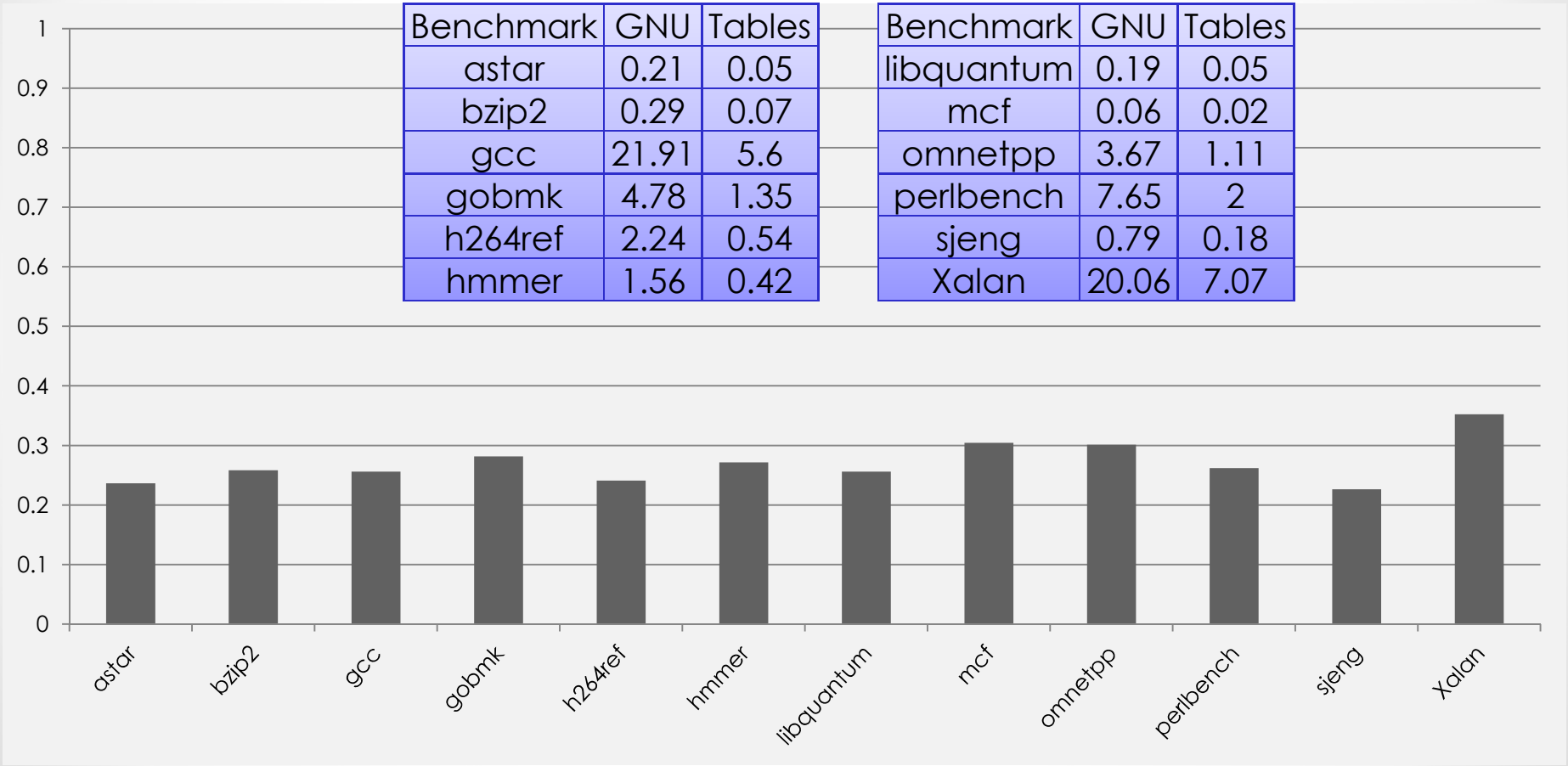
- Benchmarks
  - SPEC CINT2006
  - PETSc snes package
  - Firefox (v. 9.0.1)
- Systems
  - 64-bit Linux machines
  - server: 24-core Intel Xeon, 48 GB total memory
  - laptop: AMD Turion, 2 GB total memory
- Methodology
  - Executed running time experiments 5 times
  - Reporting mean

Normalized Parsing Time

SPEC CINT2006

Benchmark	GNU	Tables
astar	0.21	0.05
bzip2	0.29	0.07
gcc	21.91	5.6
gobmk	4.78	1.35
h264ref	2.24	0.54
hmmer	1.56	0.42

Benchmark	GNU	Tables
libquantum	0.19	0.05
mcf	0.06	0.02
omnetpp	3.67	1.11
perlbench	7.65	2
sjeng	0.79	0.18
Xalan	20.06	7.07



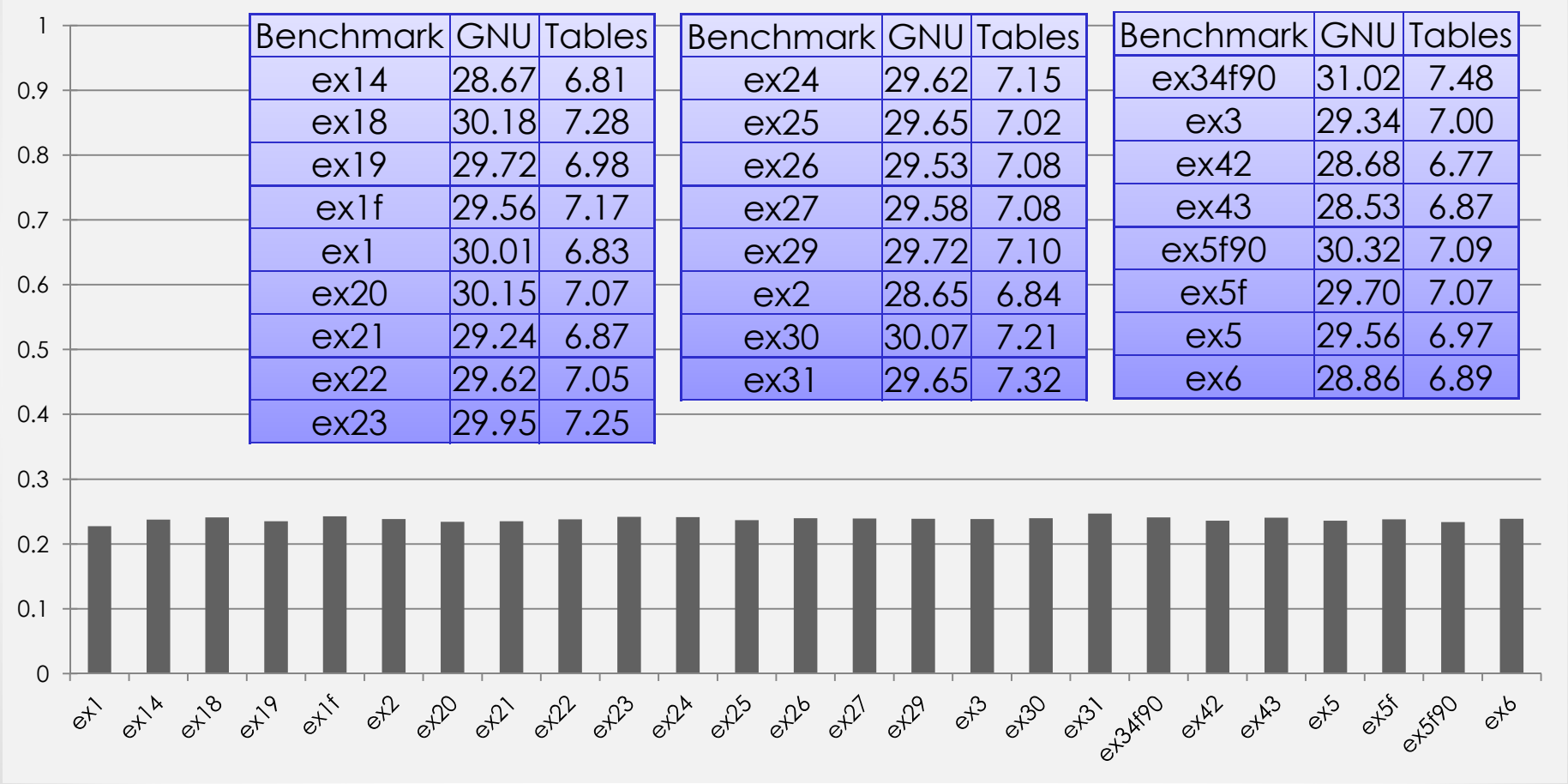
Normalized Parsing Time

PETSc snes Package

Benchmark	GNU	Tables
ex14	28.67	6.81
ex18	30.18	7.28
ex19	29.72	6.98
ex1f	29.56	7.17
ex1	30.01	6.83
ex20	30.15	7.07
ex21	29.24	6.87
ex22	29.62	7.05
ex23	29.95	7.25

Benchmark	GNU	Tables
ex24	29.62	7.15
ex25	29.65	7.02
ex26	29.53	7.08
ex27	29.58	7.08
ex29	29.72	7.10
ex2	28.65	6.84
ex30	30.07	7.21
ex31	29.65	7.32

Benchmark	GNU	Tables
ex34f90	31.02	7.48
ex3	29.34	7.00
ex42	28.68	6.77
ex43	28.53	6.87
ex5f90	30.32	7.09
ex5f	29.70	7.07
ex5	29.56	6.97
ex6	28.86	6.89



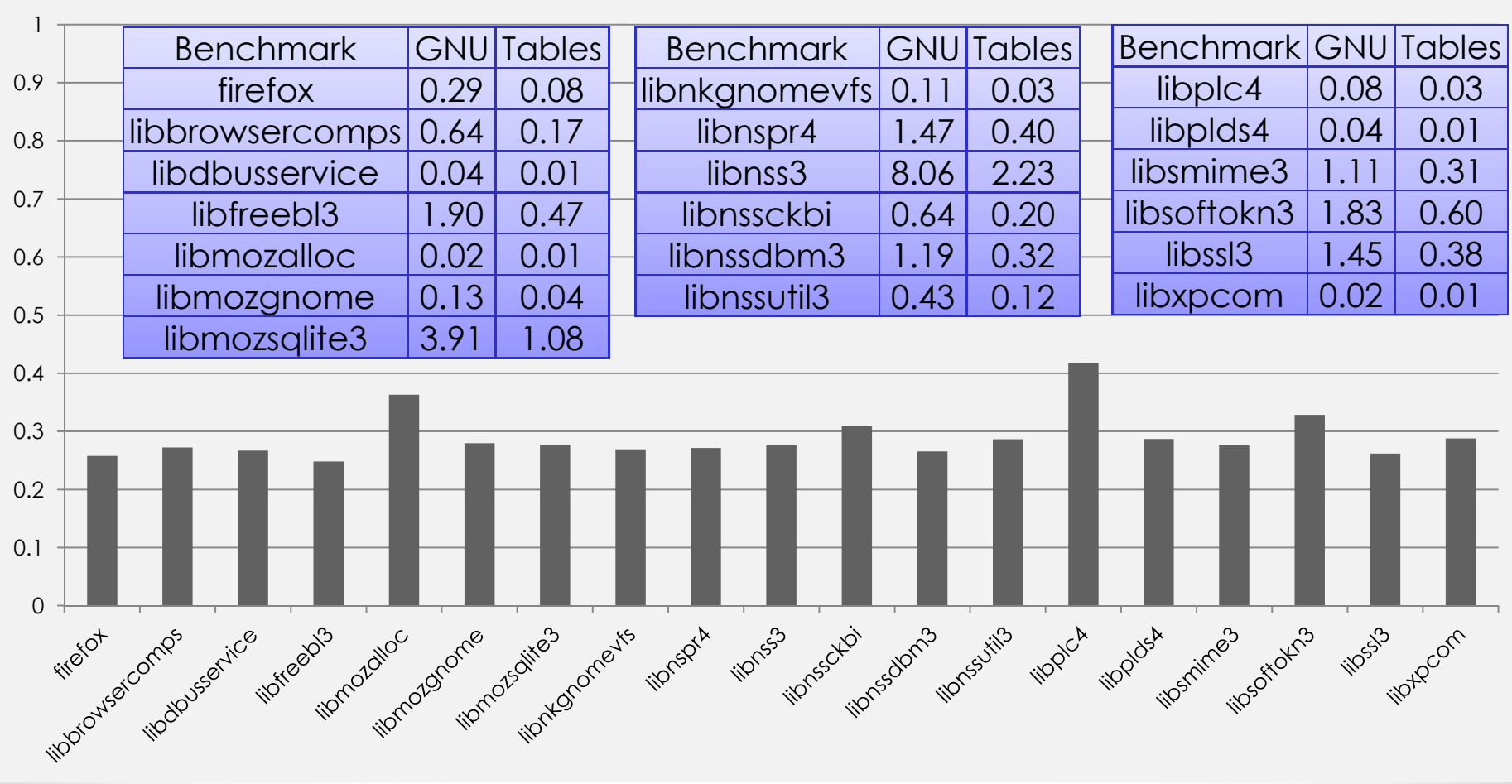
Normalized Parsing Time

Firefox Version 9.0.1

Benchmark	GNU	Tables
firefox	0.29	0.08
libbrowsercomps	0.64	0.17
libdbusservice	0.04	0.01
libfreebl3	1.90	0.47
libmozalloc	0.02	0.01
libmozgnome	0.13	0.04
libmozsqlite3	3.91	1.08

Benchmark	GNU	Tables
libnkgnomevfs	0.11	0.03
libnspr4	1.47	0.40
libnss3	8.06	2.23
libnssckbi	0.64	0.20
libnssdbm3	1.19	0.32
libnssutil3	0.43	0.12

Benchmark	GNU	Tables
libplc4	0.08	0.03
libplds4	0.04	0.01
libsmime3	1.11	0.31
libsoftokn3	1.83	0.60
libssl3	1.45	0.38
libxpcom	0.02	0.01



Build Time Metrics

	File Size		
	Without Debug	With Debug	Compilation Time
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 increase 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
- Hard to measure Firefox running time
  - No workload
  - Use V8 Benchmark

V8 Benchmarks for Firefox

	V8 Benchmark Value
Firefox with gcc	2549.2
Firefox with our mechanism	2587.6

- V8: JavaScript benchmark
  - Higher scores are better
  - Cannot be converted to time

- No significant change in performance

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
- Improved parsing speed
  - On average 73% decrease in parsing time
  - No memory or runtime overhead