Benchmarking Parsers

Nikolaos Bezirgiannis

March 22, 2012

Outline

- Intro
- Parser Combinator Libraries
- Benchmark Results
- Outro

State of Parsing

- Parser Generators

State of Parsing

- Parser Generators
  - Yacc / Bison
    - LALR(1) algorithm
    - Accompanied by a lexer (e.g. lex / flex)
    - Generates code for C / C++ / Java

State of Parsing

- Parser Generators
  - Yacc / Bison
    - LALR(1) algorithm
    - Accompanied by a lexer (e.g. lex / flex)
    - Generates code for C / C++ / Java
  - ANTLR
    - LL(*) algorithm
    - Generates code for many languages
    - GUI tools: editor, interpreter, debugger, visualization utilities

State of Parsing

- Parser Generators
  - Yacc / Bison
    - LALR(1) algorithm
    - Accompanied by a lexer (e.g. lex / flex)
    - Generates code for C / C++ / Java
  - ANTLR
    - LL(*) algorithm
    - Generates code for many languages
    - GUI tools: editor, interpreter, debugger, visualization utilities
  - Happy
    - Written in Haskell
    - Generates Haskell
    - Accompanied by Alex (also in Haskell)
    - Pretty solid & stable

State of Parsing

- Parser Generators
  - Yacc / Bison
    - LALR(1) algorithm
    - Accompanied by a lexer (e.g. lex / flex)
    - Generates code for C / C++ / Java
  - ANTLR
    - LL(*) algorithm
    - Generates code for many languages
    - GUI tools: editor, interpreter, debugger, visualization utilities
  - Happy
    - Written in Haskell
    - Generates Haskell
    - Accompanied by Alex (also in Haskell)
    - Pretty solid & stable
- Parser Combinators

Combinatory Parsing

- A **parser** is a function that consumes input and returns some structure.
- A **parser combinator** is a higher-order function that takes parsers to form a new larger parser.
- The **parsing program** is the application of some basic parsers and parsing combinators.

Combinatory Parsing (cont.)

- Advantages
  - Readable
  - Written in the same programming language
  - Can be extended by the user with new parsers and combinators
  - Maintainable
- Disadvantages
  - No grammar analysis
  - No left-recursion
  - Some cases have high time and space usage

History of Combinatory Parsing

- Wadler, 85: *List of Successes* method
- Wadler, 92: Monadic Parsers
- Hutton, 92: basic parser combinators (in Miranda)
- Röjemo, 95: Applicative Parsers
- Hutton,Meijer, 96: Standardizing parser combinators, a Hugs parsing library
- Leijen,Meijer, 01: Parsec, fast parsing library with good error reporting
- Swierstra, 01: Applicative parsers with error correction
- Swierstra, 04: Permutation parsing
- Swierstra, 06: Online results parsing

Parsec2

- Written by Daan Leijen
- Introduced in 2001
- Became de-facto
- Used to be distributed with GHC
- Now distributed with Haskell Platform

Parsec2 Features

- Parses context-sensitive grammars
- Predictive LL(1) by default
- LL(*) by explicit backtracking
- Monadic Interface
- Permutation parsing
- Good error messages (Position, Unexpected Input, Expected production)
- Haskell98-compatible

Parsec2 Drawbacks

- Lacks an applicative interface
- Accepts a token-list as input
- Cannot be used with Bytestring or Text

Parsec3

- In most cases, API-compatible to Parsec2
- Adds an applicative interface
- Parameterized over the input type
- Provides the parser as a monad transformer
- Breaks Haskell98-compatibility
- Trying to be flexible, it pays in terms of speed

UUlib General

- Written by UU
- Introduced $<$ 2005
- Stable
- The library bundles also a lexer and a prettyprinter

UUlib Features

- Applicative Interface
- Good Error reporting

UUlib Drawbacks

- Lacks a monadic interface $\Rightarrow$ difficult to construct context-sensitive grammars
- Does not have pSatisfy.It makes difficult to construct some basic parsers.
- Not documented
- Restricted only to String input

UU-parsinglib General

- Written by Doaitse Swierstra
- Introduced in 2008
- Somewhat stable

UU-parsinglib Features

- LL(*)
- Applicative and Monadic Interface
- API-compatible with uulib
- Good at dealing with ambiguity
  - Guesses out when the grammar is ambiguous and reports it
  - `amb :: Parser a -> Parser [a]`
- Compared to the other libraries, it seems to be the best abstracted away
- Great Error messages
- Error Correction
- Online parsing results
- Visits the alternatives in parallel (BFS), so it does not retain the whole input in memory
- Input can also be Bytestring,Text,etc
- Extremely easy to switch between input types (thanks to ListLike)

UU-parsinglib Drawbacks

- Error Correction cannot be used with the monadic interface
- Underdocumented
- The abstractions make it harder for beginners to understand the lib

Attoparsec General

- Written by Bryan O'Sullivan
- Introduced in 2007
- Has recently become very popular
- Starting to mature

Attoparsec Features

- LL(*) , implicit backtracking
- Applicative and Monadic Interface
- Supports Text, Bytestring
- Supports Incremental Input
- Nearly translatable/compatible with Parsec
- Easy to use, by having distinct modules

Attoparsec Drawbacks

- Trying to stay simple, it lacks some high-level combinators
- Not good in error reporting
- Does not accepts String input

Polyparse General

- Written by Malcolm Wallace
- Introduced in 2002
- Is actually a collection of different parsing libraries

Polyparse Features

- LL(*) with implicit backtracking
- Applicative and Monadic Interface
- Partial parsing results
- Supports String, ByteString and Text for input

Polyparse Drawbacks

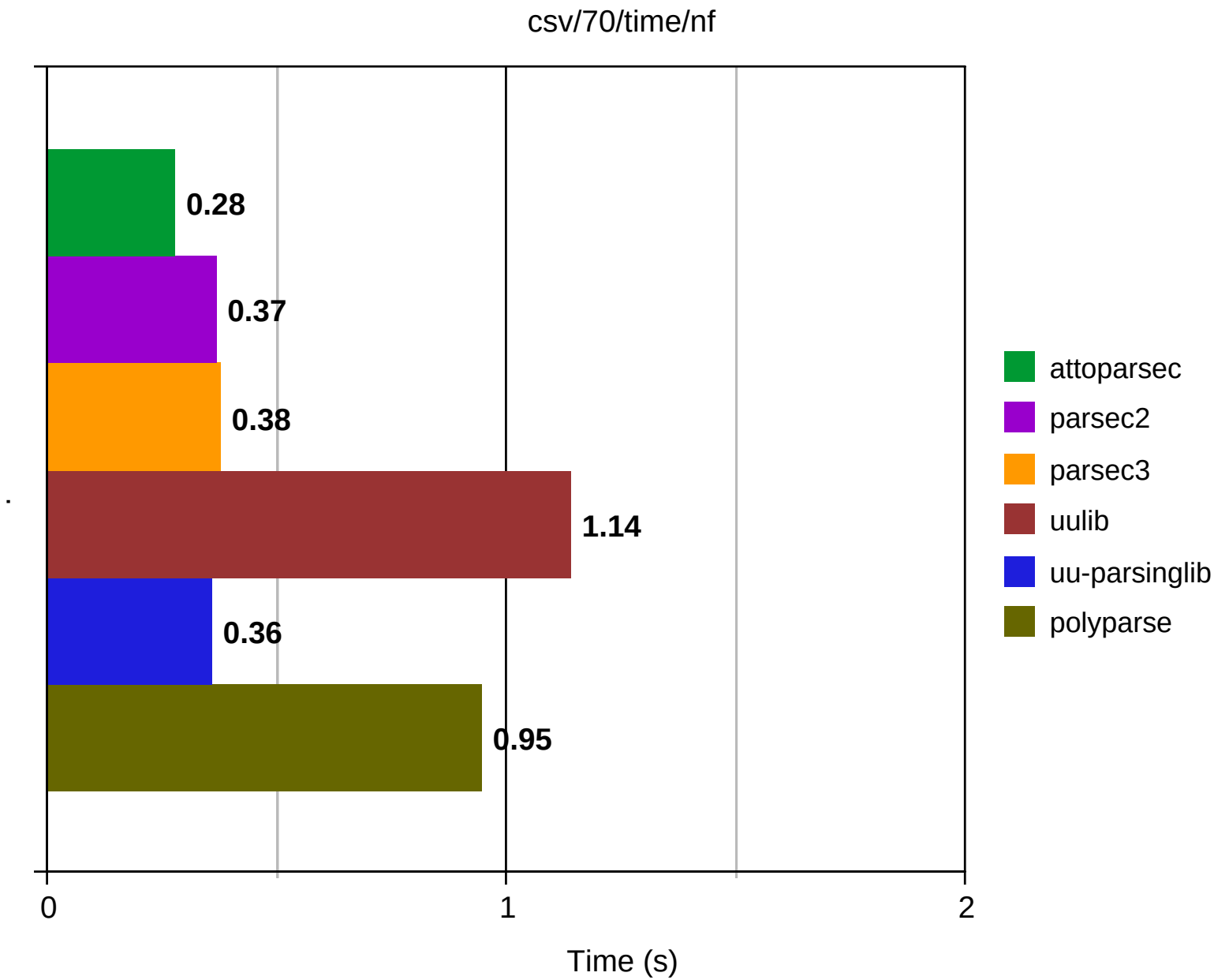
- The library's schema is confusing
- Lacks some high-level combinators for Text

CSV - Example

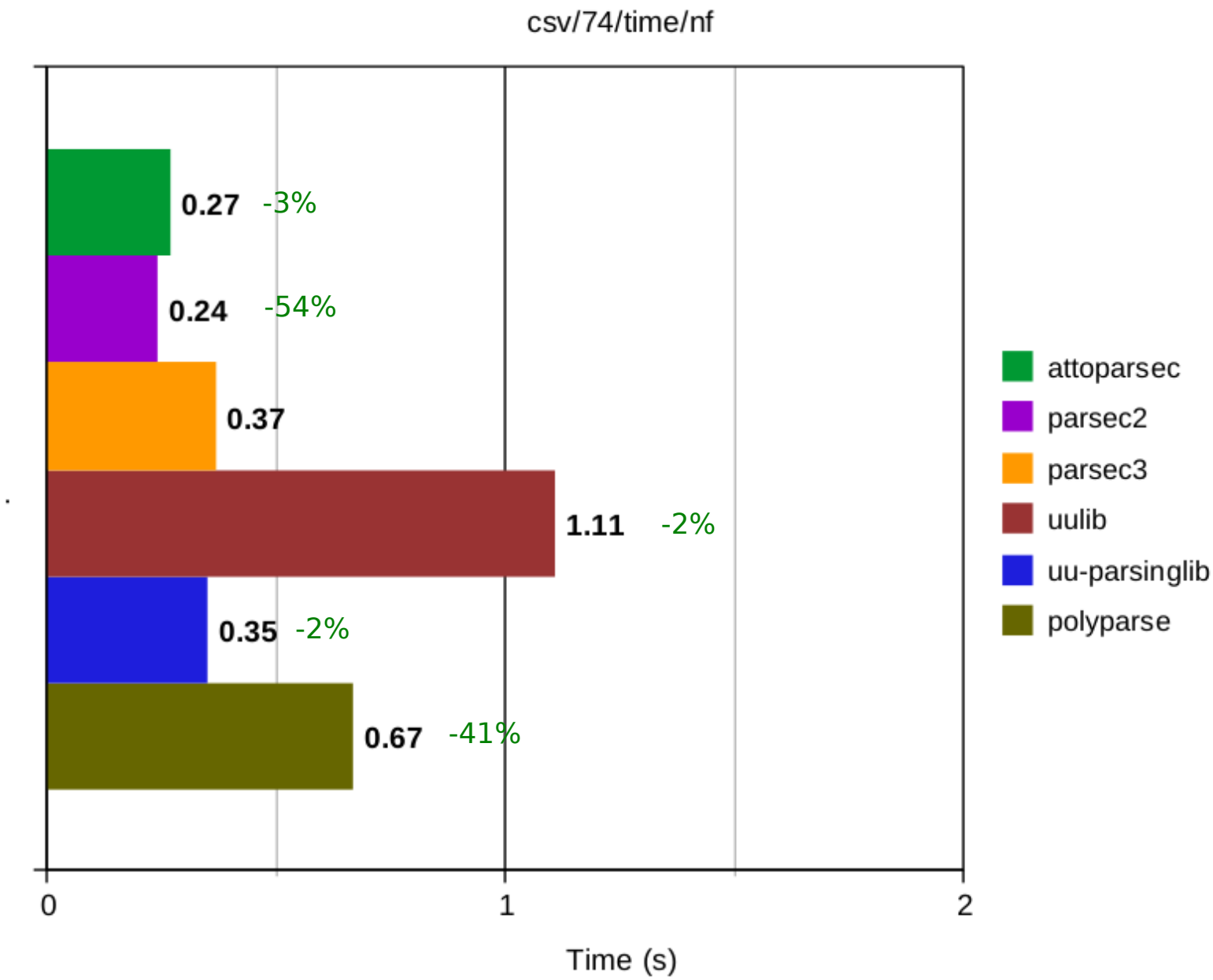
- Comma-separated values
- Has a standard but many variations exist
- Our parsers support the standard with unicode
- Datasets taken from the net

```
dma code,region,state
500,Portland-Auburn,ME
501,New York,NY
502,Binghamton,NY
503,Macon,GA
504,Philadelphia,PA
505,Detroit,MI
506,Boston,MA
507,Savannah,GA
508,Pittsburgh,PA
```

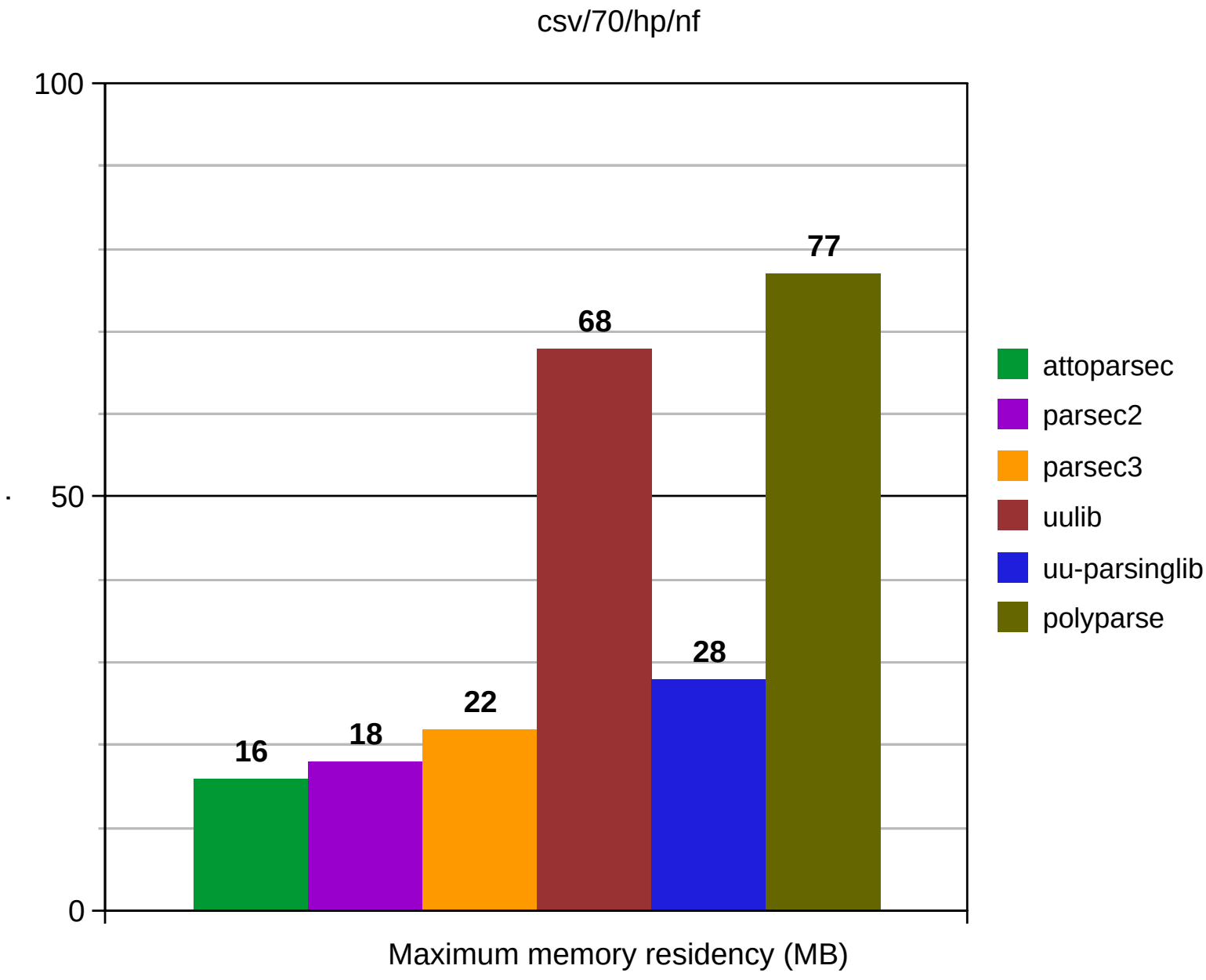
CSV - Time


csv/70/time/nf
0.28
0.37
0.38
1.14
0.36
0.95
0
1
2
Time (s)
attoparsec
parsec2
parsec3
uulib
uu-parsinglib
polyparse

CSV - Time - 74


csv/74/time/nf
0.27 -3%
0.24 -54%
0.37
1.11 -2%
0.35 -2%
0.67 -41%
0
1
2
Time (s)
attoparsec
parsec2
parsec3
uulib
uu-parsinglib
polyparse

CSV - Space

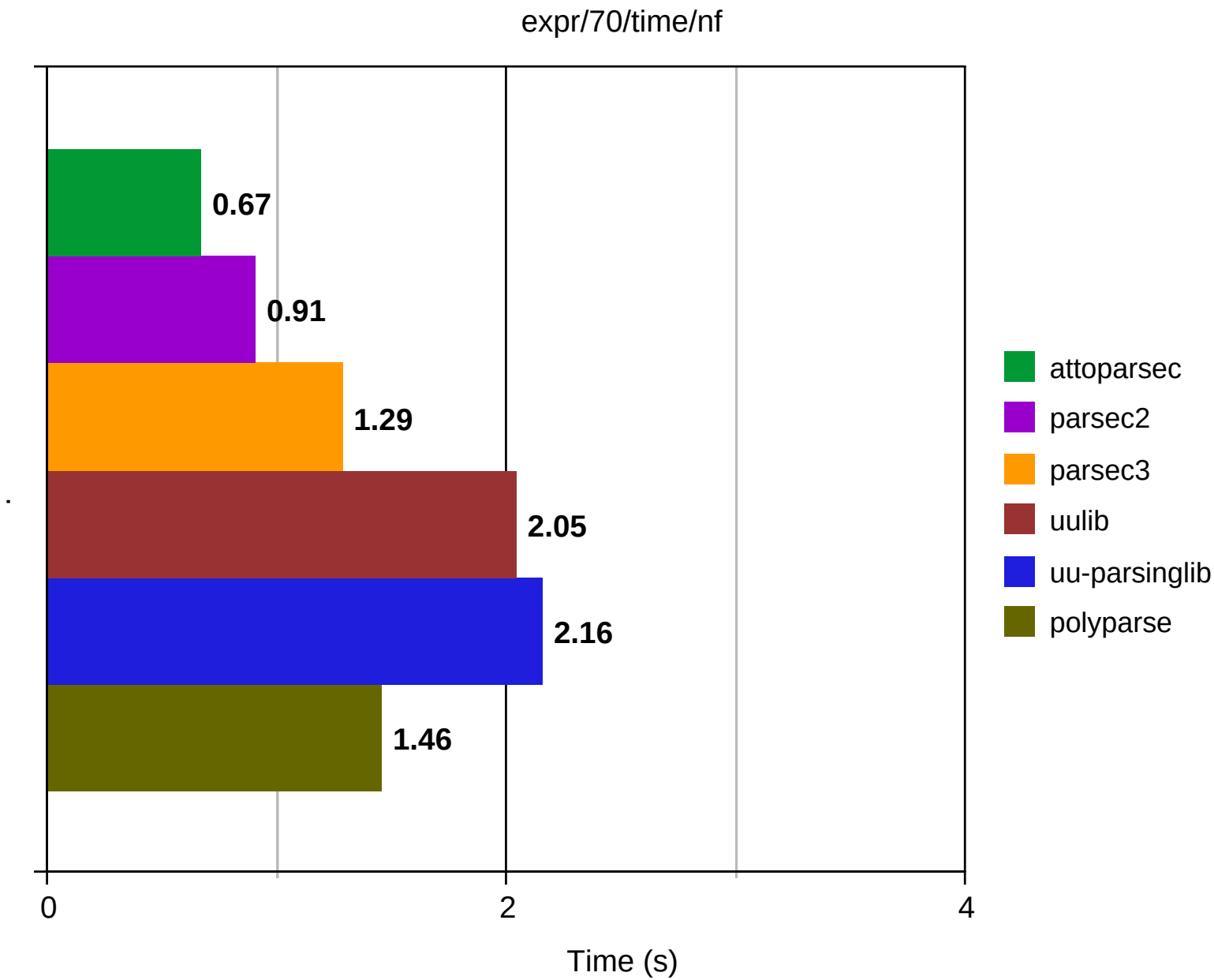

csv/70/hp/nf
100
50
0
16
18
22
68
28
77
attoparsec
parsec2
parsec3
uulib
uu-parsinglib
polyparse
Maximum memory residency (MB)

EXPR - Example

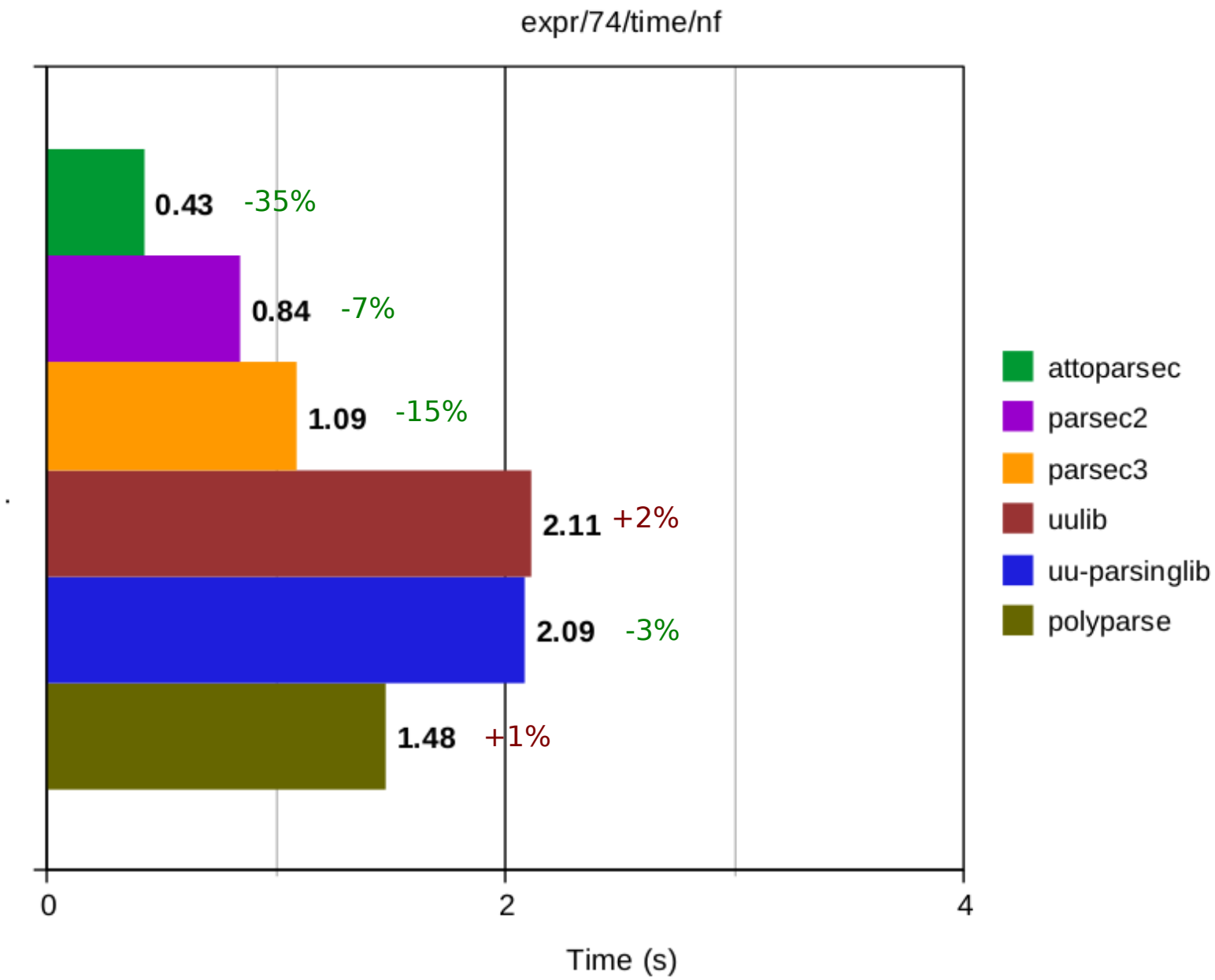
- a small custom functional language
- Datasets generated with QuickCheck's Arbitrary

```
((2 - let nngxpu = (kpjht + (lms + (medpc - l))) in
(let sk = yui in o + ((qatzrb * 2) - (8 - 2)))) -
(((let idwxvj = let ywkldp = 5 in c in (5 * az)
* (8 * let rh = yo in h)) + (((awpcu - 2) - (tcwhem *
nyt)) * let pm = (5 * lttgyt) in (cl + 5))) -
((((bf - 3) * (7 * 7)) - ((7 * vpienh) + (dy + 6)))
+ (((8 + 6) - (7 * 7)) + let zvs = poks in (2 * sfwo)))))
```

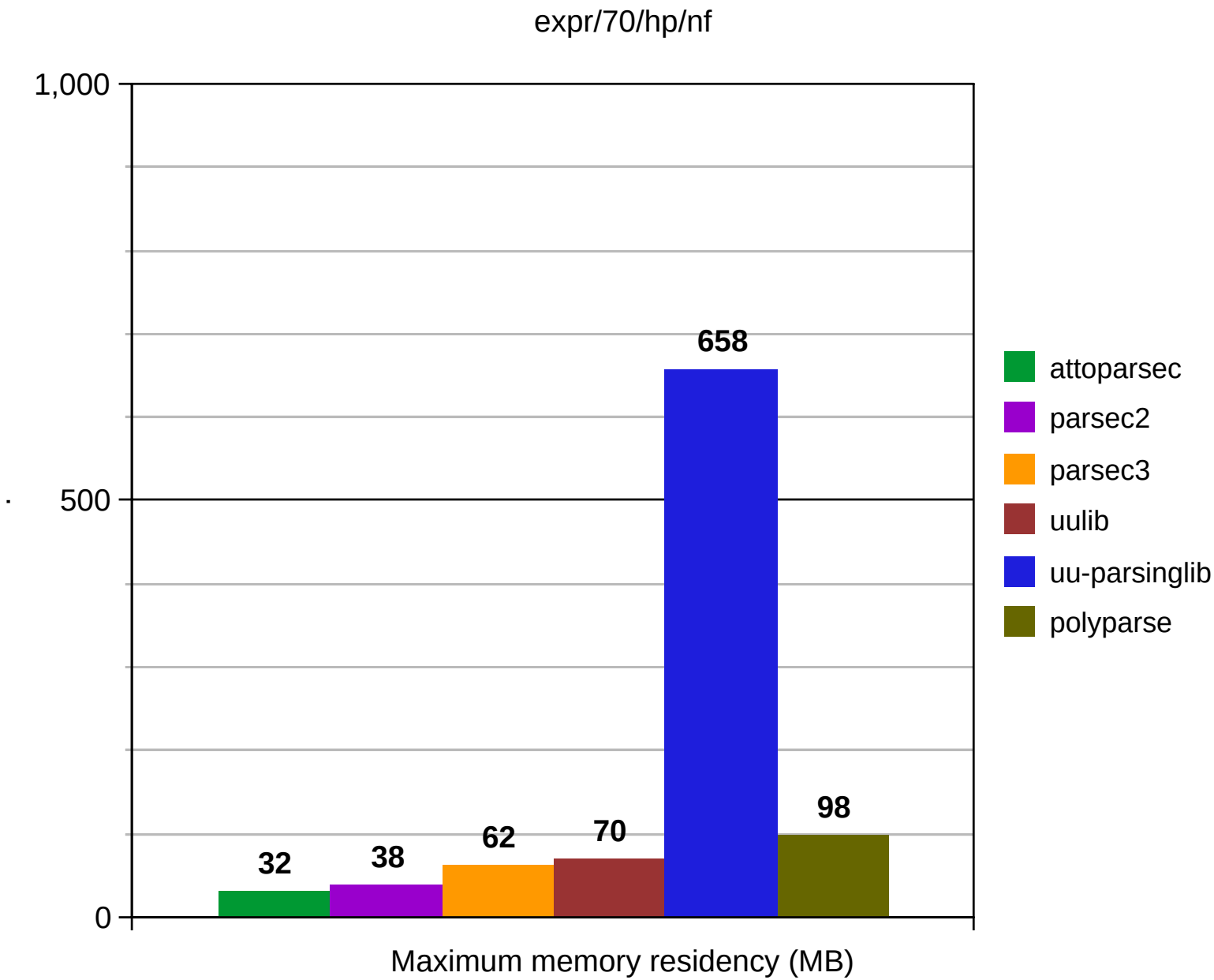
EXPR - Time


expr/70/time/nf
0.67
0.91
1.29
2.05
2.16
1.46
0
2
4
Time (s)
attoparsec
parsec2
parsec3
uulib
uu-parsinglib
polyparse

EXPR - Time - 74


expr/74/time/nf
0.43 -35%
0.84 -7%
1.09 -15%
2.11 +2%
2.09 -3%
1.48 +1%
0
2
4
Time (s)
attoparsec
parsec2
parsec3
uulib
uu-parsinglib
polyparse

EXPR - Space

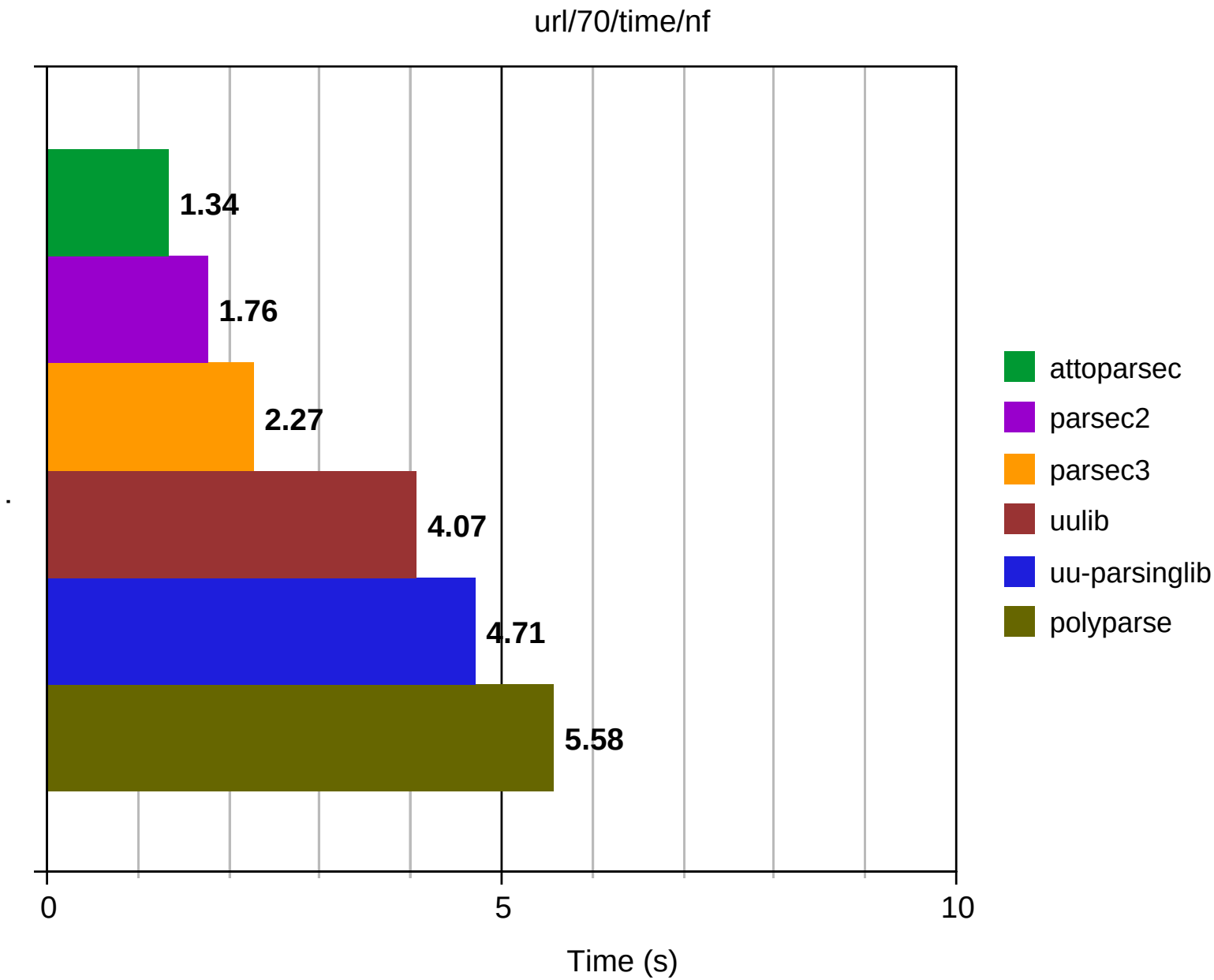

expr/70/hp/nf
1,000
500
0
658
32
38
62
70
98
attoparsec
parsec2
parsec3
uulib
uu-parsinglib
polyparse
Maximum memory residency (MB)

URL - Example

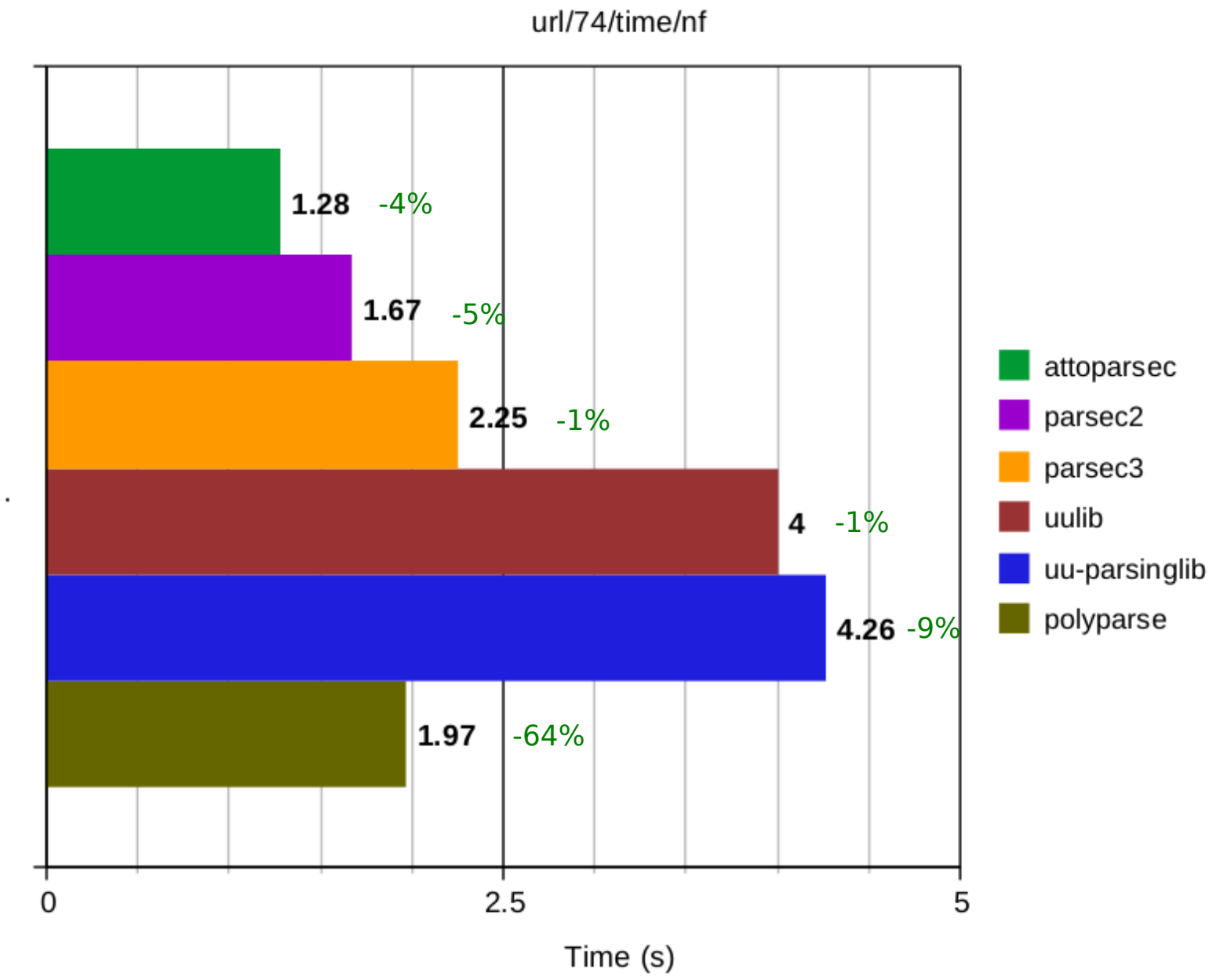
- URL query string
- Datasets generated from previous CSV

```
Aachen&Aalborg=aah&Aalesund=aahed&Aalst=aahing&Aalto=aahs
&Aarau=aal&Aargau&Aarhus&Aaron=aals&Aarons+rod=aardvark
&Aaronic=aardwolf&Ab=aargh&Abadan&Abaddon=aarrghh&Abba=
aas&Abbasid=aasvogel&Abbevillian=aba&Abbey+Theatre=abaca&
```

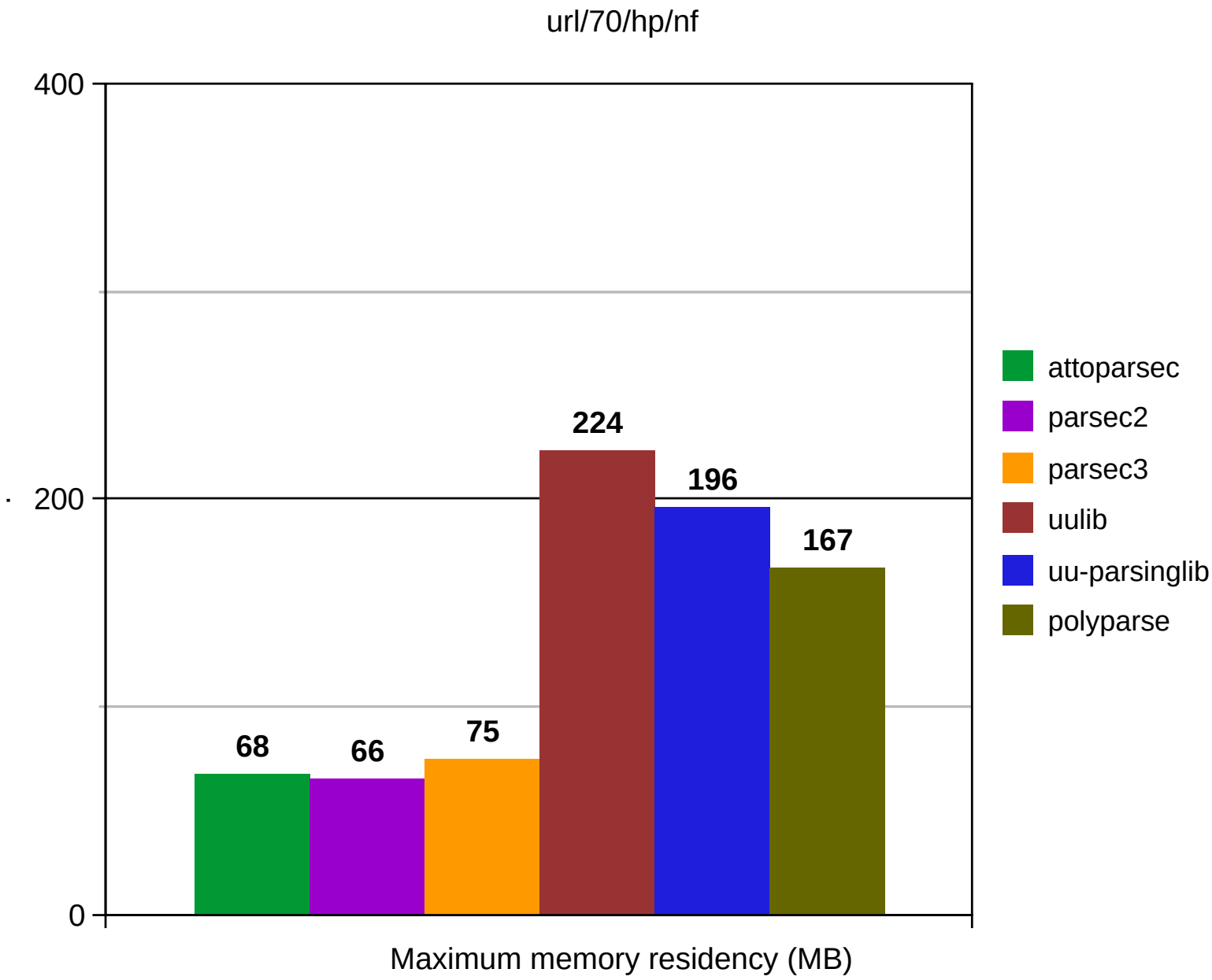
URL - Time


url/70/time/nf
1.34
1.76
2.27
4.07
4.71
5.58
0
5
10
Time (s)
attoparsec
parsec2
parsec3
uulib
uu-parsinglib
polyparse

URL - Time - 74


url/74/time/nf
1.28
-4%
1.67
-5%
2.25
-1%
4
-1%
4.26
-9%
1.97
-64%
0
2.5
5
Time (s)
attoparsec
parsec2
parsec3
uulib
uu-parsinglib
polyparse

URL - Space

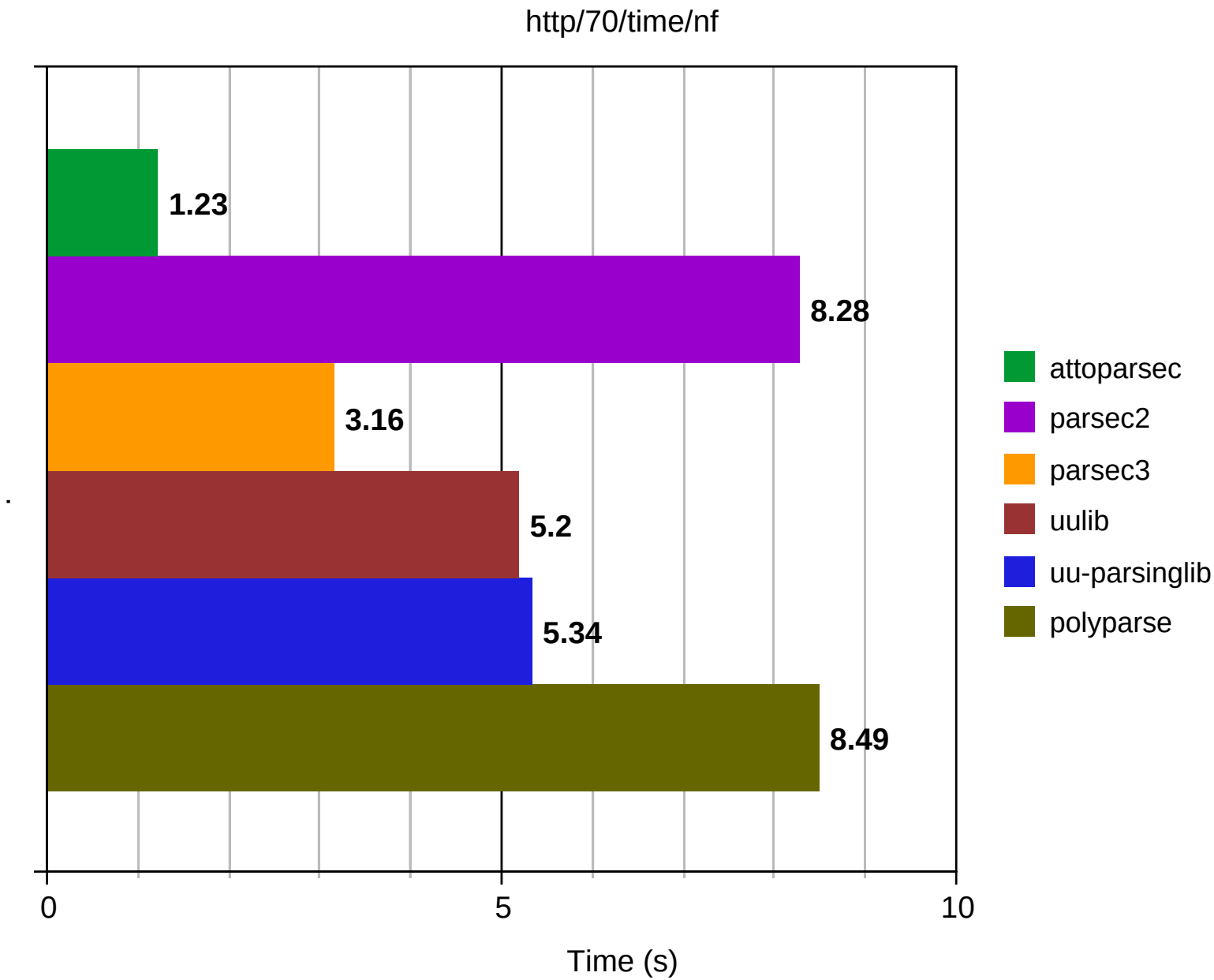

url/70/hp/nf
400
200
0
68
66
75
224
196
167
attoparsec
parsec2
parsec3
uulib
uu-parsinglib
polyparse
Maximum memory residency (MB)

HTTP - Example

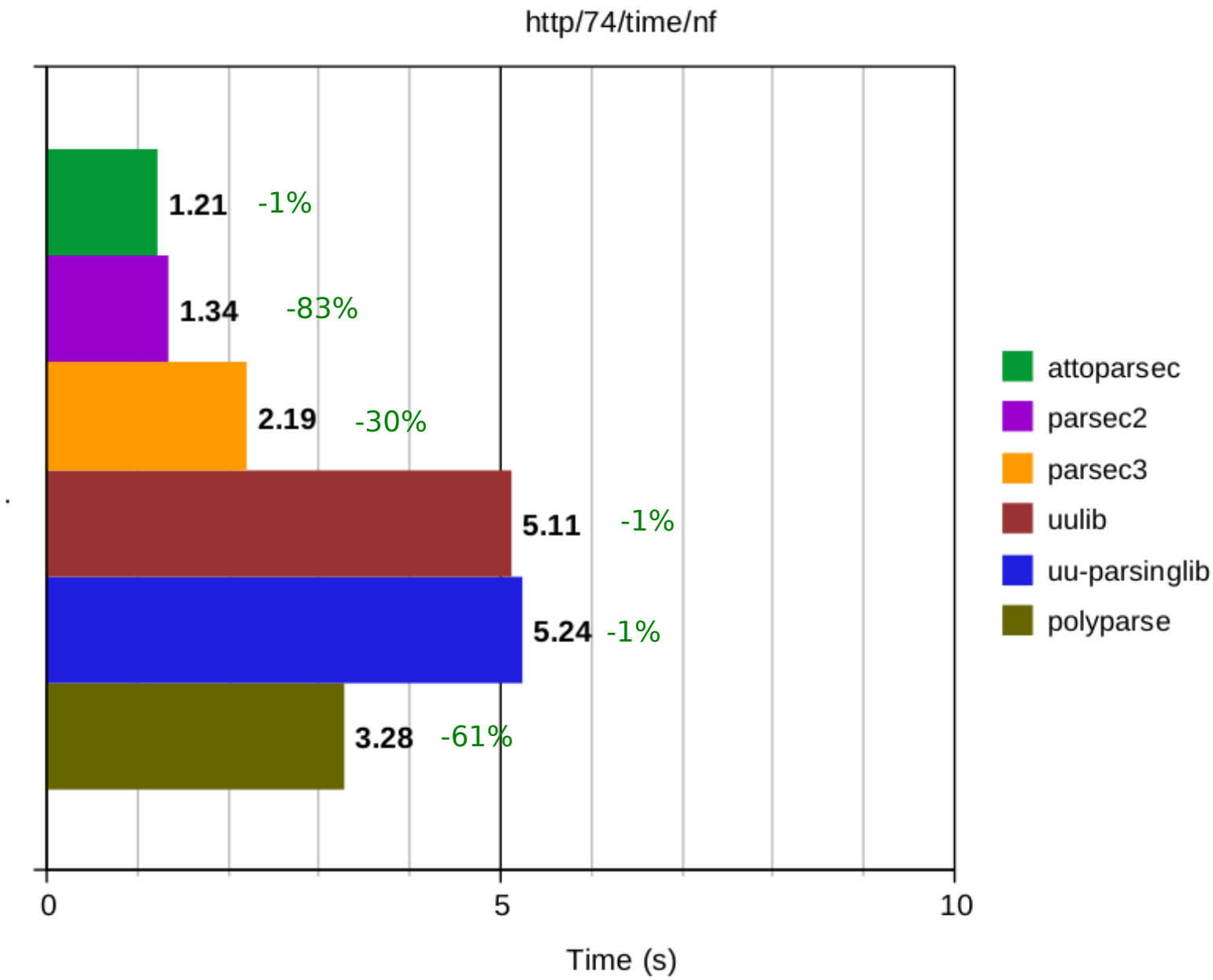
- HTTP GET and POST requests
- Datasets generated from previous CSV

```
POST http://www.w3.org/#sec3.6.1 HTTP/1.1
Aachen: aa
Aalborg: aah
Aalesund: aahed
Aalst: aahing
Aalto: aahs
Aarau: aal
Aargau: aalii
```

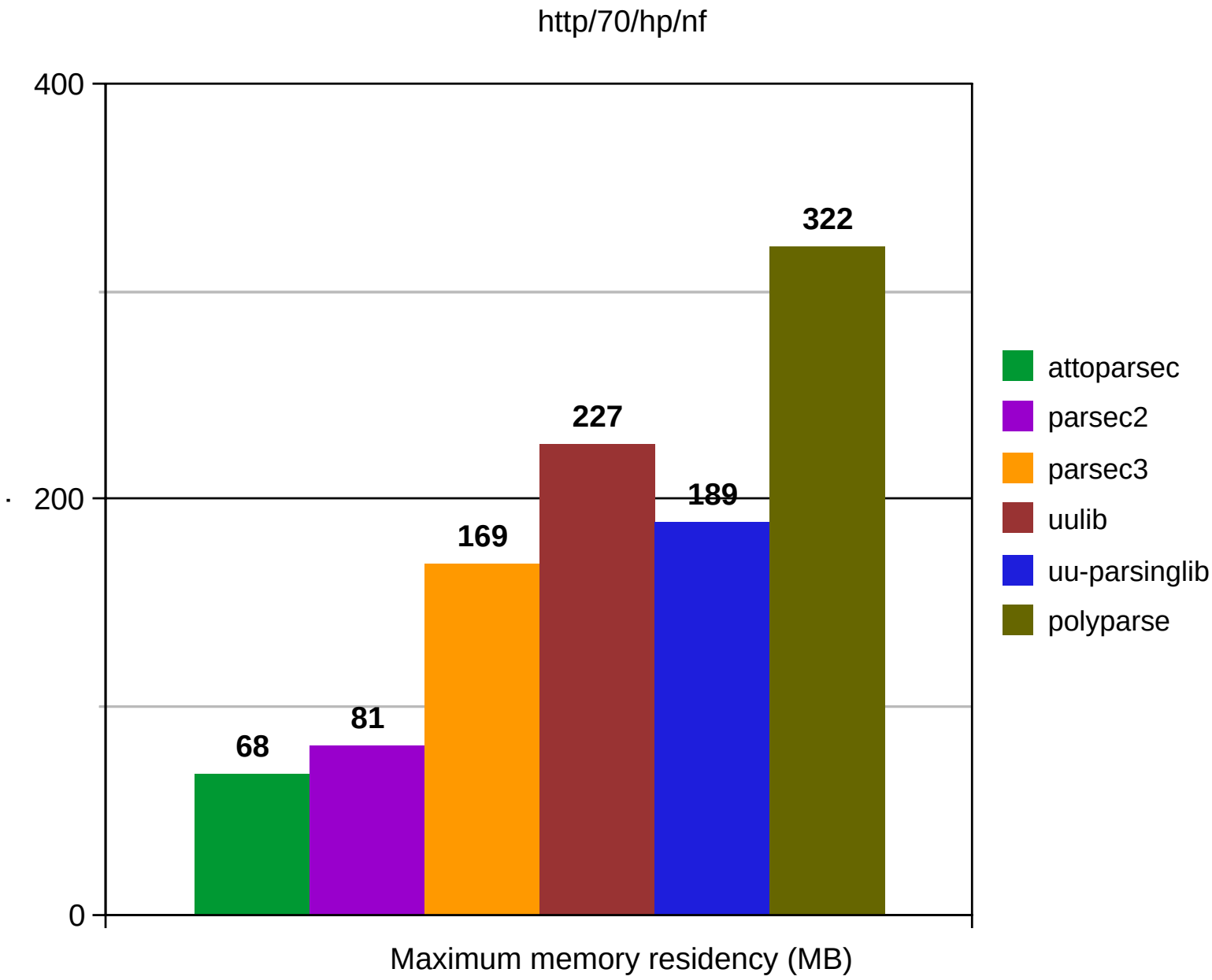
HTTP - Time


http/70/time/nf
1.23
8.28
3.16
5.2
5.34
8.49
0
5
10
Time (s)
attoparsec
parsec2
parsec3
uulib
uu-parsinglib
polyparse

HTTP - Time - 74


http/74/time/nf
1.21 -1%
1.34 -83%
2.19 -30%
5.11 -1%
5.24 -1%
3.28 -61%
0
5
10
Time (s)
attoparsec
parsec2
parsec3
uulib
uu-parsinglib
polyparse

HTTP - Space

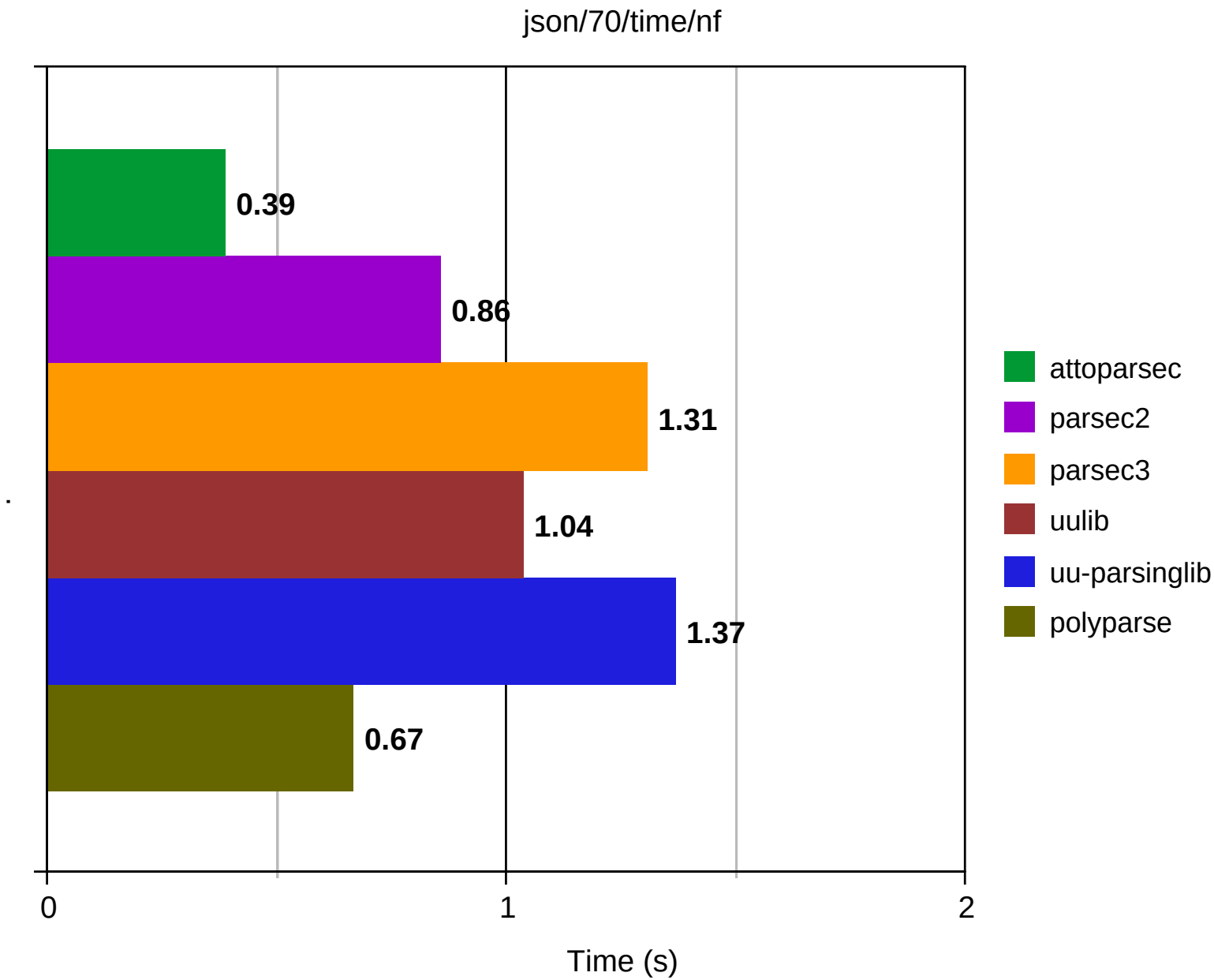

http/70/hp/nf
400
200
0
68
81
169
227
189
322
attoparsec
parsec2
parsec3
uulib
uu-parsinglib
polyparse
Maximum memory residency (MB)

JSON - Example

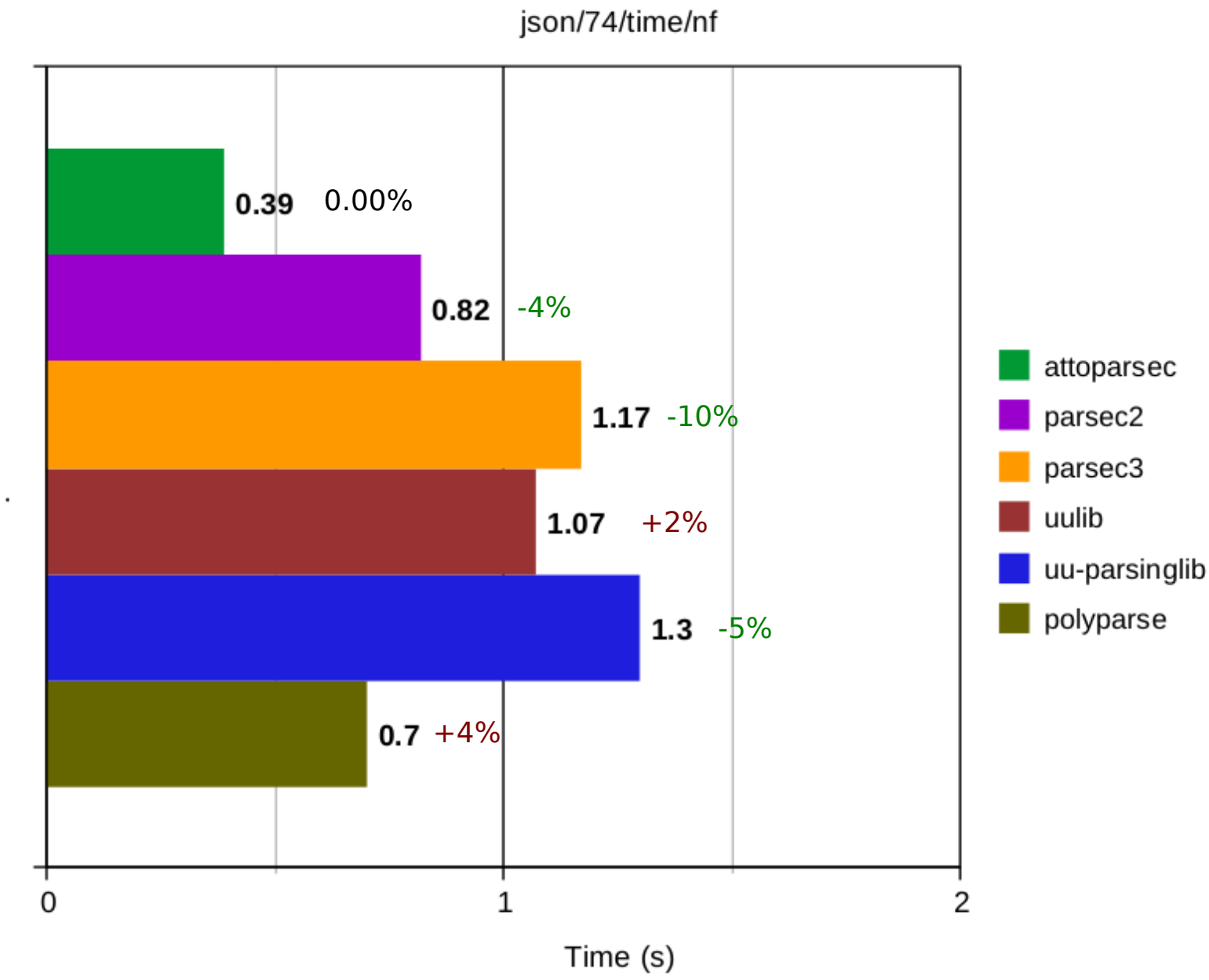
- Javascript Object Notation
- Datasets collected from GitHub


```
{"030200b_.mid": [[{"timesig": null, "keysig": 2, "st": 0,
"pitch": 50, "dur": 12.0, "fermata": 0}, {"timesig": null,
"keysig": 2, "st": 12.0, "pitch": 49, "dur": 2.0,
"fermata": 0}]]}
```

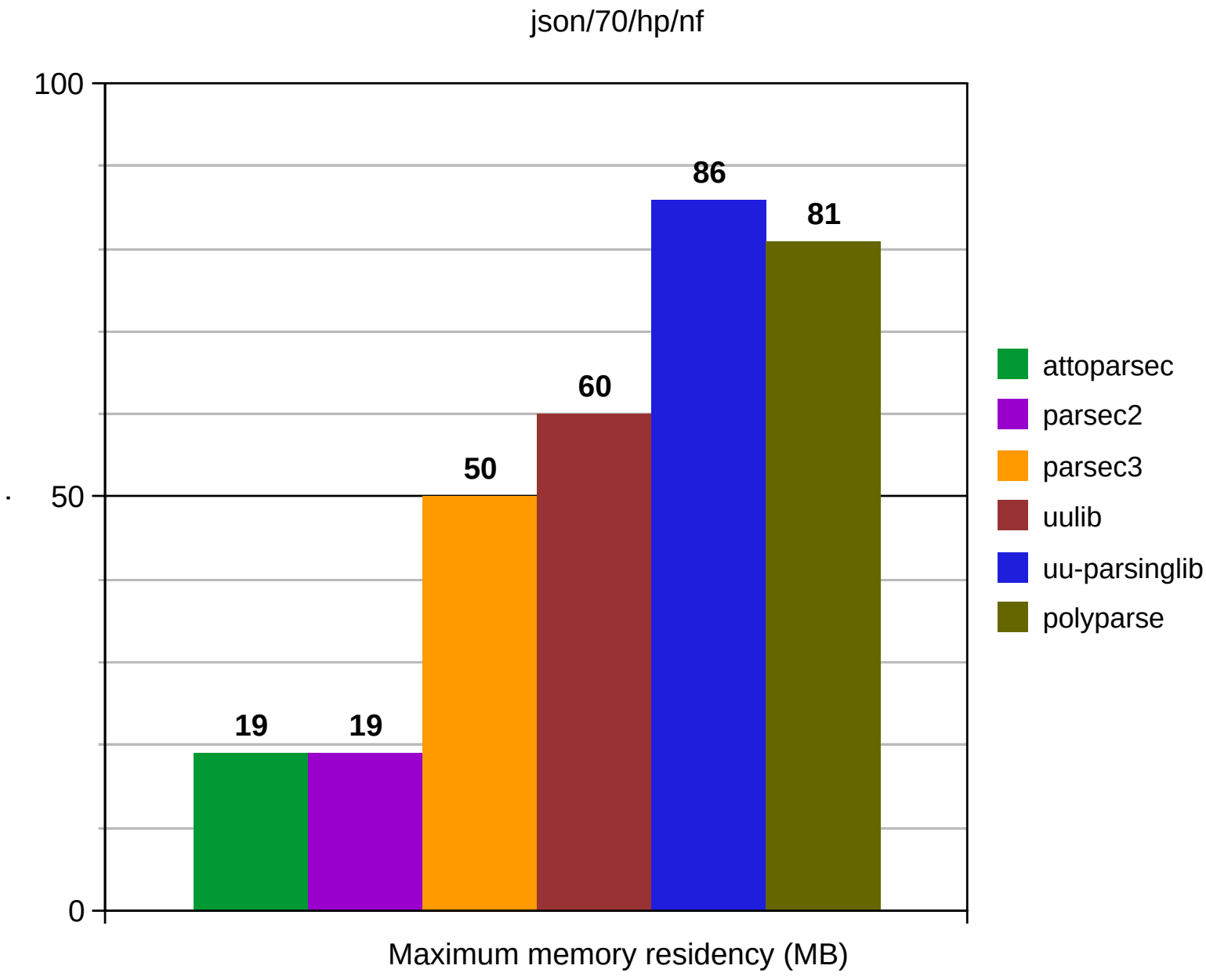
JSON - Time


json/70/time/nf
0.39
0.86
1.31
1.04
1.37
0.67
0
1
2
Time (s)
attoparsec
parsec2
parsec3
uulib
uu-parsinglib
polyparse

JSON - Time - 74


json/74/time/nf
0.39 0.00%
0.82 -4%
1.17 -10%
1.07 +2%
1.3 -5%
0.7 +4%
attoparsec
parsec2
parsec3
uulib
uu-parsinglib
polyparse
0
1
2
Time (s)

JSON - Space

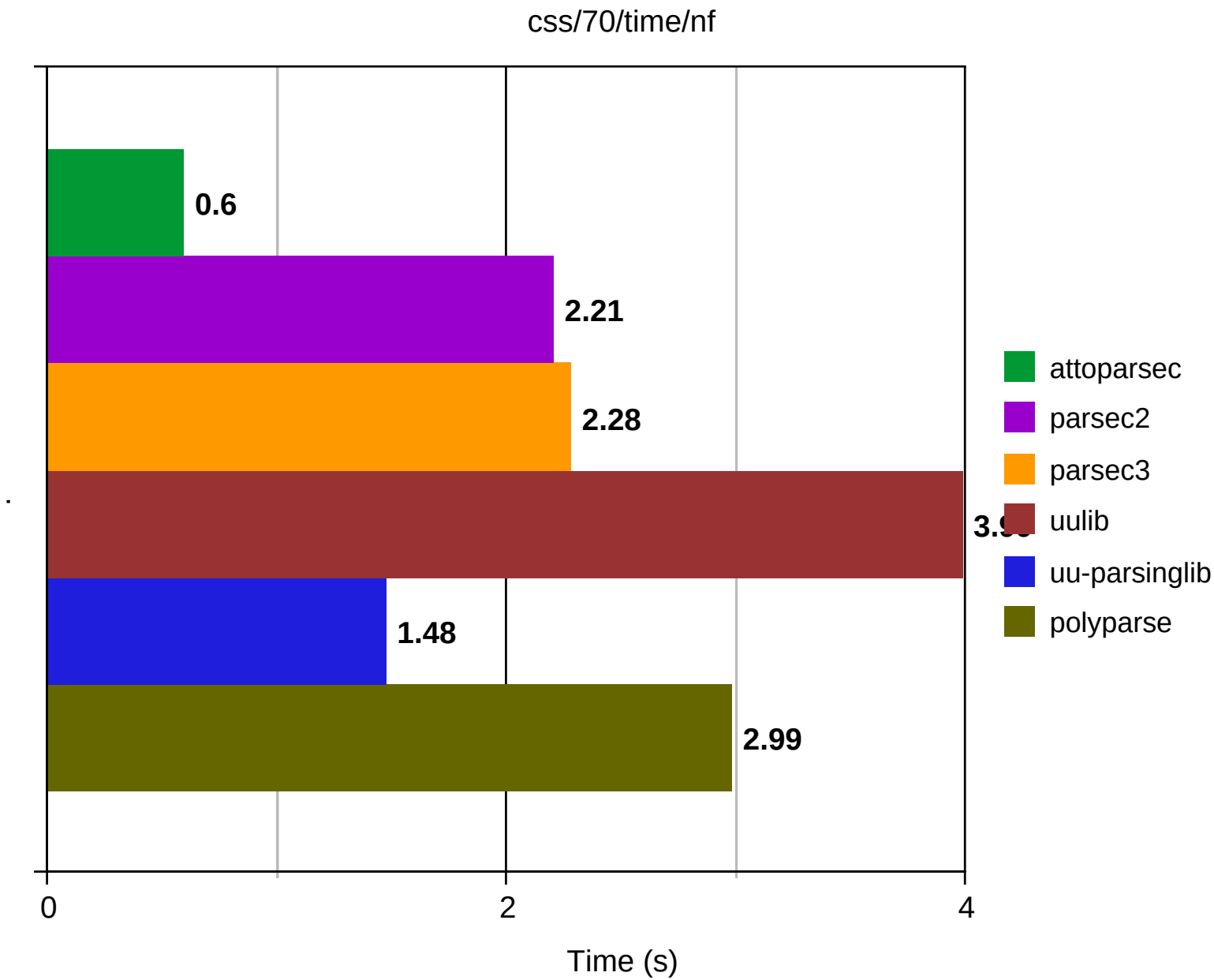

json/70/hp/nf
100
50
0
19
19
50
60
86
81
attoparsec
parsec2
parsec3
uulib
uu-parsinglib
polyparse
Maximum memory residency (MB)

CSS - Example

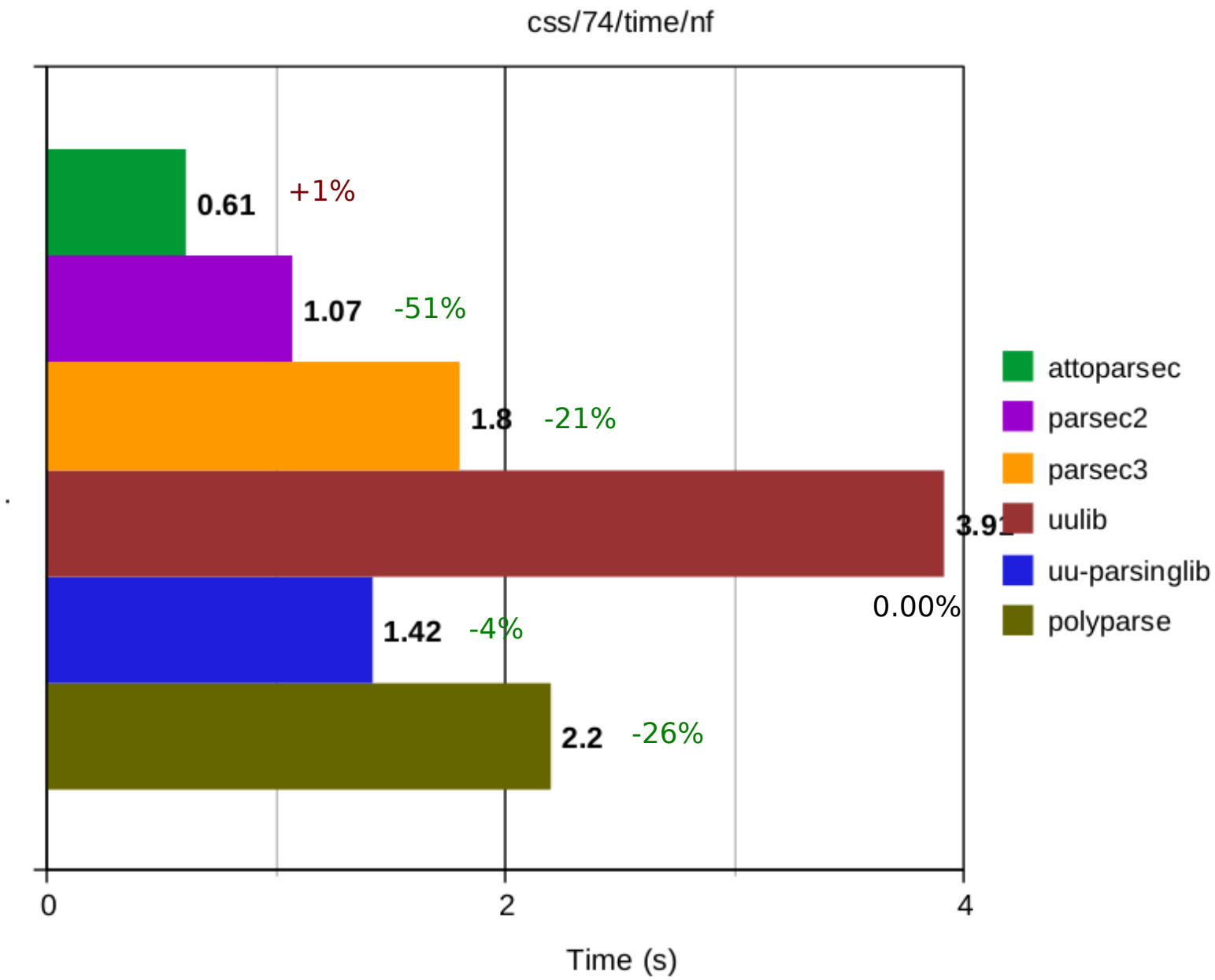
- Cascading Style Sheets
- Datasets collected from GitHub

```
body{
  background-color: #222;
  color: white;
  font-family: "Bitstream Vera Sans",sans-serif;
  margin: 0;
  padding: 0;
}
#container{
  margin: 0 10%;
  background-color: #161616;
  padding: 16px;
}
```

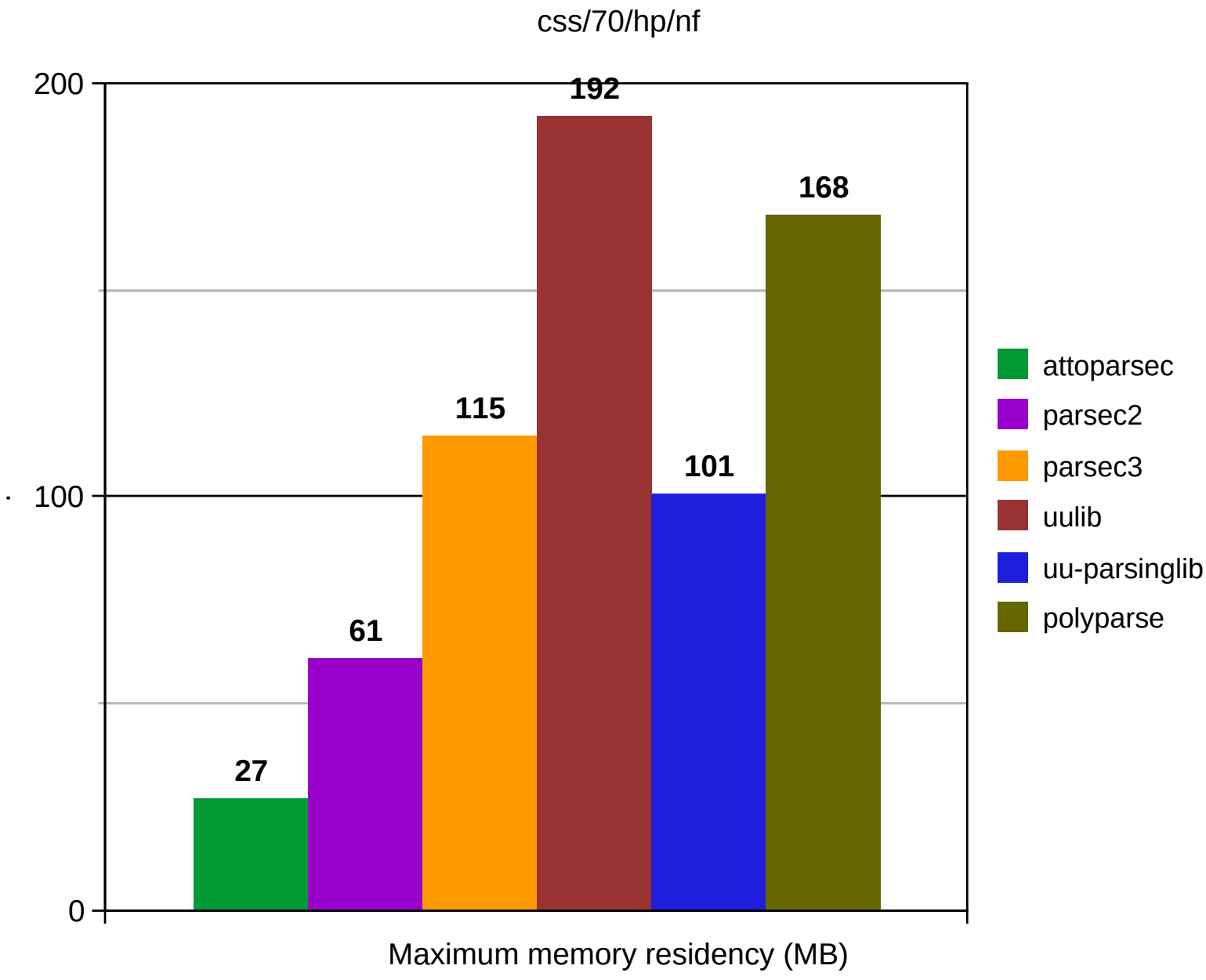
CSS - Time


css/70/time/nf
0.6
2.21
2.28
1.48
2.99
attoparsec
parsec2
parsec3
uulib
uu-parsinglib
polyparse
0
2
4
Time (s)

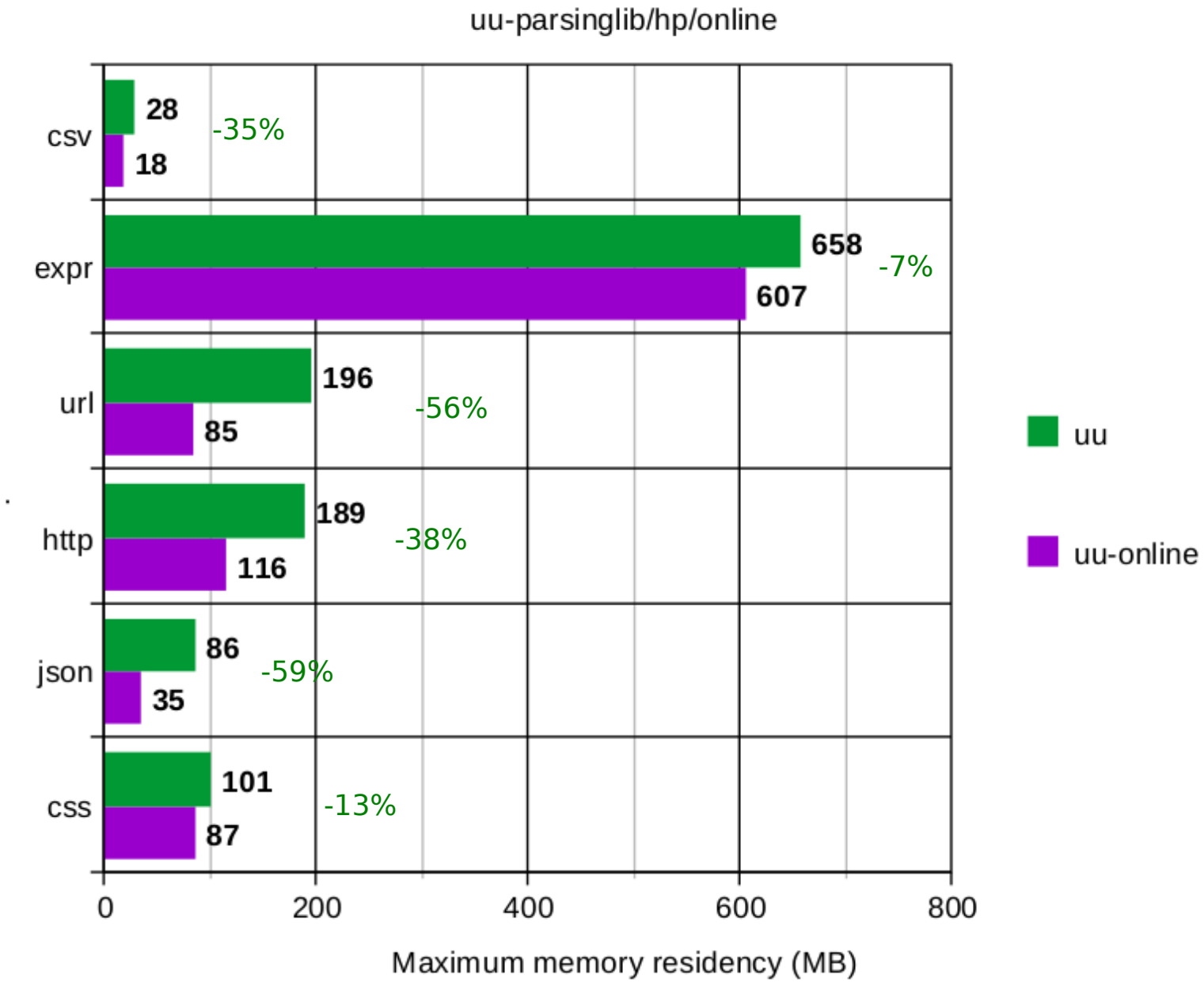
CSS - Time - 74


css/74/time/nf
0.61
+1%
1.07
-51%
1.8
-21%
3.91
0.00%
1.42
-4%
2.2
-26%
0
2
4
Time (s)
attoparsec
parsec2
parsec3
uulib
uu-parsinglib
polyparse

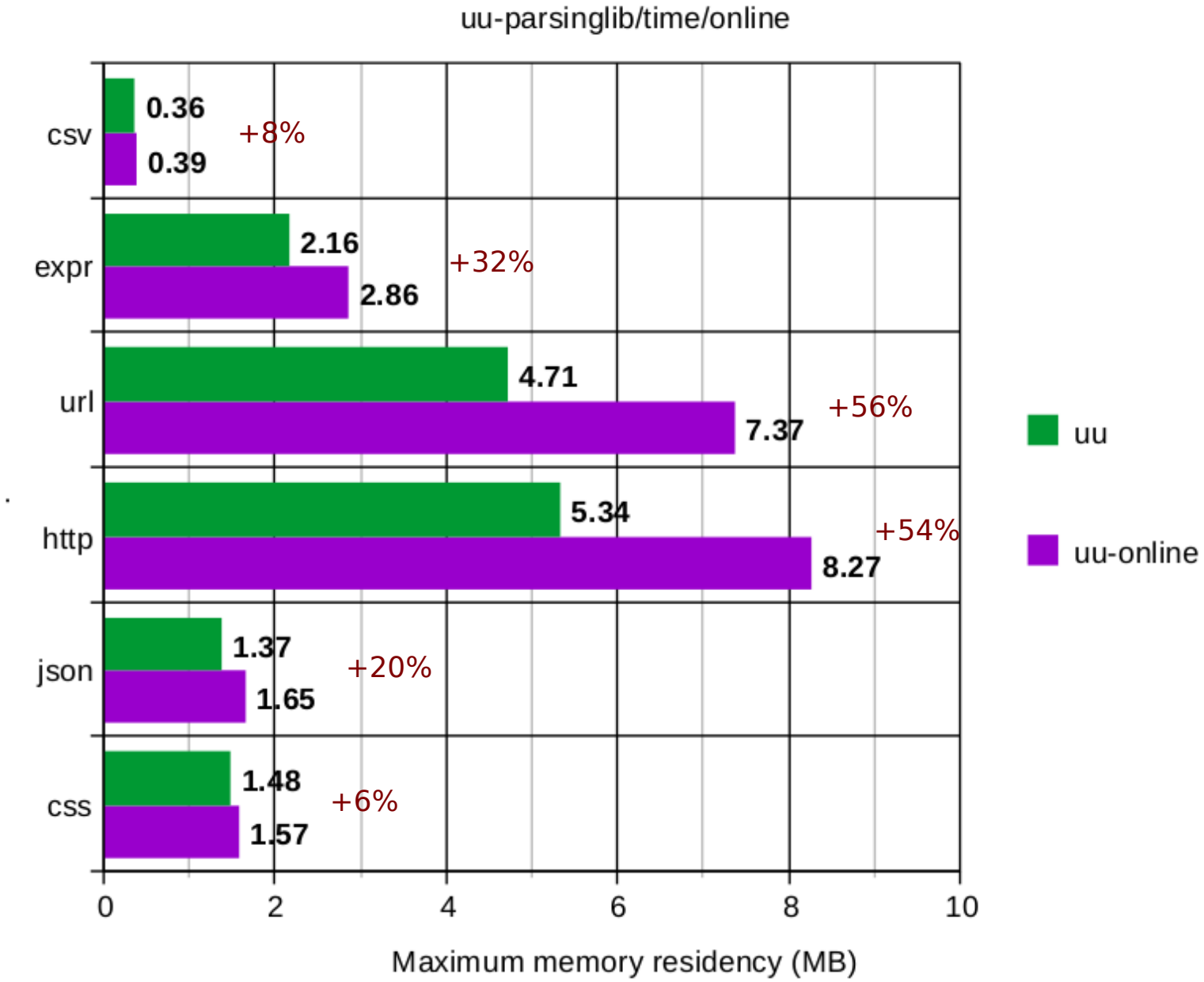
CSS - Space


css/70/hp/nf
200
100
0
192
168
115
101
61
27
attoparsec
parsec2
parsec3
uulib
uu-parsinglib
polyparse
Maximum memory residency (MB)

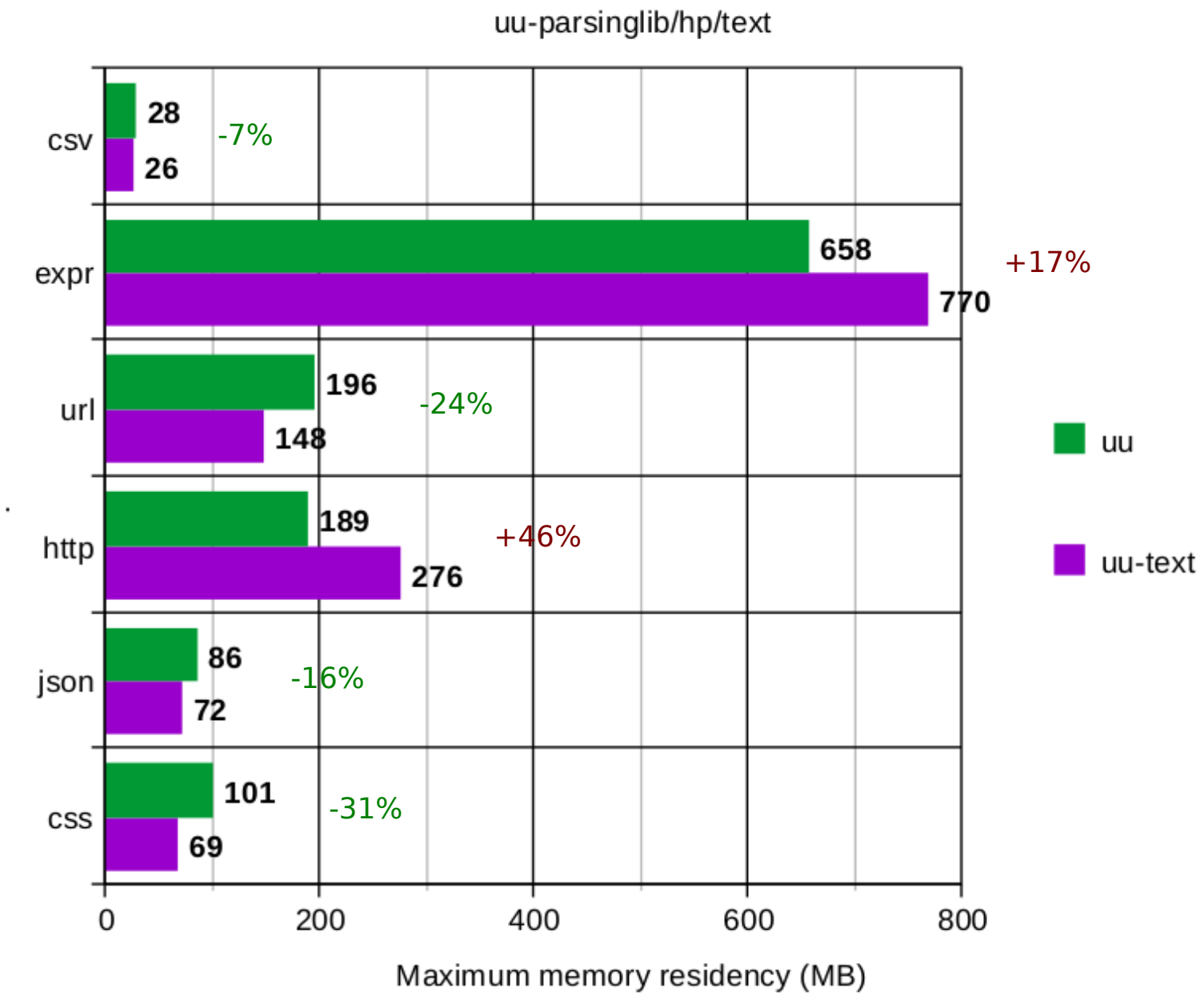
UU-parsinglib Online - Space



UU-parsinglib Online - Time



UU-parsinglib Data.Text - Space



Overall results

- GHC 7.4 brings a 15% performance increase
- WHNF 17% faster than NF on the test
- WHNF uses 16% less memory than NF

How the results were produced

- My slow laptop
- Archlinux x86_64
- GHC 7.0.3 and GHC 7.4.1
- Latest versions of parsing libraries
- Criterion
- Virthualenv
- Custom profiling scripts

Conclusion & Impressions

- attoparsec is the winner on almost all benchmarks
- parsec2 is faster than parsec 3 on all benchmarks (GHC 7.4)
- uu-parsinglib is the best abstracted away
- uu-parsinglib online feature is benifitial albeit slower
- was expecting a larger boost with Data.Text
- WHNF is slightly faster than NF for all contenstants, but does not affect the results ordering

Suggestions & Future Work

- Revise the parsers
- Try to squeeze even the last drop of speed
- Add more example cases
- It would be nice if there was a similar tool to Criterion but for profiling comparisons