Benchmarking and Performance

FORM Developers Workshop 2026, Nikhef

Josh Davies

26th June, 2026

Introduction

Since v5.0 release, I have spent some time on performance profiling and improvements.

- joined by Ana Costa Pereira, see also next talk.

Profiling mostly with Linux **`perf`**, AMD uProf, Intel VTune.

Performance improvements already in the current master branch: part of v5.1.

Collecting a “standard set” of benchmarks:

- try to aim primarily for “realistic workloads” rather than micro-benchmarks
- tests mostly run in under a minute (**`tform`**), option for “harder mode”
- if you have nice scripts to add, please contribute!

form-bench

https://github.com/form-dev/form-bench

Set of benchmarks in **tests/**

Bash script to run them: **run-compare.sh**:

- **--label** : name for results directory
- **--testdir** : running directory (tmpfs is good)
- **--timestamp** : appended to results directory
- **--nice** : **nice** level for test runs
- **--form_cmds** : comma-sep list of **form** commands to run (negative is good)
- **--tests** : comma-sep list of tests to run
- **--runs** : multiplier for number of test runs
- **--difficulty** : enable “hard mode” for supported tests

Results in **output/testdir-timestamp/results**.

Uses [hyperfine] for timings, some simple plots with **python** and **matplotlib**.

form-bench

```
./run-compare.sh --label "test-run" --runs 1 --tests "sort-small" \
                 --form_cmds "tform-5.0.0 -w12,tform-master -w12"

form-bench: 851a1e3664ffcc6296d78f638385b9ea444188af
        LABEL      = test-run
        TESTDIR    = /tmp
        TIMESTAMP  = 2026-06-19-13-37-52
        NICE       = 0
        FORM_CMDS  = tform-5.0.0 -w12,tform-master -w12
        TESTS      = sort-small
        N          = 1
        DIFFICULTY = 1
Results: /home/josh/form-bench/output/test-run-2026-06-19-13-37-52/results/

Running sort-small
Benchmark 1: tform-5.0.0 -w12
  Time (mean ± σ):      1.759 s ±  0.014 s    [User: 18.095 s, System: 0.745 s]
  Range (min...max):    1.727 s... 1.782 s    20 runs

Benchmark 2: tform-master -w12
  Time (mean ± σ):     962.6 ms ±  11.0 ms    [User: 9867.2 ms, System: 799.2 ms]
  Range (min...max):   943.6 ms...988.0 ms    20 runs

Summary
  tform-master -w12 ran
    1.83 ± 0.03 times faster than tform-5.0.0 -w12
```

Current tests

`chromatic`: compute chromatic polynomial of grid graph

- benchmark **`p15.frm`** with N=14 D=2, of [hep-ph/0702279] (**`TFORM`** paper)
- uses **`PolyFun`**, one variable
- in MMA (slowly): **`ChromaticPolynomial[GridGraph[Table[N,{d,1,D}]],q]`**

`color`: modified example which comes with [color]

`fmft`: modified example which comes with [FMFT]

- reduction of massive four-loop vacuum graphs
- uses **`PolyRatFun`**, one variable

`forcer`: reduction of four-loop propagator integrals with [forcer]

- reduces **`no{1,2,3}`** integrals with some dots and numerators
- uses **`PolyRatFun`**, one variable

`forcer-exp`: as above, with **`ep`**-expansion mode enabled

Current tests

hyperform: modified **Y2.frm** which comes with [hyperform]

- uses **PolyRatFun**

ibp-*: IBP reduction of various topologies using rules from **LiteRed** (See Kay's talk)

- **box2l** : massless two-loop box, vars: **ep,q12,q13**
- **npbox2l**: non-planar version of above
- **mbox1l** : massive one-loop box, vars: **ep,q12,q13,q33,m2**
- **pent1l** : massless one-loop pentagon, vars: **ep,s12,s23,s34,s45,s15**
- **tri3l** : massless three-loop triangle, vars: **ep,q12**
- **nptri3l**: non-planar version of above

mincer: compute Mellin moment of three-loop graviton-exchange DIS graph with [mincer]

- uses **PolyFun**, one variable

minceex: compute Mellin moment of three-loop DIS graph with [mincer-exact]

- uses **PolyRatFun**, one variable

mzv-dm: modified **hexall.frm** from [MZV Datamine]

Current tests

`sort-*`: generate lots of terms, which all cancel late in the sort

- **`small`**: full cancellation in the small buffer
- **`large`**: full cancellation only after sorting the large buffer
- **`disk`** : full cancellation only after sorting disk patches

`trace`: old benchmark of Jos: compute trace of 14 gamma matrices, 1000 times

Add your own tests?

Test files go in **`tests/test-name`**,

- **`tests/test-name/conf.sh`** configures base number of runs and warm-up runs
- benchmarking script runs **`tests/test-name/test-name.frm`** from **`tests/test-name`**
- test should verify correct result, so **`form-bench`** serves as additional offline testing

Getting reliable numbers: “benchmarking is hard”

The numbers in the following slides are mostly from my desktop system at home:

- Ubuntu 24.04, GCC 13.3.0
- AMD Ryzen 9 7900X (Zen 4, 12c24t, up to 5.6GHz — 2x16GB DDR5 6000 MT/s CL30)
  - frequency limited to 5.1GHz, can run continuous all-core load.
  - heat management: reliable numbers from a modern laptop *not easy*
  - allowing a higher single-core boost frequency affects parallel-scaling test runs
- Run with **`nice -n -10`**, on a quiet system
  - try to avoid effects due to other running processes
- Test files and **`FORMTMP`** in **`tmpfs`**
  - try to avoid effects due to disk access times, page caching

Compare mean run-times from **`hyperfine`**, including uncertainties.

Results are always somewhat system dependent!

Performance improvements: expensive copies

[PR #799] **improves a few expensive data-copying loops** [`memmove` for `NCOPY` doesn't change performance]

```
- while ( from < t ) {
-   if ( from == u ) w = m;
-   *m++ = *from++;
- }
```

```
+ LONG copy = t - from;
+ const LONG size = t - u;
+ NCOPY(m, from, copy);
+ w = m - size;
```

```
- while ( j > 0 && fill < top ) {
-   *fill++ = *t++; j--;
- }
```

```
+ LONG copy = MiN(top - fill, j);
+ j -= copy;
+ NCOPY(fill, t, copy);
```

Benchmark	Speedup
`chromatic`	1.05 ± 0.01
`fmft`	1.03 ± 0.01
`forcer`	1.07 ± 0.00
`forcer-exp`	1.08 ± 0.01
`minceex`	1.07 ± 0.02

Performance improvements: `Normalize`

[PR #803] the frequently-called **`Normalize`** function used a ∼90KB stack allocation for temp data

- **cheap in principle, expensive in practice:** stack clash protection
  - each 4k memory page is touched to probe for guard page write

Replace with dynamically allocated memory which lives in thread structs (`AT`)

- some extra logic for nested calls of **`Normalize`**
- additional benefit: **`valgrind`** will detect OOB accesses of each array

```
- WORD psym[7*NORMSIZE];  + WORD *psym = AT.NormData[AT.NormDepth-1]->psym;
- WORD pvec[NORMSIZE];    + WORD *pvec = AT.NormData[AT.NormDepth-1]->pvec;
- WORD pdot[3*NORMSIZE];  + WORD *pdot = AT.NormData[AT.NormDepth-1]->pdot;
- ...                     + ...
```

Benchmark	Speedup	Benchmark	Speedup
`chromatic`	1.10 ± 0.01	`ibp-mbox11`	1.02 ± 0.03
`color`	1.13 ± 0.01	`minceex`	1.10 ± 0.02
`fmft`	1.10 ± 0.01	`mincer`	1.25 ± 0.08
`forcer`	1.04 ± 0.02	`sort-large`	2.03 ± 0.07
`forcer-exp`	1.11 ± 0.01	`trace`	1.33 ± 0.01

Performance improvements: improve SortBot parallelism

See Ana's talk. [PR #831]

Benchmark	Speedup	Benchmark	Speedup
`chromatic`	1.12 ± 0.01	`ibp-mbox11`	1.30 ± 0.03
`color`	1.03 ± 0.02	`minceex`	1.08 ± 0.01
`fmft`	1.14 ± 0.01	`mincer`	1.02 ± 0.01
`forcer`	1.20 ± 0.01	`mzv-dm`	1.01 ± 0.04
`forcer-exp`	1.13 ± 0.00	`sort-large`	1.01 ± 0.01
`hyperform`	1.08 ± 0.01	`trace`	1.00 ± 0.01

Performance improvements: dollar variables

In **`TFORM`**, using **`ModuleOption minimum/maximum`** induced global locks for read and write.

Change: **each worker has a local copy, take the global minimum/maximum at module end.**

- changes run-time behaviour (but only behaviour which was already non-deterministic)
- manual: “The term by term order in which the $-variables obtain their value is not relevant.”

Benchmark	**Speedup** -w24 (SMT)	EPYC 9845 -w32	-w64
`fmft`	1.01 ± 0.00	1.04 ± 0.01	1.06 ± 0.01
`forcer`	1.01 ± 0.00	1.00 ± 0.01	1.00 ± 0.01
`forcer-exp`	1.02 ± 0.00	1.06 ± 0.01	1.02 ± 0.01
`hyperform`	1.01 ± 0.01	1.01 ± 0.01	1.01 ± 0.01
`minceex`	1.00 ± 0.01	1.00 ± 0.02	1.01 ± 0.02
`mincer`	1.06 ± 0.00	1.17 ± 0.01	1.24 ± 0.01
“**`dol-min-max`**”	–	2.23 ± 0.03	2.23 ± 0.04

“**`dol-min-max`**” generate 51.2M x^i terms and just compute min and max powers with $ vars.

Aside: dollar variables

Discrepancy in behaviour between **FORM** and **TFORM**:

- **works in `tform -w>=1`, not in `form`, `tform -w0`**
- **should `form` and `tform -w0` also understand $-variable `ModuleOption`?**

```
Symbol x;

#$min = 101;
Local test = <x^1>+...+<x^100>;

$min = count_(x,1);
*If (Count(x,1) < $min) $min = count_(x,1);

ModuleOption minimum $min;
.sort
#message $min: `$min'
.end
```

Performance improvements: polynomial arithmetic

[PR #838] **improve hotspots in `FLINT` interface, symbol comparison and normalization funcs**

For multivariate `PolyRatFun`, use `FORM`'s lexicographic order and use `FLINT`'s sorted result

- allows us to remove the unnecessary sort back into **`FORM`** ordering
- terms usually arrive in the correct order for the next **`FLINT`** call: sort only if necessary
- **caveat: sometimes the numerator/denominator signs change!**
  - new behaviour is more consistent
  - introduce `On OldPRFSign;` to force old behaviour (with a performance loss)

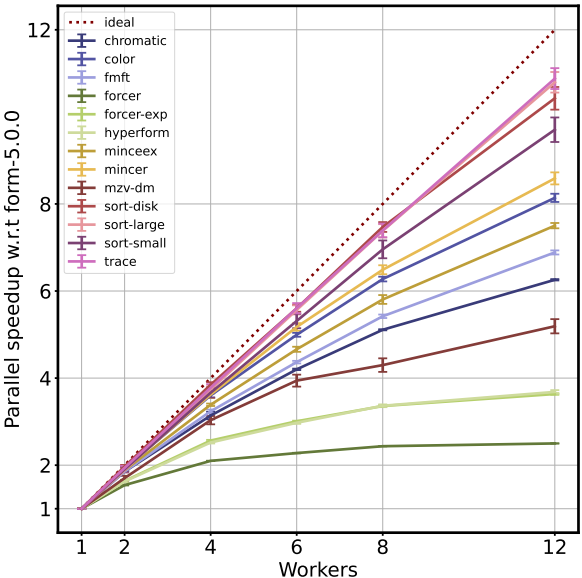
Benchmark	Speedup
`ibp-box2l`	1.09 ± 0.01
`ibp-npbox2l`	1.05 ± 0.01
`ibp-mbox1l`	1.18 ± 0.04
`ibp-pent1l`	1.31 ± 0.04
`ibp-tri3l`	1.01 ± 0.01
`ibp-nptri3l`	1.01 ± 0.02

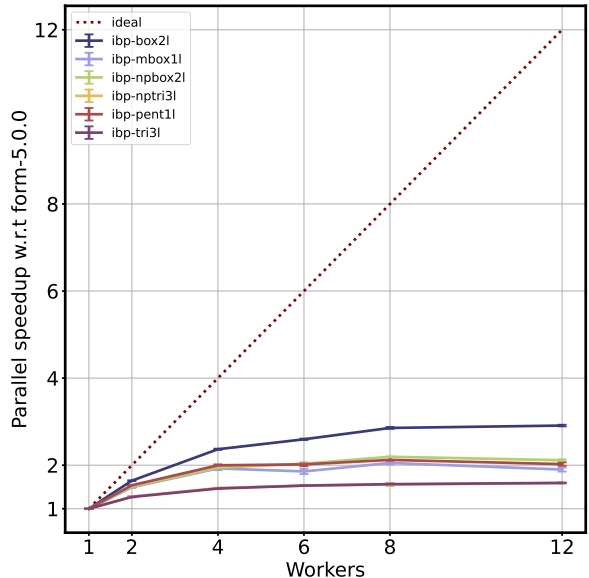
Performance improvements: overall

Combined effect of all of the above (“v5.1.0”), w.r.t. v5.0.0: `tform -w12`

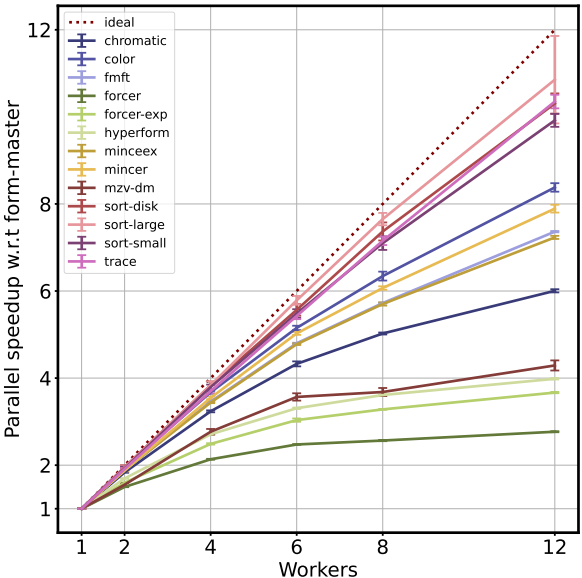
Benchmark	Speedup	Benchmark	Speedup
`chromatic`	1.24 ± 0.01	`ibp-tri3l`	1.03 ± 0.00
`color`	1.24 ± 0.02	`ibp-nptri3l`	1.05 ± 0.01
`fmft`	1.30 ± 0.01	`minceex`	1.23 ± 0.01
`forcer`	1.32 ± 0.01	`mincer`	1.35 ± 0.01
`forcer-exp`	1.37 ± 0.01	`mzv-dm`	1.28 ± 0.02
`hyperform`	1.21 ± 0.01	`sort-disk`	1.35 ± 0.04
`ibp-box2l`	1.46 ± 0.02	`sort-large`	1.75 ± 0.02
`ibp-npbox2l`	1.36 ± 0.01	`sort-small`	1.81 ± 0.04
`ibp-mbox1l`	1.64 ± 0.05	`trace`	1.35 ± 0.01
`ibp-pent1l`	1.71 ± 0.06		

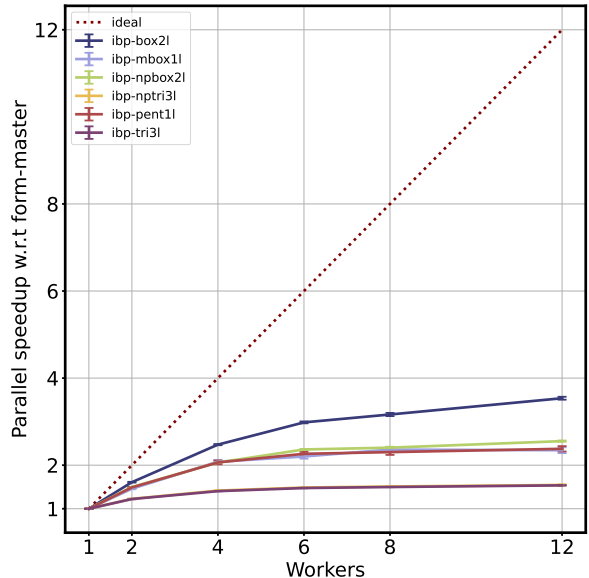
Parallel Scaling: v5.0.0





Parallel Scaling: master





Future work?

Having a standard benchmark suite is important for evaluating performance optimizations, and checking for performance regressions when making changes/fixes which are not targeting performance.

- serves as additional “test suite” with longer run times
- add more real-world tests!

Some relatively simple changes to the code have yielded excellent performance improvements.

- **continued systematic performance profiling and improvement required!**
  - `GETSTOP` in term comparison is expensive due to term layout
  - in polynomial benchmarks, conversion to/from `FLINT fmpz_(m)poly` is now comparatively expensive
  - parallel scaling is often not great: using more than 16 workers is rarely worthwhile
    - particularly if you can parallelise at a higher level (over Feynman diagrams, for eg)