Tips, Tricks and the Future

FORM and Symbolica developers meeting

Josh Davies

29th May, 2024

Introduction

Tips and Tricks:

- Resolving performance bottlenecks in my own computations.
- Not necessarily new to you.
- Not necessarily optimal for all problems (even mine!).
- See also: "**FORM** Cookbook" https://github.com/vermaseren/form/wiki/FORM-Cookbook

The Future:

- Current new features of **FORM 5** (see also Jos' talks): Float mode, Diagram generator.
- Bug fixing and Testing.
- Development ideas and discussion.

Tips and Tricks: Summing many saved expressions

Common operation: e.g. summing individually-computed Feynman diagrams, etc.

Obvious method:

✘ Not parallel in **TFORM**.

```
Load d1000.sav;
Local amp =
  #do i = 1,1000
    + d`i'
  #enddo
  ;
```

Better:

✓ Particularly if sorting is complex (**PolyRatFun**).

☞ **ifmatch** can be crucial! Likely slower than the above, otherwise.

```
Local amp = sum_(x,1,1000,tmp(x));
.sort
#do i = 1,1000
  Identify ifmatch->jump tmp(`i') = d`i';
#enddo
Label jump;
```

Example: sum 5220 files, total 2.5G. **PolyRatFun** used in merging: 2M to 43K terms.

- **442s** → **96s**. (Also: just move **PolyRatFun** to next module: **105s**). (16-worker 3955WX)

[1] But, loading a single large expression is slow. Double copy, single-thread generation.

(Try to avoid this where possible; split into a few smaller expr if terms will never merge anyway.)

Tips and Tricks: Compressing saved expressions

Saved expressions are not compressed (but are very compressible: e.g. **`gzip`** ratio >15x).

Work-around with **`#system`**:

- ✓ Save disk space.
- ✓ Save network bandwidth.
- ☞ Essential for huge expr.

```
Save /tmp/big.sav;
#system gzip < /tmp/big.sav > /network/big.sav.gz
#system rm /tmp/big.sav
```

```
#system gunzip < /network/big.sav.gz > /tmp/big.sav
Load /tmp/big.sav;
#system rm /tmp/big.sav
```

[Also: **`#pipe gunzip < /path/results.h.gz`**]

Another option, use a transparently compressing filesystem (e.g., **`ZFS`**, **`btrfs`**):

- ✓ Save disk bandwidth.
- ✓ Also compresses the scratch (hide) files (**`.sc0`**, **`.sc1`**, (**`.sc2`**))
- ☞ Measurable performance improvement for read/write heavy computations.
- ✘ Easy on your own machines, unlikely on e.g. university HPC resources.

[2] FORM could compress these files. Bonus: way to use other compression algorithms?

Tips and Tricks: Inserting (parts of) expressions into expressions

Useful when, for example:

- Determining renormalization constants (access pole-parts of an expression).
- Solving systems of equations (access coefficient of a symbol).

`dst`: 3K → 24M → 25 terms

- 21s on 8-worker 6850U.
- ✘ Very many accesses of **`src`** bracket contents.
- ☞ **`Bracket+`** : 18s

```
Local src = (1+y)^3 * (1+x)^6;
Local dst = (<f(0)>+...+<f(6)>)^8;
Bracket x;
.sort
#do i = 0,6
  Identify f(`i') = src[x^`i'];
#enddo
```

Better, use dollar variables:

- 2s.
- ✓ Only 7 accesses of **`src`** bracket contents.

```
#do i = 0,6
  #$f`i' = src[x^`i'];
  Identify f(`i') = $f`i';
#enddo
```

(Of course one could do better for this contrived example by generating fewer terms. But generating and sorting 24M terms is not a particularly big deal for **`FORM`**, clearly.)

Tips and Tricks: Hiding away parts of expressions

Often we operate on a subset of objects which appear in the terms [not terms subset: **`Spectator`**]

☞ It can improve performance a lot to hide away the irrelevant content temporarily.

`test` has 47K terms, 2.2MB.

- 88s.
- ✘ 21M terms generated.

`Collect` in function args.:

- Now 51 terms, 3.4MB.
- 22s.
- ✘ Heavy sorting, potential disk IO every module.

Use **`ArgToExtraSymbol`**:

- Now 51 terms, 1.6KB.
- 0.25s.
- ✓ Easy sorting, no disk IO.
- ✘ Costs memory.

```
Local test = (<f(0)>+...+<f(6)>)^6 * (1+x)^50;
Bracket x;
.sort
Collect hide;
#do i = 1,10
  Identify x = x+1;
  Identify x = x-1;
  .sort
#enddo
Identify hide(y?) = y;
```

```
Collect hide;
ArgToExtraSymbol hide;
...
Identify hide(y?) = y;
FromPolynomial;
```

Tips and Tricks: Hiding away parts of expressions (II)

Alternative method using **`Keep Brackets;`** and the **`Term`** environment.

- **`Bracket`** away irrelevant content.
- Use **`Keep Brackets;`** to hold bracket multiplication until the end of the module.
- Use **`Term`** environment to **`Sort;`** before bracket multiplication.

- **`Keep Brackets;`** only: 35s.
- With **`Term`** env: 0.35s.
- ✓ No large memory cost.
- ✘ No disk IO saving.

```
Local test = (<f(0)>+...+<f(6)>)^6 * (1+x)^50;

#do i = 1,10
   Bracket x;
   .sort
   Keep Brackets;
   Term;
      Identify x = x+1;
      Identify x = x-1;
   EndTerm;
#enddo
```

[3] `FORM` could make this easier; extension of the `Keep Brackets;` mechanism?

Tips and Tricks: Inserting IBP tables in FORM

I usually insert IBP tables into amplitudes in **FORM**:

- Amplitude is already in **FORM** format.
- ✘ Doing this in **Mathematica** is usually slow.
- ☞ ("Modern techniques": directly reconstruct amplitude from finte-field samples).

[4] FORM could provide more functionality for finite-field sampling?

For "small problems" (replacing $\mathcal{O}(1000)$ integrals):

- Big list of **Identify** statements is OK (from **Kira**'s **kira2form** output, for e.g.).
- ☞ Adding **ifmatch->jump** construction helps somewhat.

For anything larger, use a **TableBase**.

- Filling a **CTable** "online" OK, but not if you do it many times.
- **Kira** provides **kira2formfill** output for this purpose.

Tips and Tricks: Inserting IBP tables in FORM (II)

For multivariate problems, **MaxTermSize** often becomes an issue. For my problems, I post-process the IBP tables (**Mathematica**) before creating the **TableBase**.

Format:

- Num. **ep** and den. factors containing **ep** outside of **PolyRatFun**.
- **denep** may contain other variables.
- Smaller, but more numerous, terms.

```
+ G54(1,1,1,1,1,1,1,-1,-1) * (
   + denep(ep - 1/2)*ep^-1   * prf( ... )
   + denep(ep - 1/2)*ep      * prf( ... )
   + denep(ep - 1/2)         * prf( ... )
   + denep(ep - 1/2)^2*ep^2  * prf( ... )
   + denep(ep - 1/2)^2*ep    * prf( ... )
   + denep(ep - 1/2)^2       * prf( ... )
```

Same idea, if we will also expand in other variables. E.g., `dent(1+t)*denept(ep-2*t)*t^-1`.

Then follows a series expansion procedure: `#call denexpand(denep,ep,'DEPTH',prf)`.

AntiBracket in **ep,denep,prf** yields bracket content which will never merge with other brackets.

☞ Expand the bracket contents without **.sort**, using **Collect** and then a **Term** environment. (Related to Slides 5, 6).

Tips and Tricks: IBP reduction with LiteRed rules

For small computations/easy integral topologies, it is convinient to compute everything in **FORM**, rather than needing a separate IBP reduction step and then inserting tables after.

- ✓ Use reduction rules determined by **LiteRed** within **FORM**.
- ☞ For multivariate problems, **MaxTermSize** quickly becomes a problem.

The **FORM** code must apply **LiteRed**'s sector mappings, zero sectors, and reduction rules:

```
j[tri111,(n1_)?Positive,(n2_)?NonPositive,(n3_)?Positive]
  /;!(n2==0||n3==1) ->
((1+n2)*q33*j[tri111,n1,2+n2,-2+n3])/((-1+n3)*q11) + ...
```

```
Identify ifmatch->jump tri111(n1?pos_, n2?neg_, n3?{,>1}) =
  tri111(n1,2+n2,-2+n3) * prf(q33+n2*q33,-q11+n3*q11) + ...
```

This is something like a **MINCER**/**FORCER**-style reduction for other integral topologies.

I have a (nasty) bash script to make the conversion, but this could be automated in a cleaner way...

The Future: New Features of FORM 5

Diagram Generator:

- Direct interface to diagram generator (Toshiaki Kaneko).
- Some new syntax: **`Model`**, **`topo_`**, **`diagram_`**, ...

✓ Avoid tricky pattern matching for topology identification.

✘ Currently has some bugs, doesn't work quite as described in the manual.

Floating Point Mode:

- Arbitrary precision float representation of coefficients.
- Reconstruct rational numbers from floating representation.
- **`#StartFloat`**, **`ToRational`**, ...

See Jos' slides for more details.

The Future: Bug Fixing

Many bugs have been fixed (or have a proposed fix) over the last few months (JD, Takahiro Ueda), some of which have made computations tricky and needed to be worked around.

✓	Incorrect **`PolyRatFun`** results due to bugs in sort ordering	(PR #482)
✓	Sorting bugs in arguments or dollar vars	(PR #515 #517 #520)
✓	Crashes in **`MakeInteger`**, **`Transform`** statements	(PR #509 #516)
✓	Crashes for argument-field wildcards with $>$8192 args	(PR #490, #519)
✓	Improved **`Format Mathematica;`** output	(PR #472 #491)
✓	Some improved warnings, syntax fixes	(PR #473 #481 #500 #502 #513)
☞	? Rare crash when loading certain **`Save`** files	(Issue #420, #484)

Bug fixes are in current **`FORM 5`** code, and applied to the **`4.3.1`** branch $\rightarrow$ **`4.3.2`** bugfix release?

The current **`FORM 5`** or **`4.3.1`** branch ready for use for real computations (better than **`v4.3.1`**).

Report bugs/strange behaviour on the Issue tracker! `https://github.com/vermaseren/form/issues`

Also collect tips for developers on the wiki:

- First article, **`VSCode`** (TU): `https://github.com/vermaseren/form/wiki/VS-Code-Tips-for-FORM-Developers`

The Future: Testing

FORM has a **ruby**-based test suite in the **check** directory (Jens Vollinga, Takahiro Ueda).

- Includes examples from the manual, new features, and scripts reproducing (fixed) bugs.
- After making changes, run tests locally with **make check**.
- Runs on **GitHub**'s CI runners on commit:
  - Checks build on **Ubuntu**, **macOS**, **Windows**.
  - Runs tests for **{,t,par}form**, also with **valgrind**.

```
249 tests, 869 assertions, 0 failures, 0 errors, 5 pendings, 0 omissions,
    0 notifications
100% passed
```

More tests means better reliability!

- **Contribute your own tests:** add to **check/user-tests.frm**.
  - Add fold containing your code ***--#[GitHub_username_Test_name :**, and some assertions.
  - Particularly scripts with tricky performance optimizations, workarounds, or use rarely-used features. Tests mean that these scripts should not break in the future.
  - ☞ Should be fast-running, a few seconds at most.

The Future: Testing (II)

```
*--#[ jodavies_example :
Symbol x;
Local test = x;
Identify x = 2;
Print;
.end
assert succeeded?
assert result("test") =~ expr("2")
*--#] jodavies_example :
```

Check for error conditions:

```
assert runtime_error?("Term too complex during normalization")
assert compile_error?("Illegal position for #")
```

Skip test under certain conditions:

```
#require wordsize == 4
#pend_if mpi? || valgrind?
```

See also: `check/README.md`.

The Future: Contributing

Making contributions:

- ☞ Work on your own fork, create a pull request back into **`vermaseren/form`**.

From this workshop hopefully we'll have various ideas of things to implement.

- ✘ Inefficient if we have overlapping efforts on the same changes.
- ☞ Proposal: announce intention to work on something on **`GitHub`** with a draft pull-request/issue.

The Future: Development Ideas

[1] But, loading a single large expression is slow. Double copy, single-thread generation.

- Load data directly from sav files, no double copy?
- Single-thread term generation is also an issue with defining expressions.

[2] `FORM` could compress these files. Bonus: way to use other compression algorithms?

- Compress save files, and also scratch files (tricky: bracketing)?
- Interface to a choice of compression libraries: **`zstd`**, **`snappy`**, etc... ?

[3] `FORM` could make this easier; extension of the `Keep Brackets;` mechanism?

- Sort before multiplying brackets out?
- Multi-module version (tricky: scratch files removed)?
- Option to work inside the brackets, rather than outside?

[**`Term`** env. provides these features to] some extent.]

[4] `FORM` could provide more functionality for finite-field sampling?

- **`Modulus <value>;`** exists already. Add a way to automatically replace symbols with values?

The Future: Development Ideas (II)

`PolyRatFun` performance

- Some room for improvement in the existing code.
- Could just use **`FLINT`** (or optionally **`Symbolica`**)? Perform well for IBP-reduction software.

`Tablebase "name" Open, readonly;`

- Currently **`Tablebase "name" Open;`** requires write access to the file.
- Annoying permissions when sharing with other users.

`On InParallel;`

- Multi-module **`InParallel;`** – easier to use, e.g. when calling **`prc`** which include **`.sort`**.

`On Strict;`

- Mode which enforces some limitations (for e.g., 16 char saved expr name limit)?
- Does not promise backward compatibility?

The Future: Development Ideas (III)

Interoperability: it could be easier to import/export from/to other software:

- Read **Mathematica**-format expressions directly?
  - Deal with **[]**, **I**, **i_**, etc...
  - Have to be a bit careful on the **Mathematica** side, re: nasty denominators and brackets.
- Some intermediate format (JSON?), readable by **FORM**, **Mathematica**, **Symbolica**, ... ?

Improve documentation:

- Make **FORM** more beginner friendly?
  - Update tutorial (André Heck, 2000), add new **FORM** features, content from Jos' lectures.
  - Move to **GitHub** pages (**gh-pages** branch) `https://vermaseren.github.io/form/`
  - More searchable.
- Commentary in the source code?
  - Improve/check for missing **Doxygen** comments?

The Future: Other Things to Discuss?

Time-frame for `FORM 5` release?

- ✘ Certainly there are a few bugs to sort out in the diagram generator.
- ☞ Are there any particular open bugs that YOU would like to see fixed?
- ☞ More regular point releases for bug fixes/minor features?

Is any performance penalty acceptable, in exchange for reliability, better crash info?

- Keep debug symbols in **`form`** build? ["Probably" no noticable effect.
- Split up large memory allocations into multiple small ones? Not benchmarked in detail.]

Drop 32-bit support?

- Already various tests in the suite are disabled for 32-bit **`FORM`** builds.
- It can never support commonly-used large **`MaxTermSize`** for multivariate problems.
- Does anyone use it? Maybe, for e.g., on an older Raspberry Pi model etc.

`ParFORM` support?

- Similarly, some tests in the suite are disabled for **`ParFORM`** (some should be easy to fix).
- Does anyone use it? Modern multi-core high-memory machines out-scale **`FORM`**...
- Fix everything? Keep, but no promises that new features will work with it? Remove?

Windows support?

- Not supported, TU has branches to improve things, including building with **`MSVC`**.
- "Just use WSL"?