Bug fixing

FORM Developers Workshop 2026, Nikhef

Josh Davies

22nd June, 2026

Introduction

Working on performance optimization of a **`form`** script:

- module which replaces momenta in numerator dot products: things blow up a lot
- try: use **`Term`** environment to partially merge terms before main, ground-level sort

Disaster!

```
...
StageSort in thread 2

WTime =    8576.02 sec   Generated terms = 1456135907  (  1 B  )
            d846         Terms in output =   14693810  ( 15 M  )
                 map-3/4 Bytes used      = 2893657696  (  3 GiB)
```

```
...
StageSort in thread 1

WTime =    8604.31 sec   Generated terms = 1456135907  (  1 B  )
            d846         Terms in output =   11943854  ( 12 M  )
                 map-3/4 Bytes used      = 2388380992  (  2 GiB)
```

Let's investigate...

Investigation

Let's go through the investigation together.

1. Try to find a more minimal example.
   - we can't work with a script which creates 1B terms and takes 2.5 hours to run!

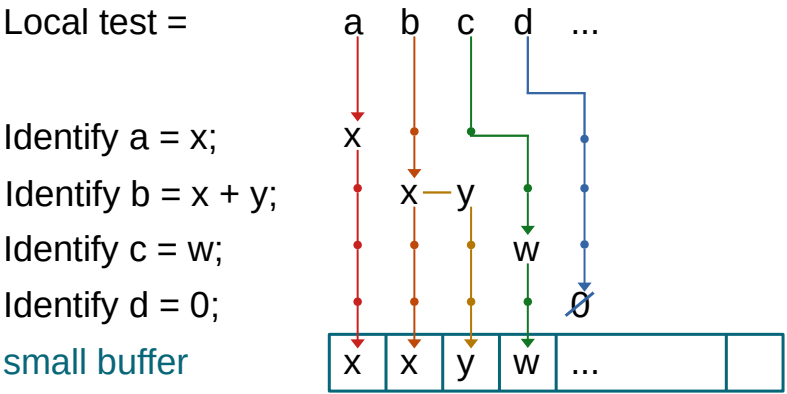
Problem: different numbers of output terms each run

- different terms are produced, non-deterministically? use of uninitialized memory?
- something goes wrong in the sorting?
  - ground-level sort was stage-sorting
  - tried to do heavy operations in **`Term`** environment
  - problems in the past, with sub-buffer sorts reaching stage-sort

Aside: sorting in FORM: step 1

Everything starts with term generation

1. terms come from the input expression
2. manipulate them somehow, possibly generating more terms in the process
   - think of this as a “depth first tree” structure: crucial feature (and sometimes subtle [Issue #835])
3. finished terms go into the **small buffer**



Aside: sorting in FORM: step 2

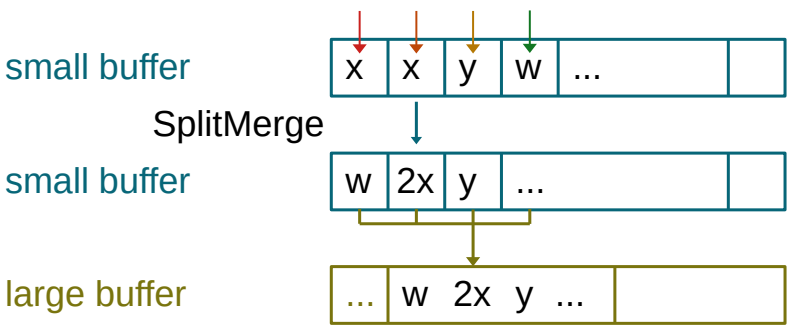
If we have finished generating terms, merge-sort (**`SplitMerge`**) the **small buffer**, copy to **output**

- sort list of pointers to term data, not too much data movement
- extra complication: terms may merge or cancel during sort
- **output** is in memory or on disk (**`.sc0`**, **`.sc1`**), depending on **`ScratchSize`** parameter
- **output** becomes the **input** for the next module

But what if we have not finished generating terms, and the small buffer is full?

- limited by **`TermsInSmall`** or **`SmallSize`**

Sort into a "large patch" in the large buffer!



Aside: sorting in FORM: step 3

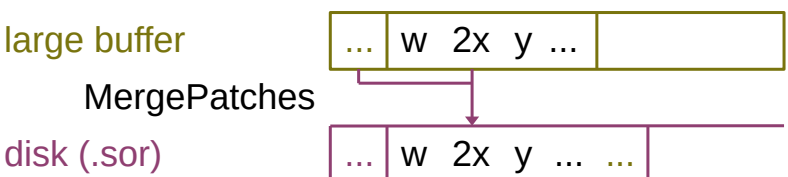
If we have now finished generating terms,

1. sort the **small buffer** into a final **large patch** in the **large buffer**
2. merge all (already sorted) **large patches** into the **output**
   - k-way merge / tournament tree / Knuth loser tree (`MergePatches`)

But what if we have not finished generating terms, and the large buffer is full?

- limited by `LargePatches` or `LargeSize`

Sort into a "file patch" on the disk (`MergePatches`)



Aside: sorting in FORM: step 4

If we have now finished generating terms,

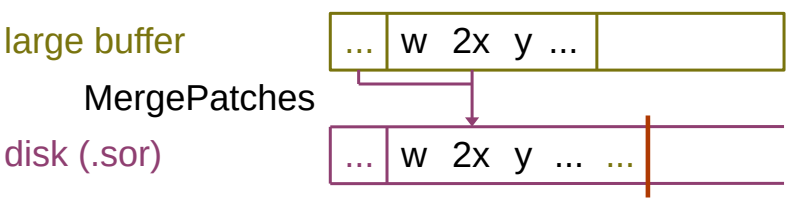
1. sort the **small buffer** into a final **large patch** in the **large buffer**
2. merge all **large patches** into a final **file patch** on the **disk**
3. merge all **disk patches** into the **output** (`MergePatches`)

But... what if we have not finished generating terms, and the disk is "full"?

- limited by `FilePatches` (if the disk is really full, it is game over)

Mark patches as a "stage 4 set" on the disk (`StageSort`)

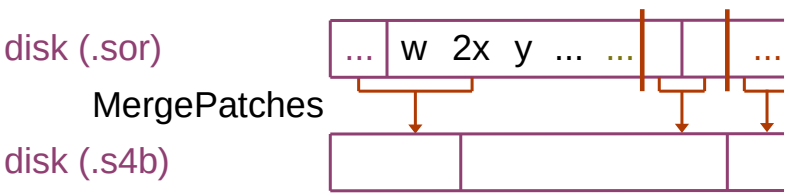
- then continue writing **disk patches** after the mark, **until term generation is finished**
- mark additional **stage 4 sets** if required



Aside: sorting in FORM: finishing up

Now we have finished generating terms, and need to clean everything up.

1. sort the **small buffer** into a final **large patch** in the **large buffer**
2. merge all **large patches** into a final **file patch** on the **disk**
3. if we have created **stage 4 sets**, merge each set into a new “super” **file patch** on the **disk**
   - if this process creates more than `FilePatches` “super” **file patches**, repeat
4. merge all **file patches** into the **output**.



Aside: sorting in FORM: stage sorting

It is not obvious if you reach a “stage 5+” sort:

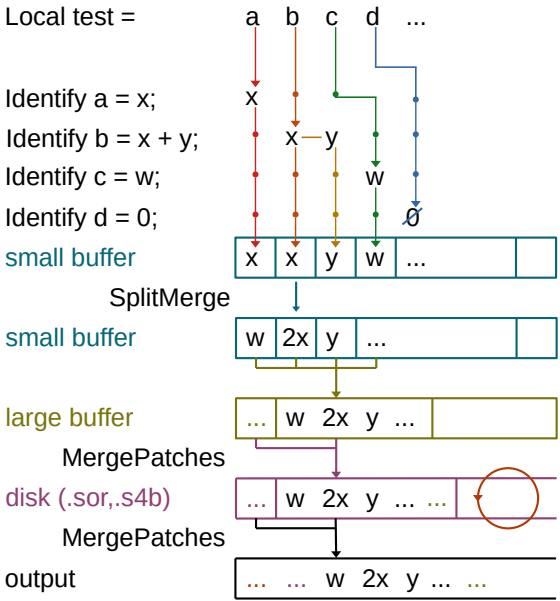
```
#: TermsInSmall 16
#: LargePatches 10
#: FilePatches 10
#define N "
    {16*10*10*10*10+1}"
Off Statistics;
Symbol x,i;
Local test = `N'*(`N'+1)/2
   - sum_(i,1,`N',i);
Print;
.end
```

```
$ strace -e trace=file form-master stage6.frm

openat(AT_FDCWD, "./xform0970804.sor", ...
StageSort
... 98 more
StageSort
openat(AT_FDCWD, "./xform0970804.s4b", ...
StageSort
... 8 more
StageSort
unlink("./xform0970804.sor")              = 0
openat(AT_FDCWD, "./xform0970804.sor", ...
StageSort
unlink("./xform0970804.s4b")              = 0
openat(AT_FDCWD, "./xform0970804.s4b", ...
unlink("./xform0970804.sor")              = 0
unlink("./xform0970804.s4b")              = 0
```

Improve `StageSort` log file information? Add more to `On SortVerbose;`?

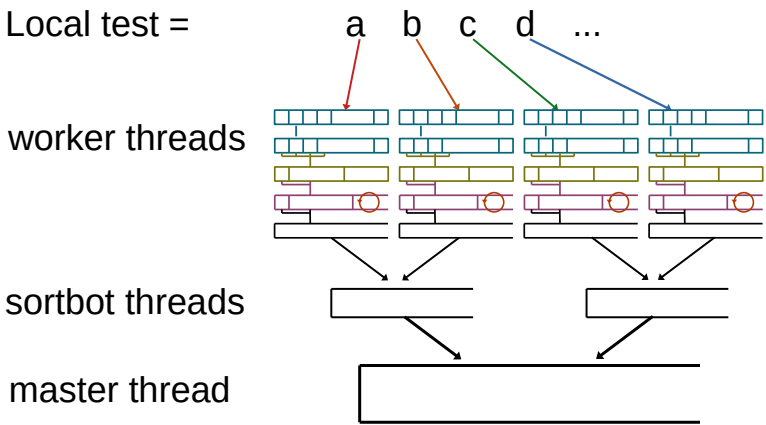
Aside: sorting in FORM: all together now



Aside: sorting in FORM: TFORM

Input shared between worker threads, which each do all of the above.

Parallel merge (with "sortbots") of each worker output into the final output (See Ana's talk).



Aside: sorting in FORM: sub-buffers

While all of this is going on, “higher-level” sorting happens in the “sub-buffers”:

- sorting of function arguments, dollar variables, `Term` environment
- buffer sizes: `SubTermsInSmall`, `SubSmallSize`, `SubLargePatches`, `SubLargeSize`, ...

This works in the same way as the ground-level sort, except “stage 4 sorting” is not allowed.

- Files can be created here: not good for performance. Best to increase sub-buffers if possible.
- Currently, it is not printed in the log file that this has happened!
  - **We should print a warning (but only once!) – Exercise**

```
Warning: sort file created in sub-buffer sort. Consider increasing sub-sizes. ?
```

Investigation: minimal script

Experiment with `N` and `BLOWUP` until something breaks!

```
#: TermsInSmall 16
#: LargePatches 10
#: FilePatches 10
#: SubTermsInSmall 16
#: SubLargePatches 10
#: SubFilePatches 10

CFunction f,sum;
Symbol i,j,x;
#define N "{1776+1}"
#define BLOWUP "{160+1}"

Local test = {1 + `N'*(`N'+1)/2} - <x^1>-...-<x^`N'>;
.sort
Term;
  Identify x^i?pos_ = sum_(j,1,`BLOWUP',i/`BLOWUP');
EndTerm;
Print;
.end
```

Investigation: minimal script

```
    #-
StageSort
StageSort

   test =
       - 300553;
```

N = 1777: `TermsInSmall*LargePatches*(FilePatches + 1) + TermsInSmall + 1`

BLOWUP = 161: `SubTermsInSmall*SubLargePatches + 1`

$$300553 = \sum_{i=183}^{796} i, \quad \sum_{i=826}^{1132} i, \quad \sum_{i=1600}^{1777} i \tag{1}$$

Investigation

Now that we have a nice simple script:

1. ~~Try to find a more minimal example.~~
2. Just `tform`? No, `form` also. ☺
3. Any `valgrind` errors? No. ☹
4. Can we bisect? No. ☹
   - ✘ master
   - ✘ 5.0
   - ✘ 4.3
   - ✘ 4.2
   - ✘ 4.0
5. `gdb`? Not so easy, it doesn't crash, 1000s of terms... ☹
6. **Good old-fashioned `printf` debugging!** 😐

Investigation

Functions to target for sorting-related issues:

- **`StoreTerm`**: generated terms enter the sorting system here
- **`PutOut`**: sorted (or intermediate-sorted) term written to some file
- **`MergePatches`**: merges large-buffer or file patches
- **`EndSort`**: after term generation, merges everything into a single sorted output

For e.g. add to **`StoreTerm`**: [See step 1 log files]

```
if ( ! AR.sLevel ) { MesPrint("StoreTerm level:%d lPatch:%d fPatchN:%d ter:[%a]",
   AR.sLevel, S->lPatch, S->fPatchN, *term, term); }
...
if ( ! AR.sLevel ) { MesPrint("---small buffer full"); }
...
if ( ! AR.sLevel ) { MesPrint("---large buffer full"); }
```

and to **`PutOut`**:

```
if ( ! AR.sLevel ) { MesPrint("PutOut level:%d lPatch:%d fPatchN:%d file:%s ter:[%
    a]",
   AR.sLevel, AT.SS->lPatch, AT.SS->fPatchN, fi->name, *term, term);}
```

Investigation

Writing `4 300553 1 -3` to the `.sc1` file at the end is weird! Expect `4 1 1 3`.

Let's check what happens in more detail in **`MergePatches`** and **`EndSort`**. [See step 2 log files]

- add more messages to **`EndSort`**, **`MergePatches`**, **`StageSort`**

In the working case, we call **`EndSort`** with **`sTerms:2`**, **`lPatch:1`**, **`fPatchN:1`**, **`stage4 sets:1`**

1. Sort **small buffer** → 2nd **large patch**
2. Merge 2 **large patches** → 2nd **file patch** in **`.sor`**
3. Call **`MergePatches(0)`** for final sort to output
4. **StageSort!** Merge 2 **file patches** into **`.s4b`**, then 10 **file patches** into **`.s4b`**
   - **this means taking `goto ConMer;` to start the second merge**
5. Take **`goto NewMerge;`** and merge 2 **file patches** (from **`.s4b`**) to output **`.sc1`**. **DONE.**

In the bad case, step **4.** is missing.

- **why is it missing, and what does it have to do with the higher-level sort?** [See **`sort.c`**]

Mystery solved!

Taking `goto ConMer;` is controlled by the value of **`AN.OldPosIn`**.

- For some reason **`AN.OldPosIn`** is 0, and we skip the stage sort. **Why?**
  - Only the second **stage 4 set** of patches ends up in the output: terms 1600 - 1777!
- **`AN.OldPosIn`** is only set once, to **`AN.OldPosOut`**
- **`AN.OldPosOut`** is set to 0, when we first sort the **large buffer** and create a sort file
- **`AN` is a thread-local global struct! It is not private to the sort.**
  - The higher-level sort creating a sort file re-set **`AN.OldPosOut`** to 0.

Solution: [PR #843]

- Move **`OldPosIn`** and **`OldPosOut`** from **`AN`** to the **`SORTING`** struct.
  - Every sort, at any level, has its own private values.

Related: moving the rest of the stage-4 related variables (**`FoStage4`**, etc, make sure file names are unique) to **`SORTING`** would allow us to re-enable higher-level stagesorts, if we want.

If you have run jobs with StageSort, with heavy sub-buffer sorts, you should re-run them...