Make pack access code thread-safe

April, 2019

Contact Info

Name Matheus Tavares Bernardino
Timezone GMT-3
IRC Nick matheustavares on #git-devel
Github https://github.com/matheustavares/
Gitlab https://gitlab.com/matheustavares
Email
Telefone
Postal address

About me

I'm a senior student at the University of São Paulo (USP), attending the *Bachelor's degree in Computer Science* course. Currently, I'm at the end of a one year undergraduate research in High-Performance Computing. The goal of this project was to accelerate astrophysical software for black hole studies using GPUs. Also, I'm working as a teaching assistant on IME-USP's Concurrent and Parallel Programming course, giving lectures and developing/grading programming assignments. Besides parallel and high-performance computing I'm very passionate about software development in general, but especially low-level coding, and FLOSS.

About me and FLOSS

Linux Kernel

Last year, I started contributing to the Linux Kernel in the IIO subsystem, together with a group of colleagues. I worked with another student, to move the ad2s90 module out of staging area to Kernel's mainline, which we accomplished by the end of the year. In total, I authored 11 patches and co-authored 3 (all of which are already at Torvald's repo). If you want to know more about my contributions to Linux Kernel, take a look at the Appendix section.

FLUSP: FLOSS at USP

After the amazing experience contributing to the Linux Kernel, we decided to found FLUSP: FLOSS at USP, a group opened to undergraduate and graduate students that aims to contribute to FLOSS software. Since then, the group has grown and evolved a lot: Currently, we have members contributing to the Kernel, GCC, IGT GPU Tools, Git and some projects of our own such as KernelWorkflow. And as a recognition of our endeavor with free software, we received some donations from AnalogDevices and DigitalOcean.

Besides administrative questions and contributions to FLOSS projects, at FLUSP, I've been mentoring people who want to start contributing to the Linux Kernel and now, to Git, as well.

About me and Git

I joined Git community in February and, so far, I have sent the following patches:

Patch	Status
clone: test for our behavior on odd objects/* content	v5 sent for review
clone: better handle symlinked files at .git/objects/	v5 sent for review
dir-iterator: add flags parameter to dir_iterator_begin	v5 sent for review
clone: copy hidden paths at local clone	v5 sent for review
clone: extract function from copy_or_link_directory	v5 sent for review
clone: use dir-iterator to avoid explicit dir traversal	v5 sent for review
clone: Replace strcmp by fspathcmp	v5 sent for review

And three more patches for git.github.io:

Patch	Status
rn-50: Add git-send-email links to light readings	Merged
SoC-2019-Microprojects: Remove git-credential-cache	Merged
SoC-2019-Microprojects: Remove all trailing spaces	Merged

Participating at FLUSP, I've also been part of some Git related activities:

- I actively helped to organize a Git workshop for newcomer students.
- I've written an article at our website to help people configure and use git-send-email to send patches.

- I've been writing a 'First steps at Git' article (not finished yet), in which I'm registering what I've learned in the Git community so far, since downloading the source, subscribing to the mailings list and joining the channel at IRC until how to use travis-ci and begin sending patches.

The Project

As direct as possible, the goal with this project is to **make more of Git's codebase thread-safe,** so that we can **improve parallelism in various commands**. The motivation behind this are the complaints from developers experiencing slow Git commands when working with large repositories[1], such as chromium and Android. And since nowadays, most personal computers have multi-core CPUs, it is a natural step trying to improve parallel support so that we can better use the available resources.

With this in mind, pack access code is a good target for improvement, since it's used by many Git commands (e.g., checkout, grep, blame, diff, log, etc.). This section of the codebase is still sequential and has many global states, which should be made local, removed or protected before we can work to improve parallelism.

The Pack Access Code

To better describe what the pack access code is, we must talk about Git's object storage (in a simplified way): Besides what are called *loose object files*, Git has a very optimized mechanism to compactly store objects (blobs, trees, commits, etc.) in packfiles[2]. These files are created by[3]:

1. listing objects;
2. sorting the list with some good heuristics;
3. traversing the list with a sliding window to find similar objects in the window, in order to do delta decomposing;
4. compress the objects with zlib and write them to the packfile.

---

[1] Some of them can be seen here:
https://groups.google.com/a/chromium.org/forum/#!topic/chromium-dev/oYe69KzyG_U
https://bugs.chromium.org/p/git/issues/detail?id=18
https://bugs.chromium.org/p/git/issues/detail?id=16
https://code.fb.com/core-data/scaling-mercurial-at-facebook/
https://public-inbox.org/git/CA+TurHgyUK5sfCKrK+3xY8AeOg0t66vEvFxX=JiA9wXww7eZXQ@mail.gmail.com/
https://public-inbox.org/git/20140213014229.GE4582@vauxhall.crustytoothpaste.net/
https://public-inbox.org/git/CACBZZX6A+35wGBYAYj7E9d4XwLby21TLbTh-zRX+fkSt_e2zeg@mail.gmail.com/

[2] https://git-scm.com/book/en/v2/Git-Internals-Packfiles
[3] https://github.com/git/git/blob/master/Documentation/technical/pack-heuristics.txt

What we are calling *pack access code* in this document, is the set of functions responsible for retrieving the objects stored at the packfiles. This process consists, roughly speaking, in three parts:

1. Locate and read the blob from packfile, using the index file;
2. If the blob is a delta, locate and read the base object to apply the delta on top of it;
3. Once the full content is read, decompress it (using zlib inflate).

Note: *There is a delta-base cache for the second step so that if another delta depends on the same base object, it is already in memory. This cache is global.*

Note 2: *When reading from the packfile, Git uses windows to map regions of the packfile into memory for better performance. (Do not confuse these with the windows used when* ***writing*** *to packfiles). These windows are global per packfile and all packfile attributes are hold by a global variable named "the_repository".*

If the previously shown steps were thread-safe, the ability to perform the delta reconstruction (together with the delta-base cache lookup) and zlib inflation in parallel could bring a good speedup. At git-blame, for example, 24%[4] of the time is spent in the call stack originated at *read_object_file_extended.* Not only this but once we have this big section of the codebase thread-safe, we can work to parallelize even more work at higher levels of the call stack. Therefore, with this project, we aim to make room for many future optimizations in many Git commands.

Plan

I will probably be working mainly with *packfile.c*, *sha1-file.c*, *object-store.h, object.c* and *pack.h*, however, I may also need to tackle other files. I will be focusing on the following three pack access call chains, found in git-grep and/or git-blame:

read_object_file → repo_read_object_file → read_object_file_extended → read_object → oid_object_info_extended → find_pack_entry → fill_pack_entry → find_pack_entry_one → bsearch_pack and *nth_packed_object_offset*
oid_object_info → oid_object_info_extended → <same as previous>
read_object_with_reference → read_object_file → <same as previous>

[4] With gprof and gprof2dot I generated the following image: https://i.imgur.com/XmyJMuE.png; which shows some of the most time consuming functions when git-blame is invoked. The call stack originate at read_object_file_extended() is responsible for pack access and consumes 24% of the total execution time.

Please note that I presented just some linear sections of these call stacks here, for simplicity. But they are much more three-alike, involving many other functions not included here that I will probably need to tackle as well.

Ideally, at the end of the project, it will be possible to call *read_object_file, oid_object_info* and *read_object_with_reference* with thread-safety, so that these operations can be, latter, performed in parallel.

Here are some threads on Git's mailing list where I started discussing my project:

- https://public-inbox.org/git/CAHd-oW7onvn4ugEjXzAX_OSVEfCboH3-FnGR00dU8iaoc+b8=Q@mail.gmail.com/
- https://public-inbox.org/git/20190402005245.4983-1-matheus.bernardino@usp.br/
- https://public-inbox.org/git/CAHd-oW7KMrDJ-cyzk63oqW9-QVpag6fKnDp+Mo5bWxg1KfzY3g@mail.gmail.com/

And also, a previous attempt to make part of the pack access code thread-safe which I may use as a base:

- https://public-inbox.org/git/20140212015727.1D63A403D3@wince.sfo.corp.google.com/

Points to work on

- Protect *packfile.c* global variables such as *pack_open_windows* and *pack_open_fds*, which are read and updated in sections that must be thread-safety.
- Just like the previous item, protect *sha1-file.c* global states such as the *cached_objects* array used by read_object_file(). This cache is intended to hold a small number of objects in memory pretending that they are available at the object storage but not really writing them to disk.
- Protect operations on the delta-base cache. Here we should study whether to add mutexes to the cache itself or to the underlying hashmap. There are advantages and disadvantages in both cases, so it should still be discussed with the community. A third option would be to try making the cache thread-local. But this could perform badly since some objects may be in one thread's cache but not the other's. And memory usage would increase because of duplicate information.
- Protect access to the windows used to read from packfiles. This is the section that I least know so far, so I will still have to study it deeper.
- Investigate pack access call chains, using GDB or GPROF, and look for other unprotected operations on global states. For each of these global states, we'll have

to decide between: making it local (embedded thought call chains), removing it (e.g. calculating the value, when needed, from other variables) or protecting it with mutexes (and perhaps adding protected access macros for them).
- Look for functions in the pack access code with static variables in their scopes. These variables are thread-shared and should be protected or the functions to which they belong be refactored.
- Take care of sequential code performance. It is expected that when we start adding mutexes, the sequential code will perform slower because calling *pthread_mutex_lock()* and *pthread_mutex_unlock()* is quite time consuming. So we should make some macros for this two functions at pack access code to skip these calls when we are working single threaded.
- Make sure tests cover functions I'll be working on and refactor/add tests as needed. It's not at all simple to do tests that check whether a function is thread-safe or not. So for this thread-safety conversion phase, I will probably rely on the existing tests to ensure refactoring doesn't break anything. Them, when we incrementally and slowly add threading (which may happen during or after GSoC), more tests should be added. And that's when we should better see if something is still non-thread-safe.
- [Bonus] Once pack access is thread-safe, refactor the critical sections at git-grep to use more fine-grained mutexes. This will hopefully increase git-grep performance, especially in large repositories.
- [Bonus] Check other mutex protected functions git-grep uses, not related with pack access, to see if we can implement a more fined-grained parallelism there. This functions are: *fill_textconv, is_submodule_active, repo_submodule_init, repo_read_gitmodules* and *add_to_alternates_memory.*
- [Bonus] Once pack access is thread-safe, ensure xdiff code used by git-blame has thread-safety. I expect this to be easier.
- [Bonus] If the previous bonus get completed, start discussing a possible parallel git-blame implementation with the community. We could work a producer-consumer mechanism at blame.c's assign_blame() function, for a very good work sharing assignment (90% of git-blame's time is spent here[5]). Or try threading at lower functions on the call stack that still uses a lot of execution time such as the libxdiff ones.

Schedule

[5] https://i.imgur.com/XmyJMuE.png

This is the planned schedule in which I should be working on. But I would like to highlight that since there's still a significant investigation period from now until early May, this can have some changes or additions during the process.

Period	**Main Task**	**Side Tasks**
Now - May, 5	Study pack access code (investigation time)	• Gather information of global states. • Trace pack access call chain used by git commands like blame and checkout. • Try to classify which global variables are updated during pack access call stack and, therefore, should be protected or made local. Also try to classify which ones can we make local or remove. • Adjust the schedule as needed.
May, 6 - 26[6]	Community Bounding and work on sha1-file.c global states	• Talk with the community about my then refined plan and ask for comments. • Protect object cache at sha1-file.c. • Work on other sha1-file.c global states and non-thread-safe functions.
May, 27 - June, 23	Work on packfile windows and other packfile.c global states	• Protect access to the windows used to map packfiles' regions into memory. • Protect packfile.c global states (*pack_open_windows*, *pack_open_fds* and etc.) and work on its non-protected functions.
June, 24 - July, 21	Work on delta-base cache	• Conclude work on packfile windows. • Protect delta-base cache operations.
July, 22 - August 19	Work on bonus and leftovers	This time will be reserved to finish any leftovers from the previous

[6] GSoC's official starting date

	Note: *I also plan to attend DebConf from july 21th to 28th*	periods and, if we still have some spare time, work on the bonus items.

Availability

My university vacations start on June 29, but since this is my last year and I'm attending just two courses plus the teaching assistance, I think it won't be a problem. Also, the classes start back in early August, but I won't have any more courses to attend in this next semester. I don't have any schedule trips besides DebConf, from July 21th to 28th (let me know if any other Git community members plan to attend too, please). All changes in availability will be communicated to the mentors in advance.

Project Relevance and after GSoC plans

As already pointed out, this project will make it feasible to improve (or add) parallelism in many Git commands which, in turn, can help many developers working on large repositories. And that's what I plan to do after [or even start during] GSoC, mainly with git-blame and git-grep. In fact, I'm working on my capstone project this whole year, whose theme is in Git. It will be divided into two fronts: In the main (and technical) side, I plan to work on Git's performance with multithreading and maybe on some other smaller patches, too. In the social side, I'm also trying to form a local community at FLUSP to contribute to Git. For this task, I will be helping other students and writing blog posts to FLUSP's website about Git development and the code snippets I'll be working on. This capstone project may also be sponsored with a scholarship by a research funding institution, but my advisor and I haven't submitted the proposal yet.

Appendix

Patches at Linux Kernel

Patches
staging: iio: ad2s1210: fix 'assignment operator' style checks
staging:iio:ad2s90: Make read_raw return spi_read's error code
staging:iio:ad2s90: Make probe handle spi_setup failure
staging:iio:ad2s90: Remove always overwritten assignment

staging:iio:ad2s90: Move device registration to the end of probe
staging:iio:ad2s90: Add IIO_CHAN_INFO_SCALE to channel spec and read_raw
staging:iio:ad2s90: Check channel type at read_raw
staging:iio:ad2s90: Add device tree support
staging:iio:ad2s90: Remove spi setup that should be done via dt
staging:iio:ad2s90: Add max frequency check at probe
dt-bindings:iio:resolver: Add docs for ad2s90
staging:iio:ad2s90: Replace license text w/ SPDX identifier
staging:iio:ad2s90: Add comment to device state mutex
staging:iio:ad2s90: Move out of staging