Parallelizing Rename Detection in Git

Samuel Lijin and Harrison Stall

What is a File Rename in Git?

- Users can make changes to both the contents and name of the file between commits
- Git recognizes these filename changes and will properly detect a single file lineage, even through rename commits
- In the process of performing diffs across commits that include file renames, git must calculate file similarities and rename possibilities on the fly

Git Renaming Semantics

- If m rename source files are part of a commit lineage with n possible destination rename files, m * n comparisons must be made between between files to determine which source file corresponds to each destination file
- These calculations happen sequentially; a function called estimate_similarity() executes for each file-pair to determine the highest similarity between files
- If this highest value is greater than a given threshold, Git labels the file destination file a renamed version of the source file
- `gif diff` properly accounts for changes in names to allows cross name-change commit diffs using this similarity estimation
- Our project parallelized the process of running `estimate_similarity()` and detecting renames across the m * n combinations

New Rename Detection Design

1. Master thread prepares matrix of diff_scores to compute.
2. Master thread kicks off the thread pool.
3. Master thread waits on all worker threads.

1. Worker thread receives pointer to matrix of scores to compute and pointer to a column in the matrix.
2. Worker thread locks matrix, increments the column pointer, and unlocks the matrix; i.e. treats the matrix as a concurrent queue of jobs, and consumes a set of jobs from it.
3. Worker thread computes diff_scores in the column the column pointer previously pointed to.
4. Worker thread repeats (2) and (3) until all work has been taken.

Performance

- All tests were performed on a two core machine running Linux on a VM with two allocated processors
- We ran the Git-provided test suite to ensure changes were non-breaking
- We developed a custom test suite to measure performance (check finaldesign.md for more detail + code)

Experiment #	Time elapsed in seconds (stable v2.12.2)	Time elapsed in seconds (patched version)	Page Faults (v2.12.2)	Page Faults (patched)
1	104.23	106.36	7924930	8783578
2	103.96	106.04	7927112	8764349
3	104.15	106.00	7928092	8777599
σ of each set of samples	0.13868	0.19732	1618.6	9840.9

Note About a Slight Adjustment in the Project

The first iteration of our project involved running git diffs in parallel but after a full day of investigation we determined that several design decisions within the Git codebase make it extraordinarily difficult to parallelize diffs. Git writes to STDOUT as it calculates each individual diff, with little care for the order in which the files are listed. Synchronization between these diffs if running in parallel would require architectural shifts that we a) knew would be 100% shut down by the Git maintainers who review PRs for the repository and b) would have required significant changes far beyond the scope of the final project for this class. Instead, we were able to parallelize rename detection, our second idea that Peff recommended.

More info can be found in finalreport.md.