Second-generation PLINK: Rising to the challenge of larger and richer datasets

Article *in* GigaScience · October 2014

DOI: 10.1186/s13742-015-0047-8 · Source: arXiv

6 authors, including:

Christopher C. Chang
Complete Genomics Incorporated
6 PUBLICATIONS **952** CITATIONS
SEE PROFILE

Carson Chow
The National Institute of Diabetes and Digestive and Kidney Diseases
211 PUBLICATIONS **8,533** CITATIONS
SEE PROFILE

Laurent Christian Asker Melchior Tellier
Genomic Prediction Inc
24 PUBLICATIONS **972** CITATIONS
SEE PROFILE

Shashaank Vattikuti
U.S. Department of Health and Human Services
28 PUBLICATIONS **1,579** CITATIONS
SEE PROFILE

TECHNICAL NOTE

Second-generation PLINK: rising to the challenge of larger and richer datasets

Christopher C Chang[1,2*], Carson C Chow[3], Laurent CAM Tellier[2,4], Shashaank Vattikuti[3], Shaun M Purcell[5,6,7,8] and James J Lee[3,9]

*Correspondence:
chrchang@alumni.caltech.edu
[1]Complete Genomics, 2071 Stierlin Court, 94043 Mountain View, CA, USA
Full list of author information is available at the end of the article

Abstract

Background: PLINK 1 is a widely used open-source C/C++ toolset for genome-wide association studies (GWAS) and research in population genetics. However, the steady accumulation of data from imputation and whole-genome sequencing studies has exposed a strong need for even faster and more scalable implementations of key functions. In addition, GWAS and population-genetic data now frequently contain probabilistic calls, phase information, and/or multiallelic variants, none of which can be represented by PLINK 1's primary data format.

Findings: To address these issues, we are developing a second-generation codebase for PLINK. The first major release from this codebase, PLINK 1.9, introduces extensive use of bit-level parallelism, $O(\sqrt{n})$-time/constant-space Hardy-Weinberg equilibrium and Fisher's exact tests, and many other algorithmic improvements. In combination, these changes accelerate most operations by 1-4 orders of magnitude, and allow the program to handle datasets too large to fit in RAM. This will be followed by PLINK 2.0, which will introduce (a) a new data format capable of efficiently representing probabilities, phase, and multiallelic variants, and (b) extensions of many functions to account for the new types of information.

Conclusions: The second-generation versions of PLINK will offer dramatic improvements in performance and compatibility. For the first time, users without access to high-end computing resources can perform several essential analyses of the feature-rich and very large genetic datasets coming into use.

Keywords: GWAS; Population genetics; Whole-genome sequencing; High-density SNP genotyping; Computational statistics

Findings

Because of its broad functionality and efficient binary file format, PLINK is widely employed in data-processing pipelines set up for gene-trait mapping and population-genetic studies. The five years since the final first-generation update (v1.07), however, have witnessed the introduction of new algorithms and analytical approaches, the growth in size of typical datasets, and wide deployment of heavily multicore processors.

In response, we have developed PLINK 1.9, a comprehensive performance, scaling, and usability update. Its speed improvements are the most notable: our data indicate that speedups frequently exceed two, and sometimes even three, orders of magnitude for several commonly used operations. Its core functional domains are unchanged from that of its predecessor (data management, summary statistics,

population stratification, association analysis, identity-by-descent estimation [1]), and it is usable as a drop-in replacement in most cases, requiring no changes to existing scripts. To support easier interoperation with newer software like BEAGLE 4 [2], IMPUTE2 [3], GATK [4], VCFtools [5], BCFtools [6], and GCTA [7], features such as the import/export of VCF and Oxford-format files and an efficient cross-platform genomic relationship matrix (GRM) calculator have been introduced. Most pipelines currently employing PLINK can expect to benefit from upgrading.

A major problem remains: PLINK's core file format can only represent unphased, biallelic data. We are developing a second update, PLINK 2.0, to address this.

Improvements in PLINK 1.9

Bitwise parallelism

Modern x86 processors are designed to operate on data in (usually 64-bit) machine word or ($\geq$ 128-bit) vector chunks. The PLINK 1 binary file format supports this exceptionally well: its packed 2-bit data elements can, with the use of bit arithmetic, easily be processed 32 or 64 at a time. However, most existing programs fail to exploit opportunities for bitwise parallelism; instead their loops painstakingly extract and operate on a single data element at a time. Replacement of these loops with bit-parallel logic is, by itself, enough to speed up numerous operations by more than one order of magnitude.

For example, the old identity-by-state calculation proceeded roughly as follows:

For every sample pair (i, j):
  For every marker k:
    1 If either i_k or j_k is a missing call, skip
    2 If $i_k = j_k$, increment IBS2 count
    3 otherwise, if both bits differ, increment IBS0 count
    4 otherwise, increment IBS1

We replaced this with:

For every sample pair (i, j):
  For every 960-marker block K:
    1 Evaluate i_K XOR j_K
    2 Mask out markers with missing calls
    3 Count number of set bits

Refer to Additional file 1 for a detailed walkthrough. Our timing data (see "Performance comparisons" below) indicate that PLINK 1.9 takes less than twice as long to handle a 960-marker block as PLINK 1.07 takes to handle a single marker.

Bit population count

The last step above—bit "population count"—merits further discussion. Post-2008 x86 processors support a specialized instruction that directly evaluates this quantity. However, thanks to 50 years of work on the problem, algorithms exist which evaluate bit population count nearly as quickly as the hardware instruction, while sticking to

universally available operations. Since PLINK is still used on some older machines, we took one such algorithm (previously discussed and refined by Dalke, Harley, Lauradoux, Mathisen, and Walisch [8]), and developed an improved SSE2-based implementation. (Note that SSE2 vector instructions are supported by even the oldest x86-64 processors.)

The applications of bit population count extend further than might be obvious at first glance. As an example, consider computation of the correlation coefficient r between a pair of markers, where some data may be missing. Letting x and y denote the markers, $i \in S$ denote sample indices, define x_i and y_i to be -1 when the corresponding genotype call is homozygous minor, 0 when the corresponding call is heterozygous or missing, and $+1$ when the corresponding call is homozygous major. Also define S_{xy} to be the subset of S for which x and y do not have missing calls, $\overline{x} := |S_{xy}|^{-1} \sum_{i \in S_{xy}} x_i$ (similarly for $\overline{y}$), and $\overline{x^2} := |S_{xy}|^{-1} \sum_{i \in S_{xy}} x_i^2$ (similarly for $\overline{y^2}$). ($|\cdot|$ denotes set size.) The correlation coefficient can then be expressed as

$$\begin{aligned} r &= \frac{|S_{xy}|^{-1} \sum_{i \in S_{xy}} (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{(\overline{x^2} - \overline{x}^2)(\overline{y^2} - \overline{y}^2)}} \\ &= \frac{|S_{xy}|^{-1} \sum_{i \in S} x_i y_i - \overline{x} \cdot \overline{y}}{\sqrt{(\overline{x^2} - \overline{x}^2)(\overline{y^2} - \overline{y}^2)}} \end{aligned}$$

Given PLINK 1 binary data, $|S_{xy}|$, $\overline{x}$, $\overline{y}$, $\overline{x^2}$, and $\overline{y^2}$ can easily be expressed in terms of bit population counts. (When no missing calls are present, these values can be precomputed since they do not vary between marker pairs; but in the general case, it is necessary to recalculate them all in the inner loop.) The dot product $\sum_{i \in S} x_i y_i$ is trickier; to evaluate it, we preprocess the data so that the genotype bit vectors G_x and G_y encode homozygote minor calls as 00_2, heterozygote and missing calls as 01_2, and homozygote major calls as 10_2, and then proceed as follows:

1. Set G_z `:=` $(G_x$ `OR` $G_y)$ `AND` $01010101\ldots_2$
2. Evaluate
   `popcount2(((`G_x `XOR` $G_y)$ `AND (`$10101010\ldots_2$ `-` $G_z))$ `OR` $G_z)$,
   where `popcount2()` sums 2-bit quantities instead of counting set bits. (This is actually cheaper than regular population count; the first step of software `popcount()` is reduction to a `popcount2()` problem.)
3. Subtract the latter quantity from $|S|$.

The key insight behind this implementation is that each $x_i y_i$ term is in $\{-1, 0, 1\}$, and can still be represented in 2 bits. (This is not strictly necessary for bitwise parallel processing—the partial sum lookup algorithm discussed later handles 3-bit outputs by padding the raw input data to 3 bits per genotype call—but it allows for unusually high efficiency.) The exact sequence of operations that we chose to evaluate the dot-product terms in a bitwise parallel fashion is somewhat arbitrary.

See `popcount_longs()` in `plink_common.c` for our primary bit population count function, and `plink_ld.c` for several correlation coefficient evaluation functions.

Multicore and cluster parallelism

Modern x86 processors also contain increasing numbers of cores, and computational workloads in genetic studies tend to contain large "embarrassingly parallel" steps which can easily exploit additional cores. Therefore, PLINK 1.9 autodetects the number of cores present in the machine it is running on, and many of its heavy-duty operations default to employing roughly that number of threads. (This behavior can be manually controlled with the `--threads` flag.) Most of PLINK 1.9's multithreaded computations use a simple set of cross-platform C functions and macros, which compile to `pthread` library idioms on Linux and OS X, and OS-specific idioms like `_beginthreadex()` on Windows.

PLINK 1.9 also contains improved support for distributed computation: the `--parallel` flag makes it easy to split large matrix computations across a cluster.

One major computational resource remains unexploited: graphics processing units. We have made development of GPU-specific code a low priority since their installed base is much smaller than that of multicore processors, and the speedup factor over well-written multithreaded code running on similar-cost, less specialized hardware is usually less than 10x [9]. However, we do plan to build out GPU support for the heaviest-duty computations after most of our other PLINK 2 development goals are achieved.

Memory efficiency

To make it possible for PLINK 1.9 to handle the huge datasets which benefit the most from these speed improvements, the program core no longer keeps the main genomic data matrix in memory; instead, most of its functions only load data for a single marker, or a small window of markers, at a time. Sample × sample matrix computations still normally require additional memory proportional to the square of the sample size, but `--parallel` gets around this:

```
plink --bfile [fileset name] --make-grm-bin --parallel 1 40
plink --bfile [fileset name] --make-grm-bin --parallel 2 40
...
plink --bfile [fileset name] --make-grm-bin --parallel 40 40
cat plink.grm.bin.1 ... plink.grm.bin.40 > plink.grm.bin
cat plink.grm.N.bin.1 ... plink.grm.N.bin.40 > plink.grm.N.bin
```

calculates 1/40th of the genomic relationship matrix per run, with correspondingly reduced memory requirements.

Other noteworthy algorithms

Partial sum lookup Each entry of a weighted distance matrix is a sum of per-marker terms. Given PLINK 1 binary data, for any specific marker, there are at most seven distinct cases:

1. Both genotypes are homozygous for the major allele.
2. One is homozygous major, and the other is heterozygous.

3 One is homozygous major, and the other is homozygous minor.
4 Both are heterozygous.
5 One is heterozygous, and the other is homozygous minor.
6 Both are homozygous minor.
7 At least one genotype is missing.

For example, the GCTA genomic relationship matrix is defined by the following per-marker increments (where q is the minor allele frequency):

1 $\frac{(2-2q)(2-2q)}{2q(1-q)}$
2 $\frac{(2-2q)(1-2q)}{2q(1-q)}$
3 $\frac{(2-2q)(0-2q)}{2q(1-q)}$
4 $\frac{(1-2q)(1-2q)}{2q(1-q)}$
5 $\frac{(1-2q)(0-2q)}{2q(1-q)}$
6 $\frac{(0-2q)(0-2q)}{2q(1-q)}$
7 0 (subtract 1 from the final denominator instead, in another loop)

This suggests the following matrix calculation algorithm, as a first draft:

1 Initialize all distance/relationship partial sums to zero.
2 For each marker, calculate and save the seven possible increments in a lookup table, and then refer to the table when updating partial sums. This replaces several floating point adds/multiplies in the inner loop with a single addition operation.

We can substantially improve on this by handling multiple markers at a time. Since seven cases can be distinguished by three bits, we can compose a sequence of operations which maps a pair of padded 2-bit genotypes to seven different 3-bit values in the appropriate manner. On 64-bit machines, 20 3-bit values can be packed into a machine word (for example, let bits 0-2 describe the relation at marker #0, bits 3-5 describe the relation at marker #1, etc., all the way up to bits 57-59 describing the relation at marker #19), so this representation lets us instruct the processor to act on 20 markers simultaneously.

Then, we need to perform the update

$$A_{jk} := A_{jk} + f_0(x_0) + f_1(x_1) + \ldots + f_{19}(x_{19})$$

where the x_i's are bit trios, and the f_i's map them to increments. This could be done with 20 table lookups and floating point addition operations. Or, the update could be restructured as

$$A_{jk} := A_{jk} + f_{\{0-4\}}(x_{\{0-4\}}) + \ldots + f_{\{15-19\}}(x_{\{15-19\}})$$

where $x_{\{0-4\}}$ denotes the lowest-order **15** bits, and $f_{\{0-4\}}$ maps them directly to $f_0(x_0) + f_1(x_1) + f_2(x_2) + f_3(x_3) + f_4(x_4)$; similarly for $f_{\{5-9\}}$, $f_{\{10-14\}}$, and

$f_{\{15-19\}}$. In exchange for some precomputation (four tables with 2^{15} entries each; total size 1 MB, which is not onerous for modern L2/L3 caches), this restructuring licenses the use of four table lookups and adds per update instead of twenty. See `fill_weights_r()` and `incr_dists_r()` in `plink_calc.c` for source code.

Hardy-Weinberg and Fisher's exact tests PLINK 1.0 used Wigginton et al.'s SNP-HWE algorithm [10] to test for Hardy-Weinberg equilibrium, and Mehta et al.'s FEXACT network algorithm [11] [12] for Fisher's exact test on 2×2 and 2×3 tables.

SNP-HWE exploits the fact that, while the absolute likelihood of a contingency table involves large factorials which are fairly expensive to evaluate, the ratios between its likelihood and that of adjacent tables are simple since the factorials almost entirely cancel out. While studying the software, we made two additional observations:

1. Its size-$O(n)$ memory allocation (where n is the sum of all contingency table entries) could be avoided by reordering the calculation; it is only necessary to track a few partial sums.
2. Since likelihoods decay super-geometrically as one moves away from the most probable table, only $O(\sqrt{n})$ of the likelihoods can meaningfully impact the partial sums; the sum of the remaining terms is too small to consistently affect even the 10th significant digit in the final p-value. By terminating the calculation when all the partial sums stop changing (due to the newest term being too tiny to be tracked by IEEE-754 double-precision numbers), computational complexity is reduced from $O(n)$ to $O(\sqrt{n})$ with no loss of precision. See Figure 1 for an example.

Fisher's exact test for 2×2 tables has the same mathematical structure, so it was straightforward to modify the early-termination SNP-HWE algorithm to handle it. The 2×3 case is more complicated, but retains the property that only $O(\sqrt{\#\text{ of tables}})$ relative likelihoods need to be evaluated, so we were able to develop a function to handle it in $O(n)$ time. Our timing data indicate that our new functions represent very large improvements over both FEXACT and Requena et al.'s updates [13] to the network algorithm.

Standalone source code for early-termination SNP-HWE and Fisher's $2 \times 2/2 \times 3$ exact test is posted at [14]. (Due to recent calls for use of mid-p adjustments in biostatistics [15] [16], all of these functions have mid-p modes, and PLINK 1.9 exposes them.) We are preparing another paper which discusses these algorithms in more detail, with attention to numerical stability and a full explanation of how the Fisher's exact test algorithm extends to larger tables.

Haplotype block estimation PLINK 1.0's `--blocks` command implements Gabriel et al.'s [17] confidence interval-based method of estimating haplotype blocks. (More precisely, it is a restricted port of Haploview's [18] implementation of the method.) Briefly, the method involves using 90% confidence intervals for D' (as defined by Wall and Pritchard [19]) to classify pairs of variants as "strong LD", "strong evidence for historical recombination", or "inconclusive"; then, contiguous groups of

variants where "strong LD" pairs outnumber "recombination" pairs by more than 19 to 1 are greedily selected, starting with the longest base-pair spans.

PLINK 1.9 accelerates this in several ways:

- Determination of the initial diplotype frequency and D' point estimates has been streamlined. We use the analytic solution to Hill's diplotype frequency cubic equation [20], and only compute log likelihoods when multiple solutions to the equation are in the valid range.
- 90% confidence intervals were originally estimated by computing relative likelihoods at 101 points (corresponding to $D' = 0, D' = 0.01, \ldots, D' = 1$) and checking where the resulting cumulative distribution function crossed 5% and 95%. However, the likelihood function rarely has more than one extreme point in $(0, 1)$ (and the full solution to the cubic equation reveals the presence of additional extrema); it is usually possible to exploit this property to establish good bounds on key cdf values after evaluating just a few likelihoods. In particular, many confidence intervals can be classified as "recombination" after inspection of just two of the 101 points; see Figure 2.
- Instead of saving the classification of every variant pair and looking up the resulting massive table at a later point, we just update a small number of "strong LD pairs within last k variants" and "recombination pairs within last k variants" counts while processing the data sequentially, saving only final haploblock candidates. This reduces the amount of time spent looking up out-of-cache memory, and also allows much larger datasets to be processed.
- Since "strong LD" pairs must outnumber "recombination" pairs by 19 to 1, it does not take many "recombination" pairs in a window before one can prove no haploblock can contain that window. When this bound is crossed, we take the opportunity to entirely skip classification of many pairs of variants.

Most of these ideas are implemented in `haploview_blocks_classify()` and `haploview_blocks()` in `plink_ld.c`. The last two optimizations were previously implemented in Taliun's "LDExplorer" R package [21].

Coordinate-descent LASSO PLINK 1.9 includes a basic coordinate-descent LASSO implementation [22] (`--lasso`), which can be useful for phenotypic prediction and related applications. See Vattikuti et al. [23] for discussion of its theoretical properties.

Newly integrated third-party software

PLINK 1.0 commands Many teams have significantly improved upon PLINK 1.0's implementations of various commands and made their work open source. In several cases, their innovations have been integrated into PLINK 1.9; examples include

- Pahl et al.'s PERMORY algorithm for fast permutation testing [24],
- Wan et al.'s BOOST software for fast epistasis testing [25],
- Ueki, Cordell, and Howey's `--fast-epistasis` variance correction and joint-effects test [26] [27], and
- Pascal Pons's winning submission to the GWAS Speedup logistic regression crowdsourcing contest [28]. (The contest was designed by Po-Ru Loh, run by Babbage Analytics & Innovation and TopCoder, and subsequent analysis

and code preparation were performed by Andrew Hill, Ragu Bharadwaj, and Scott Jelinsky. A manuscript is in preparation by these authors and Iain Kilty, Kevin Boudreau, Karim Lakhani and Eva Guinan.)

In all such cases, PLINK's citation instructions direct users of the affected functions to cite the original work.

Multithreaded gzip For many purposes, compressed text files strike a good balance between ease of interpretation, loading speed, and resource consumption. However, the computational cost of generating them is fairly high; it is not uncommon for data compression to take longer than all other operations combined. To make a dent in this bottleneck, we have written a simple multithreaded compression library function based on Mark Adler's excellent `pigz` program [29], and routed most of PLINK 1.9's gzipping through it. See `parallel_compress()` in `pigz.c` for details.

Convenience features

Import and export of VCF- and Oxford-formatted data PLINK 1.9 can import data from VCF/BCF2 (`--vcf`, `--bcf`) and Oxford-format (`--data`, `--bgen`) files. However, since it cannot handle probabilistic calls, phase information, or variants with more than two alleles, the import process is currently quite lossy. Specifically,

- With Oxford-format files, genotype likelihoods smaller than 0.9 are normally treated as missing calls (and the rest are treated as hard calls); `--hard-call-threshold` can be used to change the threshold, or request independent pseudorandom calls based on the likelihoods in the file.
- Phase is discarded.
- By default, when a VCF variant has more than one alternate allele, only the most common alternate is retained (all other alternate calls are converted to missing). `--biallelic-only` can be used to skip variants with multiple alternate alleles.

Export to these formats is also possible, via `--recode vcf` and `--recode oxford`.

Nonstandard chromosome code support When the `--allow-extra-chr` or `--aec` flag is used, PLINK 1.9 allows datasets to contain unplaced contigs or other arbitrary chromosome names, and most commands will handle them in a reasonable manner. Also, arbitrary nonhuman species (with haploid or diploid genomes) can now be specified with `--chr-set`.

Command-line help To improve the experience of using PLINK interactively, we have expanded the `--help` flag's functionality. When invoked with no parameters, it now prints an entire mini-manual. Given keyword(s), it instead searches for and prints mini-manual entries associated with those keyword(s), and handles misspelled keywords and keyword prefixes in a reasonable manner.

A comment on within-family analysis

Most of our discussion has addressed computational issues. There is one methodological issue, however, that deserves a brief comment. The online documentation

of PLINK 1.07 weighed the pros and cons of its permutation procedure for within-family analysis of quantitative traits (QFAM) with respect to the standard quantitative transmission disequilibrium test (QTDT). It pointed out that likelihood-based QTDT enjoyed the advantages of computational speed and increased statistical power. However, a comparison of statistical power is only meaningful if both procedures are anchored to the same Type 1 error rate with respect to the null hypothesis of no linkage with a causal variant, and Ewens et al. [30] have shown that the QTDT is not robust against certain forms of confounding (population stratification). The validity of a permutation procedure such as QFAM, on the other hand, only depends on the applicability of Mendel's laws. When this nicety is combined with the vast speedup of permutation in PLINK 1.9, a given user may now decide to rate QFAM more highly relative to QTDT when considering available options for within-family analysis.

Performance comparisons

In the following tables, running times are collected from seven machines operating on three datasets.

- "Mac-2" denotes a MacBook Pro with a 2.8 Ghz Intel Core 2 Duo processor and 4GB RAM running OS X 10.6.8.
- "Mac-12" denotes a Mac Pro with two 2.93 Ghz Intel 6-core Xeon processors and 64GB RAM running OS X 10.6.8.
- "Linux32-2" denotes a machine with a 2.4 Ghz Intel Core 2 Duo E6600 processor and 1GB RAM running 32-bit Ubuntu Linux.
- "Linux32-8" denotes a machine with a 3.4 Ghz Intel Core i7-3770 processor (8 cores) and 8GB RAM running 32-bit Ubuntu Linux.
- "Linux64-512" denotes a machine with sixty-four AMD 8-core Opteron 6282 SE processors and 512GB RAM running 64-bit Linux.
- "Win32-2" denotes a laptop with a 2.4 Ghz Intel Core i5-2430M processor (2 cores) and 4GB RAM running 32-bit Windows 7 SP1.
- "Win64-2" denotes a machine with a 2.3 Ghz Intel Celeron G1610T processor (2 cores) and 8GB RAM running 64-bit Windows 8.
- "synth1" refers to a 1000 sample, 100000 variant synthetic dataset generated with HAPGEN2 [31], while "synth1p" refers to the same dataset after one round of `--indep-pairwise 50 5 0.5` pruning (with 76124 markers remaining). For case/control tests, PLINK 1.9's `--tail-pheno 0` command was used to downcode the quantitative phenotype to case/control.
- "synth2" refers to a 4000 case, 6000 control synthetic dataset with 88025 markers on chromosomes 19-22 generated by resampling HapMap and 1000 Genomes data with simuRare [32] and then removing monomorphic loci. "synth2p" refers to the same dataset after one round of `--indep-pairwise 700 70 0.7` pruning (with 71307 markers remaining).
- "1000g" refers to the entire 1092 sample, 39637448 variant 1000 Genomes project phase 1 dataset [33]. "chr1" refers to chromosome 1 from this dataset, with 3001739 variants. "chr1snp" refers to chromosome 1 after removal of all non-SNPs and one round of `--indep-pairwise 20000 2000 0.5` pruning (798703 markers remaining).

All times are in seconds. To reduce disk-caching variance, timing runs are preceded by "warmup" commands like `plink --freq`. PLINK 1.07 was run with the `--noweb` flag. "nomem" indicates that the program ran out of memory and there was no low-memory mode or other straightforward workaround. A tilde indicates that runtime was extrapolated from several smaller problem instances.

Initialization and basic I/O

Table 1 displays execution times for `plink --freq`, one of the simplest operations PLINK can perform. These timings reflect fixed initialization and I/O overhead. (Due to the use of warmup runs, they do not include disk latency.)

Identity-by-state matrices, complete linkage clustering

The PLINK 1.0 `--cluster --matrix` flag combination launches an identity-by-state matrix calculation and writes the result to disk, and then performs complete linkage clustering on the data; when `--ppc` is added, a pairwise population concordance constraint is applied to the clustering process. As discussed earlier, PLINK 1.9 employs an XOR/bit population count algorithm which speeds up the matrix calculation by a large constant factor; the computational complexity of the clustering algorithm has also been reduced, from $O(n^3)$ to $O(n^2 \log n)$. (Further improvement of clustering complexity, to $O(n^2)$, is possible in some cases [34].)

In Table 2, we compare PLINK 1.07 and PLINK 1.9 execution times under three scenarios: IBS matrix calculation only (`--cluster --matrix --K` [sample count -1] in PLINK 1.07, `--distance ibs square` in PLINK 1.9), IBS matrix + standard clustering (`--cluster --matrix` for both versions), and IBD report generation (`--Z-genome`).

(Note that newer algorithms such as BEAGLE's fastIBD [35] generate more accurate IBD estimates than PLINK `--Z-genome`. However, the `--Z-genome` report contains other useful information.)

Genomic relationship matrices

GCTA's `--make-grm-bin` command (`--make-grm` in early versions) calculates the variance-standardized genomic relationship matrix used by many of its other commands. The latest implementation as of this writing is very fast, but cannot run on OS X or Windows. PLINK 1.9 includes a cross-platform implementation which is almost as fast and has a lighter memory requirement. See Table 3 for timing data. (The comparison is with GCTA v1.24 on 64-bit Linux, and v1.02 elsewhere.)

Linkage disequilibrium-based variant pruning

The PLINK 1.0 `--indep-pairwise` command is frequently used in preparation for analyses which assume approximate linkage equilibrium. In Table 4, we compare PLINK 1.07 and PLINK 1.9 execution times for some reasonable parameter choices. Note that as of this writing, `--indep-pairwise`'s implementation is single-threaded; this is why the heavily multicore machines are not faster than the 2-core machines. The r^2 threshold for "synth2" was chosen to make the "synth1p" and "synth2p" pruned datasets contain similar number of SNPs, so Tables 2-3 could clearly demonstrate scaling w.r.t. sample size.

Haplotype block estimation

Table 5 demonstrates the impact of our rewrite of `--blocks`. Due to a minor bug in PLINK 1.0's handling of low-MAF variants, we pruned each dataset to contain only variants with MAF ≥ 0.05 before running `--blocks`. 95506 markers remained in the "synth1" dataset, and 554549 markers remained in "chr1". A question mark indicates that the extrapolated runtime may not be valid since we suspect Haploview or PLINK 1.0 would have run out of memory before finishing.

Association analysis max(T) permutation tests

PLINK 1.0's basic association analysis commands were quite flexible, but the powerful max(T) permutation test suffered from poor performance. PRESTO [36] and PERMORY introduced major algorithmic improvements (including bit population count) which largely solved the problem. Table 6 shows that PLINK 1.9 successfully extends the PERMORY algorithm to the full range of PLINK 1.0's association analyses, while making Fisher's exact test practical to use in permutation tests. (There is no 64-bit Windows PERMORY build, so the comparisons on the Win64-2 machine are between 64-bit PLINK and 32-bit PERMORY.)

PLINK 2 design

Despite its computational advances, we recognize that PLINK 1.9 can ultimately still be an unsatisfactory tool for working with imputed genomic data, due to the limitations of the PLINK 1 binary file format. To address this, PLINK 2.0 will support a new core file format capable of representing essentially all information emitted by modern imputation tools, and many of its functions will be extended to account for the new types of information.

Multiple data representations

As discussed earlier, PLINK 1 binary is inadequate in three ways: probabilities strictly between 0 and 1 cannot be represented, phase cannot be stored, and variants are limited to two alleles. This can be addressed by representing *all* calls probabilistically, and introducing a few other extensions. Unfortunately, this would make PLINK 2.0's representation of PLINK 1-format data so inefficient that it would amount to a serious downgrade from PLINK 1.9 for many purposes.

Therefore, our new format defines several data representations, one of which is equivalent to PLINK 1 binary, and allows different files, or even variants within a single file, to use different representations. To work with this, PLINK 2 will include a translation layer which allows individual functions to assume a specific representation is used. As with the rest of PLINK's source code, this translation layer will be open source and usable in other programs under GPLv3 terms; and unlike most of the other source code, it will be explicitly designed to be included as a standalone library. PLINK 2 will also be able to convert files/variants from one data representation to another, making it practical for third-party tools lacking access to the library to demand a specific representation.

Data compression

PLINK 1.9 demonstrates the power of a weak form of compressive genomics [37]: by using bit arithmetic to perform computation directly on compressed genomic

data, it frequently exhibits far better performance than programs which require an explicit decompression step. But its "compressed format" is merely a tight packing which does not support the holy grail of true sublinear analysis.

To do our part to make "strong" sublinear compressive genomics a reality, the PLINK 2 file format will introduce support for "deviations from reference" storage of low-MAF variants. For datasets containing many samples, this captures much of the storage efficiency benefit of having real reference genomes available, without the drawback of forcing all programs operating on the data to have access to a library of references. Thanks to PLINK 2's translation layer and file conversion facilities, programmers will be able to ignore this feature during initial development of a tool, and then work to exploit it after basic functionality is in place.

We note that LD-based compression of variant groups is also possible, and Sambo's SNPack software [38] applies this to the PLINK 1 binary format. We do not plan to support this in PLINK 2.0 due to the additional software complexity required to handle probabilistic and multiallelic data, but we believe this is a promising avenue for development and look forward to integrating it in the future.

Remaining limitations

PLINK 2 is designed to meet the needs of tomorrow's genome-wide association studies and population-genetics research; in both contexts, it is appropriate to apply a single genomic coordinate system across all samples, and preferred sample sizes are large enough to make computational efficiency a serious issue.

Whole-exome and whole-genome sequencing also enables detailed study of structural variations which defy clean representation under a single coordinate system; and the number of individuals in such studies is typically much smaller than the tens or even hundreds of thousands which are sometimes required for effective GWAS. There are no plans to make PLINK suitable for this type of analysis; we strongly recommend the use of another software package, such as PLINK/SEQ [39], which is explicitly designed for it. This is why the PLINK 2 file format will still be substantially less expressive than VCF.

An important consequence is that, despite its ability to import and export VCF files, PLINK should not be used for management of genomic data which will be subject to both types of analysis, because it discards all information which is not relevant for its preferred type. However, we will continue to extend PLINK's ability to interpret VCF-like formats and interoperate with other popular software.

Availability and requirements

- Project name: PLINK 2
- Project (source code) home page: https://www.cog-genomics.org/plink2/ (https://github.com/chrchang/plink-ng)
- Operating systems: Linux (32/64-bit), OS X (64-bit Intel), Windows (32/64-bit)
- Programming language: C, C++
- Other requirements (when recompiling): GCC version 4, a few functions also require LAPACK 3.2
- License: GNU General Public License version 3.0 (GPLv3)
- Any restrictions to use by non-academics: none

Competing interests
The authors declare that they have no competing interests.

Author's contributions
SMP and Ch C designed the software. Ch C drafted the manuscript and did most of the v1.9 C/C++ programming. Ca C, SV, and JJL drove early v1.9 feature development and wrote MATLAB prototype code. Ca C, LCAMT, SV, SMP, and JJL assisted with v1.9 software testing. All authors read and approved the final manuscript.

Acknowledgements
We thank Stephen D.H. Hsu for helpful discussions. We also continue to be thankful to PLINK 1.9 users who perform additional testing of the program, report bugs, and make useful suggestions.
Christopher Chang was supported by BGI Hong Kong. Carson Chow and Shashaank Vattikuti were supported by the Intramural Research Program of the NIH, NIDDK.

Author details
[1]Complete Genomics, 2071 Stierlin Court, 94043 Mountain View, CA, USA. [2]BGI Cognitive Genomics Lab, Building No. 11, Bei Shan Industrial Zone, Yantian District, 518083 Shenzhen, China. [3]Mathematical Biology Section, NIDDK/LBM, National Institutes of Health, 20892 Bethesda, MD, USA. [4]Bioinformatics Centre, University of Copenhagen, 2200 Copenhagen, Denmark. [5]Stanley Center for Psychiatric Research, Broad Institute of MIT and Harvard, 02142, Cambridge, MA, USA. [6]Division of Psychiatric Genomics, Department of Psychiatry, Icahn School of Medicine at Mount Sinai, 10029, New York, NY, USA. [7]Institute for Genomics and Multiscale Biology, Icahn School of Medicine at Mount Sinai, 10029, New York, NY, USA. [8]Analytic and Translational Genetics Unit, Psychiatric and Neurodevelopmental Genetics Unit, Massachusetts General Hospital, 02114, Boston, MA, USA. [9]Department of Psychology, University of Minnesota Twin Cities, 55455 Minneapolis, MN, USA.

References

1. Purcell, S., Neale, B., Todd-Brown, K., Thomas, L., Ferreira, M., Bender, D., Maller, J., Sklar, P., de Bakker, P., Daly, M., Sham, P.: Plink: A tool set for whole-genome association and population-based linkage analyses. Am J Hum Genet **81**, 559–575 (2007)
2. Browning, B., Browning, S.: Improving the accuracy and efficiency of identity by descent detection in population data. Genetics **194**, 459–471 (2013)
3. Howie, B., Donnelly, P., Marchini, J.: A flexible and accurate genotype imputation method for the next generation of genome-wide association studies. PLoS Genet **5**, 1000529 (2009)
4. McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., Daly, M., DePristo, M.: The genome analysis toolkit: A mapreduce framework for analyzing next-generation dna sequencing data. Genome Res **20**, 1297–1303 (2010)
5. Danecek, P., Auton, A., Abecasis, G., Albers, C., Banks, E., DePristo, M., Handsaker, R., Lunter, G., Marth, G., Sherry, S., McVean, G., Durbin, R.: The variant call format and vcftools. Bioinformatics **27**, 2156–2158 (2011)
6. Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., 1000 Genome Project Data Processing Subgroup: The sequence alignment/map format and samtools. Bioinformatics **25**, 2078–2079 (2009)
7. Yang, J., Lee, S., Goddard, M., Visscher, P.: Gcta: A tool for genome-wide complex trait analysis. Am J Hum Genet **88**, 76–82 (2011)
8. Dalke, A.: Update: Faster Population Counts. http://www.dalkescientific.com/writings/diary/archive/2011/11/02/faster_popcount_update.html
9. Lee, V., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A., Satish, M., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., Singhal, R., Dubey, P.: Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu. In: Proceedings of the 37th Annual International Symposium on Computer Architecture: 19-23 June 2010; Saint-Malo, France, pp. 451–460 (2010). ACM
10. Wigginton, J., Cutler, D., Abecasis, G.: A note on exact tests of hardy-weinberg equilibrium. Am J Hum Genet **76**, 887–893 (2005)
11. Mehta, C., Patel, N.: Algorithm 643: Fexact: a fortran subroutine for fisher's exact test on unordered r×c contingency tables. ACM Transactions on Mathematical Software **12**, 154–161 (1986)
12. Clarkson, D., Fan, Y., Joe, H.: A remark on algorithm 643: Fexact: an algorithm for performing fisher's exact test in r x c contingency tables. ACM Transactions on Mathematical Software **19**, 484–488 (1993)
13. Requena, F., Martín Ciudad, N.: A major improvement to the network algorithm for fisher's exact test in 2×c contingency tables. J Comp Stat & Data Anal **51**, 490–498 (2006)
14. Chang, C.: Standalone C/C++ Exact Statistical Test Functions. https://github.com/chrchang/stats
15. Lydersen, S., Fagerland, M., Laake, P.: Recommended tests for association in 2×2 tables. Statist Med **28**, 1159–1175 (2009)
16. Graffelman, J., Moreno, V.: The mid p-value in exact tests for hardy-weinberg equilibrium. Stat Appl Genet Mol Bio **12**, 433–448 (2013)
17. Gabriel, S., Schaffner, S., Nguyen, H., Moore, J., Roy, J., Blumenstiel, B., Higgins, J., DeFelice, M., Lochner, A., Faggart, M., Liu-Cordero, S., Rotimi, C., Adeyemo, A., Cooper, R., Ward, R., Lander, E., Daly, M., Altshuler, D.: The structure of haplotype blocks in the human genome. Science **296**, 2225–2229 (2002)
18. Barrett, J., Fry, B., Maller, J., Daly, M.: Haploview: analysis and visualization of ld and haplotype maps. Bioinformatics **21**, 263–265 (2005)
19. Wall, J., Pritchard, J.: Assessing the performance of the haplotype block model of linkage disequilibrium. Am J Hum Genet **73**, 502–515 (2003)
20. Gaunt, T., Rodríguez, S., Day, I.: Cubic exact solutions for the estimation of pairwise haplotype frequencies: implications for linkage disequilibrium analyses and a web tool 'cubex'. BMC Bioinformatics **8**, 428 (2007)
21. Taliun, D., Gamper, J., Pattaro, C.: Efficient haplotype block recognition of very long and dense genetic sequences. BMC Bioinformatics **15**, 10 (2014)

22. Friedman, J., Hastie, T., Höfling, H., Tibshirani, R.: Pathwise coordinate optimization. Annals of Applied Statistics **1**, 302–332 (2007)
23. Vattikuti, S., Lee, J., Chang, C., Hsu, S., Chow, C.: Applying compressed sensing to genome-wide association studies. GigaScience **3**, 10 (2014)
24. Steiß, V., Letschert, T., Schäfer, H., Pahl, R.: Permory-mpi: A program for high-speed parallel permutation testing in genome-wide association studies. Bioinformatics **28**, 1168–1169 (2012)
25. Wan, X., Yang, C., Yang, Q., Xue, H., Fan, X., Tang, N., Yu, W.: Boost: A fast approach to detecting gene-gene interactions in genome-wide case-control studies. Am J Hum Genet **87**, 325–340 (2010)
26. Ueki, M., Cordell, H.: Improved statistics for genome-wide interaction analysis. PLoS Genet **8**, 1002625 (2012)
27. Howey, R.: CASSI: Genome-Wide Interaction Analysis Software. http://www.staff.ncl.ac.uk/richard.howey/cassi
28. CASSI: Genome-Wide Interaction Analysis Software. http://community.topcoder.com/longcontest/?module=ViewProblemStatement&rd=15637&pm=12525
29. Adler, M.: Pigz: Parallel Gzip. http://zlib.net/pigz/
30. Ewens, W., Li, M., Spielman, R.: A review of family-based tests for linkage disequilibrium between a quantitative trait and a genetic marker. PLoS Genet **4**, 1000180 (2008)
31. Su, Z., Marchini, J., Donnelly, P.: Hapgen2: Simulation of multiple disease snps. Bioinformatics **27**, 2304–2305 (2011)
32. Xu, Y., Wu, Y., Song, C., Zhang, H.: Simulating realistic genomic data with rare variants. Genet Epidemiol **37**, 163–172 (2013)
33. The 1000 Genomes Project Consortium: An integrated map of genetic variation from 1,092 human genomes. Nature **491**, 56–65 (2012)
34. Defays, D.: An efficient algorithm for a complete link method. Computer Journal **20**, 364–366 (1977)
35. Browning, B., Browning, S.: A fast, powerful method for detecting identity by descent. Am J Hum Genet **88**, 173–182 (2011)
36. Browning, B.: Presto: rapid calculation of order statistic distributions and multiple-testing adjusted p-values via permutation for one and two-stage genetic association studies. BMC Bioinformatics **9**, 309 (2008)
37. Loh, P., Baym, M., Berger, B.: Compressive genomics. Nature Biotechnology **30**, 627–630 (2012)
38. Sambo, F., Di Camillo, B., Toffolo, G., Cobelli, C.: Compression and fast retrieval of snp data. Bioinformatics **30**, 495 (2014)
39. PLINK/SEQ: A Library for the Analysis of Genetic Variation Data. https://atgu.mgh.harvard.edu/plinkseq/

Figures

Tables

Table 1 Initialization and basic I/O (`--freq`).

Dataset	Machine	PLINK 1.07	**PLINK 1.90**	Ratio
synth1	Mac-2	7.3	**0.24**	30
	Mac-12	6.2	**0.18**	34
	Linux32-2	13.1	**0.56**	23
	Linux32-8	4.3	**0.18**	24
	Linux64-512	5.4	**0.18**	27
	Win32-2	14.3	**0.68**	21
	Win64-2	9.6	**0.33**	29
synth2	Mac-2	43.3	**0.84**	52
	Mac-12	38.2	**0.34**	110
	Linux32-2	80.1	**1.9**	42
	Linux32-8	25.2	**0.53**	48
	Linux64-512	34.1	**0.40**	85
	Win32-2	83.6	**1.3**	64
	Win64-2	70.8	**0.55**	130
chr1snp	Mac-2	52.5	**3.5**	15
	Mac-12	40.5	**1.3**	31
	Linux32-2	72.9	**10.2**	7.15
	Linux32-8	29.7	**1.4**	21
	Linux64-512	36.8	**1.4**	26
	Win32-2	104.3	**4.5**	23
	Win64-2	76.8	**2.2**	35
chr1	Mac-2	403.9	**35.0**	11.5
	Mac-12	163.9	**5.3**	31
	Linux32-2	nomem	**65.3**	
	Linux32-8	134.1	**12.8**	10.5
	Linux64-512	144.7	**5.4**	27
	Win32-2	389.2	**21.4**	18.2
	Win64-2	285.3	**8.1**	35

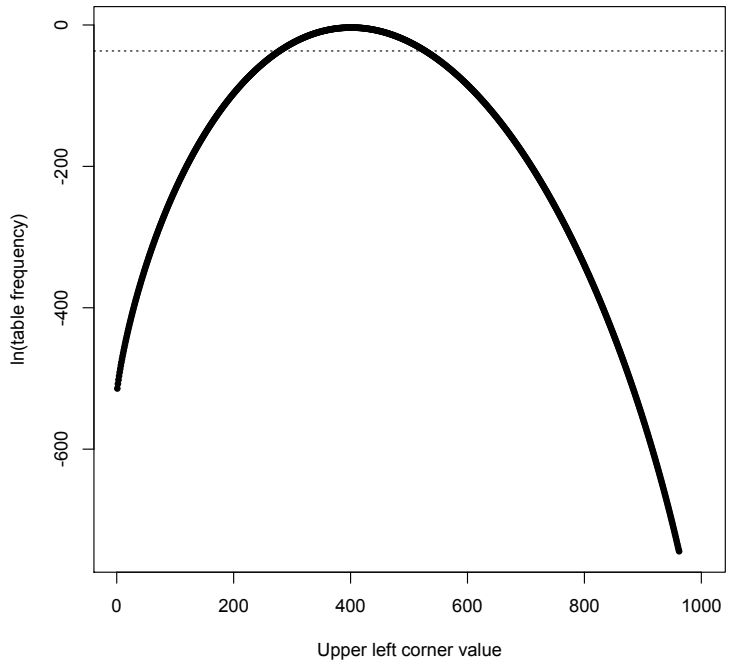


Figure 1 2x2 contingency table log-frequencies. This is a plot of relative frequencies of 2x2 contingency tables with top row sum 1000, left column sum 40000, and grand total 100000, reflecting a low-MAF variant where the difference between the chi-square test and Fisher's exact test is relevant. All such tables with upper left value smaller than 278, or larger than 526, have frequency smaller than 2^{-53} (dotted horizontal line); thus, if the obvious summation algorithm is used, they have no impact on the p-value denominator due to numerical underflow. (It can be proven that this underflow has negligible impact on accuracy, due to how rapidly the frequencies decay.) A few more tables need to be considered when evaluating the numerator, but we can usually skip at least 70%, and this fraction improves as problem size increases.

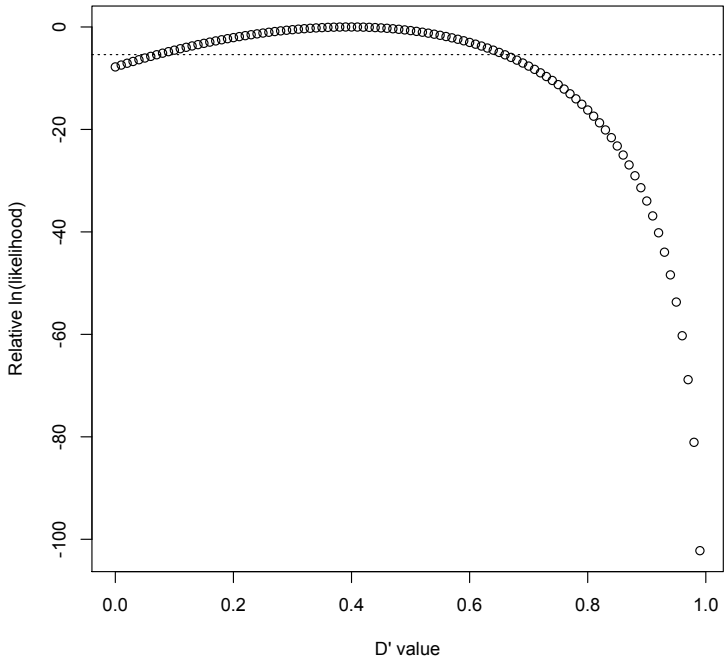


Figure 2 Rapid classification of "recombination" variant pairs. This is a plot of 101 equally spaced D' log-likelihoods for (rs58108140, rs140337953) in 1000 Genomes phase 1, used in Gabriel et al.'s method of identifying haplotype blocks. Whenever the upper end of the 90% confidence interval is smaller than 0.90 (i.e. the rightmost 11 likelihoods sum to less than 5% of the total), we have strong evidence for historical recombination between the two variants. After determining that $L(D' = x)$ has only one extreme value in [0, 1] and that it's between 0.39 and 0.40, confirming $L(D' = 0.90) < L(D' = 0.40)/220$ is enough to finish classifying the variant pair (due to monotonicity: $L(D' = 0.90) \geq L(D' = 0.91) \geq \ldots \geq L(D' = 1.00)$); evaluation of the other 99 likelihoods is now skipped in this case. The dotted horizontal line is at $L(D' = 0.40)/220$.

Table 2 Identity-by-state and complete linkage clustering times.

Calculation	Dataset	Machine	PLINK 1.07	**PLINK 1.90**	Ratio
IBS matrix only	synth1p	Mac-2	2233.6	**1.9**	1.2k
		Mac-12	1320.4	**1.2**	1.1k
		Linux32-8	1937.2	**2.8**	690
		Linux64-512	1492	**3.7**	400
		Win32-2	3219.0	**7.2**	450
		Win64-2	2674.4	**1.5**	1.8k
	synth2p	Mac-2	∼190k	**118.8**	1.6k
		Mac-12	∼99k	**23.5**	4.2k
		Linux32-8	152.5k	**214.3**	710
		Linux64-512	∼98k	**25.3**	3.9k
		Win32-2	∼270k	**654.5**	410
		Win64-2	∼200k	**104.6**	1.9k
	chr1snp	Mac-2	∼26k	**17.5**	1.5k
		Mac-12	13.4k	**12.6**	1.06k
		Linux32-8	18.4k	**30.9**	600
		Linux64-512	∼14k	**43.1**	320
		Win32-2	32.7k	**95.9**	341
		Win64-2	∼26k	**15.3**	1.7k
Basic clustering	synth1p	Mac-2	2315.7	**2.7**	860
		Mac-12	1317.9	**2.0**	660
		Linux32-8	1898.7	**4.1**	460
		Linux64-512	1496	**4.5**	330
		Win32-2	3301.7	**9.1**	360
		Win64-2	2724.5	**1.9**	1.4k
	synth2p	Mac-2	∼230k	**245.6**	940
		Mac-12	∼140k	**123.9**	1.1k
		Linux32-8	197.1k	**395.6**	498
		Linux64-512	∼125k	**143.3**	872
		Win32-2	∼440k	**976.7**	450
		Win64-2	∼270k	**127.9**	2.1k
	chr1snp	Mac-2	∼26k	**18.4**	1.4k
		Mac-12	13.6k	**13.5**	1.01k
		Linux32-8	18.5k	**33.4**	554
		Linux64-512	∼14k	**44.2**	320
		Win32-2	33.2k	**95.0**	349
		Win64-2	∼26k	**15.8**	1.6k
IBD report	synth1p	Mac-2	2230.1	**12.4**	180
		Mac-12	1346.2	**2.4**	560
		Linux32-8	2019.9	**12.4**	163
		Linux64-512	1494	**5.0**	300
		Win32-2	3446.3	**42.2**	81.7
		Win64-2	2669.8	**15.1**	177
	synth2p	Mac-2	∼190k	**447.1**	420
		Mac-12	∼99k	**50.3**	2.0k
		Linux32-8	161.4k	**618.7**	261
		Linux64-512	∼98k	**57.4**	1.7k
		Win32-2	∼270k	**1801.1**	150
		Win64-2	∼200k	**541.0**	370
	chr1snp	Mac-2	∼26k	**24.8**	1.0k
		Mac-12	13.4k	**14.6**	918
		Linux32-8	18.5k	**53.5**	346
		Linux64-512	∼14k	**46.5**	300
		Win32-2	33.1k	**199.2**	166
		Win64-2	∼26k	**25.1**	1.0k

Table 3 Genomic relationship matrix calculation times.

Dataset	Machine	GCTA	**PLINK 1.90**	Ratio
synth1p	Mac-2	222.2	**7.2**	31
	Mac-12	184.7	**5.0**	37
	Linux32-8	248.4	**10.9**	22.8
	Linux64-512	4.4	**8.3**	0.53
	Win32-2	373.1	**39.3**	9.5
	Win64-2	367.2	**6.6**	56
synth2p	Mac-2	nomem	**805.8**	
	Mac-12	17.0k	**138.3**	123
	Linux32-8	nomem	**1153.4**	
	Linux64-512	65.1	**166.0**	0.39
	Win32-2	nomem	**2007.2**	
	Win64-2	nomem	**450.1**	
chr1snp	Mac-2	nomem	**87.1**	
	Mac-12	2260.9	**50.9**	44.4
	Linux32-8	nomem	**94.3**	
	Linux64-512	58.3	**86.9**	0.67
	Win32-2	nomem	**317.5**	
	Win64-2	nomem	**65.7**	

Table 4 `--indep-pairwise` runtimes.

Parameters	Dataset	Machine	PLINK 1.07	**PLINK 1.90**	Ratio
`50 5 0.5`	synth1	Mac-2	701.3	**0.63**	1.1k
		Mac-12	569.4	**0.55**	1.0k
		Linux32-8	572.7	**0.95**	600
		Linux64-512	462	**0.60**	770
		Win32-2	1163.9	**3.2**	360
		Win64-2	1091.9	**1.0**	1.1k
`700 70 0.7`	synth2	Mac-2	∼120k	**31.9**	3.8k
		Mac-12	63.0k	**20.6**	3.06k
		Linux32-8	57.4k	**66.0**	870
		Linux64-512	∼120k	**26.4**	4.5k
		Win32-2	139.3k	**127.3**	1.09k
		Win64-2	∼200k	**22.9**	8.7k
`20000 2000 0.5`	chr1	Mac-2	nomem	**1520.1**	
		Mac-12	nomem	**1121.7**	
		Linux32-8	nomem	**4273.9**	
		Linux64-512	∼950k	**1553.3**	610
		Win32-2	nomem	**4912.7**	
		Win64-2	nomem	**1205.1**	
	1000g	Mac-2	nomem	**20.5k**	
		Mac-12	nomem	**14.5k**	
		Linux32-8	nomem	**54.5k**	
		Linux64-512	∼13000k	**20.2k**	640
		Win32-2	nomem	**64.5k**	
		Win64-2	nomem	**14.7k**	

Table 5 `--blocks` runtimes.

Parameters	Dataset	Machine	Haploview 4.2	PLINK 1.07	**PLINK 1.90**
`--ld-window-kb 500`	synth1	Mac-2	nomem	3198.4	**1.7**
		Mac-12	∼45k	3873.0	**1.3**
		Linux32-2	nomem	5441.1	**3.4**
		Linux64-512	∼57k	2323.4	**2.9**
		Win32-2	nomem	9803.4	**8.9**
		Win64-2	∼51k	5513.4	**2.8**
`--ld-window-kb 1000`	synth1	Mac-2	nomem	6185.7	**2.2**
		Mac-12	∼45k	7394.4	**9.8**
		Linux32-2	nomem	9876.8	**10.0**
		Linux64-512	∼57k	4462.1	**3.9**
		Win32-2	nomem	18925.7	**17.3**
		Win64-2	∼51k	10.3k	**3.6**
`--ld-window-kb 500`	chr1	Mac-2	nomem	∼2700k?	**550.9**
		Mac-12	nomem	∼3600k?	**426.0**
		Linux32-2	nomem	∼4300k?	**1288.4**
		Linux64-512	∼440k?	∼2600k?	**1119.7**
		Win32-2	nomem	∼17000k?	**4535.8**
		Win64-2	nomem	∼5700k?	**1037.2**

Table 6 Association analysis max(T) permutation test times. (`--mperm 10000 --seed 1`)

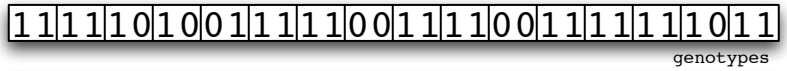
Other parameter(s)	Dataset	Machine	PLINK 1.07	PERMORY 1.1	**PLINK 1.90**	Ratio
`--trend` (C/C)	synth1	Mac-2	∼20k		**18.7**	1.1k
		Mac-12	∼16k		**2.8**	5.7k
		Linux32-2	∼21k		**65.0**	320
		Linux64-512	∼17k	285.0	**2.8**	
		Win32-2	∼35k	1444.2	**61.5**	
		Win64-2	∼25k	889.7	**14.4**	
	synth2	Mac-2	∼170k		**42.4**	4.0k
		Mac-12	∼180k		**6.4**	28k
		Linux32-2	∼410k		**391.0**	1.0k
		Linux64-512	∼200k	580.9	**13.7**	
		Win32-2	∼1100k	2362.5	**198.0**	
		Win64-2	∼370k	1423.6	**34.0**	
`--fisher` (C/C)	synth1	Mac-2	∼150k		**21.9**	6.9k
		Mac-12	∼150k		**3.7**	41k
		Linux32-2	∼170k		**57.8**	2.9k
		Linux64-512	∼120k		**3.4**	35k
		Win32-2	∼440k		**64.9**	6.8k
		Win64-2	∼200k		**22.0**	9.1k
	synth2	Mac-2	∼890k		**49.8**	18k
		Mac-12	∼690k		**7.6**	91k
		Linux32-2	∼1300k		**393.7**	3.3k
		Linux64-512	∼720k		**13.0**	55k
		Win32-2	∼3600k		**208.3**	17k
		Win64-2	∼1700k		**35.6**	48k
`--assoc` (QT)	synth1	Mac-2	∼30k		**148.0**	200
		Mac-12	∼22k		**22.6**	970
		Linux32-2	∼68k		**847.2**	80
		Linux64-512	∼29k		**29.2**	990
		Win32-2	∼58k		**896.1**	65
		Win64-2	∼36k		**264.2**	140
`--assoc lin` (QT)	synth1	Mac-2			**606.8**	
		Mac-12			**34.7**	
		Linux32-2			**3212.6**	
		Linux64-512		1259.8	**46.4**	27.2
		Win32-2		2115.7	**3062.7**	0.69
		Win64-2		972.6	**336.6**	2.89

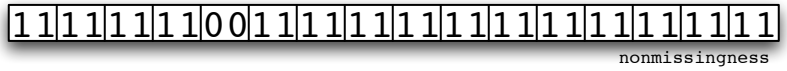
Identity-by-state and software popcount

PLINK 1.9's most important optimization is its replacement of slow loops iterating over single genotype calls with bitwise operations on many calls at a time. This document illustrates how identity-by-state (i.e. Hamming distance) between two genomes is computed in this fashion.

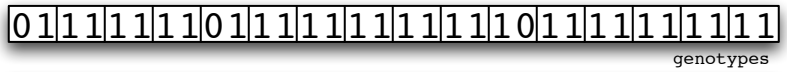
Step 1: Transposition and other preprocessing. PLINK's core file format saves genotype calls in a variant-major manner. IBS computation is faster with sample-major data, and its overall time complexity is $O(mn^2)$ while transposition is just $O(mn)$, so we transpose the data before the main loop. We also assemble a bit array tracking the presence of missing genotype calls.

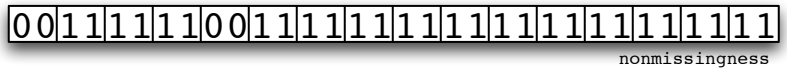
Vincent:



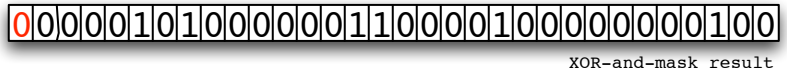


Anton:





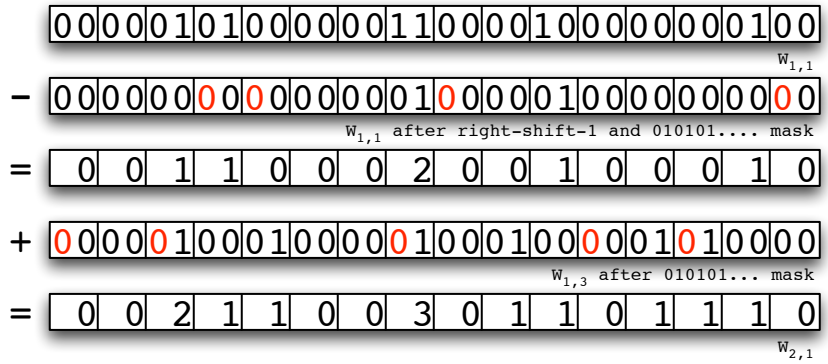
Step 2: XOR-and-mask. PLINK 1 represents homozygous major calls with binary `11`, heterozygous calls with `10`, and homozygous minor calls with `00`. Conveniently, if you take the exclusive-or of two such values, the number of set bits in the result is the number of differing allele calls; thus, the overall Hamming distance between two genomes in (transposed) PLINK 1 format is the bit population count of their XOR. Excepting missing calls (represented by `01`), that is; we "mask" (via AND operations) the final result with both nonmissingness arrays to force those bits to zero. (The red '0' below is due to the mask.)



Step 3: Software popcount. Since PLINK is still used on many machines lacking a hardware popcount instruction, we use SSE2 (in x86-64 builds) or basic word (in 32-bit builds) operations to implement the "bitslice" algorithm discussed by Dalke et al., which is almost as fast when acting on long arrays. For clarity of exposition, we illustrate what happens with six 32-bit words; our SSE2 code applies the same idea to batches of fifteen or thirty 128-bit blocks.

This can be seen as a collection of 192 one-bit values which add up to our desired result.

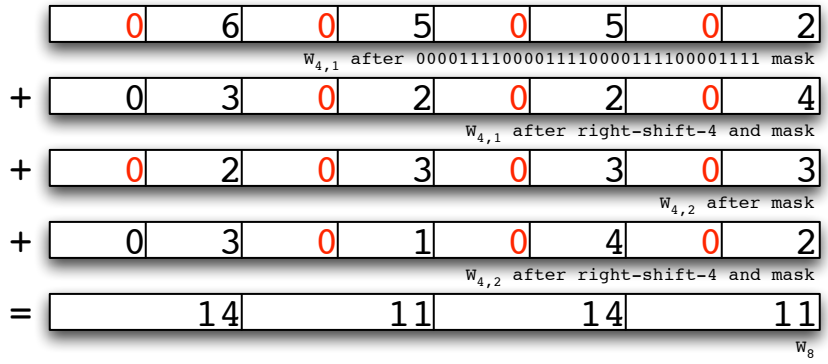
The bitslice algorithm starts by generating a collection of two-bit partial sums which add up to the same total. Specifically, the partial sums in $W_{2,1}$ aggregate two bits in $W_{1,1}$ and an even-position bit in $W_{1,3}$; $W_{2,2}$ aggregates pairs of bits in $W_{1,2}$ and odd-position bits in $W_{1,3}$; $W_{2,3}$ aggregates pairs of bits in $W_{1,4}$ and even-position bits in $W_{1,6}$; and $W_{2,4}$ aggregates pairs of bits in $W_{1,5}$ and odd-position bits in $W_{1,6}$. The actual operations are a right-shift-1, a mask with `010101...`, a subtraction, a mask (even-position) or right-shift-1-and-mask (odd-position) with `010101...`, and an addition.



Note that "2" is shorthand for binary `10` and "3" is shorthand for binary `11` here; similar shorthand will be used for four- and eight-bit partial sums on the next page.

The next step is to use these to produce an even smaller collection of four-bit partial sums with the same total. Specifically, $W_{4,1}$ aggregates two values in $W_{2,1}$ and two values in $W_{2,2}$, while $W_{4,2}$ aggregates two values in $W_{2,3}$ and two values in $W_{2,4}$.

Then we produce a single word of eight-bit partial sums from $W_{4,1}$ and $W_{4,2}$. (Since none of the four-bit partial sums can be greater than 12, and eight bits can represent values up to 255, we can actually merge up to 10 pairs of partial sums at this stage, rather than just 2; this is done by some of our SSE2 code.)



Finally, we add these eight-bit partial sums: 14 + 11 + 14 + 11 = 50, which is indeed the number of set bits among the original 192.