1 Methods

1.1 Haplotype Frequency Estimation

The MM principle involves two steps. In maximization, we first minorize and then maximize. In minimization, we first majorize and then minimize. Because we are interested in penalized maximum likelihood estimation, we interpret MM in the former sense. The minorization step creates a surrogate function $\boldsymbol{q} \mapsto g(\boldsymbol{q} \,|\, \boldsymbol{q}^n)$ anchored at the current iterate $\boldsymbol{q}^n$ of a parameter search. The surrogate function falls below the objective function $f(\boldsymbol{q})$ and is tangent to it at the current point $\boldsymbol{q}^n$. Formally, these conditions amount to

$$\begin{aligned} f(\boldsymbol{q}^n) &= g(\boldsymbol{q}^n \mid \boldsymbol{q}^n) \\ f(\boldsymbol{q}) &\geq g(\boldsymbol{q} \mid \boldsymbol{q}^n), \qquad \boldsymbol{q} \neq \boldsymbol{q}^n. \end{aligned}$$

The maximization step of the MM algorithm solves for the parameter vector $\boldsymbol{q}^{n+1}$ maximizing the surrogate function $g(\boldsymbol{q} \mid \boldsymbol{q}^n)$. In the process the objective function $f(\boldsymbol{q})$ is sent uphill.

The traditional EM algorithm for haplotype frequency estimation can be interpreted as an MM algorithm. Let $\boldsymbol{q}$ be the vector of haplotype frequencies and H_i be the set of ordered haplotype pairs (k, l) consistent with subject i's observed multilocus genotype. In this notation i's likelihood is written as

$$\ell_i(\boldsymbol{q}) = \sum_{(k,l) \in H_i} q_k q_l.$$

The full loglikelihood across all independent samples equals

$$L(\boldsymbol{q}) = \sum_i \ln \ell_i(\boldsymbol{q}).$$

To encourage parsimony, we subtract a penalty that tends to eliminate haplotypes with low explanatory power. The penalty is defined by a threshold δ, a tuning constant λ scaling the strength of the penalty, and the penalty function

$$p(q) = \begin{cases} q & q \leq \delta \\ \delta & q > \delta. \end{cases}$$

Accordingly, the haplotype frequency vector $\boldsymbol{q}$ is estimated by maximizing the objective function

$$f(\boldsymbol{q}) \quad = \quad L(\boldsymbol{q}) - \lambda \sum_j p(q_j). \tag{1}$$

The concavity of the natural logarithm entails the minorization

$$\begin{aligned} L(\boldsymbol{q}) &\geq \sum_i \sum_{(k,l)\in H_i} \frac{q_k^n q_l^n}{\ell_i(\boldsymbol{q}^n)} \ln\left[\frac{\ell_i(\boldsymbol{q}^n)}{q_k^n q_l^n} q_k q_l\right] \\ &= \sum_k c_k^n \ln q_k + c_0^n \end{aligned}$$

where

$$\begin{aligned} c_k^n &= \sum_i \sum_l \left[1_{(k,l)\in H_i} + 1_{(l,k)\in H_i}\right] \frac{q_k^n q_l^n}{\ell_i^n(\boldsymbol{q}^n)} \\ c_0^n &= \sum_i \sum_{(k,l)\in H_i} \frac{q_k^n q_l^n}{\ell_i(\boldsymbol{q}^n)} \ln\left[\frac{\ell_i(\boldsymbol{q}^n)}{q_k^n q_l^n}\right]. \end{aligned}$$

The penalty $p(q_j)$ is majorized by the linear function q_j when $q_j^n < \delta$. It is majorized by the constant δ when $q_j^n \geq \delta$. Multiplying the penalty majorization by $-\lambda$ gives a valid minorization of $-\lambda p(q_j)$. Finally, adding the minorization of $L(\boldsymbol{q})$ and the minorizations of the penalty terms $-\lambda p(q_j)$ gives the overall minorization

$$f(\boldsymbol{q}) \quad \geq \quad \sum_j c_j^n \ln q_j + c_0^n - \lambda \sum_{j:q_j^n<\delta} q_j - \lambda \sum_{j:q_j^n\geq\delta} \delta \tag{2}$$

of the objective function (1). This completes the description of the minorization step of the MM algorithm. The maximization step involves solving a sequence of quadratic equations that respect the constraints $q_j \geq 0$ for all j and $\sum_j q_j = 1$. Details are presented in our earlier paper [Ayers and Lange, 2008]. The bottom line is that the explicit MM updates are only slightly more complicated than the standard EM updates.

1.2 Genotype imputation and haplotype phasing

Our algorithms for imputing genotypes operate on a sliding window of constant width. Fixing the number of markers per window simplifies implementation. Constraining the maximum number of haplotypes considered per window to a constant $h_{\max}$ controls computational complexity. Within this overall framework, we implement two strategies that differ in their reliance on reference haplotypes and in the number of SNP positions imputed per window. In both strategies we first estimate haplotype frequencies within the window using the iterative MM algorithm just discussed.

The first strategy, introduced in our previous paper [Ayers and Lange, 2008], is relevant when there are no reference haplotypes to inform imputation. Even when reference haplotypes are available, an agnostic mode of imputation may be helpful in imputing variants that are not polymorphic in the external data. The drawback of ignoring reference haplotypes is that it is hard to guess in advance which haplotypes are pertinent to a window. In mitigation the window is slowly moved SNP by SNP across a genomic region, and partial haplotypes from the previous window are propagated and expanded to fill the current window. Because the window advances a single SNP at a time, only the central SNP of the window is imputed. Imputation is done by computing posterior probabilities. Let $r_{im}(s,t)$ denote the penetrance (likelihood) of person i's observed genotype at the central SNP m of the window given an underlying ordered genotype (s,t). Soft penetrances are generally preferable to hard genotype calls because noise is better taken into account. The posterior probability of an ordered genotype (s,t) at SNP m amounts to nothing more than the ratio

$$\frac{1}{\ell_i(\boldsymbol{q})} \sum_{(k_m,l_m)=(s,t)} q_k q_l r_{im}(s,t),$$

where the sum ranges over all ordered haplotypes (k,l) displaying the ordered genotype (s,t) at the central SNP, and $\ell_i(\boldsymbol{q})$ is the likelihood of i's observed multilocus genotypes in the window. The posterior probabilities of the various genotypes are translated into a posterior mean count of the number of minor or reference alleles at the central SNP. If hard imputations are desired, then the unordered genotype with the maximum posterior probability is reported.

Haplotype phasing operates in essentially the same manner as genotype imputation. Indeed, both procedures rely on the same algorithms for estimating haplotype frequencies and aggregating posterior evidence over pairs of haplotypes. The subtle difference is that during phasing, one distinguishes the two ordered genotypes of a heterozygote at a SNP and reports the ordered genotype with the higher posterior probability. In practice estimated haplotype frequencies are substituted for theoretical frequencies, and the likelihood $\ell_i(\boldsymbol{q})$ is computed with penetrances inserted. Likewise, the MM algorithm is performed with penetrances inserted.

Once imputation is complete at the central SNP, the existing set of haplotypes is modified in preparation for the next window. Modification is guided by several rules. First, any haplotype whose frequency falls below a threshold, say 10^{-8}, is pruned from the list of competing haplotypes. Second, if the number of remaining haplotypes is still greater than $\frac{1}{2}h_{\max}$, further rare haplotypes are pruned until this constraint is met. The remaining haplotypes are then copied to the next window and edited. The first edit crops the leftmost SNP. If haplotypes coincide after cropping, redundant copies are deleted. The second edit duplicates each remaining haplotype and appends a 0 to the right end of one member of the duplicated pair and a 1 to the right end of the other member of the duplicated pair. (Internally, the two alleles of a SNP are always represented as 0's and 1's.) The next round of haplotype frequency estimation and genotype imputation can now proceed with the revised list of haplotypes.

When reference haplotypes are available, our second strategy produces further gains in computational efficiency. The list of competing haplotypes now is equated to the unique haplotypes in the reference set spanning the current window. Instead of imputing a single SNP, we now employ a middle-third tactic. Thus, all w SNPs in the middle third of a window of width $3w$ are simultaneously imputed by posterior probabilities. Advancing to the next window requires deleting the left third and adding a new right third. The former right third now occupies the middle. Because haplotype frequency estimation is considerably more expensive than imputation and identification of the unique reference haplotypes spanning a window, this approach saves substantial computing.

1.3 GPU algorithms

Algorithms written for GPUs have moved well beyond computer games and animation, making inroads into diverse problems in computational biology such as proteomics [Hussong *et al.*, 2009], phylogenetics [Suchard and Rambaut, 2009; Zhou *et al.*, 2011a], gene-expression analysis [Magis *et al.*, 2011; Kohlhoff *et al.*, 2011; Buckner *et al.*, 2010], high dimensional optimization [Zhou *et al.*, 2010; Chen, 2012], epistasis modeling [Chikkagoudar *et al.*, 2011; Greene *et al.*, 2010; Kam-Thong *et al.*, 2011; Yung *et al.*, 2011; Ritchie and Venkatraman, 2010; Hemani *et al.*, 2011], sequence alignment [Blom *et al.*, 2011; Campagna *et al.*, 2009; Vouzis and Sahinidis, 2011; Liu *et al.*, 2012b], and systems biology [Klingbeil *et al.*, 2011; Vigelius *et al.*, 2011; Liu *et al.*, 2012a; Zhou *et al.*, 2011b; Liepe *et al.*, 2010]. Our algorithms for haplotype frequency estimation and imputation offer ample opportunities for acceleration. The sheer amount of parallelism exposed at key steps of the algorithms is impressive. Because GPU cores are optimized for parallel arithmetic operations, we expect and realize in practice major computing gains.

There are currently two development frameworks for writing code for GPU devices. CUDA is a proprietary framework from nVidia that has garnered considerable interest in the scientific community [nVidia, 2011]. OpenCL is a specification backed by nVidia, ATI, and others that offers superior code portability. Indeed, OpenCL programs will run on nVidia and ATI GPUs and even Intel and AMD multi-core CPUs. Nevertheless, it has proved less popular than CUDA, perhaps due to its lack of utilities and a steeper learning curve [Khronos, 2011]. Since we considered portability crucial, we based our software on OpenCL.

Programming a GPU device requires more attention to the underlying hardware than programming a CPU. In particular, OpenCL specifies a memory model in which programmers explicitly direct where data is to be stored among three layers of memory at each state of a run. The system RAM typically accessed by conventional CPU programs is by far the most abundant (64GB on typical high-end servers), but it has much higher latency (access times) when data is transmitted to a GPU device over a low bandwidth PCIx bus (the slot on a motherboard where a GPU device is attached). Global memory has much lower latency since it is physically located on the GPU device, but is relatively scarce (2GB on high-end GPUs). Local memory, shared across cores, is far scarcer (generally 16-64 KB per GPU) than global memory but has very low latency because it is mounted adjacent to the arithmetic cores on each GPU. Hence, good GPU programming strategy involves writing code that accesses RAM and global memory infrequently and performs the bulk of computations in local memory.

It is helpful to demonstrate these concepts in practice through actual code. Listing 1 is a short excerpt from our kernel computing haplotype pair penetrances under our first imputation scheme. Prior to computations, data is transferred from global to local memory in a coalesced manner to maximize available bandwidth (efficiency of data transfers). In other words, data is read from or written to global memory in a block of some fixed number of elements as a single instruction that runs in parallel across cores. There are 32 elements on the majority of nVidia GPUs. In our code, we set the constant BLOCK_WIDTH equal to 256, a multiple of 32, in order to minimize memory transfers that are not fully coalesced. For downstream computations, staying within the confines of local memory can lead to more than a ten-fold speed improvement over code accessing global memory.

Listing 1: Coalesced memory reads

```
// get subject identifier, mapping to workgroup
int subject = get_group_id(0);
// get thread ID, mapping to work item
int threadindex = get_local_id(0);
// iterate in chunks of BLOCK_WIDTH
for(int chunk=0;
chunk<(max_haplotypes/BLOCK_WIDTH)+1;
++chunk){
  int hapindex = chunk*BLOCK_WIDTH+threadindex;
  if (hapindex<max_haplotypes){
```

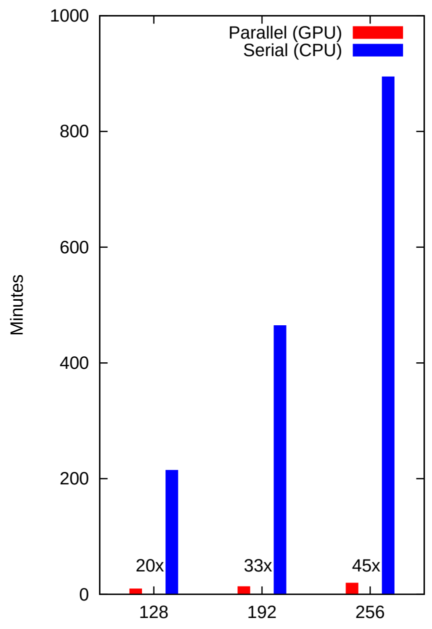

Figure 1: Speedup as a function of haplotype diversity

```
    local_active_haplotype[hapindex]
   = active_haplotype[hapindex];
  }
}
```

We have implemented serial versions of the core algorithms in C++. Code written in FORTRAN90 handles the entry point of our program and orchestrates calling of the various C++ routines. If a GPU device is available, users can activate a flag to enable execution of our OpenCL kernels, which are dispatched by the C++ routines.

2 Results

2.1 Scalability properties of the GPU implementation

We first verified the correctness of our GPU code by comparing its results to the results of the corresponding CPU code. Once we were convinced that the GPU code was correct, we ran benchmarking tests on real data to measure performance gains on our GPU device, the nVidia Tesla C2050, which has 448 cores and 2.5 GB of global memory. We downloaded the March 2012 release of Phase 1 of the 1000 Genomes Project (KGP) [Altshuler *et al.*, 2010] from the *IMPUTE2* website [Howie *et al.*, 2009]. Three imputation runs, using our first imputation scheme (no external haplotypes), were carried out on a subset of the KGP data consisting of 1,092 individuals typed across a random 1MB region on Chr 22. For each run, we set $h_{\max}$ to 128, 192, and 256 and recorded the total execution times on both the standard serial version (running on the CPU) and the GPU version. The GPU version required 10, 14, and 20 minutes respectively, while the serial version required 215, 465, and 895 minutes. We observed super-linear speedups with respect to problem size as shown in Figure 1. In other words, speedups increase as more parallelism is exposed to the device. This phenomenon is explained by the fact that when the ratio of computation to data transfers increases (as problem size increases), latency in computations in some threads are more effectively masked by data transfer in other threads.

2.2 Simulation of test data

To keep imputation times manageable, we restricted analysis to the random 1MB region on Chr 22 from the KGP data just mentioned [Altshuler *et al.*, 2010]. The underlying 2,184 haplotypes constitute a unique resource for further simulation. To mimic low-coverage sequencing, we post-processed the KGP data, assuming that the haplotypes originally deduced by KGP were true haplotypes. (Since MACH and IMPUTE2 were used in defining these haplotypes, the failure of this assumption may work to the advantage of these programs in our later assessments of imputation accuracy.) To determine sequencing depth at each person-SNP combination, we sampled from a gamma distribution with shape 3.5 and scale 0.8. Rounding the generated random deviates to the nearest integer gives an average depth of coverage of 2.8x with a standard deviation of 1.5, roughly consistent with the 2-4x coverage of Phase 1 of KGP. Finally, to regenerate the allele counts (a, b) for the two possible alleles A and B of each person-SNP combination, we sampled from the conditional binomial distributions

$$\begin{aligned}
\Pr(a, b|A/A) &= \binom{a+b}{a}(1-\epsilon)^a\epsilon^b \\
\Pr(a, b|A/B) &= \binom{a+b}{a}(1/2)^{(a+b)} \\
\Pr(a, b|B/B) &= \binom{a+b}{a}\epsilon^a(1-\epsilon)^b,
\end{aligned}$$

where ϵ is the reading error rate. In all simulations ϵ was fixed at 0.01. The binomial penetrances of each sampled pair were passed to *Mendel-GPU* and comparison programs for imputation and haplotyping.

Table 1: Accuracy and runtime as a function of MM tuning constants

	Accuracy						Total Runtime		
	Imputation			Phasing					
	λ								
δ	10	1000	10^5	10	1000	10^5	10	1000	10^5
.1	.928	.928	.915	.945	.946	.943	20:04	19:25	17:23
.001	.929	.929	.928	.944	.945	.943	20:03	19:44	19:22
10^{-5}	.928	.928	.928	.945	.945	.944	20:03	20:01	20:01

2.3 Selection of tuning constants

Based on the post-processed KGP data, we explored the impact of window width (the number of flanking markers) and the MM tuning constants δ and λ on performance. One metric of genotype imputation quality is heterozygote accuracy. This metric is preferred for rare SNPs since simply guessing the wild-type homozygote for all subjects can lead to deceptively high accuracies. Heterozygote accuracy is computed by taking the proportion of true heterozygote sites that are correctly imputed. We also evaluated the quality of haplotype phasing by monitoring switch errors over long genomic regions [Browning and Browning, 2011]. To compute switch error rates, we considered only intervals between sites where the imputed haplotype pair and the true haplotype pair are both heterozygous and hence informative. A switch error occurs on an interval if a crossover event must be included in the imputed data to be consistent with the phase of the previous interval. Switch accuracy is simply defined as one minus switch error rate.

Figure 2 plots two curves, heterozygote and switch accuracy, as a function of the number of flanking markers per window. Consistent with findings from our earlier paper [Ayers and Lange, 2008], accuracy of both phasing and imputation initially increases as window width increases. Beyond a certain point, imputation accuracy falters. A good balance between phasing and imputation accuracy can be struck by limiting the number of flanking markers to the range of 40 to 90. Given 90 flanking markers, we tracked accuracy and runtimes over a grid of nine pairs (δ, λ), with δ drawn from the set $\{10^{-5}, 0.001, .1\}$ and λ from the set $\{10^5, 1000, 10\}$. Heterozygote accuracies were virtually identical (.928-.929) except when δ and λ were set to their maximum values; in this case accuracy fell slightly to .925. Phasing accuracies were consistently in the range of .943 to .946 across all nine parameter pairs. Runtimes drop slightly with larger values of λ and δ. In particular, the combination $(\lambda, \delta) = (10^5, .1)$ strongly emphasizes the penalty and drops execution time to slightly over 17 minutes. Based on these observations, window width has a small but somewhat larger impact on accuracy and speed than the MM tuning constants. Overall, we see the kind of robustness one seeks in a penalty driven method.

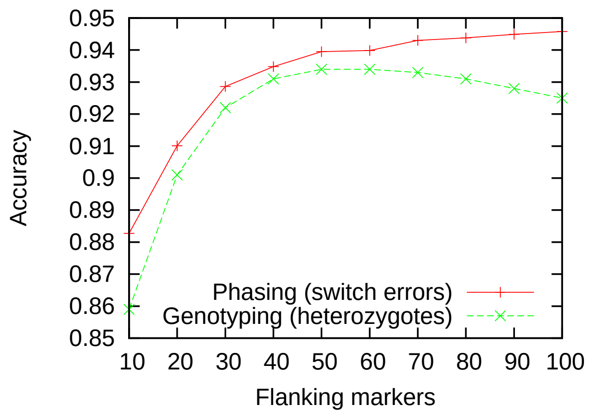

Figure 2: Accuracy as a function of number of flanking markers per window

2.4 Program settings

We compared the performance of our program *Mendel-GPU* to leading programs for genotype imputation: in particular *MaCH*, *thunder*, *IMPUTE2*, and *BEAGLE*. For all tests, we fixed the number of flanking markers at 90, δ at .001, λ at 1000, and $h_{\max}$ (the maximum number of allowed haplotypes) at 256. *thunder*, an extension of *MaCH*, supports input of genotype penetrances but does not accommodate reference haplotypes. We applied the recommended settings for each comparison programs. Specifically, we set the number of haplotype states to 200 and total MCMC rounds to 30 for *MaCH* and *thunder*. For *IMPUTE2*, we requested 80 haplotype states and 30 MCMC rounds. Finally, we set the number of sampled haplotype pairs per subject to 4 and total MCMC iterations to 10 for *BEAGLE*.

2.5 Comparison metrics

For each scenario, we tracked overall computational efficiency and accuracy. Memory usage was taken as the peak demand on RAM. Peak demand is important because users need to decide in advance how much memory to reserve for jobs on compute clusters. It is noteworthy that *BEAGLE*, a Java program, fluctuated greatly in its memory usage; average usage was substantially lower than peak usage. Runtimes were reported via the *time* command in Linux. In addition to calculating heterozygote accuracy as described earlier, we also considered the ℓ_1 norm of dosage errors. This measure of imputation accuracy is defined as the sum of the absolute differences between imputed allele counts and true allele counts. Allele dosages are preferable to hard imputations in association studies because they properly account for imputation uncertainty.

Table 2: Genotype imputation in a study based on Illumina 2.5M microarray data using KGP reference haplotypes

Program	Hetero. Accuracy	ℓ_1 norm of errors	Total Runtime	Max Memory Footprint
Mendel-GPU	.872	1975975.417	14:22	575MB
BEAGLE	.893	1739268.964	13:06:23	3.2GB
IMPUTE2	.901	1575050.833	8:25:00	1.5GB
MaCH	.882	2053479.833	11:40:00	1.1GB

2.6 Simulating data sets with reference haplotypes

To simulate experiments with reference haplotypes, we reserved one half of the entire KGP Phase 1 data as the set of reference haplotypes and the remaining half as a hypothetical study dataset. Each ethnic group was evenly divided over the two halves. To keep run times within reasonable bounds for all programs, we restricted analyses to a randomly selected 7 MB region on chromosome 22. The primary (study) data is assumed to be generated from a low-pass sequencing assay. The idea is to leverage ethnically matched reference haplotypes to improve confidence in genotypes called with low certainty. To prepare the primary dataset, we applied the same procedure as described earlier for simulating coverage depth from a gamma distribution. Since *MaCH* supports only discrete genotypes in its input and *thunder* does not support reference haplotypes, neither of these programs could be included in comparisons based on sequencing data in the study.

2.7 Imputation where GWAS data is derived from microarray chip

Here we consider a traditional configuration for genotype imputation. We suppose that GWAS data was generated by a high-coverage genotyping microarray, in particular Illumina's Infinium 2.5M Duo product, which features approximately 2.4 million SNPs. The goal here is to impute genotypes for SNPs that are present in the reference haplotype panel, but not on the chip. We split the KGP data into two halves in the same manner as above, and restricted analysis to a 7MB region. Genotypes in the hypothetical study were masked (set to missing) at a SNP unless it was listed in Illumina's manifest file for the 2.5M Duo. Table 2 presents speed, memory usage, and our two measures of accuracy. *BEAGLE* and *IMPUTE2* had slightly higher accuracies than *Mendel-GPU* and *MaCH*. Comparing just the latter two programs, *Mendel-GPU* gave better dosage predictions, while *MaCH* gave a higher heterozygote concordance rate. In terms of speed however, *Mendel-GPU* was 55, 35, and 48 times faster than *BEAGLE*, *IMPUTE2*, and *MaCH*, respectively. *Mendel-GPU* required only half as much memory as *MaCH*, the most efficient consumer of memory among the three contenders.

References

Altshuler, D. *et al.* (2010). A map of human genome variation from population-scale sequencing. *Nature*, **467**(7319), 1061–1073.

Ayers, K. L. and Lange, K. (2008). Penalized estimation of haplotype frequencies. *Bioinformatics*, **24**(14), 1596–1602.

Blom, J., Jakobi, T., Doppmeier, D., Jaenicke, S., Kalinowski, J., Stoye, J., and Goesmann, A. (2011). Exact and complete short-read alignment to microbial genomes using graphics processing unit programming. *Bioinformatics*, **27**(10), 1351–1358.

Browning, S. R. and Browning, B. L. (2011). Haplotype phasing: existing methods and new developments. *Nat. Rev. Genet.*, **12**(10), 703–714.

Buckner, J., Wilson, J., Seligman, M., Athey, B., Watson, S., and Meng, F. (2010). The gputools package enables gpu computing in r. *Bioinformatics*, **26**(1), 134–135.

Campagna, D., Albiero, A., Bilardi, A., Caniato, E., Forcato, C., Manavski, S., Vitulo, N., and Valle, G. (2009). Pass: a program to align short sequences. *Bioinformatics*, **25**(7), 967–968.

Chen, G. K. (2012). A scalable and portable framework for massively parallel variable selection in genetic association studies. *Bioinformatics*, **28**(5), 719–720.

Chikkagoudar, S., Wang, K., and Li, M. (2011). GENIE: a software package for gene-gene interaction analysis in genetic association studies using multiple GPU or CPU cores. *BMC Res Notes*, **4**, 158.

Greene, C. S., Sinnott-Armstrong, N. A., Himmelstein, D. S., Park, P. J., Moore, J. H., and Harris, B. T. (2010). Multifactor dimensionality reduction for graphics processing units enables genome-wide testing of epistasis in sporadic als. *Bioinformatics*, **26**(5), 694–695.

Hemani, G., Theocharidis, A., Wei, W., and Haley, C. (2011). Epigpu: exhaustive pairwise epistasis scans parallelized on consumer level graphics cards. *Bioinformatics*, **27**(11), 1462–1465.

Howie, B. N., Donnelly, P., and Marchini, J. (2009). A flexible and accurate genotype imputation method for the next generation of genome-wide association studies. *PLoS Genet.*, **5**(6), e1000529.

Hussong, R., Gregorius, B., Tholey, A., and Hildebrandt, A. (2009). Highly accelerated feature detection in proteomics data sets using modern graphics processing units. *Bioinformatics*, **25**(15), 1937–1943.

Kam-Thong, T., Ptz, B., Karbalai, N., MllerMyhsok, B., and Borgwardt, K. (2011). Epistasis detection on quantitative phenotypes by exhaustive enumeration using gpus. *Bioinformatics*, **27**(13), i214–i221.

Khronos (2011). Opencl specification. http://www.khronos.org.

Klingbeil, G., Erban, R., Giles, M., and Maini, P. K. (2011). Stochsimgpu: parallel stochastic simulation for the systems biology toolbox 2 for matlab. *Bioinformatics*, **27**(8), 1170–1171.

Kohlhoff, K. J., Sosnick, M. H., Hsu, W. T., Pande, V. S., and Altman, R. B. (2011). Campaign: an open-source library of gpu-accelerated data clustering algorithms. *Bioinformatics*, **27**(16), 2321–2322.

Liepe, J., Barnes, C., Cule, E., Erguler, K., Kirk, P., Toni, T., and Stumpf, M. P. (2010). Abc-sysbioapproximate bayesian computation in python with gpu support. *Bioinformatics*, **26**(14), 1797–1799.

Liu, B., Hagiescu, A., Palaniappan, S. K., Chattopadhyay, B., Cui, Z., Wong, W.-F., and Thiagarajan, P. S. (2012a). Approximate probabilistic analysis of biopathway dynamics. *Bioinformatics*, **28**(11), 1508–1516.

Liu, C.-M., Wong, T., Wu, E., Luo, R., Yiu, S.-M., Li, Y., Wang, B., Yu, C., Chu, X., Zhao, K., Li, R., and Lam, T.-W. (2012b). Soap3: ultra-fast gpu-based parallel alignment tool for short reads. *Bioinformatics*, **28**(6), 878–879.

Magis, A. T., Earls, J. C., Ko, Y.-H., Eddy, J. A., and Price, N. D. (2011). Graphics processing unit implementations of relative expression analysis algorithms enable dramatic computational speedup. *Bioinformatics*, **27**(6), 872–873.

nVidia (2011). nvidia cuda. http://www.nvidia.com.

Ritchie, D. W. and Venkatraman, V. (2010). Ultra-fast fft protein docking on graphics processors. *Bioinformatics*, **26**(19), 2398–2405.

Suchard, M. A. and Rambaut, A. (2009). Many-core algorithms for statistical phylogenetics. *Bioinformatics*, **25**(11), 1370–1376.

Vigelius, M., Lane, A., and Meyer, B. (2011). Accelerating reactiondiffusion simulations with general-purpose graphics processing units. *Bioinformatics*, **27**(2), 288–290.

Vouzis, P. D. and Sahinidis, N. V. (2011). Gpu-blast: using graphics processors to accelerate protein sequence alignment. *Bioinformatics*, **27**(2), 182–188.

Yung, L. S., Yang, C., Wan, X., and Yu, W. (2011). Gboost: a gpu-based tool for detecting genegene interactions in genomewide case control studies. *Bioinformatics*, **27**(9), 1309–1310.

Zhou, H., Lange, K., and Suchard, M. A. (2010). Graphics Processing Units and High-Dimensional Optimization. *Stat Sci*, **25**(3), 311–324.

Zhou, J., Liu, X., Stones, D. S., Xie, Q., and Wang, G. (2011a). Mrbayes on a graphics processing unit. *Bioinformatics*, **27**(9), 1255–1261.

Zhou, Y., Liepe, J., Sheng, X., Stumpf, M. P. H., and Barnes, C. (2011b). Gpu accelerated biochemical network simulation. *Bioinformatics*, **27**(6), 874–876.