GENOTYPE IMPUTATION AND PHASING WITH MendelImpute.jl

OpenMendel Programming Workshop

ASHG 2020

Benjamin Chu

Software: https://github.com/OpenMendel/MendelImpute.jl
Live demo: https://mybinder.org/v2/gh/OpenMendel/ASHG-OpenMendelWorkshop-2020-Oct/master
Paper: Should be on bioarxiv when you see this video

Imputation and Phasing

- Imputing genotypes at all markers in a reference genotype panel

- Inputs:
  - Target (sample) genotypes at ~1M markers (unphased, entries 0, 1, 2)
  - A reference panel of phased genotypes (entries 0, 1) at >40M markers
- Output:
  - Phased genotypes at all markers

Software Demonstration

Step 1: Compress reference panels

```
using MendelImpute
reffile = "./data/ref.excludeTarget.vcf.gz"
tgtfile = "./data/target.typedOnly.masked.vcf.gz"
outfile = "./data/ref.excludeTarget.jlso"
@time compress_haplotypes(reffile, tgtfile, outfile)
```

```
importing reference data...100%|████████████████████████████████| Time: 0:00:13
 27.176913 seconds (211.00 M allocations: 14.879 GiB, 9.88% gc time)
```

Step 2: Phase and Impute

```
tgtfile = "./data/target.typedOnly.masked.vcf.gz"
reffile = "./data/ref.excludeTarget.jlso"
outfile = "./data/imputed.vcf.gz"
phase(tgtfile, reffile, outfile);
```

```
Number of threads = 1
Importing reference haplotype data...
Total windows = 16, averaging ~ 708 unique haplotypes per window.

Timings:
    Data import                          = 0.73262 seconds
        import target data                  = 0.569068 seconds
        import compressed haplotypes        = 0.163552 seconds
    Computing haplotype pair             = 0.299327 seconds
        BLAS3 mul! to get M and N           = 0.026954 seconds per thread
        haplopair search                    = 0.253962 seconds per thread
        initializing missing                = 0.00432748 seconds per thread
        allocating and viewing              = 0.0138387 seconds per thread
        index conversion                    = 0.000123722 seconds per thread
    Phasing by win-win intersection = 0.059 seconds
        Window-by-window intersection       = 0.000994633 seconds per thread
        Breakpoint search                   = 0.0567691 seconds per thread
        Recording result                    = 0.00104923 seconds per thread
    Imputation                          = 0.289381 seconds
        Imputing missing                    = 0.00287408 seconds
        Writing to file                     = 0.286507 seconds

    Total time                           = 1.3809 seconds
```

Post analysis: Import & Manipulate data

```
using VCFTools

# import genotypes as double-precision matrices
Ximputed = convert_gt(Float64, "./data/imputed.vcf.gz")
Xtrue = convert_gt(Float64,"./data/target.full.vcf.gz")

# calculate error rate
n, p = size(Xtrue)
err = sum(Xtrue .!= Ximputed)/n/p
println("error rate = $err")

# do standard matrix operations
Ximputed[1:5, 1:5] * rand(5)
```

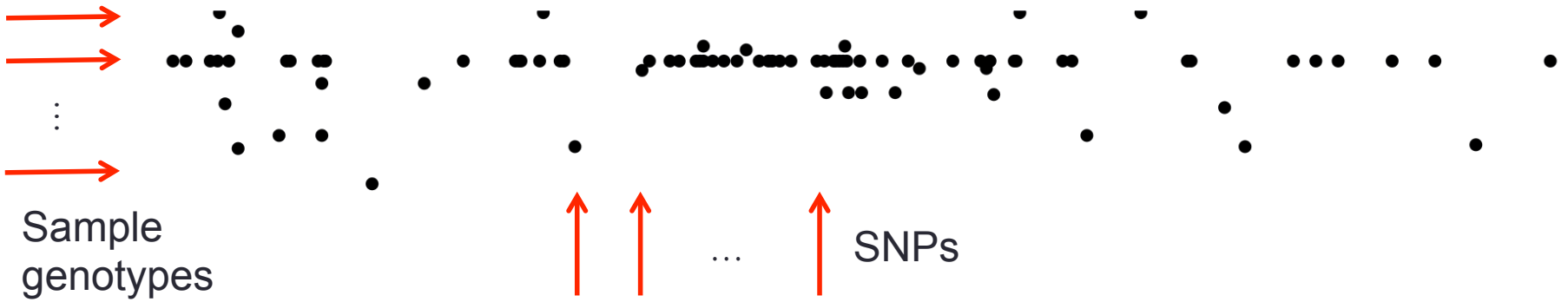
```
error rate = 0.0007245653439813659

5-element Array{Union{Missing, Float64},1}:
 0.37960123900222476
 0.7592024780044495
 0.7592024780044495
 0.7592024780044495
 0.9665350623077842
```

- VCFTools.jl imports VCF files as numeric matrices
- Instant access to statistical packages that support a “matrix” type
- BLAS and LAPACK libraries are automatically hooked up

Post analysis: Visualization

Julia is shipped with a rich plotting ecosystem. For example, we can visualize error:

```
disagreeing_entries = sparse(Xtrue .!= Ximputed)
spy(disagreeing_entries, size=(40000, 100), markersize = 3)
```

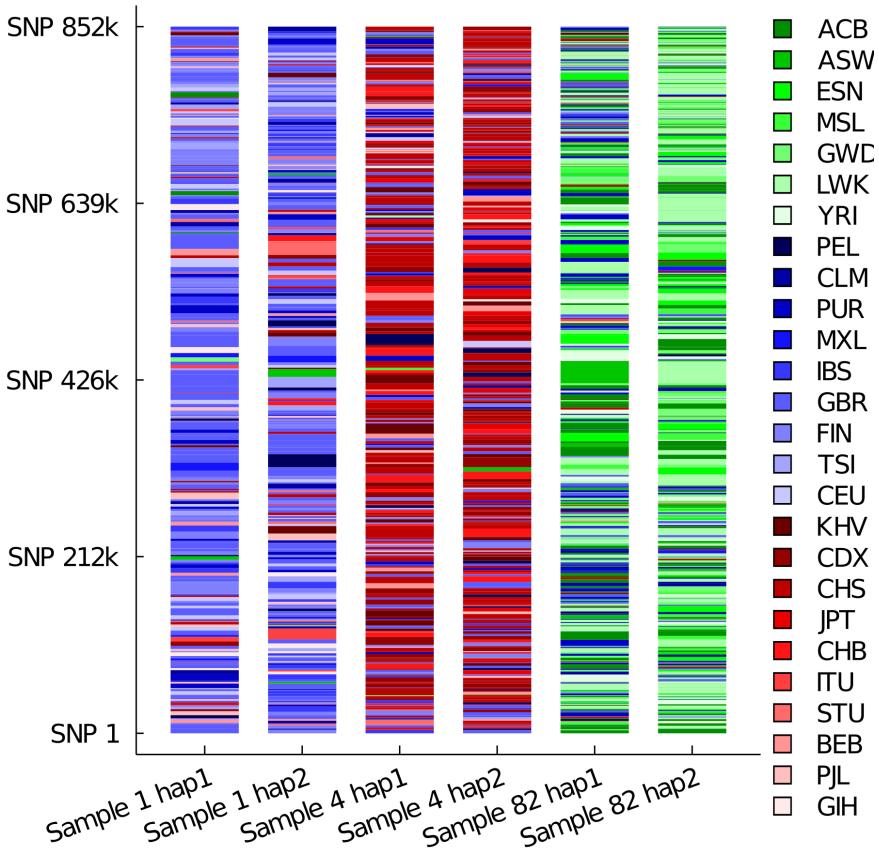


Speed, Memory, Error rate

sim 10k	Error	Time (sec)	Memory (GB)
MendelImpute	4.02E-04	16	1.6
Beagle 5.1	6.21E-05	260	8.3
Minimac 4	5.95E-05	503	6.6
sim 100k	Error	Time (sec)	Memory (GB)
MendelImpute	7.34E-05	18	2.0
Beagle 5.1	1.12E-05	343	18.2
Minimac 4	1.09E-05	3282	16
sim 1M	Error	Time (sec)	Memory (GB)
MendelImpute	1.90E-04	30	4.7
Beagle 5.1	6.60E-06	825	26.8
Minimac 4	NA	NA	≥ 64
1000G chr10	Error	Time (sec)	Memory (GB)
MendelImpute	1.10E-02	48	4.6
Beagle 5.1	5.51E-03	190	10
Minimac 4	5.24E-03	291	10.4
1000G chr20	Error	Time (sec)	Memory (GB)
MendelImpute	3.30E-02	21	2.07
Beagle 5.1	1.68E-02	51	10.3
Minimac 4	1.65E-02	588	5.8
HRC chr10	Error	Time (sec)	Memory (GB)
MendelImpute	6.41E-03	167	13.4
Beagle 5.1	1.79E-03	1889	33.3
Minimac 4	1.71E-03	13880	17.8
HRC chr20	Error	Time (sec)	Memory (GB)
MendelImpute	1.31E-03	138	9.1
Beagle 5.1	5.01E-04	2413	25.5
Minimac 4	6.05E-04	16627	33.7

- 10-150x faster than Beagle 5.1 or Minimac 4
- Increasing panel size 100x increase compute time by <2x
- Uses less RAM
- Error rate is somewhat higher but still acceptable

Extension to Ancestry Inference and new Data Compression Strategies



Dataset	.vcf.gz	ultra-compressed	compression ratio
Sim 10k	10.54	0.04	263
Sim 100k	11.12	0.04	278
Sim 1M	9.61	0.11	87
1000G Chr10	13.87	0.37	37
1000G Chr20	31.4	1.23	25
HRC Chr10	157.85	7.13	22
HRC Chr20	70.76	5.63	13

(units = Megabytes)

Conclusion

- MendelImpute.jl, with SnpArrays.jl and VCFTools.jl, offer a fast and intuitive pipeline for imputation and phasing

- After imputation, your data is automatically setup for statistical analysis and visualization

Software is on GitHub: https://github.com/OpenMendel/MendelImpute.jl
Latest documentation: https://openmendel.github.io/MendelImpute.jl/dev/

ASHG tutorial:
https://github.com/OpenMendel/ASHG-OpenMendelWorkshop-2020-Oct