Fast Epistasis Detection in Large-Scale GWAS for Intel Xeon Phi Clusters

Glenn R. Luecke[1] Nathan T. Weeks[1] Brandon M. Groth[1]
Marina Kraeva[2] Li Ma[4] Luke M. Kramer[3]
James E. Koltes[3] James M. Reecy[3]

[1]Department of Mathematics, [2]IT Services Academic Technologies,
[3]Department of Animal Science, Iowa State University, USA

[4]Department of Animal and Avian Sciences, University of Maryland, USA

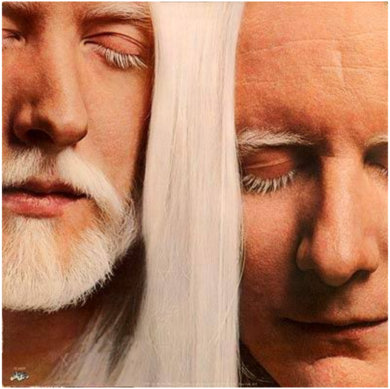
Third International Workshop on Parallelism in Bioinformatics
(PBIO 2015)

Introduction
Epistasis

Epistasis is the interaction of genes to determine a trait.

Example: albinism (recessive epistasis)

Right: Albino brothers (and famous musicians) Edgar and Johnny Winter (*Source: Edgar Winter*)



Introduction
Epistasis

A **quantitative trait** is a phenotype that is expressed as a quantity (e.g., height, weight).

> Studies suggest that epistasis is involved in variety of quantitative traits in humans, such as blood pressure, cholesterol, triglycerides...

A **Genome-Wide Association Study (GWAS)** is a statistical association of single nucleotide polymorphisms (SNPs) with phenotypes (observable traits).

Introduction
Epistasis

A **quantitative trait** is a phenotype that is expressed as a quantity (e.g., height, weight).

> Studies suggest that epistasis is involved in variety of quantitative traits in humans, such as blood pressure, cholesterol, triglycerides...

A **Genome-Wide Association Study (GWAS)** is a statistical association of single nucleotide polymorphisms (SNPs) with phenotypes (observable traits).

Introduction
EPISNPmpi

epiSNP (Ma et al.) is software for detecting epistasis in quantitative-trait GWAS.

Testing for pairwise epistasis is computationally expensive, requiring a large number of individuals, and $\mathcal{O}(n^2)$ pairwise genetic marker tests.

EPISNPmpi was created in 2008 to handle large-scale GWAS.

Falling costs of sequencing and genotyping means larger data than EPISNPmpi was designed to handle.

Introduction
EPISNPmpi

epiSNP (Ma et al.) is software for detecting epistasis in quantitative-trait GWAS.

Testing for pairwise epistasis is computationally expensive, requiring a large number of individuals, and $\mathcal{O}(n^2)$ pairwise genetic marker tests.

EPISNPmpi was created in 2008 to handle large-scale GWAS.

Falling costs of sequencing and genotyping means larger data than EPISNPmpi was designed to handle.

Introduction
EPISNPmpi

epiSNP (Ma et al.) is software for detecting epistasis in quantitative-trait GWAS.

Testing for pairwise epistasis is computationally expensive, requiring a large number of individuals, and $\mathcal{O}(n^2)$ pairwise genetic marker tests.

EPISNPmpi was created in 2008 to handle large-scale GWAS.

Falling costs of sequencing and genotyping means larger data than EPISNPmpi was designed to handle.

Introduction
EPISNPmpi

epiSNP (Ma et al.) is software for detecting epistasis in quantitative-trait GWAS.

Testing for pairwise epistasis is computationally expensive, requiring a large number of individuals, and $\mathcal{O}(n^2)$ pairwise genetic marker tests.

EPISNPmpi was created in 2008 to handle large-scale GWAS.

Falling costs of sequencing and genotyping means larger data than EPISNPmpi was designed to handle.

Introduction
Our Work

To address performance challenges posed by larger data sets, we implemented the following improvements to epiSNP:

- serial optimizations
- improved data distribution for better load balancing and to allow execution using an arbitrary number of MPI processes
- shared-memory parallelism to reduce memory footprint and further improve load balancing
- port to the Intel Xeon Phi coprocessor

Results (average speedup vs. original EPISNPmpi):

- 15X (MPI)
- 17X (MPI+OpenMP)
- 28X (MPI+OpenMP with 1 Xeon Phi per node)
- 36X (MPI+OpenMP with 2 Xeon Phis per node)

Introduction

Our Work

To address performance challenges posed by larger data sets, we implemented the following improvements to epiSNP:

- serial optimizations
- improved data distribution for better load balancing and to allow execution using an arbitrary number of MPI processes
- shared-memory parallelism to reduce memory footprint and further improve load balancing
- port to the Intel Xeon Phi coprocessor

Results (average speedup vs. original EPISNPmpi):

- 15X (MPI)
- 17X (MPI+OpenMP)
- 28X (MPI+OpenMP with 1 Xeon Phi per node)
- 36X (MPI+OpenMP with 2 Xeon Phis per node)

Serial Optimizations

Serial Optimizations
Data Type Changes

Some advantages of using smaller data types for arrays:

- Less main memory usage
- More efficient cache utilization
- SIMD instructions can operate on more data elements per instruction.

Main changes to epiSNP:

- SNP genotypes stored in 1-byte integer instead of 4-byte integer
- Phenotype values can be stored in single precision instead of double (customizable at compile time)

Serial Optimizations

Data Type Changes

Some advantages of using smaller data types for arrays:

- Less main memory usage
- More efficient cache utilization
- SIMD instructions can operate on more data elements per instruction.

Main changes to epiSNP:

- SNP genotypes stored in 1-byte integer instead of 4-byte integer
- Phenotype values can be stored in single precision instead of double (customizable at compile time)

Serial Optimizations

Dead Code Elimination

A computationally-expensive p-value calculation (accounting for 45% of the original EPISNPmpi run time) was performed for all pairs of SNPs, but this value was not output.

This p-value was recalculated and output for SNPs with the most significant interaction effects.

Compilers try to detect and eliminate *dead code*, but it was too difficult in this case.

Serial Optimizations
Dead Code Elimination

A computationally-expensive p-value calculation (accounting for 45% of the original EPISNPmpi run time) was performed for all pairs of SNPs, but this value was not output.

This p-value was recalculated and output for SNPs with the most significant interaction effects.

Compilers try to detect and eliminate *dead code*, but it was too difficult in this case.

Serial Optimizations

Dead Code Elimination

A computationally-expensive p-value calculation (accounting for 45% of the original EPISNPmpi run time) was performed for all pairs of SNPs, but this value was not output.

This p-value was recalculated and output for SNPs with the most significant interaction effects.

Compilers try to detect and eliminate *dead code*, but it was too difficult in this case.

Serial Optimizations

Loop Optimization and Vectorization

Branching (IF statements) inside of loops is bad for performance:

- Inhibits vectorization
- Xeon Phi lacks branch prediction hardware

Code was restructured to remove IF statements from loops where possible.

Serial Optimizations
CDFLIB

epiSNP uses a third-party library of cumulative distribution functions (CDFLIB).

- Replacing *cdft()* with *cumt()* improved performance
- Declaring *cumt()* as `elemental` in an interface block further improved performance
- Using equivalent CDFLIB90 subroutine (*cum_t()*) resulted in severe performance regression, despite claims of improved performance of CDFLIB90 over CDFLIB

Serial Optimizations
CDFLIB

epiSNP uses a third-party library of cumulative distribution functions (CDFLIB).

- Replacing *cdft()* with *cumt()* improved performance
- Declaring *cumt()* as `elemental` in an interface block further improved performance
- Using equivalent CDFLIB90 subroutine (*cum_t()*) resulted in severe performance regression, despite claims of improved performance of CDFLIB90 over CDFLIB

Serial Optimizations
CDFLIB

epiSNP uses a third-party library of cumulative distribution functions (CDFLIB).

- Replacing *cdft()* with *cumt()* improved performance
- Declaring *cumt()* as `elemental` in an interface block further improved performance
- Using equivalent CDFLIB90 subroutine (*cum_t()*) resulted in severe performance regression, despite claims of improved performance of CDFLIB90 over CDFLIB

MPI

MPI
EPISNPmpi Data Distribution

The data distribution used by EPISNPmpi has two limitations:

- Load imbalance: some ranks compute $5n^2$ pairwise tests, while others compute $3n + \frac{5n(n-1)}{2} = \frac{5}{2}n^2 + \frac{1}{2}n$ pairwise tests
- Number of MPI processes used (P) limited to $P \in \{1, 1+2, 1+2+3, ...\}$

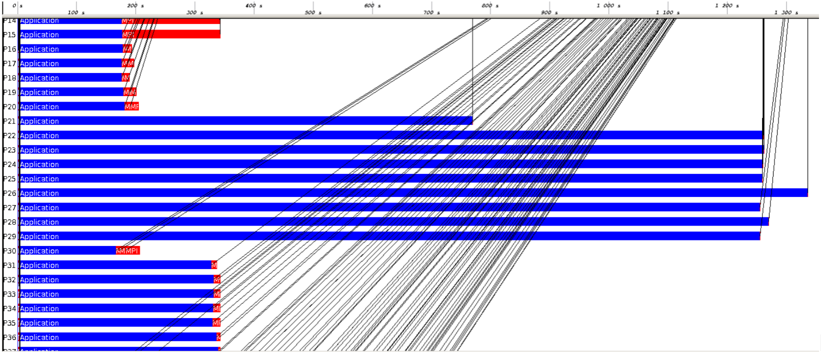
MPI
EPISNPmpi Data Distribution

The data distribution used by EPISNPmpi has two limitations:

- Load imbalance: some ranks compute $5n^2$ pairwise tests, while others compute $3n + \frac{5n(n-1)}{2} = \frac{5}{2}n^2 + \frac{1}{2}n$ pairwise tests
- Number of MPI processes used (P) limited to $P \in \{1, 1+2, 1+2+3, ...\}$

MPI

EPISNPmpi Load Imbalance



Intel Trace Analyzer Event Timeline for EPISNPmpi ranks 14-37 (out of 56) Each horizontal bar is an MPI process (blue: program execution; red: MPI communication). Black lines connect communicating processes.

MPI

Improved Load Balancing

Example: $N = 10$ SNPs, $P = 4$ MPI processes

SNP_1 \ SNP_2	1	2	3	4	5	6	7	8	9	10	
1											rank 1
2											rank 2
3											rank 3
4											rank 4
5											rank 4
6											rank 3
7											rank 2
8											rank 1
9											rank 1

MPI

Improved Load Balancing

This improved load balancing method:

- Alleviates most load imbalance
- Allows the use of an arbitrary number of MPI ranks, allowing a flexible number of nodes/cores/Xeon Phi coprocessors
- Facilitates OpenMP parallelization

MPI

Improved Load Balancing

This improved load balancing method:

- Alleviates most load imbalance
- Allows the use of an arbitrary number of MPI ranks, allowing a flexible number of nodes/cores/Xeon Phi coprocessors
- Facilitates OpenMP parallelization

MPI
Improved Load Balancing

This improved load balancing method:

- Alleviates most load imbalance
- Allows the use of an arbitrary number of MPI ranks, allowing a flexible number of nodes/cores/Xeon Phi coprocessors
- Facilitates OpenMP parallelization

OpenMP

OpenMP

OpenMP was used to implement shared-memory parallelism in epiSNP

- Facilitates more efficient load balancing
- Allows MPI rank 0 to read input data faster, using multiple threads
- Reduces MPI communication
- Reduces memory footprint by sharing data structures that would otherwise be replicated
- Required to fully utilize Intel Xeon Phi coprocessor due to per-process memory constraints

OpenMP

OpenMP was used to implement shared-memory parallelism in epiSNP

- Facilitates more efficient load balancing
- Allows MPI rank 0 to read input data faster, using multiple threads
- Reduces MPI communication
- Reduces memory footprint by sharing data structures that would otherwise be replicated
- Required to fully utilize Intel Xeon Phi coprocessor due to per-process memory constraints

OpenMP

OpenMP was used to implement shared-memory parallelism in epiSNP

- Facilitates more efficient load balancing
- Allows MPI rank 0 to read input data faster, using multiple threads
- Reduces MPI communication
- Reduces memory footprint by sharing data structures that would otherwise be replicated
- Required to fully utilize Intel Xeon Phi coprocessor due to per-process memory constraints

OpenMP

OpenMP was used to implement shared-memory parallelism in epiSNP

- Facilitates more efficient load balancing
- Allows MPI rank 0 to read input data faster, using multiple threads
- Reduces MPI communication
- Reduces memory footprint by sharing data structures that would otherwise be replicated
- Required to fully utilize Intel Xeon Phi coprocessor due to per-process memory constraints

OpenMP

OpenMP was used to implement shared-memory parallelism in epiSNP

- Facilitates more efficient load balancing
- Allows MPI rank 0 to read input data faster, using multiple threads
- Reduces MPI communication
- Reduces memory footprint by sharing data structures that would otherwise be replicated
- Required to fully utilize Intel Xeon Phi coprocessor due to per-process memory constraints

OpenMP

OpenMP was used to implement shared-memory parallelism in epiSNP

- Facilitates more efficient load balancing
- Allows MPI rank 0 to read input data faster, using multiple threads
- Reduces MPI communication
- Reduces memory footprint by sharing data structures that would otherwise be replicated
- Required to fully utilize Intel Xeon Phi coprocessor due to per-process memory constraints

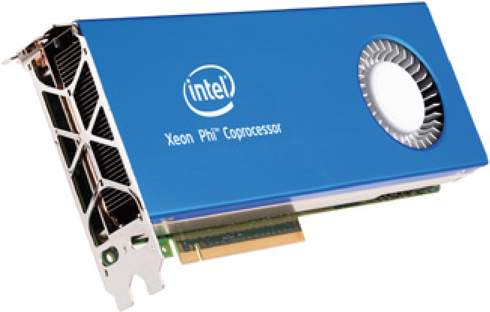


Figure: The Intel Xeon Phi coprocessor (aka MIC)

MIC
epiSNP on Accelerators

epiSNP was a good candidate for porting to an accelerator due to several characteristics:

- Abundant parallelism in computationally-intensive portions of the code
- Relatively small memory footprint in new hybrid MPI+OpenMP version
- Little data movement relative to computation

MIC
epiSNP on Accelerators

epiSNP was a good candidate for porting to an accelerator due to several characteristics:

- Abundant parallelism in computationally-intensive portions of the code
- Relatively small memory footprint in new hybrid MPI+OpenMP version
- Little data movement relative to computation

MIC
epiSNP on Accelerators

epiSNP was a good candidate for porting to an accelerator due to several characteristics:

- Abundant parallelism in computationally-intensive portions of the code
- Relatively small memory footprint in new hybrid MPI+OpenMP version
- Little data movement relative to computation

MIC
epiSNP on Accelerators

epiSNP was a good candidate for porting to an accelerator due to several characteristics:

- Abundant parallelism in computationally-intensive portions of the code
- Relatively small memory footprint in new hybrid MPI+OpenMP version
- Little data movement relative to computation

MIC
Specifications

The Intel Xeon Phi (MIC) "Knights Corner" generation features:

- Over 1 teraFLOPS double-precision peak performance
- Up to 3 times the performance per watt vs. Intel Xeon E5-2697v2
- 57 - 61 processor cores
- 1.1 - 1.238 GHz
- 4 threads per core
- 512-bit vector registers
- 6GB - 16GB memory

MIC
MIC vs. GPU

The Intel Xeon Phi coprocessor (MIC) has a couple advantages over NVIDIA GPUs as an accelerator target for epiSNP:

- Allows the use of the same programming language (Fortran)
- Optimized epiSNP could run as-is on the MIC with reasonable performance (approx. 1.5X faster than host Xeon CPUs)

MIC
MIC vs. GPU

The Intel Xeon Phi coprocessor (MIC) has a couple advantages over NVIDIA GPUs as an accelerator target for epiSNP:

- Allows the use of the same programming language (Fortran)
- Optimized epiSNP could run as-is on the MIC with reasonable performance (approx. 1.5X faster than host Xeon CPUs)

MIC
MIC vs. GPU

The Intel Xeon Phi coprocessor (MIC) has a couple advantages over NVIDIA GPUs as an accelerator target for epiSNP:

- Allows the use of the same programming language (Fortran)
- Optimized epiSNP could run as-is on the MIC with reasonable performance (approx. 1.5X faster than host Xeon CPUs)

MIC
Programming Models

An MPI application can be run on the MIC using one of three execution modes (*native*, *offload*, *symmetric*).

Symmetric mode (MPI processes run on both hosts and coprocessors) was chosen for epiSNP

- simplicity: no code modification required (initially)
- flexibility: arbitrary numbers of host processors and MIC coprocessors

MIC
Programming Models

An MPI application can be run on the MIC using one of three execution modes (*native*, *offload*, *symmetric*).

Symmetric mode (MPI processes run on both hosts and coprocessors) was chosen for epiSNP

- simplicity: no code modification required (initially)
- flexibility: arbitrary numbers of host processors and MIC coprocessors

MIC

Load Balancing Host and MIC

Problem: host processors and MIC coprocessors don't have the same performance characteristics.

- This leads to load imbalance when MPI processes execute on both the host and MIC.

Solution: Allow user to specify a *MIC Performance Factor* (F) so that additional comparisons would be assigned to the faster processes.

- Implemented as an environment variable (`EPISNP_MIC_PERF_FACTOR`)

MIC
Load Balancing Host and MIC

Problem: host processors and MIC coprocessors don't have the same performance characteristics.

- This leads to load imbalance when MPI processes execute on both the host and MIC.

Solution: Allow user to specify a *MIC Performance Factor* (F) so that additional comparisons would be assigned to the faster processes.

- Implemented as an environment variable (`EPISNP_MIC_PERF_FACTOR`)

Benchmarking

Benchmarking

Benchmark data set:

- 1,634 individuals (Angus sired cattle)
- 774,660 SNPs
- 45 traits (fatty acid content)

Since epiSNP scales linearly with the number of traits, a single trait (stearic acid content) was chosen for benchmarking.

Environment

Stampede

Source: Texas Advanced Computing Center

Environment

Compute node specs

- 2×8-core Intel Xeon E5-2680 2.7GHz CPUs
- 32 GB DDR3-1600MHz memory
- 1 or 2×61-core Xeon Phi SE10P 1.1GHz coprocessors

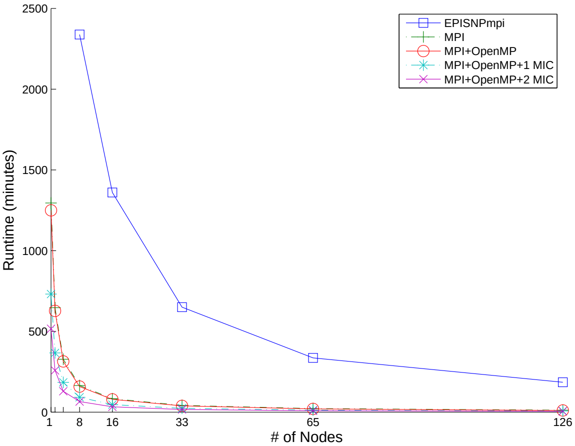
Software Environment

- Intel Fortran 14.0.1.106
- Intel MPI 4.1 Update 1[a]

[a] MVAPICH2 2.0b was used for original EPISNPmpi when run on 16 nodes

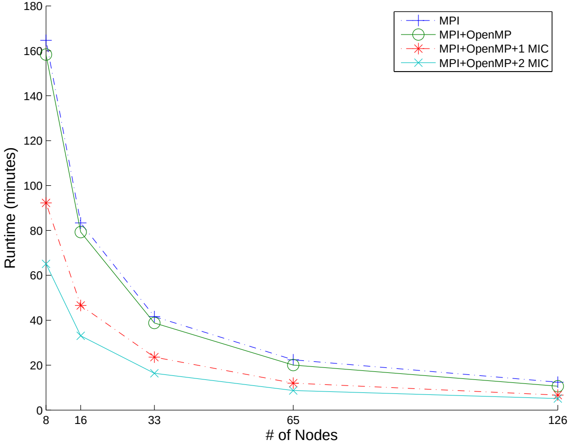
Results

Runtime



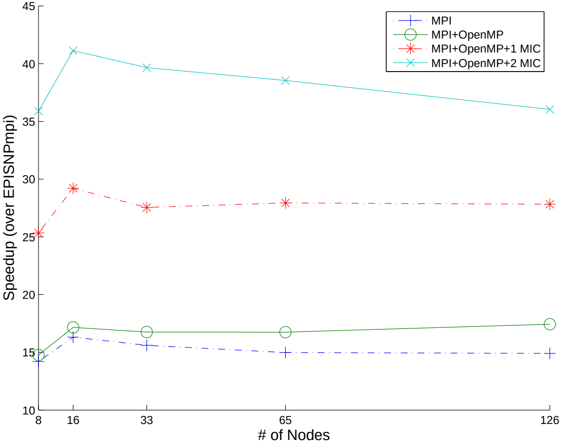
Results

Runtime (without original EPISNPmpi)

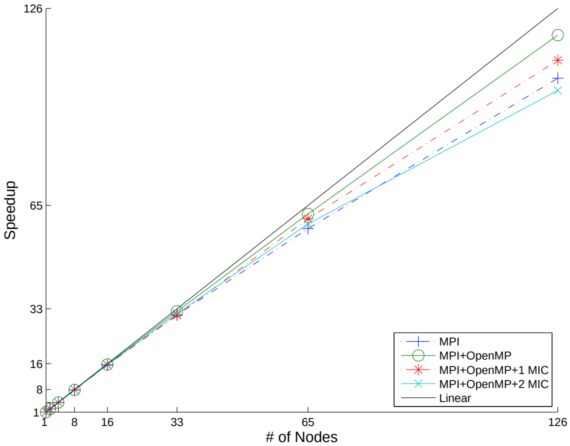


Results

Speedup (over EPISNPmpi)



Results
Speedup



Results
Node Hours

nodes	EPISNPmpi	MPI	MPI + OpenMP	MPI + OpenMP 1 MIC	MPI + OpenMP 2 MIC
1		21.58	20.82	12.19	8.61
2		21.55	20.90	12.21	8.61
4		21.84	20.91	12.30	8.62
8	311.78	21.96	21.11	12.30	8.69
16	362.68	22.22	21.14	12.42	8.82
33	357.46	22.91	21.33	12.97	9.01
65	363.65	24.27	21.71	13.01	9.44
126	388.50	26.06	22.28	13.97	10.77

Conclusions

To address performance challenges posed by larger data sets, we implemented the following improvements to epiSNP:

- serial optimizations
- improved data distribution for better load balancing and to allow execution using an arbitrary number of MPI processes
- shared-memory parallelism to reduce memory footprint and further improve load balancing
- port to the Intel Xeon Phi coprocessor

Results (average speedup vs. original EPISNPmpi):

- 15X (MPI)
- 17X (MPI+OpenMP)
- 28X (MPI+OpenMP with 1 Xeon Phi per node)
- 36X (MPI+OpenMP with 2 Xeon Phis per node)

Conclusions

To address performance challenges posed by larger data sets, we implemented the following improvements to epiSNP:

- serial optimizations
- improved data distribution for better load balancing and to allow execution using an arbitrary number of MPI processes
- shared-memory parallelism to reduce memory footprint and further improve load balancing
- port to the Intel Xeon Phi coprocessor

Results (average speedup vs. original EPISNPmpi):

- 15X (MPI)
- 17X (MPI+OpenMP)
- 28X (MPI+OpenMP with 1 Xeon Phi per node)
- 36X (MPI+OpenMP with 2 Xeon Phis per node)

Acknowledgment

Yang Da

Dossay Oryspayev

The research reported in this presentation is partially supported by the HPC@ISU equipment at Iowa State University, some of which has been purchased through funding provided by NSF under MRI grant number CNS 1229081 and CRI grant number 1205413.

This work was created using resources or cyberinfrastructure provided by the iPlant Collaborative. The iPlant Collaborative is funded by a grant from the National Science Foundation (#DBI-0735191).

The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported within this presentation.

Questions

Questions?