Everyday Multithreading

Parallel computing for genomic evaluations in *R*

C. Heuer, D. Hinrichs, G. Thaller

Institute of Animal Breeding and Husbandry, Kiel University

August 27, 2014

Introduction

Introduction

- High Dimensional livestock data sets
- New computational challenges
- Paradigm shift in breeding programs **and** computing
- From sparse to dense MME (or mixtures)
- High storage, memory and computing demands

Introduction

- High Dimensional livestock data sets
- New computational challenges
- Paradigm shift in breeding programs **and** computing
- From sparse to dense MME (or mixtures)
- High storage, memory and computing demands
- **Solution**: Making use of available hardware resources by parallel computing

What parallel computing is:

What parallel computing is:

Splitting up a big problem into chunks that are simultaneously being solved by several processing units

What parallel computing is:

Splitting up a big problem into chunks that are simultaneously being solved by several processing units

$$\begin{bmatrix} -0.09 \\ -0.65 \\ -0.08 \\ -0.95 \\ -0.02 \\ -0.59 \\ -0.55 \\ 0.37 \\ -0.05 \\ -0.16 \\ 0.59 \\ -0.55 \end{bmatrix} + \begin{bmatrix} -0.92 \\ 0.41 \\ -0.76 \\ -1.61 \\ 1.09 \\ -0.33 \\ 0.83 \\ 3.21 \\ 1.29 \\ -0.82 \\ -1.23 \\ -1.08 \end{bmatrix} = \begin{bmatrix} -1.01 \\ -0.24 \\ -0.83 \\ -2.56 \\ 1.07 \\ -0.92 \\ 0.27 \\ 3.58 \\ 1.23 \\ -0.98 \\ -0.64 \\ -1.64 \end{bmatrix}$$

What parallel computing is:

Splitting up a big problem into chunks that are simultaneously being solved by several processing units

$$\begin{bmatrix} -0.09 \\ -0.65 \\ -0.08 \\ \hline -0.95 \\ -0.02 \\ -0.59 \\ \hline -0.55 \\ 0.37 \\ -0.05 \\ \hline -0.16 \\ 0.59 \\ -0.55 \end{bmatrix} + \begin{bmatrix} -0.92 \\ 0.41 \\ -0.76 \\ \hline -1.61 \\ 1.09 \\ -0.33 \\ \hline 0.83 \\ 3.21 \\ 1.29 \\ \hline -0.82 \\ -1.23 \\ -1.08 \end{bmatrix} = \begin{bmatrix} -1.01 \\ -0.24 \\ -0.83 \\ \hline -2.56 \\ 1.07 \\ -0.92 \\ \hline 0.27 \\ 3.58 \\ 1.23 \\ \hline -0.98 \\ -0.64 \\ -1.64 \end{bmatrix}$$

Parallel Computing Paradigms

Parallel Computing Paradigms

1. Most importantly: Single Core parallelism → **Vectorization**
2. Shared Memory multi-threading → *OpenMP*
3. Distributed Memory multi-processing → *MPI*
4. GPU-programming → *CUDA*, *OpenCL*, *Intel Xeon-Phi*

Parallel Computing Paradigms

1. Most importantly: Single Core parallelism → **Vectorization**
2. Shared Memory multi-threading → *OpenMP*
3. Distributed Memory multi-processing → *MPI*
4. GPU-programming → *CUDA, OpenCL, Intel Xeon-Phi*

Efficiency/Scaling:

- Depends on size of the problem: Overhead
- The less efficient a single-threaded application, the better the scaling

Parallel Computing Paradigms

1. Most importantly: Single Core parallelism → **Vectorization**
2. Shared Memory multi-threading → *OpenMP*
3. Distributed Memory multi-processing → *MPI*
4. GPU-programming → *CUDA, OpenCL, Intel Xeon-Phi*

Efficiency/Scaling:

- Depends on size of the problem: Overhead
- The less efficient a single-threaded application, the better the scaling
- In general: First single-threaded optimization then parallelization

R-package *cpgen*

R-package *cpgen*

Advantages of R:

1. Very flexible open source interpreter language
2. Easy to use, available on all platforms and widely spread

Drawbacks of R:

R-package *cpgen*

Advantages of R:

1. Very flexible open source interpreter language
2. Easy to use, available on all platforms and widely spread

Drawbacks of R:

1. Not designed for big data problems
2. Needs a lot of effort to extend R
3. Strictly single-threaded

R-package *cpgen*

Advantages of R:

1. Very flexible open source interpreter language
2. Easy to use, available on all platforms and widely spread

Drawbacks of R:

1. Not designed for big data problems
2. Needs a lot of effort to extend R
3. Strictly single-threaded

But: Can be extended and multi-threaded through C/C++/Fortran shared libraries

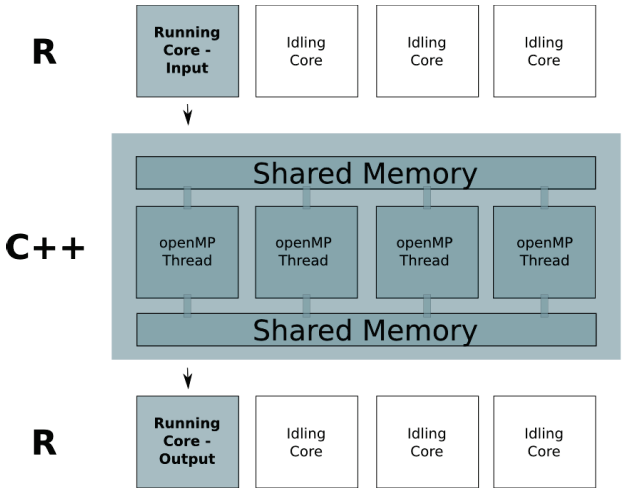
General Implementation

General Implementation

1. *R* as the basic environment for data preparation and supply
2. Linking C++ to R: *Rcpp* (Eddelbuettel and Francois, 2011)
3. Linear Algebra: *Eigen* $\rightarrow$ Vectorization! (Guennebaud et al., 2010)
4. Eigen + Rcpp + Sparse-Matrix support: *RcppEigen* (Bates and Eddelbuettel, 2013)
5. Parallelization: *OpenMP*

Parallel Computing in *cpgen*

Parallel Computing in *cpgen*



Functionality

Functionality

1. Single-Site Gibbs Sampler for Mixed Models with arbitrary number of random effects (sparse or dense design matrices)
2. Genomic Prediction Module: Different Methods, Cross Validation
3. GWAS Module: *EMMAX* - highly efficient and very flexible single marker GWAS
4. Tools: Genomic additive and dominance relationships, Crossproducts, Covariance Matrices, ...

Gibbs Sampler

Runs models of the following form:

$$\mathbf{y} = \mathbf{Xb} + \mathbf{Z}_1\mathbf{u}_1 + \mathbf{Z}_2\mathbf{u}_2 + \mathbf{Z}_3\mathbf{u}_3 + ... + \mathbf{Z}_n\mathbf{u}_n + \mathbf{e}$$

- For all u_k: $MVN(\mathbf{0}, \mathbf{I}\sigma^2_{u_k})$
- If u_k is assumed to follow some $MVN(\mathbf{0}, \mathbf{G}_k\sigma^2_k)$:

Design matrix must be constructed as: $\mathbf{Z}_k = \mathbf{Z}_k\mathbf{G}^{1/2}$, yielding independent effect in $\mathbf{u}_k$ (Waldmann et al., 2008).

Genomic BLUP

GBLUP can be accomplished very efficiently (Kang et al., 2008):

$$\mathbf{y} = \mathbf{Xb} + \mathbf{a} + \mathbf{e} \quad \text{with: } \mathbf{a} \sim MVN(\mathbf{0}, \mathbf{G}\sigma_a^2)$$

By finding the decomposition: $\mathbf{G} = \mathbf{UDU}'$ and premultiplying the model equation by $\mathbf{U}'$ we get:

$$\mathbf{U}'\mathbf{y} = \mathbf{U}'\mathbf{Xb} + \mathbf{U}'\mathbf{a} + \mathbf{U}'\mathbf{e}$$

with:

$$\begin{aligned} Var(\mathbf{U}'\mathbf{y}) &= \mathbf{U}'\mathbf{GU}\sigma_a^2 + \mathbf{U}'\mathbf{U}\sigma_e^2 \\ &= \mathbf{U}'\mathbf{UDU}'\mathbf{U}\sigma_a^2 + \mathbf{I}\sigma_e^2 \\ &= \mathbf{D}\sigma_a^2 + \mathbf{I}\sigma_e^2 \end{aligned}$$

GWAS

- Single marker regression
- Controlling for polygenic background effect through [assumed] covariance structure $\mathbf{V}$ of $\mathbf{y}$ - EMMAX (Kang et al., 2010)
- Obtaining General Least Squares estimates for marker effects:

$$\hat{\beta} = (\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}\mathbf{y}$$

This is equivalent to:

$$\hat{\beta} = (\mathbf{X}^{*\prime}\mathbf{X}^{*})^{-1}\mathbf{X}^{*\prime}\mathbf{y}^{*}, \quad \text{with } \mathbf{X}^{*} = \mathbf{V}^{-1/2}\mathbf{X}, \quad \mathbf{y}^{*} = \mathbf{V}^{-1/2}\mathbf{y}$$

Parallel Computing in *cpgen*

What is parallelized:

1. All crossproduct-like computations
2. Sampling of random effects in Gibbs Sampler (dot product, vector-vector subtraction) $\rightarrow$ Fernando et al., 2014
3. Cross Validation for genomic prediction
4. GWAS

Parallel Computing in *cpgen*

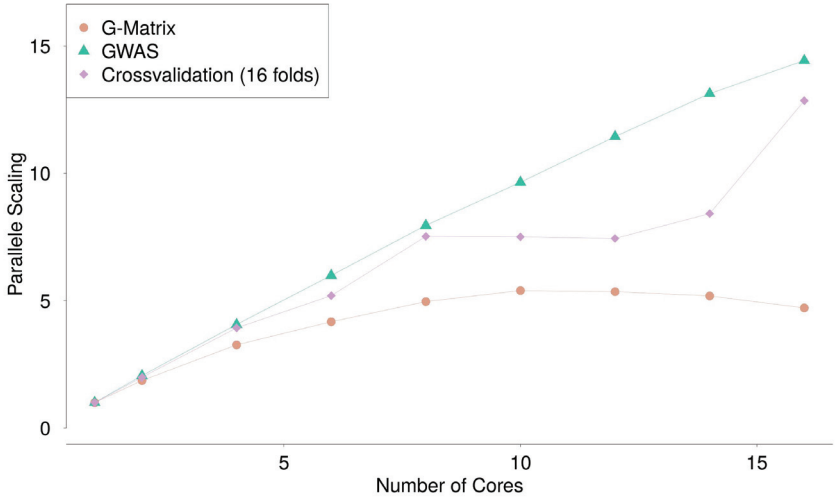
What is parallelized:

1. All crossproduct-like computations
2. Sampling of random effects in Gibbs Sampler (dot product, vector-vector subtraction) $\rightarrow$ Fernando et al., 2014
3. Cross Validation for genomic prediction
4. GWAS

Number of threads being used can be controlled during runtime

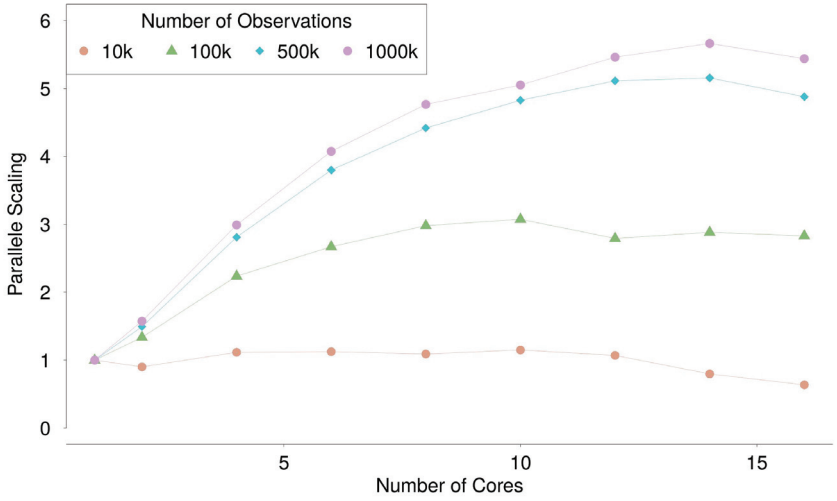
2,000 Observations, 50,000 Markers

2,000 Observations, 50,000 Markers



Gibbs Sampler (BRR) - 10,000 Markers

Gibbs Sampler (BRR) - 10,000 Markers



Conclusions

Conclusions

- Shared Memory multithreading can decrease computation time significantly
- Bridges the gap between single threaded applications and heavily parallelized standalone programs for HPC Clusters
- We can make use of the computational power present in workstation PCs $\rightarrow$ *Everyday Multithreading*

Conclusions

- Shared Memory multithreading can decrease computation time significantly
- Bridges the gap between single threaded applications and heavily parallelized standalone programs for HPC Clusters
- We can make use of the computational power present in workstation PCs $\rightarrow$ *Everyday Multithreading*
- **But**: Size of solvable problem is limited by available memory
- With 1 TB of memory, the package could fit a BRR model with 3 million observations and 40k markers

Thank you for the attention

Github `https://github.com/cheuerde/cpgen`
R-Forge `https://r-forge.r-project.org/R/?group_id=1687`

Absolute Timings

10 Cores, 30,000 markers

- BRR: 1 million observations, 30k iterations $\sim$ 100 hours
- GWAS: 10k observations $\sim$ 7 minutes
- G-Matrix: 10k observations $\sim$ 2 minutes