Utility of Graphics Processing Units for dense matrix calculations in computing and inverting genomic relationship matrices

Karin Meyer and Bruce Tier
Animal Genetics and Breeding Unit, University of New England, Armidale, Australia

AAABG 2013

Computing in the genomics age

- Requirements for mixed model analyses have changed!
  - Pre-genomics: mixed model equations sparse (few non-zero elements)
    - inverse of pedigree based relationship matrix (NRM)
    - concentrated on exploiting sparsity in computations
  - Now: Genomic relationship matrix (GRM)
    - dense (large proportion of elements non-zero)
    - require manipulation of large, dense matrices to
      - compute the GRM
      - invert the GRM
      - predict breeding values in single-step approach
    - computationally demanding
    - exploit new hardware and software libraries
      - computing in parallel

Objectives

- A first look to examine scope for
  - speeding up computations to calculate & invert GRM
  - using parallel computations
  - exploiting a Graphics Processing Unit (GPU)

What is a GPU?

- Graphical Processing Unit
  - initially: co-processor for computer gaming
    - graphics card
    - fast recalculation of pixel values
    - remember: i387 math processor for i386
  - now: GP-GPU adapted to General Purpose computing
    - hundreds to thousands of cores
    - capable of double precision calculations
    - turns desktop PC into personal super-computer
    - but: limited memory (currently 6GB max.)
  - specialised interface for application programming
    - CUDA (proprietary of NVIDIA) → Fortran interface
    - OPENCL (C++ like)

"An excursion into the land of computer gaming, modern compilers and software libraries"
GPUs for GRMs | Introduction

Tesla K20X
- 2688 cores
- 6 GB RAM
- Peak: 3.95/1.31 Tflops (single/double precision)

GPUs for GRMs | Calculating the GRM

Memory needed

<table>
<thead>
<tr>
<th>n</th>
<th>G</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>s</td>
<td>100K</td>
</tr>
<tr>
<td>5000</td>
<td>0.1</td>
<td>1.9</td>
</tr>
<tr>
<td>10000</td>
<td>0.4</td>
<td>3.7</td>
</tr>
<tr>
<td>20000</td>
<td>1.5</td>
<td>7.5</td>
</tr>
<tr>
<td>30000</td>
<td>3.4</td>
<td>11.2</td>
</tr>
<tr>
<td>50000</td>
<td>9.3</td>
<td>18.6</td>
</tr>
</tbody>
</table>

→ break calculations into blocks
→ use out-of-core storage
→ parallel computing literature

GPUs for GRMs | Calculating the GRM

Blockwise multiplication

- All-in-one
  \[ \begin{bmatrix} Z \\ Z' \end{bmatrix} = \begin{bmatrix} G \end{bmatrix} \]

- Column-wise division
  \[ \begin{bmatrix} Z_1 & \cdots & Z_4 \end{bmatrix} \begin{bmatrix} Z'_1 \\ \vdots \\ Z'_4 \end{bmatrix} = \begin{bmatrix} Z_1Z'_1 \\ \vdots \\ Z_4Z'_4 \end{bmatrix} \]

- G: symmetric \( sn(n + 1)/2 \) flops

GPUs for GRMs | Calculating the GRM

Blockwise multiplication - cont’

- Row-wise division
  \[ G_{ij} = Z_iZ'_j. \]

- Row- and columnwise division
  \[ G_{ij} = \sum_k Z_{ik}Z'_{jk}. \]
**Blockwise inversion**

\[
G = \begin{pmatrix}
G_{PP} & G_{PC} & G_{PT} \\
G_{CP} & G_{CC} & G_{CT} \\
G_{TP} & G_{TC} & G_{TT}
\end{pmatrix}
\]

1. Subdivide matrix into blocks
   - C current → choosen size
   - P previous
   - T trailing
2. Factor & invert chol(\(G_{CC}\))
3. Adjust \(G_{PP} & G_{PC}\)
4. ‘Absorb’ into \(G_{TT} & G_{CT}\)
5. Calculate inverse
6. Redefine C, P & T; repeat
   → break matrix products into blocks as needed

**Gauss-Jordan Elimination**

\[
\begin{align*}
G_{CC} & := \text{chol}(G_{CC}) & \text{POTRF} \\
G_{CC} & := G_{CC}^{-1} & \text{TRTRI} \\
G_{PC} & := G_{PC}G_{CC} & \text{TRMM} \\
G_{PP} & := G_{PP} + G_{PC}G_{PC}^{-1} & \text{SYRK} \\
G_{CT} & := G_{CC}^{-1}G_{CT} & \text{TRMM} \\
G_{TT} & := G_{TT} - G_{CT}G_{CT}^{-1} & \text{SYRK} \\
G_{PT} & := G_{PT} - G_{PC}G_{TC} & \text{GEMM} \\
G_{CT} & := -(G_{CC}G_{CT}) & \text{TRMM} \\
G_{PC} & := G_{PC}G_{CC} & \text{TRMM} \\
G_{CC} & := G_{CC}G_{CC} & \text{LAUUM}
\end{align*}
\]

**Software**

- Linux (CentOS 6)
- gfortran, gcc 4.4.7; Intel composer XE 13.0.1
- Intel MKL library 11.0
  - BLAS: SYRK, SGEMM
  - LAPACK: SPOTRF, SPOTRI
- CUDA 5.0
  - CUBLAS: CUBLAS..SYRK, CUBLAS..SGEMM
- CULA dense R16a
  - CULA..SPOTRF, CULA..SPOTRI, CULA..STRTRI
- MAGMA 1.3
  - MAGMAF..SLAUM

**Hardware**

- ‘Standard’ multi-core CPU
  - Intel i7-3930K
  - 3.2 GHz, 12M cache
  - 64GB RAM
  - 6 cores

- GPU
  - Nvidia GTX 580
  - 512 cores
  - 3GB RAM
  - capable of double precision calculations
    \(\leftarrow\) high end gaming card, not GP-GPU

**Time to ‘make’ GRM**

\[
\begin{array}{c|c|c}
\text{System time (min)} & \text{No. animals} \\
\hline
10 & 0 \\
5 & 20000 \\
1 & 40000 \\
0.1 & 60000 \\
0.01 & 80000 \\
\end{array}
\]

*Single precision calculations, \(s = 500000\)
**Time to invert GRM**

![Graph showing system time (min) vs. number of animals for CPU1 and GPU compared to CPU6.](image)

†Single precision calculations

![Graph showing system time (min) vs. number of animals for CPU1 and GPU compared to CPU6.](image)

‡Double precision calculations

### Conclusions

- **Build & invert GRM**
  - dense matrix multiplications
  - highly parallelisable

- **BLAS and LAPACK library routines awesome**
  - highly optimised
  - exploit memory architecture of modern processors
  - multi-threaded versions

- **GPU: powerful new hardware for parallel computing**
  - thousands of processors
  - can use multiple GPUs simultaneously
  - BLAS/LAPACK etc. libraries available
  - but: limited memory → blocked algorithms
  - but: specialised interface, not much Fortran support yet