Solving large models with MiX99 and parallel processing

MiX99 course: test-day models and single step genomic prediction
COURSE DAY 2, April 11th, 2025

Matti Taskinen
Natural Resources Institute Finland (Luke)

April 11, 2025

Luke
NATURAL RESOURCES INSTITUTE FINLAND

Content

- Solving large models.
- Using parallel MiX99 solver `mix99p` (instead of `mix99s`).
  - Concentrating on differences.
- Traditional models, no genomics involved.
- Monitoring parallel performance.

What is a large model?

- Large number of traits: $\gg$20.
- Large number of animals: $\gg$10 million
- Large number of records: $\gg$10 million.
- Large number of equations/effects/unknowns: $\gg$100 million.
- Can take days to solve without (and with) parallel computing.

Parallel operations of `mix99s` and `mix99p`

- MiX99 has two solvers:
  - "Serial" solver `mix99s`.
  - "Parallel" solver `mix99p`.

Parallel operations of `mix99s` and `mix99p`

- MiX99 has two solvers:
  - "Serial" solver `mix99s`.
  - "Parallel" solver `mix99p`.
- Serial solver `mix99s`:
  - Single running process.
  - Can utilize parallel computing using **multithreading** (`mp` subdir).

Parallel operations of `mix99s` and `mix99p`

- MiX99 has two solvers:
  - "Serial" solver `mix99s`.
  - "Parallel" solver `mix99p`.
- Serial solver `mix99s`:
  - Single running process.
  - Can utilize parallel computing using **multithreading** (`mp` subdir).
- Parallel solver `mix99p`:
  - Utilizes parallel computing with **Message Passing Interface (MPI)** communication.
  - Separate parallel **processes** that do not (directly) share memory.
  - Can also utilize **hybrid parallelism**, i.e. both multithreading and **multiprocessing** (`mp` subdir).

How to operate `mix99s` and `mix99p`

- Serial solver `mix99s`:
  - Run MiX99 preprocessor: `mix99i`
  - Run solver:
    `mix99s options < option_file`

How to operate `mix99s` and `mix99p`

- Serial solver `mix99s`:
  - Run MiX99 preprocessor: `mix99i`
  - Run solver:

    ```
    mix99s options < option_file
    ```

- Parallel solver `mix99p` needs additional pre-processing and starting of the actual parallel MPI processes:
  - Run MiX99 preprocessor: `mix99i`
  - Additional preprocessing: `imake99`
  - Start MPI processes:

    ```
    mpiexec -np number_or_processes mix99p options < option_file
    ```

Difficulties with MPI programs

Parallel solver `mix99p` is a MPI program:

- Depends on **compile-time** version of the MPI library:
  - **Run-time environment** MPI library version needs to match.

Difficulties with MPI programs

Parallel solver `mix99p` is a MPI program:

- Depends on **compile-time** version of the MPI library:
  - **Run-time environment** MPI library version needs to match.
- If more than one MPI libraries installed on run-time environment, correct `mpiexec` version needs to be specifed with, for example, **absolute paths**:
  `/usr/lib64/openmpi/bin/mpiexec -np number_or_processes ...`

Difficulties with MPI programs

Parallel solver `mix99p` is a MPI program:

- Depends on **compile-time** version of the MPI library:
  - **Run-time environment** MPI library version needs to match.
- If more than one MPI libraries installed on run-time environment, correct `mpiexec` version needs to be specifed with, for example, **absolute paths**:
  `/usr/lib64/openmpi/bin/mpiexec -np number_or_processes ...`
- MiX99 MPI programs `mix99p` and `apax99p` are **dynamically linked**:
  - Most external libraries **statically** included within executables.
  - But **system** and MPI libraries dynamically linked on run-time.
  - This may cause more compatibility problems than with other, fully statically linked, MiX99 programs.

Features and options available only on `mix99p` solver

- Parallel `mix99p` job can be distributed and executed on multiple separate computers (or "**nodes**"):
  - This is, however, nowadays seldomly used.
  - Instead, typically run on a single node containing one or more CPUs each having multiple **physical** or **logical** computational **cores**.

Features and options available only on `mix99p` solver

- Parallel `mix99p` job can be distributed and executed on multiple separate computers (or "**nodes**"):
  - This is, however, nowadays seldomly used.
  - Instead, typically run on a single node containing one or more CPUs each having multiple **physical** or **logical** computational **cores**.

- Additional memory model "eXtra large" (or "X") and corresponding **command line option** `"-x"`:
  - Can speed up communication between the MPI processes considerable but also uses more memory.

Features and options available only on `mix99p` solver

- Parallel `mix99p` job can be distributed and executed on multiple separate computers (or "**nodes**"):
  - This is, however, nowadays seldomly used.
  - Instead, typically run on a single node containing one or more CPUs each having multiple **physical** or **logical** computational **cores**.

- Additional memory model "eXtra large" (or "X") and corresponding **command line option** `"-x"`:
  - Can speed up communication between the MPI processes considerable but also uses more memory.

- Command line option `"-a"` to use "Allreduce" MPI communication:
  - Can speed up communication between the MPI processes.

Differences between `mix99s` and `mix99p`

- Parallel computing in `mix99s` is more targeted for models with **dense matrix** information, such as **genomic models**.

Differences between `mix99s` and `mix99p`

- Parallel computing in `mix99s` is more targeted for models with **dense matrix** information, such as **genomic models**.
- Parallel computing in `mix99p` is more targeted for models with **sparse matrix** information, such as traditional **animal models**.

Differences between `mix99s` and `mix99p`

- Parallel computing in `mix99s` is more targeted for models with **dense matrix** information, such as **genomic models**.
- Parallel computing in `mix99p` is more targeted for models with **sparse matrix** information, such as traditional **animal models**.
- Models with **non-linear** or **categorical** traits are not yet possible to analyze with `mix99p`.

Dense vs. sparse matrix operations

- Back in the old days dense matrix operations were mostly limited by the speed of **floating point calculations**:
  - Multiplication and addition of floating point numbers were slow.

Dense vs. sparse matrix operations

- Back in the old days dense matrix operations were mostly limited by the speed of **floating point calculations**:
  - Multiplication and addition of floating point numbers were slow.
- Sparse matrix operations, on the other hand, are **memory bandwidth limited**. For example:

Dense vs. sparse matrix operations

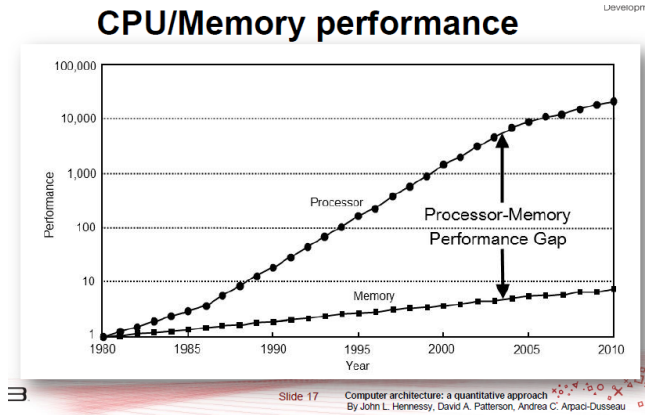
- Back in the old days dense matrix operations were mostly limited by the speed of **floating point calculations**:
  - Multiplication and addition of floating point numbers were slow.
- Sparse matrix operations, on the other hand, are **memory bandwidth limited**. For example:
  - Solving traditional animal model involves multiplying with inverse of **pedigree relationship matrix** $\mathbf{A}^{-1}$.
  - Matrix $\mathbf{A}^{-1}$ is, however, very sparse and can be expressed using only 3 indeces: id, sire id, and dam id of individuals.

Dense vs. sparse matrix operations

- Back in the old days dense matrix operations were mostly limited by the speed of **floating point calculations**:
  - Multiplication and addition of floating point numbers were slow.
- Sparse matrix operations, on the other hand, are **memory bandwidth limited**. For example:
  - Solving traditional animal model involves multiplying with inverse of **pedigree relationship matrix** $\mathbf{A}^{-1}$.
  - Matrix $\mathbf{A}^{-1}$ is, however, very sparse and can be expressed using only 3 indeces: id, sire id, and dam id of individuals.
  - So, matrix operations involving $\mathbf{A}^{-1}$ are **linear** (instead of quadratic) with respect to number of individuals.
  - Relatively more memory references than floating point calculations $\Rightarrow$ limited by memory access speed and **memory access patterns**.

CPU speed vs. memory speed

- New CPUs are getting faster generation by generation.
- Memory speed, however, has not increased at the same rate.
- Floating point calculations are so fast that memory access speed limits the computing even on dense matrices.

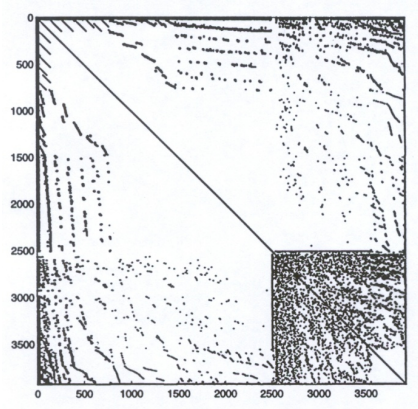


How to speed up sparse matrix operations

- Parallization does not help much with sparse matrices.
- Almost the only way to speed up sparse matrix operations is to:
  - minimize memory accesses
  - use **cache-friendly** memory accessing

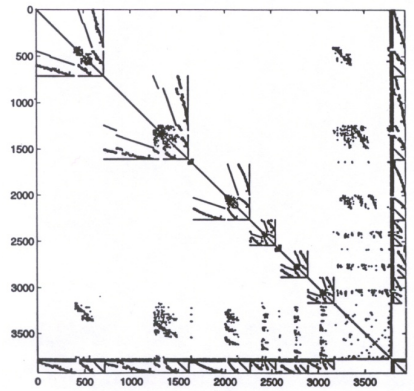
How to speed up sparse matrix operations

- Parallization does not help much with sparse matrices.
- Almost the only way to speed up sparse matrix operations is to:
  - minimize memory accesses
  - use **cache-friendly** memory accessing
- **Reordering** of the sparse matrix:
  - Instead of randomly scattered locations,



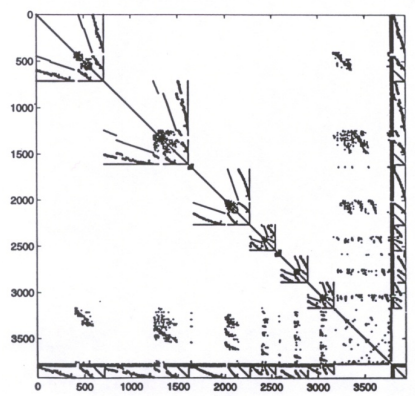
How to speed up sparse matrix operations

- Parallization does not help much with sparse matrices.
- Almost the only way to speed up sparse matrix operations is to:
  - minimize memory accesses
  - use **cache-friendly** memory accessing
- **Reordering** of the sparse matrix:
  - Instead of randomly scattered locations,
  - non-zero elements moved close to **diagonal**,



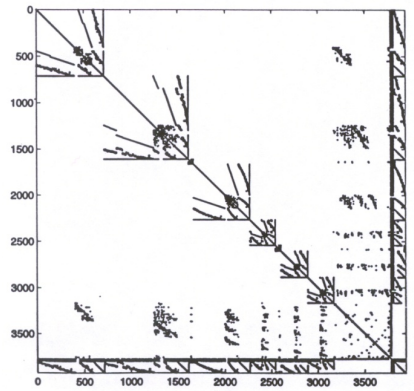
How to speed up sparse matrix operations

- Parallization does not help much with sparse matrices.
- Almost the only way to speed up sparse matrix operations is to:
  - minimize memory accesses
  - use **cache-friendly** memory accessing
- **Reordering** of the sparse matrix:
  - Instead of randomly scattered locations,
  - non-zero elements moved close to **diagonal**,
  - or collected along separate (more) dense columns and rows.



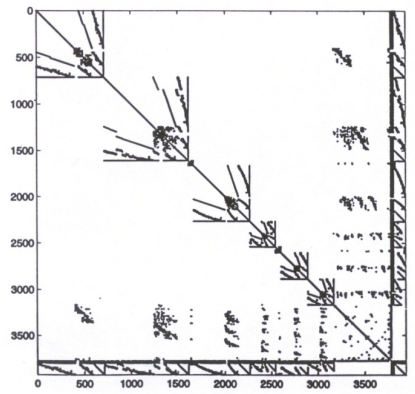
How to speed up sparse matrix operations

- Parallization does not help much with sparse matrices.
- Almost the only way to speed up sparse matrix operations is to:
  - minimize memory accesses
  - use **cache-friendly** memory accessing
- **Reordering** of the sparse matrix:
  - Instead of randomly scattered locations,
  - non-zero elements moved close to **diagonal**,
  - or collected along separate (more) dense columns and rows.
  - Allows more **sequential** memory accesses.



MiX99 equation family blocks: ordering of relations

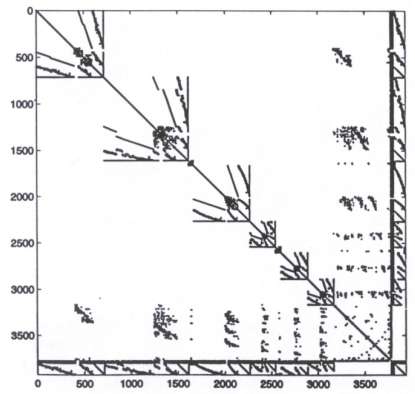
- MiX99 allows ordering of equations in so called **equation family blocks**:
  - **Block code** column in data and pedigree.
  - Both files sorted by user.



Luke
NATURAL RESOURCES INSTITUTE FINLAND

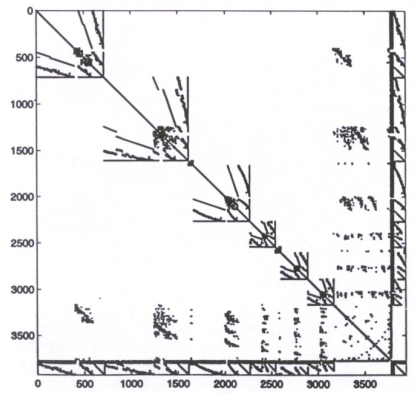
MiX99 equation family blocks: ordering of relations

- MiX99 allows ordering of equations in so called **equation family blocks**:
  - **Block code** column in data and pedigree.
  - Both files sorted by user.
- An equation family block comprises of closely related equations in the model:
  - In dairy cattle effects of the same herd.



MiX99 equation family blocks: ordering of relations

- MiX99 allows ordering of equations in so called **equation family blocks**:
  - **Block code** column in data and pedigree.
  - Both files sorted by user.
- An equation family block comprises of closely related equations in the model:
  - In dairy cattle effects of the same herd.
- Equations associated to all or most of the blocks combined to **common blocks**:
  - Common blocks are the last blocks.
  - Cache-friendly dense columns and rows.



MiX99 equation family blocks: workload balancing

MiX99 uses the equation family blocks also for **balancing the workload** between the parallel `mix99p` processes:

MiX99 equation family blocks: workload balancing

MiX99 uses the equation family blocks also for **balancing the workload** between the parallel `mix99p` processes:

- `mix99p` uses **data-parallelism**: data is split and distributed across the parallel MPI processes.

MiX99 equation family blocks: workload balancing

MiX99 uses the equation family blocks also for **balancing the workload** between the parallel `mix99p` processes:

- `mix99p` uses **data-parallelism**: data is split and distributed across the parallel MPI processes.
- Non-common equation family blocks are **partitioned** (in order) to MPI processes so that the processes have equal amount of work.

MiX99 equation family blocks: workload balancing

MiX99 uses the equation family blocks also for **balancing the workload** between the parallel `mix99p` processes:

- `mix99p` uses **data-parallelism**: data is split and distributed across the parallel MPI processes.
- Non-common equation family blocks are **partitioned** (in order) to MPI processes so that the processes have equal amount of work.
- MiX99 has two **workload division** methods: number of records (default) and number of equations.

MiX99 equation family blocks: workload balancing

MiX99 uses the equation family blocks also for **balancing the workload** between the parallel `mix99p` processes:

- `mix99p` uses **data-parallelism**: data is split and distributed across the parallel MPI processes.
- Non-common equation family blocks are **partitioned** (in order) to MPI processes so that the processes have equal amount of work.
- MiX99 has two **workload division** methods: number of records (default) and number of equations.
- Number of common blocks must be given by the user. Alternatively, the block code of the first common block can be specified:

```
PARALLEL <N processors> <N common blocks> [<work division> <buffer size>]
PARALLEL <N processors> <first common block> FIRST [...]
```

Monitoring block-to-block communication

- Parallel `mix99p` processes operate on their data part and **communicate** with each others.

```
Communication in parallel computing:
  91.33% using the linked list
   7.98% through the common blocks
   0.69% through the across blocks

   5.78% of equations need communication

Total communicated equations:      21999258
  - linked list              :      20090904
  - common blocks            :       1756440
  - across blocks            :        151914

Total number of equations    :     380436594
  - within blocks            :     380284680
  - across blocks            :        151914
```

Monitoring block-to-block communication

- Parallel `mix99p` processes operate on their data part and **communicate** with each others.
- Performance depends partially on the amount of this communication.

```
Communication in parallel computing:
  91.33% using the linked list
   7.98% through the common blocks
   0.69% through the across blocks

   5.78% of equations need communication

Total communicated equations:      21999258
  - linked list              :      20090904
  - common blocks            :       1756440
  - across blocks            :        151914

Total number of equations    :     380436594
  - within blocks            :     380284680
  - across blocks            :        151914
```

Monitoring block-to-block communication

- Parallel `mix99p` processes operate on their data part and **communicate** with each others.
- Performance depends partially on the amount of this communication.
- Communication can be inspected from output of `imake99`:
  - How much communication is sparse ("`linked list`") relative to common and across blocks.
  - How large proportion of equations needs communication.

```
Communication in parallel computing:
  91.33% using the linked list
   7.98% through the common blocks
   0.69% through the across blocks

   5.78% of equations need communication

Total communicated equations:      21999258
  - linked list              :      20090904
  - common blocks            :       1756440
  - across blocks            :        151914

Total number of equations    :     380436594
  - within blocks            :     380284680
  - across blocks            :        151914
```

Visualizing block-to-block dependencies

- Communication between blocks can be visualized using:

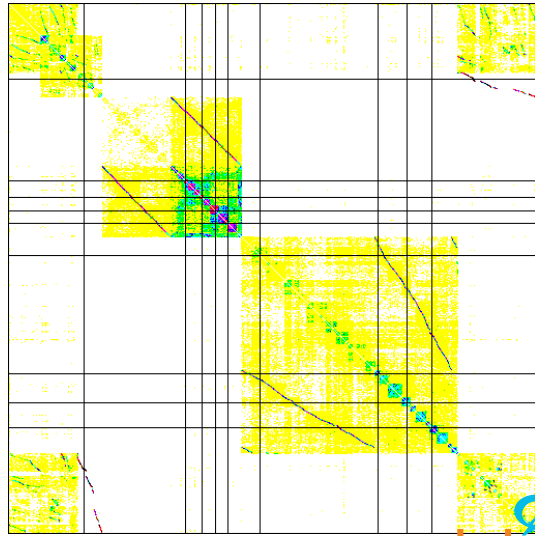
```
block_information -p
```

Visualizing block-to-block dependencies

- Communication between blocks can be visualized using:

  ```
  block_information -p
  ```

- Creates file `Bmatrix.png` showing **heat map** of communication:
  - Largest communications are dark.
  - Each 7 colors are $\frac{1}{7}$ of total amount.

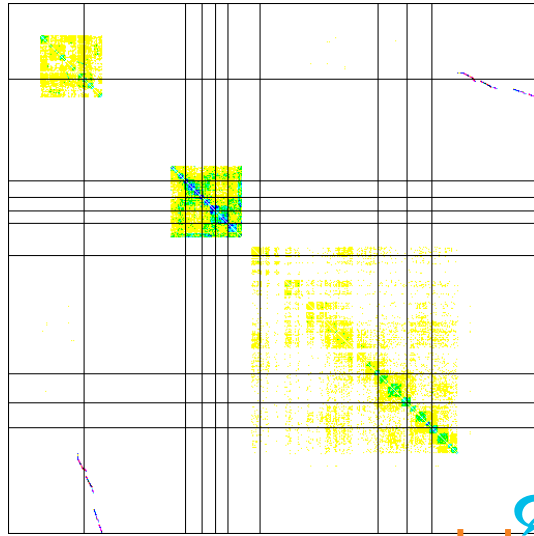


Visualizing block-to-block dependencies

- Communication between blocks can be visualized using:

  `block_information -p`

- Creates file `Bmatrix.png` showing **heat map** of communication:
  - Largest communications are dark.
  - Each 7 colors are $\frac{1}{7}$ of total amount.
- With option `-separate` creates separate files:
  - For data: `BM_data.png`

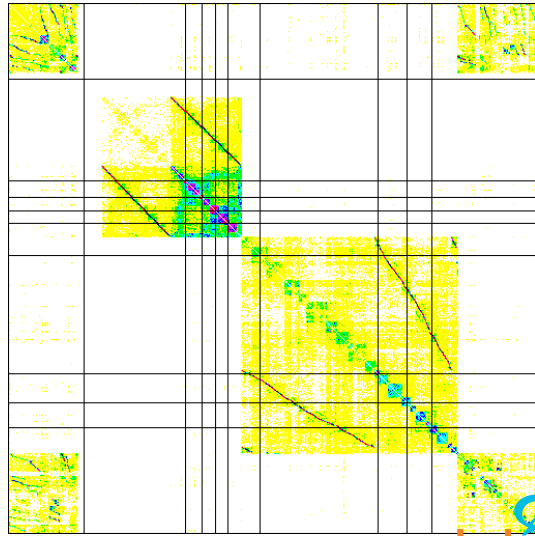


Visualizing block-to-block dependencies

- Communication between blocks can be visualized using:

  `block_information -p`

- Creates file `Bmatrix.png` showing **heat map** of communication:
  - Largest communications are dark.
  - Each 7 colors are $\frac{1}{7}$ of total amount.
- With option `-separate` creates separate files:
  - For data: `BM_data.png`
  - For variance structures: `BM_VStruct.png`



Monitoring parallel performance: `OK_mix99p` file

After successful execution of `mix99p` file `OK_mix99p` is created containing:

- Convergence status.
- Memory usage.
- Elapsed times of program segments.
- Parallel efficiency.

```
Successful execution of mix99p                    Time: 18:07:10.6  05.04.2025
Convergence information:
  Stopping criterion CA < 0.1E-7 was _NOT_ achieved in 100 iterations.
Peak Virtual Memory Usage [0]: 2208512 KB
Peak Virtual Memory Usage [1]: 1941712 KB
Peak Virtual Memory Usage [2]: 1933376 KB
Peak Virtual Memory Usage [3]: 1895952 KB
Peak Virtual Memory Usage [4]: 1894672 KB
Peak Virtual Memory Usage [5]: 1887664 KB
Peak Virtual Memory Usage: 11761888 KB = 11.22 GB
Timing information:
  Segment:                Elapsed          Cumulative       #    Parallel
                          Time:     %      Time:     %           efficiency %:
  ..........................................................................
  Start                     0.67s   0.2      0.67s   0.2     1
  Start of Iteration     6m22.35s  94.2   6m23.03s  94.3     1
    Matrix-vector product 5m06.70s  75.5                    100  [99 99 97 88 90 93] =  94.9
  End of Iteration          0.53s   0.1   6m23.56s  94.5     1
  WRITE SOLUTIONS          22.42s   5.5   6m45.99s 100.0     1
```

Example of good parallel scaling

Timo's RDC example:

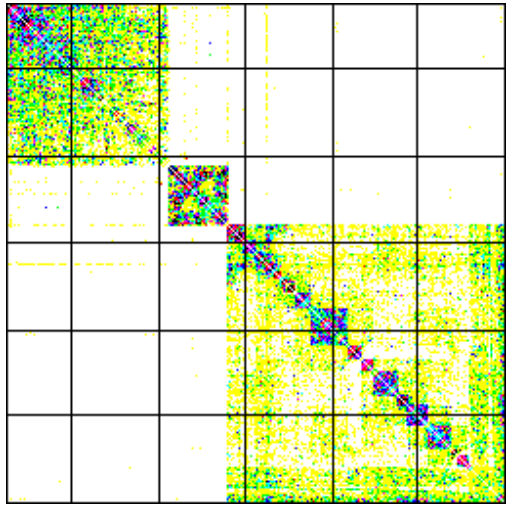
- Most communication is through common blocks (`imake99.log`).

```
Communication in parallel computing:
  14.05% using the linked list
  85.22% through the common blocks
   0.73% through the across blocks

  16.73% of equations need communication

Total communicated equations:     18150113
  - linked list             :      2549418
  - common blocks           :     15468323
  - across blocks           :       132372

Total number of equations   :    108479691
  - within blocks           :    108347319
  - across blocks           :       132372
```

Example of good parallel scaling

Timo's RDC example:

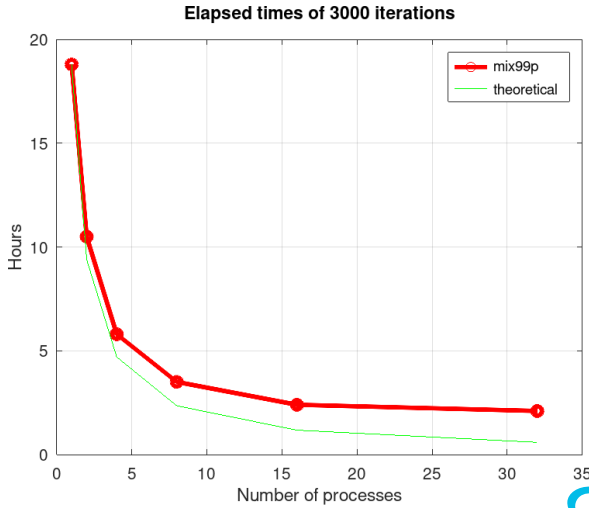
- Most communication is through common blocks (`imake99.log`).
- Blocks are well ordered and communication is quite diagonal.



Example of good parallel scaling

Timo's RDC example:

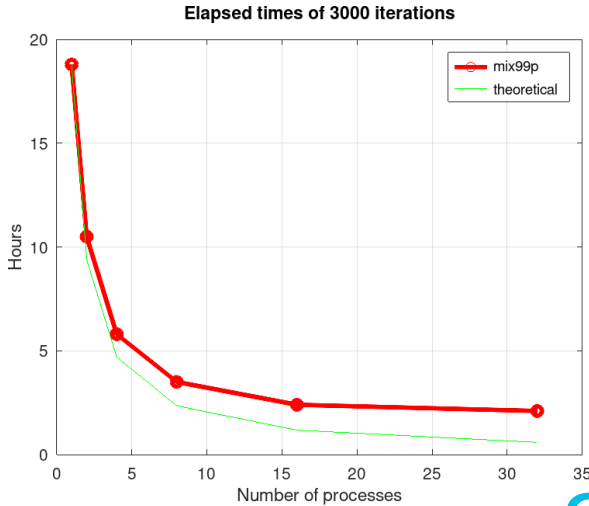
- Most communication is through common blocks (`imake99.log`).
- Blocks are well ordered and communication is quite diagonal.
- Parallel performance scales quite well.



Example of good parallel scaling

Timo’s RDC example:

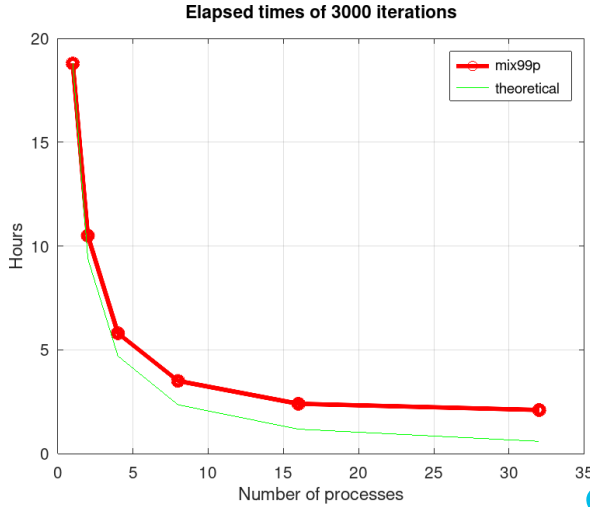
- Most communication is through common blocks (`imake99.log`).
- Blocks are well ordered and communication is quite diagonal.
- Parallel performance scales quite well.
- Parallel efficiency remains at 78.5% with 32 processes.




```
[83 82 82 83 82 83 82 83 82 83 82 83 82 82 80 74 73 73 72 73 73 73 72 73 72 74 72 73 73 73 72 99] = 78.5
```

Reasons for non-optimal parallel scaling

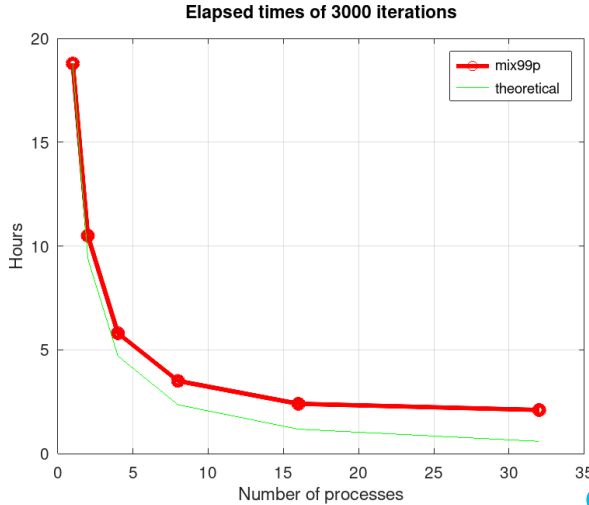
Why non-optimal parallel scaling:



Reasons for non-optimal parallel scaling

Why non-optimal parallel scaling:

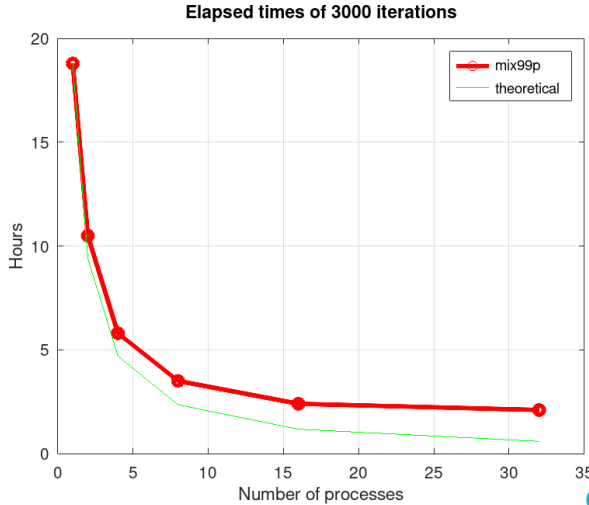
- Parts of program cannot be parallelized.



Reasons for non-optimal parallel scaling

Why non-optimal parallel scaling:

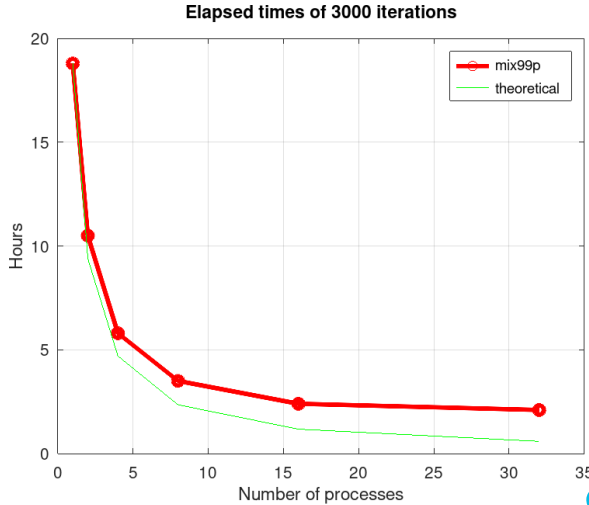
- Parts of program cannot be parallelized.
- Parallel processes compete from resources:
  - I/O.
  - Memory access.



Reasons for non-optimal parallel scaling

Why non-optimal parallel scaling:

- Parts of program cannot be parallelized.
- Parallel processes compete from resources:
  - I/O.
  - Memory access.
- Overheads from parallel operation:
  - Communication.



Example of bad block ordering: Nordic Holstein model

Nordic Holstein model had bad block ordering:

- 27 traits, 9 M animals, 350 M observations, 380 M equations.

Example of bad block ordering: Nordic Holstein model

Nordic Holstein model had bad block ordering:

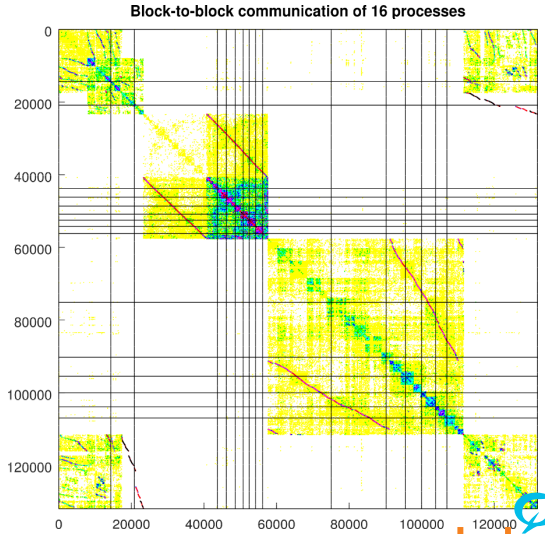
- 27 traits, 9 M animals, 350 M observations, 380 M equations.
- Over 90% of communication was through "linked list" (`imake99.log`).

```
Communication in parallel computing:
  91.33% using the linked list
   7.98% through the common blocks
   0.69% through the across blocks

   5.78% of equations need communication

Total communicated equations:     21999258
  - linked list             :     20090904
  - common blocks           :      1756440
  - across blocks           :       151914

Total number of equations   :    380436594
  - within blocks           :    380284680
  - across blocks           :       151914
```

Example of bad block ordering: Nordic Holstein model

Nordic Holstein model had bad block ordering:

- 27 traits, 9 M animals, 350 M observations, 380 M equations.
- Over 90% of communication was through "linked list" (`imake99.log`).
- Finnish herds were placed first and last causing lot of communication.



Example of bad block ordering: Nordic Holstein model

Nordic Holstein model had bad block ordering:

- 27 traits, 9 M animals, 350 M observations, 380 M equations.
- Over 90% of communication was through "linked list" (`imake99.log`).
- Finnish herds were placed first and last causing lot of communication.
- Parallel efficiency was only 41.1%:
  - Most processes were waiting $\frac{2}{3}$ of the time for few processes to finish.

```
Timing information:
  Parallel
  efficiency %:
...............

  [36 99 61 37 37 38 37 37 37 38 32 32 32 32 32 34] =  41.1
```

Block reordering and repartitioning using Metis

MiX99 includes utility script for block reordering:

- In `tools` subdir: `partition_blocks`
- Reorders and repartitions blocks.
- Minimizes communication using external **Metis** program (`gpmetis`).
- Sorts data and pedigree files.

```
Reorder and partition blocks to minimize block-to-block communication:
  Directory for MiX99 binaries: ../run
MiX99 model file: RDC_local.CLM
  CLIM file detected
  MiX99 preprocessor (mix99i) options: --usemacros
  Output pedigree file: sorted.ped
  Output data     file: sorted.dat
Analyze block-to-block communication of non-common blocks:
  ../run/mix99i --usemacros RDC_local.CLM > mix99i_partition_blocks.log
    Pedigree file: ../pedigree.ped
      Pedigree block code column: 4
    Data file: ../datafile.dat
      Data format: text
      Number of integer columns: 21
      Number of real    columns: 29
      Block        column: 1
      Relationship column: 2
      Trait group  column: 3
    Number of processes: 6
    Elapsed time: 6 minutes 33.24 seconds
  ../run/block_information -g > block_information_partition_blocks.log
    Number of blocks: 19584
    Number of common blocks: 118
    Elapsed time: 1 minute 47.82 seconds
Reorder and partition non-common blocks to 6 partitions:
  gpmetis BMatrix.graph 6 > gpmetis_partition_blocks.log
  Build PARALLEL_LASTBLOCK file containing optimized block distribution:
     5214
     8058
    10675
    13761
    16819
    19584
  Figure out new block numbers
  Combine original_blockcode.dat and new block number to BMatrix.map
  Elapsed time: 0.11 seconds
```

Block reordering and repartitioning using Metis

MiX99 includes utility script for block reordering:

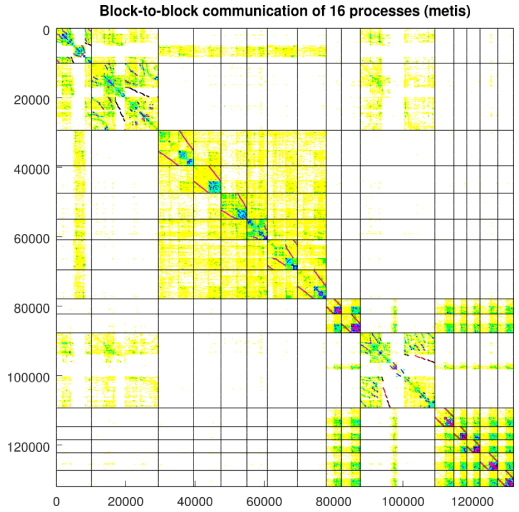
- In `tools` subdir: `partition_blocks`
- Reorders and repartitions blocks.
- Minimizes communication using external **Metis** program (`gpmetis`).
- Sorts data and pedigree files.

```
Reorder file according to block code column:
  File name: ../pedigree.ped
  File type: text
  Renumber block codes using map file: BMatrix.map
  Output file name: sorted.ped
  Block code column: 4
  File type: text
  Number of parallel sorts: 6
  Memory size for sort: 20G
  cat ../pedigree.ped | ../run/renumber_ids.pl -c BMatrix.map 4 \
    | sort -n -k4,4 --parallel=6 -S 20G > sorted.ped
  Elapsed time: 3.05 seconds
Reorder file according to block code column:
  File name: ../datafile.dat
  File type: text
  Multifile support (<filename><#>) enabled.
  Renumber block codes using map file: BMatrix.map
  Output file name: sorted.dat
  Block code column: 1
  Relationship code column: 2
  Trait group code column: 3
  File type: text
  Number of parallel sorts: 6
  Memory size for sort: 20G
  cat ../datafile.dat | ../run/renumber_ids.pl -c BMatrix.map 1 \
    | sort -n -k1,1 -k2,2 -k3,3 --parallel=6 -S 20G > sorted.dat
  Elapsed time: 2 minutes 18.46 seconds
Elapsed time: 10 minutes 42.74 seconds
```

Example of block reordering: Nordic Holstein model

Nordic Holstein model after reordering:

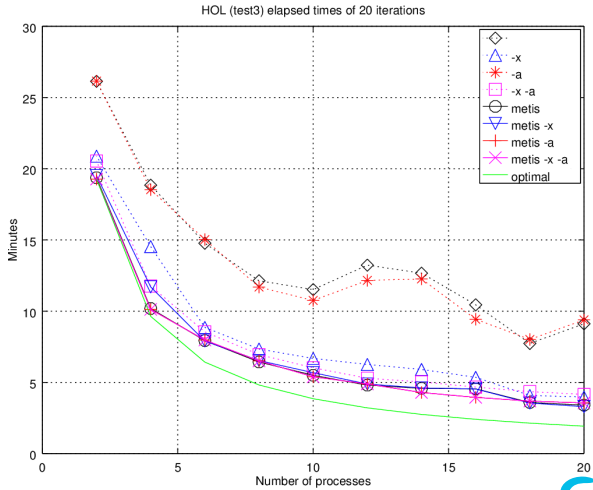
- Reordering and repartitioning minimized communication.



Example of block reordering: Nordic Holstein model

Nordic Holstein model after reordering:

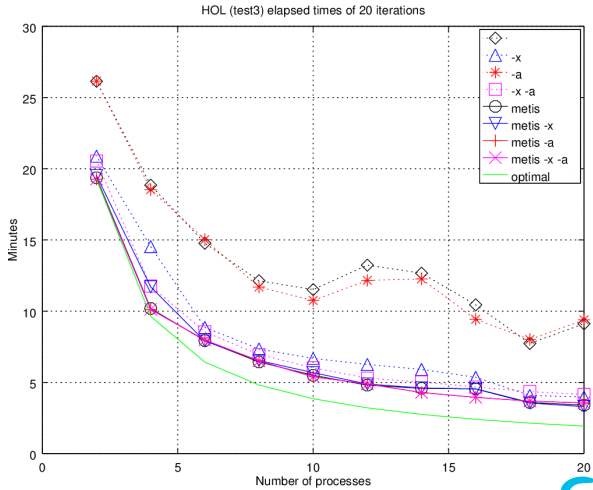
- Reordering and repartitioning minimized communication.
- Elapsed time (of 20 iterations) decreased from 9 minutes down to 4 minutes (with 20 processes).



Example of block reordering: Nordic Holstein model

Nordic Holstein model after reordering:

- Reordering and repartitioning minimized communication.
- Elapsed time (of 20 iterations) decreased from 9 minutes down to 4 minutes (with 20 processes).
- Alternatively, **eXtra large** memory option (or command line option `-x`) speeds up communication.



Example of block reordering: Nordic Holstein model

Nordic Holstein model after reordering:

- Reordering and repartitioning minimized communication.
- Elapsed time (of 20 iterations) decreased from 9 minutes down to 4 minutes (with 20 processes).
- Alternatively, **eXtra large** memory option (or command line option `-x`) speeds up communication.
- Parallel efficiency increased from 41.1% to 89.8%.

```
Timing information:
  Parallel
  efficiency %:
...............

   [85 96 83 79 81 86 79 85 90 99 92 96 90 94 98 94] =  89.8
```

Conclusions

Conclusions

- We looked how MiX99 "parallel" solver `mix99p` differs from "serial" solver `mix99s`.

Conclusions

- We looked how MiX99 "parallel" solver `mix99p` differs from "serial" solver `mix99s`.
- How to monitor parallel performance of the parallel solver `mix99p`.

Conclusions

- We looked how MiX99 "parallel" solver `mix99p` differs from "serial" solver `mix99s`.
- How to monitor parallel performance of the parallel solver `mix99p`.
- What affects the parallel performance.

Conclusions

- We looked how MiX99 "parallel" solver `mix99p` differs from "serial" solver `mix99s`.
- How to monitor parallel performance of the parallel solver `mix99p`.
- What affects the parallel performance.
- How to minimize or speed up communication.

Looking to the future: CPU architectures

- There are some new CPU architectures in which part of memory is closer to some of the computational cores than others, i.e. memory access is **heterogeneous**.

Looking to the future: CPU architectures

- There are some new CPU architectures in which part of memory is closer to some of the computational cores than others, i.e. memory access is **heterogeneous**.

- In the future, we may need to consider splitting and distributing data to memory locations that are closer to cores where they are operated.

Looking to the future: CPU architectures

- There are some new CPU architectures in which part of memory is closer to some of the computational cores than others, i.e. memory access is **heterogeneous**.
- In the future, we may need to consider splitting and distributing data to memory locations that are closer to cores where they are operated.
- This is basically how MiX99 parallel solver `mix99p` is already operating.

Looking to the future: CPU architectures

- There are some new CPU architectures in which part of memory is closer to some of the computational cores than others, i.e. memory access is **heterogeneous**.
- In the future, we may need to consider splitting and distributing data to memory locations that are closer to cores where they are operated.
- This is basically how MiX99 parallel solver `mix99p` is already operating.
- So, similar practices may need to be implemented on the "serial" solver `mix99s` side.

Few words about genomic models

- Parallel MiX99 solver `mix99p` can also utilize models containing genomic information.

Few words about genomic models

- Parallel MiX99 solver `mix99p` can also utilize models containing genomic information.
- One of the MPI processes computes dense matrix parts using multithreading, i.e. utilizing hybrid parallelism.

Few words about genomic models

- Parallel MiX99 solver `mix99p` can also utilize models containing genomic information.
- One of the MPI processes computes dense matrix parts using multithreading, i.e. utilizing hybrid parallelism.
- As the memory access is becoming limiting factor also for the dense matrix operations:
  - Memory footprint of genomic information is kept as small as possible.
  - Marker values (0,1,2) are kept in integer form.
  - **Byte-packing** is used to store 5 marker values to single byte.
  - Up to 20 or 40 times less memory needed for genomic information.


Luke
NATURAL RESOURCES
INSTITUTE FINLAND