Parallel Computing in MiX99

MiX99 course on genomic prediction

Matti Taskinen, Timo Pitkänen, Ismo Strandén

Natural Resources Institute Finland (Luke)

March 10, 2026

Luke
NATURAL RESOURCES INSTITUTE FINLAND

Contents

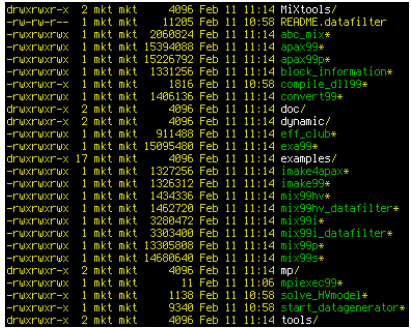
MiX99 has two different solvers:

- What are the differences.
- How to use parallel computing with them.
- How to choose which to use.

Different flavors of MiX99 executables

MiX99 binary distribution contains:

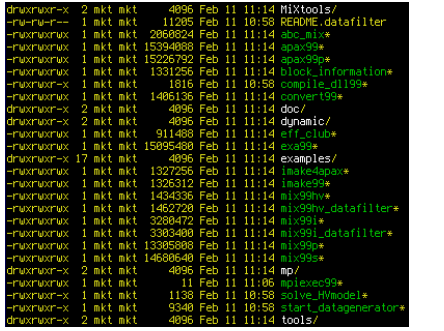
- **"Default"** executables at root level:
  - Utilize one computational core (per process).
  - Can be used in parallel by running multiple instances at the same time.



Different flavors of MiX99 executables

MiX99 binary distribution contains:

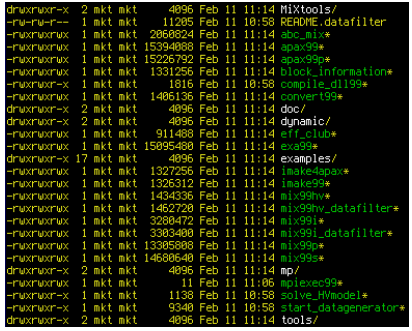
- **"Default"** executables at root level:
  - Utilize one computational core (per process).
  - Can be used in parallel by running multiple instances at the same time.
- **Multi-threaded** (or multi-process) executables in `mp` subdirectory:
  - Utilize many computational cores in parallel.
  - This presentation concentrates on these.



Different flavors of MiX99 executables

MiX99 binary distribution contains:

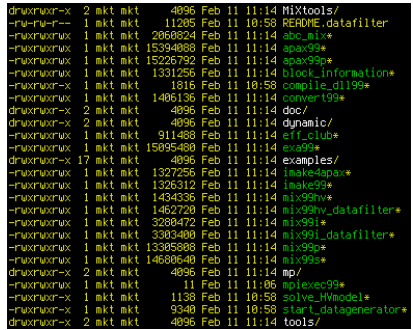
- **"Default"** executables at root level:
  - Utilize one computational core (per process).
  - Can be used in parallel by running multiple instances at the same time.
- **Multi-threaded** (or multi-process) executables in `mp` subdirectory:
  - Utilize many computational cores in parallel.
  - This presentation concentrates on these.
- **Dynamically linked** executables in `dynamic`:
  - Seldomly needed.
  - Allows updating the used libraries.



Two main types of MiX99 executables

MiX99 programs are either:

- **Single-process programs**:
  - Pre-processor: `mix99i`
  - "Serial" solver: `mix99s`
  - Reliability approximation: `apax99`
  - And most of the others.
  - First part of this presentation.



Two main types of MiX99 executables

MiX99 programs are either:

- **Single-process programs**:
  - Pre-processor: `mix99i`
  - "Serial" solver: `mix99s`
  - Reliability approximation: `apax99`
  - And most of the others.
  - First part of this presentation.
- **Multi-process MPI programs**:
  - "Parallel" solver: `mix99p`
  - Reliability approximation: `apax99p`
  - Multiple parallel processes.
  - Utilize Message Passing Interface (MPI) communication between processes.
  - Second part of this presentation.

```
drwxrwxr-x  2 mkt mkt     4096 Feb 11 11:14 MiXtools/
-rw-rw-r--  1 mkt mkt    11205 Feb 11 10:58 README.datafilter
-rwxrwxrwx  1 mkt mkt  2068824 Feb 11 11:14 abc_mix*
-rwxrwxrwx  1 mkt mkt 15394088 Feb 11 11:14 apax99*
-rwxrwxrwx  1 mkt mkt 15226792 Feb 11 11:14 apax99p*
-rwxrwxrwx  1 mkt mkt  1331256 Feb 11 11:14 block_information*
-rwxrwxr-x  1 mkt mkt     1816 Feb 11 10:58 compile_dll99*
-rwxrwxrwx  1 mkt mkt  1406136 Feb 11 11:14 convert99*
drwxrwxr-x  2 mkt mkt     4096 Feb 11 11:14 doc/
drwxrwxr-x  2 mkt mkt     4096 Feb 11 11:14 dynamic/
-rwxrwxrwx  1 mkt mkt   911488 Feb 11 11:14 elf_glub*
-rwxrwxrwx  1 mkt mkt 15095480 Feb 11 11:14 exa99*
drwxrwxr-x 17 mkt mkt     4096 Feb 11 11:14 examples/
-rwxrwxrwx  1 mkt mkt  1327256 Feb 11 11:14 imake4apax*
-rwxrwxrwx  1 mkt mkt  1326312 Feb 11 11:14 imake99*
-rwxrwxrwx  1 mkt mkt  1434336 Feb 11 11:14 mix99hv*
-rwxrwxrwx  1 mkt mkt  1462720 Feb 11 11:14 mix99hv_datafilter*
-rwxrwxrwx  1 mkt mkt  3280472 Feb 11 11:14 mix99i*
-rwxrwxrwx  1 mkt mkt  3303400 Feb 11 11:14 mix99i_datafilter*
-rwxrwxrwx  1 mkt mkt 13305088 Feb 11 11:14 mix99p*
-rwxrwxrwx  1 mkt mkt 14680640 Feb 11 11:14 mix99s*
drwxrwxr-x  2 mkt mkt     4096 Feb 11 11:14 mp/
-rwxrwxrwx  1 mkt mkt       11 Feb 11 11:06 mpiexec99*
-rwxrwxr-x  1 mkt mkt     1138 Feb 11 10:58 solve_HVmodel*
-rwxrwxrwx  1 mkt mkt     9340 Feb 11 10:58 start_datagenerator*
drwxrwxr-x  2 mkt mkt     4096 Feb 11 11:14 tools/
```

Three types of parallel computing in MiX99 programs

MiX99 utilize parallel computing through:

- High-performance **matrix libraries**:
  - Intel MKL
  - AMD AOCL

Three types of parallel computing in MiX99 programs

MiX99 utilize parallel computing through:

- High-performance **matrix libraries**:
  - Intel MKL
  - AMD AOCL
- **Dense matrix** operations:
  - Especially genomic data.

Three types of parallel computing in MiX99 programs

MiX99 utilize parallel computing through:

- High-performance **matrix libraries**:
  - Intel MKL
  - AMD AOCL
- Custom **OpenMP** parallelizations.
- **Dense matrix** operations:
  - Especially genomic data.

Three types of parallel computing in MiX99 programs

MiX99 utilize parallel computing through:

- High-performance **matrix libraries**:
  - Intel MKL
  - AMD AOCL
- Custom **OpenMP** parallelizations.
- **Dense matrix** operations:
  - Especially genomic data.
- Various places:
  - Reversed reliability approximation in `apax99`.

Three types of parallel computing in MiX99 programs

MiX99 utilize parallel computing through:

- High-performance **matrix libraries**:
  - Intel MKL
  - AMD AOCL
- Custom **OpenMP** parallelizations.
- MPI programs.
- **Dense matrix** operations:
  - Especially genomic data.
- Various places:
  - Reversed reliability approximation in `apax99`.

Three types of parallel computing in MiX99 programs

MiX99 utilize parallel computing through:

- High-performance **matrix libraries**:
  - Intel MKL
  - AMD AOCL
- Custom **OpenMP** parallelizations.
- MPI programs.
- **Dense matrix** operations:
  - Especially genomic data.
- Various places:
  - Reversed reliability approximation in `apax99`.
- `mix99p` and `apax99p`.

Controlling parallel threads/processes in MiX99 programs

Controlling level of parallel computing:

Controlling parallel threads/processes in MiX99 programs

Controlling level of parallel computing:

- Using **single-threaded** (non-`mp`) MiX99 executables:
  - Utilizes only single computational thread (except MPI programs).

Controlling parallel threads/processes in MiX99 programs

Controlling level of parallel computing:

- Using **single-threaded** (non-`mp`) MiX99 executables:
  - Utilizes only single computational thread (except MPI programs).
- Multi-threaded executables (`mp`):
  - Setting **number of parallel threads**.
  - Either various environment variables.
  - Or new command line option: `-nt`
  - If not set, uses all computational cores!

```
function set_num_threads() {
  export MKL_NUM_THREADS="$1"
  export OPENBLAS_NUM_THREADS="$1"
  export GOTO_NUM_THREADS="$1"
  export BLIS_NUM_THREADS="$1"
  export OMP_NUM_THREADS="$1"
}

set_num_threads 10
```

```
./mix99s -nt 10 ...
```

Controlling parallel threads/processes in MiX99 programs

Controlling level of parallel computing:

- Using **single-threaded** (non-`mp`) MiX99 executables:
  - Utilizes only single computational thread (except MPI programs).
- Multi-threaded executables (`mp`):
  - Setting **number of parallel threads**.
  - Either various environment variables.
  - Or new command line option: `-nt`
  - If not set, uses all computational cores!
- MPI programs (`mix99p` and `apax99p`):
  - **Number of parallel processes** in CLIM.

```
function set_num_threads() {
  export MKL_NUM_THREADS="$1"
  export OPENBLAS_NUM_THREADS="$1"
  export GOTO_NUM_THREADS="$1"
  export BLIS_NUM_THREADS="$1"
  export OMP_NUM_THREADS="$1"
}

set_num_threads 10
```

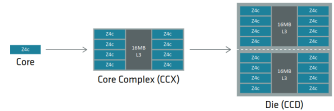
```
./mix99s -nt 10 ...
```

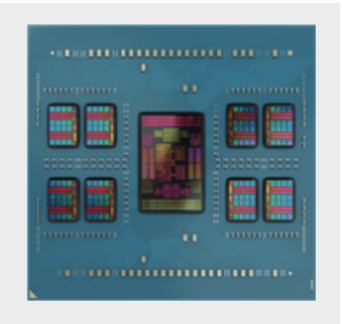
```
PARALLEL          6    10
```

What is optimal number of threads/processes?

Performance of parallelization depends on:

- Hardware:
  - Number of computational cores in CPU.
  - Modern CPUs are glued from chiplets.
  - Using all cores may not be the fastest.

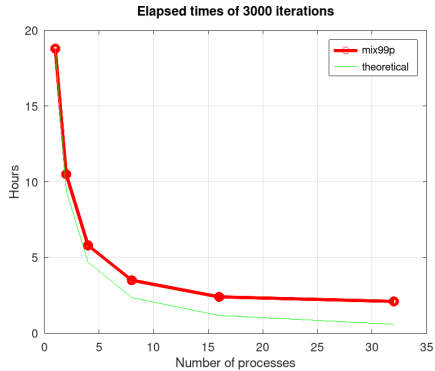




What is optimal number of threads/processes?

Performance of parallelization depends on:

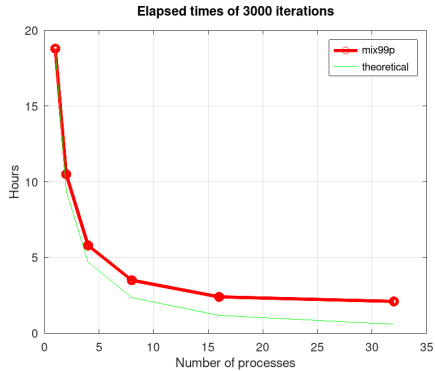
- Hardware:
  - Number of computational cores in CPU.
  - Modern CPUs are glued from chiplets.
  - Using all cores may not be the fastest.
- Task:
  - Some problems parallelize well, some don't.



What is optimal number of threads/processes?

Performance of parallelization depends on:

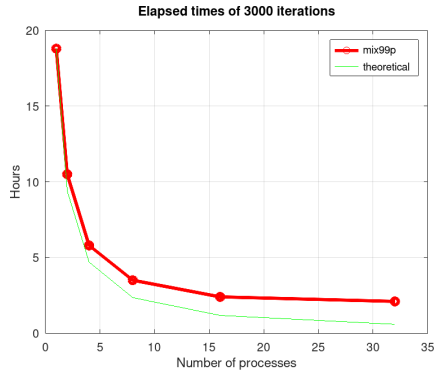
- Hardware:
  - Number of computational cores in CPU.
  - Modern CPUs are glued from chiplets.
  - Using all cores may not be the fastest.
- Task:
  - Some problems parallelize well, some don't.
- Other users:
  - Concurrent users compete on resources.



What is optimal number of threads/processes?

Performance of parallelization depends on:

- Hardware:
  - Number of computational cores in CPU.
  - Modern CPUs are glued from chiplets.
  - Using all cores may not be the fastest.
- Task:
  - Some problems parallelize well, some don't.
- Other users:
  - Concurrent users compete on resources.
- May need higher level of control:
  - Thread/process **affinity**, **CPU pinning**.
  - Slurm jobs may not have optimal settings.



Luke NATURAL RESOURCES INSTITUTE FINLAND

Multi-threading with MiX99 "serial" solver **`mix99s`**

Luke
NATURAL RESOURCES INSTITUTE FINLAND

How to operate "serial" MiX99 solver **`mix99s`**

Using "serial" MiX99 solver `mix99s`:

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

```
mix99s -nt nthreads options < option_file
```

How to operate "serial" MiX99 solver **`mix99s`**

Using "serial" MiX99 solver `mix99s`:

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

```
mix99s -nt nthreads options < option_file
```

- Solver options through:
  - **Solver option file** (above `option_file`).
  - Or using **command line options**.
  - Note: If both given, remember to use command line option: `-i`. Otherwise the solver option file is omitted.

Parallel computing with MiX99 "serial" solver **`mix99s`**

Parallelization with "serial" solver `mix99s`:

Parallel computing with MiX99 "serial" solver **`mix99s`**

Parallelization with "serial" solver `mix99s`:

- **Multi-threaded** `mix99s` (`mp` subdirectory).

Parallel computing with MiX99 "serial" solver `mix99s`

Parallelization with "serial" solver `mix99s`:

- **Multi-threaded** `mix99s` (`mp` subdirectory).
- Parallelizes mostly only **dense matrix** operations.

Parallel computing with MiX99 "serial" solver **`mix99s`**

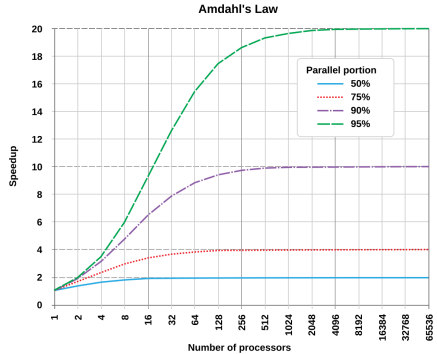
Parallelization with "serial" solver `mix99s`:

- **Multi-threaded** `mix99s` (`mp` subdirectory).
- Parallelizes mostly only **dense matrix** operations.
- Models that benefit from multi-threading have large dense matrices:
  - CLIM commands `GBLUP`, `COVFILE`, `SSGBLUP`, `REGMATRIX`, or `SNPMATRIX`.
  - Especially models with genomic data.

Parallel computing with MiX99 "serial" solver `mix99s`

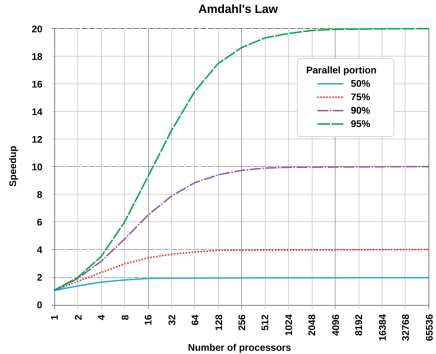
Parallelization with "serial" solver `mix99s`:

- **Multi-threaded** `mix99s` (`mp` subdirectory).
- Parallelizes mostly only **dense matrix** operations.
- Models that benefit from multi-threading have large dense matrices:
  - CLIM commands `GBLUP`, `COVFILE`, `SSGBLUP`, `REGMATRIX`, or `SNPMATRIX`.
  - Especially models with genomic data.
- **NOTE:** Other parts of the operation do not gain from this multi-threaded parallelization!



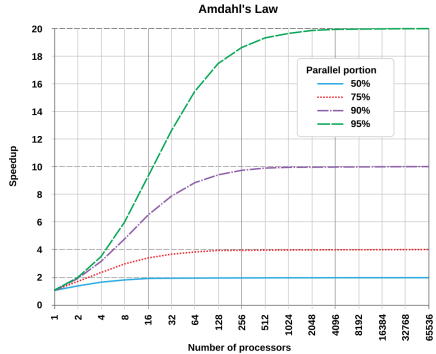
Amdahl’s law on parallelization

- If the operation has two parts:
  - One part that parallelizes well.
  - Other part that does not parallelize.



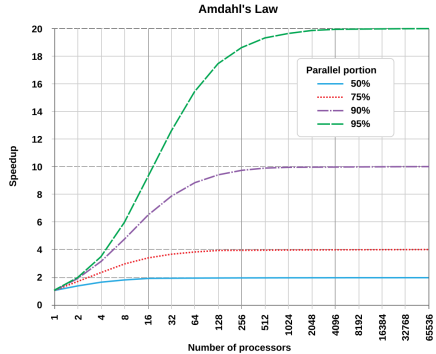
Amdahl's law on parallelization

- If the operation has two parts:
  - One part that parallelizes well.
  - Other part that does not parallelize.
- Performance improvement from parallelization is limited by the fraction of time that the parallelized part took originally.



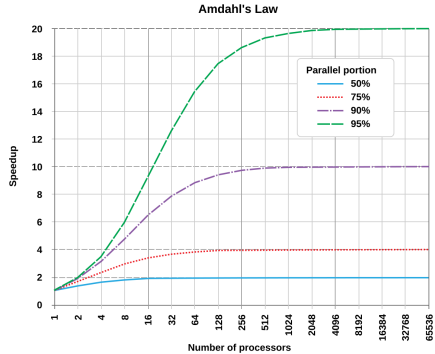
Amdahl's law on parallelization

- If the operation has two parts:
  - One part that parallelizes well.
  - Other part that does not parallelize.
- Performance improvement from parallelization is limited by the fraction of time that the parallelized part took originally.
- If parallelized part took originally **relative time** t then maximum speedup is $\frac{1}{1-t}$:



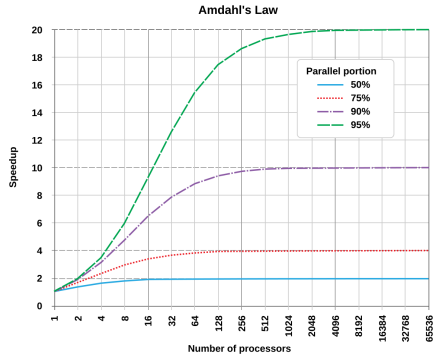
Amdahl's law on parallelization

- If the operation has two parts:
  - One part that parallelizes well.
  - Other part that does not parallelize.
- Performance improvement from parallelization is limited by the fraction of time that the parallelized part took originally.
- If parallelized part took originally **relative time** t then maximum speedup is $\frac{1}{1-t}$:
  - If 50% of the work can be parallelized, maximum speedup is 2.



Amdahl's law on parallelization

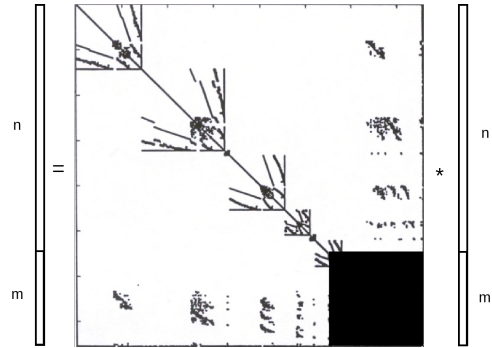
- If the operation has two parts:
  - One part that parallelizes well.
  - Other part that does not parallelize.
- Performance improvement from parallelization is limited by the fraction of time that the parallelized part took originally.
- If parallelized part took originally **relative time** t then maximum speedup is $\frac{1}{1-t}$:
  - If 50% of the work can be parallelized, maximum speedup is 2.
  - If 90% of the work can be parallelized, maximum speedup is 10.



Matrix-vector product with partially dense matrix

MiX99 solvers perform repeatedly **matrix-vector product**:

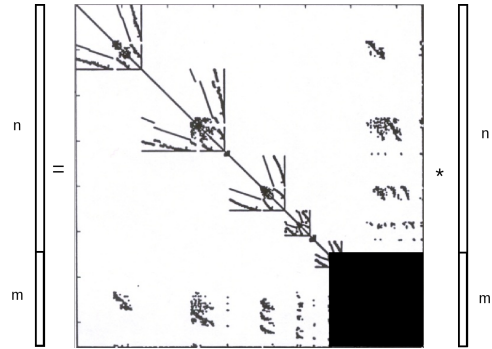
- $\mathbf{u} = \mathbf{C} * \mathbf{v}$
- Matrix $\mathbf{C}$ has **dense block** of size $m \times m$ and size of sparse part is n.
- Let **sparsity** of sparse part be d.
- **Complexity** of matrix-vector product:
  - Dense part: m^2
  - Sparse part: $dn^2 + 2dnm$



Matrix-vector product with partially dense matrix

Assuming 10,000 dense "genotypes" and there are 10 non-zeros on sparse "animal model" part per row:

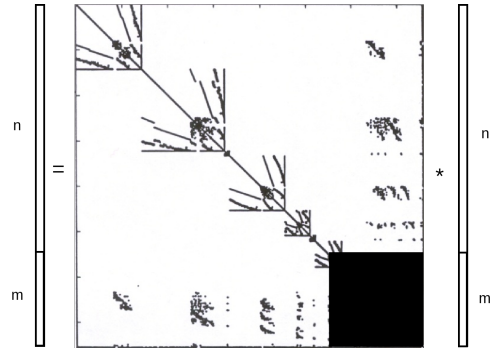
- $m = 10,000$ and $d = \frac{10}{n+m}$.



Matrix-vector product with partially dense matrix

Assuming 10,000 dense "genotypes" and there are 10 non-zeros on sparse "animal model" part per row:

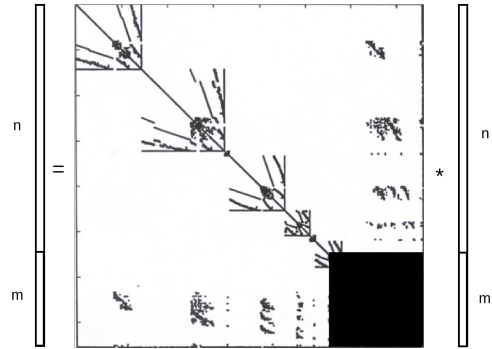
- $m = 10,000$ and $d = \frac{10}{n+m}$.
- How large is the sparse part n, in order for the dense part to take 90% of total time (in matrix-vector product)?



Matrix-vector product with partially dense matrix

Assuming 10,000 dense "genotypes" and there are 10 non-zeros on sparse "animal model" part per row:

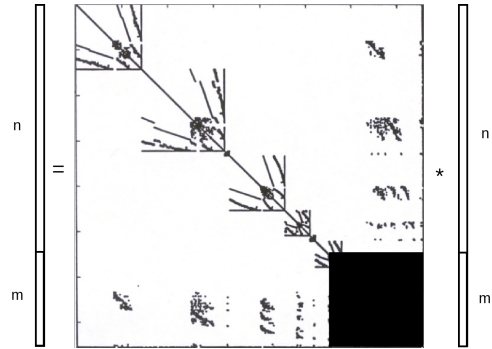
- $m = 10,000$ and $d = \frac{10}{n+m}$.
- How large is the sparse part n, in order for the dense part to take 90% of total time (in matrix-vector product)?
- $\frac{m^2}{m^2+dn(n+2m)} = \frac{9}{10} \quad \Rightarrow \quad n = 1,101,201$



Matrix-vector product with partially dense matrix

Assuming 10,000 dense "genotypes" and there are 10 non-zeros on sparse "animal model" part per row:

- $m = 10,000$ and $d = \frac{10}{n+m}$.
- How large is the sparse part n, in order for the dense part to take 90% of total time (in matrix-vector product)?
- $\frac{m^2}{m^2+dn(n+2m)} = \frac{9}{10} \quad \Rightarrow \quad n = 1,101,201$
- With 1.1 M population 10,000 "genotypes" consume 90% of computation time.



Example with large dense matrices

Model with two large dense matrices:

Example with large dense matrices

Model with two large dense matrices:

- Matrix-vector times (single-thread):
  - Dense part: 69.3s
  - Total time: 78.6s

Example with large dense matrices

Model with two large dense matrices:

- Matrix-vector times (single-thread):
  - Dense part: 69.3s
  - Total time: 78.6s
- 88% of time on dense part.

Example with large dense matrices

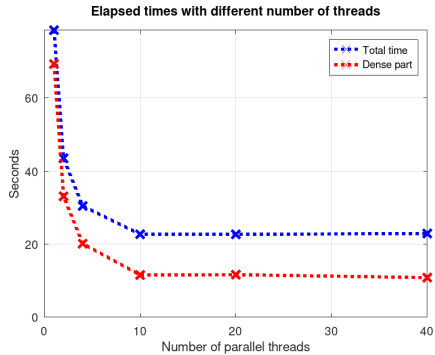
Model with two large dense matrices:

- Matrix-vector times (single-thread):
  - Dense part: 69.3s
  - Total time: 78.6s
- 88% of time on dense part.
- Amdahl: 8.5 speedup from parallelization.

Example with large dense matrices

Model with two large dense matrices:

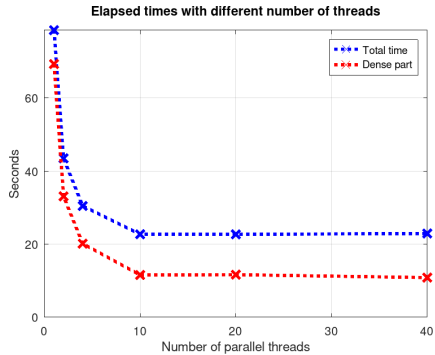
- Matrix-vector times (single-thread):
  - Dense part: 69.3s
  - Total time: 78.6s
- 88% of time on dense part.
- Amdahl: 8.5 speedup from parallelization.
- But multi-threads give only 3.4 speedup:
  - Slows down with larger number of threads.



Example with large dense matrices

Model with two large dense matrices:

- Matrix-vector times (single-thread):
  - Dense part: 69.3s
  - Total time: 78.6s
- 88% of time on dense part.
- Amdahl: 8.5 speedup from parallelization.
- But multi-threads give only 3.4 speedup:
  - Slows down with larger number of threads.
- Threads compete for the same resources:
  - Dense operations are also memory limited.



Memory options for large matrices in MiX99 solvers

Multi-threaded dense matrix computing requires the matrix to fit in memory:

- Solvers have **memory options** for controlling memory usage:
  - For $\mathbf{G}^{-1}$ and single-step.
- Either in **solver option file** or using corresponding command line option.
- **MES**, **MEM**, and **MEL** to keep the matrices in memory or not.
- **MEB b** to operate in blocks.
- **MEA** for automatic block of 2 GB.

MEL in single-step method, reads $\mathbf{G}^{-1}$ or $\mathbf{T}$ matrix from file to memory. The MEL option uses efficient matrix multiplication during PCG iteration. The multiplication can use multi-threading by the multi-threaded versions of MiX99 solvers. Note that when the number of genotyped individuals is large the matrix to read is large and consumes a lot of RAM memory.

MEM is like option MEL but slower and uses slightly less memory, when $\mathbf{T}$ matrix is used. Same as MEL when $\mathbf{G}^{-1}$.

MES does not read $\mathbf{G}^{-1}$ or $\mathbf{T}$ to memory, memory efficient but slow.

MEB b is like option MEL but makes the computations in blocks of size b in ssGTBLUP. Can be faster than MEL for very large matrices.

MEA is like option MEB but the block size b is computed automatically that leads to a block of 2 GB.

Memory options for large genotypes in MiX99 solvers

Controlling memory usage of large genotype matrices in MiX99 solvers:

- Genomic data can be **byte-packed**:
  - `SNPMATRIX` and `REGMATRIX`.
  - 5 markers byte-packed to one byte.
  - Optional centering and scaling.
  - 20 times smaller internal files.
  - 40 times less memory.
- Genomic data can be kept byte-packed in memory:
  - **Unpacking** in blocks.

```
SNPMATRIX  USE=PACK FIRST=2 LAST=7 CENTER=p FORMAT=m
SNPFILE    SNP.dat
CENTERFILE AF.dat
```

```
REGMATRIX  RANDOM SNP ID=1 FIRST=2 LAST=7 FORMAT=m &
                          CENTER=1 SCALE=0.57735
REGFILE    gs_geno.dat
REGPARFILE gs_gen_scaled.par # marker variance
```

Memory options for large genotypes in MiX99 solvers

Memory options for `REGMATRIX`:

- Either in **solver option file** or using corresponding command line option.
- **RDS**, **RDM**, and **RDL**: byte-packed in memory, different block sizes.
- **RDB b** with given block size.
- **RDU m** automatic block size.
- **RDX** in double precision memory.

RDS instructs to use a small block size when calculating contributions from regression design matrices. Keeps SNP marker matrix byte-packed in memory. Currently implemented in `mix99s` only.

RDM similar to RDS but uses a medium block size. Currently in `mix99s` only.

RDL similar to RDM but uses a large block size. Currently in `mix99s` only.

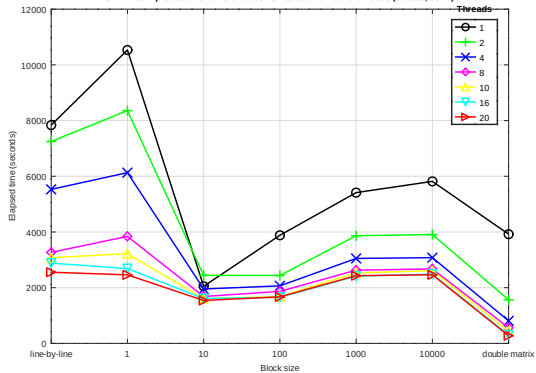
RDX instructs to keep regression design matrices fully in (double precision real) memory. Currently `mix99s` only.

RDB b similar to RDL but uses given block size abs(b) (absolute value of b). If b is positive, keeps SNP marker matrices byte-packed in memory. Currently `mix99s` only.

RDU m similar to RDB but instructs to use given amount of memory (m) when calculating constributions of regression design matrices. Full regression design matrices are kept in memory if possible or at least byte-packed SNP matrices if possible. Block size is also calculated from the given memory limit. Memory size m is given as <amount><unit> where <amount> is an integer and optional <unit> is one of K (kilo), M (mega), G (giga, default), T (tera), and P (petabytes). Example: RDU 12G . Currently `mix99s` only.

Performance testing SNP-BLUP in MiX99 solver `mix99s`

- **1M genotypes, 50k SNPs, 6 traits**:
  - `REGMATRIX` with byte-packing.
  - 373 GB $\Rightarrow$ 9 GB.
- Elapsed times of 20 iterations.
- Different memory options:
  - Line-by-line (reading from file).
  - Unpacked in blocks from memory.
  - Full double precision matrix in memory (373 GB).
- **Two sweet spots:**
  - Unpacked in small blocks.
  - Full double precision matrix in memory. Faster in this server.



Luke NATURAL RESOURCES INSTITUTE FINLAND

Multi-processing with MiX99 "parallel" solver **`mix99p`**

Luke
NATURAL RESOURCES INSTITUTE FINLAND

Parallel operations in "parallel" MiX99 solver `mix99p`

`mix99p` and `apax99p` are parallel MPI programs:

- Utilize 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 **MPI multi-processing** and **multi-threading** (executables from `mp` subdir).

How to operate `mix99p`

"Parallel" MiX99 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 nprocesses mix99p options < option_file
```

How to operate `mix99p`

"Parallel" MiX99 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 nprocesses mix99p options < option_file
```

- Solver options through:
  - Solver option file (above `option_file`).
  - Or using command line options.
  - Note: If both given, remember to use command line option: `-i`. Otherwise the solver option file is omitted.

Features and options available only in `mix99p` solver

- Parallel `mix99p` job can be distributed and executed on multiple separate computers (or "**nodes**"):
  - Seldomly used nowaways.
  - Instead, typically run on a single node containing one or more CPUs each having multiple **physical** or **logical** computational **cores**.
  - Communication between MPI processes though shared memory if single node, otherwise through much slower network.

Features and options available only in `mix99p` solver

- Parallel `mix99p` job can be distributed and executed on multiple separate computers (or "**nodes**"):
  - Seldomly used nowaways.
  - Instead, typically run on a single node containing one or more CPUs each having multiple **physical** or **logical** computational **cores**.
  - Communication between MPI processes though shared memory if single node, otherwise through much slower network.
- 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. **Highly recommended!**

Features and options available only in `mix99p` solver

- Parallel `mix99p` job can be distributed and executed on multiple separate computers (or "**nodes**"):
  - Seldomly used nowaways.
  - Instead, typically run on a single node containing one or more CPUs each having multiple **physical** or **logical** computational **cores**.
  - Communication between MPI processes though shared memory if single node, otherwise through much slower network.
- 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. **Highly recommended!**
- Command line option `"-a"` to use "Allreduce" MPI communication:
  - Can speed up communication between the MPI processes.

Difficulties with MPI programs

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

Difficulties with MPI programs

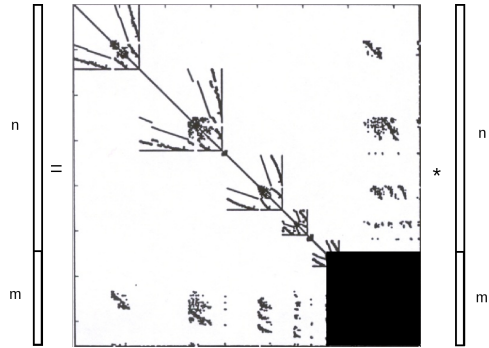
- MPI programs depend on **compile-time** version of the MPI library:
  - **Run-time environment** MPI library version needs to match.
- If more than one MPI libraries installed on run-time environment, correct `mpiexec` version needs to be specifed with, for example, **absolute paths**:
  `/usr/lib64/openmpi/bin/mpiexec -np 10 ...`

Difficulties with MPI programs

- MPI programs depend 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 10 ...`
- MiX99 MPI programs `mix99p` and `apax99p` are **dynamically linked**:
  - Most external libraries are **statically** included within executables.
  - But **system** and MPI libraries are dynamically linked on run-time.
  - This may cause more compatibility problems than with other, fully statically linked, MiX99 programs.

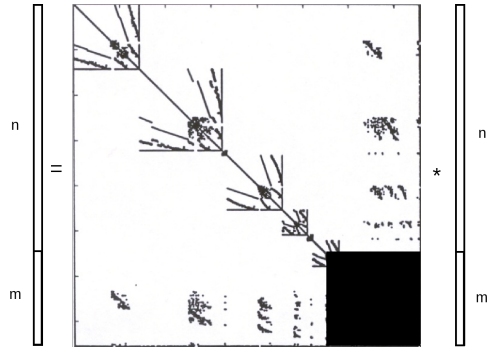
Differences between `mix99s` and `mix99p`

- 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`

- Models with **non-linear** or **categorical traits** are not yet possible to analyze with `mix99p`.
- **Variance component estimation** and **deregression** only in `mix99s`.
- **Heterogeneous variance adjustment** only in `mix99p`.

Dense vs. sparse matrix operations

- At least 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

- At least 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, have always been **memory bandwidth limited**. For example:

Dense vs. sparse matrix operations

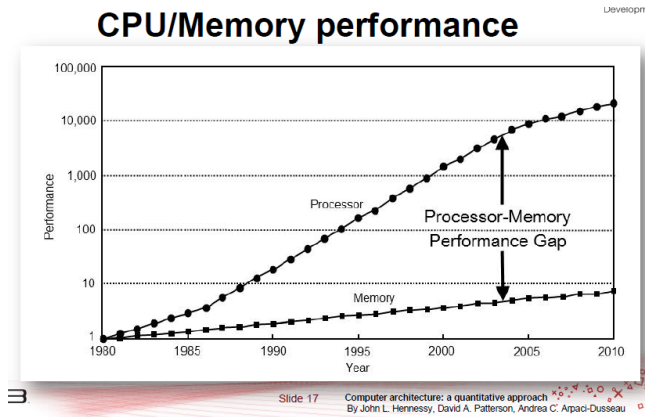
- At least back in the old days dense matrix operations were mostly limited by the speed of **floating point calculations**:
  - Multiplication and addition of floating point numbers were slow.
- Sparse matrix operations, on the other hand, have always been **memory bandwidth limited**. For example: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

- At least 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, have always been **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.

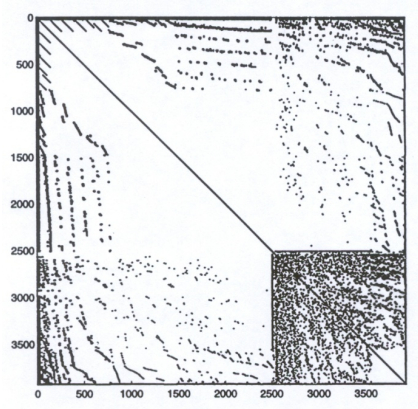


How to speed up sparse matrix operations

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

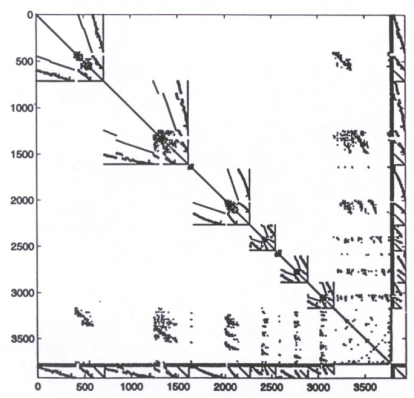
How to speed up sparse matrix operations

- Parallelization does not help so 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,



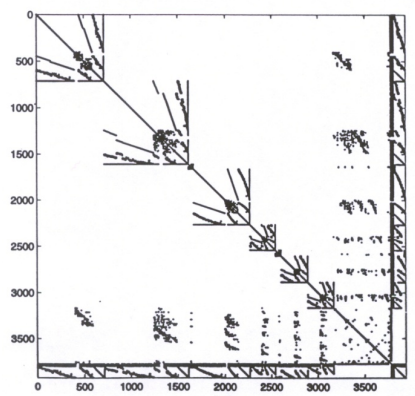
How to speed up sparse matrix operations

- Parallelization does not help so 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**,



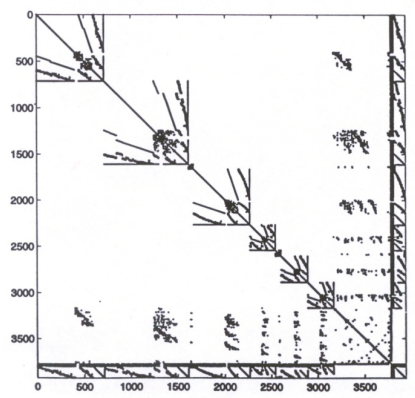
How to speed up sparse matrix operations

- Parallelization does not help so 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.



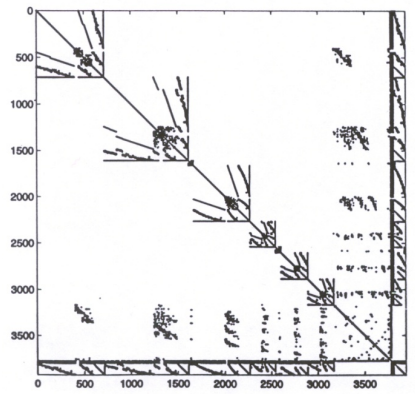
How to speed up sparse matrix operations

- Parallelization does not help so 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.



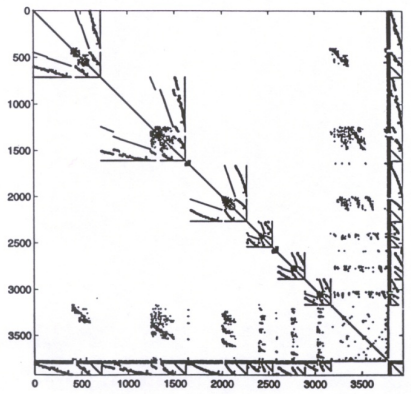
MiX99 equation family blocks: ordering of relations

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



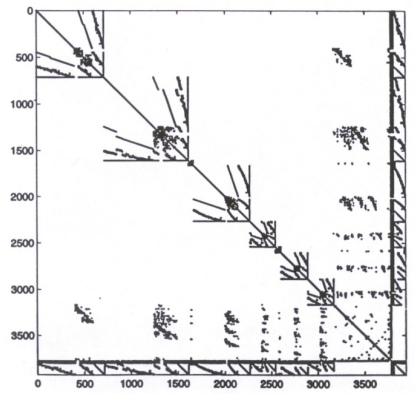
MiX99 equation family blocks: ordering of relations

- 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.



Data-parallelism and workload balancing

MiX99 MPI programs `mix99p` and `apax99p` utilize **data-parallelism**:

Data-parallelism and workload balancing

MiX99 MPI programs `mix99p` and `apax99p` utilize **data-parallelism**:

- Data is split and distributed across the parallel MPI processes.

Data-parallelism and workload balancing

MiX99 MPI programs `mix99p` and `apax99p` utilize **data-parallelism**:

- Data is split and distributed across the parallel MPI processes.
- MiX99 uses the equation family blocks on **balancing the workload** between the parallel MPI processes.

Data-parallelism and workload balancing

MiX99 MPI programs `mix99p` and `apax99p` utilize **data-parallelism**:

- Data is split and distributed across the parallel MPI processes.
- MiX99 uses the equation family blocks on **balancing the workload** between 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.

Data-parallelism and workload balancing

MiX99 MPI programs `mix99p` and `apax99p` utilize **data-parallelism**:

- Data is split and distributed across the parallel MPI processes.
- MiX99 uses the equation family blocks on **balancing the workload** between 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.

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

Data-parallelism and workload balancing

MiX99 MPI programs `mix99p` and `apax99p` utilize **data-parallelism**:

- Data is split and distributed across the parallel MPI processes.
- MiX99 uses the equation family blocks on **balancing the workload** between 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` MPI processes operate on their data parts and **communicate** with each others.

Monitoring block-to-block communication

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

Monitoring block-to-block communication

- Parallel `mix99p` MPI processes operate on their data parts and **communicate** with each others.
- Performance depends partially on the amount of the 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:

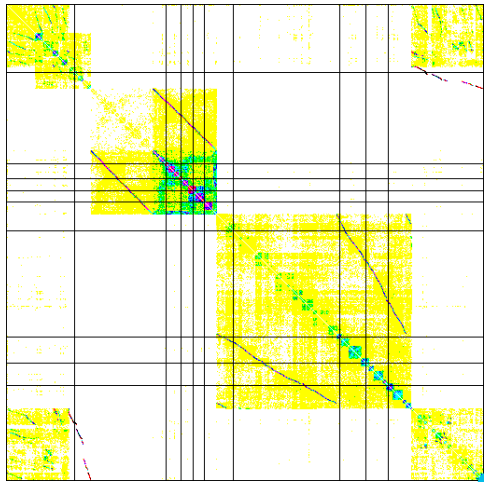
```
block_information -p
```

Visualizing block-to-block dependencies

- Communication between blocks can be visualized using:

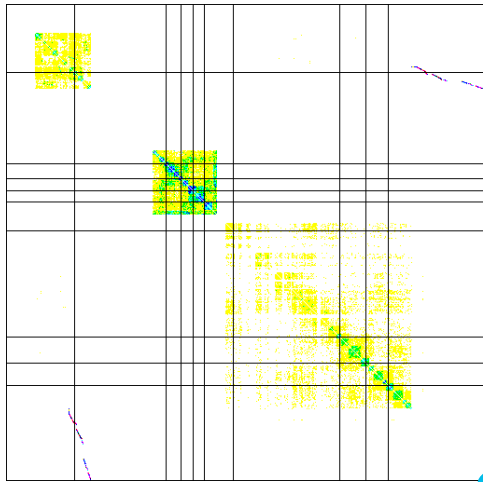
  ```
  block_information -p
  ```

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



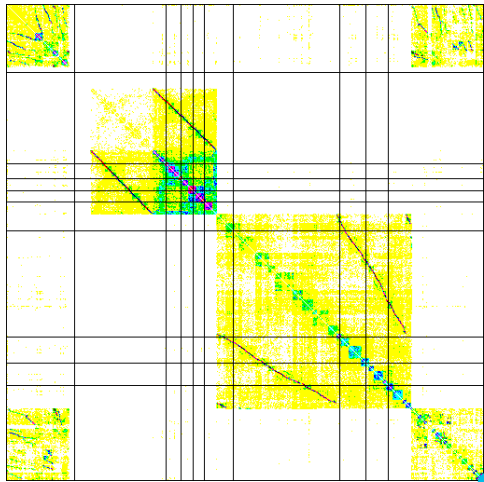
Visualizing block-to-block dependencies

- 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.
  - Black lines indicate partitioning.
- 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.
  - Black lines indicate partitioning.
- 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

Good parallel scaling:

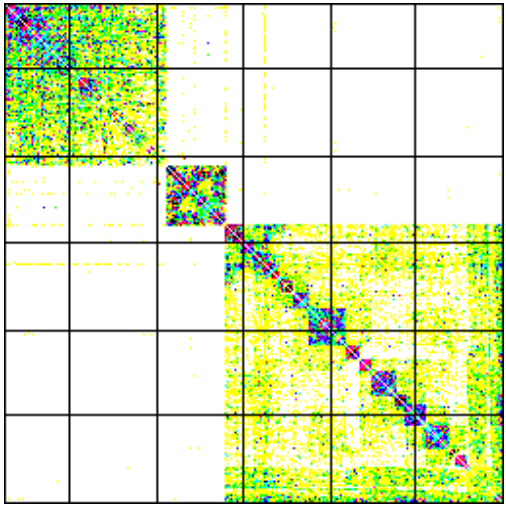
- Most communication is through common blocks (`imake99.log`).

```
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

Good parallel scaling:

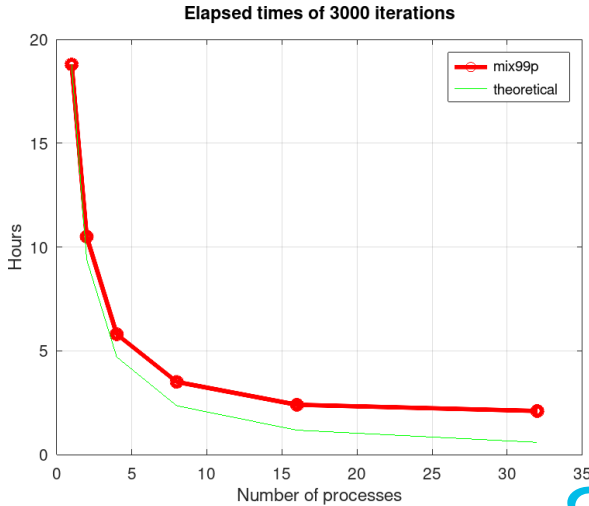
- Most communication is through common blocks (`imake99.log`).
- Blocks are well ordered and communication is quite diagonal.



Example of good parallel scaling

Good parallel scaling:

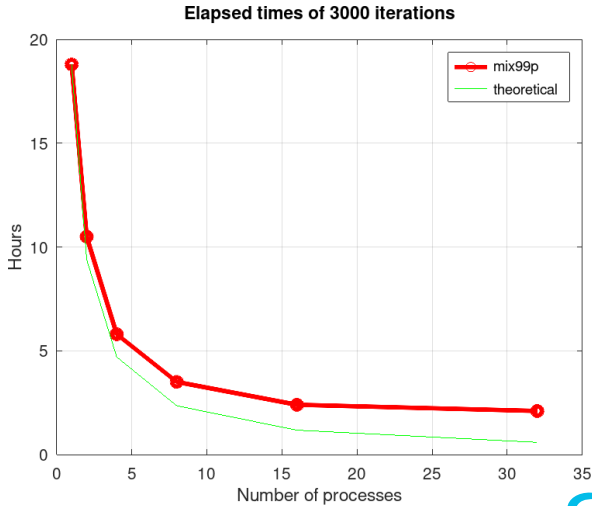
- Most communication is through common blocks (`imake99.log`).
- Blocks are well ordered and communication is quite diagonal.
- Parallel performance scales quite well.



Example of good parallel scaling

Good parallel scaling:

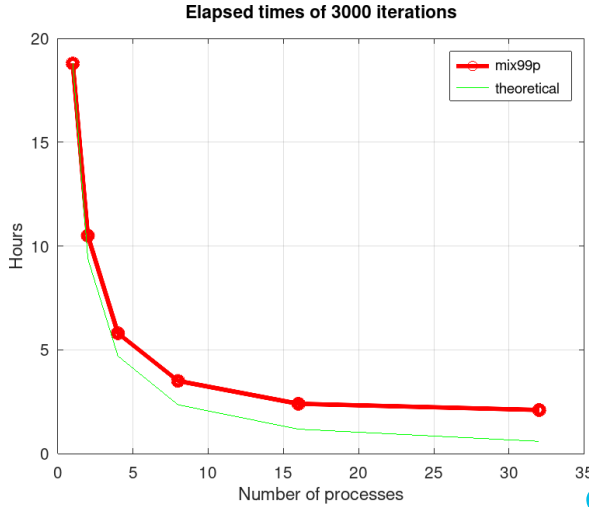
- Most communication is through common blocks (`imake99.log`).
- Blocks are well ordered and communication is quite diagonal.
- Parallel performance scales quite well.
- Parallel efficiency remains at 78.5% with 32 processes.




```
[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

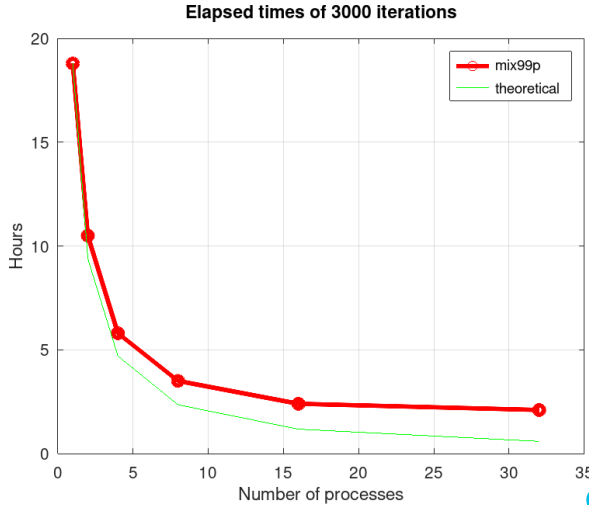
Why non-optimal parallel scaling:



Reasons for non-optimal parallel scaling

Why non-optimal parallel scaling:

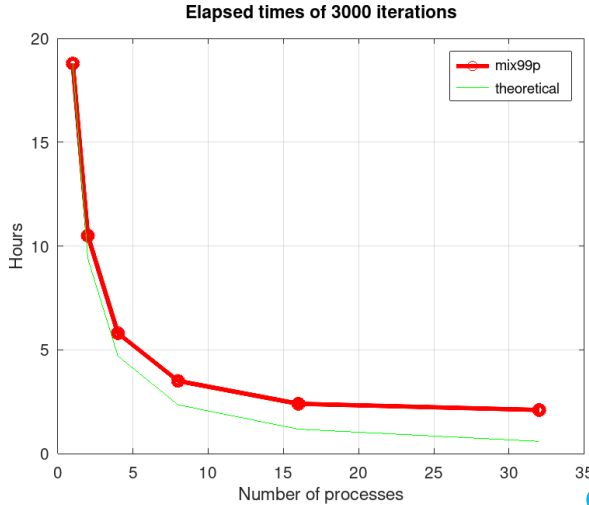
- Parts of program cannot be parallelized.



Reasons for non-optimal parallel scaling

Why non-optimal parallel scaling:

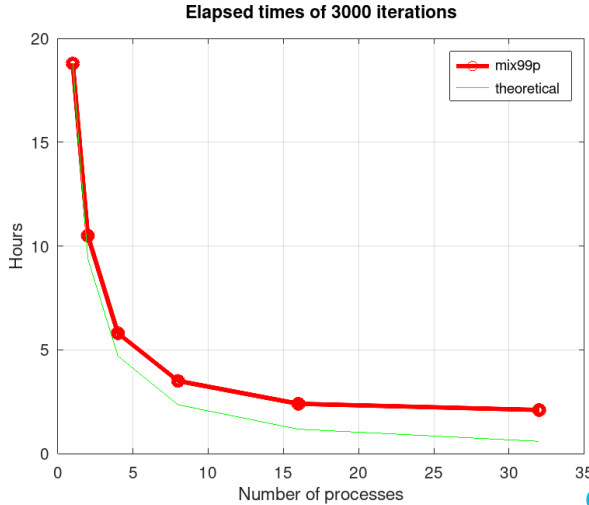
- Parts of program cannot be parallelized.
- Parallel processes compete from resources:
  - I/O.
  - Memory access.



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:

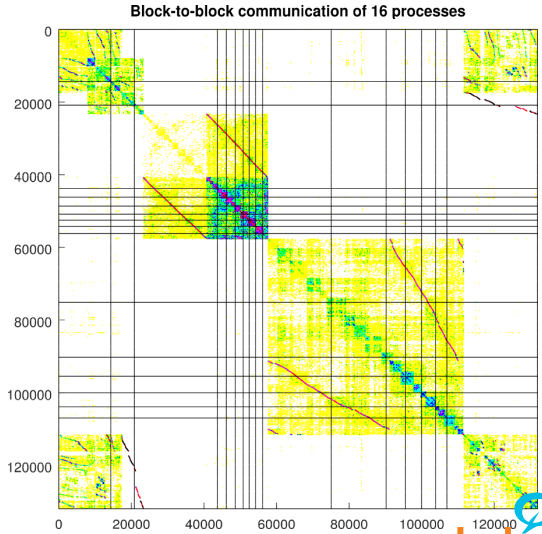
- 27 traits, 9 M animals, 350 M observations, 380 M equations.
- Over 90% of communication was through "linked list" (`imake99.log`).

```
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:

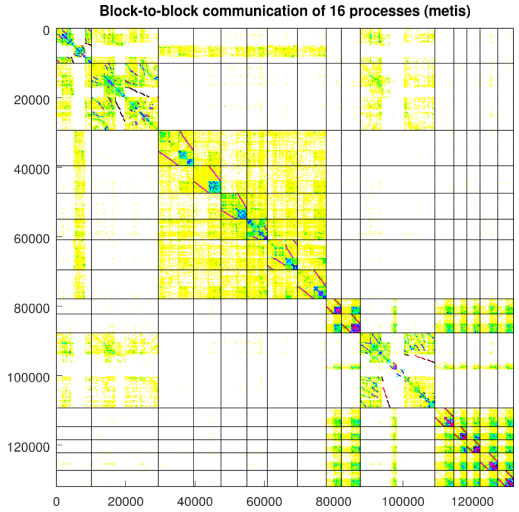
- In `tools` subdir: `partition_blocks`
- Reorders and repartitions blocks.
- Minimizes communication using external **Metis** program (`gpmetis`).
- Sorts data and pedigree files.

```
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:

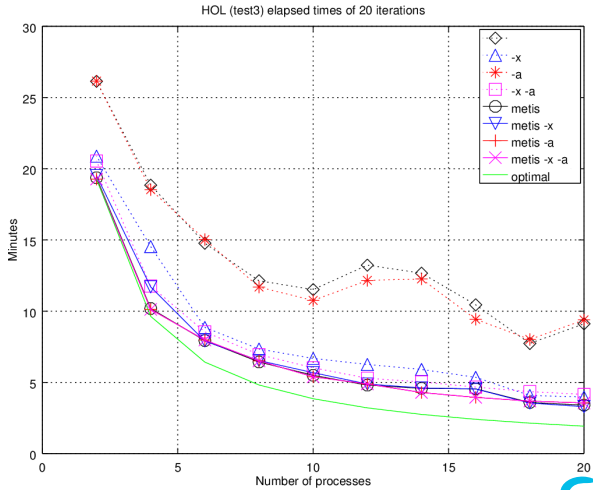
- Reordering and repartitioning minimized communication.



Example of block reordering: Nordic Holstein model

Nordic Holstein model after reordering:

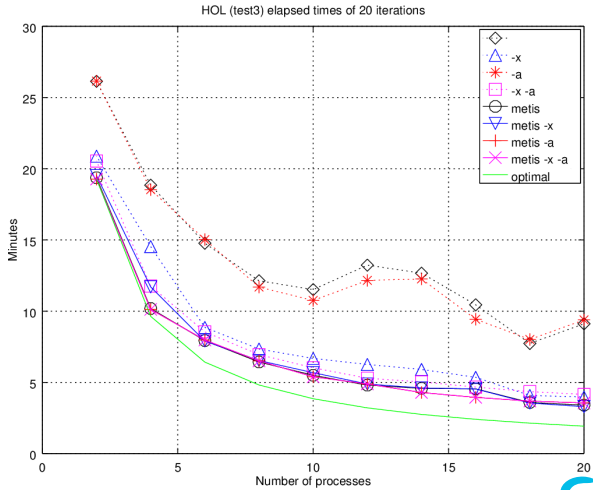
- Reordering and repartitioning minimized communication.
- Elapsed time (of 20 iterations) decreased from 9 minutes down to 4 minutes (with 20 processes).



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
```

Hybrid multi-threaded MPI parallelism with `mix99p`

MiX99 "parallel" solver `mix99p` can also utilize **hybrid parallelism**:

- Both **MPI multi-processing** and **multi-threading**.
- `mix99p` from `mp` subdir.
- Ideal operation:
  - Sparse operations: all MPI processes in single-thread.
  - Dense operations: one MPI process in multi-thread.
- Memory options affect here.

In `mix99p`:

1) no single-step model: all processes use the same number of threads given by the command.
2) single-step method with the low memory option "MES" is used: the same as for the no single-step model.
3) single-step with memory options "MEL"/"MEB"/"MEM": only the master process will use the number of threads given, all other processes use only one CPU.

The command line option is like:

```
mpiexec -np 4 mix99p -nt 10 -MEL -s
```

which uses 4 processes in MPI but 10 CPU threads for the master process. When the solver commands are in a file, the "nt" command is on the first line in format like "nt 10".

The logic behind the differences is that option "MEL"/"MEB"/"MEM" lead to the Master process to have an additional computational work that will benefit from multi-threading. However, instructing all the other processes to have as many threads may lead to use of too many threads and inefficient computations. Because this may be often the case, the optimum number of threads in non single-step or "MES" option can be small.

Choosing MiX99 solver

- If performing multiple evaluations concurrently:
  - Single-thread solver or
  - multi-threaded/MPI solver and limiting number of threads/processes.
- For large dense matrices are involved: use multi-threading.
- For large sparse models: data parallelism with `mix99p`.

	large dense	large sparse	both
Multi-thread `mix99s`	✓		
`mix99p`		✓	
Multi-thread `mix99p`			✓

Quick peek on current high-performance parallel computing hardware 2026

Luke
NATURAL RESOURCES INSTITUTE FINLAND

State-of-the-art parallel computing hardware 2026

- 5th Gen AMD EPYC servers (9005 series):
  - Up to 192 computational cores, 384 threads.
  - Up to 6 TB memory, 12 channels.
- NVIDIA RTX PRO 6000 GPUs:
  - 24,064 CUDA cores.
  - 96 GB memory.
- NVIDIA B300 accelerators:
  - 32,768 CUDA cores?
  - 288 GB memory.
- GPU prices have been high and now also memory and storage prices.

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.
- 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.
- Basically how MiX99 parallel solver `mix99p` is already operating:
  - So, similar practices may need to be implemented also on the "serial" solver `mix99s` side in the future.

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.
- Basically how MiX99 parallel solver `mix99p` is already operating:
  - So, similar practices may need to be implemented also on the "serial" solver `mix99s` side in the future.
- Similarly GPU/accelerators have much faster local GPU memory access compared to their global CPU memory access speeds.

Looking to the future: GPU/accelerator

- GPUs and accelerators are still best at:
  - Specialized non-general processing.
  - Limited floating point precision calculations.

Looking to the future: GPU/accelerator

- GPUs and accelerators are still best at:
  - Specialized non-general processing.
  - Limited floating point precision calculations.
- They are, however, well suited for, for example, processing genomic data.

Looking to the future: GPU/accelerator

- GPUs and accelerators are still best at:
  - Specialized non-general processing.
  - Limited floating point precision calculations.
- They are, however, well suited for, for example, processing genomic data.
- MiX99 does not currently support GPUs or other accelerators.

Quick poll on GPU/accelerator usage/interest

Raise your hand if:

1) **Your group already has GPU/accelerator hardware or has used them in, for example, cloud services.**

Quick poll on GPU/accelerator usage/interest

Raise your hand if:

2) **Your group is going to purchase GPU/accelerator hardware or is planning to use them in, for example, cloud services.**

Quick poll on GPU/accelerator usage/interest

Raise your hand if:

3) You would like MiX99 to have GPU/accelerator support.

Conclusions

- We looked how MiX99 both "serial" and "parallel" solver `mix99s` and `mix99p` utilize parallel computing.
- What affects the performance of parallelization and how to monitor it.
- How to choose MiX99 solver.


Luke
NATURAL RESOURCES
INSTITUTE FINLAND