

MiX99

Solving Large Mixed Model Equations



MTT

Trends in computing environments

Matti Taskinen, MTT

4.12.2014 MiX99 Workshop, Tuusula, Finland

Overview

- MiX99 has been in development for about 15 years now.
- How MiX99 is going to look like after next 15 years?
 - Hardware
 - Software
 - User interface

Overview

- MiX99 has been in development for about 15 years now.
- How MiX99 is going to look like after next 15 years?
 - Hardware
 - Software
 - User interface

Overview

- MiX99 has been in development for about 15 years now.
- How MiX99 is going to look like after next 15 years?
 - Hardware
 - Software
 - User interface

Overview

- MiX99 has been in development for about 15 years now.
- How MiX99 is going to look like after next 15 years?
 - Hardware
 - Software
 - User interface

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (optional) hard drive, 100 GB
 - 100-400 MB/s speed
 - Speed: 10-1000 gigabytes/second (limited partially by speed)
 - HDD (rotation On/Off) needs large memory as fast data storage
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made routines
 - Some use of BLAS/Lapack/Lapacke/Arpack
 - Parallel computing:
 - Parallel MPI version of the software
 - Acceleration by (Fortran) compiler
- User interface:
 - Data prepared / results analysed:
 - Excel/matrix, R/DOEwin, VAP
 - Fortran made routines, Rplots, Rplots2, ...
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs **large memory or fast data storage**.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Some home-made interfaces
 - Some use of BLAS/LAPACK/LinearAlgebra
 - Parallel computing:
 - Some use of MPI
 - Some use of OpenMP
 - Some use of Fortran compiler
- User interface:
 - Data prepared / results analysed:
 - Some use of R/Python/Matlab
 - Some use of R/Python/Julia
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs large memory or fast data storage.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made routines
 - Some use of BLAS/LAPACK/LinearAlgebra
 - Parallel computing:
 - Parallel Fortran 90 of the 90's
 - Acceleration by (Fortran) compiler
- User interface:
 - Data prepared / results analysed:
 - Spreadsheet, Fortran, C++
 - Home-made routines, R, RStudio
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs large memory or fast data storage.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made routines
 - Some use of BLAS/LAPACK/ARPACK/FFTW
 - Parallel computing:
 - Shared memory (SMP) or the old-fashioned message passing (Fortran) compiler
- User interface:
 - Data prepared / results analysed:
 - Spreadsheet software (Excel)
 - Home-made routines (Fortran/Python)
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs **large memory or fast data storage**.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made interfaces
 - Some use of BLAS/LAPACK/LinearAlgebra
 - Parallel computing:
 - Shared memory (SMP) and the MPI
 - Message Passing Interface (MPI) compiler
- User interface:
 - Data prepared / results analysed:
 - Fortran 90, Fortran 95, C
 - Fortran 90, Fortran 95, Perl, Python, C
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs **large memory or fast data storage**.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made interfaces
 - Some use of BLAS/LAPACK/Lapacke/Intel MKL
 - Parallel computing:
 - Fortran 90/95 and the MPI
 - OpenMP (Fortran) compiler
- User interface:
 - Data prepared / results analysed:
 - Fortran 90/95, C, Python, R
 - Fortran 90/95, R, Perl, JavaScript
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs **large memory or fast data storage**.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made libraries
 - Some use of BLAS/LAPACK/ARPACK/FFTW
 - Parallel computing:
 - Fortran 90/95 and the MPI
 - OpenMP (Fortran) compiler
- User interface:
 - Data prepared / results analysed:
 - Fortran 90/95/Python/C++
 - Fortran 90/95/Python/Perl/Java
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs **large memory or fast data storage**.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - LAPACK, BLAS, ScaLAPACK, LAPACK++
 - BLACS, PBLACS, ScaLAPACK, ScaLAPACK++
 - MPI
 - Parallel computing:
 - Shared memory (ScaLAPACK) or message passing (MPI)
- User interface:
 - Data prepared / results analysed:
 - Fortran 90
 - Fortran 90
 - Fortran 90
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs **large memory or fast data storage**.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made routines
 - Some use of BLAS/Linpack/Lapack/Eispack
 - Parallel computing:
 - Separate MPI version of the solver
 - Vectorization by (Fortran) compiler
- User interface:
 - Data prepared / results analysed:
 - Fortran 90
 - Fortran 90
 - Fortran 90
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs **large memory or fast data storage**.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made routines
 - Some use of BLAS/Linpack/Lapack/Eispack
 - Parallel computing:
 - Separate MPI version of the solver
 - Vectorization by (Fortran) compiler
- User interface:
 - Data prepared / results analysed:
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs **large memory or fast data storage**.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made routines
 - Some use of BLAS/Linpack/Lapack/Eispack
 - Parallel computing:
 - Separate MPI version of the solver
 - Vectorization by (Fortran) compiler
- User interface:
 - Data prepared / results analysed:
 - Home-made Fortran programs
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs **large memory or fast data storage**.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made routines
 - Some use of BLAS/Linpack/Lapack/Eispack
 - Parallel computing:
 - Separate MPI version of the solver
 - Vectorization by (Fortran) compiler
- User interface:
 - Data prepared / results analysed:
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs **large memory or fast data storage**.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made routines
 - Some use of BLAS/Linpack/Lapack/Eispack
 - Parallel computing:
 - Separate MPI version of the solver
 - Vectorization by (Fortran) compiler
- User interface:
 - Data prepared / results analysed:
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs **large memory or fast data storage**.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made routines
 - Some use of BLAS/Linpack/Lapack/Eispack
 - Parallel computing:
 - Separate MPI version of the solver
 - Vectorization by (Fortran) compiler
- User interface:
 - Data prepared / results analysed:
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs **large memory or fast data storage**.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made routines
 - Some use of BLAS/Linpack/Lapack/Eispack
 - Parallel computing:
 - Separate MPI version of the solver
 - Vectorization by (Fortran) compiler
- User interface:
 - Data prepared / results analysed:
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs **large memory or fast data storage**.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made routines
 - Some use of BLAS/Linpack/Lapack/Eispack
 - Parallel computing:
 - Separate MPI version of the solver
 - Vectorization by (Fortran) compiler
- User interface:
 - Data prepared / results analysed:
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs **large memory or fast data storage**.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made routines
 - Some use of BLAS/Linpack/Lapack/Eispack
 - Parallel computing:
 - Separate MPI version of the solver
 - Vectorization by (Fortran) compiler
- User interface:
 - Data prepared / results analysed:
 - External tools: R/Octave/SAS/...
 - Home-made routines: RelaX2/HGinv/...
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs **large memory or fast data storage**.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made routines
 - Some use of BLAS/Linpack/Lapack/Eispack
 - Parallel computing:
 - Separate MPI version of the solver
 - Vectorization by (Fortran) compiler
- User interface:
 - Data prepared / results analysed:
 - External tools: R/Octave/SAS/...
 - Home-made routines: RelaX2/HGinv/...
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs **large memory or fast data storage**.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made routines
 - Some use of BLAS/Linpack/Lapack/Eispack
 - Parallel computing:
 - Separate MPI version of the solver
 - Vectorization by (Fortran) compiler
- User interface:
 - Data prepared / results analysed:
 - External tools: R/Octave/SAS/...
 - Home-made routines: RelaX2/HGinv/...
 - Home-made command languages

MiX99 currently

- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs **large memory or fast data storage**.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made routines
 - Some use of BLAS/Linpack/Lapack/Eispack
 - Parallel computing:
 - Separate MPI version of the solver
 - Vectorization by (Fortran) compiler
- User interface:
 - Data prepared / results analysed:
 - External tools: R/Octave/SAS/...
 - Home-made routines: RelaX2/HGinv/...
 - Home-made command languages

MiX99 currently

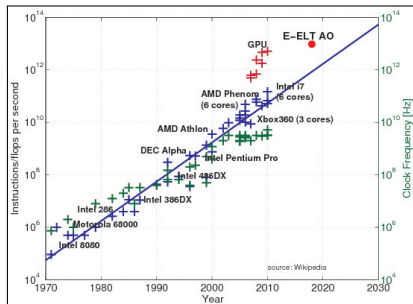
- Target hardware:
 - CPU: Intel/AMD 64-bit x86
 - Memory: 1-20 GB
 - Data storage:
 - (Spinning) hard drives
 - 100-200 MB/s speed
 - Space: 10-1000 gigabytes (limited partially by speed)
 - IOD (Iteration On Data) needs **large memory or fast data storage**.
- Software:
 - Programming language: Fortran 90
 - Software libraries:
 - Home-made routines
 - Some use of BLAS/Linpack/Lapack/Eispack
 - Parallel computing:
 - Separate MPI version of the solver
 - Vectorization by (Fortran) compiler
- User interface:
 - Data prepared / results analysed:
 - External tools: R/Octave/SAS/...
 - Home-made routines: RelaX2/HGinv/...
 - Home-made command languages

Hardware: CPU

- CPU “speed” has been increasing steadily
- Intel/AMD x86 seem to have won the race as general purpose CPU
- CPU frequencies does not seem to be increasing
- Other optimizations still making CPUs faster
- Number of processing units (multi-core) is increasing
 - Commonly: 4-12 cores
- “Special purpose” processing units exploit parallelism:
 - GPU / Cell / Xeon Phi
 - Up to thousands of cores
 - Order of magnitude “faster” (theoretically)
- ⇒ **Multi-core parallel computing**
seems to be essential

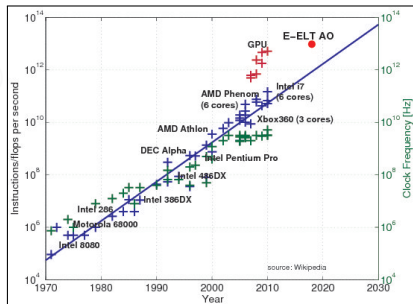
Hardware: CPU

- CPU “speed” has been increasing steadily
- Intel/AMD x86 seem to have won the race as general purpose CPU
- CPU frequencies does not seem to be increasing
- Other optimizations still making CPUs faster
- Number of processing units (multi-core) is increasing
 - Commonly: 4-12 cores
- “Special purpose” processing units exploit parallelism:
 - GPU / Cell / Xeon Phi
 - Up to thousands of cores
 - Order of magnitude “faster” (theoretically)
- ⇒ **Multi-core parallel computing** seems to be essential



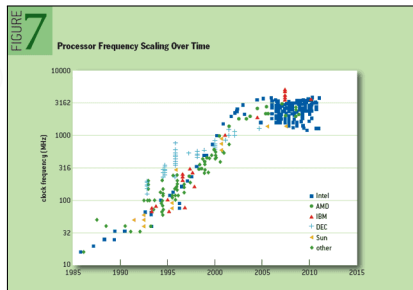
Hardware: CPU

- CPU “speed” has been increasing steadily
- Intel/AMD x86 seem to have won the race as general purpose CPU
 - CPU frequencies does not seem to be increasing
 - Other optimizations still making CPUs faster
 - Number of processing units (multi-core) is increasing
 - Commonly: 4-12 cores
- “Special purpose” processing units exploit parallelism:
 - GPU / Cell / Xeon Phi
 - Up to thousands of cores
 - Order of magnitude “faster” (theoretically)
- ⇒ **Multi-core parallel computing** seems to be essential



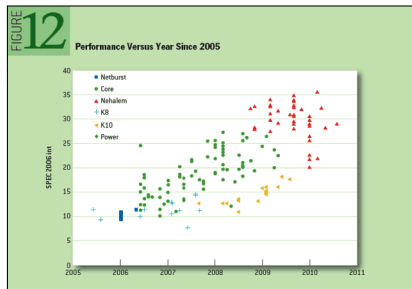
Hardware: CPU

- CPU “speed” has been increasing steadily
- Intel/AMD x86 seem to have won the race as general purpose CPU
- CPU frequencies does not seem to be increasing
- Other optimizations still making CPUs faster
- Number of processing units (multi-core) is increasing
 - Commonly: 4-12 cores
- “Special purpose” processing units exploit parallelism:
 - GPU / Cell / Xeon Phi
 - Up to thousands of cores
 - Order of magnitude “faster” (theoretically)
- ⇒ **Multi-core parallel computing** seems to be essential



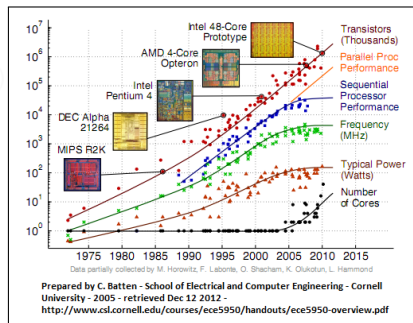
Hardware: CPU

- CPU “speed” has been increasing steadily
- Intel/AMD x86 seem to have won the race as general purpose CPU
- CPU frequencies does not seem to be increasing
- Other optimizations still making CPUs faster
- Number of processing units (multi-core) is increasing
 - Commonly: 4-12 cores
- “Special purpose” processing units exploit parallelism:
 - GPU / Cell / Xeon Phi
 - Up to thousands of cores
 - Order of magnitude “faster” (theoretically)
- ⇒ **Multi-core parallel computing** seems to be essential



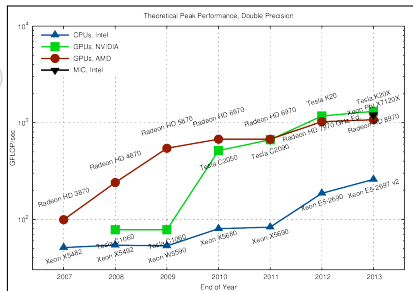
Hardware: CPU

- CPU “speed” has been increasing steadily
- Intel/AMD x86 seem to have won the race as general purpose CPU
- CPU frequencies does not seem to be increasing
- Other optimizations still making CPUs faster
- Number of processing units (multi-core) is increasing
 - Commonly: 4-12 cores
- “Special purpose” processing units exploit parallelism:
 - GPU / Cell / Xeon Phi
 - Up to thousands of cores
 - Order of magnitude “faster” (theoretically)
- ⇒ **Multi-core parallel computing** seems to be essential



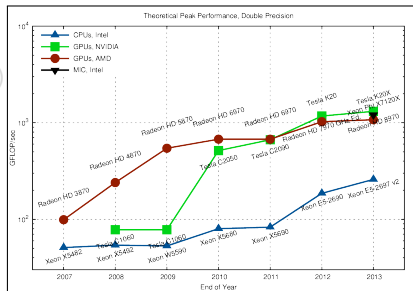
Hardware: CPU

- CPU “speed” has been increasing steadily
- Intel/AMD x86 seem to have won the race as general purpose CPU
- CPU frequencies does not seem to be increasing
- Other optimizations still making CPUs faster
- Number of processing units (multi-core) is increasing
 - Commonly: 4-12 cores
- “Special purpose” processing units exploit parallelism:
 - GPU / Cell / Xeon Phi
 - Up to thousands of cores
 - Order of magnitude “faster” (theoretically)
- ⇒ **Multi-core parallel computing** seems to be essential



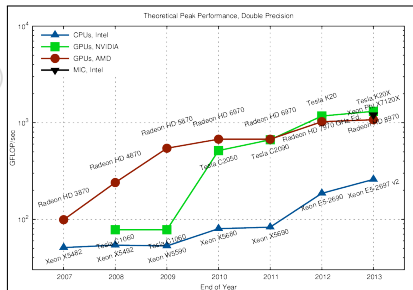
Hardware: CPU

- CPU “speed” has been increasing steadily
- Intel/AMD x86 seem to have won the race as general purpose CPU
- CPU frequencies does not seem to be increasing
- Other optimizations still making CPUs faster
- Number of processing units (multi-core) is increasing
 - Commonly: 4-12 cores
- “Special purpose” processing units exploit parallelism:
 - GPU / Cell / Xeon Phi
 - Up to thousands of cores
 - Order of magnitude “faster” (theoretically)
- ⇒ **Multi-core parallel computing** seems to be essential



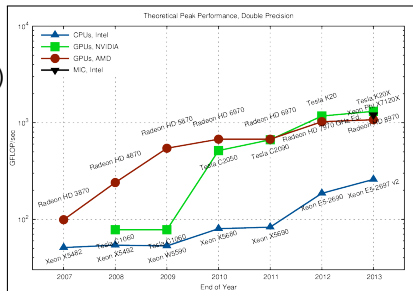
Hardware: CPU

- CPU “speed” has been increasing steadily
- Intel/AMD x86 seem to have won the race as general purpose CPU
- CPU frequencies does not seem to be increasing
- Other optimizations still making CPUs faster
- Number of processing units (multi-core) is increasing
 - Commonly: 4-12 cores
- “Special purpose” processing units exploit parallelism:
 - GPU / Cell / Xeon Phi
 - Up to thousands of cores
 - Order of magnitude “faster” (theoretically)
- ⇒ **Multi-core parallel computing** seems to be essential



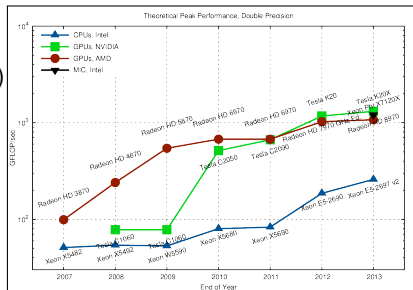
Hardware: CPU

- CPU “speed” has been increasing steadily
- Intel/AMD x86 seem to have won the race as general purpose CPU
- CPU frequencies does not seem to be increasing
- Other optimizations still making CPUs faster
- Number of processing units (multi-core) is increasing
 - Commonly: 4-12 cores
- “Special purpose” processing units exploit parallelism:
 - GPU / Cell / Xeon Phi
 - Up to thousands of cores
 - Order of magnitude “faster” (theoretically)
- ⇒ Multi-core parallel computing seems to be essential



Hardware: CPU

- CPU “speed” has been increasing steadily
- Intel/AMD x86 seem to have won the race as general purpose CPU
- CPU frequencies does not seem to be increasing
- Other optimizations still making CPUs faster
- Number of processing units (multi-core) is increasing
 - Commonly: 4-12 cores
- “Special purpose” processing units exploit parallelism:
 - GPU / Cell / Xeon Phi
 - Up to thousands of cores
 - Order of magnitude “faster” (theoretically)
- ⇒ **Multi-core parallel computing** seems to be essential



Hardware: Large memory or fast data storage

- For large genomic models very large matrices are needed to be stored in the memory or in the data storage
- Currently, up to 2 terabyte (CPU) memory possible:
 - Up to ten millions $\times$ ten millions (symmetric single-precision) matrix can be stored in memory
 - Can be increased by using **distributed memory** ($\Rightarrow$ MPI)
- **Faster data storage** could compensate smaller memory
 - (Spinning) hard drives are **cheap but slow**
 - It takes 3 hours to load 1TB file (100 MB/s)
 - **SSDs currently 3-5 times faster**
 - SSD speed will further increase
 - SSDs are **expensive**

Hardware: Large memory or fast data storage

- For large genomic models very large matrices are needed to be stored in the memory or in the data storage
- Currently, up to 2 terabyte (CPU) memory possible:
 - Up to ten millions $\times$ ten millions (symmetric single-precision) matrix can be stored in memory
 - Can be increased by using **distributed memory** ($\Rightarrow$ MPI)
- **Faster data storage** could compensate smaller memory
 - (Spinning) hard drives are **cheap but slow**
 - It takes 3 hours to load 1TB file (100 MB/s)
 - **SSDs currently 3-5 times faster**
 - SSD speed will further increase
 - SSDs are **expensive**

Hardware: Large memory or fast data storage

- For large genomic models very large matrices are needed to be stored in the memory or in the data storage
- Currently, up to 2 terabyte (CPU) memory possible:
 - Up to ten millions $\times$ ten millions (symmetric single-precision) matrix can be stored in memory
 - Can be increased by using **distributed memory** ($\Rightarrow$ MPI)
- **Faster data storage** could compensate smaller memory
 - (Spinning) hard drives are cheap but slow
 - It takes 3 hours to load 1TB file (100 MB/s)
 - SSDs currently 3-5 times faster
 - SSD speed will further increase
 - SSDs are expensive

Hardware: Large memory or fast data storage

- For large genomic models very large matrices are needed to be stored in the memory or in the data storage
- Currently, up to 2 terabyte (CPU) memory possible:
 - Up to ten millions $\times$ ten millions (symmetric single-precision) matrix can be stored in memory
 - Can be increased by using **distributed memory** ($\Rightarrow$ MPI)
- **Faster data storage** could compensate smaller memory
 - (Spinning) hard drives are **cheap but slow**
 - It takes 3 hours to load 1TB file (100 MB/s)
 - **SSDs currently 3-5 times faster**
 - SSD speed will further increase
 - SSDs are **expensive**

Hardware: Large memory or fast data storage

- For large genomic models very large matrices are needed to be stored in the memory or in the data storage
- Currently, up to 2 terabyte (CPU) memory possible:
 - Up to ten millions $\times$ ten millions (symmetric single-precision) matrix can be stored in memory
 - Can be increased by using **distributed memory** ($\Rightarrow$ MPI)
- **Faster data storage** could compensate smaller memory
 - (Spinning) hard drives are **cheap but slow**
 - It takes 3 hours to load 1TB file (100 MB/s)
 - **SSDs currently 3-5 times faster**
 - SSD speed will further increase
 - SSDs are **expensive**

Hardware: Large memory or fast data storage

- For large genomic models very large matrices are needed to be stored in the memory or in the data storage
- Currently, up to 2 terabyte (CPU) memory possible:
 - Up to ten millions $\times$ ten millions (symmetric single-precision) matrix can be stored in memory
 - Can be increased by using **distributed memory** ($\Rightarrow$ MPI)
- **Faster data storage** could compensate smaller memory
 - (Spinning) hard drives are **cheap but slow**
 - It takes 3 hours to load 1TB file (100 MB/s)
 - **SSDs currently 3-5 times faster**
 - SSD speed will further increase
 - SSDs are **expensive**

Hardware: Large memory or fast data storage

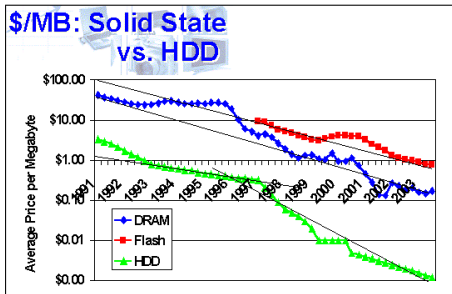
- For large genomic models very large matrices are needed to be stored in the memory or in the data storage
- Currently, up to 2 terabyte (CPU) memory possible:
 - Up to ten millions $\times$ ten millions (symmetric single-precision) matrix can be stored in memory
 - Can be increased by using **distributed memory** ($\Rightarrow$ MPI)
- **Faster data storage** could compensate smaller memory
 - (Spinning) hard drives are **cheap but slow**
 - It takes 3 hours to load 1TB file (100 MB/s)
 - **SSDs currently 3-5 times faster**
 - SSD speed will further increase
 - SSDs are **expensive**

Hardware: Large memory or fast data storage

- For large genomic models very large matrices are needed to be stored in the memory or in the data storage
- Currently, up to 2 terabyte (CPU) memory possible:
 - Up to ten millions $\times$ ten millions (symmetric single-precision) matrix can be stored in memory
 - Can be increased by using **distributed memory** ($\Rightarrow$ MPI)
- **Faster data storage** could compensate smaller memory
 - (Spinning) hard drives are **cheap but slow**
 - It takes 3 hours to load 1TB file (100 MB/s)
 - **SSDs currently 3-5 times faster**
 - SSD speed will further increase
 - SSDs are **expensive**

Hardware: Large memory or fast data storage

- For large genomic models very large matrices are needed to be stored in the memory or in the data storage
- Currently, up to 2 terabyte (CPU) memory possible:
 - Up to ten millions $\times$ ten millions (symmetric single-precision) matrix can be stored in memory
 - Can be increased by using **distributed memory** ($\Rightarrow$ MPI)
- **Faster data storage** could compensate smaller memory
 - (Spinning) hard drives are **cheap but slow**
 - It takes 3 hours to load 1TB file (100 MB/s)
 - **SSDs currently 3-5 times faster**
 - SSD speed will further increase
 - SSDs are **expensive**



Software: programming language

- MiX99 is programmed with Fortran 90
- Fortran evolves:
 - F66, F77, F90, F95, F2003, F2008, F2015
 - Vectorization, parallel execution
 - Object oriented programming
 - Interoperability with C
- Should be consider other languages?
 - C/C++
 - Java
 - .NET, C#, F#
 - Python
 - Go

Software: programming language

- MiX99 is programmed with Fortran 90
- Fortran evolves:
 - F66, F77, F90, F95, F2003, F2008, F2015
 - Vectorization, parallel execution
 - Object oriented programming
 - Interoperability with C
- Should be consider other languages?
 - C/C++
 - Java
 - .NET, C#, F#
 - Python
 - Go

Software: programming language

- MiX99 is programmed with Fortran 90
- Fortran evolves:
 - F66, F77, F90, F95, F2003, F2008, F2015
 - Vectorization, parallel execution
 - Object oriented programming
 - Interoperability with C
- Should be consider other languages?
 - C/C++
 - Java
 - .NET, C#, F#
 - Python
 - Go

Software: programming language

- MiX99 is programmed with Fortran 90
- Fortran evolves:
 - F66, F77, F90, F95, F2003, F2008, F2015
 - Vectorization, parallel execution
 - Object oriented programming
 - Interoperability with C
- Should be consider other languages?
 - C/C++
 - Java
 - .NET, C#, F#
 - Python
 - Go

Software: programming language

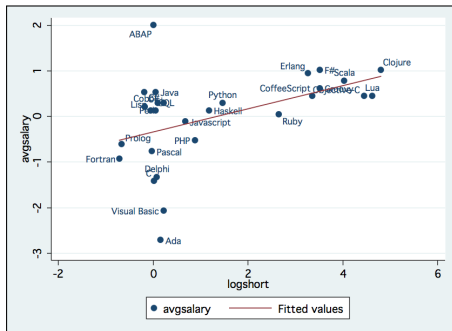
- MiX99 is programmed with Fortran 90
- Fortran evolves:
 - F66, F77, F90, F95, F2003, F2008, F2015
 - Vectorization, parallel execution
 - Object oriented programming
 - Interoperability with C
- Should be consider other languages?
 - C/C++
 - Java
 - .NET, C#, F#
 - Python
 - Go

Software: programming language

- MiX99 is programmed with Fortran 90
- Fortran evolves:
 - F66, F77, F90, F95, F2003, F2008, F2015
 - Vectorization, parallel execution
 - Object oriented programming
 - Interoperability with C
- Should be consider other languages?
 - C/C++
 - Java
 - .NET, C#, F#
 - Python
 - Go

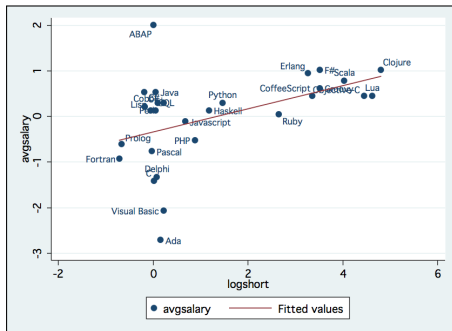
Software: programming language

- MiX99 is programmed with Fortran 90
- Fortran evolves:
 - F66, F77, F90, F95, F2003, F2008, F2015
 - Vectorization, parallel execution
 - Object oriented programming
 - Interoperability with C
- Should we consider other languages?
 - C/C++
 - Java
 - .NET, C#, F#
 - Python
 - Go



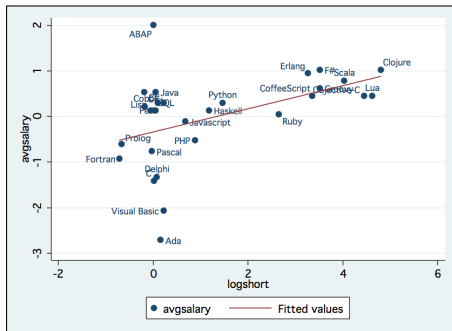
Software: programming language

- MiX99 is programmed with Fortran 90
- Fortran evolves:
 - F66, F77, F90, F95, F2003, F2008, F2015
 - Vectorization, parallel execution
 - Object oriented programming
 - Interoperability with C
- Should be consider other languages?
 - C/C++
 - Java
 - .NET, C#, F#
 - Python
 - Go



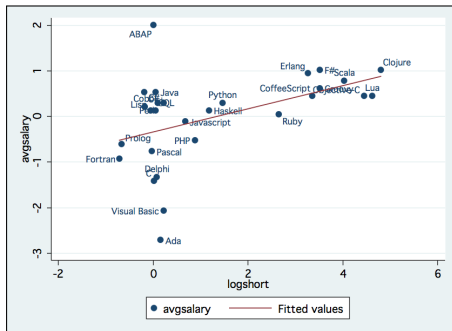
Software: programming language

- MiX99 is programmed with Fortran 90
- Fortran evolves:
 - F66, F77, F90, F95, F2003, F2008, F2015
 - Vectorization, parallel execution
 - Object oriented programming
 - Interoperability with C
- Should be consider other languages?
 - C/C++
 - Java
 - .NET, C#, F#
 - Python
 - Go



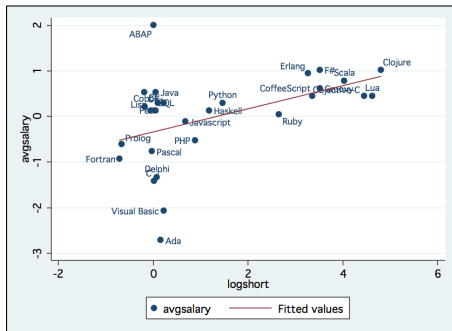
Software: programming language

- MiX99 is programmed with Fortran 90
- Fortran evolves:
 - F66, F77, F90, F95, F2003, F2008, F2015
 - Vectorization, parallel execution
 - Object oriented programming
 - Interoperability with C
- Should be consider other languages?
 - C/C++
 - Java
 - .NET, C#, F#
 - Python
 - Go



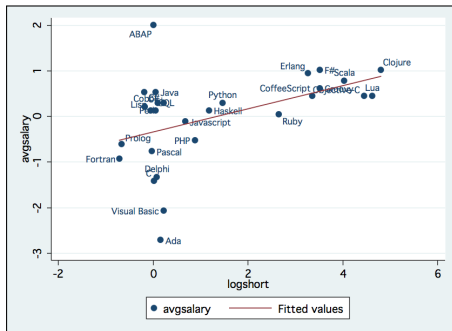
Software: programming language

- MiX99 is programmed with Fortran 90
- Fortran evolves:
 - F66, F77, F90, F95, F2003, F2008, F2015
 - Vectorization, parallel execution
 - Object oriented programming
 - Interoperability with C
- Should be consider other languages?
 - C/C++
 - Java
 - .NET, C#, F#
 - Python
 - Go



Software: programming language

- MiX99 is programmed with Fortran 90
- Fortran evolves:
 - F66, F77, F90, F95, F2003, F2008, F2015
 - Vectorization, parallel execution
 - Object oriented programming
 - Interoperability with C
- Should be consider other languages?
 - C/C++
 - Java
 - .NET, C#, F#
 - Python
 - Go



Parallel computing

- MiX99 uses currently parallel computing in separate parallel version of the solver
 - MPI, Distributed memory model
- There are different levels of parallelism:
 - Instruction level parallelism (ILP)
 - Thread/task level parallelism (TLP)
 - Data level parallelism (DLP)
- Current MiX99 code could use parallel computing “automatically”:
 - Smart compilers use ILP on multi-core CPU (SIMD, SSE, AVX)
- Existing subroutines could be replaced with parallel implementations using external libraries
 - BLAS/Linpack/Lapack/...
 - MKL
 - NAG/IMSL
 - Blitz++/Boost/PETSc/...
- Other parallel computing methods could be used:
 - OpenMP, OpenCL
 - Cuda

Parallel computing

- MiX99 uses currently parallel computing in separate parallel version of the solver
 - MPI, Distributed memory model
- There are different levels of parallelism:
 - Instruction level parallelism (ILP)
 - Thread/task level parallelism (TLP)
 - Data level parallelism (DLP)
- Current MiX99 code could use parallel computing “automatically”:
 - Smart compilers use ILP on multi-core CPU (SIMD, SSE, AVX)
- Existing subroutines could be replaced with parallel implementations using external libraries
 - BLAS/Linpack/Lapack/...
 - MKL
 - NAG/IMSL
 - Blitz++/Boost/PETSc/...
- Other parallel computing methods could be used:
 - OpenMP, OpenCL
 - Cuda

Parallel computing

- MiX99 uses currently parallel computing in separate parallel version of the solver
 - MPI, Distributed memory model
- There are different levels of parallelism:
 - Instruction level parallelism (ILP)
 - Thread/task level parallelism (TLP)
 - Data level parallelism (DLP)
- Current MiX99 code could use parallel computing “automatically”:
 - Smart compilers use ILP on multi-core CPU (SIMD, SSE, AVX)
- Existing subroutines could be replaced with parallel implementations using external libraries
 - BLAS/Linpack/Lapack/...
 - MKL
 - NAG/IMSL
 - Blitz++/Boost/PETSc/...
- Other parallel computing methods could be used:
 - OpenMP, OpenCL
 - Cuda

Parallel computing

- MiX99 uses currently parallel computing in separate parallel version of the solver
 - MPI, Distributed memory model
- There are different levels of parallelism:
 - Instruction level parallelism (ILP)
 - Thread/task level parallelism (TLP)
 - Data level parallelism (DLP)
- Current MiX99 code could use parallel computing “automatically”:
 - Smart compilers use ILP on multi-core CPU (SIMD, SSE, AVX)
- Existing subroutines could be replaced with parallel implementations using external libraries
 - BLAS/Linpack/Lapack/...
 - MKL
 - NAG/IMSL
 - Blitz++/Boost/PETSc/...
- Other parallel computing methods could be used:
 - OpenMP, OpenCL
 - Cuda

Parallel computing

- MiX99 uses currently parallel computing in separate parallel version of the solver
 - MPI, Distributed memory model
- There are different levels of parallelism:
 - Instruction level parallelism (ILP)
 - Thread/task level parallelism (TLP)
 - Data level parallelism (DLP)
- Current MiX99 code could use parallel computing “automatically”:
 - Smart compilers use ILP on multi-core CPU (SIMD, SSE, AVX)
- Existing subroutines could be replaced with parallel implementations using external libraries
 - BLAS/Linpack/Lapack/...
 - MKL
 - NAG/IMSL
 - Blitz++/Boost/PETSc/...
- Other parallel computing methods could be used:
 - OpenMP, OpenCL
 - Cuda

Parallel computing

- MiX99 uses currently parallel computing in separate parallel version of the solver
 - MPI, Distributed memory model
- There are different levels of parallelism:
 - Instruction level parallelism (ILP)
 - Thread/task level parallelism (TLP)
 - Data level parallelism (DLP)
- Current MiX99 code could use parallel computing “automatically”:
 - Smart compilers use ILP on multi-core CPU (SIMD, SSE, AVX)
- Existing subroutines could be replaced with parallel implementations using external libraries
 - BLAS/Linpack/Lapack/...
 - MKL
 - NAG/IMSL
 - Blitz++/Boost/PETSc/...
- Other parallel computing methods could be used:
 - OpenMP, OpenCL
 - Cuda

Parallel computing

- MiX99 uses currently parallel computing in separate parallel version of the solver
 - MPI, Distributed memory model
- There are different levels of parallelism:
 - Instruction level parallelism (ILP)
 - Thread/task level parallelism (TLP)
 - Data level parallelism (DLP)
- Current MiX99 code could use parallel computing “automatically”:
 - Smart compilers use ILP on multi-core CPU (SIMD, SSE, AVX)
- Existing subroutines could be replaced with parallel implementations using external libraries
 - BLAS/Linpack/Lapack/...
 - MKL
 - NAG/IMSL
 - Blitz++/Boost/PETSc/...
- Other parallel computing methods could be used:
 - OpenMP, OpenCL
 - Cuda

Parallel computing

- MiX99 uses currently parallel computing in separate parallel version of the solver
 - MPI, Distributed memory model
- There are different levels of parallelism:
 - Instruction level parallelism (ILP)
 - Thread/task level parallelism (TLP)
 - Data level parallelism (DLP)
- Current MiX99 code could use parallel computing “automatically”:
 - Smart compilers use ILP on multi-core CPU (SIMD, SSE, AVX)
- Existing subroutines could be replaced with parallel implementations using external libraries
 - BLAS/Linpack/Lapack/...
 - MKL
 - NAG/IMSL
 - Blitz++/Boost/PETSc/...
- Other parallel computing methods could be used:
 - OpenMP, OpenCL
 - Cuda

Parallel computing

- MiX99 uses currently parallel computing in separate parallel version of the solver
 - MPI, Distributed memory model
- There are different levels of parallelism:
 - Instruction level parallelism (ILP)
 - Thread/task level parallelism (TLP)
 - Data level parallelism (DLP)
- Current MiX99 code could use parallel computing “automatically”:
 - Smart compilers use ILP on multi-core CPU (SIMD, SSE, AVX)
- Existing subroutines could be replaced with parallel implementations using external libraries
 - BLAS/Linpack/Lapack/...
 - MKL
 - NAG/IMSL
 - Blitz++/Boost/PETSc/...
- Other parallel computing methods could be used:
 - OpenMP, OpenCL
 - Cuda

Parallel computing

- MiX99 uses currently parallel computing in separate parallel version of the solver
 - MPI, Distributed memory model
- There are different levels of parallelism:
 - Instruction level parallelism (ILP)
 - Thread/task level parallelism (TLP)
 - Data level parallelism (DLP)
- Current MiX99 code could use parallel computing “automatically”:
 - Smart compilers use ILP on multi-core CPU (SIMD, SSE, AVX)
- Existing subroutines could be replaced with parallel implementations using external libraries
 - BLAS/Linpack/Lapack/...
 - MKL
 - NAG/IMSL
 - Blitz++/Boost/PETSc/...
- Other parallel computing methods could be used:
 - OpenMP, OpenCL
 - Cuda

Parallel computing

- MiX99 uses currently parallel computing in separate parallel version of the solver
 - MPI, Distributed memory model
- There are different levels of parallelism:
 - Instruction level parallelism (ILP)
 - Thread/task level parallelism (TLP)
 - Data level parallelism (DLP)
- Current MiX99 code could use parallel computing “automatically”:
 - Smart compilers use ILP on multi-core CPU (SIMD, SSE, AVX)
- Existing subroutines could be replaced with parallel implementations using external libraries
 - BLAS/Linpack/Lapack/...
 - MKL
 - NAG/IMSL
 - Blitz++/Boost/PETSc/...
- Other parallel computing methods could be used:
 - OpenMP, OpenCL
 - Cuda

Parallel computing

- MiX99 uses currently parallel computing in separate parallel version of the solver
 - MPI, Distributed memory model
- There are different levels of parallelism:
 - Instruction level parallelism (ILP)
 - Thread/task level parallelism (TLP)
 - Data level parallelism (DLP)
- Current MiX99 code could use parallel computing “automatically”:
 - Smart compilers use ILP on multi-core CPU (SIMD, SSE, AVX)
- Existing subroutines could be replaced with parallel implementations using external libraries
 - BLAS/Linpack/Lapack/...
 - MKL
 - NAG/IMSL
 - Blitz++/Boost/PETSc/...
- Other parallel computing methods could be used:
 - OpenMP, OpenCL
 - Cuda

User interface

- Command language:
 - MIX/CLIM
 - Embedded language: Lua/Python/...
- Embedding MiX99:
 - Octace/R
- Operating system
 - Linux/Windows/MacOS
 - Android
- Desktop/Mobile/Tablet
- Cloud computing
 - Grid computing
 - OpenStack
 - Docker