Efficient parallel solvers for large dense systems of linear interval equations

Mariana Kolberg¹, Walter Krämer², Michael Zimmer²

¹Pontifícia Universidade Católica do Rio Grande do Sul, Brazil ²University of Wuppertal, Germany

October 2, 2008

K ロ ▶ (K @) | K 글) | K 글) | □ 글

 299

 $A \equiv \mathcal{P} \rightarrow A \equiv$

(ロ) (母)

 298

B J.

[Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0)

[The problem](#page-3-0) [Rump's Algorithm](#page-17-0) [Available software](#page-18-0)

Mariana Kolberg, Walter Krämer, Michael Zimmer [Efficient parallel linear dense interval system solvers](#page-0-0)

イロト イ部 トイモ トイモト

 298

B

[Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) [The problem](#page-6-0) [Rump's Algorithm](#page-17-0) [Available software](#page-18-0)

Task

Find a verified enclosure $[x]$ for the solution of the dense linear system $[A][x] = [b]$, $A \in \mathbb{IR}^{n \times n}$, $b \in \mathbb{IR}^n$.

Also allowed:

- Real point systems \bullet
- Complex point systems
- Complex interval systems

 $\left\{ \begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \end{array} \right.$

B

[Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) [The problem](#page-6-0) [Rump's Algorithm](#page-17-0) [Available software](#page-18-0)

Task

Find a verified enclosure $[x]$ for the solution of the dense linear system $[A][x] = [b]$, $A \in \mathbb{IR}^{n \times n}$, $b \in \mathbb{IR}^n$.

Also allowed:

- Real point systems
- Complex point systems
- Complex interval systems

イロト イ押 トイヨ トイヨト

B

[Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) [The problem](#page-6-0) [Rump's Algorithm](#page-17-0) [Available software](#page-18-0)

Task

Find a verified enclosure $[x]$ for the solution of the dense linear system $[A][x] = [b]$, $A \in \mathbb{IR}^{n \times n}$, $b \in \mathbb{IR}^n$.

Also allowed:

- Real point systems
- Complex point systems
- Complex interval systems

イロト イ押 トイヨ トイヨト

B

[Tools](#page-21-0) **[Parallelization](#page-42-0)** [Results](#page-64-0) [Summary](#page-106-0) [The problem](#page-3-0) [Rump's Algorithm](#page-17-0) [Available software](#page-18-0)

Task

Find a verified enclosure $[x]$ for the solution of the dense linear system $[A][x] = [b]$, $A \in \mathbb{IR}^{n \times n}$, $b \in \mathbb{IR}^n$.

Also allowed:

- Real point systems
- Complex point systems
- Complex interval systems

イロト イ押 トイヨ トイヨト

B

[Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) [The problem](#page-3-0) [Rump's Algorithm](#page-17-0) [Available software](#page-18-0)

Problem

Problems of higher dimension require a lot of processing power and especially a lot of memory.

- Verified solution using Rump's algorithm is 6-8 times slower than normal floating point solution using LU-decomposition.
- Memory requirements: Matrices A, R and [C]
	-
	-
	-
	-
	-

Solution: Parallelization for distributed me[m](#page-6-0)[or](#page-8-0)[y](#page-6-0) [s](#page-7-0)[y](#page-16-0)[s](#page-17-0)[t](#page-2-0)[e](#page-3-0)[m](#page-16-0)[s.](#page-1-0)

イロト イ押 トイヨ トイヨト

[The problem](#page-3-0) [Rump's Algorithm](#page-17-0) [Available software](#page-18-0)

Problem

Problems of higher dimension require a lot of processing power and especially a lot of memory.

- Verified solution using Rump's algorithm is 6-8 times slower than normal floating point solution using LU-decomposition.
- Memory requirements: Matrices A, R and [C]
	-
	-
	-
	-
	-

Solution: Parallelization for distributed me[m](#page-7-0)[or](#page-9-0)[y](#page-6-0) [s](#page-7-0)[y](#page-16-0)[s](#page-17-0)[t](#page-2-0)[e](#page-3-0)[m](#page-16-0)[s.](#page-1-0)

イロト イ押 トイヨ トイヨト

[The problem](#page-3-0) [Rump's Algorithm](#page-17-0) [Available software](#page-18-0)

Problem

Problems of higher dimension require a lot of processing power and especially a lot of memory.

- Verified solution using Rump's algorithm is 6-8 times slower than normal floating point solution using LU-decomposition.
- Memory requirements: Matrices A, R and [C]
	- Real point system, n=20000: about 12GB
	- Real point system, n=50000: about 75GB
	- Real interval system, n=50000: about 93GB
	- Complex point system, n=50000: about 150GB
	- Complex interval system, n=50000: about 186GB

Solution: Parallelization for distributed me[m](#page-8-0)[or](#page-10-0)[y](#page-6-0) [s](#page-7-0)[y](#page-16-0)[s](#page-17-0)[t](#page-2-0)[e](#page-3-0)[m](#page-16-0)[s.](#page-1-0)

 $\left\{ \begin{array}{ccc} \square & \rightarrow & \left\langle \bigoplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. \end{array}$

[The problem](#page-3-0) [Rump's Algorithm](#page-17-0) [Available software](#page-18-0)

Problem

Problems of higher dimension require a lot of processing power and especially a lot of memory.

- Verified solution using Rump's algorithm is 6-8 times slower than normal floating point solution using LU-decomposition.
- \bullet Memory requirements: Matrices A, R and $[C]$
	- Real point system, n=20000: about 12GB
	- Real point system, n=50000: about 75GB
	- Real interval system, n=50000: about 93GB
	- Complex point system, n=50000: about 150GB
	- Complex interval system, n=50000: about 186GB

Solution: Parallelization for distributed me[m](#page-9-0)[or](#page-11-0)[y](#page-6-0) [s](#page-7-0)[y](#page-16-0)[s](#page-17-0)[t](#page-2-0)[e](#page-3-0)[m](#page-16-0)[s.](#page-1-0)

 $\left\{ \begin{array}{ccc} \square & \rightarrow & \left\langle \bigoplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. \end{array}$

[The problem](#page-3-0) [Rump's Algorithm](#page-17-0) [Available software](#page-18-0)

Problem

Problems of higher dimension require a lot of processing power and especially a lot of memory.

- Verified solution using Rump's algorithm is 6-8 times slower than normal floating point solution using LU-decomposition.
- \bullet Memory requirements: Matrices A, R and $[C]$
	- Real point system, n=20000: about 12GB
	- Real point system, n=50000: about 75GB \bullet
	- Real interval system, n=50000: about 93GB
	- Complex point system, n=50000: about 150GB
	- Complex interval system, n=50000: about 186GB

Solution: Parallelization for distributed me[m](#page-10-0)[or](#page-12-0)[y](#page-6-0) [s](#page-7-0)[y](#page-16-0)[s](#page-17-0)[t](#page-2-0)[e](#page-3-0)[m](#page-16-0)[s.](#page-1-0)

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

[The problem](#page-3-0) [Rump's Algorithm](#page-17-0) [Available software](#page-18-0)

Problem

Problems of higher dimension require a lot of processing power and especially a lot of memory.

- Verified solution using Rump's algorithm is 6-8 times slower than normal floating point solution using LU-decomposition.
- \bullet Memory requirements: Matrices A, R and $[C]$
	- Real point system, n=20000: about 12GB
	- Real point system, n=50000: about 75GB \bullet
	- Real interval system, n=50000: about 93GB \bullet
	- Complex point system, n=50000: about 150GB
	- Complex interval system, n=50000: about 186GB

Solution: Parallelization for distributed me[m](#page-11-0)[or](#page-13-0)[y](#page-6-0) [s](#page-7-0)[y](#page-16-0)[s](#page-17-0)[t](#page-2-0)[e](#page-3-0)[m](#page-16-0)[s.](#page-1-0)

 QQ

(□) (母) (ヨ) (ヨ)

[The problem](#page-3-0) [Rump's Algorithm](#page-17-0) [Available software](#page-18-0)

Problem

Problems of higher dimension require a lot of processing power and especially a lot of memory.

- Verified solution using Rump's algorithm is 6-8 times slower than normal floating point solution using LU-decomposition.
- \bullet Memory requirements: Matrices A, R and $[C]$
	- Real point system, n=20000: about 12GB
	- Real point system, n=50000: about 75GB
	- Real interval system, n=50000: about 93GB \bullet
	- Complex point system, n=50000: about 150GB
	- Complex interval system, n=50000: about 186GB

Solution: Parallelization for distributed me[m](#page-12-0)[or](#page-14-0)[y](#page-6-0) [s](#page-7-0)[y](#page-16-0)[s](#page-17-0)[t](#page-2-0)[e](#page-3-0)[m](#page-16-0)[s.](#page-1-0)

 QQ

(□) (母) (ヨ) (ヨ)

[The problem](#page-3-0) [Rump's Algorithm](#page-17-0) [Available software](#page-18-0)

Problem

Problems of higher dimension require a lot of processing power and especially a lot of memory.

- Verified solution using Rump's algorithm is 6-8 times slower than normal floating point solution using LU-decomposition.
- \bullet Memory requirements: Matrices A, R and $[C]$
	- Real point system, n=20000: about 12GB
	- Real point system, n=50000: about 75GB
	- Real interval system, n=50000: about 93GB \bullet
	- Complex point system, n=50000: about 150GB
	- Complex interval system, n=50000: about 186GB

Solution: Parallelization for distributed me[m](#page-13-0)[or](#page-15-0)[y](#page-6-0) [s](#page-7-0)[y](#page-16-0)[s](#page-17-0)[t](#page-2-0)[e](#page-3-0)[m](#page-16-0)[s.](#page-1-0)

 QQ

(ロト (母) (ヨ) (ヨ)

[The problem](#page-3-0) [Rump's Algorithm](#page-17-0) [Available software](#page-18-0)

Problem

Problems of higher dimension require a lot of processing power and especially a lot of memory.

- Verified solution using Rump's algorithm is 6-8 times slower than normal floating point solution using LU-decomposition.
- \bullet Memory requirements: Matrices A, R and $[C]$
	- Real point system, n=20000: about 12GB
	- Real point system, n=50000: about 75GB
	- Real interval system, n=50000: about 93GB \bullet
	- Complex point system, n=50000: about 150GB
	- Complex interval system, n=50000: about 186GB

Solution: Parallelization for distributed me[m](#page-14-0)[or](#page-16-0)[y](#page-6-0) [s](#page-7-0)[y](#page-16-0)[s](#page-17-0)[t](#page-2-0)[e](#page-3-0)[m](#page-16-0)[s.](#page-1-0)

 QQ

(ロト (母) (ヨ) (ヨ)

[The problem](#page-3-0) [Rump's Algorithm](#page-17-0) [Available software](#page-18-0)

Problem

Problems of higher dimension require a lot of processing power and especially a lot of memory.

- Verified solution using Rump's algorithm is 6-8 times slower than normal floating point solution using LU-decomposition.
- Memory requirements: Matrices A, R and [C]
	- Real point system, n=20000: about 12GB
	- Real point system, n=50000: about 75GB
	- Real interval system, n=50000: about 93GB \bullet
	- Complex point system, n=50000: about 150GB
	- Complex interval system, n=50000: about 186GB

Solution: Parallelization for distributed me[m](#page-15-0)[or](#page-17-0)[y](#page-6-0) [s](#page-7-0)[y](#page-16-0)[s](#page-17-0)[t](#page-2-0)[e](#page-3-0)[m](#page-16-0)[s.](#page-1-0)

[Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) [The problem](#page-3-0) [Rump's Algorithm](#page-17-0) [Available software](#page-18-0)

Algorithm: Verified solution of dense linear (interval-)systems

Input: Square matrix A and right hand side b **Output:** An interval vector enclosing the solution of $Ax = b$ Compute approximate inverse R of A Compute approximate solution $\tilde{x} := Rb$ **repeat** $\widetilde{\mathbf{x}} := \widetilde{\mathbf{x}} + R(b - A\widetilde{\mathbf{x}})$ until \tilde{x} exact enough or max. iterations reached $Z := R \circ (b - A\tilde{x})$ $C := \diamond (I - RA)$ $Y = Z$ **repeat** $Y_A := \text{blow}(Y, \epsilon)$ $Y = Z + C \cdot Y_A$ **until** $Y \subset int(Y_A)$ or max. iterations reached **if** $Y \subset int(Y_A)$ **then**
Unique solutio Unique solution in $x \in \tilde{x} + Y$ **else** \mathbf{L} Algorithm failed, A is singular or condition is too bad

For badly conditioned systems: Second stage using inverse of double length and extended precision dot products.

K ロ ト K 何 ト K ヨ ト K ヨ ト

B

[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0)

> [Results](#page-64-0) [Summary](#page-106-0)

[The problem](#page-3-0) [Rump's Algorithm](#page-17-0) [Available software](#page-20-0)

• Intlab: Fast, easy to use

- Serial C-XSC solvers: Fast, high accuracy
- Intel MKL and CMKL: Krawczyk-Solver is about 20 times \bullet slower, discontinued(?)

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

[The problem](#page-3-0) [Rump's Algorithm](#page-17-0) [Available software](#page-20-0)

• Intlab: Fast, easy to use

• Serial C-XSC solvers: Fast, high accuracy

Intel MKL and CMKL: Krawczyk-Solver is about 20 times \bullet slower, discontinued(?)

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

[The problem](#page-3-0) [Rump's Algorithm](#page-17-0) [Available software](#page-18-0)

- Intlab: Fast, easy to use
- Serial C-XSC solvers: Fast, high accuracy
- **Intel MKL and CMKL: Krawczyk-Solver is about 20 times** slower, discontinued(?)

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

 $2Q$

э

[C-XSC](#page-22-0) [Dot products](#page-28-0) [BLAS, LAPACK, ScaLAPACK](#page-38-0)

Mariana Kolberg, Walter Krämer, Michael Zimmer [Efficient parallel linear dense interval system solvers](#page-0-0)

イロト イ部 トイミト イモト

 \Rightarrow

 298

[C-XSC](#page-27-0) [Dot products](#page-28-0) [BLAS, LAPACK, ScaLAPACK](#page-38-0)

● C++ library for eXtended Scientific Computing

- Basic datatypes: real, interval, complex, cinterval
- Datatypes for vectors and matrices \bullet
- Many built in functions
- Computation of sums and dot products with maximum precision using fixed-point accumulator
- Toolbox with algorithms for many problems (linear- and nonlinear systems, optimization, ...)

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

[C-XSC](#page-27-0) [Dot products](#page-28-0) [BLAS, LAPACK, ScaLAPACK](#page-38-0)

- C++ library for eXtended Scientific Computing
- Basic datatypes: real, interval, complex, cinterval
- Datatypes for vectors and matrices \bullet
- Many built in functions
- Computation of sums and dot products with maximum precision using fixed-point accumulator
- Toolbox with algorithms for many problems (linear- and nonlinear systems, optimization, ...)

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

[C-XSC](#page-27-0) [Dot products](#page-28-0) [BLAS, LAPACK, ScaLAPACK](#page-38-0)

- C++ library for eXtended Scientific Computing
- Basic datatypes: real, interval, complex, cinterval
- Datatypes for vectors and matrices \bullet
- Many built in functions
- Computation of sums and dot products with maximum precision using fixed-point accumulator
- Toolbox with algorithms for many problems (linear- and nonlinear systems, optimization, ...)

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) [C-XSC](#page-27-0) [Dot products](#page-28-0) [BLAS, LAPACK, ScaLAPACK](#page-38-0)

- C++ library for eXtended Scientific Computing
- Basic datatypes: real, interval, complex, cinterval
- Datatypes for vectors and matrices
- Many built in functions
- Computation of sums and dot products with maximum precision using fixed-point accumulator
- Toolbox with algorithms for many problems (linear- and nonlinear systems, optimization, ...)

イロト イ押 トイヨ トイヨト

 $2Q$

э

[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) [C-XSC](#page-27-0) [Dot products](#page-28-0) [BLAS, LAPACK, ScaLAPACK](#page-38-0)

- C++ library for eXtended Scientific Computing
- Basic datatypes: real, interval, complex, cinterval
- Datatypes for vectors and matrices
- Many built in functions
- Computation of sums and dot products with maximum precision using fixed-point accumulator
- Toolbox with algorithms for many problems (linear- and nonlinear systems, optimization, ...)

イロト イ押 トイヨ トイヨト

 $2Q$

∍

[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) [C-XSC](#page-22-0) [Dot products](#page-28-0) [BLAS, LAPACK, ScaLAPACK](#page-38-0)

- C++ library for eXtended Scientific Computing
- Basic datatypes: real, interval, complex, cinterval
- Datatypes for vectors and matrices
- Many built in functions
- Computation of sums and dot products with maximum precision using fixed-point accumulator
- Toolbox with algorithms for many problems (linear- and nonlinear systems, optimization, ...)

イロト イ押 トイヨ トイヨト

 $2Q$

∍

[C-XSC](#page-22-0) [Dot products](#page-30-0) [BLAS, LAPACK, ScaLAPACK](#page-38-0)

In C-XSC

Use a fix-point accumulator of sufficient length

Dot products can be computed with maximum accuracy \bullet

• Slow! (when realized in software)

K ロ ト K 何 ト K ヨ ト K ヨ ト

÷.

[C-XSC](#page-22-0) [Dot products](#page-30-0) [BLAS, LAPACK, ScaLAPACK](#page-38-0)

In C-XSC

Use a fix-point accumulator of sufficient length

$$
\boxed{g} \quad 2e_{\text{max}} \quad t \quad t \quad 2|e_{\text{min}}|
$$

• Dot products can be computed with *maximum* accuracy

• Slow! (when realized in software)

イロト イ押 トイヨ トイヨト

B

[C-XSC](#page-22-0) [Dot products](#page-28-0) [BLAS, LAPACK, ScaLAPACK](#page-38-0)

In C-XSC

Use a fix-point accumulator of sufficient length

- Dot products can be computed with *maximum* accuracy
- Slow! (when realized in software)

イロト イ押ト イヨト イヨト

B

[C-XSC](#page-22-0) [Dot products](#page-28-0) [BLAS, LAPACK, ScaLAPACK](#page-38-0)

DotK algorithm (Ogita, Rump, Oishi)

Compute dot product in K-fold working precision by using error free transformations.

Mariana Kolberg, Walter Krämer, Michael Zimmer [Efficient parallel linear dense interval system solvers](#page-0-0)

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

[C-XSC](#page-22-0) [Dot products](#page-28-0) [BLAS, LAPACK, ScaLAPACK](#page-38-0)

DotK algorithm (Ogita, Rump, Oishi)

Compute dot product in K-fold working precision by using error free transformations.

Mariana Kolberg, Walter Krämer, Michael Zimmer [Efficient parallel linear dense interval system solvers](#page-0-0)

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

[C-XSC](#page-22-0) [Dot products](#page-28-0) [BLAS, LAPACK, ScaLAPACK](#page-38-0)

DotK algorithm (Ogita, Rump, Oishi)

Compute dot product in K-fold working precision by using error free transformations.

Error free transformations

For all $a, b \in \mathbb{F}$ and $\circ \in \{+, -, \cdot\}$ there exists a $y \in \mathbb{F}$ with

$$
a\circ b=x+y
$$

and $x = fl(a \circ b)$.

イロンス 御き スミメスミメー 差

[C-XSC](#page-22-0) [Dot products](#page-28-0) [BLAS, LAPACK, ScaLAPACK](#page-38-0)

\bullet Dot product in K-fold working precision

- **Uses pure floating point operations**
- Reliable error bound can be computed with floating point \bullet operations

 $\left\{ \begin{array}{ccc} \square & \rightarrow & \left\langle \bigoplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. \end{array}$

B

[C-XSC](#page-22-0) [Dot products](#page-28-0) [BLAS, LAPACK, ScaLAPACK](#page-38-0)

- \bullet Dot product in K-fold working precision
- **Uses pure floating point operations**
- Reliable error bound can be computed with floating point \bullet operations

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B
[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) [C-XSC](#page-22-0) [Dot products](#page-28-0) [BLAS, LAPACK, ScaLAPACK](#page-38-0)

- \bullet Dot product in K-fold working precision
- **Uses pure floating point operations**
- Reliable error bound can be computed with floating point operations

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) [C-XSC](#page-22-0) [Dot products](#page-28-0) [BLAS, LAPACK, ScaLAPACK](#page-38-0)

Exact result: 9.9999999999999995E-021 Condition: 1E+020

Direct computation using C-XSC operators: -3.4944599247537239E-012 Time used: 0.003654s

Computation using accumulator: 9.9999999999999995E-021 Time used: 0.148152s

Computation using DotK: $k=2$: [3.7130847403408012E-021,1.6286857271626398E-020] Time used: 0.0236309s

 $k=3$: [9.9999999999999979E-021,1.0000000000000001E-020] Time used: 0.0542479s

 $k = 4$: [9.9999999999999994E-021,9.9999999999999995E-021] Time used: 0.0600731s

 $k = 5$: [9.9999999999999994E-021,9.9999999999999995E-021] Time used: 0.0686409s

KO K (FIX KE) K E V A CA

[C-XSC](#page-22-0) [Dot products](#page-28-0) [BLAS, LAPACK, ScaLAPACK](#page-41-0)

● BLAS and LAPACK: Highly optimized routines for numerical algebra

- **ScaLAPACK: Special version for distributed memory** parallelization
- Manipulate rounding mode to get verified enclosures.

Input: Two real matrices A and B **Output:** An interval matrix enclosing $C = AB$ SetRound(-1); SetInf(C, A*B); SetRound(1); SetSup(C, A*B);

 QQ

Ξ

イロト イ押 トイ エト イエト

- BLAS and LAPACK: Highly optimized routines for numerical algebra
- **ScaLAPACK: Special version for distributed memory** parallelization
- Manipulate rounding mode to get verified enclosures.

Input: Two real matrices A and B **Output:** An interval matrix enclosing $C = AB$ SetRound(-1); SetInf(C, A*B); SetRound(1); SetSup(C, A*B);

 QQ

Ξ

イロト イ押 トイ エト イエト

- BLAS and LAPACK: Highly optimized routines for numerical algebra
- **ScaLAPACK: Special version for distributed memory** parallelization
- Manipulate rounding mode to get verified enclosures.

Input: Two real matrices A and B **Output:** An interval matrix enclosing $C = AB$ SetRound(-1); SetInf(C, A*B); SetRound(1); SetSup(C, A*B);

 QQ

Ξ

イロト イ押 トイ エト イエト

- BLAS and LAPACK: Highly optimized routines for numerical algebra
- **ScaLAPACK: Special version for distributed memory** parallelization
- Manipulate rounding mode to get verified enclosures.

Example: Product of two real matrices

Input: Two real matrices A and B **Output:** An interval matrix enclosing $C = AB$ SetRound(-1); SetInf(C, A*B); SetRound(1); SetSup(C, A*B);

 2990

Ξ

(ロ) (削) (ミ) (ミ)

[Two-dimensional, block cyclic distribution](#page-46-0) [Parallel computations](#page-48-0)

 $\left\{ \begin{array}{ccc} \square & \rightarrow & \left\langle \bigoplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. \end{array}$

 2990

B

Mariana Kolberg, Walter Krämer, Michael Zimmer [Efficient parallel linear dense interval system solvers](#page-0-0)

[Basics](#page-45-0) [Two-dimensional, block cyclic distribution](#page-46-0) [Parallel computations](#page-48-0)

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

Þ

 $2Q$

Basic concept

All matrices are distributed equally among processes, every step of the algorithm is computed by all processes

Q Uses Scal APACK instead of LAPACK

Uses two dimensional block cyclic distribution for matrices

[Basics](#page-45-0) [Two-dimensional, block cyclic distribution](#page-46-0) [Parallel computations](#page-48-0)

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

 $2Q$

э

Basic concept

All matrices are distributed equally among processes, every step of the algorithm is computed by all processes

O Uses Scal APACK instead of LAPACK

Uses two dimensional block cyclic distribution for matrices

[Basics](#page-43-0) [Two-dimensional, block cyclic distribution](#page-46-0) [Parallel computations](#page-48-0)

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

 QQ

э

Basic concept

All matrices are distributed equally among processes, every step of the algorithm is computed by all processes

- **O** Uses Scal APACK instead of LAPACK
- Uses two dimensional block cyclic distribution for matrices

[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) [Two-dimensional, block cyclic distribution](#page-46-0) [Parallel computations](#page-48-0)

Process grid:

Number of rows and columns should be the same or nearly the same.

 $\left\{ \begin{array}{ccc} \square & \rightarrow & \left\langle \bigoplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. \end{array}$

 298

B

[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) [Two-dimensional, block cyclic distribution](#page-46-0) [Parallel computations](#page-48-0)

Matrix is distributed accordingly:

Optimal size of blocks is hardware depend[en](#page-46-0)[t.](#page-48-0)

 2990

B

イロト イ押 トイヨ トイヨト

[Two-dimensional, block cyclic distribution](#page-46-0) [Parallel computations](#page-53-0)

イロト イ伊 トイヨ トイヨト

 Ω

- Matrix-matrix products and approximate inverse are computed with ScaLAPACK.
- Matrix-vector products are computed using high precision \bigcirc dot products.
- These dot products must be split among processes in a row.
- **•** Intermediate results are stored in accumulators.

MPI communicators are introduced for every row and column of process grid.

 \rightarrow Broadcasts limited to a row or column are possible.

[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) [Two-dimensional, block cyclic distribution](#page-46-0) [Parallel computations](#page-53-0)

- Matrix-matrix products and approximate inverse are computed with ScaLAPACK.
- Matrix-vector products are computed using high precision dot products.
- These dot products must be split among processes in a row.
- **•** Intermediate results are stored in accumulators.

MPI communicators are introduced for every row and column of process grid.

 \rightarrow Broadcasts limited to a row or column are possible.

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

 Ω

- Matrix-matrix products and approximate inverse are computed with ScaLAPACK.
- Matrix-vector products are computed using high precision dot products.
- These dot products must be split among processes in a row.
- **•** Intermediate results are stored in accumulators.

 \rightarrow Broadcasts limited to a row or column are possible.

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

 QQ

- Matrix-matrix products and approximate inverse are computed with ScaLAPACK.
- Matrix-vector products are computed using high precision dot products.
- These dot products must be split among processes in a row.
- Intermediate results are stored in accumulators.

 \rightarrow Broadcasts limited to a row or column are possible.

イロト イ押 トイヨ トイヨト

- Matrix-matrix products and approximate inverse are computed with ScaLAPACK.
- Matrix-vector products are computed using high precision dot products.
- These dot products must be split among processes in a row.
- Intermediate results are stored in accumulators.

 \rightarrow Broadcasts limited to a row or column are possible.

イロト イ押 トイヨ トイヨト

- Matrix-matrix products and approximate inverse are computed with ScaLAPACK.
- Matrix-vector products are computed using high precision dot products.
- These dot products must be split among processes in a row.
- Intermediate results are stored in accumulators.

 \rightarrow Broadcasts limited to a row or column are possible.

イロト イ押 トイヨ トイヨト

[Two-dimensional, block cyclic distribution](#page-46-0) [Parallel computations](#page-48-0)

イロト イ押 トイヨ トイヨト

 Ω

Algorithm: Parallel matrix-vector product

Input: A matrix A and vector x **Output:** The result of A times x **for** all i ∈ myrows **do** compute own parts of dot product broadcast intermediate results in own row compute final result for row i broadcast final result in own column

Now all processes know the result of Ax. (Necessary to check break conditions etc.)

[Two-dimensional, block cyclic distribution](#page-46-0) [Parallel computations](#page-48-0)

 \leftarrow

 $\mathcal{A} \ \overline{\mathcal{B}} \ \rightarrow \ \mathcal{A} \ \overline{\mathcal{B}} \ \rightarrow$

B

 QQ

- Use of higher precision dot products, BLAS/LAPACK only used for inversion
- Switch to a special case of the two-dimensional block cyclic distribution

- Use of higher precision dot products, BLAS/LAPACK only used for inversion
- Switch to a special case of the two-dimensional block cyclic distribution

 $\mathcal{A} \ \overline{\mathcal{B}} \ \rightarrow \ \mathcal{A} \ \overline{\mathcal{B}} \ \rightarrow$

 \leftarrow

B

[Two-dimensional, block cyclic distribution](#page-46-0) [Parallel computations](#page-48-0)

イロト イ伊 トイヨ トイヨト

 Ω

Parallel matrix-matrix product $R = AB$ in stage two:

- R is stored in the same way as A and B.
- Every process needs the vertical blocks of size $n \times nb$ of A. \bullet
- Every process broadcasts his part of this block.
- Computations for the corresponding vertical $n \times nb$ block can then be performed in parallel.
- This is repeated for all P vertical blocks.

[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) [Two-dimensional, block cyclic distribution](#page-46-0) [Parallel computations](#page-48-0)

Parallel matrix-matrix product $R = AB$ in stage two:

- R is stored in the same way as A and B.
- **•** Every process needs the vertical blocks of size $n \times nb$ of A.
- Every process broadcasts his part of this block.
- Computations for the corresponding vertical $n \times nb$ block can then be performed in parallel.
- This is repeated for all P vertical blocks.

イロト イ部 トイモ トイモト

 QQ

[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) **[Basics](#page-43-0)** [Two-dimensional, block cyclic distribution](#page-46-0) [Parallel computations](#page-48-0)

Parallel matrix-matrix product $R = AB$ in stage two:

- R is stored in the same way as A and B.
- **•** Every process needs the vertical blocks of size $n \times nb$ of A.
- **Every process broadcasts his part of this block.**
- Computations for the corresponding vertical $n \times nb$ block can then be performed in parallel.
- This is repeated for all P vertical blocks.

イロト イ部 トイミト イモト

 QQ

[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) **[Basics](#page-43-0)** [Two-dimensional, block cyclic distribution](#page-46-0) [Parallel computations](#page-48-0)

Parallel matrix-matrix product $R = AB$ in stage two:

- R is stored in the same way as A and B.
- **E** Very process needs the vertical blocks of size $n \times nb$ of A.
- **E** Very process broadcasts his part of this block.
- Computations for the corresponding vertical $n \times nb$ block can then be performed in parallel.
- This is repeated for all P vertical blocks.

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) **[Basics](#page-43-0)** [Two-dimensional, block cyclic distribution](#page-46-0) [Parallel computations](#page-48-0)

Parallel matrix-matrix product $R = AB$ in stage two:

- R is stored in the same way as A and B.
- **E** Very process needs the vertical blocks of size $n \times nb$ of A.
- **E** Very process broadcasts his part of this block.
- Computations for the corresponding vertical $n \times nb$ block can then be performed in parallel.
- This is repeated for all P vertical blocks.

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

[Two-dimensional, block cyclic distribution](#page-46-0) [Parallel computations](#page-48-0)

イロト イ押 トイヨ トイヨト

 QQ

∍

How is the system matrix distributed in the beginning?

- ¹ Process 0 stores the complete matrix in the beginning and distributes it.
- 2 A function pointer to a function like void getA(int i, int j, real &r) is used, every process fills his part of the matrix with this function.

[Two-dimensional, block cyclic distribution](#page-46-0) [Parallel computations](#page-48-0)

イロト イ押 トイヨ トイヨト

 $2Q$

∍

How is the system matrix distributed in the beginning?

- ¹ Process 0 stores the complete matrix in the beginning and distributes it.
- 2 A function pointer to a function like void getA(int i, int j, real $&r$) is used, every process fills his part of the matrix with this function.

[ALICEnext](#page-84-0) [XC6000](#page-91-0) [JUMP](#page-100-0)

Mariana Kolberg, Walter Krämer, Michael Zimmer [Efficient parallel linear dense interval system solvers](#page-0-0)

イロト イ部 トイミト イモト

 \Rightarrow

 298

[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) **[ALICEnext](#page-84-0)** [XC6000](#page-91-0) [JUMP](#page-100-0)

Some Remarks concerning compilation:

- DotK algorithm does not work if processor uses higher precision registers. Appropriate compiler switches must be set (use of SSE floating point registers,...).
- **•** Inlining has a very high impact on the performance. Appropriate compiler switches must be used (Inlining limits may have to be extended).
- OpenMP instructions require an OpenMP capable compiler, preferrably the latest version.

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

 QQ

[ALICEnext](#page-84-0) [XC6000](#page-91-0) [JUMP](#page-100-0)

Some Remarks concerning compilation:

- DotK algorithm does not work if processor uses higher precision registers. Appropriate compiler switches must be set (use of SSE floating point registers,...).
- Inlining has a very high impact on the performance. Appropriate compiler switches must be used (Inlining limits may have to be extended).
- OpenMP instructions require an OpenMP capable compiler, preferrably the latest version.

イロト イ押 トイヨ トイヨト

B

[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) **[ALICEnext](#page-84-0)** [XC6000](#page-91-0) [JUMP](#page-100-0)

Some Remarks concerning compilation:

- DotK algorithm does not work if processor uses higher precision registers. Appropriate compiler switches must be set (use of SSE floating point registers....).
- Inlining has a very high impact on the performance. Appropriate compiler switches must be used (Inlining limits may have to be extended).
- OpenMP instructions require an OpenMP capable compiler, preferrably the latest version.

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

[PC cluster](#page-74-0) **[ALICEnext](#page-84-0)** [XC6000](#page-91-0) [JUMP](#page-100-0)

Hardware:

• 24 standard PCs

- CPU: Core2Duo 2.33GHz
- 2GB RAM
- **Standard Gigabit Ethernet**

Software:

- GNU compiler 4.2.1
- LAM MPI
- ATLAS BLAS 3.8.1

 $\left\{ \begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \end{array} \right.$

 2990

÷.

[PC cluster](#page-74-0) **[ALICEnext](#page-84-0)** [XC6000](#page-91-0) [JUMP](#page-100-0)

Hardware:

- 24 standard PCs
- CPU: Core2Duo 2.33GHz
- 2GB RAM
- **Standard Gigabit Ethernet**

Software:

- GNU compiler 4.2.1
- LAM MPI
- ATLAS BLAS 3.8.1

イロトス 伊 トス ミトス ミト

 2990

÷.

[PC cluster](#page-74-0) **[ALICEnext](#page-84-0)** [XC6000](#page-91-0) [JUMP](#page-100-0)

Hardware:

- 24 standard PCs
- CPU: Core2Duo 2.33GHz
- 2GB RAM
- **Standard Gigabit Ethernet**

Software:

- GNU compiler 4.2.1
- LAM MPI
- ATLAS BLAS 3.8.1

 $\left\{ \begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \end{array} \right.$

 2990

÷.

[PC cluster](#page-74-0) **[ALICEnext](#page-84-0)** [XC6000](#page-91-0) [JUMP](#page-100-0)

Hardware:

- 24 standard PCs
- CPU: Core2Duo 2.33GHz
- 2GB RAM
- **Standard Gigabit Ethernet**

Software:

- GNU compiler 4.2.1
- LAM MPI
- ATLAS BLAS 3.8.1

イロト イ押 トイヨ トイヨト

B
[PC cluster](#page-74-0) **[ALICEnext](#page-84-0)** [XC6000](#page-91-0) [JUMP](#page-100-0)

Hardware:

- 24 standard PCs
- CPU: Core2Duo 2.33GHz
- 2GB RAM
- **Standard Gigabit Ethernet**

Software:

- GNU compiler 4.2.1
- LAM MPI
- ATLAS BLAS 3.8.1

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

[PC cluster](#page-74-0) **[ALICEnext](#page-84-0)** [XC6000](#page-91-0) [JUMP](#page-100-0)

Hardware:

- 24 standard PCs
- CPU: Core2Duo 2.33GHz
- 2GB RAM
- **Standard Gigabit Ethernet**

Software:

- GNU compiler 4.2.1
- LAM MPI
- ATLAS BLAS 3.8.1

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

[PC cluster](#page-68-0) **[ALICEnext](#page-84-0)** [XC6000](#page-91-0) [JUMP](#page-100-0)

Hardware:

- 24 standard PCs
- CPU: Core2Duo 2.33GHz
- 2GB RAM
- **Standard Gigabit Ethernet**

Software:

- GNU compiler 4.2.1
- LAM MPI
- **ATLAS BLAS 3.8.1**

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

[PC cluster](#page-68-0) [ALICEnext](#page-84-0) [XC6000](#page-91-0)

Time in s, condition 10 10 , n $=$ 5000, \mathcal{K} $=$ 2

Mariana Kolberg, Walter Krämer, Michael Zimmer [Efficient parallel linear dense interval system solvers](#page-0-0)

イロトメ 倒 トメ ヨ トメ ヨ トー ヨー

 298

[PC cluster](#page-68-0) [ALICEnext](#page-84-0) [XC6000](#page-91-0)

Speed up, condition 10¹⁰, $n = 5000, K = 2$

Mariana Kolberg, Walter Krämer, Michael Zimmer [Efficient parallel linear dense interval system solvers](#page-0-0)

イロトメ 倒 トメ 差 トメ 差 トー 差し

 $2QQ$

[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) [PC cluster](#page-68-0) [ALICEnext](#page-84-0) [XC6000](#page-91-0)

Average number of exact digits, condition 10¹⁰, $n = 5000$, $K = 2$

Mariana Kolberg, Walter Krämer, Michael Zimmer [Efficient parallel linear dense interval system solvers](#page-0-0)

 $\mathbf{E} = \mathbf{A} \oplus \mathbf{A} + \mathbf{A$

 2990

[PC cluster](#page-68-0) [ALICEnext](#page-84-0) [XC6000](#page-91-0)

Time in s, condition 10¹⁷, $n =$ 1000, $K = 3$

イロトメ 倒 トメ ヨ トメ ヨ トー ヨー

 298

[PC cluster](#page-68-0) [ALICEnext](#page-84-0) [XC6000](#page-91-0)

Speed up, condition 10¹⁷, $n = 1000, K = 3$

Mariana Kolberg, Walter Krämer, Michael Zimmer [Efficient parallel linear dense interval system solvers](#page-0-0)

 $\mathbf{A} \equiv \mathbf{A} + \math$

 $2QQ$

Average number of exact digits, condition 10¹⁷, $n = 1000$, $K = 3$

Mariana Kolberg, Walter Krämer, Michael Zimmer [Efficient parallel linear dense interval system solvers](#page-0-0)

 298

÷.

[PC cluster](#page-68-0) [ALICEnext](#page-84-0) [XC6000](#page-91-0) [JUMP](#page-100-0)

Time in s, OpenMP version, random real matrix, $n = 1000$, $K = 2$

Mariana Kolberg, Walter Krämer, Michael Zimmer [Efficient parallel linear dense interval system solvers](#page-0-0)

 $\mathbf{A} \equiv \mathbf{B} + \mathbf{A} \sqrt{\mathbf{B}} \mathbf{B} + \mathbf{A} \sqrt{\mathbf{B}} + \mathbf{A} \sqrt{\mathbf{B}} \mathbf{B} + \mathbf{A} \mathbf{B}$

 2990

 \Rightarrow

[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) [PC cluster](#page-68-0) [ALICEnext](#page-84-0) [XC6000](#page-91-0) [JUMP](#page-100-0)

Time in s, OpenMP version, random real matrix, $n = 2000$, $K = 2$

Mariana Kolberg, Walter Krämer, Michael Zimmer [Efficient parallel linear dense interval system solvers](#page-0-0)

 $\mathbf{A} \equiv \mathbf{B} + \mathbf{A} \sqrt{\mathbf{B}} \mathbf{B} + \mathbf{A} \sqrt{\mathbf{B}} + \mathbf{A} \sqrt{\mathbf{B}} \mathbf{B} + \mathbf{A} \mathbf{B}$

 2990

 \Rightarrow

[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0) [PC cluster](#page-68-0) [ALICEnext](#page-84-0) [XC6000](#page-91-0) [JUMP](#page-100-0)

Time in s, OpenMP version, random real matrix, $n = 3000$, $K = 2$

Mariana Kolberg, Walter Krämer, Michael Zimmer [Efficient parallel linear dense interval system solvers](#page-0-0)

 $\mathbf{A} \equiv \mathbf{B} + \mathbf{A} \sqrt{\mathbf{B}} \mathbf{B} + \mathbf{A} \sqrt{\mathbf{B}} + \mathbf{A} \sqrt{\mathbf{B}} \mathbf{B} + \mathbf{A} \mathbf{B}$

 2990

 \Rightarrow

[ALICEnext](#page-89-0) [XC6000](#page-91-0) [JUMP](#page-100-0)

Hardware:

\bullet 512 \times 2 AMD Opteron 1.8GHz

- 1GB RAM per processor
- Gigabit-Ethernet + 2D-Torus

Software:

- OS: Linux
- GNU Compiler 3.3.1
- AMD Core Math Library

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

[ALICEnext](#page-89-0) [XC6000](#page-91-0) [JUMP](#page-100-0)

Hardware:

- \bullet 512 \times 2 AMD Opteron 1.8GHz
- 1GB RAM per processor
- Gigabit-Ethernet + 2D-Torus

Software:

- OS: Linux
- GNU Compiler 3.3.1
- AMD Core Math Library

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

[ALICEnext](#page-89-0) [XC6000](#page-91-0) [JUMP](#page-100-0)

Hardware:

- \bullet 512 \times 2 AMD Opteron 1.8GHz
- 1GB RAM per processor
- Gigabit-Ethernet + 2D-Torus

Software:

- OS: Linux
- GNU Compiler 3.3.1
- AMD Core Math Library

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

[ALICEnext](#page-89-0) [XC6000](#page-91-0) [JUMP](#page-100-0)

Hardware:

- \bullet 512 \times 2 AMD Opteron 1.8GHz
- 1GB RAM per processor
- Gigabit-Ethernet + 2D-Torus

Software:

- OS: Linux
- GNU Compiler 3.3.1
- AMD Core Math Library

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

[ALICEnext](#page-89-0) [XC6000](#page-91-0) [JUMP](#page-100-0)

Hardware:

- \bullet 512 \times 2 AMD Opteron 1.8GHz
- 1GB RAM per processor
- Gigabit-Ethernet + 2D-Torus

Software:

- OS: Linux
- **GNU Compiler 3.3.1**
- AMD Core Math Library

 $\left\{ \begin{array}{ccc} \square & \rightarrow & \left\langle \bigoplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. \end{array}$

 $2Q$

э

[ALICEnext](#page-84-0) [XC6000](#page-91-0) [JUMP](#page-100-0)

Hardware:

- \bullet 512 \times 2 AMD Opteron 1.8GHz
- 1GB RAM per processor
- Gigabit-Ethernet + 2D-Torus

Software:

- OS: Linux
- **GNU Compiler 3.3.1**
- AMD Core Math Library

イロト イ伊 トイヨ トイヨト

 $2Q$

э

[PC cluster](#page-68-0) [ALICEnext](#page-84-0) [XC6000](#page-91-0)

Time in s, condition 10 10 , n $=$ 5000, \mathcal{K} $=$ 2

Mariana Kolberg, Walter Krämer, Michael Zimmer [Efficient parallel linear dense interval system solvers](#page-0-0)

イロトメ 御 トメ 差 トメ 差 トー 差し

 2990

[ALICEnext](#page-84-0) [XC6000](#page-95-0) [JUMP](#page-100-0)

Hardware:

● 128 Intel Itanium 2 1.5GHz

- 6GB per processor
- Quadrics QsNet II interconnect \bullet

Software:

- **•** Intel Compiler 10.0
- **Intel Math Kernel Library 10.0**

 $\left\{ \begin{array}{ccc} \square & \rightarrow & \left\langle \bigoplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. \end{array}$

B

[ALICEnext](#page-84-0) [XC6000](#page-95-0) [JUMP](#page-100-0)

Hardware:

- 128 Intel Itanium 2 1.5GHz
- 6GB per processor
- Quadrics QsNet II interconnect \bullet

Software:

- **•** Intel Compiler 10.0
- **Intel Math Kernel Library 10.0**

 $\left\{ \begin{array}{ccc} \square & \rightarrow & \left\langle \bigoplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. \end{array}$

B

[ALICEnext](#page-84-0) [XC6000](#page-95-0) [JUMP](#page-100-0)

Hardware:

- 128 Intel Itanium 2 1.5GHz
- 6GB per processor
- Quadrics QsNet II interconnect

Software:

- **•** Intel Compiler 10.0
- **Intel Math Kernel Library 10.0**

 $\left\{ \begin{array}{ccc} \square & \rightarrow & \left\langle \bigoplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. \end{array}$

B

[ALICEnext](#page-84-0) [XC6000](#page-95-0) [JUMP](#page-100-0)

Hardware:

- 128 Intel Itanium 2 1.5GHz
- 6GB per processor
- Quadrics QsNet II interconnect

Software:

- o Intel Compiler 10.0
- **Intel Math Kernel Library 10.0**

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

[ALICEnext](#page-84-0) [XC6000](#page-91-0) [JUMP](#page-100-0)

Hardware:

- 128 Intel Itanium 2 1.5GHz
- 6GB per processor
- Quadrics QsNet II interconnect

Software:

- o Intel Compiler 10.0
- o Intel Math Kernel Library 10.0

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

 $K = 2$, well conditioned real system

Time in s	$P = 20$	$P = 50$	$P = 100$
$n = 10000$	108.1	52.0	35.3
$n = 25000$	1188.0	532.2	299.7
$n = 50000$			1978.6
Speed Up	$P = 20$	P=50	$P = 100$
$n = 10000$		80.1%	59.0%
$n = 25000$		89.3%	79.3%
$n = 50000$			

Speed Up is given as percentage of th[eo](#page-95-0)r[et](#page-97-0)[ic](#page-95-0)[al](#page-96-0) [o](#page-90-0)[p](#page-91-0)[ti](#page-99-0)[m](#page-100-0)[u](#page-64-0)[m](#page-105-0)

 \leftarrow

 QQ

 $K = 2$, well conditioned interval system

Time in s	$P = 20$	P=50	$P = 100$
$n = 10000$	136.0	64.6	42.2
$n = 25000$	1571.7	687.3	385.3
$n = 50000$			2561.1
Speed Up	$P = 20$	P=50	$P = 100$
$n = 10000$		84.2%	64.5%
$n = 25000$		91.5%	81.6%
$n = 50000$			

Speed Up is given as percentage of th[eo](#page-96-0)r[et](#page-98-0)[ic](#page-96-0)[al](#page-97-0) [o](#page-90-0)[p](#page-91-0)[ti](#page-99-0)[m](#page-100-0)[u](#page-64-0)[m](#page-105-0)

[ALICEnext](#page-84-0) [XC6000](#page-91-0) [JUMP](#page-100-0)

Since data is distributed equally among processors, huge dense systems can be solved.

Using the full XC6000 cluster (128 Itanium 2 processors, 6GB per processor), we could solve a real system of dimension 100000 × 100000 in 12118 seconds.

Mariana Kolberg, Walter Krämer, Michael Zimmer [Efficient parallel linear dense interval system solvers](#page-0-0)

イロト イ押 トイヨ トイヨト

Þ

[ALICEnext](#page-84-0) [XC6000](#page-91-0) [JUMP](#page-100-0)

Since data is distributed equally among processors, huge dense systems can be solved.

Using the full XC6000 cluster (128 Itanium 2 processors, 6GB per processor), we could solve a real system of dimension 100000×100000 in 12118 seconds.

 $\langle \vert \bar{f} \vert \vert \rangle$ $\langle \vert \bar{f} \vert \rangle$ $\langle \vert \bar{f} \vert \rangle$ $\langle \vert \bar{f} \vert \rangle$

4. 17. 30

 $2Q$

∍

[ALICEnext](#page-84-0) [XC6000](#page-91-0) [JUMP](#page-104-0)

Hardware:

● 14 nodes with 32 IBM Power6 4.7GHz processors

- Memory: 14×128 Gigabyte
- **•** Infiniband network

Software:

- \bullet OS: AIX 5.3
- **IBM Compiler XLC**

 $\left\{ \begin{array}{ccc} \square & \rightarrow & \left\langle \bigoplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. & \rightarrow & \left\langle \biguplus \right. \right. \end{array}$

B

[ALICEnext](#page-84-0) [XC6000](#page-91-0) [JUMP](#page-104-0)

Hardware:

- 14 nodes with 32 IBM Power6 4.7GHz processors
- Memory: 14×128 Gigabyte
- **•** Infiniband network

Software:

- \bullet OS: AIX 5.3
- **IBM Compiler XLC**

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

[ALICEnext](#page-84-0) [XC6000](#page-91-0) **[JUMP](#page-104-0)**

Hardware:

- 14 nodes with 32 IBM Power6 4.7GHz processors
- Memory: 14×128 Gigabyte
- **•** Infiniband network

Software:

- \bullet OS: AIX 5.3
- **IBM Compiler XLC**

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

[ALICEnext](#page-84-0) [XC6000](#page-91-0) **[JUMP](#page-104-0)**

Hardware:

- 14 nodes with 32 IBM Power6 4.7GHz processors
- Memory: 14×128 Gigabyte
- **•** Infiniband network

Software:

- \bullet OS: AIX 5.3
- **IBM Compiler XLC**

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

[ALICEnext](#page-84-0) [XC6000](#page-91-0) **[JUMP](#page-100-0)**

Hardware:

- 14 nodes with 32 IBM Power6 4.7GHz processors
- Memory: 14×128 Gigabyte
- **•** Infiniband network

Software:

- \bullet OS: AIX 5.3
- **IBM Compiler XLC**

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

B

 $K = 2$, well conditioned real system

ter ext

Speed Up is given as percentage of th[eo](#page-104-0)r[et](#page-106-0)[ic](#page-104-0)[al](#page-105-0) [o](#page-99-0)[p](#page-100-0)[ti](#page-105-0)[m](#page-106-0)[u](#page-64-0)[m](#page-105-0)

 \leftarrow \Box

 2990

 \Rightarrow

∍

[Introduction](#page-2-0)

[Tools](#page-21-0)

[Parallelization](#page-42-0)

[Results](#page-64-0)

イロト イ部 トイモト イモト

 \Rightarrow

 298

Fast and very accurate verified solvers in C-XSC.

- Efficient parallelization. \bullet
- **Parallel solvers can solve very large dense systems.**
- Tested with satisfying results (performance and accuracy) \bullet on very different architectures.

• Outlook

-
-

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

 $2Q$

э

- Fast and very accurate verified solvers in C-XSC.
- **•** Efficient parallelization.
- Parallel solvers can solve very large dense systems.
- Tested with satisfying results (performance and accuracy) \bullet on very different architectures.

-
-

イロト イ押 トイヨ トイヨト

 $2Q$

э

- Fast and very accurate verified solvers in C-XSC.
- **•** Efficient parallelization.
- **Parallel solvers can solve very large dense systems.**
- Tested with satisfying results (performance and accuracy) \bullet on very different architectures.

-
-

イロト イ押 トイヨ トイヨト

 $2Q$

э

- Fast and very accurate verified solvers in C-XSC.
- **•** Efficient parallelization.
- **Parallel solvers can solve very large dense systems.**
- Tested with satisfying results (performance and accuracy) on very different architectures.

-
-

イロト イ押 トイヨ トイヨト

B

 $2Q$

- Fast and very accurate verified solvers in C-XSC.
- **•** Efficient parallelization.
- **Parallel solvers can solve very large dense systems.**
- Tested with satisfying results (performance and accuracy) on very different architectures.

- Solvers for sparse systems.
- Use of OpenMP for dot products in parallel solvers(?)

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

 QQ

- Fast and very accurate verified solvers in C-XSC.
- **•** Efficient parallelization.
- Parallel solvers can solve very large dense systems.
- Tested with satisfying results (performance and accuracy) on very different architectures.
- **•** Outlook
	- Solvers for sparse systems.
	- Use of OpenMP for dot products in parallel solvers(?) \bullet

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

 QQ

- Fast and very accurate verified solvers in C-XSC.
- **•** Efficient parallelization.
- Parallel solvers can solve very large dense systems.
- Tested with satisfying results (performance and accuracy) on very different architectures.
- **•** Outlook
	- Solvers for sparse systems.
	- Use of OpenMP for dot products in parallel solvers(?)

≮ロト ⊀伊ト ⊀ ヨト ⊀ ヨト

 $2Q$

[Introduction](#page-2-0) [Tools](#page-21-0) [Parallelization](#page-42-0) [Results](#page-64-0) [Summary](#page-106-0)

Thank you

Mariana Kolberg, Walter Krämer, Michael Zimmer [Efficient parallel linear dense interval system solvers](#page-0-0)

K ロラ K 倒 ト K 差 ト K 差 ト

重

 298