Символьное обращение плохо обусловленных матриц в Грид-среде сервисов доступа к системе компьютерной алгебры MAXIMa.

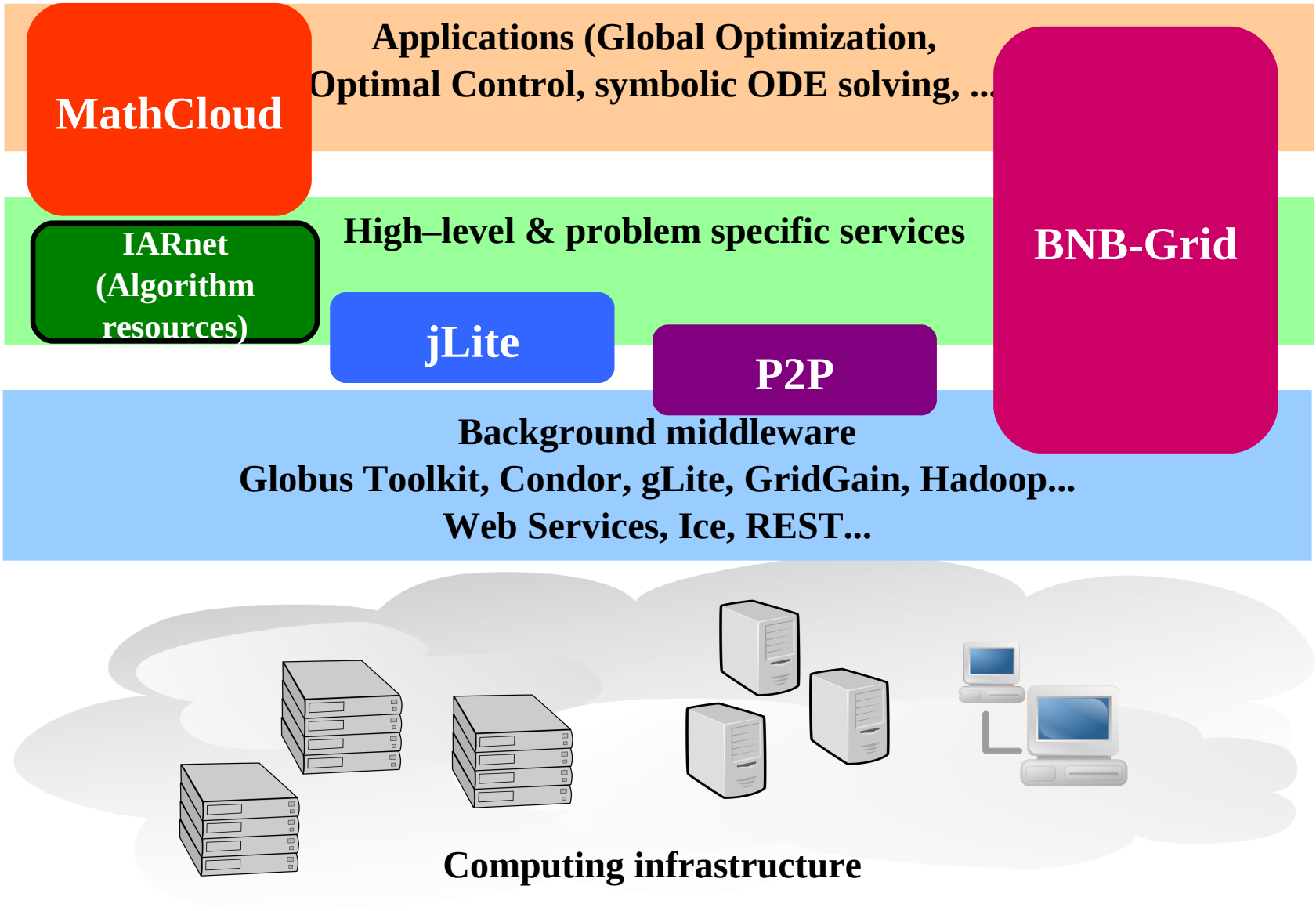
Ill-conditioned matrices symbolic inversion in Desktop Grid of CAS Maxima services.

Vladimir V. Voloshinov

Center of Grid-technologies and Distributed Computing,
Institute of System Analysis RAS, http://dcs.isa.ru
Moscow, 2009

Supported by RFBR, grant #**08-07-00430-a**

Area of our studies



Relates to the report

MathCloud

ODE solving (Chaotic dynamics)

CAS Services

Problem specific services

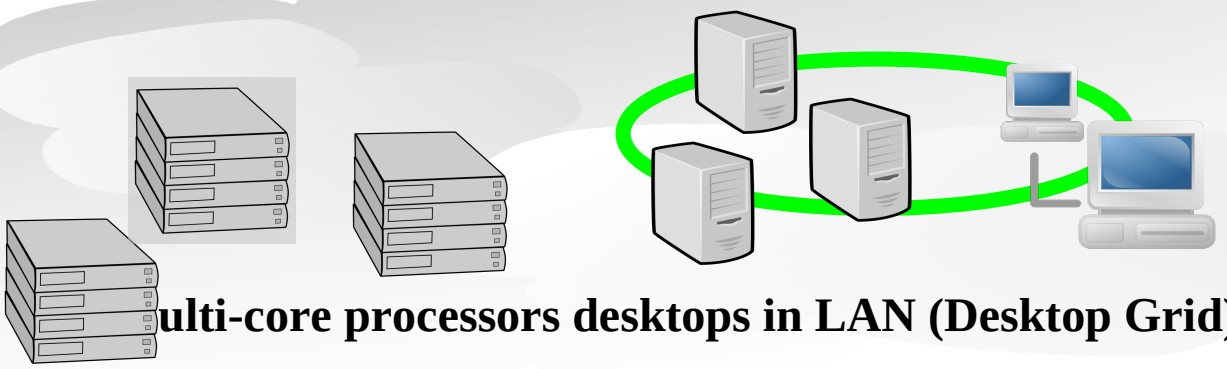
BNB-Grid

jLite

P2P

Object-oriented MIddleware

Ice (Internet Communication Engine, www.zeroc.com), since 2003



ulti-core processors desktops in LAN (Desktop Grid)

CAS (Computer Algebra System) Maxima (1).

Has been started in Massachusetts Institute of Technology, by prof. William Schelter. Since 1998 - GNU Public License.

The core - GCL (GNU Common Lisp), Windows, Linux

Single-threaded Lisp interpreter

Has almost the same basic "symbolic" capabilities as Maple and Mathematica: differentiation&integration, series, ODE solving, matrices and linear algebra, polynomials, sets, lists, tensors...

http://maxima.sourceforge.net/ (GPL)

http://maxima.sourceforge.net/ru/

CAS (Computer Algebra System) Maxima (2).

Inherent feature of float arithmetic computing — accumulation of rounding errors in intermediate operations.

Maxima handles rational number *x* as pairs of {numerator, denominator}, {p,q}: $x = \frac{p}{q}$

All arithmetic operations are performed without loss of accuracy.

Theoretically, regardless memory (and elapsed time!) unlimited accuracy may be expected.

If necessary, only final result may be rounded to float representation.

CAS (Computer Algebra System) Maxima (3).

Procedural programming language (MaximaScript).
MaximaScript «wraps» Lisp.

Examples of script:

Some initialisation.

```
/*N:100;*/
/*TID:"H";*/
path2common:"../common/";
...
load(eigen);
colInvByLUkd(LU,N,k):=lu_backsub(LU,unitVect(k,N));
luFileName(TID,N):=simplode([path2common,"LU",TID,N,".lsp"]);
invColFileName(TID,N,k):=simplode(["invTrCol",TID,N,"_",k,".lsp"]);
M:hilbert_matrix(N);
```

Composing inverse matrix from its columns written to files before.

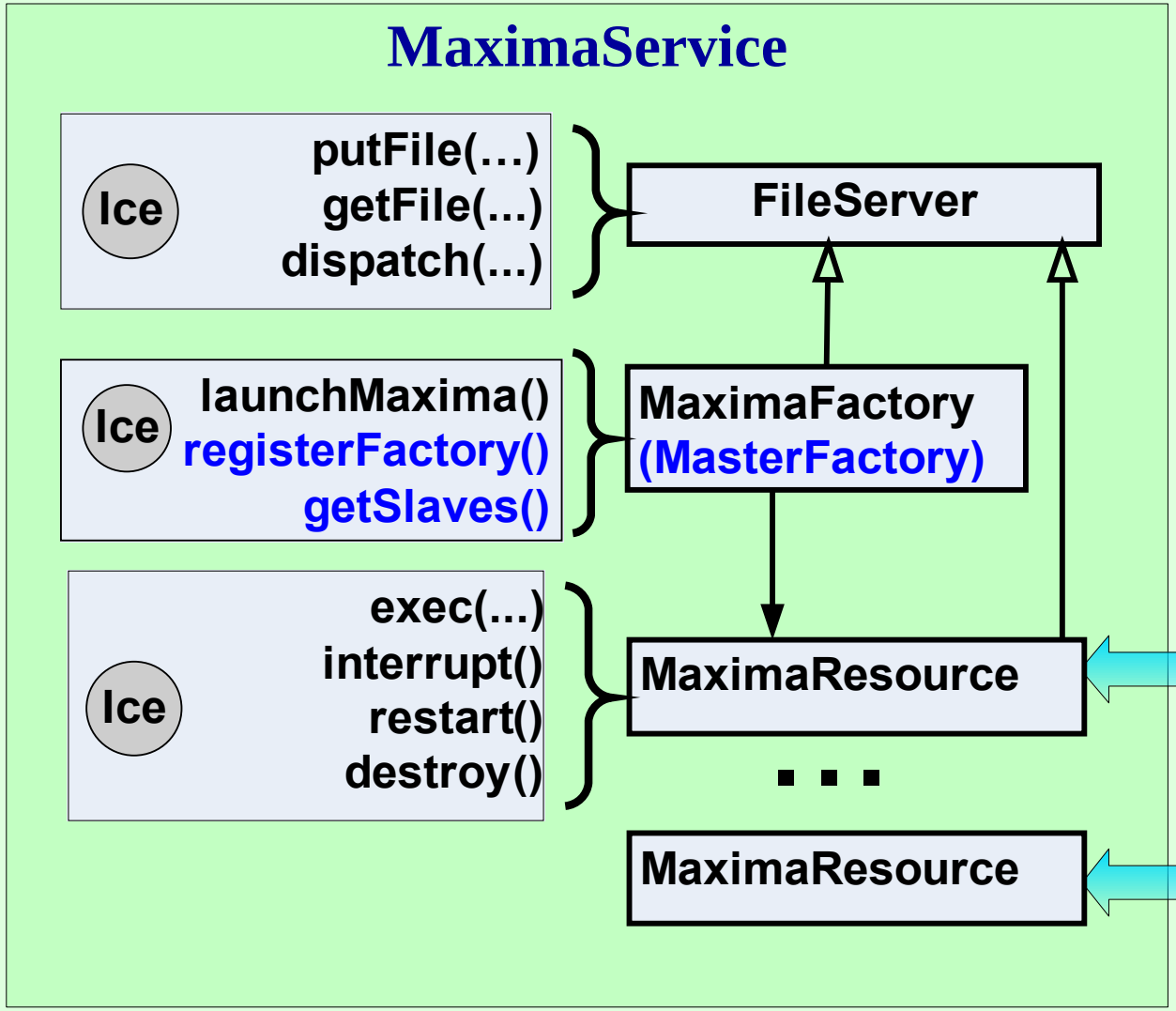
```
invColFileName(TID,N,k):=simplode(["invTrCol",TID,N,"_",k,".lsp"])$

colName(TID,N,k):=concat('invTrCol,TID,N,"_",k)$

readInv(TID,N,rpath):=block([k,fname,cmd],
    tmpM:zeromatrix(0,N),
    for k:1 thru N step 1 do (
        fname:simplode([rpath,invColFileName(TID,N,k)]),
        load(fname),
        cmd:simplode(["tmpM:addrow(tmpM,",colName(TID,N,k),")"]),
        eval_string(cmd)
    ),
    return(tmpM)
);
```

Maxima remote access service.

Node (unified services container IARnet2, Ice)

Basic service management

Resourse access control and security



Available implementation.

http://code.google.com/p/remote-maxima, GPL

Language C++, platforms Windows XP, Linux, MacOS

Ice, http://www.zeroc.ice, version 3.2.0 and later

Maxima version 5.12.0 and later.

Required additional libraries:

Boost 1.36.0, http://www.boost.org/

Boost.Process library, http://www.highscore.de/boost/process.zip

CxxTest, http://cxxtest.tigris.org

Hilbert matrices

Matrices H_N of the type: $H_N = \left\{ h_{m,n} \right\}_{m=1,n=1}^{N,N}$, где $h_{m,n} = \dfrac{1}{m+n-1}$

Gram matrix of the basic polynomials in $L_2[0,1]$ $\quad h_{mn} = \int_0^1 t^{m-1} \cdot t^{n-1} \, dt$

Well-known example of ill-conditioned matrices.

***Condition number* of H_N is growing exponentially w.r.t N**

$$cond\left(\boldsymbol{H}_N\right) = \|\boldsymbol{H}_N\| \cdot \|\left(\boldsymbol{H}_N\right)^{-1}\| \sim \mathrm{e}^{3.5 \cdot N}$$

$$\|A_{N \times N}\| = \left(\sum_{m=1,n=1}^{N,N} A_{i,j}^2 \right)^{1/2}$$

Values of $cond(\boldsymbol{H}_N)$ for some N (calculated exactly in Maxima)

N	$cond(\boldsymbol{H}_N)$
10	$1.6 \cdot 10^{13}$
50	$1.5 \cdot 10^{74}$
70	$5.5 \cdot 10^{104}$
100	$4.1 \cdot 10^{150}$
150	$1.2 \cdot 10^{227}$

«Traditionally», «well-conditioned» matrices should have *cond* less than 1000.

Possible speedup reasoning for «*LU-inversion*» of H_N (1)

Let ***M*** be [N×N] matrix. LU-factor {***L, U, P***}: ***L*** - lower-triangular; ***U*** - upper-triangular; ***P*** – permutation matrix. $L \cdot U = P \cdot M$.

Let E_N be unit matrix. To obtain M^{-1} - solve (by forward and backward substitution): $L \cdot U \cdot X = P \cdot E_N \Rightarrow X = M^{-1}$

Maxima has function ***lu_factor(M)*** (Gaussian elimination) and ***lu_backsub***(*LU,**B***) (*LU* — returned ***lu_factor(M)***) to solve $L \cdot U \cdot X = P \cdot B$ for any rectangular matrix ***B***[N×M].

invert_by_lu(M)*:=*lu_backsub(lu_factor(M), E_N)

If E_N <u>is sliced vertically</u> into *K* submatrices

$$\mathbf{E}_N = \left[\mathbf{E}_N^{1:n_1} \middle| \mathbf{E}_N^{n_1+1:n_2} \middle| \mathbf{E}_N^{n_2+1:n_3} \middle| \cdots \middle| \mathbf{E}_N^{n_{K-1}+1:N} \right], \quad 1 = n_0 \le n_1 < n_2 < n_3 < \ldots < n_{K-1} < n_K = N$$

then <u>parallel call</u> of ***lu_backsub***(*LU,* $\mathbf{E}_N^{n_{k-1}+1:n_k}$) gives K set of inverse columns $M^{-1}[n_{k-1}:n_k]$.

<u>K=N means parallel calculations of M^{-1} columns</u>.

Possible speedup reasoning for *«LU-inversion» of* H_N (2)

Durations of two phases of inversion :

LU-factorization (Gaussian elimination) and calculation of inverse matrix's columns.

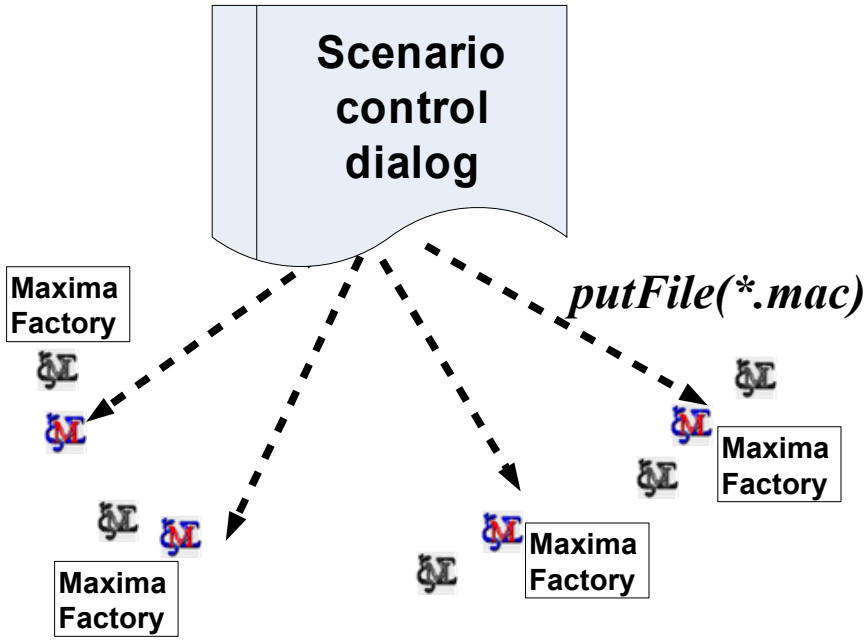
N	*invert_by_lu*($\boldsymbol{H}_N$), T_{inv}, *sec*	*LUP=lu_factor*($\boldsymbol{H}_N$), T_{LU} *sec*	*lu_backsub*(LUP, $\boldsymbol{E}_N$), T_{bs} *sec*
100	27	9	17
150	122	43	76
200	368	128	229
250	826	306	513
300	1684	631	1043

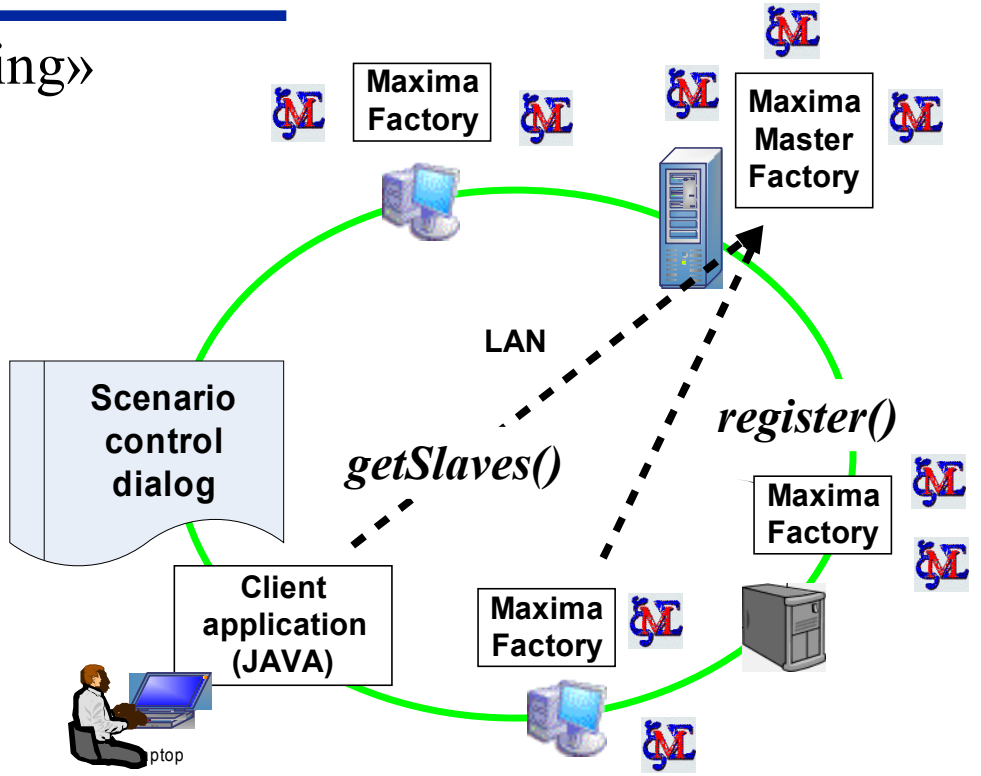
«Columns phase» (T_{bs}) takes almost twice of LU-factorization.

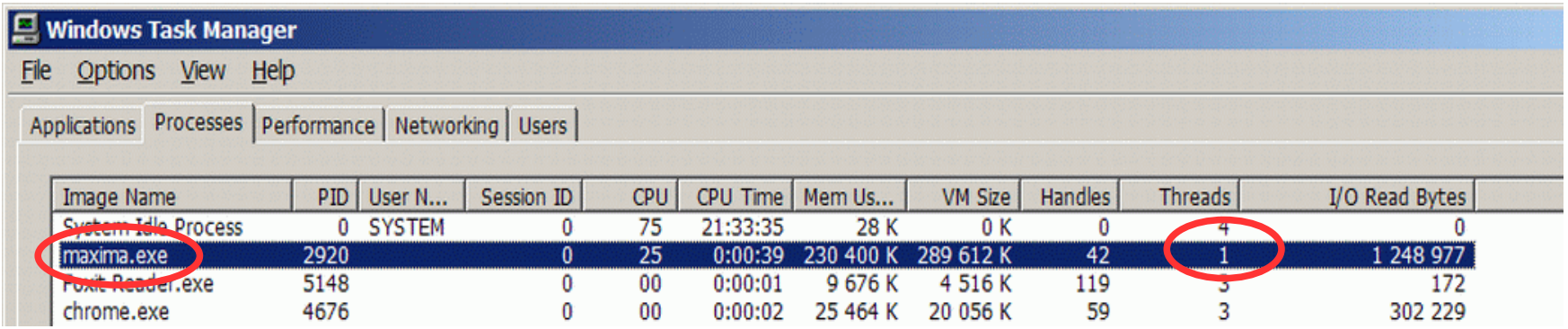
In the case of unlimited number of resources ~300% speedup may be expected.

«LU» scenario.

0 phase: start nodes, scripts «dispatching»

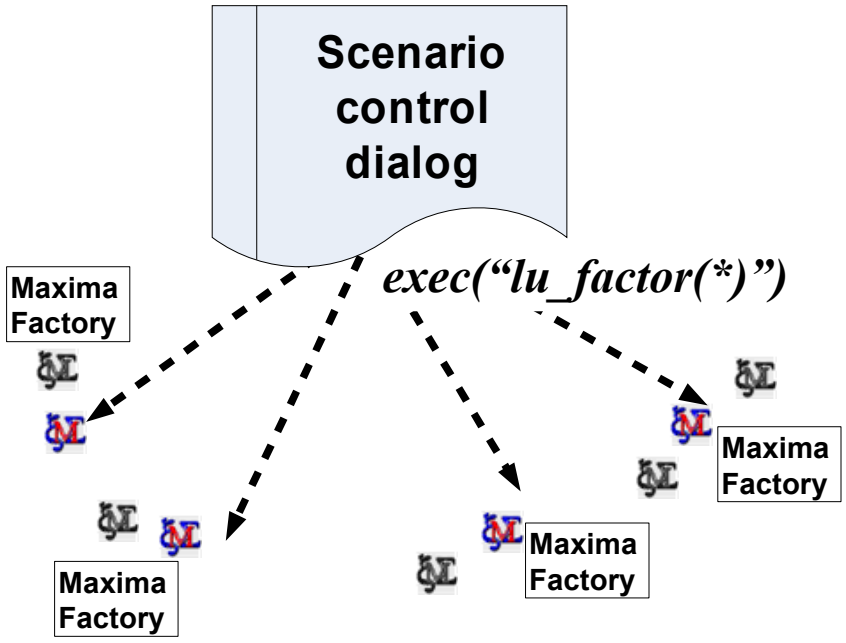


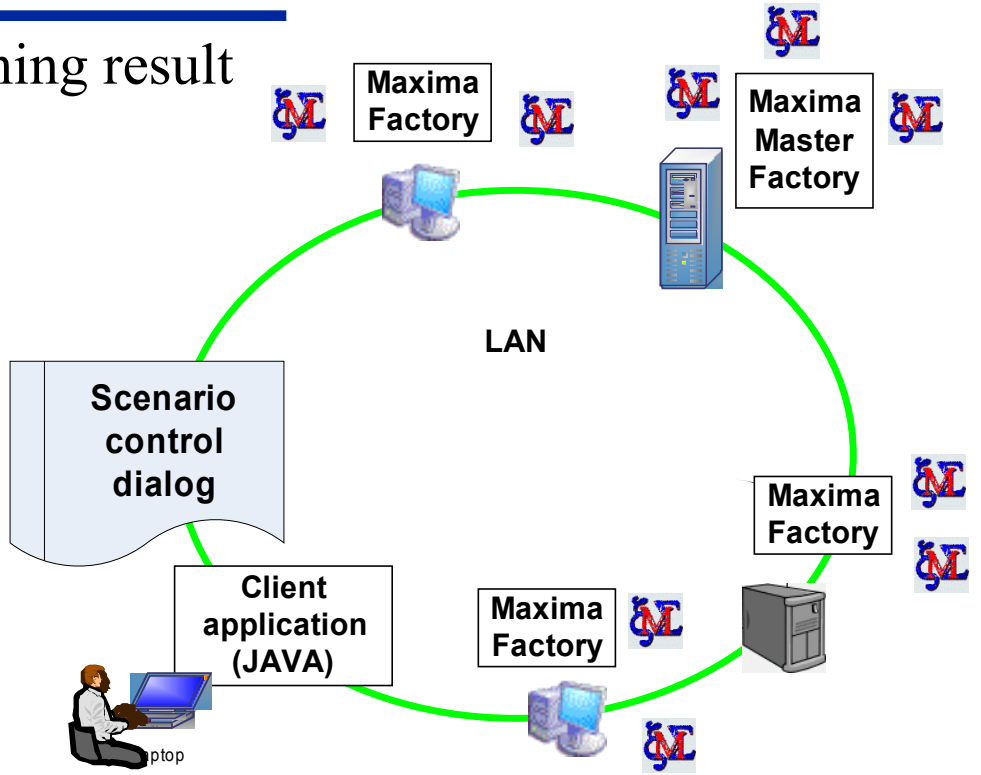




«LU» scenario.

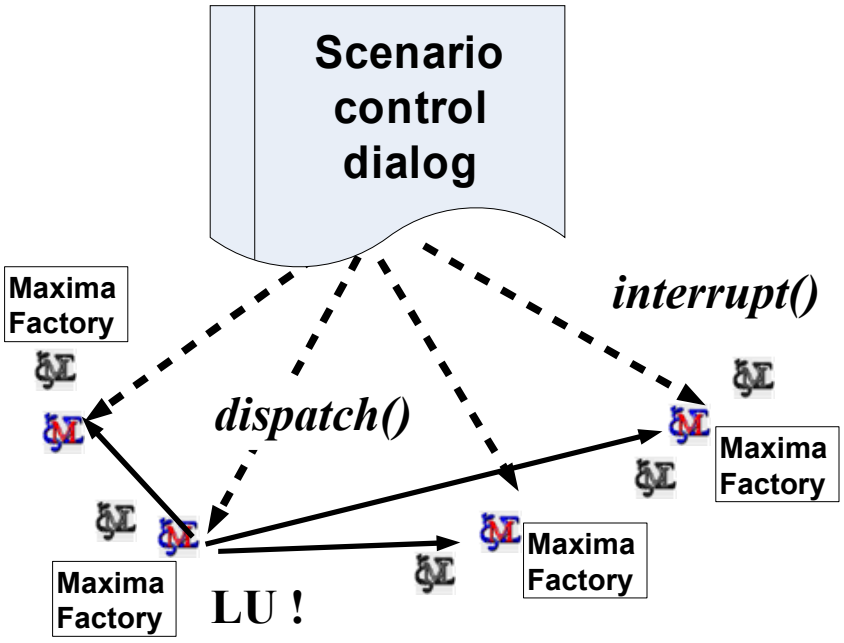
I phase: LU-factorization and dispatching result

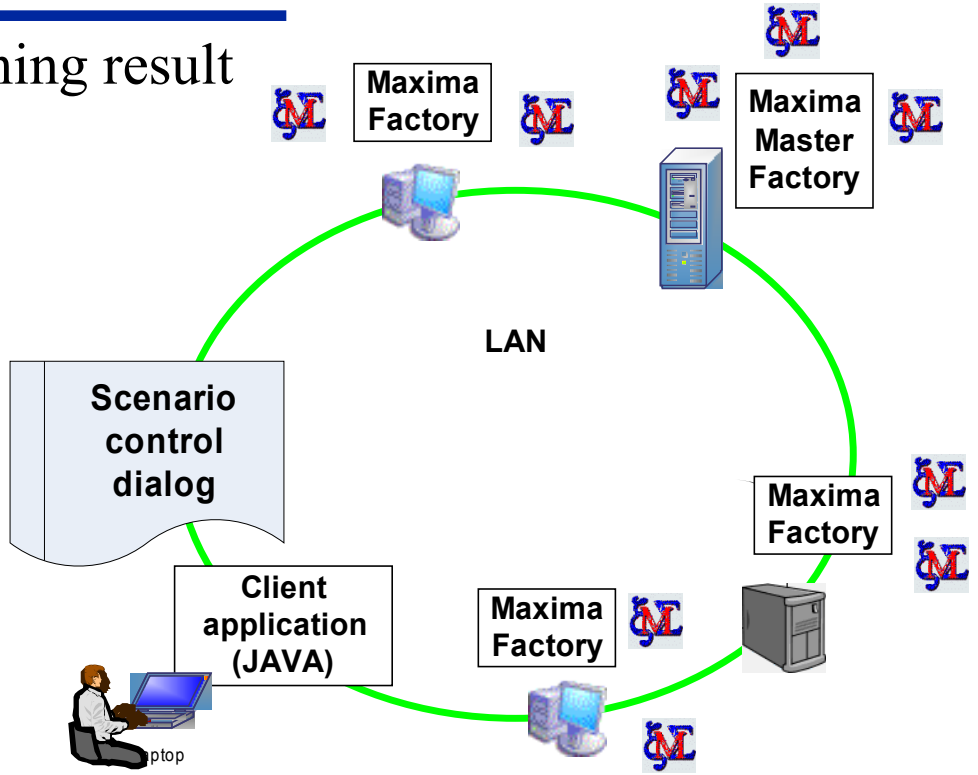




«LU» scenario.

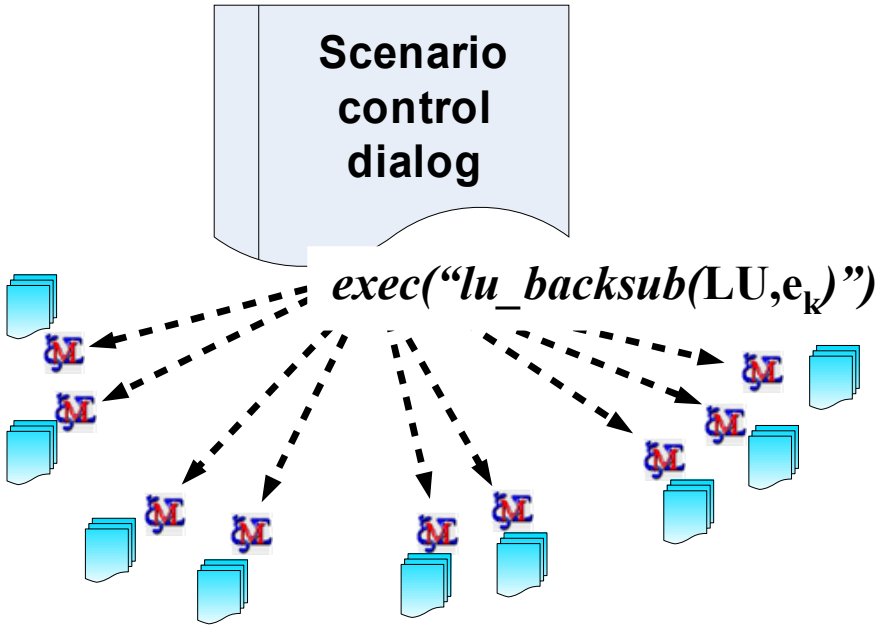
I phase: LU-factorization and dispatching result

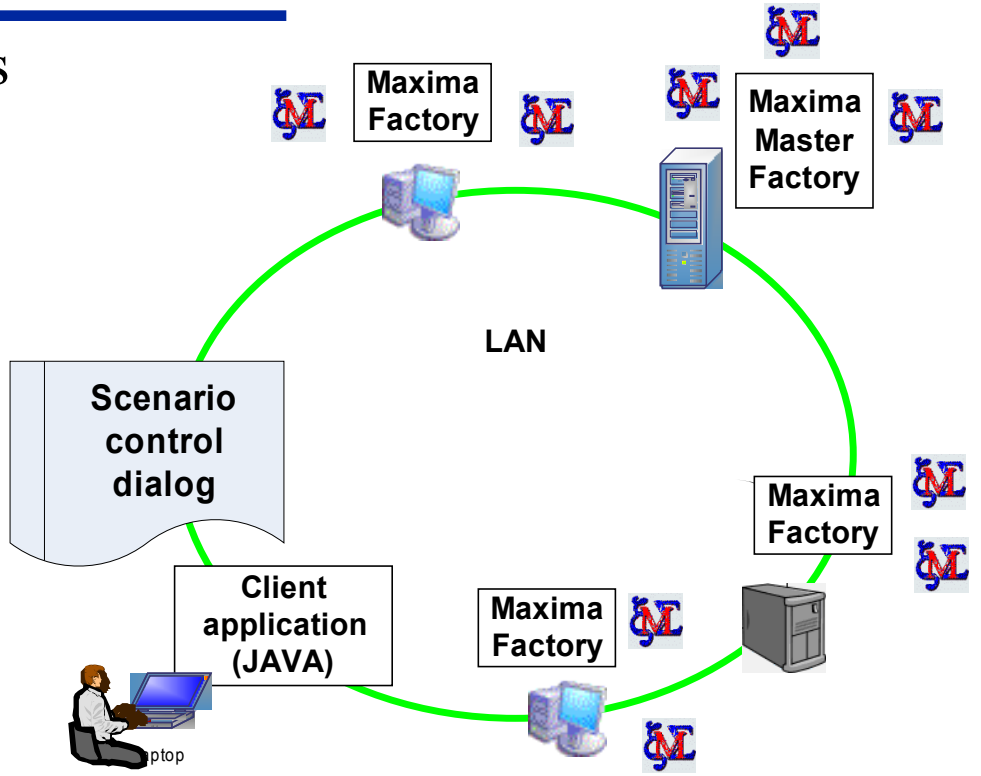




«LU» scenario.

II phase: inverse columns calculations

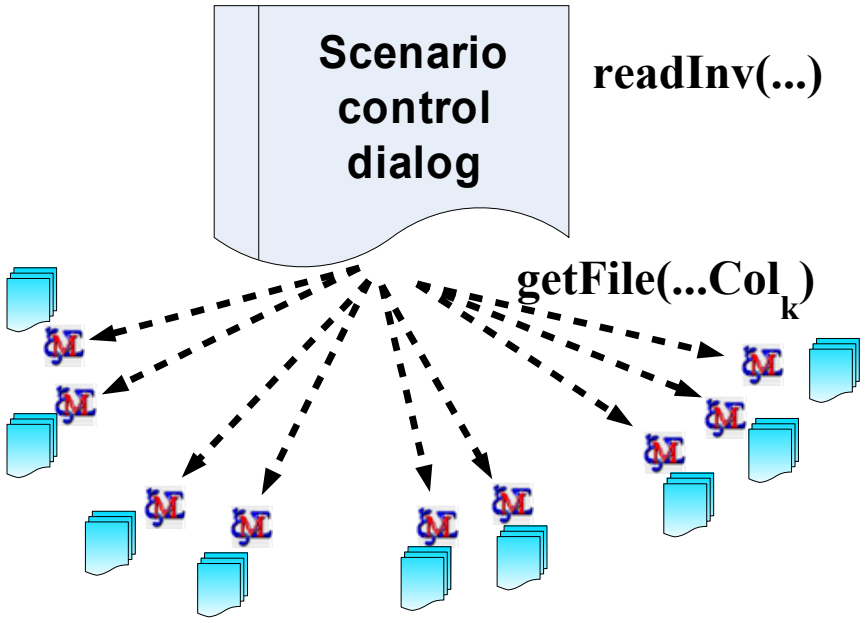


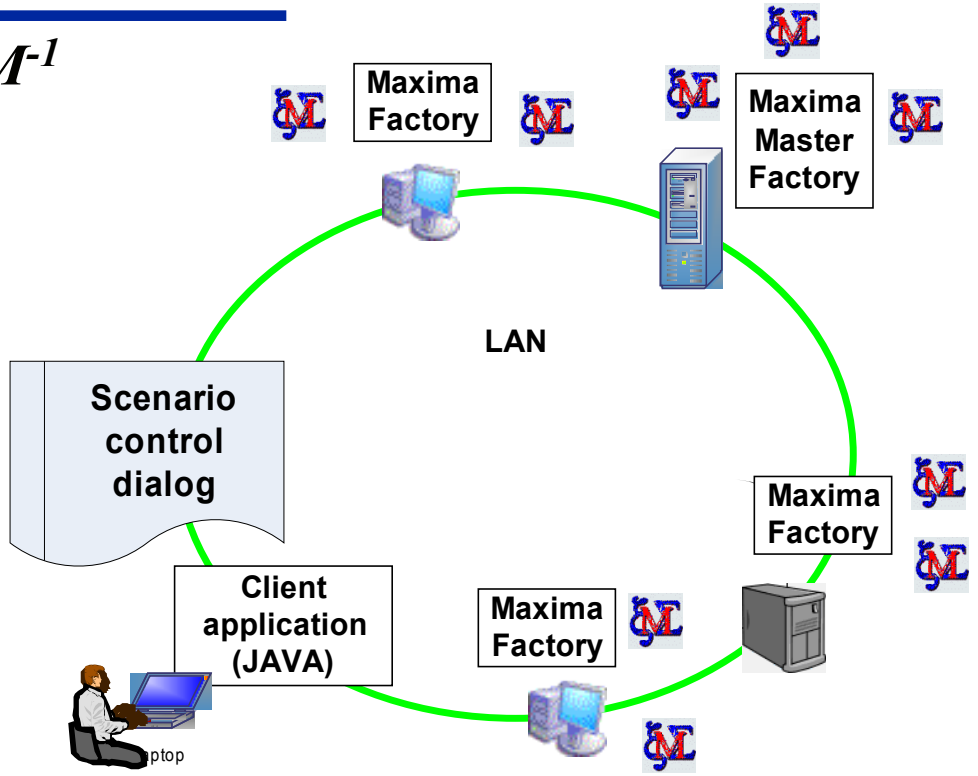


Columns M^{-1} are saved in files

«LU» scenario.

III phase: collect columns and build M^{-1}

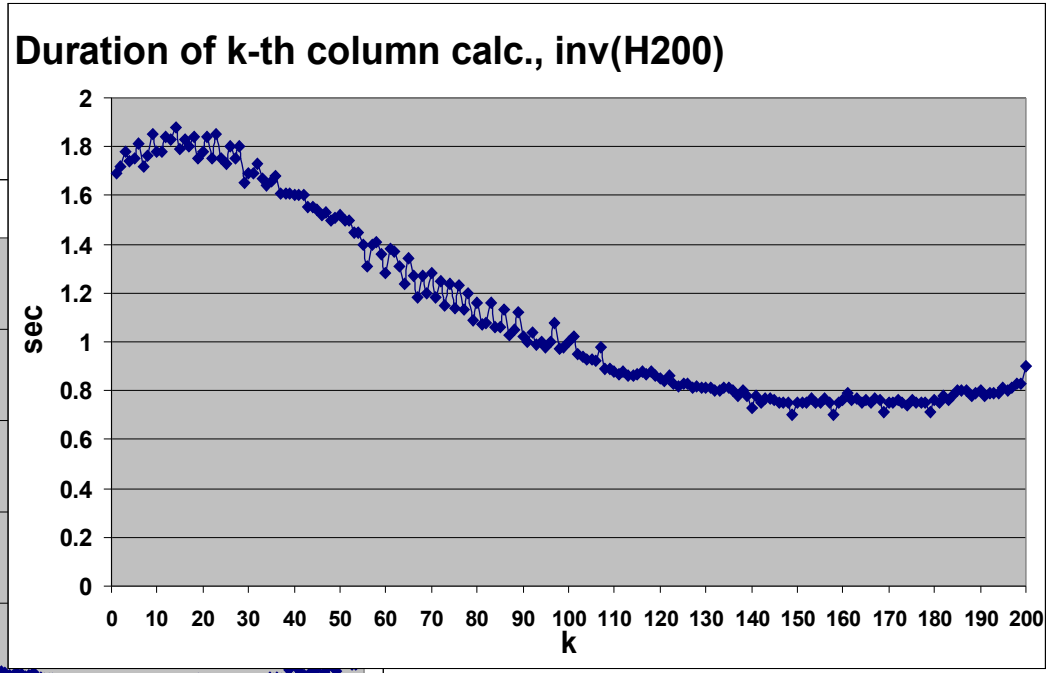


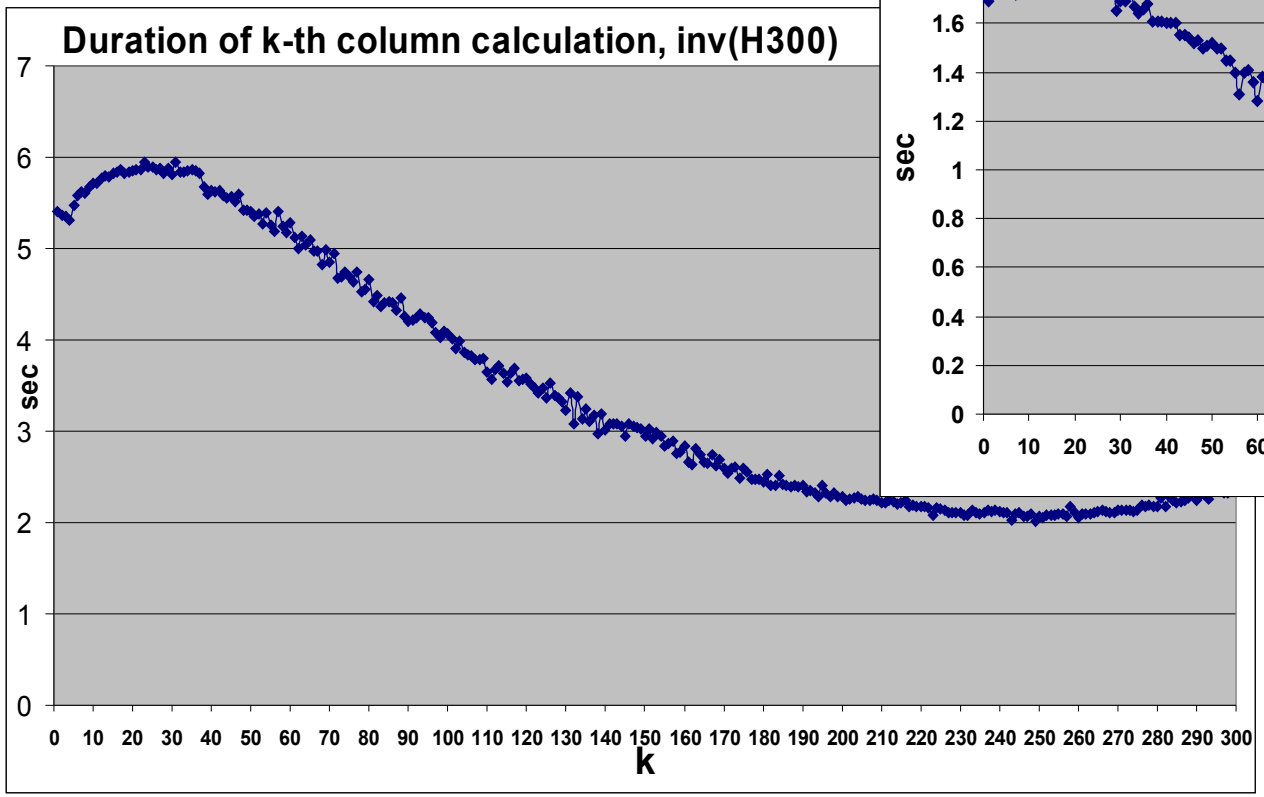


Columns M^{-1} are sent in files

Load balancing problem in «LU» scenario

Host id	Processor	Cores involved	Memory	OS	Perform.
0	Intel Core Quad Q6600 2.4 GHz	**3**	3.8 Gb	Linux	**1.1**
1	Intel Core Quad Q6600 2.4 GHz	**3**	2 Gb	WinXP	**1.0**
2	Intel Centrino Dual-Core 1.466 GHz	**2**	1 Gb	WinXP	**0.6**
3	Pentium 4, 3 GHz	**2** (HyperThreading)	1 Gb	WinXP	**0.6**
4	Pentium 4 2.8 GHz	**2** (HyperThreading)	512 Mb	WinXP	**0.5**





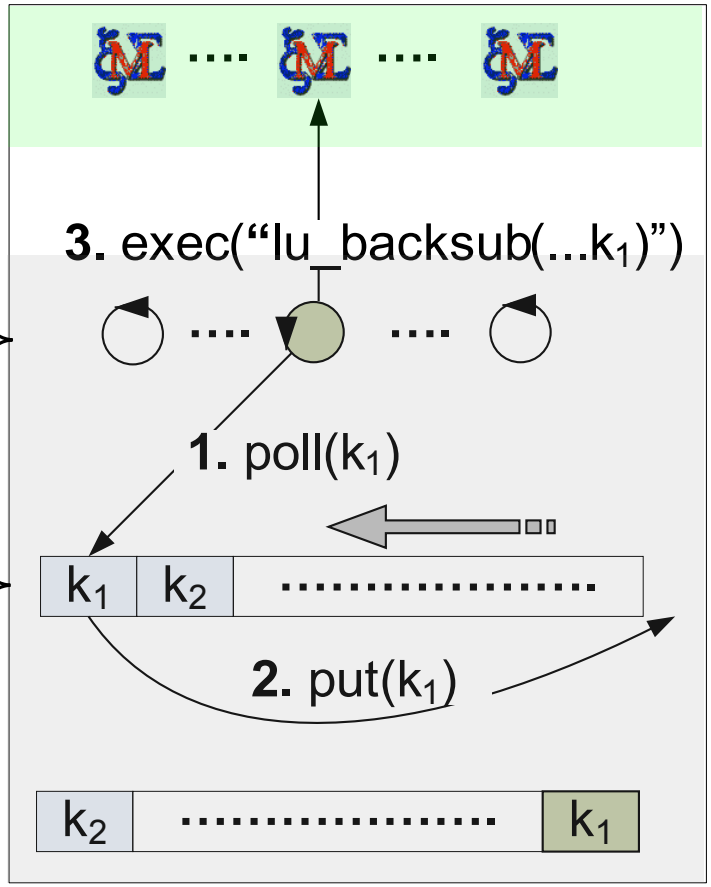
Load balancing implementation at Phase II (inv. columns calc)

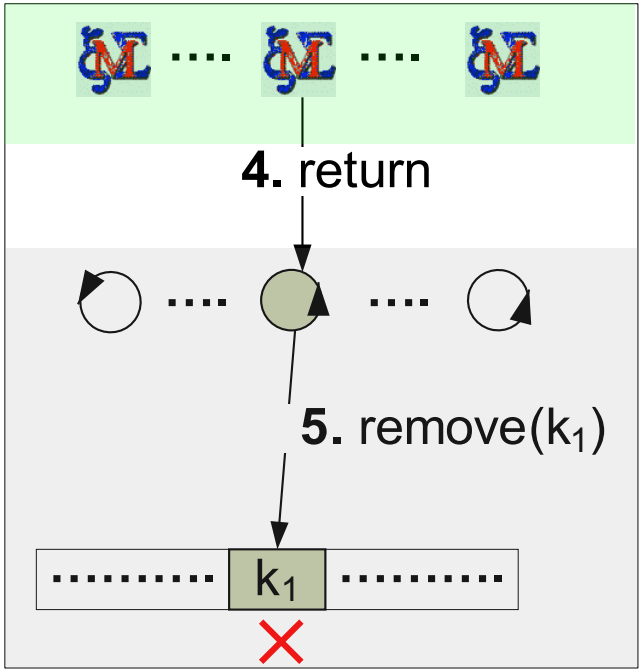
Maxima services

Client application

Worker threads

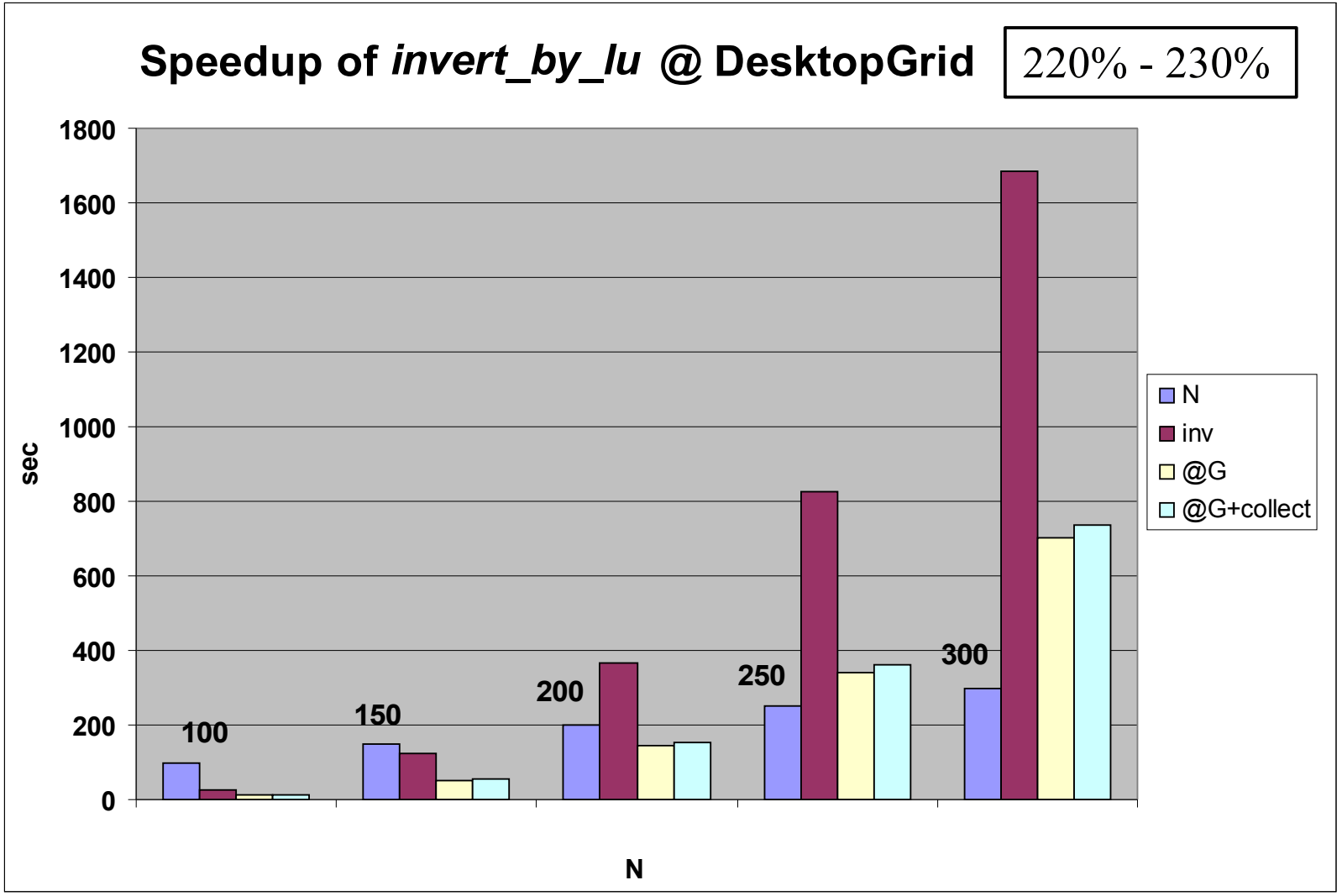
Thread-safe task queue





It turned out, that this heuristic gives ~1.5 times more than optimal schedule of Phase II (in deterministic post facto task assignment problem formulation)

Results of tests of distributed inversion via LU-decomposition



«Collection» of inverse matrix's columns (as files) takes ~ 2.5% of total scenario duration (including network data transmission)

Matrix inversion by Schur complement (1)

There is another inversion approach based on «block decomposition» and Schur complement

$$\mathbf{M}[N \times N],\ \mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{U} \\ \mathbf{V} & \mathbf{B} \end{bmatrix},\ \exists \mathbf{M}^{-1}$$

$$\mathbf{A} = [N_A \times N_A], \mathbf{B} = [(N - N_A) \times (N - N_A)].$$

$$\text{Schur complement: } \mathbf{S} = \mathbf{B} - \mathbf{V}\mathbf{A}^{-1}\mathbf{U}, \quad \text{then} \quad \exists\, \mathbf{S}^{-1} \text{ и}$$

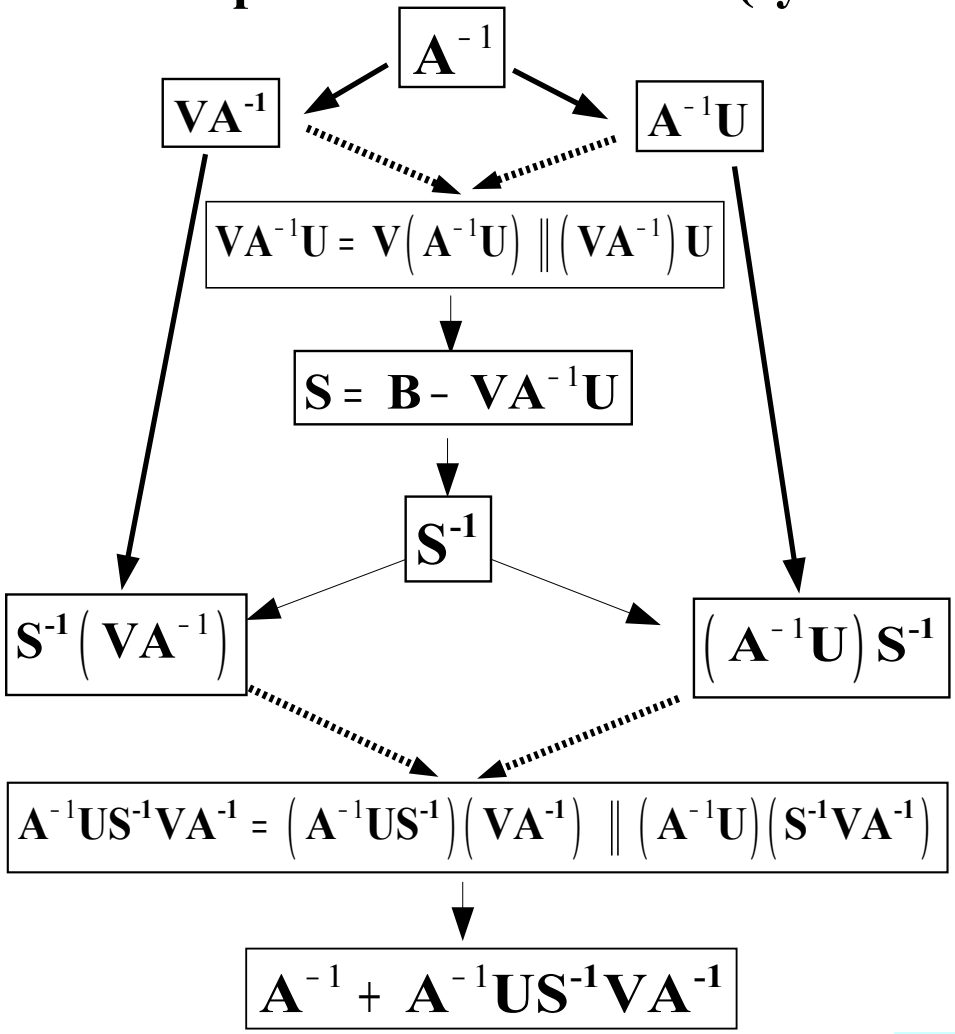
$$\mathbf{M}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{U}\mathbf{S}^{-1}\mathbf{V}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{U}\mathbf{S}^{-1} \\ -\mathbf{S}^{-1}\mathbf{V}\mathbf{A}^{-1} & \mathbf{S}^{-1} \end{bmatrix}.$$

Blocks processing provides more flexible for subsequent parallel and recursive algorithm because calculation of A^{-1} , S^{-1} and matrices multiplications may be parallelized as well.

Speedup evaluation at the next slide.

Matrix inversion by Schur complement (2)

Let M[N×N] be divided into four [$^N/_2$×$^N/_2$] blocks. The cost of inverse matrix blocks' "parallel" calculation (symbol «||») may be evaluated as follows.



A^{-1} => $\sim O((N/2)^3)$

$VA^{-1} \parallel A^{-1}U$ => $\sim O((N/2)^3)$

$VA^{-1}U$ => $\sim O((N/2)^3)$

$B - VA^{-1}U$ => $\sim O((N/2)^2)$

S^{-1} => $\sim O((N/2)^3)$

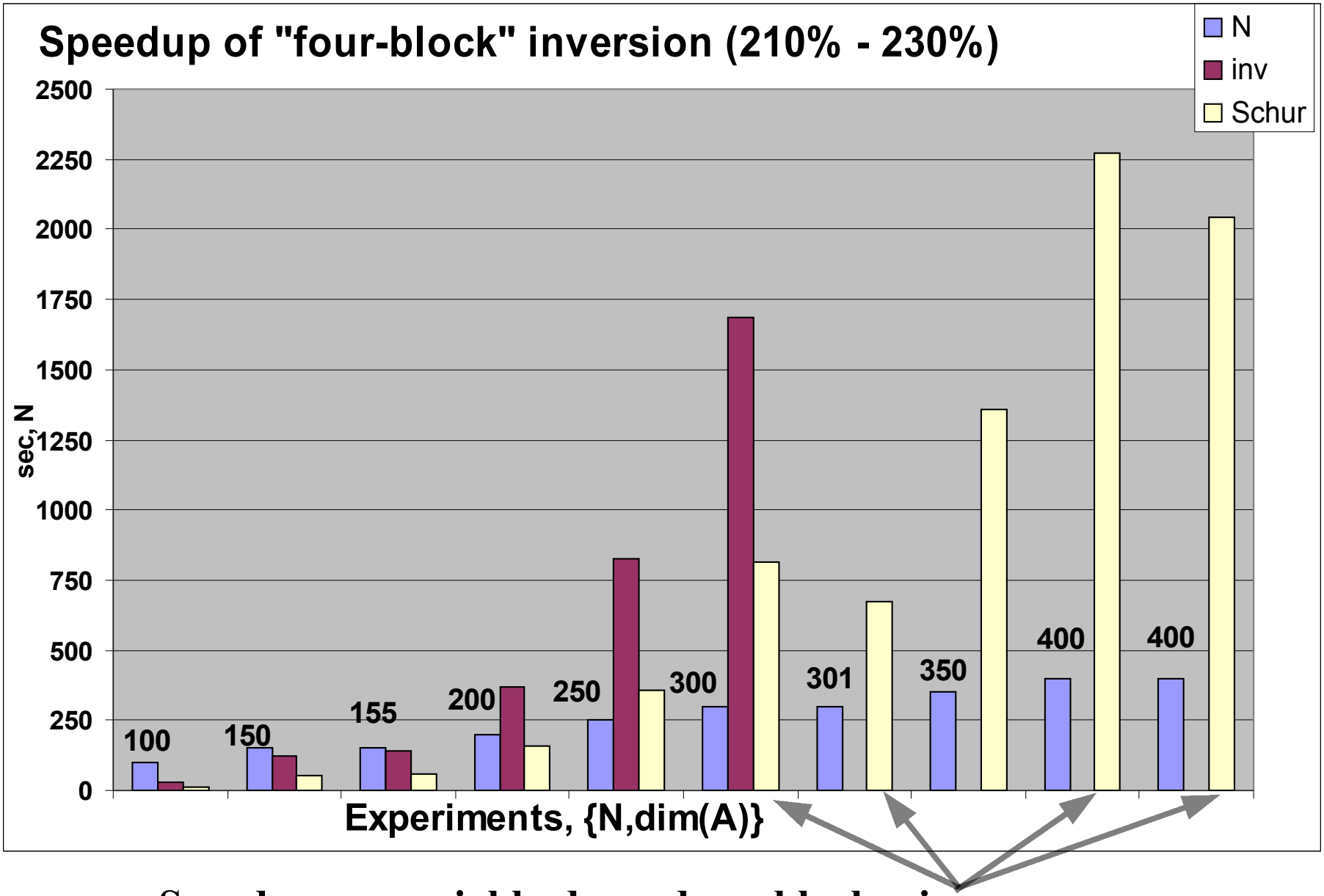
$S^{-1}(VA^{-1}) \parallel (A^{-1}U)S^{-1}$ => $\sim O((N/2)^3)$

$A^{-1}US^{-1}VA^{-1}$ => $\sim O((N/2)^3)$

$A^{-1}+A^{-1}US^{-1}VA^{-1}$ => $\sim O((N/2)^2)$

So, speedup: $\frac{4N^3}{3N^3+2N^2} \approx \frac{4}{3}$ (130% for large N)

Preliminary results of Schur complement approach (standalone «simulation»)



Speedup appreciably depends on blocks sizes.

Thank you!