UDC 5519.852+681.142

EXACT AND GUARANTEED ACCURACY SOLUTIONS OF LINEAR PROGRAMMING PROBLEMS BY DISTRIBUTED COMPUTER SYSTEMS WITH MPI

© **Anatoliy Vasilyevich Panyukov**
South Ural State University, 76 Lenina Ave., Chelyabinsk, 454080, Russia, Doctor of Physics and Mathematics, Professor, Head of Economical and Mathematical Methods and Statistics Department, e-mail: a_panykov@mail.ru

© **Vasiliy Vladimirovich Gorbik**
South Ural State University, 76 Lenina Ave., Chelyabinsk, 454080, Russia, Post-graduate Student of Economical and Mathematical Methods and Statistics Department, e-mail: gorbik@gmail.com

Key words: linear programming; tabular simplex method; distributed computing; parallel optimization; rational computations; arbitrary precision; interval arithmetic.

Techniques of obtaining both exact and guaranteed accuracy solutions of linear programming problems and methods of increasing accuracy of computations by distributed computer systems with MPI are subjects of this paper. To obtain the solutions the rational and arbitrary precision floating point interval arithmetic libraries are applied. Methods of adaptation of the used data types to MPI are presented. Results of computational experiments based on introduced parallel versions of algorithms for solving systems of linear equations and linear programming problems demonstrate effectiveness of their application.

1 Introduction

Unsubstantiated prejudices, causing errors of calculations are widespread. Some of them are: (1) distributing property of associativity of addition and multiplication in the field of real numbers to a finite set of machine "real" numbers; (2) extension of properties of continuous dependence on parameters of solutions of the system received after the "equivalent" changes to the original system. Calculations that ascribe non-existing properties to objects of the numerical analysis, are unproved. Popular commercial packages MatLab, MathCad, etc., and also free package SciLab have the marked disadvantages. Usage of different number of processors in calculations in many cases gives substantially different results, demonstrating the need for evidence-based computing. The potential of available packages supporting symbolic computations does not allow to solve the real problems of mathematical modeling. For the means of arbitrary precision computations GMP library can be used. But GMP library does not provide any interface for using it in parallel computations.

The aim of this work is implementation of exact rational and guaranteed arbitrary precision floating point interval computations software for parallel and distributed computing systems with MPI (Message Passing Interface[1]). The paper covers the usage of mpq_t and mpf_t

data types from the GNU MP library [2], and mpfi_t interval type from MPFI library [3](that is built on the top of GNU MP). An important aspect here is the possibility and effectiveness of adaptation of these types to a multiprocessor environment. MPI interface is an unofficial standard for building distributed computing systems for a long time. Serialization and rearrangement to sequential memory layout of rational and interval arithmetic objects for MPI integration are considered in this paper. We chose GNU MP library for the purposes of exact computations because it is an open source solution available in all widespread GNU/Linux distributions and has a good performance. MPFI library is also an open source project and extends GNU MP; adding interval calculations on top of arbitrary precision floating point data types.

2 Accuracy of Computations

The GNU MP package contains open source GNU MP library for arbitrary precision arithmetic: operations on signed integers, rational numbers and floating-point numbers. GNU MP library is developed for fast operation on both large and small operands. It is fast because it uses whole words as the base type, applies fast algorithms, depending on the size of the operands. It has the optimized assembly code for many types of processors and combines speed with simplicity and elegance of operations.

2.1 Exact Computations with mpq_t Type

```
//   FILE: gmp.h
//Definition of mpq_t
#ifdef __GMP_SHORT_LIMB
  typedef unsigned int mp_limb_t;
#else
#ifdef _LONG_LONG_LIMB
  typedef unsigned long \
    long int mp_limb_t;
#else
typedef unsigned long \
  int  mp_limb_t;
#endif
#endif
```

```
typedef struct
{
  int _mp_alloc;
  int _mp_size;
  mp_limb_t *_mp_d;
} __mpz_struct;

typedef struct
{
  __mpz_struct _mp_num;
  __mpz_struct _mp_den;
} __mpq_struct;

typedef __mpq_struct mpq_t[1];
```

Fig. 1. Declaration of mpq_t type

By the means of mpq_t type exact calculations with rationals are implemented, mpq_t type is based on structures and functions of C library, numerator and a denominator are mpz_struct structures which contain:

- size of allocated memory;
- size of occupied memory;
- pointer to an array representing number.

Code fragment on figure 1 shows mpq_t type declaration.

GNU MP library contains about 40 functions for mpq_t type. Besides it is possible to apply any integers functions to its numerator and denominator separately.

2.1.1 Adaptation of mpq_t Type to MPI

Effective transmission of mpq_t type for MPI environment can be carried out by the means of incomplete serialization. Details of implementation and efficiency estimation are presented in the previous papers [4], [5].

2.2 Arbitrary Precision Floating Point and Interval Computation

In a case when problem has such a scale that it is impossible to use exact computations with rational types, and inaccuracy of solution with hardware floating-point data types fall outside of admissible limits we can use arbitrary precision floating point data types. Effective implementations of such derivative data types, unlike rational, are comparable to the computation time with hardware floating point (with similar length of a mantissa). One of such data types is mpf_t (multiple precision floating-point). It allows dynamically arbitrary change accuracy (the length of mantissa). But computations with mpf_t data type are approximate, and inaccuracy is not considered. Data type mpf_t can be used in algorithms when result verification procedure exists allowing to measure inaccuracy and repeat algorithm from some step with more precision if necessary.

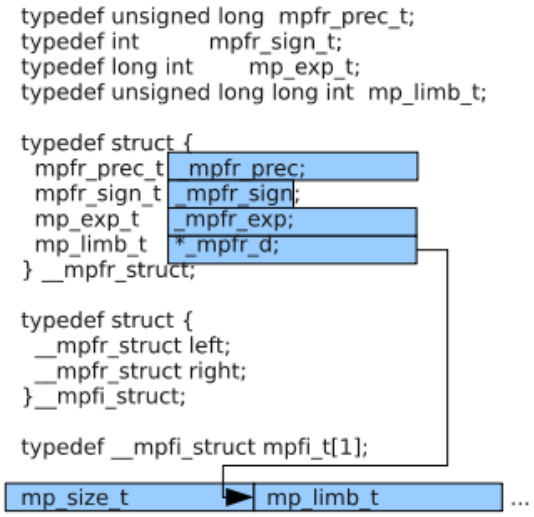

Fig. 2. Structure of mpfi_t type

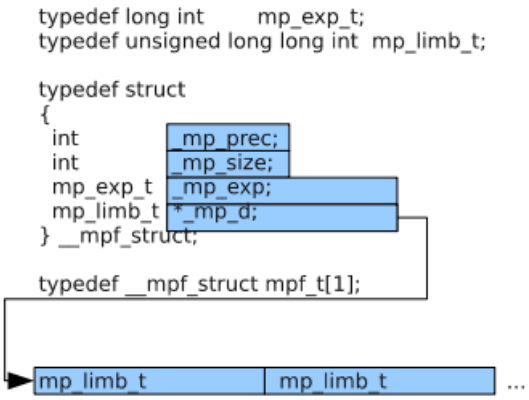

Fig. 3. Structure of mpf_t type

For the means of guaranteed solution one can use data mpfi_t type (from the multiple precision floating-point interval library [3]). The mpfi_t library is based on GNU MP and mpfr (multiple-precision floating point with correct rounding library [6]). Instead of single floating point a number in mpfi is represented by a pair of arbitrary precision floating point values (of mpfr type), they represent lower and upper interval bounds enclosing real value. Unlike mpf, mpfi allows to obtain guaranteed (due to usage of interval calculations) and accurate results (due to arbitrary precision and correct rounding according standard IEEE 754, implemented in mpfr). On figures 2 and 3 structures of data types mpfi_t and mpf_t (for 64-bit architectures) are shown.

2.2.1 Adaptation of mpfi_t and mpf_t Data Types to MPI

The complexity of effective MPI adaptation of derived types with dynamic sizes exist because MPI doesn't provide any interface for passing data types that (1) don't have sequential memory layout or (2) have variable length that could be changed multiple times at runtime. Mantissas of mpfi_t and mpf_t data types are stored out of base structure. In mpf_t structure field _mp_d points to mantissa. In a case of mpfi_t field _mpfr_d pointers of _mpfr structures points to the second elements of the allocated memory blocks (the beginning of mantissa), the current length of mantissa is stored in the first element. Thus, the data can have random memory locations, and it is impossible to define MPI data type on top of mpfi_t and mpf_t types. But still we can perform effective transmission in two ways:

- incomplete serialization;
- rearrangement to sequential memory layout.

In case of incomplete serialization it is enough packing of the necessary structure fields that have been painted over at figures 4 and 5.

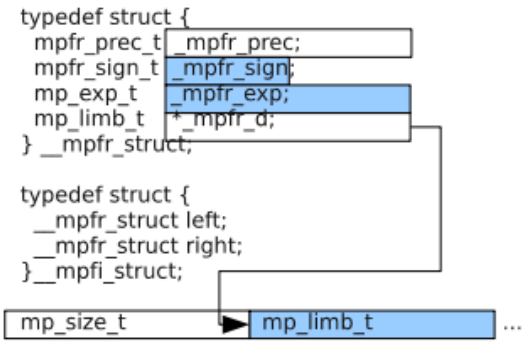

Fig. 4. Serialization of mpfi_t type

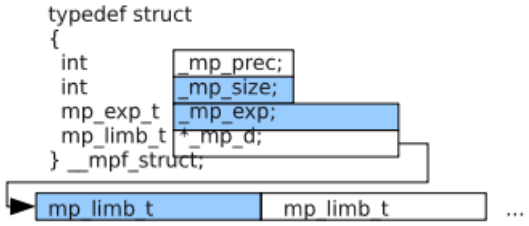

Fig. 5. Serialization of mpf_t type

Depending on the algorithm, if the receiver is not aware of the data types precision of the sender, _mpfr_prec, _mp_prec fields should be also serialized and transfered. In some cases, only _mp_size elements of the mantissa array may be transfered (in the general case _mp_size is equal to the total length _mp_prec).

This approach has obvious disadvantage, that in fact we have to implement the MPI transfer functions by transferring array of bytes (in case of incomplete mantissa transfer - array of bytes previously unknown size), and in case of collective communication it's not always an easy task itself.

2.2.2 Memory Layout Rearrangement for mpfi_t and mpf_t

This approach makes sense in the case when the algorithm operates on the numbers with precision, that is not changed during the computations. Using the macros shown on figure 6,we can define the derived types mpfin_t and mpfn_t based on mpfi_t and mpf_t.

Obviously, the structure of types mpfin_t and mpfn_t, as well as arrays of objects of these types will be allocated in memory sequentially (not including alignment). In this case, all operations (except for initialization and precision set) that are available for mpfi_t (mpf_t), may be applied to mpfin_t (mpfn_t) without any restrictions. Initialization procedure is not

```
#define mpfi_type(prec) \
typedef struct \
{ \
  __mpfr_struct left; \
  __mpfr_struct right; \
  mp_limb_t \
    _mp_d_left[L(prec)+ 1]; \
  mp_limb_t \
    _mp_d_right[L(prec)+1]; \
} __mpfi_struct##prec; \
typedef __mpfi_struct##prec \
mpfi##prec##_t[1]; \
MPI_Datatype MPI_mpfi##prec;
```

```
#define mpf_type(prec) \
typedef struct \
{ \
  int _mp_prec; \
  int _mp_size; \
  mp_exp_t _mp_exp; \
  mp_limb_t *_mp_d; \
  mp_limb_t \
    _mp_d_real[L(prec)+ 1]; \
} __mpf_struct##prec; \
typedef __mpf_struct##prec \
mpf##prec##_t[1]; \
MPI_Datatype MPI_mpf##prec;
```

Fig. 6. Declaration of mpfin_t and mpfn_t types macros

fundamentally different from the original. Instead of memory allocation for mantissa simple initialization of pointers left._mpfr_d and right._mpfr_d (_mp_d) with relevant addresses of mantissa is required (which offset is now constant).

The significant positive factor of this approach is possibility easily declare the MPI data type on top of mpfin_t and mpfn_t types (for any given accuracy), and use all functions available for MPI-derived data types without any restrictions.

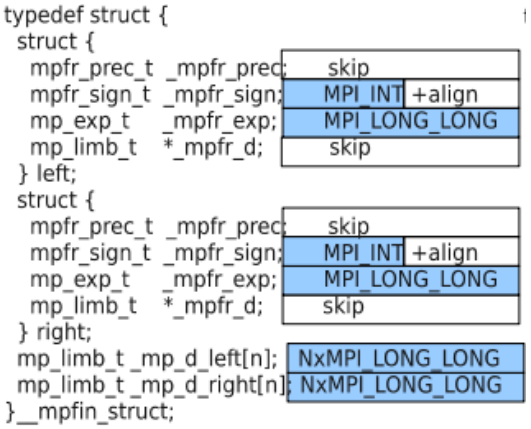

Fig. 7. Fields of mpfi_t structure included in MPI data type

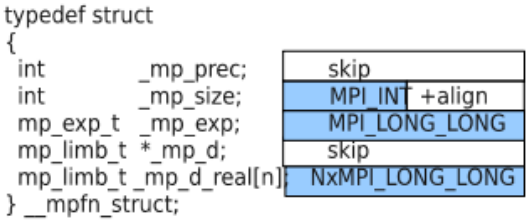

Fig. 8. Fields of mpf_t structure included in MPI data type

Fig. 7 and 8 show mpfin_t and mpfn_t data types structures in terms of MPI (for 64-bit architectures). If one declares MPI data type in a way as shown on fig. 7 and 8 skipping non-colored fields, the pointer to the mantissa won't be rewritten during data receive and no adjustments or additional steps need to be carried out during transfer at all.

Another question is memory layout rearrangement effects for performance. Table 1 shows a comparison of cache misses for initial and modified types. The comparison was made on the processor with 2 MB second-level cache (32 KB first-level) by solving the system of lineal equations by Gauss-Jordan elimination (30 equations and 192 bit mantissa precision).

Table 1

Cache miss test for initial and modified types

	mpfi	**mpfin**	**mpf**	**mpfn**
I refs:	43366674	41850711	13762859	13601951
I1 misses:	23822	21566	10938	10205
L2i misses:	3456	3447	3236	3228
D refs:	18095665	17450618	5076607	5012224
D1 misses:	84427	65950	46662	38923
L2d misses:	13121	12126	9886	9646
L2 refs:	108249	87516	57600	49128
L2 misses:	16577	15573	13122	12874

Test shows that modified types has better cache-hit rate than original. But in closer look on specific functions it turns out that the cache hit rate is better only for functions like comparison, addition, subtraction, etc., but somewhat worse for multiplication and division. There is a slight superiority of the original data types under increasing problem size to 3000 equations and mantissa precision to 1024-2048 bit.

3 Parallel Simplex Method

Simplex-method application for real-world linear program problems keeps beyond comparison in spite of appearance of polynomial algorithms [7]. At present two techniques of simplex-method software engineering are in use:

- tabular simplex-method;
- inverse pivotal matrix method (revised simplex-method).

Preserve generality let us demonstrate its characteristic property with an example linear programming problem

$$\max\left\{c^T x \ : Ax = b \geqslant 0,\ x \geqslant 0\right\}. \tag{1}$$

3.1 Tabular simplex-method

On k-th iteration it re-counts the simplex table

$$\left|\begin{array}{c|c} z^{(k)} = -c^T + c^T_{B(k)} B(k)^{-1} A & c^T_{B(k)} B(k)^{-1} b \\ \hline S^{(k)} = B(k)^{-1} A & x^{(k)} = B(k)^{-1} b \end{array}\right|,$$

here $B(k)$ is the pivotal matrix containing matrix A columns related to the basic variables, $c_{B(k)}$ is criterion function coefficient vector related to the basic variables. At that right simplex table column contains vector $x^{(k)} = B(k)^{-1} b$ of the basic variable values, and the criterion function value $c^T_{B(k)} B(k)^{-1} b$ for current solution. The upper row contains vector $z^{(k)} = -c^T +$

$c^T_{B(k)}B(k)^{-1}$ of relative evaluation replacement for nonbasic variables. Test for optimality of the current basic solution is nonnegativity of vector z.

If optimality test no passed then there is nonbasic variable $x_i:\ z_i^{(k)} < 0$. Let us introduce set $L = \{l:\ S_{li}^{(k)} > 0\}$. If $L = \emptyset$ then the criterion function is unbounded. Otherwise incoming of x_i to basic variables leads to criterion function increment

$$\Delta_i = -\frac{x_{l^*}^{(k)} \cdot z_i^{(k)}}{S_{l^*i}^{(k)}}, \tag{2}$$

here

$$l^* = \arg\min_{l \in L} \frac{x_l^{(k)}}{S_{li}^{(k)}}$$

defines the variable outgoing from basic ones.

Conversion from k-th iteration simplex table to $(k+1)$-th one is realized by Gauss-Jordan elimination for i-th column of the current simplex table. Principal computation capacity is block S converting that requires $\Theta(mn)$ algebraic operations (m is number of rows, n is number of columns of matrix A).

3.2 Inverse pivotal matrix method

On k-th iteration it re-counts matrix $B(k)^{-1}$ that requires $\Theta(m^2)$ algebraic operations ($m < n$). At that for each iteration it is necessary

- to compute basic variables value $x^{(k)} = B(k)^{-1}b$ ($\Theta(m^2)$ algebraic operations);
- to compute dual variables value $y^T = c^T_{B(k)}B(k)^{-1}$ ($\Theta(m^2)$ algebraic operations);
- to check permissibility of the dual solution $c^T \leqslant y^T A$ (no more $O(mn)$ algebraic operations).

If the dual problem is impressible there is nonbasic variable $x_i:\ z_i^{(k)} = -c_i^T + y^T A_i < 0$. If set $L = \{l:\ S_{li}^{(k)} > 0\} = \emptyset$, where $S_i^{(k)} = B(k)^{-1}A_i$ then the criterion function is unbounded. Otherwise incoming of x_i to basic variables leads to criterion function increment defined by formula (2). So application of inverse pivotal matrix method is realized when

- n essentially surpass m;
- matrix A is sparse;
- it is required to solve both primal and dual problems.

3.3 Intercomparison of methods

In the case of tabular simplex method, columns decomposition is preferred due to calculations and communication specific (fig. 9). All the columns $1, 2, \ldots, n$ can be divided in equal proportions between the processes $K = 1, 2, \ldots, N$, the vector of basic variables and the criterion function value are sent to all processes and are processed independently. So the basic steps of one algorithm iteration are following:

Process K				
$S_{00} = Z$	$z_{\lceil \frac{(K-1)n}{N} \rceil + 1}$	$z_{\lceil \frac{(K-1)n}{N} \rceil + 2}$	$\cdots$	$z_{\lceil \frac{Kn}{N} \rceil}$
$S_{10} = X_{B1}$	$S_{1\lceil \frac{(K-1)n}{N} \rceil + 1}$	$S_{1\lceil \frac{(K-1)n}{N} \rceil + 2}$	$\cdots$	$S_{1\lceil \frac{Kn}{N} \rceil}$
$S_{20} = X_{B1}$	$S_{2\lceil \frac{(K-1)n}{N} \rceil + 1}$	$S_{2\lceil \frac{(K-1)n}{N} \rceil + 2}$	$\cdots$	$S_{2\lceil \frac{Kn}{N} \rceil}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$S_{m0} = X_{B1}$	$S_{m\lceil \frac{(K-1)n}{N} \rceil + 1}$	$S_{m\lceil \frac{(K-1)n}{N} \rceil + 2}$	$\cdots$	$S_{m\lceil \frac{Kn}{N} \rceil}$

Fig. 9. Simplex table decomposition

1. Choose the leading column from non-basic coefficients of the objective function (based on some common criteria each process selects from the columns it has).

2. The global exchange between the processes with the values obtained in step 1 and choosing optimal (pivot column for all processes).

3. Process that holds leading column chooses what variable to remove from basic ones (the choice of the leading line).

4. Process holding pivot column globally distribute numbers of the variables being included and excluded from the basic variables, as well as the pivot column.

5. Each process carries out the computation of a new canonical form by the rules of simplex method on the columns it has.

From the above we can conclude that the parallel version of algorithm is not much more complicated than sequential. It uses only minimum number of collective communications. All that should lead to a uniform loading of the system and high efficiency of parallelization [8].

The main difficulty arises when procedure of generating the basic plan is introduced. If simply add the necessary slack and dummy variables, and discard that columns after the first phase, the load will not be uniform. This problem can be solved in two ways:

- Redistribution of columns after the first stage, that is difficult and costly.

- Distribute the matrix in a way that each process got approximately equal number of columns of the original problem and columns that appear during the computation of the initial basic plan. This procedure requires for each process to know how many columns to discard after the first phase.

In the case of revised simplex method, the original matrix must be available to all processes because it is unknown what variables become basic and by which process they will be handled. Additional overhead for communication during computations lead to the fact that one cannot create an simple effective parallel version of the revised simplex method algorithm [9].

3.4 Algorithm of Parallel Simplex Method

This approach to parallelization of simplex method was implemented as MPI program Plinpex (parallel lineal exact solver). It uses rational data types from GNU MP library for exact computations or arbitrary precision floating point interval data types from MPFI library

(depends on compilation flags). A set of gcc 4.4.3 compiler, gdb debugger, efence and valgrind profiler was used. Implementation, testing, and some computational experiments were performed on gentoo gnu/linux based cluster of workstations that we built from computer class resources of department laboratory.

Algorithm Plinpex:

1: for each of N processes initialize MPI environment and identify itself via its MPI *rank*.
2: **if** $rank = 0$ **then**
3: read input problem file in MPS [10] format;
4: parse MPS file and save variable names;
5: initialize and fill matrix A, vector of objective function coefficients c and linear constraints b;
6: expand matrix A with slack and dummy variables for basic plan finding; initialize basic variables vector, dummy objective function;
7: **end if**
8: call MPI barier and initiate main solve function
9: broadcast common problem data from *rank* 0 process (problem; size and the number of slack and dummy variables);
10: evaluate number of main and dummy columns by knowing its *rank* and total number of processes N;
11: **if** $rank <> 0$ **then**
12: initialize memory for the part of A matrix, vectors of linear constraints b, dummy and objective function c, basic variables vector;
13: **end if**
14: broadcast the linear constraints and basic variables vector from *rank* 0 process;
15: **if** $rank = 0$ **then**
16: **for** $i = 1$ to N **do**
17: send part of main and dummy columns of A matrix to *rank* i;
18: **end for**
19: **else**
20: receive its main and dummy columns of A matrix;
21: **end if**
22: call MPI barrier to synchronize of main computation loop
23: **repeat**
24: choose the pivot column from columns this process handles;
25: call MPI all reduce and choose pivot column globally, let's assume the process that handles pivot column has *rank* L;
26: **if** minimum found **then**
27: **if** step one **then**
28: basic plan found, exclude dummy column and replace dummy objective function with primary one;
29: **else**
30: solution is found, goto 42;
31: **end if**
32: **end if**
33: **if** $rank = L$ **then**
34: chose a variable excluding from basic variables and pivot row;

35: **end if**
36: indexes of including and excluding to/from basic variables and broadcast the pivot column from *rank* L to other processes;
37: apply simplex method transformation on columns handled;
38: **if** entering/excluding basic variables are handled locally **then**
39: update basic variables vector;
40: **end if**
41: **until** solution is found
42: **if** $rank = 0$ **then**
43: output the problem solution;
44: **end if**
45: terminate MPI environment and exit.

4 Computational Experiments

Computational experiments were performed on "SKIF Ural" cluster of South-Ural State University. Brief specifications are presented in table 2.

Table 2

The specifications of computational platform

CPU type (per 1 blade)	2 quad core Intel Xeon E5472 3.0 GHz
System Memory (per 1 blade)	8 GB
Network type	InfiniBand (20Gbit/s, max latency of 2 ms)
Operating System	SUSE Linux Enterprise Server 10 x86_64

4.1 Experiment with Guaranteed Accuracy Floating Point Types

Parallel version of Gauss-Jordan elimination algorithm adapted for computing with mpfi_t (mpfin_t) and mpf_t (mpfn_t) was used. The effectiveness of parallelization is shown on fig. 10.

It should also be noted that when precision (mantissa length) is increased the efficiency of parallelization is also increased (table 3).

4.2 Linear Programming Problems Solving Experiment

Linear programming problems from Netlib library [11] were used as input data for computational experiment. This library contains complex problems that are often used for testing linear programming solving software systems. For the experiment we chose problems with various density and ratio.

Results with exact rational mpq_t type are presented on figure 11 , arbitrary precision floating point data types (mpf_t, mpfn_t) and interval data types (mpfi_t, mpfin_t) are shown on figure 12.

5 Conclusion

Methods for the effective application of arbitrary precision floating point (interval) data types in MPI environment in this paper are suggested in the work. Interval arbitrary precision types

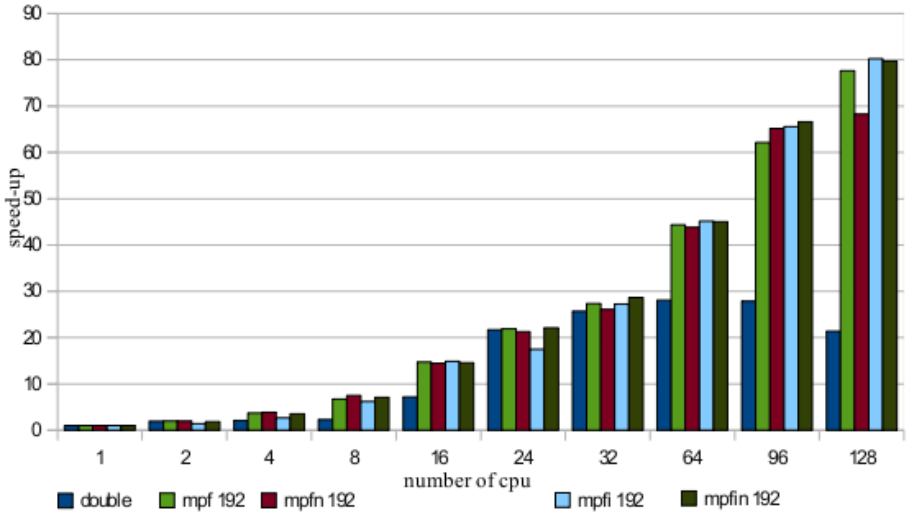

Fig. 10. Effectiveness of Guaranteed Accuracy Floating Point parallelization for Gauss-Jordan elimination

Table 3

Time resources for Gauss-Jordan elimination

	double	mpf			mpfn			mpfi	mpfin
N	**53**	**64**	**192**	**704**	**64**	**192**	**704**	**192**	**192**
1	43.04	1250.99	1817.42	6203.19	1238.50	1796.03	6219.70	3943.99	3817.11
2	22.06	627.28	913.78	3106.40	617.31	910.50	3132.24	2747.88	2073.00
4	20.41	315.35	489.06	1561.31	312.38	463.75	1573.92	1464.61	1072.29
8	18.83	171.44	269.83	795.95	166.03	239.94	804.95	634.02	537.36
16	5.98	86.91	123.35	413.09	85.19	124.01	410.43	265.07	261.93
24	1.98	60.12	83.07	325.72	61.50	84.44	277.53	225.70	172.57
32	1.68	47.70	66.41	212.97	47.30	68.72	220.01	144.66	133.10
64	1.53	27.90	40.98	135.55	28.06	40.97	123.79	87.36	84.79
96	1.54	19.85	29.25	92.61	19.88	27.57	84.55	60.14	57.31
128	2.01	16.86	23.41	77.10	18.53	26.31	76.18	49.15	47.90

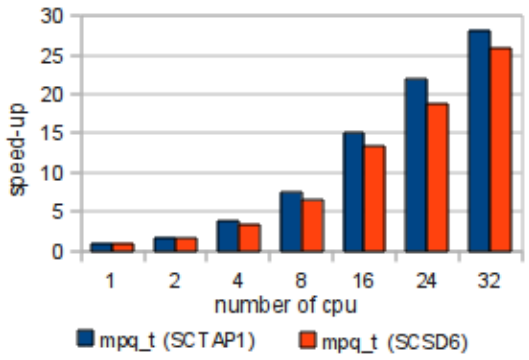

Fig. 11. effectiveness of rational type parallelization (lp)

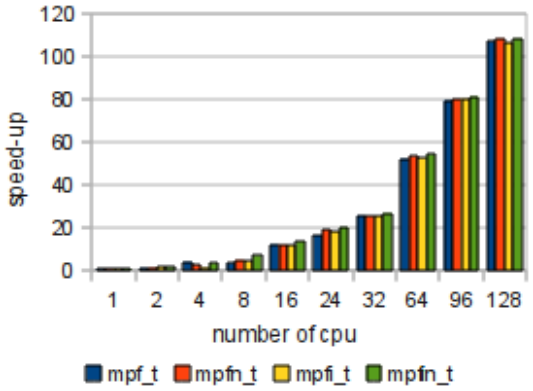

Fig. 12. Effectiveness of arbitrary precision floating point parallelization (LP)

give advantages of obtaining guaranteed results, that include rounding errors which can be analyzed to decide whether current precision of solution is sufficient. Interval arithmetic has main disadvantage of enclosures being too large but with arbitrary precision types we can cope with this problem to some extent and solve even high dimension problems. It should also be noted that the positive effect of parallelization in this case is not only in computation time reduction but also in ability to solve problems of higher dimension, because it is easy enough to reach memory limits, when the matrix will not fit entirely in memory of the single node.

All reviewed commercial programs use basic floating point data types and therefore they can not guarantee the accuracy of solutions. An exception from a number of programs, that does not provide guaranteed results are two open source implementations of simplex method, that use exact rational computations algorithms based on GNU MP library; and that fact inevitably leads to a substantial increase in computation time. However, they do not take advantage of parallel programming techniques which, with a skilful use, reduce the computation time and increase the number of problems that can be solved (problems with more dimensions).

The aim of this work was implementation of exact rational and guaranteed arbitrary precision floating point interval computations software for parallel and distributed computing systems called Plinpex. Software Plinpex use proposed methods for data types adaptation to MPI and parallel simplex method algorithm. The proposed algorithm uses only two collective communication at each iteration of the main computational loop of the program.

The computational experiments showed that the implemented methods are effective for problems of different dimensions, ratio and density. The usage of rational and arbitrary precision floating point interval data types adaptation to MPI gives ability to obtain exact and guaranteed results respectively. Examination of computational experiment result reveals efficiency of implemented parallel simplex method algorithm. According to experiments results the efficiency of parallelization depends on precision and it is about 70-80% (and grows with precision increase), and even higher for exact rational computations. However, the total time of computations can be improved with several algorithm optimizations. It is the subject for the further work.

References

1. MPI: A Message Passing Interface Standard, 1995. URL: http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html.

2. GNU Multiple Precision Arithmetic Library, 2010. URL: http://gmplib.org/.

3. MPFI library, 2008. URL: http://perso.ens-lyon.fr/nathalie.revol/software.html.

4. *Panyukov A.V., Germanenko M.I., Gorbik V.V.* Parallel algorithms for solving systems of linear algebraic equations using calculations without rounding//Parallel Computing Technologies (Pavt'2007). 2007. V. 2. P. 238-249.

5. *Panyukov A.V., Gorbik V.V.* Exact solution of linear programming problems on multiprocessor systems//Parallel Computing Technologies (Pavt'2008). 2008. P. 364-369.

6. MPFR library, 2009. URL: http://www.mpfr.org/.

7. *Pan V.Y., Reif J.H.* Fast and Efficient Parallel Linear Programming and Linear Least Squares Computations //Proceedings of the VLSI Algorithms and Architectures, Aegean Worksho on Computing. Springer-Verlag, 1986. P. 283-295.

8. *Hall Ju.* Towards a practical parallelisation of the simplex method // J. Computational Management Science. 2010. V. 7. N. 2. P. 139-170.

9. *Yarmish G.G.* A Distributed Implementation of the Simplex Method//UMI. Dissertations Publishing, 2001.

10. MPS format, 2008. URL: http://softlib.cs.rice.edu/pub/miplib/mps_format.

11. Netlib library collection, 1996. URL: ftp://netlib2.cs.utk.edu/lp/data.

GRATITUDES: The work is supported by Russian Foundation of Bounded Research (project 10-07-96003-r_ural_a).

Accepted for publication 7.06.2010.

РЕШЕНИЕ ЗАДАЧИ ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ С ПРОИЗВОЛЬНОЙ ТОЧНОСТЬЮ НА РАСПРЕДЕЛЕННЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ С MPI

© **Анатолий Васильевич Панюков**
Южно-Уральский государственный университет, пр. Ленина, 76, Челябинск, 454080, Россия, доктор физико-математических наук, профессор, зав. кафедрой экономико-математических методов и статистики, e-mail: a_panykov@mail.ru

© **Василий Владимирович Горбик**
Южно-Уральский государственный университет, пр. Ленина, 76, Челябинск, 454080, Россия, аспирант кафедры экономико-математических методов и статистики, e-mail: vgorbik@gmail.com

Ключевые слова: линейное программирование; метод симплекс-таблиц; распределенные вычисления; параллельная оптимизация; дробно-рациональные вычисления; произвольная точность; интервальная арифметика.

Предметом статьи являются способы получения как точного решения, так и приближенного решения с гарантированной точностью, а также способы повышения точности вычислений на распределенных вычислительных системах с MPI. Для получения таких решений применяются дробно-рациональные вычисления без округления, вычисления над числами с плавающей точкой произвольной наперед заданной точностью и интервальные вычисления с такими числами. Представлены способы адаптации предложенных типов данных к MPI. Результаты вычислительного эксперимента на разработанных параллельных алгоритмах решения систем линейных алгебраических уравнений и задач линейного программирования показывают эффективность их применения.