MSC 68W10, 65Y04, 65Y05, 65Y10 **DOI: 10.14529/mmp150210**

PARALLEL ALGORITHMS OF INTEGER ARITHMETIC IN RADIX NOTATIONS FOR HETEROGENEOUS COMPUTATION SYSTEMS WITH MASSIVE PARALLELISM

A.V. Panyukov, South Ural State University, Chelyabinsk, Russian Federation, anatoly.panyukov@gmail.com
V.A. Golodov, South Ural State University, Chelyabinsk, Russian Federation, avaksa@gmail.com

For the analysis of huge problems which are very sensitive to the rounding errors, the software providing rational calculations is developed. Software uses MPI interface for communication in the distributed computational environment. Improved efficiency of such software my be achieved by using heterogeneous computation systems. Local arithmetic operations with long numbers may be done in parallel mode with a lot of processes per one operation. This work introduces the research of increasing of the scalability of basic arithmetic operations.

Abilities of the massive parallelism for the heterogeneous computation systems for the efficiency improving are shown. Redundant numerical system with a constant time of the addition operation is introduced. It allows to design well scaled algorithms for all basic arithmetic operations with integer numbers. Scalability of the basic integer arithmetic algorithms is easy applied to rational arithmetic.

Keywords: integer computer arithmetic; heterogeneous computer system; radix notation; massive parallelism.

Introduction

Reliable computations are the necessary [1] and sufficient [2–4] tool for algorithmic analysis of large scale unstable problems. The library "Exact computation" [5] provides such functionality in the distributed computing environment.

Further increasing of effectiveness of such software is possible for account of heterogeneous computing environment allowing to parallelize local arithmetic operations using more than one process for the basic arithmetic operation.

Quality measure of the sequential algorithm is its computational complexity. Less computational complexity leads to less time complexity of the algorithm and its more efficiency. Computational complexity of parallel algorithm is not indicative since different operation may be performed simultaneously. Good quality measure of the parallel algorithm is given by measure of the *strong* scalability of the parallel algorithm is speedup over sequential analogs. The best parallel algorithms provide constant or logarithmic (from the input data size) estimations of the time complexity that leads to $O(n)$ or $O(n/\log_2 n)$ speedup. Linear $O(n)$ speedup usually is given by fully parallel algorithms and $O(n/\log_2 n)$ speedup is given by algorithms that use doubling scheme.

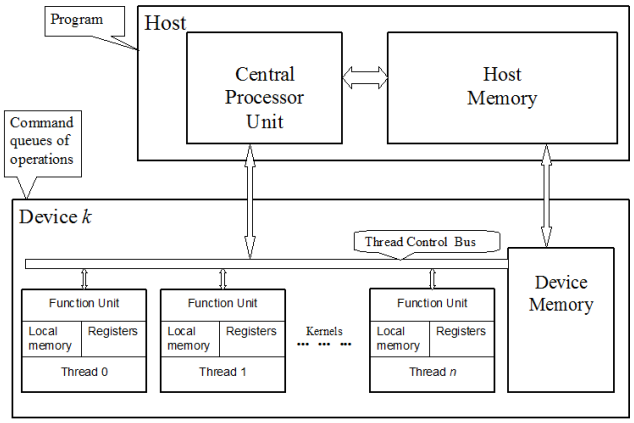
If the processes $k = 0, 1, \ldots, n$ require time t_k^ρ for the performing of operation ρ then time required to complete operation is $t^\rho = \max\{t_k^\rho : \ k = 0, 1, \ldots, n\}$.

The subject of the paper is development of the completely scalable parallel algorithms of the basic arithmetic operation with linear $O(n)$ speedup and the well scalable parallel algorithms with $O(n/\log_2 n)$ speedup.

It is demonstrated that the application of redundant positional notations gives the *completely* scalable addition/substraction algorithms and *relative* scalable algorithms for the rest basic arithmetical operations. The results about scalability algorithms for integer arithmetics that were announced at the conferences [6–8] are presented in the paper.

1. Heterogenous Computation Systems

Structure of heterogenous computation systems is presented on fig. 1.Heterogeneous



Fragment of heterogenous system architect

computation system unit consist of the managing CPUs (host side) and the set of devices (device side). CPU provides operating system functioning and program launch. Devices provide parallel execution of basic operations over the data objects of program.

Data exchange between host and device memory is carried out via PCI bus, On-chip and device-device communication is carried out via DMA (direct memory access). All local on-chip interprocess communications of the "point-to-point" type can be carried out asynchronously. Collective "one-to-all" data exchange may be carried out in two steps. First, some process sends data to the shared thread or shared device memory. Second, recipients read transmitted data. Reading of message from the shared memory may be performed simultaneously by all recipients.

In summary, such geterogeneous system allows to construct a relatively low-cost, high-performance computer with low power consumption. However, the transfer speed between the host CPU and the massive parallel device can become a bottle neck, making it unusable for applications with intensive CPU-Device-CPU data flow.

Massive parallel architecture of the devices requires algorithms with high level parallelism. Less clock rate (than CPU clock) of the device processors also decrease its efficiency for sequential tasks.

Features of the devices may dramatically vary from one to another therefore we consider only pseudocode of algorithms.

2. Analysis of Parallel Algorithms for Integer Arithmetic Operations in the Classical Radix Notation

2.1. Addition of the Nonnegative Number

Classical addition algorithm for n-digit numbers is fully sequential. The possible approach for parallel execution of the addition is two step procedure: first one is parallel digit-by-digit summation, second one is parallel carry propagation from all digits where carries are non zero. The second step can be performed after the first one is completed but the operations at each step are fully parallel.

Algorithm may be implemented by the two fully parallel procedures Digit_Addition and Carry_Propagation. Some global procedure _global_Add is need to define lengths of the summands and create the required quantity of the parallel processes. Further it is required to call procedure _global_Add to get the result of addition $(a_{n-1} \dots a_0)_R + (b_{m-1} \dots b_0)_R$. Lets estimate the time for the execution of such algorithm as well as possible speedup compared with classical sequential algorithm.

It is known that the binary notation is applied for presentation of numbers in computer memory. Therefore the value $R = 2^r$ is accepted as the base of the radix. Here r is word length of the utilized registers. Usually modern computers use 32- or 64-bit registers. We use the noted values of R and r.

Thus, each of parallel processes of the procedure Digit_Addition requires time for the addition of single digits and calculating corresponding digit of the result and carry, denote it as ts. The time of the execution of procedure Carry_Propagation can vary from 0 in the best case to $(n-1) \cdot ts$ in worst case, we neglect overhead for while loop. An Full time of execution of addition algorithm depends on values of input arguments. The full time varies from s to $n \cdot s$ plus overhead on fork of the process. Time of the execution of addition by strictly sequential algorithm (i.e. digit after the digit) in above terms is exactly $n \cdot ts$.

Lets estimate the mean time of the execution of our parallel algorithm on the assumption that the arguments belong to uniform distribution. Assume that $n = m$ for simplicity.

The probability of the non zero carry in one of the processes is equal to

$$p = \mathrm{P}\left\{a_i + b_i \geq 2^r\right\} = \sum_{l=1}^{2^r-1} \mathrm{P}\left\{a_i = l\right\} \mathrm{P}\left\{b_i \geq 2^r - l\right\} = \sum_{l=1}^{2^r-1} \frac{1}{2^r} \cdot \frac{l}{2^r} = \frac{1}{2}\left(1 - \frac{1}{2^r}\right). \quad (1)$$

Probability to have sum of two digits $a_i + b_i = 2^r - 1$ is equal to

$$q = \mathrm{P}\left\{a_i + b_i = 2^r - 1\right\} = \sum_{l=0}^{2^r-1} \mathrm{P}\left\{a_i = l\right\} \mathrm{P}\left\{b_i = 2^r - 1 - l\right\} = \sum_{l=0}^{2^r-1} \frac{1}{2^r} \cdot \frac{1}{2^r} = \frac{1}{2^r}. \quad (2)$$

Consider the probability of carry propagation chain with length $k \geq l$

$$P_l = \mathrm{P}\left\{\bigcup_{i=0}^{n-l-1} \left[(a_i + b_i \geq 2^r - 1) \bigcap_{j=1}^{l} (a_{i+j} + b_{i+j} = 2^r - 1)\right]\right\} = (n-l)pq^l. \quad (3)$$

Modern processors can address up to few terabyte of operating memory but we have probabilities $P_l \leq q^{l-1},\ l = 0, 1, \dots, m$ even for very long numbers when the digit is an r-bit number where $r = 32$ or $r = 64$.

It is easy to see that the value of the probability $P_2 \le q$, therefore, in the asymptotic behavior the mean execution time of the algorithm is equal to $2ts$. Thus, the considered algorithm has at the average $n/2$ speedup in comparison with the sequential addition algorithm.

Time of the addition in the the worst case may de decreased due to the improvement of the carry propagation algorithm. Elements of the chain of sequential carries may be processed in parallel. Carry propagation chains don't overlap.

Algorithm ACP releases advanced carry propagation scheme.

ACP1. Set $L = 1$, assume $V(i) = i$ for each process $i = 0, 1, 2, \ldots, n$.

ACP2. While exist $i = 0, 1, 2, \ldots, n$ with $c_i = 1$, for all $i:\ i \equiv (L-1) mod 2L$ perform steps *ACP3–ACP5*.

ACP3. Set $j = \min\{i + L,\ n - 1\}$.

ACP4.1. If $c_i = 1$ then set $s_{V(j)} = t_{V(j)} + 1 = \left(s^r_{V(j)}\ s^{r-1}_{V(j)}\ \ldots\ s^1_{V(j)} s^0_{V(j)}\right)_2$, $c_{V(j)} = s^r_{V(j)}, t_{V(j)} = \left(s^{r-1}_{V(j)} \ldots s^1_{V(j)} s^0_{V(j)}\right), (\forall k : i < k < V(j))\,(t_k = 0, c_k = 0)\,, V(j) = V(i)$.

ACP4.2. Else if $c_i \ne 2^r - 1$ or $V(i) \ne i$, set $V(j) = V(i)$.

ACP5. Set $L{=}2L$.

ACP6. Stop.

Under joining the low process with index $L2^k$ sends to higher process with index $(L+1)2^k$ absent ripple carry flag mark it as $NotCarry$, itself carry c, and possible ripple carry verge V. High joining process with index $(L+1)2^k$ in the case of the presence of transfer from the low-order fragment $L2^k$ is produced into all necessary digits (from $L2^k + 1$-th to $V((L+1)2^k)$-th). The verge of the propagation of the ripple-through carry in the united fragment is also refined. Excess processes are terminated for all cases.

Lets examine time complexity of addition with advanced carry propagation procedure. This loop is carried out by any of the processes not more than $\lceil \log_2 n \rceil$ times. During the appropriate optimization at heterogeneous environment these operators can be executed in parallel for one tick. Each of the active processes also executes no more than one receiving communication and not more than one sending communication. Their preparation and execution can be carried out for two tics.

Thus, the mean time of the advanced carry propagation scheme is equal to $3s$, and in the worst case it is equal to $3s\lceil \log_2 n \rceil$.

In the asymptotic behavior of the mean time of the execution of addition with the application of advanced carry propagation algorithm is equal to $4s$, i.e., exceeds mean time with the application of simple carry propagation algorithm in $4/3$ times. However, already with the presence of the carry circuits of length of more than two digits the effectiveness of advanced carry propagation algorithm usage is undoubted.

2.2. The Binary Relations

Checking truth of any binary relation $a \rho b\ : \rho \in \{<, \le, =, \ge, >, \ne\}$ may be carried out through the relations $\rho \in \{=, >\}$. Indeed $(a \le b) = \neg(a > b)$, $(a \ne b) = \neg(a = b)$, $(a \ge b) = (a > b) \vee (a = b)$, $(a < b) = \neg(a \ge b)$.

Idea of the algorithm is following. Initially data are split up n separated processes $i = 0, 1, \ldots, n-1$, and p_i and q_i are results of comparisons $p_i = (a_i = b_i)$ and $q_i = (a_i > b_i)$ for fragments $i = 0, 1, \ldots, n-1$.

The k-th iteration of the algorithm handles the fragments associated with the processes of $l2^k$ and $(l+1)2^k$ and combines them to one fragment associated with the process $l2^{k-1}$. Values of p_i and q_i are recalculated by formulas $r = p_{2i+1}\&p_{2i}$, $s = q_{2i+1}|(p_{2i+1}\&q_{2i})$, $p_{2i+1} = true$, $p_{2i} = true$, $q_{2i+1} = false$, $q_{2i} = false$, $p_i = r$, $q_i = s$.

Sequential comparison algorithm requires at mean $n/2$ operations of digit comparison. It is easy to see that palallel algorithm requires $\lceil \log_2 n \rceil$ of parallel steps. Average speedup over the sequential algorithm is $n/2\lceil \log_2 n \rceil$ times.

2.3. Determination of the Number Significant Digits

For the efficient usage of computational resources under execution of arithmetic operations it is necessary to know the number of significant digits.

Number of significant digits of the difference after subtraction can be determined only after operation completion. Therefore, the way to compute completion is an important part of the efficient arithmetics.

Algorithm for this operation is optimized simplified version of the binary relation algorithm which requires $\lceil \log_2 n \rceil$ of parallel steps. So determination of the number of significant digits is used only straight after the subtraction operation and may be performed with $\lceil \log_2 n \rceil$ parallel steps.

2.4. Multiplication of the Multidigit Number on the Digit

Algorithm M calculates the product $(c_n, ..., c_0)_R$ of given non-negative integers $a = (a_{n-1}, ..., a_0)_R$, and $b = (b_0)_R$ represented in the radix notation with the base $R = 2^r$.

M1 [Initialization]. Fork $n+1$ proceses.

M2 [Digit-on-digit multiplication]. Each process $i = 0, 1, \ldots, n-1$ calculates $(x_i^1\ x_i^0)_R = a[i]\,[i] \cdot b$.

M3 [Addtition]. Each process $i = 0, 1, 2, \ldots, n-1$ calculates

$$f[i] = \left\lfloor \frac{x_i^0 + x_{i-1}^1}{R} \right\rfloor,$$
$$c[i] = \left(x_i^0 + x_{i-1}^1\right) \mod R.$$

M4 [Carry propagation]. Each process $i = 0, 1, \ldots,\ n-1$ sets $c[i+1] += f[i]$.

M5 [Finish]. $(c[n]\ c[n-1]\ ...\ c[0])_R$ is result of the algorithm.

Algorithm calculates product of the number b on the digits a_i, $i = 0, \ldots, n-1$ (line 3). Product of two digits is a two-digit number $(x_1\, x_2)_R \le (2^r - 1)^2 = 2^r(2^r - 2) + 1$, i.e. value of the carry x_1 (lines 4,9, and 10) is not more than $2^r - 2$. Therefore there are no carry propagation chains (lines 11, 12, 13. 14) with length more than 1 and we have $t_i \le 2^r - 1$ for all $i = 0, \ldots, n-1, n$.

From the description of Algorithm M it is evident that the expenditures of time for stages *M2-M4* is $tm + 2ts$ where tm is time to calculate the result of the digit on digit multiplication and ts is time to calculate sum of two digits and carry. Thus Algorithm M has speedup n times over sequential algorithm of calculating number on digit product.

2.5. Multiplication of the Two Multidigit Numbers

We can calculate the product of the two multidigit numbers of length n and m correspondingly using Algorithm M and reduction scheme to calculate sum of the partial results.

The time of execution of such procedure consist of time to perform in parallel Algorithm M for all digit of one number and $\lceil \log_2 m \rceil$ additions of $(n + m - 2)$-digit numbers.

Thus, average execution time does not exceed $tm + ts(2 + \log_2 m)$. In the worst case execution time not exceed the value $tm + ts(2 + (n + m)\log_2 m)$.

Algorithm M uses n and there are m digit to multiply first number on. Consequently, the total quantity of created processes is equal to mn.

In the worst case with increasing length of operands the time of execution slightly exceeds the linear function, whereas execution time of the classical sequential algorithm of multiplication of two numbers has the quadratic dependence on length of operands.

2.6. Division

The classical algorithm of the long division cannot be performed in parallel. It requires $(n+m-1)$ sequential operations of the multiplication-subtraction with m-digital numbers. In the papers [7, 9] the solution of the problem of increasing of the effectiveness of the algorithm for the operation of division by means of the Newton's method application is proposed.

In order to divide integer $u = (u[n-1]\, u[n] \ldots u[1]\, u[0])_R$ by integer $v = (v[m-1] \ldots v[0])_R$ it is proposed to find sufficiently precise approximation for number $1/v$, then to multiply it on u, which will give the approximation for u/v. It is obvious that the length of integral answer is not more than $n - m + 1$. Number $1/v$ contains not more than m nonsignificant zeros in the high-order digits, for obtaining the correct result of division it is sufficient that the approximate value of $1/v$ would still contain at least $n - m + 1$ significant digits. Thus, the necessary precision of the value $1/v$ is determined by value R^{-n+1}.

The application of Newton's method to the problem of finding of the root of equation $f(x) = 0$, where $f(x) = v - 1/x$, consists of the sequential calculation of

$$x_{k+1} = (2 - v \cdot x_k) \cdot x_k,\ k = 0, 1, 2, \ldots,$$

where x_0 is initial approximation calculated with the necessary precision. Function $f(x) = v - 1/x$ is twice continuously differentiable and strictly convex when $x > 1$. In this case the Newton's method possesses the quadratic speed of convergence, i.e., a quantity of correctly calculated discharges after the execution of sequential iteration will double. The initial approximation $x_0 = 1/(v[m-1])$ of value $1/v$ has an error

$$\frac{1}{v[m-1] \cdot R^{m-1}} - \frac{1}{v} = \frac{v - v[m-1] \cdot R^{m-1}}{v \cdot v[m-1] \cdot R^{m-1}} \le \frac{1}{v \cdot v[m-1]} \le R^{-m+1},$$

i.e. it correctly calculates m digits. Thus, a quantity of iterations, which it will be necessary to carry out according to the Newton's method, will be not more than the value $4\log_2(n+1) - \log_2 m$.

On the k-th iteration $(k = 0, 1, 2, \ldots, l < \log_2(n+1) - \log_2 m)$ variable x represents $(2^{k+1} - 1)$-digit number. Therefore at this cycle body with aid of parallel algorithms one operation of multiplication of variable x by the m-digit number b, one operation of subtraction of (2^k)-digital numbers, and one operation of multiplication of (2^k)-digital numbers are performed. Consequently, time necessary for fulfilment of the cycle body does

not exceed value $2tm + ts \cdot (2 + \log_2 m + k)$ at the average case and value $2tm + ts \cdot (4 + 2^k k + 2,5 \cdot 2^k + 3m \log_2 m + k)$ at the worst case.

Since

$$\sum_{k=0}^{l} k = \tfrac{l(l+1)}{2}, \quad \sum_{k=0}^{l} 2^k = 2^{l+1} - 1,$$

$$\sum_{k=0}^{l} \left(k \cdot 2^k\right) \le \sqrt{\sum_{k=0}^{l} k^2 \sum_{k=0}^{l} 4^k} = \sqrt{\tfrac{2l^3+3l^2+l}{6} \cdot \tfrac{4^{l+1}-1}{3}} \le 2^{l+1}\sqrt{\tfrac{l^3}{3}},$$

the time of execution in the average and the worst cases will not exceed the values $O\left(\log_2 n \cdot \log_2\left(\tfrac{n+1}{m}\right)\right)$ and $O\left(\tfrac{n+1}{m} \log_2{}^{3/2}\left(\tfrac{n+1}{m}\right)\right)$ respectively.

The multiplication of two n-digit numbers completes the execution of procedure D. This step requires at the average $O(\log_2 n)$ time and $O(n \cdot \log_2 n)$ in the worst case. Thus, the final estimations of the execution time of division in the average and worst cases will be equal respectively $O\left(\log_2 n \cdot \log_2\left(\tfrac{n+1}{m}\right)\right)$ and $O\left(\tfrac{n+1}{m} \log_2{}^{3/2}\left(\tfrac{n+1}{m}\right) + n \log_2 n\right)$.

3. Usage of Sign Radix Notation

The notation system given above is unsigned, digits in the position system on the base R are numbers 0, 1, 2, ..., $R-2$, $R-1$. One of the drawback of this is that it requires the comparison of numbers to find the result of addition and subtraction. It may be avoided by the application of sign radix notation. The digits on the sign radix notation on the base R are integers

$$-\left\lfloor \frac{R}{2} \right\rfloor, -\left\lfloor \frac{R}{2} \right\rfloor + 1, \ldots, -1, 0, 1, 2, \ldots, \left\lceil \frac{R}{2} \right\rceil - 2, \left\lceil \frac{R}{2} \right\rceil - 1.$$

Note that with odd R the number of positive and negative numbers is equal, and with even R a number of positive numbers is less than the number of negative ones.

Designate presentation of the number at the sign position radix notation with base $R = 2^r$ as $(a_{n-1}, ..., a_0)_{\pm R}$, and its digits as $a_i = (a_i^{r-1}\, a_i^{r-2} \ldots a_i^1\, a_i^0)_{\pm 2}$, $i = 0, 1, \ldots, n-1$. High bit of the digit presentation is its sign (0 for positive, and 1 for negative). So digits at the sign radix notation are the objects of the `sign integer` type.

Note that all basic algorithms, for the unsigned numeration systems, with exception of the algorithms of addition / subtraction, will not change. The algorithms of addition / subtraction are united into the common algorithm.

Addition procedure calculates the sum of the unsigned presentations of `signed integer` data type (i.e. for positive numbers in high-order digit zero, for negative numbers in the elder ones digit), and forms the sum and carry into the next digit: in the absence register overflow the carry is zero, the sign of result does not change, but in its presence the sign of result changes to the opposite and the carry of the corresponding sign is formed.

Application of sign position systems simplifies the algorithm of algebraic addition, but it does not solve the problem of ripple-through carries propagation.

The accelerated calculation of the chain of ripple-through carry and its propagation is possible also for unsigned systems. Advantages and disadvantages of the accelerated carry propagation are the same as for unsigned radix notation. The truth of binary relations in the sign systems is easily realized by means of the subtraction operation. The sign of

the number is determined by the sign of high-order digit. Algorithms of definition of a number of significant digits, multiplication and division are the same as in the unsigned radix notation.

4. Usage of Redundant Radix Notation

Parallel algorithms of all arithmetic operations presented above show at the average high speedup over sequential versions. Mean computing time of results of addition, subtraction, multiplication by the single-column number has the value $O(1)$, the mean computing time of binary relations has a value of $O(\log n)$, multiplication and division of the numbers of word length n does not exceed value $O(\log_2^2 n)$. However, in the worst case the computing time of results of any operation with n-digital numbers requires $O(n)$ time with the usual carry propagation and $O(\log_2 n)$ with the accelerated carry propagation. The reason for deviations from the average value is the fact that with the execution of addition (subtraction) for carry propagation there appear the chains of the length of more than one. To exclude the noted precedents to the usage of redundant radix notation [7] is helpful.

The natural n-digital number N in the radix notation with the base R is presented in the form of ordered set of numbers

$$N = (a_{n-1} \ldots a_1\, a_0)_R = \sum_{l=0}^{n-1} a_l R^l, \quad a_{n-1}, \ldots, a_1, a_0 \in \mathbf{D} = \{0, 1, 2, \ldots, R-1\}.$$

This presentation is unique. The uniqueness of presentation is reached because the set of numbers D contains exactly R elements that present the section of natural series including zero. The expansion of the set of numbers D will lead to the expansion of presentation for number N.

Consider the method of expanding of the set of numbers D, which makes it possible to perform the operation of addition (subtraction) in time $O(1)$. Let the computing system use 2^r-bit registers. Let us accept the number base $R = 2^{r-1}$. Therefore any digit a_i has non-redundant representation $\left(0\, a_i^{r-2} \ldots\ a_i^1\, a_i^0\right)_2$. Under redundant representation we suppose that a_i may be presented with possible nonzero delayed carry a_i^{r-1}, so $a_i = \left(a_i^{r-1}\, a_i^{r-2} \ldots\ a_i^1\, a_i^0\right)_2$.

Look over one of the possible methods of realization of addition algorithm. Let the i-th digits of terms have form

$$a_i = \left(a_i^{r-1}\, a_i^{r-2} \ldots\ a_i^1\, a_i^0\right)_2, \quad b_i = \left(b_i^{r-1}\, b_i^{r-2} \ldots\ b_i^1\, b_i^0\right)_2$$

i.e. present binary r-digital numbers. After completing of digit-by-digit sum in each position there will be obtained an $(r+1)$-digital result

$$s_i = a_i + b_i = \left(a_i^{r-1}\, a_i^{r-2} \ldots\ a_i^1\, a_i^0\right)_2 + \left(b_i^{r-1}\, b_i^{r-2} \ldots\ b_i^1\, b_i^0\right)_2 = \left(s_i^r\, s_i^{r-1} \ldots\ s_i^1\, s_i^0\right)_2.$$

Use two elder bits for the transfer into the next digit, and obtain resulting r-digital form

$$\tilde{s}_i = \left(0\ 0\ s_i^{r-2}\, s_i^{r-3} \ldots\ s_i^1\, s_i^0\right)_2 + \left(s_{i-1}^r\, s_{i-1}^{r-1}\right)_2 = \left(\tilde{s}_i^{r-1}\, \tilde{s}_i^{r-2} \ldots\ \tilde{s}_i^1\, \tilde{s}_i^0\right)_2.$$

Thus, executing the operation of addition may be performed fully parallel for all cases.

Presented approach uses a radix notation with the redundant digit set. We use of the redundant bit of the digit as the postponed carry that makes it possible to perform the operation of addition in constant time.

Since the algorithms of multiplication and division are correct in this numerical notation and contain the operation of addition only, i.e. use makes the time of the execution of these operations is not worse than those estimations of the mean time of execution for algorithms examined above.

Disadvantage of the redundant radix notation is in the plurality of the number presentation, which makes effective calculation of binary relations and quantity of significant places possible only after the global carry propagation, that removes the redundancy of representation. Let us recall that in the asymptotic behavior the probability of appearance of additional carry approaches zero.

Conclusion

Effective realization of local arithmetic operations with big numbers requires device with enough amount of random-access memory. If the numbers are so huge that the volume of device's random-access memory is not sufficient then operations may be performed on several devices. Moreover, interface between the central processor and the device or between the devices has restrictions on capacity, latency and bandwith. The effectiveness of arithmetic operations with huge numbers is a subject of further researches. It is possible that implementation of Toom-Cook or Karatsuba rapid multiplication algorithms [9] will be effective for this case.

References / Литература

1. Beaumont O., Philippe B. Linear Interval Tolerance Problem and Linear Programming Ttechniques. *Reliable Computing*, 2001, vol. 6, no. 4, pp. 365–390.
2. Panyukov A.V., Gorbik V.V. Using Massively Parallel Computations for Absolutely Precise Solution of the Linear Programming Problems. *Automation and Remote Control*, 2012, vol. 73, no. 2, pp. 276–290.
3. Panyukov A.V. Exact and Guaranteed Accuracy Solutions of Linear Programming Problems by Distributed Computer Systems with mpi. *Tambov University Reports. Series: Natural and Technical Sciences*, 2010, vol. 15, no. 4, pp. 1392–1404.
4. Panyukov A.V., Golodov V.A.Computing the Best Possible Pseudo-Solutions to Interval Linear Systems of Equations. *15th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Verified Numeric (SCAN'2012, Novosibirsk, Russia, September 23-29, 2012): Book of abstracts*, Institute of Computational Technologies Publisher, 2012, pp. 134–135.
5. Golodov V.A., Panyukov A.V. *Library of Classes "Exact Computation"* Programs, Data Bases and Topologies of VLIS. Official bulletin of Russian Agency of Patients and Trademarks, Moscow, FIPS, 2013. [Голодов, В.А. Библиотека классов «Exact computation 2.0», номер гос. регистрации 2013612818 от 14 марта 2013 г. / В.А. Голодов, А.В. Панюков // Программы для ЭВМ, базы данных, топологии интегральных микросхем. Официальный бюллетень Российского агентства по патентам и товарным знакам. – М.: ФИПС, 2013.]
6. Panyukov A.V., Lesovoi S.Yu. *Using of Massive Parallel Calculations for Integer Arithmetics Realisation. Vol. 2.* Perm, PermGTU, 2010. [Панюков, А.В. Применение массивно-параллельных вычислений для реализации основных операций целочисленной арифметики / Панюков А.В., Лесовой С.Ю. – Пермь: Изд-во ПермГТУ, 2010.]

7. Panyukov A.V. Application of Redundant Positional Notations for Increasing of Arithmetic Algorithms Scalability. *15th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Verified Numeric (SCAN'2012, Novosibirsk, Russia, September 23–29, 2012): Book of abstracts*, Institute of Computational Technologies Publisher, 2012.
8. Golodov V.A. Distributed Symbolic Rational-Fractional Calculations on the Processors of Series of x86 and x64. *Proceeding of international conference "Parallel computational technologies" (Novosibirsk, 2012, on March 26 to 30)*, Chelyabinsk: Publishing center of SUSU, 2012, p. 774.
9. Knuth D.E. *The Art of Computer Programming.* Addison-Wesley Longman, 1981, vol. 2, p. 688.

Received September 16, 2014

УДК 004.222 **DOI: 10.14529/mmp150210**

ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ ЦЕЛОЧИСЛЕННОЙ АРИФМЕТИКИ В ПОЗИЦИОННЫХ СИСТЕМАХ СЧИСЛЕНИЯ ДЛЯ ГЕТЕРОГЕННЫХ КОМПЬЮТЕРНЫХ СИСТЕМ С МАССОВЫМ ПАРАЛЛЕЛИЗМОМ

А.В. Панюков, В.А. Голодов

Для алгоритмического анализа крупномасштабных проблем, чувствительных к ошибкам округления, разрабатывается программное обеспечение, реализующее точные дробно-рациональные вычисления в распределенной вычислительной среде с использованием MPI коммуникаций. Эффективность программного обеспечения может быть увеличена за счет применения гетерогенных вычислительных систем, позволяющих выполнять локальные арифметические операции с числами большой разрядности параллельно большим числом процессов. Работа посвящена повышению масшабируемости алгоритмов основных арифметических операций.

Показана возможность повышения эффективности программного обеспечения за счет применения массового параллелизма в гетерогенных вычислительных системах. Использование избыточной позиционной системы счисления, предложенной в работе, позволяет выполнять операцию алгебраического сложения за константное время, что позволяет построить хорошо масштабируемые алгоритмы выполнения всех основных арифметических операций с целыми числами. Масштабируемость основных алгоритмов целочисленной арифметики легко переносится на дробно-рациональную арифметику.

Ключевые слова: базовые арифметические операции; массивно параллельные системы; гетерогенные системы; позиционные системы счисления.

Анатолий Васильевич Панюков, доктор физико-математических наук, профессор, кафедра «Экономико-математические методы и статистика», Южно-Уральский государственный университет (г. Челябинск, Российская Федерация), anatoly.panyukov@gmail.com.

Валентин Александрович Голодов, кандидат физико-математических наук, доцент, кафедра «Экономико-математические методы и статистика», Южно-Уральский государственный университет (г. Челябинск, Российская Федерация), avaksa@gmail.com.

Поступила в редакцию 16 сентября 2014 г.