Data decomposition for formation aggregation values of hypercube in multiprocessor parallel computing systems

R Uskenbayeva, N Mukazhanov*

Department of Computer Science and Software Engineering, International University of Information Technologies
34 «A»/8 «A» Manas Str./Zhandosov Str., Almaty, Kazakhstan

*Corresponding author's e-mail: mukazhan@mail.ru

Abstract

In this paper possibilities of aggregational values calculation is considered. Aggregational values are the main element of multidimensional operative analytical processing. The main reason of using parallel computing systems in data processing is to increase productivity level. Although, parallel computing systems cannot be used in processing all data types. Data processing algorithms and processing data should be gradually adapted to parallel computing systems' usage. In this regard, data decomposition for formation aggregational values in parallel computing systems in data operative analyzing is considered in this paper. In order to identify dependence between data during the process of decomposition Bernstein's conditions are used. At the same time implemented course calculation of from 1-dimension to n-dimension and parallel computation of course interactions will also be considered.

Keywords: OLAP, multidimensional hypercube, aggregational values, parallel computing, decomposition

1 Introduction

Intensive development of information technologies and their wide range usage in all production spheres requires effective processing of large amounts of information. Accordingly, in order to increase productivity of large amounts of information processing effective multiprocessor parallel computing systems are suggested. Effective usage of high-performance parallel calculation systems requires to solve several important tasks. One of them is to make proved parallel algorithms and data that is being processed by parallel computation system.

Nowadays one of the main directions of information technologies which requires the large amounts of data processing with the help of high-performance computing systems is data operative analytical processing. OLAP (On-Line Analytical Processing) is created in order to perform operative analytical processing of data. This technology is created for working with large amounts of data sources and organizes analytical data as multidimensional hypercube and provides users with required data in the form of hypercube lays. One of the most crucial requirements of operative computation system at certain moment is to quickly calculate, formulate, suggest and store aggregational indexes in huge amounts of data sources. Summing of aggreagtional values is calculated as (sum), average value (avg), minimum value (min), maximum value (max) and e.g. Aggreagtional values size is much more less than original values, therefore request fulfillment of previously computed aggregation data would take less time. Accordingly, data bringing from aggregation increases the rate of demand. Previously computation of aggreagtional values in large amounts of data processing meets operative processing requirements.

Large size of data source which is used in analytical hypercube performance and high quantity of dimensions and dimensional elements which are used in processing, deriving possible aggregational summing require number of operation courses to be performed. In order to do these kind of processes in short time, parallel computations are used in multiprocessor systems. Parallel computing systems usage's opportunities in data operative processing with the help of OLAP are shown in workshops [1-3]. In these workshops decomposition by computation functions is considered. In processing of data parallel computation system data decomposition has to be used as well as decomposition by computation functions.

The main objective of given work is data decomposition for formation aggregational values which are the main element of multidimensional operative analytical processing in multiprocessor parallel computing systems.

2 Decomposition of aggregational values computing operators by data

In decomposition performance aggregation by summing of values and operators of summing would be considered. Summing algorithm and its operators are shown in source [5]. Division of atomic operation groups of independent values and independent operators in order to perform summing parallel is algorithm decomposition. Each of the atomic groups consist of internal groups dependent from each other: operation groups which cannot be used separately or operation groups which cannot be performed separately. Dependence between values and operations groups should be identified in order to divide them into atomic groups.

If large amounts of data is processed by certain atomic algorithm, these kind of data can be divided into separate divisions which can be processed independently from each other and processing would be performed by several performer (processor). Process results can be extracted in this way. That is called decomposition by data. Division for independent parts of numerical values produced by aggregational values in data decompositions will be discussed. The second type of decomposition is division of comuations (operations) through several performers and data identification of certain calculator. That is

called decomposition by computation functions. Decomposition, task division process not always have successful finish. For example, some algorithms work just by one performer (processor). It means that just data parallel processing of atomic divided groups or parallel processing of operations might be performed. Integration to one atomic group of data that cannot be processed separately and operation groups that cannot be performed separately is performed [4].

Fulfillment of certain task consists of activities set and action groups. Each action set can be divided into atomic activities group. If activities are to be parallel performed atomic groups will be performed in several performers simultaneously. In that case they are two performers (processors) in computation system. If output data is taken from input data with similar format at any moment in parallel computing, activities set will be determinant. In opposite way, if different output data is taken from different input data, activities group will be non-determinant. Parallel performance can be used if activities groups are determinant. Bernstein's conditions are used to identify whether program activities determinant or not. Let's try to identify aggregational values calculation determination with Bernstein's condition. Operator is the smallest atomic part of program which includes one or several operands in programming tasks or instructions.

2.1 BERNSTEIN'S CONDITION

They are input and output variableness in activities set of each program. Some activities set might not have such kind of variableness. If there is no any input variableness (data) in aggregational values calculation *NULL* value will be mentioned. Absence of input variableness means that hypercube aggregational values calculation by multidimensional index structure does not have original fact values.

Let's insert notes, activities set in program P, input variableness groups of activities set R(P), (borrowed from "read" in programming), output variableness groups of activities set W(P) (borrowed from "write" in programming). Bernstein's condition will be identified as following for P and Q activities sets [4]:

1) If intersection of W(P) and W(Q) is free group (Ø),
2) If intersection of W(P) and R(Q) is free group (Ø),
3) If intersection of R(P) and W(Q) is free group (Ø).

Performance of P and Q sets will be determinant.

Activities sets being as output data should not use just one variableness in 1-condition.

Variableness used as input data in certain activities sets at the same time should not be used as output data in another activities sets according to the 2 and 3 conditions.

If given conditions are performed P and Q activities sets are not connected with each other. That means atomic groups taken from P and Q activities set are not related. Internal groups of each of the atomic groups might be connected. Thus given sets would be parallel performed if Bernstein condition is enough to identify the determination of activities sets and if it is entirely performed. Certain atomic groups might not be determinant under activities sets consideration, values and operations groups, which are united in atomic groups might be also determinant to one another. Also several processes might be accessed to one atomic group while performance of parallel computing is

happening. Atomic operations groups might be performed separately without being crossed with each other, nevertheless several processes might be accessed to the one atomic data group as mutual resource. These all may create race condition connected with which of the processes accessed to data first which one did it second. In this regard, critical sections will be created in data parallel processing. Critical section is the result of race condition during the performance of program.

Consideration of Bernstein' condition identification of aggregational values computation operations' determination: S_1, S_2, S_3,...., S_n, S_{11}, S_{12}, S_{13},...., S_{1k_n}, ..., $S_{k_n-1k_n}$,...., $S_{all_{111...1}}$, $S_{all_{111...2}}$, $S_{all_{111...3}}$,...., $S_{all_{k_1k_2k_3...k_n}}$ - hypercube aggregational values computing operators. Accordingly, computing operators summing format (1-13 formula) [5]:

$$S_1 = \sum_{i=1}^{j_1} x_i[1], \tag{1}$$

$$S_2 = \sum_{i=1}^{j_2} x_i[2], \tag{2}$$

$$S_3 = \sum_{i=1}^{j_3} x_i[3], \tag{3}$$

$$S_n = \sum_{i=1}^{j_n} x_i[n], \tag{4}$$

$$S_{11} = \sum_{i=1}^{j_{11}} x_i[1][1], \tag{5}$$

$$S_{12} = \sum_{i=1}^{j_{12}} x_i[1][2], \tag{6}$$

$$S_{13} = \sum_{i=1}^{j_{13}} x_i[1][3], \tag{7}$$

$$S_{k_1k_2} = \sum_{i=1}^{j_{k_1k_2}} x_i[k_1][k_2], \tag{8}$$

$$S_{k_n-1k_n} = \sum_{i=1}^{j_{k_n-1k_n}} x_i[k_n-1][k_n], \tag{9}$$

$$S_{all_{111...1}} = \sum_{i=1}^{j_{111...1}} x_i[1][1][1]...[1], \tag{10}$$

$$S_{all_{111...2}} = \sum_{i=1}^{j_{111...2}} x_i[1][1][1]...[2], \tag{11}$$

$$S_{all_{111...3}} = \sum_{i=1}^{j_{111...3}} x_i[1][1][1]...[3], \tag{12}$$

$$S_{all_{k_1 k_2 k_3 \ldots k_n}} = \sum_{i=1}^{j_{k_1 k_2 k_3 \ldots k_n}} x_i[k_1][k_2][k_3]\ldots[k_n] \qquad (13)$$

Lower indexes of summing operators give aggregtional values from 1 and n-dimensional size. If indexes of operators have similar size, they can be dynamically interchanging performed. Dimensions are similar with quantity of dimension performed in lays. Intersection of input and output data of values computing operators can be identified by Bernstein' condition. In order to do this Bernstein' condition will be formulated on computing operators.

Operators are organized as determinant activities groups and they can be performed in pseudo parallel way. According to this, following $S_{all_{111\ldots1}}$, $S_{all_{111\ldots2}}$, $S_{all_{111\ldots3}}$,...., $S_{all_{k_1 k_2 k_3 \ldots k_n}}$ - if we use original aggregational values compuating operators these operations will be performed dynamically interchanged way. All original aggregational values computing operators do not have connected output and input data. $S_{all_{111\ldots1}}$, $S_{all_{111\ldots2}}$, $S_{all_{111\ldots3}}$,...., $S_{all_{k_1 k_2 k_3 \ldots k_n}}$ - operators set will be performed in program, but their parallel performance at the same time has to be considered from to point of Bernstein' condition.

There is no connection between all of operators and we can take any two of them in order to test them by Bernstein' condition because they are equal. If $S_{all_{111\ldots1}}$, $S_{all_{111\ldots2}}$ - if two operators are being performed dynamically one by one in (atomic actions set) program

– If the intersection of $W(S_{all_{111\ldots1}})$ and $W(S_{all_{111\ldots2}})$ is free group ($W(S_{all_{111\ldots1}}) \bigcap W(S_{all_{111\ldots2}}) = \varnothing$),

– If the intersection of $W(S_{all_{111\ldots1}})$ and $R(S_{all_{111\ldots2}})$ is free group ($W(S_{all_{111\ldots1}}) \bigcap R(S_{all_{111\ldots2}}) = \varnothing$),

– If the intersection of $R(S_{all_{111\ldots1}})$ and $W(S_{all_{111\ldots2}})$ is free group ($R(S_{all_{111\ldots1}}) \bigcap W(S_{all_{111\ldots2}}) = \varnothing$), operators $S_{all_{111\ldots1}}$, $S_{all_{111\ldots2}}$ might be performed at the same time in different processors in parallel computing system. This parallel computing is common for $S_{all_{111\ldots1}}$, $S_{all_{111\ldots2}}$, $S_{all_{111\ldots3}}$,...., $S_{all_{k_1 k_2 k_3 \ldots k_n}}$ - all operators.

Disturbing consequence of Bernsten's condition in dynamically performed atomic sets $S_{all_{111\ldots1}}$, $S_{all_{111\ldots2}}$, $S_{all_{111\ldots3}}$,...., $S_{all_{k_1 k_2 k_3 \ldots k_n}}$ - is considered [6]. Disturbing of the first Bernstein' condition for two operators of dynamically performed one by one program set $S_{all_{111\ldots1}}$, $S_{all_{111\ldots2}}$ is being considered.

– If the intersection of $W(S_{all_{111\ldots1}})$ and $W(S_{all_{111\ldots2}})$ is not free group ($W(S_{all_{111\ldots1}}) \bigcap W(S_{all_{111\ldots2}}) =$ is not free),

– If the intersection of $W(S_{all_{111\ldots1}})$ and $R(S_{all_{111\ldots2}})$ is free group ($W(S_{all_{111\ldots1}}) \bigcap R(S_{all_{111\ldots2}}) = \varphi$),

– If the intersection of $R(S_{all_{111\ldots1}})$ and $W(S_{all_{111\ldots2}})$ is free

group ($R(S_{all_{111\ldots1}}) \bigcap W(S_{all_{111\ldots2}}) = \varphi$),

Performance:

1) $S_{all_{111\ldots1}} = \sum_{i=1}^{j_{111\ldots1}} x_i[1][1][1]\ldots[1]$

2) $S_{all_{111\ldots2}} = \sum_{i=1}^{j_{111\ldots2}} x_i[1][1][1]\ldots[2]$

Case when output results is written down on one variableness. Entering output data into one variablness in program might is used to save the memory. By renaming the similar variableness of output data we can easily solve the problem. For example: $S_{all_{111\ldots1}}$ first output variableness is stayed untouched, accordingly the second one will be mentioned like this: $S_{all_{111\ldots2}}$. Nevertheless, while entering outgoing data into one variableness operators stay interdependent and this is called *output dependence*. Output data dependence will not hinder to perform the task in parallel way, but variableness should be renamed. This dependence is usually mentioned like this: $S_{all_{111\ldots1}} \sigma^o S_{all_{111\ldots2}}$, graphic type like this:
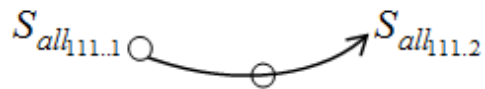

FIGURE 1 Output dependence

The next, disturbing of Bernstein's second condition for two operators (or operators group) $S_{all_{111\ldots1}}$, $S_{all_{111\ldots2}}$ of program set which is performed dynamically one by one is considered.

– If intersection of $W(S_{all_{111\ldots1}})$ and $W(S_{all_{111\ldots2}})$ is free group ($W(S_{all_{111\ldots1}}) \bigcap W(S_{all_{111\ldots2}}) = \varphi$),

– If intersection of $W(S_{all_{111\ldots1}})$ and $R(S_{all_{111\ldots2}})$ is not free group ($W(S_{all_{111\ldots1}}) \bigcap R(S_{all_{111\ldots2}}) =$ not free group),

– If intersection of $R(S_{all_{111\ldots1}})$ and $W(S_{all_{111\ldots2}})$ is free group ($R(S_{all_{111\ldots1}}) \bigcap W(S_{all_{111\ldots2}}) = \varphi$),

Performance:

1) $S_{all_{111\ldots1}} = \sum_{i=1}^{j_{111\ldots1}} x_i[1][1][1]\ldots[1]$

2) $S_{all_{111\ldots2}} = S_{all_{111\ldots1}} + \sum_{i=1}^{j_{111\ldots2}} x_i[1][1][1]\ldots[2]$

Result of $S_{all_{111\ldots1}}$ calculation will be used in $S_{all_{111\ldots2}}$ calculation. If given operators are performed dynamically one by one, they cannot be parallel performed. This kind of dependence is called flow dependence. Flow dependence mark accepted as following $S_{all_{111\ldots1}} \sigma S_{all_{111\ldots2}}$ and Figure 2 shows its graphic format.

FIGURE 2 Flow dependence

The next, disturbing of Bernstein's third condition for

19

two operators (or operators group) $S_{all_{111...1}}$, $S_{all_{111...2}}$ of program set which is performed dynamically one by one will be considered.

– If intersection of $W(S_{all_{111...1}})$ and $W(S_{all_{111...2}})$ is free group ($W(S_{all_{111...1}}) \bigcap W(S_{all_{111...2}}) = \varphi$),

– If intersection of $W(S_{all_{111...1}})$ and $R(S_{all_{111...2}})$ is free group ($W(S_{all_{111...1}}) \bigcap R(S_{all_{111...2}}) = \varphi$),

– ($R(S_{all_{111...1}}) \bigcap W(S_{all_{111...2}})$ = is not a free group),

Performance:

1) $S_{all_{111...1}} = \sum_{i=1}^{j_{111...1}} x_i[1][1][1]...[1] + S_{all_{111...2}}$

2) $S_{all_{111...2}} = \sum_{i=1}^{j_{111...2}} x_i[1][1][1]...[2]$

Result of $S_{all_{111...1}}$ calculation will be used in $S_{all_{111...2}}$ calculation, $S_{all_{111...2}}$ value will be identified in following calculation. If aggregational values are calculated first time and operators $S_{all_{111...1}}$, $S_{all_{111...2}}$ are performed in set in one performer, in calculation $S_{all_{111...1}}$ value of $S_{all_{111...2}}$ will be equal to 0. In following calculations value of $S_{all_{111...2}}$ value which is taken from one iteration will be used.

If operators $S_{all_{111...1}}$ and $S_{all_{111...2}}$ calculation is performed in different performers, operator $S_{all_{111...1}}$ use the one iteration earlier value of operator $S_{all_{111...2}}$ and in that case parallel performance can be entirely completed. Before using hyppercube formation aggreagtional values taken from atomic activities $S_{all_{111...1}}$, $S_{all_{111...2}}$, $S_{all_{111...3}}$,...., $S_{all_{k_2k_3...k_n}}$ will be equal to 0. Values taken from atomic activities will be identified after each iteration process ends and after iteration all values will be distributed to performers then will be copied to their local memory. Parallel computing will be continued after all this operations end. This type of dependence is called antidependence. Antidependence mark is accepted as following $S_{all_{111...1}} \sigma^{-1} S_{all_{111...2}}$ its graphic format is following (Figure 3):

FIGURE 3 Antidependence

Other types of dependence are used in set program parallel performance except the ones which are mentioned above. They are: connected with condition, by recursion and etc.

2.2 CYCLIC OPERATIONS AND THEIR PARALLEL PERFORMANCE IN CALCULATION OF AGGREGATIONAL VALUES HYPERCUBE

Large amounts of data processing perform consistently in

number of program. It is the main reason of careful attention on set program parallel performance by data. In aggregational values calculation structure which is characterized above is basically performed in format of massive. Massive should be divided into parts which are processed separately by certain users in parallel performance by data. In massive processing courses are used. It means that courses impact on courses parallel performance of dependence and data massive dependence should be identified.

One simple dimensional and multidimensional completed courses are used in aggregational data calculation. By accordance of indexes taken from original fact data summing of values jointed in one massive are performed like one dimensional loop. All original aggregational values will be jointed into one dimensional massive and processing will be performed by one dimensional course. In formation of certain hypercube loop quantity performed in calculation of original aggregational values is equal to multiplication of all dimensions elements of hypercube or it might be less than them $N_r \le k_1 \times k_2 \times k_3 \times ... \times k_n$. All one dimensional loops are free from each other. It was discussed above and they can be performed dynamically divided into processors. But iterations of loops might be interdependent. Calculation of original aggreagtional values operators by joining all aggregational values into same multidimensional indexes $S_{all_{111...1}}$, $S_{all_{111...2}}$, $S_{all_{111...3}}$,..., $S_{all_{k_1k_2k_3...k_n}}$ is considered:

1) for (int i = 1; i $\le$ $j_{111...1}$; i++)
2) {
3) S_1: $S_{all_{111...1}} = S_{all_{111...1\,i-1}} + x_i$;
4) }

For parallel performance of given loop its iterations should be divided into independent parts. In order to identify dependence between iterations loop is considered:

1) S_1^1: $S_{all_{111...1}}^1 = S_{all_{111...1}} + x_1$;
2) S_1^2: $S_{all_{111...1}}^2 = S_{all_{111...1}}^1 + x_2$;
3) S_1^3: $S_{all_{111...1}}^3 = S_{all_{111...1}}^2 + x_3$;
4)....
5) $S_1^{j_{111...1}}$: $S_{all_{111...1}}^{j_{111...1}} = S_{all_{111...1}}^{j_{111...1}+} + x_{j_{111...1}}$;

Operator $S_{all_{111...1}}$ - $S_{a\,l_1\,l_{1\,i-1}}$ which is calculated one iteration before as incoming value and iteration step value x_i will be considered in each iteration. Marking S_k^i in loop iteration i operator S_k (or operator S_{all_k}) are using for reincarnation. Set operators group data independence can be processed in open loop. Dependence by output data can be easily sorted by renaming variableness. It is mentioned above. Our next step is identification of dependence of input data in loops. If value $S_{all_{111...1}}^i$ is used as output data in one loop iteration in aggregational values calculation it is also used as input data. It shows that dependence which performs summing operations exists in course iteration. If loop

iterations are performed by certain users, Bernstein's condition disturbing will be noted. Input data dependence performance is considered. If iteration variableness values are " $j_{111...1}$ ", λ and κ and if they are equal to $1 \le \lambda \le j_{111...1}$, $1 \le \kappa \le j_{111...1}$ source of dependence will be the following S_1^{κ} ($S_{all_{111...1}}^{\kappa}$) (massive element – as output variableness),

S_1^{λ} ($S_{all_{111...1}}^{\lambda}$) – sink of dependence (massive element – as

input variableness). Therefore value $D = \lambda - \kappa$ is being calculated. This value is considered as loop dependence distance [4].

Loop parallel needs the dependence distance to analyzing and division. Dependence distance value gives the opportunity to identify type of dependence between data for performers and to divide iteration space into independent parts.

Dependence distance is identified to given loop iteration parallel performance. For example, iterations S_1^2 and S_1^3 use value $S_{all_{111...1}}^2$. This value is output variableness of iteration S_1^2, source of dependence, but in iteration S_1^3 – it is considered as input variableness, or sink of dependence. Accordingly, $\lambda = 3$, $\kappa = 2$. Dependence distance is $D = \lambda - \kappa = 1$.

If dependence distance is equal to $D > 0$, flow dependence will take place between course iterations. If it is equal to $D > 1$ course will be parallel performed in processors not more than D.

If dependence distance is equal to $D < 0$, antidependence will take place between loop iterations. If course iteration performance has required before input data is copied into performers, each iteration of loop can be performed in parallel performance.

If dependence distance is equal to $D = 0$, dependence between courses will not be identified. Each of iteration of loop makes parallel performance to performers.

It is known that several dimensions are used in hypercube. Multidimensional hypercube is given in the form of multi-dimensional massive, implemented loop types are used in program processing of multidimensional data massive. Implemented loops might consist 2 or more internal loops. Internal loops quantity in aggregational values calculation which are used in each of the implemented loops are connected with dimension quantity in hypercube lays production. Implemented loop for n-dimensional hypercube is considered:

```
1: for ( int i₁ = 1; i ≤ k₁ ; i₁ ++ ) {
2: for ( int i₂ = 1; i ≤ k₂ ; i₂ ++ ) {
3: for ( int i₃ = 1; i ≤ k₃ ; i₃ ++ ) {
4: ...
5: for ( int iₙ = 1; i ≤ kₙ ; iₙ ++ ) {
```

6: $S_{all_{row}}^{i_1 i_2 i_3 ... i_n} = S_{all_{row}}^{i_1 i_2 i_3 ... i_n - 1} + $ x[i_1][i_2][i_3]...[i_n];

7: $S_{all_{col}}^{i_n ... i_3 i_2 i_1} = S_{all_{col}}^{i_n ... i_3 i_2 i_1 - 1} + $ x [i_n]...[i_3][i_2][i_1];

```
8: }
9: }
10: }
11: }
```

Each of iterations in n-dimensional implemented loop are identified by all value group i_1 , i_2 , i_3 ,..., i_n of calculators. Value group of calculators which is used in each of the iteration is called n-dimensional vector $I = (i_1, i_2, i_3 ,..., i_n)$, iterational vector. All value groups in iterational vector form iterational space. We can observe the relations order between vectors in iterational space. If $\forall k$, $1 \le k \le n$ and $i_k = j_k$, it means that $I = J$, also $\exists s\ 1 \le s \le n$, $\forall k\ 1 \le k \le s$ and if $i_s < j_s$ is I < J [4, 5].

Space coordination is defined by iterational vectors when hypercube data is offered in multidimensional space. According to multidimensional index structure one aggreagtional value is given to each iteration vector. They are two operators in given loop: for aggreagtional calculation by line and column. In operators calculation process output variableness taken from one course ($S_{all_{row}}^{i_1 i_2 i_3 ... i_n}$) is used

as input variableness in next course ($S_{all_{row}}^{i_1 i_2 i_3 ... i_n - 1}$). Disturbing of Bernstein'conditions is shown in this situation. Also, when iterational vectors x[i_1][i_2][i_3]...[i_n] and x [i_n]...[i_3][i_2][i_1] are brought out, they will have common values. It means, operators $S_{all_{row}}^{i_1 i_2 i_3 ... i_n}$ and $S_{all_{col}}^{i_n ... i_3 i_2 i_1}$ are inter-sected by input data, although while using appropriate there is no any order in vectors accessing and no race condition occasion, that is why there is no any interference from iterational vectors in parallel performance. The main task is to divide iteration vectors into private parts in parallel calculation performance. We have to identify dependence distance which is mentioned above, in order to make it in real. Dependence distance for multidimensional courses will be implemented by appropriate indicators of one dimensional courses dependence distance. It is called dependence distance of vectors: $D = \Lambda - K$. For each iterational variableness vector following conditions $(1,1,1,...,1) \le K \le (k_1, k_2, k_3 ,..., k_n)$, $(1,1,1,...,1) \le \Lambda \le (k_1, k_2, k_3 ,..., k_n)$ stay unchangeable.

Hypercube lays are usually taken by two dimensions. Parallel performance opportunities in aggreagtional values calculation which are taken from dimensional elements belonged to each of lays calculation of aggreagtional values in line and column by multidimensional loop are considered. Next, dependence distance of vectors will be defined by two dimensional loops.

```
1: for ( int i₁ = 1; i ≤ k₁ ; i₁ ++ ) {
2: for ( int i₂ = 1; i ≤ k₂ ; i₂ ++ ) {
```

3: $S_{all_{row}}^{i_1 i_2} = S_{all_{row}}^{i_1 i_2 - 1} + $ x[i_1][i_2];

4: $S_{all_{col}}^{i_2 i_1} = S_{all_{col}}^{i_2 i_1 - 1} + $ x [i_2][i_1];

```
5: }
6: }
```

Loop steps are specified in order to identify dependence distance of vectors:

1: $S^{11}_{all_{row}} = S^{11-1}_{all_{row}} + \text{x}[1][1];$

2: $S^{11}_{all_{col}} = S^{11-1}_{all_{col}} + \text{x}[1][1];$

3: $S^{12}_{all_{row}} = S^{11}_{all_{row}} + \text{x}[1][2];$

4: $S^{21}_{all_{col}} = S^{11}_{all_{col}} + \text{x}[2][1];$

5: $S^{13}_{all_{row}} = S^{12}_{all_{row}} + \text{x}[1][3];$

6: $S^{31}_{all_{col}} = S^{21}_{all_{col}} + \text{x}[3][1];$

7: ...

8: $S^{21}_{all_{row}} = S^{21-1}_{all_{row}} + \text{x}[2][1];$

9: $S^{12}_{all_{col}} = S^{12-1}_{all_{col}} + \text{x}[1][2];$

10: $S^{22}_{all_{row}} = S^{21}_{all_{row}} + \text{x}[2][2];$

11: $S^{22}_{all_{col}} = S^{12-1}_{all_{col}} + \text{x}[2][2];$

12: ...

We will identify dependence distance between vectors in order to parallel performance of iteration of given multidimensional course. For example, $I = (1,2)$ and $I = (1,3)$ – vector iterations use $S^{12}_{all_{row}}$ value. This value is output variableness in vector iteration $(1,2)$, dependence source, input variableness in vector iteration$(1,3)$ is used as dependence sink. Accordingly, $\Lambda = (1,3)$, $K = (1,2)$. Dependence distance is $D = (0,1)$. Identification of dependence by data and parallel performance by distance of vectors is complicated issue. To find the solution to this issue, we use vector direction. [В.Е. Карпов. Введение в распараллеливание алгоритмов и программ]. Identification of vector directions d as following:

$$d_i = \begin{cases} "=", D_i = 0; \\ ">", D_i < 0; \\ "<", D_i > 0. \end{cases} \qquad (14)$$

Vector direction $d = ("=","<")$ is identified by given two dimensional loops [4]. Data dependence might be identified by vector directions. First one is suggested like output value, then it is used as input value in the next iteration of the loop, after all it is jointed to massive element. It is called fact dependence. If the internal course is suggested like whole operator, all dependences will be stayed in this operator, then

parallel performance could be handled by external course iterations. Parallel performance cannot be completed by internal loop and two loops in that case of dependence. If we change internal and external operators and leave calculation unchangeably, dependence distance of vector will be $D = (1,0)$, vector direction will be $d = ("<","=")$. Type of dependence stays without change. In this situation parallel performance by internal loop might be completed. Data dependence by multidimensional loop vector directions types is identified, parallel dependence might be performed by identified dependence.

If multidimensional loop structure consists of vector directions elements "<" and "=". This kind of course can be parallel performed by index number which is appropriate to vector direction component "=" without any limitation. Parallel performance by vector direction component appropriate to index may cause some problems. If multidimensional loop structure consists of vector directions elements ">" and "=". This kind of course can be parallel performed by index number which is appropriate to vector direction component "=" without any limitation. In parallel performance by appropriate indexes to vector direction component ">" input data movement might be required. Before course parallel performance implementation of original structure might be placed in appropriate save places.

If vector direction taken from multidimensional course is $d = ("=",...,"=")$, it will be *loop independent dependence* and dependence type is fact dependence. Parallel performance might be completed by any component of iterational vector. Parallel performance can be completed by changing of implemented course levels.

3 Conclusions

In this paper data decomposition formulation for aggregational values in high-performance parallel computing systems is suggested. Aggregational values are based on data operative analytical analyzing. When decomposition ends algorithm will introduce groups consisted of operations (activities) which are performed by several processors. Group operations brought via decomposition might be performed independently through certain processor. Different operations might be held in each of the group and they can be performed by different user. Parallel performance of implemented courses iteration which is used in program performance of multidimensional data processing is also considered.

References

[1] Arres B, Kabbachi N, Boussaid O, Boussaid 2013 Building OLAP cubes on a Cloud Computing environment with MapReduce. *Conference: Computer Systems and Applications (AICCSA)*

[2] Nandi A, Yu C, Bohannon P, Ramakrishnan R 2011 Distributed cube materialization on holistic measures *International Conference on Data Engineering-ICDE*

[3] Kuznecov S, Kudryavcev Y 2009 *Applying Map-Reduce Paradigm for Parallel Closed Cube Computation*

[4] Karpov V 2010 Vvedenie v rasparallelivanie algorimov i programm

Comuternye issledovaya i modelirovanie **3**(2)

[5] Uskenbayeva R K, Cho Y I, Bektemyssova G B, Mukazhanov N K, Kozhamzharova D K, Kurmangaliyeva B K 2014 Multidimensional indexing structure development for the optimal formation of aggregated indicators in OLAP hypercube *Proceedings of the 14th International Conference on Control, Automation and Systems (ICCAS 2014)* Gwangju, Korea

[6] Matthew D A 2010 *Data-driven decomposition of sequential programs for determinate parallel execution* University of Wisconsin-Madison

Authors

Raissa Uskenbayeva, 1953, Kazakhstan

Current position, grades: Vice-rector on the academic affairs, professor department of CSSE
University studies: International University of Information Technologies
Scientific interest: Macro and micro-economics, finance and banking, Industrial Automation and Control Theory, Marketing, Management and logistics, Information technology and software engineering, Informatics problems
Publications: more than 100 scientific articles, monographs on Theory of control and automation industry, Information Technology and Systems, Reliability of mathematical and software IP
Experience: 2012- at present Almaty: International University of Information Technologies, Vice-rector on the academic affaires, 2003-2012 Almaty: Kazakh National Technical University after K. I. Satpaev, The head of the Depart. «Software of systems and nets» of the Institute Information Technologies, The Doctor of Science, professor, 1999-2003 Almaty: Doctorate at Kazakh National Technical University after K. I. Satpaev, 1987–1999 Almaty: Kazakh National Technical University after K. I. Satpaev, The Candidate of Technical Science, Docent, 1981–1987 Almaty: Kazakh National Technical University after K. I. Satpaev, tutor, 1978-1981 Almaty: Research Institute of the State Planning Committee of the KazSSR, The Junior Research Fellow, 1975–1978 Almaty: Almaty Special. Design Bureau of The Ministry of Telecommunications Industry USSR, Engineer-mathematic.

Nurzhan Mukazhanov, 1986, Kazakhstan

Current position, grades: Ph.D student
University studies: International University of Information Technologies
Scientific interest: Database, Information and analytical systems, Distributed data processing, Decision support systems, Artificial intelligence
Publications: more than 10 scientific articles on Distributed data processing, Information and analytical systems
Experience: 2012 at present Almaty, International University of Information Technologies, Ph.D student, 2010–2012 Almaty: Kazakh National Technical University after K. I. Satpaev, tutor, 2008–2010 Almaty: Kazakh National Technical University after K. I. Satpaev, studied in magistracy.