ISSN 1512-1232

A New Approach to Constructing Parallel Algorithms

Natela Archvadze[1], Merab Pkhovelishvili[2], Lia Shetsiruli[3]

[1.] Department of Computer Sciences Faculty of Exact and Natural Sciences Ivane Javakhishvili Tbilisi State University, 2, University st., 0143, Tbilisi, GEORGIA, natela.archvadze@tsu.ge

[2.] Georgian Technical University, Institute Computation Mathematics, 7, Akuri st., Tbilisi, GEORGIA, merab_4@list.ru

[3.] Department of Mathematics and Computer Science Shota Rustaveli State University, 35, Ninoshvili st., 6010, Batumi, Georgia, likalika77u@yahoo.com

Abstract

The traditional parallel algorithms of data processing based on method: (for some principal the number of cores, a speed of processing etc.) separate all information which should be processed into parts and then process each part separately on a different cores (or processor). It takes quite a long time and is not always an optimal solution to separate into a parts. It's impossible to bring to minimum a standing of cores. It's not always possible to find optimal choice (or changing algorithm during execution). The algorithms provided by us have the main principal that the processing by the cores should be performed in parallel and bring to minimum the stay of cores.In the article are reviewed two algorithms working according to this principal "smart-delay" and the development of multiplication of matrix transposed ribbon-like algorithm.

Keywords: *Functional Programming, Parallel calculations, Algorithms, Matrix multiplication*

I. Introduction

The algorithm of sort is an algorithm for sorting elements in a list. The task of sort created a big number of solutions but there is no the universal algorithm. Having approximate characteristics of output values it can be chosen method which will work in optimal way. For reasonably making such a choice it's being reviewed parameters which will be used for evaluate algorithms [1].

Parameter:

- A time of sort - is the main parameter which characterizing the speed of execution of algorithm;
- A memory - some of the algorithms needs additional memory for temporal storage of data;
- A stability - stable sort doesn't change a mutual arrangement of data;
- A natural behavior - the electiveness of method during processing already sort data or partially sort data.

The main methods of sorting are:

1. Quadratic and sub-quadratic algorithms, "Select Sort", "BubbleSort" and its improvements, "InsertSor", "ShellSort".
2. Logarithmic and linear algorithms, "HeapSort", "QuickSort" and a "RadixSort".

ISSN 1512-1232

II. Algorithm " small delay"

In this article is being reviewed a new effective algorithm for sort, using a parallel programming "Small Delay" [2] separation of initial array is being performed by the principal of "card distribution". The first element of the array is being given to the first core the second element to second core until *n* element (n is a number of cores) on n core. An element n+1 is not only being given to first core it begins execution the sorting operation for the elements located before it. An element n+2 is being given similarly to second core and begins sorting of the elements there. The process continuing until all the elements of the array will be separated into sorted and located in sorted n sub array.

The traditional merge operation is being performed after. The main win of time is occurs because a sort of arrays doesn't start after decay of arrays on elements it starts in parallel with it.

Each core starts sort operation in parallel with "Small Delay" in compare with previous core. Each new received element and already located there sorted elements of array.

Factually is being used following algorithms in given algorithm?:

- Separation by card principal;
- Searching for the spot according to searching for phone numbers in phone book;
- Separating with merge;
- The parallel decomposition of package

The main newness of given algorithm consist of that the methods of decomposition and sort of array are executing in parallel mode and perhaps because of that the task for core begins with support in compare with previous core. The win of a time is due to delay during the decomposition of array into sub array [3].

III. The review of matrices multiplying methods

Multiplying matrix of matrix A with size of mxn and matrix B with size nxl gives matrix C with size mxl, each of elements of this matrix is being defined according to expression:

$$Cij = \sum_{k=0}^{n-1} Aik \cdot Bkj, \quad 0<=i<m, 0<=j<1$$

For showing the effectiveness of given below a new method for multiplying matrices let's review some existing traditional parallel methods [4].

A. A rough parallel algorithm

A rough parallel algorithm is presented as:

- On the first core is being calculated C_{11}
- On the second core C_{12}
- On *k* core - C_{1k} etc.
- On the last (s) core is being calculated $C_{1S.}$

Further:

- On the first core is being calculated $C_{1(s+1)}$

ISSN 1512-1232

- On the second core $C_{1(s+2)}$
- Etc.

B. A ribbon parallel algorithm

In the following parallel algorithms matrices A and B are split into continuous sequences of rows and columns (strips). A ribbon parallel algorithm has a form:

- On the first core is being calculated first column C, for this reason the first column is being set there matrix A and matrix B.
- On the second core is being calculated first column C, for this reason the second column is being set there matrix A and matrix B.
- On the k core is being calculated k column C, for this k column is being set there matrix A and matrix B.

C. Improved ribbon (Transposed) algorithm

The difference of this algorithm with previous algorithms is that by the trans pending the matrix *B* into *B^* matrix, and then

- On the firs core is being sent first row A and first row B^ and is being calculated C_{11} .
- On the second core is being sent second row A and second row B^ and is being calculated C_{21}.
- On k core is being sent row k -th A and first row B^ and is being calculated C_{k1}.
- On the last (n-th) core and is being sent n-th row A and first row B^ and is being calculated C_{n1}.
- On the first core the first row is left A and is being sent second row B^ and is calculated C_{21}.
- On the second core the second row is left A and is being sent second row B^ and is calculated C_{22}…
- On k core is left k row A and is being sent second row B^ and is calculated C_{k2}…
- ……. …… …….
- On the last (n-th) core is the last row (n-th) A is left and is being sent second row B^ and is calculated C_{n2}….

At the end let's present the evaluation of reviewed algorithms.

Given algorithms need n^3/p parallel multiplications. The volume of sent data of each processor for Fox and Cannon is $n^2 / p^{1/2}$. The volume of sending data of each processor in algorithms proposed in [5], is equal to $n^2 / p^{2/3}$. The volume of sending data of each processor in algorithms proposed in block-recursive algorithm is equal to $\frac{n^2 \log_2 p}{p}$,

It is not used "one-to-all" data feeds.

ISSN 1512-1232

IV. The proposed method of multiplying of matrices - optimization of a number of rows sent on cores

In this article we present a new method of multiplication of matrix which is an improved ribbon (transponder) algorithm. The Idea is to calculate an optimal number of rows for $B^\wedge$ matrix which we will sent to processor because it should not occur stay of cores.

We think that, instead of sending each row it should be calculated how many rows of $B^\wedge$ matrix needs to be sent

For example if the result of calculation is 3 then it means that on the first core was sent a first row of A matrix and first, second and third rows $B^\wedge$ *and will be calculated* C_{11}, C_{12} and C_{13}. On the second core is being sent a third, fourth and fifth rows of $B^\wedge$ matrix and will be calculated C_{14}, C_{15} and C_{16} etc. Until $B^\wedge$ will not be finished a number of cores.

Let's see how should be considered an optimal number of rows of $B^\wedge$ *matrix*.

Let's multiply large matrices (more than 1000 * 1000 elements) on a supercomputer (more than 128 cores).

Let's use following notations:

M - a number of cores;

L - a time of sending elements, (rows) on cores;

V - a time of execution of one operation (scalar multiplication) per core;

W- go to new row of array (very small)

K- a number of rows sending for cores.

It should be found a value of K for reason of minimizing stay of cores.

On one step the execution of scalar multiplication, (getting one result) should be approximately the same value as during sending to cores:

$$W*M \approx K*V+L. \quad (1)$$

Taking under consideration that the core shouldn't stay the return shouldn't be quick and at the same time it shouldn't be late. In case if for example if we sent to first core less rows than it is necessary then first core will stay if we will send more then it will take a lot of time for processing and will happen delay of sort.

(1) From this formula we are getting:

$K \approx (W*M-L)/ V.$

For Example

$W = 1$

$M = 512$

$V = 90$

$L= 5$

$K \approx (W*M-L)/ V= (512-5)/90 = 5$

It should happen sending of five rows of $B^\wedge$ *matrix at the same time*.

If V is not constant, then it's required approximate calculation of it.

For this reason it's possible to calculate it without coming out from value in previous step (for economy of time it's being calculated only on one core).

ISSN 1512-1232

On first series is being calculated *V = V1* and *K2* – the number of rows sending at the second series. It's being calculated consequently on second series *V2* if *V2>V*, then *V=V2* and for third series is being calculated и *K3* – the number of sending rows.

V. Conclusion:

In case of multiplying matrices *K* is a constant. It's according to in case of multiplying matrices V is a constant and it's according to it defines a time of execution of scalar multiplying operation according to this it will be calculated just once the number of sending rows of given matrices and then constantly will be sent a number of rows which is equal to K.

For different type of tasks, for example sort a value of V - is not a constant, that is why it's becomes necessary depending on V_1, V_2, V_3 ... values calculation of K_1, K_2, K_3 …- sending elements. For calculation of each V_i is being used as speed calculated for operation on previous loop. Ki (and correspondingly Vi) as exact and fast calculations as possible is the next step of development of algorithms offered by us.

It's possible that V_i – Vi_{-1} or V_i/V_{i-1} is a constant or can be calculated with some f function $V_i = f(V_{i-1})$.

All of these issues becoming very important during processing a very big data, which requires not only a good algorithm of sending data to cores but minimizing delaying of cores.

References

[1] http://algolist.manual.ru
[2] Simon Marlow. Parallel and Concurrent Programming in Haskell. 2014.
[3] R. Miller, L. Boxer. Algorithms Sequential & Parallel: A Unified Approach. M Binom 2015.
[4] N. Archvadze, M.Pkhovelishvili, L.Shetsiruli, O.Ioseliani. The Algorithm of Parallel Programming Using "Small Delay". SCCTW'2016 South-Caucasus Computing and Technology Workshop. 2-7 / 10/2016.
[5] https://indico.cern.ch/event/572800/contributions/2319259/attachments/1347569/2032637/WOR1.pdf.
[6] ВП Гергель. Теория и практика параллельных вычислений - ИНТУИТ, Бином. Лаборатория знаний, 2007.
[7] Гергель В. П. Введение в методы параллельного программирования. Образовательный комплекс. Н. Новгород: ННГУ, 2005.
[8] Гергель В. П., Стронгин Р. Г. Основы параллельных вычислений для многопроцессорных вычислительных систем. Н. Новгород: ННГУ, 2003.
[9] Gupta H., Sadayappan P. Communication Efficient Matrix-Multiplication on Hypercubes // Parallel Computing. 1996. N 22. P. 75-99.

Article received: 2017-11-24