Control of matrix computations on distributed memory

Gennadi Malaschonok and Alla Sidko

Abstract. Dedicated to research in the field of parallel computer algebra, in particular the parallelization of matrix recursive algorithms on a cluster with distributed memory. A new dynamic control scheme for matrix recursive algorithms is proposed. We considered in detail new software objects that ensure the effective operation of the dynamic control scheme.

Introduction

The first approach to creating parallel programs was a centralized dynamic LLP control scheme, in which one of the cluster nodes acted as the dispatcher for the entire computational process.

Next, the DDP scheme of the decentralized control was developed. In this scheme, each process node created its own dispatch process. However, in this scheme, there was no control over the depth of recruitment and the ability to switch to a new task until the current task was completed.

The new control scheme is called DAP-VAT-schemes. It differs in that it sequentially expands functions in depth, retaining all states at any nesting level until all calculations in the current computational subtree are completed. This allows any processor to freely switch from one subtask to another, without waiting for the completion of the current subtask account.

1. Recursive algorithm graph

1.1. Examples: matrix multiplication and triangular matrix inversion

For example, we present two simple block-recursive algorithms. Each of them contains a small number of types of recursive blocks.

The first algorithm is the calculation of the inverse matrix for a triangular non-degenerate matrix. After dividing the matrices into blocks, we get the equations

$$A = \begin{pmatrix} a & 0 \\ c & d \end{pmatrix}, \quad A^{-1} = \begin{pmatrix} x & 0 \\ z & k \end{pmatrix}$$

$$x = a^{-1}, \ k = d^{-1}, \ cx + dz = 0, \ z = -kcx.$$

The second algorithm is the algorithm of recursive block matrix multiplication $AB = C$:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} l & m \\ n & p \end{pmatrix} = \begin{pmatrix} w1 & w2 \\ w3 & w4 \end{pmatrix}$$

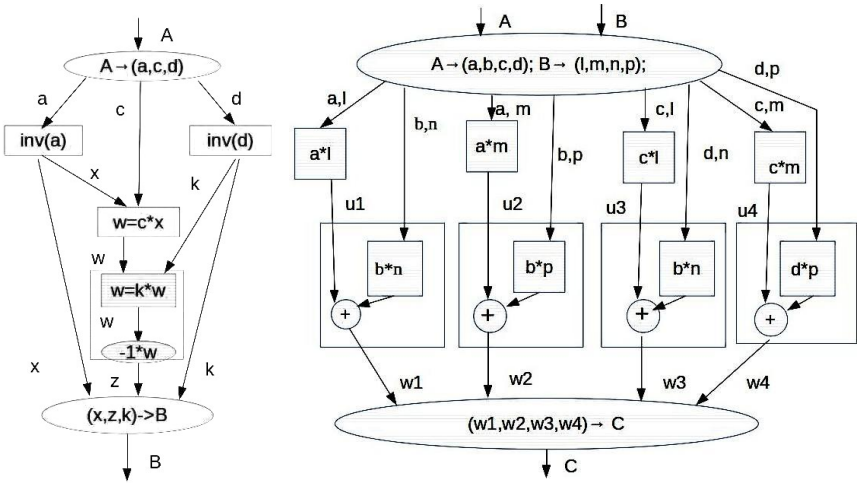

FIGURE 1. The graph of the recursive inversion algorithm of the triangular matrix (on the left) and the graph of the block-recursive matrix multiplication algorithm (on the right).

2. Computational control mechanism

Consider the components of the control mechanism of the computational process.

2.1. Drop

We divide the computational graph into separate compact subgraphs (drops). In Figure 1, the vertices, which are combined into one drop, are outlined in a square outline.

Thus, we define the drops as the smallest components of the computational graph that can be transferred to other processors.

2.2. Amine

Before the drop is calculated, we need to expand the corresponding subgraph. This subgraph is called amine. This amine also consists of drops.

For example, the amine $A \cdot B$ consists of 4 drops $A \cdot B$, 4 drops $A \cdot B + C$, one input and one output function.

2.3. Pine

All amines that are formed in one processor are stored in the general list, which is called Pine.

2.4. Vokzal

At the Vokzal are all the drop-tasks that are awaiting their direction to the calculations. These tasks are located at different levels. These levels correspond to the depth of recursion for drops.

2.5. Aerodrome

Each processor that sent a drop task is called a parent. The list of all parent processors is called an Aerodrome.

2.6. Terminal

The terminal is used to communicate with the child processors that were sent drop-tasks. All child processors are stored in the terminal.

3. Primary fields and functions

3.1. The main fields of the drop object

— PAD (np, na, nd)– address of this drop.
— Type — drop type (unique number in the list of all drop types).
— InData outData — these are vectors for input and output.
— Amine — the amine of this drop.
— RecNum — recursion number of the drop.
— Arcs — amine graph topology.

3.2. The main fields of the amine object

— PAD (np, na, nd) — address to return the result of the calculation of this drop.
— Type, inData, outData — the same as the drop.
— Drop — an array of all drops of a given amine.

4. Organization of computational threads

We use two threads: a computational thread and a dispatcher thread. These threads will run on each cluster processor.

4.1. CalcThread

The CalcThread waits for the arrival of the first drop task at the vokzal and starts the corresponding calculations.

4.1.1. CalcThread objects:

— Pine — list of amines on this processor.
— Vokzal — an array of lists of available drop tasks.
— Aerodrome — list of parent processors.
— Terminal — an array of child processor lists.
— CurrentDrop — current drop, which is calculated.

4.1.2. CalcThread functions:

— WriteResultsToAmin — the results of a drop calculation are written to its amine in the input data vectors of other drops.
— InputDataToAmin — create an amine from a drop, if a new task arrives, we make an input function.
— WriteResultsAfterInpFunc — write the result of the input function to all the amine drops.
— runCalcThread() — If a drop-result has come, we register it in another drop by topology (writeResultsToAmin). If additional components arrived, we make the input function and write to the amine its result. (inputDataToAmin & writeResultsAfterInpFunc) If a new drop arrives with a task, we look at the size of the input data. If the task is a leaf, it make a sequential calculation and it write the result to other drops (writeResultsToAmin). Otherwise, expand the amine (inputDataToAmin). If Vokzal is empty, then the isNotEmptyVokzal flag is set and the counting thread goes to wait another drop.

4.2. Dispatching Thread

The work of the dispatching thread can be divided into 10 processes:
— Waiting for completion signal.
— Reception task.
— Receive free processors.
— Receive and record the status of the child processor.
— Receive the result of the calculated drop and record these results in the corresponding amine.
— Receive non-main components and record it in the right place.
— Sending available tasks to free processors (if there are tasks and processors).
— Sending free processors to a child (if there are no drop tasks available, but there are free and child processors).
— Sending the entire list of free processors to the parent processor (if the Vokzal is empty and the Terminal does not contain child processors with positive levels).
— Sending drop results to parent processors.
— Sending additional components to child processors.

Conclusion

We gave a description of the universal dynamic paralleling scheme for recursive algorithms on the distributed memory cluster, described the main objects, their fields and functions, and also explained the operation of the two-thread system that runs on each cluster core. We have described six new objects that provide such a control mechanism and give the name of this scheme: drop, amine, pine, vokzal, aerodrome, terminal. This scheme can be applied to any matrix recursive algorithms, both with dense and sparse matrices. The scheme was implemented in the Java programming language using the OpenMPI and MathPartner [6] packages, and its work was tested on the above matrix multiplication and inversion algorithms. We plan to conduct a detailed experimental study of the effectiveness of this scheme on other recursive matrix algorithms.

References

[1] http://parca2010.parallel-computer-algebra.org/articles.pdf

[2] Abramov S., Adamovich A., Nyukhin A., Moskovsky A., Roganov V., Shevchuk E., Shevchuk Yu., Vodomerov A. OpenTS: An Outline of Dynamic Parallelization Approach.*PaCT 2005. Parallel Computing Technologies*, LNCS **3606**, Springer, eidenberg, 303-312, 2005.

[3] Malashonok G.I., Valeev Y.D. The control of parallel calculations in recursive symbolic-numerical algorithms. Proceedings of conference of PaVt'2008 (St.-Petersburg). Chelyabinsk: Publishing house JuUrGu, 2008. P. 153-165.

[4] Malashonok G.I. Control of parallel computing process. Tambov University Reports. Natural and Technical Sciences. V. 14, part. 1, 2009. P. 269-274.

[5] Ilchenko E.A. On the effective method of parallelization of block recursive algorithms. Tambov University Reports. Natural and Technical Sciences. V. 20, part. 5, 1173-1186, 2015.

[6] Malaschonok G.I., MathPartner Computer Algebra, *Programming and Computer Software*, **43**, No. 2, 112–118, 2017.

Gennadi Malaschonok
National University of Kyiv-Mohyla Academy
Kiev, Ukraine
e-mail: malaschonok@gmail.com

Alla Sidko
National University of Kyiv-Mohyla Academy
Kiev, Ukraine
e-mail: allochka.sidko@gmail.com