УДК 004.27

# Вопросы применения и реализации потоковой модели вычислений

А.В. Климов, Н.Н. Левченко, А.С. Окунев, А.Л. Стемпковский

Институт проблем проектирования в микроэлектронике РАН, klimov@ippm.ru, nick@ippm.ru, oku@ippm.ru, ippm@ippm.ru

***Аннотация*** **– В статье приведены аргументы, демонстрирующие целесообразность развития и продвижения оригинальных архитектурных принципов построения суперкомпьютеров, основанных на управлении потоком данных. Описывается потоковая модель вычислений с динамически формируемым контекстом, рассмотрены отличия предложенной модели вычислений с точки зрения изменения парадигмы вычислений и программирования (с парадигмы «сбора» на парадигму «раздачи»). Описано сопоставление с другими подходами, приводится возможная аппаратная реализация модели вычислений с управлением потоком данных, работающая в парадигме «раздачи», а также обосновывается перспективность предлагаемой модели вычислений.**

***Ключевые слова*** **– парадигма "раздачи", новая парадигма программирования, параллельное программирование, параллельная потоковая вычислительная система.**

## I. Введение

Модель вычислений с управлением потоком данных (dataflow) по сравнению с фон-неймановской моделью изначально задумывалась как более адекватная аппаратуре, в частности, разработчики утверждали, что она значительно лучше использует присущий аппаратуре «естественный» параллелизм [1, 2]. Однако в дальнейшем основные усилия разработчиков были направлены на приведение модели вычислений dataflow в соответствие сложившейся на тот момент методологии программирования: разработке адекватных ей языков программирования высокого уровня с привычной нотацией [3, 4], подгонке аппаратной реализации как под эти новые, так и старые языки [5, 6]. В результате изначальный потенциал и конкурентные преимущества были утрачены, что в конце концов (в совокупности с имеющимися на тот момент трудностями с аппаратной реализацией) и привело к «поражению» в «соревновании» с традиционной моделью программирования на основе адресуемой памяти и к фактическому сворачиванию всех работ по данному направлению за рубежом [7, 8].

В России работы над моделью вычислений с управлением потоком данных велись под руководством академика Бурцева В.С. в ИВВС РАН и ИПИ РАН, а с 2007 года – продолжены в ИППМ РАН. Авторы видят в ней огромный потенциал, особенно это касается создания новых суперкомпьютерных вычислительных систем петафлопсного и сверхпетафлопсного уровня производительности.

Тем не менее, можно утверждать, что в традиционных dataflow системах было утрачено и определенное принципиальное качество, сближавшее потоковую модель с аппаратурой, и, что особенно важно, с аппаратурой распределенной и параллельной.

В данной статье говорится об этом качестве как о парадигме «раздачи» и показывается, что оно в полной мере сохранено и поддержано в потоковой модели вычислений с динамически формируемым контекстом [9, 10]. Проблема, однако, в том, что это качество действительно плохо совместимо с привычной парадигматикой программирования, поэтому и затрудняется продвижение данной модели для ее широкого использования. Но есть основания и для оптимизма, одно из который состоит в наблюдении, что практически все современные средства параллельного программирования явно или неявно это качество приобретают, вынуждая таким образом программистское сообщество к нему привыкать.

## II. Парадигмы сбора и раздачи

Практически все современные языки программирования, начиная с Фортрана, в своей основе имеют понятия переменной и выражения. Если нам надо умножить два числа и прибавить к произведению третье, мы пишем следующее выражение: a*x+b.

Когда эти команды будут выполняться, то непосредственно перед выполнением умножения будут сделаны запросы на считывание значений a и x, а затем (перед сложением) значения b. Это показывает, что как в исходном коде (a*x+b), так и при его выполнении мы работаем в парадигме «сбора»: кому данные нужны, тот их и запрашивает.

Вообще, в ходе выполнения программы каждый элемент данных сначала где-то вычисляется, потом где-то используется, а в промежутке он еще где-то и хранится. Классический постулат распределенного программирования звучит так: «храним там, где вычисляем», или, в другой формулировке, «вычисляем там, где будем хранить». Но тогда в момент использования неизбежен запрос к месту хранения с долгим ожиданием обратного ответа – это и есть

парадигма «сбора». Но возможно обратное решение: хранить данные там, где ожидается их использование. То есть по вычислению элемента данных сразу будет инициироваться его передача – туда, где он будет использоваться. Этим и отличается парадигма «раздачи» (рис. 1, б) от парадигмы «сбора» (рис. 1, а).


**Рис. 1. Сравнение парадигм «сбора» (а) и «раздачи» (б)**

Таким образом, в парадигме «сбора» происходит сбор необходимых для вычисления данных, который инициируется там и в тот момент, где и когда эти данные будут нужны. При выполнении такой процесс влечет технические приостановки на ожидание прихода требуемых данных. На однопроцессорных системах с относительно небольшой памятью эти задержки были практически незаметны, но для многопроцессорных распределенных систем с большим объемом памяти они становятся уже критичными. Для нивелирования таких задержек появились различные механизмы, такие как: кэш, спекулятивное выполнение вычислений, архитектура «EPIC» (Explicitly Parallel Instruction Computing – микропроцессорная архитектура с явным параллелизмом команд), суперскаляр, гипертрединг (hyper-threading) и другие. И все эти механизмы приводят к увеличению объема аппаратуры – только для того, чтобы обеспечить эффективную поддержку удобного программирования в парадигме «сбора»! Таким образом, назрела необходимость смены парадигмы программирования.

## III. Потоковая модель вычислений

Итак, в потоковой модели вычислений, работающей в парадигме «раздачи», реализуется следующий принцип: всё вычисление должно быть разбито на элементарные фрагменты, каждый из которых запускается лишь тогда, когда в наличии имеются все данные, необходимые для его завершения (рис. 2). Иногда их называют нанотредами (nanothreads) [11], поскольку, как правило, фрагменты с таким свойством малы, мы же называем такие объекты программными узлами.

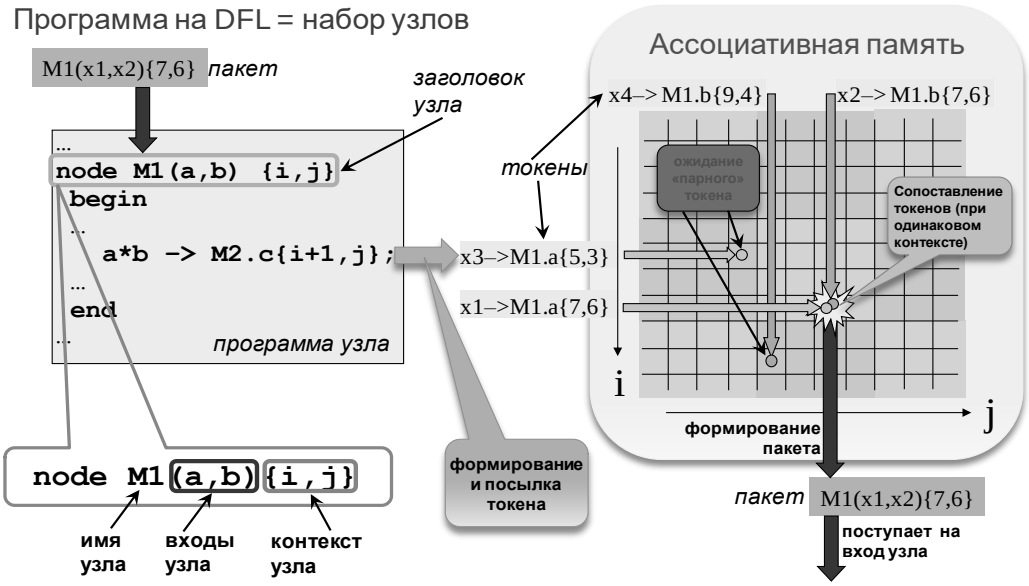


**Рис. 2. Потоковая модель вычислений с динамически формируемым контекстом**

Программный узел создается и активируется, когда присутствуют все данные, необходимые для выполнения программы этого узла без подкачки дополнительных данных. Это принцип модели вычислений с управлением потоком данных: вычисления активируются данными. Каждый программный узел в результате своего выполнения формирует данные, которые предназначены для использования в других программных узлах. Значит, по завершении или в процессе своей работы программный узел должен сам посылать свои результаты в «места» их дальнейшего использования.

Чтобы эти послания доходили куда следует, они должны быть снабжены некоторыми адресами. Эти адреса должны быть уникальны для инициируемого этими сообщениями программного узла, по меньшей мере в пределах некоторого периода времени. Иначе говоря, адресоваться должны не данные, а вычисления, точнее «места», где производится накопление данных и инициация вычисления, когда все необходимые для этого вычисления данные имеются в наличии.

Важно отметить, что в нашей же программе явно задаются правила вычисления всех полей контекста передаваемого токена. Это отличает описываемую модель вычислений от существовавших ранее потоковых моделей, поэтому она и называется потоковой моделью вычислений с динамически формируемым контекстом.

Для хорошей масштабируемости программы необходим контроль распределения вычислений и данных по процессорным элементам. Выбор распределения может значительно влиять на объемы совершаемых обменов на разных уровнях иерархии системы, а тем самым и на общую производительность. Желательно, чтобы смена распределения не приводила к необходимости перепрограммирования.

В парадигме «раздачи» (в параллельном языке программирования) программист указывает способ распределения вычислительных узлов. Для этого им задается функция распределения, которая по значению виртуального адреса выдает номер процессорного элемента, в котором и будет происходить накопление данных для узла с данным адресом и с приходом всех необходимых данных – выполняется программа узла. В программе функции распределения могут задаваться либо при заголовке узла для каждого узла отдельно:

**node** C(x:**real**) F1{t,i} **distribution** (i **div** 256);

либо при описании типа контекста сразу для всех узлов с контекстом данного типа:

**type** F1 = { time:**int**, index:**int** } **distribution** (index **div** 256).

В обоих случаях написано распределение, при котором узел C{t,i} «приписан» к элементу с номером ((i **div** 256) **mod** K), где K – число процессорных элементов. В данном случае мы имеем блочно-циклическое распределение блоками по 256 элементов, где элементами являются виртуальные узлы. Эта функция создает длинные цепочки межпроцессорных зависимостей. Мы можем легко заменить функцию распределения другой: ((i+t) **div** 256), которая дает в 128 раз меньшую длину цепочек. Заметим, что в известных реализациях вычислительных систем, работающих в парадигме «сбора», аналогичная возможность не поддерживается, и реализация аналогичной схемы потребует перепрограммирования задачи.

С точки зрения межпроцессорных коммуникаций и распределения вычислений потоковая модель, работающая в парадигме «раздачи», имеет следующие преимущества:

1) указания по распределению и их модификация никак не затрагивают сути алгоритма;

2) в парадигме «раздачи» пересылки идут в одну сторону и сразу «по назначению».

Основной недостаток парадигмы «раздачи» – это «непривычность» с точки зрения написания программ. Существуют три пути устранения этого недостатка. Первый – обеспечение «комфортных» условий для программирования на базовом параллельном потоковом языке DFL. Второй – автоматическая трансляция с существующих языков (Фортран, C) в DFL. Третий путь – создание нового языка в парадигме «сбора» – метаязыка, работа над которым ведется авторами в настоящее время. Метаязык должен легко транслироваться в парадигму «раздачи» (параллельный язык DFL), а также иметь возможность реализации и на других программно-аппаратных платформах.

## IV. Сопоставление с другими подходами

MPI. Это самый распространенный на сегодняшний день подход [12]. Его в свете нашей классификации следует охарактеризовать как смешанный: в малом – (внутри одного MPI-процесса) это обычное программирование в парадигме «сбора», в большом (на уровне совокупности процессов) – в парадигме «раздачи». Это приводит к тому, что кодирование в некотором смысле выполняется дважды.

CHARM++ [13]. Этот язык в настоящее время набирает популярность. В своей основе он – объектно-ориентированный. В нем вводятся объекты специального вида – share, которые «населяют» виртуальный мир, обмениваясь односторонними сообщениями. В отличие от узлов DFL объекты-share «живут» перманентно, имея состояние, которое изменяется обработчиками сообщений (методами). Параллелизм имеет место среди многих share, но внутри одного share все происходит строго последовательно.

Polyphonic C# [14]. Это тоже объектно-ориентированный язык. В нем имеются особые объекты с асинхронными методами. Как и в Charm++ вызовы этих методов являются односторонними, асинхронными (то есть сразу после вызова, не дожидаясь завершения, вызвавшая программа продолжает работать). Но есть одно дополнение: методы могут иметь по нескольку заголовков, и для активации метода должны быть из одного или разных мест (тредов) сделаны вызовы на все заголовки.

Как Charm++, так и Polyphonic C# являются языками с программированием в парадигме раздачи. Их особенностью, в отличие от нашего подхода, является следование объектной парадигме.

## V. Аппаратная реализация вычислительной системы

Одним из ключевых требований, предъявляемых к аппаратуре, реализующей модель вычислений с управлением потоком данных, является аппаратная поддержка процедуры сопоставления токенов. В ИППМ РАН разрабатывается архитектура параллельной потоковой вычислительной системы (ППВС), которая как раз и реализует эту модель вычислений.

За сопоставление токенов в этой системе отвечает процессор сопоставления, элементом которого является ассоциативная память ключей.

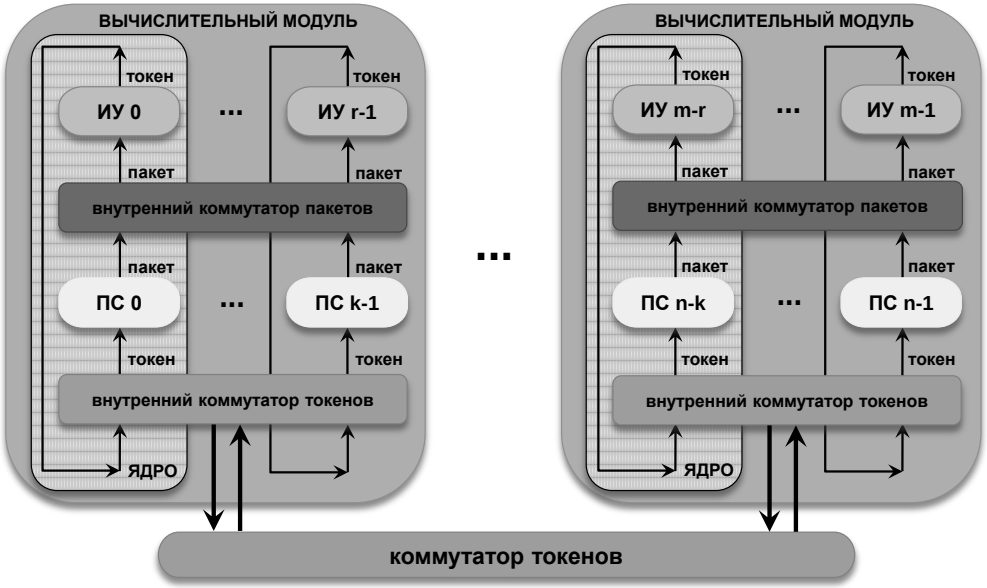


**Рис. 3. Архитектура параллельной потоковой вычислительной системы**

В целом ППВС представляет собой высокопроизводительную масштабируемую вычислительную систему, в которой вычислительные модули системы объединены коммуникационной сетью (рис. 3). По коммуникационной сети вычислительной системы циркулируют токены, являющиеся структурой, в состав которой входят передаваемое данное (операнд), ключ, код операции, а также набор служебных признаков и атрибутов. В поле ключа содержится информация, позволяющая однозначно идентифицировать токен в виртуальном адресном пространстве задачи.

Пакет представляет собой структуру, состоящую из служебных признаков, данных и ключа, полностью готовую к обработке. Во внешней коммуникационной среде циркулируют только токены.

Вычислительный модуль состоит из вычислительных ядер, которые объединены между собой посредством коммутатора пакетов и коммутатора токенов. Вычислительное ядро состоит из процессора сопоставления (ПС), исполнительного устройства (ИУ) и блока хэширования. Исполнительные устройства в вычислительном модуле обезличены, каждое из них хранит все программы вычислительных узлов задачи, то есть пакет может обрабатываться в пределах модуля на любом свободном ИУ.

Коммутация между вычислительными ядрами осуществляется по номеру вычислительного ядра, который формируется блоком хэширования с использованием настраиваемой функции распределения [15]. Токен после формирования номера вычислительного ядра либо направляется к соответствующему ядру внутри вычислительного модуля, либо передается во внешнюю сеть в соответствующий вычислительный модуль. Подробно принципы работы вычислительной системы описаны в работах [16].

## VI. Заключение

Предлагаемая модель вычислений отличается от классической тем, что изменена парадигма вычислений и программирования, что позволяет ей стать намного ближе к параллельной аппаратуре, и, соответственно, более естественным образом создавать параллельные программы, уменьшая семантический разрыв между параллельным языком высокого уровня и предлагаемой архитектурой в сравнении с современными вычислительными системами.

Основными особенностями модели вычислений с управлением потоком данных и архитектуры, её реализующей, являются:

1) масштабирование без перепрограммирования текста задачи;

2) аппаратная синхронизации по данным;

3) отсутствие проблемы когерентности кэш;

4) отсутствие адресной арифметики и в абстрактном смысле отсутствие памяти как таковой;

5) возможность выявления неявного параллелизма в задачах;

6) возможность гибкого регулирования параллелизма и др.

На программных моделях ППВС было проверено выполнение некоторых задач, являющихся «тяжелыми» для традиционных суперкомпьютерных систем.

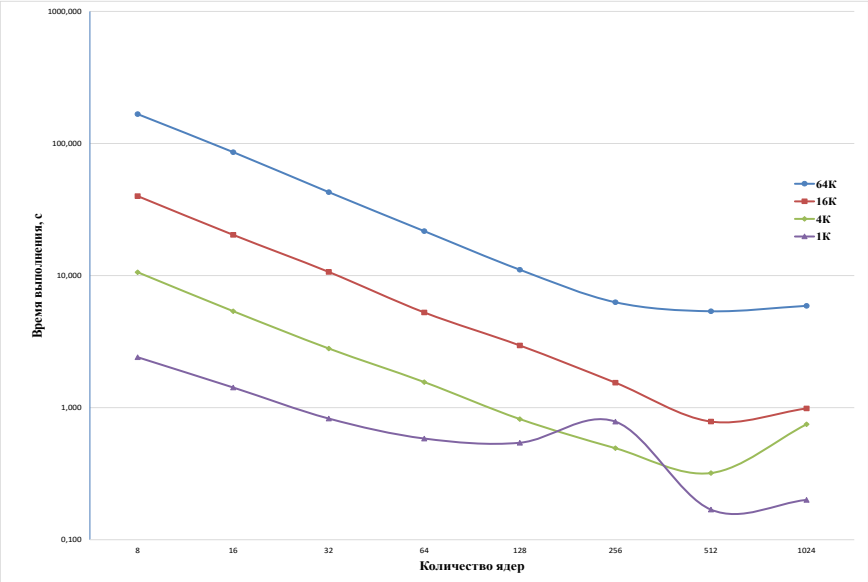


**Рис. 4. Зависимость времени прохождения задачи от количества ядер на суперкомпьютере «Ломоносов» для задачи МД (логарифмическая шкала)**

На рис. 4 приведены графики зависимости времени прохождения задачи от количества ядер на суперкомпьютере «Ломоносов» [17] для задачи «молекулярная динамика» (МД). Отчетливо видна тенденция сохранения степени масштабирования при увеличении числа ядер.

Эксперименты проводились на различных конфигурациях системы – от 8 до 1024 вычислительных ядер. На первых трех графиках видно, что с ростом количества ядер происходит практически линейное уменьшение времени выполнения задачи на эмуляторе. Это означает, что размерность самой задачи и параллелизм, заложенный в реализованном алгоритме, обеспечивают эффективное использование имеющихся вычислительных ресурсов. Время выполнения задачи начинает увеличиваться на большом количестве ядер из-за недозагруженности работой исполнительных устройств (недостаточная размерность задачи). Оптимальное количество обрабатываемых частиц на одно ядро, при котором масштабирование не падает, составляет для ППВС значение порядка $10^3$ частиц.

Проведенные эксперименты [18] показали, что предлагаемая модель вычислений с управлением потоком данных с ее уникальными особенностями может стать в перспективе основной реализуемой парадигмой программирования в области крупномасштабных параллельных вычислений, а разрабатываемая аппаратура ППВС, реализующая эту модель вычислений, позволит создавать отечественные высокопроизводительные вычислительные системы общего или специального назначения.

## Литература


[1] Lee Ben, Hurson A.R. Issues in Dataflow Computing //Advances in computers. 1993. Vol.37. P. 285-333.

[2] Ben Lee, Hurson A.R. Dataflow Architectures and Multithreading//Computer. 1994. Aug. V. 27, no. 8. P. 27-39.

[3] Gurd, J. R., Kirkham C. C. and Watson, I., "The Manchester Prototype DataFlow Computer," Commun. ACM, Vol. 28, pp. 34-52, Jan. 1985.

[4] G.Popadopoulos, D.Culler. Monsoon: an Explicit Token-Store Architecture, Proceedings of the 17th Annula International Symposium on Computer Architecture, pp. 82-91.

[5] B. Lee , A. R. Hurson , B. Shirazi, A Hybrid Scheme for Processing Data Structures in a Dataflow Environment, IEEE Transactions on Parallel and Distributed Systems, v.3 n.1, p.83-96, January 1992.

[6] Hurson, A. R., Lee, B., and Shirazi, B., "Hybrid Structure: A Scheme for handling Data Structures in a Dataflow Environment," Proc. Parallel Architectures and Languages, Lecture Notes in Computer Science, Vol. 365, 1989, pp. 323-340.

[7] Arvind, D.E. Culler, and K. Ekanadham, “The Price of Fine-Grain Asynchronous Parallelism: An Analysis of Dataflow Methods,” Proc. CONPAR 88, Sept. 1988, pp. 541-555.

[8] Arvind, D. E. Culler, and G. K. Maa. Assessing the Benefits of Fine-Grain Parallelism in Dataflow Programs. The International Journal of High Performance Computing Applications, 2(3), December 1988, pp. 60-69.

[9] Стемпковский А. Л., Левченко Н. Н., Окунев А. С., Цветков В. В. Параллельная потоковая вычислительная система – дальнейшее развитие архитектуры и структурной организации вычислительной системы с автоматическим распределением ресурсов // Журнал «ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ», 2008, №10, С. 2-7.
[10] Климов А. В., Левченко Н. Н., Окунев А. С. Преимущества потоковой модели вычислений в условиях неоднородных сетей // Журнал «Информационные технологии и вычислительные системы», 2012, № 2, С. 36-45.
[11] C. Polychronopoulos, N. Bitar, and S. Kleiman. NanoThreads: A User-Level Threads Architecture. Technical Report No. 1297, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, 1993.
[12] URL: http://www.mpi-forum.org/docs/docs.html (дата обращения: 30.03.2016).
[13] URL: http://charm.cs.uiuc.edu/ (дата обращения: 30.03.2016).
[14] N. Benton, L. Cardelli, C. Fournet, ”Modern concurrency abstractionsfor C#,” ACM Transactions on Programming Languages and Systems (TOPLAS), 25(5):769-804, 2004.
[15] Стемпковский А.Л., Климов А.В., Левченко Н.Н., Окунев А.С. Методы адаптации параллельной потоковой вычислительной системы под задачи отдельных классов // журнал «Информационные технологии и вычислительные системы», 2009 г. № 3, С. 12-21.
[16] Климов А. В., Левченко Н. Н., Окунев А. С., Стемпковский А. Л. Суперкомпьютеры, иерархия памяти и потоковая модель вычислений // Программные системы: теория и приложения: электрон. научн. журн. 2014, Т. 5, № 1(19), С. 15-36, URL: http://psta.psiras.ru/read/psta2014_1_15-36.pdf (дата обращения: 30.03.2016).
[17] Воеводин Вл.В., Жуматий С.А., Соболев С.И., Антонов А.С., Брызгалов П.А., Никитенко Д.А., Стефанов К.С., Воеводин Вад.В. Практика суперкомпьютера "Ломоносов" // Открытые системы, Москва: Издательский дом "Открытые системы", 2012, № 7, с. 36-39.
[18] Д.Н. Змеев, А.В. Климов, Н.Н. Левченко, А.С. Окунев, А.Л. Стемпковский. Эмуляция аппаратно-программных средств параллельной потоковой вычислительной системы «Буран» // Информационные технологии. 2015. Т. 21. №10. С. 757-762.

# The application and implementation issues of dataflow computing system

A.V. Klimov, N.N. Levchenko, A.S. Okunev, A.L. Stempkovsky

Institute for Design Problems in Microelectronics of RAS, klimov@ippm.ru, nick@ippm.ru, oku@ippm.ru, ippm@ippm.ru

***Keywords* — paradigm of "scattering", new paradigm of programming, parallel programming, parallel dataflow computing system.**

ABSTRACT

Computing model with dataflow management was originally conceived as a more adequate for hardware in comparison with von-Neumann model. It uses a proper "natural" parallelism of the hardware much better.

A classic postulate of distributed programming is: "to store there where is computing" or, in another formulation, "to compute there where to be stored". But then, at the time of use it is inevitable to have a request to data storage with a long feedback waiting time - it is the paradigm of "gathering". But the opposite solution is possible: to store data there where they are expected to be used. That is, the computing of data item will immediately initiates its transfer to the place where it will be used. This distinguishes the paradigm of "scattering" from the paradigm of "gathering".

In terms of inter-processor communication and computation distribution, the dataflow model, which works in the paradigm of "scattering", has the following advantages:

1) Instructions for distribution and modification do not affect the essence of the algorithm.

In the paradigm of "scattering" the data transfer in one direction and at once "on destination".

To date, the most common approaches to the creation of parallel programs are MPI, Charm++, Polyphonic C#.

The MPI, in the light of our classification, should be described as mixed: inside one MPI-process it is a common programming in the paradigm of "gathering", on the level of the group of processes - in the paradigm of "scattering".

Charm++ and Polyphonic C# are the languages with programming in the paradigm of "scattering", however, their peculiarity, in contrast to our approach, is to follow the object paradigm.

The IPPM RAS develops the architecture of the parallel dataflow computing system (PDCS), which implements the dataflow computing model with dynamically formed context. The matching processor, which element is an associative memory of keys, responds for matching of tokens in the system.

The proposed computing model is distinguished from the classic one the fact that the paradigms of computation and programming were changed. This allows it to be much closer to the parallel hardware, to provide a more natural

way for parallel programs creation, to reduce the semantic gap between the high-level parallel language and the proposed architecture in comparison with modern computer systems.

## REFERENCES


[1] Lee Ben, Hurson A.R. Issues in Dataflow Computing //Advances in computers. 1993. Vol.37. P. 285-333.

[2] Ben Lee, Hurson A.R. Dataflow Architectures and Multithreading//Computer. 1994. Aug. V. 27, no. 8. P. 27-39.

[3] Gurd, J. R., Kirkham C. C. and Watson, I., "The Manchester Prototype DataFlow Computer," Commun. ACM, Vol. 28, pp. 34-52, Jan. 1985.

[4] G.Popadopoulos, D.Culler. Monsoon: an Explicit Token-Store Architecture, Proceedings of the 17th Annula International Symposium on Computer Architecture, pp. 82-91.

[5] B. Lee , A. R. Hurson , B. Shirazi, A Hybrid Scheme for Processing Data Structures in a Dataflow Environment, IEEE Transactions on Parallel and Distributed Systems, v.3 n.1, p.83-96, January 1992.

[6] Hurson, A. R., Lee, B., and Shirazi, B., "Hybrid Structure: A Scheme for handling Data Structures in a Dataflow Environment," Proc. Parallel Architectures and Languages, Lecture Notes in Computer Science, Vol. 365, 1989, pp. 323-340.

[7] Arvind, D.E. Culler, and K. Ekanadham, “The Price of Fine-Grain Asynchronous Parallelism: An Analysis of Dataflow Methods,” Proc. CONPAR 88, Sept. 1988, pp. 541-555.

[8] Arvind, D. E. Culler, and G. K. Maa. Assessing the Benefits of Fine-Grain Parallelism in Dataflow Programs. The International Journal of High Performance Computing Applications, 2(3), December 1988, pp. 60-69.

[9] Stempkovskij A.L., Levchenko N.N., Okunev A.S., Cvetkov V.V. Parallel Dataflow Computing System: Further Development of Architecture and Structural Organization of the Computing System with Automatic Distribution of Resources. Zhurnal «INFORMACIONNYE TEHNOLOGII». 2008, № 10, pp. 2-7 (In Russian).

[10] Klimov A.V., Levchenko N.N., Okunev A.S. The Advantages of Dataflow Calculation Model in Heterogeneous Networks Conditions. Zhurnal «Informacionnye tehnologii i vychislitel'nye sistemy», 2012, № 2, pp. 36-45 (in Russian).

[11] C. Polychronopoulos, N. Bitar, and S. Kleiman. NanoThreads: A User-Level Threads Architecture. Technical Report No. 1297, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, 1993.

[12] Available at: http://www.mpi-forum.org/docs/docs.html (accessed 30.03.2016).

[13] Available at: http://charm.cs.uiuc.edu/ (accessed 30.03.2016).

[14] N. Benton, L. Cardelli, C. Fournet, ”Modern concurrency abstractionsfor C#,” ACM Transactions on Programming Languages and Systems (TOPLAS), 25(5):769-804, 2004.

[15] Stempkovskij A.L., Klimov A.V., Levchenko N.N., Okunev A.S. Methods of Parallel Dataflow Computing System Adaptation for Problems of Individual Classes. zhurnal «Informacionnye tehnologii i vychislitel'nye sistemy», 2009. №3, pp. 12-21 (In Russian).

[16] Klimov A.V., Levchenko N.N., Okunev A.S., Stempkovskij A.L. Supercomputers, memory hierarchy and the dataflow computing system. Programmnye sistemy: teoriya i prilozheniya: elektron. nauchn. zhurn. 2014, vol. 5, № 1(19), pp. 15-36 (in Russian). Available at: http://psta.psiras.ru/read/psta2014_1_15-36.pdf (accessed 30.03.2016).

[17] Voevodin Vl.V., Zhumatij S.A., Sobolev S.I., Antonov A.S., Bryzgalov P.A., Nikitenko D.A., Stefanov K.S., Voevodin Vad.V. The practice of supercomputer "Lomonosov". Otkrytye sistemy, Moscow, 2012, no. 7, pp. 36-39 (in Russian).

[18] Zmeev D.N., Klimov A.V., Levchenko N.N., Okunev A.S., Stempkovskij A.L. Emulation on Hardware and Software of the Parallel Dataflow Computing System "Buran". «Informacionnye tehnologii», 2015, vol. 21, №10, pp. 757-762 (In Russian).