НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ноябрь–декабрь 2018 Том 18 № 6 ISSN 2226-1494 http://ntv.ifmo.ru/
SCIENTIFIC AND TECHNICAL JOURNAL OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS
November–December 2018 Vol. 18 No 6 ISSN 2226-1494 http://ntv.ifmo.ru/en
УНИВЕРСИТЕТ ИТМО

УДК 004/27

ОТКРЫТЫЕ *DATAFLOW*-СИСТЕМЫ С СЕТЕВОЙ СТРУКТУРОЙ

Р.А. Ланцов[a]

[a] ООО «Стримлайн ДС», Москва, 125212, Российская Федерация
Автор для переписки: ruslantsov@yandex.ru

Информация о статье
Поступила в редакцию 28.08.18, принята к печати 26.09.18
doi: 10.17586/2226-1494-2018-18-6-1023-1033
Язык статьи – русский

Ссылка для цитирования: Ланцов Р.А. Открытые *dataflow*-системы с сетевой структурой // Научно-технический вестник информационных технологий, механики и оптики. 2018. Т. 18. № 6. С. 1023–1033. doi: 10.17586/2226-1494-2018-18-6-1023-1033

Аннотация

Рассмотрены открытые вычислительные системы, позволяющие по мере необходимости путем механического добавления новых конструктивных единиц наращивать производительность и память, не затрагивая существующей программной среды. Такие системы основаны на применении специальной функционально полной элементной базы (планировщик, функтор, коммуникатор и др.), реализующей параллельную обработку с использованием управляющих потоков данных (*dataflow*), когда вместе с данными переносятся и необходимые фрагменты программы. Для этого при обнаружении готовности к запуску определенной процедуры (в планировщике есть все необходимые для нее данные) в планировщике раскрывается соответствующий фрагмент программы – оператор, который затем передается вместе с данными в свободное исполнительное устройство – функтор. Результат всегда возвращается по тому же маршруту, по которому происходила активация процедуры. Рассмотрены варианты компоновки открытых систем с использованием двух конструктивных единиц – ячеек на приведенной элементной базе, и серверов, собранных из этих ячеек. Использована двухуровневая распределенная коммутационная среда. На уровне ячеек она обеспечивается транзитными свойствами планировщиков и функторов, а на уровне серверов – коммуникаторами, входящими в состав ячеек. Выделены три типа ячеек, позволяющих наращивать функции вычислительных систем: ячейки для увеличения числа шлюзов, используемых для обмена с внешней средой; ячейки для расширения управляющей и оперативной памяти; ячейки для повышения производительности. Отказ от сосредоточенной коммутационной среды позволил наращивать вычислительные системы независимо и без ограничения на их размеры. Описана трехмерная структура открытой системы, которая может быть использована для построения суперкомпьютеров.

Ключевые слова

открытые системы, *dataflow*-архитектура, операторный уровень управления, планировщик, функтор, коммуникатор

OPEN *DATAFLOW*-SYSTEMS WITH NETWORK STRUCTURE

R.A. Lantsov[a]

[a]Streamline DS, Moscow, 125212, Russian Federation
Corresponding author: ruslantsov@yandex.ru

Article info
Received 28.08.18, accepted 26.09.18
doi: 10.17586/2226-1494-2018-18-6-1023-1033
Article in Russian

For citation: Lantsov R.A. Open *dataflow*-systems with network structure. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2018, vol. 18, no. 6, pp. 1023–1033 (in Russian). doi: 10.17586/2226-1494-2018-18-6-1023-1033.

Abstract

The paper considers open computing systems, which provide the necessary growth of performance and memory by mechanical addition of new units without affecting the existing software environment. Such computing systems are based on the application of a special functionally complete element base (planner, functor, communicator) that implements parallel processing using dataflow control flows when necessary program fragments are transferred along with the data. Aimed at this, when a certain procedure is found ready for starting (the planner is ready for all the data it needs), the corresponding part of the program is opened in the planner – the operator, which is then transferred along with the data to the free execution device – the functor. The result is always returned along the same route by which the procedure was activated. The layouts of the open systems using two units of design are considered: cells on the reduced element base, and servers assembled from these cells. A two-level distributed switching environment is used. At the cell level, it is provided by the transit properties of the planners and functors, and at the server level – by communicators that are part of the cells. Three types of cells are

identified, which enable the growth of computing system functions: cells for increasing the number of gateways used to exchange with the external environment, cells for increasing control and working memory, cells for increasing performance. The recession of the concentrated switching environment allowed for these types of computing systems extensions to be performed independently and without any restrictions on their size. A three-dimensional structure of open systems is described, which can be used to build supercomputers.

Keywords
open systems, *dataflow*-architecture, operator control level, planner, functor, communicator

Введение

Развитие технологий при создании элементной базы, используемой в многопроцессорных вычислительных системах (ВС), приводит наряду с повышением производительности и к существенному росту стоимости таких систем. В результате при ужесточении требований к производительности существующей ВС актуальной становится задача достичь этой цели путем расширения текущих ресурсов – памяти и производительности. Целесообразно решать такую задачу, используя конструктивные единицы – *ячейки* и *серверы*, электрическое подключение которых к системе должно сопровождаться автоматическим встраиванием новых ресурсов в существующую среду программного управления без какой-либо специальной подстройки. ВС, которые могут быть расширены указанным способом, называют *открытыми*. Такие ВС не имеют логических и конструктивных ограничений в форме некоего «каркаса». В существующих ВС таким «каркасом», в частности, являются ограниченная разрядность адресов и коммутационная среда, обеспечивающая доступ к отдельным компонентам решающего поля и рассчитываемая на определенный размер этого поля.

Реализация идеи «бескаркасных» параллельных ВС связана с использованием распределенной обработки, и предполагает наличие соответствующей элементной базы. В настоящее время идея идет исключительно с ориентацией на класс фон-неймановских архитектур (*controlflow*-архитектур). В этом классе используется конечно-автоматная модель центрального управления вычислениями, реализуемая с помощью универсальных процессоров и компонентов коммутационной среды. Построение ВС с распределенным управлением на указанной элементной базе представляет собой лишь эмуляцию модели такого управления с помощью конечно-автоматной модели, что не может увеличить отношение производительность/цена в сравнении с чисто центральным управлением.

Следуя работе [1], конечно-автоматное управление распараллеливаемыми вычислениями можно представить компактней – в форме параллельной композиции более простых конечных автоматов. Эта композиция далее легко представляется на языке сетей Петри [2], описывающих параллельные вычисления. Главные компоненты таких сетей – *условия* и *действия* – представляют собой, по сути, управляющую и обрабатывающую часть ВС. Действия активируются уже не императивным способом – последовательно по командам, – а по мере готовности необходимых данных. Архитектуру ВС с моделью вычислений, основанной на сетях Петри, принято относить к классу *dataflow*-архитектур [3, 4], в которых поток данных всегда сопровождается соответствующей управляющей информацией (образуется поток *токенов*).

Важно отметить, что класс *dataflow*-архитектур не является альтернативным классу *controlflow*-архитектур, а является его расширением, которое обеспечивает активацию одних программируемых автоматов при получении результатов из других автоматов. Поэтому утверждения о разработке архитектур, не попадающих ни в один из упомянутых классов, необходимо воспринимать критически. Например, мультиклеточная архитектура [5] – один из вариантов dataflow-архитектуры, выполненной в форме регулярной вычислительной среды из четырех клеток,

Пока dataflow-архитектуры, несмотря на потенциально высокую способность параллельных вычислений, так и не были реализованы на уровне серийных образцов. Основная причина этого лежит в «мелкозернистости» обработки, когда токен несет в себе простые команды, время выполнения которых соизмеримо со временем его доставки. Увеличение числа обрабатывающих блоков приводит к их простоям. Положительный эффект от перехода к «крупнозернистой» обработке был смоделирован в [6] с использованием универсальных процессоров. В токены включались достаточно большие наборы команд и данных, которые могли обрабатываться автономно. Сами процессоры при этом были соединены согласно укрупненному информационному графу.

С развитием технологии интегральных схем снова возник интерес к dataflow-архитектурам, в первую очередь – в плане их использования в суперкомпьютерах. В работах [7, 8] исследована dataflow-архитектура с использованием специальной элементной базы, непосредственно реализующей компоненты сети Петри – *действия* и *условия*. Им были поставлены в соответствие вычислительные узлы и узлы ассоциативной памяти, которые попарно объединялись в модули. В вычислительные узлы предварительно загружались небольшие программы-шаблоны, задающие функции обработки. Это позволило формировать более «крупнозернистые» вычислительные кванты, что сразу снизило простои вычислительных узлов. Однако основные недостатки ранних вариантов *dataflow*-архитектуры остались:

– при обмене узлов между собой виртуальные адреса, заложенные в программах-шаблонах, необходимо преобразовывать в физические, что добавляет времени на обработку;
– для замены отдельного отказавшего узла нужны корректировка функции распределения узлов и восстановление в узле соответствующих программ-шаблонов;
– для обмена между узлами необходима внешняя коммутационная среда, ограничивающая размеры ВС.

Позволяющее довести *dataflow*-архитектуры до практического применения решение, связанное также с использованием специальной элементной базы, предложено в [9, 10]. Его идея заключается в реконфигурации архитектуры перед решением каждой задачи. Для минимизации времени решения выбирается необходимое число макропроцессоров, выполняется их настройка, устанавливаются необходимые связи между ними и распределенной памятью. Для обеспечения реконфигурации элементная база строится на программируемых логических интегральных схемах (ПЛИС). ВС с такой перестраиваемой архитектурой обеспечивают реальную производительность, близкую к пиковой, однако обладают очевидными недостатками: принципиальной однозадачностью, сложностью программирования, а также аппаратной избыточностью, являющейся характерной особенностью ПЛИС.

В [11] расширено число уровней управления до трех – к командному уровню добавлены *операторный* и *файловый*, реализуемые с привлечением специальной элементной базы, главными компонентами которой явились планировщики П (условия) и функторы Ф (действия). Использующие эти компоненты как основной строительный материал сети названы ПФ-сетями. В результате удалось перейти к более гибкой «крупнозернистой» обработке, и существенно сократить время простоев обрабатывающих блоков – функторов. Чтобы отказаться от упомянутого выше «каркаса», коммутационные функции заложены в сами компоненты решающего поля путем введения второго порта, обеспечивающего транзитную прокладку маршрутов при доставке данных и управляющей информации. Кроме того, программы-шаблоны (операторы) загружаются вместе с данными, что позволило отказаться от адресации вычислительных узлов (функторов), которые стали выбираться из числа свободных, а результаты возвращаться по ранее проложенному оператором маршруту. При этом каждый функтор вел учет всех использованных в нем операторов и в случае попытки повторного ввода какого-либо оператора в него вводились только переменные.

В [12] разработаны приемы параллельного программирования в ПФ-сетях на командном и операторном уровне, а в [13] с использованием этих приемов продемонстрированы простые и эффективные реализации различных задач обработки сигналов.

В настоящей работе развита тема применения специальной элементной базы при построении открытых ВС на основе ПФ-сетей с использованием двухуровневой компоновки – ПФ-ячеек на нижнем уровне и ПФ-серверов – на верхнем. Рассмотрены конфигурации ВС, которые могут представлять интерес также при создании суперкомпьютеров следующего поколения, в которых вместо универсальных процессоров будет использоваться специальная элементная база.

Принципы управления

Будем полагать, что имена данных и операторов задаются адресной парой (НС, А), где НС – номер страницы, А – начальный адрес на странице. Множество страниц, в которых удается поместить всю задачу или ее часть с собственной системой имен переменных и операторов, назовем *файлом*. Файлы одной задачи размещаются в одном планировщике. На каждой странице могут размещаться компоненты только одного файла. Когда в планировщик загружается файл, он включается в систему индексации, общую для всех задач, загруженных в этот планировщик. При перемещении пакета данных и операторов к их именам добавляется индекс файла (ИФ) и индекс планировщика в ячейке (ИП), а также, в зависимости от местоположения пакета, номера тех портов, через которых происходит передача. Это позволяет, возвращая результаты, использовать маршруты, ранее проложенные при загрузке операторов в функторы, и загружая переменные, выбирать маршруты, сформированные при загрузке файлов в концентраторы и планировщики.

На традиционном командном уровне небольшие программы (скрипты) и данные поступают в функторы в составе операторов. Другими словами, в отличие от обычных процессоров с полной структурой, программная память в функторе заполняется каждый раз при запуске очередного оператора. Результаты вычислений отправляются на операторный уровень в планировщик, который инициировал соответствующее вычисление. Использование перемещаемых операторов позволяет механически наращивать производительность вычислительной системы добавлением новых функторов.

Операторный уровень управления разбивается на два подуровня – функциональный и объектный.

Функциональный подуровень. Задача на этом подуровне представляется множеством F имен f_i формул и связями между ними. В это же множество включаются имена свободных переменных x_j как случай вырожденных формул.

На множестве F для каждого имени f_i формируются два подмножества. Первое из них $F^-(f_i)$ содержит имена формул, которые используются в качестве переменных в формуле f_i , а второе $F^+(f_i)$ включает имена формул, в которых f_i используются в качестве переменной. Для переменных x_j формируются только подмножества $F^+(f_i)$. Для формулы, определяющей конечный результат, формируется только подмножество $F^-(f_i)$.

Для каждой формулы f_i вводится вектор готовности $R(f_i)= \{p_1, p_2, \ldots\}$, где p_k – предикат с одним или двумя аргументами, в качестве которых используются значения формул $f_k \in F$. С помощью предикатов подготавливаются логические условия, на основании которых далее будет определяться готовность соответствующей формулы к вычислениям. Предикаты с двумя аргументами реализуют операции сравнения (больше, меньше и т.д.). Если условие сравнения выполняется, результату сравнения присваивается логическое значение «1». Предикат с одним аргументом-формулой принимает значение «1», когда появляется новое вычисленное значение этой формулы. Значения предикатов далее участвуют в качестве аргументов при вычислении полной готовности формулы f_i с помощью логической функции $r(f_i)$.

На рис. 1 приведен пример схемы вычислений, для которой на функциональном подуровне имеем:

$F=\{x_1, x_2, x_3, f_1, f_2, f_3\}$;

$F^+(x_1)=\{f_1, f_3\}$; $F^+(x_2)=\{f_1, f_2\}$;

$F^-(f_1)=\{x_1, x_2\}$; $F^+(f_1)=\{f_2\}$; $F^-(f_2)=\{x_2, f_1\}$; $F^-(f_3)=\{x_1\}$;

$R(f_1)=\{p(x_1), p(x_2)\}$; $r(f_1)=p(x_1) \wedge p(x_2)$;

$R(f_2)=\{p(x_2), p(x_3<f_1), p(x_1 \geq x_2)\}$; $r(f_2)=(p(x_2) \wedge (p(x_3<f_1) \vee p(x_1 \geq x_2))$;

$R(f_3)=\{p(x_3 \geq f_1), p(x_1 < x_2)\}$; $r(f_3)=p(x_3 \geq f_1) \vee p(x_1 < x_2)$.

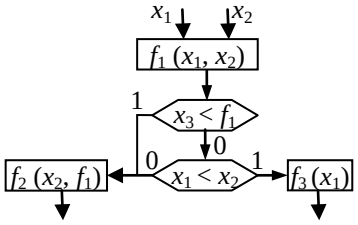

Рис. 1. Пример вычислений

Как только появляется новое значение переменной x_k или результата вычисления формулы f_s, выбирается множество $F^+(x_k/f_s)$, и для каждого его элемента f_t выбирается и корректируется вектор готовности $R(f_t)$. Затем вычисляется значение соответствующей функции $r(f_t)$. Если $r(f_t)=1$, то происходит переход на объектный подуровень операторного уровня. Таким образом, управление на функциональном подуровне задает *редукционный* способ вычислений, в котором выбор для вычислений формулы (*редекса*) происходит с помощью функции готовности r, идентифицирующей наличие в памяти значений всех аргументов, используемых в выбранной формуле,

Объектный подуровень. Для вычисления f_t выбирается оператор $H= \{f_{t_1}, f_{t_2}, \ldots f_{t_n}, f_i, S\}$, в котором f_{t_1}, f_{t_2}, f_{t_n}, – имена переменных, используемых в формуле f_t, в качестве аргументов, f_i – имя переменной, которой будет назначен результат (имя формулы, вычисление которой выполняет рассматриваемый оператор), S – программа вычисления формулы f_t. После раскрытия ссылок на переменные оператор поступает на самый нижний уровень – *командный*, который обеспечивает прием оператора, его распаковку и пошаговое исполнение.

Кроме операторов, непосредственно реализующих обработку в функторах, используются также транспортные операторы, обеспечивающие передачу результатов обработки из одних файлов в другие той же задачи. Например, в выполняемую подпрограмму с помощью транспортного оператора переносятся необходимые переменные из родительского файла, в который затем результат возвращается другим транспортным оператором [12].

Различаются две категории обрабатывающих операторов – нерегулярные (одноразовые) и регулярные (периодически повторяющиеся). Первые загружаются в выбранный функтор всегда целиком. В регулярных операторах допустима обработка переменных по частям (отдельными фрагментами), что позволяет распараллеливать вычисление формулы на множестве функторов, повторяя одну и ту же процедуру вычисления с различными данными. После обработки отдельных фрагментов регулярным оператором производится сборка полученных результатов. Такой вид обработки характерен для различных интегральных преобразований, в частности, быстрого преобразования Фурье (БПФ). В нем на функциональном подуровне множество F содержит формулы, задающие отдельные ступени этого преобразования.

Использование операторов, и в первую очередь регулярных, увеличивает «зернистость» обработки, что повышает эффективность использования многопроцессорных структур за счет уменьшения времени их простоя. Действительно, пусть программа содержит N команд, которые сгруппированы в m операторов. Для простоты положим, что операторы одного размера и одинаково время T_B выполнения элементарной арифметической команды и время T_A активации оператора (выборки оператора и его доставки в функтор). При загрузке каждого оператора в отдельный функтор общее время T выполнения программы будет $T= T_B N/m + m\ T_A$. Найдем значение m, при котором T будет минимально. Продифференцируем выражение для T по m и приравняем производную нулю:

$$dT/dm= -T_B \cdot N/m^2 + T_A=0, \text{ откуда } m= \sqrt{\frac{NT_B}{T_A}}\ . \quad (1)$$

Полагая, например, $T_B=T_A=T_0$, N=100, получаем в случае «крупнозернистой» обработки: m=10 и $T=10T_B +10T_A=20T_0$. В случае обычного командного управления $N=m$=100 и $T=T_B + 100T_A= 101T_0$. Таким образом, время при «крупнозернистой» обработке, в которой размеры m выбраны с учетом (1), быстродействие удается увеличить более чем в пять раз и использовать при этом в десять раз меньше обрабатывающих блоков, чем при «мелкозернистой» обработке.

Файловый уровень управления. Распараллеливание возможно также на более высоком уровне – файловом, например, при вычислении многомерных интегральных преобразований. Так, в двумерном БПФ можно организовать вычисления одновременно для нескольких строк (столбцов). В такой задаче на функциональном подуровне файлового уровня множество F состоит из одного имени f, назначенного файлу одномерного БПФ. Как только в концентраторе появляется исходный двумерный массив M для обработки, на объектном подуровне файлового уровня активируется файл под именем f (аналогично активации на операторном уровне оператора). К этому файлу в качестве исходных данных присоединяется первая строка массива M, с которой файл спускается на операторный уровень. На нем выбирается свободный планировщик, в котором отсутствует указанный файл, либо он есть, но уже обработан (в этом случае в планировщик вводится только обрабатываемая строка), после чего на операторном уровне выбранный планировщик инициирует вычисление БПФ введенной строки. Далее тот же файл f, но с другой строкой массива M, может быть направлен на операторный уровень в другой планировщик. Результат вычисления возвращается с операторного уровня на файловый, замещая в массиве M исходную строку на обработанную.

На файловом уровне также удобно хранить резервные копии файлов, которые в случае отказа планировщиков будут использоваться для восстановления их функций на других планировщиках.

Организация нижнего уровня в открытой ПФ-системе

Этот конструктивный уровень представляется ПФ-ячейками, компонентами которых являются планировщики, функторы, концентраторы, коммуникаторы и шлюзы. С помощью коммуникаторов обеспечиваются связи ячеек между собой. Коммуникатор имеет два внешних одноканальных порта $П_1^0$ и $П_1^1$ и один внутренний двухканальный порт $П_0^i$ (с временным разделением). В каждом канале используются буферные накопители. Каждому порту $П_1^i$ назначается номер.

Типы ячеек. Используются три типа ячеек: А, Б и В.

Ячейка переднего плана (*А-ячейка*) (рис. 2, а). Данный тип ПФ-ячейки используется для связи с внешними устройствами, а также для решения несложных задач с частым (периодическим) обращением к новым исходным данным. А-ячейка также удобна для предварительной обработки (сжатия) данных, которая позволяет далее ограничиться меньшим количеством данных на основном уровне обработки.

Для связи с периферийными устройствами используется *шлюз* (Ш). Задача переднего плана сразу поступает в планировщик А-ячейки. Запрос на загрузку задачи заднего плана перенаправляется через коммуникатор КМ на выход *X* или *Y*. Основной функцией шлюза является согласование форматов данных – внутреннего (параллельный код с плавающей запятой) с форматом внешних устройств (последовательный код с фиксированной запятой). Шлюз также накапливает данные для дальнейшей передачи и формирует интервалы обработки. Через шлюз под управлением внешнего процессора загружаются задачи и настраиваются заголовки пакетов, которые будут использоваться при обмене данными с внешними устройствами. В шлюзе используются: $П_0$ – ведомый последовательный *USB*-подобный порт для связи с внешним хост-процессором; $П_1$, …, $П_4$ – ведущие *USB*-подобные порты для связи с внешними устройствами. Внутренний порт идентичен портам планировщика и функтора.

Концентратор КН принимает загружаемые файлы и далее распределяет их по Б-ячейкам. При сбое в одних Б-ячейках концентратор переустанавливает необходимые файлы в других Б-ячейках. В случае больших массивов данных, допускающих параллельную обработку, концентратор размножает соответствующие файлы в различных Б-ячейках, и по ним далее распределяет для обработки отдельные части массива.

Ячейка заднего плана (*Б-ячейка*) (рис. 2, б). Ячейка этого типа предназначена для решения задач с достаточно большим временем выполнения, не требующих частой подкачки данных. Если возможностей концентратора А-ячейки по объему памяти недостаточно для всех обслуживаемых Б-ячеек, в них дополнительно может включаться местный концентратор.

Ячейка обработки (*В-ячейка*) (рис. 2, в). Ячейка содержит только функторы. Используется в качестве расширения Б-ячеек, если в них не хватает своих функторов для параллельной обработки.

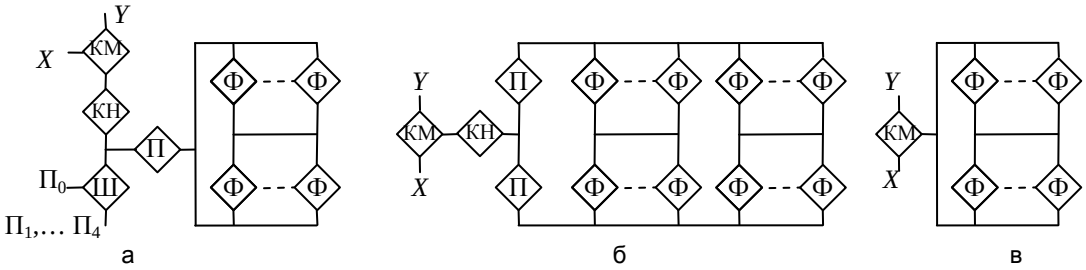

Рис. 2. Типы ячеек: ячейка переднего плана (а), ячейка заднего плана (б), ячейка обработки (в)

Организация связи между ячейками. При выходе из порта ячейки запроса на передачу нерегулярного оператора, номер этого порта добавляется к запросу. Это позволяет возвращать результат обработки в файл, из которого был активирован оператор.

Если прокладывается маршрут для регулярного оператора, коммуникатор, расположенный в начальной ячейке, включает в запрос имя управляющей точки (ИУТ), назначенной ячейке при загрузке в нее задач. В каждом коммуникаторе во время прохождения запроса также формируются два параметра:

- δ_1 – тип передачи: 0 – передача между наружными портами ($П_1^0 \leftrightarrow П_1^1$); 1 – передача через внутренний порт ($П_0^i \leftrightarrow П_1^j$);
- δ_2 – индекс наружного порта (0 – порт $П_1^0$, 1 – порт $П_1^1$), который в случае δ_1=0 задает входной наружный порт, а в случае δ_1=1 – наружный порт, используемый в передаче.

На рис. 3 приведены варианты передач в коммуникаторе в зависимости от параметров δ_i и δ_2.

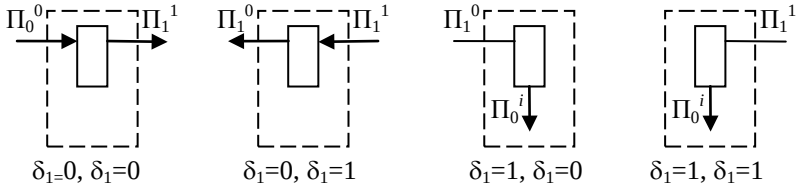

Рис. 3. Управление маршрутом внутри коммуникатора

Организация верхнего уровня открытых ПФ-систем

ПФ-сервер. Под этим будем понимать функционально и конструктивно законченный вычислитель, компонентами которого являются ПФ-ячейки. Частота обмена между ПФ-ячейками существенно ниже частоты обмена внутри ячеек, при обмене используется последовательно-параллельный интерфейс. В сравнении с последовательными шинами это позволяет уменьшить потери на электромагнитное излучение, исключить хабы и сохранить при этом высокую производительность. ПФ-серверы далее могут объединяться в кластеры, используя уже последовательную шину, например, *Gigabit Ethernet*. Возможность организации вычислений многих пользователей, независимо от их местоположения, позволяет просто реализовать грид-технологию [14], в которой для получения определенного ресурса на каком-то ПФ-сервере не требуется выполнять каких-либо настроек. Достаточно просто включиться в специальным образом организованную «розетку» и начать работу.

На рис. 4, а, решающее поле (пунктир) линейно сгруппировано. Если после загрузки задачи в Б-ячейку в ней для обработки не найдено свободных ресурсов, их ищут в В-ячейках сначала на шине в смежном порту. При отсутствии результата поиск переводится транзитом через одну из В-ячеек на следующую шину. В зависимости от занятости В-ячеек будет происходить переход ко все более удаленным В-ячейкам.

На рис. 4, б, доступ к элементам решающего поля ускорен за счет использования матричной структуры. Увеличение числа шин позволяет обеспечить не более двух переходов из любой Б-ячейки к любой В-ячейке поля. Матричная структура дает возможность сформировать еще одну линейку Б-ячеек.

Наращивание производительности в структурах на рис. 4 ограничено нагрузочными возможностями шин. Расширить эти возможности можно, используя трехмерную структуру решающего поля, как показано на

рис. 5. В ней увеличивается число линеек с Б-ячейками, которые образуют теперь прямоугольную сетку. Таким же образом «размножаются» сетки с В-ячейками, принимающие трехмерную структуру.

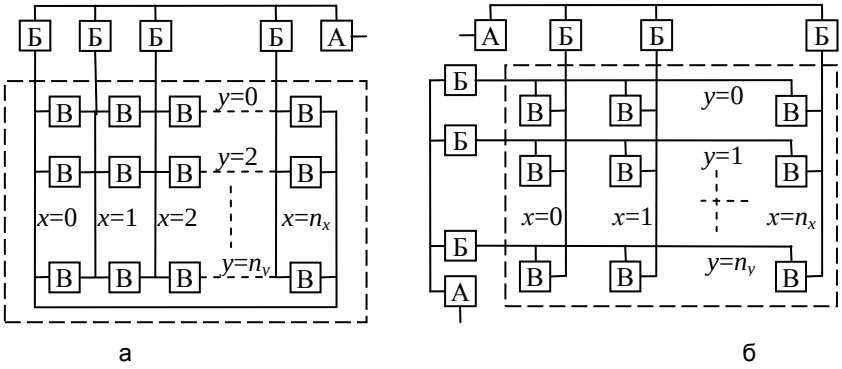

Рис. 4. Варианты ПФ-серверов с двумерной структурой решающего поля: с линейным группированием (а), с матричным группированием (б)

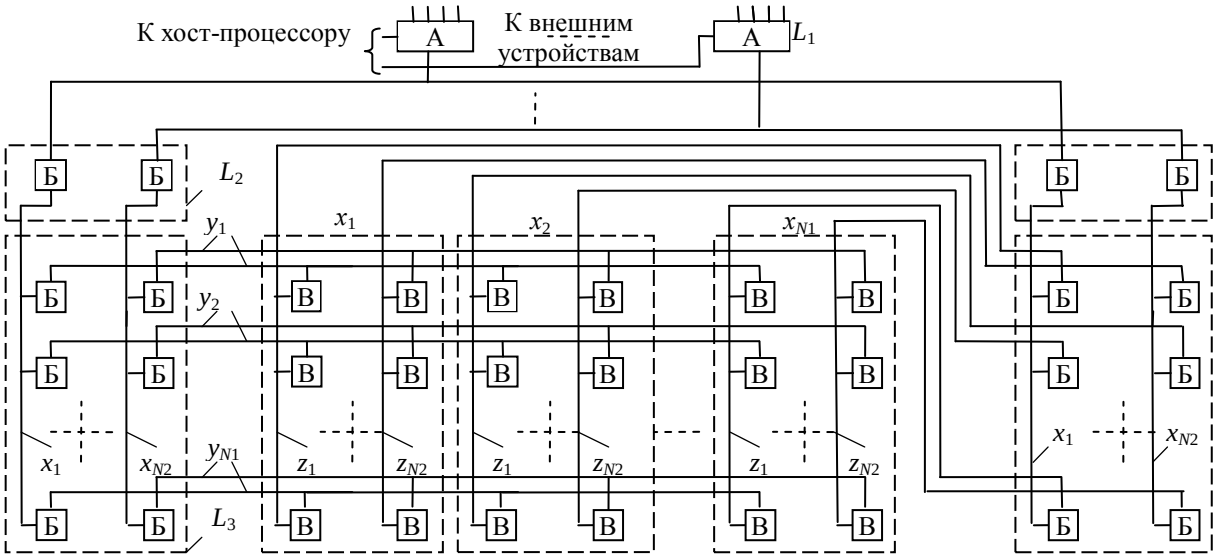

Рис. 5. ПФ-сервер с трехмерной структурой решающего поля

На рис. 5 выделены три уровня обработки. На уровне L_1 размещаются несложные задачи с высокой частотой обращения к новым исходным данным. На L_2 (верхняя строка Б-ячеек) – задачи с относительно редкой подкачкой новых данных. Уровень L_3 предназначен для задач с большим временем выполнения, исходные данные в которых вводятся, как правило, вместе с загружаемой задачей. Связь с периферией нечастая и нерегулярная. Повышение производительности вычислений достигается использованием трехмерного решающего поля, в котором доступ к любым В-ячейкам обеспечивается за один или два шага в случае прямого обращения, либо за два или три шага при перемещении транзитом через другие Б-ячейки.

Ввод задачи. *Поиск приемной ячейки и загрузка в нее задачи*. Вначале хост-процессор (ХП) выбирает А-ячейку, в которой шлюз связан с внешними устройствами (ВУ), используемыми в задаче. Все А-ячейки размещаются в позициях, которым назначены номера шлюзов (НШ). НШ являются базой для исходных адресов переменных, представленных первоначально адресной парой (НС, А), и включаются в запрос при поиске ячейки, которая должна принять задачу.

Задачи заднего плана размещаются в Б-ячейках вначале на втором уровне. Если на нем за время Δ на запрос от нескольких Б-ячеек получено подтверждение, что они обладают нужными ресурсами для размещения задачи, выбирается одна, например, с учетом близости к началу шины. Если за время Δ не обнаружено Б-ячейки с нужными ресурсами, происходит поиск Б-ячейки уже на третьем уровне.

К перемещающемуся по уровням запросу добавляются номера портов (НП) ячеек, через которые он перемещался. В итоге в коммуникаторе выбранной ячейки к запросу с номером НШ оказывается добавлена последовательность номеров, образующих маршрут загрузки МЗ={$НП_0$, $НП_1$ [$НП_2$]}, где $НП_0$ – номер выходного порта А-ячейки, выставившей запрос, $НП_1$ и $НП_2$ – номера шин передающих портов ячеек на втором и третьем уровне, через которые транзитом прошел запрос. Принятая последовательность номеров объединяется в коммуникаторе приемной ячейки в одно слово как ИУТ и сохраняется в приемной ячейке.

Ячейка может иметь несколько ИУТ по числу шлюзов, выполнивших загрузку задач в эту ячейку. Это имя будет далее использоваться при прокладке маршрутов для регулярных операторов (и возврата результатов их выполнения) и исходных данных, поступающих в задачу из определенных устройств.

В приемной Б-ячейке задача загружается вначале в концентратор, который назначает каждому файлу задачи в ячейке локальный индекс ИФ и индекс ИП приемного планировщика. Затем концентратор приемной ячейки возвращает в шлюз (по маршруту [$НП_2$] $НП_1$, $НП_0$) параметры загрузки файла – ИУТ, ИФ и ИП, которые будут использоваться для настройки пакетов с переменными, формируемыми периферийными устройствами. В процессе передачи указанных параметров в коммуникаторах всех ячеек на маршруте сохраняется ИУТ, а также формируются параметры δ_i: ячейке со шлюзом $\delta_1=0$, $\delta_2=i$, в транзитных ячейках $\delta_1=0$, в приемной ячейке $\delta_1=1$.

Настройка шлюза для передачи переменных (рис. 6). Возвращенные в шлюз параметры ИУТ, ИФ и ИП сохраняются в хэш-таблице заголовков (ТЗ) как часть полного имени переменных (ПИП), которые будут далее вводиться в только что загруженный файл. Дополнительно хост-процессор добавляет в ПИП для каждой входной переменной адресную пару (НС, А), а также реквизиты периферийных устройств, передающих или принимающих переменные: НУ – номер устройства; НКТ – номер конечной точки в устройстве (обычно это номер соответствующего регистра для ввода или вывода переменной). Таким образом, в хэш-таблице ТЗ полное имя переменной будет иметь вид:

ПИП=(НУ, НКТ, ИУТ, ИП, ИФ, НС, А).

Выполнение задачи (рис. 7). *Передача нерегулярного оператора.* Оператор выполняется, как правило, один раз и потому в функтор поступает всегда в полном составе. Когда запрос на передачу оператора из Б-ячейки $Я_1$ достигает В-ячейки $Я_3$ со свободным функтором, этот функтор захватывает шину m_2, с которой пришел запрос. Порт коммуникатора, через который только что был передан запрос, обнаружив захват, считывает свой НП в направлении приемной ячейки $Я_3$ и дает указание смежному наружному порту, через который был принят этот запрос, также установить захват на своей шине m_1. Как только захват достигает $Я_1$, она передает вначале свой $НП_1$, и затем остальные слова из пакета. К моменту, когда пакет достигнет $Я_3$, в нее будет уже введена последовательность $НП_2$, $НП_1$. Согласно этой последовательности результат будет возвращаться из $Я_3$ в $Я_1$.

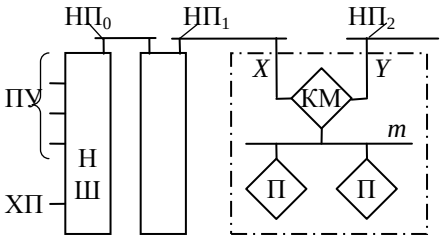

Рис. 6. Настройка планировщика

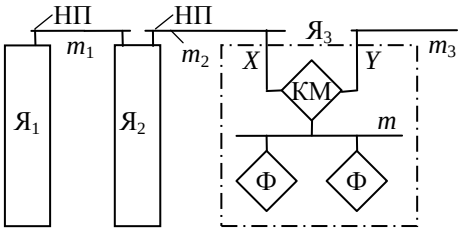

Рис. 7. Настройка функтора

Передача регулярного оператора. Этот вид оператора используется периодически и поэтому целесообразно сохранять его постоянную часть в функторе, чтобы сократить время ввода, и включать в состав оператора только переменную часть – значения переменных и маршрут возврата. Этот маршрут, в отличие от нерегулярного оператора, задается иным образом. В коммуникаторе каждой ячейки, через которую прошел запрос, запоминается ИУТ ячейки, которая инициировала передачу оператора. Это имя присоединяется к запросу в момент его выхода из $Я_1$. При первом проходе пакета с регулярным оператором ИУТ запоминается в коммуникаторах на маршруте. Если в коммуникаторе при последующих проходах найдено нужное ИУТ с параметром $\delta_1=1$ (концевая приемная ячейка $Я_3$), это означает, что передаваемый оператор ранее уже вводился в текущую ячейку. Если в ней в течение интервала времени Δ обнаруживается свободный функтор, происходит захват шины m_2 в приемном порту $Я_3$. Иначе запрос направляется через смежный порт ячейки на смежную шину m_3.

Если нужное ИУТ в коммуникаторе $Я_2$ найдено с параметром δ_1=0 (транзит), и на шине m_1 в течение времени Δ_1 не произошел захват, это означает, что на шине m_1 также нет других ячеек с требуемым ИУТ и признаком δ_1=1. В результате запрос транслируется через смежный порт $Я_2$ на шину m_2. Если далее ни в одном из наружных портов $Я_2$ в течение времени Δ_2 не обнаруживается захват, но в самой ячейке $Я_2$ появляется свободный функтор, в коммуникаторе этой ячейки сохраняется ИУТ с параметром δ_1=1 и формируется захват шины m_1, с которой пришел запрос. При обнаружении планировщиком в $Я_1$ захвата, пришедшего со стороны приемного функтора, передается остальная часть пакета с оператором.

Возврат результата выполнения нерегулярного оператора. Результат возвращается по маршруту согласно сохраненным в функторе номерам портов, через которые проходил запрос при прокладке маршрута. Если возврат происходит из ячейки $Я_2$, то она формирует маршрутную последовательность $\{НП_2, НП_1\}$.

Возврат результатов выполнения регулярного оператора. Результат возвращается согласно ИУТ, включенному в запрос. Ячейка, которая находит такое же имя в своем коммуникаторе, далее согласно параметрам δ_1 и δ_2 , сопровождающим ИУТ, определяет дальнейшее движение запроса – транзитом или внутрь ячейки,

Ввод исходных данных. Шлюз, как хост, при очередном опросе периферийного устройства с НУ и НКТ обнаруживает в этом устройстве новые данные, которые помещаются в буфер данных шлюза. Далее формируется пакет с этими данными для приемной ячейки следующим образом. Согласно параметрам опроса – НУ и НКТ, используемым в качестве ключа, в хэш-таблице ТЗ находится запись с ПИП, из которого извлекается заголовок (ИУТ, ИП, ИФ, НС, А) пакета. По нему последовательно находятся Б-ячейка, планировщик, страница и начальная позиция на ней. В это место считываются данные из буфера данных шлюза.

Вывод результатов из планировщика. Результат выводится по маршруту под ИУТ через порт, использовавшийся при загрузке задачи (это исключает выбор маршрутов при выводе регулярных операторов, которые также используют ИУТ). Результат поступает в буфер данных шлюза. Имя результата (ИП, ИФ, НС, А) применяется в качестве ключа в хэш-таблице ТЗ, в которой выбирается НУ приемного периферийного устройства и НКТ в нем. Согласно этим реквизитам результат из буфера данных выводится в выбранное устройство.

Пример обработки данных с использованием трех уровней управления

Рассмотрим преобразование $M^*=Ф_0(M, Ф_1(M_i), Ф_1(M_j))$, в котором массив М данных под общим управлением материнского файла $Ф_0$ преобразуется в Б-ячейке с помощью файла $Ф_1$ вначале по строкам M_i, а затем по столбцам M_j. Структурная схема вычислений показана на рис. 8. Сначала в накопитель НО объектного подуровня файлового уровня вводятся файлы $Ф_0$ и $Ф_1$, а затем массив М. Далее под управлением файла $Ф_0$ файл $Ф_1$ и первая строка M_1 массива М копируются в один из планировщиков ячейки, который, считывая соответствующие операторы, запускает обработку строки в функторах. Обработанная строка из планировщика возвращается на старое место в НО концентратора, который затем выбирает для обработки следующую строку. Аналогично выполняется обработка столбцов.

Для доступа к НО файлового и операторного уровня используются хэш-таблицы адресов ТА. В таблице концентратора согласно индексам массива (ИМ) или файла (ИФ) выбирается физическая страница и на ней – адресное смещение. В планировщике для доступа к переменным и операторам сначала согласно ИФ, в котором они определены, и пользовательскому номеру страницы (НС), определяется физическая страница. Затем на этой странице согласно пользовательскому смещению А выбирается ячейка. Процедура вычислений выглядит следующим образом.

1. В концентратор из шлюза А-ячейки через коммуникатор на вход *D* накопителя НО поступает пакет $П_1$, в состав которого входят МЗ и файлы $Ф_0$ и $Ф_1$. Перед вводом в НО файла $Ф_0$ ему назначается $ИФ_0$ и в таблице ТА делается запись, которая $ИФ_0$ ставит в соответствие номер физической страницы, используемый как базовый адрес $А_б$. Начальный адрес $А_с$ на странице при этом для материнского файла устанавливается равным нулю. Далее делается то же самое для файла $Ф_1$, но начальным адресом становится увеличенный на один текущий адрес на странице.
2. Распаковывается $Ф_0$, и для всех переменных, непосредственно входящих в $Ф_0$, назначаются индексы в продолжение индексации файлов. В нашем случае для М назначается ИМ= $ИФ_0$+ПН, где ПН – пользовательский номер массива М. Для вложенных файлов в пакете $П_1$ номер ПН явно не задается, а определяется порядковым номером размещения в материнском файле. Далее делаются записи в ТА, резервирующие необходимое количество памяти для этих переменных. В статусе $Ф_0$, организованного в НФ, сохраняются ИУТ, полученное путем сжатия МЗ до одного слова, и индексы вложенных файлов.

3. По ранее проложенному МЗ, но в обратном направлении ($МЗ^{-1}$) в шлюз в составе $П_2$ возвращаются параметры ИУТ и $ИФ_0$, При этом в каждом коммуникаторе на маршруте сохраняется ИУТ и устанавливаются параметры δ_1 и δ_2.
4. В ячейку поступает пакет $П_3$ с исходным массивом М данных, который сохраняется в зарезервированной области НО согласно записи в ТА. Для выбора этой записи вычисляется ИМ=$ИФ_0$+ПН. Начальный адрес массива на выбранной странице устанавливается в статусе файла $Ф_0$ (в поле А значение 0).
5. Файл $Ф_0$, обнаружив ввод М, активирует транспортный оператор, который из НО в составе пакета $П_4$ считывает файл $Ф_1$ и за ним согласно ИМ и начальному адресу, выбранному в поле А статуса файла $Ф_0$, первую строку $М_1$ под именем НС. В пакет $П_4$ из статуса включается в качестве адреса $А_{ВК}$ возврата содержимое поля А, индекс $ИФ_0$ и имя ТО транспортного оператора, который будет возвращать результат обработки строки в концентратор. Пакет вводится в планировщик, не занятый обработкой М. Если в планировщике ранее уже был загружен $Ф_1$, он повторно не вводится. Далее в планировщике в файле с $ИФ_1$ находится ТО, в который вводятся $А_{ВК}$ и $ИФ_0$. Если ячейка содержит несколько планировщиков, может сразу начаться активация обработки следующей строки $М_{i+1}$, для чего в поле А статуса обновляется начальный адрес.
6. При вводе файл $Ф_1$ сразу инсталлируется. Обнаружив далее ввод строки $М_i$, планировщик выбирает свободный функтор (из своей ячейки либо из В-ячеек решающего поля) и вводит в него пакет $П_5$ с обрабатывающим оператором Н. В случае выбора функтора из В-ячейки и использования в ней регулярного оператора в маршрут возврата добавляется еще параметр ИУТ.
7. Функтор, обработав фрагмент $М_i(s)$ строки, возвращает в планировщик на старое место согласно $А_{ВП}$ результат $М_i^*(s)$ в составе $П_6$.
8. Если обработан последний фрагмент строки, планировщик возвращает в концентратор в составе пакета $П_7$ обработанную строку $М_i^*$ (на место строки $М_i$), используя ТО, настроенный в пункте 5.
9. Если обработанная строка не последняя, то в статусе $Ф_0$ в поле А формируется начальный адрес новой строки $А_{ВК}^+$=А+Р, где Р – расстояние между строками. Иначе активируется аналогичная процедура обработки столбцов.

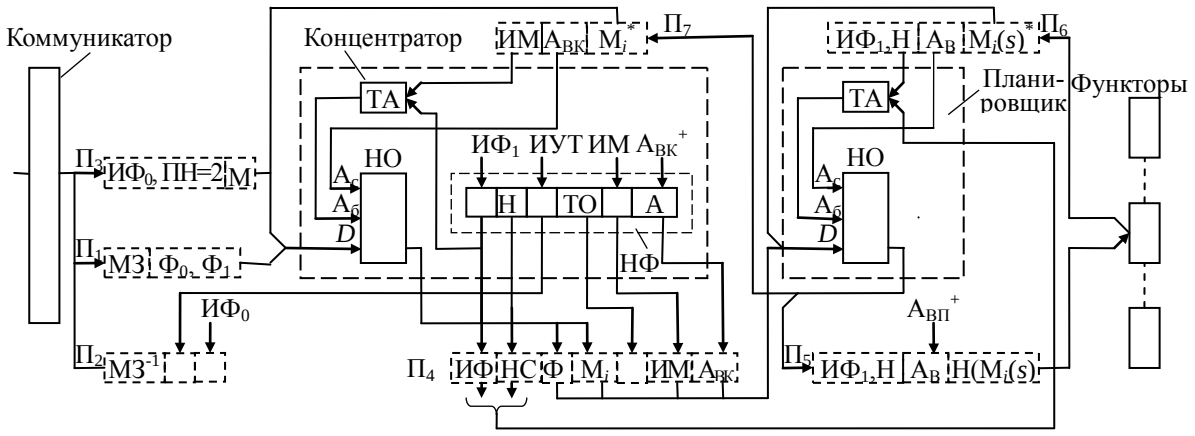

Рис. 8. Пример обработки с использованием трех уровней управления

Заключение

Благодаря наличию в рассмотренной элементной базе многоуровневого управления возможно просто организовывать эффективное «крупнозернистое» распараллеливание на уровне отдельных операторов и файлов, а также практически снять ограничения на размеры адресного пространства. На нижнем уровне используется быстрая традиционная адресация с помощью пары (НС, А), а на верхнем уровне определяется число А-ячеек и Б-ячеек, обеспечивающих соответственно параллельный и последовательный способ расширения адресного пространства.

Простота масштабирования открытых вычислительных систем обеспечена использованием двухуровневой распределенной коммутирующей среды – внутри ячеек путем организации транзитных передач через планировщики и функторы, и между ячейками путем использования специального компонента – коммуникатора. Кроме того, применение в решающих полях вместо универсальных процессоров специализированных компонентов – планировщиков и функторов – позволило более гибко выполнять масштабирование, независимо наращивая соответственно управляющую и обрабатывающую часть. При этом пользователю достаточно лишь погрузить задачу в свободную управляющую память планировщиков, и задача будет сама разворачиваться внутри вычислительной системы практически без какого-либо взаимодействия с внешней средой и при необходимости периодически обращаться к

средствам подкачки новых данных при работе в реальном времени. Важно отметить, что получаемая в результате масштабирования избыточность автоматически используется для обеспечения живучести вычислительной системы путем использования копий файлов, сохраненных в концентраторах, которые в случае отказов вновь инсталлируются на других исправных планировщиках.

В процессе вычислений, благодаря распределенному управлению, исключается большая часть функций операционной системы, которые теперь реализуются на аппаратном уровне в составе элементной базы – управление вводом и выводом данных, доступ к общим ресурсам (памяти, шинам), обнаружение сбоев и отказов с последующим восстановлением системы и др.

Предложенный состав элементной базы обеспечивает функциональную полноту при построении законченных вычислительных систем, включая организацию ввода и вывода информации.

Литература

1. Ланцов Р.А. 3-уровневая dataflow-архитектура для обработки больших потоков сигналов // Инновационные внедрения в области технических наук. Москва, 2018. С. 7–12.
2. Питерсон Дж. Теория сетей Петри и моделирование систем. М.: Мир, 1984. 264 с.
3. Dennis J.B., Misunas D.P. A preliminary architecture for a basic data flow processor // ACM SIGARCH Computer Architecture News. 1974. V. 3. N 4. P. 126–132. doi: 10.1145/641675.642111
4. Ланцов Р.А. *Dataflow*-архитектура для обработки сигналов сегодня и в перспективе // Вестник КГТУ им. А. Туполева. 2016. № 3. С. 141–149.
5. Стрельцов Н.В. Организация мультиклеточной обработки // Труды IV Международной научной конференции «Параллельные вычисления и задачи управления». Москва, 2008.
6. Young S.D., Wills R.W. Performance Analysis of a Large-Grain Dataflow Scheduling Paradigm. NASA Langley, Virginia, 1993. 8 p.
7. Климов А., Окунев А., Степанов А. Исследование возможностей управления распределением вычислений по вычислительным модулям [Электронный ресурс]. URL: ipmce.ru/about/press/articles/issled_rasvich, своб. (дата обращения: 03.10.2018).
8. Климов А.В., Окунев А.С., Степанов А.М. Проблемы развития модели вычислений dataflow и особенности ее архитектурной реализации [Электронный ресурс]. URL: ipmce.ru/about/press/articles/problem_dataflow, своб. (дата обращения: 03.10.2018).
9. Дмитриенко Н.Н., Каляев И.А. и др. Система многопроцессорных вычислительных систем с динамически перестраиваемой структурой [Электронный ресурс]. Таганрог. URL: fpga.parallel.ru/papers/dmitrenko.pdf, своб. (дата обращения: 03.10.2018).
10. Voigt S., Baesler M., Teufel T. Dynamically reconfigurable dataflow architecture for high-performance digital signal processing // Journal of Systems Architecture. 2010. V. 56. N 11. P. 561–576. doi: 10.1016/j.sysarc.2010.07.010
11. Ланцов А.Л., Ланцов Р.А. Использование ПФ-сетей для скоростных математических вычислений // Вестник КГТУ им. А. Туполева. 2016. № 4. С. 125–135.
12. Ланцов Р.А. Основы параллельного управления в ПФ-сетях // Вестник КГТУ им. А. Туполева. 2017. № 1. С. 96–105.
13. Ланцов А.Л., Ланцов Р.А. Реализация быстрого преобразования Фурье в ПФ-сетях // Вестник КГТУ им. А. Туполева. 2017. № 1. С. 106–115.
14. Демичев А.П., Ильин В.А., Крюков А.П. Введение в грид-технологии. Препринт НИИЯФ МГУ-2007-11/832.

References

1. Lantsov R.A. Level dataflow-architecture for processing of large signal flows. *Proc. Innovative Implementations in Technical Sciences*. Moscow, 2018, pp. 7–12. (in Russian)
2. Peterson J.L. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, 1981, 257 p.
3. Dennis J.B., Misunas D.P. A preliminary architecture for a basic data flow processor. *ACM SIGARCH Computer Architecture News*, 1974, vol. 3, no. 4, pp. 126–132. doi: 10.1145/641675.642111
4. Lantsov R.A. Dataflow-architecture for processing signals today and in perspective. *Vestnik KGTU im. A.N. Tupoleva*, 2016, no. 3, pp. 141–149. (in Russian)
5. Strel'tsov N.V. Organization of multicellular processing. *Proc. 4th Int. Conf. on Parallel Computing and Control Problems*. Moscow, 2008. (in Russian)
6. Young S.D., Wills R.W. Performance Analysis of a Large-Grain Dataflow Scheduling Paradigm. NASA Langley, Virginia, 1993, 8 p.
7. Klimov A., Okunev A., Stepanov A. *Study of the possibilities of managing the computations distribution by computational modules.* Available at: ipmce.ru/about/press/articles/issled_rasvich (accessed: 03.10.2018).
8. Klimov A., Okunev A., Stepanov A. *Problems of development of dataflow computing model and features of its architectural implementation*. Available at: ipmce.ru/about/press/articles/problem_dataflow (accessed: 03.10.2018).
9. Dmitrienko N.N., Kalyaev I.A. et al. *System of multiprocessor computing systems with a dynamically tunable structure*. Available at: fpga.parallel.ru/papers/dmitrenko.pdf (accessed: 03.10.2018).
10. Voigt S., Baesler M., Teufel T. Dynamically reconfigurable dataflow architecture for high-performance digital signal processing. *Journal of Systems Architecture*, 2010, vol. 56, no. 11, pp. 561–576. doi: 10.1016/j.sysarc.2010.07.010
11. Lantsov A.L., Lantsov R.A. Use PF-networks for speed mathematical calculation. *Vestnik KGTU im. A.N. Tupoleva*, 2016, no. 4, pp. 125–135. (in Russian)
12. Lantsov R.A. Basis of parallel control in PF-networks. *Vestnik KGTU im. A.N. Tupoleva*, 2017, no. 1, pp. 96–105. (in Russian)
13. Lantsov A.L., Lantsov R.A. Implementation of FFT in PF-network. *Vestnik KGTU im. A.N. Tupoleva*, 2017, no. 1, pp. 106–115. (in Russian)
14. Demichev A.P., Il'in V.A., Kryukov A.P. Introduction to Grid Technology. Preprint *NIINP MSU-2007-11/832*. (in Russian)

Автор

Ланцов Руслан Александрович – директор, ООО «Стримлайн ДС», Москва, 125212, Российская Федерация, ORCID ID: 0000-0003-2219-0522, ruslantsov@yandex.ru

Author

Ruslan A. Lantsov – Director, Streamline DS, Moscow, 125212, Russian Federation, ORCID ID: 0000-0003-2219-0522, ruslantsov@yandex.ru