УДК 004.27

Варианты реализации контроллера памяти параллельной потоковой вычислительной системы для работы с векторными и многовходовыми узлами

Н.Н. Левченко, А.С. Окунев, И.О. Шурчков, Д.Е. Яхонтов

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

***Аннотация* — В статье рассматривается архитектура параллельной потоковой вычислительной системы «Буран», описаны механизмы работы одного из основных модулей системы – процессора сопоставления. Исследована возможность работы системы с многовходовыми и векторными вычислительными узлами. Обозначены требования к контроллеру памяти, необходимые для работы с новыми структурами данных, а также предложены алгоритмы их реализации.**

***Ключевые слова* — параллельная потоковая вычислительная система; векторные вычисления; контроллер памяти; алгоритмы распределения памяти.**

I. Введение

В настоящее время в ИППМ РАН ведется разработка параллельной потоковой вычислительной системы (ППВС) "Буран", основанной на гибридной модели вычислений [1]. Данная модель объединяет элементы фон-неймановской архитектуры с принципами управления потоком данных (dataflow).

Вычислительная задача представляется в виде потокового графа, каждый узел которого является последовательной программой. Ребра графа отражают зависимости между узлами по данным. От узла к узлу информация передается в виде токенов. Токен представляет собой структуру, которая содержит служебный заголовок, поле данных, а также ключ – идентификатор, однозначно определяющий положение токена на вычислительном графе задачи.

Система "Буран" состоит из большого числа вычислительных модулей, каждый из которых, в свою очередь, состоит из вычислительных ядер (рис. 1). Ядро включает в себя три основных элемента:

1) *Процессор сопоставления* (или ассоциативный процессор) служит для поиска наборов токенов, адресованных в один узел, на основе их ключей. На выходе процессора сопоставления формируется *пакет* – структура, содержащая данные из всех токенов набора.

2) *Исполнительное устройство* служит для выполнения программ узлов. Входные аргументы принимаются в виде пакета, результат выполнения программы – набор токенов.

3) *Блок хэширования и коммутации* предназначен для маршрутизации токенов между ядрами на основе вычисления хэш-функции от значения ключа. Токены с совпадающими ключами направляются в одно ядро.

II. Архитектура процессора сопоставления

Процессор сопоставления выполняет две основные функции: поиск токенов с совпадающими ключами и хранение токенов, которые ожидают сопоставления. Для выполнения поиска и сравнения ключей токенов служит ассоциативное запоминающее устройство (АЗУ), называемое памятью ключей (ПК).

Аппаратная ассоциативная память позволяет максимально быстро выполнять поиск и сравнение клю-

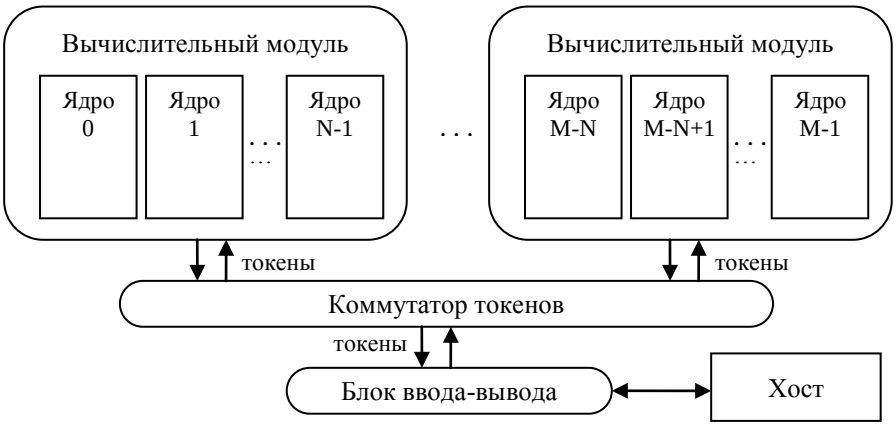

Рис. 1. Структурная схема ППВС «Буран»

чей, причем, в отличие от программно-аппаратных реализаций, таких как хэш-таблицы, поддерживается работа с масками и обработка множественных совпадений. Вместе с тем, максимальный объем блока ассоциативной памяти ограничен энергетическими соображениями. Кроме того, ячейка ассоциативной памяти устроена сложнее, то есть занимает большую площадь на кристалле, чем ячейка прямоадресуемой памяти (RAM).

Указанные причины заставляют искать пути уменьшения объема АЗУ без снижения эффективности работы. Был предложен следующий механизм: в АЗУ хранятся только ключи, в то время как для хранения самих токенов используется прямоадресуемая память, которая обладает гораздо лучшими показателями по энергопотреблению и занимаемой площади.

Каждый токен, поступающий на вход процессора сопоставления, запускает ассоциативный поиск среди ключей, хранящихся в ПК. При наличии соответствия между ключом входного токена с каким-либо из ключей, хранящихся в ПК, формируется адрес совпадения на выходе ассоциативной памяти.

Полученный адрес совпадения поступает на адресный вход памяти дескрипторов (ПД). ПД представляет собой прямоадресуемое запоминающее устройство, число ячеек которого равно числу ячеек ПК. Каждая ячейка ПД содержит дескриптор токена, а также поля, необходимые для выбора дальнейшего алгоритма обработки токена – "код операции" и "кратность".

Дескриптор представляет собой адрес, по которому токен хранится в памяти токенов (ПТ). ПТ – это также прямоадресуемое ЗУ, в котором хранятся токены целиком, включая заголовки, ключи и поля данных. Токен, считанный из ПТ, направляется на формирователь пакетов, вместе с токеном, пришедшим на вход процессора сопоставления.

Запись токена в память процессора сопоставления происходит в обратной последовательности: токен записывается в ПТ, его адрес – в ПД, а ключ – в ПК. Последние две операции могут выполняться параллельно.

III. Многовходовые и векторные токены

В ряде вычислительных задач [2] отмечается рост производительности в случае, когда пакет содержит не два, а большее количество полей данных. Также в этом случае снижается удельная загрузка АЗУ, так как на фиксированное количество обрабатываемых данных приходится меньше операций ассоциативного поиска. Было предложено два новых типа токенов: многовходовые токены (*M-токены*) и векторные токены (*B-токены*).

A. *Многовходовые токены*

В отличие от стандартных токенов, при каждом взаимодействии которых образуется пакет, M-токены объединены в *серии*. Пакет формируется только после того, как получены все токены серии (рис. 2). До этого момента полученные токены хранятся в памяти сопоставления, причем на всю серию выделяется одна ячейка АЗУ и один дескриптор.

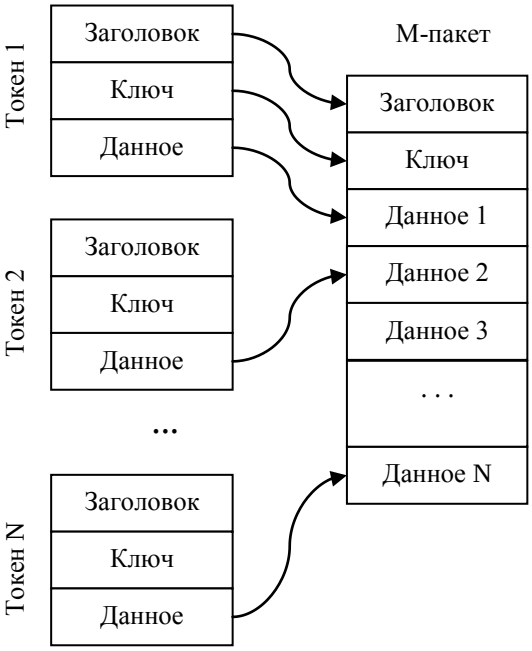

Рис. 2. Схема формирования M-пакета

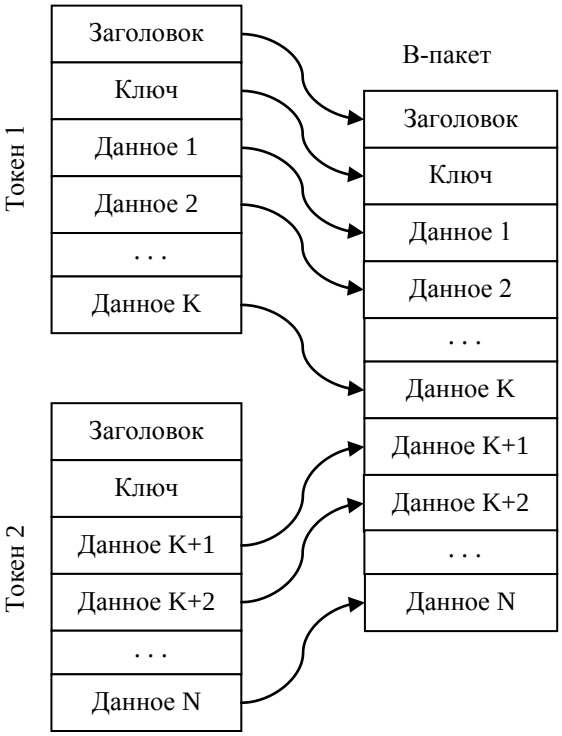

Рис. 3. Схема формирования B-пакета

B. *Векторные токены*

B-токен содержит не одно поле данных, как стандартный, а несколько. Возможность работать не с отдельными операндами, а сразу с векторами, увеличивает эффективность работы исполнительных уст-

ройств, снижает нагрузку на АЗУ и устройства маршрутизации. Работа с B-токенами со стороны устройства сопоставления ничем не отличается от описанной выше, за исключением того, что в ПТ хранится не одно поле данных, а множество (рис. 3).

IV. Контроллер памяти токенов

Введение многовходовых и векторных токенов требует нового контроллера памяти токенов с возможностью распределять память блоками переменного размера. К контроллеру памяти токенов предъявляются следующие требования:

1) выделение блоков памяти размером от 4 до 64 машинных слов (длина слова 64 бита);

2) доступ на чтение и запись по произвольному адресу внутри блока;

3) возможность многократного перераспределения памяти на блоки.

Предложено несколько механизмов распределения памяти, удовлетворяющих указанным требованиям.

A. Статическое распределение

Все адресное пространство ПТ делится на несколько сегментов. Память в пределах первого сегмента выделяется и освобождается блоками максимального размера – по 64 слова, в пределах второго – блоками по 32 слова и т.д. вплоть до размера блока 4 слова в последнем сегменте (рис. 4).

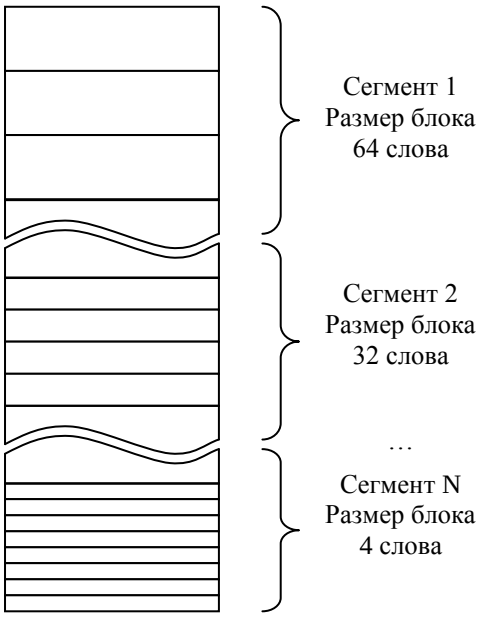

Рис. 4. Статическое распределение памяти токенов

Поскольку ожидаемое количество данных в токене заранее известно, для хранения токена выделяется блок минимально необходимого размера в соответствующем сегменте. Свободное место в конце блока не используется. Размеры сегментов могут быть фиксированными либо могут настраиваться перед запуском вычислительной задачи, например, на основе информации о статистическом распределении токенов по размеру.

Данный алгоритм просто реализуется и обеспечивает высокую производительность. Недостаток описанного механизма в том, что возможно заполнение одного из сегментов во время работы ВС, в то время как остальные могут оставаться свободными. Данная проблема решается применением динамического распределения.

B. Одноуровневое динамическое распределение

Изначально вся память разбита на блоки максимального размера (64 слова). Каждый такой блок верхнего уровня может быть выделен под хранение токена целиком (рис. 5), а может быть разбит на более мелкие подблоки (2 по 32 слова, 4 по 16, 8 по 8 или 16 по 4). Перед записью токена выполняется поиск свободного подблока минимально необходимого размера. Если таковой найден, выполняется запись в него, если не найден, один свободный блок верхнего уровня разбивается в соответствии с требуемым размером.

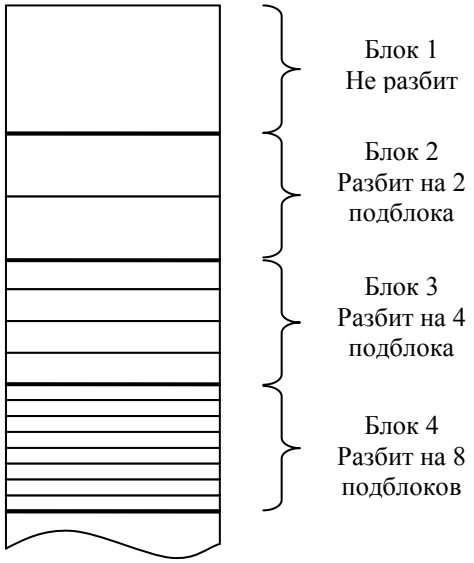

Рис. 5. Одноуровневое динамическое распределение памяти токенов

При описанном алгоритме работы возникает проблема фрагментации памяти. Для борьбы с ней необходимо время от времени перемещать данные внутри ПТ так, чтобы минимизировать число блоков верхнего уровня, заполненных частично. При этом разбитый блок верхнего уровня, все подблоки которого освобождаются, должен помечаться как свободный и целый. Дефрагментация может проводиться путем обмена содержимым между занятым подблоком и свободным (рис. 6).

Для того, чтобы сохранить возможность доступа к токенам по дескрипторам, хранящимся в ПД, вводится таблица обмена дескрипторов. В данной таблице хранятся пары "дескриптор до обмена – дескриптор после обмена". При получении дескриптора от ПД выполняется поиск по таблице обмена. Если блок, на который указывает дескриптор, был перемещен, из таблицы

обмена извлекается новый адрес, и операции с ПТ производятся уже по нему. Также новый дескриптор записывается в ПД вместо старого, а соответствующая запись из таблицы обмена удаляется.

По сравнению со статическим, механизм динамического распределения позволяет более полно использовать память. В то же время, реализация данного алгоритма требует больших аппаратных ресурсов. В связи с использованием таблицы обмена дескрипторов возрастает время доступа к памяти токенов со стороны ПД.

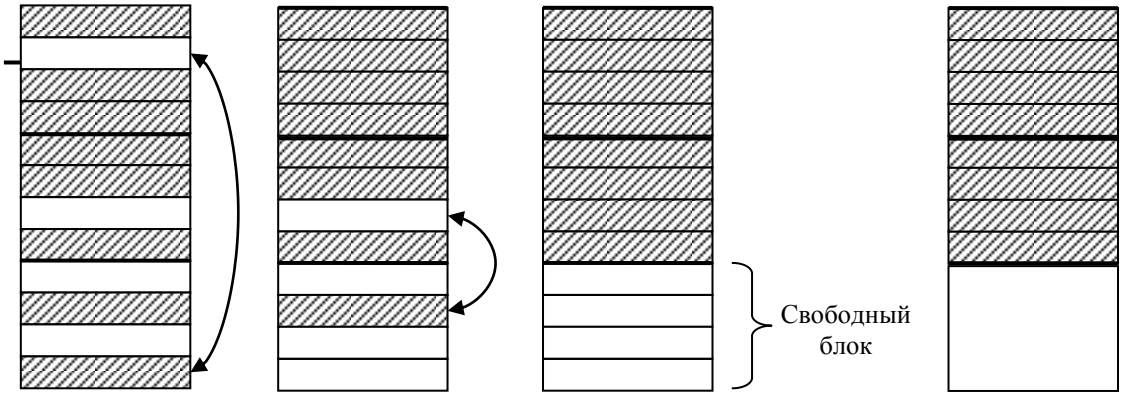

Рис. 6. Процесс дефрагментации памяти токенов (занятые подблоки заштрихованы)

C. Многоуровневое динамическое распределение

Данный алгоритм является развитием предыдущего. Память разбивается на блоки первого уровня размером 64 слова. Каждый блок первого уровня может быть выделен под хранение токена, а может быть разбит на два блока второго уровня размером по 32 слова. Аналогично каждый блок второго уровня может быть разбит на два блока третьего уровня и т.д. (рис. 7).

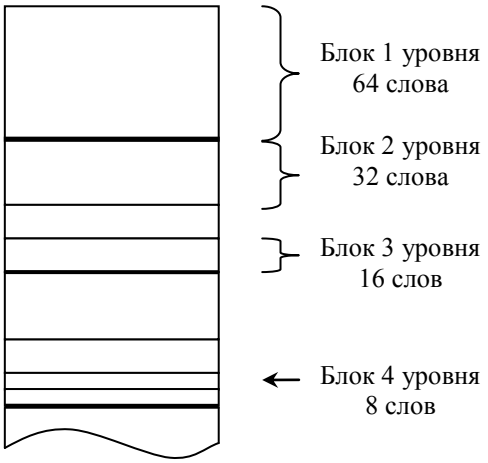

Рис. 7. Многоуровневое динамическое распределение памяти токенов

При записи токена выполняется поиск свободного блока минимально необходимого размера. Если такой блок не найден, ищется блок на один уровень выше, который затем разбивается на два.

Как и в случае одноуровневого распределения, требуется периодическая дефрагментация памяти. При этом освободившиеся блоки уровня N "склеиваются" в блоки уровня N-1 и т.д. до блоков первого уровня. По сравнению с одноуровневым, механизм многоуровневого распределения сложнее реализуется, но позволяет эффективнее распределять память и требует менее частой дефрагментации.

V. Заключение

В настоящей статье рассмотрена архитектура параллельной потоковой вычислительной системы, исследованы возможности работы системы с многовходовыми и векторными вычислительными узлами. Предложено несколько механизмов работы контроллера памяти токенов, обеспечивающих хранение и обработку новых структур данных.

Дальнейшая работа в данном направлении предполагает экспериментальную проверку предложенных механизмов путем RTL-моделирования. Также планируется провести более детальный анализ алгоритмов дефрагментации памяти в ППВС, применяя их к реальным вычислительным задачам.

Литература

[1] 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 // Информационные технологии. 2008. №10. С. 2 – 7.

[2] Левченко Н.Н., Окунев А.С. Исследование применения многовходовых узлов в параллельной потоковой вычислительной системе для задач фильтрации // Материалы Пятой Международной молодежной научно-технической конференции «Высокопроизводительные вычислительные системы (ВПВС 2008)». 2008, Таганрог, Россия. С. 367 – 370.