УДК 004.2;004.31

Исследование производительности процессора обработки структур в системе с многими потоками команд и одним потоком данных

© А.Ю. Попов

МГТУ им. Н.Э. Баумана, Москва, 105005, Россия

В ходе проекта, проводимого на кафедре «Компьютерные системы и сети» МГТУ им Н.Э. Баумана, разработана вычислительная система с многими потоками команд и одним потоком данных, в которой реализованы новые архитектурные принципы обработки структурированной информации. Механизмы хранения структур данных и доступа к ним реализованы на специализированном процессоре обработки структур, который способен на аппаратном уровне выполнять такие операции, как добавление, удаление, поиск, пересечение, дополнение, объединение структур и др. Преимущество этой системы — возможность параллельного исполнения частей вычислительных задач, связанных с доступом к структурам данных и арифметико-логической обработкой информации. Описаны основные механизмы доступа к данным, приведены результаты экспериментов измерения производительности процессора обработки структур при выполнении основных операций. Представлены результаты сравнения аппаратной сложности реализации процессора обработки структур и универсальных микропроцессоров, выполняющих аналогичные действия.

Ключевые слова: *структура данных, вычислительная система, МКОД, процессор, обработка структур, B+ дерево.*

Вычислительная система с многими потоками команд и одним потоком данных (МКОД), разработанная и тестируемая на кафедре «Компьютерные системы и сети» МГТУ им. Н.Э. Баумана, использует параллельную обработку структур данных на основе специализированного аппаратного блока — процессора обработки структур. Принцип функционирования системы МКОД основан на параллельной обработке двух составляющих данных: информационной и структурной [1, 2]. Структурная определяет взаимосвязь данных, в то время как информационная состоит из самих данных, используемых в ходе вычислительного процесса. В традиционных вычислительный машинах и системах структурная составляющая позволяет организовать данные в одном из известных видов структур: массив, список, дерево.

В [3, 4] показано, что в реализованном варианте построения системы можно выполнять параллельную обработку тех ветвей алгоритма, в которых ранее не выделяли параллельных нитей вычисле-

ний. Например, в известном алгоритме Дейкстры поиска кратчайших путей на графе можно выделить два потока команд. Команды вычисления длин путей выполняются на универсальном микропроцессоре, в то время как процессор обработки структур выполняет поиск кратчайшего из них.

Принципы функционирования процессора обработки структур. Выполнение программ в ЭВМ начинается с фазы инициализации, когда происходит загрузка информации в оперативную память. Эта фаза достаточно плохо подвергается параллельной обработке, так как предполагает последовательное чтение информации из источника (например, периферийного устройства, диска или оперативной памяти), обработку и последовательную запись их в оперативную память. Форматы представления данных, целесообразные при их передаче, часто не соответствуют форматам представления данных при выполнении программ. «Узким местом» в процедуре инициализации является центральный процессор, который принимает, обрабатывает и сохраняет информацию в виде структуры данных в оперативной памяти. При использовании нескольких микропроцессоров удается совместить действия по загрузке информации и обработке структур данных.

Во время фазы выполнения программы в ЭВМ с одним потоком команд и одним потоком данных центральный процессор осуществляет как арифметическую и логическую обработку информации, так и действия со структурами данных. Последовательность запросов к структуре может быть выделена из общего алгоритма и обработана отдельно, однако этого не происходит по причине недостаточной аппаратной поддержки. В настоящее время не применяются какие-либо специализированные устройства, способные хранить и обеспечивать доступ к структурам данных независимо от центрального процессора. В системах с многими потоками команд и многими потоками данных такая обработка возможна на одном из вычислительных узлов, но требует применения специальных технологий параллельного программирования, затрудняющих процесс проектирования программного обеспечения.

Разработанная вычислительная система МКОД, в отличие от традиционных, имеет в своем составе специализированный процессор обработки структур (СП), который обеспечивает аппаратную поддержку всех операций над структурами данных, их хранение в локальной памяти, выдачу информации в центральный процессор (ЦП) для ее последующей обработки. СП может выполнять команды, хранимые в его локальной памяти команд, в то время как структуры данных хранятся в независимой локальной памяти данных. Прямой доступ со стороны ЦП в локальную память данных СП невозможен.

Все необходимые для ЦП данные выдаются СП в соответствии с заложенной в его память программой обработки структуры. Общие принципы функционирования системы МКОД более подробно изложены в [1–4].

Функции процессора обработки структур. СП выполняет ряд функций ассоциативной обработки, которые в других типах вычислительных структур, таких как ЭВМ с одним потоком команд и одним потоком данных (ОКОД) или системы с многими потоками команд и многими потоками данных (МКМД), выполняются на ЦП кодом вычислительной программы или операционной системы. Основные функции СП:

• ***Реализация операций над структурами данных в соответствии с ключами.*** Ключ является основой структурной части информации и позволяет идентифицировать данные. В исследовании использован вариант СП с размером ключа 32 бит и аналогичным полем данных. Реализованы следующие операции: добавление, удаление, поиск, поиск минимума и максимума, мощность, удаление структуры, пересечение, дополнение, объединение структур, срез структуры, сжатие. Результаты выполнения команд передаются в ЦП для последующей обработки вычислительным алгоритмом.

• ***Управление памятью при хранении информации структур данных.*** В универсальных ЭВМ управление памяти относится к функциям операционной системы и является одной из важнейших задач. Менеджеры памяти операционных систем, в свою очередь, строятся на основе структур данных: красно-черных деревьев, очередей и пр. В системе МКОД выделение и освобождение памяти, используемой при хранении структур данных, выполняются аппаратно при помощи СП. Помимо этого, СП обеспечивает хранение одновременно нескольких структур данных в своей локальной памяти, что позволяет выполнять такие операции, как объединение, пересечение и дополнение. Память для хранения структур данных реализована в виде модуля памяти DDR или DDR2 SDRAM емкостью до 8 Гб.

• ***Исполнение управляющих программ обработки структур данных.*** СП выполняет код, хранимый в локальной памяти команд в виде программ, которые составлены таким образом, чтобы результаты их исполнения соответствовали ходу вычислительного процесса в ЦП. Если, например, к некоторому моменту времени в основной вычислительной программе требуется получить данные, соответствующие минимальному значению ключа, СП заблаговременно выполняет команду поиска минимума и направляет данные в ЦП. В этом случае достигается наибольшая параллельность при выполнении вычислительного алгоритма. В случае зависимых данных, когда результат обработки одной команды влияет на работу следую-

щих, ускорение может быть достигнуто благодаря высокому внутреннему параллелизму СП.

• ***Синхронизация с вычислительным процессом в ЦП.*** СП функционирует под управлением собственной программы, выбираемой им из локальной памяти команд. При выполнении ветвлений в программе ЦП требуется обеспечение синхронизации кода в обоих вычислительных узлах: СП и ЦП. Эта синхронизация выполняется на основе передачи специальных команд перехода, реализующих механизм сложных событий. Во втором случае СП должен ожидать от ЦП передачу информации для выполнения некоторых операций, таких как *Поиск*, *Добавление*, *Удаление*. В этом варианте используется примитив синхронизации «порт завершения ввода-вывода», представляющий собой аппаратную очередь. В третьем варианте взаимодействия ЦП ожидает результат выполнения операции от СП (например, *Поиска*), для чего также используется примитив «порт завершения ввода-вывода».

Принципы хранения и организации доступа к структурам в СП. Информация представляется в памяти данных СП в виде структуры *B*+ дерева. Отличительной особенностью этого вида деревьев является то, что данные хранятся исключительно на нижнем уровне, а соответствующие им ключи организованы в виде непересекающихся множеств. В отличие от программной реализации, при аппаратной обработке *B*+ дерева его высота является постоянной и соответствует максимуму для выделенного объема памяти. Это связано с необходимостью реализации фиксированного количества аппаратных блоков для обработки уровней дерева и хранения нескольких структур разных размеров в единой локальной памяти. На всех уровнях, кроме нижнего (уровень n–1), *B*+ дерево хранит границы подмножеств ключей [Low, High], по которым можно определить, какая именно вершина соответствует искомому значению ключа. Состав и алгоритмы обработки множеств ключей для уровней 0…n–2 и блоков данных на уровне n–1 существенно отличаются (рис. 1). Обработку множеств ключей для уровней 0…n–2 обеспечивает *Каталог* СП. Данные уровня n–1 обрабатываются в *Операционном буфере* [4]. Все основные операции выполняются в *B*+ дереве за $O(\log_b n)$ операций, где b — количество вершин на одном уровне дерева. Аппаратная реализация *B*+ деревьев в СП отличается от классической программной реализации [5, 6] в части алгоритмов таких операций, как *Добавление* и *Удаление*.

Каждый уровень дерева, кроме нижнего, реализован в СП в виде набора взаимосвязанных вершин, осуществляющих хранение и обработку информации о структуре дерева, которая загружается по мере необходимости в *Каталог* СП. Для ускоренного поиска и модифика-

ции информации в вершинах *Каталога* они объединены в независимые списки, которые автоматически модифицируются по мере увеличения или сокращения размеров структур. Максимальное количество структур, хранимых в СП, соответствует количеству блоков на верхнем уровне дерева (в текущей версии — восемь). Структура с номером «0» выполняет специальную функцию: хранение информации о свободных вершинах дерева. В начальный момент времени эта структура занимает всю локальную память данных и становится пустой в случае полного заполнения памяти информацией. Инициализация структуры 0, включающая указание всех атрибутов и адресов локальной памяти данных, выполняется автоматически за один такт работы СП.

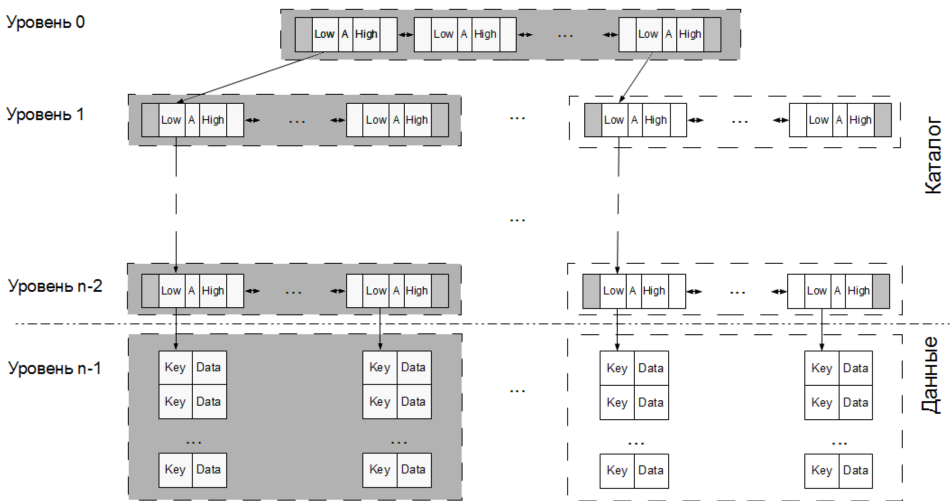

Рис. 1. Аппаратная реализация *B*+ дерева

На нижнем уровне *B*+ дерева ключи и информация хранятся в виде блоков данных, которые содержат в упорядоченном виде от одной до шести пар «ключ-значение». Для осуществления основных операций со структурами все блоки, соответствующие вершинам *Каталога* n–1 уровня, загружаются из памяти данных в *Операционный буфер* СП [4]. Все загруженные в данный момент вершины каталога для всех уровней дерева, а также заполненный *Операционный буфер* представляют собой трассу дерева (выделена на рис. 1 серым цветом), обращение к которой возможно за наименьшее время, так как при этом не требуется доступ в локальную память данных СП. При выполнении операции *Поиска* происходит восходящий поиск информации в трассе до тех пор, пока не будет найден искомый путь, либо не будет достигнут уровень 0. После этого выполняется нисходящее движение в дереве до уровня n–1. Использование загруженной

в СП трассы позволяет существенно ускорить доступ к соседним частям дерева.

В каждой вершине *Каталога* хранится следующая информация:

• параметры Low и High — нижняя и верхняя границы ключей в поддереве. Эти параметры используются для определения вершины *Каталога*, в котором может находиться искомый ключ;

• атрибуты A вершины *Каталога*: номер структуры; номера предшествующей и последующей вершин; адрес дочернего набора *Каталога* в локальной памяти данных СП; флаг переполнения вершины; некоторые дополнительные атрибуты, ускоряющие обработку.

Реализация основных команд. Рассмотрим реализацию команды *Добавление* ADD(structure, key, data), где *structure* — номер структуры, *key* — ключ, *data* — информационная часть. Команда состоит в поиске вершины, соответствующей заданному значению ключа и продвижению к нижнему уровню *Каталога*. Если ключ *key* выходит за границы раздела *Каталога* [Low, High], выполняется подъем в дереве, после чего операция повторяется. Если же найдена вершина с соответствующими *key* границами (Low ≤ key ≤ High), то происходит модификация информации в *Каталоге* и начинается нисходящее движение вплоть до уровня n–1. Для нижнего уровня n–1 выполняется вставка в блок данных в *Операционном буфере*. Алгоритм работы операции *Добавление* представлен ниже:

НАЧАЛО:

    Направление поиска — вверх: $d = -1$

ДОБАВЛЕНИЕ:

    Поиск вершины на текущем уровне t

ЕСЛИ (*Поиск* успешен) ИЛИ ($t == 0$) ИЛИ ($d == +1$)

Направление поиска — вниз: $d = +1$

Изменить *Каталог*

        ЕСЛИ (*Каталог* переполнен)

            ЕСЛИ ($t \neq 0$)

                Деление вершины

            ИНАЧЕ

                Ошибка: Структура заполнена

                КОНЕЦ

            ВСЕ ЕСЛИ

        ВСЕ ЕСЛИ

        ЕСЛИ ($t == n$–1)

            Изменить *Операционный буфер*

            КОНЕЦ

        ИНАЧЕ

            $t = t + d$

Переход к **ДОБАВЛЕНИЕ**
ВСЕ ЕСЛИ
ИНАЧЕ
$t = t + d$
Переход к **ДОБАВЛЕНИЕ**
ВСЕ ЕСЛИ
КОНЕЦ.

Операция *Деление* выполняется в случае переполнения *Каталога*. Данная операция заключается в выделении для структуры новой вершины на вышестоящем уровне и переносе в нее части поддеревьев текущего уровня. На рис. 2 представлен пример выполнения операции *Деление* вершины для *B*+ дерева кратности 4. В примере необходимо произвести добавление в вершину 1 на уровне *k*–1, далее — добавление нового элемента, ключ которого находится в интервале между границей High вершины 5 и Low вершины 6. Однако *Каталог* на уровне *k* оказывается переполнен, и добавление в вершину 5 или 6 невозможно. В этом случае на уровне *k*–1 выполняется добавление новой вершины номер 3 в структуру 1. Вершина 3 занимает место между вершинами 1 и 2 и удаляется из структуры 0. На уровне *k* происходит перенос вершины 8 из поддерева вершины 1 в поддерево вершины 3. Вершина 9 из структуры 0 переносится из поддерева вершины 3 в поддерево вершины 1 и занимает позицию между вершинами 5 и 6.

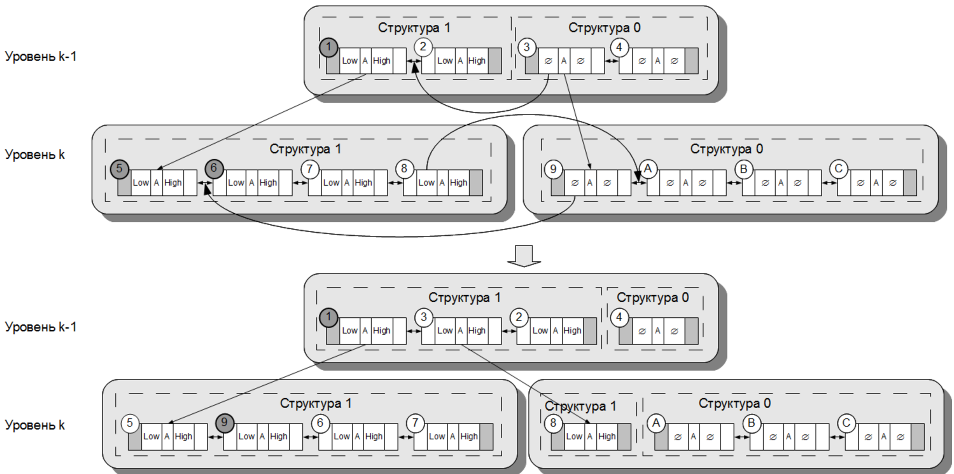

Рис. 2. Операция *Деление* вершины *Каталога*

Таким образом, уровень *k Каталога* освобождается и становится возможным произвести *Добавление* в него новой информации в вершину 9. В случае, когда на вышестоящем уровне *k*–1 отсутствуют пу-

стые вершины (структура 0 пуста), необходимо выполнить *Деление* на уровне k–2 и после этого повторить *Деление* на уровне k–1. При необходимости подъем повторяется до уровня 0. Операция *Деление* в худшем случае требует $O(\log_b n)$ тактов и не требует перезаписи ключей на нижнем уровне.

В алгоритме управления СП также учтен случай, когда локальная память данных будет полностью заполнена информацией и операция *Деление* станет невозможной. Когда локальная память данных заполнена не на 100 %, происходит частичная или полная дефрагментация структуры дерева. В результате дерево перестраивается и освобождается некоторое количество вершин, что делает *Деление* возможным. Если же пустых поддеревьев и в этом случае не образуется, происходит цепочечный сдвиг элементов. Такое происходит только при существенном заполнении локальной памяти данных СП. Например, когда из заполненной памяти удален минимальный элемент структуры и происходит добавление элемента с ключом, бóльшим максимального. Проведенное тестирование СП показало работоспособность данного алгоритма управления при заполнении памяти СП до 100 %.

Операция *Удаление* в текущем варианте реализации СП, в отличие от программных вариантов, не связана с дефрагментацией структуры. Дефрагментация в программной реализации *B*+ дерева служит для сохранения компактности представления структуры в оперативной памяти и выполняется с помощью операции *Слияние*, при которой происходит объединение информации двух вершин в одну, в результате чего вторая вершина освобождается. Команда *Удаление* в СП сводится лишь к выполнению операции *Поиск*, удалению информации из *Операционного буфера* и последующей модификации *Каталога* без дефрагментации и *Слияния*. Такое упрощение операции связано с тем, что, вне зависимости от размера структур, СП хранит их в виде деревьев постоянной высоты. СП самостоятельно управляет выделением свободных блоков и выполняет дефрагментацию по необходимости во время операции *Деление* вершин. Предполагается также, что при паузах между выполнениями основных команд СП будет проводить частичную дефрагментацию структур самостоятельно, для чего в СП предусмотрена команда *Сжатие*. Таким образом, дефрагментация производится только тогда, когда это необходимо для размещения новых элементов, или во время вынужденных простоев СП.

Операция *Поиск* реализована в СП аналогично программным вариантам данного алгоритма в *B*+ дереве [5, 6] и заключается в поиске вершины *Каталога*, удовлетворяющей условию (Low ≤ key ≤ High). На нижнем уровне n–1 в *Операционном буфере* выполняется поиск

полностью совпадающего ключа. Ускорение относительно программной реализации на микропроцессоре достигается благодаря тому, что в СП заложен больший параллелизм при поиске, а также реализован механизм хранения трассы дерева. В программной же реализации ускоренное обращение к ранее загруженной части дерева возможно только в том случае, если эта информация осталась в кэш-памяти. Операция *Поиск* в худшем случае требует $O(\log_b n)$ тактов.

Экспериментальное исследование временной сложности основных операций. Экспериментальные исследования временной сложности операций *Добавление*, *Поиск* и *Удаление* проводились при помощи измерения количества тактов, затраченных на выполнение определенного количества итераций. В первом эксперименте был получен график зависимости количества тактов, необходимых для создания различных структур при выполнении операции *Добавление* случайных ключей, от количества элементов в структуре (рис. 3). Во втором и третьем экспериментах были получены графики зависимостей количества тактов, необходимых для выполнения 10^3 операций *Удаления* (рис. 4) и *Поиска* (рис. 5) случайных ключей для структур с различным количеством элементов. Замер количества тактов вместо измерения времени проведения экспериментов дает более четкое представление об эффективности аппаратного решения, так как уменьшает влияние технологической разницы в реализации тестовых платформ.

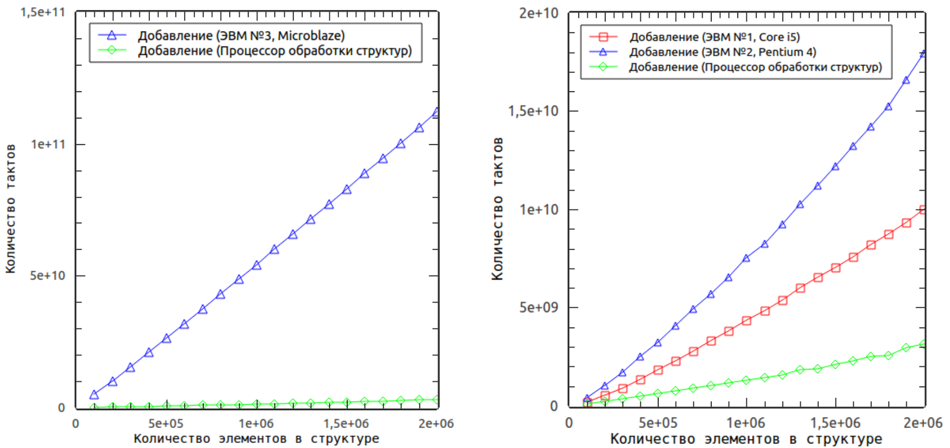

Рис. 3. Экспериментальное исследование временной сложности операции *Добавление*

В экспериментах использовалась реализация СП со следующими параметрами: разрядность ключей и данных — 32 бит; максимальное количество ключей в структуре — $2{\cdot}10^6$; количество уровней *B*+ де-

рева — 8; количество вершин на одном уровне *Каталога* — 8; локальная память — DDR 256 Мб; ширина шины памяти — 64 бит; частота шины памяти — 100 МГц; частота СП — 100 МГц, ПЛИС FPGA Virtex II Pro.

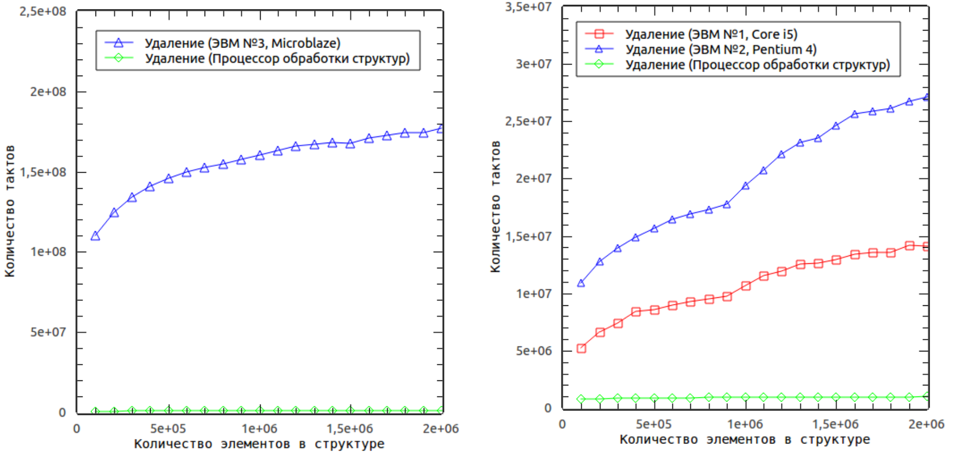

Рис. 4. Экспериментальное исследование временной сложности операции *Удаление*

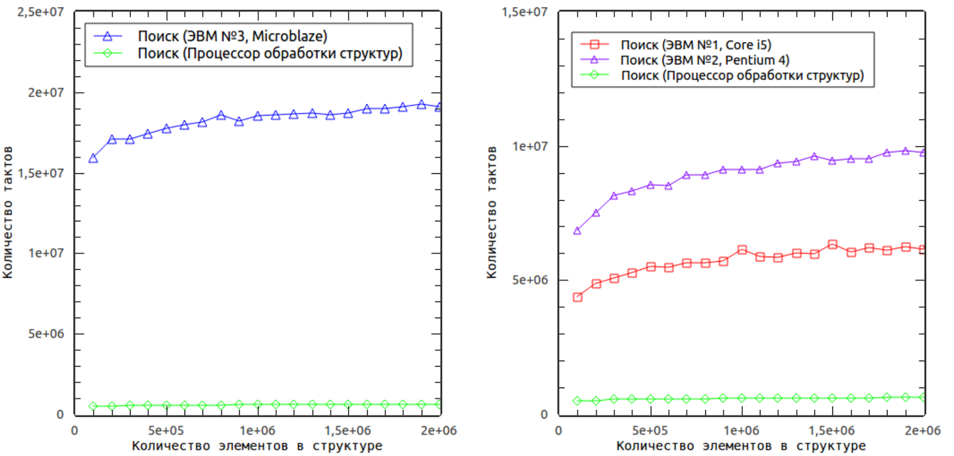

Рис. 5. Экспериментальное исследование временной сложности операции *Поиск*

С целью выявления эффективности алгоритмов управления СП была разработана программная реализация *B+* дерева и проведены эксперименты на трех ЭВМ с различной конфигурацией. Для сравнения результатов измерений программной и аппаратной реализации потребовалось обеспечить схожие условия проведения экспериментов. В частности, объем доступной оперативной памяти на ЭВМ № 1

существенно больше объема локальной памяти СП (3,5 Гб, DDR3 в сравнении с 256 МБ DDR). Другие параметры экспериментальной ЭВМ № 1: количество процессорных ядер — 4, Core i5; частота процессоров — 2530 МГц; операционная система — Ubuntu Linux 12.04 i386; компилятор — c99; локальная память — DDR3 4 Гб; ширина шины памяти — 64 бит; частота шины памяти — 1333 МГц. Параметры экспериментальной ЭВМ № 2: количество процессорных ядер — 1, Pentium 4; частота процессоров — 1500 МГц; операционная система — Ubuntu Linux 10.04 i386; компилятор — c99; локальная память — SDRAM 256 МБ; ширина шины памяти — 64 бит; частота шины памяти — 133 МГц.

ЭВМ № 3 реализована на основе микропроцессорного ядра Microblaze на аппаратной платформе ПЛИС FPGA Virtex II Pro. Параметры экспериментальной ЭВМ № 3: количество процессорных ядер — 1, Microblaze v4.00a; частота процессора — 100 МГц; операционная система — отсутствует; компилятор — mb-gcc; локальная память — DDR SDRAM 256 Мб; ширина шины памяти — 64 бит; частота шины памяти — 100 МГц.

Так как использование полного объема локальной памяти при размещении структуры приводит к большому количеству операций дефрагментации, результаты экспериментов в этом случае искажаются. Поэтому все эксперименты проводились при объемах структур, заведомо меньших объема локальной памяти СП (максимальное допустимое количество элементов структуры в локальной памяти для тестового варианта СП: $6{\cdot}8{\cdot}8{\cdot}8{\cdot}8{\cdot}8{\cdot}8{\cdot}8 = 12{,}6{\cdot}10^6$ пар «ключ-значение»). Измерение количества тактов работы программного алгоритма проводилось с помощью счетчика тактов, получаемого командой RDTSC. Для учета дополнительных программных издержек из общего количества тактов, потраченных для проведения программного эксперимента, было вычтено количество тактов, потраченное на генерацию случайных значений.

Проведенные эксперименты показали, что временная сложность аппаратной реализации операции *Добавление* для СП в 3,2 раза ниже сложности программной реализации на многоядерной платформе ЭВМ № 1 (см. рис. 3). Для одноядерных платформ результаты сравнения количества тактов с аппаратной реализацией на основе СП следующие: для ЭВМ № 2 потребовалось в 5,7 раза больше тактов; для ЭВМ № 3 потребовалось в 42,7 раз больше тактов.

Следует отметить, что в программном алгоритме операции *Добавление* используется функция выделения памяти malloc(), которая для ЭВМ № 1 и № 2 выполняет обращение к менеджеру памяти операционной системы, функционирующему независимо от тестовой программы. В связи с этим в полученных результатах для ЭВМ № 1 и

№ 2 не учитываются влияние параллельного выполнения кода менеджера памяти на разных ядрах и работа менеджера памяти до и после кода эксперимента. Эксперимент на ЭВМ № 2 показал, что при запуске программного эксперимента и ОС на одном ядре временная сложность возрастает приблизительно в два раза.

Полный учет времени выделения и освобождения памяти выполнен при программной реализации эксперимента на ЭВМ № 3, функционирующей в однозадачном режиме без разделения времени.

Для операции *Удаление* временная сложность аппаратного решения на основе СП в 11,8 раза ниже временной сложности программной реализации на многоядерной платформе № 1, в 22,2 раза ниже ЭВМ № 2 и в 164,4 раза ниже ЭВМ № 3. Временная сложность аппаратного выполнения операции *Поиск* в 9,8 раза ниже временной сложности программной реализации на ЭВМ № 1, в 15,3 раза ниже ЭВМ № 2 и в 31,4 раза ниже ЭВМ № 3.

Для сравнения аппаратной сложности экспериментальных платформ был использован отчет о задействованных в аппаратном проекте на ПЛИС FPGA Virtex II Pro ресурсах кристалла, использованных при реализации СП (см. таблицу). Аппаратная сложность проекта составила 840 тыс. эквивалентных вентилей. Микропроцессоры, использованные в эксперименте, имеют следующую аппаратную сложность: ЭВМ № 1 Core i5 ~ 730 млн вентилей (в 869 раз больше); ЭВМ № 2 Pentium 4 ~ 42 млн вентилей (в 50 раз больше). Аппаратная сложность встраиваемой системы на основе микропроцессорного ядра Microblaze (процессор, шина, контроллер памяти) составляет около 40 тыс. эквивалентных вентилей, что в 21 раз меньше аппаратной сложности СП.

Сравнение временной и аппаратной сложностей СП и микропрограммных ЭВМ

ЭВМ	Операция, такты			Аппаратная сложность, вентили
	Добавление	Удаление	Поиск	
На основе СП	A*	B*	C*	D**
№ 1 (Core i5, 4 Гб RAM)	3,2·A	11,8·B	9,8·C	869·D
№ 2 (Pentium 4, 256 Мб RAM)	5,7·A	22,2·B	15,3·C	42·D
№ 3 (Microblaze, 256 Мб RAM)	42,7·A	164,4·B	31,4·C	D/21

* A, B, C — усл. ед. измерения временной сложности такта.
* D — усл. ед. измерения аппаратной сложности, вентили.

Выводы:

• Временная сложность выполнения основных команд в процессоре обработки структур существенно ниже временной сложности выполнения команд при программной реализации аналогичных алгоритмов на универсальных микропроцессорах.

• Временная сложность выполнения основных операций, выраженная в секундах, для СП оказывается в 1,5–8 раз выше, чем в ЭВМ на основе ×86 микропроцессоров, что объясняется высокой разницей в тактовых частотах СБИС и ПЛИС.

• Анализ полученных результатов позволяет сделать вывод о высокой архитектурной эффективности предложенного аппаратного устройства — процессора обработки структур. В случае реализации данного устройства на основе ПЛИС целесообразным можно считать его применение в специализированных встраиваемых системах, где требуется реализовать высокую скорость доступа к структурам данных при невысокой программной и аппаратной сложности системы (например, сетевых устройствах, устройствах управления роботами, летательными аппаратами и пр.).

• Высокая архитектурная эффективность принципов обработки информации, заложенных в СП, позволяет считать целесообразной дальнейшую реализацию компонентов MISD-системы на основе СБИС.

ЛИТЕРАТУРА

[1] Попов А.Ю. Электронная вычислительная машина с аппаратной поддержкой операций над структурами данных. Аэрокосмические технологии, 2009. Т. 1: *Тр. Второй Междунар. научно-техн. конф., посвященной 95-летию со дня рождения академика В.Н. Челомея.* ОАО «ВПК «НПО машиностроения», МГТУ им. Н.Э. Баумана, Москва, 2012, с. 296–301.

[2] Попов А.Ю. Электронная вычислительная машина с многими потоками команд и одним потоком данных. Пат. № 71016, Российская Федерация, 2008, бюл. № 5, 1 с.

[3] Попов А.Ю. Применение вычислительных систем с многими потоками команд и одним потоком данных для решения задач оптимизации. *Инженерный журнал: наука и инновации*, 2012, вып. 1. URL: http://engjournal.ru/catalog/it/hidden/80.html

[4] Попов А.Ю. Реализация электронной вычислительной машины с аппаратной поддержкой операций над структурами данных. *Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение,* спецвыпуск «Информационные технологии и компьютерные системы», 2011, с. 83–87.

[5] Кнут Д.Э. *Искусство программирования. Т. 3: Сортировка и поиск.* 2-е изд. Москва, Вильямс, 2000, 832 с.

[6] Кормен Т., Лейзерсон Ч., Ривест Р. *Алгоритмы: построение и анализ.* Москва, МЦНМО, 2000, 960 с.

Статья поступила в редакцию 28.06.2013

Ссылку на эту статью просим оформлять следующим образом:

Попов А.Ю. Исследование производительности процессора обработки структур в системе с многими потоками команд и одним потоком данных. *Инженерный журнал: наука и инновации,* 2013, вып. 11. URL: http://engjournal.ru/catalog/it/hidden/1048.html

Попов Алексей Юрьевич родился в 1974 г., окончил МГТУ им. Н.Э. Баумана в 1997 г. Канд. техн. наук, доцент кафедры «Компьютерные системы и сети». Автор 28 научных работ в области разработки программно-аппаратных средств вычислительной техники. e-mail: alexpopov@bmstu.ru