УДК 681.324

Ю.С. ЯКОВЛЕВ, Е.В. ЕЛИСЕЕВА

ОСНОВНЫЕ ЗАДАЧИ И МЕТОДЫ РЕАЛИЗАЦИИ ФУНКЦИЙ УПРАВЛЕНИЯ ПАМЯТЬЮ В PIM-СИСТЕМАХ

Abstract: *Tasks and methods of implementation of functions of control by memory in PIM-systems are considered – creation and translation of the address information, an optimal memory distribution, allocation of data, etc. Examples of two various approaches are led to construction of controllers of a memory management. Sentences on implementation of an effective way management by the memory grounded on features of construction and operation of PIM-systems are formulated.*
Key words: *PIM-systems, addressing, distribution of memory, allocation of date, ways and controllers of management by the memory.*

Анотація: *Розглянуто завдання і методи реалізації функцій керування пам'яттю в PIM-системах – формування і трансляція адресної інформації, оптимальний розподіл пам'яті, розміщення даних та ін. Наведені приклади двох різних підходів до побудови контролерів керування пам'яттю. Сформульовані пропозиції щодо реалізації ефективного способу керування пам'яттю, засновані на особливостях побудови і функціонування PIM-систем.*
Ключові слова: *PIM-системи, адресування пам'яті, розподіл пам'яті, розміщення даних, способи і контролери управління пам'яттю.*

Аннотация: *Рассмотрены задачи и методы реализации функций управления памятью в PIM-системах – формирование и трансляция адресной информации, оптимальное распределение памяти, размещение данных и др. Приведены примеры двух различных подходов к построению контроллеров управления памятью. Сформулированы предложения по реализации эффективного способа управления памятью, основанные на особенностях построения и функционирования PIM-систем.*
Ключевые слова: *PIM-системы, адресация, распределение памяти, размещение данных, способы и контроллеры управления памятью.*

1. Введение

PIM-системы (Processor-in-Memory) реализуются на базе больших интегральных схем (БИС) памяти путем разделения всего массива памяти, размещенного на кристалле БИС, на банки памяти и привнесения в каждый банк памяти процессорных элементов, которые берут на себя часть вычислений, выполняемых в компьютерных системах (КС) с классической архитектурой ведущим процессором (хост-процессором) [1]. Это, как правило, интенсивные вычисления широко распараллеленных процессов с массовым обращением к памяти за данными. Такая память с элементами обработки также может выполнять свои классические функции: запись, хранение и выдачу информации, и поэтому может быть использована в качестве основной памяти хост-процессора с классическими функциями; дополнительной памяти, расширяющей емкость основной памяти хост-процессора; эффективного средства обработки, используемого в виде приставки – ускорителя вычислений хост-процессора, а также в обособленном виде как основы для построения высокопроизводительных распределенных КС. PIM-системы могут быть реализованы как на основе БИС динамической памяти (DRAM), так и статической (SRAM). В первом случае при реализации большой емкости банков памяти часто применяют КЭШ-память для согласования скорости работы процессорного узла со скоростью работы приписанного ему банка памяти, образуя тем самым иерархическую систему памяти [2].

Особенность управления памятью PIM-систем состоит в том, что функции управления памятью в системе разделены между хост-процессором и процессорными ядрами (ПЯ) банков памяти. На хост-процессоре функциональные возможности стандартной операционной системы расширены, чтобы поддерживать работу PIM. На каждом PIM-процессоре во время выполнения

ISSN 1028-9763. Математичні машини і системи, 2008, № 2

имеется своя небольшая программа управления (далее её будем определять как стержень операционной системы (ОС)), которая всегда является резидентом. Основное назначение стержня ОС во время выполнения: управление поступающими пакетами и их коммутацией между PIM; программное вмешательство в ответ на прерывания на процессоре PIM. В дополнение к этим функциям PIM стержень ОС также должен взаимодействовать с хостом при выполнении операций системного уровня, типа загрузки программы PIM и данных, управления памятью PIM-видимых сегментов и коммутаторов контекста PIM между различными пользовательскими программами.

Как правило, для реализации процесса управления памятью используется контроллер памяти, размещенный на той же подложке БИС, что и другие компоненты системы.

Расширенные функциональные возможности и способы применения памяти PIM-систем накладывают определенный отпечаток на решение задач управления памятью, которые существенно отличаются от решения соответствующих задач управления памятью КС с классической архитектурой. К таким задачам, прежде всего, следует отнести:

– формирование и передачу адресной информации к любой ячейке массива памяти с учетом разбиения массива памяти на банки и подбанки (проблема адресации);

– оптимизацию разделения (распределения) всего массива памяти в зависимости от разделения решаемой задачи между хост-процессором и ядрами процессоров, приписанных к банкам и подбанкам памяти, а также в зависимости от емкости памяти всего массива, размещенного на кристалле, количества узлов, типа реализуемых алгоритмов и других параметров КС (проблема разделения памяти);

– оптимизацию размещения данных в виртуальных и физических пространствах памяти по разделенным областям (проблема размещения данных);

– динамическую корректировку информации (замещение страниц, подкачка информации в нужную область памяти и др.);

– сборку «мусора» и дефрагментацию памяти и др.

Далее кратко рассмотрим особенности решения некоторых из перечисленных выше задач управления памятью на примерах практической реализации описанных в литературе конкретных PIM-систем.

2. Особенности решения задачи формирования и передачи адресной информации

Возможны различные подходы к формированию и передаче адресной информации PIM-систем в зависимости от доминирующих функций, выполняемых системой памяти, и соответственно её архитектурно-структурной организации. Один из подходов описан в [3] применительно к PIM-системе DIVA, где память выполняет не только классические функции (запись, хранение, чтение информации), но и функции среды обработки, являясь в то же время единственной памятью для хост-процессора.

Чтобы интерпретировать адреса в коде PIM и данных, микропроцессор PIM должен поддерживать механизм трансляции, для упрощения которого память классифицируется, согласно её использованию в системе, на глобальную, локальную и так называемую “немую” память. Глобальная память образует адресное пространство, распределенное по всем PIM-узлам,

видимым приложениям, которые выполняются на хосте и узлах PIM. Локальная память является областью памяти узла, используемой подпрограммами PIM. Определенные функции хост-операционной системы, такие как инициализация и контекстное управление, также иногда обращаются к этой памяти при соответствующих соглашениях совместно используемых данных. “Немая” память является областью памяти узла, распределенной как обычные страницы в виртуальном пространстве хост-приложения и неиспользованные в процессе обработки узла PIM. Она управляется исключительно хост-операционной системой стандартными способами с адресной трансляцией, поддержанной аппаратным обеспечением управления памятью хост-процессора.

Физическая память на каждом чипе PIM разделена в соответствии с этими тремя направлениями её использования. DRAM-память, связанная с глобальным адресным пространством, физически распределяется на все узлы PIM, использованные при вычислении. Хост-процессор может обратиться к памяти PIM, используя ее шину памяти. Чтобы избежать режима насыщения этой шины, связь PIM-to-PIM осуществляется через широкополосную сеть между PIM-чипами с помощью пакетов. Структурно пакет имеет рабочее поле и заголовок, который включает адрес памяти целевого объекта, выраженного в адресном пространстве приложения, и команду, идентифицирующую функцию, которая будет выполнена узлом по прибытии пакета.

При обращении к памяти за данными соответствующий адресный узел системы должен, прежде всего, определить, располагается ли адрес в его собственной памяти, и если да, то найти физический адрес. Поэтому каждый узел поддерживает в текущее время трансляцию виртуальных адресов, включая локальную память и часть его глобальной памяти. Чтобы уплотнить информацию трансляции, использованы сегменты, каждый из которых определяется регистрами сегмента, содержащими физический базовый адрес. Сегменты отображаются картой адресов PIM, которая показана в табл. 1.

Таблица 1. Карта адресов PIM-узла

Функциональное назначение регистров сегмента	Виртуальный размер области памяти, Мбайт	Используемая часть памяти, Кбайт
Область операционной системы хоста	1024	Резервная
Глобальные сегменты (разделенные окна данных)	2816	Сотни Кбайт в каждом картографируемом окне
PIM-физическое пространство	128	Нетранслируемая область
Стек ядра	16	Несколько Кбайт
Буфер ядра пакета	16	Несколько сотен байт
Ядро данных	16	Десятки Кбайт
Ядро кода	16	Десятки Кбайт
Стек пользователя	16	Несколько Кбайт
Буфер пакета пользователя	16	Несколько сотен байт
Данные пользователя	16	Сотни Кбайт
Код пользователя	16	Десятки Кбайт

При трансляции отдаленных адресов система определяет размещение отдаленных данных, используя двухступенчатую технологию. Виртуальный адрес данного разделяется аппаратным обеспечением, чтобы определить его “домашний узел", который выполнит операцию непосредственно, если данное является резидентом, или отправит запрос другому узлу, если данное находится в другом месте.

3. Особенности решения задачи распределения памяти

Проблема распределения памяти является весьма сложной и требует выполнения отдельных исследований, так как значительная часть времени (до 30%) центрального процессора при выполнении приложений тратится на динамическое управление информационными объектами и соответственно на распределение и освобождение соответствующих областей памяти [4, 5].

Управление распределением памяти выполняется путем формирования двух запросов менеджеру ресурса системы на обслуживание программного обеспечения: "создание распределенной памяти" и "размещение распределенной памяти" [6]. Служба "создание распределенной памяти" сначала исследует таблицу системного распределения, чтобы определить, было ли выполнено такое распределение, имеется ли достаточно места в системном адресном пространстве, чтобы реализовать распределение необходимого размера, и т.д.

Другой менеджер ресурса системы обслуживания "размещение распределенной памяти" связывает распределение локальной памяти с распределением системной памяти путем записи надлежащей информации в память распределения типа RAM для конкретного узла и активации соответствующих входов RAM-распределения.

При распределении памяти в системе DIVA [3] часть памяти, используемая хостом как “немая” память, управляется операционной системой хоста, используя стандартное распределение, замещение и механизмы свопинга (подкачки). Локальная и глобальная, совместно используемая память PIM управляется при взаимодействии между хостом и PIM. Наиболее важный аспект этого взаимодействия – распределение памяти. Табл. 2 отображает программно реализуемые функции, связанные с распределением памяти, которые выполняются хостом или PIM.

Таблица 2. Программно реализуемые функции распределения памяти

№ п/п	Реализуемые функции	Сущность реализации
1	GlobalMallocToNode (int numBytes, int virtualNode, int segmentName) GlobalMallocToAddress (int numBytes, void *existingObject) Malloc (int numBytes)	Распределение пространства физической памяти для новых объектов (хост или PIM)
2	GlobalUnMap (void *existingObject) GlobalMap (int numBytes,void *existingObject)	Выборка существующих объектов (хост или PIM)
3	GlobalFree free (void *existingObject)	Разрушение существующих объектов (хост или PIM)
4	ReservGlobalSegment (int numBytes, int virtualNote) ReserveLocalHeapSegment ReserveLocalCodeSegment ReserveLocalStackSegment (int numBytes, int virtualNode)	Распределение виртуального адресного пространства

Стандартные функции распределения памяти *Malloc* могут использоваться или на хосте или на PIM, в зависимости от того, где функции выполняются. На хосте функция *Malloc* реализует стандартное распределение “немой” памяти. На PIM она распределяет память локального множества сегментов PIM. Память, полученная от *Malloc*, является приватной для процесса и не используется для обеспечения совместного доступа.

Группа общих данных создается хостом или узлом PIM через функции *GlobalMalloc*, которые выполняют две роли: распределение группы элементов физической памяти и отображение

части (предварительно зарезервированной) глобального виртуального адресного пространства на физическую память.

Чтобы получить хорошие рабочие характеристики, приложение должно совместить выборку данных с обработкой. С этой целью используют две функции распределения памяти. Функция *GlobalMallocToNode* (int numBytes, int virtualNode, int segmentName) распределяет *numBytes* от названного сегмента *segmentName* на виртуальном PIM узле *virtualNode*. Дополнительный *segmentName* аргумент позволяет реализовать распределение в пределах специфического глобального сегмента.

Чтобы расположить два связанных объекта без виртуального PIM-идентификатора, функция *GlobalMallocToAddress* разрешает динамическое распределение объектов к одному и тому же виртуальному PIM-узлу и глобальному сегменту, выше которого постоянно находится другая заданная величина.

Процесс может ограничить использование объекта данных путем вызова любой функции *GlobalUnMap* или *GlobalFree*. Вызов *GlobalUnMap* не отображает объект текущего процесса, а указывает, что его содержание должно сохраниться. Объект будет перемещен к его домашнему узлу, где другие процессы могут впоследствии обратиться к нему путем запроса функции *GlobalMap*. Объект будет разрушен, когда некоторый процесс, хост или PIM запрашивает *GlobalFree*, или когда ограничиваются вычисления. Вызов *GlobalFree* указывает, что его физическая память может быть перераспределена, а её содержание разрушено.

Применение технологии сегментирования требует, чтобы данные в сегменте постоянно находились в непрерывной виртуальной адресации. Виртуальное распределение памяти выполняется хостом, используя функции *Reserve* для глобальных и локальных сегментов. Поскольку виртуальное адресное пространство является весьма большим, необходимо оценить пространственные требования сегмента, особенно начиная с выхода размеров сегмента за пределы того, что было первоначально зарезервировано.

Множество глобальных сегментов могут быть зарезервированы с помощью запросов к функциям *ReserveGlobalSegment*. Каждое глобальное резервирование сегмента создает новый сегмент с уникальным названием, которое впоследствии может использоваться произвольно функциями распределения.

Физическое распределение выполняется при взаимодействии хоста и PIM. На хосте при этом заполняются входы таблицы страниц. На стороне PIM могут быть обновлены регистры сегмента. Такая технология использует хост при распределении памяти, однако разрешает PIM распределять память независимо, если это необходимо для управления на основе указателя или другой динамической структуры в течение PIM-вычислений.

DIVA – программируемая модель отражает глобально адресуемое, распределенное адресное пространство на разделенных данных. Для упрощения модели программирования функции *GlobalMalloc* выполнены с учетом PIM согласованного интерфейса хоста. Обычно эти функции используются, чтобы распределить данные локально резидентских глобальных сегментов PIM, но можно также выполнить распределение и на удаленном PIM-узле, т.е. распределить виртуальные адреса отдаленного узла и локально отобразить объект на глобальных сегментах PIM

и физической памяти. Такое распределение выполняется с тем, чтобы эффективно поддержать обновления отдаленных данных до отправления их к отдаленному узлу. Подобно функциям *GlobalMalloc,* применительно к отдаленному узлу, иногда желательно временно отобразить нерезидентные глобальные данные, чтобы облегчить совместное их использование различными PIM. Функция *GlobalMap* выполняет это отображение к глобальным регистрам сегмента, и *GlobalUnMap* возвращает данные к его домашнему узлу.

4. Особенности решения задачи размещения данных в памяти PIM-систем

Одно из главных препятствий к достижению высокой производительности PIM-систем – это межпроцессорные связи, которые удлиняют общее время выполнения приложения [7, 8]. Поэтому одна из проблем при создании PIM-систем состоит в поиске оптимального способа размещения и перемещения данных в массиве PIM. Прежде чем начнется обработка приложения, оно разбивается на разделы, а его данные распределяются по процессорам.

Принимая во внимание актуальность этой проблемы, целесообразно выделить несколько подходов к размещению данных: один подход основан на использовании так называемых “окон выполнения” [8], другой – на так называемых коллекциях и макросерверах [9, 10], третий – на применении специальной программной системы (например, SAGE) для разбиения инструкций и построения трафика [11].

При использовании “окон выполнения” для решения проблемы размещения данных могут быть применены 2 метода [8]: метод единственного центра (Single-Center Data Scheduling – SCDS) и метод многих центров. Метод единственного центра размещения данных не рассматривает перемещение данных во время выполнения, они остаются на том же самом месте в течение всех шагов выполнения, которые для приложения собраны в одном “окне выполнения“. Если процессор выбран в качестве центра данных и память процессора полностью занята, чтобы сохранить данные, должно быть указано другое место их размещения. Метод многих центров состоит из двух частей. Первая часть – инициализация, которая размещает исходные данные для первого “окна выполнения“. Вторая часть обеспечивает перемещение данных во время выполнения, реализуя размещение каждого данного для различных шагов выполнения.

В [8] рассматриваются два алгоритма планирования (размещения) данных, которые позволяют перемещать данные во время выполнения. Первый назван локально – оптимальным многоцентровым планированием данных (Local-Optimal Multiple-Center Data Scheduling –LOMCDS) и второй – глобально оптимальным многоцентровым планированием данных (Global-Optimal Multiple-Center Data Scheduling – GOMCDS). Первый обеспечивает оптимальное локальное размещение данных относительно каждого индивидуального окна выполнения. При этом все окна выполнения известны. Определяется центр в каждом окне выполнения относительно каждого данного, соответствующий минимальному расстоянию относительно окна выполнения и данных. Во время выполнения приложения данные перемещаются в центры, соответствующие этим окнам выполнения. Когда емкость памяти процессора ограничена, используется соответствующая методика, чтобы найти первый доступный процессор для каждого данного в каждом окне выполнения.

При глобально оптимальном многоцентровом планировании данных рассматриваются все окна выполнения. Определяются центры каждого окна выполнения относительно каждого данного, и создается диаграмма стоимости в виде взвешенного направленного графа без петель (Directed Acyclic Graph – DAG). При этом используется метод самого короткого пути для нахождения оптимального размещения каждого данного в каждом окне выполнения.

Чтобы понизить полную стоимость связи при реализации приложения, данные не должны быть помещены в идентичной позиции для всех окон выполнения. Кроме того, размер окна выполнения может также играть существенную роль в снижении общей стоимости связи. Если окно выполнения слишком мало, стоимость перемещающихся данных между центрами окон может быть большой. Можно группировать соответствующие окна выполнения относительно данных в новые окна, большие по размерам. Помещая данные в центре нового окна выполнения, можно понижать стоимость перемещающихся данных между центрами так же, как и общую стоимостью связи.

Методы размещения данных, основанные на применении коллекций и макросерверов, целесообразно рассматривать на примере решения проблемы распределения и управления данными при создании PIM-системы типа Gilgamesh [9, 10].

Gilgamesh система (Giga Logic Gate Assemblies with Mesh Integration) – это гомогенная масштабируемая архитектура, которая может иметь сотни тысяч узлов и содержит набор чипов типа MIND, связанных системной локальной сетью. Каждый чип MIND содержит множество банков памяти DRAM (динамической оперативной памяти), обрабатывающую логику, внешние интерфейсы ввода-вывода и межчиповые каналы связи. Каждый узел чипа MIND предполагает доступ ко всей строке разрядностью R, содержащей множество G r-разрядных слов $(r \subset R)$ для чтения-записи в память и набор параллельных арифметико-логических устройств, оперирующих одновременно r-разрядными словами и образующими в совокупности результаты для всей R-разрядной строки.

Gilgamesh реализует глобальное виртуальное общедоступное адресное пространство в том смысле, что виртуальные адресации могут быть отображены (с поддержкой аппаратных средств) на физические адреса для всей системы. Однако нелокальный прямой доступ (например, через команду чтения или команду записи) от данного модуля до адреса с физическим размещением в другом модуле прямо не поддерживается аппаратными средствами, а реализуется через механизм коммутации и пакеты, связанные с активными сообщениями. Поэтому латентность нелокального доступа, по крайней мере, на один порядок величины больше, чем латентность локального доступа, учитывая важность локализации для программируемой модели.

Архитектура MIND управляет виртуальным пространством имён, которое разделено среди множества узлов MIND на данном чипе. Поддержка трансляции виртуального адреса к физическому включена как встроенная функция каждого узла и работает с распределенной директивной таблицей так же, как с локальными кэшируемыми отображениями адресов. Таким образом, поддерживается полное виртуальное адресное пространство, общедоступное для всех чипов MIND.

Важная особенность архитектуры MIND – прямая обработка виртуального адресного пространства для распределенной общедоступной памяти. Текущая модель памяти основана на

определенном размере страницы (например, 4 Кбайт) и позволяет размещать страницы на любой узел массива MIND.

Ключом к схеме распределения данных является таблица каталога адресов. Наборы входов, составляющие директивную таблицу для групп виртуальных страниц, назначаются в чипе на основании их физических имен.

Чипы MIND могут быть сгруппированы так, что они находятся в области адреса. Любая страница в таком домене представляется в директивном сегменте таблицы, где указываются все виртуальные страницы, которые ей приписаны, или же какая-либо страница является гостевой страницей, которая не входит в область определенных чипов MIND, но тем не менее сохранена в этой таблице. Предпочтительное размещение страницы – в пределах ее области. Если это удовлетворено, управляющий пакет определяет через прямое частичное отображение, какая группа чипов MIND должна выполняться и затем найдет точное местоположение целевой страницы. Если целевая страница – гостевая страница другого домена, то требуется выполнить второй шаг прежде, чем пакет обратится к предназначенному объекту.

Для структурирования, обозначения и доступа к данным в системе Gilgamesh используются так называемые *коллекции*. Коллекции являются гомогенными или гетерогенными агрегатами, охватывающими широкий диапазон методов. Они включают многомерные, плотные гомогенные массивы, списки структур, разреженных структур данных и наборов. Любая коллекция связана с индексным доменом, который использует уровень микропрограммных средств, названный *уровнем макросервера,* поддерживающим основанное на объектах управление во время выполнения потоков данных и выполняющим динамический контроль балансирования загрузки и местоположения. Макросерверы – это распределенные динамические объекты, формирующие переменные, методы, потоки и внешние интерфейсы (в смысле объектно-ориентированных вычислений), но с расширенными функциональными возможностями для выполнения ориентированных вычислений на PIM-системах. Приложения выполняются как асинхронные сети макросерверов. Сложное приложение может отобразить параллелизм на многих уровнях абстракции.

На уровне макросервера распределение производится отображением в параметризованную абстрактную архитектуру, которая понимается как набор абстрактных узлов, связанных друг с другом через управляющие пакеты, используя сеть связи. При этом какое-либо предположение о топологии связи не делается. Абстрактные узлы отображаются системным программным обеспечением нижнего уровня на основную физическую архитектуру.

Метод распределения данных с помощью специальных программных систем включает распределение задачи по процессорам с различными характеристиками и разделение первоначальной программы на несколько частей (блоков) с целью их параллельного выполнения. В [11] предложена программная система SAGE (Statement-Analysis-Grouping-Evaluation), которая не только реализует эти функции, но и генерирует взвешенный граф (*Weight Partition Dependence Graph* – WPG), определяет вес каждого блока и затем отправляет соответствующие задания процессорам памяти и хост-компьютеру. Укрупненная схема компиляции программы с помощью системы SAGE приведена на рис.1.

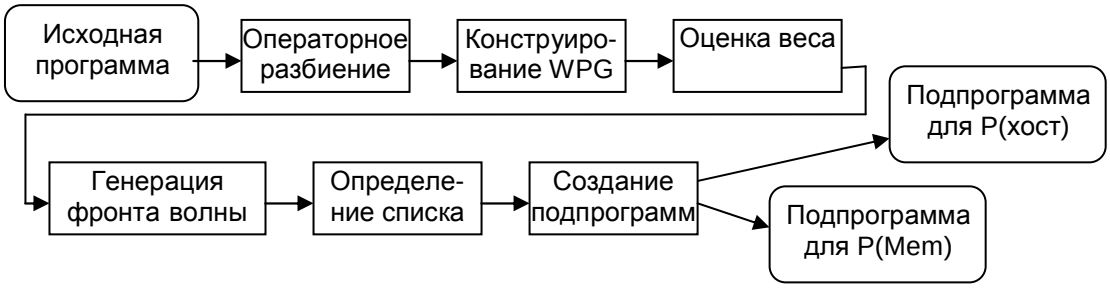

Рис. 1. Укрупненная схема компиляции программы с помощью системы SAGE

SAGE-система ориентирована на решение перечисленных задач применительно к архитектуре интеллектуальной памяти типа FlexRAM.

5. Особенности решения задачи динамической корректировки информации

Для реализации вычислений, требующих доступа к глобальным структурам данных, большим, чем имеющаяся емкость физической памяти, необходимо поддерживать "механизм замещения страницы" виртуальной памяти для PIM-процесса. Если домашний узел возвращает сообщение, отмечающее, что требуемое данное не является резидентным в физической памяти системы, инициализирующее ядро PIM обращается к службе замещения страниц хоста, который связан с дисковой памятью [3]. На рис. 2 показана последовательность шагов замещения страницы хоста и PIM. Хост-операционная система ответственна за создание контекстов для PIM, а также за обновление контекстов в соответствии с большими системными контекстными коммутациями. Чтобы облегчить хост-управление контекстами, хост в течение инициализации создает структуру данных, отображенную в памяти PIM, которую он совместно использует с PIM текущим стержнем.

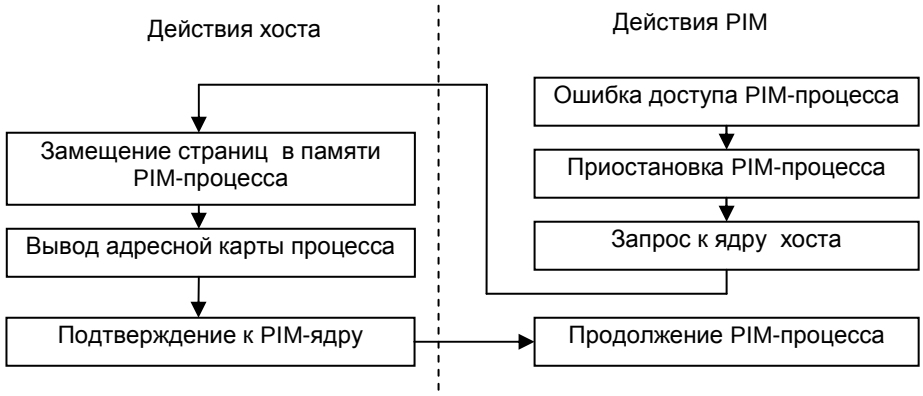

Рис. 2. Последовательность шагов замещения страниц

Табл. 3 отражает содержание контекста. При инициализации пользовательской программы хост выполняет виртуальное распределение сегментов памяти и записывает уровень распределенных виртуальных адресаций в структуре контекстных данных в PIM-стержневом ядерном сегменте.

Таблица 3. Содержание контекста

Контекст	**Содержание**
Отображения локального сегмента	Код, стек, локальная куча
Отображения глобального сегмента	Распределенные данные ОС
Набор скалярных регистров	32 входа шириной 32 бит
Набор регистров широкого слова	32 входа шириной 256 бит
Набор скалярных регистров с плавающей точкой	32 входа шириной 64 бит
Коды условий и др.	Скалярное и широкое слово
Состояние буфера пакета	Состояние сетевого интерфейса

Подкачка (обмен) является другим механизмом для того, чтобы поддержать вычисления с требованиями большой емкости памяти. Многие вычисления могут быть разбиты на фазы (этапы), которые не должны быть одновременно активными. Требуемые пиковые объемы и конфигурации памяти могут быть уменьшены при выгрузке неактивных процессов или низких по приоритетам активных процессов.

Механизм распределенной совместно используемой памяти применяется для простого блочно-ориентированного совместного использования данных приложений, где пропускная способность является более приоритетным показателем, чем задержка.

6. Контроллеры управления памятью PIM-систем

Для управления памятью используют контроллер, который обеспечивает согласованное взаимодействие функциональных узлов, размещенных как на данном чипе, так и на других, связанных с ним чипах, реализуя следующие функции:

– формирование и выдача управляющих сигналов, необходимых для работы памяти при реализации классических функций (запись, хранение, чтение информации), а также функций обработки данных с помощью процессорных элементов (процессорных ядер), размещенных в банках памяти;

– управление обменом данными между чипом системы памяти и другими аналогичными чипами или устройствами ввода-вывода;

– определение направленности запросов к памяти от процессоров или процессорных ядер (к одному или нескольким банкам DRAM на данном чипе или к внешней памяти) и передача этих запросов к банкам DRAM на данном чипе или к внешней памяти через интерфейс внешней памяти;

– устранение несогласованности в состояниях памяти, например, путем запуска программного протокола когерентности КЭШ;

– поддержка обмена данными между процессорными ядрами и другими процессорами вне кристалла и связанными с ними блоками памяти или устройствами ввода-вывода через интерфейс ввода-вывода.

При обмене данными или совместном использовании устройств данного чипа с подключенными к нему устройствами контроллер памяти использует два механизма связи: один – для поддержки локальной коммутации и запросов ввода-вывода и второй – для поддержки отдаленной коммутации и запросов ввода-вывода. На контроллер памяти также могут быть возложены функции управления сообщением и маршрутизацией ввода-вывода. Для этого он должен содержать таблицы маршрутизации или команд маршрутизации и должен быть запрограммирован таким образом, чтобы обрабатывать эти функции.

Архитектура контроллера памяти PIM-системы типа Computational RAM (CRAM)

Основное назначение CRAM-контроллера – обеспечение эффективного управления выполнением прикладных программ в памяти со стороны внешнего интерфейса хост-процессора [12]. При этом хост-процессор выдает команду высокого уровня на контроллер памяти, который генерирует микрокоманды для процессорных ядер (PE), подключенных к банкам памяти. Информация управления, данные и адрес передаются от контроллера ко всем PE

широковещательно в стиле SIMD. На рис. 3 показана архитектура CRAM-контроллера для управления памятью PIM-системы.

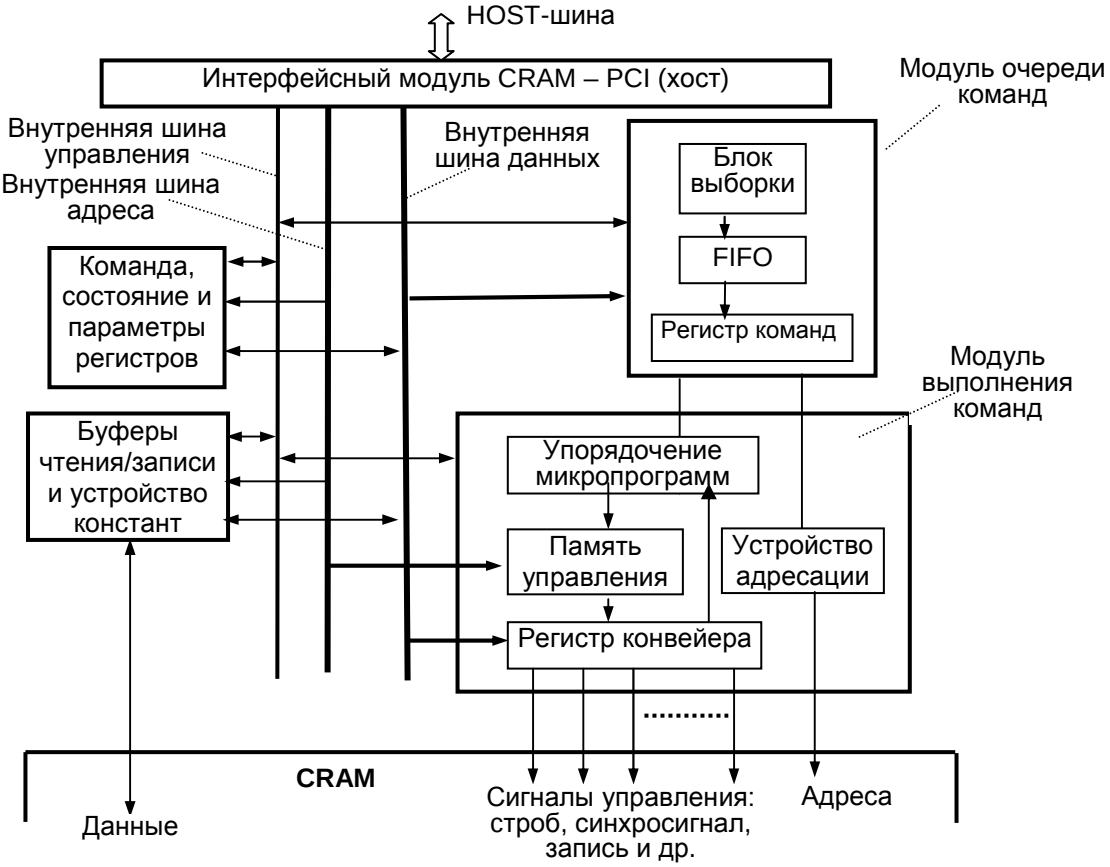

Рис. 3. Архитектура CRAM-контроллера

Интерфейсный модуль CRAM – PCI выдает для устройства управления CRAM-контроллера сигнал управления чтением – записью и передает ему состояние логики управления шины ISA. Для связи с персональным компьютером (PC) в принципе могут быть использованы три типа шин: шина PCI, которая является стандартом для большинства PC, шина ISA и шина EISA. При применении в контроллере программируемых устройств типа Xilinx FPGA используется интерфейсная шина ISA.

Все команды CRAM от ведущего процессора сначала загружаются в модуль очереди команды (МОК). При этом выполняемые CRAM-контроллером команды в МОК полностью независимы от хост-процессора.

Модуль выполнения команд загружает команду в регистр команд и выдает сигналы управления, коды адреса и данных CRAM. Все команды CRAM 32-разрядной ширины и имеют однородный RISC-подобный формат (рис. 4).

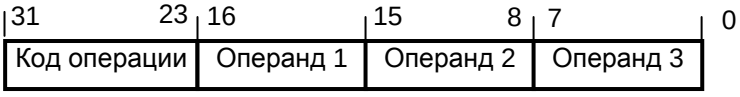

Рис. 4. Формат команды

Слово микрокоманды представляет собой 32-разрядное CRAM – слово, формат которого показан на рис. 5.

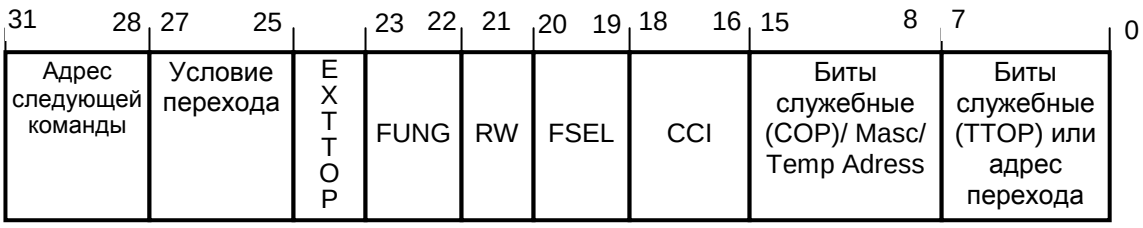

Рис. 5. Слово микрокоманды

EXTTOP определяет значение любого операнда слова команды или битов [7:0] текущей микрокоманды.

FUNC вместе с RW используется для указания пустой команды, операции PE или доступа к внутренней или внешней CRAM-памяти.

FSEL выбирает информацию для адресной шины CRAM. Это может быть информация одного из трех регистров адреса или COP, или адрес системной маски, или временного (рабочего) местоположения (TMPA).

CCI используются для того, чтобы загрузить параметры регистров и увеличить их адреса.

Биты служебные (управления) [15:8] представляют собой COP или TMPA, которые всегда используются раздельно и, следовательно, могут занять те же самые биты слова управления, не требуя аппаратных средств для их мультиплексирования.

TTOP закодирован с помощью восьми наименьших значащих бит микрокоманды. Чтобы ограничить размер микрокоманды, эти биты могут также быть заняты признаками переходов.

Модуль упорядочения микрограмм определяет стартовый адрес микрокоманд, которые должны быть выполнены. Логика управления генерирует сигналы для выбора следующего адреса памяти управления (следующая микрокоманда), уменьшая счетчик цикла и читая следующую команду из регистра команды.

Память управления перезаписывает информацию от внешней шины, разрешая динамическое вторичное наполнение микрокомандами.

Устройство адресации (адресный модуль) состоит из трех 8-битовых регистров адреса (AR0-AR2), трех соответствующих регистров расширения адреса (AX0-AX2), 8-битового адресного регистра (CBA) и 8-битового регистра инкремента/декремента. CBA реализуется как 6-битовый регистр, если CRAM-чип имеет 512 PE.

Как команды CRAM, так и данные от ведущего процессора передаются до CRAM-чипа, используя буферы чтения-записи. Когда CRAM выполняет команды, находящиеся в FIFO, хост-процессор может предварительно загрузить данные в буфер записи, затем выдать команду (которая будет поставлена на очередь в FIFO) и передать эти данные от буфера до CRAM. Такой параллелизм уменьшает время инициализации или чтения переменных CRAM. Кроме того, наличие буферов данных упрощает синхронизацию передачи данных между хостом и CRAM и снижает нагрузку на шину. Чтобы уменьшить общее время передачи данных, загрузка данных и команда записи от хоста выполняются параллельно.

Контроллер также содержит две группы регистров, к которым можно обратиться программисту. Первая группа – с картографируемой памятью. Вторая группа содержит регистры, состояние которых обычно изменяется между командами. Они используются, чтобы установить

параметры контроллера, типа расширения адреса и длины слова. Чаще всего параметры регистров не картографированы памятью и могут быть только инициализированы со специализированными командами регистра загрузки.

Кроме параметров регистров, описанных выше, все модули контроллера CRAM, к которым можно обратиться программистам, картографируемы памятью. Карта памяти распределена таким образом, чтобы минимизировать декодирование адреса модулей контроллера.

Применение интеллектуальной памяти для реализации функций распределения памяти компьютерной системы

В [13] описана интеллектуальная система памяти (ИСП), которая берет на себя функции обслуживания распределением и освобождением памяти, выполняемые обычно центральным процессором. Показано, что перемещение функций управления памятью от центрального процессора к интеллектуальной динамической оперативной памяти устраняет в среднем 60% неудачных обращений в КЭШ по сравнению с обычным методом, где функции размещения/освобождения выполняются центральным процессором. Такая интеллектуальная система содержит специализированный процессор, встроенный в чип динамической оперативной памяти (DRAM). Она определена как интеллектуальный менеджер памяти (ИМП). Архитектура ИМП может быть выполнена с использованием ASIC (специализированных интегральных схем реконфигурируемой логики) или в виде традиционного конвейерного проекта. Рассматриваются два направления выполнения интеллектуальной памяти: расширение централизованного управления (eController) и использование RAM вместо DRAM устройств (eRAM). Можно использовать также программируемый конвейер или специализированные интегральные схемы (ASIC).

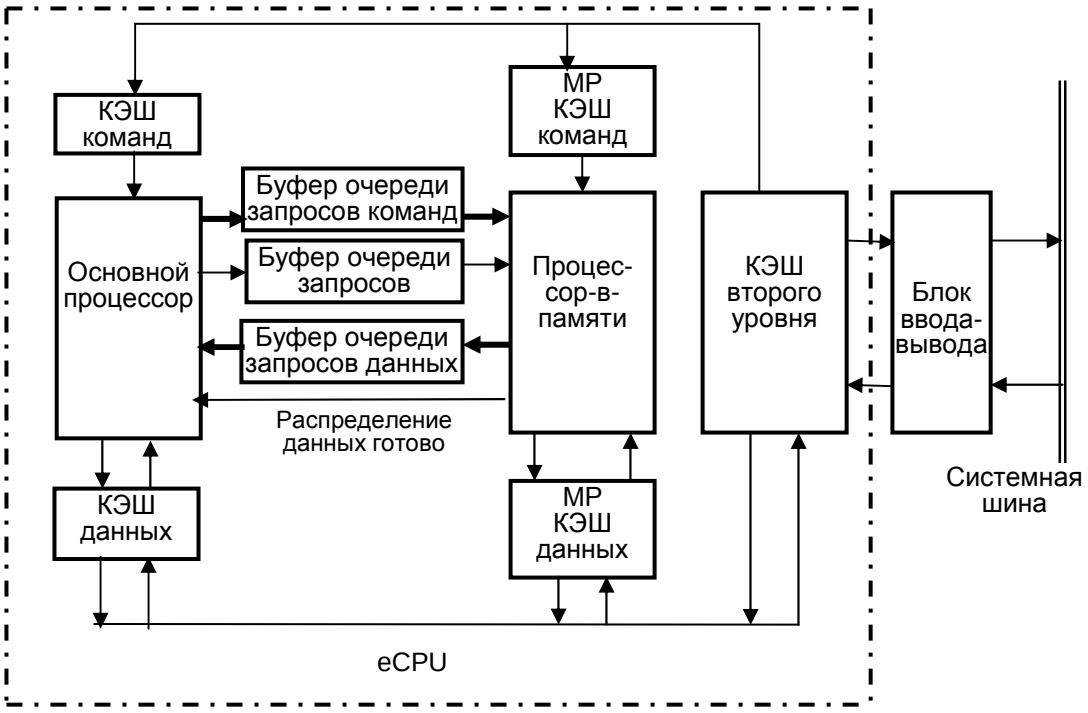

Рис. 6. Архитектура интеллектуальной системы памяти для управления ресурсами

Процессор памяти представляет собой простое целочисленное устройство, поэтому компилятор должен разделить коды памяти приложений и задачи центрального процессора. Предпочтительно, чтобы задачи, которые предназначаются для выполнения процессором памяти, были информационно насыщенными. Центральный процессор и процессор памяти могут работать параллельно, и, следовательно, зависимость данных между их кодами должна быть минимизирована.

С имеющейся в настоящее время технологией реализации динамической оперативной памяти можно разместить на одном чипе памяти типа DRAM (емкостью 256 Мбайт) логику, достаточную, чтобы выполнить необходимые операции управления памятью. Например, можно поместить интеллектуальный процессор системы памяти на одном чипе с центральным процессором, обозначив полученную систему как расширенный центральный процессор (eCPU), упрощенная схема которого показана на рис. 6. Содержательная сущность шагов процесса распределения с помощью интеллектуальной памяти состоит в следующем [13]:

1. Поиск свободной области памяти достаточного размера, чтобы удовлетворить запрос.

2. Формирование программой распределения начального адреса доступной области.

3. Изменение программой распределения векторных бит, которые соответствуют найденной свободной области; этот шаг называют переброской бита.

7. Выводы

Оптимальное управление памятью и данными, включающее распределение памяти, размещение данных, состоящее из нахождения исходных данных и перемещения данных во время выполнения, может существенно уменьшить время выполнения приложения (до 30%) по сравнению с известными методами распределения данных типа построчных или постолбцовых распределений.

Расширенные функциональные возможности и способы применения памяти PIM-систем накладывают определенный отпечаток на решение задач управления памятью, которые могут существенно отличаться от решения соответствующих задач управления памятью КС с классической архитектурой. Так, при распределении памяти в PIM-системе DIVA часть памяти, используемая хостом как “немая” память, управляется операционной системой хоста, используя стандартное распределение, замещение и механизмы свопинга (подкачки). Локальная и глобальная, совместно используемая память PIM, управляются при взаимодействии главного процессора (хоста) с процессорами памяти (PIM-процессорами).

В общем случае определены несколько подходов к управлению размещением данных в PIM-системах: один основан на использовании “окон выполнения”, другой – на коллекциях и макросерверах и третий – на применении специальной программной системы (например, SAGE).

При решении проблемы распределения и управления данными PIM-системы типа Gilgamesh распределение производится на уровне макросерверов отображением в параметризованную абстрактную архитектуру, которая понимается как набор абстрактных узлов, связанных друг с другом через управляющие пакеты с использованием сети связи. При этом абстрактные узлы отображаются системным программным обеспечением нижнего уровня на

основную физическую архитектуру. Макросерверы реализуют специальные методы, которые позволяют также полностью или частично перераспределять распределенные структуры данных.

Метод операторного разбиения сообщения и механизм планирования данных для архитектур интеллектуальной памяти типа FlexRAM с помощью специализированной программной системы типа SAGE может дать ускорение до 3,87 раза по сравнению с другими методами управления памяти в PIM-системах.

Применение отдельного процессора на кристалле PIM-системы (или специального контроллера), на который можно переместить от ведущих процессоров функции размещения/освобождения памяти, позволяет повысить эффективность использования каждого процессора для решения непосредственно задач обработки и устранить загрязнение КЭШ, вызванное этими функциями. Такое перераспределение функций позволяет устранить в среднем более 50% неудачных обращений в КЭШ (по сравнению с обычным методом, где функции размещения/освобождения выполняются ведущими процессорами).

В качестве такого контроллера может быть использован интеллектуальный модуль памяти (ИМП), т.е. специальный вариант PIM-компьютера, который встраивается в чип динамической оперативной памяти для выполнения функции управления памятью. Архитектура ИМП может быть выполнена с использованием FPGA (ПЛИС) или в виде традиционного конвейерного проекта. При этом к стандартному набору функций добавляются функции распределения и освобождения памяти.

Всесторонне оценивая рассмотренные выше решения по управлению памятью PIM-систем, можно сделать вывод, что при решении проблемы распределения памяти и оптимального размещения данных целесообразно основываться на положительных особенностях систем этого типа (по сравнению с классическими системами на одном кристалле), принимая во внимание, прежде всего, следующее:

– возможность передачи (с помощью управляющего пакета) набора операций (работы) над данными к местам расположения требуемого массива данных на соответствующих узлах распределенной многопроцессорной системы, а не наоборот – массива данных к процессорам, как это обычно делается;

– возможность размещения в одной строке данных, содержащей достаточно большое количество полноразрядных слов-операндов (несколько десятков или сотен), данных или их идентификаторов, которые могут потребоваться на последующих тактах обработки приложения.

Таким образом, вместо массовых пересылок данных к средствам обработки и динамического перераспределения памяти в процессе реализации приложений наиболее эффективно, по нашему мнению, направлять команды (работу) и ресурсы обработки информации разного уровня к соответствующим массивам данных, расположенным в банках и подбанках памяти различных узлов распределенной PIM-системы.

СПИСОК ЛИТЕРАТУРЫ

1. Палагин А.В. Основные принципы построения вычислительных систем с архитектурой «Процессор-в-памяти» / А.В. Палагин, Ю.С. Яковлев, Б.М. Тихонов // Управляющие системы и машины. – 2004. – № 5. – С. 30–37.

2. Renau Jose Memory hierarchies in intelligent memories: Energy/performance design. – http://iacoma.cs.uiuc.edu/~renau/docs/msuiuc.pdf.
3. Hall M., Steele C. Memory Management in a PIM-Based Architecture. – http:// www.isi.edu/~mhall/diva-mem00.ps.
4. Яковлев Ю.С., Тихонов Б.М. О распределении памяти в компьютерных системах // Управляющие системы и машины. – 2006. – № 5. – С. 40–47.
5. Rezaei Mehran Intelligent memory manager: towards improving the locality behavior of allocation-intensive applications. – http://ranger.uta.edu/~rezaei/research/dissertation.pdf.
6. Parrish O., Peiffer R., Thomas J., Hilpert J., Edwin J. Memory address mechanism in a distributed memory architecture. United States Patent, № 5.117.350, G06F 012/06; G06F 015/16. – 1992. – May 26. – 44 c.
7. Яковлев Ю.С., Тихонов Б.М. Об оптимизации размещения данных в PIM-системе // Математичні машини і системи. – 2006. – № 3. – С. 24–35.
8. Yi Tian Edwin H.-M. Sha Chantana Chantrapornchai Peter M. Kogge Optimizing Data Scheduling on Processor-In-Memory Arrays. Dept. of Computer Science and Engineering. University of Notre Dame. – http:// yipdps.eece.unm.edu/1998/papers/074.pdf.
9. Zimaa Hans P., Sterlingb Thomas L. Gilgamesh: A Multithreaded Processor-In-Memory Architecture for Petaflops Computing. – http//sc-2002.org/paperpdfs/pap.pap105.pdf.
10. Zimaa Hans P., Sterlingb Thomas L. The Gilgamesh Processor_in_Memory Architecture and Its Execution Model. – http://www.icsa.informatics.ed.ac.uk/cpc2001/proceedings/zima.ps.
11. Tsung-Chuan Huang, Slo-Li Chu, Sun Yat-sen A Parallelizing Framework for Intelligent Memory Architectures. Department of Electrical Engineering National University Kaohsiung, Taiwan. – parallel. its. sinica. edu. tw/ cth pc/7th pc /03 CTHPC 2001-Hardcopy.pdf.
12. Peter M. Nyasulu, B.Sc., M.Eng. System Design for a Computational-RAM. Logic-In-Memory Parallel-Processing Machine. – http://129.215.96.3:1234/~chrb/papers/cram_thesis.pdf.
13. Mehran Rezaei Intelligent memory manager: towards improving the Locality behavior of allocation-intensive applications. Dept. of Computer Science and Engineering. University of North Texas, 2004. – http//ranger.uta.edu/~rezaei/research/dissertation.pdf.

Стаття надійшла до редакції 19.11.2007