УДК 681.322.00; 681.324.00

Ю.С. ЯКОВЛЕВ, Б.М. ТИХОНОВ

ОБ ОПТИМИЗАЦИИ РАЗМЕЩЕНИЯ ДАННЫХ В PIM-СИСТЕМЕ

Abstract: *On the basis of the analysis of existing approaches to accommodation of data in PIM-systems ("Processor-in-Memory") it is drawn a conclusion on specialization of the approach for each created PIM-system. Substantive provisions for the construction of a universal method of optimum accommodation of data based on features of the architecturally-structural organization of PIM-systems are formulated. The opportunity of its practical realization on the basis of production decisions of the declaration patent of authors is noted.*
Key words: *"processor-in-memory", method of data distribution.*

Анотація: *На основі аналізу існуючих підходів до розміщення даних у PIM-системах ("Processor-in-Memory") зроблений висновок про спеціалізацію підходу для кожної створеної PIM-системи. Сформульовані основні положення для побудови універсального методу оптимального розміщення даних, заснованого на особливостях архітектурно-структурної організації PIM-систем. Відзначена можливість його практичної реалізації на основі інженерно-технічних рішень деклараційного патенту авторів.*
Ключові слова: *"процесор-в-пам'яті", метод розподілу даних.*

Аннотация: *На основе анализа существующих подходов к размещению данных в PIM-системах ("Processor-in-Memory") сделан вывод о специализации подхода для каждой созданной PIM-системы. Сформулированы основные положения для построения универсального метода оптимального размещения данных, основанного на особенностях архитектурно-структурной организации PIM-систем. Отмечена возможность его практической реализации на основе инженерно-технических решений декларационного патента авторов.*
Ключевые слова: *"процессор-в-памяти", метод распределения данных.*

1. Введение

Развитие полупроводниковой технологии обеспечило плотное размещение динамической оперативной памяти (DRAM) и логики КМОП на том же самом кристалле (чипе), что привело к появлению нового класса компьютерных систем, известного как класс "Процессор-в-памяти" (Processor-in-memory – PIM) [1, 2].

PIM отличается от классической системы на чипе тем, что реализуется широкая полоса пропускания память-процессор на уровне 100 Гб/с, обеспечивая производительность до 10 Гипс (32-разрядные операнды) на чипе памяти с емкостью 16 Мбайт. При этом логические схемы для обработки информации имеют прямой доступ ко всем битам строки памяти (например, 2048 бит), используя новое поколение очень широких арифметико-логических устройств и соответствующие системы команд.

Несмотря на то, что архитектура "Процессор-в-памяти" была предложена как целевая архитектура для достижения высокой эффективности вычислительных операций, одно из главных препятствий к достижению указанной цели – это межпроцессорные связи, которые удлиняют общее время выполнения приложения.

Поэтому одной из самых больших проблем при создании PIM-систем является поиск оптимального способа размещения и перемещения данных в массиве PIM, чтобы минимизировать критерий размещения $K_{разм}$, который условно определен как "стоимость связи" [3]. Стоимость связи между любыми двумя процессорами определяется как расстояние между этими процессорами по оси X и оси Y в 2-мерной сетке, взвешенной перенесенным объемом данных. Полная стоимость коммуникаций процессора – сумма стоимости связи для всех его данных, а полная стоимость связи для приложения определяется суммированием полной стоимости связи каждого процессора в массиве процессоров.

Прежде чем начинается обработка приложения, это приложение разбивается на разделы, а его данные распределяются по процессорам. Хорошее планирование данных, включающее нахождение исходных данных и их перемещение во время выполнения, может давать существенное понижение полной стоимости связи и времени выполнения приложения.

Несмотря на то, что идеология PIM-систем в современной интерпретации развивается уже десятки лет (примерно с 1986 г.) и за рубежом создано по этой идеологии значительное количество действующих систем, но в настоящее время отсутствуют теория, концепция, а также универсальные методы оптимального размещения исходных и полученных при реализации приложений данных во всем PIM-массиве памяти, которые бы существенно уменьшали потоки данных между узлами обработки PIM-системы. Это частично объясняется тем, что создавались, как правило, либо специализированные, либо в лучшем случае проблемно-ориентированные PIM-системы, в которых каждый разработчик по-своему и специфично решал проблему управления данными.

Принимая во внимание актуальность этой проблемы, мы посчитали целесообразным выделить несколько подходов к размещению данных: один подход основан на использовании так называемых *окон выполнения* [3], другой – на так называемых *коллекциях* и *макросерверах* [4, 5], третий – на применении специальной программной системы разбиения инструкций и построения трафика – SAGE [6]. Информация об этих подходах может быть полезной при решении проблемы управления данными для конкретных направлений построения PIM-систем.

2. Методы распределения данных, основанные на окнах выполнения

Для решения проблемы планирования данных могут быть использованы 2 метода [3]: *метод единственного центра* (Single-Center Data Scheduling – SCDS), в котором данные остаются на том же самом месте для всего периода выполнения, и *метод многих центров*, который позволяет перемещать данные во время выполнения.

Метод единственного центра планирования данных не рассматривает перемещение данных во время выполнения, они остаются в 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. При этом список процесса, содержащий список процессоров, создается для каждого данного. Он сортируется в восходящем порядке стоимости связи, учитывая данные, назначенные каждому процессору. Первый процессор в списке – оптимальное место, которое приводит к минимальной стоимости полной связи для каждого данного. Этот вид размещения данных назван единственным центром планирования данных.

Многоцентровое планирование данных 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 из данных различных шагов выполнения.

В [3] рассматриваются два алгоритма планирования данных, позволяющие перемещать данные во время выполнения. Первый называется *локально-оптимальным многоцентровым*

планированием данных (Local-Optimal Multiple-Center Data Scheduling – LOMCDS), второй – *глобально-оптимальным многоцентровым планированием данных* (Global-Optimal Multiple-Center Data Scheduling – GOMCDS). Первый обеспечивает локальное оптимальное размещение данных для каждого данного относительно каждого индивидуального окна выполнения. Второй рассматривает все окна выполнения и использует метод самого короткого пути для нахождения глобального оптимального размещения данных для каждого данного в каждом окне выполнения.

При локально-оптимальном многоцентровом планировании данных все окна выполнения известны. Определяют центр в каждом окне выполнения относительно каждого данного, соответствующий минимальной стоимости 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 (список процессоров), чтобы найти первый доступный процессор для каждого данного в каждом окне выполнения.

При глобально-оптимальном многоцентровом планировании данных находят центры каждого окна выполнения относительно каждого данного и создают диаграмму стоимости $G = (V, E, w)$ в виде взвешенного направленного графа без петель (Directed Acyclic Graph – DAG). Здесь V – набор вершин, включая псевдоисходную вершину s, псевдовершину адреса d. Каждая вершина $v_{ij} \in V$ соответствует j-му процессору окна выполнения i для $0 \le i \le n-1$, $0 \le j \le m-1$, где n – общее число окон выполнения, а m – общее число процессоров. При этом E – набор дуг, соединяющих вершину v_{ij} с вершиной $v_{i+1;k}$ для всех i и m, где $0 \le i \le n-2$ и $j \ge 0$; $k \le m-1$, включая дуги от источника s к вершине $v_{0;j}$ и дуги от вершины $v_{n-1;j}$ к вершине d для $0 \le j \le m-1$.

Весовая функция w, которая отображает E в целое число, определяется следующим образом: для каждой дуги $e = (s; v_{0;j})$ функция $w(e)$ определяет полную стоимость связи при приёме данных, которые хранятся в процессоре j в окне выполнения 0. Для каждой дуги $e = (v_{i;j}; v_{i+1;k})$ функция $w(e)$ отражает суммирование стоимости связи между процессорами j и k, и полная стоимость связи при приеме данных запоминается в процессоре k в окне выполнения $i+1$. Для $e = (v_{n-1;j;d})$ функция $w(e)$ равна нулю.

Пусть $v_{i;ji}$ – центр, отобранный алгоритмом. При этом

$$s \to v_{0;j0} \to v_{1;j1} \to \ldots . v_{n-1;jn-1} \to d$$

представляет последовательность перемещения данных среди этих центров. Суммирование w по всему пути – это полная стоимость связи, включая стоимость перемещающихся данных. Поэтому самый короткий путь от вершины s к d дает глобально-оптимальные центры каждого окна выполнения.

При глобально-оптимальном многоцентровом планировании данных входом алгоритма планирования данных являются строки процессора для всех окон выполнения относительно каждого данного. Чтобы понизить полную стоимость связи при реализации приложения, данные не должны быть помещены в той же самой позиции для всех окон выполнения. Кроме того, размер окна выполнения может также играть существенную роль в снижении общей стоимости связи. В зависимости от числа команд, выполняемых в окне, окно выполнения может быть меньше или больше. Если окно выполнения слишком мало, стоимость перемещающихся данных между центрами окон может быть большой. Можно группировать соответствующие окна выполнения относительно каждых данных в большие, если объединение последовательных окон выполнения и расположение данных в центре нового окна могут понизить общую стоимость связи. Помещая данные в центре нового окна выполнения, мы можем понижать стоимость перемещающихся данных между центрами так же, как и общую стоимость связи.

Учитывая изложенное, исходные посылки общего характера для оптимального размещения данных можно представить следующим образом [3]:

– массив процессоров может быть сформирован в виде 2-мерной сетки, где каждый процессор имеет свою собственную локальную память, при этом метод "X-У маршрутизации" используется для связи между процессорами;

– выполняемое приложение может быть разделено на множественные шаги, а последовательность параллельных шагов выполнения – сгруппирована в так называемое исполнительное окно;

– строка ссылки процессора относительно данных в одном окне выполнения отражает последовательность процессоров, требующих данные в этом окне выполнения;

– строка ссылки данных процессора в одном окне выполнения отражает последовательность данных, на которые ссылается процессор в этом окне выполнения;

– центром по отношению к данным окна выполнения является процессор, запоминающий данные;

– локальный оптимальный центр по отношению к данным окна выполнения является центром окна выполнения с полной минимальной стоимостью связи относительно данных.

При реализации группировки окон можно воспользоваться следующими положениями:

– если p_{i1} и p_{i2} – самая близкая пара локальных оптимальных центров данных D для окон выполнения T_0 и T_1, то стоимость связи $\cos t(D, T_{0,} p_i)$ по направлению от p_{i1} к p_{i2} увеличивается строго монотонно;

– если для 2-мерного массива процессоров $p_{i1;j1}$ и $p_{i2;j2}$ – самая близкая пара локальных оптимальных центров относительно данных D для окон выполнения T_0 и T_1, то стоимость связи $\cos t(D, T_0, p_{i,j})$ по любому пути, который дает самое короткое расстояние между процессором $p_{i1,j1}$ и $p_{i2,j2}$ увеличивается строго монотонно;

– если p_1 и p_2 – самая близкая пара локальных оптимальных центров относительно данных D для двух последовательных окон выполнения T_0 и T_1, то группировка T_0 и T_1,не понижает полную стоимость связи относительно данных D.

Из свойств группировки можно сделать вывод, что число окон выполнения, рассматриваемых для каждой группы, может воздействовать на понижение полной стоимости связи. Однако исчерпывающее нахождение всех возможных выборов группировки может быть весьма трудоемким и дорогостоящим.

3. Методы распределения данных, основанные на применении коллекций и макросерверов

Сущность этих методов целесообразно рассмотреть на примере решения проблемы распределения и управления данными при создании PIM-системы типа Gilgamesh [4, 5].

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 множественные банки 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 образующих 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 пакеты, механизм коммутации, 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 использует специальные управляющие пакеты для поддержки управляемого сообщением вычисления среди чипов MIND и их узлов. Пакет – это коммуникационно-управляющий пакет переменной длины, который включает значения параметров и спецификаторы действий, целенаправленные определяемому адресату. Пакеты назначают виртуальные адреса и могут использоваться, чтобы реализовать обычные операции типа отдаленной загрузки или хранения, так же, как и вызывать методы выполнения операций на отдаленный чип MIND.

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 включают область 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, не входящей в область определенных чипов MIND, но сохраненной в этой таблице. Например, чтобы удовлетворить некоторые ограничения местоположения, гостевая страница может быть назначена на чип MIND вне области виртуальной адресации. 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 С связана с индексным доменом, который обеспечивает однозначное имя для его примитивных элементов.

Большие коллекции распределяются между узлами архитектуры. Распределение данных делит индексный домен коллекции на взаимно непересекающиеся классы, названные дистрибутивными долями. Они характеризуются следующими свойствами:

1) определенная дистрибутивная доля определяет область местоположения, при этом все ее элементы отображаются в памяти отдельным узлом;

2) различные дистрибутивные доли могут быть отображены к различным узлам, таким образом допуская параллельную операцию между долями, при условии, что никакие зависимости не нарушены.

Gilgamesh обеспечивает уровень микропрограммных средств, названный *уровнем макросервера,* который поддерживает основанное на объектах управление во время выполнения потоков данных, позволяя явный и 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. Макросерверы обеспечивают богатую семантику

для распределения групп данных через узлы архитектуры, включая правильные и нерегулярные отображения, динамическое перераспределение и средство для определяемых пользователем специализированных классов, поддерживающих специфические методы доступа и стратегии трансляции.

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.

При распределении данных и линеаризации для макросерверов сначала идентифицируют атомарные узлы (компоненты) структуры данных отображением $D : I \to \Omega$, где I – индексный домен (область), а Ω – некоторый "универсальный" набор значений.

Явное управление отображением на физическое пространство зачастую вынуждает транслятор или систему поддержки выполнения разбивать непрерывную часть виртуального адресного пространства на меньшие части, совместимые с дистрибутивной стратегией.

Распределение структур данных будет в основном создаваться во время, когда структуры данных размещаются. Следовательно, макросерверы реализуют специальные методы, которые позволяют полностью или частично перераспределить распределенные структуры данных.

Если (D_1, δ^{D1}) и (D_2, δ^{D2}) – распределенные структуры данных, обработанные в общем контексте, тогда δ^{D1}, δ^{D2} и их зависимость определяют степень параллелизма и местоположения в алгоритме.

Например, если D_1 и D_2 – матрицы с тем же самым индексным доменом, чья сумма должна быть вычислена и отнесена к третьей матрице, тогда распределение всех трех матриц идентично и тождественно приводит полностью к локальным операциям, выполненным параллельно во всех блоках памяти, включенных в распределение. Следует отметить, что две распределенные структуры данных выровнены, если в течение стадии выполнения зависимость линеаризации установлена между их распределениями.

Далее приведены простые примеры результата распределения массива структуры данных различного типа. Во всех рассматриваемых случаях принята абстрактная архитектура с четырьмя узлами (от P_1 до P_4).

Пример 1. Обычное блочное распределение массив. Обычное блочное распределение представляет собой простое обобщение регулярных блочных распределений, допуская переменный размер блока при сохранении смежности индексного поддомена, связанного с каждой дистрибутивной долей. В комбинации с перестановкой элементов матрицы обычное блочное распределение может использоваться для неструктурных сеток. Рис. 1 представляет пример для

одномерного массива $A(1:12)$. Связанное распределение сегментов дает: $\lambda(1) = [1:5], \lambda(2) = [6:7], \lambda(3) = [8:10], \lambda(4) = [11:12].$

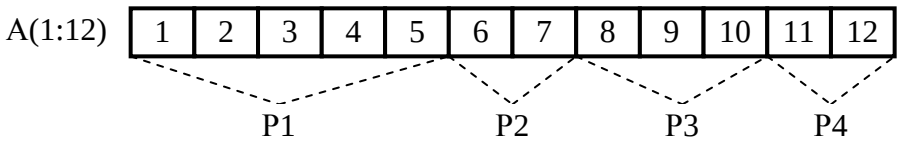

Рис. 1. Обычное блочное распределение

Пример 2. Распределение структуры данных типа дерева. Рис. 2 представляет пример для распределения структуры данных типа дерева. Стиль блоков определяет отображение, приводя к дистрибутивным долям $\lambda(1) = \{1,\ 1.1,\ 1.1.1,\ 1.1.1.1\}$, $\lambda(2) = \{1.2,\ 1.2.1,\ 1.2.2\}$, $\lambda(3) = \{1.3,\ 1.3.1\}$, $\lambda(4) = \{1.3.1.1,\ 1.3.1.2,\ 1.3.1.3\}$.

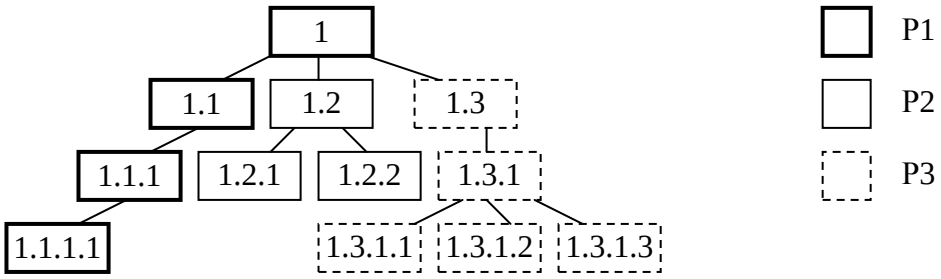

Рис. 2. Дерево распределения данных

Пример 3. Распределение разреженной матрицы. Рассмотрим разреженную матрицу $(n \times m)$, представленную на рис. 3 (n = 10 и m = 8). Элементы, значения которых равны нулю, представлены пустыми блоками; ненулевые элементы явно определены и пронумерованы. Распределение – нерегулярное, основано на содержании матрицы; цель состоит в том, чтобы иметь приблизительно тот же самый номер ненулевых элементов в каждом дистрибутивном сегменте.

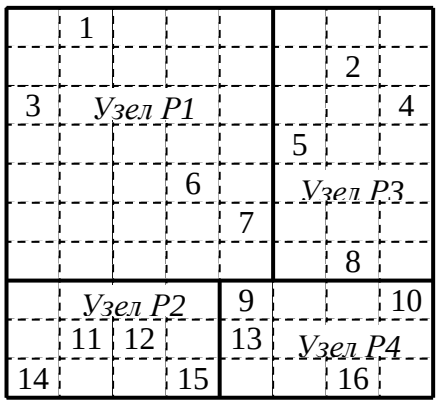

Рис. 3. Распределение разреженной матрицы

На рис. 3 четыре дистрибутивных сегмента определяются как $\lambda(1) = [1{:}7,\ 1{:}5]$ с ненулевыми элементами (1; 2), (3; 1), (5; 4), (6; 5); $\lambda(2) = [8{:}10,\ 1{:}4]$ с ненулевыми элементами (9; 2), (9; 3), (10; 1), (10; 4); $\lambda(3) =$ [1:7, 6:8] с ненулевыми элементами (2; 7), (3; 8), (4; 6), (7; 7) и $\lambda(4) = [8{:}10,\ 5{:}8]$ с ненулевыми элементами (8; 5), (8; 8), (9; 5), (10; 7).

Чтобы управлять данными и, в первую очередь, отображением виртуальной памяти на физическую память, необходимо ответить на вопросы, возникающие в контексте распределения, перераспределения и линеаризации данных. К таким вопросам следует отнести:

– что представляют из себя аппаратные средства для реализации рассмотренных выше функций и какое нужно программное обеспечение, чтобы квалифицированно воспользоваться преимуществом распределения и линеаризации информации;

– как может такая информация перемещаться через иерархические уровни программной системы к ассемблеру машинного языка.

4. Метод распределения данных с помощью специальных программных систем

В [6] предложена программная система SAGE (Statement-Analysis-Grouping-Evaluation), с помощью которой распределяют задачи по процессорам с различными характеристиками и разделяют первоначальную программу на несколько частей с целью их параллельного выполнения. Она не только может анализировать исходную программу, но и генерировать взвешенный граф (диаграмму) зависимости части (*Weight Partition Dependence Graph* – WPG), определять вес каждого блока и затем отправлять соответствующие задания процессорам памяти и главному (HOST) компьютеру. SAGE – система была ориентирована на решение этих задач применительно к архитектуре интеллектуальной памяти типа FlexRAM. Укрупненная схема компиляции программы с помощью системы SAGE приведена на рис. 4.

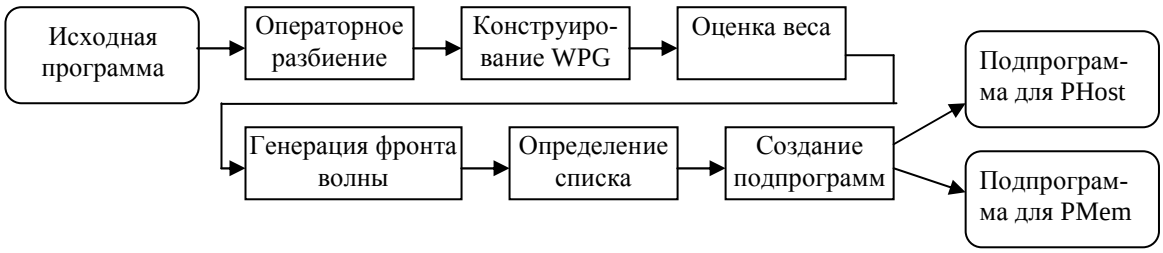

Рис. 4. Укрупненная схема компиляции программы с помощью системы SAGE

В этих стадиях используют распределение цикла (Loop Distribution), чтобы разбить первоначальную диаграмму зависимости на узловые части (Node Partition) Π и создать взвешенную диаграмму (граф) зависимости раздела (Weighted Partition Dependence Graph –WPG), которая используется в оценке веса и стадиях определения расписания.

Далее приведен укрупненный алгоритм разбиения сообщения.

Дан цикл $L = (I_1, I_2, \cdots I_d)(S_1, S_2 \cdots S_d)$.

Шаг 1. Создается граф зависимости G, анализируя выражения нижнего индекса и индексную модель.

Шаг 2. Граф G разбивают на узловые разделы Π согласно следующим положениям.

На графе зависимости G, для данного цикла L, определяют вершины (узлы) Π из $\{S_1, S_2, \cdots S_d\}$ таким образом, что S_k и S_I, $k \neq I$ находятся в том же самом подмножестве, если и только если $S_k \Delta S_I$ и $S_I \Delta S_k$, где Δ – косвенно зависимое отношение по данным. На разделе $\Pi = \{\pi_1, \pi_2\}$ определяют частично упорядочивающие отношения $\alpha, \overline{\alpha}$ и α^0 следующим образом. Для $i \neq j$:

1) $\pi_i \; \alpha \pi_j$ тогда и только тогда, если существуют $S_k \in \pi_i$ и $S_i \in \pi_j$ такие, что $S_k \; \delta \; S_I$, где δ – истинное соотношение зависимости;

2) $\pi_i \overline{\alpha} \; \pi_j$ тогда и только тогда, если существуют $S_k \in \pi_i$ и $S_i \in \pi_j$ такие, что $S_k \; \overline{\delta S_I}$, где $\overline{\delta}$ – отрицание отношения зависимости;

3) $\pi_i \; \alpha^0 \pi_j$ тогда и только тогда, если существуют $S_k \in \pi_i$ и $S_i \in \pi_j$ такие, что $S_k \; \delta^0 S_I$, где δ^0 – выходное соотношение.

В системе SAGE эти отношения используются в качестве механизма разбиения на предварительные циклы, чтобы сгенерировать диаграмму WPG.

Если имеются большие блоки, являющиеся результатом соотношений зависимости управления, сначала конвертируют управляющую зависимость в зависимость данных, а затем выделяют разделы графа зависимости.

Шаг 3. На разделении Π устанавливают граф взвешенной зависимости разделов $WPG(PE)$ согласно положениям.

Для данной вершины раздела Π, выделенного выше, определяют граф зависимости взвешенных разделов $WPG(PE)$.

Для каждого $\pi_i \in \Pi$ существует вершина (узел) $b_i(I_i, S_i, W_i, O_i) \in P$, где I_i – индекс цикла и S_i – инструкция тела, которая может быть инструкцией назначения или другого цикла, W_i – вес вершины (узла) i в форме $W_i(PH, PM)$, где PH и PM – веса в $P(Host)$ и $P(Mem)$ соответственно, и O_i – порядок выполнения этой вершины (узла). Существует дуга $e_{ij} \in E$ от b_i до b_j, если b_i и b_j имеют отношения зависимости $\alpha, \overline{\alpha}$ или α^0, обозначенные выше.

Для генерирования волнового фронта и определения расписания используют специальный алгоритм для планирования $P(Host)$ и $P(Mem)$. Сначала определяют веса блоков в разделе P, а затем – порядок выполнения для каждого блока согласно их соотношению зависимости и лексикографическому порядку. Блоки, которые могут выполняться одновременно, образуют фронт волны. Блоки в том же самом фронте волны назначаются на $P(Host)$ и $P(Mem)$ процессоры на основании их весов.

5. Выводы

Несмотря на то, что архитектура “Процессор-в-памяти” была предложена как архитектура для достижения высокой производительности, одно из главных препятствий к достижению указанной цели – это межпроцессорные связи, которые удлиняют общее время выполнения приложения. Хорошее планирование данных, состоящее из нахождения исходных данных и перемещения данных во время выполнения, может давать существенное понижение времени выполнения приложения (до 30%) по сравнению с известными методами распределения данных типа построчных или поколонных распределений.

Метод операторного разбиения (разбиения сообщения) и механизм планирования данных для архитектур интеллектуальной памяти типа FlexRAM с помощью специализированной программной системы SAGE [6] может дать ускорение до 3,87 раза в FlexRAM-архитектуре.

Однако рассмотренные выше методы распределения данных являются достаточно трудоемкими и дорогостоящими как с точки зрения создания специализированных программных средств, так и с точки зрения затрат времени непосредственно на процесс распределения, что явно подтверждено достаточно длинной цепочкой последовательных процедур (рис. 4) по реализации распределения. По нашему мнению, при решении проблемы оптимального распределения данных в PIM-системах следует основываться на положительных особенностях систем этого типа (по сравнению с классическими системами на одном кристалле), принимая во внимание 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;

– возможность управляемого аппаратного подключения каждого ведущего процессора PIM-системы к массивам данных, размещенным в соответствующих банках памяти вместе со своими процессорами памяти P_{MEM};

– возможность управляемого аппаратного подключения каждого процессора P_{MEM} к данным, размещенным в соответствующих подбанках памяти каждого банка.

Таким образом, вместо массовых пересылок данных к средствам обработки и динамического перераспределения памяти в процессе реализации приложений наиболее эффективно, по мнению авторов, направлять команды (работу) и ресурсы обработки информации разного уровня к соответствующим массивам данных, расположенным в банках и подбанках памяти различных узлов распределенной PIM-системы. Частичное решение этой проблемы по предложенному авторами принципу отражено в [7].

СПИСОК ЛИТЕРАТУРЫ

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

2. Палагин А.В., Яковлев Ю.С., Тихонов Б.М., Першко И.М. Архитектурно-структурная организация компьютерных средств класса “Процессор-в-памяти” // Математичні машини і системи. – 2005. – № 3.– С.3 –16.
3. 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, IN 46556, yipdps.eece.unm.edu/1998/papers/074.pdf.
4. Zimaa Hans P., Sterlingb Thomas L. Gilgamesh: A Multithreaded Processor-In-Memory Architecture for Petaflops Computing.– http// sc-2002.org/paperpdfs/pap.pap105.pdf.
5. Zimaa Hans Pand, Sterlingb Thomas L. The Gilgamesh Processor_in_Memory Architecture and Its Execution Model. – http://www.icsa.informatics.ed.ac.uk/cpc2001/proceedings/zima.ps.
6. 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.
7. Сергієнко І.В., Кривонос Ю.Г., Палагін О.В., Коваль В.М., Яковлєв Ю.С., Тихонов Б.М. Система пам’яті з інтеграцією функцій зберігання та обробки інформації на одному кристалі. Деклараційний патент на корисну модель. № 6259, G06F 13/00, G06F 12/00; Заявлен 15.02.2005; Опубл.15.04.2005; Бюл. № 4. – 26 с.