УДК.004.27; 004.25; 004.382.2

Ю.С. ЯКОВЛЕВ, Е.В. ЕЛИСЕЕВА

МАТЕМАТИЧЕСКАЯ МОДЕЛЬ И СТРАТЕГИЯ РАСПРЕДЕЛЕНИЯ ПРИЛОЖЕНИЙ ДЛЯ ИНТЕЛЛЕКТУАЛЬНОЙ ПАМЯТИ РАСПРЕДЕЛЕННЫХ КОМПЬЮТЕРНЫХ СИСТЕМ

Abstract: *The model of the strategy of allocation of applications for intellectual memory which is grounded on model of the PIM-system containing objects, tags and the properties distinguishing it from computer systems with the classical architecture is offered. The set of the main procedures of the strategy of allocation of the applications, the integrated flowchart of the strategy and the flowchart of the hardware-software environment for support of implementation of this strategy are offered.*
Key words: *PIM-systems, the allocation of applications.*

Анотація: *Запропоновано модель стратегії розподілу додатків для інтелектуальної пам'яті, яка заснована на моделі PIM-системи, що містить об'єкти, ознаки та властивості, які відрізняють її від комп'ютерних систем із класичною архітектурою. Запропоновано набір основних процедур стратегії розподілу додатків, укрупнену блок-схему стратегії та блок-схему апаратно-програмного середовища для підтримки реалізації цієї стратегії.*
Ключові слова: *PIM-системи, розподіл додатків.*

Аннотация: *Предложена модель стратегии распределения приложений для интеллектуальной памяти, которая основана на модели PIM-системы, содержащей объекты, признаки и свойства, отличающие её от компьютерных систем с классической архитектурой. Предложены набор основных процедур стратегии распределения приложений, укрупненная блок-схема стратегии и блок-схема аппаратно-программной среды для поддержки реализации этой стратегии.*
Ключевые слова: *PIM-системы, распределение приложений.*

1. Введение

Как отмечено в [1, 2], ведущий процессор (ВП), размещенный вместе с процессорными ядрами (ПЯ) и банками памяти (БП) на кристалле распределенной интеллектуальной памяти, которая часто идентифицируется как PIM-система, реализует функции обработки наиболее сложных операций, плохо поддающихся распараллеливанию. Совсем другие функции возлагаются на ПЯ, основное назначение которых – массовая реализация простых, но зато доминирующих в алгоритме операций (например, сложение, сдвиг и др.) над огромным количеством данных, требующих массового обращения к памяти для чтения и записи как исходных, так и их результатов после обработки. Естественно, что при незначительном наборе функций по управлению простыми вычислительными операциями нет необходимости применять достаточно сложные полнофункциональные операционные системы (ОС). Поэтому целесообразно использовать усеченные ядра ОС (ЯОС), причем такие ядра размещают в соответствующей памяти каждого ПЯ, что придает им определенную независимость.

Если же рассматривать служебные функции и процедуры системных служб классических ОС, имеющих отношение к распределению приложений, то, с точки зрения их возможного применения к решению соответствующих задач в PIM-системах, необходимо отметить следующее.

Главным требованием, предъявляемым к ОС компьютерных систем с классической архитектурой, как известно, является выполнение ею функций управления ресурсами, обеспечения удобного интерфейса для пользователя и прикладных программ. Современная ОС, как правило, должна поддерживать мультипрограммную обработку, виртуальную память, свопинг, многооконный

ISSN 1028-9763. Математичні машини і системи, 2009, № 4

графический интерфейс пользователя, а также выполнять служебные функции [3, 4]. Так, функциями ОС по управлению памятью являются: отслеживание свободной и занятой памяти; выделение памяти процессам и освобождение памяти при завершении процессов; защита памяти; вытеснение процессов из оперативной памяти на диск, когда размеры основной памяти недостаточны для размещения в ней процессов, и возвращение их в оперативную память, когда в ней освобождается место, а также настройка адресов на конкретную область физической памяти.

Анализ существующих способов распределения приложений и организации управления памятью [5–9], используемых для классических компьютерных систем (КС), показал, что они без соответствующей доработки не могут быть использованы для решения аналогичных задач интеллектуальной памяти (PIM-системы) ввиду особенностей её архитектурно-структурной организации [2, 9–11] хотя бы потому, что, во-первых, в распределенных КС с классической архитектурой разделение приложений осуществляется на небольшое количество фрагментов [4] и, во-вторых, в КС с классической архитектурой (типа кластера) существует одна операционная система, в то время как в PIM-системе используется несколько. Поэтому требуется другой подход к решению задачи разделения приложений для PIM-систем, который учитывал бы особенности их архитектуры, сохраняя тем самым её преимущества перед КС с классической архитектурой.

Известные методы разделения приложений в PIM-системах [12–14] концентрируются на так называемой итерационно-основанной технологии – технологии трансформации циклов, чтобы выполнить все или некоторые итерации одновременно. При этом не рассматриваются различия возможностей ВП и ПЯ. Другой подход (так называемый операторно-основанный метод) использует в качестве основной единицы анализа оператор в цикле. Такой подход применяется в системе SAGE [12, 13], где исходная программа разбивается на несколько частей (модулей операторов), которые могут выполняться одновременно на ВП и ПЯ. В основе подхода принята упрощенная модель PIM-системы: модель содержит один ВП и один ПЯ (фактически PIM-система, размещенная на одном кристалле, содержит один ВП и множество ПЯ). Кроме того, судя по описаниям, в алгоритме разбиения не учитывается ряд факторов, имеющих важное значение, например, условные и безусловные переходы внутри и вне циклов, что может привести к необоснованному вычислению веса соответствующих модулей и, следовательно, к некорректному разбиению. Также может оказаться, что количество модулей, принадлежащих ПЯ, и в том числе их вес, на порядок и более отличаются соответственно от количества и веса модулей, принадлежащих ВП. Это означает, что ВП большую часть времени будет простаивать. Может быть и обратная картина, когда ПЯ будут простаивать, а ВП перегружен, т.е. наблюдается дисбаланс загрузки процессоров, откорректировать который трудно.

Использование оператора в цикле в качестве основной единицы анализа “оставляет в тени” объединение и планирование операторов, находящихся вне циклов, а это означает, что не все операторы могут быть проанализированы на предмет их параллельного выполнения.

Следует иметь ввиду, что PIM-системы ориентированы на решение сложных и трудоемких задач, требующих массового обращения к памяти при параллельной обработке большого количества (до нескольких тысяч и даже до нескольких сот тысяч) операндов в Petaflops-ном диапазоне. При этом они допускают обработку операндов большой разрядности (например, из

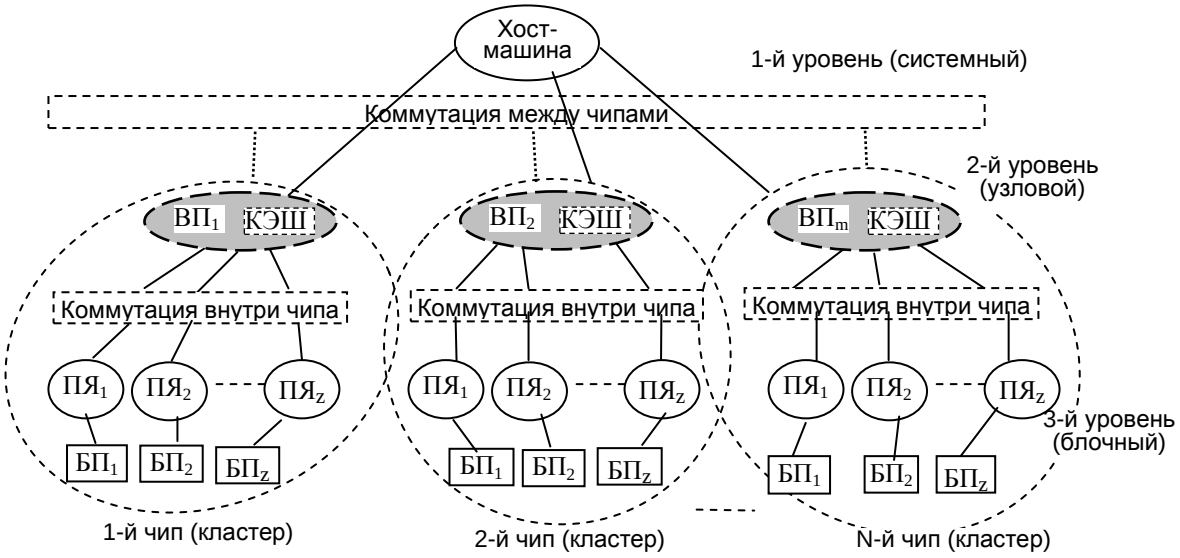
ряда 64, 128, 256, 1024 бит и т.д.), одновременно считывая или записывая за один цикл обращения к памяти большое количество таких операндов. Поэтому рассмотренные выше методы распределения приложений по процессорам PIM-системы, если принять во внимание указанные выше факторы, являются чрезвычайно сложными и трудоемкими. При этом вопросы сходимости алгоритма разбиения являются серьезной проблемой.

Другие подходы к разбиению приложений для PIM-систем как в зарубежной, так и в отечественной литературе, авторам не известны.

Далее предлагается достаточно обозримый вариант подхода к распределению приложений для PIM-систем с учетом особенностей их архитектурно-структурной организации, включая предлагаемую модель стратегии и процедуры распределения приложений с описанием входящих в них компонентов.

2. Модель PIM-системы для распределения приложений

Структурная интерпретация исходной многоуровневой модели PIM-системы для распределения приложений приведена на рис. 1.



Обозначения: $БП_i$ – i -й банк памяти; $КЭШ$ – память типа $КЭШ$; ВП – ведущий процессор; ПЯ – процессорные ядра.

Рис. 1. Структурная интерпретация модели PIM-системы для распределения приложений

1. Верхний уровень (системный) определяется хост-машиной с соответствующей ОС. На этом уровне на хост-машину возлагаются не только функции управления работой всей системы, но и функции распределения ресурсов по всем чипам. На этом уровне алгоритм A приложения разделяется на фрагменты $A \supset (A_1, A_2, ... A_i, ..., A_N)$, подлежащие распараллеливанию, которые распределяются между всеми кристаллами (чипами PIM-системы), принимая во внимание баланс загрузки $(\Delta_{ВП})$ каждого ВП.

2. Средний (узловой) уровень имеет непосредственное отношение к организации одного чипа, который содержит один ВП и множество ПЯ, подключенных к ВП через схему внутричиповой коммутации. На этом уровне решается задача, какая часть фрагмента $(A_{i(ВП)})$ передается для

реализации на ВП и какая часть $(A_{i(ПЯ*)})$ – для параллельной обработки на всем наборе ПЯ внутри одного чипа. Принимая во внимание, что все ПЯ на одном чипе являются одинаковыми и одновременно могут выполнять параллельные участки алгоритма, на начальном этапе построения модели распределения приложения можно представить, что фрагмент A_i алгоритма приложения распределяется между одним ВП и одним эквивалентным ПЯ, которому назначаются следующие параметры: система команд аналогична системе команд одного ПЯ, однако при этом тактовая частота работы (из-за параллелизма ПЯ) примерно в z раз выше (z – количество ПЯ на чипе согласно рис. 1), чем тактовая частота работы одного ПЯ, а емкость памяти равна сумме емкостей всех банков памяти, размещенных на данном чипе. При этом принимается во внимание функциональная ограниченность системы команд ПЯ, что в конечном итоге должно быть учтено при распределении фрагмента приложения на данном уровне.

3. Нижний уровень определяется непосредственно набором ПЯ, размещенным на чипе, количество которых z в наборе может быть больше или меньше количества частей, на которые распределена $A_{i(ПЯ*)}$, т.е.

$$A_{i(ПЯ*)} \supset (a_1, a_2, ..., a_j, ..., a_k), \text{ где } k \geq z \text{ или } k \leq z .$$

Основные функции ЯОС на этом уровне – реализовать одновременную высокоскоростную обработку данных выделенной части $A_{i(ПЯ*)}$ фрагмента приложения при массовом обращении к памяти, в максимальной степени используя особенности архитектурно-структурной организации PIM-системы, в том числе широкополосный канал “процессор-память”. При этом каждое ПЯ работает под управлением собственного ядра операционной системы $(ЯОС_{ПЯ})$, которая функционирует независимо от других $ЯОС_{ПЯ}$ в том смысле, что каждая из них принимает независимые решения о создании и завершении на конкретном ПЯ своих собственных процессов и об управлении своими локальными ресурсами.

3. Модель стратегии распределения приложения

Принимая во внимание рис. 1, модель стратегии распределения приложения можно представить в виде кортежа следующих процедур:

$$\xi = (R_{ПМ}, R_{Д}, R_{РПЛ}, Y_{КМ}), \quad (1)$$

где $R_{ПМ}$ – распределение памяти, $R_{Д}$ – размещение данных по выделенным областям памяти, $R_{РПЛ}$ – многоуровневое (согласно рис. 1) разделение приложений на составные части и распределение их по процессорам, $Y_{КМ}$ – управление коммутацией между процессорами распределенной системы.

Вопросы распределения памяти, размещения данных и управления коммутацией применительно к распределенным компьютерным системам (в том числе для PIM-систем) были освещены ранее в работах [9, 10, 15, 16] соответственно, поэтому в данной статье мы уделим основное внимание процедуре распределения приложений, т.е. параметру $R_{РПЛ}$ выражения (1). Эта процедура является наиболее сложной и наиболее ответственной для организации

распараллеливания вычислительного процесса, поскольку при её реализации на PIM-системе (в отличие от реализации аналогичной процедуры на классических КС) должны быть учтены следующие особенности архитектурно-структурной организации этого класса машин.

Каждый узел PIM работает под управлением своей собственной операционной системы ЯОС. Узлы должны быть в состоянии управлять их собственной памятью и вычислительными ресурсами при сохранении сильной интеграции в полную систему. При этом у узлов PIM нет доступа к информации о состоянии других узлов, в отличие от узлов классических КС, которые могут исследовать массив параметров, сохраняемых операционной системой для каждого узла классических КС (например, выполненная очередь).

PIM – узлы относительно бедны ресурсом и имеют ограниченную мощность обработки относительно традиционных вычислительных платформ. Пользователи должны использовать более мелкомодульный уровень разделения программы, чем на современных машинах, и системное программное обеспечение должно занимать небольшой объем памяти, чтобы остались ресурсы памяти для пользовательских программ.

PIM-система составляется из многих узлов PIM. Высокая степень параллелизма и мелкомодульное разделение программы обеспечиваются эффективной коммуникацией между узлами PIM. Время коммуникации в системах PIM меньше, чем в кластерах PC, так как узлы PIM – часть единственной машины на кристалле, а не автономных машин, подключенных с помощью внешней хост-сети. Коммуникация в среде PIM включает пакеты, которые посылают между объектами.

Широкая шина памяти, подключающая узлы PIM, также обеспечивает большую полосу пропускания памяти, чем многие традиционные кластерные решения. Эффективность коммуникации делает системы PIM лучшими для мелкомодульного распределенного вычисления, чем их традиционные кластерные копии.

Чипы PIM являются компонентами памяти и должны быть в состоянии выступить в качестве памяти (то есть, они должны быть в состоянии быстро ответить на запросы о данных). Кроме того, как части памяти, узлы PIM непосредственно не подключаются к широкому разнообразию периферийных устройств, к которым обычно подключаются универсальные компьютеры, что требует новых решений некоторых старых проблем (например, подкачки информации в память).

Сервисные службы операционной системы PIM-машины должны учитывать все эти особенности архитектурно-структурной организации PIM. При этом способность переместить вычисление к узлу PIM-системы, обладающему необходимым набором данных для текущего процесса, вместо того чтобы переместить данные к процессору для их обработки, является одним из самых интересных аспектов среды PIM.

Учитывая эти особенности, процедура разделения приложений может быть представлена в виде кортежа, который содержит следующие компоненты:

$$R_{РПЛ} = (M_{РПЛ}, R_F, B_{РПЛ}, R_{ТС}, B_{ЗГ}), \quad (2)$$

где $M_{РПЛ}$ – построение математической модели процедуры распределения приложений и описание её компонентов, R_F – выделение признаков (функций), отличающих PIM-систему от КС с

классической архитектурой при соответствующей целевой функции распределения, $B_{РПЛ}$ – выбор способа распараллеливания вычислительных алгоритмов, $R_{ТС}$ – формирование таблиц соответствия набора операций приложения и набора команд процессоров PIM-системы, $B_{ЗГ}$ – проверка баланса загрузки процессоров PIM-системы.

Принимая во внимание (1) и (2), математическую модель разделения приложений для распределения их частей по процессорам можно представить следующим образом:

$$E_{РПЛ} = \{(\forall a_{РПЛ} \in A_{РПЛ})[\exists R_{РПЛ} \in \xi] : (f(R_{РПЛ}) = W_{РПЛ}, W_{РПЛ} \in \Omega_{РПЛ}) \Rightarrow F_{РПЛ}\}|_{Z_{ОГР}}, \quad (3)$$

где $F_{РПЛ}$ – целевая функция распределения приложения (например, повышение производительности, масштабирование параллелизма, эффективное использование ресурсов), т.е. для всех значений параметров $a_{РПЛ}$ приложения $A_{РПЛ}$ существует способ разделения приложения $R_{РПЛ}$, являющийся составной частью стратегии распределения приложений ξ, такой, что реализация этого способа $f(R_{РПЛ})$, представленного через параметры множества $\Omega_{РПЛ}$, отражающие признаки (функции), отличающие PIM-системы от классических КС, приводит к выполнению заданной целевой функции $F_{РПЛ}$ разделения приложения при ограничениях $Z_{ОРГ}$ на параметры PIM-системы.

4. Описание компонентов модели

Выделение признаков (функций), отличающих PIM-систему от КС с классической архитектурой (процедура R_F), при соответствующей целевой функции распределения $F_{РПЛ}$. В связи с тем, что эти параметры (признаки) рассматриваются по отношению к соответствующим параметрам КС с классической архитектурой, они определяются нечеткими значениями типа “в пределах“, “больше”, “больше или равно“ и др., характерными для нечетких множеств, причем некоторые из параметров могут быть определены переменными типа “повышение”, “расширение”, “упрощение” и др., что также характерно для нечетких множеств. Поэтому мы считаем целесообразным для выполнения процедуры R_F использовать таблицы параметров разработанной ранее авторами модели PIM-системы [17], основанной на использовании теории нечетких множеств и теории гранулирования.

Если при распределении приложения ставится функция цели $F_{РПЛ}$ – повышение производительности системы (этот параметр в [17] обозначен β), то для обеспечения высокой производительности PIM-система должна иметь набор средств и способов, обеспечивающих соответствующие параметры (признаки, функции), отличающие её от КС с классической архитектурой (табл. 1).

Соответствующая гранула при этом будет иметь вид

$$\Gamma(\beta) = (s_1, s_4, s_5, w_1, w_6, w_{10}, c_1, c_2, c_8, p_8, p_{11}, a_1, a_3, a_5, a_6), \quad (4)$$

где $s_j \in S$, $w_q \in W$, $c_h \in C$, $p_r \in P$, $a_i \in A$, при этом S – структурно-архитектурный срез модели PIM-системы [17], W – срез программного обеспечения, C – срез управления процессами внутри

чипа, P – пользовательский срез, A – алгоритмический срез, j,q,h,r,i,k определяют номера элементов множеств, а их количество определяет мощности $m_j, m_q, m_h, m_r, m_i, m_k$.

Если же ставится в качестве функции цели F_{PIJI} – масштабирование параллелизма (этот параметр в [17] обозначен b), то PIM-система должна иметь соответствующий набор средств и способов, отличающих её от КС с классической архитектурой.

Таблица 1. Набор средств и способов, обеспечивающих сверхвысокую производительность, отличающих PIM-систему от КС с классической архитектурой

Обознач. парамет-ров	Средства и способы, обеспечивающие признаки (функции), отличающие PIM-систему от КС с классической архитектурой
s_1	Широкая полоса пропускания по каналу процессор-память (существенно больше по сравнению с классическими КС)
s_4	В качестве ВП в составе PIM-чипа используются как серийные, так и специализированные процессоры
s_5	В качестве средств коммутации внутри чипа используются скоростные коммутаторы (селекторы выбора), а между чипами – высокоскоростная межчиповая сеть
w_1	Модифицированная операционная система (OC_1), например, модифицированная Unix, Linux и т.п.
w_6	Программные средства для поддержки работы межчиповой коммутационной сети – МЧКП (ПО/МЧКП) и иерархической системы памяти (ПО/ПАМ) в различных режимах
w_{10}	Набор системных сервисных программ – ПО разбиения алгоритма на гранулы трех уровней: системного, узлового и блочного
c_1	Управление процессами PIM-системы основано на концепции потоковой обработки информации и управляемого сообщением вычисления
c_2	Поддержка управляемого сообщением вычисления основана на концепции пакетов, которые содержат значения параметров и спецификаторы действий, а также дополнительные поля, необходимые для транспортировки, обнаружения ошибок, маршрутизации и управления контекстом
c_8	Используется мультипроцессирование для быстрого удовлетворения входящих запросов на обслуживание и обеспечение перекрытия по времени выполнения вычислений, коммуникаций и доступа к памяти
p_8	PIM интегрируются в очень больших количествах, обеспечивая большую возможность параллелизма и распределенных вычислений, чем их традиционные кластерные копии
p_{11}	Эффективность коммуникации делает системы PIM лучшими для мелкомодульного распределенного вычисления, чем их традиционные кластерные копии
a_1	Возможность разделения реализуемого алгоритма на три уровня: системный, узловой и блочный
a_3	Наличие часто повторяющихся участков алгоритма, реализация которых требует максимальных ресурсов памяти и производительности
a_5	Высокая степень параллелизма алгоритма решаемой задачи с помощью ресурсов одного кристалла (чипа)
a_6	Масштабирование параллелизма за счет наращивания PIM-чипов (до нескольких тысяч чипов)

При этом гранула будет иметь вид

$$\Gamma(b) = (s_1, w_{12}, c_1, p_8, p_{11}, a_1, a_2, a_3, a_4, a_6). \quad (5)$$

Аналогично могут быть получены гранулы для других значений функций цели.

Необходимо отметить, что формирование такого типа гранул при определенных целевых установках и манипуляция с ними дает возможность при отсутствии прототипа предварительно оценить требуемую конфигурацию, свойства и функции PIM-системы, для которой в дальнейшем формируются стратегия, алгоритм разделения и размещения приложения по процессорам. Например, если поставить задачу повышения быстродействия только за счет масштабирования параллелизма, то эквивалентная гранула, согласно (4) и (5), будет иметь вид

$$\Gamma(\beta / b) = \Gamma(\beta) \cap \Gamma(b) = (s_1, c_1, p_8, p_{11}, a_1, a_3, a_6). \quad (6)$$

В соответствии с этим PIM-система (по сравнению с КС с классической архитектурой) должна иметь (табл. 1):

– широкую полосу пропускания по каналу процессор-память, существенно большую по сравнению с классическими КС (параметр s_1);

– потоковую обработку информации и управляемое сообщением вычисление (параметр c_1);

– интеграцию PIM в очень больших количествах, обеспечивая большую возможность параллелизма и распределенных вычислений, чем их традиционные кластерные копии (параметр p_8);

– более мелкомодульное распределенное вычисление, чем в традиционных кластерных копиях, и эффективную коммуникацию (параметр p_{11});

– возможность разделения реализуемого алгоритма на три уровня: системный, узловой и блочный (параметр a_1);

– наличие часто повторяющихся участков алгоритма, реализация которых требует максимальных ресурсов памяти и производительности (параметр a_3);

– масштабирование параллелизма за счет наращивания до нескольких тысяч PIM-чипов (параметр a_6).

Выбор способа распараллеливания вычислительных алгоритмов (процедура $B_{РПЛ}$). Выбор способа распараллеливания вычислений основывается на анализе вычислительной схемы решения исходной задачи. Требования, которым должен удовлетворять выбираемый подход, обычно состоят в обеспечении равного объема вычислений в выделяемых подзадачах и минимума информационных зависимостей между этими подзадачами (при прочих равных условиях нужно отдавать предпочтение редким операциям передачи сообщений большего размера по сравнению с частыми пересылками данных небольшого объема).

Для большого класса задач вычисления сводятся к выполнению однотипной обработки большого набора данных. К такому классу задач относятся, например, матричные вычисления, численные методы решения уравнений в частных производных и др. В этом случае говорят, что существует параллелизм по данным, и выделение подзадач сводится к разделению имеющихся данных [15, 16, 18]. Для другой части задач вычисления могут состоять в выполнении разных операций над одним и тем же набором данных. В этом случае говорят о существовании функционального параллелизма.

Важный вопрос при выделении подзадач состоит в выборе нужного уровня декомпозиции вычислений. Формирование максимально возможного количества подзадач обеспечивает использование предельно достижимого уровня параллелизма решаемой задачи, но затрудняет анализ параллельных вычислений. Применение при декомпозиции вычислений только достаточно "крупных" подзадач может затруднить эффективное использование достаточно большого количества процессоров. Возможное разумное сочетание этих двух подходов может состоять в применении в качестве конструктивных элементов декомпозиции только тех подзадач, для которых

методы параллельных вычислений являются известными. Выбираемые подзадачи при таком подходе считаются базовыми, так как не допускают дальнейшего разделения, или составными в противном случае.

При применении параллельных алгоритмов решения сложных задач принципиальным моментом является анализ эффективности использования параллелизма, состоящий обычно в оценке получаемого ускорения процесса вычислений (сокращения времени решения задачи). Формирование подобных оценок ускорения может осуществляться применительно к выбранному вычислительному алгоритму (оценка эффективности распараллеливания конкретного алгоритма).

Ускорение, получаемое при использовании параллельного алгоритма для p процессоров, по сравнению с последовательным вариантом выполнения вычислений, определяется [15, 16]:

$$S_p(n) = T_1(n) / T_p(n),$$

т.е. как отношение времени решения задач на скалярной ЭВМ к времени выполнения параллельного алгоритма (величина n используется для параметризации вычислительной сложности решаемой задачи и может пониматься, например, как количество входных данных задачи).

Эффективность использования параллельным алгоритмом процессоров при решении задачи определяется соотношением

$$E_p(n) = T_1(n) / pT_p(n) = S_p(n) / p .$$

(Величина эффективности определяет среднюю долю времени выполнения алгоритма, в течение которой процессоры реально используются для решения задачи.) Как следует из приведенных соотношений, в наилучшем случае получим

$$S_p(n) = p \text{ и } E_p(n) = 1.$$

В [15, 16, 18] приведены выражения для оценки параметров ускорения и эффективности применения различных способов распараллеливания для некоторых типов распространенных приложений. Используя эту информацию, можно определить эффективность использования соответствующего способа распараллеливания алгоритма и соответственно ускорение вычислений по сравнению с последовательным вариантом вычислений, что в конечном итоге позволяет выполнить балансировку загрузки процессоров. При этом объем вычислений для каждого используемого процессора должен быть примерно одинаков, а распределение подзадач между процессорами должно быть выполнено таким образом, чтобы количество информационных связей (коммуникационных взаимодействий) между подзадачами было минимальным.

Формирование таблиц соответствия набора операций приложения и набора команд процессоров PIM-системы (процедура R_{TC}). Как было отмечено во введении, каждый чип PIM-системы (в наиболее распространенном варианте архитектуры) содержит один ВП и множество ПЯ (рис. 1), при этом функциональные возможности ПЯ, как правило, существенно ниже функциональных возможностей ВП. Это продиктовано тем, что вся область подложки одного кристалла разделена между массивом памяти и процессорными элементами, и чтобы разместить как можно больше процессорных элементов, они по своим схемотехническим решениям должны быть как можно проще и тем самым – с ограниченным набором команд с ориентацией на

выполнение простых, но наиболее массовых для реализуемого алгоритма операций. ВП по своему функциональному назначению должен иметь более мощную систему команд по сравнению с ПЯ, и поэтому в качестве ВП может быть применен аналог одного из типов коммерческих процессоров.

В связи с этим, если решать задачу распределения приложений по процессорам, то для PIM-систем, в отличие от кластерных систем, где наборы команд всех процессоров практически можно рассматривать универсальными, разделение приложений на части и распределение этих частей по процессорам необходимо выполнять при условии соответствия набора операций выделенной части приложения $(H_{ОП})_{ПРЛ^*}$ набору команд $H_к(ПЯ)$ или $H_к(ВП)$ при отображении их через набор операций $H_к(ПЯ) := H_{ОП}(ПЯ)$ или $H_к(ВП) := H_{ОП}(ВП)$, к которому эта часть приложения намечается для реализации, т.е.

$$H_{ОП}(ПЯ) \subseteq (H_{ОП})_{ПРЛ^*}, \quad H_{ОП}(ВП) \subseteq (H_{ОП})_{ПРЛ^*}.$$

При этом мощности m множеств этих наборов должны соответствовать условиям

$$m_{(ПЯ)} \geq m_{(ПРЛ)^*}, \; m_{(ВП)} \geq m_{(ПРЛ)^*}, \tag{7}$$

а частота встречаемости f набора операций приложения, соответствующих набору операций ПЯ, должна быть наибольшей по сравнению с частотой встречаемости других операций, принадлежащих ВП и хост-машине:

$$f_{ПЯ} >> f_{ВП} >> f_{хост}, \quad f_{ПЯ} = f_{макс}. \tag{8}$$

Результаты анализа приложения представляются в виде таблицы, содержащей коды операций, которые сравниваются с кодами операций ПЯ и ВП. Результаты сравнения затем представляются в виде таблицы соответствия, по которой проверяется выполнение условия (7), а также в виде таблицы частот встречаемости операций, по которой проверяется выполнение условий (8). Это позволяет предварительно оценить возможность распределения приложения по процессорам PIM-системы. В [19] авторами изложены основные положения концепции и принципы построения форматов и набора команд гипотетической PIM-системы, которые могут быть использованы при выполнении такого анализа.

Проверка баланса загрузки процессоров PIM-системы (процедура $B_{ЗГ}$). Наиболее объективным вариантом проверки баланса загрузки является запуск программы приложения после её распределения по процессорам и определение времени выполнения распределенных частей приложения каждым процессором. При этом баланс загрузки определяется как между ВП и ПЯ, так и между каждым ПЯ, принимающим участие в реализации приложения. Такая процедура может выполняться как при реализации конкретных этапов, так и по завершении разделения, и поэтому более подробно её целесообразно рассмотреть в рамках описания процесса разделения приложения.

5. Основные положения стратегии разделения приложений для интеллектуальной памяти распределенных компьютерных систем

Основные положения стратегии разделения приложений исходят из сущностей предлагаемых выше моделей и их компонентов и сформулированы следующим образом:

1) Отображение аппаратно-программного базиса построения конфигурации PIM-системы (при отсутствии прототипа), для которой формируется стратегия, алгоритм разделения и размещение приложения по процессорам осуществляются с помощью набора объектов, признаков, свойств и процедур, отличающих PIM-систему от КС с классической архитектурой, которые определяются функционально-структурной гранулой, полученной на основе математической модели PIM-системы при заданной целевой функции её архитектурно-структурной организации.

2) Разделение приложений выполняется на трех уровнях (системном, узловом и блочном) в соответствии с многоуровневой организацией PIM-системы (рис. 1). При этом архитектурно-структурная интерпретация многоуровневой модели распределения приложения должна отражать особенности реализации приложения на каждом уровне.

На верхнем (системном) уровне модель представляется в виде “хост-машина – множество чипов (ВП)”, на среднем (узловом уровне): “один ВП – эквивалентное ПЯ”, где эквивалентное ПЯ содержит систему команд одного ПЯ и эквивалентную емкость памяти, равную сумме емкостей банков памяти, размещенных на чипе. При этом сокращение времени реализации за счет распараллеливания фрагментов приложения на все ПЯ одного кристалла учитывается путем умножения тактовой частоты работы ПЯ на величину, равную количеству параллельно работающих ПЯ.

На нижнем (блочном уровне) модель распределения представляется в виде “линейка однородных ПЯ”.

3) Исходное разделение приложения реализуется по циклам, которые могут быть вложенными, где каждый уровень вложенности может содержать только один цикл и несколько операторов.

4) Дальнейшее разделение приложения и наполнение операторами выделенных в результате исходного разделения основных модулей $M_{осн}$ выполняется по принципу операторной зависимости (связности) по данным.

5) Формирование составных $(M_{сост})$ модулей из модулей $M_{осн}$ малой величины, а также объединение некоторых основных модулей $M_{осн}$ с соседними операторами, не входящими в выделенные циклы, выполняется по принципу операторной зависимости (связности) по данным.

6) Принятие решения об объединении конкретных модулей, а также модулей с операторами, не входящими в выделенные циклы, и их распределение по процессорам осуществляется по принципам оценки “параметрического веса” модуля, который определяется временем выполнения модуля на конкретном процессоре. Например, если модуль реализуется на ВП быстрее, чем на ПЯ, то его “параметрический вес” ниже, и в дальнейшем он может быть приписан для реализации на ВП. “Параметрический вес” (родственность) модуля оценивается при использовании модели статических параметров.

7) При разделении приложения в соответствии с моделью верхнего уровня предпочтение отдается разделению на крупные фрагменты, где каждый фрагмент предназначается для самостоятельной реализации на отдельном чипе (узле). Следует стремиться, чтобы взаимосвязь по данным между этими фрагментами осуществлялась в основном итоговыми значениями результатов их реализации.

8) При разделении выделенной части (фрагмента) приложения между ВП и ПЯ конкретного чипа (узловой уровень) необходимо учитывать ограничения на функциональные возможности (ограниченный набор команд) ПЯ, что приводит к необходимости выявления соответствия между набором операций, требуемых для реализации со стороны выделенного фрагмента приложения, и набором операций (системы команд) ПЯ и ВП, а также учет частоты встречаемости в реализуемом алгоритме этих операций.

9) При разделении части приложения, выделенного для реализации на ПЯ (блочный уровень), необходимо учитывать ограничения на емкость каждого банка памяти, подключенного к соответствующему ПЯ, который предназначается как для хранения атрибутов ЯОС (ядра операционной системы ПЯ) и соответствующих служб, так и для хранения данных. Это приводит к необходимости разделения выделенного фрагмента приложения на более мелкие части и распределения их по ПЯ.

10) Оценка загрузки каждого процессора PIM-системы и установление баланса загрузки между всеми процессорами выполняется путем сравнения “параметрических весов” и перераспределения модулей и операторов к модулям, используя принцип связности по данным.

11) Целесообразно оценить возможность использования на любом уровне разделения приложения известных способов распараллеливания задач (например, способами распараллеливания при умножении матриц, операций на графе и др.) и определение эффективности применения этих способов по соответствующим аналитическим выражениям (например, согласно [15, 16, 18]).

На основе приведенных положений можно представить укрупненную схему стратегии распределения приложений (рис. 2). При этом на верхнем (системном) уровне алгоритм приложения A разделяется на фрагменты $A \supset (A_1, A_2, ..., A_i, ..., A_N)$, каждый из которых приписывается для реализации на соответствующем чипе.

На среднем (узловом) уровне каждый фрагмент, например, A_i, согласно модели распределения на данном уровне, разделяется на часть фрагмента $A_{i(ВП)}$, которая приписывается для реализации на соответствующем ВП, и на часть фрагмента $A_{i(ПЯ*)}$, которая приписывается для реализации на эквивалентном $ПЯ*$, объединяющем память всех ПЯ, находящихся на данном чипе.

И, наконец, на нижнем (блочном) уровне часть фрагмента приложения $A_{i(ПЯ*)}$, согласно модели распределения на этом уровне, разделяется на более мелкие части $A_{i(ПЯ*)} \supset (a_1, a_2, ..., a_j, ..., a_k)$, которые распределяются для реализации по всем $ПЯ$, размещенным на данном чипе.

Как видно из рис. 2, состав и последовательность процедур на каждом уровне разбиения приложения фактически одинаковые. Отличие состоит в некоторых исходных данных и ограничениях на параметры системы, которые необходимо учитывать на соответствующих уровнях разделения приложения. Это создает определенные предпосылки для построения ограниченного набора программных блоков (модулей) и, что очень важно, для создания ограниченного набора

достаточно простых аппаратных средств, необходимых для поддержки реализации отдельных процедур стратегии (рис. 3).

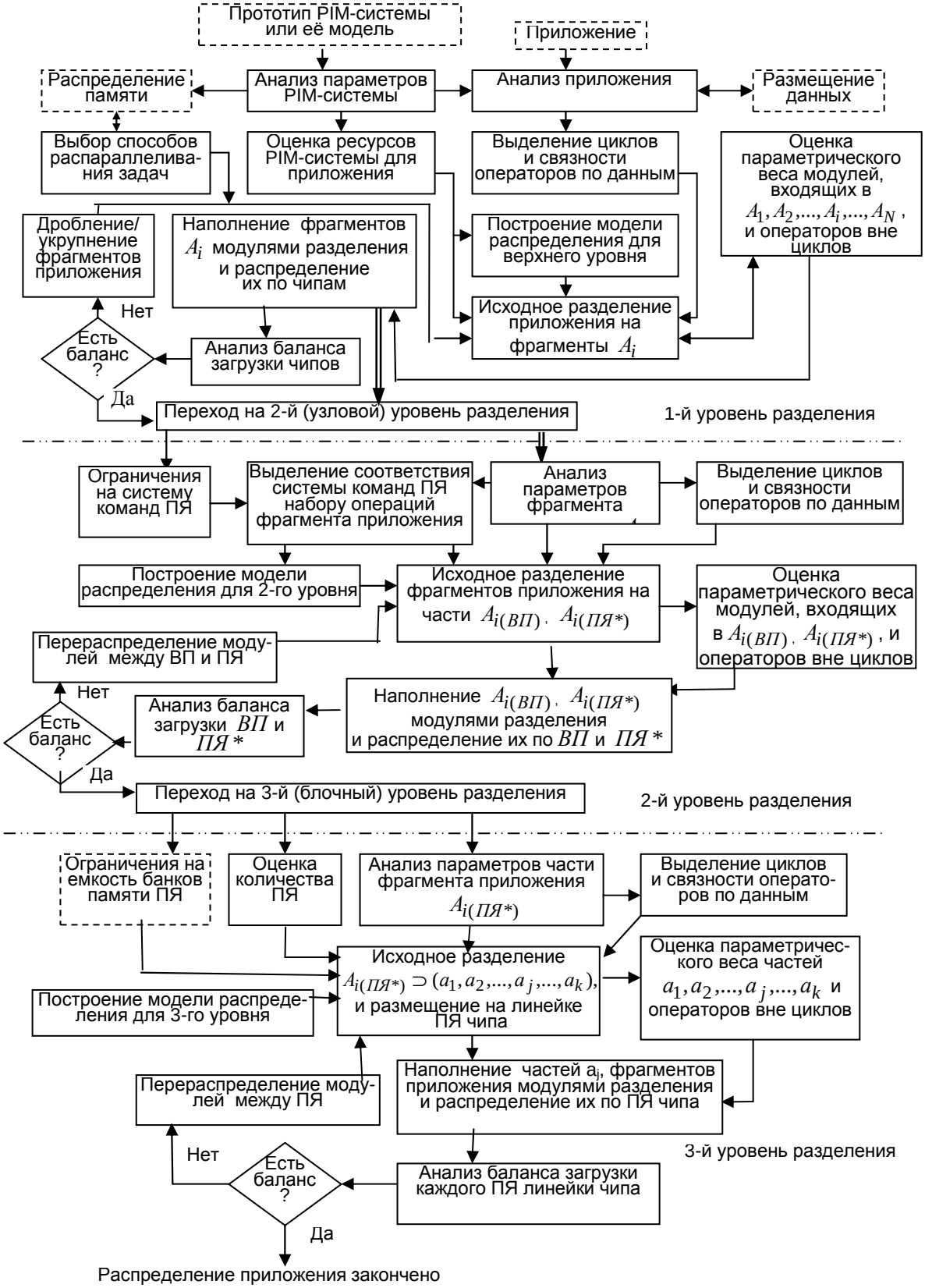
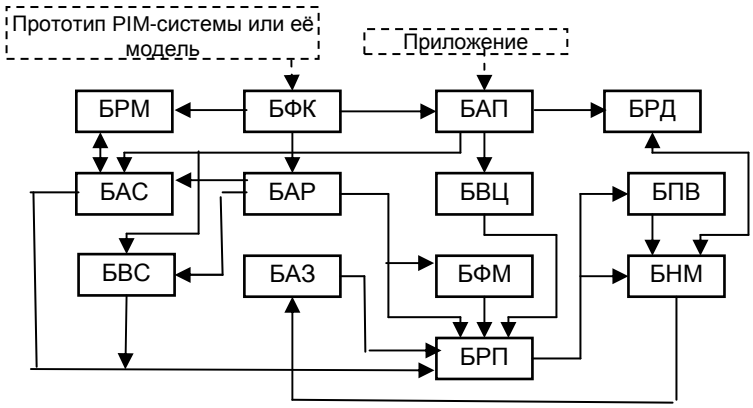


Рис. 2. Вариант блок-схемы многоуровневой стратегии разделения приложений для PIM-системы



Обозначения:
БФК – блок формирования конфигурации и определения параметров PIM-системы.
БАП – блок анализа приложения и определения его характеристик (наличие циклов и других параллельных участков, набор операций, частота встречаемости операций и др.).
БВЦ – блок выделения циклов из приложения.
БАР – блок анализа ресурсов, необходимых для распределения приложений на каждом уровне.
БАС – блок анализа способов распараллеливания задач на многопроцессорной системе с функциями выбора и определения эффективности способа применительно к конкретному алгоритму реализации приложения.
БПВ – блок расчета параметрических весов модулей и операторов с функциями их сравнения.
БФМ – блок формирования моделей распределения приложений по уровням.
БРП – блок исходного разделения приложения в соответствии с моделями распределения на каждом уровне.
БНМ – блок наполнения основных и составных модулей, выделенных на этапе исходного разделения приложения.
БАЗ – блок анализа и корректировки баланса загрузки процессоров PIM-системы.
БРМ – блок распределения памяти.
БРД – блок распределения данных.
БВС – блок выделения соответствия системы команд процессоров набору операций приложения.

Рис. 3. Укрупненная блок-схема аппаратно-программной среды реализации стратегии разделения приложений для PIM-систем

6. Выводы

В сложных компьютерных системах типа “процессор-в-памяти” (PIM-системы), содержащих огромное количество процессорных элементов (ядер), большое значение для получения высокой производительности приобретают оптимальное разделение приложений и размещение полученных частей по процессорам. Специфика архитектурно-структурной организации PIM-систем наложила свои отпечатки на процедуру распределения приложений, которая, вследствие этого, является многоуровневой. В соответствии с этим предложенная в статье модель стратегии и процедуры распределения приложений основываются на разработанной ранее многоуровневой модели PIM-системы, которая по своей сущности содержит признаки (свойства), отличающие PIM-систему от КС с классической архитектурой [17]. Применение такой модели распределения приложений при заданной целевой функции распределения с помощью процедуры формирования соответствующей гранулы, используя теорию нечетких множеств и теорию гранулирования, позволяет получить конфигурацию PIM-системы, которая может быть исходной для распределения приложений.

Исследования параметров модели распределения позволили сформулировать процедуры стратегии распределения приложения, на основе которых построена блок-схема стратегии, а также блок-схема аппаратно-программных средств для поддержки её практической реализации. Естественно, что функции некоторых блоков этой схемы могут быть реализованы с помощью

имеющихся программных средств (например, функции выявления циклов в приложении и их маркировка). Однако основной набор блоков отражает специфику архитектуры PIM-системы, и реализация их основных функций просматривается в приведенном наборе процедур стратегии распределения.

Таким образом, рассмотренные в статье вопросы, имеющие непосредственное отношение к распределению приложений в сложных компьютерных системах типа PIM, перекрывают широкий спектр задач, начиная от построения модели разделения приложения и описания её компонентов и заканчивая набором сформулированных процедур распределения приложения с отражением блок-схемы стратегии распределения и блок-схемы аппаратно-программной среды, поддерживающей реализацию этих процедур.

СПИСОК ЛИТЕРАТУРЫ

1. Архитектурно-структурная организация компьютерных средств класса “Процессор-в-памяти” / А.В. Палагин, Ю.С. Яковлев, Б.М. Тихонов и др. // Математичні машини і системи. – 2005. – № 3.– С. 3 – 16.
2. Елисеева Е.В., Яковлев Ю.С. О концепции построения программной среды PIM-систем // Управляющие системы и машины. – 2008. – № 4. – С. 58 – 67.
3. Олифер В.Г., Олифер Н.А. Сетевые операционные системы: Учебник для ВУЗов. – СПб.: ПИТЕР, 2003. – 544 с.
4. Зарудный Д.И. Управление распределенными ресурсами в ОС UNIX. – М.: Московский государственный институт электроники и математики, 2005.– 50 с.
5. Rezaei M. 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.
6. Пат. 5.701.503, G06F 012/00. Method and apparatus for transferring information between a processor and a memory system: United States Patent. Singh; Gurbir; Wang; Wen-Hann; Rhodehamel; W. Michael; Bauer; M. John; Sarangdhar; V. Nitin; Опубл. 23.12.97. – 24 с.
7. Пат. 5.117.350, G06F 012/06; G06F 015/16. Memory address mechanism in a distributed memory architecture. United States Patent. Parrish; C. Osey; Jr. Peiffer; E. Robert; Thomas; H. James; Jr. Hilpert; J. Edwin; Опубл. 26.05.92.
8. Управление распределенными ресурсами. –
http://doc.trecom.tomsk.su/citforum.ru/win/operating_systems/sos/glava_11.shtml.
9. Яковлев Ю.С., Тихонов Б.М. О распределении памяти в компьютерных системах // Управляющие системы и машины. – 2006. – № 5. – С. 40 – 47.
10. Яковлев Ю.С., Тихонов Б.М. Об оптимизации размещения данных в PIM-системе // Математичні машини і системи. – 2006. – № 3. – С. 24 – 35.
11. Яковлев Ю.С., Елисеева Е.В. Основные задачи и методы реализации функций управления памятью в PIM-системах // Математичні машини і системи. – 2008. – № 2. – С. 47 – 62.
12. Parallelizing A. Framework for Intelligent Memory Architectures; Tsung-Chuan Huang, Slo-Li Chu // Proc. of The Seventh Workshop on Compiler Techniques for High-Performance Computing (CTHPC 2001). – Hsinchu, Taiwan, 2001. – Ar. 15-16. – P. 96 – 10. – http://parallel.iis.sinica.edu.tw/cthpc/7th/03CTHPC2001-Hardcopy.PDF.
13. Exploiting Application Parallelism for Processor-in-Memory Architecture; Slo-Li Chu, Tsung-Chuan Huang // Proc. of 2003 National Computer Symposium. – Taichung, Taiwan, 2003. – Dec 18–19. – P. 2293 – 2303. – http://dspace.lib.fcu.edu.tw/bitstream/2377/564/1/OT_1022003305.pdf.
14. Solihin Ya. Improving memory performance using intelligent memory. THESIS of Doctor of Philosophy in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 2002. – http//portal.acm.org/citation.cfm?id=936938&coll=&dl=&CFID=15151515&CFTOKEN=6184618
15. Гергель В.П., Стронгин Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных систем: Учебное пособие. – 2-е изд., доп. – Нижний Новгород: Издательство Нижегородского госуниверситета, 2003. – 82 с.
16. Гергель В.П. Теория и практика параллельных вычислений: Учеб. пособие. – М.: Интернет-университет информационных технологий, 2007. – 423 с.
17. Елисеева Е.В., Яковлев Ю.С. Математическая модель функциональной среды PIM-системы на основе теории нечетких множеств и теории гранулирования // Математичні машини і системи. – 2009. – №1.– С.40–54.
18. Ефимов С.С. Обзор методов распараллеливания алгоритмов решения некоторых задач вычислительной дискретной математики // Математические структуры и моделирование. – 2007. – Вып. 17.– С. 72 – 93. –
http://www.parallel.ru/tech/tech_dev/par_libs.html.
19. Елисеева Е.В., Яковлев Ю.С. Концепция и принципы построения форматов и набора команд гипотетической PIM-системы // Комп’ютерні засоби, мережі та системи: Зб. наук. праць. – 2008. – № 7. – С. 39 – 47.

Стаття надійшла до редакції 26.07.2009