УДК 004.27; 004.25; 004.382.2

Ю.С. ЯКОВЛЕВ, Е.В.ЕЛИСЕЕВА

Институт кибернетики имени В.М. Глушкова НАН Украины, Киев

О РЕАЛИЗАЦИИ РАСПРЕДЕЛЕНИЯ ПРИЛОЖЕНИЯ ДЛЯ ПАРАЛЛЕЛЬНОГО ВЫПОЛНЕНИЯ НА PIM-СИСТЕМЕ

Аннотация. Предложены эффективный алгоритм и непосредственно на языке С++ программа реализации алгоритма распределения фрагмента приложения между процессорами распределенной гетерогенной компьютерной системы типа PIM, которые в максимальной степени учитывают особенности архитектурно-структурной организации систем такого класса и тем самым обеспечивают быструю сходимость и сокращение времени реализации алгоритма распределения.
Ключевые слова: PIM- система, алгоритм распределения приложения, параллельная обработка, система команд.
Анотація. Запропоновані ефективний алгоритм і безпосередньо на мові С++ програма реалізації алгоритму розподілу фрагменту програми користувача між процесорами розподіленої гетерогенної комп'ютерної системи типу PIM, які в максимальному ступені враховують особливості архітектурно-структурної організації систем такого класу і тим самим забезпечують швидку збіжність і скорочення часу реалізації алгоритму розподілу.
Ключові слова: PIM-система, алгоритм розподілення програми користувача, паралельна обробка, система команд.
The abstract. Are offered effective algorithm and it is direct in language C ++ the program of implementation of a scheduling algorithm of applications between processors of the distributed heterogeneous computer system of type PIM, which in the maximum extent consider features of the is architectural-structural organisation of systems of PIM-system and by that provides sweeping convergence and abbreviation of a time of implementation of a scheduling algorithm.
Keywords: PIM-system, the application scheduling algorithm, parallel machining, the system of commands.

Введение

История развития средств вычислительной техники показала, что с точки зрения повышения производительности эффект использования новых архитектурно-структурных решений и новых методов распараллеливания приложений существенно превосходит эффект, который может быть получен путем применения новой элементной базы. Достижения интегральной технологии сделали возможным появление новых классов архитектур типа “Процессор-в-памяти” (PIM), которые реализованы на одном кристалле (чипе) и по сравнению с КС с классической архитектурой при одинаковых ресурсах процессоров и памяти обладают более высокими параметрами производительности при улучшенных значениях других пользовательских характеристик (габаритов, веса и др.), а также возможностями решать сложные и трудоемкие задачи, которые плохо поддаются решению или вообще не могут быть решены с помощью классических КС [1]. PIM-система по своим принципам построения имеет ряд особенностей: она образует иерархическую гетерогенную многопроцессорную среду, включающую ведущий процессор (ВП) со своей памятью и соединенные с ВП процессорные ядра (ПЯ), подключенные каждый к своему банку памяти. При этом ВП имеет развитую систему команд, широкие функциональные возможности и является более мощным, по сравнению с каждым ПЯ, который представлен в виде упрощенного процессора с сокращенной системой команд и поэтому является функционально ограниченным и менее мощным, но имеет малое время доступа к своему банку памяти.

Актуальность

Известные способы распределения приложений для PIM-систем [2, 3] основаны на упрощенной модели распределения, и соответственно процесс распределения реализуют только между хост-машиной и одним ВП. При этом в качестве основной единицы при анализе и разделении исходной программы пользователя используют операторы в цикле (применен операторный метод распределения), из-за сложности которого рассматривают только операторы внутри циклов, и обнаруживают зависимости по данным только для некоторых конструкций программы пользователя (для идеально вложенных циклов), при этом другие конструкции не рассматривают, что указывает на ограниченные функциональные возможности способов. Несмотря на эти упрощения, процесс распределения приложения является трудоемким и занимает большое количество времени, которое может существенно превышать время непосредственного решения задачи пользователя, увеличивая тем самым энергозатраты и стоимость эксплуатации системы в целом. Поэтому поиск нового решения проблемы распределения приложений, которое устранило бы недостатки известных способов распределения, является задачей актуальной.

Цель

Создать эффективный алгоритм и непосредственно на языке С++ программу реализации фрагмента алгоритма распределения приложений между процессорами распределенной гетерогенной компьютерной системы типа PIM, которые в максимальной степени учитывают особенности архитектурно-структурной организации систем такого класса и тем самым обеспечивает быструю сходимость и сокращение времени реализации алгоритма распределения.

Постановка задач

1). Исследовать существующие алгоритмы распределения приложений для PIM-систем и выполнить анализ их недостатков.

2). Исследовать существующие критерии распределения приложения и предложить новые, учитывающие особенности архитектурно-структурной организации PIM-систем.

2). Разработать алгоритм программы распределения приложений для параллельной реализации на PIM-системе с учетом особенностей архитектурно-структурной организации этого класса машин и выполнить его описание (комментарии).

3). Разработать программу реализации предложенного алгоритма распределения приложения по процессорам PIM-системы с использованием языка программирования C++.

Решение задач

В основу решения задачи по созданию алгоритма распределения приложения положена многоуровневая стратегия распределения, которая применительно к PIM-системе на одном кристалле содержит два уровня: сначала внутри чипа разделяют фрагмент алгоритма приложения на блоки и распределяют их внутри чипа по процессорам. Этот фрагмент алгоритма был приписан данному чипу на предыдущем уровне распределения (между хост-машиной и чипом), который здесь не рассматривается. Распределение приложения на программные блоки выполняют согласно модели распределения – между ВП и эквивалентным процессорным ядром (ПЯ*), параметры которого формируют следующим образом: за систему команд ПЯ* принимают систему команд одного ПЯ, так как все ПЯ на чипе одинаковые, величину емкости памяти ПЯ* принимают равной величине емкости всей памяти, подключенной ко всем ПЯ на этом чипе, а время реализации команды ПЯ* принимают равной времени реализации одного ПЯ, поделенного на количество ПЯ, входящих в состав ПЯ*, учитывая, что все ПЯ одинаковые и работают параллельно. Тем самым на этом уровне создают модель стратегии распределения программ пользователя из двух элементов (ВП и ПЯ*), что упрощает процесс распределения и не приводит к значительным ошибкам, поскольку на этом уровне выполняют проверку баланса загрузки процессоров ВП и ПЯ* и при необходимости из-за отсутствия баланса выполняют корректировку распределения. При этом в качестве основного критерия распределения каждого фрагмента программы пользователя на этом уровне (внутри одного чипа) принимают соответствие систем команд ВП и ПЯ* набору операций фрагмента алгоритма, который был приписан этому чипу при распределении всего приложения между хост-машиной и чипом.

На следующем уровне выполняют распределение программных блоков на модули между всеми ПЯ, входящими в набор ПЯ* одного чипа. При этом, так как все ПЯ в наборе ПЯ* одинаковые, то за основной критерий распределения модулей принимают рассчитанные их параметрические веса (времена их реализации) и соответствующие связи по данным. По значениям параметрических весов оценивают баланс загрузки модулями каждого ПЯ внутри чипа, и при отсутствии баланса выполняют перераспределение модулей между ПЯ, входящих в состав ПЯ*, для которых баланс не выполняется.

По окончании распределения формируют так называемые волновые фронты (ВФ), при этом в состав каждого ВФ включают программные блоки, независимые по данным и которые на PIM-системе могут быть выполнены одновременно. Последовательность выполнения ВФ определяют в соответствие с последовательностью выполнения программных блоков в исходном фрагменте алгоритма приложения согласно связности этих блоков по данным. Результаты распределения представляют в виде соответствующих пакетов (файлов), которые направляют в память соответствующих ВП и ПЯ для параллельного выполнения распределенных блоков приложения внутри чипа.

Предложенная стратегия распределения основана на следующих принципах [4]**:**

1. Принцип соответствия набора операций фрагмента (блока) алгоритма приложения набору системы команд процессора, на котором этот блок должен быть выполнен.

2. Принцип соответствия структуры алгоритма распределения иерархической структуре имеющихся ресурсов PIM-системы.

3. Принцип допустимой замены одинаковых ресурсов множества процессорных элементов, ориентированных на параллельную работу, и модулей памяти на один эквивалентный комплексный ресурс, вложенный в ПЯ*, содержащий один процессор с системой команд одного ПЯ и временем такта его работы, равный времени такта работы одного ПЯ, поделенной на k, где k – количество ПЯ на кристалле (чипе), и с емкостью памяти равной сумме емкостей памяти всех ПЯ на этом кристалле.

4. Принцип проверки и корректировки баланса загрузки процессоров по специфическим критериям, в частности – по критерию параметрического веса.

Согласно сформулированными выше стратегией и основными принципами распределения, разработан алгоритм программы распределения приложения по процессорам PIM-системы, часть которого, отражающая распределение между ВП и эквивалентным ПЯ*, а также между множеством ПЯ, входящих в состав ПЯ*, приведена на рисунке 1. Описание уровней распределения приложений более подробно изложено в [5]. Для части алгоритма, отображенной на рисунке 1, разработана программа на C++, описание блоков которой приведено ниже.

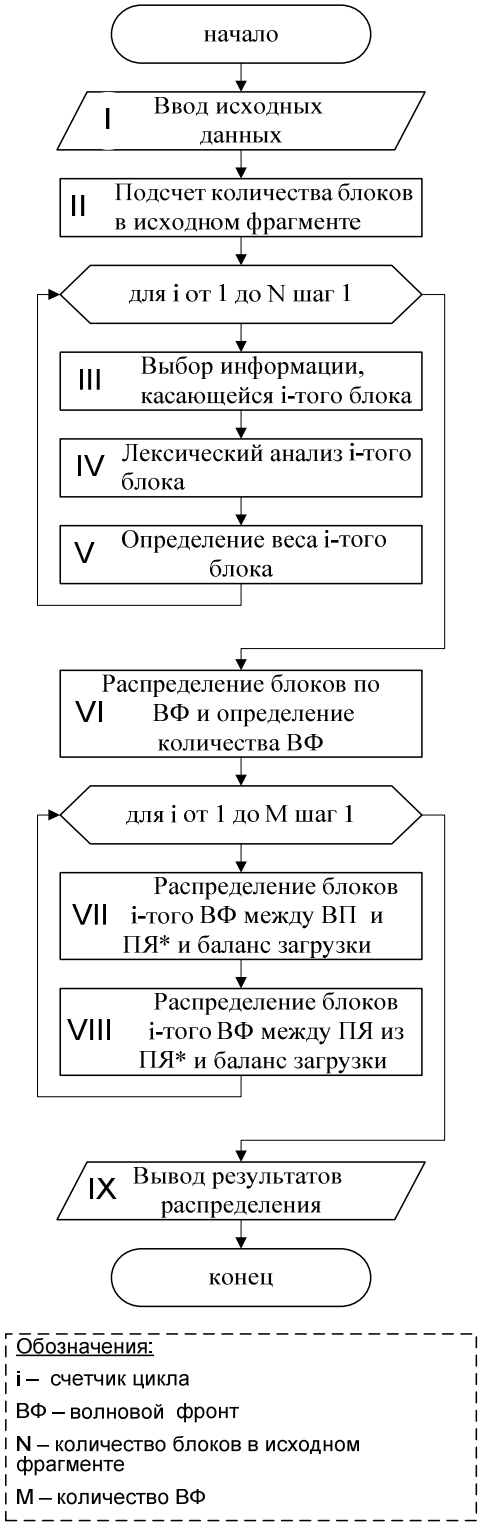

Рисунок 1 – Фрагмент алгоритма распределения приложения

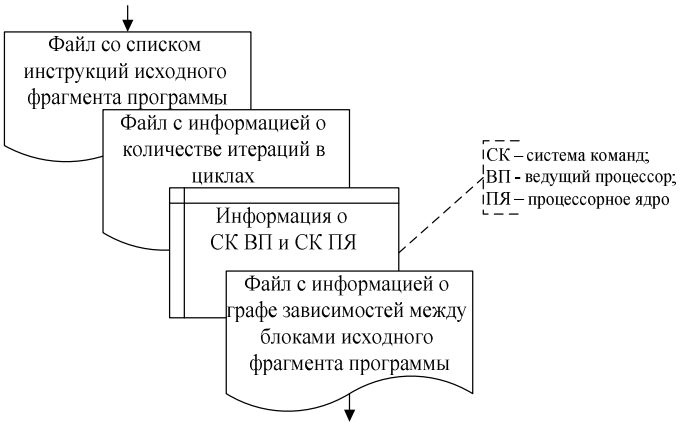

Рисунок 2 – Исходные данные

I. Исходные данные.

Блок-схема исходных данных приведена на рисунке 2. К исходным данным, которые необходимы для работы программы, относятся:

I.1. Список команд, входящих в исходный фрагмент программы. Представлен в текстовом файле в следующем виде: идентификатор блока 1, список команд входящих в блок 1, идентификатор блока 2, список команд блока 2 и т. д. При этом к фрагменту программы предъявляется ряд требований, в частности, фрагмент программы пользователя должен быть написан и откомпилирован на C++ и содержать только те команды, которые входят в систему команд (СК) ВП.

I.2. Файл с информацией о количестве итераций для каждого цикла, входящего в исходный фрагмент программы.

I.3. Описание системы команд ВП и ПЯ, хранится в оперативной памяти ВП в виде структур, состоящих из двух полей: код команды и вес команды для соответствующего процессора. Под весом команды для данного процессора понимается время выполнения этой команды на данном процессоре. Причем, предполагается, что система команд ПЯ является подмножеством системы команд ВП .

I.4. Файл с описанием графа зависимости между блоками фрагмента исходной программы. Структура файла следующая: идентификатор вершины, список дочерних вершин, идентификатор следующей вершины, список дочерних вершин и так далее. Пока не будут перечислены все вершины графа. Под вершинами графа здесь понимаются блоки, на которые разбит фрагмент исходной программы. Множество ребер графа - это зависимости между блоками.

II. Подсчет N -количества блоков, входящих в исходный фрагмент.

В данном блоке анализируют файл, содержащий список инструкций исходного фрагмента программы приложения и подсчитано число **N**-количество блоков, входящих в этот фрагмент.

III. Выбор информаии, касающейся I – ого блока.

Из файла I.1. (см. *исходные данные*) выбирают список инструкций, относящихся к i–тому блоку, а из файла I.2. выбирают информацию о количестве итераций для каждого цикла, входящего в состав i-того блока.

IV. Лексический анализ и получение списка токенов блока.

В процессе лексического анализа считывают исходный текст блока, распознают и выделяют лексемы (при этом убираются лишние пробелы и символы табуляции) и выдают последовательность токенов, которая используется в **V** для проверки сооветсвия команд, входящих в данный блок, системам команд ВП и ПЯ и дальнейшего определения возможности выполнения блока на ВП и ПЯ, а также подсчета веса блока для ВП и ПЯ.

Под лексемой здесь понимают последовательность допустимых символов языка С++, имеющая смысл для компилятора.

Токен представлен в виде структуры, которая содержит:

1) идентификатор класса токена (код токена);
2) лексему, выделенную в исходном тексте (необязательный параметр).

Во входном потоке будет существовать целый ряд лексем, для которых на выходе будет получен один и тот же токен. Этот набор лексем описывается правилом (шаблоном), связанным с токеном. В таблице 1 приведены примеры токенов, используемых при работе алгоритма и их описание.

Таблица 1 – Примеры токенов и их описание

Код токена	Неформальное описание шаблона	Примеры лексем
NUM	Любая целая числовая константа	15, 136, -12345
NUM1	Любая числовая константа с плавающей точкой	3.14, -23.002345671
ID	Идентификатор (произвольная последо-вательность латинских букв и цифр, начинающаяся с буквы)	for, sum1, X
COMM	Коментарий(любой набор символов, находящийся между /* и */, или все символы до конца строки, расположенные после //)	/* вычисление веса*/ // количество записей
LIT	ЛИТЕРАЛ (любые символ между парными кавычками или парными апострофами)	"S=%d" 'g' "house"
LOGIC	Логическая операция (&& или || или !)	&&, ||
REL	Отношение сравнения(== или > или < или >= или <= или !=)	>,>=
OP	Арифметическая операция (+ или - или * или / или % или INC или DEC)	+, DEC
BI	Побитовая операция ()	&, ^
…	…	…
ER	При выдедлении лексемы найдена ошибка	
NONE	Входая лексема не относится ни к одному из вышеперечисленных токенов	

V Определение веса блока. Вес каждого блока вычисляют в виде пары чисел ($W_{ВП}$, $W_{ПЯ}$), где $W_{ВП}$ – время выполнения блока на ВП, а $W_{ПЯ}$ – время выполнения блока на ПЯ, при этом если значение $W_{ПЯ}$ – окажется равным -1, то выполнение данного блока на ПЯ невозможно.

В процессе анализа списка токенов поочередно выделяются команды и записываются в таблицу команд блока. Каждая запись данной таблицы состоит из следующих полей:

– код команды;
– вес команды для ВП;
– вес команды для ПЯ;
– L (число, показывающее, сколько раз данная команда будет выполнена в блоке).

Изначально последние три поля таблицы заполнены 0. Для каждой команды проверяется входит ли данная команда в СК ПЯ. Реализация функции для такой проверки представлена на рисунке 3. Эта функция, возвращает значение веса для ПЯ рассматриваемой команды, при отсутствии данной команды в СК ПЯ функция возвращает значение -1.

```
int look_skpp()  // проверка: входит ли данная команда в СК ПЯ
{
 int i,k=-1;
 for(i=0;i<skppnum;i++)
       if (skpp[i].instr ==tokenval) k=i;
 if (k!=-1) return skpp[k].w;
 else return -1;
}

// tokenval –  код команды, поиск которой осуществляется
// skppnum – количество записей в таблице skpp
// skpp – таблица для хранения системы команд ПЯ
```

Рисунок 3 – Поиск команды в СК ПЯ.

При обнаружении очередной команды в списке токенов проверяется есть ли эта команда в таблице команд. Реализация функции для такой проверки представлена на рисунке 4.

```
int lookup(int opcod)
       /* Возвращает p1 - номер записи в таблице команд для opcod */
       /* Возвращает 0, если такой команды в таблице команд нет */
       /* optable -  таблица команд */
{
 int p1;
 for(p1=lastentry; p1>0; p1--)
 if (optable[p1].cod == opcod)
 return p1;
 return 0;
}
```

Рисунок 4 – Поиск в таблице команд

Далее возможны два варианта действий:

1. Такой команды в таблице команд еще нет. В этом случае, в таблицу добавляется новая строка, в первое поле этой строки вносится код команды. Во второе поле из таблицы, описывающей систему команд ВП, выбирается вес данной команды для ВП. Осуществляется поиск в таблице, описывающей систему команд ПЯ (рис. 3), и в третье поле вносится вес команды для ПЯ или -1 (если команда не входит в систему команд ПЯ) Число L равно 1, если команда находится вне циклов или вычисляется в зависимости от количества итераций циклов, внутри которых находится команда.
2. Такая команда в таблице уже есть. В этом случае находится строка таблицы, которая содержит данную команду, и изменяется только число L (увеличивается на единицу или на другое число, в зависимости от того находится ли эта команда внутри циклов (одного или нескольких) или вне циклов вообще).

На основании таблицы команд блока вычисляется вес данного блока для ВП и ПЯ.

Вес для ПЯ вычисляется формуле:

$$W_{ПЯ} = \sum_{i=1}^{n} p_{ПЯ\,i} \times l_i ,$$

где n - количество записей в таблице команд, $p_{ПЯ\,i}$ - вес i -той команды для ПЯ, l_i - сколько раз i- тая команда может быть выполнена в данном блоке

Вес для ВП вычисляется по аналогичной формуле:

$$W_{ВП} = \sum_{i=1}^{n} p_{ВП\,i} \times l_i$$

где n - количество записей в таблице команд, $p_{ВП\,i}$ - вес i -той команды для ВП, l_i - сколько раз i- тая команда может быть выполнена в данном блоке.

Реализация функции для вычисления веса блока приведена на рисунке 5

```
void weightblock()
{ int i;
  WPP=0;WVP=0;
  for(i=1;i<=lastentry;i++)
    {if (optable[i].vp!=-1)
       WPP=WPP+optable[i].vp * optable[i].count;
        else {WPP=-1; break;}
    }
  for(i=1;i<=lastentry;i++)
    {if  (optable[i].pp!=-1)
       WVP=WVP+optable[i].pp* optable[i].count;
       else {WVP=-1; break;}
    }
}
```

Рисунок 5 – Вычисление веса блока

Значение веса для каждого блока сохраняется в специально предназначенном для этого файле. Структура такого файла следующая: идентификатор блока 1, $W_{ВП}$ для блока 1, $W_{ПЯ}$ для блока 1, идентификатор блока 2, $W_{ВП}$ для блока 2, $W_{ПЯ}$ для блока 2, и т.д.

VI Распределение блоков по волновым фронтам и определение количества фронтов волн. На основании графа зависимостей между блоками исходного фрагмента программы (I.4) для каждого блока осуществляется выбор списка родителей, подсчитывается их количество и определяется порядок выполнения каждого блока по индуктивному принципу:

если у блока нет родителей (предшествующих блоков), то его порядок выполнения 0;

для всех остальных блоков порядок выполнения определяется как максимальный порядок выполнения всех родительских блоков (по отношению к данному) плюс один.

Блоки, которые имеют одинаковый порядок выполнения, могут выполняться параллельно и, соответственно, назначаются на один и тот же волновой фронт. Подсчитывается M - количество таких волновых фронтов. Реализация фрагмента программы, в котором показано нахождение порядка выполнения каждого блока, приведена на рис. 6.

```
                                                  . . .
//поиск блоков без родителей, им назначается нулевой порядок выполнения
int flag; // flag - показывает все ли порядки вычислены(0 -все, а -1 нет)
for(i=0;i<=Nodes-1;i++){
if (GR[i].CPAR==0) GR[i].O=0;
else {GR[i].O=-1;flag=-1;}
                        }
int Omax=0; // максимальный порядок блоков, количество фронтов волн определяется как Omax+1
while (flag==-1)
{
         for (i=0;i<Nodes;i++)
           {
              if (GR[i].O==-1)
                {
                   //k - это счетчик родителей
                   int max=-8;//max - максимальный порядок всех родителей
                   for(int k=1;k<=GR[i].CPAR;k++)
                       {
                         if (i==0)l=0;
                         else l=GR[i-1].PINDEX;
                         r=poisk(TIME_PARENTS[l+k-1]);
                         if (GR[r].O==-1){max=-8;break; }
                         if (max<GR[r].O) {max=GR[r].O;};
                       }
                         if (max>=0) {GR[i].O=max+1; if (Omax<max+1){Omax=max+1;};}
                 }
           }
          flag=0;
          for (i=0;i<Nodes;i++){if (GR[i].O==-1) {flag=-1;break;}}
 }
                                                  . . .
```

Рисунок 6 – Определение порядка выполнения блоков

*VII. Распределение блоков і-того волнового фронта между ВП и эквивалентным ПЯ**. Внутри одного волнового фронта те блоки, которые не могут быть выполнены на ПЯ объединяются в один составной блок, который будет выполнен на ВП, вес этого блока равна сумме весов для ВП блоков, входящих в этот составной блок. Все остальные блоки внутри данного волнового фронта назначаются на эквивалентный ПЯ* и упорядочиваются в порядке убывания $W_{ПЯ}$. Выполняется баланс загрузки ВП и ПЯ*: если вес $W_{ВП}$ составного блока, назначенного на ведущий процессор, меньше, чем вес $W_{ПЯ}$ первого блока (с наибольшим весом для ПЯ) назначенного на ПЯ*, и количество блоков назначенных на ПЯ* превышает число ПЯ на кристалле, то на ВП назначаются дополнительные блоки, подходящие по весу.

*VIII. Распределение блоков і-того волнового фронта между ПЯ из ПЯ** Блоки, назначенные для выполнения на ПЯ*, распределяются между всеми ПЯ на кристалле. Выполняется баланс загрузки ПЯ.

IX. Вывод результатов распределения. По окончании работы алгоритма результаты распределения сохраняются в отдельном файле в виде идентификаторов процессоров и списков идентификаторов блоков, назначенных каждому процессору, и затем передаются в память процессоров PIM-системы для параллельной реализации.

Выводы

Приведенный в статье алгоритм является частью программы распределения приложения между процессорами PIM-системы, в основу которой положены предложенная авторами стратегия распределения приложений, новые критерии и принципы распределения, которые в максимальной степени учитывают особенности архитектурно-структурной организации PIM-систем. Это позволяет исключить на начальном этапе распределения множество итераций и тем самым уменьшить время распределения, за счет того, что распределение приложения по процессорам сначала выполняют целенаправленно согласно новому критерию – соответствие системы команд процессоров набору операций распределяемого приложения. Кроме того, сходимость и корректность (точность) алгоритма распределения улучшаются при таком подходе за счет того, что на каждом уровне распределения осуществляют проверку и при необходимости корректировку баланса загрузки процессоров PIM-системы.

Разработанный алгоритм и приведенные в статье фрагменты программ, написанные на языке C++, подтверждают работоспособность программы, а также правильность теоретических и методических положений применительно к процедуре распределения.

Особенности архитектуры PIM-системы и следовательно применимость предложенных решений можно перенести на кластерные классические системы, процессор которых в каждом узле кластера отличается от остальных процессоров этого узла по своим функциональным возможностям и соответственно системой команд. Однако в классических кластерных системах каждый узел кластера выполнен не на одном кристалле как в PIM – системе, а на множестве кристаллов, что сказывается на достижении предельных значений параметров производительности.

Список литературы

1. Яковлев Ю.С. Однокристальные компьютерные системы высокой производительности. Особенности архитектурно-структурной организации и внутренних процессов: монография / Ю.С. Яковлев. – Винница: ВНТУ, 2009. – 294 с.
2. Slo-Li Chu. Exploiting Application Parallelism for Processor-in-Memory Architecture / Slo-Li Chu, Tsung-Chuan Huang // Proc. of National Computer Symposium, Taiwan, 2003, December 18-19. – 2003. – P. 293–303. – Режим доступа: http://dspace.lib.fcu.edu.tw/bitstream/2377/564/ 1/OT_1022003305.pdf . – Дата доступа: 17.08.11.
3. Елисеева Е.В. О распараллеливании пользовательских задач в распределенных компьютерных системах типа “процессор-в-памяти” / Елисеева Е.В. //Математические системы и машины.– № 4, 2010. – С. 68-81.
4. Яковлев Ю.С. Основные принципы и методика распределения приложений в сложных компьютерных системах типа “процессор-в-памяти”/Яковлев Ю.С., Елисеева Е.В.// Управляющие машины и системы.– 2009.– № 6.– С. 56-63.
5. Яковлев Ю.С. Математическая модель и стратегия распределения приложений для интеллектуальной памяти распределенных компьютерных систем / Яковлев Ю.С., Елисеева Е.В.// Математичні машини і системи. – 2009. – № 4. – С. 3-17.

Сведения об авторах

Яковлев Юрий Сергеевич – докт. техн. наук, зав. отделом, Институт кибернетики имени В.М. Глушкова Национальной академии наук Украины, 03680, ГСП, г. Киев, пр. Академика Глушкова, 40, тел. 044-526-32-07, моб: +38-067-408-59-07, e-mail: jakus@bigmir.net.

Елисеева Елена Владимировна – молодший науковий співробітник, Институт кибернетики имени В. М. Глушкова Национальной академии наук Украины, 03680, ГСП, г. Киев, Пр. Академика Глушкова, 40, тел. 044-526-32-07, e-mail: evo55555@ukr.net.