05.00.00 Engineering science

05.00.00 Технические науки

UDC 004.4

Mathematical Methods and Algorithms of Mobile Parallel Computing on the Base of Multi-core Processors

[1] Alexander B. Bakulev
[2] Marina A. Bakuleva
[3] Svetlana B. Avilkina

[1] MESI, Russia
Yablochkova street 6, Ryazan city, 390023
Dr. PhD (technical), Assistant Professor
E-mail: alex.bakulev@gmail.com
[2] MESI, Russia
Yablochkova street 6, Ryazan city, 390023
PhD (technical), Assistant Professor
E-mail: marina.bakuleva@gmail.com
[3] MESI, Russia
Yablochkova street 6, Ryazan city, 390023
PhD (pedagogical), Assistant Professor
E-mail: asv@rfmesi.ru

Abstract. This article deals with mathematical models and algorithms, providing mobility of sequential programs parallel representation on the high-level language, presents formal model of operation environment processes management, based on the proposed model of programs parallel representation, presenting computation process on the base of multi-core processors.

Keywords: mobile parallel computing; multi-core processors; computer speed; portable software.

Введение. Развитие технологий программирования для персональных компьютеров с точки зрения повышения производительности программ традиционно происходило в основном экстенсивными методами. По большей части разработчики универсального программного обеспечения (ПО) уповали на постоянное совершенствование аппаратных ресурсов компьютера, в частности технологического процесса производства микропроцессоров. Таким образом, очередная установка процессора с увеличенной тактовой частотой позволяла получить ускорение работы прикладных программ без малейшего их изменения.

В настоящее время достигнут фактический предел роста тактовой частоты процессоров и дальнейшее увеличение их производительности становится возможным только за счет перехода к многоядерной архитектуре, по существу к параллельной архитектуре. Так на сегодняшний день для персональных компьютеров уже широко распространены процессоры с 6 и даже 12 ядрами, а в ближайшей перспективе количество ядер может измеряться сотнями. Уже появились первые портативные устройства (смартфоны и планшетные ПК), использующие микропроцессоры с 4 ядрами. Подобная технологическая революция в архитектуре персонального компьютера предоставляет существенный потенциал для роста производительности ПО, доступный ранее только в мире суперкомпьютеров. Однако для реализации этого потенциала необходимы столь же серьезные революционные изменения современных технологий программирования, а также решение задачи адаптации огромного объема существующего последовательного ПО для эффективного высокопроизводительного выполнения в параллельной вычислительной среде.

Термин последовательное ПО подразумевает программы, написанные в расчете на их исполнение на однопроцессорной ВС и не предусматривающие явно организацию

параллельных вычислений. В этом смысле без перевода последовательного ПО в параллельную форму выигрыш от использования многоядерных микропроцессоров будет незначительным.

Большое разнообразие существующих сегодня архитектурных платформ, операционных систем (ОС), использующих многоядерные процессоры, а также технологий и языков программирования, положенных в основу последовательного ПО, заставляет искать решение указанной задачи для каждой из них в отдельности. Это создает вторую проблему — проблему переносимости параллельных программ, то есть ставит задачу приведения ПО в соответствие с требованием мобильности. Термин мобильное ПО (portable software) подразумевает программы, которые могут быть откомпилированы и корректно выполнены на вычислительных системах (ВС) с различной архитектурой, без каких-либо изменений.

Актуальным представляется создание доступных инструментальных средств для разработки мобильного параллельного программного обеспечения, ориентированного на различные многоядерные архитектуры, а также библиотек параллельных подпрограмм, реализующих базовые алгоритмы обработки данных.

Материалы и методы. Решение проблемы организации параллельного программирования лежит в области создания новых параллельных языков высокого уровня (ЯВУ), либо в модификации наиболее популярных последовательных ЯВУ путём добавления параллельной парадигмы. Наибольшее распространение получил второй подход, что обусловлено:

а) широким распространением наиболее популярных ЯВУ (Фортран, Си, Паскаль);

б) наличием строгих спецификаций на уровне международных стандартов для этих языков;

в) большим объемом созданного на этих языках ПО;

г) широкой практикой программирования на этих языках.

Создание принципиально новых ЯВУ обычно оправдано лишь при невозможности использования модификаций ни одного из распространенных ЯВУ, например, для нетрадиционных архитектур ВС — машина потока данных или при разработке проблемно ориентированных ЯВУ. Таким образом, можно сделать вывод, что решение задачи разработки новых параллельных языков программирования поможет лишь частично снять проблему повышения эффективности многоядерных вычислений. Подобный подход применим только для вновь разрабатываемого ПО, но никак не решает проблему адаптации существующих последовательных программ, а также не снимает проблемы обеспечения мобильных параллельных вычислений.

Решение проблемы переноса и распараллеливания последовательных программ с учетом требования обеспечения последующей мобильности является трудоемкой задачей. Необходимо производить исследование вычислительного процесса на различных его стадиях, начиная с анализа исходных текстов программ, заданных на формальном языке программирования, с последующим анализом структуры программы и зависимостей составляющих её элементов, выбором подходящих способов и методов распараллеливания, и заканчивая анализом способов организации параллельных вычислений на уровне ВС. В результате этого встаёт задача выбора или создания подходящих математических моделей объектов исследования — данных, алгоритмов и программ.

Формальная модель алгоритма используется при анализе и доказательстве корректности новых создаваемых алгоритмов и является предметом исследования теории алгоритмов. Задача анализа алгоритмов состоит в оценке их свойств, для выяснения факта существования или отсутствия эффективных алгоритмов организации определенных вычислений, в том числе из оптимальности.

Модель программы является базовой, так как предметом исследования работы ставится задача преобразования последовательных программ в параллельную форму. При этом необходимо трактовать термин программа в широком аспекте, поскольку область интересов затрагивает особенности программ как текста на формальном языке, так и формы представления вычислительного алгоритма.

В работе предложено использовать семейство алгебраических систем [1] для представления типов данных и управляющих конструкций универсальных ЯВУ. При выборе базиса алгебры управляющих операций, предложенной алгебраической системы

управляющих операций, надлежало соблюсти оптимальный компромисс между явным представлением высокоуровневых структур управления ЯВУ и требованием устранения недостатков непосредственного высокоуровневого представления с позиции выявления последующего параллелизма операций. Ключом к реализации подобного компромисса явилось использование предложенной Дейкстрой концепции «охраняемых операторов», что отражается введением в состав базиса управляющих операций O_c^0 так называемых «стражей» (guards), и дальнейшим развитием этой идеи в работе [2], чему соответствует введение в O_c^0 двух типов «объединителей» (merges). Подобное представление позволяет совместить явное представление ветвлений и высокоуровневое представление условных структур. В работе разработаны алгоритмы исключения операций выхода за пределы ветви управляющей конструкции [3], основанные на предложенной модели мобильного параллельного промежуточного кода, обеспечивающие корректность представления исходных текстов программ. Разработан алгоритм выделения программных регионов [4], основанный на предложенной модели мобильного параллельного промежуточного кода, обеспечивающий предварительное крупноблочное распараллеливание программы, отличающийся меньшей трудоёмкостью в сравнении с аналогами.

На стадии анализа производится выявление скрытого параллелизма в исходной последовательной программе. Для этого используются методы выявления зависимостей между операционными объектами программы (зависимостей по управлению) и зависимостей между информационными объектами программы (зависимостей по данным). В качестве подобных объектов, в зависимости от выбранного масштаба рассмотрения — мелкозернистого или крупнозернистого, могут рассматриваться отдельные операторы (инструкции), группы операторов присваивания, блоки, итерации цикла, условные операторы, выполнение процедур после вызова и т.д. Большинство методов анализа зависимостей базируются на графовом представлении программы и предполагают построение графа зависимости по данным и графа зависимости по управлению и делятся в свою очередь на две большие группы: статические и динамические.

Статические методы выполняются на этапе трансляции исходного текста программы. Возможности статических методов являются ограниченными, так как не всегда возможно выявить полностью все информационные зависимости между операторами, в связи с тем, что при анализе текста программы никогда не известны значения переменных, используемых в ней. Кроме того, в современных ЯВУ существует широкий набор средств, позволяющих осуществлять неявный (косвенный) доступ к информационным объектам. Примером может служить использование указателей и их разыменования, организация доступа к элементам массивов по индексу, использование формальных параметров, процедурных переменных, виртуальных методов классов. Всё это существенно затрудняет задачу анализа потока данных. Применение методов анализа синонимов (alias analysis) позволяет частично решить данную проблему на стадии статического анализа программы во время трансляции, но в общем случае полная информация о потоке данных может быть получена только на этапе исполнения программы. Всё это существенно снижает эффективность выявления параллелизма статическими методами.

Динамические методы исследуют программу на этапе её выполнения. Динамический анализ программ основан на внедрении в исходную программу дополнительных операторов, проводящих анализ. Полученная программа выполняется на некотором тестовом наборе входных данных (или нескольких наборах), и во время выполнения собирается информация о фактических зависимостях, присутствующих в программе на данном конкретном наборе данных.

Такой подход позволяет производить выявление зависимостей во многих ситуациях, когда статический анализ невозможен. Поскольку анализ происходит во время выполнения программы, анализатору доступны значения всех переменных, присутствующих в программе. Поэтому появляется возможность проанализировать любые сложные и запутанные виды зависимостей. Динамический анализатор всегда может однозначно установить ячейку памяти, которая используется в любом операторе программы.

Недостатки динамического анализа также следуют из того, что он производится во время выполнения. Динамический анализ показывает только те зависимости, которые

возникают на данном конкретном запуске программы. Поэтому, используя динамический анализ, никогда нельзя быть уверенным, что найдены все зависимости в программе. Некоторые зависимости могут не проявиться на тестовых данных. Это может привести к генерации некорректной параллельной программы. Таким образом, можно сделать вывод о необходимости использования как статических, так и динамических методов этапа анализа зависимостей последовательной программы, а также их адаптации к особенностям мобильного параллельного представления программ.

Синтез параллельной программы включает выбор схемы распределения данных и планирования вычислений. Планирование вычислений подразумевает поиск такого способа назначения работ по процессорам, при котором достигается некоторая цель в процессе функционирования параллельной ВС. Обычно стремятся минимизировать общее время исполнения всего множества работ. Для решения подобных задач могут использоваться методы теории построения расписаний. Так как в общем случае задачи теории расписаний относятся к классу NP-полных, это говорит о том, что к ним не применимы переборные алгоритмы поиска оптимальных расписаний, поэтому на практике используются эвристические алгоритмы, один из которых предложен в данной работе [5], обладающие полиномиальной и псевдополиномиальной сложностью в зависимости от строгости накладываемых ограничений.

На стадии синтеза, как и на стадии анализа, может также использоваться две группы методов: статические и динамические. Статические методы осуществляют построение предварительного расписания, после чего запускают на выполнение параллельные процессы. Динамические методы могут успешно сочетать решение задач анализа (выявления зависимостей) и планирования параллельных вычислений. Наиболее подходящим методом динамического распараллеливания последовательных программ для мультипроцессорных (многоядерных) вычислительных систем с общей памятью является метод спекулятивной многопоточности [6].

Суть метода состоит в следующем. Среди множества всех фрагментов последовательной программы (регионов), выявляются регионы, имеющие зависимости, характер которых не может быть установлен на этапе трансляции программы из-за неоднозначности. Статические методы при планировании параллельного исполнения подобных регионов, чтобы исключить нарушение информационных зависимостей в случае их возникновения, вынуждены использовать более сильные виды отношений, прибегая к позиции крайнего пессимизма. Такой подход негативно отражается на результатах распараллеливания. Метод спекулятивной многопоточности предписывает параллельное многопоточное выполнение подобных регионов, в расчёте на то, что информационные зависимости на стадии исполнения не проявятся, прибегая к позиции крайнего оптимизма (спекулируя на удаче). В случае если подобные надежды оправдают себя, будет получен выигрыш в производительности, в противном случае, результаты вычислений региона должны быть аннулированы, и он будет выполнен повторно, что приведет к накладным расходам.

Эффективность применения метода спекулятивной многопоточности зависит во многом от качества прогнозирования успешности спекулятивного выполнения тех или иных регионов программы, что требует разработки способов и алгоритмов повышения эффективности использования данного метода, в том числе с учётом специфики обеспечения мобильности. Для этого в работе предложена формализация метода спекулятивной многопоточности [7], используемого для динамического планирования параллельных вычислений, основанная на предложенной модели мобильного параллельного промежуточного кода, обеспечивающая оценку перспективности использования метода спекулятивной многопоточности для циклических участков программы.

Рассмотрим на примере работу данного метода. На рис. 1а изображен фрагмент программы — цикл с параметром.

Если при распараллеливании осуществить развёртку цикла по итерациям, выделяя каждую из них в отдельно выполняемую эпоху

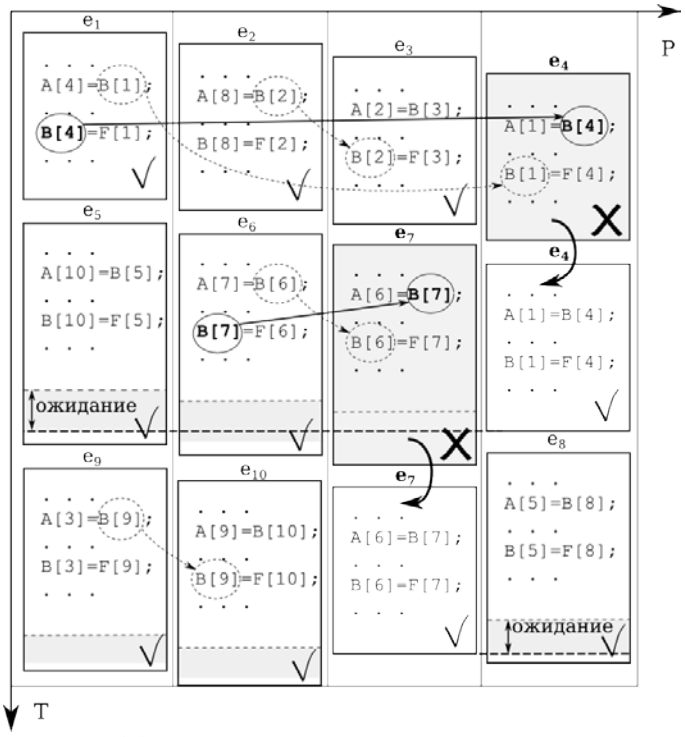
$$\forall e_i \in EP \mid i = \overline{1, N},\ N = \mid EP \mid,$$

то между операторами присваивания v_1 и v_2 (в строках с метками 1 и 2 соответственно), находящимися в разных эпохах, возникнут отношения:

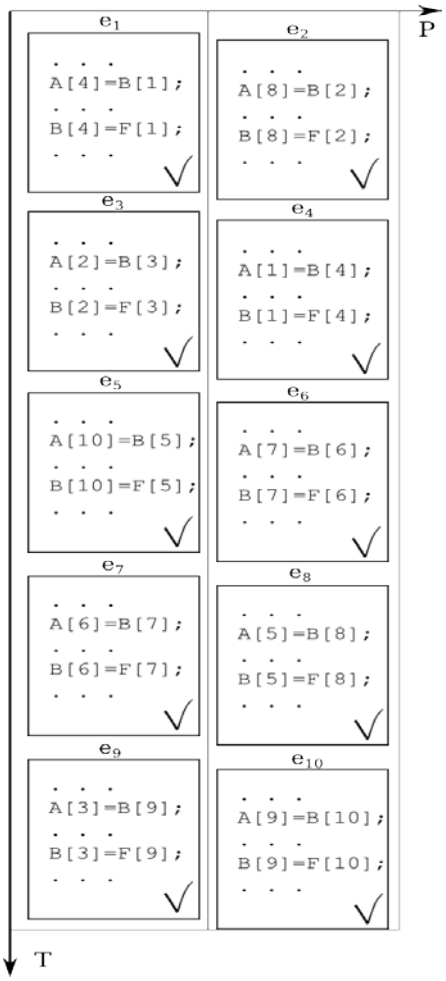
$$v_1^i \, \delta_{in-out}^{\infty} \, v_2^j \mid v_1^i \in e_i, v_2^j \in e_j, i = \overline{1,N}, \; j = \overline{1,N}, i < j, \forall e_i, e_j \in EP,$$

$$v_2^i \, \delta_{out-in}^{\infty} \, v_1^j \mid v_1^i \in e_i, v_2^j \in e_j, i = \overline{1,N}, \; j = \overline{1,N}, i < j, \forall e_i, e_j \in EP,$$

$$v_2^i \, \delta_{out-out}^{\infty} \, v_2^j \mid v_1^i \in e_i, v_2^j \in e_j, i = \overline{1,N}, \; j = \overline{1,N}, i < j, \forall e_i, e_j \in EP.$$



(б) Параллельное выполнение при p = 4



(в) Параллельное выполнение при p = 2

```
   for (i = 0; i < N; i++) {
            . . .
1           A[D[i]] = B[C[i]];
            . . .
2           B[D[i]] = F[C[i]];
            . . .
   }
```

(а) Фрагмент программы с циклом

Рис. 1. Пример работы метода спекулятивной многопоточности

Пусть в процессе выполнения программы перед началом цикла переменные примут следующие значения:

N ← 10,

C ← {1, 2, 3, 4, 5, 6, 7, 8, 9, 10},

D ← {4, 8, 2, 1, 10, 7, 6, 5, 3, 9},

тогда особенности параллельного выполнения итераций цикла на вычислительной системе с четырьмя процессорными элементами будут выглядеть, как представлено на рис. 1б, а для системы с двумя процессорными элементами — на рис. 1в.

В случае спекулятивного распределения эпох по двум процессорным элементам нарушения зависимостей не произойдёт, все спекулятивные эпохи завершатся успешно (на рисунке об этом информирует галочка, проставленная в конце каждой эпохи). Для случая четырёх процессорных элементов, во время параллельного исполнения эпох будут выявлены следующие зависимости: $v_1^1 \delta_{in-out} v_2^4$, $v_1^1 \delta_{out-in} v_2^4$, $v_1^1 \delta_{in-out} v_2^4$, $v_1^6 \delta_{in-out} v_2^7$, $v_1^6 \delta_{out-in} v_2^7$, $v_1^9 \delta_{in-out} v_2^{10}$ (на рис. 3.5б зависимые части операторов помечены кружочками, а зависимости указаны стрелками). Тем не менее, только две из них: $v_1^1 \delta_{out-in} v_2^4$ и $v_1^6 \delta_{out-in} v_2^7$ будут нарушены из-за параллельного выполнения операторов, что потребует аннулирования результатов выполнения эпох e_4 и e_7 (на рисунке отмечено крестиком) и повторного их перезапуска (отмечено стрелкой между одноименными эпохами). В результате, будут произведены избыточные вычисления и потеряно время на ожидание начала проверки корректности. Потери показаны на рис. 1б серым фоном.

Обсуждение. Приведенный пример показывает, что благодаря использованию метода спекулятивной многопоточности, время выполнения цикла было сокращено вдвое, при использовании двух процессорных элементов, и втрое, при использовании четырёх процессорных элементов.

В реальной ситуации следует учесть ещё некоторые издержки подобного метода, например, накладные системные расходы на выделение каждой эпохи в отдельный процесс ОС, на перезапуск процессов в случае неудачной спекуляции. Если этого не сделать, то совокупные накладные расходы могут свести на нет преимущества от распараллеливания.

Для организации исполнения мобильного параллельного промежуточного кода были разработаны программные модули, представляющие из себя прототип среды исполнения параллельной виртуальной машины (рис. 2).

Модуль загрузки параллельного промежуточного кода принимает на вход промежуточный код [3, 4, 7]. Модуль статического распараллеливания осуществляет синтез параллельной реализации программы в соответствии с предложенными алгоритмами [4]. Модуль динамического планирования исполнения параллельных потоков реализует алгоритмы спекулятивной многопоточной обработки [7].

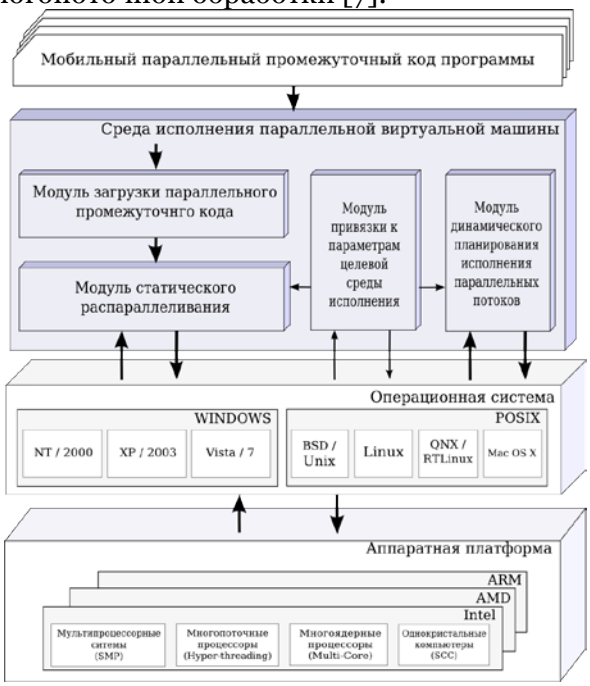


Рис. 2. Программные средства поддержки этапа исполнения мобильного параллельного промежуточного кода

Представленные программные модули были разработаны на языке C++ в кроссплатформенной среде QT 4.6 с компилятором GCC 4.4.0, что в свою очередь позволило создать прототип среды исполнения параллельной виртуальной машины для семейства ОС Windows и для семейства Posix-совместимых ОС.

Результаты. Для оценки разработанных программных средств были проведены эксперименты. Был получен параллельный промежуточный код C++ программы (LZW) архивации файлов по методу LZW и параллельный промежуточный код Паскаль-программы (OLAP) реализации алгоритмов оперативного анализа, основанных на кратномасштабном представлении данных [8].

С помощью разработанных программных модулей, в соответствии со схемой рис. 2, было организовано многопоточное исполнение параллельного промежуточного кода этих программ в среде ОС Window XP и Mandriva Linux 2010.1 на ВС с двухъядерным процессором Intel Core 2 Duo E7200 и 4-х ядерным процессором AMD Phenom X4 9650.

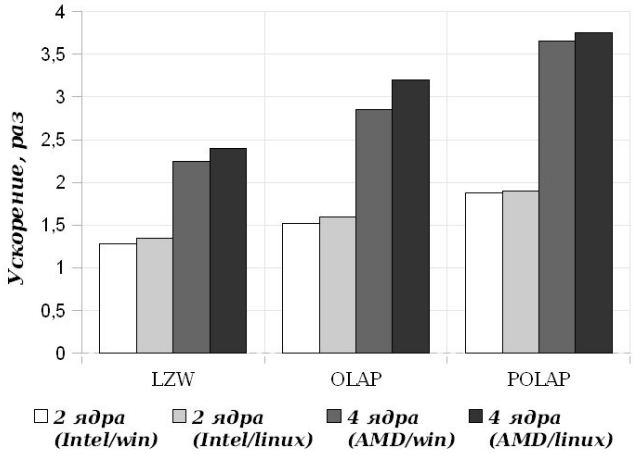


Рис. 3. Результаты ускорения многопоточного выполнения программ на двухъядерной и 4-х ядерной ВС

Для оценки качества результатов автоматического распараллеливания, было организовано исполнение параллельной версии программы (POLAP) реализации алгоритмов оперативного анализа [9], созданной вручную. Все программы неоднократно прогонялись на различных наборах данных, средний показатель ускорения вычислений по каждой программе представлен на рис. 3. Из рисунка видно, что среднее ускорение выполнения программ LZW и OLAP на 2-х и 4-х ядерной ВС составило 70 % от предельной величины. Ускорение выполнения параллельного промежуточного кода программы OLAP в среднем составляет 80 % от ускорения выполнения параллельной версии этой программы (POLAP), выполненной вручную.

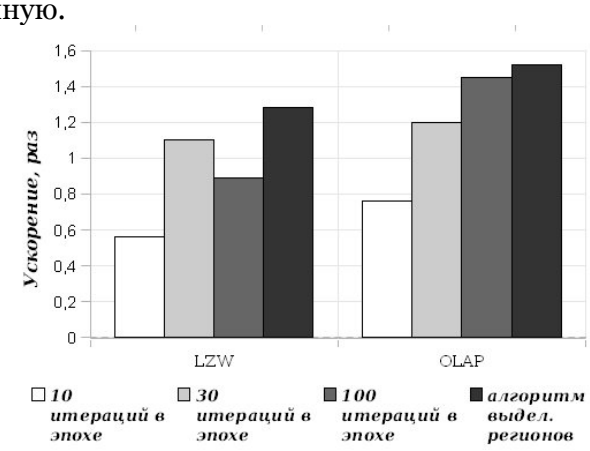


Рис. 4. Результаты ускорения многопоточного выполнения программ на двухъядерной ВС в среде Windows

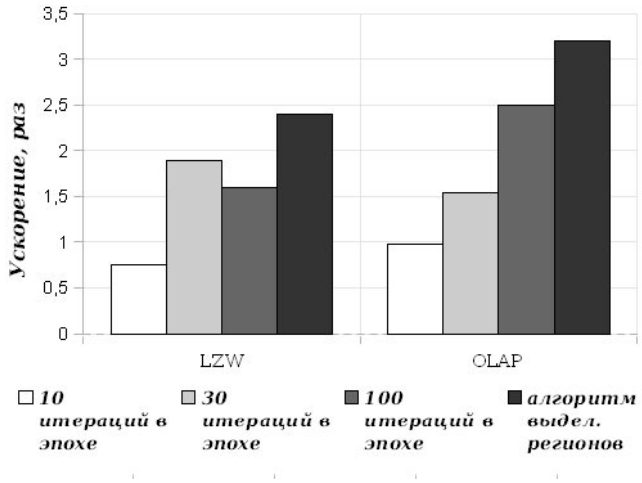


Рис. 5. Результаты ускорения многопоточного выполнения программ на 4-х ядерной ВС в среде Linux

Для того, чтобы оценить влияние предложенного алгоритма выделения спекулятивных регионов и эпох на эффективность использования метода спекулятивной многопоточности, было выполнено сравнение результатов спекулятивного многопоточного выполнения разных версий кода программ LZW и OLAP с фиксированным количеством смежных итераций в каждой спекулятивной эпохе (10, 30 и 100) и выделенным динамически с помощью алгоритма. Результаты ускорения от использования метода спекулятивной многопоточности для двухъядерной системы на базе Intel/Windows представлены на рис. 4, для 4-х ядерной системы на базе AMD/Linux представлены на рис. 5.

Заключение. Исходя из результатов экспериментов, можно заключить, что использование предложенного алгоритма позволило повысить производительность параллельной обработки метода спекулятивной многопоточности от 5% до 65% за счет оптимального выделения спекулятивных регионов и определения оптимального количества смежных итераций, выполняемых внутри каждой спекулятивной эпохи [10].

Дальнейшая работа планируется в направлении разработки программных и инструментальных средств, автоматизирующих создание многопоточных приложений на основе последовательных программ, включая подсистему динамического планирования параллельных вычислений для компьютерных систем, использующих многоядерные архитектуры.

Примечания:

1. Бакулев А.В., Телков И.А. Алгебраическая модель операций мобильного промежуточного кода // Новые информационные технологии: Межвуз. сб. науч. трудов. Рязань: РГРТА, 2002. С. 19-22.

2. Brandis M.M. Optimizing Compilers for Structured Programming Languages. A dissertation for the degree of Doctor of Technical Sciences. 1995.

3. Бакулев А.В., Бакулева М.А. Алгоритм исключения операторов выхода за пределы ветви управляющей конструкции в произвольной точке // Тез. докл. 14-ой Всероссийской научно-технической конференции студентов, молодых ученых и специалистов "Новые информационные технологии в научных исследованиях и в образовании". Рязань: РГРТУ, 2009. С. 191-192.

4. Бакулев А.В., Бакулева М.А. Алгоритм выделения параллельных регионов на основе управляющей структуры программы // Информационные технологии: Межвузовский сборник научных трудов. Рязань: РГРТУ, 2011. С. 27-30.

5. Бакулев А.В. Алгоритм синтеза параллельной реализации последовательной программы для вычислительных систем, построенных на базе многоядерных процессоров // Вестник РГРТУ. Выпуск 30. Рязань: РГРТУ, 2009. С. 43-49.

6. Iffat Hoque Kazi. Dynamically Adaptive Parallelization Model Based on Speculative Multithreading. PhD thesis. Minnesota, 2000. 188 p.

7. Бакулев А.В., Бакулева М.А. Использование метода спекулятивной многопоточности для динамического распараллеливания последовательных программ // Сборник 16-й Всероссийской научно-технической конференции студентов, молодых ученых и специалистов «Новые информационные технологии в научных исследованиях». Рязань: РГРТУ, 2011. С. 145-146.

8. Бакулева М.А. Применение вейвлет-преобразований для представления данных хранилища // Вестник РГРТА. 2006. №18.

9. Бакулева М.А. Применение вейвлет-преобразования для анализа данных хранилища. Вестник РГРТУ. Научно-технический журнал. Выпуск 21. Рязань: РГРТУ, 2007. С. 57-60.

10. Бакулев А.В., Бакулева М.А. Параллельная реализация алгоритмов оперативного анализа, основанных на кратномасштабном представлении данных хранилища. Информационные технологии и телекоммуникации в образовании и науке. Межвуз. сб. научн. трудов. Рязань: РГРТА, 2006.

УДК 004.4

Математических модели и алгоритмы организации мобильных параллельных вычислений в среде многоядерных процессоров

[1] Александр Валериевич Бакулев
[2] Марина Алексеевна Бакулева
[3] Светлана Викторовна Авилкина

[1] МЭСИ (Рязанский филиал), Россия
390023, Рязань, пр.Яблочкова, 6
Кандидат технических наук, доцент
E-mail: alex.bakulev@gmail.com
[2] МЭСИ (Рязанский филиал), Россия
390023, Рязань, пр.Яблочкова, 6
Кандидат технических наук
E-mail: marina.bakuleva@gmail.com
[3] МЭСИ (Рязанский филиал), Россия
390023, Рязань, пр.Яблочкова, 6
Кандидат педагогических наук
E-mail: asv@rfmesi.ru

Аннотация. Предложены формальные модели и алгоритмы, обеспечивающие мобильность параллельного представления последовательных программ, заданных на различных языках высокого уровня. Предложена формальная модель подсистемы управления процессами операционной среды, основанная на предложенной модели параллельного представления программ, обеспечивающая представление вычислительного процесса на уровне среды многоядерных процессоров.

Ключевые слова: мобильные параллельные вычисления; многоядерные процессоры; производительность компьютеров; мобильное программное обеспечение.