УДК 519 716.35

**РЕАЛИЗАЦИЯ И ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ УПРЕЖДАЮЩЕГО ПАРАЛЛЕЛИЗМА**1

В.П. Кутепов, М.И. Зубов

В статье предложен метод реализации параллельных вычислений с упреждением, позволяющий увеличивать степень распараллеливания программ и при определённых условиях существенно сократить время их выполнения на компьютерных системах. Типичным случаем возможного применения упреждающих вычислений в языках программирования является условный оператор. При его выполнении можно одновременно вычислять значения предиката и с упреждением значения ветвей. Такого рода параллелизм не реализован в настоящее время во многих известных средах параллельного программирования: PVM, MPI, MULTITHREADING, HOPE и др. Предложенный метод реализован как расширение системы функционального параллельного программирования, созданной на базе языка FPTL (Functional Parallel Typified Language).

Дано краткое описание языка FPTL и предоставляемых в нём возможностей. Определены формулы для оценки эффективности упреждающих вычислений по критериям ускорения и уменьшения времени выполнения параллельных программ. Описаны ключевые механизмы интерпретатора FPTL и выполненная в нём реализация предложенного метода. Показан пример эквивалентного представления программы для эффективно использования упреждающих вычислений. Приведены результаты экспериментов, демонстрирующие эффективность разработанного расширения и возможные границы эффективности упреждающих вычислений при выполнении параллельных FPTL программ на многоядерных компьютерах.

Работа выполнена при финансовой поддержке РФФИ, грант 18-01-00548.

*Ключевые слова*: параллельное программирование, управление параллельными процессами, упреждающие вычисления.

**IMPLEMENTATION AND EXPERIMENTAL RESEARCH OF THE EFFICIENCY OF PREEMPTION PARALLELISM**1

V.P. Kutepov, M.I. Zubov

The article proposes a method of implementing parallel computations with preemption, which allows to increase the degree of parallelization of programs and, under certain conditions, significantly reduce their execution time on computer systems. A typical case of the possible application of proactive calculations in programming languages ​​is the conditional operator. When executed, it is possible to simultaneously calculate the values of the predicate and in advance the value of the branches. This kind of parallelism is not currently implemented in many well-known parallel programming environments: PVM, MPI, MULTITHREADING, HOPE, etc. The proposed method is implemented as an extension of the functional parallel programming system created on the basis of the FPTL (Functional Parallel Typified Language).

A brief description of the FPTL language and its capabilities is given. The formulas for evaluating the effectiveness of pre-emptive calculations based on the criteria of acceleration and reduction of the execution time of parallel programs are determined. The key mechanisms of the FPTL interpreter and the implementation of the proposed method are described. An example of an equivalent representation of a program for effectively using proactive calculations is shown. The results of experiments demonstrating the effectiveness of the developed method and the possible limits of the efficiency of proactive computations when executing parallel FPTL programs on multi-core computers are presented.

The work was supported by RFBR grant 18-01-00548.

*Keywords*: parallel programming, control of parallel processes, proactive calculations.

**Введение**

Для ускорения выполнения программ в настоящее время используются разнообразные методы и приёмы их распараллеливания и выполнения на компьютерных системах. Для их реализации сегодня разработаны различной ориентации программные средства: PVM[1], MPI[2], ERLANG[3] для многокомпонентных систем с распределённой памятью, MULTITHREADING[4], HOPE[5], FPTL[6,7] для систем с общей памятью, в частности для многоядерных компьютеров.

Во всех этих программных средах параллелизм описывается в программе либо в виде множества параллельно выполняемых и взаимодействующих процессов, представляемых в виде фрагментов программы (PVM, MPI, MULTITHREADING, HOPE), либо с использованием функциональной нотации (FPTL, ERLANG), предполагающей динамическое выявление параллелизма в программе при её выполнении.

В статье [8] дан анализ различных форм параллелизма, который возникает в реальных задачах, где рассмотрена ещё одна важная и не реализованная в упомянутых выше средах параллельного программирования форма так называемого упреждающего параллелизма.

Этот достаточно часто используемый приём уменьшения времени выполнения работ и программ, когда некоторые их части реализуются с забеганием вперёд и результат того, какая часть действительно потребуется для завершения всей работы или программы, решается после выполнения определённого условия.

Типичным примером является условный оператор в языках программирования, для сокращения времени выполнения которого можно одновременно вычислять значения предиката и функций и , однако итоговый результат (значение или ) определится только по завершении выполнения предиката .

Реализация подобной формы упреждающего параллелизма является нетривиальной, так как требуются достаточно сложные и затратные по времени механизмы прерывания выполняемых, но уже не влияющих на результат фрагментов программы (в данном примере – вычисление значения или ).

В настоящей статье проблема упреждающих вычислений исследуется и практически решается применительно к языку функционального параллельного программирования FPTL, реализованному на многоядерных компьютерах [7].

Этот язык в определённом смысле уникален тем, что в нём параллелизм в программе описывается не на процессном уровне в виде явного указания, какие фрагменты программы должны выполняться параллельно, а явно представляется посредством применяемых операций композиции функций, из которых только одна является последовательной.

Язык реализован на многоядерных компьютерах и успешно используется для обучения студентов кафедры прикладной математики и информатики методам и средствам параллельного программирования, а также аспирантами и научными работниками для ускорения решения сложных задач путём распараллеливания.

**Язык функционального программирования FPTL**

FPTL – язык со строгой неявной динамической типизацией. Данные в нём представляются в виде кортежей – последовательностей в общем случае разнотипных элементов.

Функции в языке являютсятипизированными -арными соответствиями между кортежами данных , где – длина входного кортежа функции, – выходного, и – типы элементов входного и выходного кортежей. Функции арности (0, 1) – константы. Для m и n, равных 0, имеем кортеж нулевой длины, обозначаемый λ, со свойствами λα = αλ = α, где α – произвольный кортеж.

Формально функции определяются как системы функциональных уравнений , где – функциональные термы, построенные из заданных (базисных) функций и функциональных переменных путем применения операций композиции функций: →, \*, •. Для функций и функциональных переменных задана их арность, а для базисных функций – также их тип. Тип функциональных переменных совпадает с типом определяющих их термов и однозначно определяется из задания типов базисных функций и правил вывода типов для функций, построенных путем применения операций композиции [9].

В языке помимо встроенных типов данных (*bool*, *real*, *int*, *string*) можно определять абстрактные типы данных; также существует данное специального типа, обозначающее неопределённое значение и обозначаемое . Для любого кортежа выполняется свойство .

Пусть – -арная функция , – результат ее применения к кортежу , а – заданные функции. Синтаксис и семантика операций композиции определяются следующим образом.

1. Последовательная композиция «•»:

*;* .

1. Операция конкатенации кортежей значений функций (параллельная композиция) «\*»:

*;* .

1. Операция условной композиции «→»:

*;*

FPTL позволяет определить любой структурный тип данных, называемый *абстрактным типом данных* (АТД). Для определения АТД применяются те же операции композиции, которые применяются для задания функций, а также конструкторы, деструкторы (обратные конструкторам функции) и операция объединения двух множеств данных «++» [7]. Используемые в определении абстрактного типа данных (системе реляционных уравнений) конструкторы и деструкторы выполняют роль базисных функций.

Приведем пример определения в FPTL абстрактного типа данных (списка натуральных чисел):

**data** ListOfNat {

Nat = c\_null ++ Nat • c\_succ;

ListOfNat = c\_nil ++ (Nat \* ListOfNat) • c\_cons;}

Здесь функции-конструкторы c\_null, c\_succ, c\_nil и с\_сons имеют арности (0,1), (1,1), (0,1) и (2,1) соответственно и следующие типы: {λ}→Nat, Nat → Nat, {λ} → ListOfNat, Nat \* ListOfNat → ListOfNat.

Обратные к ним функции (деструкторы), обозначаемые как ~c\_null, ~c\_succ, ~c\_nil, ~c\_cons, автоматически извлекаются из определения типа и имеют следующую интерпретацию:

Приведем пример функции предиката, которая проверяет принадлежность кортежа типу данных ListOfNat:

IsListOfNat = ~c\_nil ++ ~c\_cons.(IsNat • IsListOfNat);

IsNat = ~c\_null ++ ~c\_succ.IsNat.

Функция IsListOfNat определена на любом кортеже данных, принадлежащих ListOfNat, и ее значением является λ, что может трактоваться как “истина”. Для других отличных от ListOfNat данных в качестве результата применения IsListOfNat будет неопределенное значение ω, которое может также трактоваться как “ложь”.

Пример программы вычисления длины списка приведен ниже.

**data** ListOfNat {

Nat = c\_null ++ Nat.c\_succ;

ListOfNat = c\_nil ++ (Nat • ListOfNat).c\_cons;}

**scheme** Length {

Length = ~c\_nil -> c\_null, ~c\_cons.[2].Length.c\_succ;}

**application**

list = (c\_null.c\_succ • (c\_nil • c\_null).c\_cons).c\_cons;

%Length(list)

Язык FPTL реализован на многоядерных компьютерах с использованием интерпретации при выполнении программы.[7] При этом программист свободен от необходимости явного указания в программе какие её фрагменты должны выполняться параллельно, что приходится делать во всех перечисленных во введении языках программирования. Интерпретатор языка FPTL динамически определяет какие фрагменты программы могут выполняться одновременно, а планировщик осуществляет назначение этих фрагментов на выполнение вычислительным ядрам компьютера.

**Упреждающий параллелизм и оценка его эффективности**

Возможность упреждающих вычислений при выполнении FPTL программы обязана условной композиции, которая является прямой аналогией условного оператора в языках программирования. Достигаемое при этом ускорение зависит от ряда факторов: сложности FPTL программы и её частей, количества используемых операций условной композиции, вероятности выбора фрагментов программы необходимых для продолжения выполнения программы. В свою очередь, от реализации упреждающего параллелизма зависит время, требуемое для синхронизации, для создания копий данных, которые изменяются в частях программы, выполняемых с упреждением, а также время, необходимое для прерывания выполняемых фрагментов, результаты которых с некоторого момента становятся ненужными.

Рассмотрим фрагмент FPTL программы , полученный путём применения условной композиции. Пусть – среднее время выполнения фрагмента-условия и фрагментов-функций и ; – среднее время необходимое для прерывания выполнения  и ; – среднее время затрачиваемое на синхронизацию в случае выполнения и ; – вероятность выполнения , а – вероятность выполнения . Тогда среднее время выполнения этой условной конструкции при последовательном выполнении задаётся формулой 1, при параллельном выполнении на трёх вычислительных ядрах формулой 2.

(1)

(2)

При малой вычислительной сложности нет смысла использовать упреждающий параллелизм, как и в случае, если и не являются вычислительно сложными, так как время, затрачиваемое на прерывание вычислений, передачу данных и синхронизацию, будет превышать возможный выигрыш от реализации параллелизма. Наибольшее среднее ускорение достигается при и равно 2. Рисунок 1 иллюстрирует зависимость среднего ускорения от времени выполнения условия при фиксированном времени выполнения и и равновероятном выполнении обеих ветвей для случая, когда прерывание функций происходит мгновенно, а синхронизации не требуются. При увеличении разницы между вычислительной сложностью , и ускорение сильно падает. Можно также заметить, что ситуация, когда время выполнения фрагмента больше времени выполнения фрагментов и , предпочтительнее, чем обратная. Наиболее частым на практике является случай, когда фрагменты и сильно различаются по временам их выполнения. Среднее ускорение в этом случае сильно зависит от вероятности выбора сложной с вычислительной точки зрения ветви.

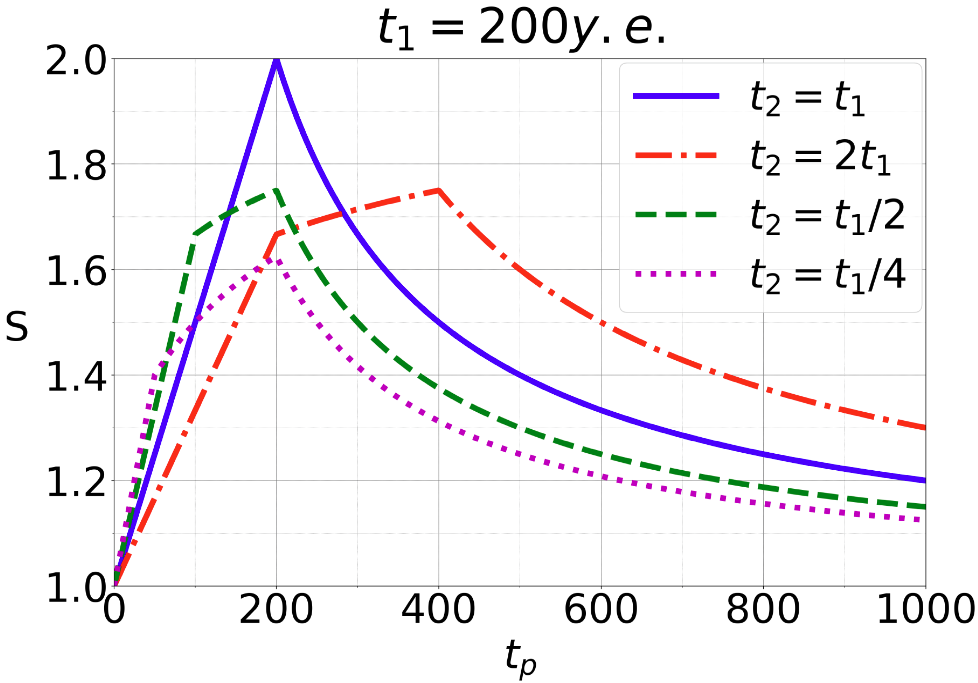


Рис.1. Среднее ускорение в зависимости от времени выполнения фрагментов .

Необходимо отметить, что параллельные программы выполняются на ограниченных ресурсах (заданном количестве ядер, процессоров, узлов), а динамически изменяемая степень параллелизма (количество порождаемых параллельно выполняемых фрагментов программы) может существенно превышать имеющиеся ресурсы. Поэтому возникает естественная проблема эффективного планирования могущих параллельно выполняться фрагментов программы. Очевидно, что с точки зрения эффективности использования ресурсов фрагменты программы, отнесённые к упреждающим вычислениям, должны иметь более низкий приоритет. Это обстоятельство вместе с возникающей сложностью при реализации упреждающего параллелизма были причинами отказа от него в выполненной реализации FPTL на многоядерных компьютерах [7].

**Реализация упреждающего параллелизма в языке FPTL**

Ранее выполненная реализация языка функционального параллельного программирования FPTL на многоядерных компьютерах условно состоит из двух частей: интерпретирующей, в функции которой входит динамическое развёртывание вычислений путём анализа синтаксической структуры программы и выявление в ней частей (функций), которые могут выполняться последовательно или одновременно, и, собственно, операционных средств, осуществляющих планирование, инициализацию и контроль выполнения фрагментов программы [7].

Работу интерпретатора языка FPTL можно кратко описать следующим образом: разбор кода программы и построение абстрактного синтаксического дерева (АСД), классификация узлов АСД и объединение их в функциональные схемы, генерация промежуточного кода и его выполнение [7].

Перед началом выполнения программы создаются рабочие потоки, осуществляющие параллельное выполнение фрагментов программы, соответственно заданному их количеству в параметрах запуска программы. Каждый поток имеет свою двунаправленную очередь задач (параллельно выполняемых фрагментов) с неблокирующей синхронизацией. Задача для выполнения берётся потоком либо из начала своей очереди, либо, при отсутствии в ней задач, из конца чужой очереди.

В ходе выполнения программы сложные функций, соединённые операцией параллельной композиции, выделяются в отдельные задачи и помещаются в конец очереди породившего их потока. Сложными считаются функции, которые являются рекурсивными или содержат другие сложные функции.

Реализация описанного в предыдущем разделе упреждающего параллелизма выполнена как расширение существующих программных средств, созданных для управления параллельным выполнением FPTL программ на многоядерных компьютерах.

При этом были решены следующие задачи:

* явное разделение могущих одновременно выполняться частей FPTL программы на упреждающие и не упреждающие и организация их хранения в динамически создаваемых очередях;
* организация планирования, при котором фрагменты, которые отнесены к упреждающим, должны иметь более низкий приоритет при назначении на выполнение;
* прерывание упреждающих вычислений, необходимость в продолжении которых теряет смысл после выполнения соответствующих условий.

Ниже опишем решения и созданные программные средства с целью эффективной реализации упреждающих вычислений при выполнении FPTL программ на многоядерных компьютерах.

Для хранения упреждающих задач в каждом рабочем потоке создаётся дополнительный экземпляр неблокирующей очереди. Задачи из этих очередей выполняются только при условии, если все очереди с неупреждающими задачами пусты. В отличие от основных очередей из упреждающих очередей задачи на выполнение берутся из начала очереди всеми потоками, так как у задач в конце очереди больший уровень вложенности и, соответственно, меньшая вероятность, что их результаты потребуются.

Для порождения упреждающих задач во время генерации промежуточного кода при анализе функциональной схемы программы условного узла производится следующая проверка: если условие сложное, то для каждой сложной ветви производится генерация промежуточного кода находящихся в этой ветви узлов, который во время выполнения будет выделен в отдельную задачу. Иначе генерируется промежуточный код узлов ветви вместе с узлами, следующими за условной конструкцией. Схема данного алгоритма показана на рисунке 2.

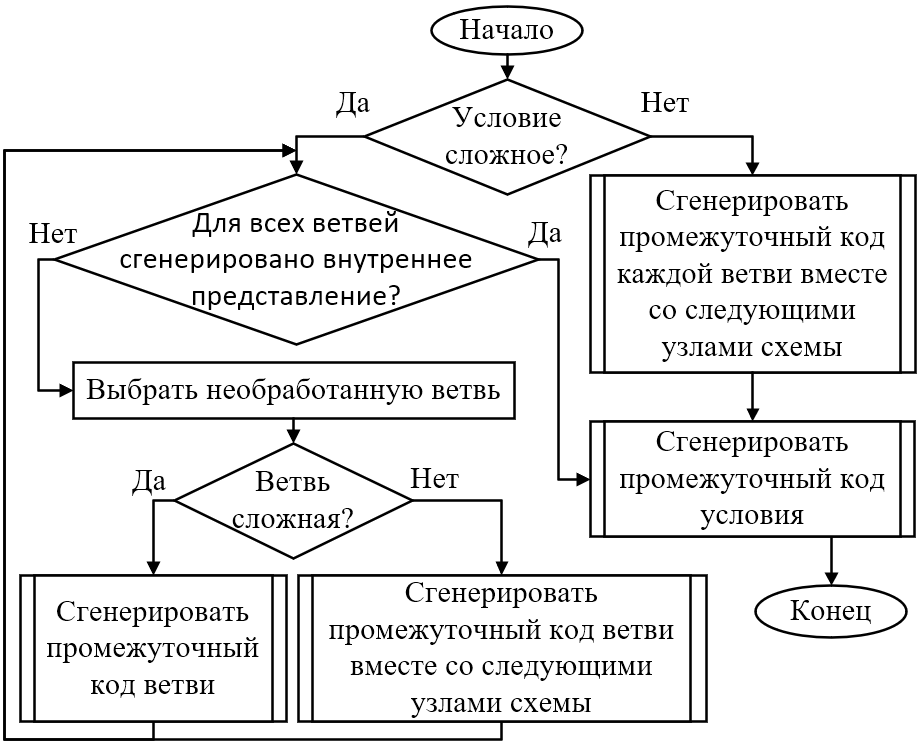


Рис.2. Алгоритм обработки условной конструкции.

Созданные таким образом задачи помечаются как упреждающие и являющиеся частью условной конструкции. Задачи, порождаемые при выполнении упреждающих задач, также помечаются как упреждающие.

Как только завершается вычисление предиката условной конструкции, начинается процесс отмены задач, результаты которых не потребуются, а также перенос не начавших выполняться задач «верной» ветви из упреждающей очереди в основную.

Перемещение задач в основную очередь происходит следующим образом. Если родительская задача упреждающая, флаг принадлежности к условной конструкции выставляется в ложное значение, чтобы при перемещении родительской задачи в основную очередь эта задача также была перемещена. Иначе флаг упреждения выставляется в ложное состояние с целью предотвратить выполнение этой задачи из упреждающей очереди. После этого задача добавляется в основную очередь, если она ещё не завершена или не выполняется. Следом инициируется перемещение порождённых ею задач, которые не помечены как часть условной конструкции. На рисунке 3 приведена схема алгоритма перемещения задач.

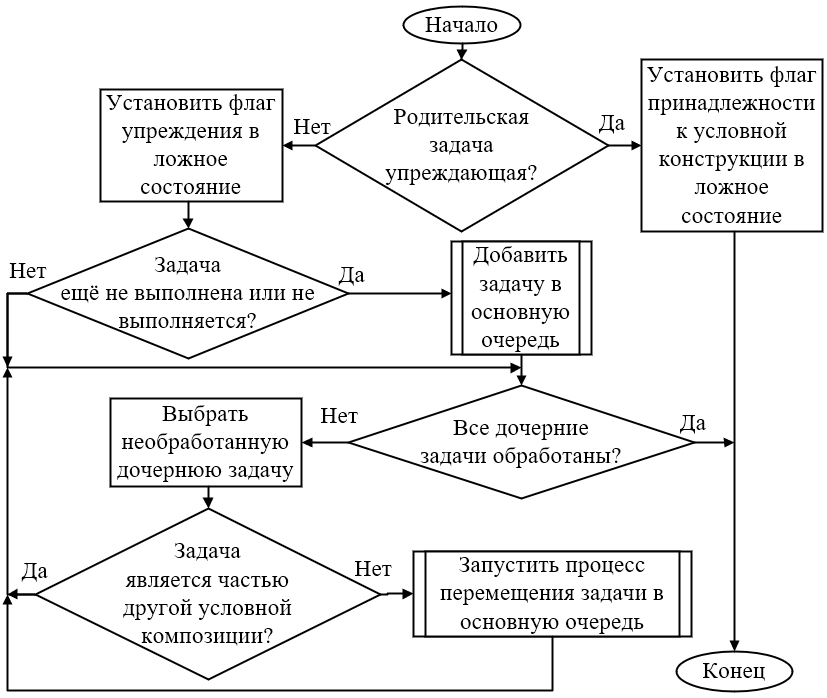


Рис.3. Алгоритм перемещения задачи в основную очередь.

Отмену задач упрощённо можно описать следующим алгоритмом. Если задача ещё не выполнена, флаг готовности выставляется в истинное значение, чтобы выполнение задачи не начиналось. После этого, если задача уже выполняется, начинается замена промежуточного кода узлов задачи на признак конца программы. Далее инициируется отмена дочерних задач. Схематически алгоритм отмены задач проиллюстрирован на рисунке 4.

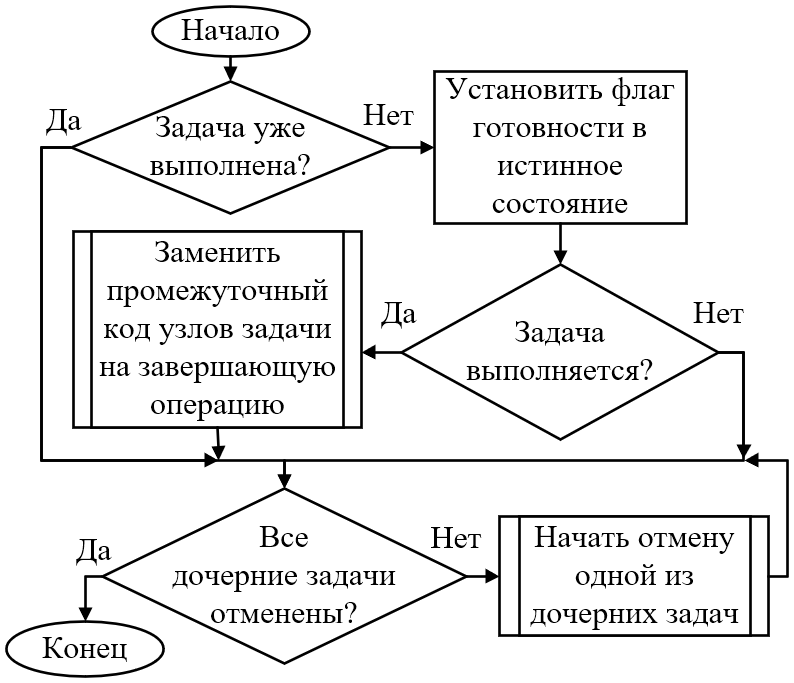


Рис.4. Алгоритм отмены задач.

Описанные алгоритмы управления упреждающими вычислениями реализованы в виде соответствующего программного расширения общей системы управления параллельным выполнением FPTL программ на многоядерных компьютерах [7].

**Результаты экспериментальных исследований**

Исследование эффективности выполненной реализации упреждающих вычислений проводились на компьютере с 8-ядерным процессором Intel Xeon E5-2670.

В качестве примера взята программа численного интегрирования методом трапеций, в которой только одна из ветвей условной конструкции вычислительно сложная и имеет высокую вероятность выполнения. Программа оптимизирована для увеличения эффективности упреждающих вычислений. Каждая итерация при выполнении программы содержит две сложные задачи: вычисление интеграла с шагами h и h/2. Вычислительная сложность задач с каждой итерацией возрастает в четыре раза. Погрешность оценивается по правилу Рунге [11].

Основные шаги алгоритма выглядят следующим образом:

1. , где – левая граница отрезка интегрирования, – правая;
2. Если , то перейти к пункту 4. Здесь – точность, – значение интеграла функции , вычисленное методом трапеции с шагом , а – с шагом .
3. , перейти к пункту 2.
4. Вывести результат .

Код FPTL программы приведён в приложении 1.

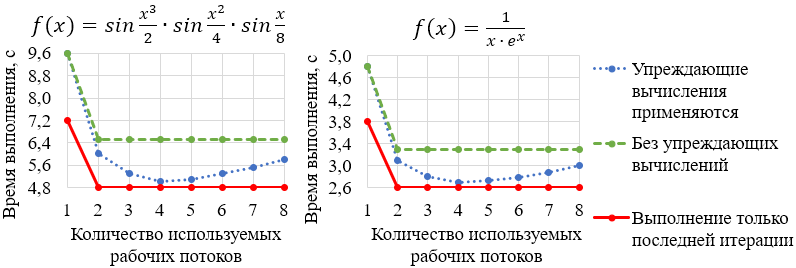


Рис.5. Время выполнения программы численного интегрирования.

На рисунке 5 показано время вычисления интеграла с применением упреждающих вычислений и без них. А также время выполнения последней итерации, чтобы можно было оценить эффективность упреждающих вычислений. Производились вычисления интеграла функции на интервале с точностью и интеграла функции на интервале с точностью .

Видно, что наилучший результат для рассмотренных функций достигается при использовании упреждающих вычислений на 4 рабочих потоках. С дальнейшим увеличением количества рабочих потоков время выполнения возрастает из-за многократного увеличения необходимого для отмены задач времени.

**Заключение**

Из проведённых экспериментов можно сделать вывод, что реализованная модификация интерпретатора FPTL позволяет эффективно выполнять упреждающие вычисления и значительно сокращает время выполнения программ. Также следует отметить, что упреждающие вычисления большой глубины вложенности негативно влияют на время выполнения программы из-за увеличения времени, затрачиваемого на управление, в частности необходимости отмены большого количества задач.

**Приложение 1**

**Программа численного интегрирования**

Scheme Program{ //(a, b, Eps)

@ = Integrate(Func);

Func = ((((([1]\*[1]).mul\*[1]).mul \* 0.5).mul.sin \*

(([1]\*[1]).mul \* 0.25).mul.sin).mul \* ([1] \* 0.125).mul.sin).mul;

Fun Integrate[fFunction] {

@ = ([3] \* [1] \* [2] \* ([2] \* [1]).sub).

([1] \* [2] \* [3] \* ([4] \* 2).div \* ([4] \* 4).div ).Integr;

Integr = ((([2] \* [3] \* [5]).Trp \* ([2] \* [3] \* [4]).Trp \* [1]).

(Runge -> true, false \* [1].print.("\n".print)) ->

([1] \* [2] \* [3] \* ([5] \* 2).div \* ([5] \* 4).div ).Integr);

Runge = ((([1] \* [2]).sub.abs \* 3).div \* [3]).greater;

Trp = ([3] \* ((([1].fFunction \* [2].fFunction ).add \* 2).div \*

(([1] \* [3]).add \* ([2] \* [3]).sub \* [3]).Sum).add).mul;

Sum = ([1] \* [2]).equal -> [1].fFunction, ([1] \* [2]).greater -> 0,

([1].fFunction \* ([2].fFunction \*

(([1] \* [3]).add \* ([2] \* [3]).sub \* [3]).Sum).add).add;}}

**Литература**

1. **Al G****eist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, Vaidy Sunderam.** PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing. IEEE, 1996.
2. **Snir M., Otto S. W., Huss-Lederman S., Walker D., Dongarra J.** MPI: The Complete Reference. MIT press, 1996.
3. **Cesarini F., Thompson S.** Erlang Programming: A Concurrent Approach to Software Development. O'Reilly, 2009.
4. **RICHARD H. CARVER, KUO-CHUNG TAI.** Modern Multithreading: Implementing, Testing, and Debugging Multithreaded Java/C++. Wiley-Interscience, 2005.
5. **Burstall, R.M., Sannella, D.T.** HOPE User's Manual. In preparation. //LFP '80 Proceedings of the 1980 ACM conference on LISP and functional programming, 1980.
6. **Бажанов C.Е., Кутепов В.П., Шестаков Д.А.** Язык функционального параллельного программирования и его реализация на кластерных системах. //Программирование, 2005, № 5, c. 237-269.
7. **Кутепов В.П., Шамаль П.Н.** Реализация языка функционального параллельного программирования FPTL на многоядерных компьютерах. //Известия РАН. Теория и системы управления, 2014, № 3, с. 46-60.
8. **Кутепов В.П., Фальк** **В.Н.** Формы, языки представления, критерии и параметры сложности параллелизма. //Программные продукты и системы, 2010, № 3, с. 16-26.
9. **Бочаров И.А., Кутепов В.П., Шамаль П.Н.** Система типового контроля программ на языке функционального программирования FPTL. //РФФИ №13-07-00810 2014г.
10. **Ахо А.В., Лам М.С., Сети Р., Ульман Д.Д.** Компиляторы: принципы, технологии и инструментарий. 2 изд. Вильямс, 2018.
11. **Амос****ов А.А., Дубинский Ю.А., Копченова Н.В***.* Вычислительные методы для инженеров: Учебное пособие. 2 изд. Изд-во МЭИ, 2003.

**References**

1. **Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, Vaidy Sunderam.** PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing. IEEE, 1996.
2. **Snir M., Otto S. W., Huss-Lederman S., Walker D., Dongarra J.** MPI: The Complete Reference. MIT press, 1996.
3. **Cesarini F., Thompson S.** Erlang Programming: A Concurrent Approach to Software Development. O'Reilly, 2009.
4. **RICHARD H. CARVER, KUO-CHUNG TAI.** Modern Multithreading: Implementing, Testing, and Debugging Multithreaded Java/C++. Wiley-Interscience, 2005.
5. **Burstall, R.M., Sannella, D.T.** HOPE User's Manual. In preparation. //LFP '80 Proceedings of the 1980 ACM conference on LISP and functional programming, 1980.
6. **Bazhanov C.E., Kutepov V.P., Shestakov D.A.** Functional parallel programming language and its implementation on cluster systems. //Programming and Computer Software, 2005, No. 5, p. 237-269.
7. **Kutepov V.P., Shamal’ P.N.** Implementation of functional parallel typified language FPTL on multicore computers. //News of the RAS. Theory and control systems, 2014, No. 3, p. 46-60.
8. **Kutepov V.P., Falk V.N.** Forms, languages of the representation, criteria and parameters of the complexity of parallelism. //Software products and systems, 2010, № 3, p. 16-26.
9. **Bocharov I.A., Kutepov V.P., Shamal P.N.** The system of typical program control in the functional programming language FPTL. //RFBR No. 13-07-00810 2014
10. **Aho A.V., Lam M.S., Sethi R., Ullman D.D.** Compilers: principles, technologies and tools. 2nd ed. Williams, 2018.
11. **Amosov A.A., Dubinsky Yu.A., Kopchenova N.V.** Computational methods for engineers: Study guide. 2nd ed. Publishing house MEI, 2003.

**Сведения об авторах**

**Кутепов Виталий Павлович** – доктор технических наук, профессор кафедры прикладной математики НИУ «МЭИ», e-mail: KutepovVP@mpei.ru

**Зубов Михаил Игоревич** – магистрант кафедры прикладной математики НИУ «МЭИ», e-mail: zumisha@yandex.ru

**Information about authors**

**Kutepov Vitaliy P.** – Dr.Sci.(Techn.), Professor of Applied Mathematics Dept., NRU MPEI, e-mail: KutepovVP@mpei.ru

**Zubov Mikhail I.** – master student of Applied Mathematics Dept., NRU MPEI, e-mail: zumisha@yandex.ru