Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей (05.13.11)

УДК 519.716.35
DOI: 10.24160/1993-6982-2019-4-119-126

Реализация и экспериментальное исследование эффективности упреждающего параллелизма

В.П. Кутепов, М.И. Зубов

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 с упреждением значения функций. Такого рода параллелизм не реализован в настоящее время во многих известных средах параллельного программирования: PVM, MPI, MULTITHREADING, HOPE и др. Предложенный метод выполнен как расширение системы функционального параллельного программирования, созданной на базе языка FPTL (Functional Parallel Typified Language).
Дано краткое описание языка FPTL и предоставляемых им возможностей, определены формулы для оценки эффективности упреждающих вычислений по критериям ускорения и уменьшения времени выполнения параллельных программ, описаны ключевые части интерпретатора FPTL и выполненная программная реализация предложенного метода, приведены результаты экспериментов, демонстрирующих возможные границы эффективности упреждающих вычислений при выполнении параллельных FPTL программ на многоядерных компьютерах.

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

Для цитирования: Кутепов В.П., Зубов М.И. Реализация и экспериментальное исследование эффективности упреждающего параллелизма // Вестник МЭИ. 2019. № 4. С. 119—126. DOI: 10.24160/1993-6982-2019-4-119-126.

Implementation of Lookahead Parallelism and Experimental Implementation of Its Efficiency

V.P. Kutepov, M.I. Zubov

A method for implementing parallel lookahead computations is proposed, the use of which makes it possible to increase the degree to which computer programs are set to run in parallel flows and, under certain conditions, significantly reduce their execution time on computer systems. A conditional operator is the typical case of possible application of lookahead computations in programming languages. In executing it, it is possible to simultaneously compute the predicate values with predicting the function value. This kind of parallelism has not been 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 Functional Parallel Typified Language (FPTL).
The article gives a brief description of the FPTL language and the possibilities available in it, presents formulas for evaluating the effectiveness of lookahead computations in terms of criteria of accelerating and reducing the execution time of parallel programs, describes the key parts of the FPTL interpreter and the software implementation of the proposed method, and presents the results of experiments that demonstrate the possible efficiency limits of using lookahead computations in executing parallel FPTL programs on multicore computers.

Key words: parallel programming, control of parallel processes, lookahead computations, FPTL.

For citation: Kutepov V.P., Zubov M.I. Implementation of Lookahead Parallelism and Experimental Implementation of Its Efficiency. Bulletin of MPEI. 2019;4:119—126. (in Russian). DOI: 10.24160/1993-6982-2019-4-119-126.

Введение

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

Во всех 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 работе [8] дан анализ различных форм параллелизма, возникающего в реальных задачах, где рассмотрена ещё одна важная и нереализованная в упомянутых 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.

Типичный пример — условный оператор if $p(x)$ then $f_1(x)$ else $f_2(x)$, для сокращения времени выполнения которого можно одновременно вычислять значения предиката $p(x)$ и функций $f_1(x)$, $f_2(x)$, однако итоговый результат (значение $f_1(x)$ или $f_2(x)$) определится только по завершении выполнения предиката $p(x)$.

Реализация подобной формы упреждающего параллелизма нетривиальна, поскольку требуются достаточно сложные и затратные по времени механизмы прерывания выполняемых, но уже не влияющих на результат фрагментов программы (вычисление значения $f_1(x)$ или $f_2(x)$).

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

FPTL 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 — типизированные (m,n)-арные соответствия между кортежами данных:

$$t_1' \times t_2' \times \ldots \times t_m' \to t_1'' \times t_2'' \times \ldots \times t_n'',$$

где $m \in \{0, 1, 2, ...\}$, $n \in \{0, 1, 2, ...\}$ — длины входного и выходного кортежей функции; $t_i', i = \overline{1,m}$, $t_j'', j = \overline{1,n}$ — типы элементов входного и выходного кортежей.

Функции арности (0, 1) — константы. Для m и n, равных 0, имеется кортеж λ нулевой длины со свойствами $\lambda\alpha = \alpha\lambda = \alpha$, где α — произвольный кортеж.

Формально функции определяются как системы функциональных уравнений $F_i = \tau_i$, $i = \overline{1,n}$, где τ_i — функциональные термы, построенные из заданных (базисных) функций и функциональных переменных F_i путем применения операций композиции функций: $\to$, $*$, $\bullet$. Для функций и функциональных переменных задана их арность, а для базисных функций — их тип. Типы функциональных переменных и определяющих их термов совпадают и однозначно определяются 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 ω, обозначающее неопределённое значение. Для любого кортежа α выполняется свойство $\omega\alpha = \alpha\omega = \omega$.

Пусть $f^{(m,\,n)}$ — $(m,\,n)$-арная функция $m,\ n \geq 0$, а $f(\alpha)$ — результат ее применения к кортежу α; f_1, f_2 — заданные функции. Синтаксис и семантика операций композиции определяются следующим образом.

Последовательная композиция «$\bullet$»:

$$f^{(m,n)} \Leftrightarrow f_1^{(m,k)} \bullet f_2^{(k,\,n)};\ f(\alpha) \Leftrightarrow f_2\left(f_1(\alpha)\right).$$

Операция конкатенации кортежей значений функций (параллельная композиция) «$*$»:

$$f^{(m,n_1+n_2)} \Leftrightarrow f_1^{(m,n_1)} * f_2^{(m,\,n_2)};\ f(\alpha) \Leftrightarrow f_1(\alpha) f_2(\alpha).$$

Операция условной композиции «$\to$»:

$$f^{(m,n_1|n_2)} \Leftrightarrow f_1^{(m,k)} \to f_2^{(m,\,n_1)}, f_3^{(m,n_2)};$$

$$f(\alpha) \Leftrightarrow \begin{cases} f_2(\alpha), \text{ если } f_1(\alpha) \text{ отлично} \\ \text{от значения «ложь» и } \omega; \\ f_3(\alpha), \text{ иначе.} \end{cases}$$

FPTL позволяет определить любой структурный тип данных, называемый абстрактным типом данных (АТД). Для определения АТД применяются те же операции композиции, которые работают 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 c_cons 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:

$$\sim\text{c\_null}(x) = \begin{cases} \lambda, \text{ если } x = \text{c\_null}; \\ \omega, \text{ иначе}; \end{cases}$$

$$\sim\text{c\_succ}(x) = \begin{cases} y, \text{ если } x = \text{c\_succ}(y); \\ \omega, \text{ иначе}; \end{cases}$$

$$\sim\text{c\_nil}(x) = \begin{cases} \lambda, \text{ если } x = \text{c\_nil}; \\ \omega, \text{ иначе}. \end{cases}$$

Приведем пример функции предиката, проверяющей принадлежность кортежа типу данных ListOfNat:

```
IsListOfNat = ~cnil ++ ~ccons.(IsNat • IsListOfNat);
      IsNat = ~cnull ++ ~csucc.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 понимается как «ложь».

Пример программы вычисления длины списка.

```
data ListOfNat {
    Nat = cnull ++ Nat.csucc;
    ListOfNat = cnil ++ (Nat • ListOfNat).ccons;}
scheme Length {
    Length = ~cnil -> cnull, ~ccons.[2].Length.csucc;}
application
    list = (cnull.csucc • (cnil • cnull).ccons).ccons;
    %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 остальных языках программирования. Интерпретатор языка FPTL динамически определяет, какие фрагменты программы могут выполняться одновременно, а планировщик назначает эти фрагменты на выполнение вычислительным ядрам компьютера.

Упреждающий параллелизм и оценка его эффективности

Возможностью упреждающих вычислений при выполнении FPTL-программа 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, необходимых для продолжения выполнения программы. В свою очередь, от реализации упреждающего параллелизма зависит время, требуемое для синхронизации, создания копий данных, которые меняются в частях программы, выполняемых с упреждением, а также время, необходимое для прерывания выполняемых фрагментов, результаты которых с некоторого момента становятся ненужными.

Проанализируем фрагмент FPTL-программы $\tau_p \rightarrow \rightarrow \tau_1, \tau_2$, полученный путём применения условной композиции.

Пусть t_p, t_1, t_2 — среднее время выполнения фрагмента-условия τ_p и фрагментов-функций τ_1, τ_2; t_{1b}, t_{2b} — среднее время, необходимое для прерывания выполнения τ_1, τ_2; t_{1s}, t_{2s} — среднее время, затрачиваемое на синхронизацию в случае выполнения τ_1, τ_2; q — вероятность выполнения τ_1; $(1 - q)$ — вероятность выполнения τ_2. Тогда среднее время выполнения условной конструкции при последовательном выполнении задаётся формулой

$$T_1 = t_p + t_1 q + t_2(1 - q),$$

а при параллельном выполнении на трёх вычислительных ядрах:

$$T_3 = (\max(t_p + t_{2b}, t_1) + t_{1s})q + (\max(t_p + t_{1b}, t_2) + t_{2s})(1 - q).$$

При малой вычислительной сложности τ_p нет смысла использовать упреждающий параллелизм, как и в случае, если τ_1 и τ_2 не являются вычислительно сложными, поскольку время, затрачиваемое на прерывание вычислений, передачу данных и синхронизацию, будет превышать возможный выигрыш от реализации параллелизма. Наибольшее среднее ускорение $S = T_1/T_3$ достигается при $t_p = t_1 = t_2$; $t_{1b} = t_{2b} = t_{1s} = t_{2s} = 0$ и равно 2. На рисунке 1 проиллюстрирована зависимость среднего ускорения от времени выполнения условия τ_p при фиксированном времени выполнения τ_1 и τ_2 и равно-

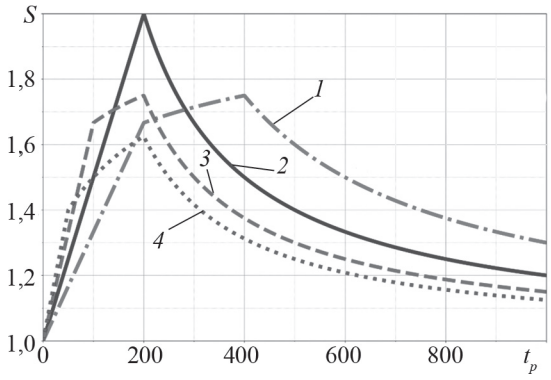

Рис. 1. Среднее ускорение в зависимости от времени выполнения фрагментов τ_p, τ_1, τ_2:

1 — $t_2 = 2t_1$; *2* — $t_2 = t_1$; *3* — $t_2 = t_1/2$; *4* — $t_2 = t_1/4$

вероятном выполнении обеих ветвей для случая, когда прерывание функций происходит мгновенно, а синхронизации не требуются. При увеличении разницы между вычислительной сложностью τ_p, τ_1 и τ_2 ускорение сильно падает. Можно также заметить, что ситуация, когда время выполнения фрагмента τ_p больше времени выполнения фрагментов τ_1 и τ_2, предпочтительнее, чем обратная. Наиболее частым на практике является случай, когда τ_1 и τ_2 сильно различаются по времени их выполнения. Среднее ускорение в этом случае сильно зависит от вероятности выбора сложной с вычислительной точки зрения ветви.

Следует 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 их превышать. Возникает естественная проблема эффективного планирования параллельно выполняющихся 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, стало причиной отказа от него в выполненной реализации FPTL на многоядерных компьютерах [7].

Реализация упреждающего параллелизма в языке FPTL

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, соответственно заданному их количеству в параметрах запуска. 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 функции, 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 упреждающего параллелизма выполнена как расширение существующих программных средств, созданных для управления параллельным выполнением FPTL-программ на многоядерных компьютерах.

Решены следующие задачи:

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

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 проводится следующая проверка: если условие сложное, то для каждой сложной ветви назначается генерация проме-

жуточного 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 рис. 2.

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 данная задача также была перемещена. Иначе флаг упреждения выставляется в ложное состояние с целью предотвратить выполнение 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.

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 дан на рис. 4.

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 проводили на компьютере с восьмиядерным 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. Она оптимизирована для увеличения эффективности упреждающих вычислений.

Каждая итерация при выполнении программы содержит две сложные задачи: вычисления интегралов с шагами h и $h/2$. Вычислительная сложность задач с каждой итерацией возрастает в четыре раза. Погрешность оценивается по правилу Рунге [10, 11].

Основные шаги алгоритма выглядят следующим образом:

1. $h = b - a$, где a, b — левая и правая границы отрезка интегрирования.

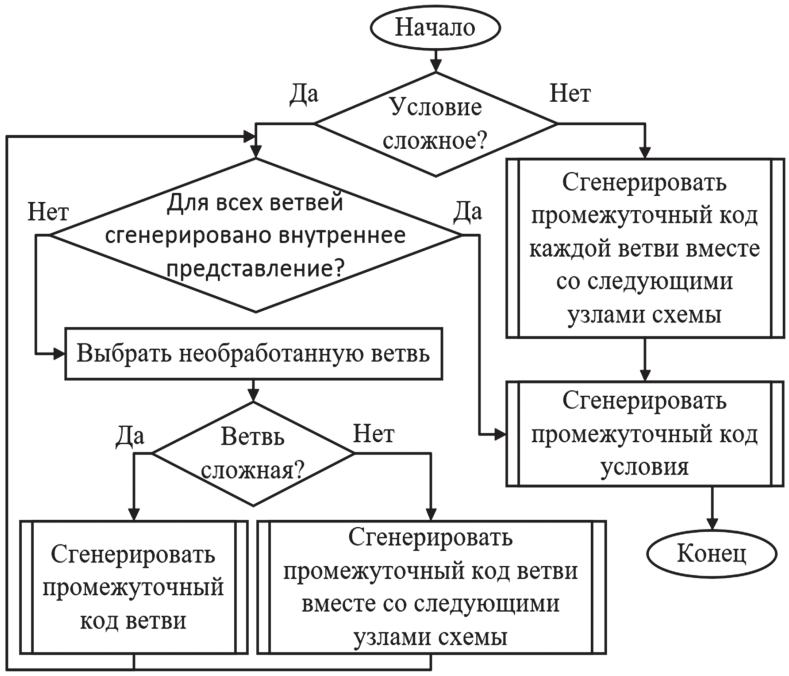

Рис. 2. Алгоритм обработки условной конструкции

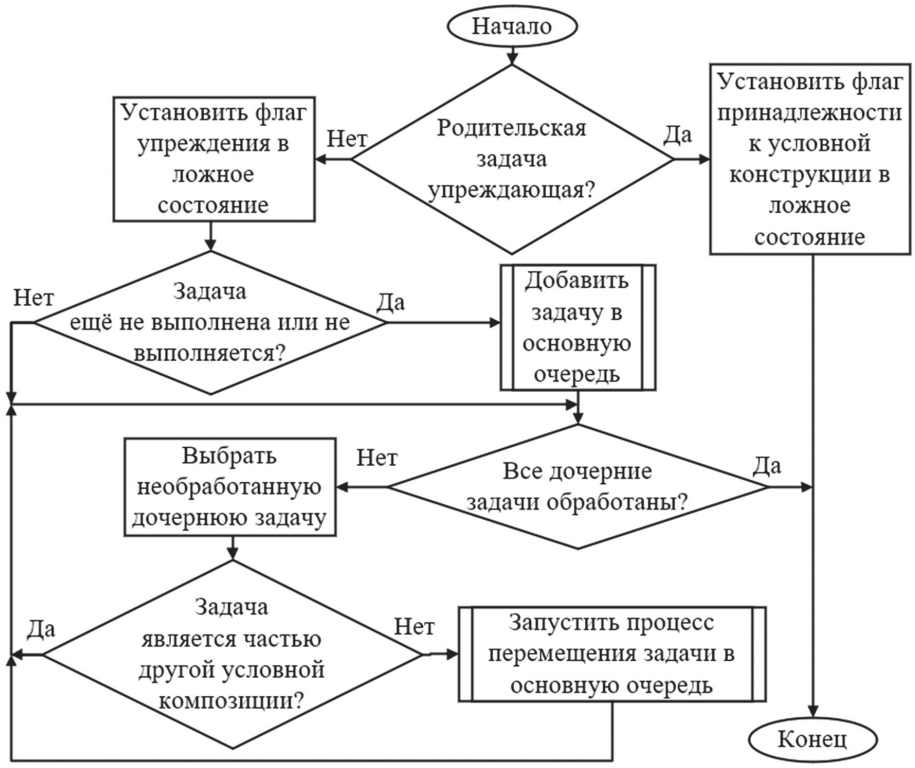

Рис. 3. Алгоритм перемещения задачи в основную очередь

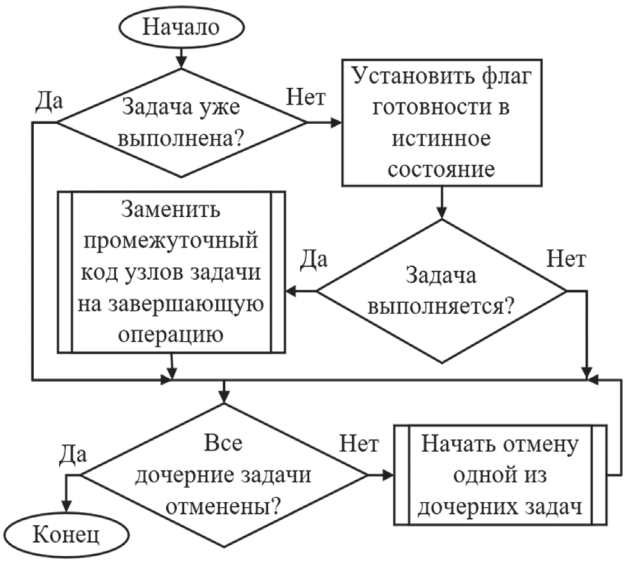

Рис. 4. Алгоритм отмены задач

2. Если $\left|\frac{Trp(h/2)-Trp(h)}{3}\right| \le \varepsilon$, то перейти к пункту 4. Здесь E — точность; $Trp(h)$, $Trp(h/2)$ — значения интеграла функции $f(x)$, вычисленные методом трапеции с шагами h и $h/2$.

3. При $h = h/4$ перейти к пункту 2.

4. Вывести результат $Trp(h/2)$.

Приведем код программы 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;}}
```

На рисунке 5 показано время вычисления интеграла с применением упреждающих вычислений и без них, а также время выполнения последней итерации для оценки эффективности упреждающих вычислений.

Сделаны вычисления интегралов функций

$$f(x) = \sin\frac{x^3}{2}\sin\frac{x^2}{4}\sin\frac{x}{8}$$

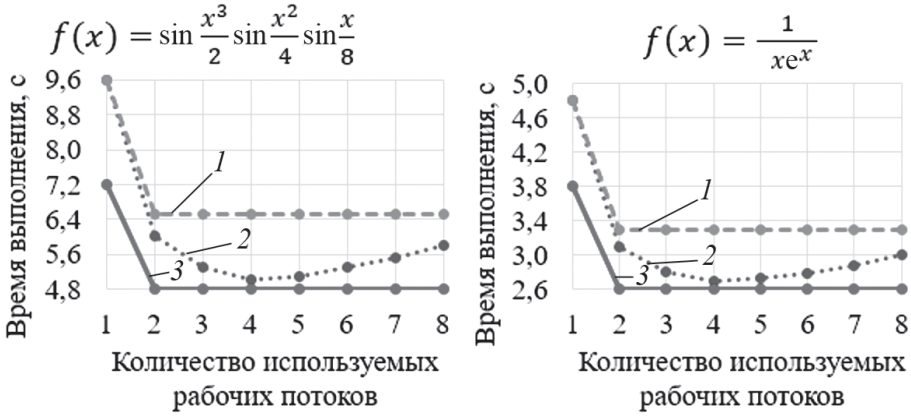

Рис. 5. Время выполнения программы численного интегрирования:
1 — без упреждающих вычислений; *2* — с упреждающими вычислениями; *3* — выполнение только последней итерации

на интервале [0; 25] с точностью 10^{-10} и $f(x) = \frac{1}{x\mathrm{e}^x}$ на интервале [0,01; 20] с точностью 10^{-7}.

Видно, что наилучший результат для анализируемых функций достигается при использовании упреждающих вычислений на четырех 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. 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.

Литература

1. **Al Geist e. a.** PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing. Massachusetts: MIT Press, 1996.

2. **Snir M., Otto S.W., Huss-Lederman S., Walker D., Dongarra J.** MPI: The Complete Reference. Massachusetts: MIT Press, 1996.

3. **Cesarini F., Thompson S.** Erlang Programming: a Concurrent Approach to Software Development. Sebastopol: O'Reilly, 2009.

4. **Carver R.H., Kuo-chung Tai.** Modern Multithreading: Implementing, Testing, and Debugging Multithreaded Java/C++. N.-Y.: Wiley-Interscience, 2005.

5. **Burstall R.M., Sannella D.T.** HOPE User's Manual. In preparation // Proc. 1980 ACM Conf. LISP and Functional Programming. Stanford: Stanford University, 1980. Pp. 136—143.

6. **Бажанов С.Е., Кутепов В.П., Шестаков Д.А.** Язык функционального параллельного программирования и его реализация на кластерных системах // Программирование. 2005. № 5. С. 237—269.

7. **Кутепов В.П., Шамаль П.Н**. Реализация языка функционального параллельного программирования FPTL на многоядерных компьютерах // Известия РАН. Серия «Теория и системы управления». 2014. № 3. С. 46—60.

References

1. **Al Geist e. a.** PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing. Massachusetts: MIT Press, 1996.

2. **Snir M., Otto S.W., Huss-Lederman S., Walker D., Dongarra J.** MPI: The Complete Reference. Massachusetts: MIT Press, 1996.

3. **Cesarini F., Thompson S.** Erlang Programming: a Concurrent Approach to Software Development. Sebastopol: O'Reilly, 2009.

4. **Carver R.H., Kuo-chung Tai.** Modern Multithreading: Implementing, Testing, and Debugging Multithreaded Java/C++. N.-Y.: Wiley-Interscience, 2005.

5. **Burstall R.M., Sannella D.T.** HOPE User's Manual. In preparation. Proc. 1980 ACM Conf. LISP and Functional Programming. Stanford: Stanford University, 1980: 136—143.

6. **Bazhanov C.E., Kutepov V.P., Shestakov D.A.** Yazyk Funktsional'nogo Parallel'nogo Programmirovaniya i Ego Realizatsiya na Klasternykh Sistemakh. Programmirovanie. 2005;5:237—269. (in Russian).

7. **Kutepov V.P., Shamal' P.N.** Realizatsiya Yazyka Funktsional'nogo Parallel'nogo Programmirovaniya FPTL na Mnogoyadernykh Komp'yuterakh. Izvestiya RAN. Seriya «Teoriya i Sistemy Upravleniya». 2014;3:46—60. (in Russian).

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 // Программные продукты и системы. 2014. № 2. С. 11—17.

10. **Ахо А.В., Лам М.С., Сети Р., Ульман Д.Д**. Компиляторы: принципы, технологии и инструментарий. М.: Вильямс, 2018.

11. **Амосов А.А., Дубинский Ю.А., Копченова Н.В.** Вычислительные методы для инженеров. М.: Изд-во МЭИ, 2003.

8. **Kutepov V.P., Fal'k V.N.** Formy, Yazyki Predstavleniya, Kriterii i Parametry Slozhnosti Parallelizma. Programmnye Produkty i Sistemy. 2010;3:16—26. (in Russian).

9. **Bocharov I.A., Kutepov V.P., Shamal' P.N.** Sistema Tipovogo Kontrolya Programm na Yazyke Funktsional'nogo Programmirovaniya. Programmnye Produkty i Sistemy. 2014;2:11—17. (in Russian).

10. **Akho A.V., Lam M.S., Seti R., Ul'man D.D.** Kompilyatory: Printsipy, Tekhnologii i Instrumentariy. M.: Vil'yams, 2018. (in Russian).

11. **Amosov A.A., Dubinskiy Yu.A., Kopchenova N.V.** Vychislitel'nye Metody Dlya Inzhenerov. M.: Izd-vo MEI, 2003. (in Russian).

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's Degree Student of Applied Mathematics Dept., NRU MPEI, e-mail: zumisha@yandex.ru

Работа выполнена при поддержке: РФФИ (грант № 18-01-00548)
The work is executed at support: RFBR (grant No. 18-01-00548)

Конфликт интересов: авторы заявляют об отсутствии конфликта интересов
Conflict of interests: the authors declare no conflict of interest

Статья поступила в редакцию: *18.11.2018*
The article received to the editor: *18.11.20180*