УДК 004.021

МЕТОДИКА ПРИМЕНЕНИЯ ПСЕВДОПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ

Лелейкин С.С.

ОАНО ВО «Московский технологический институт», Москва, e-mail: ssoft@mail.ru

В данной статье автор предлагает ознакомиться с методикой применения псевдопараллельных вычислительных алгоритмов при решении задач, требующих от программного обеспечения непрерывной работы в режиме реального времени в случае, когда архитектура исполняющей вычислительной машины не позволяет реализовать многопроцессность или мультипоточность. Подобные случаи часто возникают при необходимости оптимизации использования вычислительных мощностей и ограничении их выбора ввиду специфики решения конкретной задачи. Методика базируется на подходе, характеризующемся использованием алгоритма, подразумевающего наличие в программе базового цикла, контролирующего счетчики и связанные с ними триггеры, при срабатывании вызывающие задачи. В зависимости от времени, затрачиваемого на полезную нагрузку, происходит коррекция вызова триггеров, что позволяет программе контролировать и соблюдать время запуска задач. Данная методика была успешно использована автором при разработке и внедрении в промышленную среду системы термоконтроля серверных комнат, выполненной с применением таких компонентов, как микроконтроллерная сборка NodeMCU (ESP8266) и цифровой датчик температуры и влажности DHT11. Задача была выполнена с максимальной утилизацией вычислительных мощностей и рациональным использованием выделенного бюджета, что подкрепляет вышеуказанную методологию практической базой и повышает уровень потенциальной целесообразности ее внедрения и применения в промышленных средах.

Ключевые слова: параллелизм, алгоритмы, прикладная информатика, вычисления, системы реального времени

METHODOLOGY OF APPLICATION OF PSEUDO-PARALLEL ALGORITHMS

Leleykin S.S.

Moscow Technological Institute, Moscow, e-mail: ssoft@mail.ru

In this article, the author suggests getting acquainted with the methodology for using pseudo-parallel computing algorithms in solving problems that require software to work continuously in real time when the architecture of the executing computer does not allow for multiprocessing or multithreading. Such cases often arise when it is necessary to optimize the use of computing power and limit their choice due to the specifics of solving a particular problem. The technique is based on an approach characterized by the use of an algorithm that implies the presence of a basic loop in the program that controls counters and related triggers that trigger tasks. Depending on the time spent on the payload, trigger calls are corrected, which allows the program to monitor and observe the start time of tasks. This technique has been successfully used by the author in the development and implementation in the industrial environment of a thermal monitoring system for server rooms, made using components such as the NodeMCU microcontroller assembly (ESP8266) and the DHT11 digital temperature and humidity sensor. The task was completed with maximum utilization of computing power and rational use of the allocated budget, which supports the above methodology with a practical base and increases the level of potential expediency of its implementation and application in industrial environments.

Keywords: concurrency, algorithms, applied computer science, computing, real-time systems

Введение

В данной статье автор описывает методику применения псевдопараллельных вычислительных алгоритмов, позволяющую решать задачи, требующие обработки информации/вычислений/получения сигналов/изменения состояний в режиме реального времени параллельно друг с другом, но с применением электронной вычислительной машины, архитектура которой подразумевает работу в однопоточном/однопроцессорном режиме.

Автором были изучены и проанализированы существующие публикации по тематике «Параллельные вычисления» [1–3]. Анализ показал косвенное упоминание подобия псевдопараллельных вычислений некоторыми авторами вышеуказанных статей, но подробный разбор в статьях не приводится, что подчеркивает необходимость изучения и описания научным языком данной тематики для дальнейшего ее применения в промышленных средах.

Данная методика основывается на построении работы программы таким образом, при котором основной алгоритм – повторяющийся цикл с жестко заданным временем выполнения и набором счетчиков, количество которых равно количеству выполняемых задач.

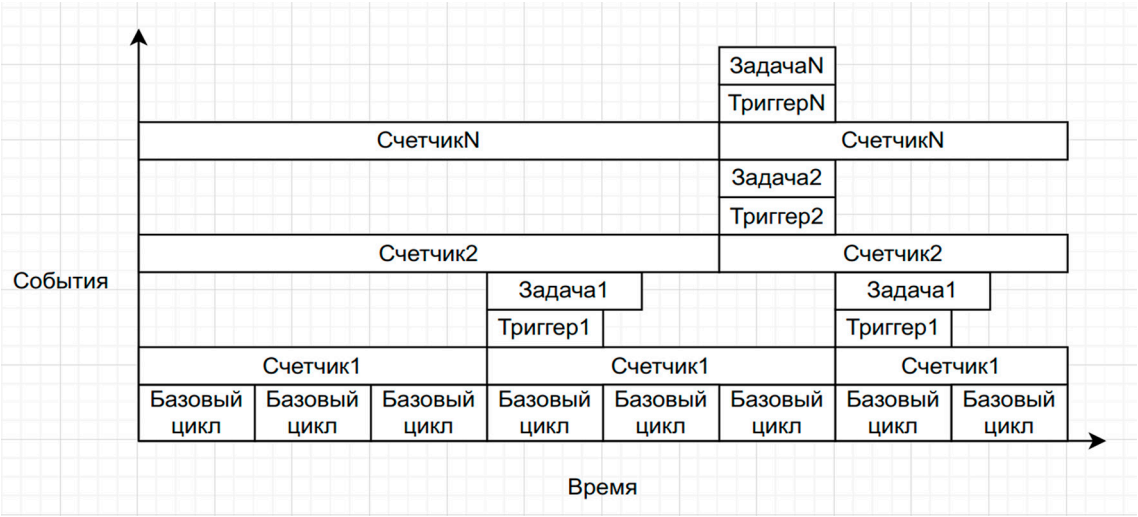

Рис. 1. Диаграмма Ганта, отражающая принцип работы псевдопараллельного алгоритма Горизонтальная плоскость является временной шкалой, вертикальная – отражает слои происходящих в тот или иной отрезок времени событий. Нижний слой содержит блоки повторения базового цикла, над ними расположены слои счетчиков, связанных с ними триггеров и выполняемых задач
Источник: составлено автором на основе анализа задачи и последующего синтеза структуры целевого алгоритма

Внутри цикла реализованы проверки, базирующиеся на состоянии счетчиков и, при достижении ими определенных значений, вызывающие необходимые функции. Счетчики инкрементируются при каждом проходе цикла. Так же алгоритм поддерживает коррекцию в зависимости от затраченного на выполнение задач времени. Диаграмма работы алгоритма отражена на рис. 1.

Цель исследования – освоить и описать научным языком методику применения псевдопараллельных алгоритмов для последующего внедрения в промышленные контуры.

Материалы и методы исследования

Изначально исследования проводились автором на базе процесса разработки системы термоконтроля серверных комнат методом анализа, синтеза и моделирования, но довольно быстро были масштабированы и на другие случаи решения задач, связанных с автоматизацией различных процессов (внутренняя автоматизация организации).

В результате исследования было выявлено, что данный алгоритм имеет практическую ценность при построении автоматизированных вычислительных систем реального времени, требующих от вычислительных мощностей параллельного решения различного рода задач, легко применим и повторяем.

Для исследования автором применялись следующие технологии: микроконтроллерная платформа NodeMCU [4], DHT11, скриптовый язык программирования lua [5].

Для удобочитаемости, универсальности и простоты освоения в данной статье будут использованы примеры, реализованные на языке программирования Python [6] в силу его распространенности и легкости восприятия исходного кода.

Результаты исследования и их обсуждение

Результатом исследования является алгоритм, позволяющий реализовать параллельные вычисления на платформах, архитектурно не поддерживающих параллельный запуск нескольких процессорных команд (последовательная платформа).

Итак, непосредственно алгоритм:

1. Базовый цикл. Алгоритм строится вокруг основного цикла, выполняющегося бесконечно. Цикл имеет жестко заданный период выполнения. Внутри цикла находятся счетчики и логика их инкрементирования. Такой цикл в статье будет называться базовым циклом.

В интерпретации языка программирования Python [6] аналогом является следующий код:

```
import time
cyclePeriod = 0.001
counter1 = 0
```

```
counter2 = 0
counter... = 0
while(True):
   counter1 += 1
   counter2 += 1
   counter... += 1
… код с функциональной логикой ...
   time.sleep(cyclePeriod)
```

В приведенном примере период равен одной миллисекунде. Счетчики через 1 с примут значение 1000, через 2 с – 2000 и т.д.

2. Задачи. Условия вызова задач определяются внутри тела базового цикла. Каждая задача – функция, на выполнение которой уходит определенное количество времени и которая занимает определенное количество процессорных тактов [7]. Вызов осуществляется при достижении счетчиком функции определенного значения (срабатывает триггер). После выполнения функции счетчик сбрасывается в значение 0.

Расчет триггера выполняется следующим способом:

$$n_{\text{триггер}} = T_{\text{функции}} / T_{\text{базовогоцикла}},$$

где $n_{\text{триггер}}$ – результирующее значение триггера, $T_{\text{функции}}$ – желаемый период запуска функции, $T_{\text{базовогоцикла}}$ – период выполнения базового цикла.

Пример, реализованный на языке программирования Python [6] в контексте кода приведенного в 1 разделе:

```
... тело цикла
   # Задача 1 (1 раз в секунду)
   if counter1 >= 1000:
      print('1 раз в секунду')
      counter1 = 0

   # Задача 2 (2 раза в секунду)
   if counter2 >= 500:
      print('2 раза в секунду')
      counter2 = 0
… тело цикла
```

При достижении первым счетчиком значения больше или равно 1000 будет выполнена первая задача, счетчик будет возвращен в значение 0. Аналогично отработает вторая задача при достижении вторым счетчиком значения 500 или больше.

3. Компенсация задержек. Глобальная корректировка счетчиков в результате возникновения задержек при выполнении задач является необходимостью в случае, когда программа должна выполнять задачи с соблюдением временных периодов. Например, первая задача – строго каждые 500 мс, вторая задача – строго каждые 1000 мс и т.д. [8]

За счет сокращения времени цикла достигается отсутствие сдвига из-за затрачиваемого на выполнение задач времени. Если необходимости в соблюдении жестких временных рамок при запуске задач нет, реализация данной функциональности необязательна, в таком случае периоды между запусками задач будут отличаться в зависимости от количества их вызовов и времени обработки, проще говоря время старта каждой задачи будет «плавать» в зависимости от времени, затраченного на выполнение остальных задач.

Первый этап: необходимо определить время, затраченное на выполнение полезной нагрузки задачи и зафиксировать его. Надежным и автоматическим способом является снятие временной метки непосредственно перед запуском задачи и сразу после ее выполнения. Разница временных меток позволит получит время выполнения задачи:

$$T_{\text{задачи}} = T_{\text{выполнение}} – T_{\text{запуск}},$$

Второй этап: при срабатывании триггера и выполнении вышеуказанной задачи добавить ко всем счетчикам компенсационное значение, равное отношению $T_{\text{задачи}}$ к $T_{\text{базовогоцикла}}$

$$n_{\text{компенс}} = T_{\text{задачи}} / T_{\text{базовогоцикла}},$$

Простыми словами, время до срабатывания всех триггеров сократится на количество тактов, соответствующее времени, которое выполнялась запущенная задача. Таким образом, произойдет компенсация времени выполнения задачи за счет сокращения тактов до запуска триггеров всех задач.

На языке программирования Python [6] описанный выше алгоритм реализуется следующим образом:

```
   ...
cyclePeriod = 0.001
counter = [0, 0, 0]
# Получение времени в миллисекундах
def current_milli_time():
   return round(time.time() * 1000)
# Корректировка счетчиков
def counterAdjustment(correction, counter):
   counter = [x + correction for x in counter]
   print(f'Корректировка: {correction}')
   return counter
… внутри цикла:
   # Задача 1 (1 раз в секунду)
   if counter[0] >= 1000:
      preMillis = current_milli_time()
      print('1 раз в секунду')
      … полезная нагрузка ...
         counter = counterAdjustment(current_
milli_time() – preMillis, counter)
      counter[0] = 0
```

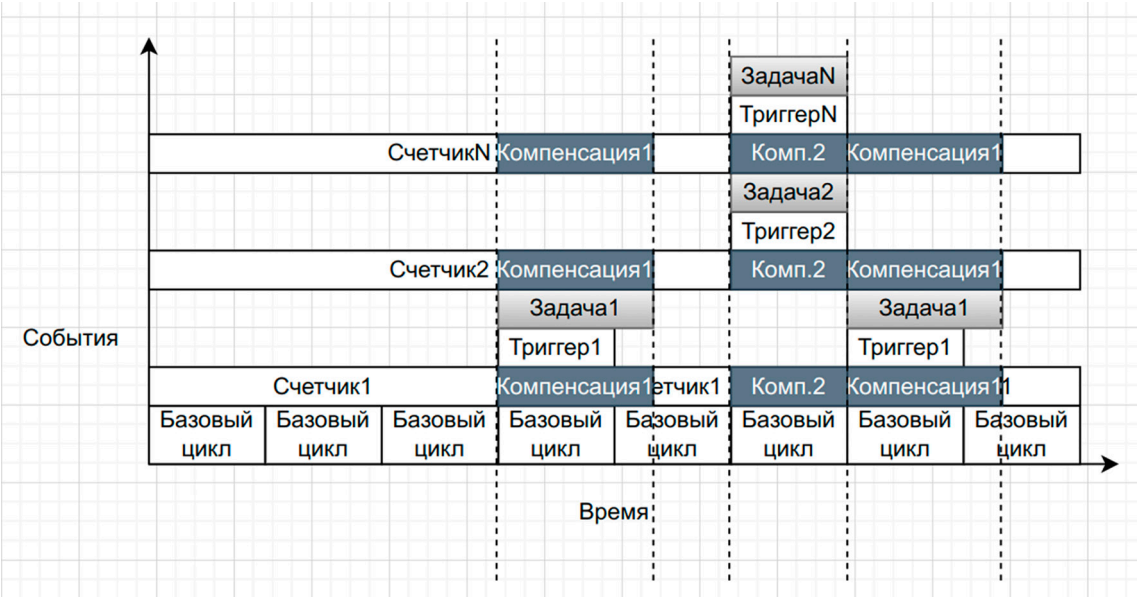

Рис. 2. Диаграмма Ганта, отражающая суть компенсации значения счетчиков при возникновении задержек во время выполнения задач. От блоков задач проведены вертикальные прерывистые линии, иллюстрирующие проекцию времени выполнения задач на счетчики
Источник: составлено автором на основе реализованного алгоритма с интегрированной компенсацией времени выполнения задач

В приведенном примере Тбазового-цикла равен 1 мс, соответственно, расчет nкомпенс достаточно прост: количество тактов на сокращение равно количеству мс, затраченному на выполнение задачи.

Принцип работы компенсационного алгоритма проиллюстрирован на рис. 2.

4. Выполнение задач в фоновом режиме или с отложенными эффектами. При выполнении кода в теле задачи нельзя использовать функции задержки (в случае Python [6] – time.sleep()), так как вызов такой функции полностью прервет выполнение программы на заданный в параметрах период, что остановит базовый цикл и нарушит принцип работы алгоритма.

При необходимости ожидания какого-либо условия или отсчета заданного периода времени внутри задачи, используется подход с флагом запуска ожидания и связкой триггер-счетчик, работающих параллельно с другими задачами и срабатывающих при выполнении заданных условий.

Пример вышеописанного алгоритма, реализованный на языке программирования Python [6]:

```
… Тело цикла
   # Задача 3 (запускается каждые 3 секунды, ждет 10 секунд, выполняется)
   if waitTaskFlag == False and counter[2] >= 3000: # Запуск ожидания
      print('Ожидающая задача запущена')
      waitTaskFlag = True

   if waitTaskFlag == True and waitCounter < 10000: # Ожидание, пока счетчик меньше 10000
      waitCounter += 1

   if waitTaskFlag == True and waitCounter >= 10000: # Запуск задачи по истечении 10 секунд
после запуска
      print('Ожидающая задача выполнена спустя 10 секунд после запуска')
      waitTaskFlag = False
      waitCounter = 0
      counter[2] = 0
… Тело цикла
```

Этапы алгоритма реализации задержки
при необходимости ее выполнения внутри задачи

Шаг	Описание	Условия выполнения
1	Запуск ожидания Поднимает флаг ожидания	Если флаг ожидания опущен и триггер базового счетчика сработал (при необходимости)
2	Ожидание Инкрементирует счетчик ожидания	Если флаг ожидания поднят и триггер счетчика ожидания не сработал
3	Запуск задачи Сброс флага, сброс базового счетчика и счетчика ожидания	Если флаг ожидания поднят и триггер счетчика ожидания сработал

Источник: составлено автором на основе подхода собственной разработки, функционирующего за счет системы, состоящей из флага запуска ожидания и связки триггер – счетчик, работающих параллельно с другими задачами и срабатывающих при выполнении заданных условий. Назначение: более структурированно представить алгоритм, описанный выше.

Алгоритм, лежащий в основе данного кода, описан более подробно в таблице.

5. Пример кода. Ниже приведен полный листинг кода программы, основанной на использовании псевдопараллельного вычислительного алгоритма.

```
import time
import math

cyclePeriod = 0.001
counter = [0, 0, 0]
waitTaskFlag = False
waitCounter = 0

# Получение времени в миллисекундах
def current_milli_time():
    return round(time.time() * 1000)

# Корректировка счетчиков
def counterAdjustment(correction, counter):
    counter = [x + correction for x in counter]
    print(f'Корректировка: {correction}')
    return counter

# Основной цикл
while(True):
    # Инкремент счетчика
    counter = [x + 1 for x in counter]

    # Задача 1 (1 раз в секунду)
    if counter[0] >= 1000:
        preMillis = current_milli_time()
        print('1 раз в секунду')

        for x in range(100000):
            temp = math.cos(999999)

        counter = counterAdjustment(current_milli_time() – preMillis, counter)
        counter[0] = 0

    # Задача 2 (2 раза в секунду)
    if counter[1] >= 500:
        preMillis = current_milli_time()
        print('2 раза в секунду')
        counter = counterAdjustment(current_milli_time() – preMillis, counter)
        counter[1] = 0
```

```
    # Задача 3 (запускается каждые 3 секунды, ждет 10 секунд, выполняется)
    if waitTaskFlag == False and counter[2] >= 3000: # Запуск ожидания
        print('Ожидающая задача запущена')
        waitTaskFlag = True

    if waitTaskFlag == True and waitCounter < 10000: # Ожидание, пока счетчик меньше 10000
        waitCounter += 1

    if waitTaskFlag == True and waitCounter >= 10000: # Запуск задачи по истечении 10 секунд
после запуска
        print('Ожидающая задача выполнена спустя 10 секунд после запуска')
        waitTaskFlag = False
        waitCounter = 0
        counter[2] = 0

    time.sleep(cyclePeriod)
```

В коде приведенной программы импортированы две библиотеки: time и math. Обе библиотеки входят в базовую поставку дистрибутива Python, применяются для работы со временем (time) и выполнения математических операций (math). Библиотеки, предназначенные для многопоточных и многоядерных вычислений [9] (например, популярные библиотеки threads и multiprocessing), в программе не использованы.

Заключение

Разработанная и описанная автором статьи методика, построенная на псевдопараллельном подходе к разработке программного обеспечения, включающая такие понятия, как базовый цикл, счетчики, триггеры, корректировка и задачи, успешно зарекомендовала себя при практическом использовании в промышленной среде и рекомендуется автором к освоению и повторению.

Новизна результатов исследования подтверждается отсутствием ранее опубликованных минимально схожих материалов, столь подробно описывающих применение подобных алгоритмов в промышленной эксплуатации.

Представленный псевдопараллельный алгоритм применим в различных встраиваемых системах, требующих использования микроконтроллерных вычислительных платформ, но также может быть применен и при разработке программного обеспечения для других видов ЭВМ, например, при невозможности использования по ряду причин библиотек, реализующих многопоточные или многоядерные вычисления.

Список литературы

1. Клименко В.И. Параллельные вычисления и создание параллельных программ // Научные труды КубГТУ. 2016. № 15. URL: https://ntk.kubstu.ru/data/mc/0036/1298.pdf (дата обращения: 02.06.2025).

2. Волосова А.В. Параллельные методы и алгоритмы: учебное пособие М.: МАДИ 2020. [Электронный ресурс]. URL: https://lib.madi.ru/fel/fel1/fel20E533.pdf (дата обращения: 02.06.2025).

3. Хардиков М.В., Эминджонов Д.Е. Обзор архитектур многопроцессорных и многопоточных систем // Научный Лидер. 2025. № 6 (207). С. 46–48. URL: https://scilead.ru/media/journal_pdf_207.pdf (дата обращения: 12.05.2025).

4. Гусев В.В., Гусев И.В., Христофоров Р.П., Домрачева Т.С. Интернет вещей и устройства, подключаемые к интернету // Аллея науки. 2018. Т. 2. № 11 (27). С. 859–865. URL: https://alley-science.ru/domains_data/files/Journal_Dec18/2%20tom%20Dekabr.pdf?x14474 (дата обращения: 03.06.2025).

5. Волков В.Д. Сравнительный анализ языков программирования на основе решения тестовой задачи сортировки данных // Вестник РГГУ. Серия «Информатика. Информационная безопасность. Математика». 2023. № 3. С. 61–70. URL: https://www.rsuh.ru/vestnik/digest/informatika-informatsionnaya-bezopasnost-matematika/informatika-informatsionnaya-bezopasnost-matematikat-3-2023.php (дата обращения: 12.05.2025).

6. Таршхоева Ж.Т. Язык программирования Python. Библиотеки Python // Молодой ученый. 2021. № 5 (347). С. 20–21. URL: https://moluch.ru/archive/347/78102/ (дата обращения: 12.05.2025).

7. Фролов В., Галактионов В., Санжаров В. RISC-V: стандарт, изменивший мир микропроцессоров // Открытые системы. СУБД. 2020. № 2. С. 30–34. URL: https://www.osp.ru/os/2020/02/13055471 (дата обращения: 12.05.2025).

8. Глонина А.Б. Обобщенная модель функционирования модульных вычислительных систем реального времени для проверки допустимости конфигураций таких систем // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2017. Т. 6. № 4. С. 43–59. URL: https://cyberleninka.ru/article/n/obobschennaya-model-funktsionirovaniya-modulnyh-vychislitelnyh-sistem-realnogo-vremeni-dlya-proverki-dopustimosti-konfiguratsiy (дата обращения: 12.05.2025).

9. Шавтикова Л.М., Текеев М.Б. Параллельное вычисление потоков на языке программирования Python // Тенденции развития науки и образования. 2020. № 68–1. С. 153–156. URL: https://doicode.ru/doifile/lj/68/lj-12-2020-46.pdf (дата обращения: 12.05.2025).