МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт Кибернетики

Базовая кафедра №252 - Информационной безопасности

КУРСОВАЯ РАБОТА

Тема курсовой работы: **«Методы синхронного распараллеливания циклических конструкций для SIMD-архитектуры»**

Студент группы ККСО-01-19 Никишина А.А. _____ (подпись)

Руководитель курсовой работы Лукинова О.В. _____ (подпись)

Работа представлена к защите «____» _____ 2020 г.

Допущен к защите «____» _____ 2020 г.

Москва 2020

Содержание

1. Введение

В данной курсовой работе рассмотрены методы синхронного распараллеливания циклических конструкций для SIMD-архитектуры. Эта тема является актуальной на сегодняшний день, так как люди создают всё более сложные программы, требующие множество времени для выполнения даже на современной технике. Большую часть времени программы тратят на выполнение циклов. Поэтому, для ускорения их работы и увеличения производительности вычислительной системы выполнение задач происходит параллельно. Это существенно оптимизирует процесс работы вычислительной машины.

Целью курсовой работы положим изучение методов синхронного распараллеливания циклических конструкций для SIMD-архитектуры.

В качестве задач были выбраны следующие пункты:

1) Рассмотрение типов архитектур вычислительных систем.

2) Рассмотрение методов распараллеливания в архитектурах.

3) Классификация методов синхронного распараллеливания циклических конструкций для SIMD-архитектуры.

2. Теоретическая часть

В данном разделе мы рассмотрим некоторые схемы работы архитектуры ЭВМ. Также познакомимся с классификацией типов ЭВМ, узнаем различные способы организации распараллеливания в вычислительных системах, а затем проведём сравнительный анализ этих способов и подведём итоги.

Архитектура ЭВМ - это общее описание структуры и функций ЭВМ на уровне, достаточном для понимания принципов работы и системы команд ЭВМ, не включающее деталей технического и физического устройства компьютера.

2.1. Классификация основных типов ЭВМ

Ниже приведена классификация основных типов ЭВМ по Флинну. Она берёт за предмет рассмотрения способы взаимодействия потоков команд и потоков данных. Разделяя вычислительные системы по такому признаку мы можем выделить следующие типы:

- **SISD** (Single Instruction, Single Data) – системы с одиночным потоком команд и одиночным потоком данных. К данному классу можно отнести вычислительные системы с последовательным характером выполнения команд.
- **SIMD** (Single Instruction, Multiple Data) – системы с одиночным потоком команд и множественным потоком данных. В качестве примера можно рассмотреть многопроцессорные системы, имеющие единое устройство управления.
- **MISD** (Multiple Instruction, Single Data) – системы с множественным потоком команд и одиночным потоком данных. Пример – систолические вычислительные системы или системы с конвейерной обработкой данных.

- **MIMD** (Multiple Instruction, Multiple Data) – системы с множественным потоком команд и множественным потоком данных. К данному типу относится очень широкий круг вычислительных систем - по сути большинство параллельных многопроцессорных систем.

В связи с классификацией для систематизации знаний можно составить таблицу классификации основных типов ЭВМ по Флинну.

	Одиночный поток команд	Множественный поток команд
Одиночный поток данных	SISD	MISD
Множественный поток данных	SIMD	MIMD

Таблица 1. «Классификация ЭВМ по Флинну»

2.2. ЭВМ на базе SIMD архитектуры

DM-SIMD-компьютеры «матричные процессоры» состоят из одного командного процессора (модуля управления), называемого контроллером, и нескольких модулей обработки данных - процессорных элементов. Управляющий модуль принимает, анализирует и выполняет команды. В том случае если в команде встречаются данные, контроллер рассылает на все процессорные элементы команду, и эта команда выполняется на нескольких или на всех процессорных элементах. Каждый из которых имеет свою собственную память для хранения данных.

Одним из преимуществ данной архитектуры считается то, что в этом случае более эффективно реализована логика вычислений. В обычном процессоре до половины логических инструкций связаны с управлением выполнением

машинных команд, а остальная их часть относится к работе с внутренней памятью процессора и выполнению арифметических операций. В SIMD-компьютере управление выполняется контроллером, а арифметические-логические операции отданы процессорным элементам. Примерами SIMD-машин являются системы CPP DAP, Gamma II и Quadrics Apemille.

Другим подклассом SIMD-систем являются векторные компьютеры (SM-SIMD). Векторные компьютеры манипулируют массивами сходных данных подобно тому, как скалярные машины обрабатывают отдельные элементы таких массивов. Это делается за счет использования специально сконструированных векторных центральных процессоров. Когда данные обрабатываются посредством векторных модулей, результаты могут быть выданы на один, два или три такта частотогенератора.

Циклы подлежащие векторизации должны отвечать ряду следующих требований:

- в цикле не должно быть вложенных циклов
- в теле цикла не должно быть ветвлений
- в теле цикла не должно быть вызовов процедур и функций
- в теле цикла не должно быть рекурсий

При работе в векторном режиме векторные процессоры обрабатывают данные практически параллельно, что делает их в несколько раз более быстрыми, чем при работе в скалярном режиме. Благодаря этому одной командой могут обрабатываться векторы размером до нескольких тысяч элементов. Примерами систем подобного типа являются, например, компьютеры Hitachi S3600.

Ниже схематично представлен принцип работы вычислительной системы под архитектурой SIMD.

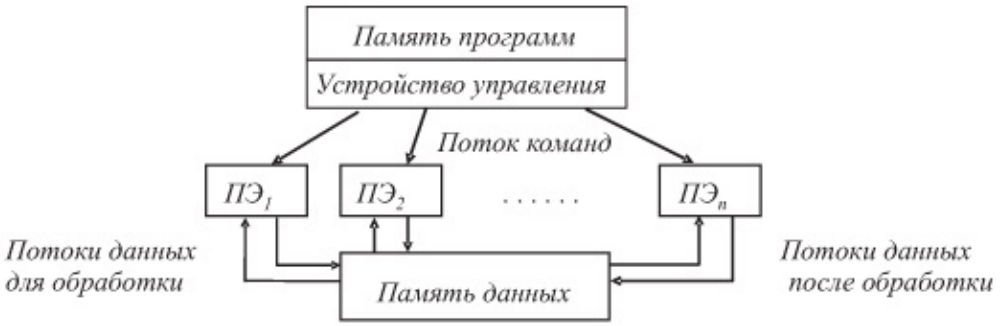

Рис. 1. «Схема работы SIMD-компьютера»

2.3. Классификация методов распараллеливания

Прежде чем начинать описание методов распараллеливания, напомним, что в нашем случае мы будем говорить о синхронном распараллеливании циклических конструкций.

То есть в общем случае мы рассматриваем последовательную программу, имеющую внутри себя массивный цикл, работу которого нам нужно оптимизировать для ускорения работы программы. Для детального понимания описываемых процессов ниже приведён небольшой глоссарий терминов описывающих рассматриваемую нами модель.

Синхронная программная модель – это такая программная модель, при реализации которой потоку назначается одна задача и начинается выполнение. Когда завершено выполнение задачи - тогда появляется возможность перейти к осуществлению другой задачи. В этой модели невозможно останавливать выполнение задачи, чтобы в промежутке выполнить другую задачу.

Распараллеливание – оптимизация программного кода, предназначенного для выполнения в системах с массовым параллелизмом.

Однопоточность – если мы имеем несколько задач, которые надлежит выполнить, и текущая система предоставляет один поток, который может работать со всеми задачами, то он берет поочередно одну за другой задачи и

реализует их последовательно. Данный процесс схематично изображён ниже:

Рис. 2. «Схема работы однопоточной ЭВМ»

Многопоточность – в этом сценарии, мы используем много потоков, которые могут брать задачи и приступать к работе с ними. У нас есть пулы потоков (новые потоки также создаются, основываясь на потребности и доступности ресурсов) и множество задач. Итак, поток может работать вот так:

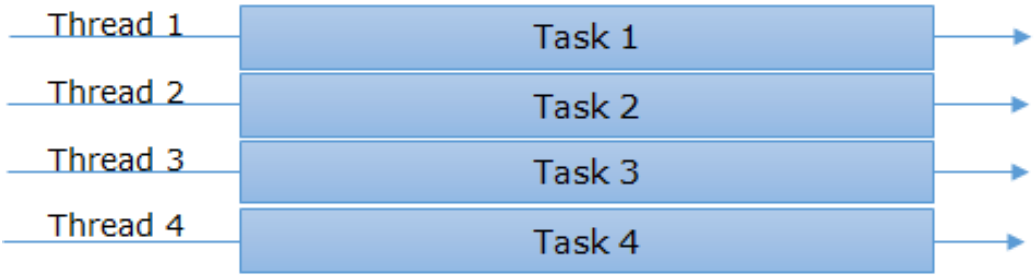

Рис. 3. «Схема работы многопоточной ЭВМ»

Здесь мы можем видеть, что у нас есть 4 потока и столько же задач для выполнения, и каждый поток начинает работать с ними. Это идеальный сценарий, но в обычных условиях мы используем большее количество задач чем количество доступных потоков, таким образом освободившийся поток получает другое задание. Как уже говорилось создание нового потока не происходит каждый раз потому что для этого требуются системные ресурсы такие как процессор, память и начальное количество потоков должно быть определенным.

Итак, зарезюмировав базовые понятия, на которых основывается наша система, можем перейти непосредственно к классификации методов синхронного распараллеливания циклических конструкций для SIMD-архитектуры.

Существуют три основных способа классификации методов распараллеливания последовательных программ:

1) По способу организации.

2) По степени участия человека.

3) По способу реализации.

2.3.1. Способы организации

- Распараллеливание на уровне данных.
  Исходный массив данных разбивается на несколько фрагментов, которые распределяются между процессорами. Команды выполняются одновременно над несколькими элементами массива данных на разных процессорах.

- Распараллеливание на уровне команд.
  Исходная программа разбивается на несколько подпрограмм и каждый процессор выполняет свою подпрограмму.

2.3.2. Степень участия человека

- Ручное распараллеливание.
  Программист сам выделяет параллельные блоки и сам занимается синхронизацией вычислений. Ручное распараллеливание требует от программиста знания языков параллельного программирования, а также архитектуры системы, для которой пишется программа. Методы программирования для различных параллельных вычислительных систем существенно отличаются друг от друга, что делает процесс ручного распараллеливания весьма трудоемким и длительным.
  Существуют различные пакеты для параллельного программирования. Некоторые из них помогают программисту, самостоятельно выделяя параллельные блоки, но, как правило, языки параллельного программирования ограничиваются специфическими моделями параллелизма, ко-

торые не всегда подходят для создаваемой программы. При ручном распараллеливании безусловно можно добиться высокой эффективности конечного кода, но процесс ручного распараллеливания является очень трудоемким.

- Полуавтоматическое распараллеливание.
  В случае полуавтоматического распараллеливания компилятор руководствуется директивами программиста при трансляции кода, но вопросы синхронизации решает сам, автоматически. Все полуавтоматические методы требуют взаимодействия между программистом и распараллеливающим компилятором. Программист обязан сам указывать потенциально параллелизуемые области.
  Примером такой системы является система ParaScope. В системе поддерживается визуальное представление параллелизма для облегчения его идентификации.

- Автоматическое распараллеливание.
  При использовании данного способа компилятор выполняет распараллеливание сам, с учетом конфигурации конкретной системы. Такие методы реализованы в компиляторах, которые преобразовывают исходные последовательные программы в параллельную форму (Paradigm, SUIF, RHODOS). Преимуществом таких компиляторов является снижение требований к программисту. Программист пишет программу на последовательном языке, а компилятор сам обнаруживает и выделяет параллелизм в программе.
  Использование автоматического распараллеливания сокращает время разработки параллельного кода, но эффективность полученных таким способом параллельных программ может быть достаточно низкой.

2.3.3. Способы реализации

- Программные методы распараллеливания.
  Данные методы являются низкоуровневыми и отличаются, как правило, временной избыточностью.

- Аппаратные методы распараллеливания.

  Среди этих методов сформировались два направления: реализация аппаратного распараллеливателя на ПЛИС и на одном кристалле с процессором. Такие устройства получили намного меньшее распространение, так как имеют узкую направленность и предназначены для работы в конкретной вычислительной системе.

2.4. Анализ методов распараллеливания

Исходя из вышеизложенного подведём промежуточный итог по собранной информации о теоретической классификации способов распараллеливания процессов в циклических конструкциях для SIMD-архитектур.

Анализ известных методов распараллеливания позволяет сделать вывод о том, что существующие методы распараллеливания последовательных программ, в основном, имеют программную реализацию, что приводит к дополнительной нагрузке на хост-процессор вычислительной системы.

3. Практическая часть

В данном разделе мы разберём программную реализацию методов синхронного распараллеливания. В качестве примера возьмём задачу на нахождение суммы элементов массива. Итак, приведём реализацию нашей задачи несколькими методами а языке программирования C# а после подведём итоги и составим сравнительную таблицу.

Первый способ - самый простой и очевидный:

```
1 public int Naive() {
2     int result = 0;
3     foreach (int i in Array) {
4         result += i;
5     }
6     return result;
7 }
```

Второй - с использованием LINQ:

```
1 public long LINQ() => Array.Aggregate<int, long>(0,
(current, i) => current + i);
```

Для третьего метода реализации используем векторы из System.Numerics:

```
1 public int Vectors() {
2     int vectorSize = Vector<int>.Count;
3     var accVector = Vector<int>.Zero;
4     int i;
5     var array = Array;
6     for (i = 0; i < array.Length - vectorSize; i += vectorSize) {
7         var v = new Vector<int>(array, i);
8         accVector = Vector.Add(accVector, v);
9     }
10     int result = Vector.Dot(accVector, Vector<int>.One);
```

```
11     for (; i < array.Length; i++) {
12         result += array[i];
13     }
14     return result;
15 }
```

Четвёртый реализован с использованием кода из пространства System.Runtime.Intrinsics:

```
1 public unsafe int Intrinsics() {
2     int vectorSize = 256 / 8 / 4;
3     var accVector = Vector256<int>.Zero;
4     int i;
5     var array = Array;
6     fixed (int* ptr = array) {
7         for (i = 0; i < array.Length - vectorSize; i += vectorSize) {
8             var v = Avx2.LoadVector256(ptr + i);
9             accVector = Avx2.Add(accVector, v);
10         }
11     }
12     int result = 0;
13     var temp = stackalloc int[vectorSize];
14     Avx2.Store(temp, accVector);
15     for (int j = 0; j < vectorSize; j++) {
16         result += temp[j];
17     }
18     for (; i < array.Length; i++) {
19         result += array[i];
20     }
21     return result;
22 }
```

Запустив на компьютере бенчмарк на эти 4 метода можно увидеть следующий результат:

Method	ItemsCount	Median
Naive	10	75.12 ns
LINQ	10	1 186.85 ns
Vectors	10	60.09 ns
Intrinsics	10	255.40 ns
Naive	100	360.56 ns
LINQ	100	2 719.24 ns
Vectors	100	60.09 ns
Intrinsics	100	345.54 ns
Naive	1000	1 847.88 ns
LINQ	1000	12 033.78 ns
Vectors	1000	240.38 ns
Intrinsics	1000	630.98 ns
Naive	10000	18 403.72 ns
LINQ	10000	102 489.96 ns
Vectors	10000	7 316.42 ns
Intrinsics	10000	3 365.25 ns
Naive	100000	176 630.67 ns
LINQ	100000	975 998.24 ns
Vectors	100000	78 828.03 ns
Intrinsics	100000	41 269.41 ns

Таблица 2. «Анализ методов реализации»

Можем заметить, что решения с Vectors и Intrinsics очень сильно выигрывают в скорости по сравнению с очевидным решением и с LINQ. Что наглядно показывает эффективность распараллеливания.

4. Заключение

В данной курсовой работе были изучены методы синхронного распараллеливания циклических конструкций для SIMD-архитектуры. Так же была приведена классификация этих методов и сравнительный анализ эффективности программной реализации.

Наблюдение за существующими вычислительными системами показало, что высокая скорость работы параллельных вычислительных систем достигается за счет параллельного выполнения операций. Время выполнения циклических участков составляет значительную часть времени выполнения всей программы, вследствие чего возникает необходимость распараллеливания циклических участков последовательных программ. Что доказывает актуальность рассматриваемой нами темы.

5. Список литературы

1) Алексеев Е.Г., Богатырев С.Д.: «Информатика. Мультимедийный электронный учебник», [Электронный ресурс] - http://inf.e-alekseev.ru/text/toc.html

2) Баканов В.М.: «Параллельные вычисления», 2006г.

3) Давыдов А.В.: «Архитектура параллельных вычислительных систем. Классификация.», 2013г.

4) Кудин А.В., Линёв А.В.: «Архитектура и операционные системы параллельных вычислительных систем», 2007г.

5) Ткачев П.Ю.: «Метод, алгоритм и устройство распараллеливания циклических участков последовательных программ для вычислительных систем высокой готовности», 2016г.

6) Цилькер Б.Я, Орлов С.А.: «Организация ЭВМ и систем», 2004г.

7) Вычислительные системы [Электронный ресурс] - https://studfile.net/preview/2204769/

8) Классификация параллельных вычислительных систем [Электронный ресурс] - https://ru.wikipedia.org/wiki/Классификация_параллельных_вычислительных_систем

9) Небольшой обзор SIMD в .NET/C# [Электронный ресурс] - https://habr.com/ru/post/435840/

10) Открытая распараллеливающая система [Электронный ресурс] - https://www.osp.ru/os/2007/09/4567122

11) Параллелизм против многопоточности против асинхронного программирования: разъяснение [Электронный ресурс] - https://habr.com/ru/post/337528/

12) Разновидности SIMD [Электронный ресурс] - https://habr.com/ru/post/441536/

13) Таксономия Флинна [Электронный ресурс] - https://ru.wikipedia.org/wiki/Таксономия_Флинна#SIMD

14) SIMD (Single Instruction, Multiple Data) [Электронный ресурс] - https://ru.bmstu.wiki/SIMD_(Single_Instruction,_Multiple_Data)