Жибарь Марк

Лабораторная работа №1 «Проверка закона Амдала с использованием OpenMP»

Оглавление

1. Цель работы

Получить навык анализа программ, выявления в них участков потенциального параллелизма с наибольшей трудоемкостью, применить для распараллеливания OpenMP.

2. Введение

Существуют различные способы написания программ, которые условно можно разделить на три группы:

1. Последовательное программирование с дальнейшим автоматическим распараллеливанием.
2. Непосредственное формирование потоков параллельного управления, с учетом особенностей архитектур параллельных вычислительных систем или операционных систем.
3. Описание параллелизма без использования явного управления обеспечивается заданием только информационных связей. Предполагается, что программа будет выполняться на вычислительных системах с бесконечными ресурсами, операторы будут запускаться немедленно по готовности их исходных данных.

Каждый из перечисленных подходов обладает своими достоинствами и недостатками параллельное программирование.

Параллельные вычисления - способ организации компьютерных вычислений, при котором программы разрабатываются, как набор взаимодействующих вычислительных процессов, работающих асинхронно и при этом одновременно.

Параллельное программирование - это техника программирования, которая использует преимущества многоядерных или многопроцессорных компьютеров и является подмножеством более широкого понятия многопоточности (multithreading).

Использование параллельного программирования становится наиболее необходимым, поскольку позволяет максимально эффективно использовать возможности многоядерных процессоров и многопроцессорных систем. По ряду причин, включая повышение потребления энергии и ограничения пропускной способности памяти, увеличивать тактовую частоту современных процессоров стало невозможно. Вместо этого производители процессоров стали увеличивать их производительность за счет размещения в одном чипе нескольких вычислительных ядер, не меняя или даже снижая тактовую частоту. Поэтому для увеличения скорости работы приложений теперь следует по-новому подходить к организации кода, а именно - оптимизировать программы под многоядерные системы.

3. Теория

3.1. Ускорение (Speedup)

Ускорением параллельного алгоритма называется отношение:

$$S_n = \frac{T_1}{T_n}$$

где T_n - время вычисления задачи на n процессорах, T_1 - время выполнения однопоточной программы

$T(n) < T(1)$, если параллельная версия алгоритма эффективна.

$T(n) > T(1)$, если накладные расходы (издержки) реализации параллельной версии алгоритма чрезмерно велики.

С ускорением связана эффективность параллельного алгоритма. Эффективностью параллельного алгоритма называется величина:

$$E_n = \frac{S_n}{n}$$

По определению, $E_1 = 1$. Теоретически должно быть $S_n \leq n$ и $E_n \leq 1$. Если алгоритм достигает максимального ускорения $(S_n = n)$, то $E_n = 1$. На практике эффективность убывает при увеличении числа процессоров.

Если же результат получается $E_n > 1$(суперлинейное ускорение). Эта аномалия вызвана, чаще всего, двумя причинами:

- В качестве последовательного алгоритма был применён не самый быстрый алгоритм из существующих.
- С увеличением количества вычислителей растёт суммарный объём их оперативной и кэш памяти. Поэтому всё большая часть данных задачи умещается в оперативной памяти и не требует подкачки с диска, или (чаще всего) умещается в кэше.

3.2. Закон Амдала

Зако́н Амдала (англ. *Amdahl's law*, иногда также *Закон Амдаля-Уэра*) — иллюстрирует ограничение роста производительности вычислительной системы с увеличением количества вычислителей. Джин Амдал сформулировал закон в 1967 году, обнаружив простое по существу, но непреодолимое по содержанию ограничение на рост производительности при распараллеливании вычислений: «В случае, когда задача разделяется на несколько частей, суммарное время её выполнения на параллельной системе не может быть меньше времени выполнения самого длинного фрагмента»[1]. Согласно этому закону, ускорение выполнения программы за счёт распараллеливания её инструкций на множестве вычислителей ограничено временем, необходимым для выполнения её последовательных инструкций.

Пусть необходимо решить некоторую вычислительную задачу. Предположим, что её алгоритм таков, что доля α от общего объёма вычислений может быть получена только последовательными расчётами, а, соответственно, доля $1 - \alpha$ может быть распараллелена идеально (то есть время вычисления будет обратно пропорционально числу задействованных узлов p). Тогда ускорение, которое может быть получено на вычислительной системе из p процессоров, по сравнению с однопроцессорным решением не будет превышать величины $S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}}$

Закон Амдала показывает, что прирост эффективности вычислений зависит от алгоритма задачи и ограничен сверху для любой задачи с $\alpha \neq 0$. Не для всякой задачи имеет смысл наращивание числа процессоров в вычислительной системе.

Более того, если учесть время, необходимое для передачи данных между узлами вычислительной системы, то зависимость времени вычислений от числа узлов будет иметь минимум. Это накладывает ограничение на масштабируемость вычислительной системы, то есть означает, что с определенного момента добавление новых узлов в систему будет увеличивать время расчёта задачи.

3.3. Основные принципы OpenMP

OpenMP - это интерфейс прикладного программирования для создания многопоточных приложений, предназначенных в основном для параллельных вычислительных систем с общей памятью. OpenMP состоит из набора директив для компиляторов и библиотек специальных функций.

OpenMP позволяет легко и быстро создавать многопоточные приложения на алгоритмических языках Fortran и C/C++. При этом директивы OpenMP аналогичны директивам препроцессора для языка C/C++ и являются аналогом комментариев в алгоритмическом языке Fortran. Это позволяет в любой момент разработки параллельной реализации программного продукта при необходимости вернуться к последовательному варианту программы.

В настоящее время OpenMP поддерживается большинством разработчиков параллельных вычислительных систем: компаниями Intel, Hewlett-Packard, Silicon Graphics, Sun, IBM, Fujitsu, Hitachi, Siemens, Bull и другими.

3.4. Принципиальная схема программирования в OpenMP

Любая программа, последовательная или параллельная, состоит из набора областей двух типов: последовательных областей и областей распараллеливания. При выполнении последовательных областей порождается только один главный поток (процесс). В этом же потоке инициируется выполнение программы, а также происходит ее завершение. В последовательной программе в областях распараллеливания порождается также только один, главный поток, и этот поток является единственным на протяжении выполнения всей программы. В параллельной программе в областях распараллеливания порождается целый ряд параллельных потоков. Порожденные параллельные потоки могут выполняться как на разных процессорах, так и на одном процессоре вычислительной системы. В последнем случае параллельные процессы (потоки) конкурируют между собой за доступ к процессору. Управление конкуренцией осуществляется планировщиком операционной системы с помощью специальных алгоритмов. Принципиальная схема параллельной программы изображена на рисунке 1.

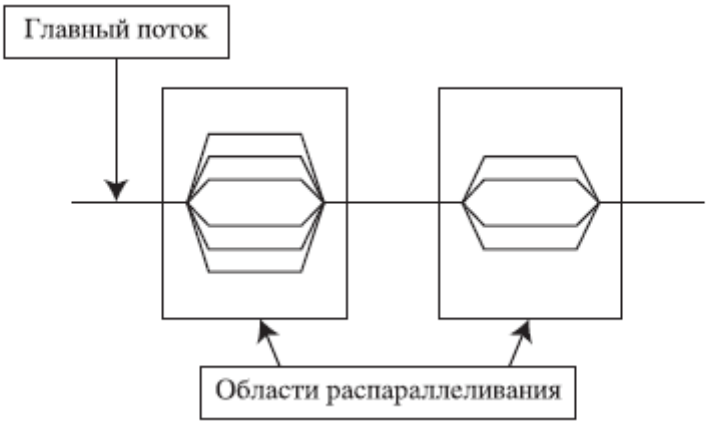

Рисунок 1 - Принципиальная схема параллельной программы

При выполнении параллельной программы работа начинается с инициализации и выполнения главного потока (процесса), который по мере необходимости создает и выполняет параллельные потоки, передавая им необходимые данные. Параллельные потоки из одной параллельной области

программы могут выполняться как независимо друг от друга, так и с пересылкой и получением сообщений от других параллельных потоков. Последнее обстоятельство усложняет разработку программы, поскольку в этом случае программисту приходится заниматься планированием, организацией и синхронизацией посылки сообщений между параллельными потоками. Таким образом, при разработке параллельной программы желательно выделять такие области распараллеливания, в которых можно организовать выполнение независимых параллельных потоков. Для обмена данными между параллельными процессами (потоками) в OpenMP используются общие переменные. При обращении к общим переменным в различных параллельных потоках возможно возникновение конфликтных ситуаций при доступе к данным. Для предотвращения конфликтов можно воспользоваться процедурой синхронизации (synchronization). При этом надо иметь в виду, что процедура синхронизации - очень дорогая операция по временным затратам и желательно по возможности избегать ее или применять как можно реже. Для этого необходимо очень тщательно продумывать структуру данных программы.

Выполнение параллельных потоков в параллельной области программы начинается с их инициализации. Она заключается в создании дескрипторов порождаемых потоков и копировании всех данных из области данных главного потока в области данных создаваемых параллельных потоков. Эта операция чрезвычайно трудоемка - она эквивалентна примерно трудоемкости 1000 операций. Эта оценка чрезвычайно важна при разработке параллельных программ с помощью OpenMP, поскольку ее игнорирование ведет к созданию неэффективных параллельных программ, которые оказываются зачастую медленнее их последовательных аналогов. В самом деле: для того чтобы получить выигрыш в быстродействии параллельной программы, необходимо, чтобы трудоемкость параллельных процессов в областях распараллеливания программы существенно превосходила бы трудоемкость порождения параллельных потоков. В противном случае

никакого выигрыша по быстродействию получить не удастся, а зачастую можно оказаться даже и в проигрыше.

После завершения выполнения параллельных потоков управление программой вновь передается главному потоку. При этом возникает проблема корректной передачи данных от параллельных потоков главному. Здесь важную роль играет синхронизация завершения работы параллельных потоков, поскольку в силу целого ряда обстоятельств время выполнения даже одинаковых по трудоемкости параллельных потоков непредсказуемо (оно определяется как историей конкуренции параллельных процессов, так и текущим состоянием вычислительной системы). При выполнении операции синхронизации параллельные потоки, уже завершившие свое выполнение, простаивают и ожидают завершения работы самого последнего потока. Естественно, при этом неизбежна потеря эффективности работы параллельной программы. Кроме того, операция синхронизации имеет трудоемкость, сравнимую с трудоемкостью инициализации параллельных потоков, т. е. эквивалентна примерно трудоемкости выполнения 1000 операций.

На основании изложенного выше можно сделать следующий важный вывод: при выделении параллельных областей программы и разработке параллельных процессов необходимо, чтобы трудоемкость параллельных процессов была не менее 2000 операций деления. В противном случае параллельный вариант программы будет проигрывать в быстродействии последовательной программе. Для эффективной работающей параллельной программы этот предел должен быть существенно превышен.

3.5. Синтаксис директив в OpenMP

Основные конструкции OpenMP - это директивы компилятора или прагмы (директивы препроцессора) языка C/C++. Ниже приведен общий вид директивы OpenMP прагмы.

#pragma omp конструкция [предложение [предложение] …]

Для обычных последовательных программ директивы OpenMP не изменяют структуру и последовательность выполнения операторов. Таким образом, обычная последовательная программа сохраняет свою работоспособность. В этом и состоит гибкость распараллеливания с помощью OpenMP.

Количество потоков в параллельной программе определяется либо значением переменной окружения OMP_NUM_THREADS, либо специальными функциями, вызываемыми внутри самой программы.

Каждый поток имеет свой номер thread_number, начинающийся от нуля (для главного потока) и заканчивающийся OMP_NUM_THREADS-1. Определить номер текущего потока можно с помощью функции omp_get_thread_num().

3.6. Загрузка процессов в OpenMP. Директива schedule

Для исключения простоев или уменьшения времени простоев процессоров применяют различные методы балансировки процессов. Существующие в OpenMP различные методы загрузки процессов также могут быть применены для улучшения балансировки работы параллельных вычислительных систем.

Для распределения работы между процессами в OpenMP имеется директива schedule с параметрами, позволяющими задавать различные режимы загрузки процессоров. Ниже приведен общий вид предложения schedule в OpenMP.

schedule(type [, chunk])

Здесь type - параметр, определяющий тип загрузки, а chunk - параметр, который определяет порции данных, пересылаемых между процессами (по умолчанию значение параметра chunk равно 1).

В OpenMP параметр type принимает одно из следующих значений:

- static,
- dynamic,

- guided,
- runtime.

Загрузка типа static:

В этом случае вся совокупность загружаемых процессов разбивается на равные порции размера chunk, и эти порции последовательно распределяются между процессорами (или потоками, которые затем и выполняются на этих процессорах) с первого до последнего и т. д.

Загрузка типа dynamic:

В этом случае вся совокупность загружаемых процессов, как и в предыдущем варианте, разбивается на равные порции размера chunk, но эти порции загружаются последовательно в освободившиеся потоки (процессоры).

4. Ход работы

В эксперименте реализуются операции умножения массивов A, B и сложение с C на конфигурации с процессором Intel Xeon E5 с 16 физическими ядрами. Будет определяться время выполнения программы в зависимости от изменения числа потоков и различных режимов загрузки потоков Static и Dynamic. Также переменной NUM будет меняться размер массивов.

В работе будет меняться размер массивов 300x300, 1000x1000 и 1300x1300, число потоков от 1 до 18 и режимы загрузки static и dynamic.

Проведём эксперимент при NUM=300:

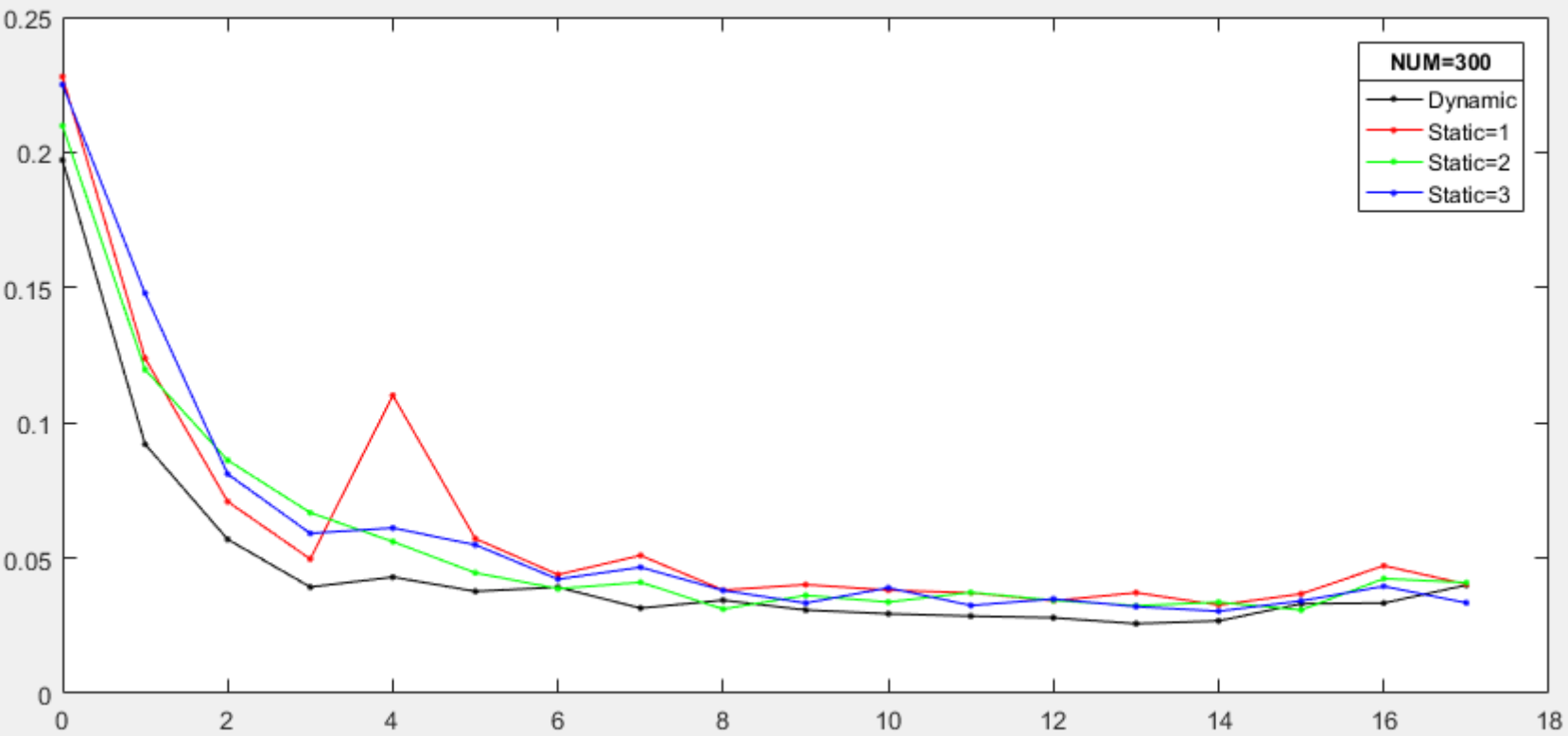

Рисунок 2 – Время выполнения программы с размерностью массивов 300x300

Заметно, что при относительно небольшой трудоёмкости вычислений, изменение режимов загрузки на скорость выполнения особо не влияет, но кривая режима Dynamic находится ниже остальных, о чем можно сказать, что в задачах малых объемов вычислений такой режим может подойти лучше всего.

Также можно увидеть небольшое ухудшение ускорения в режиме static=1 при пяти потоках, что говорит о попадании в точку рассинхронизации в распределении между порциями.

Проведем эксперимент при NUM=1000:

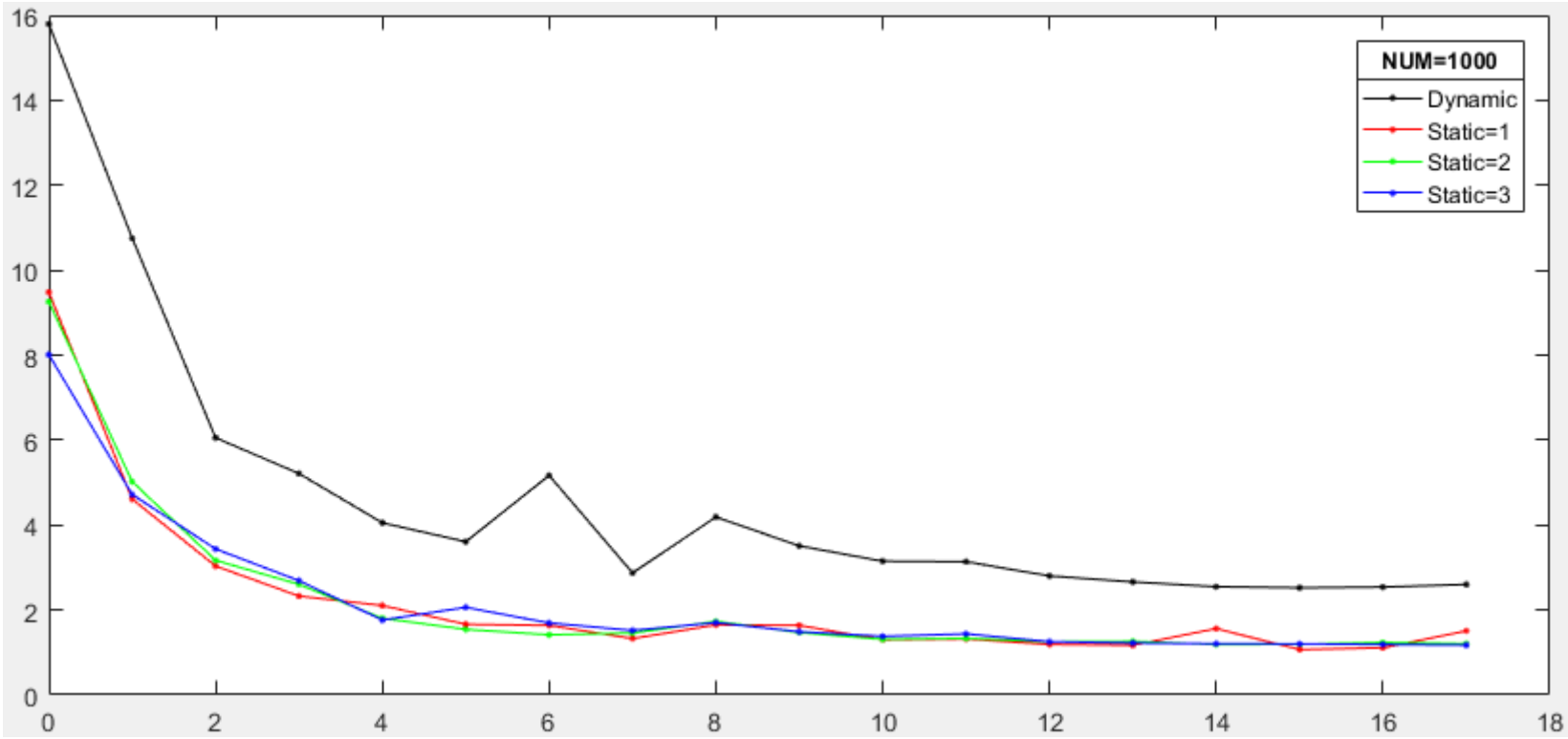

Рисунок 3 - Время выполнения программы с размерностью массивов 1000x1000

Здесь наиболее заметно преимущество управляемого распределения порций перед динамическим. Изменение размеров порций в режиме static влияет несущественно.

Проведём эксперимент при NUM=1300:

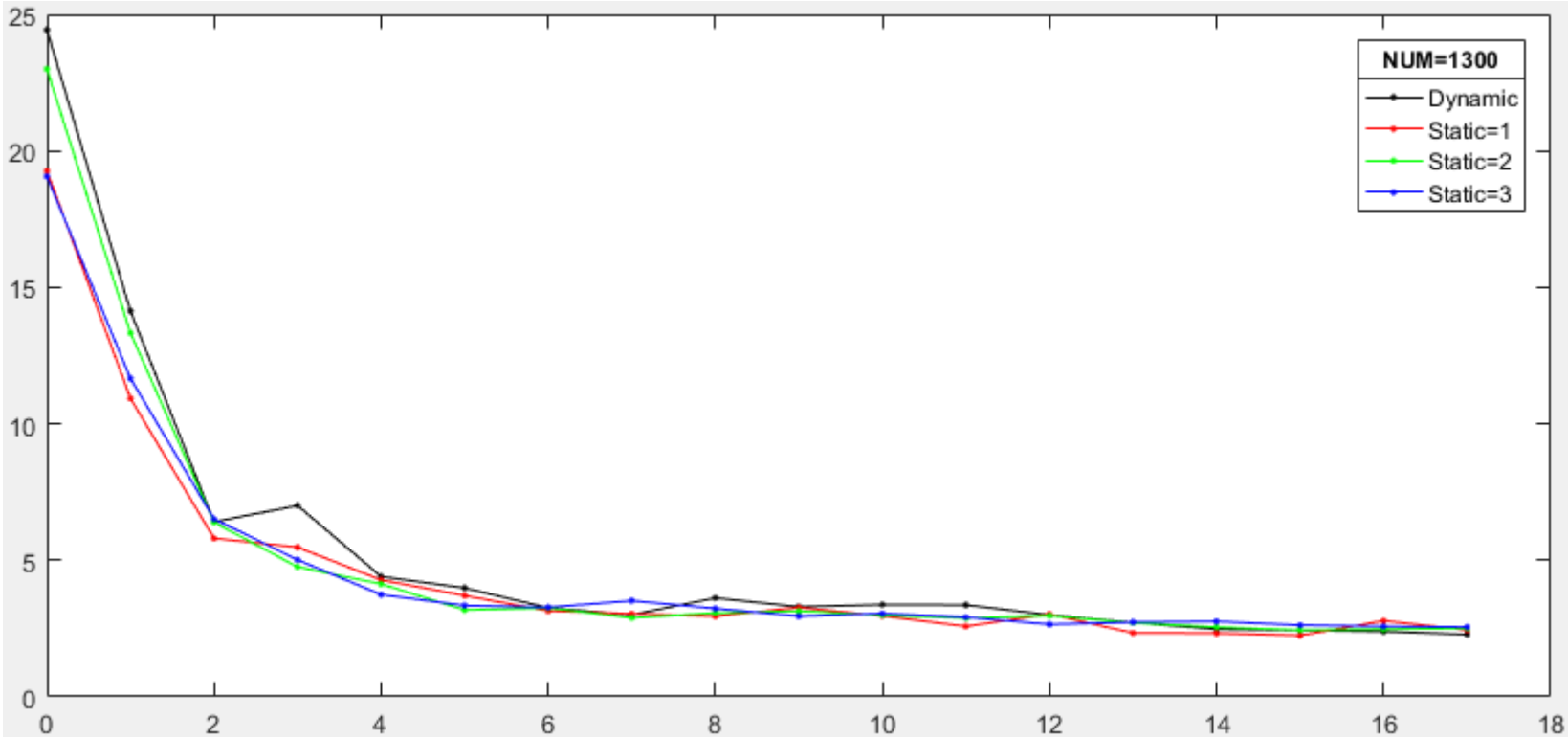

Рисунок 4 - Время выполнения программы с размерностью массивов 1300x1300

Время вычислений в зависимости от режимов static и dynamic вновь меняется слабо. Но динамическое распределение всё ещё уменьшает быстродействие, особенно при малом числе потоков.

5. Выводы

В результате выполнения лабораторной работы, был проверен второй закон Амдала, исследованы особенности изменения режимов загрузки процессоров с помощью директив в OpenMP.

Наглядно отражено ограничение распараллеливания программы, упирающееся во время вычисления последовательной её части. Для данной задачи заметно, что самым оптимальным значением будет являться 4 потока.

Динамический режим загрузки Dynamic выигрышен при малых объемах вычислений, так как в этом режиме затраты на управляемое распределение порций минимальны, в отличии от режима Static. А управляемое планирование повышает время выполнения программы на средних и высоких объемах вычислений, причём на средних наблюдается наибольший эффект, в отличии от динамического режима.

Литература

[1] Основы параллельного программирования с использованием Visual Studio 2010: http://www.intuit.ru/studies/professional_skill_improvements/10496/courses/1055/lecture/16369

[2] Параллельное программирование с использованием OpenMP: https://www.intuit.ru/studies/courses/1112/232/lecture/6021

[3] Закон Амдала — Википедия: https://ru.wikipedia.org/wiki/Закон_Амдала