Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра суперкомпьютеров и квантовой информатики

Отчёт по заданию:

Проблемы масштабируемости в OpenMP:

Многократное порождение/уничтожение нитей

Аксой Тевфик Огузхан
423 группа

OpenMP - это набор директив компилятора и библиотечных функций для параллельного программирования на многопроцессорных системах с общей памятью. Он предоставляет возможность разработчикам программного обеспечения использовать несколько процессоров в одной машине для ускорения выполнения задач, которые могут быть распараллелены.

OpenMP основывается на модели параллельного программирования с использованием общей памяти (shared-memory model), при которой несколько нитей (threads) могут одновременно обращаться к общей области памяти. Каждая нить выполняет свою задачу в параллельном режиме и синхронизируется с другими нитями при необходимости.

OpenMP позволяет разработчикам легко создавать параллельные программы, используя директивы компилятора и библиотеки, не требуя при этом специального оборудования или операционной системы. Однако, при масштабировании параллельной программы с помощью OpenMP, могут возникнуть проблемы масштабируемости и эффективности, связанными с накладными расходами на порождение и уничтожение нитей, накладнымы расходами на барьеры, неравномерная загрузка нитей, ожидание в точках синхронизации, законом Амдала (увеличение доли последовательных операций), неравномерной загрузкой процессов/нитей, ожидание в точках синхронизации, неравномерной загрузкой нитей, ограниченностью ресурса параллелизма и невозможностью одновременного выполнения множества операций.

Компиляция и запуск программы

Программа компилируется с помощью команды **`xlc -fopenmp code.c -o code.o`** **на Полюсе** и **`gcc -fopenmp code.c -o code.o`** локально. В ходе работы написал на языке Python скрипт, с помощью которого можно компилировать файл с кодом, написанный на языке C и создать нужный файл-планировщик с расширением LSF для запуска программы на Полюсе, а так же размещать программу на очередь вызывая команду **bsub**. Скрипт берёт номер задачи(варианта – в моём случае 1) в качестве аргумента(файлы хранятся в директории под названием номера варианта) и букву **“u”(unoptimized)** или **“o”(optimized)** для запуска неоптимизированный или оптимизированный версии программы.

Проблемы масштабируемости

Многократное порождение/уничтожение нитей

Проблема масштабируемости в OpenMP, связанная с накладными расходами на порождение/уничтожение нитей, проявляется при использовании небольших параллельных задач, когда время порождения и уничтожения нитей может превышать время выполнения самих задач.

При порождении нити выделяется ресурс для её выполнения, что сопровождается накладными расходами на выделение памяти, инициализацию, синхронизацию с другими нитями и т.д. Поэтому при выполнении небольших задач накладные расходы могут стать существенными по сравнению с временем выполнения самих задач, что приводит к снижению масштабируемости.

Еще одной причиной низкой масштабируемости при порождении/уничтожении нитей может быть использование неправильных параметров при создании параллельной области, например, создание слишком маленьких параллельных областей, которые не соответствуют структуре задачи.

Для решения этой проблемы можно использовать техники оптимизации, такие как сокращение числа порождаемых нитей, использование библиотеки OpenMP Task, объединение задач в более крупные задачи для уменьшения количества порождаемых нитей и т.д. Иногда создание и уничтожение нитей в OpenMP-программах может занимать значительное время. Это может происходить, когда количество создаваемых нитей меньше, чем количество физических ядер на процессоре, и время на создание и уничтожение нитей может превышать время выполнения вычислений на этих нитях.

Кроме того, частое создание и уничтожение нитей может привести к большим накладным расходам на синхронизацию и координацию между ними. Это может привести к снижению производительности программы и ухудшению ее масштабируемости при увеличении числа нитей.

Для решения этой проблемы можно использовать пул нитей, то есть заранее создать определенное количество нитей и использовать их повторно для выполнения задач. Это сокращает время на создание и уничтожение нитей и улучшает масштабируемость программы при увеличении числа нитей. Также можно использовать методы динамического управления нитями, которые позволяют программе эффективно использовать доступные ресурсы и улучшить масштабируемость.

Фрагмент программы, иллюстрирующиий проблему с масштабируемостью

```
  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
      double sum = 0;
#pragma omp parallel for private(k) shared(a, b, c) reduction(+ : sum)
      for (k = 0; k < N; k++) {
        sum += a[i][k] * b[k][j];
      }
      c[i][j] = sum;
    }
  }
```

В этом примере каждый раз, когда порождается параллельная область, создаются новые нити. В многомерном цикле это происходит для каждого выполнения внутреннего цикла, что приводит к значительным накладным расходам на порождение и уничтожение нитей.

Оптимизированная версия программы, выносящая порождение нитей выше по уровням вложенности гнезда циклов, выглядит следующим образом:

```
#pragma omp parallel private(i, j, k) shared(a, b, c)
  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
#pragma omp for reduction(+ : c[i][j])
      for (k = 0; k < N; k++) {
        c[i][j] += a[i][k] + b[k][j];
      }
    }
```

В оптимизированной версии мы вынесли порождение нитей на уровень выше по иерархии гнезда циклов, то есть на уровень цикла по переменной i. Это позволяет снизить накладные расходы на порождение/уничтожение нитей за счет того, что каждая нить будет выполнять более длительное время и не будет пересоздаваться каждый раз на каждой итерации внутреннего цикла.

Результаты выполнения программы должны быть идентичны для обеих версий. Однако, при увеличении размера массива и числа нитей, мы должны наблюдать улучшение масштабируемости и более высокую эффективность выполнения в оптимизированной версии.

Результаты

После запуска и обработки программы в Полюсе, файлы с результатами ***«1_unoptimized.out»*** и ***«1_optimized.out»*** на свой компьютер скопировал, дальше для составления графиков по результатам, перенёс данные от результатов в Excel с помощью скрипта. В Excel составил графики ускорения в зависимости от количества нитей для обеих версий. При N=1000 результаты:

MatrixSize	Threads	TIME(Unoptimized) (s)	TIME(Optimized) (s)	SPEEDUP(Unoptimized)	SPEEDUP(Optimized)
1000x1000	1	0,540117	0,197363	1,00	1,00
1000x1000	2	0,747433	0,102649	0,72	1,92
1000x1000	4	0,754871	0,055413	0,72	3,56
1000x1000	8	0,786182	0,033265	0,69	5,93
1000x1000	16	0,922865	0,027968	0,59	7,06
1000x1000	32	1,088393	0,025395	0,50	7,77