Нижегородский государственный университет им. Н.И. Лобачевского
Факультет вычислительной математики и кибернетики

Образовательный комплекс «Параллельные численные методы»

Описание технологии Intel Cilk Plus

Кустикова В.Д.

При поддержке компании Intel

Нижний Новгород
2011

Содержание

Введение

В данном документе приводится краткое описание технологии Intel® Cilk Plus. Акцент делается на механизмах распараллеливания циклических конструкций с известным числом повторений.

Распараллеливание циклов демонстрируется на простом и понятном примере умножения плотной матрицы на вектор. Рассматривается задача с квадратной матрицей. Ниже представлена последовательная реализация алгоритма умножения. При выполнении умножения вычисляются скалярные произведения строк матрицы на заданный вектор, поэтому данная операция вынесена в отдельную функцию **dotProd**.

```
double dotProd(const double *a, const double *b,
  const int n)
{
  double res = 0.0;
  for (int i = 0; i < n; i++)
  {
    res += a[i] * b[i];
  }
  return res;
}

void Multiplication(const double *A, const double *x,
  const int n, double *b)
{
  for (int i = 0; i < n; i++)
  {
    b[i] = dotProd(&(A[i * n]), x, n);
  }
}
```

1. Intel® Cilk Plus – расширение C/C++. Основные возможности Cilk Plus

Intel® Cilk Plus [1, 3] является расширением языка C++, которое упрощает разработку приложений, параллельных по задачам и данным в системах с общей памятью. Задачи могут быть реализованы в виде отдельных функций или итераций цикла. При этом система исполнения автоматически распределяет их по доступным ядрам.

Для обозначения потока операционной системы, который используется планировщиком Cilk Plus для исполнения задачи, вводится термин «*обработчик*» (*worker*). Набор инструкций, выполняемых последовательно, в Cilk Plus называется *«нитью»* (*strand*). Причем нити могут выполняться как в одном, так и в разных потоках.

Для технологии Cilk Plus характерна простота разработки параллельного приложения, что обусловлено интуитивно понятными синтаксическими конструкциями и ключевыми словами. Интеграция с инфраструктурой компилятора позволяет применять многочисленные компиляторные оптимизации к параллельному коду.

Cilk Plus включает четыре основные компоненты:

- *Ключевые слова* (**`cilk_for`**, **`cilk_spawn`**, **`cilk_sync`**). Эти конструкции предоставляют мощный инструмент для организации параллелизма по задачам.
- *Гипер-объекты* или *преобразователи* (reducers). Устраняют конкуренцию за переменные, разделяемые между задачами, автоматически создавая представление этих переменных для каждой задачи и собирая их обратно в разделяемое значение по завершении всех задач.
- *Специальное представление массивов* (C/C++ Extensions for Array Notation (CEAN)). Данный компонент обеспечивает параллелизм по данным (векторизацию) для всего массива или его частей при выполнении однотипной операции со всеми его элементами.
- *Элементарные функции*. Включают параллелизм по данным для всех функций и операций, которые применяются к специальным массивам или их частям.

Таким образом, разработчики Cilk Plus предоставляют средства для организации циклического и рекурсивного («разделяй и властвуй») параллелизма.

2. Подключение возможности использования Intel® Cilk Plus

Intel® Cilk Plus поддерживается компилятором **Intel® Windows C/C++ Compiler**. Поэтому чтобы использовать синтаксические конструкции Cilk Plus, необходимо применить данный компилятор.

В работе используется **Intel® Windows C/C++ Compiler**, входящий в состав **Intel® Parallel Composer XE 2011**. Чтобы применить указанный компилятор, в окне **Solution Explorer** выберите проект и выполните команду

контекстного меню **Intel® Parallel Composer→Use Intel® C++…**. В диалоговом окне **Confirmation** нажмите **OK**.

В настройках проекта убедитесь, что включена возможность использования Cilk Plus. Для этого кликните правой кнопкой мыши по проекту, в контекстном меню выберите пункт **Properties** и выполните команду **Configuration Properties→C/C++→Language.** Для свойств **Disable Intel® Cilk Plus Keywords for Serial Semantics** и **Disable All Intel® Language Extensions** установите значение **No.**

Далее в исходных кодах проекта необходимо подключить заголовочные файлы. Перечень подключаемых заголовочных файлов определяется компонентами, которые будут задействованы в разработке параллельной реализации. В настоящей работе используется только синтаксическая конструкция **`cilk_for`**, поэтому достаточно подключить файл, приведенный ниже.

```
#include <cilk/cilk.h>
```

3. Конструкции cilk_spawn и cilk_sync. Синтаксис и механизм исполнения

В данном разделе будут рассмотрены синтаксические конструкции **`cilk_spawn`** и **`cilk_sync`**.

Ключевое слово **`cilk_spawn`** – это конструкция, которая может быть использована непосредственно перед вызовом функции, чтобы указать системе, что данная функция может выполняться параллельно с вызывающей. При этом вызвавшая функция называется *родительской*, а вызванная – *дочерней*.

`cilk_spawn` можно использовать несколькими способами:

```
// func() возвращает значение типа type и принимает
// на вход список аргументов args
type var = cilk_spawn func(args);
var = cilk_spawn func(args);
// func() может возвращать void
cilk_spawn func(args);
```

В ситуации, когда дальнейшие вычисления в родительской функции невозможны без результатов дочерней, необходимо использовать конструкцию синхронизации **`cilk_sync`**.

```
cilk_sync;
```

Отметим, что **`cilk_spawn`** – это только рекомендация планировщику Cilk Plus выполнять нити вызванной и вызывающей функций параллельно. Решение о создании нового обработчика (worker) для выполнения вызывае-

мой функции принимается динамически при условии, что в системе имеются свободные ядра. В противном случае нить вызываемой функции выполняется тем же обработчиком, что и нить вызывающей. В качестве примера рассмотрим следующий код[1]:

```
do_init_stuff(); // выполнение нити 1
cilk_spawn func3(); // создание нити 3 (дочерняя)
do_more_stuff(); // выполнение нити 2 (продолжение 1)
cilk_sync;
do_final_stuff; // выполнение нити 4
```

Схематично диаграмму нитей можно представить так, как показано на Рис. 1. Т.к. планировщик принимает решение о создании потока динамически, то существует два возможных способа исполнения этих нитей:

1. Последовательное выполнение нитей в одном обработчике (worker) в порядке 1-3-2-4.
2. Параллельное выполнение нитей 2 и 3 в разных обработчиках (worker). В этом случае сначала выполняются инструкции нити 1 в исходном потоке, затем при вызове **`cilk_spawn`** создается новый поток. Нить 2 продолжает выполняться в исходном потоке, а нить 3 – в новом. По завершении нитей 2 и 3 начинает работать нить 4 в исходном потоке.

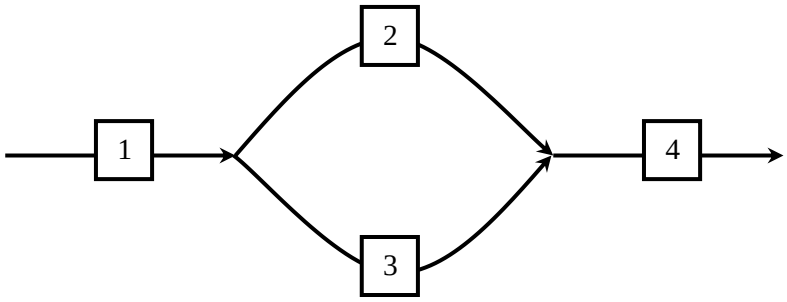

Рис. 1. Диаграмма создания нитей

4. Конструкция cilk_for. Синтаксис и механизм исполнения

Для распараллеливания циклов с известным числом повторений в Cilk Plus используется конструкция **`cilk_for`**. По существу **`cilk_for`** выглядит как обычный цикл **`for`**:

[1] Пример взят из документации к Cilk Plus [3].

```
cilk_for (int i = 0; i < 1000; ++i)
{
  //...
}
```

Основное требование к телу цикла состоит в том, чтобы итерации цикла были независимы по данным. Такое требование накладывается, чтобы каждая итерация могла быть выполнена в параллели с любой другой итерацией.

Необходимо отметить, что использование **`cilk_for`** предполагает наличие только одной переменной-счетчика и возможность перехода к любой другой итерации, т.е. цикл не должен содержать операторы принудительного перехода (**`return`**, **`break`**, **`goto`**). Более подробную спецификацию можно найти в [3].

Обратимся к вопросу о том, как работает конструкция **`cilk_for`**. В процессе компиляции тело цикла конвертируется в функцию, которая вызывается рекурсивно в соответствии со стратегией «разделяй и властвуй». Предположим, что **`first`** и **`last`** – это первое и последнее значения счетчика в цикле, тогда данную функцию можно представить следующим псевдокодом:

```
void run_loop(first, last)
{
  if (last - first < grainsize)
  {
    for (int i = first; i < last; ++i) LOOP_BODY;
  }
  else
  {
    int mid = (last - first) / 2;
    cilk_spawn run_loop(first, mid);
    run_loop(mid, last);
  }
}
```

Фактически множество итераций делится пополам до тех пор, пока после очередного деления количество итераций не будет превышать значения **`grainsize`**, **`grainsize`** – это максимальное количество итераций, на которое расщепляется множество итераций (может быть определено программно разработчиком).

Рассмотрим диаграмму нитей при исполнении такой рекурсии (Рис. 2) в случае, когда **`first = 0`**, **`last = 7`**, **`grainsize = 1`**. Каждой дуге диаграммы соответствует нить. Нить обрабатывает набор итераций цикла, номера которых подписаны над дугой.

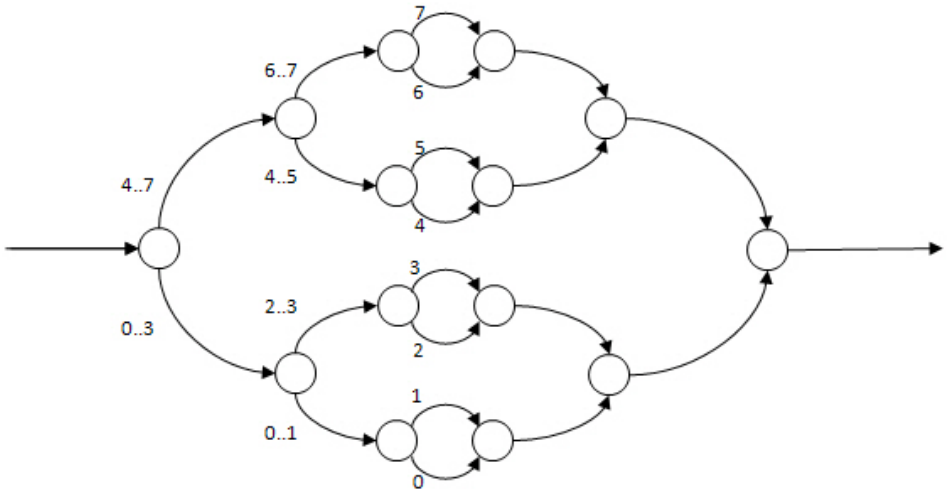

Рис. 2. Диаграмма создания нитей при распараллеливании цикла с известным числом итераций с помощью конструкции cilk_for [4]

На этапе исполнения нити отдаются обработчикам (worker). Планировщик Cilk Plus автоматически распределяет нагрузку между обработчиками.

Отметим, что механизм очень похож на тот, что реализуется при разделении итерационного пространства в библиотеке TBB (разница только в терминологии). При этом синтаксически организация параллелизма с использованием **`cilk_for`** значительно проще, чем с **`tbb::parallel_for`**.

Ниже показан пример использования **`cilk_for`** в задаче умножения плотной матрицы на вектор:

```
#include <cilk/cilk.h>

double dotProd(double *a, double *b, int n);

void Multiplication(double *A, double *x, int n, double *b)
{
  cilk_for (int i = 0; i < n; i++)
  {
    b[i] = dotProd(&(A[i * n]), x, n);
  }
}
```

5. Литература

5.1. Ресурсы сети Интернет

1. Страница интерфейса Cilk Plus на сайте корпорации Intel – [http://software.intel.com/en-us/articles/intel-cilk-plus/].
2. Руководство по использованию Intel® Cilk Plus – [http://software.intel.com/sites/products/evaluation-guides/docs/cilk-plus-evaluation-guide.pdf].
3. Спецификация синтаксических конструкций Intel® Cilk Plus – [http://software.intel.com/sites/products/cilk-plus/cilk_plus_language_specification.pdf].
4. Документация к Intel® Cilk Plus – [http://software.intel.com/sites/products/documentation/studio/composer/en-us/2011/compiler_c/index.htm#cref_cls/common/cilk_for.htm].