ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

TÍNH TOÁN SONG SONG

Bài Tập Lớn:

LẬP TRÌNH OPEN-MP VÀ

CÁC PHƯƠNG PHÁP LẬP LỊCH STATIC DYNAMIC

Giáo viên hướng dẫn: PGS.TS Thoại Nam

Lớp: L01

DANH SÁCH THÀNH VIÊN

1. Nguyễn Trung Tính – 1713521
2. Đặng Văn Dũng – 1710853
3. Cao Đăng Dũng – 1710849

TP. HỒ CHÍ MINH - THÁNG 11/2019

MỤC LỤC

GIỚI THIỆU ĐỀ TÀI 2
NỘI DUNG 3
1. LÝ THUYẾT 3
1.1 Lập trình song song với OpenMP 3
1.2 Lập lịch Static (Static schedule) 5
1.3 Lập lịch Dynamic (Dynamic schedule) 6
1.4 Công cụ trực quan hóa: AfterMath – Trace-based Performance Analysis for OpenMP and OpenStream 8
2. HIỆN THỰC 8
1.1 Lập lịch static: 9
1.2 Lập lịch dynamic 10
1.3 So sánh hai phương pháp lập lịch 11
DANH MỤC TÀI LIỆU THAM KHẢO 12

GIỚI THIỆU ĐỀ TÀI

Đề 1: Trực quan hoá các phương thức lập lịch static, dynamic cho các threads trong OpenMP.

Lý thuyết:

- Tìm hiểu về lập trình OpenMP
- Tìm hiểu về lập lịch static, dynamic
- Tìm hiểu công cụ trực quan hoá.

Hiện thực:

- Viết chương trình.
- Trực quan hoá và phân tích kết quả.

NỘI DUNG

1. LÝ THUYẾT

1.1 Lập trình song song với OpenMP

1.1.1 Định nghĩa

OpenMP là một giao diện lập trình ứng dụng (Application Programming Interface – API), đem lại mô hình lập trình linh động cho những nhà phát triển ứng dụng song song điều khiển các luồng (Thread) dựa trên cấu trúc chia sẻ bộ nhớ (shared memory).

OpenMP được hợp thành bởi: chỉ thị chương trình dịch (Compiler Directives), thư viện hàm thời gian chạy (Library Runtime Routines), các biến môi trường (Environment Variables).

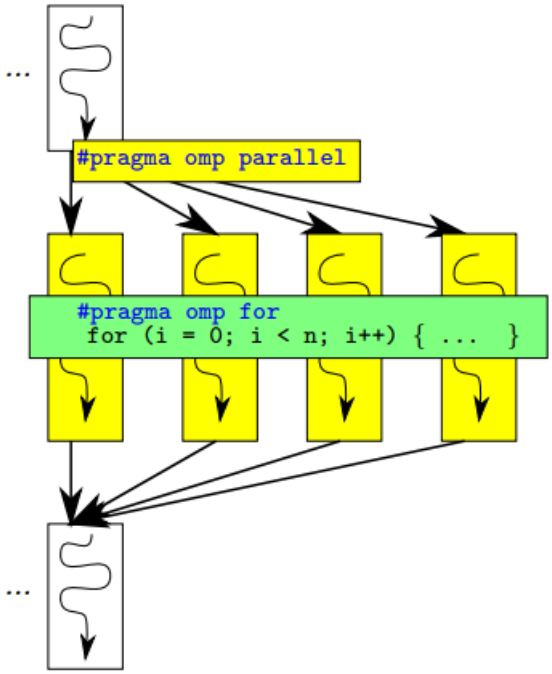
Tự động song song hóa chương trình, người lập trình phải tự ý thức về tính song song của công việc, OpenMP cung cấp cơ chế để chỉ định việc thực hiện song song.

Chương trình sẽ đạt hiệu suất và khả năng mở rộng tốt nếu được thiết kế đúng; tính khả chuyển cao; chương trình viết ra có thể dịch bởi nhiều chương trình dịch khác nhau; dễ sử dụng nhờ sự đơn giản và số lượng ít các chỉ thị; cho phép song song hóa tăng dần chương trình tuần tự.

1.1.2 Mô hình song song OpenMP

OpenMP cung cấp mô hình lập trình đa luồng cấp cao, xây dựng trên thư viện lập trình đa luồng của hệ thống. Vd: POSIX Threads,…

Thực hiện theo mô hình Fork – Join, chương trình OpenMP bắt đầu việc thực hiện như một luồng chủ duy nhất – master thread; luồng chủ thực hiện tuần tự cho đến vùng song song (vùng Fork) đầu tiên; luồng chủ tạo nhóm các luồng để chia sẻ thực hiện các công việc song song.



Hình 1 Mô hình Fork - Join

Khi các luồng thực thi các đoạn mã trong vùng song song kết thúc, chúng sẽ đồng bộ (Join) sau đó công việc lại được thực hiện bởi luồng chủ.

Mọi luồng có quyền truy cập đến vùng nhớ chung toàn cục; dữ liệu hoặc là chia sẻ hoặc là riêng tư; việc truyền dữ liệu là trong suốt với người lập trình; cần thiết phải đồng bộ hóa nhưng hầu như được thực hiện ngầm.

Tính năng chính của OpenMP: Tạo nhóm các luồng cho thực hiện song song; chỉ rõ cách các chia sẻ công việc giữa các luồng thành viên của nhóm; khai báo dữ liệu chia sẻ và riêng tư; đồng bộ các luồng và cho phép các luồng thực hiện thực hiện công việc một các độc lập; cung cấp hàm thời gian chạy; quản lý số lượng luồng.

1.1.3 Viết chương trình song song với OpenMP

Chia tách bài toán thành các công việc, lý tưởng nhất khi các cộng việc là hoàn toàn độc lập.

Gán công việc cho các luồng thực thi.

Viết mã trên môi trường lập trình song song.

Thiết kế chương trình phụ thuộc vào nền tảng phần cứng, cấp độ song song, bản chất của bài toán.

Song song theo dữ liệu khuyến kích lập trình song song có cấu trúc dựa trên phân chia công việc trong vòng lặp `(#pragma omp parallel for)`.

Song song theo công việc hỗ trợ việc gán các công việc cụ thể cho các luồng thông qua chỉ số của luồng `(#pragma omp parallel sections)`.

Chương trình C/C++ cần thêm khai báo: `#include <omp.h>`.

1.1.4 Khuôn dạng của chỉ thị

Chỉ thị trong OpenMP được cho dưới dạng sau

```
#pragma omp directive-name [clause...] newline
```

- `#pragma omp:` yêu cầu bắt buộc đối với mọi chỉ thị OpenMP C/C++.
- `directive-name:` là tên của chỉ thị phải xuất hiện sau `#pragma omp` và đứng trước bất kì mệnh đề nào: `parallel, for, sections, single,…`
- `[clause...]:` các mệnh đề này không bắt buộc trong chỉ thị: `private(list),num_threads(integer-expression), shared(list), schedule (kind [, chunk_size]), …`
- `newline:` yêu cầu bắt buộc với mỗi chỉ thị nó là tập mã lệnh nằm trong khối cấu trúc được bao bọc bởi chỉ thị.

  Ví dụ:

```
#prama omp parallel shared(a,b) private(i)
{
        //các đoạn mã
}
```

1.2 Lập lịch Static (Static schedule)

Được xác định bằng mệnh đề `schedule(static,chunk-size)`. OpenMP chia các lần lặp thành các khối có kích thước `chunk-size` (mặc định `chunk-size = 1`) và nó phân phối các khối cho các luồng theo thứ tự vòng tròn.

Trong lập lịch tĩnh, phân công nhiệm vụ cho các bộ xử lý được thực hiện trước khi thực hiện chương trình bắt đầu. Một nhiệm vụ luôn được thực thi trên bộ vi xử lý mà nó được giao, nghĩa là phương pháp lập lịch tĩnh là bộ xử lý nonpreemptive.

Thông thường, mục tiêu của phương pháp lập kế hoạch tĩnh là để giảm thiểu thời gian thực hiện tổng thể của một chương trình, dự đoán hành vi thực hiện tại thời điểm biên dịch, thực hiện một phân vùng nhiệm vụ nhỏ hơn, phân bổ quy trình để xử lý.

Ưu điểm chính của phương pháp lập lịch tĩnh là đòi hỏi ít thời gian và chi phí lập lịch nhất, kết quả tạo ra một môi trường thời gian thực hiện hiệu quả hơn so với phương pháp lập lịch động. Tuy nhiên, nó có thể gây ra cân bằng tải kém trên các ứng dụng mà chi phí tính toán trên các lần lặp là khác nhau.

Một số ví dụ về lập lịch tĩnh: giả sử song song hóa vòng lập for với 64 lần lặp, và sử dụng 4 threads, mỗi hàng ngôi sao đại diện cho một thread, mỗi cột đại diện cho một lần lặp.

```
schedule(static):
****************
                ****************
                                ****************
                                                ****************
schedule(static, 4):
****            ****            ****            ****
    ****            ****            ****            ****
        ****            ****            ****            ****
            ****            ****            ****            ****
schedule(static, 8):
********                        ********
        ********                        ********
                ********                        ********
                        ********                        ********
```

Ở ví dụ đầu tiên, mỗi dòng có 16 ngôi sao, nghĩa là mỗi thread sẽ thực hiện 16 lần lặp, hay nói cách khác thread 1: 0, 1, 2,…15 ; thread 2: 16, 17, 18,…31; tương tự đối với thread 3, 4.

Ta thấy rằng, với `schedule(static)` OpenMP sẽ chia số lần lặp thành 4 khối với kích thước 16 và chia cho 4 threads. Với `schedule(static, 4)` và `schedule(static, 8)` OpenMP chia số vòng lặp thành các khối 4 và 8.

1.3 Lập lịch Dynamic (Dynamic schedule)

Được xác định bằng mệnh đề `schedule(dynamic,chunk-size).` Các công việc lặp đi lặp lại của vòng lặp được chia làm các `chunk-size` công việc, nhưng khác với STATIC các công việc ở đây được tự động lên lịch với kích thước `chunk-`

`size` đã chỉ định trong thời gian chạy của ứng dụng và phân phối chúng bất cứ khi nào có một PU nhàn rỗi.

Ưu điểm của việc lập lịch Dynamic là hệ thống không cần phải biết trước được thời gian chạy hành vi của các ứng dụng trước khi thực hiện. Sự linh hoạt vốn có trong cân bằng tải động cho phép thích ứng với yêu cầu ứng dụng không lường trước được tại thời gian chạy. Lập lịch Dynamic đặc biệt hữu ích trong một hệ thống bao gồm một mạng lưới các máy trong đó mục tiêu hiệu suất chính là tối đa hóa việc sử dụng của các quá trình thay vì giảm thiểu thời gian thực hiện của các ứng dụng. Những bất lợi chính của lập lịch Dynamic là thời gian chạy cao do:

- Việc truyền tải thông tin giữa các bộ vi xử lý.
- Quá trình ra quyết định cho việc lựa chọn các quy trình và bộ xử lý để chuyển giao công việc.
- Sự chậm trễ giao tiếp do nhiệm vụ di dời của chính nó.

Một số ví dụ về lập lịch động: với bài toán như trên, lập lịch động chia công việc như sau:

```
schedule(dynamic):
*   ** **  * * *  *      *  *    **   *  *  * *       *  *   *
  *       *     *    * *     * *   *    *        * *   *    *
 *       *    *     * *   *   *     *  *       *  *  *  *  *   *
   *  *     *    * *    *  *    *    *    ** *  *   *     *   *
schedule(dynamic, 1):
    *    *     *        *   *    * *  *  *         *  * *  * *
*  *  *   * *     *  * * *    * *      *   ***  *   *         *
 *   *  *  *  *    ** *    *      *  *  * *   *  *   *   *
  *    *     *  **        *  * *    *          *  *    *  * *  *
schedule(dynamic, 4):
           ****                    ****                    ****
****            ****    ****            ****        ****
    ****            ****    ****            ****        ****
        ****                    ****            ****
schedule(dynamic, 8):
                ********                                ********
                        ********        ********
********                        ********        ********
        ********
```

Với `schedule(dynamic)` và `schedule(dynamic,1)` OpenMP thực hiện lập lịch như nhau, kích thước mỗi khối là 1. Phân chia công việc cho các threads là tùy

ý. Tương tự đối với `schedule(dynamic,4)` và `schedule(dynamic,8)` OpenMP chia thành các khối có kích thước là 4 và 8.

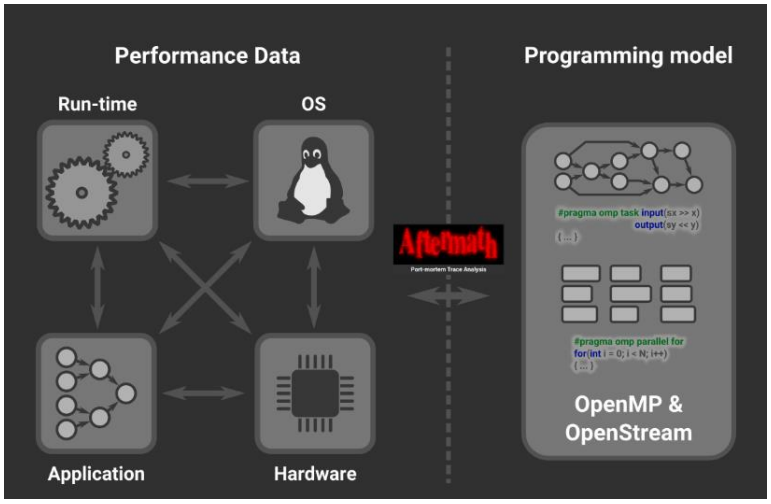
1.4 Công cụ trực quan hóa: AfterMath – Trace-based Performance Analysis for OpenMP and OpenStream

Aftermath là một công cụ đồ họa để phân tích hiệu suất dựa trên vết của các chương trình OpenMP và OpenStream. Công cụ cho phép các lập trình viên tính toán được hiệu suất dữ liệu từ các mô hình thực thi song song, ví dụ: các vòng lặp và tác vụ trong OpenMP.

Aftermath hỗ trợ kiểm tra sự tương tác giữa các vết, tạo ra các số liệu thống kê chi tiết cho các tập hợp con tùy ý của một vết đang hoạt động và kiểm tra các sự kiện riêng lẻ một cách chi tiết.

Một số tính năng của Aftermath:

- Giao diện người dùng trực quan, ngay cả đối với các tệp truy vết lớn.
- Nhiều views được kết nối để kiểm tra, thống kê và phân tích chi tiết.
- Trực quan và kiểm tra chi tiết các vòng lặp song song.
- Thời gian chạy được ghi lại dựa trên thời gian chạy LLVM/clang OpenMP.



Hình 2 Cơ chế hoạt động Aftermath

2. HIỆN THỰC

Cấu trúc vòng for trong OpenMP và mệnh đề `schedule`:

```
#pragma omp parallel for schedule(scheduling-type)
for (...)
```

```
{ ... }
```

OpenMP sẽ dùng `scheduling-type` để lập lịch cho các lần lặp trong vòng lặp for.

1.1 Lập lịch static:

Minh họa việc phân tích các tasks, hãy xem xét mã dưới đây, một phiên bản của chương trình xác định số lượng số nguyên tố trong một khoảng cho trước.

```
#include <stdio.h>
#include <math.h>
#include <omp.h>
#define N 1000000
int isprime_naive(int n)
{
  if(n % 2 == 0 && n != 2)
    return 0;
  for(int j = 3; j <= sqrt(n); j += 2)
    if(n % j == 0)
      return 0;
  return 1;
}
int main(int argc, char** argv)
{
  int n = 1;
  #pragma omp parallel
  {
    #pragma omp for schedule(static) reduction(+:n)
    for(int i = 0; i < N; i+=2)
    {
       n += isprime_naive(i);
    }
  }
  printf("There are %d prime numbers in the interval\n", n);
  return 0;
}
```

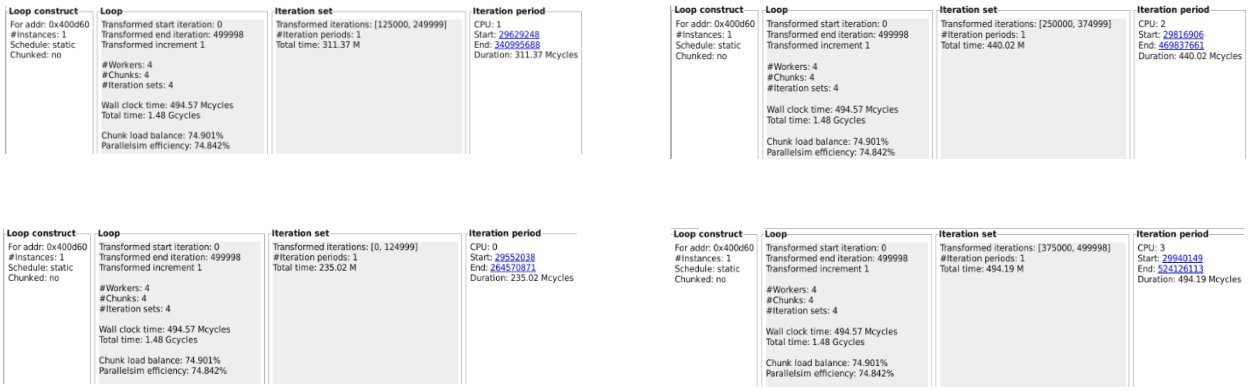
Kết quả thu được sau khi thực thi chương trình trên

Hình 3 Thời gian chạy của các CPU trong lập lịch tĩnh

Có 4 hàng ứng với 4 CPU khác nhau thực thi. Trên mỗi CPU có mỗi màu ứng với một tập các lần lặp của một CPU, ở đây dùng lập lịch static nên ta chỉ thấy mỗi CPU chỉ thực thi một tập duy nhất trong toàn vòng lặp.

Ta còn thấy được, độ dài của mỗi hàng ứng với thời gian chạy của mỗi CPU. Rõ ràng có sự mất cân bằng tải giữa các CPU.

Ngoài ra, Aftermath còn ghi nhận được `scheduling-type`, `chunk-size`, bắt đầu ở lần lặp nào, kết thúc ở lần lặp nào và thời gian chính xác của chúng.



Hình 4 Thông tin chi tiết của các CPU từ 0 đến 3

Hiệu suất song song tính toán đạt được 74.842%. Được tính theo công thức sau:

$$E_{par} = \frac{t_{tot}^{loop}}{t_{wct}^{loop}.N}$$ với: t_{tot}^{loop} : total time

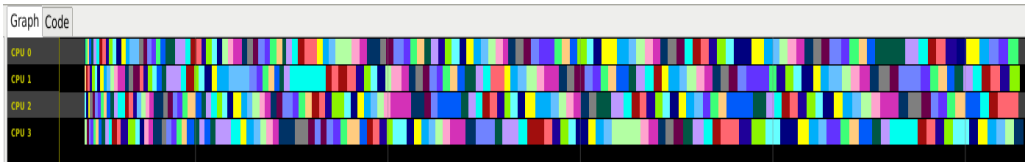
t_{wct}^{loop}: wall clock time

N: number of cores

1.2 Lập lịch dynamic

Cùng một bài toán trên, ta thay `#pragma omp for schedule(static)` thành `#pragma omp for schedule(dynamic,1000)` để thay đổi thành lập lịch động.

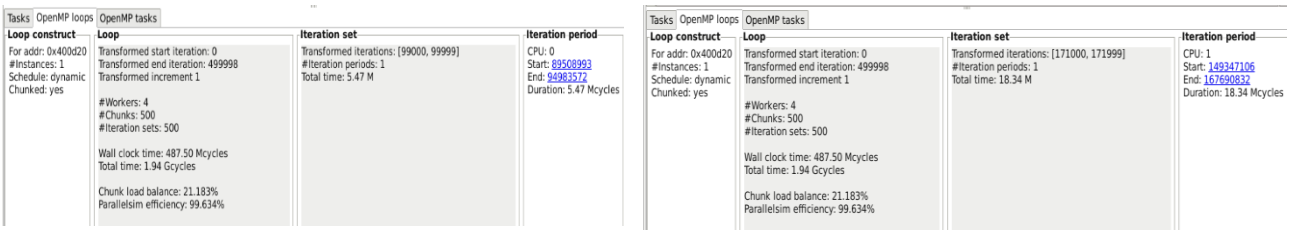
Kết quả thu được



Hình 5 Thời gian chạy của các CPU trong lập lịch động

Đúng như lý thuyết, ta có thể thấy các CPU thay nhau xử lí các vòng lặp (ứng với mỗi màu khác nhau), và các CPU dường như xử lí xong cùng lúc, không xảy ra mất cân bằng tải.

Khác với static, mỗi CPU thay nhau thực hiện các lần lặp, dưới đây là thông tin của một số tập lần lặp (`chunk-size = 1000`)



Hình 6 Thông tin chi tiết của một số tập lần lặp

Hiệu suất song song tính toán đạt đến 99.643%

1.3 So sánh hai phương pháp lập lịch

a) Tăng số thread

Thực hiện chạy chương trình trên, với số lượng thread tăng liên tục, và bằng hai phương pháp lập lịch, thu được kết quả thời gian chạy và hiệu suất như bảng sau:

Số thread	Static		Dynamic	
	Thời gian (Gcycles)	Hiệu suất (%)	Thời gian (Gcycles)	Hiệu suất (%)
1	1.43	50.00	1.43	50
2	1.51	54.36	1.70	99.87
3	1.60	77.95	1.83	99.92
4	1.81	77.61	1.97	97.86

b) Tăng số chunk_size

Thực hiện chạy chương trình trên, với chunk_size tăng liên tục, số thread bằng 4, và bằng hai phương pháp lập lịch, thu được kết quả thời gian chạy và hiệu suất như bảng sau:

Chunk_size	Static		Dynamic	
	Thời gian (Gcycles)	Hiệu suất (%)	Thời gian (Gcycles)	Hiệu suất (%)
1000	1.89	94.49	1.97	97.86
5000	2.38	89.39	1.85	96.39
10000	1.95	93.50	2.45	81.20
20000	2.12	95.02	1.96	96.10

DANH MỤC TÀI LIỆU THAM KHẢO

1. **Corner, Jaka's.** *OpenMP: For & Scheduling.* [Online] June 13, 2016. http://jakascorner.com/blog/2016/06/omp-for-scheduling.html.

2. **Aftermath.** Aftermath: Trace-based Performance Analysis for OpenMP and OpenStream. [Online] https://www.aftermath-tracing.com/legacy/.