MÔ TẢ ĐỀ TÀI

*Nhóm* :

*Thành viên*:

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

*Đề tài*: Trực quan hóa các phương thức lập lịch static, dymamic cho các threads trong OpenMP

1. Lý thuyết
   1. Lâp trình OpenMp
      1. OpenMp: Open specifications for Multi Processing

* OpenMP là Application Programming Interface (API), đem lại mô hình lập trình linh động cho những nhà phát triển ứng dụng song song chia sẻ bộ nhớ.
* 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).
* Có thể dùng được trên hầu hết các máy với kiến trúc một không gian nhớ (single memory space).
* OpenMP không phải một ngôn ngữ lập trình mới, thực ra OpenMP hoạt động trên sự liên kết chặt chẽ với ngôn ngữ lập trình làm cơ sở. Vd: Fortran, C/C++.
* 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.
* Phương tiện lập trình cho hệ thống có bộ nhớ phân tán.

Ưu điểm của OpenMP

* Một chuẩn hoàn chỉnh và được công nhận trên thực tế; hiệu suất và khả năng mở rộng tốt nếu chương trình 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. 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 đầu tiên; luồng chủ tạo nhóm các luồng để chia sẻ thực hiện các công việc song song.
* Song song đa luồng, song song có khai báo, song song động.

Mô hình bộ nhớ OpenMP

* 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 quyền; cung cấp hàm thời gian chạy; quản lý số lượng luồng.
  + 1. 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).
  + 1. Các hàm của OpenMP
* OpenMP cung cấp các hàm giúp người dùng điều khiển và truy vấn môi trường song song, thủ tục semaphore/lock đa mục đích...
* Hàm có độ ưu tiên cao hơn các biến môi trường tương ứng.
* Chương trình C/C++ cần thêm khai báo.

#include

* Nên sử dụng kèm macro

#ifdef \_OPENMP

* Cấu trúc song song:

#pragma omp parallel [clause [ [, ]clause] ...] new-line structured-block clause:

* if(scalar-expression)
* num\_threads(integer-expression)
* default (shared | none)
* private(list)
* firstprivate(list)
* shared(list)
* copyin(list)
* reduction (operator: list)

#pragma omp for [clause [[,] clause]...] new-line for-loops clause:

* private(list)
* firstprivate(list)
* lastprivate(list)
* reduction (operator: list)
* schedule (kind [, chunk\_size])
* collapse(n)
* ordered
* nowait

#pragma omp sections [clause [[,] clause] ...] new-line

{

[#pragma omp section new-line]

structured-block

[#pragma omp section new-line structured- block ]

...

}

Clause:

* private(list)
* firstprivate(list)
* lastprivate(list)
* reduction (operator: list)
* nowait

#pragma omp single [clause [[,] clause] ...] new-line structured-block clause:

* private(list)
* firstprivate(list)
* lastprivate(list)
* reduction (operator: list)
* nowait

#pragma omp task [clause [[, ] clause] ...] new-line structured-block clause:

* if(scalar-expression)
* untied default (shared | none)
* private(list)
* firstprivate(list)
* shared(list)
  1. Lập lịch static, dynamic
     1. Lập lịch static

Trong lập lịch tĩnh, phân công nhiệm vụ cho các bộ xử lý được thực hiện trước khi thực hiện chương trình bắt đầu. Trong việc hình thành liên quan đến thời gian thực hiện nhiệm vụ và các nguồn lực xử lý được giả được biết đến tại thời gian biên dịch. 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 đồng thời trong khi giảm thiểu sự chậm trễ thông tin liên lạc. Với muc tiêu đó phương pháp lập lịch của [Lo 1988, Sarkar 1986, Shirazi 1990, Stone 1977] cố gắng:

* Dự đoán hành vi thực hiện tại thời điểm biên dịch (ước tính quá trình hoặc công việc, thời gian thực hiện, và sự chậm trễ giao tiếp).
* Thực hiện một phân vùng nhiệm vụ nhỏ hơn vào các quá trình giảm chi phí giao tiếp.
* Phân bổ quy trình để xử lý.

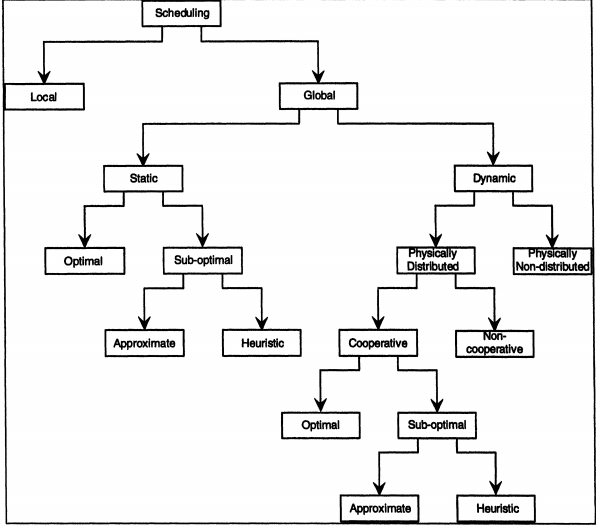
Ưu điểm chính của phương pháp lập kế hoạch tĩnh là tất cả các chi phí của quá trình lập kế hoạch phát sinh tại thời gian biên dịch, kết quả trong một môi trường thời gian thực hiện hiệu quả hơn so với phương pháp lập lịch động. Tuy nhiên lập lịch tĩnh có nhiều nhược điểm.

Phương pháp lập kế hoạch tĩnh có thể được phân thành tối ưu và ít tối ưu.

Một trong những thiếu sót quan trọng nhất của lập lịch tĩnh là việc tạo lịch trình tối ưu là một vấn đề hoàn chỉnh NP. Chỉ có thể tạo các giải pháp tối ưu trong các trường hợp hạn chế (ví dụ: khi thời gian thực hiện

của tất cả các tác vụ là như nhau và chỉ có hai bộ xử lý được sử dụng). Tính đầy đủ của việc lập lịch tĩnh tối ưu, có hoặc không có các cân nhắc về chi phí truyền thông, đã được chứng minh trong tài liệu [Chretienne 1989,

Papadimitriou 1990, Sarkar 1989].



Hình 1.1: Phân loại các phương pháp lập lịch

* + 1. Lập lịch dynamic

Lập lịch động dựa trên sự phân phối lại các quy trình giữa các bộ xử lý trong thời gian thực hiện. Sự phân phối lại này được thực hiện bằng cách chuyển các tác vụ từ các bộ xử lý tải nặng nề với bộ xử lý tải nhẹ (gọi là cân bằng tải) với mục đích cải thiện hiệu năng của ứng dụng. [Eager 1986, Lin 1987, Shivaratri 1992, Wang 1985], một thuật toán cân bằng tải điển hình được xác định bởi ba

chính sách:

* Chính sách thông tin, trong đó nêu rõ số lượng thông tin tải được cung cấp cho việc sắp xếp người ra quyết định
* Chính sách chuyển giao, xác định các điều kiện theo đó một công việc sẽ được chuyển giao, nghĩa là tải trọng hiện tại của máy chủ và quy mô của công việc đang xem xét (chính sách chuyển giao có thể hoặc không bao gồm di chuyển tác vụ, nghĩa là tạm dừng thực thi nhiệm vụ và chuyển nó sang bộ xử lý khác để tiếp tục thực hiện)
* Chính sách vị trí, trong đó xác định các yếu tố xử lý mà một công việc nên được chuyển giao

Các hoạt động cân bằng tải có thể được tập trung trong một bộ xử lý hoặc được phân phối giữa tất cả các yếu tố xử lý tham gia vào quá trình cân bằng tải. Nhiều chính sách kết hợp cũng có thể tồn tại. Ví dụ, chính sách thông tin có thể được tập trung nhưng chính sách chuyển nhượng và vị trí có thể được phân phối. Trong trường hợp đó, tất cả các bộ xử lý gửi thông tin tải của chúng đến bộ xử lý trung tâm và nhận thông tin tải hệ thống từ bộ xử lý đó. Tuy nhiên, các quyết định liên quan đến khi nào và nơi một công việc nên được chuyển giao được thực hiện tại địa phương của mỗi bộ xử lý.Nếu một chính sách thông tin phân phối được sử dụng, mỗi phần tử xử lý giữ hình ảnh địa phương riêng của mình tải của hệ thốngTuy nhiên, các quyết định liên quan đến khi nào và nơi một công việc nên được chuyển giao được thực hiện tại địa phương của mỗi bộ xử lý.Nếu một chính sách thông tin phân phối được sử dụng, mỗi phần tử xử lý giữ hình ảnh địa phương riêng của mình tải của hệ thống. Chính sách hợp tác này thường được thực hiện bằng một bản phân phối của thông tin tải một trong những yếu tố xử lý [Lin 1987]. Mỗi bộ xử lý chuyển tải thông tin hiện tại của mình cho các nước láng giềng tại các khoảng thời gian định trước, dẫn đến sự phân tán thông tin tải giữa tất cả các yếu tố xử lý trong một khoảng thời gian ngắn. Chính sách phân phối thông tin cũng có thể không hợp tác. Lập lịch ngẫu nhiên là một ví dụ về lập lịch bất hợp tác, trong đó tải nặng bộ xử lý chọn ngẫu nhiên một bộ xử lý khác để chuyển công việc. Cân bằng tải ngẫu nhiên hoạt động tốt hơn tốt khi tải của tất cả các bộ xử lý tương đối cao, nghĩa là khi nó không tạo ra nhiều khác biệt

nơi một công việc được thực thi.

Ưu điểm của việc nạp năng động cân bằng trên lịch tĩnh là hệ thống không cần phải nhận thức 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.cân bằng tải động là đặc biệt hữu ích trong một hệ thống bao gồm một mạng lưới các máy trạm trong đó mục tiêu hiệu suất chính được tối đa hóa việc sử dụng của quá trình xử lý điện thay vì giảm thiểu thời gian thực hiện của các ứng dụng.Những bất lợi chính của chương trình cân bằng tải động là thời gian chạy trên 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 chuyển cho công việc.
* Sự chậm trễ giao tiếp do nhiệm vụ di dời của chính nó.
  1. Công cụ trực quan hóa

*Sử dụng DOT tool*: Chúng ta cần điều chỉnh source code để xử lí việc phân phối vòng lặp vào thread. Trong thời gian runtime thì ta lưu trữ thông tin phân phối đó vào bộ nhớ buffer. Khi thời gian thực thi kết thúc thì, thì thông tin được lưu trữ trong bộ nhớ buffer đó sẽ được chuyển đổi thành một file text trong định dạng DOT, sau đó nó sẽ được hiện thực bằng DOT tool.

1. Hiện thực