ĐẠI HỌC BÁCH KHOA THÀNH PHỐ HỒ CHÍ MINHKHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Báo cáo bài tập lớn

PARALLEL PROGRAMMING

Đề tài 20
Hiện thực giải thuật PSO cho bài toán lập lịch máy ảo với thời gian cố định.

Giáo viên bộ môn: ***Thầy Nguyễn Quang Hùng***

Danh sách thành viên

1. Vũ Thế Đệ 51300854
2. Bùi Minh Đức 51300902

Mục lục

1. Giới thiệu.

Với sự phát triển của công nghệ tính toán mạng và tính toán song song, việc giải quyết vấn đề về lập lịch trở nên rất cần thiết. Nếu có 1 số lượng lớn các task cần phải được tính toán trên 1 số lượng máy nhất định cho trước, 1 giải thuật lập lịch hợp lý cần phải đạt được sao cho thời gian xử lý là nhỏ nhất. Lập lịch công việc là một trong những bài toán của vấn đề NP đầy đủ, và trở thành 1 vấn đề mà các nhà nghiên cứu quan tâm.

Những thuật toán tối ưu hóa sử dụng heuristic thường được sử dụng rộng rãi để giải quyết vấn đề NP đầy đủ. Một số những thuật toán lấy cảm hứng từ tự nhiên như giải thuật di truyền (Genetic Algorithm), giải thuật giả lập luyện kim (Simulated Annealing), Tabu Search. Giải thuật tối ưu bầy đàn (Particle Swarm Optimization – PSO) cũng là một trong những giải thuật lấy cảm hứng từ tự nhiên và báo cáo này sẽ tập trung vào giải quyết vấn đề lập lịch công việc bằng giải thuật PSO này.

2. Mô tả bài toán.

2.1 Đề bài.

Hiện thực giải thuật PSO cho bài toán lập lịch máy ảo với thời gian cố định bằng lập trình song song trên MPI.

2.2 Mô tả bài toán.

Một hệ thống có M máy tính (*machine*) khác nhau. Mỗi máy có 1 tài nguyên về CPU và RAM cố định. Có T nhiệm vụ (*task*) **độc lập** với nhau được đưa ra. Mỗi nhiệm vụ yêu cầu 1 tài nguyên nhất định về CPU và RAM để có thể thực thi và hoàn tất.

Yêu cầu bài toán là sắp xếp tất cả T nhiệm vụ vào M máy tính sao cho thời gian thực thi tất cả các nhiệm vụ là nhỏ nhất.

3. Sơ lược về giải thuật PSO.

3.1 Giới thiệu.

Giải thuật PSO (Particle Swarm Optimization), nghĩa tiếng Việt - giải thuật tối ưu bầy đàn, là một kỹ thuật tối ưu ngẫu nhiên được phát triển bởi Dr. Eberhart và Dr. Kennedy vào năm 1995. Giải thuật này được lấy cảm hứng từ những hành vi tự nhiên mang tính bầy đàn của chim hoặc cá.

Giải thuật PSO có rất nhiều nét tương đồng so với giải thuật di truyền (Genetic Algorithms - GA). Hệ thống sẽ khởi tạo 1 loạt các giải pháp ngẫu nhiên và tìm kiếm kết quả tối ưu dựa trên việc cập nhật giá trị của tập ngẫu nhiên đó thông qua nhiều thế hệ. Tuy nhiên, không như GA, PSO không có sự biến đổi như di truyền chéo và đột biến. Trong giải thuật PSO, những giải pháp tiềm năng, gọi là particles, sẽ di chuyển trong không gian trạng thái dựa vào thông tin của những trạng tối ưu hiện tại.

So với giải thuật GA, lợi thế của giải thuật PSO là dễ dàng hiện thực và có nhiều tham số để điều chỉnh. PSO đã được áp dụng thành công trong một số lĩnh vực như tối ưu chức năng,

huấn luyện tập neural network, điều khiển hệ thống mờ và một số lĩnh vực khác mà GA có thể áp dụng được.

3.2 Giải thuật tổng quát.

Giả sử có 1 kịch bản như sau: Một đàn chim đang tìm kiếm thức ăn một cách ngẫu nhiên trong một vùng địa lý. Chỉ có 1 mảnh của thức ăn trong vùng địa lý này có thể được tìm thấy. Tất cả những con chim không biết mảnh thức ăn đó ở đâu. Nhưng chúng biết còn bao xa để tới được mảnh thức ăn sau mỗi vòng lặp. Vậy chiến thuật tốt nhất để tìm ra mảnh thức ăn là gì? Cách hiệu quả nhất là đi theo con chim có vị trí gần nhất so với mảnh thức ăn.

Giải thuật PSO sẽ dựa trên kịch bản trên và sử dụng nó để tìm ra kết quả tối ưu. Trong PSO, mỗi giải pháp đơn là một con chim ở trong 1 không gian trạng thái. Nó được gọi là *particle*. Tất cả những particle đều có một giá trị tốt nhất được lượng giá bởi hàm số nào đó, và một vector vận tốc để định hướng cho particle. Tất cả những particle sẽ di chuyển trong không gian tối ưu hiện tại dựa theo hướng của 1 particle có giá trị tối ưu.

PSO khởi tạo 1 nhóm các giải pháp (particle) ngẫu nhiên và tìm kiếm giá trị tối ưu qua nhiều lần lặp. Ở mỗi vòng lặp, mỗi particle sẽ cập nhật 2 giá trị tốt nhất. Giá trị thứ nhất là giá trị tốt nhất đạt được ở trong particle đó, hay còn gọi là pBest. Giá trị thứ hai là giá trị tốt nhất đạt được trong toàn cục của swarm, hay còn gọi là gBest.

Trong khi tìm kiếm 2 giá trị pBest và gBest, các particle sẽ cập nhật vector vận tốc và vị trí của nó theo công thức sau:

$$v_{i,d} = \omega v_{i,d} + c_p r_p\left(p_{i,d} - x_{i,d}\right) + c_g r_g\left(g_d - x_{i,d}\right) \quad (\mathbf{1})$$

$$x_i = x_i + v_i \quad (\mathbf{2})$$

Ở công thức (**1**), ω, c_p, c_g là những giá trị được chọn bởi người lập trình để tối ưu cho giải thuật PSO. Thông thường, $c_p = c_g = 2$. r_p, r_g là những số ngẫu nhiên trong khoảng (0, 1).

3.3 Mã giả.

Mã giả của giải thuật PSO như sau:

```
for each particle i = 1, ..., S do
  Khởi tạo vị trí particle với phân phối chuẩn: xi ~ U(b_low, b_up)
  Khởi tạo pBest: pi ← xi
  if f(pi) < f(g) then
    cập nhật vị trí tốt nhất gBest: g ← pi
  Khởi tạo vector vận tốc: vi ~ U(-|bup-blo|, |bup-blo|)
while (điều kiện dừng chưa thỏa mãn) do
  for each particle i = 1, ..., S do
    for each dimension d = 1, ..., n do
      Chọn ngẫu nhiên : rp, rg ~U(0, 1)
      Cập nhật vận tốc: vi,d ← ω*vi,d + cp*rp*(pi,d-xi,d) + cg*rg*(gd-xi,d)
      Cập nhật vị trí: xi ← xi + vi
      if f(xi) < f(pi) then
        Cập nhật vị trí pBest: pi ← xi
        if f(pi) < f(g) then
          Cập nhật vị trí gBest: g ← pi
```

4. Áp dụng giải thuật PSO để giải quyết bài toán.

Các thông số:

- Hàm mục tiêu (Objective function): Thời gian để hoàn thành hết tất cả công việc hiện có là nhỏ nhất.
- Position sẽ là **trạng thái** của việc lập lịch ở thời điểm hiện tại. Ví dụ: task1, task2 được xếp vào machine1, task3 được xếp vào machine 2…
- Velocity V_{tm} là việc chọn 1 task **t** vào máy **m** sao cho hàm mục tiêu là nhỏ nhất.

Mã giả, ý tưởng để giải quyết bài toán:

```
InitParticles(n)
   Khởi tạo n particle ngẫu nhiên (ngẫu nhiên ở thứ tự các machine và các
   task). Ta sẽ có 2 vector Machines và Tasks chứa các machine và các task đã
   được random thứ tự.
FindGlobalBest():
   for (i = 0; i < n; i++)
       LocalBest_i  =  FindLocalBest();
       GlobalBest = Min(LocalBest_i, 0 < i < n );
FindLocalBest():
   for (j = 1; j < số task; j++) {
       BestMachine = ChooseMachineForTask(j);
   }
ChooseMachineForTask(j):
   for (m = 0 ; m < số machine; m++) {
        Value_m = Hàm mục tiêu(m, j);
      }
      bestMachine = Machines[Min(Valuem, 0 <= m < số machine)]
      return bestMachine;
```

4.1 Giải quyết bài toán bằng giải thuật PSO dùng phương pháp tuần tự.

- Ý tưởng: Lặp khắp các particle để tìm kết quả tối ưu.
- Mã giả:

```
InitParticles(n);
FindGlobalBest();
```

4.2 Giải quyết bài toán bằng giải thuật PSO dùng phương pháp song song.

- Ý tưởng: Chia đều số particle cho mỗi process, mỗi process sẽ làm công việc tương tự giải thuật tuần tự. Sau đó lấy kết quả tốt nhất từ các kết quả tốt nhất trả về từ các process.
- Mã giả:

  + Giả sử có m số processes, thì mỗi process sẽ làm làm như sau:

```
InitParticles(n / m);
GlobalBest_k  = FindGlobalBest();
```

  + Lấy kết quả tốt nhất từ kết quả tốt nhất của các process:

```
Min (GlobalBest_k , 0 <= k < m);
```

4.3 Phân tích độ hiệu quả và speedup của giải thuật song song trên lí thuyết và thực tiễn.

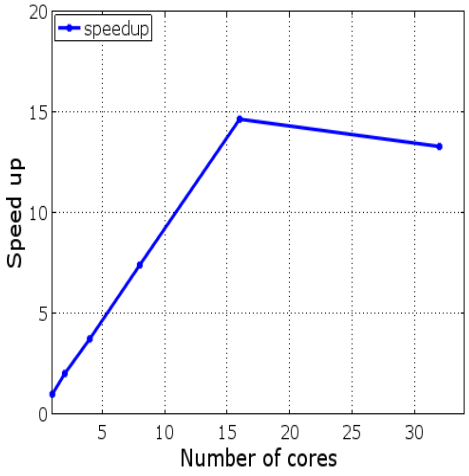
– Theo lí thuyết:

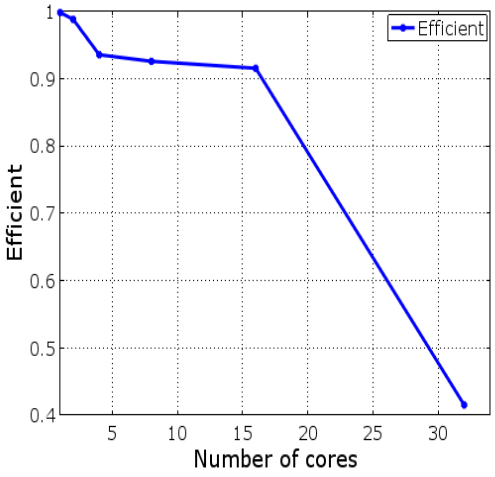
Theo Amdahl's law thì $\boldsymbol{speedup} = \frac{1}{\alpha + \frac{1-\alpha}{N}}$. Giá trị α và N có thể có rất nhiều giá trị khác nhau. Ở bài toán xếp lịch với giải thuật song song chúng tôi dùng ở trên thì giá trị α sẽ rất là nhỏ vì phần việc song song chiếm chủ yếu, chỉ có một khoảng thời gian rất nhỏ dành cho phần việc tuần tự (tìm min của các localbest được trả về từ các process). Hơn nữa việc song song hóa của chúng tôi theo dạng data parallel, chi phí giao tiếp giữa các process sẽ là rát nhỏ nên có thể bỏ qua. Ví vậy speedup theo lí thuyết cho bài toán của chúng tôi sẽ sấp xỉ bằng $\frac{1}{0 + \frac{1-0}{N}} = N$ và độ hiệu quả theo lí thuyết cho bài toán của chúng tôi sẽ sấp xỉ bằng 1.

– Theo thực tiễn:

Để kiểm chứng lại kết quả speedup và độ hiệu quả trên lí thuyết, chúng tôi đã thử chạy giải thuật tuần tự và song song để xếp lịch cho **2000 tasks** trên **20 machines** trên server có 16 cores vật lí và tài nguyên Ram là 32 G. Kết quả đo được tương đối phù hợp với kết quả lí thuyết

- Thời gian chạy giải thuật tuần tự để tìm ra kết quả **trong vùng** tối ưu: 502.111(s).
- Thời gian chạy song song với:
  - ✓ 1 cores: 503.212(s)
  - ✓ 2 cores: 254.245(s)
  - ✓ 4 cores : 134.232(s)
  - ✓ 8 cores: 67.814(s)
  - ✓ 16 cores: 34.574(s)





- **Giải thích kết quả đo được:**
  +Vì giải thuật song song của chúng tôi dựa trên **data parallel** nên thời gian giao tiếp giữa các process và thời gian overhead sẽ nhỏ hơn nên speedup đo được sẽ gần như tăng tuyến tính so với số core (như hình 1 ở trên). Do máy server chỉ có 16 cores vật lí nên khi tăng số process vượt quá 16 thì các process sẽ tranh chấp tai nguyên CPU, dẫn đến speedup giảm.
  + Khi tăng số process từ 1->16 thì ta thấy speedup gần như tăng tuyên tính, nên độ hiệu quả giảm rất chậm. Khi vượt quá 16 cores thì speedup giảm mạnh nên độ hiệu quả cũng sẽ giảm mạnh, như hình trên.

4.4 Phân chia công việc và Sorce Code.

- Phân chia công việc trong nhóm: Cả 2 thành viên đều bàn bạc và đưa ra ý tưởng chung để hiện thực bài toán. Bùi Minh Đức hiện thực giải thuật tuần tự và đo speedup và độ hiệu quả của giải thuật song song so với giải thuật tuần tự. Vũ Thế Đệ hiện thực giải thuật song song và dùng MatLab để vẽ hình từ số liệu đo được.

-Source code của nhóm được up lên github theo link sau :

https://github.com/ParallelProcessing/Assignment1--PSO--Scheduling.git

5. Tài liệu tham khảo.

[1] PSO Tutorial http://www.swarmintelligence.org/tutorials.php
[2] Particle swarm optimization https://en.wikipedia.org/wiki/Particle_swarm_optimization