PHƯƠNG PHÁP GIẢI BÀI TOÁN TỐI ƯU HÀM NHIỀU BIẾN BẰNG THUẬT TOÁN SONG SONG VÀ DI TRUYỀN

Nguyễn Mậu Hân, Nguyễn Đình Quý, Nguyễn Hoàng Hà
Trường Đại học Khoa học, Đại học Huế

TÓM TẮT

Các phương pháp giải bài toán tối ưu của hàm nhiều biến được biết đến khá sớm trong toán học. Nhưng đối với một số bài toán phức tạp các phương pháp này khó có thể tìm kiếm được lời giải tối ưu. Trong phạm vi bài báo này, chúng tôi đề xuất phương pháp giải bài toán tối ưu hàm nhiều biến bằng cách sử dụng giải thuật di truyền và tính toán song song.

1. Giới thiệu

Xử lý song song là xử lý trên nhiều bộ xử lý và các bộ xử lý này phải tham gia giải quyết cùng một bài toán [1],[4], [5]. Xử lý song song cần kết hợp giữa lập trình song song và thuật toán song song.

Thông thường để tối ưu hóa một hàm số nào đó người ta phải tính đạo hàm rồi tìm những điểm mà tại đó đạo hàm triệt tiêu. Tuy nhiên, đối với các hàm phức tạp thì việc này khó thực hiện. Giải thuật di truyền (Genetic Algorithms – GA) là một trong những giải thuật thích hợp nhất cho vấn đề này. Tuy nhiên, thời gian chạy GA là rất dài đối với bài toán có không gian tìm kiếm lớn. Mặt khác, GA là một giải thuật mang bản chất song song (luôn duy trì n lời giải chứa trong quần thể), vì vậy, song song hóa GA là hướng tiếp cận phù hợp cho việc cải thiện thời gian tính toán.

2. Lập trình song song và thuật toán song song

Trong môi trường song song, ở cùng một thời điểm có thể có nhiều hơn một chương trình được thực hiện, nghĩa là mỗi chương trình sẽ tự thực hiện các tiến trình của mình và chúng tương tác với nhau để không làm ảnh hưởng tới nhịp độ thực hiện của nhau [6], [7]. Do đó, người lập trình không chỉ viết chương trình, dữ liệu như trong môi trường tuần tự mà còn phải cung cấp các công cụ để đồng bộ hoá và điều khiển sự tương tác giữa các tiến trình.

Lập trình song song có các cách tiếp cận tương ứng với các loại kiến trúc của các máy tính song song như lập trình song song kiểu SIMD, kiểu MIMD với bộ nhớ chia sẻ hay phân tán [8]. Đối với những máy tính song song thì ngôn ngữ lập trình phải là mô hình được sử dụng để thực hiện song song và giải quyết bài toán đặt ra một cách

hiệu quả nhất. Một trong các mô hình lập trình song song đang được sử dụng phổ biến là mô hình truyền thông điệp. Trong mô hình truyền thông điệp, các tiến trình chia sẻ với nhau kênh truyền thông.

Thuật toán song song là một tập các tiến trình hoặc các tác vụ có thể thực hiện đồng thời và có thể trao đổi dữ liệu với nhau để kết hợp cùng giải một bài toán đặt ra [1], [7]. Thuật toán song song có thể xem như là một tập hợp các đơn thể độc lập, một số trong số chúng có thể thực hiện tương tranh trên máy tính song song. Vấn đề đặt ra là làm thế nào để xây dựng được những thuật toán song song để giải bài toán một cách hiệu quả trên những máy tính mà chúng ta có. Cách làm khá thông dụng là biến đổi các thuật toán tuần tự về song song nhưng vẫn bảo toàn được tính tương đương trong tính toán.

3. Giải thuật di truyền

Giải thuật di truyền là giải thuật tìm kiếm, chọn lựa các giải pháp tối ưu để giải quyết các bài toán thực tế khác nhau, dựa trên cơ chế chọn lọc của di truyền học: từ tập lời giải ban đầu, thông qua nhiều bước tiến hoá, hình thành tập lời giải mới phù hợp hơn, và cuối cùng tìm ra lời giải tối ưu nhất. Trong tự nhiên, mỗi cá thể muốn tồn tại và phát triển phải thích nghi với môi trường, cá thể nào thích nghi hơn thì tồn tại, cá thể nào kém thích nghi thì bị tiêu diệt. Thủ tục giải thuật di truyền được trình bày như sau:

Input: số cá thể, số lần lặp tối đa, xác suất lai ghép và đột biến, hàm tính độ thích nghi.

Output: quần thể đã tiến hóa.

Procedure *Giải thuật di truyền*;

Begin *t:=0;*

Khởi tạo ngẫu nhiên tập cá thể P(t);

Đánh giá độ phù hợp từng cá thể trong P(t);

while (*not điều kiện kết thúc*) ***do***

begin

t:=t+1;

chọn các cá thể từ tập P(t - 1);

lai tạo các cá thể được chọn tạo ra tập các cá thể mới P(t);

đột biến các cá thể trong tập P(t) theo xác suất;

đánh giá độ phù hợp các cá thể trong tập P(t);

end;

End;

Giải thuật di truyền gồm 5 thao tác cơ bản [2], đó là: mã hóa – mô tả di truyền cho lời giải của bài toán, tạo lập lời giải ban đầu, xây dựng hàm thích nghi, xây dựng các toán tử di truyền, xác định các tham số cho giải thuật.

3.1. Mã hóa – mô tả di truyền cho lời giải bài toán

Đây là vấn đề cần giải quyết trước khi giải bài toán với GA. Tuỳ thuộc vào nội dung của mỗi bài toán mà chúng ta có một cách mã hoá khác nhau. Các phương pháp mã hoá hay được sử dụng [2], [3], [9]:

+ Mã hóa dạng chuỗi nhị phân: mỗi NST (lời giải) là một chuỗi các bits 0 và 1.

+ Mã hóa thứ tự: mỗi NST là một chuỗi các số nguyên thể hiện thứ tự phân bố lời giải của bài toán.

+ Mã hóa theo giá trị: mỗi NST là một chuỗi các giá trị có mối quan hệ tương ứng với bài toán.

+ Mã hóa dạng cây: mỗi NST là một cây của một nhóm đối tượng nào đó.

3.2. Tạo lập lời giải ban đầu

Tập lời giải ban đầu thường được khởi tạo ngẫu nhiên từ không gian các lời giải. Cách tạo lập tập lời giải ban đầu phụ thuộc rất nhiều vào cách mã hoá NST.

3.3. Xây dựng hàm thích nghi

Hàm thích nghi đánh giá khả năng thích nghi của tập lời giải theo yêu cầu bài toán.

3.4. Xây dựng các toán tử di truyền

+ Toán tử chọn lọc: Trong GA, sau mỗi lần tiến hoá chúng ta chỉ giữ lại các cá thể có độ phù hợp cao còn các cá thể xấu bị loại bỏ. Một số quy tắc chọn lọc thường được sử dụng là: quy tắc chọn lọc xén, chọn lọc theo bàn Roulete, quy tắc chọn lọc nhiều lần.

+ Toán tử lai ghép: Bước 1: tạo ra tập NST từ các NST đã được chọn lọc bằng cách chọn ngẫu nhiên N (số lượng NST của quần thể) NST. Bước 2: sau khi chọn được N NST, lần lượt lấy ra từng cặp NST để lai ghép tạo ra 2 NST mới. Một số phương pháp lai ghép: lai ghép 1 điểm, lai ghép hai điểm, lai ghép mặt nạ, lai ghép vòng tròn.

+ Toán tử đột biến: đột biến gây ra thay đổi ngẫu nhiên trên từng bit của NST tạo ra một NST mới. Các phương pháp đột biến: đột biến đều, đột biến không đều.

3.5. Xác định các tham số cho giải thuật

+ Xác suất lai ghép: là mức độ thường xuyên của thao tác lai ghép [2]. Nếu không lai ghép thì NST con sao chép nguyên vẹn NST cha mẹ. Nếu có lai ghép thì NST con được xây dựng từ những phần của NST cha mẹ.

+ Xác suất đột biến: là mức độ thường xuyên của việc đột biến một phần của NST [2]. Xác suất đột biến phải nhỏ, vì nếu xác suất này cao thì sẽ làm giảm khả năng hội tụ.

+ Kích thước quần thể: nếu kích thước quần thể quá nhỏ thì khả năng lai ghép sẽ nhỏ và chỉ một phần nhỏ của tập tìm kiếm được xử lý. Ngược lại, nếu kích thước quá lớn thì tốc độ xử lý chậm. Số lượng cá thể tối ưu phụ thuộc vào bài toán cụ thể và phương pháp mã hoá cụ thể.

4. Thuật toán di truyền tuần tự cho bài toán tối ưu hóa hàm nhiều biến

4.1. Phát biểu bài toán

Phát biểu bài toán: Tìm cực trị hàm f với n biến liên tục f(x1, x2, ..., xn), trong đó xi có giá trị trong miền D = [a, b] là tập con của tập số thực.

4.2. Giải thuật di truyền tuần tự cho bài toán tối ưu hóa hàm nhiều biến

Không mất tính tổng quát, chúng ta có thể giả sử các biến của phương trình nằm trong đoạn [0, 1].

Mã hóa – mô tả di truyền cho lời giải bài toán

Như phần trên đã đề cập ở phần trên, các biến của phương trình có giá trị là kiểu số thực và nằm trong đoạn [0, 1], do đó đối với bài toán này chúng ta sử dụng phương pháp mã hóa theo giá trị là hợp lý nhất. Như vậy, mỗi lời giải sẽ là một dãy số thực được lưu vào trong một mảng một chiều (mỗi phần tử sẽ có kiểu dữ liệu là kiểu số thực) với các giá trị được sinh ngẫu nhiên trong đoạn [0, 1].

Tạo lập lời giải ban đầu

Theo cách mã hóa đã được xác định, chúng ta xây dựng một mảng hai chiều V[Size,N] để lưu tập lời giải ban đầu. Trong đó, Size là số hàng (số cá thể trong quần thể), N là số cột (số biến trong hàm mục tiêu). Trong chương trình, thủ tục **InitPopulation** sẽ có chức năng sinh ra tập lời giải ban đầu một cách ngẫu nhiên và lưu vào trong mảng V[Size,N].

Đánh giá độ thích nghi của mỗi cá thể

Việc đánh giá độ thích nghi của mỗi cá thể trong chương trình này chính là hàm mục tiêu mà bài toán đưa ra. Dưới đây là hàm mục tiêu của bài toán:

$$f(x_i) = \sum_{i=1}^{N} \left(\frac{9}{256} - (x_i - 0.25)^2 \times (x_i - 0.75)^2 + \log(1 + x_i) \right)$$

Trong chương trình, thủ tục tính độ thích nghi của mỗi cá thể là **CalculateFitness**. Thủ tục này sẽ tính tuần tự độ thích nghi của mỗi cá thể và lưu chúng vào một mảng một chiều Fitness[Size] với số phần tử chính là số cá thể được sinh ra ban đầu, tức là bằng chính số hàng của mảng V.

Chọn lọc các cá thể tốt dựa vào độ thích nghi

Trong giải thuật di truyền, tác vụ chọn lọc thường sử dụng hai phương pháp chọn lọc xén hay chọn lọc theo bàn quay Roulete để chọn lọc ra những cá thể có độ thích nghi tốt. Theo cách mã hóa lời giải và khởi tạo tập lời giải ban đầu, nếu sử dụng phương pháp chọn lọc xén, chúng ta phải sắp xếp mảng Fitness theo thứ tự tăng dần hoặc giảm dần và sắp xếp lại các hàng của mảng V tương ứng với các vị trí của Fitness đã thay đổi một cách đồng thời. Điều này sẽ làm cho thuật toán của chúng ta tốn khá nhiều thời gian. Do đó, với bài toán đang xét, chúng ta sử dụng phương pháp chọn lọc theo bàn quay Roulete để chọn lọc.

Thủ tục có chức năng chọn lọc trong chương trình là thủ tục **Select**. Trước tiên, chúng ta tính tổng độ thích nghi của i cá thể lưu vào mảng một chiều PartialSum[i], trong đó $i = \overline{1 - Size}$. Tổng độ thích nghi của cả quần thể TotalSum = PartialSum[Size]. Tiếp theo, chúng ta sinh ngẫu nhiên một giá trị RandVal trong đoạn [0, 1], nhân với TotalSum rồi gán lại cho chính nó. Thực hiện Size lần lặp, nếu PartialSum[i] ≥ RandVal thì chọn cá thể thứ i.

Lai ghép các cá thể

Sau khi chọn lọc được các cá thể có độ thích nghi tốt, chúng ta chọn ngẫu nhiên từng cặp trong chúng để lai ghép với xác suất lai ghép cho trước.

Trong chương trình, việc chọn ngẫu nhiên từng cặp để lai ghép được thực hiện bởi thủ tục **SelectParrent**. Thủ tục này sử dụng lại thủ tục Select để chọn ra từng cặp cá thể có độ thích nghi tốt một cách ngẫu nhiên để lai ghép. Thủ tục này chỉ dừng lại khi chọn ngẫu nhiên hai cá thể mà hai cá thể đó trùng nhau.

Sau khi thực hiện **SelectParrent** xong, chương trình sẽ sử dụng thủ tục **CrossOver** để lai ghép với xác suất cho trước. Trong thủ tục này, trước tiên chúng ta sinh một số R ngẫu nhiên trong đoạn [0, 1], nếu R > CrossOverProbability (xác suất lai ghép cho trước) thì giữ nguyên cặp cá thể đó, ngược lại thì cho chúng lai ghép. Việc lai ghép được thực hiện khá đơn giản. Sử dụng một vòng **for** chạy từ biến đầu tiên của các thể đến biến cuối cùng, cứ mỗi lần như vậy chúng ta thực hiện như sau:

```
a = (double)ran.Next(sMax)/ (double)sMax;
Child1[i] = a * Parrent1[i] + (1 - a) * Parrent2[i];
a = (double)ran.Next(sMax)/ (double)sMax;
Child2[i] = a * Parrent1[i] + (1 - a) * Parrent2[i];
```

Đột biến

Sau khi thực hiện tác vụ lai ghép, tác vụ đột biến sẽ được thực hiện với xác suất đột biến cho trước. Tác vụ đột biến sẽ giúp chương trình tránh được việc nhận giá trị tối

ưu cục bộ. Trong chương trình, thủ tục **Mute** sẽ thực hiện tác vụ này. Trong thủ tục **Mute**, chúng ta sinh ngẫu nhiên một số R trong đoạn [0, 1], nếu R ≤ MutationProb (xác suất đột biến cho trước) thì chúng ta đột biến cá thể được chọn, ngược lại thì giữ nguyên. Tác vụ đột biến được thực hiện như sau:

+ Chọn ngẫu nhiên một ví trí trên cá thể được chọn để đột biến bằng cách: sinh ngẫu nhiên một số nguyên rồi chia lấy dư cho số biến của cá thể đó, chúng ta sẽ lấy được ví trí cần đột biến.

+ Đột biến: sinh ngẫu nhiên một số trong đoạn [0, 1], gán giá trị của biến ở vị trí đã tìm được ở thao tác trên bằng số vừa mới được sinh ngẫu nhiên.

Thay thế quần thể hiện tại bởi quần thể đã được tiến hóa

Sau khi thực hiện các tác vụ gene, chương trình sẽ thay thế quần thể hiện tại bởi quần thể được tiến hóa bằng thủ tục **ReplacePopulation**. Thủ tục này sẽ ghi lại các cá thể mới được sinh ra vào mảng V.

Quá trình tiến hóa trên (tính độ thích nghi của quần thể, chọn lọc, lai ghép, đột biến, thay thế quần thể hiện tại bởi quần thể đã được tiến hóa) được lặp đi lặp lại cho đến khi điều kiện dừng (được cho trước) được thỏa mãn.

Để kiểm tra tính hội tụ của thuật toán, cứ mỗi lần thực hiện quá trình tiến hóa chương trình sẽ tính độ thích nghi trung bình của quần thể và độ phương sai về độ thích nghi của các cá thể trong quần thể.

Thủ tục tính toán chính trong chương trình mô phỏng GA tuần tự

Input: N (số biến), Size (số cá thể), MaxNumOfGen (số lần lặp tối đa), CrossOverProb (xác suất lai ghép), MutationProb (xác suất đột biến), hàm $f(x_i)$.

Output: mảng V[Size,N] (mảng lưu quần thể đã tiến hóa).

```
Procedure    SerialGA;
  Begin
    NumberOfGeneration:=0; //biến lặp
    InitPopulation; //Khởi tạo quần thể ban đầu
    while (NumberOfGeneration < MaxNumOfGen) do
      begin
      CalculateFitness; //Tính độ thích nghi của các cá thể trong quần thể
      For i=1 to Size/2 do
        begin
          SelectParrent; //Chọn các cá thể có độ thích nghi tốt và lấy từng cặp
```

```
        CrossOver; //Lai ghép các cặp cá thể đã chọn với xác suất cho trước
        Mute; //Đột biến cá thể với xác suất cho trước
      end;
    ReplacePopulation; //Thay thế quần thể hiện tại bằng quần thể đã tiến hóa
    NumberOfGeneration = NumberOfGeneration + 1;
  end;
End;
```

5. Song song hóa giải thuật di truyền trong bài toán tối ưu hóa hàm nhiều biến

Với chương trình tuần tự như trên, chúng ta thấy có rất nhiều tác vụ được tính toán độc lập với nhau, nhất là tác vụ tính độ thích nghi trung bình của các cá thể trong quần thể.

Trong môi trường song song, giả sử chúng ta có S là số cá thể trong quần thể, N là số processor chạy song song với nhau. Mô hình được sử dụng trong phương pháp này là mô hình master – slave, trong đó có một processor làm master còn N-1 processor còn lại làm slave.

- Mã hóa – mô tả lời giải di truyền cho bài toán: Việc mã hóa được thừa kế lại từ phương pháp mã hóa trong chương trình tuần tự, tức là mã hóa theo giá trị.

- Khởi tạo quần thể ban đầu: việc khởi tạo quần thể ban đầu được thực hiện giống như giải thuật tuần tự và công việc này được giao cho master làm.

- Tại mỗi lần lặp: các bước mà master và các slave sẽ thực hiện được liệt kê trong bảng 1.

Bảng 1. *Các bước mà master và slave thực hiện tại mỗi lần lặp*

Master	**Slave i** ($1 \leq i \leq N-1$)
Gởi toàn bộ các cá thể trong quần thể tới tất cả các slave	Nhận toàn bộ các cá thể trong quần thể tới tất cả các slave
	Tính độ thích nghi của S/(N-1) cá thể
Nhận và tổng hợp các kết quả	Gởi kết quả về cho master
Gởi độ thích nghi của toàn quần thể đến tất cả các slave	Nhận độ thích nghi của toàn quần thể
	Dựa vào độ thích nghi để chọn lọc, lai ghép, đột biến và sinh ra S/(N-1) cá thể của thế hệ tiếp theo

Nhận các cá thể được sinh ra ở thế hệ tiếp theo	Gởi các cá thể được sinh ra ở thế hệ tiếp theo về cho master
Tổng hợp các cá thể mới lại	
Thay thế quần thể hiện tại bởi quần thể mới	

Nhận xét:

+ Tất cả các tác vụ của giải thuật di truyền tuần tự (tính độ thích nghi, chọn lọc, lai ghép, đột biến) đều được song song hóa.

+ Cấu trúc thuật toán tuần tự không bị phá vỡ cho nên chất lượng lời giải của giải thuật sau khi song song hóa được đảm bảo như giải thuật tuần tự.

+ Chi phí truyền thông cao, thời gian thực hiện thuật toán phụ thuộc vào chất lượng máy móc, các tác vụ gene thực hiện trên chỉ một quần thể.

+ Thuật toán không đòi hỏi topology mạng phức tạp.

Các hàm và thủ tục trong chương trình mô phỏng giải thuật di truyền được song song hóa đa phần được thừa kế từ chương trình mô phỏng giải thuật di truyền tuần tự. Các thủ tục chọn lọc, lai ghép, đột biến được thừa kế hoàn toàn từ giải thuật di truyền tuần tự. Trong chương trình song song, chỉ có thủ tục tính độ thích nghi của các cá thể được thay đổi, và có một số thao tác được phát sinh thêm. Để thực hiện chương trình song song này, chúng ta cần thực hiện những bước sau:

+ Xây dựng một **interface Ical** dùng để khai báo các hàm cần tính song song (thực hiện các tác vụ được song song hóa).

+ Xây dựng **project CalServer** gồm nội dung của các hàm đã được khai báo trong **Ical**. Trong project này phải bổ sung References file Ical.dll được sinh ra sau khi xây dựng interface **Ical**.

+ **Ical** và **CalServer** được lưu ở master và các slave để thực hiện tác vụ được song song hóa.

+ Xây dựng **project Client** được lưu ở master để chạy chương trình chính. Trong project này có hai lớp chính: **Ketnoi** và **GlobalMembersGA**. Trong lớp **Ketnoi**, công việc chủ yếu là triệu gọi master và các slave tính toán các tác vụ được song song hóa. **GlobalMembersGA** là lớp chính để chạy chương trình. Trong lớp **GlobalMembersGA**, các hàm và thủ tục được giữ nguyên giống như giải thuật tuần tự, chỉ có thay đổi ở thủ tục **CalculateFitness**. Ở chương trình song song, thủ thục này chỉ có công việc là tập hợp lại độ thích nghi của các cá thể đã được master và các slave tính toán.

+ Trong lớp **GlobalMembersGA**, thủ tục tính toán chính **PGA** cũng có một số sự thay đổi nhỏ. Trong thủ tục này, chúng ta phải mở kênh truyền thông, tạo các tuyến

đoạn (tương ứng với số processor), khởi động các tuyến đoạn. Các tuyến đoạn này sẽ được lớp **Ketnoi** sử dụng lại để triệu gọi master và các slave tính toán.

+ Lớp **GlobalMembersGA** phải có khai báo các địa chỉ IP của master và các slave để lớp **Ketnoi** sử dụng cho việc triệu gọi.

Thủ tục tính toán chính trong chương trình được song song hóa:

Input: số tiến đoạn được mở tương ứng với số máy, địa chỉ IP của các máy tham gia vào quá trình tính toán, N (số biến), Size (số cá thể), MaxNumOfGen (số lần lặp tối đa), CrossOverProb (xác suất lai ghép), MutationProb (xác suất đột biến), hàm $f(x_i)$.

Output: mảng V[Size,N] (mảng lưu quần thể đã tiến hóa).

```
Procedure    PGA;
  Begin
    Mở kênh truyền thông và khởi tạo các tuyến đoạn;
    NumberOfGeneration:=0;   InitPopulation;
    while (NumberOfGeneration < MaxNumOfGen) do
      begin
        Các slave tính độ thích nghi của các cá thể do master gởi đến;
        Master tập hợp độ thích nghi của các cá thể trong quần thể;
        For i=1 to Size/2 do
          begin
            SelectParrent;
            CrossOver;
            Mute;
          end;
        ReplacePopulation;
        NumberOfGeneration = NumberOfGeneration + 1;
      end;
  End;
```

Kết quả thực nghiệm của chương trình song song:

+ Chương trình thử nghiệm được chạy trên 2 máy. Máy thứ nhất là máy đã dùng để chạy chương trình tuần tự. Máy này được sử dụng vừa là server vừa là client thứ nhất.

+ Do mỗi lần chạy chương trình sẽ tạo các biến ngẫu nhiên trong đoạn [0,1] nên

mỗi lần chạy sẽ cho các kết quả khác nhau, vì vậy chúng ta chỉ xét kết quả mà nó thể hiện tính hội tụ của thuật toán.

+ Độ thích nghi trung bình của quần thể tăng dần và phương sai về độ thích nghi của các cá thể trong quần thể giảm dần.

6. Đánh giá chương trình song song với chương trình tuần tự

6.1. Đánh giá chương trình song song

+ Chương trình song song không phá vỡ cấu trúc của thuật toán tuần tự cho nên chất lượng lời giải của nó tương tự như chương trình tuần tự.

+ Chương trình chỉ mới song song hóa được một phần dựa theo ý tưởng đã nêu (song song hóa tác vụ tính toán độ thích nghi của các cá thể trong quần thể).

+ Chương trình không đòi hỏi topology mạng phức tạp, chi phí giao tiếp hơi lớn, thời gian thực hiện phụ thuộc vào chất lượng của các máy thực hiện.

+ Chương trình phải khai báo địa chỉ IP của các máy một cách thủ công. Do vậy, cứ mỗi lần muốn nâng hay giảm số máy thực hiện thì chúng ta phải khai báo thêm hay xóa bớt đi các địa chỉ IP của các máy.

6.2. So sánh chương trình song song với chương trình tuần tự dựa trên các kết quả thực nghiệm

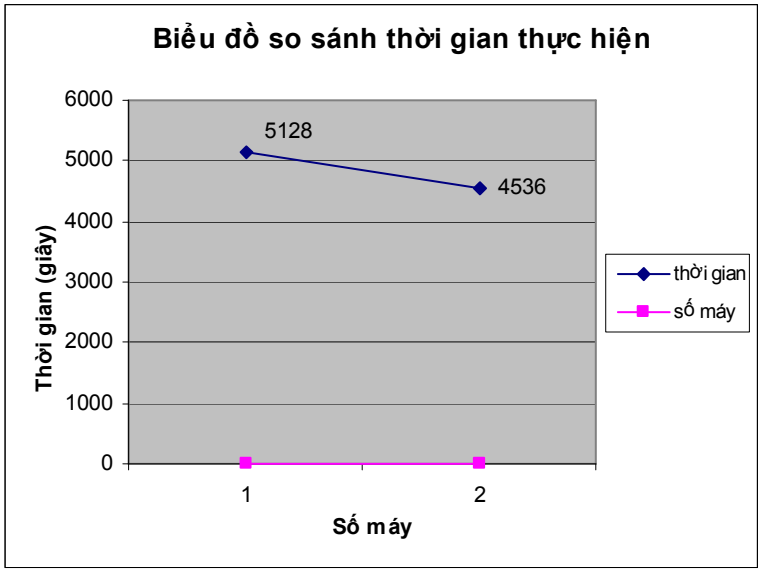
+ Hai chương trình song song và tuần tự đều có: N=500, Size=1000, lặp 500 lần, xác suất lai ghép là 0.1, xác suất đột biến là 0.01, hàm tính độ thích nghi là f(xi) được cho như trên.

+ Dựa vào kết quả đưa ra các file của hai chương trình, chúng ta thấy rằng kết quả của chương trình tuần tự và chương trình song song là tương tự nhau. Kết quả của cả hai chương trình đều cho cho chúng ta thấy được tính hội tụ của thuật toán.

+ Chi phí để thực hiện chương trình song song tốn hơn chương trình tuần tự.

+ Thời gian thực hiện thuật toán song song ngắn hơn thuật toán tuần tự. Tuy nhiên, chương trình song song hiện tại chỉ mới song song được một phần cho nên thời gian rút ngắn không đáng kể. Thời gian thực hiện giải thuật tuần tự là 1 giờ 26 phút 08 giây (5.128 giây), thời gian thực hiện trên 2 máy 1 giờ 15 phút 36 giây (4.536 giây).

+ Với các thời gian của các kết quả thực nghiệm trên, chúng ta có biểu đồ so sánh thời gian thực hiện giữa chương trình tuần tự với chương trình được song song hóa chạy trên 2 máy như hình sau.



Hình 1. *Biểu đồ so sánh thời gian thực hiện tuần tự và song song*

7. Kết luận

Trong phạm vi bài báo này, chúng tôi ứng dụng thuật toán song song và giải thuật di truyền cho bài toán tối ưu hoán hàm nhiều biến. Đặc biệt, tập trung phân tích, chuyển đổi giải thuật di truyền cho bài toán tối ưu hoán hàm nhiều biến từ môi trường tuần tự sang môi trường song song. Các kết quả thực nghiệm cho thấy tính hiệu quả của thuật toán song song.

Tuy nhiên, chương trình song song hiện tại vẫn còn một số nhược điểm. Đó chính là động lực để chúng tôi đưa ra hướng phát triển của bài báo:

+ Tiếp tục song song hóa thêm một số tác vụ trong chương trình song song đã mô phỏng được để rút ngắn hơn nữa thời gian thực hiện thuật toán.

+ Sau khi mô phỏng được chương trình song song hóa nhiều tác vụ, chúng tôi sẽ thử nghiệm trên nhiều máy (3 máy, 4 máy, ...) để tìm được ngưỡng về số máy mà với ngưỡng này thời gian thực hiện chương trình sẽ tối ưu nhất.

TÀI LIỆU THAM KHẢO

1. Barry Wilkingson, Michael Allen, *Parallel Programming,Technigues and Applications Using Networked Workstations and Parallel Computers,* Prentice Hall New Jersey, 2005.
2. Darrent Whitley, *A Genetic Algorithms Tutorial*, Computer Science Department, Colorado State University, 2000.
3. Randy L. Haupt, Sue Ellen Haupt, *Practical Genetic Algorithms, Second Edition,* John Wiley & Sons, Inc., Hoboken, New Jersey, 2004.

4. M. Sasikumar, Dinesh Shikhare, P. Ravi Prakash, *Introduction to Parallel Processing*, Prentice - Hall, 2000.

5. Seyed H. Roosta, *Parallel Processing and Parallel Algorithms*, Theory and Computation, Springer, 1999.

6. M. Sasikumar, Dinesh Shikhare, and P. Ravi Prakash, *Introduction to Parallel Processing*, Prentice – Hall, 2000.

7. Michael J. Quinn, *Parallel Programming in C with MPI and OpenMP*, McGraw – Hill, 2004.

8. B.H.V. Topping, J. Sziveri, A. Bahreinejad, J.P.B. Leite, B. Cheng, *Parallel processing, neural network and genetic algorithms*, 1998.

9. Alga,E.,and M.Tomassini, *Parallelism and evolutionary algorithms*, 2002.

A METHOD OF SOLVING THE MULTI VARIABLE FUNCTIONS OPTIMIZATION USING PARALLEL AND GENETIC ALGORITHMS

Nguyen Mau Han, Nguyen Dinh Quy, Nguyen Hoang Ha
College of Sciences, Hue University

SUMMARY

Methods to solve multi variable functions optimization have been early acknowledged in Mathematics. In this field some complex problems have been dealt with the difficulty in finding out solutions. In this paper, we propose a method to solve the variable functions optimization using genetic algorithms and parallel processing.