CÁC CHIẾN LƯỢC SONG SONG HÓA THUẬT TOÁN TUẦN TỰ

Nguyễn Việt Anh

Khoa Công Nghệ, ĐHQG HN

Phạm Trần Nhu

Viện Công Nghệ Thông tin, TT KHTN QG

Tóm tắt: Trong nhiều chiến lược song song hoá thuật toán tuần tự có ba chiến lược thiết kế chương trình song song tương đối phổ biến là song song hoá kết quả, song song hoá đại diện và song song hoá chuyên biệt. Thông thường để song song hoá một bài toán chỉ sử dụng một trong ba chiến lược thiết kế nêu trên. Bài báo này sẽ phân tích và so sánh các cách tiếp cận này trong môi trường Parallel Virtual Machine (PVM) đối với bài toán sinh ra số nguyên tố.

Mở đầu

Trong rất nhiều chiến lược khác nhau để song song hoá thuật toán tuần tự có ba chiến lược thiết kế chương trình song song tương đối phổ biến là song song hoá kết quả, song song hoá đại diện và song song hoá chuyên biệt. Mặc dù, trong từng bài toán cụ thể việc kết hợp các chiến lược thiết kế có thể cho ta kết quả thú vị, song thông thường tuỳ thuộc tính chất bài toán chỉ một chiến lược thiết kế thuật toán song song được chọn nhằm đạt hiệu quả cao nhất. Trong phần đầu chúng tôi trình bày ba chiến lược thiết kế nêu trên. Phần thứ hai chúng tôi trình bày môi trường Parallel Virtual Machine (PVM) được dùng trong quá trình thử nghiệm bài toán sinh ra số nguyên tố trong thuật toán mã hoá RSA . Tiếp theo chúng tôi trình bày một số thử nghiệm và kết quả của nó.

1. Các chiến lược song song hoá phổ biến

1.1 Song song hoá kết quả

Việc phân loại các chiến lược thiết kế thuật toán song song phụ thuộc vào tính chất bài toán. Song song hoá kết quả (xem [1]) là cơ chế tính toán song song tập trung trên toàn bộ dữ liệu của bài toán. Mỗi bộ xử lý sẽ cho một phần kết quả của bài toán và các bộ xử lý hoạt động song song sao cho các phần việc được thực hiện độc lập tối đa có thể. Sau khi các phần việc hoàn thành, công đoạn cuối cùng là kết hợp các thành phần để được kết quả hoàn chỉnh. Các lớp bài toán chia - để - trị, bảng đen (trong hệ chuyên gia) và lơi - lặp (iterative relaxtion) thường sử dụng chiến lược song song hoá kết quả để thiết kế chương trình song song.

Mỗi bài toán trong lớp các bài toán chia-để -trị (xem [2]) thường được chia thành các bài toán thành phần. Mỗi bài toán thành phần được giải quyết độc lập và kết quả cuối cùng là sự kết hợp các kết quả của bài toán thành phần.

Trong hệ thống bảng đen (xem [3]) cấu trúc dữ liệu thể hiện mô hình tính toán. Mỗi bộ xử lý độc lập có trách nhiệm quản lý, kiểm tra trạng thái hiện thời và cập nhật nó nếu cần.

Trong lợi - lặp (xem [4]) người ta chia không gian dữ liệu thành các vùng có quan hệ láng giềng, mỗi vùng này được một hay nhiều bộ vi xử lý đảm nhiệm. Mỗi bộ vi xử lý này đảm trách công việc theo từng vùng song song và khi cần có sự truyền thông điệp với các láng giềng.

Việc thiết kế chương trình theo chiến lược song song hoá kết quả thông qua năm giai đoạn. Giai đoạn đầu cần phải mô hình kết qủa dưới dạng cấu trúc dữ liệu gồm nhiều thành phần, đồng thời xác định được sự phụ thuộc giữa chúng. Giai đoạn hai, phân mỗi bộ xử lý đảm nhiệm công việc cho một hay nhiều thành phần và việc phân chia này phải bảo đảm vấn đề hiệu suất của các bộ xử lý. Giai đoạn ba, xác định nguồn tài nguyên cần thiết để thực hiện việc tính toán các thành phần. Bước tiếp theo xác định cách lấy các giá trị kết quả của các thành phần khi thực hiện xử lý song song. Cuối cùng, là giai đoạn kết hợp các kết quả thành phần để được kết quả bài toán và kết thúc các xử lý.

1.2 Song song hoá đại diện

Thiết kế chương trình theo cơ chế song song hoá đại diện xác định cụ thể công việc phải thực hiện để song song hoá. Như vậy, để giải quyết một bài toán có nhiều công đoạn, mỗi công đoạn của bài toán được giải quyết song song cho đến khi hoàn thành công đoạn đó và các công đoạn tiếp theo cũng được thực hiện tương tự cho đến khi bài toán được giải quyết. Các mô hình của chiến lược này là chủ - tớ, tính toán - tổng hợp - truyền thông.

Trong mô hình chủ - tớ (xem [4]) bài toán cần giải được chia thành các vấn đề phụ thuộc lẫn nhau. Các bộ xử lý đóng vai trò tớ trong mô hình có nhiệm vụ xử lý các vấn đề này và giữa chúng được điều phối bởi bộ xử lý đóng vai trò chủ. Khác với phương pháp chia - để - trị, các vấn đề ở đây không nhất thiết cùng được tiến hành giải quyết, mà có thể được giải quyết song song một cách tuần tự.

Trong mô hình tính toán - tổng hợp - truyền thông (xem [2]) thiết kế bao gồm ba giai đoạn. Giai đoạn tính toán thực hiện các tính toán cơ bản có tính chất cục bộ, giai đoạn tổng hợp kết hợp các dữ liệu cục bộ thành dữ liệu toàn cục, giai đoạn truyền thông trả lại các thông tin toàn cục cho các bộ xử lý.

Việc thiết kế chương trình theo chiến lược song song hoá đại diện được thực hiện thông qua ba giai đoạn. Giai đoạn đầu xác định các công việc cần phải thực hiện bởi các bộ xử lý. Trong giai đoạn tiếp theo cần phải xác định bộ xử lý đóng vai trò điều khiển các công việc.Cuối cùng cần phải nhận biết được kết quả công việc.

1.3 Song song hoá chuyên biệt

Trong chiến lược song song hoá chuyên biệt, bài toán cần giải quyết bao gồm nhiều công việc, mỗi công việc có đặc thù riêng được giao cho một bộ xử lý chuyên

dụng. Ngoài ra hệ thống cần một bộ xử lý giữ vai trò agent, điều phối quá trình thực hiện công việc. Trong mỗi công việc đặc thù, các phần việc nhỏ hơn được thực hiện song song. Sau khi các phần việc này hoàn thành, cần tiến hành "phối hợp" các kết quả để hoàn thành công việc.

Một số mô hình thường gặp trong chiến lược thiết kế này là: ống dẫn và bộ lọc, khách - phục vụ.

Trong mô hình ống dẫn và bộ lọc [3] dòng dữ liệu kết quả ra của quá trình xử lý A là dữ liệu vào của quá trình xử lý B.

Trong mô hình khách - phục vụ [4], các yêu cầu từ các bộ xử lý đóng vai trò khách được gửi đến các bộ xử lý đóng vai trò phục vụ để xử lý.

2. Parallel Virtual Machine (PVM)

PVM là tập hợp phần mềm công cụ và thư viện kết hợp các máy tính khác nhau về cấu hình như một máy ảo. Mục tiêu chính của hệ thống PVM là tăng hiệu suất tính toán bằng cách kết hợp các máy tính thông qua xử lý song song trên các bộ sử lý.

Hệ thống PVM gồm hai thành phần chính. Thứ nhất là tiến trình thường trú (daemon) thường gọi là pvmd3 hay pvmd chạy trên tất cả các máy tính thành viên của máy tính song song ảo. Để thực hiện ứng dụng PVM, trước hết tạo ra một máy ảo bằng cách chạy PVM. Khi đó ứng dụng PVM sẽ được khởi động trên tất cả các máy tính thành viên. Thứ hai là thư viện các hàm giao diện PVM thường gọi là libPVM3.a. Thư viện gồm các hàm phục vụ cho việc truyền thông điệp, sinh tiến trình, phối hợp công việc v.v...

Mô hình tính toán PVM dựa trên khái niệm một ứng dụng bao gồm nhiều công việc. Hai phương pháp thường được PVM sử dụng là song song hoá chức năng và song song hoá dữ liệu. Phương pháp song song các chức năng, ứng dụng bao gồm nhiều chức năng, các chức năng được thực hiện song song. Phương pháp song song hoá dữ liệu các tiến trình xử lý như nhau nhưng mỗi tiến trình chỉ xử lý một phần dữ liệu của ứng dụng.

Hệ thống PVM hỗ trợ ngôn ngữ lập trình C, Fortran và C++. Trong đó ngôn ngữ C++ được sử dụng phổ biến nhất. Các thử nghiệm của chúng tôi cũng sử dụng ngôn ngữ C++.

2.1 Đặc điểm môi trường PVM

- Khả năng cấu hình các máy: Các công việc thực hiện trên các máy được lựa chọn bởi chính người dùng khi thực hiện PVM. Trong quá trình tính toán, việc thêm hay bớt các máy thành viên thực hiện rất dễ dàng.

- Khả năng truy nhập đến cấu hình phần cứng cho phép người dùng có thể chọn các máy thành viên phù hợp trong quá trình tính toán.

- Tính toán dựa trên bộ xử lý: Đơn vị của PVM là công việc, nhiều công việc có thể thực hiện trên một bộ xử lý, có một luồng điều khiển tuần tự độc lập thực hiện việc chuyển đổi giữa quá trình truyền thông và tính toán.

- Sử dụng mô hình truyền thông điệp: Điều phối việc truyền thông giữa các máy thành viên qua việc gửi/nhận thông điệp.

- Hỗ trợ mạng hỗn tạp: Cho phép các máy thành viên có cấu hình khác nhau, hỗ trợ nhiều bộ xử lý.

2.2 Nguyên lý hoạt động

Thông thường chương trình được viết trên PVM qua các bước khởi tạo các tiến trình, quản lý truyền thông điệp, tính toán, tổng hợp kết quả và hiển thị chúng, kết thúc các tiến trình.

- Khởi tạo tiến trình

Mỗi tiến trình trong hệ thống PVM được định danh bởi một số nguyên do pvmd cung cấp. Hàm **pvm_mytid()** thực hiện công việc này. Nó kết nạp tiến trình vào hệ thống PVM nếu chúng chưa được kết nạp. Hàm **pvm_spawn(char *task, char **argv, int flag, char *where, int intask, int *tids)** sinh ra các bản sao của tiến trình cho bởi tham số **task** trên máy ảo. Danh sách các đối số của tiến trình đó thông qua tham số **argv**. Cờ **flag** cho một số tuỳ chọn đặc biệt, thường sử dụng giá trị mặc định cho phép PVM tự xác định nơi khởi tạo tiến trình. Hàm trả về số nguyên cho biết số lượng các tiến trình được khởi tạo thành công, hoặc mã lỗi trong trường hợp ngược lại. Nếu thành công tham số **tids** danh sách các số nguyên định danh tiến trình.

- Quản lý truyền thông điệp

Quá trình gửi thông điệp trong hệ thống PVM gồm ba giai đoạn. Đầu tiên một vùng đệm được khởi tạo thông qua hàm **pvm_initsend(int encoding)** sẽ tạo ra vùng đệm dành cho dữ liệu mới. Giai đoạn tiếp theo thông điệp được nạp vào vùng đệm thông qua hàm **pvm_pk*()**. Tùy vào kiểu dữ liệu nạp vào vùng đệm, PVM xây dựng hàm tương ứng, ví dụ **pvm_pkint()** dành cho dữ liệu ,kiểu nguyên, **pvm_pkfloat()** dành cho kiểu thực. Công đoạn cuối cùng là việc thực hiện gửi thông điệp đến tiến trình khác thông qua hàm **pvm_send(int tid, int msgtag)** gửi thông điệp định danh bởi **msgtag** đến tiến trình định danh bởi **tid.**

Quá trình nhận thông điệp được thực hiện thông qua hàm **pvm_recv(int tid, int msgtag)**. Sau khi nhận thông điệp, dữ liệu được lấy ra từ vùng đệm thông qua hàm **pvm_upk*()**. Cần phải sử dụng hàm phù hợp với kiểu dữ liệu, ví dụ **pvm_upkint()** dùng cho kiểu dữ liệu kiểu nguyên, **pvm_upkfloat()** dùng cho dữ liệu kiểu thực.

- Kết thúc tiến trình

Trong quá trình hoạt động PVM cho phép loại bỏ tiến trình khi tiến trình đã thực hiện xong công việc hoặc không cần thiết nữa thông qua hàm **pvm_exit()**, cho tiến trình thường trú pvmd biết rằng tiến trình đó được loại khỏi hệ thống PVM.

3. Thử nghiệm, so sánh các chiến lược song song hoá

Trong quá trình thực hiện đề tài "Xử lý song song trên PVM ứng dụng trong bài toán bảo mật". Chúng tôi có sử dụng thuật toán RSA để mã hoá. Vấn đề mấu chốt của thuật toán RSA là việc sinh ra các số nguyên tố. Trong quá trình thử nghiệm chúng tôi thực hiện cài đặt bài toán tìm số nguyên tố theo ba chiến lược trên. Giải thuật mà chúng tôi dùng là giải thuật "sàng Eratosthenes".

Giải thuật sàng Eratosthenes:

Bước 1: Nhập 1.. n

Bước 2 : Loại bỏ 1

Bước 3: Lặp các bước sau đây cho đến $\sqrt{n}$

Bước 3.1: Tìm tiếp số đầu chưa loại bỏ, kí hiệu p, nếu tìm được p là số nguyên tố

Bước 3.2: Loại bỏ các số là bội của p kể từ p^2

3.1 Sử dụng chiến lược song song hoá kết quả

Chúng tôi dùng mảng logic để kiểm tra số nguyên tố. Phần tử đầu tiên của mảng biểu diễn cho số 1, phần tử thứ 2 biểu diễn cho số 2, tiếp tục như vậy. Nếu phần tử có giá trị là true thì số ứng với phần tử đó là số nguyên tố và ngược lại.

Tiến trình cha khởi tạo mảng, sinh các tiến trình con xử lý mỗi phần tử của mảng, các tiến trình con này có nhiệm vụ kiểm tra số ứng với vị trí cửa phần tử đó có phải là số nguyên tố hay không, sau đó tiến trình cha có nhiệm vụ tập hợp kết quả để hiển thị.

```
#define MAX 100000
#define TRUE 1
#define FALSE 0
void main(){
     int kq[ MAX] ,tid,tids[ MAX-2] ,i,num,res;
     (void) pvm_mytid() // Khởi tạo tiến trình
     kq[ 0] =FALSE;kq[ 1] =TRUE;
     //Sinh tiến trình con
     (void) pvm_spawn("nguyento",(char**)0,0,"",MAX-2,tids);
     //Gửi cho các tiến trình con để kiểm tra
     for(i=2;i<MAX;i++){
      pvm_intsend(PvmDataRaw);
       pvm_pkint(&i,1,1);
       pvm_send(tids[ i] ,1);
    }
     //Nhận kết quả
     for(i=2;i<MAX;i++){
      pvm_recv(-1,2);
       pvm_upkint(&num,1,1); pvm_upkint(&res,1,1);
       kq[ num] =res;
```

```
    }
    //Hiển thị kết quả
    for(i=0;i<MAX;i++)
     if(kq[ i] ) printf("%d",i);
    pvm_exit()// Kết thúc tiến trình
    exit(0);
}
```

Mỗi tiến trình con sẽ là nhiệm vụ kiểm tra và trả lại giá trị TRUE nếu đúng là số nguyên tố và ngược lại.

```
#define TRUE 1
#define FALSE 0
void main(){
     int p_tid,i,num,res;
     p_tid=pvm_parent(); //Nhận số định danh tiến trình
     //Nhận dữ liệu
     pvm_recv(p_tid,1);pvm_upkint(&num,1,1);
      //Kiểm tra số nguyên tố
      res=TRUE;i=2;

          while(res &&(i<sqrt(num)){
               res &&=((num %i)!=0);
               i++;
          }
          //Gửi kết quả
          pvm_initsend(PvmDataRaw);
          pvm_pkint(&num,1,1);pvm_pkint(&res,1,1);
          pvm_send(p_tid,2);
          pvm_exit();
          exit(0);
}
```

3.2 Sử dụng chiến lược song song hoá dại diện

Cũng giống như chiến lược song song hóa kết quả, sử dụng mảng logic để kiểm tra số nguyên tố. Tuy nhiên chỉ có một số lượng nhỏ các tiến trình kiểm tra thực hiện công việc kiểm tra, sau khi kiểm tra xong, các tiến trình này tiếp tục kiểm tra các phần tử khác.

```
#define MAX 100000
#define NPROCS 3
#define TRUE 1
#define FALSE 0

void main()
{
```

```
  int
kq[ MAX] ,tids[ NPROCS] ,bufid,msgtag,msglength,msgsrc,from,upto;
  int allocated=2,halt=0;
  (void) pvm_mytid();//Khởi tao tiến trình
  kq[ 0] = FALSE; kq[ 1] = TRUE;
  //Sinh tiến trình
  (void) pvm_spawn ("nguyento",(char **) 0,0,"",NPROCS,tids);
   while(NPROCS>0)
    {
      bufid = pvm_recv (-1,-1);
      pvm_bufinfo(bufid,&msglength,&msgtag,&msgsrc);
      switch(msgtag) {
        case REQUESTTAG :
          pvm_initsend(PvmDataRaw);
          pvm_pkint(&allocated, 1, 1);
          pvm_send(msgsrc,2);
          if (allocated < MAX)
            allocated += 100;
          else
            halted++;
          break;
        case RESULTTAG :
          // Nhận dữ liệu
          pvm_unpkint(&from, 1, 1);
          pvm_unpkint(&upto, 1, 1);
          if ((from < upto) && (upto < MAX))
            pvm_unpkint((&kq) + from, from - upto + 1, 1);
          break;
      };
    }
  // Hiển thị kết quả
  for(int i = 0; i < MAX; i++)
    if(kq[ i] )
      printf ("%d  ", i);
  pvm_exit();
  exit(0);
}
```

Đối với các tiến trình con, thực hiện kiểm tra số nguyên tố, sau khi kiểm tra xong gửi kết quả cho tiến trình cha và nhận các phần tử tiếp theo để thực hiện.

```
#define TRUE 1
#define FALSE 0
void main()
{
  int ptid,num,kq[ 100] ,dum,task_end,i;
```

```
    int task_start=0;
    ptid = pvm_parent();
    while(task_start < MAX)
      {
        // Gửi yêu cầu nhận số mới để kiểm tra
        pvm_initsend(PvmDataRaw);
        pvm_pkint(&dum,1, 1);
        pvm_send(ptid, 1);
        pvm_recv(ptid,2);
        pvm_upkint(&task_start,1, 1);
        if (task_start<MAX)
          {
            task_end = task_start + 100;
            if (task_end > MAX)
              task_end = MAX;
            for (num= task_start; num < task_end; num++)
              {
                kq[ num - task_start] = TRUE;
                i = 2;
                while(kq[ num - task_start] && (i < sqrt (num)))
                  {
                    kq[ num - task_start] &&= ((num % i) != 0);
                    i++;
                  }
              }
            // Gửi kết quả cho tiến trình cha
            pvm_initsend (PvmDataRaw);
            pvm_pkint (&task_start, 1, 1);
            pvm_pkint (&task_end, 1, 1);
            pvm_pkint (kq, task_end - task_start + 1, 1);
            pvm_send (ptid, 3);
          }
      }
    pvm_exit();
    exit(0);
}
```

3.3 Sử dụng chiến lược song song hoá chuyên biệt

Trong chiến lược này, chúng tôi khởi tạo các tiến trình kiểm tra số nguyên tố. Các số được tiến trình chính chuyển cho các tiến trình con để kiểm tra. Nếu số đó không phải là số nguyên tố, tiến trình tiếp tục kiểm tra số tiếp theo, nếu số đó là số nguyên tố và tiến trình khác chưa kiểm tra số đó thì sẽ kết nạp số đó vào tập kết quả. Nếu đã có tiến trình khác đã chọn số đó, tiến trình này sẽ tiếp tục kiểm tra như đã làm với tiến trình trước.

```
#define MAX 100000
#define TRUE 1
#define FALSE 0
void main()
{
  int kq[MAX],tid,num,i;
 (void) pvm_mytid();//Khởi tạo tiến trình
 (void) pvm_spawn ("nguyento",(char**)0,0,"",1, &tid);
  for (num = 2; number <= MAX; num++)
    {
      pvm_initsend (PvmDataRaw);
      pvm_pkint (&num, 1, 1);
      pvm_send (tid,1);
    }

  // Nhận kết quả
  pvm_recv (tid,2);
  pvm_unpkint (&num,1,1);
  pvm_unpkint (kq,num + 1, 1);
  kq[0] = FALSE;
  //Hiển thị kết quả
  for (i = 0; i < (num + 1), i++)
    printf ("%d  ", kq[i]);
  pvm_exit();
  exit(0);
}
```

Các tiến trình con thực hiện việc kiểm tra, đồng thời các tiến trình động được PVM được tạo ra khi có yêu cầu.

```
#define MAX 100000
#define TRUE 1
#define FALSE 0
void main()
{
  int ptid,ngto,n,kq[MAX];
  int next_tid = 0;
  int length = 0;
  ptid = pvm_parent();
  pvm_recv(ptid,1);
  pvm_unpkint(&ngto, 1, 1);
  while (n < MAX)
    {
      //Nhận số nguyên mới để kiểm tra
      pvm_recv (ptid,1);
```

```
        pvm_unpkint(&n, 1, 1);
        if (n < MAX)
          {
              if ((n % ngto) != 0)
              {
                if (!next_tid)
                  //Tao tiến trình mới
                  pvm_spawn("nguyento",(char**)0,0,"",1,&next_tid);
                  pvm_initsend(PvmDataRaw);
                  pvm_pkint(&n,1, 1);
                  pvm_send(next_tid, 1);
              }
          }
      }
    if (next_tid)
      {
        pvm_initsend(PvmDataRaw);
        pvm_pkint(&n, 1, 1);
        pvm_send(next_tid,1);
        pvm_recv(next_tid, 2);
        pvm_unpkint(&length, 1, 1);
        pvm_unpkint((kq + 1), length, 1);
      }
    kq[ 0] = ngto;
    length++;
    pvm_initsend(PvmDataRaw);
    pvm_pkint(&length, 1, 1);
    pvm_pkint(kq,length, 1);
    pvm_send(ptid,2);
    pvm_exit();
    exit(0);
}
```

3.4 Nhận xét

Chúng tôi tiến hành thử nghiệm trên ba máy tính với các cấu hình Pentium IV 2.4 Mhz bộ nhớ 512MB, Pentium III 750 Mhz bộ nhớ 256 MB và Xeron 1.7Mhz bộ nhớ 128MB. Sử dụng chiến lược song song hoá kết quả, chương trình thực hiện hiệu quả nhất. Tuy nhiên do dữ liệu chương trình nhỏ nên sử dụng các chiến lược này chưa cho hiệu quả chênh lệch rõ rệt. Về phương diện cài đặt, cài đặt chương trình theo chiến lược song song hoá kết quả dễ dàng hơn so với song song hoá đại diện và song song hoá chuyên biệt.

4. Kết luận

Để song song hóa thuật toán tuần tự có rất nhiều chiến lược, để cho hiệu quả cần phải chọn chiến lược phù hợp với thuật toán. Trong quá trình thực hiện thực hiện đề tài "Xử lý song song trên PVM ứng dụng trong bài toán bảo mật" chúng tôi đã thử nghiệm một số chiến lược tiếp cận khác nhau để tìm ra chiến lược phù hợp nhất bài toán của mình.

Tài liệu tham khảo

1. Carriero, N., Gelernter, D., How to write parallel programs: A guide to the perplexed, *Journal of the ACM*, 21(3), Sept 1989, pp 323-357.
2. Philip A.Nelson, Lawrence Snyder, *Programing paradigms for nonshared memory parallel computers*. In Leah H. Jamieson, Dennis Gannon and Robert J Douglas editors, The characteristics of Parallel Algorithms, The MIT Press, Cambridge, Massachusetts and London, England, 1987, pages 3-20.
3. Mary Shaw, Larger Scale Systems Require Higher-Level Abstractions, *5th International workshop on software specification and design*, 14(**2**), May,1989, 143-146.
4. Raphael A.Finkel, *Large - grain parallelism - Three case studies*. In Leah H. Jamieson, Dennis Gannon and Robert J Douglas editors, The characteristics of parallel algorithms, The MIT Press, Cambridge, Massachusetts and London, England, 1987, pages 21-63.
5. Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek and Vaidy Sunderam, *PVM: Parallel virtual machine - A Users' guide and tutorial for networked parallel computing*. The MIT Press, Cambridge, Massachusetts and London, England,1994.

VNU JOURNAL OF SCIENCE, Nat., Sci., & Tech., T.XIX, N$_o$4, 2003

PARALLELIZATION STRATEGIES OF SEQUENTIAL ALGORITHM

Nguyen Viet Anh

Faculty of Technology, VNU

Pham Tran Nhu

National center for Natural Sciecnces and Technology

Parallelization strategies: result parallelism, agenda parallelism and specialist parallelism will be analysed and compared on an experiment in Parallel Virtual Machine enviroment for prime algorithm.