



BÀI 5. LẬP TRÌNH SONG SONG VỚI OPENMP



NỘI DUNG BÀI HỌC

- Lập trình đa luồng
- Khái niệm OpenMP
- Các thành phần của OpenMP



5.1.LẬP TRÌNH ĐA LUỒNG (MULTITHREADED PROGRAMMING)



TIẾN TRÌNH (PROCESS)

- Một thực thể thực thi của chương trình, đã bắt đầu nhưng chưa kết thúc.
- Là đơn vị nhỏ nhất cấp phát tài nguyên.
- Tiến trình được tạo qua lời gọi hệ thống, vd `fork()` trong UNIX
- Hệ thống quản lý tiến trình thông qua khối điều khiển tiến trình
- Liên lạc giữa các tiến trình thông qua giao thức IPC

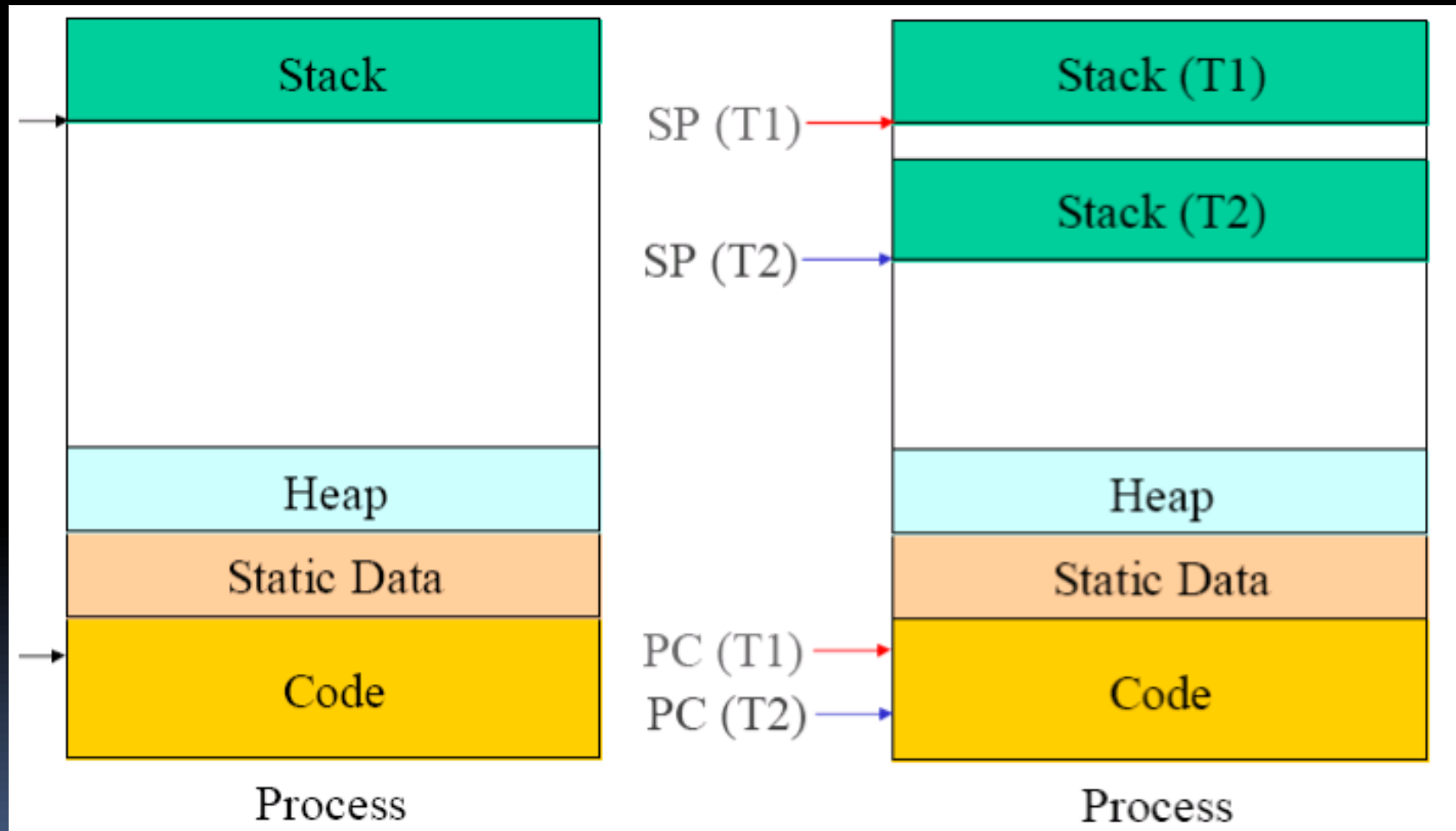
TIẾN TRÌNH (PROCESS)

- Theo quan điểm về hệ thống:
 - Tiến trình là đơn vị chiếm dụng tài nguyên: CPU, bộ nhớ, thanh ghi, thẻ tên.
 - Các tiến trình là riêng biệt: không được phép truy cập tài nguyên đến tiến trình khác.
 - Liên lạc giữa các tiến trình rất tốn chi phí.
- Tiến trình được nhìn theo 2 góc độ:
 - Chiếm dụng tài nguyên và thực thi lệnh
 - Tập hợp các luồng.

LUỒNG (THREAD)

- Luồng là đơn vị thực thi của tiến trình
- Một tiến trình bao gồm một hoặc nhiều luồng, mỗi luồng thì thuộc vào 1 tiến trình.
- Luồng có vùng nhớ ngăn xếp riêng, con trỏ lệnh riêng, các thanh ghi riêng.
- Các luồng trong tiến trình chia sẻ các tài nguyên khác của tiến trình. VD: bộ nhớ.
- Liên lạc giữa các luồng thông qua vùng nhớ của tiến trình

THREAD & PROCESS

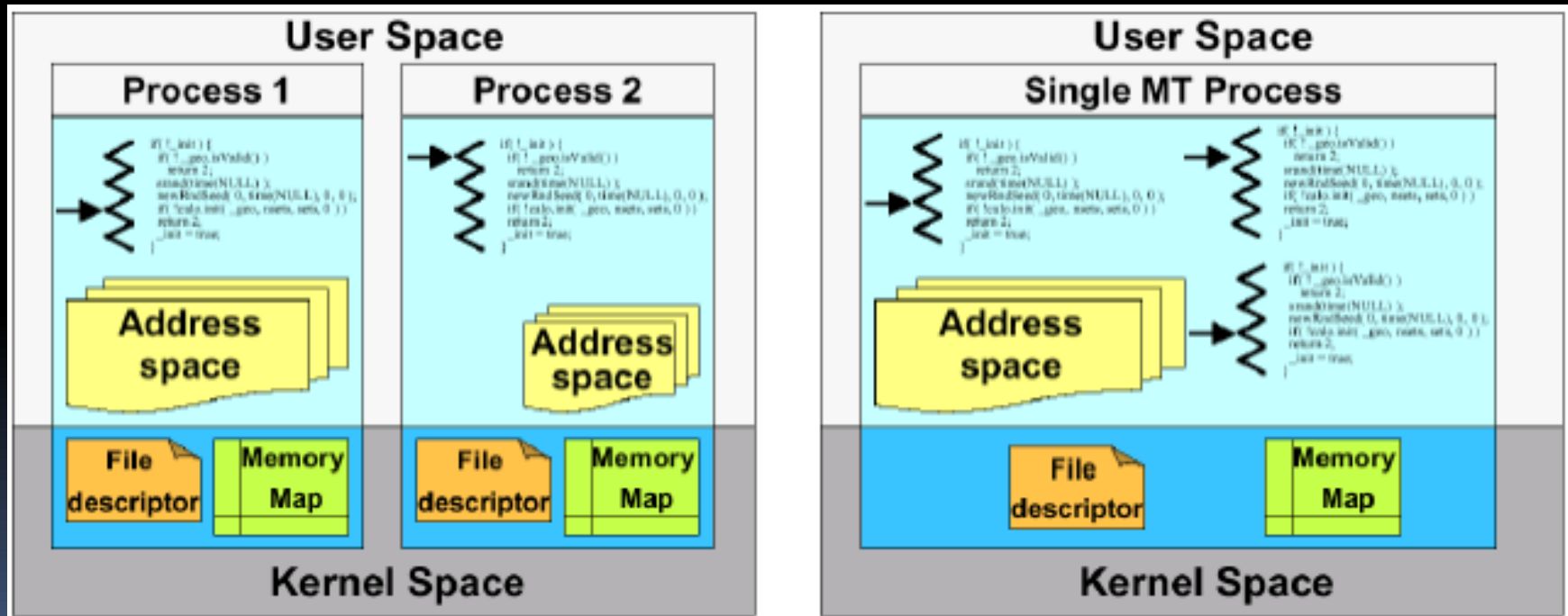


LẬP TRÌNH ĐA LUỒNG

- Theo quan điểm lập trình:
 - Luồng là dòng điều khiển độc lập (hàm)
 - Tham số của hàm là dữ liệu của luồng
 - Mỗi hàm thực hiện một công việc cụ thể -> một tiến trình có thể thực hiện nhiều công việc một lúc bằng cách chia nó thành các luồng -> lập trình đa luồng.
- Phân biệt với lập trình đa tiến trình
 - Chi phí khởi tạo, quản lý, kết thúc công việc
 - Chi phí trao đổi dữ liệu giữa các công việc
 - Hệ thống máy tính đem triển khai

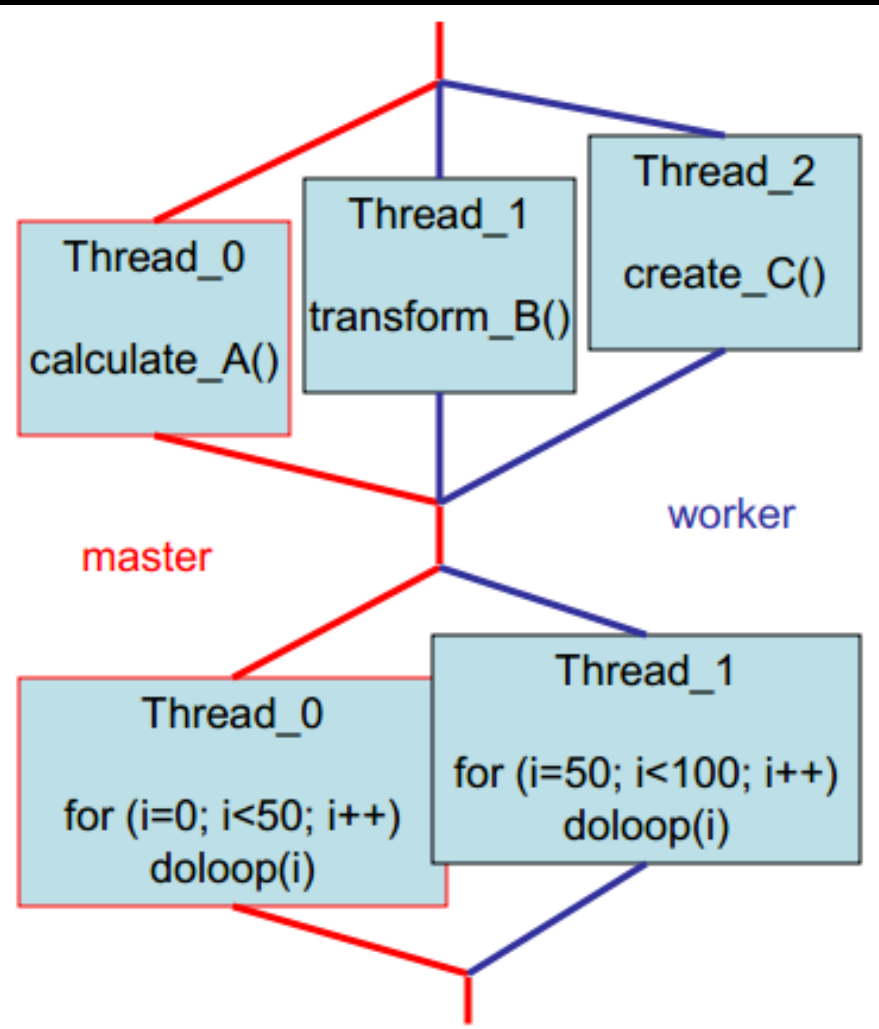
LẬP TRÌNH ĐA LUỒNG

- Mô hình lập trình áp dụng tốt cho hệ thống song song SMP (Symmetric Multi-Processing)



LẬP TRÌNH ĐA LUỒNG

```
01 main()
02 {
03     ...
04     ...
05     ...
06     calculate_A();
07     transform_B();
08     create_C();
09     ...
10     ...
11     for (i=0; i<100; i++)
12         doloop(i);
13     ...
14     ...
15     ...
16     ...
17 }
```



ƯU ĐIỂM LẬP TRÌNH ĐA LUỒNG

- Khai thác tối đa tính song song của hệ thống xử lý đối xứng (SMP)
- Sử dụng tối đa khả năng của BXL
- Tăng hiệu suất của chương trình ngay cả với máy tính đơn xử lý
- Tăng khả năng đáp ứng của chương trình
- Đưa ra cơ chế liên lạc giữa các công việc nhanh và hiệu quả hơn.

KHÓ KHĂN LẬP TRÌNH ĐA LUỒNG

- Đồng bộ hóa các công việc
- An toàn và toàn vẹn với dữ liệu chia sẻ
- Xử lý điều kiện đua tranh
- Dò lỗi chương trình



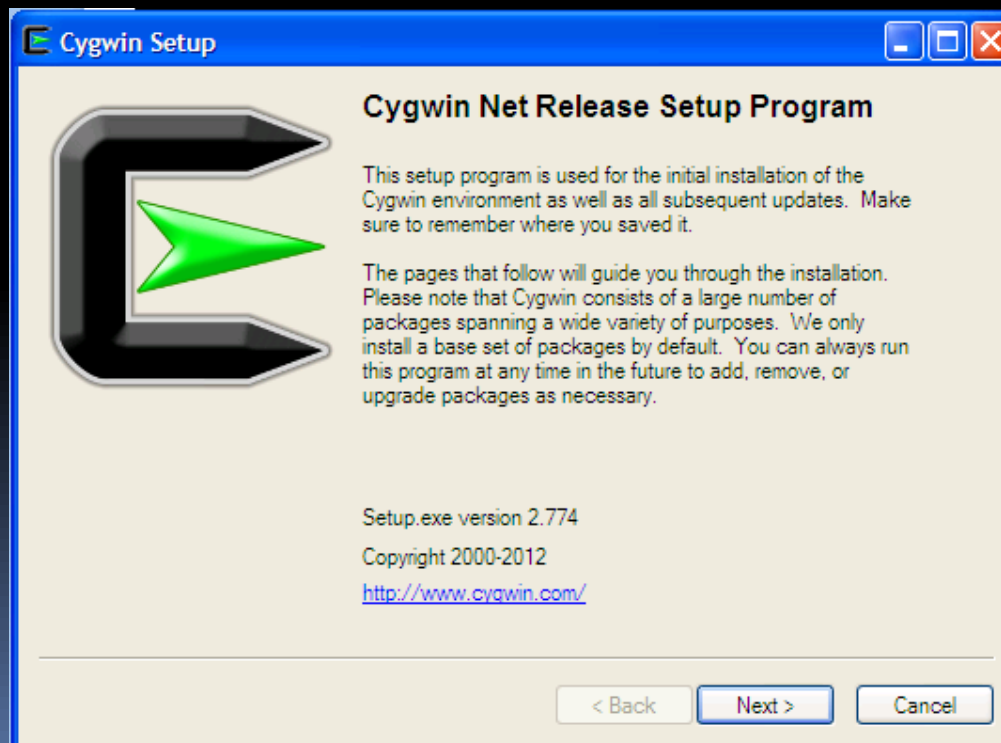
CYGWIN?

CYGWIN

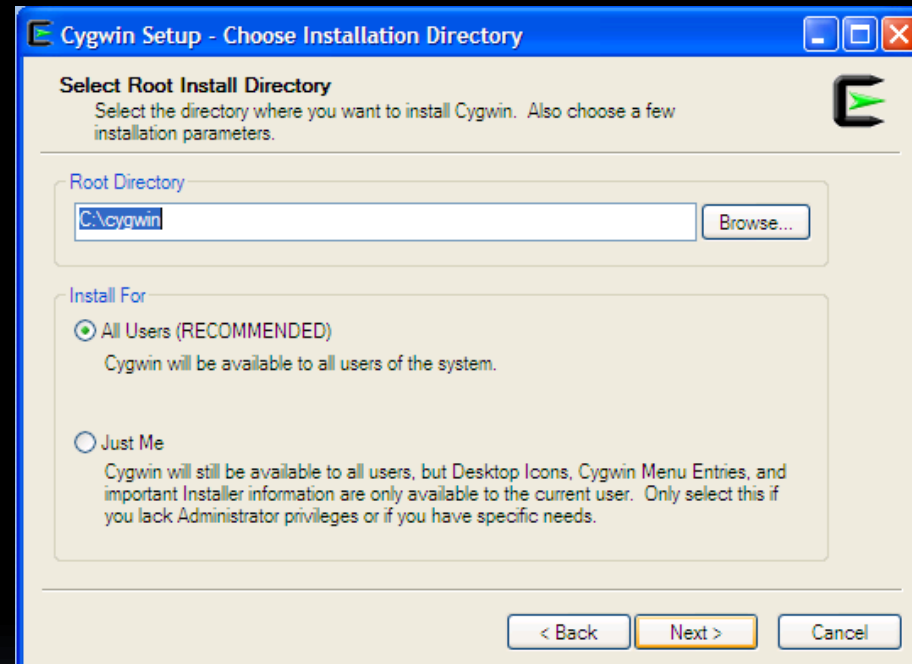
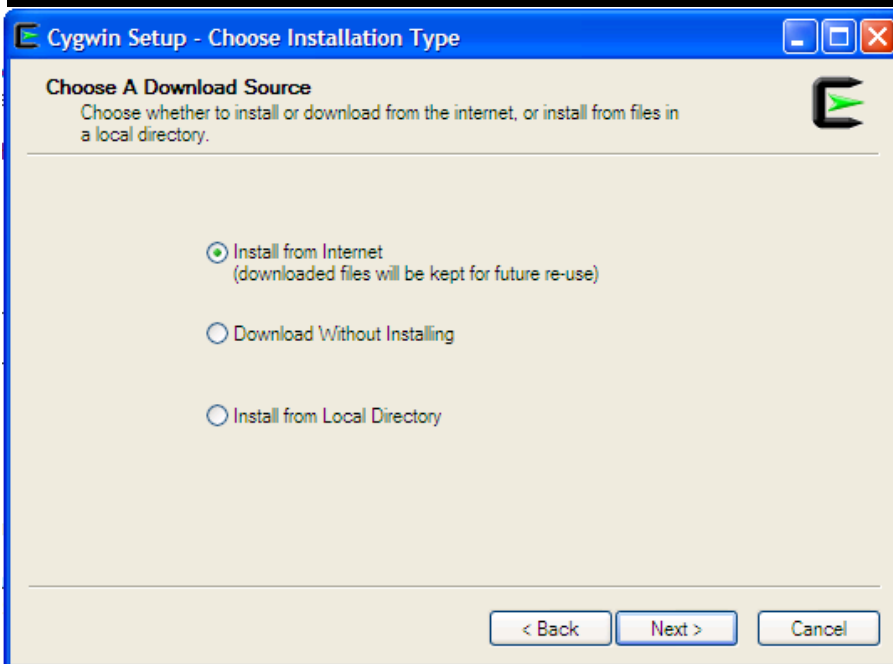
- Môi trường giống LINUX cho WINDOWS, bao gồm hai phần:
 - Một DLL (cygwin.dll) cung cấp các chức năng của Linux API
 - Một tập các công cụ, cung cấp cách nhìn và cảm giác như Linux

CÀI ĐẶT CYGWIN

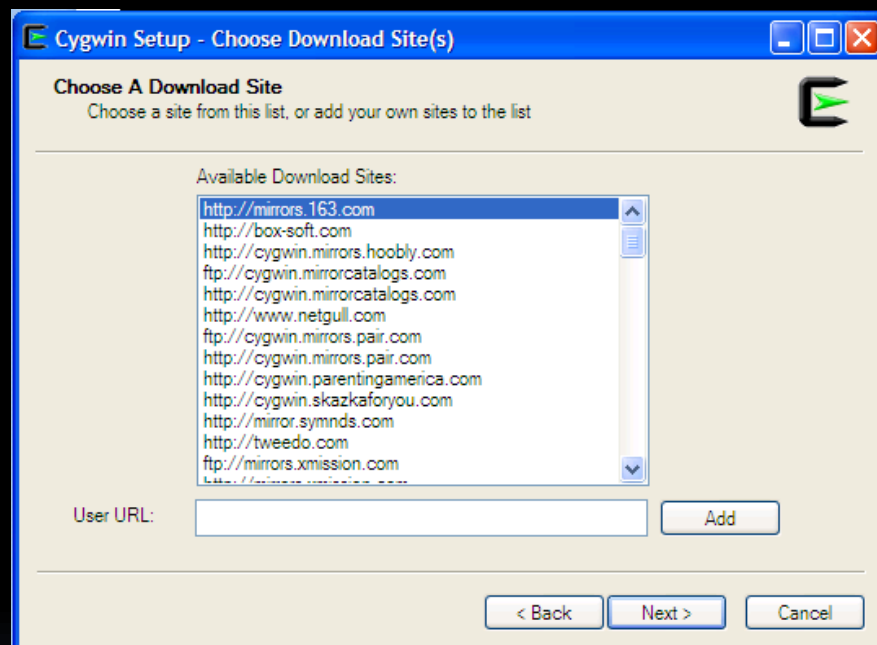
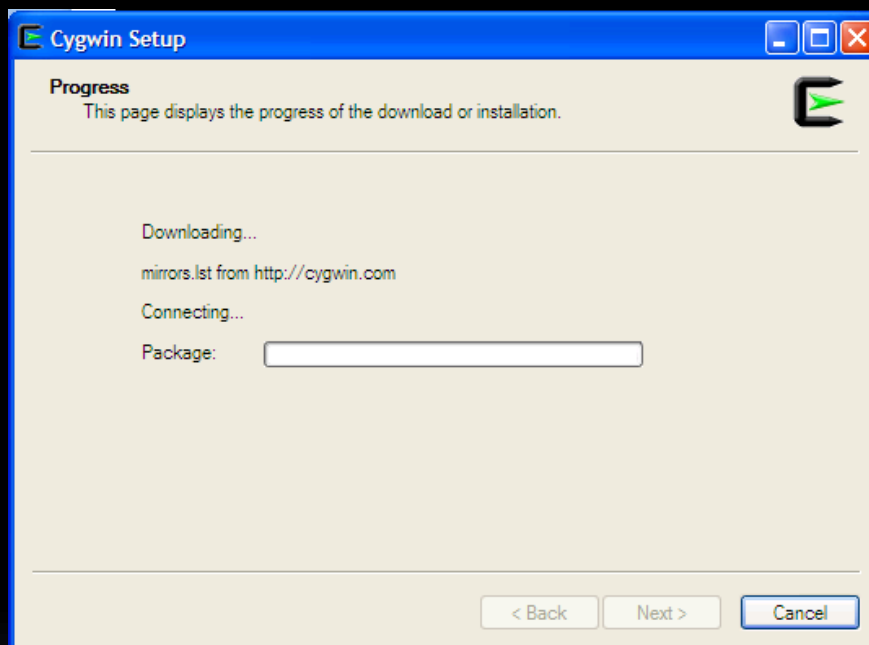
- Truy cập trang chủ <http://cygwin.com> download file setup.exe.
- Tiến hành chạy file setup.exe trên máy tính



CÀI ĐẶT CYGWIN

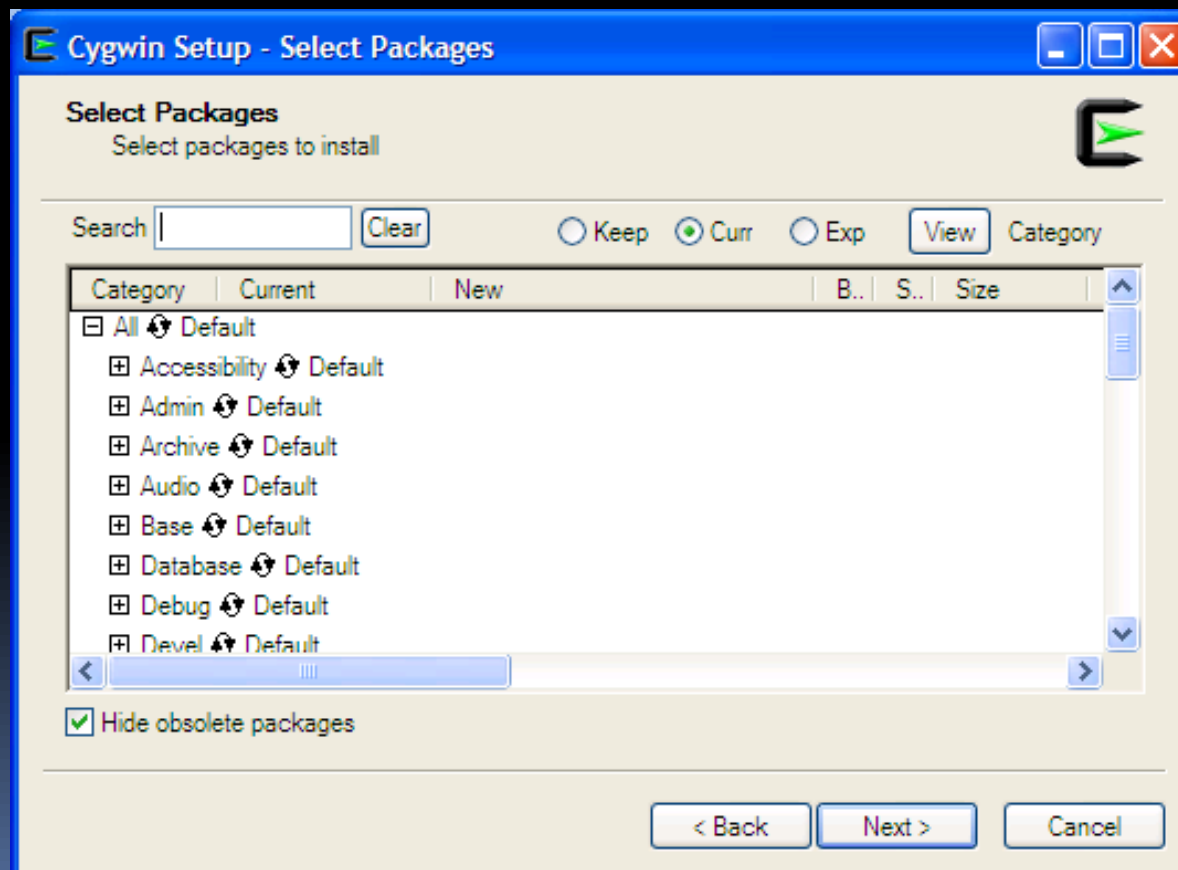


CÀI ĐẶT CYGWIN



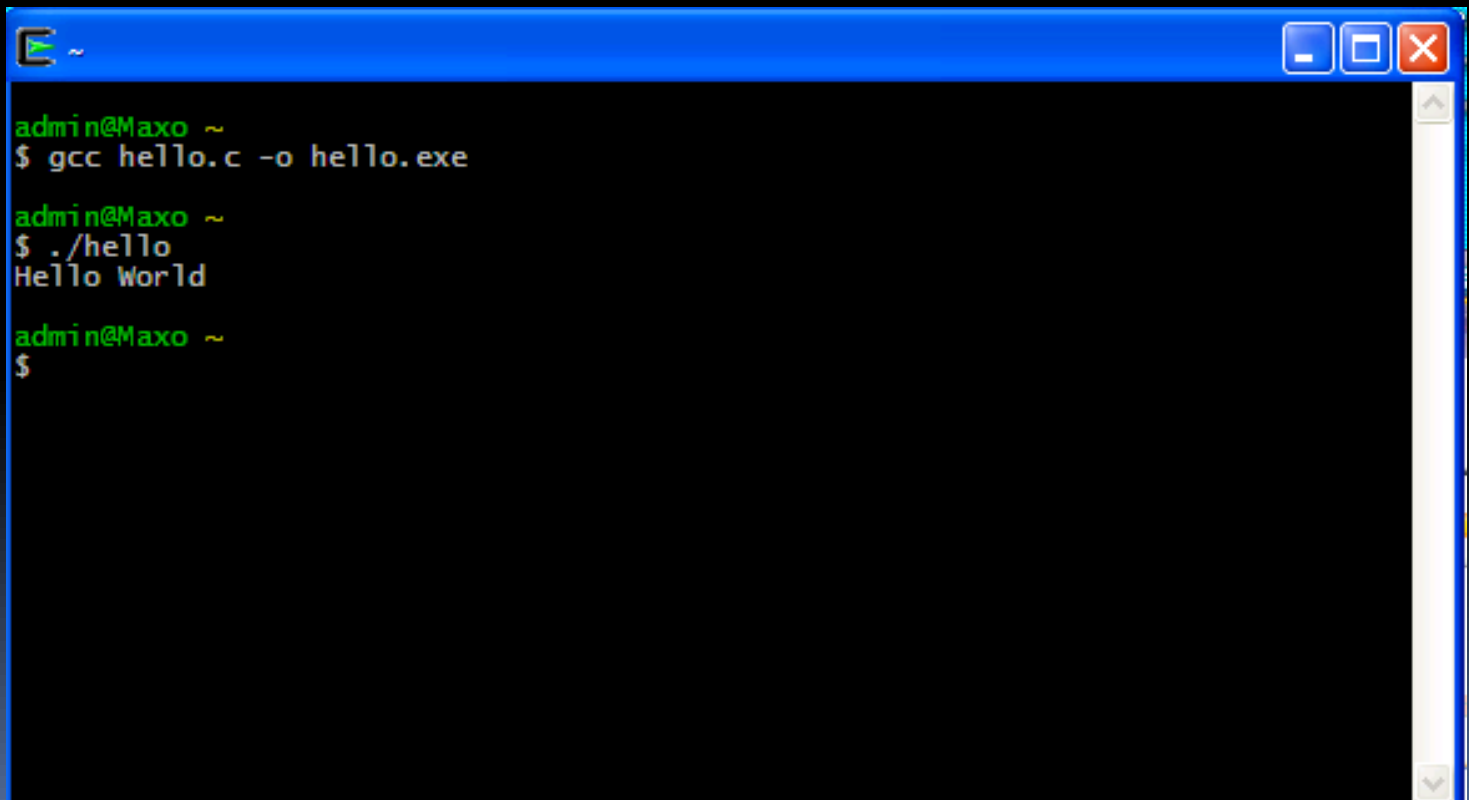
CÀI ĐẶT CYGWIN

- Lựa chọn các gói cần cài đặt: gcc, gdb và make



BUILD & RUN IN CYGWIN

- Build: `gcc hello.c -o hello.exe`
- Run: `./hello`



```
admin@Maxo ~  
$ gcc hello.c -o hello.exe  
  
admin@Maxo ~  
$ ./hello  
Hello World  
  
admin@Maxo ~  
$
```



5.2.OPENMP?

OPENMP ?

- OpenMP (Open Multiprocessing): Giao diện lập trình ứng dụng (API), dùng để điều khiển các luồng (threads) trên cấu trúc chia sẻ bộ nhớ chung.

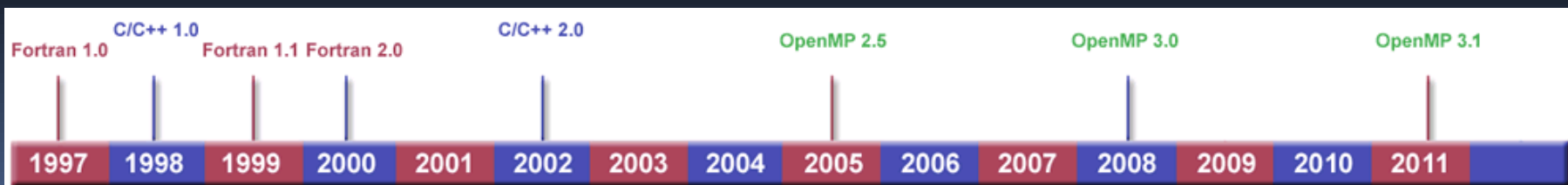
The screenshot displays the OpenMP website. At the top, the OpenMP logo is prominently featured, with the text "THE OPENMP® API SPECIFICATION FOR PARALLEL PROGRAMMING" underneath. A navigation menu on the left includes links for "Subscribe to the News Feed", "OpenMP Specifications", "About the OpenMP ARB", "Compilers", "Resources", "Who's Using OpenMP?", "Press Releases", "Discussion Forums", "Events", "Public OpenMP Calendar", "Input Register", "Search OpenMP.org", and "Archives". The main content area is titled "OpenMP News" and contains several articles. The first article, "IWOMP 2012 Proceedings Available", features a graphic for the "International Workshop on OpenMP" and mentions the proceedings are now available. The second article, "Convey Computer Joins the OpenMP Effort", includes the Convey Computer logo and text about their participation in the OpenMP ARB. A right sidebar provides information about the OpenMP API, links to "OpenMP specs", "OpenMP Compilers", and "Using OpenMP" resources, including a book, examples, forum, Wikipedia, and a tutorial.

OPENMP

- OpenMP bao gồm:
 - Các chỉ thị biên dịch (Compiler Directives)
 - Các thư viện runtime (Runtime Library Routines)
 - Các biến môi trường (Environment Variables)
- Hỗ trợ nhiều phần cứng phần mềm như DEC, Intel, IBM, SGI, Numerical Algorithms Group.
- Sử dụng trên UNIX và Windows NT

OPENMP

- OpenMP hoạt động trên 2 ngôn ngữ Fortran và C/C++.
- Tự động song song hóa chương trình:
 - Người lập trình phải tự ý thức về tính song song của công việc
 - Cung cấp cơ chế chỉ định việc thực hiện song song.
- Công cụ lập trình cho hệ thống có bộ nhớ phân tán



MỤC ĐÍCH CỦA OPENMP

- Cung cấp chuẩn chung cho kiến trúc và nền tảng phần cứng.
- Thiết lập các chỉ thị biên dịch hỗ trợ việc lập trình song song trên máy tính chia sẻ bộ nhớ chung.
- Giúp việc lập trình song song dễ dàng, cung cấp khả năng song song hóa chương trình tuần tự mà không cần dùng đến thư viện thông điệp.

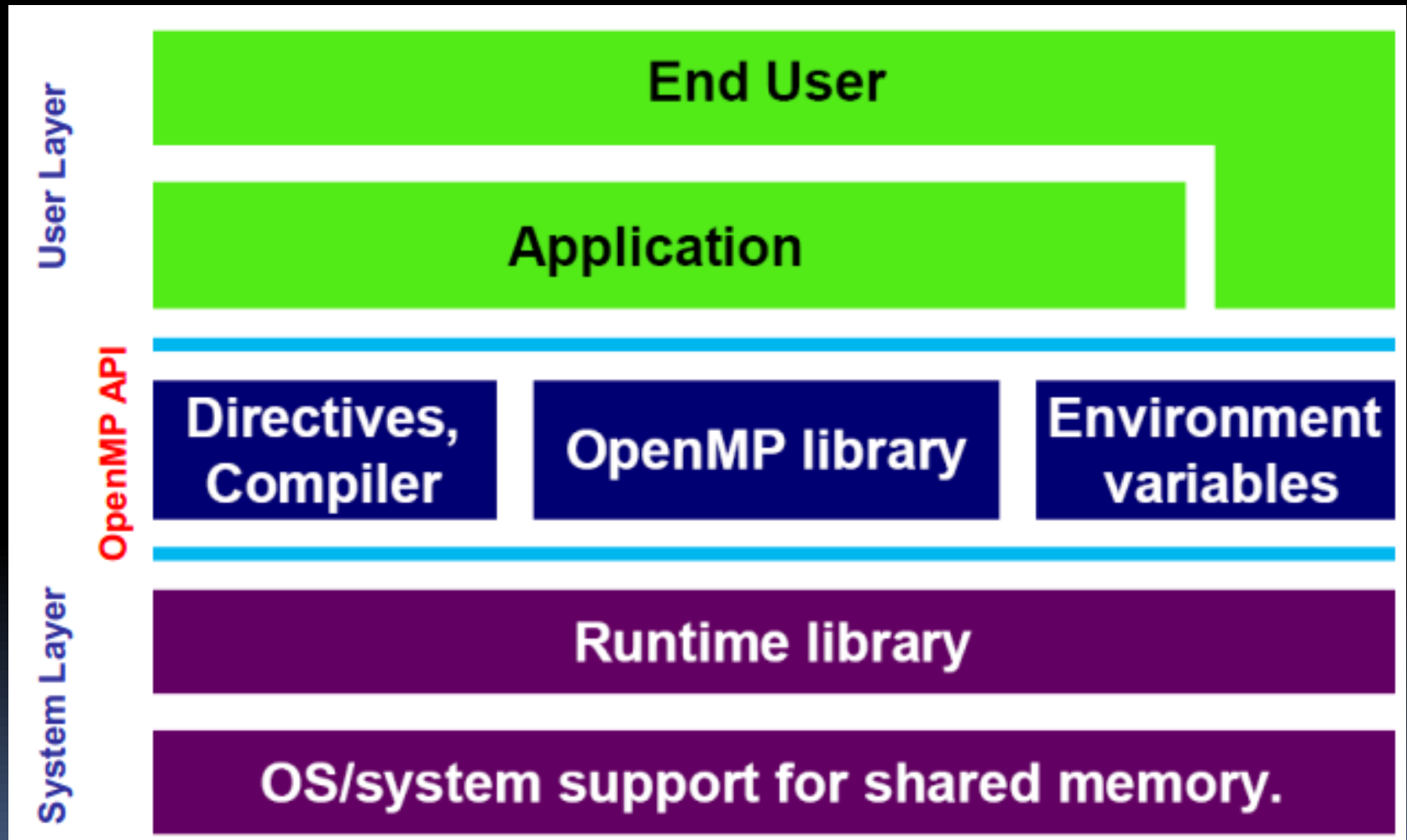
ỨNG DỤNG CỦA OPENMP

- Giải quyết các vấn đề giới hạn bài toán:
 - Dự báo thời tiết
 - Mô phỏng tai nạn xe hơi
 - Bài toán mô phỏng N-Body...

ƯU ĐIỂM OPENMP

- Chuẩn hoàn chỉnh và được công nhận trên thực tế.
- Hiệu suất và khả năng mở rộng tốt
- Tính khả chuyển cao: Chương trình viết ra có thể dịch bằng nhiều trình dịch khác nhau.
- Dễ sử dụng, đơn giản, ít các chỉ thị
- Cho phép song song hóa nhiều chương trình tuần tự

OPENMP



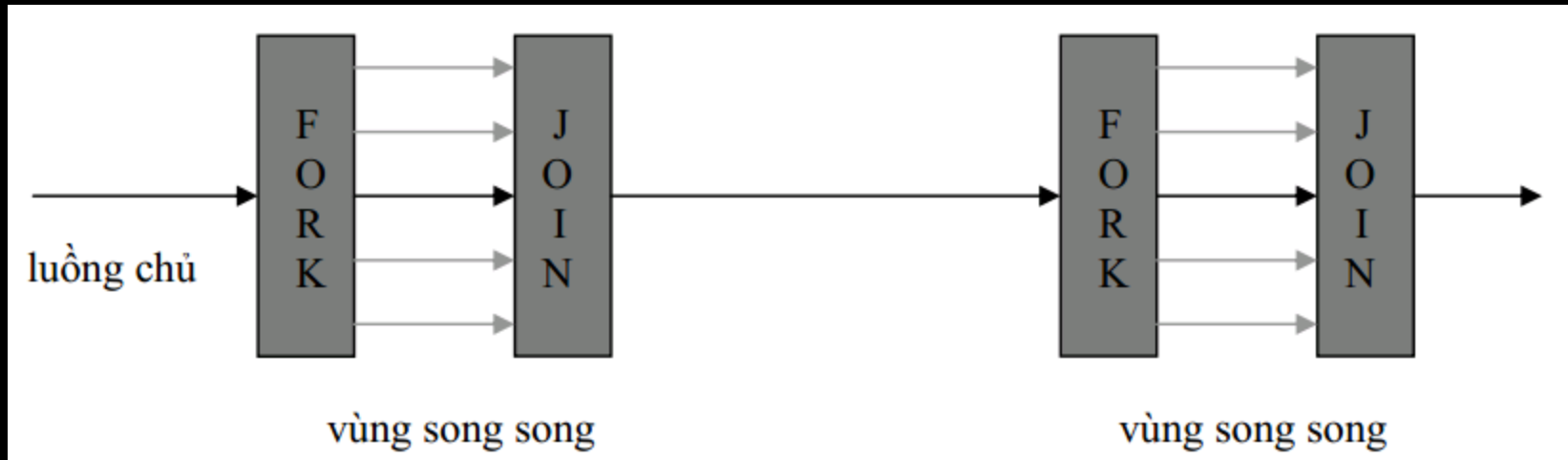
MÔ HÌNH LẬP TRÌNH SONG SONG OPENMP

- Song song hóa dựa trên cơ chế luồng (Thread based parallelism)
 - Xử lý trên bộ nhớ toàn cục
 - Nhiều luồng thực thi đồng thời
- Mô hình song song hiện (Explicit Parallelism)
 - Mô hình lập trình không tự động
 - Điều khiển song song hóa tự động
- Mô hình Fork-Join
 - OpenMP sử dụng mô hình Fork-Join để thực thi việc song song hóa.

MÔ HÌNH LẬP TRÌNH SONG SONG OPENMP

- OpenMP cung cấp mô hình lập trình đa luồng cấp cao, xây dựng thư viện lập trình đa luồng của hệ thống. VD: POSIX Threads.
- Mô hình Fork-Join:
 - Chương trình OpenMP bắt đầu 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 song song.

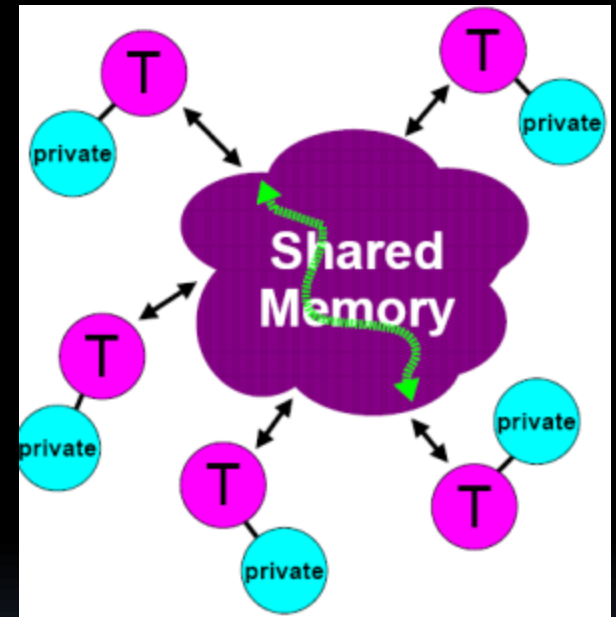
MÔ HÌNH FORK - JOIN



- Fork: Luồng chủ (Master Thread) sẽ tạo ra một tập các luồng song song, đoạn mã trong vùng song song được thực thi bởi luồng song song vừa tạo ra
- Join: Tập luồng song song hoàn thành, đoạn mã trong vùng song song sẽ được đồng bộ và kết thúc và công việc lại được thực hiện bởi luồng chủ.

MÔ HÌNH BỘ NHỚ OPENMP

- Mọi luồng có quyền truy cập đến vùng nhớ chung toàn cục
- Dữ liệu hoặc chia sẻ, hoặc riêng tư
- Cần thiết phải đồng bộ hóa



TÍNH NĂNG CHÍNH CỦA OPENMP

- Tạo nhóm các luồng (threads) thực hiện việc song song.
- Chỉ rõ cách 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 công việc một cách độc quyền.
- Cung cấp hàm thời gian chạy
- Quản lý số lượng luồng.

LẬP TRÌNH VỚI OPENMP

- Chia tách bài toán thành các công việc
 - Trường hợp tốt nhất là khi công việc độc lập nhau
- Gán công việc cho luồng thực thi
- Viết code trên môi trường lập trình song song
- Các yếu tố ảnh hưởng khi lập trình:
 - Nền tảng phần cứng.
 - Cấp độ song song
 - Bản chất bài toán

BIÊN DỊCH OPENMP

- OpenMP biên dịch với các chương trình sau:
 - GNU gcc/g++, gfortran
 - IBM xlc, xls
 - Intel icc, ifort
 - Microsoft Visual C++ 2005 (Professional)
 - Portland C/C++/Fortran, pgcc, pgf95
 - Sun Studio C/C++/Fortran

BIÊN DỊCH CHƯƠNG TRÌNH OPENMP

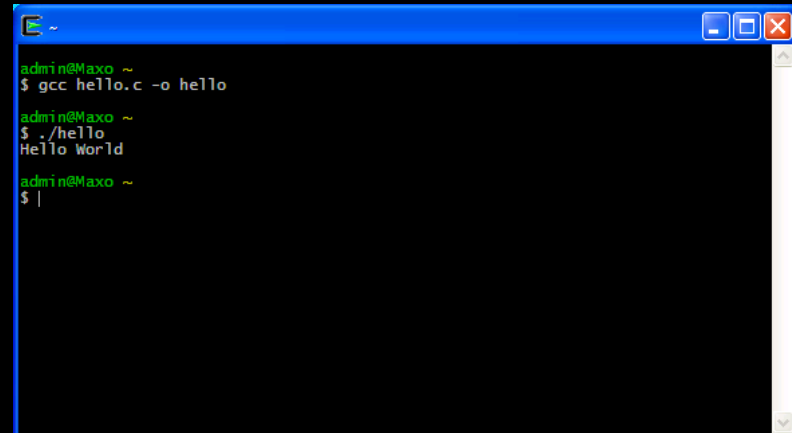
- Sử dụng GNU GCC
 - OpenMP 3.1 support GCC 4.7
 - `#include <omp.h>`
 - `gcc -fopenmp`
- Microsoft Visual C++
 - Version VC++ 2005, 2008, 2010, OpenMP 2.0
 - Win 32 Console Project -> Empty Project
 - Project Properties -> Configuration properties -> C, C++/language
 - Activate OpenMP options
 - Build Project
 - Run “without debug”

LẬP TRÌNH VỚI OPENMP

- Song song theo dữ liệu:
 - 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:
 - Gán công việc cụ thể cho luồng thông qua chỉ số của luồng.
 - `#pragma omp parallel sections`

HELLO WORLD VỚI OPENMP (GCC)

```
1  #include <stdio.h>
2  void main()
3  {
4      printf("Hello World \n");
5  }
```

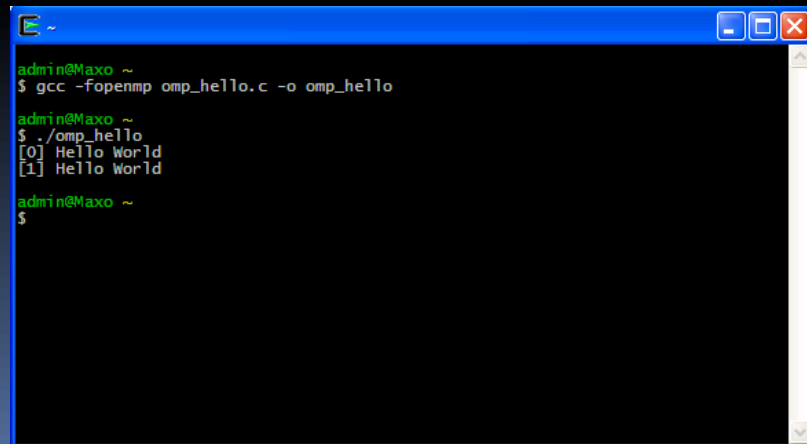


```
admin@Maxo ~
$ gcc hello.c -o hello

admin@Maxo ~
$ ./hello
Hello World

admin@Maxo ~
$ |
```

```
1  #include <omp.h>
2  #include <stdio.h>
3  void main()
4  {
5      #pragma omp parallel
6      printf("[%d] Hello World \n",omp_get_thread_num());
7  }
```

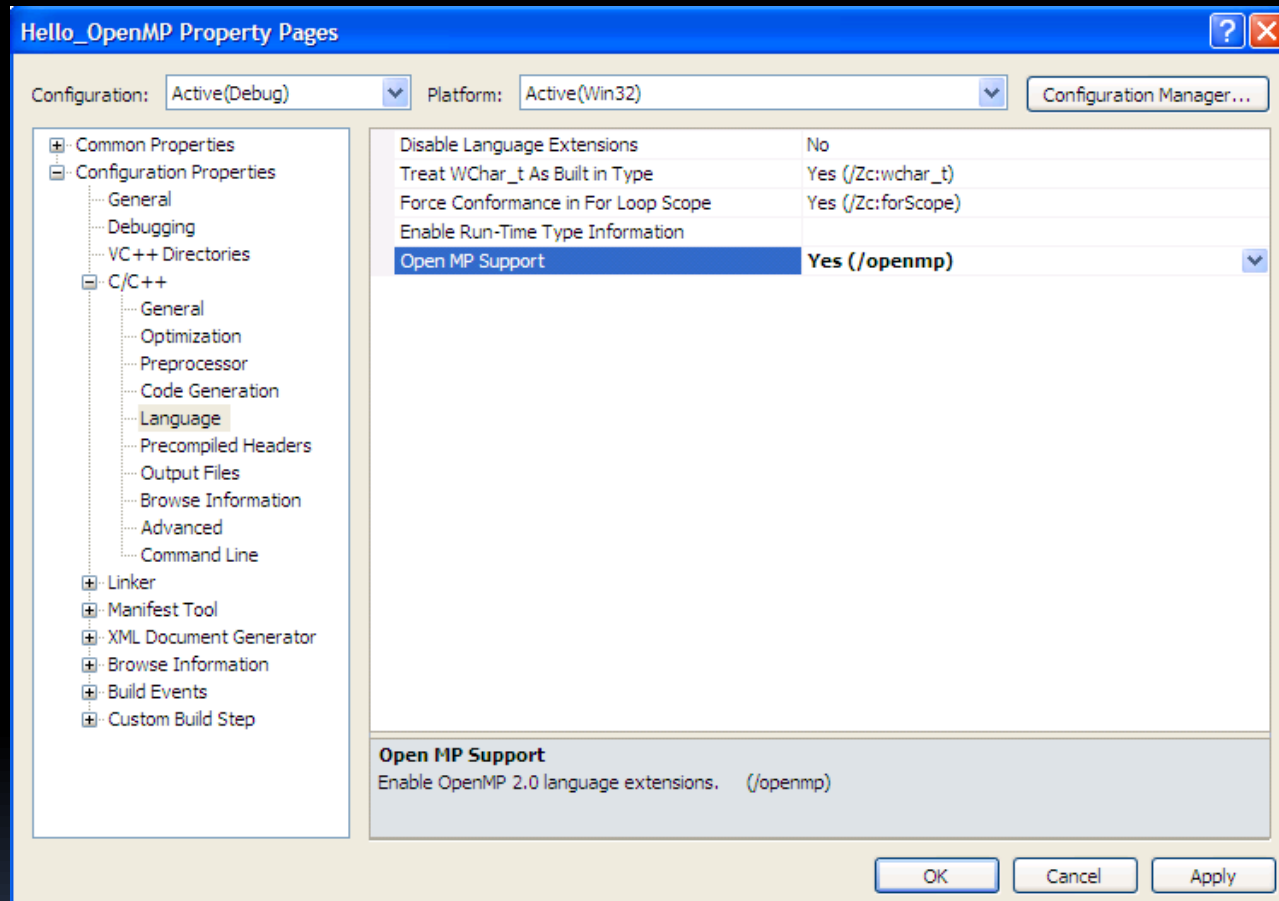
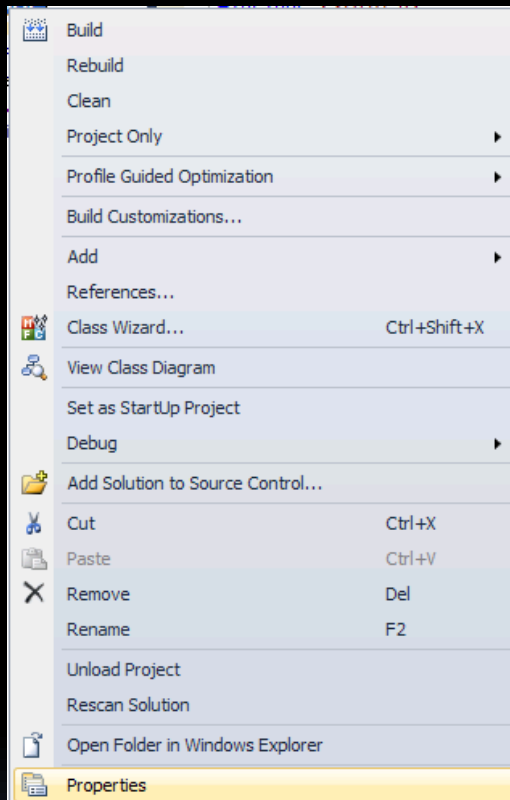


```
admin@Maxo ~
$ gcc -fopenmp omp_hello.c -o omp_hello

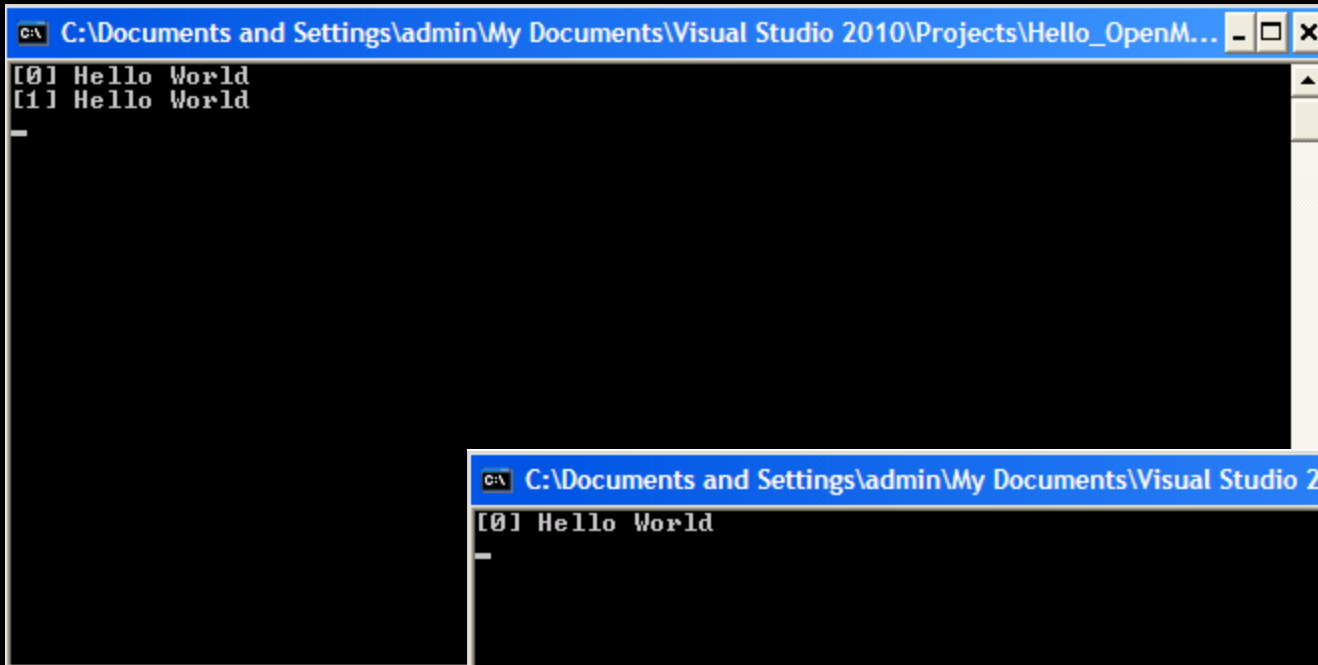
admin@Maxo ~
$ ./omp_hello
[0] Hello World
[1] Hello World

admin@Maxo ~
$
```

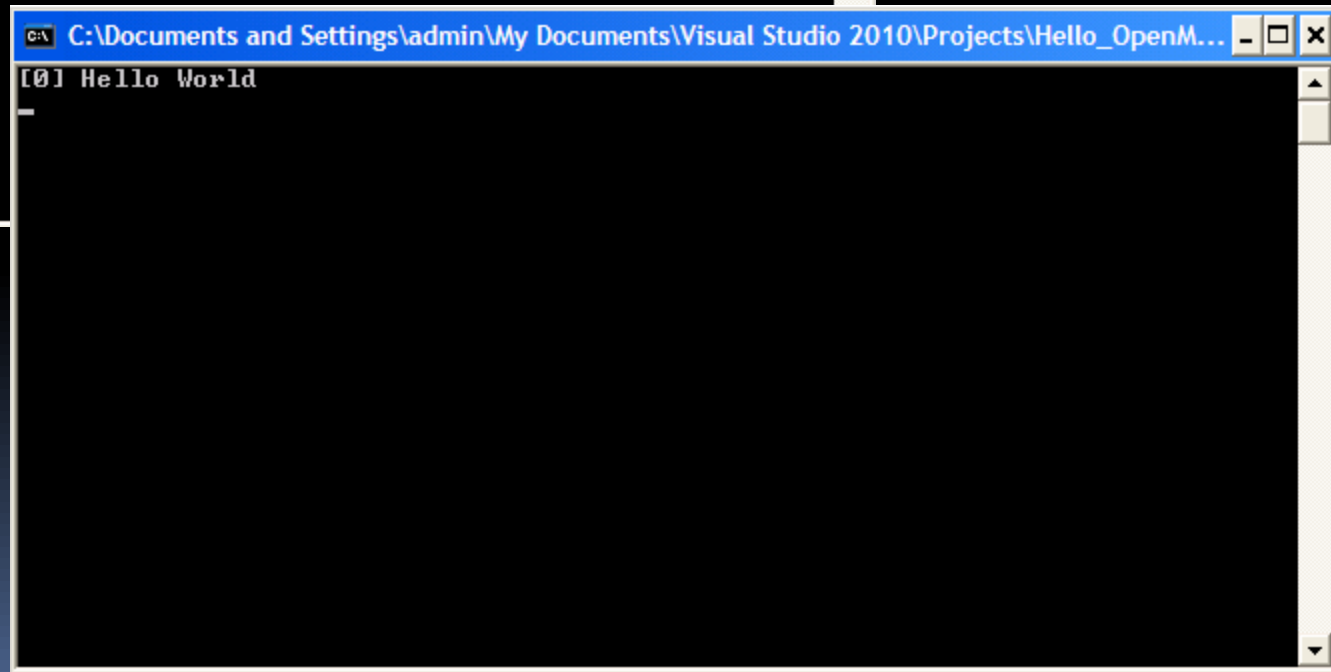
HELLO WORLD VỚI OPENMP VISUAL STUDIO



HELLO WORLD VỚI OPENMP VISUAL STUDIO



```
C:\Documents and Settings\admin\My Documents\Visual Studio 2010\Projects\Hello_OpenM...  
[0] Hello World  
[1] Hello World
```



```
C:\Documents and Settings\admin\My Documents\Visual Studio 2010\Projects\Hello_OpenM...  
[0] Hello World
```

Các chỉ thị trong OpenMP

- OpenMP sử dụng chỉ thị chương trình dịch để điều khiển sự song song
- Cấu trúc chỉ thị song song:
 - ▣ **#pragma omp directive-name [clause...] newline**
 - #pragma omp: Yêu cầu bắt buộc đối với mọi chỉ thị OpenMP C/C++
 - directive-name: Tên chỉ thị xuất hiện sau #pragma omp và đứng trước bất kì mệnh đề nào đó.
 - [clause]: Mệnh đề không bắt buộc.
 - newline: Yêu cầu bắt buộc đối với mỗi chỉ thị.

Phạm vi của chỉ thị

- Static Extent:
 - ▣ Đoạn mã nguyên bản trong phạm vi từ đầu đến cuối khối cấu trúc cho sau mỗi chỉ thị. Phạm vi tĩnh không mở rộng đến các thủ tục và các tệp chứa mã
- Orphaned Directive:
 - ▣ Xuất hiện độc lập với các chỉ thị khác, tồn tại ở ngoài phạm vi tĩnh của chỉ thị khác, cho phép mở rộng với các thủ tục và các tệp mã nguồn.
- Dynamic Extent:
 - ▣ Bao gồm phạm vi 2 chỉ thị trên

Chỉ thị kết hợp parallel for

- Sử dụng trước câu lệnh for
- Tạo vùng thực hiện song song và chỉ định vòng lặp được phân phối cho các luồng.
- Mệnh đề tiếp theo xác định tính chất:
 - Shared: Chia sẻ
 - Private: Riêng tư
- default (none): Không sử dụng tính chất dựng sẵn của OpenMP cho tính chất của dữ liệu

Khai báo tính chất dữ liệu

- Shared(list): Các biến dùng chung cho các luồng.
- Private(list): Các biến dùng riêng cho mỗi luồng, mỗi luồng.
- VD: Dữ liệu chia sẻ cho các luồng thì nên dùng Shared .



5.3.CÁC THÀNH PHẦN OPENMP

Thuật ngữ

- Thread Teams: Master + workers
- Parallel region: Vùng song song:
 - Khối mã được thực hiện bởi các luồng
 - {}
- Work-sharing construct: Khối chia sẻ công việc.

Thành phần của OpenMP

Directives

- ◆ *Parallel regions*
- ◆ *Work sharing*
- ◆ *Synchronization*
- ◆ *Data scope attributes*
 - *private*
 - *firstprivate*
 - *lastprivate*
 - *shared*
 - *reduction*
- ◆ *Orphaning*

Environment variables

- ◆ *Number of threads*
- ◆ *Scheduling type*
- ◆ *Dynamic thread adjustment*
- ◆ *Nested parallelism*

Runtime environment

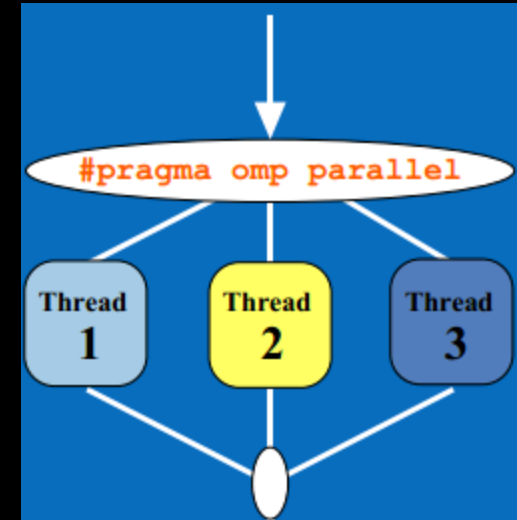
- ◆ *Number of threads*
- ◆ *Thread ID*
- ◆ *Dynamic thread adjustment*
- ◆ *Nested parallelism*
- ◆ *Timers*
- ◆ *API for locking*

Khối mã của OpenMP

- Chỉ thị thực thi OpenMP kèm theo đoạn mã gồm: câu lệnh, vòng lặp, { }
- Khối mã OpenMP:
 - Vùng song song – parallel region
 - Khối chia sẻ công việc – work-sharing construct
 - Khối đồng bộ
 - Biến môi trường
 - Hàm chỉ định thời gian chạy

VÙNG SONG SONG TRONG OPENMP

- Định nghĩa vùng song song (parallel region) trên cùng đoạn mã cấu trúc.
- Các luồng được tạo ra song song.
- Luồng được kết thúc tại đoạn cuối của vùng song song.



C/C++ :

```
#pragma omp parallel
{
    block
}
```


Đặc điểm

- Chỉ thị `#pragma omp parallel`
- Đoạn mã của khối được thực hiện bởi tất cả các luồng.
- Kết thúc khối barrier – điểm đồng bộ
- `Nowait <> barrier`
- Luồng gặp khối song song sẽ trở thành luồng chủ
- Luồng chủ tạo nhóm các luồng thực hiện công việc
- Cuối khối luồng chủ đợi luồng trong nhóm kết thúc mới thoát ra ngoài
- Bên ngoài khối duy nhất luồng chủ làm việc tuần tự

KHAI BÁO VÙNG PARALLEL

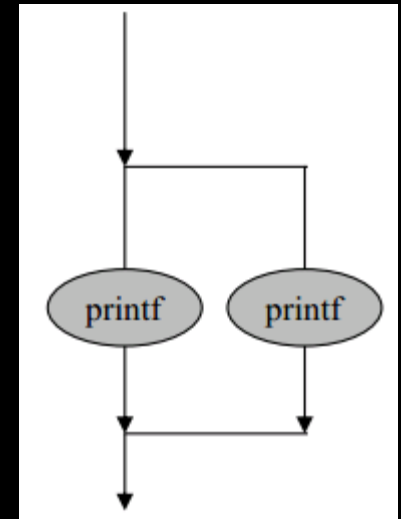
- `#pragma omp parallel [clause[clause...]]`
 - `{`
 - `/*block executed by all thread*/`
 - `}`

```
#pragma omp parallel firstprivate (a)
{
```

```
1  #include <omp.h>
2  #include <stdio.h>
3  void main()
4  {
5      #pragma omp parallel
6      printf("[%d] Hello World \n",omp_get_thread_num());
7  }
```

Ví dụ thực hiện khối song song

```
1  #include <omp.h>
2  #include <stdio.h>
3  void main()
4  {
5      #pragma omp parallel
6      printf("[%d] Hello World \n",omp_get_thread_num());
7  }
```



Các mệnh đề sử dụng khối song song

- `if (scalar-expression)`
- `num_threads(integer-expression)`
- `private(list)`
- `firstprivate(list)`
- `shared(list)`
- `default(none|shared)`
- `copyin(list)`
- `reduction(operator:list)`

Khối chia sẻ công việc

- Chi thị:

- #pragma omp for
- #pragma omp sections
- #pragma omp single

```
#pragma omp for
{
  ....
}
```

```
#pragma omp sections
{
  ....
}
```

```
#pragma omp single
{
  ...
}
```

- Công việc trong khối phân chia cho các luồng
- **Đặt trong khối song song**
- Tiếp cận với tất cả các luồng hoặc không luồng nào, điểm đồng bộ đặt cuối khối.
- **Không tạo thêm luồng mới**

Khối chia sẻ for

- Cú pháp:
 - `#pragma omp for [clause[]....]`
`<for-loop>`
- Đặc điểm
 - Nội dung của khối là câu lệnh lặp for.
 - Song song dữ liệu: mỗi đoạn của lệnh for được thực hiện bởi một luồng.
 - Khối chia sẻ chuẩn và được dùng nhiều
 - Luồng được phân chia thành các tập hợp độc lập
 - Đôi khi toàn bộ công việc được kết thúc

Khối chia sẻ for

- Chú ý:

- Số bước lặp trong lệnh for phải đếm được
- Biến chạy phải là biến riêng của luồng
- Biến kích thước vòng lặp là biến chia sẻ

- Mệnh đề:

- private(list)
- firstprivate(list)
- lastprivate(list)
- reduction(operator: list)
- ordered
- schedule
- nowait

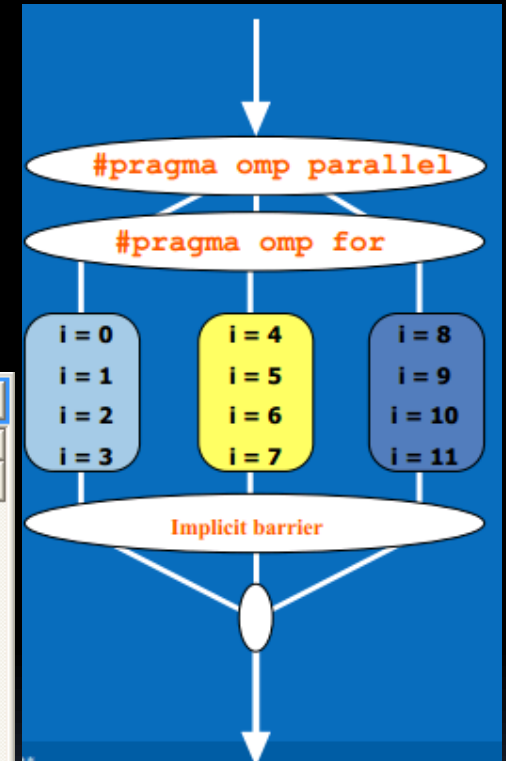
```
#pragma omp parallel
#pragma omp for
    for (i=0; i<N; i++){
        Do_Work(i);
    }
```

Ví dụ về khối chia sẻ for

```
void main()
{
    int i;
    omp_set_num_threads(3);
    #pragma omp parallel
    #pragma omp for
    for (i=0;i<12;i++)
        printf("Hello world, No Thread = %d\n",omp_get_thread_num());
    getch();
}
```

C:\Documents and Settings\admin\My Documents\Visual Studio 2010\Projects\Hello_OpenM... - □ ×

```
Hello world, No Thread = 0
Hello world, No Thread = 1
Hello world, No Thread = 0
Hello world, No Thread = 1
Hello world, No Thread = 0
Hello world, No Thread = 1
Hello world, No Thread = 1
Hello world, No Thread = 2
Hello world, No Thread = 2
Hello world, No Thread = 2
Hello world, No Thread = 2
```



Khối chia sẻ section

- Chỉ thị: `#pragma omp sections`
- Nội dung: gồm các khối con, bắt đầu bởi chỉ thị `#pragma omp section`
- Đặc điểm:
 - Song song công việc: mỗi khối section con được thực hiện bởi 1 luồng
- Chú ý:
 - Số section lớn hơn số luồng: Tồn tại luồng thực hiện nhiều section
 - Số luồng lớn hơn số section: Có luồng được nghỉ

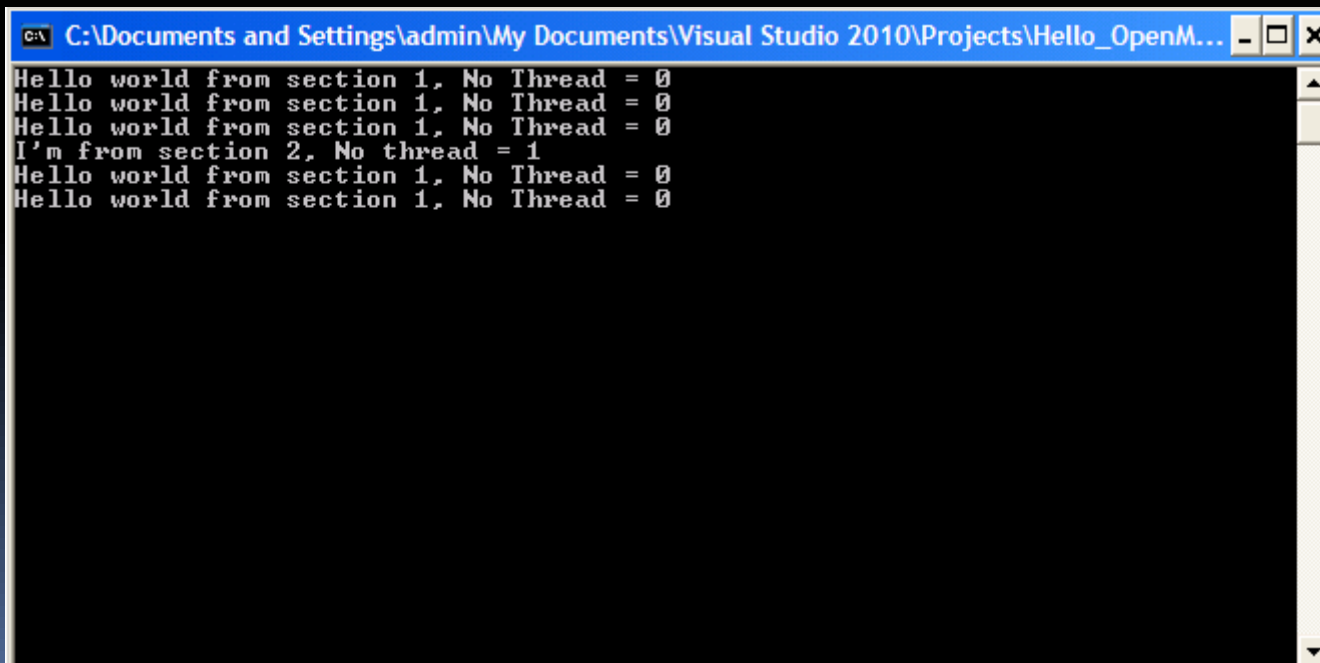
Cú pháp & mệnh đề

```
#pragma omp sections [clause(s)]  
{  
    #pragma omp section  
        <code 1>  
    #pragma omp section  
        <code 2>  
    ....  
}
```

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

Ví dụ section

```
omp_set_num_threads(3);
#pragma omp parallel
#pragma omp sections
{
    #pragma omp section
        for (i=0;i<5;i++)
            printf("Hello world from section 1, No Thread = %d\n",omp_get_thread_num());
    #pragma omp section
        printf("I'm from section 2, No thread = %d\n", omp_get_thread_num());
}
```



```
C:\Documents and Settings\admin\My Documents\Visual Studio 2010\Projects\Hello_OpenM...
Hello world from section 1, No Thread = 0
Hello world from section 1, No Thread = 0
Hello world from section 1, No Thread = 0
I'm from section 2, No thread = 1
Hello world from section 1, No Thread = 0
Hello world from section 1, No Thread = 0
```

Cú pháp rút gọn

```
#pragma omp parallel
#pragma omp for
    for()
```



```
#pragma omp parallel for
    for()
```

```
#pragma omp parallel
{
    #pragma omp sections
    {
        [#pragma omp section]
        section 1
        [#pragma omp section]
        section 2
        ....
    }
}
```



```
#pragma omp parallel sections
{
    [#pragma omp section]
    section 1
    [#pragma omp section]
    section 2
    ....
}
```

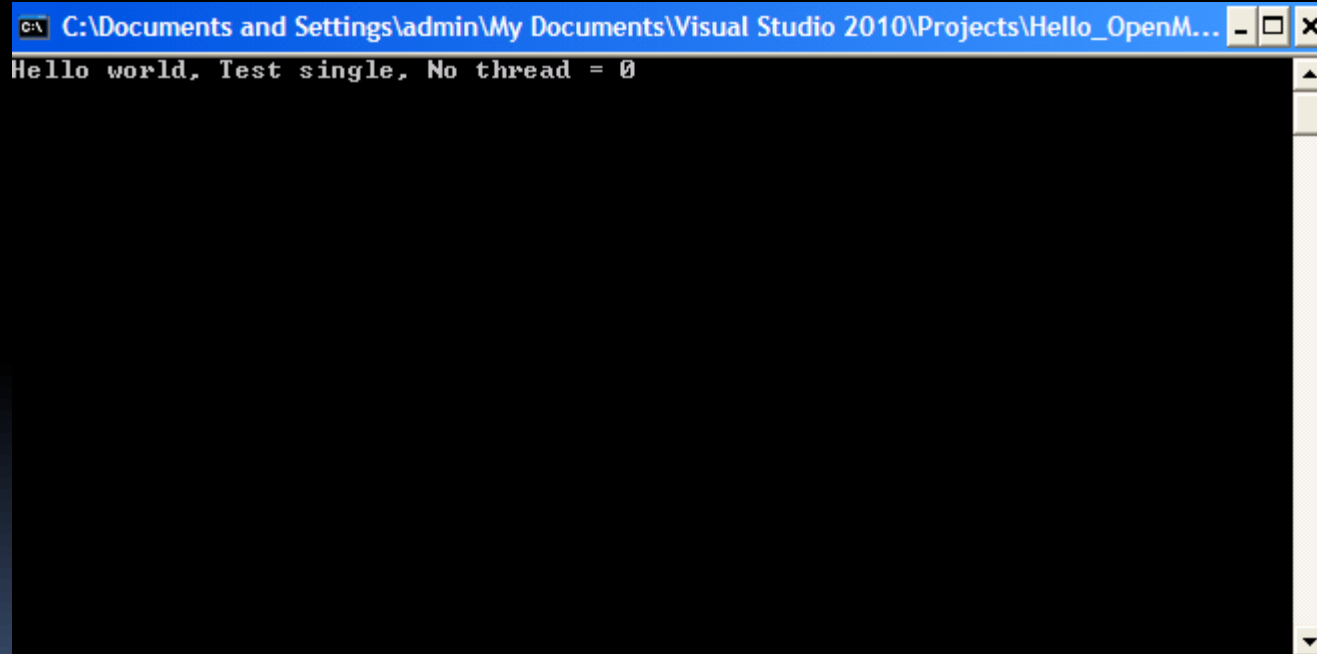
Khối chia sẻ single

- Chỉ thị: `#pragma omp single`
- Nội dung là đoạn mã trong khối `{ }`
- Chỉ được thực hiện bởi 1 luồng
- Luồng thực hiện được chọn ngẫu nhiên
- Kết thúc khối có điểm đồng bộ ngầm định

```
#pragma omp single [clause...]  
{  
    code  
}
```

Ví dụ về single

```
omp_set_num_threads(3);  
#pragma omp parallel  
#pragma omp single  
    printf("Hello world, Test single, No thread = %d",omp_get_thread_num());
```



The screenshot shows a Windows command prompt window with a blue title bar. The title bar text is "C:\Documents and Settings\admin\My Documents\Visual Studio 2010\Projects\Hello_OpenM...". The window content is black, and the text "Hello world, Test single, No thread = 0" is displayed in white. The window has standard Windows controls (minimize, maximize, close) in the top right corner.

```
C:\Documents and Settings\admin\My Documents\Visual Studio 2010\Projects\Hello_OpenM...  
Hello world, Test single, No thread = 0
```

Một số mệnh đề điều khiển

- shared
- private
- firstprivate
- lastprivate
- default
- num_threads
- nowait
- reduction
- schedule

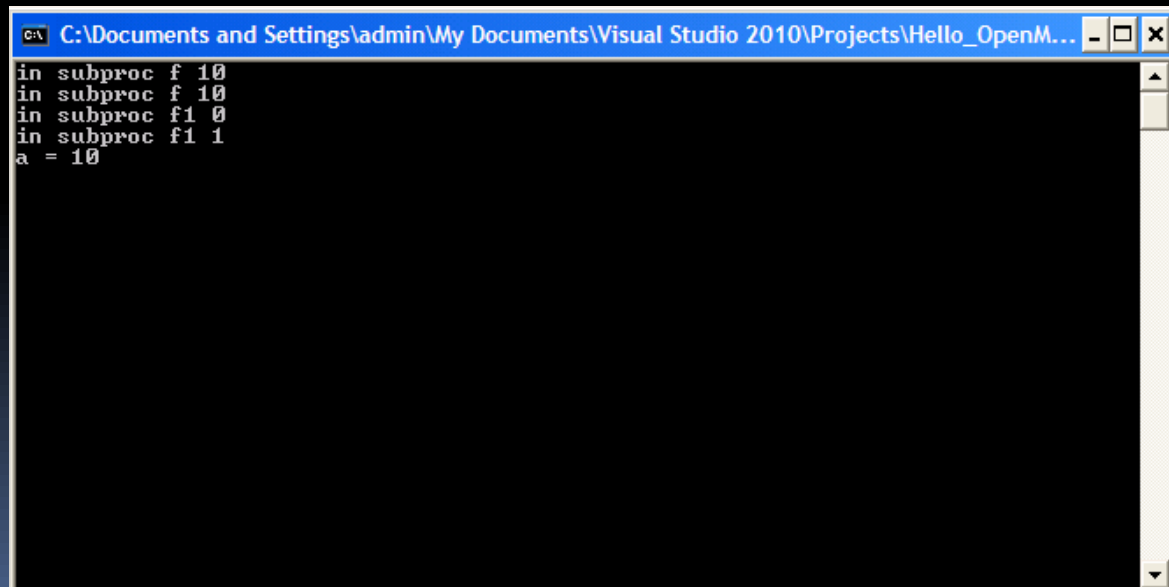


Phạm vi truy cập của biến
đã khai báo

Private(list)

- ❑ Chỉ định các biến là cục bộ của luồng
- ❑ Giá trị không xác định tại điểm vào và ra luồng
- ❑ Biến đã khai báo gọi là đối tượng tham chiếu của biến cục bộ

```
static int a = 10;
void f()
{
    printf("in subproc f %d \n",a);
}
void f1(int x)
{
    printf ("in subproc f1 %d \n",x);
}
void main()
{
    #pragma omp parallel private (a)
    {
        a = omp_get_thread_num();
        f();
        f1(a);
    }
    printf ("a = %d",a);
    getch();
}
```



```
C:\Documents and Settings\admin\My Documents\Visual Studio 2010\Projects\Hello_OpenM...
in subproc f 10
in subproc f 10
in subproc f1 0
in subproc f1 1
a = 10
```


Shared(list)

- Chỉ định biến là toàn cục, có thể được truy cập tại mọi luồng
- Mọi luồng đều truy cập đến cùng địa chỉ trên không gian nhớ.

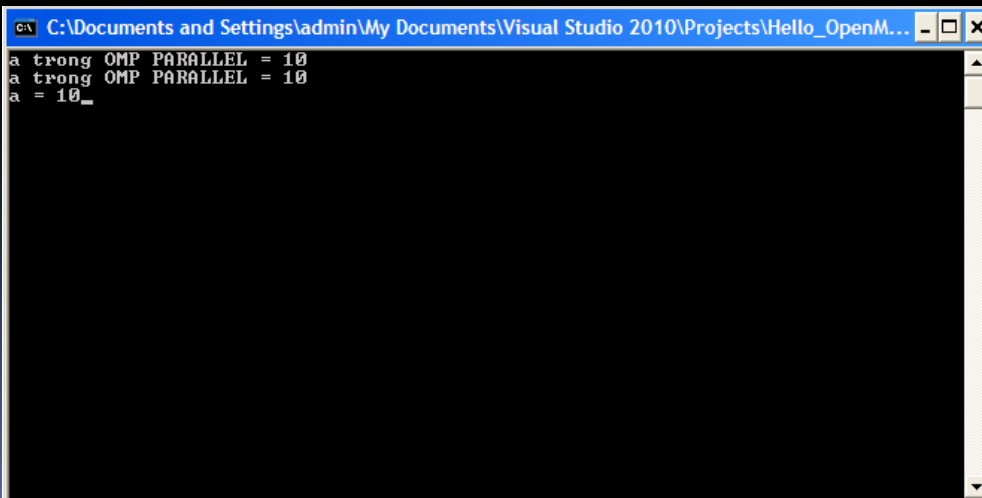
```
#pragma omp parallel shared (n,a,b) private (i)
{
    #pragma omp for
    for (i = 0; i < n; i++)
        a[i] += b[i];
}
```

Firstprivate(list)

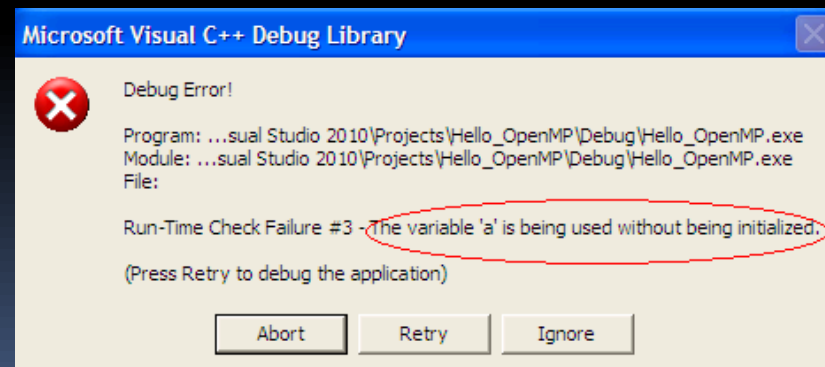
- Giống như private nhưng các biến sẽ nhận được giá trị khởi tạo ban đầu trước khi vào từng threads.

```
static int a = 10;
void main()
{
#pragma omp parallel firstprivate (a)
{
    printf("a trong OMP PARALLEL = %d\n",a);
    a = omp_get_thread_num();
}
printf ("a = %d",a);
```

```
static int a = 10;
void main()
{
#pragma omp parallel private (a)
{
    printf("a trong OMP PARALLEL = %d\n",a);
    a = omp_get_thread_num();
}
printf ("a = %d",a);
```



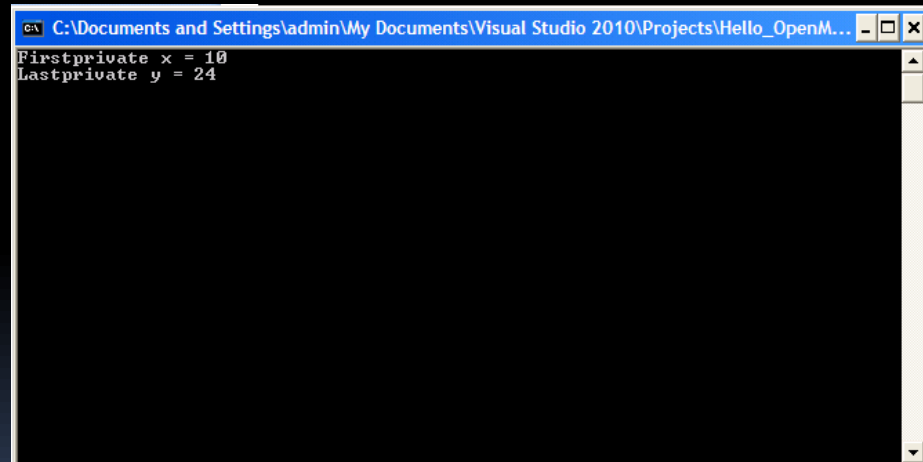
```
C:\Documents and Settings\admin\My Documents\Visual Studio 2010\Projects\Hello_OpenM...
a trong OMP PARALLEL = 10
a trong OMP PARALLEL = 10
a = 10_
```



Lastprivate(list)

- Giá trị của biến cục bộ tại đoạn lặp cuối for hoặc tại khối section cuối được gán cho đối tượng tham chiếu.

```
int i;  
int x = 10;  
int y = 100;  
#pragma omp parallel  
{  
#pragma omp for private (i) firstprivate(x) lastprivate(y)  
for (i = 0; i < 10; i++)  
{  
    x += 1;  
    y = x + i;  
}  
}  
printf ("Firstprivate x = %d\n",x);  
printf ("Lastprivate y = %d\n",y);
```



```
C:\Documents and Settings\admin\My Documents\Visual Studio 2010\Projects\Hello_OpenM...  
Firstprivate x = 10  
Lastprivate y = 24
```

Default & nowait

- Default(none|shared)
 - none: Bỏ thiết lập ngầm định đối với phạm vi truy xuất của biến -> người lập trình phải tự xác định
 - shared: Các biến đều là biến chia sẻ giữa các luồng, trừ khi sử dụng mệnh đề private.
- nowait
 - Bỏ các điểm đồng bộ (barrier) ngầm định

```
#pragma omp for nowait  
{  
      
}
```

How many threads?

- Chỉ thị `parallel` tạo ra một tập các luồng và luồng ban đầu (luồng chủ) có chỉ thị là 0 (default).
- `OMP_GET_NUM_THREADS()`: Số lượng luồng mặc định
- `OMP_GET_THREAD_NUM()`: Số thứ tự mỗi luồng.

VÍ DỤ

```
#pragma omp parallel private(nthreads, tid)
{
    tid = omp_get_thread_num();
    if (tid == 0)
    {
        nthreads = omp_get_num_threads();
        printf("Number of threads = %d\n", nthreads);
    }
}
getch();
```

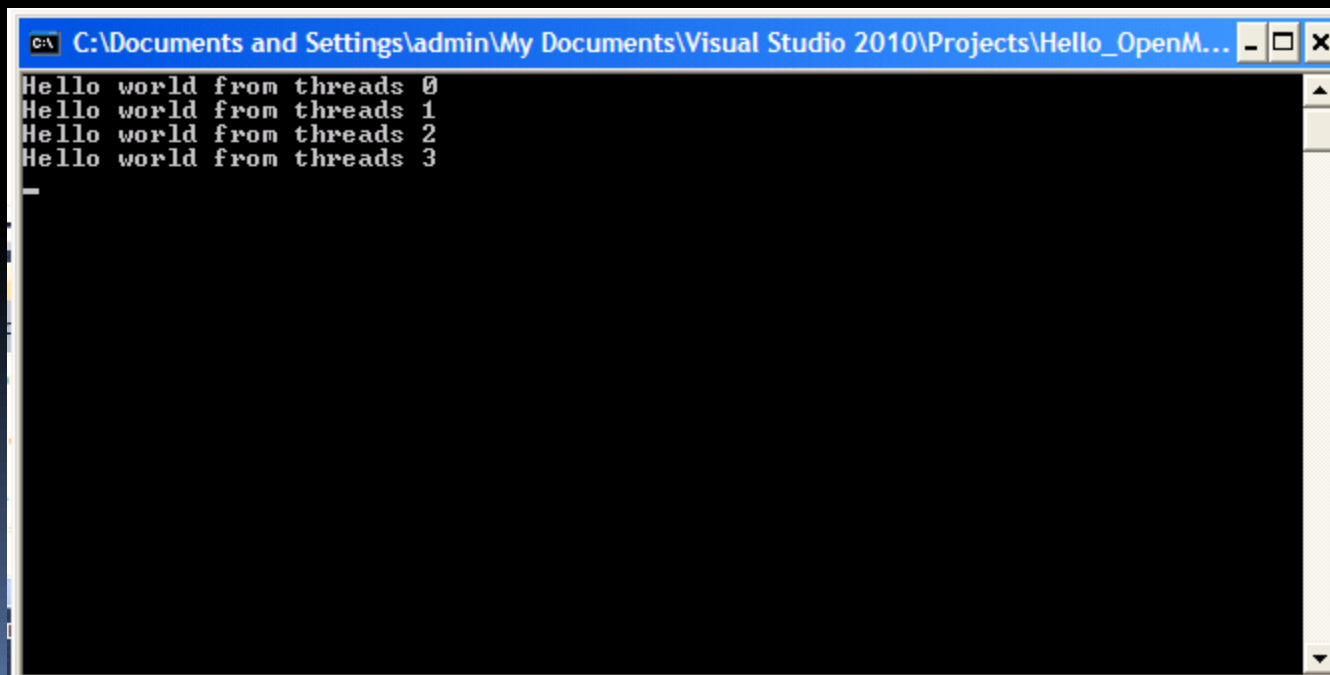
C:\Documents and Settings\admin\My Documents\Visual Studio 2010\Projects\He

Number of threads = 2

Thiết lập số Threads

- `omp_set_num_threads();`

```
omp_set_num_threads(4);  
#pragma omp parallel  
printf("Hello world from threads %d \n",omp_get_thread_num());
```



The screenshot shows a Windows command prompt window with the title bar "C:\Documents and Settings\admin\My Documents\Visual Studio 2010\Projects\Hello_OpenM...". The window contains the following output:

```
Hello world from threads 0  
Hello world from threads 1  
Hello world from threads 2  
Hello world from threads 3  
-
```

num_threads()

- Chỉ định số luồng cho khối song song

```
void main()
{
    int ThreadID;
    int Num_Thread;
    #pragma omp parallel num_threads(4)
    {
        ThreadID = omp_get_thread_num();
        Num_Thread = omp_get_num_threads();
        printf ("Thread %d in %d threads\n", ThreadID, Num_Thread);
    }
    getch();
}
```

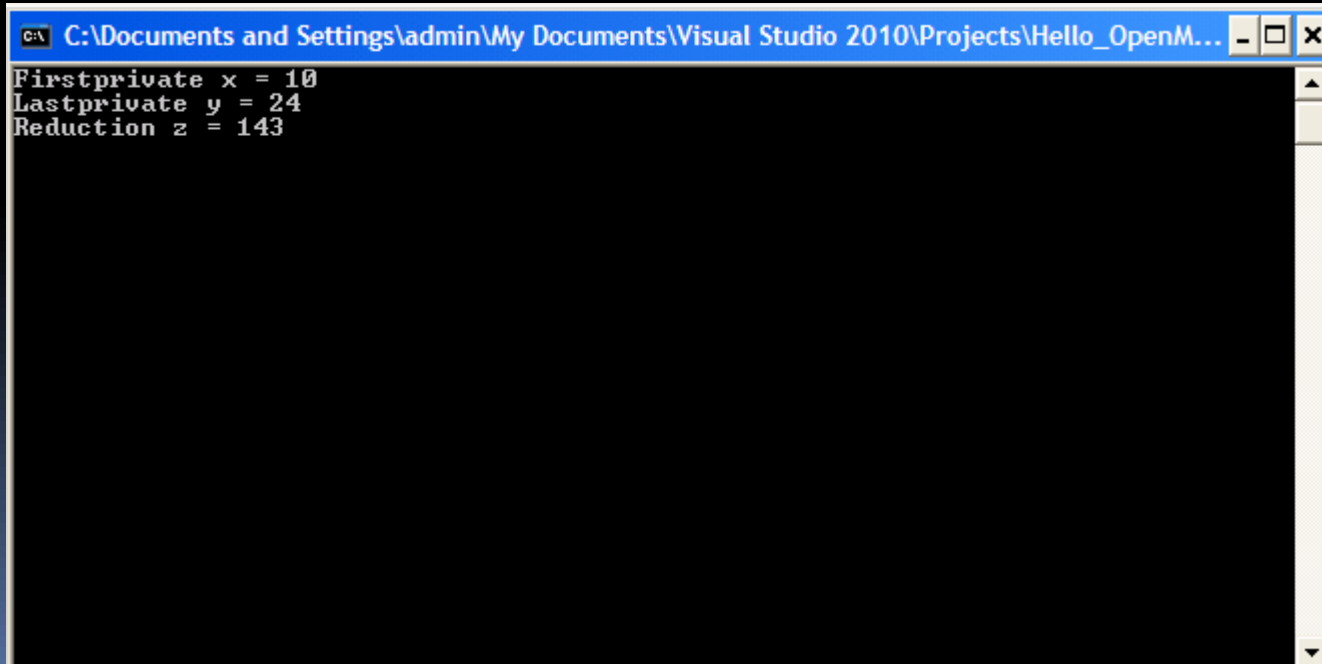
```
C:\ C:\Documents and Settings\admin\My Documents\Visual Studio 2010\Projects\Hello_OpenM.
Thread 2 in 4 threads
Thread 0 in 4 threads
Thread 1 in 4 threads
Thread 3 in 4 threads
```


reduction()

- Cú pháp: `reduction(op:list)`
- Các biến trong “list” được chia sẻ cho các vùng song song.
- Bên trong vùng song song hoặc khối chia sẻ:
 - Một bản copy của mỗi biến trong list được tạo ra và phụ thuộc vào toán tử op
 - Mỗi bản copy được cập nhật bởi các luồng
 - Cuối cùng tổng hợp các bản copy lại với nhau thông qua toán tử op.

Ví dụ reduction

```
int i;  
int x = 10;  
int y = 100;  
int z = 100;  
#pragma omp parallel for private(i) firstprivate(x) lastprivate(y) reduction(+:z)  
for (i=0;i<10;i++)  
{  
    x += 1;  
    y = x + i;  
    z = x + i;  
}  
  
printf ("Firstprivate x = %d\n",x);  
printf ("Lastprivate y = %d\n",y);  
printf ("Reduction z = %d\n",z);
```



The screenshot shows a Windows command prompt window with the title bar "C:\Documents and Settings\admin\My Documents\Visual Studio 2010\Projects\Hello_OpenM...". The window contains the following output:

```
Firstprivate x = 10  
Lastprivate y = 24  
Reduction z = 143
```

C/C++ Reduction Operations

Operator	Initial Value
+	0
*	1
-	0
^	0
&	~0
	0
&&	1
	0

$x = x \text{ op } \text{expr}$
$x \text{ binop} = \text{expr}$
$x = \text{expr op } x$
$x++$
$++x$
$x--$
$--x$

- x là biến vô hướng trong danh sách biến
- expr là biểu thức vô hướng không tham chiếu đến x
- op là một trong các phép toán $+, *, -, /, \&, ^, |, \&\&, ||$
- binop là một trong các phép toán $+, *, -, /, \&, ^, |$

schedule

schedule (static | dynamic | guided [, chunk])
schedule (runtime)

- chunk: kích thước mỗi đoạn lặp
- static[,chunk]
 - Chia cố định các đoạn lặp trong luồng cho “chunk”
 - Trong trường hợp không có “chunk” mỗi luồng thực hiện chia đều N/P

TID	0	1	2	3
no chunk	1-4	5-8	9-12	13-16
chunk = 2	1-2 9-10	3-4 11-12	5-6 13-14	7-8 15-16

schedule

```
schedule ( static | dynamic | guided [, chunk] )  
schedule (runtime)
```

- **dynamic**: hàng đợi chung chứa các đoạn lặp, khi luồng kết thúc một đoạn lặp nó lấy đoạn lặp tiếp theo từ hàng đợi
- **guided**: như dynamic, kích thước các đoạn lặp giảm theo hàm mũ, mỗi đoạn lặp có kích thước không nhỏ hơn chunk
- **runtime**: cách phân chia được xác định tại thời điểm chạy thông qua biến OMP_SCHEDULE

Khởi tạo sự đồng bộ

- barrier
- critical
- atomic
- master
- ordered
- flush

barrier

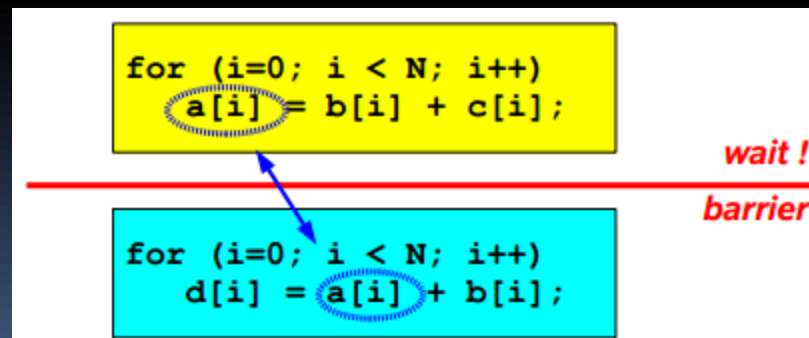
```
#pragma omp barrier
```

- Xét ví dụ sau:

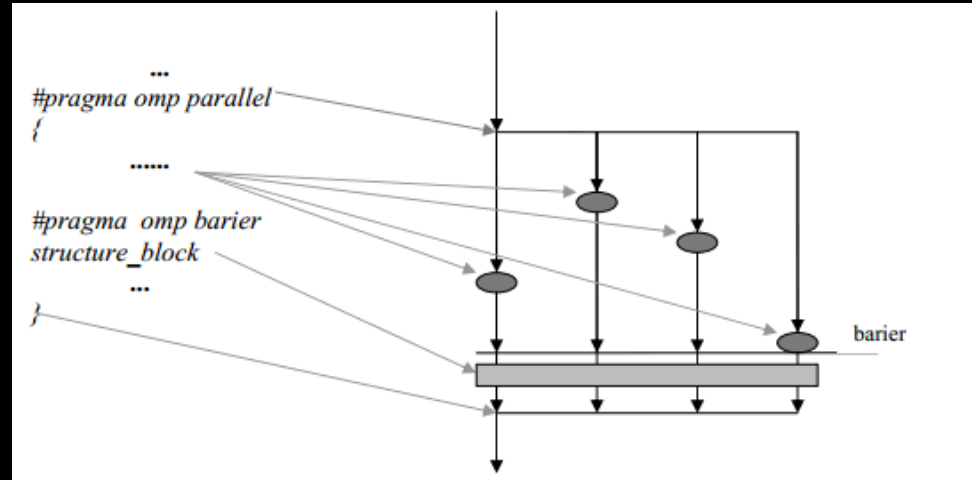
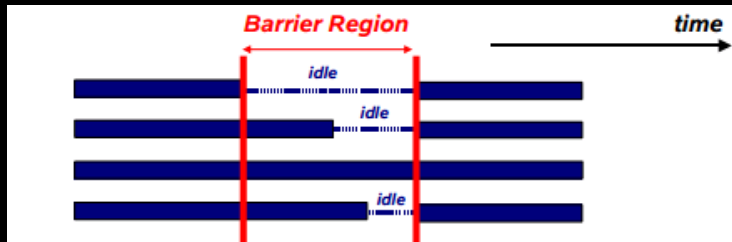
```
for (i=0; i < N; i++)  
    a[i] = b[i] + c[i];
```

```
for (i=0; i < N; i++)  
    d[i] = a[i] + b[i];
```

- Lỗi xảy ra, tính giá trị mảng a[i] trước khi sử dụng a[i]



barrier



- Tạo điểm đồng bộ trong chương trình
- Không luồng nào được phép vượt qua barrier trước khi mọi luồng tiếp cận barrier
- Buộc các luồng phải đợi nhau
- Barrier ngầm định đặt cuối các khối song song

Ví dụ barrier

```
int main ()
{
    int id;
    #pragma omp parallel private (id) num_threads(6)
    {
        id = omp_get_thread_num();
        if ((id % 2) == 0)
        {
            Sleep(3000);
            printf("Even numbers: %d \n",id);
        }

        //#pragma omp barrier
        if ((id % 2) != 0)
            printf("Odd numbers: %d \n", id);
    }
    return 0;
}
```

C:\WINDOWS\system32\cmd.exe

```
Even numbers: 2
Even numbers: 4
Even numbers: 0
Odd numbers: 3
Odd numbers: 5
Odd numbers: 1
Press any key to continue . . . _
```

critical

```
#pragma omp critical [(name)]  
{<code-block>}
```

- Hạn chế nhiều luồng đồng thời cập nhật dữ liệu chia sẻ
- Hỗ trợ tính toán song song thành tuần tự
- Name: tên của từng khối critical

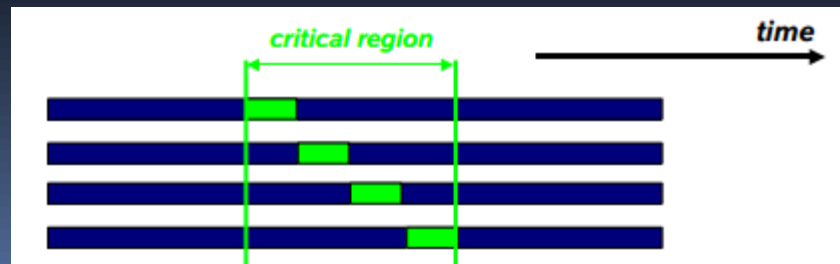
```
for (i=0; i < N; i++){  
    .....  
    sum += a[i];  
    .....  
}
```



```
for (i=0; i < N; i++){  
    .....  
    sum += a[i];  
    .....  
}
```

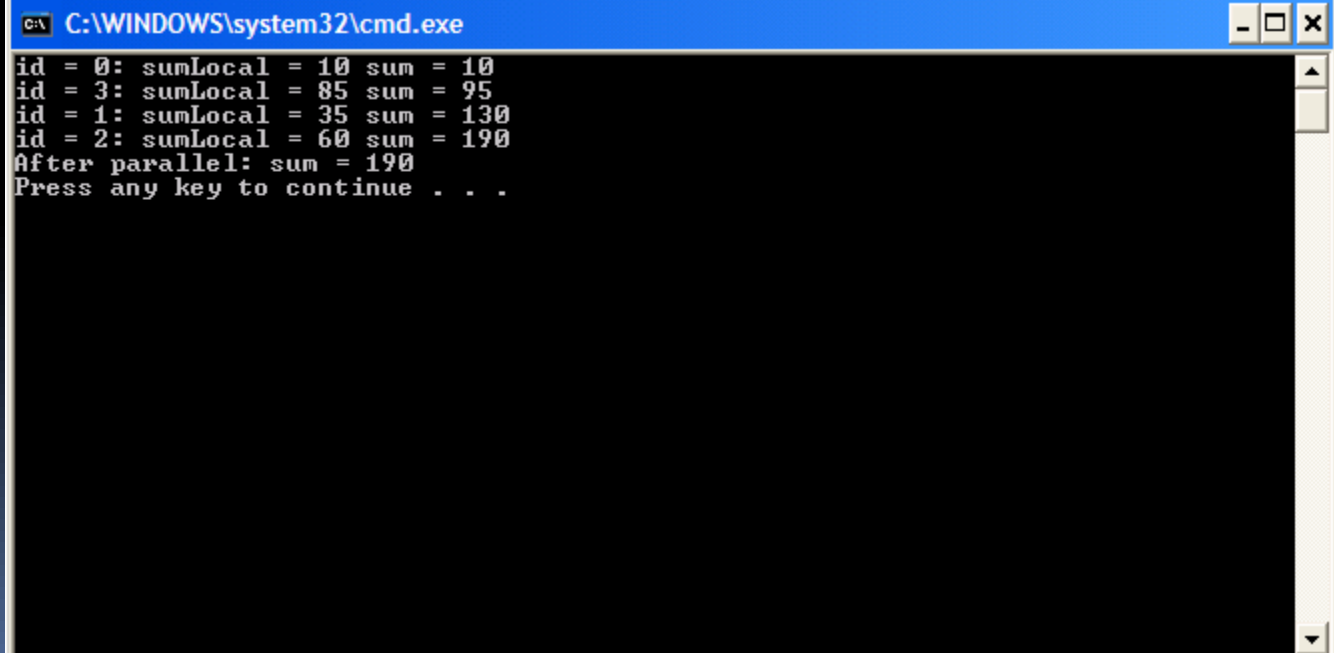
one at a time can proceed

next in line, please



Ví dụ sử dụng critical

```
#pragma omp parallel num_threads(4) shared (A, sum) private (id, sumLocal)
{
    id = omp_get_thread_num();
    sumLocal = 0;
    #pragma omp for
    for (i = 0; i < 20; i++)
        sumLocal += A[i];
    #pragma omp critical (calculate_sum)
    {
        sum += sumLocal;
        printf("id = %d: sumLocal = %d sum = %d\n", id, sumLocal, sum);
    }
}
printf ("After parallel: sum = %d \n",sum);
```



```
C:\WINDOWS\system32\cmd.exe
id = 0: sumLocal = 10 sum = 10
id = 3: sumLocal = 85 sum = 95
id = 1: sumLocal = 35 sum = 130
id = 2: sumLocal = 60 sum = 190
After parallel: sum = 190
Press any key to continue . . .
```

atomic

```
#pragma omp atomic  
<statement>
```

- Một kiểu của critical, đảm bảo vùng bộ nhớ chỉ định được cập nhật một cách tự động:
 - Khối hoạt động chỉ là câu lệnh đơn (statement)
 - Câu lệnh thuộc dạng sau: $x \text{ binop} = \text{expr} \mid x++ \mid x-- \mid --x$
- Trong đó:
 - x là kiểu vô hướng
 - expr : biểu thức không tham chiếu đến x và trả về giá trị vô hướng
 - binop : $+, -, *, /, \&, ^, <<, >$

Ví dụ sử dụng atomic

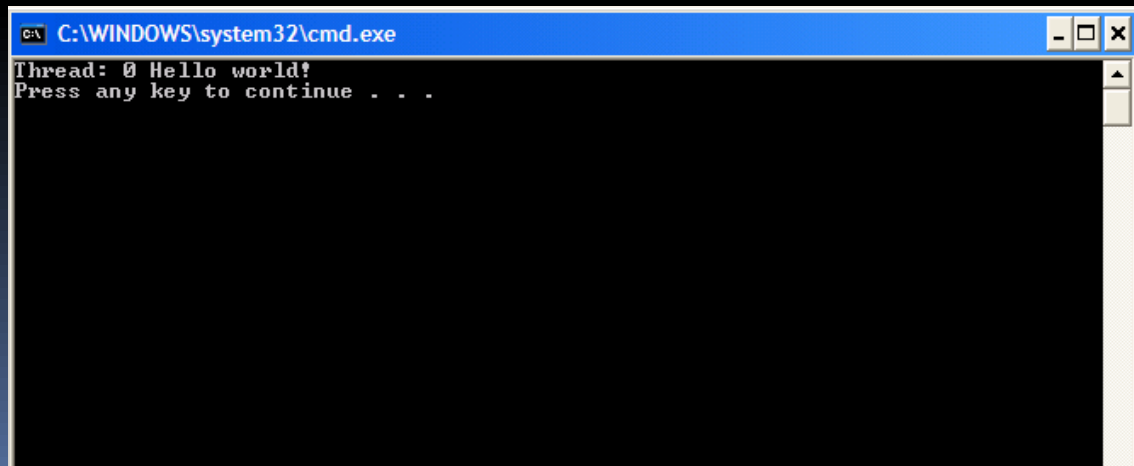
```
int main() {  
    int count = 0;  
    #pragma omp parallel num_threads(MAX)  
    {  
        #pragma omp atomic  
        count++;  
    }  
    printf("Number of threads: %d\n", count);  
    return 0;  
}
```

master

```
#pragma omp master  
{<code-block>}
```

- Khối mã chỉ được thực hiện bởi luồng chủ
- Không được sử dụng barrier ở cuối khối

```
int main()  
{  
    #pragma parallel omp num_threads(4)  
    {  
        #pragma omp master  
        printf ("Thread: %d Hello world!\n", omp_get_thread_num());  
    }  
    return 0;  
}
```



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The output of the program is displayed as follows:

```
Thread: 0 Hello world!  
Press any key to continue . . .
```

ordered

```
#pragma omp ordered  
{ <code-block> }
```

- Đoạn mã trong khối được thực hiện theo thứ tự.
- Xuất hiện trong chỉ thị for hoặc parallel for có kèm theo ordered

Ví dụ ordered

```
int main()
{
    int i;
    #pragma omp parallel for ordered private (i)
    for (i = 0; i < 10; i++)
    {
        #pragma omp ordered
        printf("i = %d in Thread: %d Hello world!\n", i, omp_get_thread_num());
    }
    return 0;
}
```

C:\WINDOWS\system32\cmd.exe

```
i = 0 in Thread: 0 Hello world!
i = 1 in Thread: 0 Hello world!
i = 2 in Thread: 0 Hello world!
i = 3 in Thread: 0 Hello world!
i = 4 in Thread: 0 Hello world!
i = 5 in Thread: 1 Hello world!
i = 6 in Thread: 1 Hello world!
i = 7 in Thread: 1 Hello world!
i = 8 in Thread: 1 Hello world!
i = 9 in Thread: 1 Hello world!
Press any key to continue . . .
```

```
int main()
{
    int i;
    #pragma omp parallel for private (i)
    for (i = 0; i < 10; i++)
    {
        printf("i = %d in Thread: %d Hello world!\n", i, omp_get_thread_num());
    }
    return 0;
}
```

C:\WINDOWS\system32\cmd.exe

```
i = 0 in Thread: 0 Hello world!
i = 5 in Thread: 1 Hello world!
i = 1 in Thread: 0 Hello world!
i = 6 in Thread: 1 Hello world!
i = 2 in Thread: 0 Hello world!
i = 7 in Thread: 1 Hello world!
i = 3 in Thread: 0 Hello world!
i = 8 in Thread: 1 Hello world!
i = 4 in Thread: 0 Hello world!
i = 9 in Thread: 1 Hello world!
Press any key to continue . . .
```


orphaned directive – chỉ thị mồ côi

- OpenMP chuẩn không giới hạn các chỉ thị chia sẻ và chỉ thị đồng bộ (omp for, omp single, critical, barrier, ...) trong phạm vi của vùng song song -> orphaned.
- Có thể xuất hiện ngoài vùng song song.

```
(void) dowork(); !- Sequential FOR  
  
#pragma omp parallel  
{  
    (void) dowork(); !- Parallel FOR  
}
```

```
void dowork()  
{  
    #pragma for  
    for (i=0;....)  
    {  
        :  
    }  
}
```

Biến môi trường

- Điều khiển chương trình OpenMP khi chạy
- Tương tác thông qua biến điều khiển nội tại của chương trình OpenMP:
 - Quản lý bởi trình dịch OpenMP
 - Không thể truy cập hay thay đổi trực tiếp
 - Bị thay đổi qua hàm OpenMP hoặc biến môi trường.

Một số biến môi trường

- OMP_NUM_THREADS n
- OMP_SCHEDULE “schedule,[chunk]”
- OMP_DYNAMIC {TRUE | FALSE}
- OMP_NESTED {TRUE | FALSE}
- OMP_STACKSIZE
- OMP_WAIT_POLICY
- OMP_MAX_ACTIVE_LEVELS
- OMP_THREAD_LIMIT

Hàm OpenMP

- OpenMP cung cấp các hàm:
 - ▣ Điều khiển, 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.
- C/C++: `#include <omp.h>`
- Nên sử dụng macro `#ifdef _OPENMP`

Hàm OpenMP

Name	Functionality
omp_set_num_threads	Set number of threads
omp_get_num_threads	Return number of threads in team
omp_get_max_threads	Return maximum number of threads
omp_get_thread_num	Get thread ID
omp_get_num_procs	Return maximum number of processors
omp_in_parallel	Check whether in parallel region
omp_set_dynamic	Active dynamic thread adjustment
omp_get_dynamic	Check for dynamic thread adjustment
omp_set_nested	Active nested parallelism
omp_get_nested	Check for nested parallelism
omp_get_wtime	Returns wall clock time
omp_get_wtick	Number of seconds between clock ticks

OpenMP 3.0

- Hỗ trợ song song hóa cho những bài toán lặp không chính quy:
 - Lặp với số bước không xác định
 - Đệ quy
 - Producer/consumer
- `#pragma omp task [clause list]`
 - Thực hiện bởi luồng
 - Gắn chặt với luồng

Duyệt danh sách

- Nhiệm vụ được tạo ra khi một luồng gặp chỉ thị task
- Nhiệm vụ này được gán cho một luồng nào đó
- Duyệt song song danh sách liên kết

```
Element first, e;  
#pragma omp parallel  
#pragma omp single  
{  
    for (e = first; e; e = e->next)  
#pragma omp task firstprivate(e)  
    process(e);  
}
```

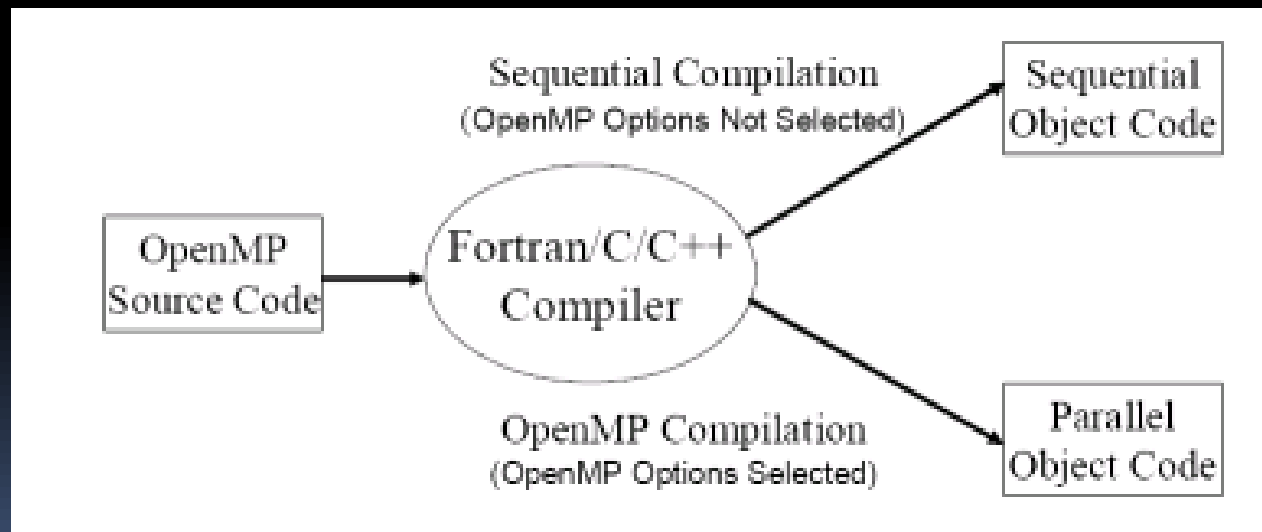


Biên dịch OpenMP



Trình dịch hỗ trợ OpenMP

- Nền tảng: C/C++ hoặc Fortran
- Chương trình dịch bằng cách thiết lập lựa chọn OpenMP trong tham số của lệnh dịch.



Thư viện lập trình đa luồng

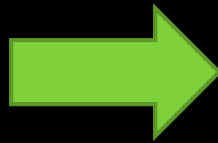
- Các chỉ thị OpenMP được dịch thành lời gọi các hàm quản lý và gán công việc cho luồng
- Chương trình dịch sử dụng thư viện lập trình đa luồng để dịch.
- Trình dịch khác nhau sử dụng thư viện lập trình đa luồng khác nhau
 - POSIX threads
 - Solaris Threads
 - Quick Threads

Dịch chương trình OpenMP

- OpenMP tách người lập trình với thư viện lập trình đa luồng
- Chương trình dịch xử lý các chỉ thị OpenMP và dùng chúng để tạo các đoạn mã đa luồng.
- Các mã này sẽ gọi hàm thời gian chạy của thư viện đa luồng
 - ▣ Những hàm của OpenMP được thực hiện qua các lời gọi đến các hàm thư viện trên.

Chuẩn hóa khối OpenMP

```
#pragma omp sections
{
    #pragma omp section
    section1();
    #pragma omp section
    section2();
    #pragma omp section
    section3();
}
```



```
#pragma omp for
for (id_section = 0; id_section <= 2; id_section++)
{
    switch (id_section)
    {
        case 0: section1(); break;
        case 1: section2(); break;
        case 2: section3(); break;
    }
}
```



Hết bài!!!!