

Programación con Multitareas

Junio/2012

Ing. Laura Sandoval Montaña, Andrés Mondragón Contreras

PAPIME 104911

Programación con Multitareas

- Fundamentos de procesamiento con multitareas
- Programación de multitareas utilizando hilos Posix
- **Programación de multitareas utilizando un compilador de C paralelo**

Ing. Laura Sandoval Montaña, Andrés Mondragón Contreras

PAPIME 104911

Programación de multitareas utilizando un compilador de C paralelo

- Intel Cilk++ SDK

Conjunto de herramientas, bibliotecas, documentación y ejemplos. Se tiene tanto para el sistema operativo Windows (32-bit) como para Linux (32 y 64-bit)

Soporte: <http://whatif.intel.com>

Programación de multitareas utilizando un compilador de C paralelo

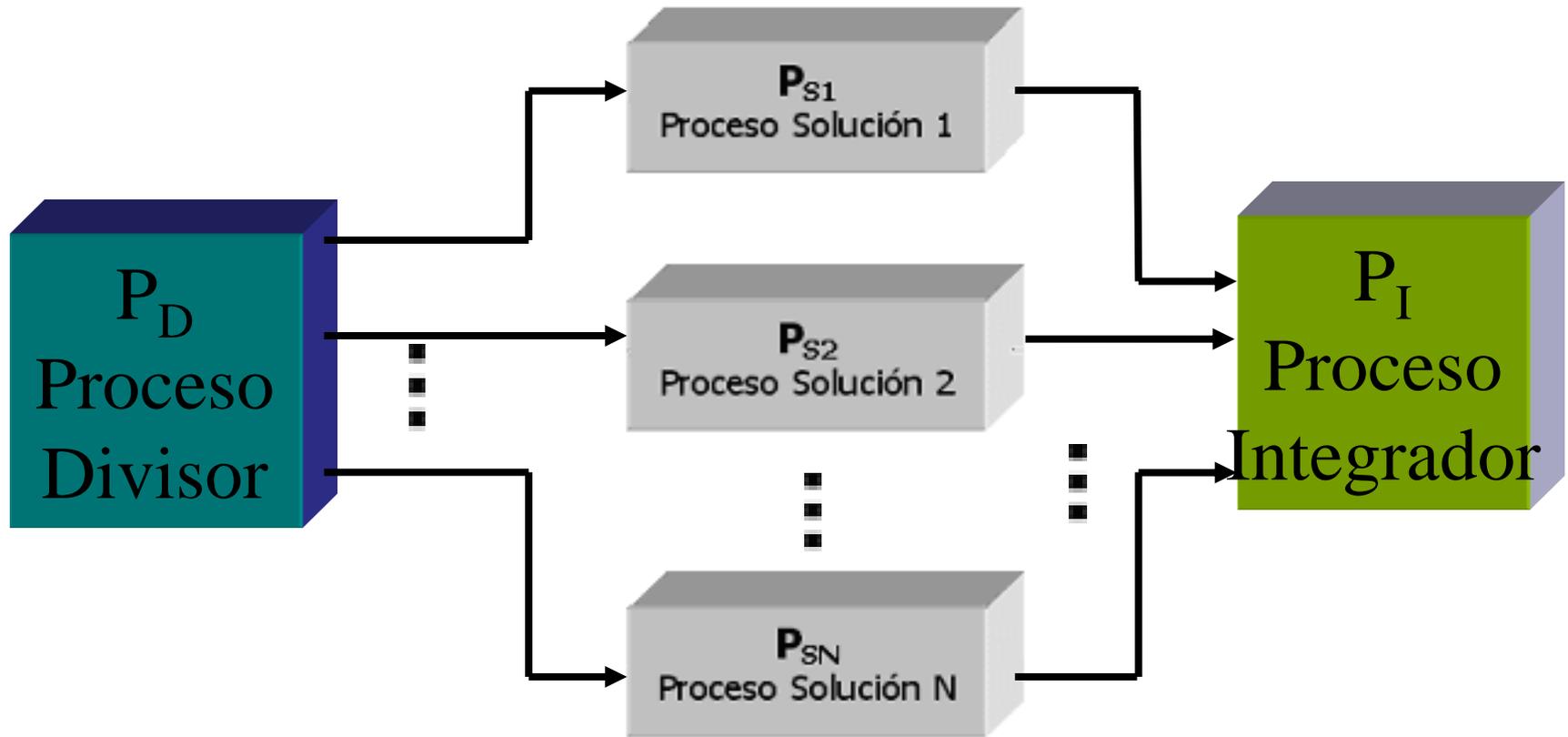
- Lenguaje Cilk++

Es una extensión del lenguaje C++ para simplificar el desarrollo de aplicaciones que aprovechen de manera eficiente un ambiente de varios procesadores.

Por lo regular, emplea el algoritmo divide y conquista, el cual resuelve los problemas partiéndolos en sub-problemas (tareas) que pueden ser resueltos independientemente, para después integrar los resultados.

Lenguaje Cilk++

Divide y Conquista



Programación de multitareas utilizando un compilador de C paralelo

- **Lenguaje Cilk++**

Las tareas pueden ser implementadas en funciones separadas o bien por iteraciones en un ciclo.

Cilk++ cuenta con palabras reservadas para identificar las funciones o las iteraciones de un ciclo que puedan ser ejecutadas en paralelo.

Programación de multitareas utilizando un compilador de C paralelo

- **Lenguaje Cilk++**

El Intel *Cilk++ runtime system* planifica dichas tareas para ejecutarlas de manera eficiente sobre varias unidades de procesamiento.

Un ***trabajo*** es un hilo del sistema operativo que el planificador de Cilk++ emplea para ejecutar una tarea definida en un programa escrito en Cilk++

Programación de multitareas utilizando un compilador de C paralelo

- **Lenguaje Cilk++**

Para identificar las tareas que se pueden ejecutar en paralelo, Cilk++ utiliza las siguientes palabras clave:

cilk_spawn llama a una función (hija) que puede ser ejecutada en paralelo con el que la llama (padre).

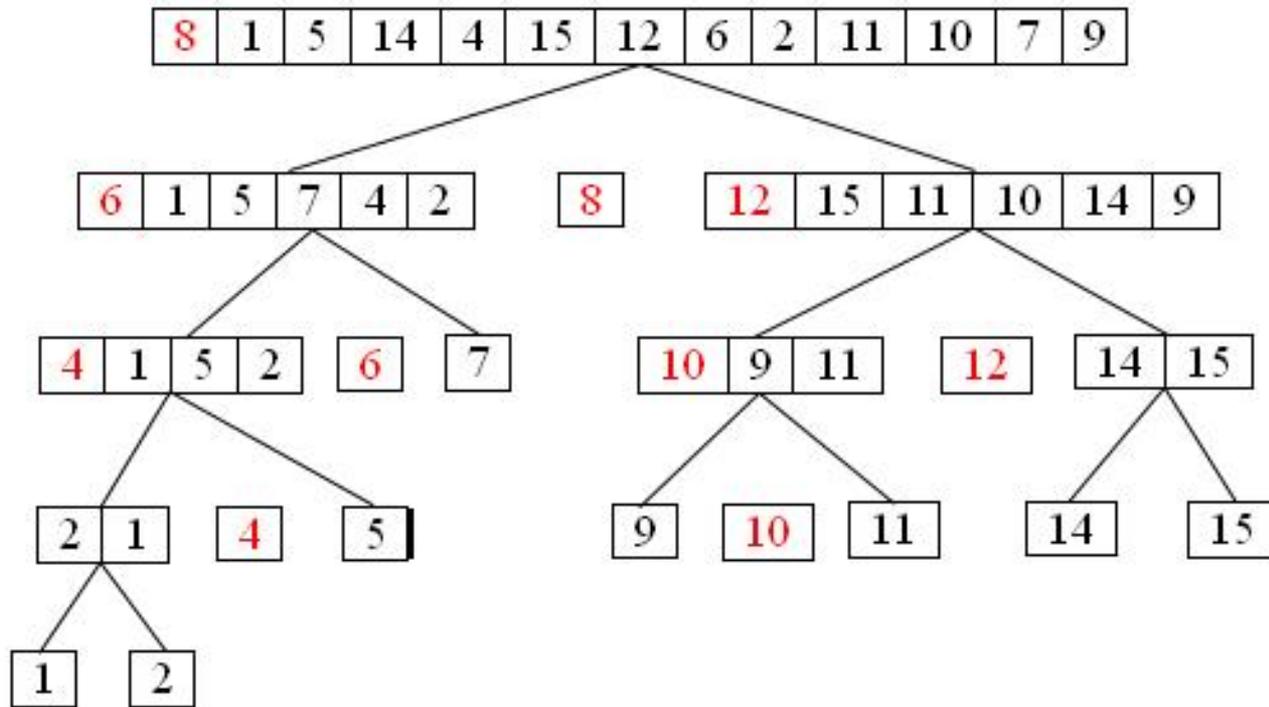
cilk_sync espera a que todas las funciones hijas se completen antes de que el padre prosiga.

cilk_for identifica un ciclo for cuyas iteraciones (todas) se pueden hacer en paralelo.

Lenguaje Cilk++ en Windows

- Requerimientos
 - Microsoft Visual Studio* 2005 y 2008
 - Asegurarse que el ambiente de Visual Studio esté correctamente establecido para cilk++, corriendo el archivo script vcvarsall.bat ubicado en ProgramFiles\Visual Studio 8\VC (para Visual Studio* 2005) o en ProgramFiles\Visual Studio 9.0\VC para Visual Studio* 2008.

Ejercicio: Algoritmo de ordenamiento Quicksort



Juntando los elementos, el arreglo quedaría ordenado

1 2 4 5 6 7 8 9 10 11 12 14 15

Lenguaje Cilk++

- **Ejercicio: qsort**

Desde Visual Studio compilar `qsort.cilk` o desde la línea de comandos: `>cilkpp qsort.cilk`

En la línea de comandos, ejecutar el programa: `>qsort`

Ejecutar nuevamente el programa indicándole al sistema operativo el número de trabajos que ocupará:

```
>qsort -cilk_set_worker_count=2
```

Lenguaje Cilk++

- **Carrera de Datos**

Ocurre cuando dos o más hilos paralelos, acceden a la misma localidad de memoria y al menos uno realiza una escritura.

El resultado del programa depende de qué hilo “gana la carrera” y accede primero a la memoria.

Ejemplo de carrera de datos

```
#include <iostream>
using namespace std;
#include <cilk.h>
#include <cilkview.h>
int a = 2; // variable global, visible para todo trabajo
void Strand1( ) {
    a = 1;
}
int Strand2( ) {
    return a;
}
void Strand3( ) {
    a = 2;
}
int cilk_main( ){
    int result;
    cilk_spawn Strand1( );
    result = cilk_spawn Strand2( );
    cilk_spawn Strand3( );
    cilk_sync;
    cout << "a = " << a << ", result = " << result;
    return 0;
}
```

Lenguaje Cilk++

- **Carrera de Datos**

Cilk++ cuenta con un detector de carrera de datos *cilkscreen*, el cual verifica si el código cuenta con este tipo de problema.

Ejercicio:

```
>cilkscreen qsort 1000  
Sorting 1000 integers  
0.078 seconds  
Sort succeeded.  
No errors found by Cilkscreen
```

Lenguaje Cilk++

- **Medición de desempeño y escalabilidad**

Con *cilkview*, se puede observar tanto el desempeño como la escalabilidad de nuestro programa.

El desempeño está dado en Speedup el cual se calcula de acuerdo a la siguiente fórmula:

$$S_p = \frac{T_1}{T_p}$$

donde:

p es el número de procesadores

T_1 es el tiempo de ejecución del algoritmo en forma serial

T_p es el tiempo de ejecución del algoritmo paralelo con p procesadores

Lenguaje Cilk++

- **Medición de desempeño y escalabilidad**

Para que cilkview muestre sus resultados, en el programa se debe crear un objeto

```
cilk::cilkview
```

y llamar a los métodos

```
start( ), stop( ) y dump( )
```

Ejercicio:

```
>cilkview -trials all 2 -verbose qsort
```

Lenguaje Cilk++

- **cilk_spawn**

var = cilk_spawn func(args); // func() regresa un valor
cilk_spawn func(args); // func() regresa void

- **cilk_sync**

Sólo espera a que terminen los hijos de la función que la invoca. Los hijos de otras funciones no son afectadas

`cilk_sync;`

Lenguaje Cilk++

- **cilk_for**

Al reemplazar el ciclo for serial con `cilk_for`, permite ejecutar las iteraciones del ciclo de forma paralela.

Ejemplos:

```
cilk_for (int i = begin; i < end; i += 2)  
    f(i);
```

```
cilk_for (T::iterator i(vec.begin()); i != vec.end(); ++i)  
    g(i);
```

Lenguaje Cilk++

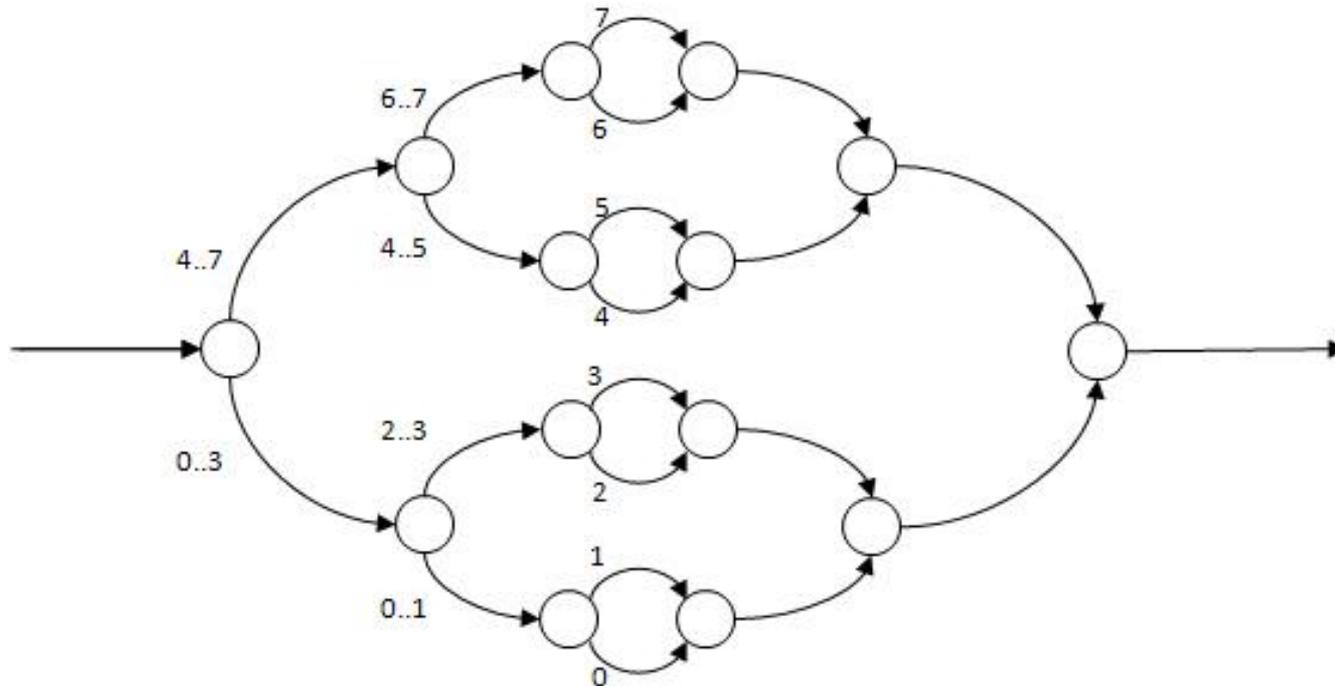
- **cilk_for (continuación)**

El compilador Cilk++ convierte el cuerpo del ciclo a una función que es llamada de forma recursiva usando la estrategia de divide y conquista.

De esta forma el planificador Cilk++ proporciona un mucho mejor desempeño.

Lenguaje Cilk++

Diagrama que representa la ejecución de un *cilk_for* de 8 iteraciones



Ing. Laura Sandoval Montaña, Andrés Mondragón Contreras

PAPIME 104911

Lenguaje Cilk++

- **cilk_for (continuación)**

cilk_for divide el ciclo en “pedazos” que contienen una o más iteraciones.

Cada pedazo se ejecuta de forma serial y se considera como un hilo dentro del ciclo.

El máximo número de iteraciones de un “pedazo” se le nombra *tamaño de grano (grain size)*

Lenguaje Cilk++

- **cilk_for (continuación)**

Para indicar el tamaño del grano se emplea el pragma `cilk_grainsize`:

```
#pragma cilk_grainsize = expresión
```

Ejemplo:

```
#pragma cilk_grainsize = 1  
cilk_for (int i=0; i<IMAX; ++i) { . . . }
```

Referencias

Bibliografía:

Cilk++ Programmer's Guide
Intel Cilk++ SDK

Sitios WEB

<http://whatif.intel.com>