OpenMP Application Program Interface

Ejemplos

### Versión 4.0.0 - Noviembre 2013

Copyright © 1997-2013 Junta de Revisión de la arquitectura OpenMP.

se concede permiso para copiar y sin Gastos de la totalidad o parte de este material, siempre que el aviso de copyright OpenMP Arquitectura Review Board y el título de este documento aparece. Se notifica que la copia es con permiso de Architecture Review Board OpenMP.

*Esta página está en blanco intencionadamente.*

CONTENIDO

[Introducción 5](#_bookmark0)

[Examples7](#_bookmark1)

1. [Un paralelo simple Loop7](#_bookmark2)
2. [La memoria OpenMP MODEL9](#_bookmark3)
3. [Condicional Compilation15](#_bookmark4)
4. [Variables de control internos (ICVs) dieciséis](#_bookmark5)
5. [los](#_bookmark6) paralela Construct19
6. [Controlar el número de hilos de anidamiento múltiple niveles 21](#_bookmark7)
7. [La interacción entre el](#_bookmark8) NUM\_THREADS Claus y

omp\_set\_dynamic 24

1. [Restricciones en el Fortran](#_bookmark9) **hacer** Construct26
2. [Fortran iteración del bucle privado Variables28](#_bookmark10)
3. [los](#_bookmark11) no, espera clause30
4. [los](#_bookmark12) colapso cláusula 33
5. [los](#_bookmark13) secciones paralelas Construct38
6. [los](#_bookmark14) Cláusula firstprivate y las secciones Construct39
7. [los](#_bookmark15) soltero Construct41
8. [tareas Constructs43](#_bookmark16)
9. [los](#_bookmark17) taskyield Directiva 67
10. [los](#_bookmark18) compartir el trabajo Construct69
11. [los](#_bookmark19) dominar Construct73
12. [los](#_bookmark20) crítico Construct75
13. [Las construcciones de trabajo compartido Dentro de una](#_bookmark21) **crítico** Construir 77
14. [Unión de](#_bookmark22) barrera Regions79
15. [los](#_bookmark23) atómico Construct82
16. [Las restricciones a la](#_bookmark24) **atómico** Construct88
17. [los](#_bookmark25) **rubor** Construir sin List92
18. [Lugar de](#_bookmark26) ras, barrera, taskwait

[y taskyield Directives95](#_TOC_250005)

1. [los](#_bookmark27) Cláusula ordenada y la ordenada Construct99
2. [Cancelación construcciones 103](#_bookmark28)
3. [los](#_bookmark29) threadprivate Directive108
4. [Paralelo iterador de acceso aleatorio Lazo 114](#_bookmark30)
5. [Restricciones en Fortran](#_bookmark31) Las cláusulas compartidas y privadas con Common bloques 115
6. [los](#_bookmark32) por defecto (ninguno) Cláusula 117
7. [Las condiciones de carrera causados ​​por copias implícita de variables compartidas en Fortran 119](#_bookmark33)
8. [los](#_bookmark34) privado Clause120
9. [Restricciones en Fortran Asociación de almacenamiento](#_bookmark35)

[con el](#_bookmark35) privado Cláusula 124

1. [C / Arrays C ++ en una](#_bookmark36) firstprivate Cláusula 127
2. [los](#_bookmark37) lastprivate Clause129
3. [los](#_bookmark38) reducción Cláusula 130
4. [los](#_bookmark39) copyin Clause136
5. [los](#_bookmark40) copyprivate Clause138
6. [Bucle anidado construcciones 142](#_bookmark41)
7. [Las restricciones a la jerarquización de Regiones 146](#_bookmark42)
8. [los](#_bookmark43) omp\_set\_dynamic y omp\_set\_num\_threads Routines152
9. [los](#_bookmark44) omp\_get\_num\_threads Rutina 154
10. [los](#_bookmark45) omp\_init\_lock Rutina 156
11. [Propiedad de Cabellos 157](#_bookmark46)
12. [Bloqueo sencillo rutinas 159](#_bookmark47)
13. [Bloqueo encajable rutinas 161](#_bookmark48)
14. [objetivo Construir 164](#_TOC_250004)
15. [datos de destino Construct171](#_TOC_250003)
16. [actualización de destinos Construir 182](#_TOC_250002)
17. [declarar objetivo Construir 186](#_TOC_250001)
18. [equipos construcciones 194](#_TOC_250000)
19. [La ejecución asíncrona de una](#_bookmark54) **objetivo** Usando región Tareas 202
20. [Secciones matriz en Device construcciones 206](#_bookmark55)
21. [Dispositivo rutinas 210](#_bookmark56)
22. [ASOCIADO Fortran Construct214](#_bookmark57)

# Introducción

Esta colección de ejemplos de programación complementa la API OpenMP para las especificaciones de memoria compartida Paralelización, y no es parte de las especificaciones formales. Se supone familiaridad con las especificaciones de OpenMP, y comparte las convenciones tipográficas utilizadas en este documento.

**Nota -** Esta primera versión de los ejemplos de OpenMP OpenMP refleja la versión 4.0 especificaciones. Ejemplos adicionales se están desarrollando y se publicarán en futuras versiones de este documento.

La especificación API OpenMP proporciona un modelo para la programación paralela que es portable a través de arquitecturas de memoria compartida de diferentes proveedores. Los compiladores de numerosos vendedores soportan la API OpenMP.

Las directivas, las rutinas de la biblioteca, y variables de entorno demostrados en este documento permiten a los usuarios crear y gestionar programas paralelos al tiempo que permite la portabilidad. Las directivas se extienden la C, C ++ y Fortran idiomas base con un solo programa de datos múltiples (SPMD) construye, tareas construcciones, las construcciones de dispositivos, constructos de trabajo compartido, y los constructos de sincronización, y proporcionan apoyo para los datos de la privatización de compartir y. La funcionalidad para controlar el entorno de ejecución es proporcionada por las rutinas de biblioteca y variables de entorno. Los compiladores que soportan la API OpenMP a menudo incluyen una opción de línea de comandos para el compilador que activa y permite la interpretación de todas las directivas OpenMP.

La información completa sobre el API OpenMP y una lista de los compiladores que soportan la API OpenMP se puede encontrar en el sitio web OpenMP.org

[**http://www.openmp.org**](http://www.openmp.org/)

*Esta página está en blanco intencionalmente.*

# Ejemplos

Los siguientes son ejemplos de las directivas API OpenMP, construcciones, y rutinas.

C / C ++



Una declaración siguiente directiva está compuesta sólo cuando sea necesario, y una declaración no tiene sangría compuesto con respecto a una directiva que lo precede.

C / C ++



## Un bucle paralelo simple

El siguiente ejemplo demuestra cómo paralelizar un bucle simple usando la construcción de lazo paralelo. La variable de iteración del bucle es privada por defecto, por lo que no es necesario especificar explícitamente en una cláusula privado.

*Ejemplo 1.1c*



C / C ++

void sencilla (int n, flotar \* a, float \* b)

{

int i;

#pragma omp paralelo para

for (i = 1; i <n; i ++) / \* i es privada por defecto \* / b [i] = (a [i] + a [i-1]) / 2,0;

}

C / C ++



Fortran



*Ejemplo 1.1f*

SIMPLE SUBRUTINA (N, A, B) INTEGER I, N

VERDADERO B (N), A (N)

! $ OMP PARALELO hacer! I es privada por defecto DO I = 2, N

B (I) = (A (I) + A (I-1)) / 2,0 ENDDO

! PARALELO $ OMP FIN DO

FIN SUBRUTINA SIMPLE

##### Fortran



## El modelo de memoria OpenMP

En el siguiente ejemplo, en Print 1, el valor de x puede ser 2 o 5, dependiendo de la sincronización de los hilos, y la aplicación de la asignación a x. Hay dos razones por las que el valor en Print 1 podría no ser 5. En primer lugar, impresión 1 puede ser ejecutado antes de ejecutar la tarea de x. En segundo lugar, incluso si de impresión 1 se ejecuta después de la asignación, el valor 5 no está garantizada para ser visto por hilo 1 porque un color puede no haber sido ejecutado por hilo 0 desde la asignación.

La barrera después de impresión 1 contiene rubores implícitos en todas las discusiones, así como una sincronización de hilo, para que el programador se garantiza que el valor 5 será impreso por tanto de impresión 2 y 3 de impresión.

C / C ++



*Ejemplo 2.1c*

#include <stdio.h> #include <omp.h>

int main () {int x;

x = 2;

#pragma omp NUM\_THREADS paralelos (2) compartido (x)

{

si (omp\_get\_thread\_num () == 0) {x = 5;

} Else {

/ \* Imprimir 1: la siguiente lectura de x tiene una carrera \* /

printf ( "1: Thread #% d: x =% d \ n", omp\_get\_thread\_num (), x);

}

barrera omp #pragma

si (omp\_get\_thread\_num () == 0) {

/ \* \* Imprimir 2 /

printf ( "2: Thread #% d: x =% d \ n", omp\_get\_thread\_num (), x);

} Else {

/ \* \* Imprimir 3 /

printf ( "3: Thread #% d: x =% d \ n", omp\_get\_thread\_num (), x);

}

}

return 0;

}

##### C / C ++



Fortran



*Ejemplo 2.1f*

PROGRAMA MEMMODEL

INCLUIR "Omp\_lib.h"! o usar OMP\_LIB entero x

X = 2

! NUM\_THREADS $ OMP paralelo (2) Piscina (X)

IF (OMP\_GET\_THREAD\_NUM () .EQ. 0), entonces X = 5

MÁS

! IMPRIMIR 1: La siguiente lectura de x tiene una carrera

PRINT \*, "1: Hilo #", OMP\_GET\_THREAD\_NUM () "X =", X ENDIF

! $ OMP BARRERA

IF (OMP\_GET\_THREAD\_NUM () .EQ. 0) ENTONCES

! IMPRESIÓN 2

PRINT \*, "2: Hilo #", OMP\_GET\_THREAD\_NUM () "X =", X ELSE

! IMPRESIÓN 3

PRINT \*, "3: Hilo #", OMP\_GET\_THREAD\_NUM () "X =", X ENDIF

! PARALELO $ OMP FIN

FIN DEL PROGRAMA MEMMODEL

##### Fortran



El siguiente ejemplo demuestra por qué sincronización es difícil de realizar correctamente a través de variables. El valor de la bandera no está definida en ambas impresiones en hilo 1 y el valor de datos sólo es bien definido en la segunda impresión.

C / C ++



*Ejemplo 2.2c*

#include <omp.h> #include <stdio.h> int main ()

{

datos int; bandera int = 0;

#pragma omp NUM\_THREADS paralelos (2)

{

si (omp\_get\_thread\_num () == 0)

{

/ \* Escribir en el búfer de datos que será leído por el hilo \* /

datos = 42;

/ \* Lavar los datos para enhebrar 1 y estrictamente el orden de escritura de datos

en relación con la escritura al ras de la bandera \* / #pragma omp (bandera, datos)

/ \* Bandera Set para liberar hilo 1 \* / bandera = 1;

/ \* Bandera Flush para asegurar que el hilo 1 se hace el cambio \* /

ras #pragma omp (bandera)

}

else if (omp\_get\_thread\_num () == 1)

{

/ \* Bucle hasta que veamos la actualización a color de la bandera \* / #pragma omp (bandera, datos)

mientras que (bandera <1)

{

#pragma omp flush (bandera, datos)

}

/ \* Los valores de bandera y los datos se indefinido \* / printf ( "flag =% d de datos =% d \ n", bandera, datos); #pragma omp flush (bandera, datos)

/ \* Los datos valores serán 42, el valor de la bandera todavía indefinido \* /

printf ( "flag =% d de datos =% d \ n", bandera, datos);

}

}

return 0;

}

##### C / C ++



Fortran



*Ejemplo 2.2f*

programa de ejemplo

Incluir "omp\_lib.h"! o uso OMP\_LIB INTEGER DATOS

INTEGER BANDERA

FLAG = 0

! NUM\_THREADS $ OMP paralelo (2) SI (OMP\_GET\_THREAD\_NUM () .EQ. 0) ENTONCES

! Escribir en el búfer de datos que será leído por hilo 1 DATA = 42

! DATOS ras de hilo 1 y estrictamente ordenan la escritura a los datos

! con respecto a la escritura a la bandera

! $ OMP DE DESCARGA (FLAG, DATOS)

! Conjunto FLAG para liberar hilo 1 FLAG = 1;

! BANDERA ras para asegurar que el hilo 1 ve el cambio \* /

! $ OMP DE DESCARGA (FLAG)

ELSE IF (OMP\_GET\_THREAD\_NUM () .EQ. 1), entonces

! Bucle hasta que veamos la actualización a la bandera

! $ OMP DE DESCARGA (FLAG, DATOS) DO WHILE (.LT FLAG. 1)

! $ OMP DE DESCARGA (FLAG, DATOS)

ENDDO

! Los valores de la bandera y de datos están sin definir IMPRESIÓN \*, 'BANDERA =', bandera, 'DATOS =', DE DATOS

! $ OMP DE DESCARGA (FLAG, DATOS)

TERMINARA SI

! Los valores de datos será de 42 años, el valor de la bandera todavía indefinido \* / PRINT \*, 'BANDERA =', bandera, 'DATOS =', DE DATOS

! $ OMP FIN FIN PARALELO

##### Fortran



El siguiente ejemplo demuestra por qué la sincronización es difícil de realizar correctamente a través de variables. Debido a que la escritura (1) -flush (1) -flush (2) -Leer (2) la secuencia no se puede garantizar en el ejemplo, las declaraciones sobre hilo 0 y el hilo 1 puede ejecutar en cualquier orden.

C / C ++



*Ejemplo 2.3c*

#include <omp.h> #include <stdio.h> int main ()

{

bandera int = 0;

#pragma omp NUM\_THREADS paralelos (3)

{

si (omp\_get\_thread\_num () == 0)

{

/ \* La bandera listo para lanzar el hilo 1 \* / #pragma omp actualización atómica

bandera ++;

/ \* Ras de la bandera está implícito en la directiva atómica \* /

}

else if (omp\_get\_thread\_num () == 1)

{

/ \* Bucle hasta que veamos que la bandera sea de 1 \* / #pragma omp ras (bandera)

mientras que (bandera <1)

{

ras #pragma omp (bandera)

}

printf ( "Thread 1 despertado \ n");

/ \* La bandera listo para lanzar rosca 2 \* / #pragma omp actualización atómica

bandera ++;

/ \* Ras de la bandera está implícito en la directiva atómica \* /

}

else if (omp\_get\_thread\_num () == 2)

{

/ \* Bucle hasta que veamos que la bandera llega a 2 \* / #pragma omp ras (bandera)

mientras que (bandera <2)

{

ras #pragma omp (bandera)

}

printf ( "Thread 2 despertado \ n");

}

}

return 0;

}

##### C / C ++



Fortran



*Ejemplo 2.3f*

programa de ejemplo

Incluir "omp\_lib.h"! o uso OMP\_LIB INTEGER FLAG

FLAG = 0

! NUM\_THREADS $ OMP paralelo (3) SI (OMP\_GET\_THREAD\_NUM () .EQ. 0) ENTONCES

! bandera listo para lanzar hilo 1

! ACTUALIZACIÓN $ OMP ATÓMICA

FLAG = FLAG + 1

! Ras de FLAG está implícito en la directiva atómica else if (OMP\_GET\_THREAD\_NUM () .EQ. 1) ENTONCES

! Bucle hasta que veamos que alcanza 1 BANDERA

! $ OMP DE DESCARGA (FLAG, DATOS) DO WHILE (.LT FLAG. 1)

! $ OMP DE DESCARGA (FLAG, DATOS)

ENDDO

IMPRESIÓN \*, 'Tema 1 despertado'

! Conjunto FLAG para liberar hilo 2

! ACTUALIZACIÓN $ OMP ATÓMICA

FLAG = FLAG + 1

! Ras de FLAG está implícito en la directiva atómica else if (OMP\_GET\_THREAD\_NUM () .EQ. 2) ENTONCES

! Bucle hasta que veamos que alcanza BANDERA 2

! $ OMP DE DESCARGA (FLAG, DATOS) DO WHILE (.LT FLAG. 2)

! $ OMP FLUSH (FLAG, DATA)

ENDDO

IMPRESIÓN \*, 'Tema 2 despertado'

TERMINARA SI

! $ OMP FIN FIN PARALELO

##### Fortran



## Compilación condicional

C / C ++



El siguiente ejemplo ilustra el uso de la compilación condicional utilizando la macro \_OPENMP OpenMP. Con la compilación OpenMP, la macro \_OPENMP queda definido.

*Ejemplo 3.1c*

#include <stdio.h> int main ()

{

#Ifdef \_OPENMP

printf ( "Compilado por una aplicación compatible con OpenMP. \ n"); # terminara si

return 0;

}

##### C / C ++



Fortran



El siguiente ejemplo ilustra el uso del centinela compilación condicional. Con la compilación OpenMP, el centinela la compilación condicional! $ Es reconocido y tratado como dos espacios. En la fuente de forma fija, declaraciones vigilados por el centinela deben comenzar después de la columna 6.

*Ejemplo 3.1f*

Programa de ejemplo C234567890

! $ PRINT \*, "Compilado por una aplicación compatible con OpenMP."

Ejemplo de programa END

##### Fortran



## Control interno Variables (ICVs)

De acuerdo con $, una implementación OpenMP debe actuar como si hay ICVs que controlan el comportamiento del programa. Este ejemplo ilustra dos ICVs, nthreads-var y max--niveles activos-var. La ICV nthreads-var controla el número de hilos requeridos para las regiones paralelas encontradas; hay una copia de este ICV por tarea. La ICV niveles-var max-activo- controla el número máximo de regiones paralelas activos anidados; hay una copia de este ICV para todo el programa.

En el siguiente ejemplo, el nido-var,-Active-niveles max-Var, dyn-var, y ICVs nthreads-var se modifican a través de llamadas a las rutinas de biblioteca de tiempo de ejecución omp\_set\_nested, omp\_set\_max\_active\_levels, omp\_set\_dynamic, y omp\_set\_num\_threads respectivamente. Estos ICVs afectan al funcionamiento de regiones paralelas. Cada tarea implícita generada por una región paralela tiene su propia copia de los ICVs nido-var, var-din, y nthreads-var.

En el siguiente ejemplo, el nuevo valor de nthreads-var aplica sólo a las tareas implícitas que ejecutar la llamada a omp\_set\_num\_threads. Hay una copia de la Max-ICV-niveles activos-var para todo el programa y su valor es el mismo para todas las tareas. Este ejemplo asume que el paralelismo anidado es compatible.

La región exterior paralelo crea un equipo de dos hilos; cada uno de los hilos se ejecutará una de las dos tareas implícitas generadas por la región exterior paralelo.

Cada tarea implícita generada por la región exterior paralelo llama omp\_set\_num\_threads (3), asignando el valor 3 a su respectiva copia de nthreads-var. Entonces cada tarea implícita encuentra una región paralela interior que crea un equipo de tres hilos; cada uno de los hilos se ejecutará una de las tres tareas implícitas generadas por esa región paralela interior.

Dado que la región exterior paralelo es ejecutado por 2 hilos, y el interior por 3, habrá un total de 6 tareas implícitas generadas por las dos regiones paralelas interiores.

Cada tarea implícita generada por una región paralela interior ejecutará la llamada a omp\_set\_num\_threads (4), asignando el valor 4 a su respectiva copia de nthreads-var.

La declaración de impresión en la región exterior paralelo es ejecutado por uno solo de los hilos en el equipo. Por lo tanto, se ejecutará sólo una vez.

La declaración de impresión en una región paralela interno también es ejecutado por uno solo de los hilos en el equipo. Ya que tenemos un total de dos regiones internas paralelas, la declaración de impresión se ejecuta dos veces - una vez por cada región paralela interior.

C / C ++



*Ejemplo 4.1c*

#include <stdio.h> #include <omp.h>

int main (void)

{

omp\_set\_nested (1); omp\_set\_max\_active\_levels (8); omp\_set\_dynamic (0); omp\_set\_num\_threads (2); #pragma omp paralelo

{

omp\_set\_num\_threads (3);

#pragma omp paralelo

{

omp\_set\_num\_threads (4); #pragma omp sola

{

/ \*

* **Lo siguiente debe imprimir:**
* **Interior: max\_act\_lev = 8, num\_thds = 3, max\_thds = 4**
* **Interior: max\_act\_lev = 8, num\_thds = 3, max\_thds = 4**

\* /

printf ( "interno: max\_act\_lev =% d, num\_thds =% d, max\_thds =% d \ n", omp\_get\_max\_active\_levels (), (), omp\_get\_num\_threads omp\_get\_max\_threads ());

}

}

barrera #pragma omp #pragma omp sola

{

/ \*

* **Lo siguiente debe imprimir:**
* **Outer: max\_act\_lev = 8, num\_thds = 2, max\_thds = 3**

\* /

printf ( "Outer: max\_act\_lev =% d, num\_thds =% d, max\_thds =% d \ n", omp\_get\_max\_active\_levels (), (), omp\_get\_num\_threads omp\_get\_max\_threads ());

}

}

return 0;

}

##### C / C ++



Fortran



*Ejemplo 4.1f*

programa de ICV uso omp\_lib

llamar omp\_set\_nested (.TRUE).

llaman omp\_set\_max\_active\_levels (8) llaman omp\_set\_dynamic (.FALSE.) llamar omp\_set\_num\_threads (2)

! $ OMP paralelo

llamar omp\_set\_num\_threads (3)

! $ OMP paralelo

llamar omp\_set\_num\_threads (4)

! $ OMP sola

! Lo siguiente debe impresión:

Inner: max\_act\_lev = 8, num\_thds = 3, max\_thds = 4

Inner: max\_act\_lev = 8, num\_thds = 3, max\_thds = 4

print \*, "Inner: max\_act\_lev =", omp\_get\_max\_active\_levels (), y "num\_thds =", omp\_get\_num\_threads (),

& "Max\_thds =", omp\_get\_max\_threads ()

! Solo $ final omp

! $ Final omp paralelo

! Barrera omp $

! $ OMP sola

! Lo siguiente debe impresión:

! Externo: max\_act\_lev = 8, num\_thds = 2, = max\_thds 3

print \*, "externo: max\_act\_lev =", omp\_get\_max\_active\_levels (), y "num\_thds =", omp\_get\_num\_threads (),

& "Max\_thds =", omp\_get\_max\_threads ()

! Solo $ final omp

! $ OMP de extremo a extremo en paralelo

##### Fortran



## El constructo paralelo

El constructo paralelo se puede utilizar en programas paralelos de grano grueso. En el siguiente ejemplo, cada hilo en la región paralelo decide qué parte de la matriz global X para trabajar en, basado en el número del hilo:

C / C ++



*Ejemplo 5.1c*

# include <omp.h>

subdominio vacío (float \* x, int istart, int maximiles)

{

int i;

for (i = 0; i <maximiles; i ++) x [istart + i] = 123,456;

}

void sub (float \* x, NPOINTS INT)

{

int iam, nt, maximiles, istart;

#pragma omp predeterminado paralelo (compartido) privada (IAM, NT, maximiles, istart)

{

iam = omp\_get\_thread\_num (); nt = omp\_get\_num\_threads ();

maximiles = NPOINTS / nt; / \* tamaño de la partición \* / iStart = iam maximiles \*; / \* De partida índice de matriz \* / If (== iam NT-1) / \* último hilo puede hacer más \* /

iPoints = NPOINTS - iStart; subdominio (x, iStart, maximiles);

}

}

int main ()

{

flotar array [10000]; sub (array, 10000);

return 0;

}

##### C / C ++



Fortran



*Ejemplo 5.1f*

SUBRUTINA SUBDOMINIO (X, ISTART, iPoints) INTEGER ISTART, iPoints

Real x (\*) entero i

DO 100 I = 1, iPoints

X (ISTART + I) = 123,456

100CONTINUE

FIN SUBRUTINA SUBDOMINIO SUBRUTINA SUB (X, NPOINTS)

INCLUIR "Omp\_lib.h"! o uso OMP\_LIB

Real x (\*) INTEGER NPOINTS

INTEGER IAM, NT, iPoints, ISTART

! $ OMP PARALELO DEFAULT (PRIVADO) COMPARTIDA (X, NPOINTS) IAM = OMP\_GET\_THREAD\_NUM ()

NT = OMP\_GET\_NUM\_THREADS ()

IPoints = NPOINTS / NT ISTART = IAM \* iPoints IF (IAM .EQ. NT-1), entonces

IPoints = NPOINTS - ISTART ENDIF

LLAMADA SUBDOMINIO (X, ISTART, iPoints)

! PARALELO $ OMP FIN

END SUB SUBRUTINA

PROGRAMA PAREXAMPLE REAL MATRIZ (10000)

SUB CALL (ARRAY, 10000) END PROGRAM PAREXAMPLE

##### Fortran



## Controlador El número de procesos en múltiples niveles de anidamiento

Los siguientes ejemplos demuestran cómo utilizar la variable OMP\_NUM\_THREADS medio ambiente para controlar el número de hilos en varios niveles de anidamiento:

C / C ++



*Ejemplo 6.1c*

#include <stdio.h> #include <omp.h> int main (void)

{

omp\_set\_nested (1); omp\_set\_dynamic (0); #pragma omp paralelo

{

#pragma omp paralelo

{

#pragma omp sola

{

/ \*

* **Si OMP\_NUM\_THREADS = 2,3 se estableció, el siguiente debe imprimir:**
* **Interior: num\_thds = 3**
* **Interior: num\_thds = 3**

\*

* **Si no se admite la anidación, el siguiente debe imprimir:**
* **Interior: num\_thds = 1**
* **Interior: num\_thds = 1**

\* /

printf ( "interiores: num\_thds =% d \ n", omp\_get\_num\_threads ());

}

}

barrera #pragma omp omp\_set\_nested (0); #pragma omp paralelo

{

#pragma omp sola

{

/ \*

* **Incluso si = 2,3 se estableció OMP\_NUM\_THREADS, el siguiente debe**
* **impresión, debido a la anidación está desactivada:**
* **Interior: num\_thds = 1**
* **Interior: num\_thds = 1**

\* /

printf ( "interiores: num\_thds =% d \ n", omp\_get\_num\_threads ());

}

}

barrera #pragma omp #pragma omp sola

{

/ \*

* **Si OMP\_NUM\_THREADS = 2,3 se estableció, el siguiente debe imprimir:**
* **Outer: num\_thds = 2**

\* /

printf ( "Outer: num\_thds =% d \ n", omp\_get\_num\_threads ());

}

}

return 0;

}

##### C / C ++



Fortran



*Ejemplo 6.1f*

programa de ICV uso omp\_lib

llamar omp\_set\_nested (.TRUE.) llamar omp\_set\_dynamic (.FALSE).

! $ OMP paralelo

! $ OMP paralelo

! $ OMP sola

! Si OMP\_NUM\_THREADS = 2,3 se estableció, el siguiente debe imprimir:

! Interior: num\_thds = 3

! Interior: num\_thds = 3

! Si no se admite la anidación, el siguiente debe imprimir:

! Interior: num\_thds = 1

! Interior: num\_thds = 1

print \*, "Interiores: num\_thds =", omp\_get\_num\_threads ()

! Solo $ final omp

! $ Final omp paralelo

! Barrera omp $

llamar omp\_set\_nested (.FALSE).

! $ OMP paralelo

! $ OMP sola

! Incluso si OMP\_NUM\_THREADS = 2,3 se estableció, el siguiente debe imprimir,

! porque anidación está desactivada:

! Interior: num\_thds = 1

! Interior: num\_thds = 1

print \*, "Interiores: num\_thds =", omp\_get\_num\_threads ()

! Solo $ final omp

! $ Final omp paralelo

! Barrera omp $

! $ OMP sola

! Si OMP\_NUM\_THREADS = 2,3 se estableció, el siguiente debe imprimir:

! Outer: num\_thds = 2

print \*, "externos: num\_thds =", omp\_get\_num\_threads ()

! Solo $ final omp

! $ Final omp paralelo

fin

##### Fortran



1. **Interacción entre los NUM\_THREADS**

**Cláusula y omp\_set\_dynamic**

El siguiente ejemplo demuestra la cláusula NUM\_THREADS y el efecto de la

**omp\_set\_dynamic** rutinaria en él.

La llamada a la rutina omp\_set\_dynamic con el argumento 0 en C / C ++ o .FALSE. en Fortran, desactiva el ajuste dinámico del número de hilos en las implementaciones de OpenMP que lo soportan. En este caso, se proporcionan 10 hilos. Tenga en cuenta que en caso de un error de la aplicación OpenMP es libre para abortar el programa o para suministrar cualquier número de temas disponibles.

C / C ++



*Ejemplo 7.1c*

#include <omp.h> int main ()

{

omp\_set\_dynamic (0);

#pragma omp NUM\_THREADS paralelas (10)

{

/ \* Trabajo aquí \* /

}

return 0;

}

##### C / C ++



Fortran



*Ejemplo 7.1f*

programa de ejemplo

INCLUIR "Omp\_lib.h"! o llame USO OMP\_LIB OMP\_SET\_DYNAMIC (.FALSE.)

! $ OMPPARALLEL NUM\_THREADS (10)

! trabajo aquí

! $ OMPEND FIN DEL PROGRAMA PARALELO EJEMPLO

##### Fortran



La llamada a la rutina omp\_set\_dynamic con un argumento distinto de cero en C / C ++, o

**.CIERTO.** en Fortran, permite la implementación OpenMP para elegir cualquier número de hilos de entre 1 y 10.

C / C ++



*Ejemplo 7.2c*

#include <omp.h> int main ()

{

omp\_set\_dynamic (1);

#pragma omp NUM\_THREADS paralelas (10)

{

/ \* Trabajo aquí \* /

}

return 0;

}

##### C / C ++



Fortran



*Ejemplo 7.2F*

programa de ejemplo

INCLUIR "Omp\_lib.h"! o usar OMP\_LIB LLAMADA OMP\_SET\_DYNAMIC (.TRUE.)

! $ OMPPARALLEL NUM\_THREADS (10)

! trabajo aquí

! $ OMPEND FIN DEL PROGRAMA PARALELO EJEMPLO

##### Fortran



Es una buena práctica para establecer el ICV dyn-var explícitamente llamando al omp\_set\_dynamic

rutina, ya que su configuración por defecto se define la aplicación.

Fortran



## Restricciones Fortran En la pantalla ¿Construir

Si un fin hacer Directiva sigue un do-construcción en la que varias declaraciones DO comparten una declaración de terminación hacen, entonces hacer una directiva sólo puede especificarse para el más exterior de estas declaraciones. El ejemplo siguiente contiene usos correctos de construcciones de bucle:

*Ejemplo 8.1f*

TRABAJO SUBRUTINA (I, J) INTEGER I, J

FIN DE TRABAJO SUBRUTINA

SUBRUTINA DO\_GOOD () INTEGER I, J

A real (1000)

|  |  |  |
| --- | --- | --- |
| **! $ OMP** | **DO 100 I**  **HACER** | **= 1,10** |
| **100** | **DO 100 de continuación de llamada** | **J = 1,10 TRABAJO (I, J)**  **! ! $ OMP ENDDO implícita aquí** |
| **! $ OMP** | **HACER**  **DO 200 J** | **= 1,10** |
| **200**  **! $ OMP** | **A (I) = ENDDO** | **I + 1** |
| **! $ OMP** | **HACER**  **DO 300 I** | **= 1,10** |
|  | **DO 300** | **J = 1,10** |
| **300** | **LLAMADA**  **CONTINUAR** | **TRABAJO (I, J)** |
| **! $ OMP** | **ENDDO** |  |

FIN SUBRUTINA DO\_GOOD

El siguiente ejemplo no es conforme porque el juego no directiva para la

**final hacer** no precede el bucle más externo:

*Ejemplo 8.2f*

TRABAJO SUBRUTINA (I, J) INTEGER I, J

FIN DE TRABAJO SUBRUTINA

SUBRUTINA DO\_WRONG INTEGER I, J

DO 100 I = 1,10

! $ OMPDO

DO 100 J = 1,10 identificador de llamada (I, J)

100CONTINUE

! $ OMPENDDO

FIN SUBRUTINA DO\_WRONG

##### Fortran



Fortran



## Fortran iteración del bucle privado Variables

En general, bucle de iteración las variables serán privados, cuando se utiliza en el do-loop de un do y

**Do paralelo** construir o en los bucles secuenciales en una construcción paralela (véase $ y

PS En el siguiente ejemplo de un bucle secuencial en un paralelo construir la variable de bucle de iteración seré privado.

*Ejemplo 9.1f*

SUBRUTINA PLOOP\_1 (A, N)

INCLUIR "Omp\_lib.h"! o usar OMP\_LIB

Un VERDADERO (\*)

INTEGER I, MYOFFSET, N

! $ OMP PARALELO PRIVADO (MYOFFSET) MYOFFSET = OMP\_GET\_THREAD\_NUM () \* N DO I = 1, N

A (MYOFFSET + I) = FLOAT (I) ENDDO

! PARALELO $ OMP FIN

FIN SUBRUTINA PLOOP\_1

En casos excepcionales, las variables de iteración de bucle se pueden hacer compartidos, como en el siguiente ejemplo:

*Ejemplo 9.2f*

SUBRUTINA PLOOP\_2 (A, B, N, I1, I2) de REAL A (\*), B (\*)

INTEGER I1, I2, N

! PARALELO $ OMP compartido (A, B, I1, I2)

! $ SECCIONES OMP

! $ OMP SECCIÓN

DO I1 = I1, N

IF (A I1) .NE.0.0 () SALIDA ENDDO

! $ OMP SECCIÓN

DO I2 = I2, N

IF (B (I2) .NE.0.0) ENDDO SALIDA

! $ SECCIONES OMP END

! $ OMP SOLA

SI (I1.LE.N) IMPRESIÓN \*, 'elementos en una HASTA', I1, 'son todos cero. SI (I2.LE.N) IMPRESIÓN \*, 'B ARTÍCULOS EN HASTA', I2, 'son todos cero.

! $ OMP único extremo

! PARALELO $ OMP FIN

FIN SUBRUTINA PLOOP\_2

Tenga en cuenta sin embargo, que el uso de variables de iteración de bucle compartido puede llevar fácilmente a las condiciones de carrera.

Fortran



## La cláusula nowait

Si hay múltiples bucles independientes dentro de una región paralela, se puede utilizar la

**no, espera** cláusula para evitar la barrera implícita al final de la construcción de lazo, como sigue:

C / C ++



*Ejemplo 10.1c*

#include <math.h>

nowait\_example void (int n, int m, flotador \* a, float \* b, float \* y, flotar \* z)

{

int i;

#pragma omp paralelo

{

omp #pragma para nowait for (i = 1; i <n; i ++)

b [i] = (a [i] + a [i-1]) / 2,0;

omp #pragma para nowait for (i = 0; i <m; i ++)

y [i] = sqrt (z [i]);

}

}

##### C / C ++



Fortran



*Ejemplo 10.1f*

SUBRUTINA NOWAIT\_EXAMPLE (N, M, A, B, Y, Z)

INTEGER N, M

VERDADERO A (\*), B (\*), Y (\*), Z (\*) INTEGER I

! $ OMP PARALELO

! $ OMP DO

DO I = 2, N

B (I) = (A (I) + A (I-1)) / 2,0 ENDDO

! $ OMP FIN HACER NOWAIT

! $ OMP DO

DO I = 1, M

Y (I) = SQRT (Z (I)) ENDDO

! $ OMP FIN HACER NOWAIT

! PARALELO $ OMP FIN

FIN SUBRUTINA NOWAIT\_EXAMPLE

##### Fortran



En el siguiente ejemplo, la programación estática distribuye los mismos números de iteración lógicas a los hilos que se ejecutan las tres regiones de bucle. Esto permite que la cláusula nowait a ser utilizado, a pesar de que existe una dependencia de datos entre los bucles. La dependencia es satisfecha mientras el mismo hilo ejecuta los mismos números de iteración lógicos en cada bucle.

Tenga en cuenta que el número de iteraciones de los bucles debe ser el mismo. El ejemplo satisface este requisito, ya que el espacio iteración de los dos primeros bucles es de 0 a n-1 (de 1 a N en la versión Fortran), mientras que el espacio iteración del último bucle es de 1 a n (2 a N 1 en la versión Fortran).



*Ejemplo 10.2c*

C / C ++

#include <math.h>

nowait\_example2 void (int n, flotar \* a, float \* b, float \* c, float \* y, flotar \* z)

{

int i;

#pragma omp paralelo

{

omp #pragma para el horario nowait (estática) para (i = 0; i <n; i ++)

c [i] = (a [i] + b [i]) / 2.0f; omp #pragma para el horario nowait (estática)

for (i = 0; i <n; i ++) z [i] = sqrtf (c [i]);

omp #pragma para el horario (estática) nowait for (i = 1; i <= n; i ++)

y [i] = z [i-1] + a [i];

}

}

##### C / C ++



Fortran



*Ejemplo 10.2f*

SUBRUTINA NOWAIT\_EXAMPLE2 (N, A, B, C, Y, Z) INTEGER N

VERDADERO A (\*), B (\*), C (\*), Y (\*), Z (\*) INTEGER I

! $ OMP PARALELO

! $ OMP DO PROGRAMA (estático)

DO I = 1, N

C (I) = (A (I) + B (I)) / 2,0 ENDDO

! $ OMP FIN HACER NOWAIT

! $ OMP DO PROGRAMA (estático) DO I = 1, N

Z (I) = SQRT (C (I)) ENDDO

! $ OMP FIN HACER NOWAIT

! $ OMP DO PROGRAMA (estático) DO I = 2, N + 1

Y (I) = Z (I-1) + A ENDDO (I)

! $ OMP FIN HACER NOWAIT

! PARALELO $ OMP FIN

FIN SUBRUTINA NOWAIT\_EXAMPLE2

##### Fortran



## La cláusula de colapso

En el siguiente ejemplo, los bucles k y j están asociados con la construcción de lazo. Así las iteraciones de los bucles k y j se contraen en un bucle con un espacio iteración más grande, y que bucle se dividen entonces entre los hilos en el equipo actual. Desde el bucle i no está asociado con el constructo de bucle, no se colapsó, y el bucle i se ejecuta secuencialmente en su totalidad en cada iteración del bucle k y j colapsado.

C / C ++



La variable j se puede omitir de la cláusula privada cuando se utiliza la cláusula colapso ya que es implícitamente privado. Sin embargo, si se omite la cláusula colapso continuación, j será compartida si se omite la cláusula privado. En cualquier caso, k es implícitamente privada y podría ser omitida de la cláusula privado.

*Ejemplo 11.1c*

void bar (flotar \* a, int i, j int, int k); int kl, ku, ks, JL, ju, js, il, iu, es; sin efecto secundario (float \* a)

{

int i, j, k;

omp #pragma para el colapso (2) privada (i, k, j) para (k = kl; k <= ku; k + = ks)

para (j = jl; j <= ju; j + = js) para (i = il; i <= iu; i + = IS)

bar (a, i, j, k);

}

##### C / C ++



Fortran



*Ejemplo 11.1f*

sub subrutina (a) venta de un (\*)

kl número entero, ku, ks, JL, ju, js, il, iu, es común / csub / kl, ku, ks, JL, ju, js, il, iu, es número entero i, j, k

! $ OMP hacer colapso (2) privada (i, j, k) no k = kl, ku, ks

hacer J = jl, ju, js hacer i = il, iu, es

bar llamada (a, i, j, k) enddo

enddo enddo

FIN $ OMP hacer

subrutina final

##### Fortran



En el siguiente ejemplo, los bucles k y j están asociados con la construcción de lazo. Así las iteraciones de los bucles k y j se contraen en un bucle con un espacio iteración más grande, y que bucle se dividen entonces entre los hilos en el equipo actual.

La ejecución secuencial de las iteraciones en los bucles k y j determina el orden de las iteraciones en el espacio de iteración colapsado. Esto implica que en la secuencialmente última iteración del espacio iteración colapsado, k tendrá el valor 2 y j tendrá el valor 3. Desde Klast y jlast se lastprivate, sus valores son asignados por el secuencialmente última iteración de la k colapsado y j lazo. Este ejemplo imprime: 2 3.

C / C ++



*Ejemplo 11.2c*

#include <stdio.h> prueba de vacío ()

{

int j, k, jlast, Klast; #pragma omp paralelo

{

omp #pragma para el colapso (2) lastprivate (jlast, Klast) para (k = 1; k <= 2; k ++)

para (j = 1; j <= 3; j ++)

{

jlast = j; Klast = k;

}

#pragma omp sola

printf ( "% d% d \ n", Klast, jlast);

}

}

##### C / C ++



Fortran



*Ejemplo 11.2f*

Control de elaboración

! $ OMP paralelo

! $ OMP hacer privada (j, k) colapso (2) lastprivate (jlast, Klast) hacer k = 1,2

hacer j = 1,3 jlast = j = k Klast

enddo enddo

FIN $ OMP hacer

! $ OMP sola

\* impresión, Klast, jlast

! Solo $ final omp

! $ Final omp paralelo

Control de elaboración final

##### Fortran



El siguiente ejemplo ilustra la interacción del colapso y cláusulas ordenados.

En el ejemplo, la construcción de lazo tiene tanto una cláusula de colapso y una cláusula de ordenado. La cláusula colapso hace que las iteraciones de la k y j bucles para ser colapsado en un bucle con un espacio iteración más grande, y que bucle se divide entre los hilos en el equipo actual. Una cláusula ordenado se añade a la construcción de lazo, debido a que una región ordenado se une a la región del bucle que surge de la construcción de lazo.

De acuerdo con $, un hilo no debe ejecutar más de una región ordenado que se une a la misma región de bucle. Por lo que la cláusula de colapso se requiere para el ejemplo que se va conformando. Con la cláusula de colapso, las iteraciones de los bucles k y j se contraen en un bucle, y por lo tanto sólo una región ordenado se unirán al bucle k y j colapsado. Sin la cláusula de colapso, habría dos regiones ordenados que se unen a cada iteración del bucle k (un derivado de la primera iteración del bucle j, y la otra derivada de la segunda iteración del bucle j).

La impresora de código

0 1 1

0 1 2

0 2 1

1 2 2

1 3 1

1 3 2

##### C / C ++



*Ejemplo 11.3c*

# include <omp.h> #include <stdio.h>

trabajo void (int a, int j, int k); sub void ()

{

int j, k, a;

#pragma omp NUM\_THREADS paralelos (2)

{

omp #pragma para el colapso (2) ordenó horario privado (j, k) (estático, 3) para (k = 1; k <= 3; k ++)

para (j = 1; j <= 2; j ++)

{

#pragma omp ordenó

printf ( "% d% d% d \ n", omp\_get\_thread\_num (), k, j);

/ \* Final ordenó \* / trabajo (a, j, k);

}

}

}

##### C / C ++



Fortran



*Ejemplo 11.3f*

prueba del programa incluyen 'omp\_lib.h'

! $ OMP NUM\_THREADS paralelas (2)

! $ OMP hacer colapso (2) ordenó horario privada (j, k) (estática, 3) de hacer k = 1,3

hacer j = 1,2

! $ OMP ordenó

impresión \*, omp\_get\_thread\_num (), k, j

FIN $ OMP ordenó

llamada de trabajo (a, j, k) enddo

enddo

FIN $ OMP hacer

! $ Final omp paralelo

Control de elaboración final

##### Fortran



## Las secciones paralelas Construct

En el siguiente ejemplo rutinas XAXIS, eje Y, y ZAXIS se pueden ejecutar concurrentemente. La primera directiva sección es opcional. Tenga en cuenta que todas las directivas de sección deben aparecer en las secciones paralelas construyen.

##### C / C ++



*Ejemplo 12.1c*

anular XAXIS (); anular eje Y (); void ZAXIS ();

sect\_example void ()

{

#pragma omp secciones paralelas

{

sección #pragma omp XAXIS ();

sección #pragma omp eje Y ();

sección #pragma omp ZAXIS ();

}

}

##### C / C ++



*Ejemplo 12.1f*

SECT\_EXAMPLE SUBRUTINA ()

! $ SECCIONES PARALELAS OMP

! $ OMP SECCIÓN

LLAMADA XAXIS ()

! $ OMP SECCIÓN

LLAMADA eje Y ()

! $ OMP SECCIÓN

LLAMADA ZAXIS ()

! $ secciones paralelas OMP END

FIN SUBRUTINA SECT\_EXAMPLE



##### Fortran

Fortran



1. **La cláusula y el firstprivate**

**secciones Construct**

En el siguiente ejemplo de las secciones de la construcción de la cláusula firstprivate se utiliza para inicializar la copia privada de section\_count de cada hilo. El problema es que las construcciones sección modifican section\_count, que rompe la independencia de la sección construcciones. Cuando diferentes hilos ejecutan cada sección, ambas secciones se imprimirán el valor 1. Cuando el mismo hilo ejecuta las dos secciones, una sección imprimirá el valor 1 y el otro se imprimirá el valor 2. Puesto que el orden de ejecución de las dos secciones en este caso no se especifica, no está especificado impresiones de sección, que qué valor.

*Ejemplo 13.1c*



C / C ++

# include <omp.h> #include <stdio.h> #define NT 4

int main () {

int section\_count = 0; omp\_set\_dynamic (0); omp\_set\_num\_threads (NT);

#pragma omp paralelo

#pragma secciones OMP firstprivate (section\_count)

{

sección #pragma omp

{

section\_count ++;

/ \* Puede imprimir el número uno o dos \* /

printf ( "section\_count% d \ n", section\_count);

}

sección #pragma omp

{

section\_count ++;

/ \* Puede imprimir el número uno o dos \* /

printf ( "section\_count% d \ n", section\_count);

}

}

volver 1;

}

##### C / C ++



Fortran



*Ejemplo 13.1f*

sección de uso del programa omp\_lib

número entero :: section\_count = 0 número entero, el parámetro :: NT = 4 llamada omp\_set\_dynamic (.FALSE.) omp\_set\_num\_threads de llamadas (NT)

! $ OMP paralelo

! $ Secciones OMP firstprivate (section\_count)

! $ Sección omp

section\_count = section\_count + 1

! puede imprimir el número uno o dos

print \*, 'section\_count', section\_count

! $ Sección omp

section\_count = section\_count + 1

! puede imprimir el número uno o dos

print \*, 'section\_count', section\_count

! $ secciones extremas omp

! $ Final omp sección final del programa paralelo

##### Fortran



## La única construcción

El siguiente ejemplo demuestra el único constructo. En el ejemplo, solo se imprimirá una rosca de cada uno de los mensajes de progreso. Todos los otros hilos se saltará la región única y parar en la barrera en el extremo de la única construcción hasta que todos los hilos en el equipo han llegado a la barrera. Si otros hilos pueden proceder sin esperar a que el hilo de ejecución de la región única, una cláusula nowait se puede especificar, como se hace en la tercera construcción única en este ejemplo. El usuario no debe hacer ninguna suposición en cuanto a que el hilo ejecutará una sola región.

C / C ++



*Ejemplo 14.1c*

#include <stdio.h> vacío trabajo 1 () {}

work2 void () {}

single\_example vacío ()

{

#pragma omp paralelo

{

#pragma omp sola printf ( "trabajo 1 Comenzando. \ n");

trabajo1 ();

#pragma omp sola printf ( "trabajo1 Acabado. \ n");

#pragma omp sola nowait

printf ( "Terminado trabajo 1 y work2 comenzando. \ n");

work2 ();

}

}

##### C / C ++



*Ejemplo 14.1f*



Trabajo1 SUBRUTINA () END SUBRUTINA Trabajo1

Trabajo2 SUBRUTINA ()

##### Fortran

FIN SUBRUTINA Ofic2

PROGRAMA SINGLE\_EXAMPLE

! $ OMP PARALELO

! $ OMP SOLA

print \*, "A partir del trabajo 1".

! $ OMP único extremo

LLAMADA Trabajo1 ()

! $ OMP SOLA

print \*, "Acabado trabajo 1".

! $ OMP único extremo

! $ OMP SOLA

print \*, "Terminado trabajo 1 y empezando work2."

! $ OMP único extremo NOWAIT

LLAMADA Trabajo2 ()

! PARALELO $ OMP FIN

FIN DEL PROGRAMA SINGLE\_EXAMPLE

##### Fortran



1. **Las construcciones de asignación de tareas**

El siguiente ejemplo muestra cómo recorrer una estructura en forma de árbol utilizando tareas explícitas. Tenga en cuenta que la función de desplazamiento debe ser llamado desde dentro de una región paralela para las diferentes tareas específicas a ser ejecutadas en paralelo. También tenga en cuenta que las tareas se ejecutarán en ningún orden específico porque no hay directivas de sincronización. Por lo tanto, suponiendo que el recorrido se hará en orden posterior, como en el código secuencial, es erróneo.

*Ejemplo 15.1c*



C / C ++

nodo struct {

nodo struct \* dejó; nodo struct \* derecha;

};

extern proceso void (nodo struct \*); travesía vacío (struct nodo \* p) {

si (p> izquierda)

#pragma omp // p es tarea firstprivate por transversal por omisión (p-> izquierda);

si (p> derecha)

#pragma omp // p es tarea firstprivate por transversal por omisión (p-> derecha);

proceso (p);

}

##### C / C ++



Fortran



*Ejemplo 15.1f*

traverse SUBRUTINA RECURSIVO (P)

nodo TIPO

TIPO (nodo), PUNTERO :: izquierdo, extremo derecho nodo Tipo

TIPO (Node) :: P

IF (asociado (P% izquierda)), entonces

! $ OMP ¡TAREA! P es firstprivate por defecto llamar transversal (P% izquierda)

! TAREA $ OMP FIN

TERMINARA SI

IF (asociado (P% derecha)) ENTONCES

! $ OMP ¡TAREA! P es firstprivate por defecto llamar transversal (P% derecha)

! TAREA $ OMP FIN

TERMINARA SI

proceso de llamada (P) SUBRUTINA FIN

##### Fortran



En el siguiente ejemplo, forzar un recorrido en orden posterior del árbol mediante la adición de una directiva taskwait. Ahora, podemos asumir con seguridad que los hijos izquierdo y derecho han sido ejecutadas antes de procesar el nodo actual.

*Ejemplo 15.2c*



C / C ++

nodo struct {

nodo struct \* dejó; nodo struct \* derecha;

};

extern proceso void (nodo struct \*);

postorder\_traverse vacío (struct nodo \* p) {if (p-> izquierda)

#pragma omp // p es tarea firstprivate por postorder\_traverse defecto (p-> izquierda);

si (p> derecha)

#pragma omp // p es tarea firstprivate por postorder\_traverse defecto (p-> derecha);

#pragma omp proceso taskwait (p);

}

##### C / C ++



Fortran



*Ejemplo 15.2f*

traverse SUBRUTINA RECURSIVO (P)

nodo TIPO

TIPO (nodo), PUNTERO :: izquierdo, extremo derecho nodo Tipo

TIPO (Node) :: P

IF (asociado (P% izquierda)), entonces

! $ OMP ¡TAREA! P es firstprivate por defecto llamar transversal (P% izquierda)

! $ OMP Finalizar tarea ENDIF

IF (asociado (P% derecha)) ENTONCES

! $ OMP ¡TAREA! P es firstprivate por defecto llamar transversal (P% derecha)

! $ OMP Finalizar tarea ENDIF

! $ OMP TASKWAIT

proceso de llamada (P) SUBRUTINA FIN

##### Fortran



El siguiente ejemplo muestra cómo utilizar el constructo tarea de procesar los elementos de una lista enlazada en paralelo. El hilo de ejecución de la región solo genera todas las tareas explícitas, que luego son ejecutadas por los hilos en el equipo actual. El puntero p es firstprivate por defecto en la construcción de tareas por lo que no es necesario especificar en una cláusula firstprivate.

C / C ++



*Ejemplo 15.3C*

typedef struct nodo nodo; nodo struct {

datos int; nodo \* siguiente;

};

proceso de vacío (nodo p \*)

{

/ \* Trabajo aquí \* /

}

increment\_list\_items void (\* nodo de cabeza)

{

#pragma omp paralelo

{

#pragma omp sola

{

nodo \* p = cabeza; mientras que (p) {

#pragma omp tarea

// p es firstprivate por defecto del proceso (p);

p = p-> siguiente;

}

}

}

}

##### C / C ++



Fortran



*Ejemplo 15.3f*

MÓDULO DE LISTA tipo de nodo

INTEGER :: CARGA

TIPO (NODO), PUNTERO :: SIGUIENTE

FIN tipo de nodo CONTIENE

PROCESO SUBRUTINA (p)

TIPO (nodo), el puntero P ::

! trabajo aquí FIN SUBRUTINA

INCREMENT\_LIST\_ITEMS SUBRUTINAS (HEAD) TIPO (NODO), PUNTERO :: CABEZA

TIPO (nodo), el puntero P ::

! $ OMP PARALELO PRIVADO (P)

! $ OMP SOLA

P => CABEZA DO

! $ OMP TAREA

! P es firstprivate por el proceso de llamada predeterminado (P)

! $ OMP FIN TAREA P% = SIGUIENTE> P

IF (.NOT. ASSOCIATED (P)) EXIT END DO

! $ OMP único extremo

! $ OMP FIN FIN PARALELO SUBRUTINA

MÓDULO DE FIN

##### Fortran



La función fib () debe ser llamado desde dentro de una región paralela para las diferentes tareas especificadas a ser ejecutadas en paralelo. Además, sólo un hilo de la región paralela debe llamar fib () a menos que se desean múltiples cálculos concurrentes de Fibonacci.

*Ejemplo 15.4c*



C / C ++

int fib (int n) {int i, j;

si (n <2) de retorno n;

else {

tarea #pragma omp compartido (i) i = fib (n-1);

tarea #pragma omp compartido (j) j = fib (n-2);

#pragma omp retorno taskwait i + j;

}

}

##### C / C ++



Fortran



*Ejemplo 15.4f*

RECURSIVO fib INTEGER FUNCIÓN (n) resultado (res) INTEGER n, i, j

IF (n .LT. 2), entonces

res = n ELSE

! $ OMP tarea compartida (i)

i = fib (n-1)

! TAREA $ OMP FIN

! $ OMP tarea compartida (j)

j = fib (n-2)

! TAREA $ OMP FIN

! $ OMP TASKWAIT

res = i + j END IF

FUNCIÓN DE FIN

##### Fortran



Nota: Existen algoritmos más eficientes para el cálculo de los números de Fibonacci. Este algoritmo recursión clásico es con fines ilustrativos.

El siguiente ejemplo demuestra una manera de generar un gran número de tareas con un hilo y ejecutarlos con los hilos en el equipo. Mientras que la generación de estas tareas, la aplicación puede llegar a su límite en las tareas asignadas. Si lo hace, se permite la puesta en práctica para hacer que el hilo de la ejecución del bucle de generación de tareas a suspender su tarea en el punto de programación de tareas en la directiva tarea y comenzar a ejecutar las tareas asignadas. Una vez que el número de tareas sin asignar es suficientemente baja, el hilo podría reanudar la ejecución del bucle de generación de tareas.

C / C ++



*Ejemplo 15.5c*

LARGE\_NUMBER #define 10000000

doble elemento [LARGE\_NUMBER]; extern proceso void (doble);

int main () {

#pragma omp paralelo

{

#pragma omp sola

{

int i;

for (i = 0; i <LARGE\_NUMBER; i ++)

#pragma omp // i es tarea firstprivate, Articulo proceso compartido (punto [i]);

}

}

}

##### C / C ++



Fortran



*Ejemplo 15.5f*

real \* 8 artículo (10000000) entero i

! $ OMP paralelo

! $ OMP sola! repetición del bucle variable i es Do privada i = 1,10000000

! $ Tarea omp

! i es firstprivate, el punto es compartido proceso de llamada (ítem (i))

! $ Tarea final omp

final hacer

! Solo $ final omp

! $ OMP de extremo a extremo en paralelo

##### Fortran



El siguiente ejemplo es el mismo que el anterior, excepto que las tareas se generan en una tarea condicionada. Mientras que la generación de las tareas, la aplicación puede llegar a su límite en las tareas asignadas. Si lo hace, se permite la puesta en práctica para hacer que el hilo de la ejecución del bucle de generación de tareas a suspender su tarea en el punto de programación de tareas en la directiva tarea y comenzar a ejecutar las tareas asignadas. Si ese hilo se inicia la ejecución de una tarea que lleva mucho tiempo en completarse, los otros hilos pueden completar todas las otras tareas antes de que termine.

En este caso, dado que el lazo está en una tarea no condicionado, cualquier otro hilo es elegible para reanudar el bucle de generación de tareas. En los ejemplos anteriores, los otros hilos serían obligados a ralentí hasta que el hilo de generación termina su larga tarea, ya que el bucle de generación de tarea fue en una tarea de atado.

C / C ++



*Ejemplo 15.6c*

LARGE\_NUMBER #define 10000000

doble elemento [LARGE\_NUMBER]; extern proceso void (doble); int main () {

#pragma omp paralelo

{

#pragma omp sola

{

int i;

#pragma omp tarea desatados

// i es firstprivate, el punto es compartido

{

for (i = 0; i <LARGE\_NUMBER; i ++)

#pragma omp tarea

proceso (punto [i]);

}

}

}

return 0;

}

##### C / C ++



Fortran



*Ejemplo 15.6f*

real \* 8 artículo (10000000)

! $ OMP paralelo

! $ OMP sola

! $ Tarea omp desatados

! repetición del bucle variable i es Do privada i = 1,10000000

! $ Tarea OMP! i es firstprivate, el punto es compartido proceso de llamada (ítem (i))

! $ Tarea final omp

final hacer

! $ Tarea final omp

! Solo $ final omp

! $ OMP de extremo a extremo en paralelo

##### Fortran



Los dos ejemplos siguientes demuestran cómo las reglas de programación ilustrados en $ afectan el uso de variables threadprivate en las tareas. Una variable threadprivate puede ser modificada por otra tarea que es ejecutada por el mismo hilo. Así, el valor de una variable threadprivate no se puede suponer que ser sin cambios a través de un punto de programación de tareas. En las tareas no vinculadas, los puntos de programación de tareas se pueden añadir en cualquier lugar por la aplicación.

Un cambio de tarea se puede producir en un punto de programación de tareas. Un solo hilo puede ejecutar tanto de las regiones de tareas que modifican tp. Las partes de estas regiones de la tarea en la que se modifica tp pueden ser ejecutados en cualquier orden de modo que el valor resultante de var puede ser 1 o 2.

C / C ++



*Ejemplo 15.7c*

int tp;

threadprivate #pragma omp (tp)

int var; trabajo vacío ()

{

#pragma omp tarea

{

/ \* Trabajo aquí tarea OMP \* / #pragma

{

tp = 1;

/ \* Trabajo aquí tarea OMP \* / #pragma

{

/ \* Ninguna modificación del tp \* /

}

var = tp; // valor de tp puede ser 1 o 2

}

tp = 2;

}

}

##### C / C ++



Fortran



*Ejemplo 15.7f*

módulo ejemplo número entero tp

! Threadprivate $ OMP (tp) número entero var contiene variables globales del uso del trabajo subrutina

! $ Tarea omp

! trabajo aquí

! $ Tarea omp

tp = 1

! trabajo aquí

! $ Tarea omp

! ninguna modificación de tp

! $ Tarea final omp

var = tp! valor de var puede ser 1 o 2

! $ Tarea final omp

tp = 2

! $ Tarea final omp

extremo del módulo final subrutina

##### Fortran



En este ejemplo, limitaciones de programación prohíben un hilo en el equipo de la ejecución de una tarea nueva que modifica tp mientras que otra región las tareas atado al mismo hilo se suspende. Por lo tanto, el valor escrito persistirá a través del punto de programación de tareas.

C / C ++



*Ejemplo 15.8c*

int tp;

#pragma omp threadprivate (tp) int var;

trabajo vacío ()

{

#pragma omp paralelo

{

/ \* Trabajo aquí tarea OMP \* / #pragma

{

tp ++;

/ \* Trabajo aquí tarea OMP \* / #pragma

{

/ \* Hacer el trabajo aquí, pero no modificar tp \* /

}

var = tp; // El valor no cambia después de escribir por encima de

}

}

}

##### C / C ++



Fortran



*Ejemplo 15.8f*

módulo ejemplo número entero tp

! Threadprivate $ OMP (tp) número entero var contiene el trabajo subrutina

! $ OMP paralelo

! trabajo aquí

! $ Tarea omp

tp = tp + 1

! trabajo aquí

! $ Tarea omp

! no trabajar aquí, pero no modificar tp

! $ Tarea final omp

var = tp! valor no cambia después de la escritura encima

! $ Tarea final omp

! $ Final omp paralelo

extremo del módulo final subrutina

##### Fortran



Los dos ejemplos siguientes demuestran cómo las reglas de programación ilustrados en $ afectan el uso de cerraduras y las secciones críticas en las tareas. Si un bloqueo se mantiene a través de un punto de programación de tareas, debe hacerse ningún intento de adquirir la misma cerradura en cualquier código que pueden ser intercalados. De lo contrario, es un callejón sin salida posible.

En el siguiente ejemplo, supongamos que el hilo de ejecución de tarea 1 prorroga tarea 2. Cuando se encuentra el punto de programación de tareas en la tarea 3, se podría suspender la tarea 1 y comenzar la tarea 2 que resultará en un punto muerto cuando se intenta entrar región crítica 1.

C / C ++



*Ejemplo 15.9c*

trabajo vacío ()

{

#pragma omp tarea

{ //Tarea 1

#pragma omp tarea

{ //Tarea 2

#pragma omp crítico // región crítica 1

{/ \* Hacer el trabajo aquí \* /}

}

#pragma omp crítico // región crítica 2

{

// Captura de datos para la siguiente tarea #pragma omp tarea

{/ \* Hacer el trabajo aquí \* /} // Tarea 3

}

}

}

##### C / C ++



Fortran



*Ejemplo 15.9f*

ejemplo módulo contiene trabajo subrutina

! $ Tarea omp

! Tarea 1

! $ Tarea omp

! Tarea 2

! $ OMP crítica

! región crítica 1

! trabajo aquí

! $ OMP finalidad crítica

! $ Tarea final omp

! $ OMP crítica

! región crítica 2

! Capture datos para la siguiente tarea

! $ Tarea omp

! Tarea 3

! trabajo aquí

! $ Tarea final omp

! $ OMP finalidad crítica

! $ Tarea final omp

extremo del módulo final subrutina

##### Fortran



En el siguiente ejemplo, el bloqueo se lleva a cabo a través de un punto de programación de tareas. Sin embargo, de acuerdo con las restricciones de programación, el hilo de ejecución no puede comenzar a ejecutar una de las tareas que no son descendientes que también adquiere bloqueo antes de la región tarea se ha completado. Por lo tanto, no es posible estancamiento.

C / C ++



*Ejemplo 15.10c*

# include <omp.h> trabajo void () {

cerradura omp\_lock\_t; omp\_init\_lock (y la cerradura);

#pragma omp paralelo

{

int i; #pragma omp para

for (i = 0; i <100; i ++) {tarea omp #pragma

{

// bloqueo es compartida por defecto en el omp\_set\_lock tarea (y bloquear);

// Captura de datos para la siguiente tarea #pragma omp tarea

// Programación de tareas del punto 1

{/ \* Hacer el trabajo aquí \* /} omp\_unset\_lock (y bloquear);

}

}

}

omp\_destroy\_lock (y la cerradura);

}

##### C / C ++



Fortran



*Ejemplo 15.10f*

ejemplo módulo incluyen 'omp\_lib.h'

número entero (tipo = omp\_lock\_kind) número entero de bloqueo i

contiene el trabajo subrutina

llamar omp\_init\_lock (bloqueo)

! $ OMP paralelo

! $ OMP hacer

hacer i = 1100

! $ Tarea omp

! tarea externa

llamada omp\_set\_lock (bloqueo)! cerradura se comparte por

! defecto en la tarea

! Capture datos para la siguiente tarea

! $ OMP ¡tarea! Programación de tareas Point 1

! trabajo aquí

! $ Tarea final omp

llamar omp\_unset\_lock (bloqueo)

! $ OMP extremo final tarea hacer

! $ Final omp paralelo

llamar omp\_destroy\_lock (bloqueo) subrutina final

módulo final

##### Fortran



Los siguientes ejemplos ilustran el uso de la cláusula fusionable en el constructo tarea. En este primer ejemplo, el constructo de tarea se ha anotado con la cláusula fusionables. La incorporación de esta cláusula permite la puesta en práctica de reutilizar el entorno de datos (incluyendo el ICVs) de la tarea principal de la tarea dentro de foo si la tarea se incluye o undeferred. Por lo tanto, el resultado de la ejecución puede ser diferente dependiendo de si la tarea se fusiona o no. Por lo tanto la cláusula fusionables necesita ser utilizado con precaución. En este ejemplo, el uso de la cláusula fusionables es seguro. Como x es una variable compartida el resultado no depende de si la tarea se fusionó (es decir, la tarea siempre se incrementará la misma variable y siempre computar el mismo valor de x).

C / C ++



*Ejemplo 15.11c*

#include <stdio.h> void foo ()

{

int x = 2;

#pragma omp tarea compartida (x) fusionables

{

x ++;

}

#pragma omp taskwait printf ( "% d \ n", x); // imprime 3

}



##### C / C ++

*Ejemplo 15.11f*



foo subrutina () número entero :: x

##### Fortran

x = 2

! $ OMP tarea compartida (x) fusionables x = x + 1

! $ Tarea final omp

! $ OMP taskwait

impresión \*, ¡X! huellas dactilares 3 subrutina final

##### Fortran



Este segundo ejemplo muestra un uso incorrecto de la cláusula fusionables. En este ejemplo, la tarea creada será acceder a las diferentes instancias de la variable x si la tarea no se fusiona, cuando x es firstprivate, pero va a tener acceso a los mismos variable x si se combina la tarea. Como resultado, el comportamiento del programa es indeterminado y puede imprimir dos valores diferentes de x en función de las decisiones tomadas por la implementación.

C / C ++



*Ejemplo 15.12c*

#include <stdio.h> void foo ()

{

int x = 2;

#pragma omp tarea fusionables

{

x ++;

}

#pragma omp taskwait printf ( "% d \ n", x); // imprime 2 o 3

}

##### C / C ++



*Ejemplo 15.12f*

foo subrutina () número entero :: x

x = 2

! $ OMP tarea fusionables x = x + 1

! $ Tarea final omp

! $ OMP taskwait

impresión \*, ¡X! 2 o impresiones 3 subrutina final

##### Fortran

Fortran



El siguiente ejemplo muestra el uso de la cláusula final y la llamada a la API omp\_in\_final en un programa de búsqueda binaria recursiva. Para reducir la sobrecarga, una vez que una cierta profundidad de recursión se alcanza el programa utiliza la cláusula final para crear tareas sólo incluidos, que permiten optimizaciones adicionales.

El uso de la llamada a la API permite a los programadores omp\_in\_final optimizar su código especificando qué partes del programa no son necesarios cuando una tarea puede crear tareas sólo incluidos (es decir, el código está dentro de una tarea final). En este ejemplo, el uso de una variable de estado diferente no es necesario para que una vez que el programa llega a la parte del cálculo que se ha finalizado y la copia del estado padre hacia el nuevo estado se elimina. La asignación de new\_state en la pila también podría ser evitado, pero que haría este ejemplo tan claro. La cláusula final es más eficaz cuando se utiliza junto con la cláusula fusionarse ya que todas las tareas creadas en una región tarea final se incluyen las tareas que se pueden combinar si la cláusula fusionables está presente.

*Ejemplo 15.13c*



C / C ++

#include <string.h> #include <omp.h>

#define LIMIT 3 / \* límite arbitrario en la recursión profundidad \* / void check\_solution (char \*);

bin\_search vacío (int pos, int n, char \* Estado)

{

si (pos == n) {check\_solution (estado); regreso;

}

#pragma omp última tarea (pos> LIMIT) fusionables

{

Char new\_state [n];

Si {memcpy (new\_state, estado, pos) (omp\_in\_final ()!); estado = new\_state;

}

estado [pos] = 0; bin\_search (pos + 1, n, Estado);

}

#pragma omp última tarea (pos> LIMIT) fusionables

{

Char new\_state [n];

Si {memcpy (new\_state, estado, pos) (omp\_in\_final ()!); estado = new\_state;

}

estado [pos] = 1; bin\_search (pos + 1, n, Estado);

}

#pragma omp taskwait

}

##### C / C ++



Fortran



*Ejemplo 15.13f*

recursiva subrutina bin\_search (pos, n, estado) utilizan omp\_lib

enteros :: pos, n

carácter, puntero :: Estado (:)

carácter, objetivo, dimensión (n) :: new\_state1, número entero new\_state2, parámetro :: LIMIT = 3

si (.EQ pos. n) entonces

check\_solution llamada (estado) de retorno

terminara si

! $ OMP última tarea (pos> LIMIT) fusionarse si (.no. Omp\_in\_final ()) entonces

new\_state1 (1: pos) = estado (1: pos) estado => new\_state1

terminara si

estatales (pos + 1) = 'z'

llamar bin\_search (pos + 1, n, estado)

! $ Tarea final omp

! $ OMP última tarea (pos> LIMIT) fusionarse si (.no. Omp\_in\_final ()) entonces

new\_state2 (1: pos) = estado (1: pos) estado => new\_state2

terminara si

estatales (pos + 1) = 'y'

llamar bin\_search (pos + 1, n, estado)

! $ Tarea final omp

! $ OMP subrutina final taskwait

##### Fortran



El siguiente ejemplo ilustra la diferencia entre el Si y las cláusulas finales. El si cláusula tiene un efecto local. En el primer nido de tareas, el que tiene el si se undeferred cláusula pero la tarea anidado dentro de esa tarea no se verá afectado por la cláusula y si se creará como de costumbre. Alternativamente, la cláusula final afecta a todas las construcciones de tareas en la región tarea final, pero no la tarea final en sí. En el segundo nido de tareas, las tareas anidadas serán creados como tareas incluidas. Tenga en cuenta también que las condiciones para el caso y cláusulas finales son por lo general lo contrario.

C / C ++



*Ejemplo 15.14c*

void foo ()

{

int i;

#pragma omp tarea si (0) // Esta tarea es undeferred

{

#pragma omp // tarea Esta tarea es un habitual tarea for (i = 0; i <3; i ++) {

#pragma omp // tarea Esta tarea es un habitual tarea bar();

}

}

última tarea omp #pragma (1) // Esta tarea es una tarea periódica

{

tarea omp #pragma // Esta tarea se incluye para (i = 0; i <3; i ++) {

#pragma omp // tarea Esta tarea es también incluido bar();

}

}

}

##### C / C ++



Fortran



*Ejemplo 15.14f*

foo subrutina () i número entero

! $ OMP tarea si (.FALSE.)! Esta tarea se undeferred

! $ OMP ¡tarea! Esta tarea es un habitual tarea hacer i = 1, 3

! $ OMP ¡tarea! Esta tarea es un habitual tarea llamar bar ()

! $ OMP final enddo tarea

! $ Tarea final omp

! $ Tarea final omp

! $ Última tarea OMP (.TRUE.)! Esta tarea es una tarea periódica

! $ OMP ¡tarea! Esta tarea es Incluyó ¿i = 1, 3

! $ OMP ¡tarea! Esta tarea es también prohibición de llamadas incluidos ()

! $ OMP final enddo tarea

! $ Tarea final omp

! $ OMP extremo final subrutina tarea

##### Fortran



**Dependencias de tareas**

**La dependencia fluir**

En este ejemplo se muestra una dependencia simple flujo expresa utilizando la cláusula dependerá de la construcción de tarea.

C / C ++



*Ejemplo 15.15c*

#include <stdio.h> int main ()

{

int x = 1;

#pragma omp omp #pragma paralelo sola

{

tarea #pragma omp compartido (x) dependerá (out: x) x = 2;

tarea #pragma omp compartido (x) depende (en: x) printf ( "x =% d \ n", x);

}

return 0;

##### C / C ++



Fortran



*Ejemplo 15.15f*

programa de ejemplo número entero :: xx = 1

! $ OMP paralelo

! $ OMP sola

! $ OMP tarea compartida (x) dependerá (Salida: x) = x 2

! $ Tarea final omp

! $ OMP tarea compartida (x) depende (en: x) de impresión \*, "x =", x

! $ Tarea final omp

! Solo $ final omp

! $ OMP extremo final del programa paralelo

##### Fortran



El programa imprimirá siempre "x = 2", debido a que las cláusulas dependen hacer cumplir el orden de las tareas. Si se han omitido las cláusulas dependerá, a continuación, las tareas podrían ejecutar en cualquier orden y el programa y el programa tendrían una condición de carrera.

**Anti-dependencia**

En este ejemplo se muestra un anti-dependencia expresa utilizando la cláusula dependen de la

**tarea** construir.

##### C / C ++



*Ejemplo 15.16c*

#include <stdio.h> int main ()

{

int x = 1;

#pragma omp omp #pragma paralelo sola

{

tarea #pragma omp compartido (x) depende (en: x) printf ( "x =% d \ n", x);

tarea #pragma omp compartido (x) dependerá (out: x) x = 2;

}

return 0;

}

##### C / C ++



Fortran



*Ejemplo 15.16f*

programa de ejemplo número entero :: xx = 1

! $ OMP paralelo

! $ OMP sola

! $ OMP tarea compartida (x) depende (en: x) de impresión \*, "x =", x

! $ Tarea final omp

! $ OMP tarea compartida (x) dependerá (Salida: x) = x 2

! $ Tarea final omp

! Solo $ final omp

! $ OMP extremo final del programa paralelo

##### Fortran



El programa imprimirá siempre "x = 1", debido a que las cláusulas dependen hacer cumplir el orden de las tareas. Si se han omitido las cláusulas dependerá, a continuación, las tareas podrían ejecutar en cualquier orden y el programa tendrían una condición de carrera.

**La dependencia de salida**

En este ejemplo se muestra una dependencia de salida expresa utilizando la cláusula dependerá de la construcción de tarea.

C / C ++



*Ejemplo 15.17c*

#include <stdio.h> int main ()

{

int x;

#pragma omp omp #pragma paralelo sola

{

#pragma omp tarea compartida (x) dependerá (Salida: x)

x = 1;

tarea #pragma omp compartido (x) dependerá (out: x) x = 2;

#pragma omp taskwait printf ( "x =% d \ n", x);

}

return 0;

}

##### C / C ++



Fortran



*Ejemplo 15.17f*

programa de ejemplo número entero :: x

! $ OMP paralelo

! $ OMP sola

! $ OMP tarea compartida (x) dependerá (Salida: x) x = 1

! $ Tarea final omp

! $ OMP tarea compartida (x) dependerá (Salida: x) = x 2

! $ Tarea final omp

! $ OMP de impresión taskwait \*, "x =", x

! Solo $ final omp

! $ OMP extremo final del programa paralelo

##### Fortran



El programa imprimirá siempre "x = 2", debido a que las cláusulas dependen hacer cumplir el orden de las tareas. Si se han omitido las cláusulas dependerá, a continuación, las tareas podrían ejecutar en cualquier orden y el programa tendrían una condición de carrera.

**La ejecución concurrente con Dependencias**

En este ejemplo se muestra la ejecución potencialmente concurrente de tareas utilizando múltiples dependencias en el flujo expresa utilizando la cláusula dependerá de la construcción de tarea.

C / C ++



*Ejemplo 15.18c*

#include <stdio.h> int main ()

{

int x = 1;

#pragma omp omp #pragma paralelo sola

{

tarea #pragma omp compartido (x) dependerá (out: x) x = 2;

tarea #pragma omp compartido (x) depende (en: x) printf; (, x + 1 "x + 1 =% d.")

tarea #pragma omp compartido (x) depende (en: x) printf ( "x + 2 =% d \ n", x + 2);

}

return 0;

}

##### C / C ++



Fortran



*Ejemplo 15.18f*

programa de ejemplo número entero :: xx = 1

! $ OMP paralelo

! $ OMP sola

! $ OMP tarea compartida (x) dependerá (Salida: x) = x 2

! $ Tarea final omp

! $ OMP tarea compartida (x) depende (en: x) de impresión \*, "x + 1 =", x + 1 ""

! $ Tarea final omp

! $ OMP tarea compartida (x) depende (en: x) de impresión \*, "x + 2 =", x + 2, ""

! $ Tarea final omp

! Solo $ final omp

! $ OMP extremo final del programa paralelo

##### Fortran



Las dos últimas tareas dependen de la primera tarea. Sin embargo no hay dependencia entre los dos últimos tareas, que pueden ejecutar en cualquier orden (o concurrentemente si más de un hilo está disponible). Por lo tanto, las salidas posibles son "x + 1 = 3. x + 2 =

4. "y "x + 2 = 4. x + 1 = 3.". Si las cláusulas dependerá habían omitido,

entonces todas las tareas podría ejecutar en cualquier orden y el programa tendría una condición de carrera.

**La multiplicación de matrices**

Este ejemplo muestra una multiplicación de matrices bloqueado basado en tareas. Las matrices son de elementos de N x N, y la multiplicación se realiza utilizando bloques de elementos BSxBS.

*Ejemplo 15.19c*



C / C ++

// Supongamos BS divide N perfectamente

matmul\_depend void (int N, int BS, flotador A [N] [N], flotador B [N] [N], flotador C [N] [N])

{

int i, j, k, ii, jj, kk; for (i = 0; i <N; i + = BS) {

para (j = 0; j <N; j + = BS) {for (k = 0; k <N; k + = BS) {

tarea #pragma omp depende (en: A [i: BS] [k: BS], B [k: BS] [j: BS]) \ dependerá (inout: C [i: BS] [j: BS])

para (ii = i; ii <i + BS; ii ++) para (jj = j; jj <j + BS; jj ++)

para (k kk =; kk <k + BS; kk ++)

C [ii] [jj] = C [ii] [jj] + A [ii] [kk] \* B [kk] [jj];

}

}

}

}

##### C / C ++



Fortran



*Ejemplo 15.19f*

! Supongamos BS divide N matmul\_depend perfectamente subrutina (N, BS, A, B, C)

número entero :: N, BS, BM

real, dimensión (N, N) :: A, B, C

número entero :: i, j, k, ii, jj, kk BM = BS -1

hacer i = 1, N, BS

hacer j = 1, N, BS

hacer k = 1, N, BS

! $ OMP tarea depende (en: A (i: i + BM, k: k + BM), B (k: k + BM, j: j + BM)) y

! $ OMP dependen (inout: C (i: i + BM, j: j + BM)) hacer ii = i, i + BS

hacer jj = j, j + BS

hacer kk = k, k + BS

C (jj, ii) = C (jj, ii) + A (kk, ii) \* B (jj, kk) final no

final terminan hacer

! $ Tarea final omp

final terminan hacer

final hacer

subrutina final

##### Fortran



## La Directiva taskyield

El siguiente ejemplo ilustra el uso de la directiva taskyield. Las tareas en el ejemplo computan algo útil y luego hacer algún cálculo que se debe hacer en una región crítica. Mediante el uso de taskyield cuando una tarea no puede obtener acceso a la región crítica de la aplicación puede suspender la tarea actual y programar alguna otra tarea que puede hacer algo útil.

C / C ++



*Ejemplo 16.1c*

# include <omp.h>

something\_useful void (void); something\_critical void (void);

void foo (omp\_lock\_t \* cerradura, int n)

{

int i;

for (i = 0; i <n; i ++) tarea #pragma omp

{

algo útil();

while (! omp\_test\_lock (bloqueo)) {#pragma omp taskyield

}

something\_critical (); omp\_unset\_lock (bloqueo);

}

}

##### C / C ++



Fortran



*Ejemplo 16.1f*

foo subrutina (bloqueo, n) utilizar omp\_lib

número entero (tipo = omp\_lock\_kind) :: cerradura número entero n

i número entero

hacer i = 1, n

! $ Tarea omp

llamar something\_useful ()

hacer mientras (.no. omp\_test\_lock (bloqueo))

FIN $ OMP taskyield hacer

llamar something\_critical llamada omp\_unset\_lock () (cerrar)

! $ OMP extremo final tarea hacer

subrutina final

##### Fortran



Fortran



1. **los compartir el trabajo Construir**

Los siguientes son ejemplos de la construcción de Workshare.

En el siguiente ejemplo, se extiende Workshare trabajo a través de los hilos de ejecución de la región paralela, y hay una barrera después de la última declaración. Implementaciones deben cumplir las reglas de ejecución Fortran en el interior del bloque de trabajo compartido.

*Ejemplo 17.1f*

SUBRUTINA WSHARE1 (AA, BB, CC, DD, EE, FF, N) INTEGER N

AA real (N, N), BB (N, N), CC (N, N), DD (N, N), EE (N, N), FF (N, N)

! $ OMP PARALELO

! $ OMP Workshare AA = CC = BB DD EE FF =

! $ OMP FIN Workshare

! PARALELO $ OMP FIN

FIN SUBRUTINA WSHARE1

En el siguiente ejemplo, la barrera en el extremo de la primera región Workshare se elimina con una cláusula nowait. Hilos haciendo CC = DD comienzan inmediatamente a trabajar en EE = FF cuando se hacen con CC = DD.

Fortran (cont.)

*Ejemplo 17.2f*

SUBRUTINA WSHARE2 (AA, BB, CC, DD, EE, FF, N) INTEGER N

AA real (N, N), BB (N, N), CC (N, N)

VERDADERO DD (N, N), EE (N, N), FF (N, N)

! $ OMP PARALELO

! $ OMP Workshare AA BB CC = = DD

! $ OMPEND Workshare NO, ESPERA

! $ OMPWORKSHARE EE = FF

! $ OMPEND COMPARTIR EL TRABAJO

! $ OMPEND PARALELA

FIN SUBRUTINA WSHARE2

El siguiente ejemplo muestra el uso de una directiva atómica dentro de un Workshare

construir. El cómputo de SUM (AA) se trabajo compartido, pero la actualización a R es atómica.

*Ejemplo 17.3f*

SUBRUTINA WSHARE3 (AA, BB, CC, DD, N) INTEGER N

AA real (N, N), BB (N, N), CC (N, N), DD (N, N) real r

R = 0

! $ OMPPARALLEL

! $ OMPWORKSHARE AA = cama y desayuno

! $ OMPATOMIC ACTUALIZAR

R = R + SUM (AA) CC = DD

! $ OMPEND COMPARTIR EL TRABAJO

! $ OMPEND PARALELA

FIN SUBRUTINA WSHARE3

Fortran WHERE y FORALL declaraciones son declaraciones compuestas, formadas por una parte de control y una parte de la declaración. Cuando trabajo compartido se aplica a uno de estos estados compuestos, tanto de control como las partes de los estados son de trabajo compartido. El siguiente ejemplo muestra el uso de una declaración donde en un constructo Workshare.

Cada tarea se trabajó en el fin de los hilos:

**AA = BB entonces CC = DD entonces .NE EE. 0entonces**

**FF = 1 / EE entonces**

**GG = HH**

Fortran (cont.)

*Ejemplo 17.4f*

SUBRUTINA WSHARE4 (AA, BB, CC, DD, EE, FF, GG, HH, N) INTEGER N

AA real (N, N), BB (N, N), CC (N, N)

VERDADERO DD (N, N), EE (N, N), FF (N, N) GG real (N, N), HH (N, N)

! $ OMP PARALELO

! $ OMP Workshare AA BB CC = = DD

DONDE (EE .NE. 0) FF = 1 / EE GG = HH

! $ OMP FIN Workshare

! PARALELO $ OMP FIN

FIN SUBRUTINA WSHARE4

En el siguiente ejemplo, una asignación a una variable escalar compartida se realiza por un hilo en una Workshare mientras que todos los otros hilos en la espera de equipo.

*Ejemplo 17.5f*

SUBRUTINA WSHARE5 (AA, BB, CC, DD, N) INTEGER N

AA real (N, N), BB (N, N), CC (N, N), DD (N, N) INTEGER SHR

! PARALELO $ OMP COMPARTIDA (SHR)

! $ OMP Workshare AA BB = SHR = 1

CC = DD \* SHR

! $ OMP FIN Workshare

! PARALELO $ OMP FIN

FIN SUBRUTINA WSHARE5

El ejemplo siguiente contiene una asignación a una variable escalar privada, que se realiza por un hilo en una Workshare mientras que todos los otros hilos esperar. Es disconforme porque la variable escalar privada no está definido después de la instrucción de asignación.

*Ejemplo 17.6f*

SUBRUTINA WSHARE6\_WRONG (AA, BB, CC, DD, N) INTEGER N

AA real (N, N), BB (N, N), CC (N, N), DD (N, N) INTEGER PRI

! $ OMP PARALELO Privado (PRI)

! $ OMP Workshare AA = BB PRI = 1

CC = DD \* PRI

! $ OMP FIN Workshare

! PARALELO $ OMP FIN

FIN SUBRUTINA WSHARE6\_WRONG

reglas de ejecución Fortran deben aplicarse dentro de un constructo Workshare. En el ejemplo siguiente, el mismo resultado se produce en el siguiente fragmento de programa, independientemente de si el código se ejecuta secuencialmente o dentro de un programa de OpenMP con múltiples hilos:

*Ejemplo 17.7f*

SUBRUTINA WSHARE7 (AA, BB, CC, N) INTEGER N

AA real (N), BB (N), CC (N)

! $ OMPPARALLEL

! $ OMPWORKSHARE

AA (1:50) = BB (11:60)

CC (11:20) = AA (1:10)

! $ OMPEND COMPARTIR EL TRABAJO

! $ OMPEND PARALELA

FIN SUBRUTINA WSHARE7

##### Fortran



## La construcción principal

El siguiente ejemplo demuestra el constructo maestro. En el ejemplo, el maestro mantiene un registro de cuántas iteraciones se han ejecutado e imprime un informe de avance. Los otros hilos omitir la región maestro sin esperar.

C / C ++



*Ejemplo 18.1c*

#include <stdio.h>

promedio flotador externo (float, float, float);

master\_example void (float \* x, flotar \* xOld, int n, flotador tol)

{

int c, i, toobig; error flotador, y;

c = 0;

#pragma omp paralelo

{

hacer{

omp #pragma para privado (i) para (i = 1; i <n-1; ++ i) {

xOld [i] = x [i];

}

#pragma omp sola

{

toobig = 0;

}

omp #pragma para privado (i, y, error) reducción (+: toobig) para (i = 1; i <n-1; ++ i) {

y = x [i];

x [i] = promedio (xOld [i-1], x [i], xOld [i + 1]); error = y - x [i];

si (error> tol || error <tol) ++ toobig;

}

#pragma omp maestro

{

++ c;

printf ( "iteración% d, toobig =% d \ n", c, toobig);

}

} While (toobig> 0);

}

}

##### C / C ++



Fortran



*Ejemplo 18.1f*

SUBRUTINA MASTER\_EXAMPLE (X, xOld, N, TOL) real x (\*), xOld (\*), TOL

INTEGER N

INTEGER C, I, TOOBIG ERROR REAL, Y, MEDIA MEDIA EXTERNO

C = 0

TOOBIG = 1

! $ OMP PARALELO

DO WHILE (TOOBIG> 0)

! $ OMPDO PRIVADO (I) DO I = 2, N-1

XOld (I) = X (I) ENDDO

! $ OMPSINGLE TOOBIG = 0

! $ OMPEND SOLTERO

! $ OMPDO PRIVADO (I, Y, ERROR), Reducción (+: TOOBIG) DO I = 2, N-1

Y = X (I)

X (I) = PROMEDIO (xOld (I-1), X (I), xOld (I + 1)) ERROR = YX (I)

IF (ERROR> TOL .OR. ERROR <Tol-) TOOBIG = TOOBIG + 1 ENDDO

! $ OMPMASTER

C = C + 1

PRINT \*, 'Iteration', C 'TOOBIG =', TOOBIG

! $ OMPEND DOMINAR ENDDO

! PARALELO $ OMP FIN

FIN SUBRUTINA MASTER\_EXAMPLE

##### Fortran



## El constructo crítico

El ejemplo siguiente incluye varias construcciones críticos. El ejemplo ilustra un modelo de espera en el cual se quita de la cola una tarea y trabajando. Para protegerse de múltiples hilos desencola la misma tarea, la operación desencola debe estar en una región crítica. Debido a que las dos colas en este ejemplo son independientes, que están protegidos por construcciones críticos con diferentes nombres, eje x y eje Y.

C / C ++



*Ejemplo 19.1c*

int dequeue (float \* a);

trabajo vacío (int i, float \* a);

critical\_example vacío (float \* x, float y \*)

{

int ix\_next, iy\_next;

#pragma omp paralelo compartido (x, y) privada (ix\_next, iy\_next)

{

#pragma omp crítico (eje x) ix\_next = dequeue (x);

trabajo (ix\_next, x);

#pragma omp crítico (eje Y) iy\_next = dequeue (y);

trabajo (iy\_next, y);

}

}

##### C / C ++



Fortran



*Ejemplo 19.1f*

SUBRUTINA CRITICAL\_EXAMPLE (X, Y)

Real x (\*), Y (\*)

INTEGER IX\_NEXT, IY\_NEXT

! PARALELO $ OMP compartido (X, Y) PRIVADO (IX\_NEXT, IY\_NEXT)

! $ OMP crítico (XAXIS)

LLAMADA quitar de la cola (IX\_NEXT, X)

! $ OMP FIN crítico (XAXIS)

Identificador de llamada (IX\_NEXT, X)

! $ OMP crítico (eje Y)

LLAMADA quitar de la cola (IY\_NEXT, Y)

! $ OMP FIN crítico (eje Y)

Identificador de llamada (IY\_NEXT, Y)

! PARALELO $ OMP FIN

FIN SUBRUTINA CRITICAL\_EXAMPLE

##### Fortran



1. **trabajo compartido Dentro de un constructos**

**Construir crítica**

El siguiente ejemplo demuestra el uso de un constructo de trabajo compartido dentro de un constructo crítico. Este ejemplo es conforme porque la región única de trabajo compartido no está estrechamente anidado dentro de la región crítica. Un solo hilo ejecuta el uno y sólo la sección en la región de las secciones, y ejecuta la región crítica. El mismo hilo se encuentra con la región paralela anidada, crea un nuevo equipo de hilos, y se convierte en el maestro del nuevo equipo. Uno de los hilos en el nuevo equipo entra en la región única y en incrementos de i en 1. Al final de este ejemplo i es igual a 2.

C / C ++



*Ejemplo 20.1c*

critical\_work vacío ()

{

int i = 1;

#pragma omp secciones paralelas

{

sección #pragma omp

{

#pragma omp crítico (nombre)

{

#pragma omp paralelo

{

#pragma omp sola

{

i ++;

}

}

}

}

}

}

##### C / C ++



*Ejemplo 20.1f*



SUBRUTINA CRITICAL\_WORK () entero i

I = 1

##### Fortran

! $ OMPPARALLEL SECCIONES

! $ OMPSECTION

! $ OMPCRITICAL (NOMBRE)

! $ OMPPARALLEL

! $ OMPSINGLE

I = i + 1

! $ OMPEND SOLTERO

! $ OMPEND PARALELA

! $ OMPEND CRÍTICO (NOMBRE)

! $ OMPEND PARALELO SECCIONES

FIN SUBRUTINA CRITICAL\_WORK

##### Fortran



## Unión de las Regiones de barrera

Las normas de obligado cumplimiento exigen una región de barrera para unirse a la envolvente más cercano

**paralela** región.

En el siguiente ejemplo, la llamada desde el programa principal a Sub2 está conforme porque la región de barrera (en SUB3) se une a la región paralelo en sub2. La llamada desde el programa principal a SUB1 está conforme porque la región de barrera une a la región paralelo en sub2 subrutina.

La llamada desde el programa principal a SUB3 está conforme porque la región de barrera se une a la región paralela inactivo implícito que encierra la parte secuencial. También tenga en cuenta que la región de barrera en SUB3 cuando se llama desde sub2 sólo sincroniza el equipo de hilos paralelos en la región que rodea y no todos los hilos creados en SUB1.

C / C ++



*Ejemplo 21.1c*

trabajo vacío (int n) {} SUB3 vacío (int n)

{

trabajo (n);

#pragma omp trabajo de barrera (n);

}

void sub2 (int k)

{

compartido (k) SUB3 #pragma omp paralelo (k);

}

void SUB1 (int n)

{

int i;

#pragma omp paralelo privada (i) compartida (n)

{

#pragma omp para

for (i = 0; i <n; i ++) sub2 (i);

}

}

int main ()

{

SUB1 (2);

sub2 (2);

SUB3 (2); return 0;

}

##### C / C ++



*Ejemplo 21.1f*



TRABAJO SUBRUTINA (N) INTEGER N

FIN DE TRABAJO SUBRUTINA

SUB3 SUBRUTINA (N) INTEGER N

Identificador de llamada (N)

! $ OMPBARRIER

Identificador de llamada (N)

##### Fortran

FIN SUBRUTINA SUB3

SUB2 SUBRUTINA (K) INTEGER K

! $ OMPPARALLEL Compartido (K)

LLAMADA SUB3 (K)

! $ OMPEND PARALELA

FIN SUBRUTINA SUB2

SUB1 SUBRUTINA (N) INTEGER N

entero i

! $ OMPPARALLEL PRIVADO (I) Compartido (N)

! $ OMPDO

DO I = 1, N LLAMADA SUB2 (I)

FIN DO

! $ OMPEND PARALELA

FIN SUBRUTINA SUB1

Programa de ejemplo LLAMADA SUB1 (2) LLAMADA SUB2 (2) SUB3 CALL (2)

Ejemplo de programa END

##### Fortran



## La construcción atómica

El siguiente ejemplo evita condiciones de carrera (actualizaciones simultáneas de un elemento de x

por múltiples hilos) usando el constructo atómica.

La ventaja de utilizar el constructo atómica en este ejemplo es que permite actualizaciones de dos elementos diferentes de x que se produzca en paralelo. Si se utilizara un constructo crítico en lugar, a continuación, todos los cambios a los elementos de x serían ejecutados en serie (aunque no en cualquier orden garantizado).

Tenga en cuenta que la directiva atómica se aplica sólo a la declaración inmediatamente después de ella. Como resultado, los elementos de y no se actualizan atómicamente en este ejemplo.

C / C ++



*Ejemplo 22.1c*

flotar trabajo 1 (int i)

{

volver 1.0 \* i;

}

flotar work2 (int i)

{

volver 2.0 \* i;

}

atomic\_example void (float \* x, float \* y, int índice \*, int n)

{

int i;

#pragma omp paralelo para compartido (x, y, índice, n) para (i = 0; i <n; i ++) {

#pragma omp atómica actualización x [índice [i]] + = trabajo1 (i);

y [i] + = work2 (i);

}

}

int main ()

{

flotar x [1000]; flotar y [10000]; int index [10000]; int i;

for (i = 0; i <10000; i ++) {

índice [i] = i% 1000; y [i] = 0,0;

}

for (i = 0; i <1,000; i ++) x [i] = 0,0;

atomic\_example (x, y, índice, 10000); return 0;

}

##### C / C ++



Fortran



*Ejemplo 22.1f*

VERDADERO Trabajo1 FUNCIÓN (I) entero i

Trabajo1 = 1,0 \* I RETURN

FIN DE FUNCIONES Trabajo1

VERDADERO Ofic2 FUNCIÓN (I) entero i

Trabajo2 = 2,0 \* I RETURN

FIN DE FUNCIONES Ofic2

SUB SUBRUTINA (X, Y, INDEX, N) real x (\*), Y (\*)

INTEGER ÍNDICE (\*), N INTEGER I

! $ OMPPARALLEL DO compartido (X, Y, INDEX, NORTE) DO I = 1, N

! $ OMPATOMIC ACTUALIZAR

X (índice (i)) = X (índice (I)) + Trabajo1 (I) Y (I) = Y (I) + Trabajo2 (I)

ENDDO

FIN DEL PROGRAMA SUB SUBRUTINA ATOMIC\_EXAMPLE

X real (1000), Y (10000)

Índice entero (10000) entero i

DO I = 1,10000

Índice (I) = MOD (I, 1000) + 1

Y (i) = 0,0 ENDDO

DO I = 1,1000 X (i) = 0,0

ENDDO

SUB CALL (X, Y, INDEX, 10000)

FIN DEL PROGRAMA ATOMIC\_EXAMPLE

##### Fortran



El siguiente ejemplo ilustra las cláusulas de lectura y escritura para la directiva atómica. Estas cláusulas garantizan que la variable dada se lee o escribe, respectivamente, en su conjunto. De lo contrario, algún otro hilo podría leer o escribir parte de la variable, mientras que el hilo actual estaba leyendo o escribiendo otra parte de la variable. Tenga en cuenta que la mayoría del hardware proporciona atómica lee y escribe para un conjunto de variables correctamente alineadas de tamaños específicos, pero no necesariamente para todos los tipos de variables soportados por la API OpenMP.

C / C ++



*Ejemplo 22.2C*

int atomic\_read (const int \* p)

{

valor int;

/ \* Garantizar que todo el valor de p \* se lee de forma atómica. Ninguna parte de

\* \* P puede cambiar durante la operación de lectura.

\* /

#pragma omp valor leído atómica = \* p;

valor de retorno;

}

atomic\_write vacío (int \* p, int value)

{

/ \* Garantizar que el valor se almacena en forma atómica \* p. Ninguna parte de \* p puede cambiar

\* Hasta después de que se haya completado toda la operación de escritura.

\* /

#pragma omp escritura atómica

\* P = valor;

}

##### C / C ++



Fortran



*Ejemplo 22.2f*

función atomic\_read (p)

número entero :: atomic\_read número entero, la intención (in) :: p

! Garantiza que todo el valor de p se lee de forma atómica. Ninguna parte de

! p puede cambiar durante la operación de lectura.

! $ OMP lectura atómica

atomic\_read = p retorno

función atomic\_read final

atomic\_write subrutina (p, valor) número entero, la intención (out) :: p número entero, la intención (in) :: valor

! Garantiza que el valor se almacena en forma atómica p. Ninguna parte de p puede cambiar

! hasta después de que se haya completado toda la operación de escritura.

! $ OMP de escritura atómica

p = valor

atomic\_write subrutina final

##### Fortran



El siguiente ejemplo ilustra la cláusula de captura para la directiva atómica. En este caso el valor de una variable es capturado, y luego se incrementa la variable. Estas operaciones se producen de forma atómica. Este ejemplo particular se podría implementar usando la extracción de instrucciones y suma disponible en muchos tipos de hardware. El ejemplo también muestra una forma de implementar un bloqueo de giro mediante la captura y leer las cláusulas.

*Ejemplo 22.3c*



C / C ++

int fetch\_and\_add (int p \*)

{

/ \* Atómicamente leer el valor de p \* y luego incrementarlo. El valor anterior es

\* Devuelto. Esto puede ser usado para implementar una cerradura simple como se muestra a continuación.

\* /

int edad;

#pragma omp captura atómica

{Old = \* p; (\* P) ​​++; } Return de edad;

}

/ \*

\* Utilizar fetch\_and\_add para implementar una cerradura

\* /

locktype struct {

int ticketnumber; int a su vez;

};

do\_locked\_work vacío (struct locktype \* bloqueo)

{

int atomic\_read (const int \* p); trabajo vacío ();

// obtener el bloqueo

int myturn = fetch\_and\_add (y Lock-> ticketnumber);

mientras que (atomic\_read (y lock-> vuelta)! = myturn)

;

// Haz algo de trabajo. Se necesita el rubor para asegurar la visibilidad

// variables no participan en las directivas atómicas

#pragma omp trabajo flush ();

#pragma omp ras

// Libera la fetch\_and\_add bloqueo (lock-Y> turno);

}

##### C / C ++



Fortran



*Ejemplo 22.3f*

función fetch\_and\_add (p) número entero :: número entero fetch\_and\_add, intención (inout) :: p

! Atómicamente leer el valor de p y luego se incrementará. El valor anterior es

! devuelto. Esto puede ser usado para implementar una cerradura simple como se muestra a continuación.

! $ OMP captura atómica

fetch\_and\_add = pp = p + 1

! $ OMP final atómica

función fetch\_and\_add final

! Utilice fetch\_and\_add implementar un módulo de bloqueo m

interfaz

función fetch\_and\_add (p) número entero :: número entero fetch\_and\_add, intención (inout) :: p

end function

función atomic\_read (p) número entero entero :: atomic\_read, la intención (in) :: p

de extremo a extremo función tipo de interfaz locktype

ticketnumber número entero su vez número entero

tipo final contiene

do\_locked\_work subrutina (bloqueo) de tipo (locktype), la intención (inout) :: bloqueo myturn número entero

basura número entero

! obtener el bloqueo

myturn = fetch\_and\_add (bloquear% ticketnumber)

hacer mientras (atomic\_read (bloqueo% vuelta) .NE myturn).

continuar enddo

! Haz algo de trabajo. Se necesita el rubor para asegurar la visibilidad de las variables

! que no participan en las directivas atómicas

! $ OMP ras

identificador de llamada

! $ OMP ras

! Liberar el bloqueo

chatarra = fetch\_and\_add (bloqueo% turno) subrutina final

módulo final

##### Fortran



## restricciones en el Constructo atómica

Los siguientes ejemplos no conformes ilustran las restricciones a la atómica

construir.

C / C ++



*Ejemplo 23.1c*

atomic\_wrong vacío ()

{

union {int n; float x;} U;

#pragma omp paralelo

{

#pragma omp actualización atómica u.n ++;

#pragma omp atómica actualización ux + = 1,0;

/ \* Referencia incorrecta debido a las construcciones atómicas la misma ubicación a través de tipos incompatibles \* /

}

}

##### C / C ++



Fortran



*Ejemplo 23.1f*

ATOMIC\_WRONG SUBRUTINA () INTEGER :: I

Real :: R equivalencia (I, R)

! $ OMPPARALLEL

! $ OMPATOMIC ACTUALIZAR I = i + 1

! $ OMPATOMIC ACTUALIZAR R = R + 1,0

! incorrecta porque I y R referencia a la misma ubicación

! pero tienen diferentes tipos

! $ OMPEND PARALELA

FIN SUBRUTINA ATOMIC\_WRONG

##### Fortran



C / C ++



*Ejemplo 23.2c*

atomic\_wrong2 vacío ()

{

int x; int \* i; flotador \* R;

i = & x;

R = (float \*) y X;

#pragma omp paralelo

{

#pragma omp actualización atómica

\* I + = 1;

#pragma omp actualización atómica

\* R + = 1,0;

/ \* Referencia incorrecta debido a las construcciones atómicas la misma ubicación a través de tipos incompatibles \* /

}

}

##### C / C ++



Fortran



El siguiente ejemplo no es conforme porque I y R referencia a la misma ubicación pero tienen diferentes tipos.

*Ejemplo 23.2f*

SUB SUBRUTINA () COMÚN / BLK / R real r

! $ OMPATOMIC ACTUALIZAR

R SUBRUTINA SUB = R + 1,0 FIN

SUBRUTINA ATOMIC\_WRONG2 () COMÚN / BLK / I

entero i

! $ OMPPARALLEL

! $ OMPATOMIC ACTUALIZAR I = i + 1

SUB CALL ()

! $ OMPEND PARALELA

FIN SUBRUTINA ATOMIC\_WRONG2

Aunque el siguiente ejemplo se podría trabajar en algunas implementaciones, esto también es disconforme:

*Ejemplo 23.3f*

INTEGER ATOMIC\_WRONG3 SUBRUTINA :: I

Real :: R equivalencia (I, R)

! $ OMPPARALLEL

! $ OMPATOMIC ACTUALIZAR I = i + 1

! incorrecta porque I y R referencia a la misma ubicación

! pero tienen diferentes tipos

! $ OMPEND PARALELA

! $ OMPPARALLEL

! $ OMPATOMIC ACTUALIZAR R = R + 1,0

! incorrecta porque I y R referencia a la misma ubicación

! pero tienen diferentes tipos

! $ OMPEND PARALELA

FIN SUBRUTINA ATOMIC\_WRONG3

##### Fortran



## La construcción ras sin una lista

El siguiente ejemplo distingue las variables compartidas afectadas por una construcción a nivel con ninguna lista de los objetos compartidos que no están afectadas:

C / C ++



*Ejemplo 24.1c*

int x, \* p = & x; f1 void (int \* q)

{

\* Q = 1;

#pragma omp ras

/ \* X, p, y q \* se vacían \* /

/ \*, Ya que son compartidos y accesible \* /

/ \* Q no se vacía, ya que no se comparte. \* /

}

f2 void (int \* q)

{

barrera omp #pragma

\* Q = 2;

barrera omp #pragma

/ \* Una barrera implica un rubor \* /

/ \* X, p, y q \* se vacían \* /

/ \*, Ya que son compartidos y accesible \* /

/ \* Q no se vacía, ya que no se comparte. \* /

}

int g (int n)

{

int i = 1, j, suma = 0;

\* P = 1;

#pragma omp reducción paralela (+: suma) NUM\_THREADS (10)

{

f1 (& j);

/ \* I, n y la suma no estaban rojas \* /

/ \* Porque no eran accesibles en f1 \* /

/ \* J se lavó abundantemente porque era accesible \* / sum + = j;

f2 (& j);

/ \* I, n, y la suma no eran tiraba \* /

/ \* Porque no eran accesibles en f2 \* /

/ \* J se lavó abundantemente porque era accesible \* /

suma + = i + j + \* p + n;

}

suma de retorno;

}

int main ()

{

resultado int = g (7); return resultado;

}

##### C / C ++



*Ejemplo 24.1f*



F1 SUBRUTINA (Q) COMÚN / DATA / X, P

INTEGER, INTEGER OBJETIVO :: X, PUNTERO :: P Q INTEGER

##### Fortran

Q = 1

! $ OMP RUBOR

! X, P y Q se vacían

! ya que son compartidos y extremo accesible SUBRUTINA F1

F2 SUBRUTINA (Q) COMÚN / DATA / X, P

INTEGER, INTEGER OBJETIVO :: X, PUNTERO :: P Q INTEGER

! $ OMPBARRIER Q = 2

! $ OMPBARRIER

! una barrera implica un rubor

! X, P y Q se vacían

! ya que son compartidos y extremo accesible F2 SUBRUTINA

INTEGER función G (N) COMÚN / DATA / X, P INTEGER, OBJETIVO :: X INTEGER, PUNTERO :: P INTEGER N

INTEGER I, J, SUM

I = 1

SUM = 0

P = 1

! $ OMPPARALLEL reducción (+: SUMA) NUM\_THREADS (10) LLAMADA F1 (J)

! I, N y SUM no se barrieron

! Porque no eran accesibles en F1

! J se lavó abundantemente porque era accesible SUM = SUM + J

LLAMADA F2 (J)

! I, N, y SUM no se barrieron

! Porque no eran accesibles en f2

! J se lavó abundantemente porque era accesible SUM = SUM + i + j + P + N

! $ OMPEND PARALELA

G = SUM

END FUNCTION G

PROGRAMA FLUSH\_NOLIST COMÚN / DATA / X, P INTEGER, OBJETIVO :: X INTEGER, PUNTERO :: P INTEGER RESULTADO, G

P => X

RESULT = G (7) PRINT \*, RESULTADO

FIN DEL PROGRAMA FLUSH\_NOLIST

##### Fortran



## Colocación de ras, barrera, taskwait

## y las Directivas taskyield

El siguiente ejemplo es no conforme, porque las rasantes, barrera, taskwait y directivas taskyield son directivas independientes y no pueden ser el substatement inmediata de una sentencia if.

C / C ++



*Ejemplo 25.1c*

standalone\_wrong vacío ()

{

int a = 1;

si (a! = 0)

#pragma omp flush (a)

/ \* Incorrecto, ya que no puede ser al ras substatement inmediata de sentencia if \* /

si (a! = 0)

barrera omp #pragma

/ \* Incorrecto, ya que la barrera no puede ser substatement inmediata de sentencia if \* /

si (a! = 0)

#pragma omp taskyield

/ \* Incorrecto, ya que no puede haber taskyield substatement inmediata de sentencia if \* /

si (a! = 0)

#pragma omp taskwait

/ \* Incorrecto, ya que no puede haber taskwait substatement inmediata de sentencia if \* /

}

##### C / C ++



El siguiente ejemplo es no conforme, porque el rubor, barrera, taskwait, y directivas taskyield son directivas independientes y no pueden ser la instrucción de acción de una sentencia if o un objeto de bifurcación marcada.

Fortran



*Ejemplo 25.1f*

SUBRUTINA STANDALONE\_WRONG () INTEGER A

A = 1

! la directiva descarga no debe ser la instrucción de acción

! en una instrucción IF

Si (a .NE. 0)! $ OMP DE DESCARGA (A)

! la directiva BARRERA no debe ser la instrucción de acción

! en una instrucción IF

Si (a .NE. 0)! $ OMP BARRERA

! la directiva TASKWAIT no debe ser la instrucción de acción

! en una instrucción IF

Si (a .NE. 0)! $ OMP TASKWAIT

! la directiva TASKYIELD no debe ser la instrucción de acción

! en una instrucción IF

Si (a .NE. 0)! $ OMP TASKYIELD GOTO 100

! la directiva descarga no debe ser un objetivo rama etiquetada

! declaración

100! $ OMP DE DESCARGA (A) GOTO 200

! la directiva BARRERA no debe ser un objetivo rama etiquetada

! declaración

200! $ OMP BARRERA GOTO 300

! la directiva TASKWAIT no debe ser un objetivo rama etiquetada

! declaración

300! $ OMP TASKWAIT GOTO 400

! la directiva TASKYIELD no debe ser un objetivo rama etiquetada

! declaración

400! $ OMP TASKYIELD

SUBRUTINA FIN

##### Fortran



La versión siguiente del ejemplo anterior es conforme porque el rubor, barrera, taskwait, y directivas taskyield están encerrados en una instrucción compuesta.

C / C ++



*Ejemplo 25.2c*

standalone\_ok vacío ()

{

int a = 1;

#pragma omp paralelo

{

si (a! = 0) {omp ras #pragma (a)

}

si (a! = 0) {barrera #pragma omp

}

si (a! = 0) {#pragma omp taskwait

}

si (a! = 0) {

#pragma omp taskyield

}

}

}

##### C / C ++



El siguiente ejemplo es conforme porque el rubor, barrera, taskwait, y directivas taskyield están encerrados en un constructo o si siguen el objeto de bifurcación marcada.

Fortran



*Ejemplo 25.2f*

SUBRUTINA STANDALONE\_OK () INTEGER A

A = 1

IF (A .NE. 0) ENTONCES

! $ OMP DE DESCARGA (A) ENDIF

IF (A .NE. 0) ENTONCES

! $ OMP BARRERA ENDIF

IF (A .NE. 0) ENTONCES

! $ OMP TASKWAIT ENDIF

IF (A .NE. 0) ENTONCES

! $ OMP TASKYIELD ENDIF

GOTO 100

100 continuar

! $ OMP DE DESCARGA (A) GOTO 200

200 CONTINUAR

! $ OMP BARRERA GOTO 300

300 CONTINUAR

! $ OMP TASKWAIT GOTO 400

400 CONTINUAR

! SUBRUTINA FIN $ OMP TASKYIELD

##### Fortran



1. **La Cláusula ordenada y la ordenada**

**Construir**

construcciones ordenadas son útiles para secuencialmente ordenar la salida del trabajo que se realiza en paralelo. El siguiente programa imprime los índices en orden secuencial:

C / C ++



*Ejemplo 26.1c*

#include <stdio.h> trabajo vacío (int k)

{

omp #pragma ordenó printf ( "% d \ n", k);

}

ordered\_example vacío (int lb, ub int, int zancada)

{

int i;

#pragma omp paralelo para el horario ordenado (dinámica) para (i = lb; i <ub; i + = zancada)

trabajar (i);

}

int main ()

{

ordered\_example (0, 100, 5);

return 0;

}

##### C / C ++



*Ejemplo 26.1f*



TRABAJO SUBRUTINA (K) entero K

! $ OMP ORDENADO

WRITE (\*, \*) K

! $ OMP FIN ORDENADO

FIN DE TRABAJO SUBRUTINA

##### Fortran

SUBRUTINA SUB (LB, UB, STRIDE) INTEGER LB, UB, STRIDE INTEGER I

Horario! PARALELO $ OMP DO ordenado (dinámico) DO I = LB, UB, ZANCADA

Identificador de llamada (I) END DO

! $ OMP PARALELO FIN terminan SUBRUTINA SUB

PROGRAMA ORDERED\_EXAMPLE SUB CALL (1,100,5)

FIN DEL PROGRAMA ORDERED\_EXAMPLE

##### Fortran



Es posible tener múltiples constructos ordenados dentro de una región de bucle con la cláusula ordenado especificado. El primer ejemplo es disconforme porque todas las iteraciones ejecutan dos regiones ordenadas. Una iteración de un bucle no deben ejecutar más de una región ordenada:

C / C ++



*Ejemplo 26.2C*

trabajo vacío (int i) {}

void ordered\_wrong (int n)

{

int i;

omp #pragma para ordenado for (i = 0; i <n; i ++) {

/ \* Incorrecta porque una iteración no puede ejecutar más de una región encargados \* /

#pragma omp trabajo ordenado (i);

#pragma omp ordenado de trabajo (i + 1);

}

}

##### C / C ++



Fortran



*Ejemplo 26.2f*

TRABAJO SUBRUTINA (I) entero i

FIN DE TRABAJO SUBRUTINA

ORDERED\_WRONG SUBRUTINA (N) INTEGER N

entero i

! $ OMPDO ORDENADO

DO I = 1, N

! incorrecta porque una iteración no puede ejecutar más de una

! región ordenada

! $ OMPORDERED

Identificador de llamada (I)

! $ OMPEND ORDENADO

! $ OMPORDERED

Identificador de llamada (I + 1)

! $ OMPEND ORDENADO FIN DO

FIN SUBRUTINA ORDERED\_WRONG

##### Fortran



El siguiente es un ejemplo conforme con más de una construcción ordenada. Cada iteración se ejecutará sólo una región ordenada:

C / C ++



*Ejemplo 26.3c*

trabajo vacío (int i) {}

void ordered\_good (int n)

{

int i;

omp #pragma para ordenado for (i = 0; i <n; i ++) {

si (i <= 10) {omp #pragma ordenó

trabajar (i);

}

si (i> 10) {

#pragma omp ordenado de trabajo (i + 1);

}

}

}

##### C / C ++



*Ejemplo 26.3f*

ORDERED\_GOOD SUBRUTINA (N) INTEGER N

! $ OMPDO ORDENADO

DO I = 1, N

IF (I <= 10), entonces

! $ OMPORDERED

Identificador de llamada (I)

! $ OMPEND ORDENADO TERMINARA SI

Si (i> 10) ENTONCES

! $ OMPORDERED

Identificador de llamada (I + 1)

! $ OMPEND ORDENADO TERMINARA SI

ENDDO

FIN SUBRUTINA ORDERED\_GOOD



##### Fortran

Fortran



## Cancelación construcciones

*Ejemplo 27.1c*



C / C ++

El siguiente ejemplo muestra cómo cancelar la directiva puede ser utilizado para terminar una región OpenMP. Aunque la construcción de cancelar termina la región de trabajo compartido OpenMP, los programadores aún debe realizar un seguimiento de la excepción a través del puntero ex y emitir una cancelación de la región paralelo si una excepción se ha planteado. El hilo maestro comprueba el puntero excepción para asegurarse de que la excepción se maneja adecuadamente en la parte secuencial. Si se ha solicitado la cancelación de la región paralela, algunos temas podrían haber ejecutado phase\_1 (). Sin embargo, se garantiza que ninguno de los hilos ejecutado phase\_2 ().

void ejemplo () {std :: excepción \* ex = NULL;

#pragma omp paralelo compartido (ex)

{

#pragma omp para

for (int i = 0; i <N; i ++) {

// si no hay '' que impide que las optimizaciones del compilador try {

causes\_an\_exception ();

}

captura (const std :: excepción \* e) {

// aún debe recordar excepción para el manejo posterior #pragma omp escritura atómica

ex = e;

// cancelar constructo trabajo compartido

#pragma omp para cancelar

}

}

// si una excepción se ha planteado, cancelación región paralela si (ex) {

#pragma omp cancelar paralelo

}

fase 1(); barrera omp #pragma

fase 2();

}

// continuar aquí si una excepción ha sido lanzada en el circuito de trabajo compartido si (ex) {

// excepción de identificador almacenado en ex

}

}

##### C / C ++



Fortran



*Ejemplo 27.1f*

El siguiente ejemplo ilustra el uso de la construcción de cancelar en el tratamiento de errores. Si hay una condición de error de la declaración asignar, se activa la cancelación. El hilo Encountering establece el err variable compartida y otros hilos del conjunto de la rosca de unión proceden a la final del trabajo compartido construir después de la cancelación ha sido activado.

ejemplo subrutina (n, DIM) número entero, la intención (in) :: n, entero dim (n) :: i, s, err

reales, asignable :: B (:) err = 0

! $ OMP paralelo compartida (err)

! ...

! $ OMP hacer privada (s, B) hacer i = 1, n

! Punto $ cancelación omp asignar (B (dim (i)), stat = s) si (s .gt. 0), entonces

! $ OMP de escritura atómica err = s

! $ OMP cancelar hacer endif

! ...

! deallocate array privado B si (asignada (B)), entonces

deallocate (B) terminara si

enddo

! $ OMP extremo final subrutina paralelo

##### Fortran



*Ejemplo 27.2c*



C / C ++

El siguiente ejemplo muestra cómo cancelar una búsqueda paralela en un árbol binario, tan pronto como se ha detectado el valor de búsqueda. El código crea una tarea a descender en los nodos secundarios del nodo de árbol actual. Si se ha encontrado el valor de búsqueda, el código recuerda el nodo de árbol con el valor encontrado a través de una escritura atómica a la variable de resultado y luego cancela la ejecución de todas las tareas de búsqueda. Los grupos de función search\_tree\_parallel todas las tareas de búsqueda en un solo grupo de tareas para controlar el efecto de la directiva cancelar taskgroup. El argumento de nivel se utiliza para crear tareas undeferred después de los primeros diez niveles del árbol.

binary\_tree\_t \* search\_tree (\* binary\_tree\_t árbol, valor int, int level) {\* binary\_tree\_t encontró = NULL;

si (árbol) {

si (árbol-> valor == valor) {encontrado = árbol;

}

else {

#pragma omp tarea compartida (encontrado) si (nivel <10)

{

binary\_tree\_t \* found\_left = NULL;

found\_left = search\_tree (de árboles> izquierda, el valor, nivel + 1); si (found\_left) {

#pragma omp escritura atómica

encontrado = found\_left; #pragma omp cancelar taskgroup

}

}

#pragma omp tarea compartida (encontrado) si (nivel <10)

{

binary\_tree\_t \* found\_right = NULL;

found\_right = search\_tree (de árboles> derecha, el valor, nivel + 1); si (found\_right) {

#pragma omp escritura atómica

encontrado = found\_right; #pragma omp cancelar taskgroup

}

}

#pragma omp taskwait

}

}

volver encontrado;

}

binary\_tree\_t \* search\_tree\_parallel (\* binary\_tree\_t árbol, int value) {\* binary\_tree\_t encontró = NULL;

#pragma omp paralelo compartida (que se encuentra, árbol, valor)

{

#pragma omp maestro

{

#pragma omp taskgroup

{

encontrado = search\_tree (árbol, el valor, 0);

}

}

}

volver encontrado;

}

##### C / C ++



Fortran



*Ejemplo 27.2f*

El siguiente es el ejemplo de búsqueda en paralelo equivalente en Fortran.

módulo de tipo parallel\_search binary\_tree

valor entero

Tipo (binary\_tree), puntero :: tipo correcto (binary\_tree), puntero izquierdo ::

tipo de final

!

contiene

search\_tree subrutina recursiva (árbol, valor, nivel, encontrado) tipo (binary\_tree), la intención (en), puntero del árbol ::

número entero, la intención (in) :: valor, tipo de nivel (binary\_tree), puntero :: encontrado

Tipo (binary\_tree), puntero :: found\_left => NULL (), found\_right => NULL ()

!

si (.not. asociado (encontrado)) a continuación, asignar (encontrado)

terminara si

!

si (asociado (árbol)), entonces

si (árbol valor% .EQ. valor) a continuación, encontrado = árbol

más

! $ OMP tarea compartida (encontrado) si (nivel <10)

llamar search\_tree (árbol% a la izquierda, el valor, el nivel + 1, found\_left) si (asociado (found\_left)), entonces

! $ OMP de escritura atómica

encontrado = found\_left

!

! $ OMP cancelar endif taskgroup

! $ Tarea final omp

!

! $ OMP tarea compartida (encontrado) si (nivel <10)

llamar search\_tree (árbol de la derecha%, valor, nivel + 1, found\_right) si (asociado (found\_right)), entonces

! $ OMP de escritura atómica

encontrado = found\_right

!

! $ OMP cancelar endif taskgroup

! $ Tarea final omp

!

! $ OMP taskwait

endif endif

subrutina final

!

search\_tree\_parallel subrutina (árbol, valor, que se encuentra) tipo (binary\_tree), la intención (en), número entero puntero :: árbol, la intención (en) :: valor

Tipo (binary\_tree), puntero :: encontrado

!

si (asociado (encontrado)) a continuación, asignar (encontrado)

terminara si

! $ OMP paralelo compartida (que se encuentra, árbol, valor)

! $ OMP maestro

! $ OMP taskgroup

llamar search\_tree (árbol, valor 0, que se encuentra)

! $ OMP final taskgroup

! $ Maestro extremo omp

! $ OMP extremo final subrutina paralelo

!

módulo parallel\_search final

##### Fortran



## La Directiva threadprivate

Los siguientes ejemplos demuestran cómo utilizar la directiva threadprivate para dar a cada hilo un contador independiente.



*Ejemplo 28.1c*

int contador = 0;

threadprivate #pragma omp (contador)

int increment\_counter ()

{

contador ++; RETURN (contador);

}



##### C / C ++

C / C ++



Fortran



*Ejemplo 28.1f*

INCREMENT\_COUNTER INTEGER FUNCIÓN () COMÚN / INC\_COMMON / COUNTER

! $ OMPTHREADPRIVATE (/ INC\_COMMON /)

CONTADOR = CONTADOR 1 INCREMENT\_COUNTER = RETURN COUNTER

FIN DE FUNCIONES INCREMENT\_COUNTER

##### Fortran



C / C ++



El ejemplo siguiente utiliza threadprivate en una variable estática:

*Ejemplo 28.2c*

int increment\_counter\_2 ()

{

int contador estático = 0;

threadprivate #pragma omp (contador) contador ++;

RETURN (contador);

}

El siguiente ejemplo muestra el comportamiento no especificado para la inicialización de una variable threadprivate. Una variable threadprivate se inicializa una vez en un punto no especificado antes de su primera referencia. Debido a que una se construye utilizando el valor de x (que se modifica mediante la declaración x ++), el valor de a.val en el inicio de la región paralela podría ser 1 o 2. Este problema se evita para b, que utiliza un auxiliar variables const y un constructor de copia.

*Ejemplo 28.3c*

clase T {public:

int val;

T (int);

T (const T &);

};

T :: T (int v) {val = v;

}

T :: T (const T & t) {val = t.val;

}

void g (T a, T b) {a.val + = b.val;

}

int x = 1; T a (x);

const T b\_aux (x); / \* Valor de captura de x = 1 \* / T b (b\_aux);

#pragma omp threadprivate (a, b)

void f (int n) {x ++;

#pragma omp paralelo para

/ \* En cada hilo:

* **una se construye a partir de x (con valor de 1 o 2?)**
* **b es copia-construido a partir de b\_aux**

\* /

for (int i = 0; i <n; i ++) {

g (a, b); / \* Valor de una no está especificado. \* /

}

}

C / C ++



Fortran



Los siguientes ejemplos muestran usos no conformes y usos correctos de la

**threadprivate** directiva.

El siguiente ejemplo no es conforme porque el bloque común no se declara local a la subrutina que se refiere a ella:

*Ejemplo 28.2f*

MÓDULO INC\_MODULE COMÚN / T / A

FIN MÓDULO INC\_MODULE

SUBRUTINA INC\_MODULE\_WRONG () INC\_MODULE USO

! $ OMPTHREADPRIVATE (/ t /)

! Disconforme porque / T / no declarado en INC\_MODULE\_WRONG INC\_MODULE\_WRONG FIN SUBRUTINA

El siguiente ejemplo es también no conforme porque el bloque común no se declara local a la subrutina que se refiere a ella:

*Ejemplo 28.3f*

SUBRUTINA INC\_WRONG () COMÚN / T / A

! $ OMPTHREADPRIVATE (/ t /)

CONTIENE

INC\_WRONG\_SUB SUBRUTINA ()

! $ OMPPARALLEL Copyin (/ T /)

! Disconforme porque / T / no declarado en INC\_WRONG\_SUB

! $ OMPEND PARALELA

FIN FIN SUBRUTINA INC\_WRONG\_SUB INC\_WRONG SUBRUTINA

##### Fortran (cont.)

El siguiente ejemplo es una reescritura correcta del ejemplo anterior:

*Ejemplo 28.4f*

INC\_GOOD SUBRUTINA () COMÚN / T / A

! $ OMPTHREADPRIVATE (/ t /)

CONTIENE

INC\_GOOD\_SUB SUBRUTINA () COMÚN / T / A

! $ OMPTHREADPRIVATE (/ t /)

! $ OMPPARALLEL Copyin (/ T /)

! $ OMPEND PARALELA

FIN FIN SUBRUTINA INC\_GOOD\_SUB INC\_GOOD SUBRUTINA

El siguiente es un ejemplo del uso de threadprivate para las variables locales:

*Ejemplo 28.5f*

PROGRAMA INC\_GOOD2

INTEGER, ALLOCATABLE, SAVE :: A (:) INTEGER, PUNTERO, Save :: PTR INTEGER, SAVE :: I

INTEGER, OBJETIVO :: :: TARG LÓGICO primeras entradas = .TRUE.

! $ OMPTHREADPRIVATE (A, I, PTR)

ASIGNAR (A (3)) A = (/ 1,2,3 /) PTR => TARG

I = 5

! $ OMPPARALLEL copyin (I, PTR)

! $ OMPCRITICAL

IF (primeras entradas) ENTONCES

TARG = 4! Actualización de destino ptr I = i + 10

IF (ASIGNADO (A)) A = A + 10 primeras entradas = .FALSE.

TERMINARA SI

IF (ASIGNADO (A)) a continuación, imprimir \*, 'a =', A

MÁS

##### Fortran (cont.)

IMPRESIÓN \*, 'A no se asigna' END IF

IMPRESIÓN \*, 'ptr =', PTR IMPRESIÓN \*, 'i =', imprimo \*

! $ OMPEND CRÍTICO

! $ OMPEND PARALELA

FIN DEL PROGRAMA INC\_GOOD2

El programa anterior, si se ejecuta por dos hilos, se imprimirá una de las siguientes dos conjuntos de salida:

a = 11 12 13

ptr = 4

i = 15

A no se asigna ptr = 4

i = 5

o

A no se asigna ptr = 4

i = 15

a = 1 2 3

ptr = 4

i = 5

El siguiente es un ejemplo del uso de threadprivate para variables de módulo:

*Ejemplo 28.6f*

MÓDULO INC\_MODULE\_GOOD3 REAL, EL PUNTERO DE TRABAJO :: (:) guardar el trabajo

! $ OMPTHREADPRIVATE (TRABAJO)

FIN MÓDULO INC\_MODULE\_GOOD3

SUB1 SUBRUTINA (N) USO INC\_MODULE\_GOOD3

! $ OMPPARALLEL PRIVADO (THE\_SUM) ASIGNAR (TRABAJO (N))

SUB2 CALL (THE\_SUM) WRITE (\*, \*) THE\_SUM

! $ OMPEND PARALELA

FIN SUBRUTINA SUB1

SUBRUTINA SUB2 (THE\_SUM) USO INC\_MODULE\_GOOD3 TRABAJO (:) = 10 THE\_SUM = SUM (TRABAJO)

FIN SUBRUTINA SUB2

PROGRAMA INC\_GOOD3 N = 10

LLAMADA SUB1 (N)

FIN DEL PROGRAMA INC\_GOOD3

##### Fortran



C / C ++



El siguiente ejemplo ilustra la inicialización de variables threadprivate para-tipo de clase T. T1 está predeterminado construido, t2 se construye tomando un constructor aceptar un argumento de tipo entero, t3 se copia construye con argumento f ():

*Ejemplo 28.4c*

t1 T estática;

#pragma omp threadprivate (t1) t2 T estática (23);

#pragma omp threadprivate (t2) T estática t3 = f ();

threadprivate #pragma omp (t3)

El siguiente ejemplo ilustra el uso de threadprivate para miembros de la clase estáticas. La directiva threadprivate para un miembro de la clase estática debe ser colocado dentro de la definición de clase.

*Ejemplo 28.5c*

clase T {public:

static int i;

threadprivate #pragma omp (i)

};

##### C / C ++



C / C ++



## Paralelo iterador de acceso aleatorio Loop

El siguiente ejemplo muestra un bucle iterador de acceso aleatorio en paralelo.

*Ejemplo 29.1c*

#include <vector>

iterator\_example anular ()

{

std :: vector <int> vec (23); std :: vector <int> :: iterador ella;

#pragma omp paralelo para defecto (ninguno) compartido (vec) para (it = vec.begin (); it <vec.end (); se ++)

{

// hacer trabajo con \* se //

}

}

##### C / C ++



Fortran



1. **Restricciones Fortran sobre compartido y**

**Las cláusulas privadas con bloques comunes**

Cuando se especifica un bloque común con nombre en una, firstprivate o cláusula lastprivate privada de una construcción, ninguno de sus miembros puede ser declarado en otro atributo cláusula de intercambio de datos sobre esa construcción. Los siguientes ejemplos ilustran este punto.

El siguiente ejemplo es conforme:

*Ejemplo 30.1f*

COMMON\_GOOD SUBRUTINA () COMÚN / C / X, Y

Real X, Y

! $ OMPPARALLEL PRIVADO (/DO/)

! trabajo aquí

! $ OMPEND PARALELA

! $ OMPPARALLEL COMPARTIDA (X, Y)

! trabajo aquí

! $ OMPEND PARALELA

FIN SUBRUTINA COMMON\_GOOD

El ejemplo siguiente también es conforme:

*Ejemplo 30.2f*

SUBRUTINA COMMON\_GOOD2 () COMÚN / C / X, Y

Real X, Y INTEGER I

! $ OMPPARALLEL

! $ OMPDO PRIVADO (/ C /)

DO I = 1,1000

! trabajo aquí ENDDO

! $ OMPEND HACER

! $ OMPDO PRIVADO (X) DO I = 1,1000

! trabajo aquí ENDDO

! $ OMPEND HACER

! $ OMPEND PARALELA

FIN SUBRUTINA COMMON\_GOOD2

El siguiente ejemplo es conforme:

*Ejemplo 30.3f*

SUBRUTINA COMMON\_GOOD3 () COMÚN / C / X, Y

! $ OMPPARALLEL PRIVADO (/DO/)

! trabajo aquí

! $ OMPEND PARALELA

! $ OMPPARALLEL COMPARTIDA (/DO/)

! trabajo aquí

! $ OMPEND PARALELA

FIN SUBRUTINA COMMON\_GOOD3

El siguiente ejemplo no es conforme porque x es un elemento constitutivo de c:

*Ejemplo 30.4f*

SUBRUTINA COMMON\_WRONG () COMÚN / C / X, Y

! Incorrecta porque X es un elemento constitutivo de C

! $ OMPPARALLEL PRIVADO (/ C /), Compartido (X)

! trabajo aquí

! $ OMPEND PARALELA

FIN SUBRUTINA COMMON\_WRONG

El siguiente ejemplo no es conforme, porque un bloque común no puede declararse ambos compartían y privada:

*Ejemplo 30.5f*

COMMON\_WRONG2 SUBRUTINA () COMÚN / C / X, Y

! Incorrecto: C bloque común no puede ser declarado tanto

! compartida y privada

! $ OMPPARALLEL PRIVADO (/ C /), Compartido (/ C /)

! trabajo aquí

! $ OMPEND PARALELA

FIN SUBRUTINA COMMON\_WRONG2

##### Fortran



## El (ninguno) Cláusula predeterminado

El siguiente ejemplo distingue las variables que se ven afectados por el

**por defecto (ninguno)** cláusula de las que no lo son.

##### C / C ++



*Ejemplo 31.1c*

#include <omp.h> int x, y, z [1000];

threadprivate #pragma omp (x)

void default\_none (int a) {int const c = 1;

int i = 0;

#pragma omp predeterminado paralelo (ninguno) privado (a) compartida (z)

{

int j = omp\_get\_num\_threads ();

/ \* OK - j se declara dentro de la región en paralelo \* / a = z [j]; / \* OK - una aparece en la cláusula privada \* /

/ \* - z aparece en la cláusula compartida \* / x = c; / \* OK - x es threadprivate \* /

/ \* - c tiene const cualificado tipo \* / z [i] = y; / \* Error - no puede hacer referencia i o Y aquí \* /

omp #pragma para firstprivate (y)

/ \* Error - No se puede hacer referencia y en la cláusula firstprivate \* / for (i = 0; i <10; i ++) {

z [i] = i; / \* OK - i es la variable de iteración de bucle \* /

}

z [i] = y; / \* Error - no puede hacer referencia i o Y aquí \* /

}

}

##### C / C ++



Fortran



*Ejemplo 31.1f*

DEFAULT\_NONE SUBRUTINA (A)

INCLUIR "Omp\_lib.h"! o usar OMP\_LIB INTEGER

INTEGER X, Y, Z (1000) COMÚN / BLOCKX / X COMÚN / BLOCKY / Y COMÚN / Blockz / Z

! $ OMP threadprivate (/ BLOCKX /)

INTEGER I, J

i = 1

! $ OMPPARALLEL DEFAULT (NINGUNO) PRIVADA (A) compartido (Z) PRIVADO (J) J = OMP\_GET\_NUM\_THREADS ();

! OK - J aparece en la cláusula PRIVADO A = Z (J)! OK - Un aparece en la cláusula PRIVADO

- Z aparece en COMPARTIDO cláusula X = 1! OK - X es threadprivate

Z (I) = Y! Error - no puede hacer referencia I o Y aquí

! $ OMP DO firstprivate (y)

! Error - No se puede hacer referencia a en la cláusula firstprivate DO I = 1,10

Z (I) = I! OK - I es la repetición del bucle FIN variables sí

Z (I) = Y! Error - no puede hacer referencia I o Y aquí

! $ OMPEND PARALELA

FIN SUBRUTINA DEFAULT\_NONE

##### Fortran



Fortran



## Las condiciones de carrera Causado por Copias implícitas de variables compartidas en Fortran

El ejemplo siguiente contiene una condición de carrera, porque la variable compartida, que es una sección de matriz, se pasa como un argumento real a una rutina que tiene una matriz de tamaño asumido como su argumento ficticio. La llamada a una subrutina que pasa una sección argumento de matriz puede causar el compilador para copiar el argumento en una ubicación temporal antes de la llamada y copia de la ubicación temporal en la variable original cuando el subprograma vuelve. Esta copia causaría carreras en la región paralelo.

*Ejemplo 32.1f*

SHARED\_RACE SUBRUTINA

INCLUIR "Omp\_lib.h"! o usar OMP\_LIB VERDADERO A (20)

INTEGER MYTHREAD

! PARALELO $ OMP compartido (A) PRIVADO (MYTHREAD) MYTHREAD = OMP\_GET\_THREAD\_NUM ()

IF (MYTHREAD .EQ. 0) ENTONCES

SUB LLAMADA (A (1:10))! compilador puede introducir escribe a A (6:10) ELSE

A (06:10) = 12 ENDIF

! PARALELO $ OMP FIN

FIN SUBRUTINA SHARED\_RACE

SUB SUBRUTINA (X) real x (\*) X (1: 5) = 4

END SUB SUBRUTINA

##### Fortran



## La Cláusula privada

En el siguiente ejemplo, los valores de elementos de lista originales I y J son retenidos en la salida de la región paralela, mientras que los elementos de la lista privadas i y j se modifican dentro de la construcción paralelo.

C / C ++



*Ejemplo 33.1c*

#include <stdio.h> #include <assert.h>

int main ()

{

int i, j;

int \* ptr\_i, \* ptr\_j;

i = 1;

J = 2;

ptr\_i = & i; ptr\_j = & j;

#pragma omp firstprivate privado paralelo (i) (j)

{

i = 3;

j = j + 2;

afirmar (\* ptr\_i == 1 && \* ptr\_j == 2);

}

afirmar (i == 1 && j == 2);

return 0;

}

##### C / C ++



*Ejemplo 33.1f*



PROGRAMA PRIV\_EXAMPLE INTEGER I, J

I = 1

J = 2

##### Fortran

! $ OMPPARALLEL PRIVADO (I) FIRSTPRIVATE (J)

I = 3

J = J + 2

! $ OMPEND PARALELA

PRINT \*, I, J! Me .EQ. 1 .y. J .EQ. 2 FIN PROGRAMA PRIV\_EXAMPLE

##### Fortran



En el siguiente ejemplo, todos los usos de la variable una dentro del bucle construyen en el f rutina referirse a un elemento de la lista privada a, mientras que es sin especificar si las referencias a una en la g rutina son para un elemento de lista privada o el elemento de la lista original de .

C / C ++



*Ejemplo 33.2c*

int a;

void g (int k) {

a = k; / \* Se accede en la región pero fuera de la construcción;

* **por lo tanto, sin especificar si la lista original o privada**
* **artículo se modificó. \* /**

}

void f (int n) {int a = 0;

#pragma omp paralelo para privado (a) for (int i = 1; i <n; i ++) {

a = i;

g (a \* 2); / \* copia privada de "a" \* /

}

}

##### C / C ++



Fortran



*Ejemplo 33.2f*

MÓDULO PRIV\_EXAMPLE2 REAL

CONTIENE

SUBRUTINA G (K) venta de K

A = K! Visitada en la región pero fuera de la

! construir; por lo tanto, sin especificar si

! elemento de la lista original o privada se modifica.

FIN SUBRUTINA G

SUBRUTINA F (N) INTEGER N

Un VERDADERO

entero i

! $ OMPPARALLEL DO PRIVADO (A) DO I = 1, N

A = I

LLAMADA G (A \* 2) ENDDO

! $ OMPEND PARALELO HACER FIN SUBRUTINA F

FIN MÓDULO PRIV\_EXAMPLE2

##### Fortran



El siguiente ejemplo demuestra que un elemento de la lista que aparece en una cláusula privada en una construcción paralela también puede aparecer en una cláusula privada en una construcción de trabajo compartido cerrado, lo que resulta en una copia privada adicional.

C / C ++



*Ejemplo 33.3c*

# include <assert.h> priv\_example3 vacío ()

{

int i, a;

#pragma omp paralelo privada (a)

{

a = 1;

#pragma omp paralelo para privado (a) para (i = 0; i <10; i ++)

{

a = 2;

}

afirmar (a == 1);

}

}

##### C / C ++



*Ejemplo 33.3f*



SUBRUTINA PRIV\_EXAMPLE3 () INTEGER I, A

##### Fortran

! $ OMPPARALLEL PRIVADO (A)

A = 1

! $ OMPPARALLEL DO PRIVADA (A) DO I = 1, 10

A = 2 FIN DO

! $ OMPEND PARALELO HACER

PRINT \*, A! Outer A todavía tiene valor 1

! $ OMPEND PARALELA

FIN SUBRUTINA PRIV\_EXAMPLE3

##### Fortran



Fortran



1. **Restricciones Fortran en la Asociación de almacenamiento con la Cláusula privada**

Los siguientes ejemplos no conformes ilustran las implicaciones de la privado

cláusula gobierna con respecto a la asociación de almacenamiento.

*Ejemplo 34.1f*

SUB SUBRUTINA () COMÚN / BLOQUE / X

IMPRESIÓN \*,¡X! X es FIN indefinido SUBRUTINA SUB

PROGRAMA PRIV\_RESTRICT COMÚN / BLOQUE / X

X = 1,0

! $ OMPPARALLEL PRIVADO (X)

X = 2,0 SUB CALL ()

! $ OMPEND PARALELA

PRIV\_RESTRICT FIN DEL PROGRAMA

#### *Ejemplo 34.2f*

PROGRAMA PRIV\_RESTRICT2 COMÚN / Bloque2 / X

X = 1,0

! $ OMPPARALLEL PRIVADO (X) X = 2,0

SUB CALL ()

! PARALELO $ OMPEND CONTIENE

SUB SUBRUTINA () COMÚN / Bloque2 / Y

IMPRESIÓN \*,¡X! X es indefinido

IMPRESIÓN \*, Y! Y es indefinido SUBRUTINA FIN SUB

FIN DEL PROGRAMA PRIV\_RESTRICT2

##### Fortran (cont.)

*Ejemplo 34.3f*

PROGRAMA PRIV\_RESTRICT3 de equivalencia (X, Y)

X = 1,0

! $ OMPPARALLEL Privada (X)

IMPRESIÓN \*, Y! Y es indefinido Y = 10

IMPRESIÓN \*,¡X! X es indefinido

! $ OMPEND PARALELA

FIN DEL PROGRAMA PRIV\_RESTRICT3

#### *Ejemplo 34.4f*

PROGRAMA PRIV\_RESTRICT4 INTEGER I, J

INTEGER A (100), B (100) de equivalencia (A (51), B (1))

! PARALELO $ OMP HACER POR DEFECTO (PRIVADO) PRIVADO (I, J) LASTPRIVATE (A) DO I = 1,100

DO J = 1,100 B (J) = J - 1

ENDDO

DO J = 1,100

A (J) = J! B se convierte en indefinido en este punto ENDDO

DO J = 1,50

B (J) = B (J) + 1! B no está definido

! A queda indefinido en este momento

ENDDO ENDDO

! PARALELO $ OMP FIN ¡HACER! La escritura de un LASTPRIVATE tiene

! resultados indefinidos

IMPRESIÓN \*, ¡SEGUNDO! B no está definido ya que la LASTPRIVATE

! escritura de A no se definió FIN DEL PROGRAMA PRIV\_RESTRICT4

#### *Ejemplo 34.5f*

SUB1 SUBRUTINA (X) DIMENSION X (10)

! Este uso de X no se ajusta a la

! especificación. Sería legal Fortran 90,

! pero la directiva OpenMP privada permite al

! compilador para romper la asociación secuencia que

! A tenía con el resto del bloque común.

FORALL (I = 01:10) X (i) = i End SUB1 SUBRUTINA

PROGRAMA PRIV\_RESTRICT5 COMÚN / Bloque 5 / A

DIMENSIÓN B (10) de equivalencia (A, B (1))

! el bloque común tiene que ser de al menos 10 palabras A = 0

! $ OMPPARALLEL PRIVADO (/ Bloque 5 /)

! Sin la cláusula privada,

! estaríamos pasando un miembro de una secuencia

! que es al menos diez elementos de largo.

! Con la cláusula privada, A ya no puede ser

! secuencia asociada.

SUB1 CALL (A)

! $ OMPMASTER PRINT \*, UNA

! $ OMPEND DOMINAR

! $ OMPEND PARALELA

FIN DEL PROGRAMA PRIV\_RESTRICT5

##### Fortran



C / C ++



## C / Arrays C ++ en una cláusula firstprivate

El ejemplo siguiente ilustra el tamaño y el valor de elementos de la lista de matriz o tipo de puntero en una cláusula firstprivate. El tamaño de los nuevos elementos de la lista se basa en el tipo de elemento de la lista original correspondiente, según lo determinado por el idioma base.

En este ejemplo:

* + El tipo de **UNA** es array de dos conjuntos de dos enteros.
  + El tipo de **segundo** se ajusta a puntero a la matriz de **norte** enteros, ya que es un parámetro de función.
  + El tipo de **do** se ajusta a puntero a int, ya que es un parámetro de función.
  + El tipo de **re** es array de dos conjuntos de dos enteros.
  + El tipo de **mi** es gama de **norte** matrices de **norte** enteros.

Tenga en cuenta que B y E implican tipos de matriz longitud variable.

Los nuevos elementos de tipo de matriz se inicializan como si cada elemento de número entero de la matriz original se asigna al elemento correspondiente de la nueva matriz. Los de tipo de puntero se inicializan como por la asignación del elemento original al nuevo elemento.

*Ejemplo 35.1c*

# include <assert.h>

int A [2] [2] = {1, 2, 3, 4};

f void (int n, int B [n] [n], int C [])

{

int D [2] [2] = {1, 2, 3, 4};

int E [n] [n];

afirmar (n> = 2);

E [1] [1] = 4;

#pragma omp firstprivate paralelo (B, C, D, E)

{

afirmar (sizeof (B) == sizeof (int (\*) [n])); afirmar (sizeof (C) == sizeof (int \*)); afirmar (sizeof (D) == 4 \* sizeof (int)); valer (sizeof (E) == n \* n \* sizeof (int));

/ \* B privada y C tienen valores de B original y C. \* / assert (& B [1] [1] == & A [1] [1]);

afirmar (& C [3] == & A [1] [1]);

afirmar (D [1] [1] == 4);

afirmar (E [1] [1] == 4);

}

}

int main () {f (2, A, A [0]);

return 0;

}

##### C / C ++



## La Cláusula lastprivate

correcta ejecución depende a veces el valor que la última iteración de un bucle asigna a una variable. Tales programas deben enumerar todas estas variables en una cláusula lastprivate de modo que los valores de las variables son los mismos que cuando el bucle se ejecuta secuencialmente.

*Ejemplo 36.1c*



C / C ++

lastpriv void (int n, flotar \* a, float \* b)

{

int i;

#pragma omp paralelo

{

omp #pragma para lastprivate (i) para (i = 0; i <n-1; i ++)

a [i] = b [i] + b [i + 1];

}

a [i] = b [i]; / \* i == n-1 aquí \* /

}

##### C / C ++



Fortran



*Ejemplo 36.1f*

SUBRUTINA LASTPRIV (N, A, B)

INTEGER N

Un VERDADERO (\*), B (\*) entero i

! $ OMP PARALELO

! $ OMP DO LASTPRIVATE (I)

DO I = 1, N-1

A (I) = B (I) + B (I + 1) ENDDO

! PARALELO $ OMP FIN

A (I) = BI)! I tiene el valor de N aquí

FIN SUBRUTINA LASTPRIV

##### Fortran



## La cláusula de reducción

El siguiente ejemplo demuestra la cláusula de reducción; en cuenta que algunas reducciones se pueden expresar en el bucle de varias maneras, como se muestra para los MAX y MIN reducciones por debajo de:

*Ejemplo 37.1c*



C / C ++

#include <math.h>

reduction1 void (float \* x, int \* y, int n)

{

int i, b, c; flotar a, d; a = 0,0;

b = 0;

c = y [0];

d = x [0];

#pragma omp paralelo para uso privado (i) compartido (x, y, n) \

reducción (+: a) reducción de (^: b) \ reducción (min: c) reducción (max: d)

for (i = 0; i <n; i ++) {a + = x [i];

b ^ = y [i];

si (c> y [i]) c = y [i];

d = fmaxf (d, x [i]);

}

}

##### C / C ++



Fortran



*Ejemplo 37.1f*

SUBRUTINA REDUCTION1 (A, B, C, D, X, Y, N) real :: X (\*), A, D

INTEGER :: Y (\*), N, B, C INTEGER :: I

A = 0

B = 0

C = Y (1)

D = X (1)

! PARALELO $ OMP DO PRIVADO (I) COMPARTIDA (X, Y, N) Reducción (+: A) y

! $ OMP y reducción (IEOR: B) Reducción (MIN: c) reducción (MAX: D) DO I = 1, N

A = A + X (I)

B = IEOR (B, Y (I))

C = MIN (C, Y (I))

Si D = X (I) END (<X (I) D) DO

FIN SUBRUTINA REDUCTION1

##### Fortran



Una implementación común del ejemplo anterior es tratarlo como si hubiera sido escrito de la siguiente manera:

*Ejemplo 37.2c*



C / C ++

# include <limits.h> #include <math.h>

reduction2 void (float \* x, int \* y, int n)

{

int i, b, b\_p, c, c\_p; flotar a, a\_p, d, D\_P; a = 0.0f;

b = 0;

c = y [0];

d = x [0];

#pragma omp paralelo compartido (a, b, c, d, x, y, n) \

privada (a\_p, b\_p, c\_p, D\_P)

{

a\_p = 0.0f; b\_p = 0;

c\_p = INT\_MAX; D\_P = -HUGE\_VALF;

omp #pragma para privado (i) para (i = 0; i <n; i ++) {

a\_p + = x [i]; b\_p ^ = y [i];

si (c\_p> y [i]) c\_p = y [i]; D\_P = fmaxf (D\_P, x [i]);

}

#pragma omp crítica

{

a + = a\_p; b ^ = b\_p;

si (c> c\_p) c = c\_p; d = fmaxf (d, D\_P);

}

}

}

##### C / C ++



Fortran



*Ejemplo 37.2f*

SUBRUTINA REDUCTION2 (A, B, C, D, X, Y, N) real :: X (\*), A, D

INTEGER :: Y (\*), N, B, C real :: A\_P, D\_P

INTEGER :: I, B\_P, C\_P A = 0

B = 0

C = Y (1)

D = X (1)

! PARALLEL $ OMP compartido (X, Y, A, B, C, D, N) y

! $ OMP Y PRIVADO (A\_P, B\_P, C\_P, D\_P) A\_P = 0,0

B\_P = 0

C\_P = enorme (C\_P) D\_P = -Enorme (D\_P)

! $ OMP DO PRIVADO (I) DO I = 1, N

A\_P = A\_P + X (I)

B\_P = IEOR (B\_P, Y (I)) C\_P = MIN (C\_P, Y (I))

IF (D\_P <X (I)) D\_P = X (I) END DO

! $ OMP CRÍTICA A = A + A\_P

B = IEOR (B, B\_P) C = MIN (C, C\_P) D = MAX (D, D\_P)

! $ OMP finalidad crítica

! PARALELO $ OMP FIN

FIN SUBRUTINA REDUCTION2

El siguiente programa no es conforme a causa de la reducción está en el nombre del procedimiento intrínseca MAX pero ese nombre se ha redefinido a ser la variable llamada MAX.

Fortran (cont.)

*Ejemplo 37.3f*

PROGRAMA REDUCTION\_WRONG MAX = enorme (0)

M = 0

! PARALELO $ OMP DO reducción (MAX: M)

! MAX ya no es la intrínseca por lo que este no es conforme DO I = 1, 100

LLAMADA SUB (M, I)

FIN DO

PROGRAMA FIN REDUCTION\_WRONG SUBRUTINA SUB (M, I)

M = MAX (M, I) END SUB SUBRUTINA

El siguiente programa de conformación realiza la reducción utilizando el nombre del procedimiento intrínseca MAX pesar de que el MAX intrínseca ha sido renombrado a REN.

#### *Ejemplo 37.4f*

MÓDULO M

INTRINSIC MAX FIN módulo M

PROGRAMA REDUCTION3 USO M, REN => MAX N = 0

! PARALELO $ OMP DO reducción (REN: NORTE)! todavía lo hace MAX DO I = 1, 100

N = MAX (N, I) END DO

FIN DEL PROGRAMA REDUCTION3

El siguiente programa de conformación realiza la reducción utilizando intrínseca nombre del procedimiento MAX pesar de que el MAX intrínseca ha sido renombrado a MIN.

#### *Ejemplo 37.5f*

MÓDULO MOD

INTRÍNSECO MAX, MIN FIN módulo MOD

PROGRAMA REDUCTION4

USO MOD, MIN => MAX, MAX => MIN real :: R

R = -Enorme (0,0)

! PARALELO $ OMP DO reducción (MIN: R)! todavía lo hace MAX DO I = 1, 1000

R = MIN (R, SIN (REAL (I))) END NO

PRINT \*, R

FIN DEL PROGRAMA REDUCTION4

##### Fortran



El siguiente ejemplo no es conforme porque la inicialización (a = 0) de la lista de elementos original A no está sincronizado con la actualización de un como un resultado del cálculo de reducción en el bucle. Por lo tanto, el ejemplo puede imprimir un valor incorrecto para a.

A evitar este problema, la inicialización del original lista de elementos de una debe completar antes de cualquier actualización de una como resultado de la cláusula de reducción. Esto se puede lograr mediante la adición de una barrera explícito después de la asignación a = 0, o encerrando la asignación a = 0 en un solo directiva (que tiene una barrera implícita), o por la inicialización de un antes del inicio de la región paralela.

C / C ++



*Ejemplo 37.3c*

#include <stdio.h> int main (void)

{

int a, i;

#pragma omp paralelo compartido (a) privado (i)

{

#pragma principal omp a = 0;

// Para evitar las condiciones de carrera, añadir una barrera aquí. omp #pragma para la reducción de (+: a)

for (i = 0; i <10; i ++) {

a + = i;

}

#pragma omp sola

printf ( "Sum es% d \ n", a);

}

}

##### C / C ++



*Ejemplo 37.6f*



INTEGER A, I

##### Fortran

! $ OMP PARALELO COMPARTIDA (A) PRIVADO (I)

! MAESTRO $ OMP

A = 0

! MAESTRO $ OMP FIN

! Para evitar las condiciones de carrera, añadir una barrera aquí.

! $ OMP DO reducción (+: A) DO I = 0, 9

A = A + I FIN DO

! $ OMP SOLA

PRINT \*, "Sum es", A

! $ OMP único extremo

! PARALELO $ OMP FIN

FIN

##### Fortran



## La Cláusula copyin

La cláusula copyin se utiliza para inicializar los datos threadprivate partir de la entrada a una región paralelo. El valor de la variable threadprivate en el hilo principal se copia a la variable threadprivate de cada otro miembro del equipo.

C / C ++



*Ejemplo 38.1c*

#include <stdlib.h> float \* trabajo;

tamaño int; flotar tol;

#pragma omp threadprivate (trabajo, tamaño, tol)

acumulación vacío ()

{

int i;

trabajo = (float \*) malloc (sizeof (float) \* tamaño); for (i = 0; i <tamaño; ++ i) trabajar [i] = tol;

}

void copyin\_example (float t, int n)

{

tol = t; size = n;

#pragma omp copyin paralelo (tol, tamaño)

{

construir();

}

}

##### C / C ++



Fortran



*Ejemplo 38.1f*

MÓDULO M

REAL PUNTERO, SAVE :: TRABAJO (:) INTEGER :: TAMAÑO

Real :: TOL

! $ OMP Threadprivate (TRABAJO, tamaño, TOL) FIN módulo M

COPYIN\_EXAMPLE SUBRUTINA (T, N) USO M

Real :: T INTEGER :: N TOL = T

SIZE = N

! $ OMPPARALLEL Copyin (TOL, tamaño) BUILD LLAMADA

! $ OMPEND PARALELA

FIN SUBRUTINA COPYIN\_EXAMPLE

SUBRUTINA USO BUILD M

ASIGNAR (TRABAJO (tamaño)) TRABAJO = TOL

FIN SUBRUTINA BUILD

##### Fortran



## La Cláusula copyprivate

La cláusula copyprivate se puede utilizar para transmitir valores adquiridos por un solo hilo directamente a todas las instancias de las variables privadas de los otros hilos. En este ejemplo, si la rutina se llama desde la parte secuencial, su comportamiento no se ve afectada por la presencia de las directivas. Si se llama desde una región paralela, a continuación, los argumentos reales con las que A y B están asociados deben ser privadas.

El hilo que ejecuta el bloque estructurado asociado con el único constructo transmite los valores de las variables privadas a, b, x, y y de entorno de datos de su tarea implícita a los entornos de datos de las otras tareas implícitas en el equipo hilo. La transmisión completa antes de cualquiera de los hilos han salido de la barrera en el extremo de la construcción.

C / C ++



*Ejemplo 39.1c*

#include <stdio.h> float x, y;

#pragma omp threadprivate (x, y)

anular init (flotar a, float b) {

#pragma omp copyprivate solo (a, b, x, y)

{

scanf ( "% f% f% f% f", & a, y b, y x, y y);

}

}

##### C / C ++



Fortran



*Ejemplo 39.1f*

SUBRUTINA INIT (A, B) VERDADERO A, B

COMÚN / XY / X, Y

! $ OMPTHREADPRIVATE (/ XY /)

! $ OMPSINGLE

READ (11) A, B, X, Y

! $ OMPEND SOLA COPYPRIVATE (A, B, / XY /)

FIN SUBRUTINA INIT

##### Fortran



En este ejemplo, se supone que la entrada debe ser realizada por el hilo principal. Puesto que el constructo maestro no soporta la cláusula copyprivate, no puede transmitir el valor de entrada que se lee. Sin embargo, copyprivate se utiliza para transmitir una dirección donde se almacena el valor de entrada.

C / C ++



*Ejemplo 39.2c*

#include <stdio.h> #include <stdlib.h>

flotar read\_next) {float \* tmp (; flotar return\_val;

#pragma omp sola copyprivate (TMP)

{

tmp = (float \*) malloc (sizeof (float));

} / \* Copia el puntero solamente \* /

#pragma omp maestro

{

scanf ( "% f", tmp);

}

#pragma barrera omp return\_val = \* tmp; barrera omp #pragma

#pragma omp sola nowait

{

libre (TMP);

}

volver return\_val;

}

##### C / C ++



*Ejemplo 39.2f*



Función real READ\_NEXT () REAL, EL PUNTERO :: TMP

##### Fortran

! $ OMPSINGLE

ASIGNAR (TMP)

! $ OMPEND SOLA COPYPRIVATE (TMP)! copia el puntero solamente

! $ OMPMASTER

READ (11) TMP

! $ OMPEND DOMINAR

! $ OMPBARRIER

READ\_NEXT = TMP

! $ OMPBARRIER

! $ OMPSINGLE

DEALLOCATE (TMP)

! $ OMPEND SOLA NO, ESPERA

FIN DE FUNCIONES READ\_NEXT

##### Fortran



Supongamos que el número de variables de bloqueo requeridas dentro de una región paralela no puede ser fácilmente determinada antes de entrar en él. La cláusula copyprivate se puede utilizar para proporcionar acceso a las variables de bloqueo compartido que se asignan dentro de esa región paralela.

C / C ++



*Ejemplo 39.3c*

#include <stdio.h> #include <stdlib.h> #include <omp.h>

omp\_lock\_t \* new\_lock ()

{

omp\_lock\_t \* lock\_ptr;

#pragma omp copyprivate sola (lock\_ptr)

{

lock\_ptr = (omp\_lock\_t \*) malloc (sizeof (omp\_lock\_t)); omp\_init\_lock (lock\_ptr);

}

volver lock\_ptr;

}

##### C / C ++



Fortran



*Ejemplo 39.3f*

NEW\_LOCK FUNCIÓN ()

UTILIZAR OMP\_LIB! o incluir "omp\_lib.h" INTEGER (OMP\_LOCK\_KIND), PUNTERO :: NEW\_LOCK

! $ OMPSINGLE

ASIGNAR (NEW\_LOCK)

LLAMADA OMP\_INIT\_LOCK (NEW\_LOCK)

! $ OMPEND SOLA COPYPRIVATE (NEW\_LOCK) NEW\_LOCK END FUNCTION

Tenga en cuenta que el efecto de la cláusula copyprivate en una variable con el atributo asignable es diferente que en una variable con el atributo puntero. El valor de A se copia (como por asignación intrínseca) y se copia el puntero B (como por asignación de puntero) para los elementos de la lista correspondientes en las otras tareas implícitas que pertenecen a la región paralela.

*Ejemplo 39.4f*

SUBRUTINA S (N) INTEGER N

REAL DIMENSION (:), ALLOCATABLE :: Un REAL DIMENSION (:), PUNTERO :: B

ASIGNAR (A (n))

! $ OMPSINGLE

ASIGNAR (B (N)) READ (11) A, B

! $ OMPEND SOLA COPYPRIVATE (A, B)

! Una variable es privado y está

! asignado el mismo valor en cada hilo

! Variable B es compartida

! $ OMPBARRIER

! $ OMPSINGLE DEALLOCATE (SEGUNDO)

! $ OMPEND SOLA NO, ESPERA SUBRUTINA FIN S

##### Fortran



## Bucle anidado construcciones

El siguiente ejemplo de anidación construcción de lazo es conforme porque las regiones interior y exterior de bucle se unen a diferentes regiones paralelas:

C / C ++



*Ejemplo 40.1c*

trabajo void (int i, int j) {} good\_nesting void (int n)

{

int i, j;

#pragma omp predeterminado paralelo (compartido)

{

#pragma omp para

for (i = 0; i <n; i ++) {

#pragma omp paralelo compartido (i, n)

{

#pragma omp para

para (j = 0; j <n; j ++) trabajo (i, j);

}

}

}

}

##### C / C ++



*Ejemplo 40.1f*



TRABAJO SUBRUTINA (I, J) INTEGER I, J

FIN DE TRABAJO SUBRUTINA

GOOD\_NESTING SUBRUTINA (N) INTEGER N

entero i

! $ OMPPARALLEL DEFAULT (compartida)

! $ OMPDO

DO I = 1, N

! $ OMPPARALLEL Compartido (I, N)

! $ OMPDO

DO J = 1, N

Identificador de llamada (I, J)

##### Fortran

FIN DO

! $ OMPEND FIN PARALELO HACER

! $ OMPEND PARALELA

FIN SUBRUTINA GOOD\_NESTING

##### Fortran



La siguiente variación del ejemplo anterior también es conforme:

C / C ++



*Ejemplo 40.2c*

trabajo void (int i, int j) {}

void trabajo 1 (int i, int n)

{

int j;

#pragma omp predeterminado paralelo (compartido)

{

#pragma omp para

para (j = 0; j <n; j ++) trabajo (i, j);

}

}

void good\_nesting2 (int n)

{

int i;

#pragma omp predeterminado paralelo (compartido)

{

#pragma omp para

for (i = 0; i <n; i ++) trabajo1 (i, n);

}

}

##### C / C ++



*Ejemplo 40.2f*



TRABAJO SUBRUTINA (I, J) INTEGER I, J

FIN DE TRABAJO SUBRUTINA

SUBRUTINA Trabajo1 (I, N) INTEGER J

! $ OMP PARALELO DEFAULT (Compartido)

! $ OMP DO

DO J = 1, N

Identificador de llamada (I, J) END DO

! PARALELO $ OMP FIN

FIN SUBRUTINA Trabajo1

##### Fortran

GOOD\_NESTING2 SUBRUTINA (N) INTEGER N

! $ OMP PARALELO DEFAULT (Compartido)

! $ OMP DO

DO I = 1, N

LLAMADA Trabajo1 (I, N) END DO

! PARALELO $ OMP FIN

FIN SUBRUTINA GOOD\_NESTING2

##### Fortran



## restricciones el anidamiento de las Regiones

Los ejemplos en esta sección ilustran las reglas de anidación región.

El siguiente ejemplo no es conforme porque las regiones interior y exterior de bucle están estrechamente anidados:

C / C ++



*Ejemplo 41.1c*

trabajo void (int i, int j) {} wrong1 void (int n)

{

#pragma omp predeterminado paralelo (compartido)

{

int i, j; #pragma omp para

for (i = 0; i <n; i ++) {

/ \* Anidamiento incorrecto de OMP bucle regiones \* / #pragma para

para (j = 0; j <n; j ++) trabajo (i, j);

}

}

}

##### C / C ++



Fortran



*Ejemplo 41.1f*

TRABAJO SUBRUTINA (I, J) INTEGER I, J

FIN SUBRUTINA TRABAJO SUBRUTINA WRONG1 (N) INTEGER N

INTEGER I, J

! $ OMPPARALLEL DEFAULT (compartida)

! $ OMPDO

DO I = 1, N

! $ OMPDO! anidación incorrecta de bucle regiones DO J = 1, N

Identificador de llamada (I, J) END DO

FIN DO

! $ OMPEND PARALELA

FIN SUBRUTINA WRONG1

##### Fortran



La siguiente versión huérfanos del ejemplo anterior es también Disconformes:

C / C ++



*Ejemplo 41.2c*

trabajo void (int i, int j) {} void trabajo 1 (int i, int n)

{

int j;

/ \* Anidamiento incorrecto de OMP bucle regiones \* / #pragma para

para (j = 0; j <n; j ++) trabajo (i, j);

}

void wrong2 (int n)

{

#pragma omp predeterminado paralelo (compartido)

{

int i;

#pragma omp para

for (i = 0; i <n; i ++) trabajo1 (i, n);

}

}

##### C / C ++



Fortran



*Ejemplo 41.2f*

SUBRUTINA Trabajo1 (I, N) INTEGER I, N

INTEGER J

! $ OMP hacer! anidación incorrecta de regiones de bucle DO J = 1, N

Identificador de llamada (I, J) END DO

FIN SUBRUTINA Trabajo1 SUBRUTINA WRONG2 (N) INTEGER N

entero i

! $ OMP PARALELO DEFAULT (Compartido)

! $ OMP DO

DO I = 1, N

LLAMADA Trabajo1 (I, N) END DO

! PARALELO $ OMP FIN

FIN SUBRUTINA WRONG2

##### Fortran



El siguiente ejemplo no es conforme porque el bucle y regiones individuales están estrechamente anidados:

C / C ++



*Ejemplo 41.3c*

trabajo void (int i, int j) {} wrong3 void (int n)

{

#pragma omp predeterminado paralelo (compartido)

{

int i;

#pragma omp para

for (i = 0; i <n; i ++) {

/ \* Anidamiento incorrecto de las regiones \* / #pragma omp sola

trabajo (i, 0);

}

}

}

##### C / C ++



Fortran



*Ejemplo 41.3f*

WRONG3 SUBRUTINA (N) INTEGER N

entero i

! $ OMPPARALLEL DEFAULT (compartida)

! $ OMPDO

DO I = 1, N

! $ OMPSINGLE! anidamiento incorrecto de regiones Identificador de llamada (I, 1)

! $ OMPEND SOLTERO FIN DO

! $ OMPEND PARALELA

FIN SUBRUTINA WRONG3

##### Fortran



El siguiente ejemplo no es conforme porque una región de barrera no puede ser estrechamente anidado dentro de una región de bucle:

C / C ++



*Ejemplo 41.4c*

trabajo void (int i, int j) {} wrong4 void (int n)

{

#pragma omp predeterminado paralelo (compartido)

{

int i;

#pragma omp para

for (i = 0; i <n; i ++) {trabajo (i, 0);

/ \* Anidación incorrecta de región de barrera en una barrera omp \* / #pragma región de bucle

trabajo (i, 1);

}

}

}

##### C / C ++



Fortran



*Ejemplo 41.4f*

WRONG4 SUBRUTINA (N) INTEGER N

entero i

! $ OMPPARALLEL DEFAULT (compartida)

! $ OMPDO

DO I = 1, N

Identificador de llamada (I, 1)

! anidación incorrecta de región de barrera en una región de bucle

! $ OMPBARRIER

Identificador de llamada (I, 2) END DO

! $ OMPEND PARALELA

FIN SUBRUTINA WRONG4

##### Fortran



El siguiente ejemplo no es conforme, porque la región de barrera no puede ser estrechamente anidado dentro de la región crítica. Si esto se permite, que se traduciría en un punto muerto debido al hecho de que sólo un hilo a la vez puede entrar en la región crítica:

C / C ++



*Ejemplo 41.5c*

trabajo void (int i, int j) {} wrong5 void (int n)

{

#pragma omp paralelo

{

#pragma omp crítica

{

trabajo (n, 0);

/ \* Anidación incorrecta de región de barrera en una barrera omp crítico región \* / #pragma

trabajo (n, 1);

}

}

}

##### C / C ++



Fortran



*Ejemplo 41.5f*

WRONG5 SUBRUTINA (N) INTEGER N

! $ OMPPARALLEL DEFAULT (compartida)

! $ OMPCRITICAL

Identificador de llamada (N, 1)

! anidación incorrecta de región de barrera en una región crítica

! $ OMPBARRIER

Identificador de llamada (N, 2)

! $ OMPEND CRÍTICO

! $ OMPEND PARALELA

FIN SUBRUTINA WRONG5

##### Fortran



El siguiente ejemplo no es conforme, porque la región de barrera no puede ser estrechamente anidado dentro de la región individual. Si esto se permite, que se traduciría en un punto muerto debido al hecho de que sólo un hilo ejecuta la única región:

C / C ++



*Ejemplo 41.6c*

trabajo void (int i, int j) {} wrong6 void (int n)

{

#pragma omp paralelo

{

#pragma omp sola

{

trabajo (n, 0);

/ \* Anidación incorrecta de región de barrera en una barrera omp sola región \* / #pragma

trabajo (n, 1);

}

}

}

##### C / C ++



Fortran



*Ejemplo 41.6f*

WRONG6 SUBRUTINA (N) INTEGER N

! $ OMPPARALLEL DEFAULT (compartida)

! $ OMPSINGLE

Identificador de llamada (N, 1)

! anidación incorrecta de región de barrera en una sola región

! $ OMPBARRIER

Identificador de llamada (N, 2)

! $ OMPEND SOLTERO

! $ OMPEND PARALELA

FIN SUBRUTINA WRONG6

##### Fortran



1. **El omp\_set\_dynamic y**

**rutinas omp\_set\_num\_threads**

Algunos programas se basan en un número especificado de antemano fijo de hilos para ejecutar correctamente. Debido a que el ajuste para el ajuste dinámico del número de hilos por defecto se define la aplicación, estos programas pueden optar por desactivar la capacidad de las discusiones dinámicas y establecer el número de hilos de forma explícita para asegurar la portabilidad. El siguiente ejemplo muestra cómo hacer esto utilizando omp\_set\_dynamic y omp\_set\_num\_threads.

En este ejemplo, el programa ejecuta correctamente sólo si se ejecuta por 16 hilos. Si la aplicación no es capaz de soportar 16 hilos, el comportamiento de este ejemplo es definido por la implementación. Tenga en cuenta que el número de hilos de ejecución de una región paralela permanece constante durante la región, independientemente de las discusiones dinámicas de ajuste.

El mecanismo de hilos dinámico determina el número de hilos de rosca para usar en el inicio de la región paralela y la mantiene constante durante la duración de la región.

C / C ++



*Ejemplo 42.1c*

# include <omp.h> #include <stdlib.h>

do\_by\_16 vacío (float \* x, int iam, Int maximiles) {}

dynthreads void (float \* x, int NPOINTS)

{

int iam, maximiles;

omp\_set\_dynamic (0); omp\_set\_num\_threads (16);

#pragma omp paralelo compartido (x, NPOINTS) privada (IAM, maximiles)

{

if (!) = omp\_get\_num\_threads (16) abortan ();

iam = omp\_get\_thread\_num (); maximiles = NPOINTS / 16; do\_by\_16 (x, iam, maximiles);

}

}

##### C / C ++



Fortran



*Ejemplo 42.1f*

SUBRUTINA DO\_BY\_16 (X, IAM, iPoints) real x (\*)

INTEGER IAM, iPoints FIN DO\_BY\_16 SUBRUTINA

DYNTHREADS SUBRUTINAS (X, NPOINTS)

INCLUIR "Omp\_lib.h"! o usar OMP\_LIB INTEGER NPOINTS

Real x (NPOINTS)

INTEGER IAM, iPoints

LLAMADA OMP\_SET\_DYNAMIC (.FALSE.) Omp\_set\_num\_threads de llamada (16)

! $ OMPPARALLEL compartido (X, NPOINTS) PRIVADO (IAM, IPoints) IF (OMP\_GET\_NUM\_THREADS () .NE. 16), entonces

DETENER

TERMINARA SI

IAM = OMP\_GET\_THREAD\_NUM () iPoints = NPOINTS / 16

LLAMADA DO\_BY\_16 (X, IAM, iPoints)

! $ OMPEND PARALELA

DYNTHREADS FIN SUBRUTINAS

##### Fortran



## Los omp\_get\_num\_threads de rutina

En el siguiente ejemplo, los omp\_get\_num\_threads llamada devuelve 1 en la parte secuencial del código, por lo np siempre será igual a 1. Para determinar el número de hilos que se implementarán para la región paralela, la llamada debe estar dentro de la región paralela .

C / C ++



*Ejemplo 43.1c*

# include <omp.h> trabajo vacío (int i);

anulará incorrecta ()

{

int np, i;

np = omp\_get\_num\_threads (); / \* Extraviado \* /

#pragma omp paralelo para el horario (estática) para (i = 0; i <np; i ++)

trabajar (i);

}

##### C / C ++



Fortran



*Ejemplo 43.1f*

TRABAJO SUBRUTINA (I) entero i

I = i + 1

FIN DE TRABAJO SUBRUTINA

SUBRUTINA incorrecto ()

INCLUIR "Omp\_lib.h"! o uso OMP\_LIB INTEGER I, notario público

= NP OMP\_GET\_NUM\_THREADS () fuera de lugar: volverá 1

! $ OMPPARALLEL DO PROGRAMA (estático) DO I = 0, NP-1

Identificador de llamada (I) ENDDO

! $ OMPEND PARALELO HACER

FIN SUBRUTINA INCORRECTO

##### Fortran



El siguiente ejemplo muestra cómo reescribir este programa sin incluir una consulta para el número de hilos:

C / C ++



*Ejemplo 43.2c*

# include <omp.h> trabajo vacío (int i);

anular la correcta ()

{

int i;

#pragma omp paralelo privada (i)

{

i = omp\_get\_thread\_num (); trabajar (i);

}

}

##### C / C ++



*Ejemplo 43.2f*



TRABAJO SUBRUTINA (I) entero i

I = i + 1

FIN DE TRABAJO SUBRUTINA

##### Fortran

SUBRUTINA CORRECTO ()

INCLUIR "Omp\_lib.h"! o usar OMP\_LIB entero i

! $ OMPPARALLEL PRIVADO (I)

I = OMP\_GET\_THREAD\_NUM () identificador de llamada (I)

! $ OMPEND PARALELA

FIN SUBRUTINA CORRECTO

##### Fortran



## La rutina omp\_init\_lock

El siguiente ejemplo demuestra cómo inicializar un conjunto de cerraduras en un paralelo

región mediante el uso de omp\_init\_lock.

##### C / C ++



*Ejemplo 44.1c*

# include <omp.h> omp\_lock\_t \* new\_locks ()

{

int i;

omp\_lock\_t \* Bloqueo = new omp\_lock\_t [1000];

#pragma omp paralelo para privado (i) para (i = 0; i <1,000; i ++)

{

omp\_init\_lock (& ​​Lock [i]);

}

bloqueo de retorno;

}

##### C / C ++



Fortran



*Ejemplo 44.1f*

NEW\_LOCKS FUNCIÓN ()

UTILIZAR OMP\_LIB! o incluir "omp\_lib.h" INTEGER (OMP\_LOCK\_KIND), DIMENSION (1000) :: NEW\_LOCKS

entero i

! $ OMPPARALLEL DO PRIVADO (I) DO I = 1,1000

LLAME OMP\_INIT\_LOCK (NEW\_LOCKS () I) FIN DO

! $ OMPEND PARALELO HACER

NEW\_LOCKS FIN DE FUNCIÓN

##### Fortran



## Propiedad de Esclusas

La propiedad de esclusas ha cambiado desde OpenMP 2.5. En OpenMP 2.5, cerraduras son propiedad de hilos; por lo que un bloqueo dado a conocer por la rutina omp\_unset\_lock debe ser propiedad de la misma rosca ejecución de la rutina. Con OpenMP 3.0, cerraduras son propiedad de las regiones de tareas; por lo que un bloqueo dado a conocer por la rutina omp\_unset\_lock en una región tarea debe ser propiedad de la misma región tarea.

Este cambio en la propiedad requiere un cuidado especial cuando se utilizan los bloqueos. El programa siguiente es conforme en OpenMP 2.5 porque el hilo que libera la lck de bloqueo en la región paralela es el mismo hilo que adquirió la cerradura en la parte secuencial del programa (rosca maestro de región paralela y la rosca inicial son los mismos). Sin embargo, no es conforme en OpenMP 3.0 y 3.1, ya que la región tarea que libera el bloqueo lck es diferente de la región tarea que adquiere el bloqueo.

C / C ++



*Ejemplo 45.1c*

#include <stdlib.h> #include <stdio.h> #include <omp.h>

int main ()

{

int x; lck omp\_lock\_t;

omp\_init\_lock (y lck); omp\_set\_lock (y lck); x = 0;

#pragma omp paralelo compartido (x)

{

#pragma omp maestro

{

x = x + 1; omp\_unset\_lock (y lck);

}

/ \* Algunas cosas más. \* /

}

omp\_destroy\_lock (y lck); return 0;

}

##### C / C ++



Fortran



*Ejemplo 45.1f*

programa utilice el bloqueo del número entero omp\_lib :: x

número entero (tipo = omp\_lock\_kind) :: lck

llamar omp\_init\_lock (LCK) llamada omp\_set\_lock (LCK)

x = 0

! $ OMP paralelo compartido (x)

! $ OMP maestro

x = x + 1

llamar omp\_unset\_lock (LCK)

! $ Maestro extremo omp

!Algo mas cosas.

! $ Final omp paralelo

llamar omp\_destroy\_lock extremo (lck)

##### Fortran



## Las rutinas de bloqueo sencillo

En el siguiente ejemplo, las rutinas de bloqueo hacen que los hilos estén en reposo a la espera de la entrada a la primera sección crítica, pero que se pueden hacer otras tareas mientras se espera para la entrada a la segunda. Los bloques de función omp\_set\_lock, pero no, lo que permite el trabajo en el salto a realizar la función omp\_test\_lock.

C / C ++



Tenga en cuenta que el argumento de las rutinas de bloqueo debe tener omp\_lock\_t tipo, y que no hay necesidad de tirar de la cadena.

*Ejemplo 46.1c*

#include <stdio.h> #include <omp.h> vacío de salto (int i) {} trabajo vacío (int i) {} int main ()

{

lck omp\_lock\_t; Identificación del int;

omp\_init\_lock (y lck);

#pragma omp paralelo compartida (LCK) privada (id)

{

id = omp\_get\_thread\_num ();

omp\_set\_lock (y lck);

/ \* Sólo un hilo a la vez puede ejecutar esta printf \* / printf ( "Mi ID del tema es% d. \ N", id); omp\_unset\_lock (y lck);

while (! omp\_test\_lock (y LCK)) {

skip (id); / \* que aún no tenemos la bloquear,

por lo que tenemos que hacer otra cosa \* /

}

trabajo (id); / \* ahora tenemos la bloquear

y puede hacer el trabajo \* /

omp\_unset\_lock (y lck);

}

omp\_destroy\_lock (y lck);

return 0;

}

##### C / C ++



Fortran



Tenga en cuenta que no hay necesidad de eliminar la variable de cerradura.

*Ejemplo 46.1f*

SUBRUTINA SKIP (ID) SKIP FIN SUBRUTINA

TRABAJO SUBRUTINA (ID) TRABAJO FIN SUBRUTINA

PROGRAMA SIMPLELOCK

INCLUIR "Omp\_lib.h"! o usar OMP\_LIB INTEGER (OMP\_LOCK\_KIND) LCK

ID INTEGER

LLAMADA OMP\_INIT\_LOCK (LCK)

! $ OMPPARALLEL compartido (LCK) PRIVADO (ID) ID = OMP\_GET\_THREAD\_NUM ()

LLAMADA OMP\_SET\_LOCK (LCK)

IMPRESIÓN \*, 'Mi ID del tema es', ID OMP\_UNSET\_LOCK CALL (LCK)

DO WHILE (.NOT. OMP\_TEST\_LOCK (LCK))

LLAMADA SKIP (ID)! Todavía no tenemos la bloquear

! por lo que tenemos que hacer otra cosa

FIN DO

LLAMADA TRABAJO (ID)! Ahora tenemos la bloquear

! y puede hacer lo OMP\_UNSET\_LOCK identificador de llamada (LCK)

! $ OMPEND PARALELA

LLAMADA OMP\_DESTROY\_LOCK (LCK)

SIMPLELOCK FIN DEL PROGRAMA

##### Fortran



## Las rutinas de bloqueo encajables

El siguiente ejemplo demuestra cómo una cerradura encajables se puede utilizar para sincronizar las actualizaciones tanto a una estructura de conjunto y de uno de sus miembros.

C / C ++



*Ejemplo 47.1c*

# include <omp.h> typedef struct {

int a, b;

lck omp\_nest\_lock\_t; } par;

int trabajo 1 (); int work2 (); int Trabajo3 ();

incr\_a void (\* p par, int a)

{

/ \* Llamado sólo de incr\_pair, no hay necesidad de bloquear. \* / P> a + = a;

}

void incr\_b (par \* p, int b)

{

/ \* Llamado tanto desde incr\_pair y en otros lugares, \* /

/ \* Por lo tanto necesitan un bloqueo encajables. \* /

omp\_set\_nest\_lock (y p-> LCK); p-> b + = b;

omp\_unset\_nest\_lock (y p-> LCK);

}

incr\_pair void (\* p par, int a, int b)

{

omp\_set\_nest\_lock (y p-> LCK); incr\_a (p, a);

incr\_b (p, b); omp\_unset\_nest\_lock (y p-> LCK);

}

void nestlock (par p \*)

{

#pragma omp secciones paralelas

{

sección #pragma omp

incr\_pair (p, trabajo1 (), work2 ()); sección #pragma omp

incr\_b (p, Trabajo3 ());

}

}

##### C / C ++

Fortran



*Ejemplo 47.1f*

MÓDULO DE DATOS

USO OMP\_LIB, SOLAMENTE: OMP\_NEST\_LOCK\_KIND TIPO LOCKED\_PAIR

INTEGER un entero B

INTEGER (OMP\_NEST\_LOCK\_KIND) TIPO LCK FIN

MÓDULO DE FIN DE DATOS

SUBRUTINA INCR\_A (P, A)

! llama sólo de INCR\_PAIR, no hay necesidad de bloquear datos sobre el uso

TIPO (LOCKED\_PAIR) :: P INTEGER A

P% A = P% A + A

FIN SUBRUTINA INCR\_A

SUBRUTINA INCR\_B (P, B)

! llamado desde ambos INCR\_PAIR y en otros lugares,

! por lo que necesitamos una cerradura encajable

UTILIZAR OMP\_LIB! o incluir Datos de uso "omp\_lib.h"

TIPO (LOCKED\_PAIR) :: P INTEGER B

LLAMADA OMP\_SET\_NEST\_LOCK (P% LCK) P% B = P% B + B

LLAMADA OMP\_UNSET\_NEST\_LOCK (P% LCK) INCR\_B FIN SUBRUTINA

SUBRUTINA INCR\_PAIR (P, A, B)

UTILIZAR OMP\_LIB! o incluir Datos de uso "omp\_lib.h"

TIPO (LOCKED\_PAIR) :: P INTEGER A

entero B

LLAMADA OMP\_SET\_NEST\_LOCK (P% LCK) Llamada INCR\_A (P, A)

LLAMADA INCR\_B (P, B)

LLAMADA OMP\_UNSET\_NEST\_LOCK (P% LCK) INCR\_PAIR FIN SUBRUTINA

NESTLOCK SUBRUTINA (P)

UTILIZAR OMP\_LIB! o incluir Datos de uso "omp\_lib.h"

TIPO (LOCKED\_PAIR) :: P INTEGER Trabajo1, Trabajo2, Trabajo3 EXTERNO Trabajo1, Trabajo2, Trabajo3

! $ OMPPARALLEL SECCIONES

! $ OMPSECTION

LLAMADA INCR\_PAIR (P, Trabajo1 (), Trabajo2 ())

! $ OMPSECTION

LLAMADA INCR\_B (P, Trabajo3 ())

! $ OMPEND PARALELO SECCIONES

FIN SUBRUTINA NESTLOCK

##### Fortran



## Construir objetivo

**Construir objetivo en Construct paralelo**

Este ejemplo siguiente muestra cómo el constructo diana descarga a una región código a un dispositivo de destino. Las variables p, v1, v2, y N se asignan implícitamente al dispositivo del objetivo.

*Ejemplo 48.1c*



C / C ++

extern void init (float \*, float \*, int); extern salida nula (float \*, int);

void vec\_mult (int N)

{

int i;

flotar p [N], v1 [N], v2 [N]; init (v1, v2, N);

objetivo omp #pragma

#pragma omp paralelo para privado (i) para (i = 0; i <N; i ++)

p [i] = v1 [i] \* v2 [i]; de salida (p, N);

}

##### C / C ++



*Ejemplo 48.1f*

vec\_mult subrutina (N) número entero :: i, N

real :: p (N), v1 (N), v2 (N) llamar a init (v1, v2, N)

! $ OMP objetivo

! $ OMP paralelo hagas i = 1, N

p (i) = v1 (i) \* v2 (i) final hacer

! Salida de llamada de destino final $ OMP (p, n)

subrutina final



##### Fortran

Fortran



**constructo objetivo con Cláusula mapa**

Este ejemplo siguiente muestra cómo el constructo diana descarga a una región código a un dispositivo de destino. Las variables p, v1, v2, se asignan explícitamente al dispositivo del objetivo utilizando la cláusula mapa. La variable N se asigna implícitamente al dispositivo de destino.

*Ejemplo 48.2c*



C / C ++

extern void init (float \*, float \*, int); extern salida nula (float \*, int);

void vec\_mult (int N)

{

int i;

flotar p [N], v1 [N], v2 [N]; init (v1, v2, N);

mapa de destino omp #pragma (v1, v2, p) #pragma omp paralelo para

for (i = 0; i <N; i ++) p [i] = v1 [i] \* v2 [i];

de salida (p, N);

}

##### C / C ++



*Ejemplo 48.2f*

vec\_mult subrutina (N) número entero :: i, N

real :: p (N), v1 (N), v2 (N) llamar a init (v1, v2, N)

! Mapa de destino $ OMP (v1, v2, p)

! $ OMP paralelo hagas i = 1, N

p (i) = v1 (i) \* v2 (i) final hacer

! Salida de llamada de destino final $ OMP (p, n)

subrutina final



##### Fortran

Fortran



**Cláusula mapa con a / de mapa-tipos**

El ejemplo siguiente muestra cómo el constructo diana descarga a una región código a un dispositivo de destino. En la cláusula mapa, las hacia y desde mapa-tipos definen la correspondencia entre los datos originales (host) y los datos de destino (dispositivo). El tipo de mapa que especifica que los datos sólo se pueden leer en el dispositivo, y la del mapa de tipo especifica que los datos sólo se escribirán en el dispositivo. Mediante la especificación de un acceso garantizado en el dispositivo, las transferencias de datos pueden ser reducidos para la región diana.

El al mapa de tipo indica que en el inicio de la región diana de las variables V1 y V2 se inicializan con los valores de las variables correspondientes en el dispositivo anfitrión, y al final de la región de destino de las variables V1 y V2 no están asignados a sus correspondientes variables en el dispositivo host.

El de mapa de tipo indica que en el inicio de la región diana de la variable p no se inicializa con el valor de la variable correspondiente en el dispositivo anfitrión, y al final de la región diana se asigna la variable p a la variable correspondiente en el dispositivo host.

*Ejemplo 48.3c*



C / C ++

extern void init (float \*, float \*, int); extern salida nula (float \*, int);

void vec\_mult (int N)

{

int i;

flotar p [N], v1 [N], v2 [N]; init (v1, v2, N);

mapa de destino omp #pragma (a: v1, v2) mapa (a partir de: p) #pragma omp paralelo para

for (i = 0; i <N; i ++) p [i] = v1 [i] \* v2 [i];

de salida (p, N);

}

##### C / C ++



Fortran



*Ejemplo 48.3f*

El mapa hacia y desde los tipos permiten a los programadores para optimizar el movimiento de datos. Puesto que los datos para los v arrays no se devuelven, y los datos para la matriz p no se transfieren al dispositivo, sólo la mitad de los datos se mueven, en comparación con el comportamiento por defecto de una asignación implícita.

vec\_mult subrutina (N) número entero :: i, N

real :: p (N), v1 (N), v2 (N) llamar a init (v1, v2, N)

Mapa del objetivo $ OMP (a: v1, v2) mapa (a partir de: p)

! $ OMP paralelo hagas i = 1, N

p (i) = v1 (i) \* v2 (i) final hacer

! Salida de llamada de destino final $ OMP (p, n)

subrutina final

##### Fortran



**Cláusula mapa con las secciones de matriz**

El ejemplo siguiente muestra cómo el constructo diana descarga a una región código a un dispositivo de destino. En la cláusula mapa, mapa-tipos se utilizan para optimizar la asignación de variables para el dispositivo de destino. Dado que las variables p, v1 y v2 son punteros, sección de notación de matriz debe ser utilizado para mapear las matrices. La notación: N es equivalente a 0: N.

*Ejemplo 48.4c*



C / C ++

extern void init (float \*, float \*, int); extern salida nula (float \*, int);

vec\_mult void (float \* p, float \* v1, float \* v2, int N)

{

int i;

init (v1, v2, N);

mapa de destino omp #pragma (a: v1 [0: N], v2 [: N]) mapa (de: p [0: N]) #pragma omp paralelo para

for (i = 0; i <N; i ++) p [i] = v1 [i] \* v2 [i];

de salida (p, N);

}

##### C / C ++



Fortran



*Ejemplo 48.4f*

En C, la longitud de la punta a matriz debe ser especificado. En Fortran la medida de la matriz es conocida y la longitud no tiene que ser especificado. Una sección de la matriz se puede especificar con la sintaxis de Fortran usual, como se muestra en el siguiente ejemplo. El valor 1 se asume para el límite inferior para la sección de matriz v2 (: N).

multiplicadores módulo contiene

vec\_mult subrutina (p, v1, v2, N) real, puntero, dimensión (:) :: p, v1, número entero v2 :: N, i

llamar a init (v1, v2, N)

Mapa del objetivo $ OMP (a: v1 (1: N), V2 (N)) mapa (a partir de: p (1: N))

! $ OMP paralelo hagas i = 1, N

p (i) = v1 (i) \* v2 (i) final hacer

! Salida de llamada de destino final $ OMP (p, n)

extremo del módulo final subrutina

Una situación más realista en la que se pasa una matriz de tamaño supone que vec\_mult requiere que se especifique la longitud de las matrices, debido a que el compilador no sabe el tamaño del almacenamiento. Una sección de la matriz se debe especificar con la sintaxis Fortran usual, como se muestra en el siguiente ejemplo. El valor 1 se asume para el límite inferior para la sección de matriz v2 (: N).

multiplicadores módulo contiene

vec\_mult subrutina (p, v1, v2, N) real, dimensión (\*) :: p, v1, v2 entero:: N, i

llamar a init (v1, v2, N)

Mapa del objetivo $ OMP (a: v1 (1: N), V2 (N)) mapa (a partir de: p (1: N))

! $ OMP paralelo hagas i = 1, N

p (i) = v1 (i) \* v2 (i) final hacer

llamar de salida (p, N)

! $ OMP final subrutina final de destino

módulo final

##### Fortran



**constructo objetivo con Si cláusula**

El ejemplo siguiente muestra cómo el constructo diana descarga a una región código a un dispositivo de destino.

El si cláusula en el constructo diana indica que si la variable N es menor que un umbral dado, a continuación, la región diana será ejecutado por el dispositivo host.

El si cláusula en el constructo paralelo indica que si la variable N es menor que un segundo umbral, entonces la región paralelo está inactivo.

*Ejemplo 48.5c*



C / C ++

#define Threshold1 1000000

#define threshold2 1000

extern void init (float \*, float \*, int); extern salida nula (float \*, int);

vec\_mult void (float \* p, float \* v1, float \* v2, int N)

{

int i;

init (v1, v2, N);

objetivo #pragma omp si (N> Threshold1) mapa (a: v1 [0: N], v2 [: N]) \ map (de: p [0: N])

#pragma omp paralelo para si (N> threshold2) para (i = 0; i <N; i ++)

p [i] = v1 [i] \* v2 [i]; de salida (p, N);

}

##### C / C ++



Fortran



*Ejemplo 48.5f*

params módulo

número entero, el parámetro :: Threshold1 = 1.000.000, THRESHHOLD2 = 1000

módulo final

subrutina vec\_mult (p, v1, v2, N) utiliza params

real :: p (N), v1 (N), v2 (N) número entero :: i

llamar a init (v1, v2, N)

! Objetivo de $ OMP si (N> THRESHHOLD1) mapa (a: v1, v2) mapa (a partir de: p)

! $ OMP paralelo hacer si (N> threshold2) hacer i = 1, N

p (i) = v1 (i) \* v2 (i)

final hacer

! $ Objetivo final omp

salida de llamada (p, N) subrutina final

##### Fortran



## Constructo de datos de destino

**Constructo de datos de destino sencilla**

Este ejemplo muestra cómo los datos de destino construir variables mapas a un entorno de datos del dispositivo. La construcción de datos de destino crea un nuevo entorno de los datos del dispositivo y los mapas de las variables v1, v2, yp al nuevo entorno de datos del dispositivo. El constructo diana cerrado en la región de datos de destino crea un nuevo entorno de datos del dispositivo, que hereda las variables v1, v2, y p desde el entorno de datos del dispositivo de cerramiento. La variable N es mapeado en el nuevo entorno de datos del dispositivo de entorno de datos de la tarea encontrar.

*Ejemplo 49.1c*



C / C ++

extern void init (float \*, float \*, int); extern salida nula (float \*, int);

vec\_mult void (float \* p, float \* v1, float \* v2, int N)

{

int i;

init (v1, v2, N);

mapa #pragma omp de datos de destino (a: v1 [0: N], v2 [: N]) mapa (de: p [0: N])

{

#pragma omp #pragma objetivo omp paralelo para for (i = 0; i <N; i ++)

p [i] = v1 [i] \* v2 [i];

}

de salida (p, N);

}

##### C / C ++



Fortran



*Ejemplo 49.1f*

El código Fortran pasa una referencia y especifica el alcance de las matrices en la declaración. No información de longitud es necesario en la cláusula de mapa, como se requiere con / punteros C C ++.

vec\_mult subrutina (p, v1, v2, N) real :: p (N), v1 (N), v2 (N) número entero :: i

llamar a init (v1, v2, N)

Mapa de datos de destino $ OMP (a: v1, v2) mapa (a partir de: p)

! $ OMP objetivo

! $ OMP paralelo hagas i = 1, N

final hacer

! $ Objetivo final omp

p (i) = v1 (i) \* v2 (i)

! $ Datos de destino final omp llaman salida (p, n)

subrutina final

##### Fortran



**Adjuntando datos de destino múltiple Región**

**Regiones de destino**

Los siguientes ejemplos muestran cómo los datos de destino construir variables mapas a un entorno de datos del dispositivo de una región objetivo. La construcción de datos de destino crea un entorno de los datos del dispositivo y encierra las regiones de destino, los cuales tienen sus propios entornos de datos de dispositivo. El entorno de datos del dispositivo de la región de datos de destino se hereda por el entorno de datos del dispositivo de una región objetivo cerrado. El constructo de datos de destino se utiliza para crear las variables que persistirán en toda la región de datos de destino.

En el siguiente ejemplo de las variables V1 y V2 se asignan a cada constructo diana. En lugar de la cartografía de la variable p dos veces, una vez en cada constructo diana, p se asigna una vez por la construcción de datos de destino.

*Ejemplo 49.2c*



C / C ++

extern void init (float \*, float \*, int); extern init\_again vacío (float \*, float \*, int); extern salida nula (float \*, int);

vec\_mult void (float \* p, float \* v1, float \* v2, int N)

{

int i;

init (v1, v2, N);

mapa #pragma omp de datos de destino (a partir de: p [0: N])

{

mapa de destino omp #pragma (a: v1 [: N], v2 [: N]) #pragma omp paralelo para

for (i = 0; i <N; i ++)

p [i] = v1 [i] \* v2 [i]; init\_again (v1, v2, N);

mapa de destino omp #pragma (a: v1 [: N], v2 [: N]) #pragma omp paralelo para

for (i = 0; i <N; i ++)

p [i] = p [i] + (v1 [i] \* v2 [i]);

}

de salida (p, N);

}

##### C / C ++



Fortran



*Ejemplo 49.2f*

El código Fortran utiliza referencia y especifica la medida de las matrices de p, V1 y V2. No información de longitud es necesario en la cláusula de mapa, como se requiere con / punteros C C ++. El v1 arrays y v2 se asignan a cada constructo diana. En lugar de la cartografía de la matriz de p dos veces, una vez en cada constructo diana, p se asigna una vez por la construcción de datos de destino.

vec\_mult subrutina (p, v1, v2, N) real :: p (N), v1 (N), v2 (N) número entero :: i

llamar a init (v1, v2, N)

Mapa de datos de destino $ OMP (a partir de: p)

Mapa del objetivo $ OMP (a: v1, v2)

! $ OMP paralelo hagas i = 1, N

p (i) = v1 (i) \* v2 (i) final hacer

! $ Objetivo final omp

llamar init\_again (v1, v2, N)

Mapa del objetivo $ OMP (a: v1, v2)

! $ OMP paralelo hagas i = 1, N

p (i) = P (i) + v1 (i) \* v2 (i) final hacer

! $ Objetivo final omp

! $ Datos de destino final omp llaman salida (p, n)

subrutina final

##### Fortran



C / C ++



En el siguiente ejemplo, los valores por defecto tmp variables para tofrom mapa de tipo y se asigna a cada constructo diana. La matriz Q se asigna una vez en la región de datos de destino que encierra en lugar de en cada constructo diana.

#include <math.h>

gramschmidt vacío (restringir el flotador Q [] [COLS], const int filas, const int cols)

{

#pragma omp mapa de datos de destino (Q [0: filas] [0: cols]) for (int k = 0; k <cols; k ++)

{

doble tmp = 0,0; objetivo omp #pragma

#pragma omp paralelo para la reducción de (+: tmp) for (int i = 0; i <filas; i ++)

tmp + = (Q [i] [k] \* Q [i] [k]); tmp = 1 / sqrt (TMP);

#pragma omp #pragma objetivo omp paralelo para

for (int i = 0; i <filas; i ++) Q [i] [k] \* = tmp;

}

}

##### C / C ++



Fortran



*Ejemplo 49.3f*

En el siguiente ejemplo las matrices v1 y v2 se asignan a cada constructo diana. En lugar de la cartografía de la matriz Q dos veces en cada constructo diana, Q se asigna una vez por la construcción de datos de destino. Nota, la variable tmp se reasigna implícitamente para cada región diana, mapeando el valor desde el dispositivo para el huésped al final de la primera región diana, y desde el host al dispositivo para la segunda región diana.

subrutina gramschmidt (Q, filas, cols) Entero :: filas, cols, i, k doble de precisión :: Q (filas, collados), tmp

Mapa de datos de destino $ OMP (Q) de hacer k = 1, collados

tmp = 0.0d0

! $ OMP objetivo

! $ OMP paralelo hacer reducción (+: tmp) hacer i = 1, filas

tmp = tmp + (Q (i, k) \* Q (i, k)) final hacer

! $ Objetivo final omp

tmp = 1.0d0 / sqrt (tmp)

! $ OMP objetivo

! $ OMP paralelo hagas i = 1, filas

Q (i, k) = Q (i, k) \* tmp enddo

! $ Objetivo final omp

final hacer

! $ OMP final subrutina extremo datos de destino

##### Fortran



**Construir datos de destino con Call Huérfano**

Los dos ejemplos siguientes muestran cómo los datos de destino construir variables mapas a un entorno de datos del dispositivo. entorno de datos del dispositivo del constructo de datos de destino encierra entorno de datos del dispositivo de la construcción diana en el vec\_mult función ().

Cuando el tipo de la variable que aparece en una sección de matriz es puntero, la variable puntero y la ubicación de almacenamiento de la sección de matriz correspondiente se asignan al entorno de datos del dispositivo. La variable puntero se trata como si hubiera aparecido en una cláusula mapa con un mapa de tipo de alloc. ubicación de almacenamiento de la sección de matriz se asigna de acuerdo con el tipo de mapa en la cláusula mapa (el mapa de tipo predeterminado es tofrom).

entorno de datos de dispositivo del constructo objetivo hereda las ubicaciones de almacenamiento de las secciones de matriz v1 [0: N], v2 [: N], y p0 [0: n] de entorno de datos del dispositivo del constructo de datos de destino que encierra. Ni la inicialización ni asignación se lleva a cabo para las secciones de matriz en el nuevo entorno de datos del dispositivo.

El variables de puntero p1, v3, y v4 son mapeados en el entorno de los datos del dispositivo del constructo diana con una implícita mapa de tipo de alloc y se les asigna la dirección de la ubicación de almacenamiento asociada con sus secciones de matriz correspondientes. Tenga en cuenta que los siguientes pares de ubicaciones de almacenamiento de la sección de gama son equivalentes (p0 [: N], p1 [: N]), (v1 [: N], v3 [: N]), y (v2 [: N], v4 [ :NORTE]).

*Ejemplo 49.3c*



C / C ++

vec\_mult vacío (float \*, \* float, float \*, int); extern void init (float \*, float \*, int); extern salida nula (float \*, int);

void foo (float \* p0, float \* v1, float \* v2, int N)

{

init (v1, v2, N);

mapa #pragma omp de datos de destino (a: v1 [0: N], v2 [: N]) mapa (a partir de: p0 [0: N])

{

vec\_mult (p0, V1, V2, N);

}

de salida (p0, N);

}

vec\_mult void (float \* p1, float \* v3, float \* v4, int N)

{

mapa de destino omp #pragma (a: v3 [0: N], v4 [: N]) mapa (a partir de: p1 [0: N]) #pragma omp paralelo para

for (i = 0; i <N; i ++) p1 [i] = v3 [i] \* v4 [i];

}

##### C / C ++



Fortran



*Ejemplo 49.4f*

El código Fortran mapea los punteros y de almacenamiento de una manera idéntica (misma medida, pero utiliza los índices de 1 a N).

entorno de datos de dispositivo del constructo objetivo hereda los lugares de almacenamiento de los v1 arrays, V2 y p0 de entorno de datos del dispositivo de las construcciones de datos de destino de encerramiento. Sin embargo, en Fortran es conocida los datos asociados del puntero, y no se requiere la forma.

El variables de puntero p1, v3, y v4 son mapeados en el entorno de los datos del dispositivo del constructo diana con una implícita mapa de tipo de alloc y se les asigna la dirección de la ubicación de almacenamiento asociada con sus secciones de matriz correspondientes. Tenga en cuenta que el siguiente par de ubicaciones de almacenamiento de matriz son equivalentes (P0, P1), (V1, V3), y (v2, v4).

multiplicadores módulo contiene

Foo subrutina (p0, V1, V2, N) real, puntero, de dimensión (:) :: p0, v1, número entero v2 :: N, i

llamar a init (v1, v2, N)

! $ OMP mapa de datos de destino (a: v1, v2) mapa (a partir de: P0) llamada vec\_mult (P0, v1, v2, N)

FIN omp meta de producción llamada de datos (P0, N)

subrutina final

vec\_mult subrutina (p1, V3, V4, N) real, puntero, de dimensión (:) :: p1, v3, número entero v4 :: N, i

Mapa del objetivo $ OMP (a: V3, V4) mapa (de p1)

! $ OMP paralelo hagas i = 1, N

P1 (i) = v3 (i) \* v4 (i) finales hacer

! $ OMP final subrutina final de destino

módulo final

##### Fortran



*Ejemplo 49.4c*



C / C ++

En el siguiente ejemplo, las variables p1, v3, y v4 son referencias a las variables de puntero P0, v1 y v2, respectivamente. entorno de datos de dispositivo del constructo objetivo hereda las variables de puntero P0, v1, v2 y de entorno de datos del dispositivo del constructo de datos de destino de encerramiento. Por lo tanto, p1, V3 y V4 están ya presentes en el entorno de datos del dispositivo.

vec\_mult vacío (float \* Y, flotador y \*, \* y flotar, int &); extern void init (float \*, float \*, int);

extern salida nula (float \*, int);

void foo (float \* p0, float \* v1, float \* v2, int N)

{

init (v1, v2, N);

mapa #pragma omp de datos de destino (a: v1 [0: N], v2 [: N]) mapa (a partir de: p0 [0: N])

{

vec\_mult (p0, V1, V2, N);

}

de salida (p0, N);

}

vec\_mult void (float \* y p1, float \* y v3, flotar \* y v4, int & N)

{

int i;

mapa de destino omp #pragma (a: v3 [0: N], v4 [: N]) mapa (a partir de: p1 [0: N]) #pragma omp paralelo para

for (i = 0; i <N; i ++) p1 [i] = v3 [i] \* v4 [i];

}

##### C / C ++



Fortran



*Ejemplo 49.5f*

En el siguiente ejemplo, el enfoque de Fortran habitual se utiliza para la memoria dinámica. El p0, v1, v2 matrices se asignan en el programa principal y pasa como referencias de una rutina a otra. En vec\_mult, p1, v3 y v4 son referencias a los P0, v1, v2 y matrices, respectivamente. entorno de datos de dispositivo del constructo objetivo hereda las matrices de p0, v1, v2 y de entorno de datos del dispositivo del constructo de datos de destino de encerramiento.

Por lo tanto, p1, V3 y V4 están ya presentes en el entorno de datos del dispositivo.

my\_mult módulo contiene

subrutina foo (p0, v1, v2, N)

real dimensión (:) :: p0, v1, v2 entero:: N, i

llamar a init (v1, v2, N)

! $ OMP mapa de datos de destino (a: v1, v2) mapa (a partir de: P0) llamada vec\_mult (P0, v1, v2, N)

FIN omp meta de producción llamada de datos (P0, N)

subrutina final

vec\_mult subrutina (p1, V3, V4, N) real, dimensión (:) :: p1, V3, V4 entero:: N, i

Mapa del objetivo $ OMP (a: V3, V4) mapa (de p1)

! $ OMP paralelo hagas i = 1, N

P1 (i) = v3 (i) \* v4 (i) finales hacer

! $ OMP final subrutina final de destino

módulo final

!

programa principal uso my\_mult

número entero, el parámetro :: N = 1024 real, asignable, dimensión (:) :: p, v1, v2

asignar (p (N), v1 (N), v2 (N)) llamada foo (p, v1, v2, N)

fin del programa

##### Fortran



**Constructo de datos de destino Con si la cláusula**

Los dos ejemplos siguientes muestran cómo los datos de destino construir variables mapas a un entorno de datos del dispositivo.

En el siguiente ejemplo, el si cláusula en el constructo de datos de destino indica que si la variable N es menor que un umbral dado, a continuación, el constructo de datos de destino no crear un entorno de datos del dispositivo.

Las construcciones diana cerrados en la región de datos de destino también deben utilizar un si cláusula en la misma condición, de lo contrario la variable puntero P se asigna implícitamente con un mapa de tipo de tofrom, pero la ubicación de almacenamiento de la sección de matriz de p [0: N] no será asignada en los entornos de los datos del dispositivo de las construcciones diana.

C / C ++



*Ejemplo 49.5c*

UMBRAL #define 1000000

extern void init (float \*, float \*, int); extern init\_again vacío (float \*, float \*, int); extern salida nula (float \*, int);

vec\_mult void (float \* p, float \* v1, float \* v2, int N)

{

int i;

init (v1, v2, N);

datos de destino #pragma omp si (N> umbral) mapa (de: p [0: N])

{

objetivo #pragma omp si (N> umbral) mapa (a: v1 [: N], v2 [: N]) #pragma omp paralelo para

for (i = 0; i <N; i ++) p [i] = v1 [i] \* v2 [i];

init\_again (v1, v2, N);

objetivo #pragma omp si (N> umbral) mapa (a: v1 [: N], v2 [: N]) #pragma omp paralelo para

for (i = 0; i <N; i ++)

p [i] = p [i] + (v1 [i] \* v2 [i]);

}

de salida (p, N);

}

##### C / C ++



Fortran



*Ejemplo 49.6f*

Las cláusulas si funcionan de la misma manera para el siguiente código Fortran. Las construcciones diana cerrados en la región de datos de destino también debe utilizar un si cláusula con la misma condición, de modo que la región de datos de destino y la región diana son o bien ambos creados para el dispositivo, o son ambos ignorado.

params módulo

número entero, el parámetro :: UMBRAL = 1000000 módulo extremo

subrutina vec\_mult (p, v1, v2, N) utiliza params

real :: p (N), v1 (N), v2 (N) número entero :: i

llamar a init (v1, v2, N)

! Datos de destino $ OMP si (N> UMBRAL) mapa (a partir de: p)

! Objetivo de $ OMP si (N> UMBRAL) mapa (a: v1, v2)

! $ OMP paralelo hagas i = 1, N

p (i) = v1 (i) \* v2 (i) final hacer

! $ Objetivo final omp

llamar init\_again (v1, v2, N)

! Objetivo de $ OMP si (N> UMBRAL) mapa (a: v1, v2)

! $ OMP paralelo hacer

hacer i = 1, N

p (i) = P (i) + v1 (i) \* v2 (i) final hacer

! $ Objetivo final omp

! $ Datos de destino final omp llaman salida (p, n)

subrutina final

##### Fortran



En el siguiente ejemplo, cuando la cláusula si la expresión condicional en el constructo diana se evalúa como falsa, la región diana se ejecutará en el dispositivo host. Sin embargo, el constructo de datos de destino creó un entorno de datos del dispositivo de cerramiento que mapea p [0: N] a un entorno de datos del dispositivo en el dispositivo por defecto. Al final de la región de datos de destino la sección de matriz p [0: N] será asignado desde el entorno de datos del dispositivo a la variable correspondiente en el entorno de datos de la tarea que se encontró con el constructo de datos de destino, lo que resulta en valores no definidos en p [ 0: N].

*Ejemplo 49.6c*



C / C ++

UMBRAL #define 1000000

extern void init (float \*, float \*, int); extern salida nula (float \*, int);

vec\_mult void (float \* p, float \* v1, float \* v2, int N)

{

int i;

init (v1, v2, N);

mapa #pragma omp de datos de destino (a partir de: p [0: N])

{

objetivo #pragma omp si (N> umbral) mapa (a: v1 [: N], v2 [: N]) #pragma omp paralelo para

for (i = 0; i <N; i ++) p [i] = v1 [i] \* v2 [i];

} / \* Comportamiento indefinido si N <= UMBRAL \* / salida (p, N);

}

##### C / C ++



Fortran



*Ejemplo 49.7f*

Las cláusulas si funcionan de la misma manera para el siguiente código Fortran. Cuando la cláusula de si la expresión condicional en el constructo diana se evalúa como falsa, la región diana se ejecutará en el dispositivo host. Sin embargo, el constructo de datos de destino creó un entorno de datos del dispositivo de cerramiento que asigna la matriz de p (y v1 y v2) a un entorno de datos del dispositivo del dispositivo de destino predeterminada. Al final de la región de datos de destino

la matriz p será asignado desde el entorno de datos del dispositivo a la variable correspondiente en el entorno de datos de la tarea que se encontró con el constructo de datos de destino, lo que resulta en valores no definidos en p.

params módulo

número entero, el parámetro :: UMBRAL = 1000000 módulo extremo

subrutina vec\_mult (p, v1, v2, N) utiliza params

real :: p (N), v1 (N), v2 (N) número entero :: i

llamar a init (v1, v2, N)

Mapa de datos de destino $ OMP (a partir de: p)

! Objetivo de $ OMP si (N> UMBRAL) mapa (a: v1, v2)

! $ OMP paralelo hagas i = 1, N

p (i) = v1 (i) \* v2 (i) final hacer

! $ Objetivo final omp

! $ datos de destino final omp

llamar de salida (p, N)! \*\*\* comportamiento indefinido si N <= umbral de subrutina final

##### Fortran



## objetivo actualización Construct

**datos de destino sencilla y la actualización de destinos**

**construcciones**

El siguiente ejemplo muestra cómo la actualización objetivo construir variables actualizaciones en un entorno de datos del dispositivo.

Los datos de destino construyen secciones correlaciones de matrices v1 [: n] y v2 [: n] (arrays v1

y v2 en el código Fortran) en un entorno de datos del dispositivo.

La tarea se ejecuta en el dispositivo principal se encuentra con la primera región diana y espera a la finalización de la región.

Después de la ejecución de la primera región diana, la tarea se ejecuta en el dispositivo host a continuación, asigna nuevos valores a V1 [: N] y v2: (V1 y V2 matrices en código Fortran) en entorno de datos de la tarea llamando a la función [N] init\_again ().

La actualización objetivo constructo asigna los nuevos valores de v1 y v2 de entorno de datos de la tarea a las secciones de matriz mapeadas correspondientes en el entorno de los datos del dispositivo de la construcción de datos de destino.

La tarea se ejecuta en el dispositivo principal se encuentra entonces con la segunda región diana y espera a la finalización de la región.

La segunda región de destino utiliza los valores actualizados de v1 [: n] y v2 [: n].

*Ejemplo 50.1c*



C / C ++

extern void init (float \*, float \*, int); extern init\_again vacío (float \*, float \*, int); extern salida nula (float \*, int);

vec\_mult void (float \* p, float \* v1, float \* v2, int N)

{

int i;

init (v1, v2, N);

mapa #pragma omp de datos de destino (a: v1 [: N], v2 [: N]) mapa (de: p [0: N])

{

#pragma omp #pragma objetivo omp paralelo para for (i = 0; i <N; i ++)

p [i] = v1 [i] \* v2 [i]; init\_again (v1, v2, N);

#pragma actualización objetivo omp a (v1 [: N], v2 [: N]) objetivo #pragma omp

#pragma omp paralelo para for (i = 0; i <N; i ++)

p [i] = p [i] + (v1 [i] \* v2 [i]);

}

de salida (p, N);

}

##### C / C ++



Fortran



*Ejemplo 50.1f*

vec\_mult subrutina (p, v1, v2, N) real :: p (N), v1 (N), v2 (N) número entero :: i

llamar a init (v1, v2, N)

Mapa de datos de destino $ OMP (a: v1, v2) mapa (a partir de: p)

! $ OMP objetivo

! $ OMP paralelo hagas i = 1, N

p (i) = v1 (i) \* v2 (i) final hacer

! $ Objetivo final omp

llamar init\_again (v1, v2, N)

! Actualización de destinos a $ OMP (v1, v2)

! $ OMP objetivo

! $ OMP paralelo hagas i = 1, N

p (i) = P (i) + v1 (i) \* v2 (i) final hacer

! $ Objetivo final omp

! $ Datos de destino final omp llaman salida (p, n)

subrutina final

##### Fortran



**objetivo actualización Construct Con si la cláusula**

El siguiente ejemplo muestra cómo la actualización objetivo construir variables actualizaciones en un entorno de datos del dispositivo.

Los datos de destino construyen secciones correlaciones de matrices v1 [: n] y v2 [: n] (arrays v1 y v2 en el código Fortran) en un entorno de datos del dispositivo. Entre las dos regiones de destino, la tarea se ejecuta en el dispositivo host asigna condicionalmente nuevos valores de V1 y V2 en el entorno de datos de la tarea. El maybe\_init\_again función () devuelve verdadero si se escriben nuevos datos.

Cuando la expresión condicional (el valor de retorno de maybe\_init\_again ()) en el caso de la cláusula es cierto, el constructo de actualización objetivo asigna los nuevos valores de v1 y v2 de entorno de datos de la tarea a las secciones de matriz mapeadas correspondientes en los datos del dispositivo de la construcción de datos de destino ambiente.

*Ejemplo 50.2c*



C / C ++

extern void init (float \*, float \*, int); extern maybe\_init\_again int (float \*, int); extern salida nula (float \*, int);

vec\_mult void (float \* p, float \* v1, float \* v2, int N)

{

int i;

init (v1, v2, N);

mapa #pragma omp de datos de destino (a: v1 [: N], v2 [: N]) mapa (de: p [0: N])

{

int cambió; objetivo omp #pragma

#pragma omp paralelo para for (i = 0; i <N; i ++)

p [i] = v1 [i] \* v2 [i];

cambiado = maybe\_init\_again (v1, N);

#pragma actualización objetivo omp si (cambiado) a (v1 [: N]) cambiado = maybe\_init\_again (v2, N);

#pragma actualización objetivo omp si (cambiado) a (v2 [: N]) objetivo omp #pragma

#pragma omp paralelo para for (i = 0; i <N; i ++)

p [i] = p [i] + (v1 [i] \* v2 [i]);

}

de salida (p, N);

}

##### C / C ++



Fortran



*Ejemplo 50.2f*

vec\_mult subrutina (p, v1, v2, N) de interfaz

función maybe\_init\_again lógico (v1, N) real :: v1 (N)

número entero :: función extremo N

interfaz de extremo

real :: p (N), v1 (N), v2 (N) número entero :: i

lógico :: cambiado init llamada (v1, v2, N)

Mapa de datos de destino $ OMP (a: v1, v2) mapa (a partir de: p)

! $ OMP objetivo

! $ OMP paralelo hagas i = 1, N

p (i) = v1 (i) \* v2 (i) final hacer

! $ Objetivo final omp

cambiado = maybe\_init\_again (v1, N)

! Objetivo de $ OMP si (cambiado) la actualización a (v1 (N)) cambió = maybe\_init\_again (v2, N)

! Objetivo de $ OMP si (cambiado) actualización de (v2 (N))

! $ OMP objetivo

! $ OMP paralelo hagas i = 1, N

p (i) = P (i) + v1 (i) \* v2 (i) final hacer

! $ Objetivo final omp

! $ Datos de destino final omp llaman salida (p, n)

subrutina final

##### Fortran



## declarar Construct objetivo

**declarar objetivo y al final declaran objetivo para una Función**

El siguiente ejemplo muestra cómo se utiliza la directiva objetivo de declarar para indicar que la llamada correspondiente dentro de una región objetivo es una función fib que se pueden ejecutar en el dispositivo de destino por defecto.

Una versión de la función también está disponible en el dispositivo host. Cuando la cláusula de si la expresión condicional en el constructo diana se evalúa como falsa, la región diana (por lo tanto FIB) se ejecutará en el dispositivo host.

Para C / C ++ códigos de la declaración de la fib función aparece entre el objetivo declarar y final declaran directivas de destino.

##### C / C ++



*Ejemplo 51.1c*

objetivo declare #pragma omp extern fib void (int N); final omp #pragma declarar objetivo #define UMBRAL 1000000

void fib\_wrapper (int n)

{

objetivo #pragma omp si (n> umbral)

{

fib (n);

}

}

##### C / C ++



Fortran



*Ejemplo 51.1f*

La subrutina fib Fortran contiene una declaración objetivo de declarar para indicar al compilador para crear una versión ejecutable del dispositivo del procedimiento. El nombre de la subrutina no se ha incluido en la directiva de destino declarar y está, por lo tanto, implícitamente asumido.

El programa utiliza el módulo module\_fib, que presenta una interfaz explícita al compilador con las declaraciones objetivo de declarar para el procesamiento de la llamada fib.

module\_fib módulo contiene

fib subrutina (N) número entero :: N

! $ Objetivo de declarar omp

! ...

extremo del módulo final subrutina

params módulo

número entero :: UMBRAL = 1000000

módulo final params uso my\_fib programa

uso module\_fib

! Objetivo de $ OMP si (N> UMBRAL) llamada fib (N)

! Programa final de destino final omp $

El siguiente ejemplo Fortran muestra el uso de una subrutina externa. Sin una interfaz explícita (a través del uso del módulo o un bloque de interfaz) las declaraciones de destino declare dentro de una subrutina externa son desconocidas a la unidad principal del programa; Por lo tanto, un objetivo de declarar debe ser proporcionada dentro del alcance del programa para el compilador para determinar que un binario de destino debe estar disponible.

*Ejemplo 51.2f*

programa my\_fib número entero :: N = 8

! $ OMP objetivo declare (FIB)

! $ OMP llamada de destino fib (N)

! $ OMP extremo final de destino subrutina programa de fib (N) número entero N ::

! $ OMP declarar impresión de destino \*, "hola de fib"

! ...

subrutina final

##### Fortran



**declarar Construct objetivo Tipo de Clase**

El siguiente ejemplo muestra cómo el objetivo declarar y final declaran directivas de destino se utilizan para encerrar la declaración de una variable variar con un typeY tipo de clase. La función miembro typeY :: foo () no se puede acceder en un dispositivo de destino porque su declaración no apareció entre el objetivo y fin de declarar declarar directivas de destino.

##### C / C ++



*Ejemplo 51.2c*

struct Typex

{

int a;

}

clase typeY

{

int foo () {return a ^ 0x01;} int a;

}

#pragma objetivo omp declare struct Typex varX; // Okay

clase typeY variar; // bien si varY.foo () no se llama el dispositivo de destino final #pragma omp declaran objetivo

void foo ()

{

objetivo omp #pragma

{

varX.a = 100; // Okay

varY.foo (); // foo error () no está disponible en un dispositivo de destino

}

}

##### C / C ++



**declarar objetivo y al final declarar objetivo para Variables**

Los siguientes ejemplos muestran cómo el objetivo declarar y final declaran directivas de destino se utilizan para indicar que las variables globales se asignan al entorno de datos dispositivo implícita de cada dispositivo de destino.

En el siguiente ejemplo, las declaraciones de las variables p, v1, v2 y aparecer entre objetivo declarar y final declaran directivas de destino que indican que las variables se asignan al entorno de datos dispositivo implícita de cada dispositivo de destino. los

**actualización de destinos** directiva se utiliza a continuación para gestionar la consistencia de las variables p, v1, v2 y entre el entorno de datos de la tarea dispositivo host Encountering y el entorno de datos dispositivo implícita del dispositivo de destino predeterminada.

*Ejemplo 51.3c*



C / C ++

#define N 1000

#pragma omp flotador objetivo declare p [N], v1 [N], v2 [N]; objetivo declare extremo omp #pragma

extern void init (float \*, float \*, int); extern salida nula (float \*, int);

vec\_mult vacío ()

{

int i;

init (v1, v2, N);

#pragma omp actualización de destinos a (v1, v2) #pragma omp objetivo

#pragma omp paralelo para for (i = 0; i <N; i ++)

p [i] = v1 [i] \* v2 [i];

#pragma actualización objetivo omp a partir de (p) de salida (p, N);

}

##### C / C ++



Fortran



*Ejemplo 51.3f*

La versión Fortran del código C anterior utiliza una sintaxis diferente. módulos Fortran utilizan una sintaxis de lista de la Directiva objetivo de declarar para declarar variables asignadas.

my\_arrays módulo

! $ Target omp declare (N, p, v1, v2) número entero, el parámetro :: N = 1000

real :: p (N), v1 (N), v2 (N) módulo final

vec\_mult subrutina () utiliza my\_arrays

número entero :: i

llamar a init (v1, v2, N);

! Actualización de destinos a $ OMP (v1, v2)

! $ OMP objetivo

! $ OMP paralelo hagas i = 1, N

p (i) = v1 (i) \* v2 (i)

final hacer

! $ Objetivo final omp

! Actualización de destinos de $ OMP (p) de salida de llamadas (p, n)

subrutina final

##### Fortran



El siguiente ejemplo también indica que la función Pfun () está disponible en el dispositivo de destino, así como la variable Q, que está asignada al entorno de datos dispositivo implícita de cada dispositivo de destino. La directiva de actualización de destinos se utiliza para gestionar la consistencia de la Q variable entre el entorno de datos de la tarea dispositivo host Encountering y el entorno del dispositivo implícita del dispositivo de destino por defecto.

C / C ++



En el siguiente ejemplo, las declaraciones de funciones y variables aparecen entre el

**declarar objetivo** y al final declarar directivas de destino.

#### *Ejemplo 51.4c*

#define N 10000

#pragma omp objetivo declare flotador Q [N] [N];

flotar Pfun (const int i, const k int)

{Return Q [i] [k] \* Q [k] [i]; } End #pragma omp declarar objetivo acum flotador (int k)

{

flotar tmp = 0,0;

#pragma omp actualización de destinos a (Q) #pragma omp objetivo

#pragma omp paralelo para la reducción de (+: tmp) for (int i = 0; i <N; i ++)

tmp + = Pfun (i, k); volver tmp;

}

##### C / C ++



Fortran



*Ejemplo 51.4f*

La versión Fortran del código C anterior utiliza una sintaxis diferente. En los módulos Fortran una sintaxis lista de la directiva de destino declare es usada para declarar variables y procedimientos asignados. Las variables N y Q se declaran como una lista separada por comas. Cuando se utiliza la directiva objetivo de declarar a declarar simplemente el procedimiento, el nombre del procedimiento no tiene que estar en la lista - se asume implícitamente, como se ilustra en la función Pfun ().

my\_global\_array módulo

! $ Objetivo omp declare (N, Q) número entero, el parámetro :: N = 10 reales :: Q (N, N)

contiene

función Pfun (i, k)

! $ Objetivo de declarar omp

real:: Pfun

número entero, la intención (in) :: i, k Pfun = (Q (i, k) \* Q (k, i))

módulo de extremo de función final

función acum (k) resultado (tmp) utiliza my\_global\_array

real :: :: tmp número entero i, k

tmp = 0.0e0

! $ OMP objetivo

! $ OMP paralelo hacer reducción (+: tmp) hacer i = 1, N

tmp = tmp + Pfun (k, i) final hacer

! $ OMP función de extremo a extremo de destino

##### Fortran



**declarar objetivo y al final declarar objetivo**

**con SIMD declarar**

El siguiente ejemplo muestra cómo el objetivo declarar y final declaran directivas de destino se utilizan para indicar que una función está disponible en un dispositivo de destino. La directiva declare SIMD indica que hay una versión de SIMD de la función P () que está disponible en el dispositivo de destino, así como uno que está disponible en el dispositivo host.

*Ejemplo 51.5c*



C / C ++

#define N 10000

#define M 1024

#pragma omp objetivo declare flotador Q [N] [N];

#pragma omp declarar uniforme SIMD (i) lineal (k) notinbranch flotar P (const int i, const int k)

{

volver Q [i] [k] \* Q [k] [i];

}

final omp #pragma declarar objetivo acum flotador (void)

{

flotar tmp = 0,0; int i, k;

objetivo omp #pragma

#pragma omp paralelo para la reducción de (+: tmp) para (i = 0; i <N; i ++) {

flotador tmp1 = 0,0;

#pragma omp reducción SIMD (+: tmp1) para (k = 0; k <M; k ++) {

tmp1 + = P (i, k);

}

tmp + = tmp1;

}

volver tmp;

}

##### C / C ++



Fortran



*Ejemplo 51.5f*

La versión Fortran del código C anterior utiliza una sintaxis diferente. módulos Fortran utilizan una sintaxis de lista de la declaración de destino de declarar para el mapeo. Aquí las variables N y Q se declaran en la forma lista como una lista separada por comas. La declaración de la función no utiliza una lista e implícitamente asume el nombre de la función. En este ejemplo Fortran índices de fila y de columna están invertidos en relación con el ejemplo de la C / C ++, como es habitual para los códigos optimizados para el acceso de memoria.

my\_global\_array módulo

! $ Objetivo omp declare (N, Q)

número entero, el parámetro :: N = 10000, M = 1024 real:: Q (N, N)

contiene la función P (k, i)

! $ OMP declarar uniforme SIMD (i) lineal (k) notinbranch

! $ OMP objetivo de declarar real :: PAG

número entero, la intención (in) :: k, i P = (Q (k, i) \* Q (i, k))

módulo de extremo de función final

función acum () resultado (tmp) utiliza my\_global\_array

real :: tmp, tmp1 número entero :: i

tmp = 0.0e0

! $ OMP objetivo

! $ OMP paralelo hacer reducción privado (tmp1) (+: tmp)

hacer i = 1, N

tmp1 = 0.0e0

! $ Reducción omp SIMD (+: tmp1) de hacer k = 1, M

tmp1 = tmp1 + P (k, i) final hacer

tmp = tmp + tmp1 final hacer

! $ OMP función de extremo a extremo de destino

##### Fortran



## equipos construcciones

**objetivo y los equipos de construcciones con** omp\_get\_num\_teams y rutinas omp\_get\_team\_num

El siguiente ejemplo muestra cómo se usan el objetivo y los equipos de construcciones para crear una liga de equipos de rosca que ejecutan una región. Los equipos de construcción crea una liga de un máximo de dos equipos en los que el hilo principal de cada equipo ejecuta la región de los equipos.

La rutina devuelve omp\_get\_num\_teams el número de equipos que se ejecutan en los equipos de una región. La rutina omp\_get\_team\_num devuelve el número del equipo, que es un número entero entre 0 y uno menos que el valor devuelto por omp\_get\_num\_teams. El siguiente ejemplo distribuye manualmente un bucle a través de dos equipos.

*Ejemplo 52.1c*



C / C ++

#include <stdlib.h> #include <omp.h>

flotar dotprod (float B [], flotador C [], int N)

{

flotar sum0 = 0,0; flotar sum1 = 0,0;

mapa de destino omp #pragma (a: B [: N], C [: n]) Equipos #pragma omp num\_teams (2)

{

int i;

if (! omp\_get\_num\_teams () = 2) abortan ();

si (omp\_get\_team\_num () == 0)

{

#pragma omp paralelo para la reducción de (+: sum0) para (i = 0; i <N / 2; i ++)

sum0 + = B [i] \* C [i];

}

else if (omp\_get\_team\_num () == 1)

{

#pragma omp paralelo para la reducción de (+: sum1) para (i = N / 2; i <N; i ++)

sum1 + = B [i] \* C [i];

}

}

sum0 + sum1 retorno;

}

##### C / C ++



Fortran



*Ejemplo 52.1f*

resultado de la función dotprod (B, C, N) (suma)

omp\_lib utilizar, únicamente: omp\_get\_num\_teams, omp\_get\_team\_num real :: B (N), C (N), suma, sum0, sum1

número entero :: N, i = sum0 0.0e0 sum1 = 0.0e0

Mapa del objetivo $ OMP (a: B, C)

! $ OMP equipos num\_teams (2)

si (omp\_get\_num\_teams () / = 2) dejar de "2 equipos requeridos" si (omp\_get\_team\_num () == 0), entonces

! $ Omp paralelo hacer reducción (+: sum0) hacer i = 1, N / 2

sum0 = sum0 + B (i) \* C (i) final hacer

else if (omp\_get\_team\_num () == 1), entonces

! $ Omp paralelo hacer reducción (+: sum1) do i = N / 2 + 1, N

sum1 = sum1 + B (i) \* C (i) final hacer

terminara si

! $ equipos de gama OMP

! $ OMP objetivo final = suma + sum0 sum1

end function

##### Fortran



**objetivo, equipos y distribuir constructos**

El siguiente ejemplo muestra cómo el objetivo, los equipos, y distribuir construcciones se utiliza para ejecutar un nido de bucle en una región diana. Los equipos de construcción crea una liga y el hilo principal de cada equipo ejecuta la región de los equipos. La distribuir horarios construir el iteraciones del bucle posteriores a través de los hilos maestras de cada equipo.

El número de equipos de la liga es inferior o igual a los num\_blocks variables. Cada equipo de la liga tiene un número de hilos de menos de o igual a la block\_threads variables. Las iteraciones en el bucle exterior se distribuyen entre las roscas maestras de cada equipo.

Cuando hilo principal de un equipo se encuentra con la construcción de lazo paralelo antes de que el bucle interior, los otros hilos en su equipo se activan. El equipo ejecuta la región paralela y luego workshares la ejecución del bucle.

Cada hilo principal la ejecución de la región de los equipos tiene una copia privada de la suma variable generada por la cláusula de reducción en la construcción de equipos. El hilo principal y todos los hilos en su equipo tienen una copia privada de la suma variable que se crea por la cláusula de reducción en la construcción de lazo paralelo. La segunda suma privada se reduce a la copia privada del hilo maestro de suma creado por la construcción de equipos. Al final de la región de los equipos, la copia privada de cada hilo maestro de suma se reduce a la suma final que se asigna de forma implícita en la región de destino.

*Ejemplo 52.2c*



C / C ++

flotar dotprod (float B [], flotador C [], int N, int block\_size, int num\_teams, int block\_threads)

{

flotador suma = 0; int i, i0;

mapa de destino omp #pragma (a: B [0: N], C [0: N])

equipos #pragma omp num\_teams (num\_teams) thread\_limit (block\_threads) \ reducción (+: Suma)

#pragma omp distribuir

para (i0 = 0; i0 <N; i0 + = block\_size)

#pragma omp paralelo para la reducción de (+: suma) para (i = i0; i <min (i0 + block\_size, N); i ++)

suma + = B [i] \* C [i]; suma de retorno;

}

##### C / C ++



Fortran



*Ejemplo 52.2f*

función dotprod (B, C, N, block\_size, num\_teams, block\_threads) resultado (suma) implícita ninguno

real :: B (N), C (N), suma

número entero :: N, block\_size, num\_teams, block\_threads, i, suma i0 = 0.0e0

Mapa del objetivo $ OMP (a: B, C)

! $ OMP equipos num\_teams (num\_teams) thread\_limit (block\_threads) y reducción (+: Suma)

! $ OMP distribuir

hacer i0 = 1, N, block\_size

! $ OMP paralelo hacerlo reducción (+: Suma) hacer i = i0, min (i0 + block\_size, N)

suma = suma + B (i) \* C (i) final hacer

final hacer

! $ equipos de gama OMP

! $ OMP función de extremo a extremo de destino

##### Fortran



**dirigirse a los equipos, y distribuir de construcciones de bucle paralelo**

El siguiente ejemplo muestra cómo los equipos de destino y distribuir construcciones bucle paralelas se utilizan para ejecutar una región diana. El equipo de destino constructo crea una liga de equipos en los que el hilo principal de cada equipo ejecuta la región de los equipos.

La distribuir horarios construcción de lazo paralelas las iteraciones de bucle a través de los hilos de capitán de cada equipo y luego a través de los hilos de cada equipo.

*Ejemplo 52.3c*



C / C ++

flotar dotprod (float B [], flotador C [], int N)

{

flotador suma = 0; int i;

#pragma omp equipos de destino mapa (a: B [0: N], C [0: N]) #pragma omp distribuir paralelo para la reducción de (+: suma) para (i = 0; i <N; i ++)

suma + = B [i] \* C [i]; suma de retorno;

}

##### C / C ++



Fortran



*Ejemplo 52.3f*

función dotprod (B, C, N) resultado (suma) real :: B (N), C (N), suma número entero :: N, yo

= suma 0.0e0

! Equipos de destino $ OMP mapa (a: B, C)

! $ OMP distribuir paralelamente hacer reducción (+: Suma) hacer i = 1, N

suma = suma + B (i) \* C (i) final hacer

! $ equipos de gama OMP

! $ OMP función de extremo a extremo de destino

##### Fortran



**equipos de destino Distribuir y de construcciones de bucle en paralelo con las cláusulas de programación**

El siguiente ejemplo muestra cómo los equipos de destino y distribuir construcciones bucle paralelas se utilizan para ejecutar una región diana. Los equipos de construcción crea una liga de un máximo de ocho equipos en los que el hilo principal de cada equipo ejecuta la región de los equipos. El número de hilos en cada equipo es menor que o igual a 16.

La distribuir los horarios de construcción de bucle paralelo las iteraciones del bucle posteriores a través de los hilos de capitán de cada equipo y luego a través de los hilos de cada equipo.

La cláusula dist\_schedule en la construcción de lazo distribuir paralelo indica que iteraciones del bucle se distribuyen a la rosca maestro de cada equipo en trozos de 1024 iteraciones.

La cláusula de horario indica que los 1024 iteraciones distribuidos a un hilo principal se asignan a los hilos en su equipo asociado en trozos de 64 iteraciones.

*Ejemplo 52.4c*



C / C ++

#define N 1024 \* 1024

flotar dotprod (float B [], flotador C [], int N)

{

flotador suma = 0; int i;

mapa de destino omp #pragma (a: B [0: N], C [0: N]) equipos #pragma omp num\_teams (8) thread\_limit (16)

#pragma omp distribuir paralelo para la reducción de (+: suma) \ dist\_schedule (, 1024 estática) horario (estático, 64)

for (i = 0; i <N; i ++)

suma + = B [i] \* C [i]; suma de retorno;

}

##### C / C ++



Fortran



*Ejemplo 52.4f*

matrices de módulos

número entero, el parámetro :: N = 1024 \* 1024 real :: B (N), C (N)

módulo final

dotprod función () resultado (SUM) matrices de uso

real:: suma número entero :: i resumir = 0.0e0

Mapa del objetivo $ OMP (a: B, C)

! $ OMP equipos num\_teams (8) thread\_limit (16)

! $ OMP distribuir paralelamente hacer reducción (+: Suma) y

! $ OMP y dist\_schedule (, 1024 estática) horario (estática, 64) DO I = 1, N

suma = suma + B (i) \* C (i) final hacer

! $ equipos de gama OMP

! $ OMP función de extremo a extremo de destino

##### Fortran



**equipos de destino y distribuir SIMD**

**construcciones**

El siguiente ejemplo muestra cómo los equipos de destino y distribuir construcciones SIMD se utiliza para ejecutar un bucle en una región diana. El equipo de destino constructo crea una liga de equipos en los que el hilo principal de cada equipo ejecuta la región de los equipos.

La distribuir horarios constructo SIMD las iteraciones de bucle a través de la rosca principal de cada equipo y luego utiliza SIMD paralelismo para ejecutar las iteraciones.

*Ejemplo 52.5c*



C / C ++

extern void init (float \*, float \*, int); extern salida nula (float \*, int);

vec\_mult void (float \* p, float \* v1, float \* v2, int N)

{

int i;

init (v1, v2, N);

#pragma omp equipos de destino mapa (a: v1 [0: N], v2 [: N]) mapa (de: p [0: N]) #pragma omp distribuir SIMD

for (i = 0; i <N; i ++) p [i] = v1 [i] \* v2 [i];

de salida (p, N);

}

##### C / C ++



Fortran



*Ejemplo 52.5f*

vec\_mult subrutina (p, v1, v2, N) real :: p (N), v1 (N), v2 (N) número entero :: i

llamar a init (v1, v2, N)

! Equipos de destino $ OMP mapa (a: v1, v2) mapa (a partir de: p)

! $ OMP distribuyen SIMD hacen i = 1, N

p (i) = v1 (i) \* v2 (i) final hacer

! Equipos de destino final omp $ llaman salida (p, n)

subrutina final

##### Fortran



**equipos de destino Distribuir y paralelas Loop SIMD constructos**

El siguiente ejemplo muestra cómo se utilizan los equipos de destino y los distribuir construcciones SIMD bucle paralelas para ejecutar un bucle en una región equipos de destino. El equipo de destino constructo crea una liga de equipos en los que el hilo principal de cada equipo ejecuta la región de los equipos.

La distribuir bucle paralelo horarios SIMD construir las iteraciones del bucle a través de la rosca principal de cada equipo y luego a través de los hilos de cada equipo donde cada hilo utiliza SIMD paralelismo.

*Ejemplo 52.6c*



C / C ++

extern void init (float \*, float \*, int); extern salida nula (float \*, int);

vec\_mult void (float \* p, float \* v1, float \* v2, int N)

{

int i;

init (v1, v2, N);

#pragma omp equipos de destino mapa (a: v1 [0: N], v2 [: N]) mapa (de: p [0: N]) #pragma omp distribuir paralelo para SIMD

for (i = 0; i <N; i ++) p [i] = v1 [i] \* v2 [i];

de salida (p, N);

}

##### C / C ++



Fortran



*Ejemplo 52.6f*

vec\_mult subrutina (p, v1, v2, N) real :: p (N), v1 (N), v2 (N) número entero :: i

llamar a init (v1, v2, N)

! Equipos de destino $ OMP mapa (a: v1, v2) mapa (a partir de: p)

! $ OMP distribuir paralelamente hacer SIMD hacer i = 1, N

p (i) = v1 (i) \* v2 (i) final hacer

! Equipos de destino final omp $ llaman salida (p, n)

subrutina final

##### Fortran



1. **La ejecución asíncrona de un objetivo**

**Región Uso de las tareas**

El siguiente ejemplo muestra cómo se utilizan los constructos de tareas y de destino para ejecutar múltiples regiones diana de forma asincrónica. La tarea que se encuentra con el constructo tarea genera una tarea explícita de que contiene una región diana. El hilo de la ejecución de la tarea explícita se encuentra con un punto de programación de tareas a la espera de la ejecución de la región de destino para completar, lo que permite el hilo para volver a la ejecución de la tarea Encountering o una de las tareas explícitas generados anteriormente.

C / C ++



*Ejemplo 53.1c*

#pragma omp objetivo declare flotador F (float);

final omp #pragma declarar objetivo #define N 1000000000

CHUNKSZ #define 1000000

anulará init (float \*, int); flotar Z [N];

pipedF anular ()

{

int C, i; init (Z, N);

para (C = 0; C <N; C + = CHUNKSZ)

{

#pragma omp tarea

mapa de destino omp #pragma (Z [C: CHUNKSZ]) #pragma omp paralelo para

for (i = 0; i <CHUNKSZ; i ++) Z [i] = F (Z [i]);

}

#pragma omp taskwait

}

##### C / C ++



Fortran



*Ejemplo 53.1f*

La versión Fortran tiene un bloque de interfaz que contiene el destino declare. Una afirmación idéntica existe en la declaración de la función (que no se muestra aquí).

parámetros del módulo

número entero, el parámetro :: N = 1000000000, módulo CHUNKSZ = 1000000 final

!!

pipedF subrutina ()

parámetros de uso, únicamente: N, número entero CHUNKSZ :: C, yo

real:: z (N)

!!

interfaz

la función F (z)

! $ OMP objetivo de declarar real intención (EN) :: z real ::F

función final interfaz F extremo

!!

llamar a init (z, N)

!!

hacer C = 1, N, CHUNKSZ

!!

! $ Tarea omp

! Mapa de destino $ OMP (z (C: C + CHUNKSZ-1))

! $ OMP paralelo hacer

hacer i = C, C + CHUNKSZ-1

z (i) = F (z (i)) final hacer

! $ Objetivo final omp

! $ Tarea final omp

!!

final hacer de impresión \*, z

!!

pipedF subrutina final

##### Fortran



El siguiente ejemplo muestra cómo se utilizan los constructos de tareas y de destino para ejecutar múltiples regiones diana de forma asincrónica. La dependencia de tareas asegura que el almacenamiento se asigna y se inicializa en el dispositivo antes de que se accede.

*Ejemplo 53.2c*



C / C ++

#include <stdlib.h>

extern void init (float \*, float \*, int); extern salida nula (float \*, int);

vec\_mult void (float \* p, float \* v1, float \* v2, int N, int dev)

{

int i; init (p, N);

#pragma omp tarea depende (Salida: v1, v2) del dispositivo de destino #pragma omp (dev) mapa (v1, v2)

{

// comprobar si el dispositivo si dev (omp\_is\_initial\_device ())

abortar();

v1 = malloc (N \* sizeof (float)); v2 = malloc (N \* sizeof (float)); init (v1, v2);

}

foo (); // ejecutar otros trabajos asychronously #pragma omp tarea dependerá (en: v1, v2)

dispositivo de destino omp #pragma (dev) mapa (a: v1, v2) mapa (de: p [0: N])

{

// comprobar si el dispositivo si dev (omp\_is\_initial\_device ())

abortar(); #pragma omp paralelo para for (i = 0; i <N; i ++)

p [i] = v1 [i] \* v2 [i];

de salida (p, N);

libre (v1);

libre (v2);

}

}

##### C / C ++



Fortran



*Ejemplo 53.2f*

El ejemplo Fortran utiliza matrices asignables para la memoria dinámica en el dispositivo.

subrutina mult (p, N, iDev)

omp\_lib utilizar, únicamente: omp\_is\_initial\_device :: reales p (N)

reales, asignable :: v1 (:), v2 (:) :: número entero i, iDev

! $ OMP objetivo declare (init)

!!

! $ OMP tarea depende (Salida: v1, v2)

! Dispositivo de destino $ OMP (iDev) mapa (v1, v2) si (omp\_is\_initial\_device ()) y

STOP "no ejecutar en el dispositivo de objetivo" asignar (v1 (N), v2 (N))

llamar a init (v1, v2, N)

! $ Objetivo final omp

! $ Tarea final omp

!!

llamar a foo ()! ejecutar otros trabajos asychronously

!!

! $ OMP tarea depende (en: v1, v2)

! Dispositivo de destino $ OMP (iDev) mapa (a: v1, v2) mapa (a partir de: p)

si (omp\_is\_initial\_device ()) y

dejar de "no ejecutar en el dispositivo de destino"

! $ OMP paralelo hagas i = 1, N

p (i) = v1 (i) \* v2 (i) final hacer

deallocate (v1, v2)

!!

! $ Objetivo final omp

! $ Tarea final omp

!!

llamar de salida (p, N)

!!

subrutina final

##### Fortran



## Las secciones de matriz en construcciones de dispositivo

Los siguientes ejemplos muestran el uso de secciones de matriz en mapa cláusulas sobre objetivo

y apuntar a constructos de datos.

Este ejemplo muestra el uso no válido de dos secciones separadas de la misma matriz dentro de un constructo diana.

C / C ++



*Ejemplo 54.1c*

void foo ()

{

int A [30];

#pragma omp mapa de datos de destino (A [0: 4])

{

/ \* No puede asignar distintas partes de la misma matriz \* / #pragma mapa de destino omp (A [07:20])

{

A [2] = 0;

}

}

}

##### C / C ++



Fortran



*Ejemplo 54.1f*

foo subrutina () número entero :: A (30)

A = 1

! $ OMP mapa de datos de destino (A (1: 4))

! No se puede asignar distintas partes de la misma matriz

Mapa del objetivo $ OMP (A (8:27)) A (3) = 0

! $ OMP mapa de destino final

! $ OMP final subrutina extremo datos de destino

##### Fortran



Este ejemplo muestra el uso no válido de dos secciones separadas de la misma matriz dentro de un constructo diana.

C / C ++



*Ejemplo 54.2c*

void foo ()

{

int A [30], \* p;

#pragma omp mapa de datos de destino (A [0: 4])

{

p = & A [0];

/ \* Válido porque p [3] y A [3] son ​​los mismos

* **ubicación en el host pero la sección de matriz**
* **especificado a través de p [...] no es un subconjunto de A [0: 4] \* / #pragma mapa de destino omp (p [03:20])**

{

A [2] = 0;

p [8] = 0;

}

}

}

##### C / C ++



Fortran



*Ejemplo 54.2f*

foo subrutina () número entero, meta :: A (30) entero, puntero :: p (:)

A = 1

! $ OMP mapa de datos de destino (A (1: 4)) p => A

! válido porque p (4) y A (4) son los mismos

! ubicación en el host pero la sección de matriz

! especificado a través de p (...) no es un subconjunto de A (1: 4)

! Mapa de destino $ OMP (p (04:23))

A (3) = 0

p (9) = 0

! $ Objetivo final omp

! $ OMP final subrutina extremo datos de destino

##### Fortran



Este ejemplo muestra el uso válido de dos secciones separadas de la misma matriz dentro de un constructo diana.

C / C ++



*Ejemplo 54.3c*

void foo ()

{

int A [30], \* p;

#pragma omp mapa de datos de destino (A [0: 4])

{

p = & A [0];

#pragma mapa de destino omp (p [07:20])

{

A [2] = 0;

p [8] = 0;

}

}

}

##### C / C ++



Fortran



*Ejemplo 54.3f*

foo subrutina () número entero, meta :: A (30) entero, puntero :: p (:)

! $ OMP mapa de datos de destino (A (1: 4)) p => A

! Mapa de destino $ OMP (p (8:27))

A (3) = 0

p (9) = 0

! $ OMP mapa de destino final

! $ OMP final subrutina extremo datos de destino

##### Fortran



Este ejemplo muestra el uso válido de una sección de matriz totalmente contenida de una sección de matriz ya asignada dentro de un constructo diana.

C / C ++



*Ejemplo 54.4c*

void foo ()

{

int A [30];

#pragma omp mapa de datos de destino (A [doce y diez])

{

p = & A [0];

#pragma mapa de destino omp (p [3: 7])

{

A [2] = 0;

p [8] = 0;

A [8] = 1;

}

}

}

##### C / C ++



Fortran



*Ejemplo 54.4f*

foo subrutina () número entero, meta :: A (30) entero, puntero :: p (:)

! $ OMP mapa de datos de destino (A (1:10)) p => A

! Mapa de destino $ OMP (p (04:10))

A (3) = 0

p (9) = 0

A (9) = 1

! $ Objetivo final omp

! $ OMP final subrutina extremo datos de destino

##### Fortran



## rutinas de dispositivos

**rutina omp\_is\_initial\_device**

El siguiente ejemplo muestra cómo la rutina de biblioteca omp\_is\_initial\_device tiempo de ejecución puede ser utilizado para consultar si un código está ejecutando en el dispositivo host inicial o en un dispositivo de destino. El ejemplo a continuación, establece el número de hilos en la región paralelos basados ​​en donde se está ejecutando el código.

*Ejemplo 55.1c*



C / C ++

#include <stdio.h> #include <omp.h>

objetivo omp declare #pragma

vec\_mult void (float \* p, float \* v1, float \* v2, int N); extern float \* p \*, v1, v2 \*;

extern int N;

objetivo declare extremo omp #pragma

init\_vars extern void (\* float, float \*, int); extern salida nula (float \*, int);

void foo ()

{

N = init\_vars (& p, y v1, v2 y);

dispositivo de destino omp #pragma (42) mapa (p [: N], v1 [: N], v2 [: N])

{

vec\_mult (p, v1, v2, N);

}

de salida (p, N);

}

vec\_mult void (float \* p, float \* v1, float \* v2, int N)

{

int i;

int nthreads = omp\_is\_initial\_device ()? 8: 1024; if (! omp\_is\_initial\_device ())

{

printf ( "1024 roscas de dispositivo de destino \ n"); nthreads = 1,024;

}

más

{

printf ( "8 hilos en el dispositivo inicial \ n"); nthreads = 8;

}

#pragma omp paralelo para (I) NUM\_THREADS privadas (nthreads);

for (i = 0; i <N; i ++) p [i] = v1 [i] \* v2 [i];

}

##### C / C ++



Fortran



*Ejemplo 55.1f*

módulo entero params, parámetro :: N = 1024

extremo del módulo params módulo vmult contiene

subrutina vec\_mult (p, v1, v2, N)

omp\_lib uso, SOLAMENTE: omp\_is\_initial\_device

! $ Objetivo de declarar omp

real :: p (N), v1 (N), v2 (N) número entero :: i, nthreads, N

si (.not. omp\_is\_initial\_device ()) a continuación de impresión \*, "1024 roscas de dispositivo de destino" nthreads = 1024

más

impresión \*, "8 hilos en el dispositivo inicial" nthreads = 8

terminara si

! $ OMP paralelo do (i) NUM\_THREADS privadas (nthreads) hacen i = 1, N

p (i) = v1 (i) \* v2 (i) final hacer

final subrutina vec\_mult vmult módulo extremo

params uso prog\_vec\_mult programa

utilizar vmult

real :: P (N), v1 (N), V2 (N) llamada init (v1, v2, N)

! Dispositivo de destino $ OMP (42) Mapa (p, v1, v2) llamada vec\_mult (p, v1, v2, N)

! Salida de llamada de destino final $ OMP (p, n)

fin del programa

##### Fortran



**omp\_get\_num\_devices rutina**

El siguiente ejemplo muestra cómo la rutina de biblioteca omp\_get\_num\_devices tiempo de ejecución se puede utilizar para determinar el número de dispositivos.

*Ejemplo 55.2c*



C / C ++

# include <omp.h>

extern void init (float \*, float \*, int); extern salida nula (float \*, int);

vec\_mult void (float \* p, float \* v1, float \* v2, int N)

{

int i;

init (v1, v2, N);

int ndev = omp\_get\_num\_devices ();

int do\_offload = (ndev> 0 && N> 1.000.000);

objetivo #pragma omp si (do\_offload) mapa (a: v1 [0: N], v2 [: N]) mapa (de: p [0: N]) #pragma omp paralelo para si (N> 1000) privada (i )

for (i = 0; i <N; i ++) p [i] = v1 [i] \* v2 [i];

de salida (p, N);

}

##### C / C ++



Fortran



*Ejemplo 55.2f*

subrutina vec\_mult (p, v1, v2, N)

omp\_lib utilizar, únicamente: omp\_get\_num\_devices real :: p (N), v1 (N), v2 (N)

número entero :: N, i, ndev lógico :: do\_offload

llamar a init (v1, v2, N)

ndev = omp\_get\_num\_devices ()

do\_offload = (ndev> 0) .y. (N> 1.000.000)

! Objetivo de $ OMP si (do\_offload) mapa (a: v1, v2) mapa (a partir de: p)

! $ OMP paralelo hacer si (N> 1000) hacer i = 1, N

p (i) = v1 (i) \* v2 (i) final hacer

! Salida de llamada de destino final $ OMP (p, n)

subrutina final

##### Fortran



**omp\_set\_default\_device y**

**rutinas omp\_get\_default\_device**

El siguiente ejemplo muestra cómo las omp\_set\_default\_device y de tiempo de ejecución omp\_get\_default\_device rutinas de biblioteca se pueden utilizar para fijar el dispositivo predeterminado y determinar el dispositivo predeterminado, respectivamente.

C / C ++



*Ejemplo 55.3c*

# include <omp.h> #include <stdio.h> foo (void)

{

int default\_device = omp\_get\_default\_device (); printf ( "Default device =% d \ n", default\_device); omp\_set\_default\_device (default\_device + 1);

if (! omp\_get\_default\_device () = default\_device + 1) printf ( "Dispositivo predeterminado es todavía =% d \ n", default\_device);

}

##### C / C ++



Fortran



*Ejemplo 55.3f*

foo programa

utilizar omp\_lib, SOLAMENTE: omp\_get\_default\_device, omp\_set\_default\_device número entero :: old\_default\_device, new\_default\_device

old\_default\_device = omp\_get\_default\_device () print \*, "Dispositivo predeterminado =", old\_default\_device new\_default\_device = old\_default\_device + 1 llamada omp\_set\_default\_device (new\_default\_device)

si (omp\_get\_default\_device () == old\_default\_device) e imprimir \*, "Dispositivo predeterminado todavía es =", old\_default\_device

fin del programa

##### Fortran



## Fortran Construir ASOCIADO

Fortran



*Ejemplo 56.1f*

Este es un ejemplo válido de especificar un nombre asociado en una cláusula de intercambio de datos de atributos. La restricción en el atributo compartido de datos de reglas en la sección 4.0 de las especificaciones API OpenMP establece que un nombre asociado conserva la asociación con el selector establecido en la declaración ASOCIADO. El nombre asociado b está asociado con la variable compartida a. Con la regla de atributo de intercambio de datos predeterminado, no se permite el nombre asociado b que se especifica en la cláusula privado.

ejemplo de programa

real :: a, c asociado (b => a)

! $ OMP paralelo privada (b, do)! no válido para privatizar segundo c = 2,0 \* b

! $ Final omp paralelo

fin al programa final asociado

#### *Ejemplo 56.2f*

En este ejemplo, dentro de la construcción paralela, el nombre de la asociación thread\_id se asocia con la copia privada de i. La sentencia print debe hacer salir el número del hilo único.

ejemplo de programa

utilizar omp\_lib i número entero

! $ OMP paralelo privada (i)

i = omp\_get\_thread\_num () asociado (thread\_id => i)

impresión \*, thread\_id! imprimir i privada valor asociado finales

! $ Final omp paralelo

fin del programa

#### *Ejemplo 56.3f*

Este ejemplo ilustra el efecto de especificar un nombre de selector en una cláusula atributo de intercambio de datos. El nombre asociado u se asocia con v y la variable v se especifica en la cláusula privado de la construcción en paralelo. La asociación constructo se estableció antes de la región paralela. La asociación entre u y v la original se conserva (ver el Atributo Sharing datos de reglas sección en el OpenMP

4.0 Especificaciones API). Dentro de la región paralela, v tiene el valor de -1 y u tiene el valor de la original v.

programa de ejemplo número entero :: vv = 15

asociado (u => v)

! $ OMP paralelo privada (v) v = -1

impresión \*, v! v privado = -1

impresión \*, u! original v = 15

! $ OMP extremo asociado finales paralelo

fin del programa

Fortran

