“Paralelizar” un Código usando OpenMP

Adolfo J. Banchio
FaMAF, IFEG-CONICET,
Universidad Nacional de Córdoba

Contenido

- Introducción
  - Paralelización: ¿por qué OpenMP?
  - Qué es OpenMP?
  - OpenMP vs. MPI
- OpenMP: una introducción
  - Modelo de paralelización
  - Algunas directivas y su uso
  - Library routines y Variables de entorno
- Conclusiones

¿Qué es OpenMP?

- Es un **API** (Application Program Interface) para programación en paralelo (multi-threaded) con memoria compartida.
- Basado en tres componentes primarias:
  - **Directivas** para el compilador
  - **Runtime library routines**
  - **Variables de entorno**
- Portable:
  - Especificada para **FORTRAN**, C y C++ (Necesita soporte por parte del compilador)
  - Multiplataforma (Unix/Linux, Windows)
- Estandarizado
  - Definido y apoyado conjuntamente por un grupo de grandes firmas de Hardware y Software (Intel, HP, SGI, SUN, etc.)

¿Qué NO es OpenMP?

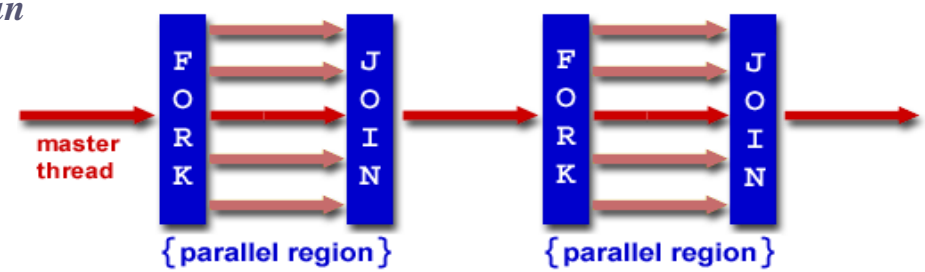
- No está ensado para sistemas de memoria distribuida (solo).
- No debe ser implementado en forma idéntica por las distintas firmas.
- No garantiza el uso más eficiente de la memoria compartida
- No controla: dependencias, conflictos en los datos, *race conditions*, ni *deadlocks*.
- No controla secuencias de código que puedan causar que el programa sea clasificado como *non-conforming*.
- No está diseñado para garantizar que el ***input*** o el ***output*** al mismo archivo sea síncrono cuando es ejecutado en paralelo. **El programador es responsable de la sincronización de la entrada y salida.**

MPI *vs.* OpenMP

- MPI puro
  - Pros:
    - Portable a máquinas con memoria distribuida o compartida.
    - “Escalea” más alla de un nodo
    - No data placement problem
  - Contras:
    - Difícil de desarrollar y corregir
    - High latency, low bandwidth
    - Communicación explícita
    - Difficult load balancing
- OpenMP puro
  - Pros:
    - Fácil de implementar paralelismo
      - Simple y limitado set de directivas
      - Permite paralelizar un código serial en forma incremental
      - Paralelismo *Coarse-grain* y *fine-grain*
    - Low latency, high bandwidth
    - Communicación implícita
    - Dynamic load balancing
  - Contras:
    - Sólo en máquinas con memoria compartida
    - “Escalea” sólo en un nodo
    - Possible data placement problem

OpenMP: Modelo de paralelización

- Paralelismo en **memoria compartida** basado en hilos (**threads**): un proceso cuenta con varios hilos
- Paralelismo explícito: no automático, el programador tiene el control.
- Modelo ***Fork-Join***



Todos los programas comienzan como un proceso simple: el *master thread*. Éste ejecuta el código en forma secuencial hasta la primera región paralela. ...

- Paralelismo basado en directivas para el compilador incluidas el el código fuente.
- Admite *Nested paralelism*
- *I/O*: OpenMP no especifica nada acerca de I/O.
- Modelo de memoria: los hilos pueden tener sus datos en *cache* y estos no necesariamente tienen que ser consistentes con la memoria real todo el tiempo.

Ejemplo de estructura de programa con OpenMP

```
        PROGRAM HELLO
        INTEGER VAR1, VAR2, VAR3
        Serial code
              .
              .
              .
        Beginning of parallel section. Fork a team of threads.
        Specify variable scoping
!$OMP PARALLEL PRIVATE(VAR1, VAR2) SHARED(VAR3)
        Parallel section executed by all threads
              .
              .
              .
        All threads join master thread and disband
!$OMP END PARALLEL
        Resume serial code
              .
              .
              .
        END
```

Directivas OpenMP

- FORTRAN

SENTINEL	NOMBRE DE LA DIRECTIVA	[CLAUSULA ...]
Todas las directivas de OpenMP deben empezar con un sentinel. Los sentinels aceptados, dependiendo el tipo de fuente Fortran, son: **!$OMP** **C$OMP** ***$OMP**	Una directiva de OpenMP válida. Debe aparecer despues del sentinel y antes que cualquier cláusula.	Opcionales. Las cláusulas pueden estar en cualquier orden, y repetidas de acuerdo a la necesidad, a menos que este explicitamente restringido.

- Ejemplo

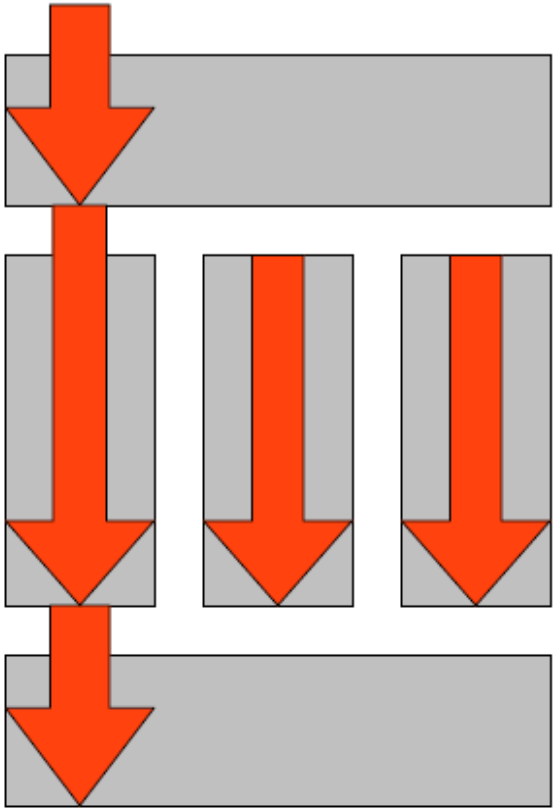
```
!$OMP PARALLEL PRIVATE(VAR1, VAR2) SHARED(VAR3)
```

- C / C++ (case sensitive!)

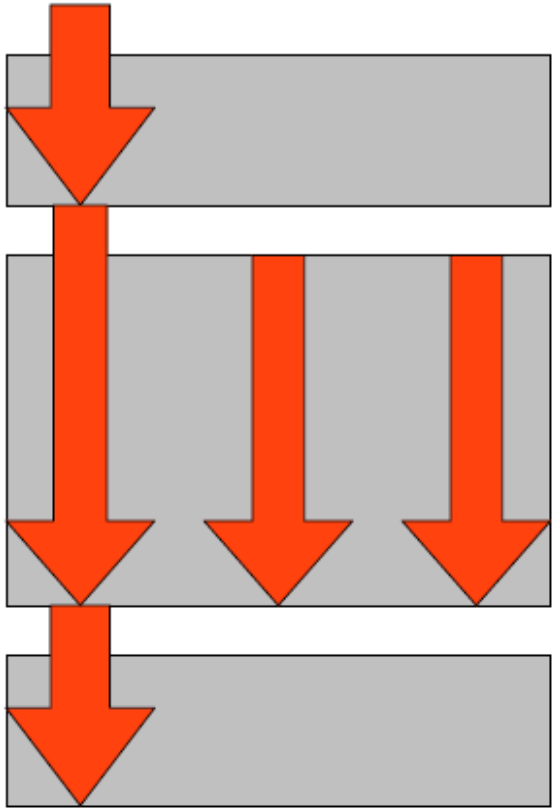
```
#pragma omp parallel private(VAR1, VAR2) shared(VAR3)
```

Formas de paralelismo

Secciones Independientes



Loops



Directivas: PARALLEL

- Es un bloque de programa que va a ser ejecutado por todos los hilos.

```
!$OMP PARALLEL DEFAULT(NONE), PRIVATE(i)   &
!$OMP SHARED(a,b,n)
!
 ... bloque estructurado de programa ...
!
!$OMP END PARALLEL
```

Cláusulas: DEFAULT(SHARED | PRIVATE | FIRSTPRIVATE | NONE),
PRIVATE(var. list.), SHARED(var. list.)
FIRSTPRIVATE (var. list), REDUCTION (operator list
COPYIN (list), NUM_THREADS (scalar-int.-expr),
IF (scalar-logical-expr)

- Cuando un hilo llega a una region PARALLEL crea un conjunto de hilos del cual pasa a ser el master.
- El código es replicado y todos los hilos ejecutan el mismo código
- El número de hilos creados puede ser controlado

Directivas: SECTIONS

SECTIONS

```
!$OMP PARALLEL DEFAULT(NONE), PRIVATE(i) &
!$OMP SHARED(a,b,n)
!
 ... bloque estructurado de programa ...
!
!$OMP SECTIONS
   do i=1,n
       a(i)=a(i)+b(i)
    enddo

!$OMP SECTION
... bloque estructurado de programa ...

!$OMP SECTION
... bloque estructurado de programa ...

!$OMP END SECTIONS
!$OMP END PARALLEL
```

Directivas: DO

DO

```
!$OMP PARALLEL DO PRIVATE(i) &
!$OMP SHARED(a,b,n)

!$OMP DO

      do i=1,n
        a(i)=a(i)+b(i)
      enddo

!$OMP END DO  [NOWAIT]

!$OMP END PARALLEL
```

Cláusulas: DEFAULT(SHARED | PRIVATE | NONE),
PRIVATE(var. list.), SHARED(var. list.)
REDUCTION(operator:variable),
SCHEDULE(STATIC | DYNAMIC | GUIDED | RUNTIME)

Directivas combinadas

PARALLEL DO

```
!$OMP PARALLEL DO PRIVATE(i)      &
!$OMP SHARED(a,b,n)
        do i=1,n
          a(i)=a(i)+b(i)
        enddo
!$OMP END PARALLEL DO
```

PARALLEL SECTIONS

```
!$OMP PARALLEL SECTIONS DEFAULT(NONE), PRIVATE(i) &
!$OMP SHARED(a,b,n)
!
!$OMP SECTIONS
... bloque estructurado de programa ...

!$OMP SECTION
... bloque estructurado de programa ...

!$OMP END PARALLEL SECTIONS
```

Directivas: WORKSHARE

WORKSHARE

```
SUBROUTINE A11_1(AA, BB, CC, DD, EE, FF, N)
INTEGER N
REAL AA(N,N), BB(N,N), CC(N,N), DD(N,N), EE(N,N),
FF(N,N)

!$OMP PARALLEL
!$OMP WORKSHARE
    AA = BB
    CC = DD
    EE = FF
!$OMP END WORKSHARE
!$OMP END PARALLEL

END SUBROUTINE A11_1
```

OTRAS Directivas

- SINGLE: espicifica una región que solo será ejecutada por un hilo del conjunto. Los demás esperan.
- MASTER: espicifica una región que solo será ejecutada por el master. No hay barrera al
- CRITICAL: espicifica una región que solo será ejecutada por un hilo a la vez.
- ATOMIC: la ubicación de memoria va a ser acutalizada por un hilo a la vez. *(mini critical region)*
- BARRIER: sincroniza todos los hilos del conjunto.
- FLUSH: las variables visibles al hilo son escritas a la memoria en este punto.

- TASK (nuevo en la version 3.0)

Dependencias (Race Condition)

Hay dependencia, por ejemplo, si en una iteración (digamos i_1) escribe una variable a(k), y otra iteración (i_2) lee el valor de esa varible a(k).

- Do loop con dep.

```
do i=1,n
    a(i) = a(i) + a(i-1)
enddo
```

- Do loop sin dep.

```
!$OMP PARALLEL DO PRIVATE(i) &
!$OMP  SHARED(a,n)

 do i=1,n,2
     a(i)    = a(i)    + a(i-1)
 enddo

!$OMP END PARALLEL DO
```

REDUCTION (Cláusula)

- Realiza una reducción de la variable que aparece en la lista
- Crea una copia privada de cada variable de la lista para cada hilo. Al final se aplica la reducción y el resultado final es escrito en la varible global.

```
!$OMP PARALLEL DO PRIVATE(i)  &
!$OMP    SHARED(a,n)  &
!$OMP  Reduction(+:result)

 do i=1,n
    result    = result  + a(i)
 enddo

!$OMP END PARALLEL DO
```

- Aplican varias restricciones. Entre ellas, la variable debe ser escalar y compartida.
- Operadores: +, *, -, .AND., .OR., .EQV., .NEQV.

Library Routines y Enviroment Variables

Algunas rutinas:

- OMP_SET_NUM_THREADS( )
- OMP_GET_NUM_THREADS( )
- OMP _GET_THREAD_NUM( )
- OMP_GET_WTIME()

Algunas variables de entorno:

- OMP_NUM_THREADS *int_literal*
- OMP_SCHEDULE “schedule[, chunk_size]”

Conclusions

- **Paralelización usando OpenMP para usar en máquinas con memoria compartida, es SIMPLE, no necesita mayores modificaciones al programa, y en gral. da buenos resultados para modestos números de procesadores.**
- **En gral. la paralelizacion está basada en los DO LOOPS (aunque no necesariamente -> TASKs en OpenMP v. 3.0).**
- **Para optimizar la ejecución en paralelo, es necesario considerar la "alocación" de memoria, la forma en que se "reparte" (SCHEDULING) el trabajo en los loops, considerar el uso de los caches, etc..**

REFERENCIAS:

OpenMP website: openmp.org

API specifications, FAQ, presentations, discussions, media releases, calendar, membership application and more...

Scheduling for OpenMP

- **Static**: Loops are divided into *#thrds* partitions, each containing *ceiling(#iters/#thrds)* iterations.
- **Guided**: Loops are divided into progressively smaller chunks until the chunk size is 1. The first chunk contains *ceiling(#iters/#thrds)* iterations. Subsequent chunk contains *ceiling(#left_iters/#thrds)* iterations.
- **Dynamic, n**: Loops are divided into chunks containing *n* iterations. Each thread executes a chunk of iterations, then requests another chunk until no chunks remains to be distributed.
- **Runtime**: Schedule and chunk size are taken from run-sched-var.
- **Auto**: The decision regarding scheduling is delegated to the compiler and/or runtime system.