Buenos días Don Daniel. ¡Vargas Cipollo!.

Seminario Práctico 3. Arquitectura de Computadores

- 2º curso / 2º cuatr.
- Grado en Ing. Informática
- Material elaborado por Mancia Anguita

Seminario 3. Herramientas de programación paralela III: Interacción con el entorno en OpenMP y evaluación de prestaciones

- Arquitectura de Computadores

Interacción con el entorno (v3.0 en gris)

- **Objetivos:**
  - Consultar: obtener información (p. ej. nº de threads o tipo de planificación de tareas)
  - Modificar: influir en la ejecución (p. ej. fijar nº de threads o fijar el tipo de planificación de tareas)
- **Relacionado con el entorno de ejecución:**
  - **Variables de control internas**
    - V2.5: nthreads-var, dyn-var, nest-var, run-sched-var, def-sched-var
    - V3.0: thread-limit-var…
  - **Variables de entorno** (ámbito: los códigos que se ejecuten a partir de su modificación)
    - V2.5: OMP_NUM_THREADS, OMP_DYNAMIC, OMP_NESTED, OMP_SCHEDULE
    - V3.0: OMP_THREAD_LIMIT, …
  - **Funciones del entorno de ejecución** (ámbito: el código que las usa)
    - V2.5: omp_get_dynamic(), omp_set_dynamic(), omp_get_max_threads(), omp_set_num_threads(), omp_get_nested(), omp_set_nested(), omp_get_thread_num(), omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel()
    - V3.0: omp_get_thread_limit(), omp_get_schedule(kind,modifier), omp_set_schedule(kind, modifier) …
  - **Cláusulas** (no modifican variables de control) (ámbito: directiva que las usa)
    - V2.5: if, schedule, num_threads

Contenido

1. Variables de control
2. Variables de entorno
3. Funciones del entorno de ejecución
4. Cláusulas para interaccionar con el entorno

5. Clasificación de las funciones de la biblioteca OpenMP
6. Funciones para obtener el tiempo de ejecución

Variables de control internas que afectan a una región parallel

Variable de control	Ámbito	Valor (inicial)	¿Qué controla?	Consultar / Modificar
dyn-var	entorno de datos	true/false (implem.)	Ajuste dinámico del nº de threads	sí(f) / sí(ve,f)
nthreads-var	entorno de datos	número (implem.)	threads en la siguiente región parallel	sí(f) / sí(ve,f)
thread-limit-var (v3)	entorno de datos	número (implem.)	Máximo nº de threads para todo el programa	sí(f) / sí(ve,-)

(f: función, ve: variable de entorno)

Variables de control internas que afectan a regiones DO/loop

Variable de control	Ámbito	Valor (inicial)	¿Qué controla?	Consultar / Modificar
run-sched-var	entorno de datos	(kind[,chunk]) (implem.)	Planificación de bucles para runtime	sí(f) / sí(ve,f)
def-sched-var	dispositivo	(kind[,chunk]) (implem.)	Planificación de bucles por defecto (ámbito: programa)	no / no

(v3.0 en gris)

Variables de entorno

Variable de control	Variable de entorno	Ejemplos de modificación (bash/ksh)
dyn-var	OMP_DYNAMIC	`export OMP_DYNAMIC=FALSE` `export OMP_DYNAMIC=TRUE`
nthreads-var	OMP_NUM_THREADS	`export OMP_NUM_THREADS=8`
thread-limit-var	OMP_THREAD_LIMIT	`export OMP_THREAD_LIMIT=8`
run-sched-var	OMP_SCHEDULE	`export OMP_SCHEDULE="static,4"` `export OMP_SCHEDULE="dynamic"` …
def-sched-var	(—)	(v3.0 en gris)

Funciones del entorno de ejecución

Variable de control	Rutina para consultar	Rutina para modificar
dyn-var	omp_get_dynamic()	omp_set_dynamic()
nthreads-var	omp_get_max_threads()	omp_set_num_threads()
thread-limit-var	omp_get_thread_limit()	(—)
nest-var	omp_get_nested()	omp_set_nested()
run-sched-var	omp_get_schedule(&kind,&mod)	omp_set_schedule(kind,modifier)
def-sched-var	no	no

typedef enum omp_sched_t

- omp_sched_static = 0x1
- omp_sched_dynamic = 0x2
- omp_sched_guided = 0x3
- omp_sched_auto = 0x4
- omp_sched_monotonic = 0x80000000u

Otras rutinas del entorno v2.5

- omp_get_thread_num(): devuelve el identificador del thread en el grupo
- omp_get_num_threads(): nº de threads en una región parallel (1 si es secuencial)
- omp_get_num_procs(): nº de procesadores disponibles
- omp_in_parallel(): true si se llama dentro de una región parallel activa

---

Cláusulas que interaccionan con el entorno

Tipo de cláusula y directivas que las admiten

Cláusula	parallel	parallel DO/for	parallel sections	single
if	X	X	X	
num_threads	X	X	X	
shared	X	X	X	X
private	X	X	X	X
lastprivate		X		X
firstprivate	X	X	X	X
default	X			X
reduction	X	X	X	X
copyin	X	X	X	
copyprivate			X	

Cláusula	parallel	parallel DO/for	parallel sections	single
schedule		X		
ordered		X		
nowait		X		X

¿Cuántos threads se usan? Orden de precedencia para fijar el nº de threads:

1. Evaluación de la cláusula `if`
2. Cláusula `num_threads`
3. Función `omp_set_num_threads()`
4. Variable de entorno `OMP_NUM_THREADS`
5. Por defecto (implementación): normalmente nº de cores del nodo

Ejemplo de uso:

```
#pragma omp parallel num_threads(8) if(N>20)
```

Cláusula `if`

- Sintaxis (C/C++): `if(scalar-exp)`
- No hay ejecución paralela si la condición es falsa
- Precaución: sólo válida en construcciones `parallel`

Ejemplo de código en C con OpenMP

```
#pragma omp parallel if(n>4) default(none) \
  private(sumalocal,tid) shared(a,suma,n)
{
  // …
}
```

Cláusula `schedule`

- Sintaxis: `schedule([modifier:]kind[,chunk])`
  - **kind**: static, dynamic, guided, auto, runtime
  - **chunk**: granularidad de iteraciones
  - **modifier**: monotonic, nonmonotonic
- Precauciones: sólo en bucles; no asumir granularidad por defecto

Tipos de `schedule`:

1. **static,chunk**: unidades de `chunk` en round-robin

2. **dynamic,chunk**: asignación en tiempo de ejecución; buena si duración iteraciones variable
3. **guided,chunk**: bloques decrecientes; menos sobrecarga que dynamic
4. **runtime**: tipo fijo por variable de control `run-sched-var`

Ejemplo:

```
#pragma omp parallel for schedule(dynamic,chunk)
for (i=0; i<n; i++) { … }
```

Funciones de la biblioteca OpenMP

- **Sincronización con cerrojos**
  - V2.5: omp_init_lock(), omp_destroy_lock(), omp_set_lock(), omp_unset_lock(), omp_test_lock()
  - V3.0: omp_destroy_nest_lock(), omp_set_nest_lock(), omp_unset_nest_lock(), omp_test_nest_lock()
- **Tiempos de ejecución**
  - omp_get_wtime(), omp_get_wtick()

Ejemplo: cálculo de PI en C secuencial

```
clock_gettime(CLOCK_REALTIME,&cgt1);
// cálculo sumatorio
clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt = (cgt2-cgt1) en segundos;
printf("Iteraciones:%d PI:%26.24f Threads:1 Tiempo:%8.6f\n", intervals,sum,ncgt);
```

Datos de ejemplo:

- Iteraciones: 10 000 000 → Tiempo: 0.194065 s
- Iteraciones: 40 000 000 → Tiempo: 0.561454 s

Ejemplo: cálculo de PI con OpenMP/C

```
t = omp_get_wtime();
#pragma omp parallel
{
  #pragma omp for reduction(+:sum) private(x)
  for (i=0; i<intervals; i++) { … }
}
t = omp_get_wtime() - t;
printf("Iteraciones:%d Pi:%26.24f Threads:%d Tiempo:%8.6f\n",
       intervals,sum,omp_get_max_threads(),t);
```

Datos de ejemplo (4 proc. × 4 cores):

- Threads 8: 10 000 000 → 0.016534 s
- Threads 4: 10 000 000 → 0.029227 s
- Threads 2: 10 000 000 → 0.055943 s
- Threads 1: 10 000 000 → 0.105901 s

Ejemplo: cálculo de PI en MPI/C

```
MPI_Init(...);
MPI_Comm_size(...,&nproc);
MPI_Comm_rank(...,&iproc);
for (i=iproc; i<intervals; i+=nproc) { … }
MPI_Reduce(&lsum,&sum,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
t = MPI_Wtime() - t;
if (!iproc)
  printf("Iteraciones:%d PI:%26.24f Procesos:%d Tiempo:%8.6f\n",
         intervals,sum,nproc,t);
```

Datos de ejemplo (4 proc. × 4 cores):

- Procesos 8: 10 000 000 → 2.281467 s
- Procesos 4: 10 000 000 → 1.116629 s
- Procesos 2: 10 000 000 → 0.119861 s
- Procesos 1: 10 000 000 → 0.156071 s

Fin del Seminario 3: Interacción con el entorno en OpenMP