Padreando-el-BP3.pdf

BlackTyson

Arquitectura de Computadores

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Interacción en el entorno:
Variables de control que afectan a Parallell:

- **dyn- var**:
    - Ámbito: entorno de datos
    - Valor inicial: true/false
    - Controla nº dinámico de threads
    - Puede consultarse y modificarse
    - Variable de entorno asociada: OMP_DYNAMIC
    - Función para consultar: omp_get_dynamic()
    - Función para modificar: omp_set_dynamic()

- **nthreads-var**:
    - Ámbito: entorno de datos
    - Valor inicial: número.
    - Controla los threads de la siguiente instruccion paralela
    - Puede consultarse y modificarse.
    - Variable de entorno asociada: OMP_NUM_THREADS
    - Función para consultar: omp_get_max_threads()
    - Función para modificar: omp_set_num_threads()
- **thread-limit-var**:
    - Ámbito: entorno de datos
    - Valor inicial: número
    - Controla el máximo número de threads del programa
    - Puede consultarse y modificarse
    - Variable de entorno asociada: OMP_THREAD_LIMIT
    - Función para consultar: omp_get_thread_limit()
- **nest-var**:
    - Entorno: n/A
    - Valor inicial: true/false
    - Controla el paralelismo anidado
    - Puede consultarse y modificarse
    - variable de entorno asociada: OMP_NESTED
    - En desuso a partir de 5.0
    - Función para consultar: omp_get_nested()
    - Función para modificar: omp_set_nested()

Variables de control que afectan a **do/loop:**

- **run-sched-var:**
    - Ámbito: entorno de datos
    - Valor inicial: kind,[chunk]
    - Planifica los bucles para *runtime*
    - Puede consultarse y modificarse
    - Variable de entorno asociada: OMP_SCHEDULE
    - Función para consultar: omp_get_schedule(&kind,&chunk)
    - Función para modificar: omp_set_schedule(kind,chunk)

- **def-sched-var:**
    - Ámbito: dispositivo
    - Valor inicial: kind,[chunk]
    - Planifica bucles por defecto
    - No puede consultarse ni modificarse
    - No tiene variable de entorno asociada.
    - No hay función para consultar ni modificar.

Otras funciones:

- **omp_get_thread_num()**: devuelve al thread su id dentro del grupo
- **omp_get_num_thread()**: si estamos en región paralela devuelve el número de threads que hay y si estamos en código secuencial devuelve 1
- **omp_get_num_procs()**: devuelve el número de procesadores disponibles para ejecución
- **omp_in_parallel()**: si se llama dentro de una región paralela activa se devuelve true, en caso contrario false.

Orden de precedencia para fija threads:
1º If
2º num_threads
3º set_num_threads()
4º omp_num_threads()
5º Fijado por defecto.

Tipos de cláusulas:

- **if**(escalar-exp):
    - No se da ejecución paralela si no se cumple la condicion(excalar-exp)
    - Solo puede utilizarse en construcciones parallel
- **schedule**(kind[,chunk]):
    - Kind nos dice el tipo de schedule.
    - chunk nos dice la granularidad de la distribución.
    - Solo puede usarse en bucles.
    - Por defecto es static
    - Es recomendable no asumir un chunk por defecto.
    - Tipos de schedule:
        - .static:
            - Las iteraciones se dividen en unidades de chunk iteraciones, que se asignan de manera round robin. Por tanto el chunk de entrada es el número de iteraciones.
            - Se asigna un único chunk a cada thread.
        - .dynamic:
            - Es una distribución en tiempo de ejecución, útil cuando se desconoce el tiempo de ejecución de las iteraciones.
            - Tiene chunk iteraciones
            - Añade sobrecarga adicional.
        - .guided:

        - Es una distribución en tiempo de ejecución, útil cuando se desconoce el tiempo de ejecución de las iteraciones o su número.
        - Comienza con un bloque largo que va disminuyendo en función del número de iteraciones que quedan entre el nº de threads.
        - Añade sobrecarga pero menos que dynamic
    - .runtime
        - Se fija en tiempo de ejecución
        - Depende de la variable de control run-sched-var

Funciones para sincronizar con cerrojos:
omp_init_lock(), **omp_destroy_lock()**, **omp_set_lock()**, **omp_unset_lock()**, **omp_test_lock()**.

Funciones para obtener tiempo de ejecución:
omp_get_wtime(), **omp_get_wtick()**.

Preguntas que pueden caer:

1. **Que función usar para fijar el numero de hebras a n:** omp_set_num_threads(n)
2. **Que cláusula schedule no se fija en tiempo de ejecución:** .static
3. **En tamaños de problemas pequeños es conveniente paralelizar:** falso, la creación y destrucción de hebras consume tiempo y a veces no es worsit.
4. **El tamaño de chunk usando dynamic varía:** falso, pero puede asignarse un nuevo valor a chunk en tiempo de ejecución.
5. **Si queremos darle el máximo de prioridad a la condición que afecta a una directiva, que cláusula usarías:** if.
6. **Que schedule es el mayor generador de sobrecarga:** dynamic.
7. **Si tenemos un bucle paralelizado por schedule(runtime), que controla el reparto de código:** runtime
8. **Y si tenemos schedule (static) y dentro de la región un omp_set_schedule(kind, chunk):** el tipo de kind.
9. **Y si en lugar de ser omp_set_schedule es omp_get_schedule**: static
10. **Si tenemos un bucle anidado con iteradores con cargas diferentes, que tipo de reparto sería el más óptimo**: dynamic.
11. **Y si tuvieran la misma carga**: static