Sumario OpenMP

1º Variables

- **KMP_AFFINITY:** Variable de entorno Indica que núcleo tiene asignada la ejecución de un hilo concreto. Algunos valores que puede tomar esta variable son:
  - **SCATTER:** Asigna cada uno de los hilos a un núcleo físico diferente.
- **OMP_NUM_TREADS:** Permite modificar el número de hilos que se utilizarán en la siguiente sección paralela. El njúmero de hilos viene indicado por al variable *nthreads-var.*

2º Llamadas a la API

- **omp_get_num_threads():** Devuelve el número de hilos.
- **omp_get_max_threads():** Devuelve el número máximo de hilos.
- **omp_get_threads_num():** Devuelve el identificador del hilo que ejecuta la función.
- **omp_set_num_threads():** Modifica el número de hilos.
- **omp_get_num_procs():** Devuelve el número de procesadores de la máquina.
- **omp_get_time():** Devuelve el número de segundos conforme al reloj del sistema.
- **omp_set_scheudule():** Modifica la variable utilizada en la planificación de iteraciones.
- **omp_set_num_threads()**: Indica el número que se utilizarán para ejecutar la siguiente región paralela. Este número viene dado por la variable *nthreads-var.*
- **omp_is_initial_device():** Devuelve True si se ejecuta en el Host o False si se ejecuta en cualquier otro dispositivo.
- **omp_get_num_devices():** Devuelve el número de dispositivos disponibles.
- **omp_get_device_num():** Devuelve el número del dispositivo desde donde se ejecuta.
- **omp_get_default_device():** Fija el dispositivo indicado como dispositivo por defecto.
- **omp_get_num_teams():** Devuelve el número de equipos en la región paralela.

3º Pragmas

- **#pragma omp parallel:** Genera una región paralela, la cual abarcará la siguiente expresión o el bloque delimitado entre llaves. Esta pragma puede usar las siguientes clausulas:
  - **num_threads, if, shared, private, firstprivate, default y reduction.**
- **#pragma omp threadprivate (“varList”):** Permite crear un copia de las variable globales por cada hilo, las cuales son inicializadas automáticamente. Las variables persisten a lo largo de las regiones *parallel* si el número de hilos es el mismo.
- **#pragma omp parallel for:** Esta pragma referencia a un bucle y sirve para dividir las iteraciones del mismo con el objetivo de que ejecutadas de forma conjunta por los diferentes hilos del equipo. Esta pragma puede usar todas las clausulas propias de *#pragma omp parallel*, además de:
  - **num_threads, if, shared, private, firstprivate, default. Reduction, lastprivate, scheude, nowait, collapsed y reduction.**
- **#pragma omp section:** Genera una región paralela dividida en secciones separadas y discretas, de manera que cada sección puede ser ejecutada por un hilo de forma independiente. Esta pragma puede usar las siguientes clausulas:
  - ***private, lastprivate, firstprivate, reduction y nowait*.**
- **#pragma omp single:** Genera una región que se ejecuta de forma secuencial.
- **#pragma omp simd:** Indica al compilador la generación de una sección vectorial que utilice las operaciones SIMD del procesador. Esta pragma puede usar las siguientes clausulas
  - ***private, reductión, collapsed*, safelen, linear y aligned**
- **#pragma omp barrier:** Crea una barrera que los hilos no pueden superar hasta que todos los hilos que puedan alcanzarla hayan llegado a la misma. Algunas pragmas ya incorporan la barrera al final de su sección.
- **#pragma omp critical [name]:** Crea una sección de exclusión mutua donde únicamente un hilo puede estar al mismo tiempo. Las exclusión se realiza con respecto al nombre dado a la sección crítica, todas aquellas sin nombre se excluirán juntas.
- **#pragma omp atomic:** Realiza una exclusión mutua en torno a la variable indicada a continuación con el objetivo de realzar acciones de lectura y escritura sobre la misma. Normalmente es más eficiente que utilizar una sección crítica, ya que no provoca que los procesos queden esperando a que la variable deje de estar en uso.
- **#pragma omp task:** Genera una tarea con el objetivo de ejecutar el bloque de código referenciado entre llaves y asigna el mismo a un hilo de ejecución. Esta pragma puede usar las siguientes clausulas:
  - ***shared, private, firstprivate, default, in_reduction, depend, untied, detach***
- **#pragma omp taskyield:** Indica al asignador de tareas que migre las tareas desde el hilo que la esta ejecutando actualmente a otro hilo diferente. Esto no obliga al asignador a hacer la migración, sino que lo hará si determina que merece la pena llevarlo a cabo.
- **#pragma omp taskwait:** Suspende la ejecución de una tarea hasta que sean completadas todas las tareas hijas de la misma.
- **#pragma omp target:** Descarga el código del kernel en el dispositivo indicado para su posterior ejecución en el mismo, aunque sin explotar el paralelismo. Se pueden utilizar las clausulas:
  - **device, if, private, firstprivate, map, reduction, nowait**

- **#pragma omp target enter data:** Asigna variables al entorno de datos del dispositivo seleccionado. La instrucción puede reducir las copias de datos que van hacia y desde el dispositivo en el caso de que varias regiones del mismos utilicen los mismos datos. Se pueden utilizar las siguientes clausulas:
  - **device, if, map, nowait**

- **#pragma omp target exit data:** Elimina las variables mapeadas en el entorno de datos del dispositivo seleccionado. La instrucción puede limitar la cantidad de memoria del dispositivo en el caso de que se haya utilizado la instrucción "*pragma omp target enter data"* inicialmente para mapear aquellos datos que se pretenden eliminar. Se pueden utilizar las siguientes clausulas:
  - **device, if, map, nowait**

- **#pragma omp target update:** Hace que los elementos indicados, los cuales se encuentran mapeados en el entorno de datos del dispositivo, sean coherentes con los datos almacenados en sus correspondientes versiones dentro del host. Se pueden utilizar las siguientes clausulas:
  - **to, from, device, if, nowait**

- **#pragma omp teams:** Crea uno o más equipos de hilos que ejecutarán el código indicado. Sin embargo, esta pragma no distribuye la carga del computo, luego todo el trabajo será realzado por el hilo maestro. Se pueden utilizar las siguientes clausulas:
  - **default, firstprivate, private, reduction, shared, num_teams, thread_limit**
- **#pragma omp distribute:** Especifica que los equipos creados en una determinada región paralela, ejecutarán las diferentes interacciones del bucle que compone dicha región distribuyendo las interlineaciones del mismo de manera equitativa. Se pueden utilizar las siguientes clausulas:
  - **firstprivate, lastprivate, private, collapse,**

- **#pragma omp distribute parallel for:** Distribuye las diferentes iteraciones del bucle que compone la región paralela entre los diferentes hilos que conforman todos los equipos. Se pueden utilizar las siguientes clausulas:
- **num_threads, if, shared, private, firstprivate, default. Reduction, lastprivate, scheude, nowait, collapsed, reduction.**

4º Listado de clausulas

- **num_threads (“num”):** Número de hilos que ejecutarán la siguiente región paralela. De forma predeterminada se utilizarán tantos hilos como núcleos tenga la máquina.
- **if (“condition”):** La pragma que utiliza dicha clausula se aplicará si la condición indicada se cumple.
- **shared (“varList”):** Asigna las variables indicadas como compartidas a todos los hilos.
- **private (“varList”):** Asigna las variables indicadas como privadas a todos los hilos. Cada uno de los hilos tendrá su propia copia de dicha variable.
- **firstprivate (“varList”):** Asigna el valor original de cada una de las variables indicadas como parámetro, a las respectivas copias privadas que tendrán los hilos.
- **default (none | shared | private | firstprivate):** Indica el tipo por defecto de las variables.
- **reduction (“varList”):** Determina que las variables indicadas, las cuales son privadas, sufrirán una operación de reducción al final de la región paralela.
- **lastprivate (“varList”):** Después de salir de la sección paralela, la última iteración conserva los valores de las variables privadas indicadas como parámetro. Una variable puede ser al mismo tiempo *fristprivate* y *lastprivate*.
- **Schedule ():** Permite determinar la distribución de la ejecución de los hilos dentro de una región paralela. Pueden usarse los siguientes valores:
  - **STATIC [,chunck]:** El planificador distribuye las iteraciones de forma estática entre los hilos y en bloques del mismo tamaño. Se puede definir el tamaño del *chunck* como parámetro.
  - **DYNAMIC [,chunck]:** El planificador distribuye las iteraciones y cuando un hilo termina se le asigna la siguiente. Se puede definir el tamaño del *chunck* como parámetro.
  - **GUIDED [,chunck]:** Semejante a *dynamic* con la diferencia de que el tamaño de los bloques se reduce conforme avanza la ejecución, con el objetivo de distribuir de una forma más equitativa. Se puede definir el tamaño del *chunck* como parámetro.
  - **AUTO:** La planificación sera realizada automáticamente con OpenMP.
  - **RUNTIME:** La decisión de planificación se retarda hasta el bucle de acuerdo con la variable *sched-nvar*.
- **nowait ():** Elimina la barrera de sincronización que se crea por defecto al final del bucle, permitiendo que se puedan solapar ejecuciones independientes del bucle.
- **collapsed ():** Paraleliza varios bucles for anidados como si estuvieran fusionados, de manera que se comporten como un único bucle.
- **reduction (“operator”:“variable”):** Crea una variable privada para cada uno de los hilos, la cual se llama de la manera indicada como parámetro y sobre la que se realizará la operación indicada. Cuando los hilos terminen la ejecución, dicha variables se reducirán obteniendo el valor conjunto. Más óptimo que usar sincronizaciones, ya que no fuerza la serialización.
- **safelen (int):** Número de iteraciones en las que no se rompe la dependencia entre los datos del vector utilizado. Se utiliza cuando los índices del vector se indican mediante operaciones.
- **linear (“intList”):** Indica el ritmo con el que se actualiza la variable indicada con respecto al índice del bucle. Se utiliza para realizar accesos a datos que se encuentran separados un número concreto de posiciones.
- **aligned (“intList”):** Especifica el alineamiento del vector utilizado en la sección vectorial.

- **in_reduction (“operator”:“var”):** Especifica que para cada uno de los argumentos dados a la tarea, estos participarán en un proceso de reducción definido una vez haya finalizado la ejecución de la sección que conforma dicha tarea para todos los hilos que la ejecutan.
- **task_reduction (“operator”:“var”):** Indica que todos los elementos de la lista indicada como parámetro, participarán en una tarea de reducción, al cual se llevará a cabo una vez todas las tareas han terminado su ejecución.
    - Todas las copias asociadas a la reducción se inicializan antes de que cualquiera de ellas acceda a la tarea de reducción. El elemento de la lista original contendrá el resultado de la reducción.
- **depend (“type”:“list”):** Impone restricciones adicionales a programación de una tarea o de las iteraciones de un bucle. Estas restricciones establecen dependencias únicamente entre las ejecuciones de una misma tarea o entre las iteracionm es de un mismo bucle.
- **untied:** Indica que las tareas creadas no estarán asociadas a un hilo de ejecución concreto, lo que permite que se pueda modificar el hilo que ejecuta una tarea en concreto
- **detach (“event”):** Indica que las tareas creadas pueden separarse de su proceso de ejecución sin la necesidad de haberse completado. Se pueden aplicar mecanismos de sincronización para esperar a la ejecución de tareas separadas.
- **device (“int”):** Nos permite especificar el dispositivo que vamos a utilizar y crea el entono de datos en el mismo para poder llevar a cabo la descarga y ejecución del kernel. El valor indicado como parámetro se evaluará como un número entero no negativo, cuyo valor será menor que el proporcionado por la llamada “*omp_get_num_devices()*”.
- **map (“parametro”:”varName”[“tamIni”:“tamFin”]):** Especifica la transferencia de datos entre el host y el dispositivo indicado para ejecutar el kernel. Se pueden utilizar los siguientes parámetro:
    - **to:** Copia los datos indicados del host al dispositivo al iniciar la ejecución del kernel.
    - **from:** Copia los datos del dispositivo al host una vez terminada la ejecución del kernel.
    - **tofrom:** Copia los datos indicados del host al dispositivo al iniciar la ejecución del kernel. Cuando este finaliza vuelve a copiarlos del dispositivo al host.
    - **alloc:** Reserva el espacio de memoria necesario en el dispositivo para los datos indicados, pero sin copiarlos.
    - **delete:** Libera el espacio reservado en el dispositivo para los datos indicados. Dicho espacio se liberará cuando el contador de referencia para dichos datos llegue a 0.
    - **release:** Decrementa el contador de referencias del dato indicado.
- **to (“varList”):** El valor de cada uno de los elementos indicados se copia desde la correspondiente variable almacenada en el host a su contratarte en el dispositivo.
- **from (“varList”):** El valor de cada uno de los elementos indicados se copia desde la correspondiente variable almacenada en el dispositivo a su contratarte en el host.
- **num_teams (“int”):** Especifica el límite superior del número de equipos que se crearán.
- **thread_limit (“int”):** Especifica el límite superior del número de hilos que conformarán cada uno de los diferentes equipos creados.