Perfecto, Don Daniel. ¡Vargas Cipollo!

Voy a redactarte unos apuntes completos, fusionando explicaciones claras con estructura tipo resumen más tabla de preguntas clave, todo basado únicamente en el contenido de los tests tipo examen que has subido. Así tendrás todo lo necesario para repasar de forma autónoma y sacar un 10.

Te aviso en cuanto los tenga listos.

# Apuntes de OpenMP y paralelismo (Arquitectura de Computadores BP3)

## Conceptos básicos de hebras y OpenMP

- **Hebras (threads)**: Son los flujos de ejecución ligeros creados en un programa paralelo. OpenMP facilita la programación multinúcleo con directivas (`#pragma omp`) que permiten dividir el trabajo entre hebras.
- **Region paralela**: `#pragma omp parallel` indica que el bloque de código se ejecutará por varias hebras en paralelo. Dentro de una región paralela, cada hebra ejecuta el mismo código de forma concurrente.

## Establecer el número de hebras

- **Variables de entorno**: `OMP_NUM_THREADS` fija el número de hebras que usarán las regiones paralelas por defecto. Por ejemplo, si `OMP_NUM_THREADS=8`, el primer `#pragma omp parallel` puede lanzar hasta 8 hebras.

- **Funciones de la API**: La llamada `omp_set_num_threads(n)` fija el número de hebras para la próxima región paralela. Esta llamada prevalece sobre la variable de entorno.

- **Cláusula `num_threads`**: Se puede especificar directamente en la directiva:

  ```
  omp_set_num_threads(4);
  #pragma omp parallel num_threads(2)
  printf("hello\n");
  ```

  En este ejemplo, aunque se fijó 4 con `omp_set_num_threads` y `OMP_NUM_THREADS=8`, la cláusula `num_threads(2)` tiene **mayor prioridad**, por lo que se crearán 2 hebras. En general la prioridad es: cláusula `num_threads` > `omp_set_num_threads()` > `OMP_NUM_THREADS`.

## Distribución de iteraciones en bucles (`schedule`)

- **`schedule(static)` (reparto estático)**: Divide las iteraciones del bucle en *chunks* (bloques) iguales entre las hebras de forma fija. Por defecto (sin especificar *chunk*), OpenMP reparte las iteraciones de forma equitativa. Por ejemplo:

```
#pragma omp parallel for schedule(static)
for(int i = 0; i < N; i++) {
    // código de cada iteración
    v3[i] = v1[i] + v2[i];
}
```

En este caso, si hay 4 hebras y N=10000, cada hebra obtiene ~2500 iteraciones fijas. El reparto es **estático** (igualitario).

- `schedule(dynamic[, chunk])` **(reparto dinámico)**: Las iteraciones se asignan de forma dinámica: cada hebra toma un *chunk* de iteraciones, y cuando termina toma otro hasta agotarlas. Si no se especifica `chunk`, el tamaño por defecto es 1. En planificación *dynamic*, el tamaño del *chunk* es **constante** a lo largo de la ejecución. (La afirmación de que el tamaño decrece corresponde a `schedule(guided)`, no a `dynamic`.)

- `schedule(guided[, chunk])` **(planificación guiada)**: Similar a *dynamic* pero los *chunks* van decreciendo: comienza con bloques grandes y luego bloques cada vez más pequeños. Esto intenta balancear la carga cuando algunas iteraciones tardan más que otras.

- `schedule(runtime)`: Indica que el tipo de planificación y *chunk* se decidirá en tiempo de ejecución, según la variable de entorno `OMP_SCHEDULE` o llamadas a funciones. Por ejemplo:

  ```
  #pragma omp for schedule(runtime)
  for(int i = 0; i < N; i++) { /*...*/ }
  ```

  En este caso, la planificación se puede cambiar sin recompilar el código. Para modificarla: se puede usar la variable de entorno `OMP_SCHEDULE` (por ejemplo `OMP_SCHEDULE="dynamic,4"`) o la función `omp_set_schedule(kind, chunk)` (por ejemplo `omp_set_schedule(omp_sched_static, 10)`) antes de la región paralela. Ambas formas son válidas (variable de entorno **o** función), y se aplican cuando se usa `schedule(runtime)`.

## Variables de entorno y de control internas de OpenMP

- **Variables de entorno comunes**:
  - `OMP_NUM_THREADS`: número máximo de hebras por defecto.
  - `OMP_SCHEDULE`: define el tipo de planificación por defecto y el tamaño de *chunk* para `schedule(runtime)`. Formato: `"tipo,chunk"` (por ejemplo `"static,5"` o `"dynamic,1"`).
  - `OMP_THREAD_LIMIT`: límite superior de hebras que puede usar el programa (por ejemplo, con varios niveles de paralelismo anidado).
  - `OMP_DYNAMIC`: si está en `true/1`, permite a OpenMP ajustar dinámicamente el número de hebras en paralelo (útil para evitar exceso de hebras).

- **Variables de control internas** (por ejemplo `run-sched-var`, `def-sched-var`): Son variables que OpenMP utiliza internamente para determinar la planificación del bucle. No se acceden directamente desde el código del usuario. Sólo se pueden cambiar mediante **API** de OpenMP o variables de entorno adecuadas, pero generalmente no se accede directamente a ellas desde el programa. En particular, *no*

se manipulan vía llamadas C directas; en su lugar se usan las funciones estándar de OpenMP o las variables de entorno específicas.

- **Funciones de consulta**:
  - `omp_get_thread_num()`: devuelve el ID (0..n-1) de la hebra actual dentro de la región paralela.
  - `omp_get_num_threads()`: devuelve el número total de hebras en la región paralela actual.
  - `omp_get_max_threads()`: número máximo de hebras que se usarán sin cláusula explícita.
  - `omp_get_schedule(int *kind, int *chunk)`: obtiene el tipo de planificación (`kind`: estático, dinámico, etc.) y tamaño de *chunk* activos cuando se usa `schedule(runtime)`. No modifica nada (por eso en el código de ejemplo sólo se usa `get_schedule`, no `set`).
  - `omp_set_schedule(omp_sched_kind kind, int chunk)`: fija la planificación y tamaño de *chunk* para directivas con `schedule(runtime)` a partir de ese momento.

## Ejemplo de código: multiplicar matriz triangular por vector

Para multiplicar una **matriz triangular inferior** `M[N][N]` por un vector `v[N]` y acumular en `r[N]`, lo más eficiente es solo iterar sobre los elementos no nulos (parte triangular). Un código correcto sería:

```
int m[N][N], v[N], r[N] = {0};
for(int i = 0; i < N; i++) {
    for(int j = 0; j <= i; j++) {      // solo j ≤ i (triangular inferior)
        r[i] += m[i][j] * v[j];
    }
}
```

Este bucle anidado recorre únicamente la mitad inferior de la matriz (incluyendo la diagonal), evitando multiplicaciones con ceros. (Otras opciones que recorren la matriz completa serían incorrectas e ineficientes).

## Cláusula `if` en paralelismo

La cláusula `if(condición)` en una región paralela controla si la región se ejecuta en paralelo o en serie según la condición. Por ejemplo:

```
#pragma omp parallel if(n > 20)
{ /*...*/ }
```

Significa que sólo se ejecuta en paralelo si `n>20`. En caso contrario (`n<=20`), el bloque se ejecuta con una sola hebra (serie). Esto es útil para **evitar la sobrecarga** de crear hebras cuando el problema es muy pequeño. Así, si `n` es pequeño, al no paralelizar se ahorra tiempo de sincronización.

## Eficiencia y escalabilidad

- **Eficiencia (speedup)**: Es la mejora de velocidad al ejecutar en paralelo respecto a serie. Se mide por *speedup* (tiempo serie / tiempo paralelo). Un *speedup* ideal con `p` hebras es igual a `p`, pero en la práctica suele ser menor por sobrecarga de paralelismo.

- **Escalabilidad**: Capacidad de un programa paralelo para obtener mayor rendimiento al aumentar el número de hebras o núcleos. Un **estudio de escalabilidad** típico representa la *ganancia en velocidad* (speedup) en función del número de hebras/núcleos. Esto indica cuánto mejora (o no) el rendimiento al añadir más recursos. Una gráfica común de escalabilidad fuerte es: en el eje Y el *speedup* (o la reducción de tiempo) y en el eje X el número de hebras. Idealmente la curva es lineal (speedup = núcleos), pero en la práctica se aplanará por costes de comunicación y desequilibrios.
- **Balance de carga**: Un factor importante para la eficiencia. Planificaciones estáticas equilibran bien cuando cada iteración tarda similar tiempo. Dinámicas o guiadas son útiles si algunas iteraciones tardan más (ya que reasignan trabajo según disponibilidad).

## Preguntas de autoevaluación y respuestas

| **Pregunta representativa** | **Respuesta correcta** |
|---|---|
| **¿Qué tipo de reparto de iteraciones hace** `schedule(static)` en `#pragma omp for`? | Estático (chunk fijo por defecto). |
| **¿Cómo modificar el reparto si** `schedule(runtime)`? | Usando la variable de entorno `OMP_SCHEDULE` o `omp_set_schedule()`. |
| **¿Cuántas hebras se ejecutan si** `OMP_NUM_THREADS=8`; luego `omp_set_num_threads(4);` y `#pragma omp parallel num_threads(2)`? | 2 (la cláusula `num_threads` tiene prioridad). |
| **¿Qué muestra una gráfica de escalabilidad?** | La ganancia en velocidad (*speedup*) vs número de núcleos. |
| **¿Cuál es el tamaño de chunk en** `schedule(dynamic)` **sin especificar?** | Siempre es constante (por defecto 1). |
| **¿Qué código es correcto para multiplicar matriz triangular inferior por vector?** | El bucle con `for(i=0..N) for(j=0..i) r[i] += m[i][j]*v[j];` (option d del test). |

Estos puntos resumen los conceptos clave que aparecen en los exámenes tipo test de OpenMP y ayudaran a prepararse para responder correctamente las preguntas de la asignatura.