Índice

1. Vectorización

1.1. Tipos de vectorización

Tipos de vectorización:

- Intrínseca (Uso de estructuras específicas de datos)
- Automática (Compilador vectoriza)
- Guiada (Uso de sintaxis especial en el código)

1.1.1. Vectorización intrínsecas

- SSE (128b)
- AVX (256b)
- AVX2 (512b)

Formato de las instrucciones intrínsecas

```
_mm_instruction_suffix(...)
// Ejemplos
// aligned: Los datos tienen que estar alineados a la linea de cache
// Requiere un solo acceso para traer los datos
_mm_load_ps(float const* mem_addr);
// unaligned: Los datos pueden estar en bloques diferentes
// Requiere dos accesos a memoria
_mm_loadu_ps(float const* mem_addr);
```

```
#include <xmmintrin.h>
int main () {
...
float a[4]={1.0,2.0,3.0,4.0};//a must be 16-byte aligned
__m128 x = _mm_load_ps(a);
__m128 a, b;
__m128 c = _mm_add_ps(a, b);
...
}
```

1.1.2. Vectorización automática

El compilador realiza automáticamente la vectorización en tiempo de compilación, aunque tiene **limitaciones**:

- Es solo válido en bucles **internos**
- Deben conocerse previamente el número de iteraciones
- No ha de haber dependencias entre iteraciones
- Se debe especificar aquellas funciones que están vectorizadas

1.1.3. Vectorización guiada

```
// Fuerza la vectorización de bucles
// Permite bucles con funciones explicitamente definidas como SIMD
// Para bucles internos
#pragma omp simd

#pragma omp declare simd
float my_simple_add(float x1, float x2){
  return x1 + x2;
}
...
// May be in a separate file
#pragma omp simd
for (int i = 0; i < N, ++i) {
output[i] = my_simple_add(inputa[i], inputb[i]);
}

// Informa al compilador que no tiene dependencias
#pragma ivdep
```

1.1.3.1 Alineamiento de datos

```
  float buf_static[1000] __attribute__((aligned(64)));
  float *buf_dynamic = (float*) _mm_malloc(buffer_size, 64);
```

```
// Para informar de que el vector está alineado
# pragma vector aligned
```

1.1.3.2 Otros pragmas

```
#pragma loop count(n)
#pragma vector always
#pragma novector
```

2. Paralelización basada en memoria compartida

```
void main ( ) {
  #pragma omp parallel
  ... /* An unkoum number of threads here. Use OMP_NUM_THREADS */
  omp_set_num_threads(2) ;
  #pragma omp parallel
  ... /* A team of two threads here */
  #pragma omp parallel num_threads(random() %4+1) if(0)
  ... /* A team of 1 thread here */
}
```

2.1. Compartición de datos

Tipo de clausulas

2.2. Trabajo compartido

3. Paralelización basada en memoria distribuida

4. Computación heterogénea