Lab 2

Parallel Programming
2025/10/2

Lab 2-1
Approximate pixels

Parallel Programming
2025/10/2

Approximate pixels

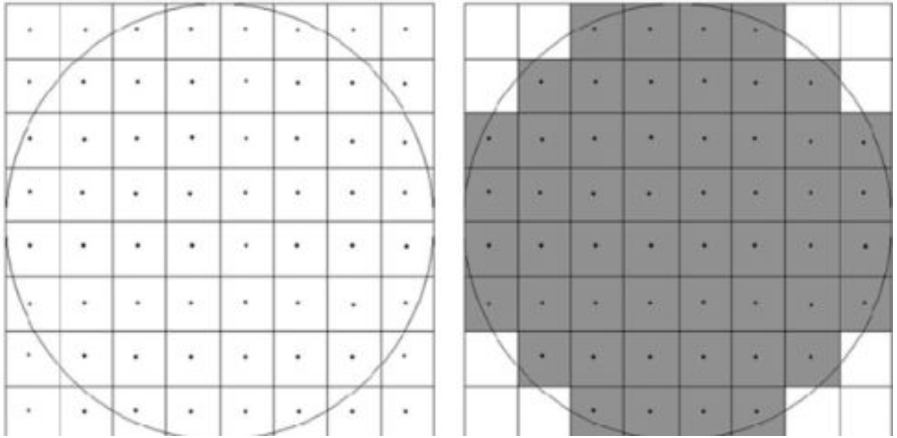
Copy source files to your home directory on origo apollo:

cp -r /home/pp25/share/lab2 ~/

Suppose we want to draw a filled circle of radius *r* on a 2D monitor, how many pixels will be filled?

We fill a pixel when any part of the circle overlaps with the pixel. We also assume that the circle center is at the boundary of 4 pixels.

For example, 88 pixels are filled when r = 5.



其實就是LAB1

Linking Pthread & Openmp

Linking pthread

gcc your_program.c -o your_program -pthread

Linking openmp

gcc your_program.c -o your_program -fopenmp

Linking both

gcc your_program.c -o your_program -pthread -fopenmp

Compile and Run

- lab2_pthread.cc
  - Compile / Run
    - g++ lab2_pthread.cc -o lab2_pthread -pthread -lm
    - srun -c4 -n1 ./lab2_pthread r k

-c4 # 4 CPU cores per process
-n1 # 1 process

- lab2_openmp.cc
  - Compile / Run
    - g++ lab2_openmp.cc -o lab2_openmp -fopenmp -lm
    - srun -c4 -n1 ./lab2_openmp r k
  - You can start with different scheduling choice and threads number setup

Approximate pixels using Hybrid MPI with OpenMP

- Modify the sequential code lab2_hybrid.cc with MPI and OpenMP
  - mpicxx lab2_hybrid.cc -o lab2_hybrid -fopenmp -lm
  - srun -n6 -c4 ./lab2_hybrid r k

Judge

You may verify your code by using our judge:

lab2_pthread

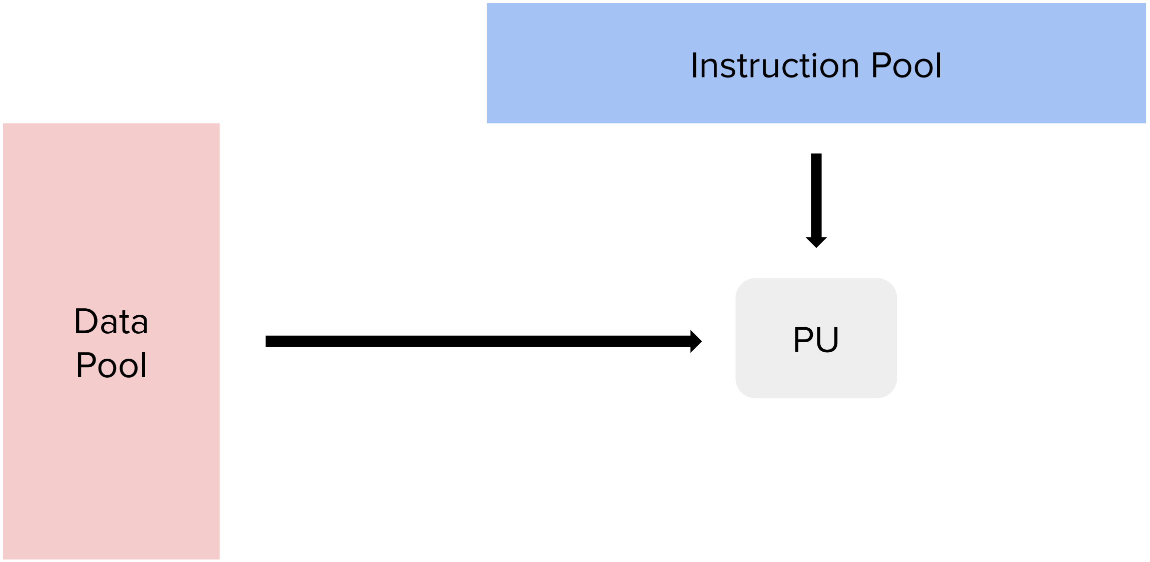
lab2_omp

lab2_hybrid

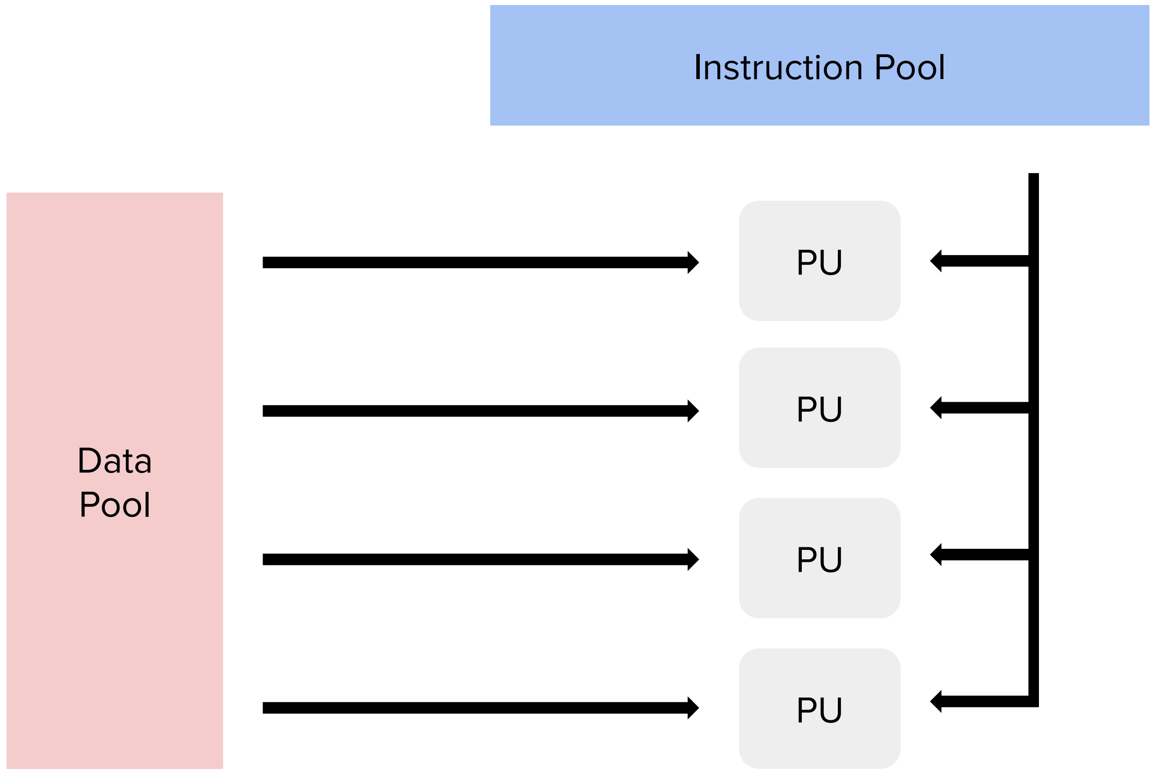
Lab 2-2
Vectorization

Parallel Programming
2025/10/2

SISD



SIMD



Automatic Vectorization

GCC Vectorization

- -ftree-vectorize : enabled vectorization
- -O3 : enabled vectorization by default
- -march=native : use instructions supported by the CPU
- -fopt-info-vec-all : print vectorization log
- #pragma GCC ivdep : tells compiler there is no data dependency in the following loop

Vector Instruction Set

lscpu can be used to display the vector instruction sets supported by the CPU

On Origo Apollo, sse/sse2 and more are available.

```
Flags:                  fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ht tm pbe syscall nx pdp
                        e1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor
                         ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm pcid dca sse4_1 sse4_2 popcnt aes lahf_lm epb pti ssbd ibrs ibpb stibp tpr_shadow vnmi
                         flexpriority ept vpid dtherm ida arat flush_l1d
```

Vector Instruction Set

SSE（Streaming SIMD Extensions）

- ❑ 128-bit registers, doubling the width of the 64-bit MMX registers
- ❑ SSE only supports 32-bit floating point
- ❑ SSE2 adds double, long long, int, char

Vector Instruction Set

AVX（Advanced Vector Extensions）

- ❑ Expanded SIMD registers from 128 bits (in SSE) to 256 bits
- ❑ Supports both single-precision and double-precision floating-point operations
- ❑ AVX512 expands SIMD registers to 512 bits

Vector Instruction Set

1. Codes have to be executed many times, will probably benefit from vectorization

2. If there are no data dependency, it will be easier to vectorize

Data dependency means the value of one data elements depends on another

Intel Intrinsics

Allows developers to use advanced instruction sets of processors directly in C/C++

Procedure :

1. Load data from memory to the special registers
2. Perform vector instructions
3. Save data from the special registers to memory

Intel Intrinsics

```
void multiple_and_add(float *a, float *b, float *c, float *d, int size){
    for(int i = 0; i < size; i++){
        a[i] = b[i] * c[i] + d[i];
    }
}
```

Intel Intrinsics

```
void vec_multiple_and_add(float *a, float *b, float *c, float *d, int size){

    int i;

    for(i = 0; i < size - 15; i += 16){
        // load data to special registers
        __m512 b_vec = _mm512_loadu_ps(&b[i]);
        __m512 c_vec = _mm512_loadu_ps(&c[i]);
        __m512 d_vec = _mm512_loadu_ps(&d[i]);

        // _mm512_fmadd_ps finish the multiplae and add operation
        // _mm512_storeu_ps store the result to a array
        _mm512_storeu_ps(&a[i], _mm512_fmadd_ps(b_vec, c_vec, d_vec));
    }

    // remaining elements
    for(; i < size; i++){
        a[i] = b[i] * c[i] + d[i];
    }
}
```

AVX512 can handle 512 bits = 16 * 32-bits floating point

Load data from memory to special registers, using intel intrinsics

Perform vector instructions and store back to memory

HW2

Parallel Programming
2025/10/2

SPEC

Hint: You can use intel intrinsics in your homework.