Abdelrahman Adel Abdelfattah Sheet 4 1701296 NOTES

① Performance of Convolution :-

- CPU with a single core at 2.5 GHz that can perform eight single precision floating-point operations per clock cycle.
- CPU is connected to a DRAM with a peak memory transfer rate at 25.6 GB/s
- Cache size of 256 KB as fast as registers
- Convolution of 2D image I of n×n size in main memory with a mask M of size (2k+1) × (2k+1)
- Compute an image C of n×n

$$C[x][y] = \sum_{i=-k}^{k} \sum_{j=-k}^{k} I[x+i][y+j] \times M[k-i][k-j]$$

for all $0 \le x, y \le n-1$

$I[x+i][y+j] = 0$ if $x+i < 0$ or $x+i \ge n$
$y+j < 0$ or $y+j \ge n$

```
for x = 0 to n
  for y = 0 to n
    for i = -k to k
      for j = -k to k
        if (x+i) ≥ n or (x+i) < n or (y+j) < 0 or (y+j) ≥ n
          I_cell = 0
        else
          I_cell = I[x+i][y+j]
        C[x][y] += I_Cell × M[k-i][k-j]
```

- CPU: ~~3 GHz~~ 2.5 GHZ
  8 single precision Point per cycle

FLOPS = 8 × 1 × 2.5 GHZ = 20 × 10⁹ FLOPS

- CPU ⇒ 20 GFLOP/s
- DRAM ⇒ 25.6 GB/s
- Cache ⇒ 256 KB
- size of C n×n ⇒ 8B × 256 × 256 = 512 KB size of C
- 2 Flops per loop × loops = n × n × 2k × 2k
  = 4k²n² loops

(i) loops = 4k²n²
= 4 × 1² × 256² = 256K Loops

- Total Flops = 256K × 2 = 512 KFLops
- CPU computation = 512 KFlops / 20 GFLops = 24.4 × 10⁻⁶ seconds
  = 24.4 μs
- Storage = n × n × 8 + k × K × 8
  = 524296 Bytes
  ≃ 512 KB
- 2 cache cycles = 2 × 9.54 μs
  = 19.08 μs

2)

$t_{exec} \geq 24.4\,\mu s + 19.1\,\mu s = 43.5\,\mu s$

$\frac{512k}{43.5\,\mu s} = 11.226\ GFlop/s$, Performance = 56.124%.

(ii) $loops = 4k^2n^2 = 6.25\,M\,loops$

$Flops = 12.5\,M\,FLops$

$Storage = 512.2\,kB$

3 cache cycle $= 9.54 \times 3 = 28.62\,\mu s$

$t_{computer} = \frac{12.5\,M}{20\,G} = 610.35\,\mu s$

$t_{exec} \geq 610.35 + 28.62 \cong 639\,\mu s$

$\frac{12.5\,M}{639\,\mu s} = 19.10335 \cong 19.1\ GFLOP/s$

Performance = 95%

(iii) using both Spacial & Temporal Locality

#

② Cache line $L = \text{size of}(\text{float}) * 16$

NOTES.

$A, B, N \times N$. $N = 1024$

$C_{ij} = A_{ij} + B_{ij}$ for all i, j

a) Calculate misses of cache

1 miss each new row

Total Misses (1024)

b) 1 miss each cell

Totall misses (1024×1024)

$= 1,048,576$ misses

row wise would be faster less cache misses

```cpp
#include <stdio.h>
#include <iostream>
#include <math.h>
#include <pthread.h>
#define N 1024
uint64_t A[N][N];
uint64_t B[N][N];
uint64_t C[N][N];

uint64_t n = 1024;
uint64_t X[1024];


using namespace std;

double Riemann_Zeta(double s, uint64_t k){
    double result = 0.0;
    for (uint64_t i = 1; i < k; i++){
        for (uint64_t j = 1; j < k; j++){
            result += (2 *(i&1)-1/pow(i+j, s));
        }
    }
    return result*pow(2, s);
}
void *worker_thread_example(void *arg){
    uint64_t k = (uint64_t)arg;
    X[k] = Riemann_Zeta(2,k);
    pthread_exit(NULL);
}
void example(){
    pthread_t threads[n];
    uint64_t Y[1024];

    for(uint64_t k = 0; k < n; k++){
        int ret = pthread_create(&threads[k], NULL, &worker_thread_example, (void*)k);
        if(ret != 0){
            printf("Error: pthread_create() failed\n");
            exit(EXIT_FAILURE);
        }
    }
    for (int i =0; i < n; i++){
        pthread_join(threads[i], NULL);
    }
    for(uint64_t k = 0; k < n; k++){
        Y[k] = Riemann_Zeta(2, k);
    }
    for(uint64_t k = 0; k < n; k++){
        cout << X[k] << "-" << Y[k] << "=" << X[k]-Y[k] << endl;
    }
}


void row_sum(uint64_t row){
    for(uint64_t i = 0; i < N; i++){
        C[row][i] = A[row][i] + B[row][i];
    }
}
void *worker_thread(void *arg){
    uint64_t row = (uint64_t)arg;
    row_sum(row);
    pthread_exit(NULL);
```

```
    }
}

void row_sum(uint64_t row){
    for(uint64_t i = 0; i < N; i++){
        C[row][i] = A[row][i] + B[row][i];
    }
}
void *worker_thread(void *arg){
    uint64_t row = (uint64_t)arg;
    row_sum(row);
    pthread_exit(NULL);
}
void matrix_addition(){
    pthread_t threads[N];

//  Init A and B
    for (int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++){
            A[i][j] = i*N + j;
            B[i][j] = i*N + j;
        }
    }
    for(uint64_t i = 0; i < N; i++){
        int ret = pthread_create(&threads[i], NULL, &worker_thread, (void *)i);
        if(ret != 0){
            printf("Error: pthread_create() failed\n");
            exit(EXIT_FAILURE);
        }
    }
    for (int i =0; i < N; i++){
        pthread_join(threads[i], NULL);
    }
}
int main(){
    matrix_addition();
    return 0;
}
~
```

main.cpp 52,0-1 Bot