Convolution

Team 9
113062573 余伽璇
112078502 楊婷婷

Outline

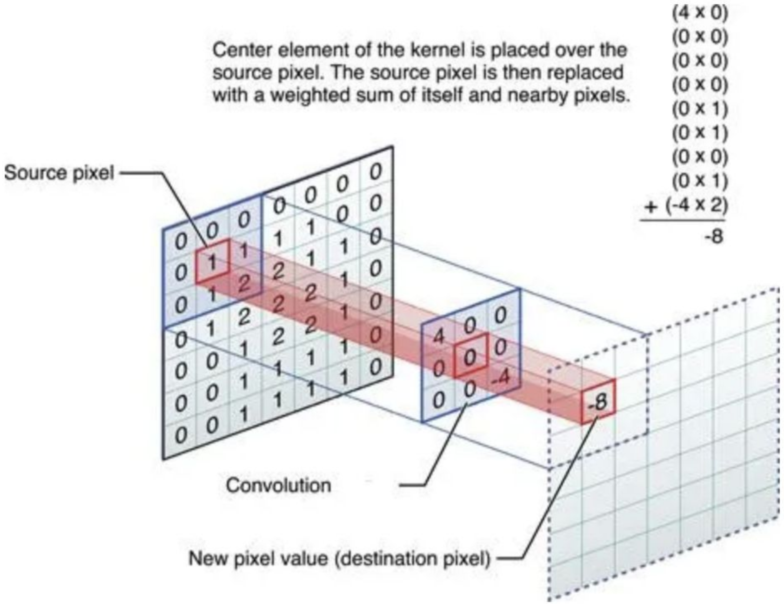
- Problem formulation
- Implementation
- Expirement
- Future work

Outline

- Problem formulation
- Implementation
- Expirement
- Future work

Convolution and Applications

Convolution is a fundamental operation in image processing, deep learning or signal processing etc. It involves the element-wise multiplication of two functions, one typically being a signal or image, and the other a kernel or filter. This operation captures local relationships and patterns, making it a powerful tool for feature extraction, system analysis.



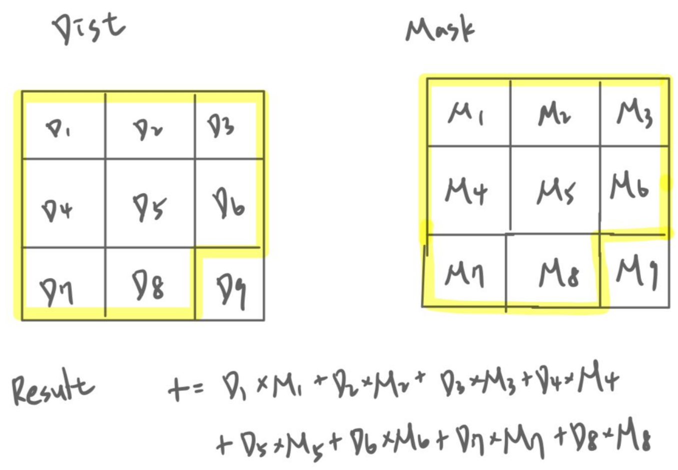
Convolution Operation on a 7×7 matrix with a 3×3 kernel

Outline

- Problem formulation
- Implementation
  - sequential with vectorization
  - OpenMP
  - Pthread
  - MPI
  - Hiybrid
  - single-GPU (regular version: global v.s shared)
- Expirement
- Future work

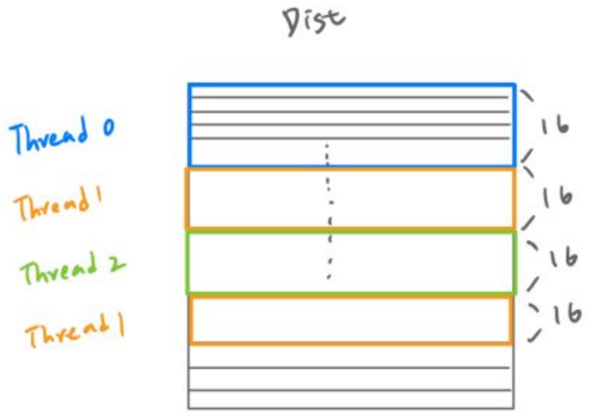
Implementation - Vectorization

- version 1
  - 256 bits-registers in AVX
  - 256 / 32 = 8, 8 floats can be computed simultaneously
- version 2
  - Same as verion 1, but using two __m256 registers
  - 16 floats can be computed simultaneously

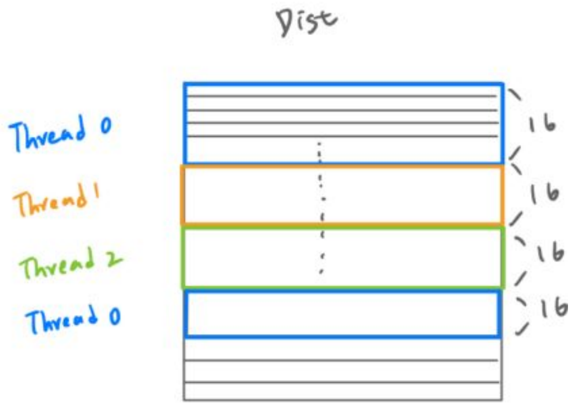


Implementation - OpenMP

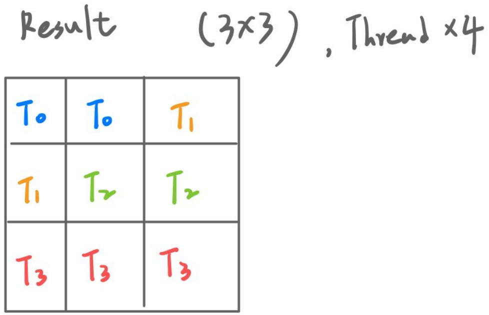
- Dynamic Scheduling
  - #pragma omp parallel for schedule(dynamic, 16)
  - height partitioning



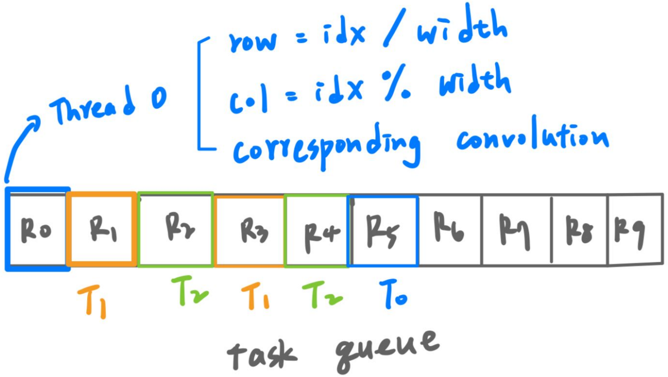
- Static Scheduling
  - #pragma omp parallel for schedule(static, 16)
  - height partitioning



Implementation - Pthread

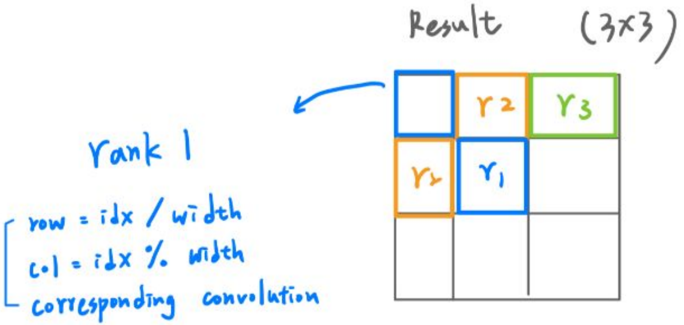


- Static Scheduling
  - divides the total number of elements in the output matrix equally among threads, with each thread handling a contiguous range of elements based on its thread ID
- Dynamic Scheduling
  - master thread
  - task_queue : store #(result matrix size) tasks and one termination signal (-1) at the end
  - mutex lock



Implementation - MPI

- master process : rank 0
- send -1 to work process if all tasks are done
  - Ex : 4 process, input = 5*5, kernel = 3*3, result = 3*3



- How the master process collects result data ?

```
MPI_Recv(&task_index, 1, MPI_INT, MPI_ANY_SOURCE, 1, MPI_COMM_WORLD, &status);
MPI_Recv(&result, 1, MPI_FLOAT, status.MPI_SOURCE, 2, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

Result[task_index] = result;
```

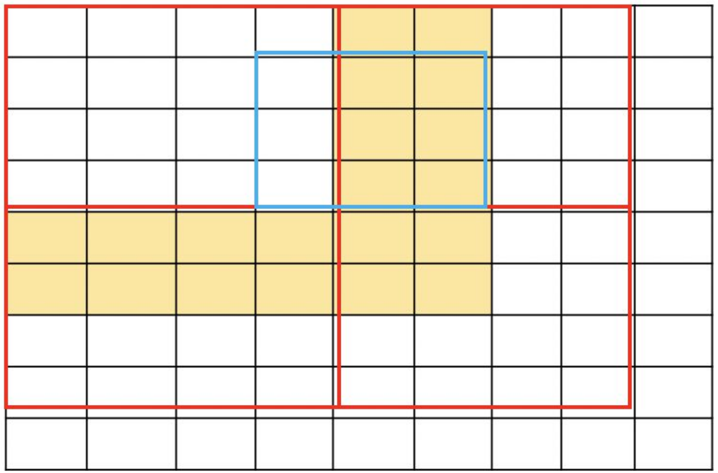
Implementation - Hybrid

- Task partition and process communication same as MPI
- Convolution of each work process use :

```
#pragma omp simd reduction(+:sum)
for (ki = 0; ki < k; ++ki)
{
    for (kj = 0; kj < k; ++kj)
    {
        sum += Dist[(row + ki) * width + (col + kj)] * Mask[ki * k + kj];
    }
}
```

CUDA - Regular Convolution

- Global Memory:
  all compute fetches from global, no boundary issue
- Share Memory :
  - need apron to load additional data to the thread block
  - e.g when threadblock = 4*4 , kernel = 3*3 at block (0,0) need the data from the other threadblocks
    → need additional data in share mem



Outline

- Problem formulation
- Implementation
- Expirement
- Future work

Experiment - Set up

- Environment of CPU

```
pp24s111@nthu-master:~/final$ lscpu
Architecture:             x86_64
  CPU op-mode(s):         32-bit, 64-bit
  Address sizes:          52 bits physical, 57 bits virtual
  Byte Order:             Little Endian
CPU(s):                   192
  On-line CPU(s) list:    0-191
Vendor ID:                GenuineIntel
  Model name:             INTEL(R) XEON(R) PLATINUM 8568Y+
    CPU family:           6
    Model:                207
    Thread(s) per core:   2
    Core(s) per socket:   48
    Socket(s):            2
    Stepping:             2
    BogoMIPS:             4600.00
    Flags:                fpu vme de pse tsc msr pae mce cx8 a
                          nonstop_tsc cpuid aperfmperf tsc_kno
                          d lahf_lm abm 3dnowprefetch cpuid_fa
                          rms invpcid cqm rdt_a avx512f avx512
                          ni avx512_bf16 wbnoinvd dtherm ida a
                          ovdir64b enqcmd fsrm md_clear serial
Virtualization features:
  Virtualization:         VT-x
Caches (sum of all):
  L1d:                    4.5 MiB (96 instances)
  L1i:                    3 MiB (96 instances)
  L2:                     192 MiB (96 instances)
  L3:                     600 MiB (2 instances)
NUMA:
  NUMA node(s):           2
  NUMA node0 CPU(s):      0-47,96-143
  NUMA node1 CPU(s):      48-95,144-191
Vulnerabilities:
  Gather data sampling:   Not affected
```

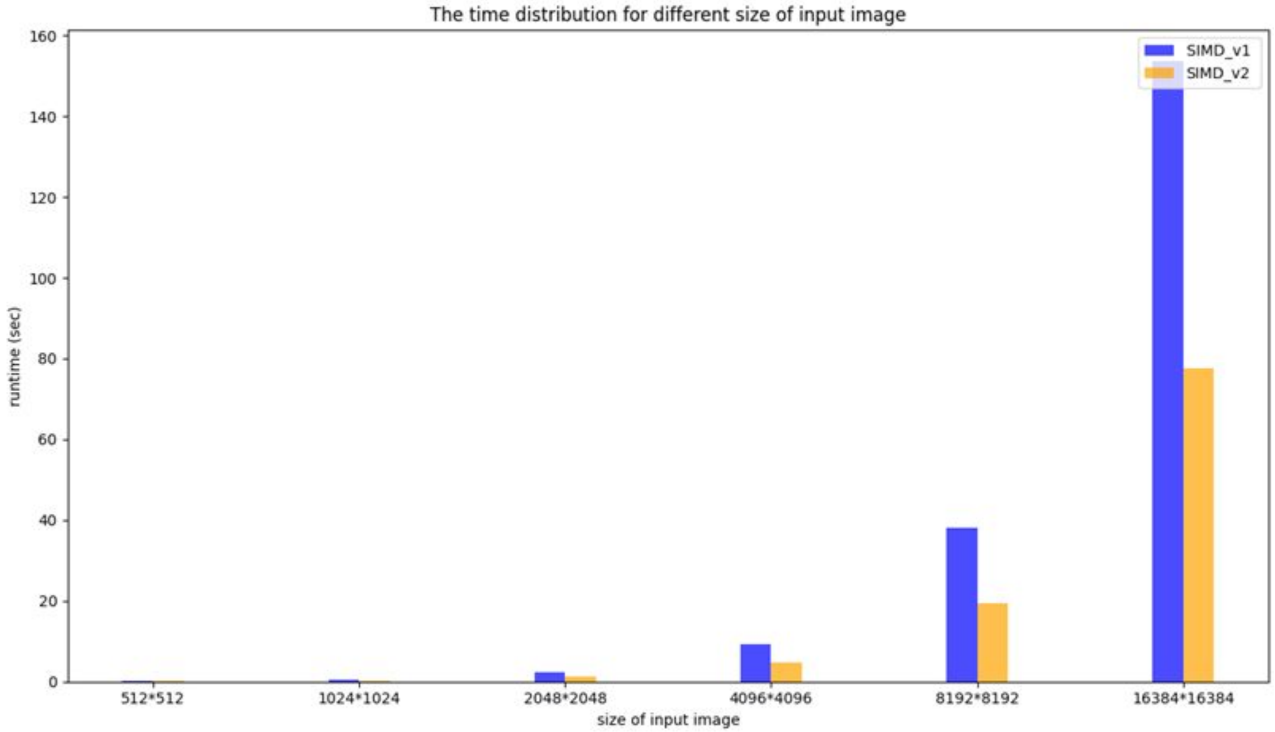
- Environment of GPU

```
Device 0: "NVIDIA GeForce GTX 1080"
  CUDA Driver Version / Runtime Version          12.6 / 12.6
  CUDA Capability Major/Minor version number:    6.1
  Total amount of global memory:                 8107 MBytes (8500871168 bytes)
  (20) Multiprocessors, (128) CUDA Cores/MP:     2560 CUDA Cores
  GPU Max Clock rate:                            1835 MHz (1.84 GHz)
  Memory Clock rate:                             5005 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 2097152 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 0 / 1
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 12.6, CUDA Runtime Version = 12.6, NumDevs = 1
Result = PASS
```

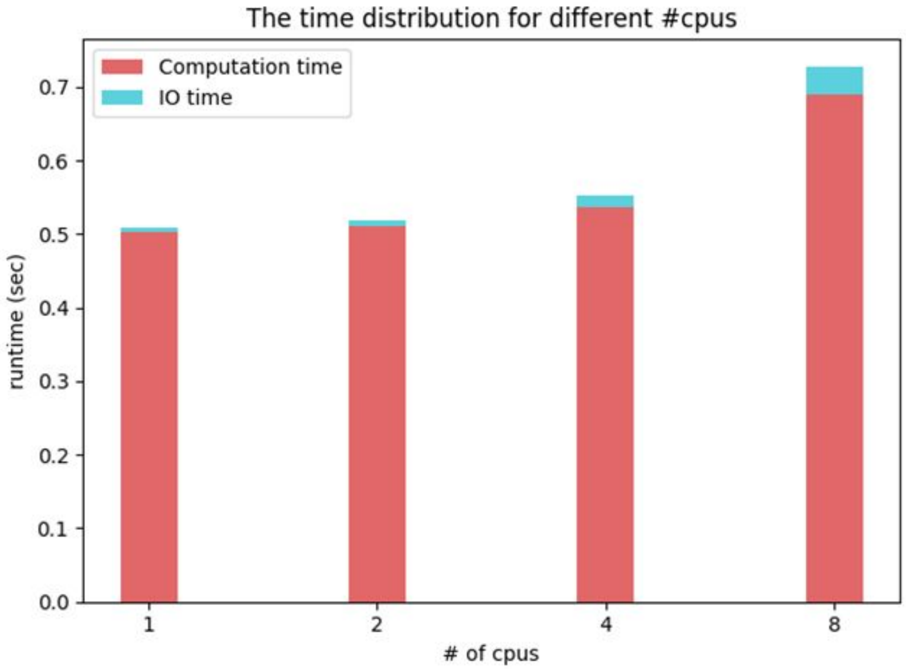
Experiment - Vectorization

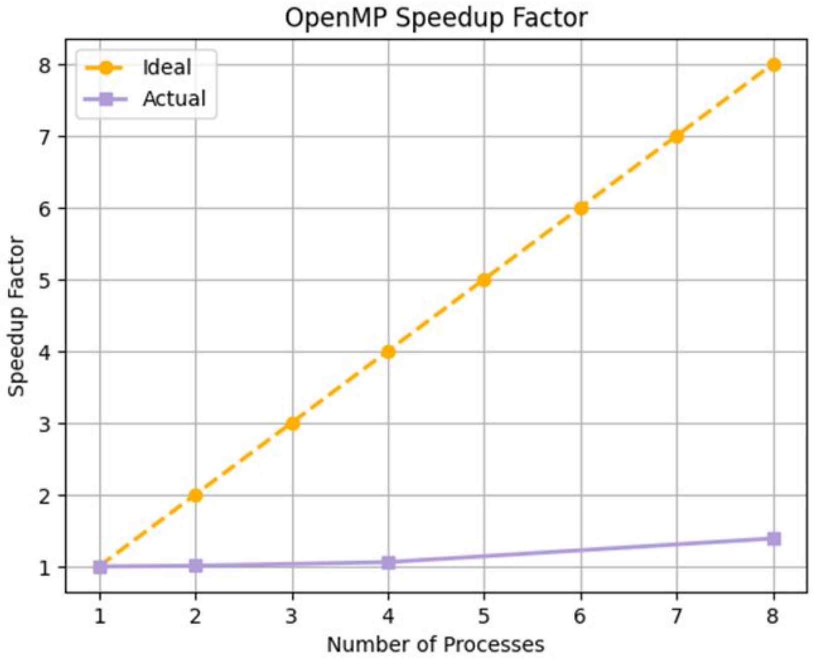
- Sequential version with SIMD-v1 vs SIMD-v2
  - kernel size : 96*96



Experiment - OpenMP

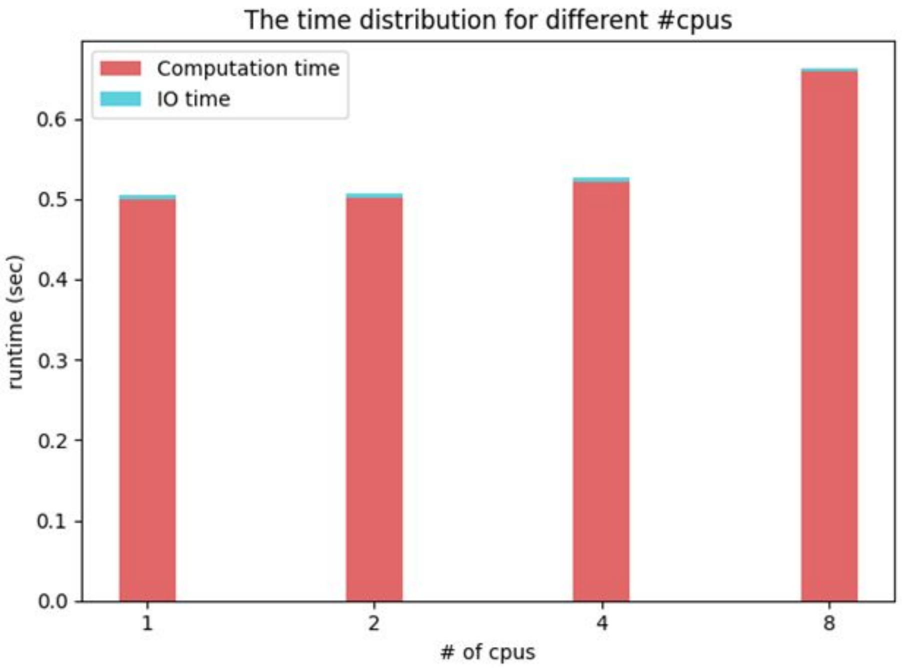
- static scheduling with SIMD-v1
  - testcase : input size = 1024*1024, kernel size = 96*96

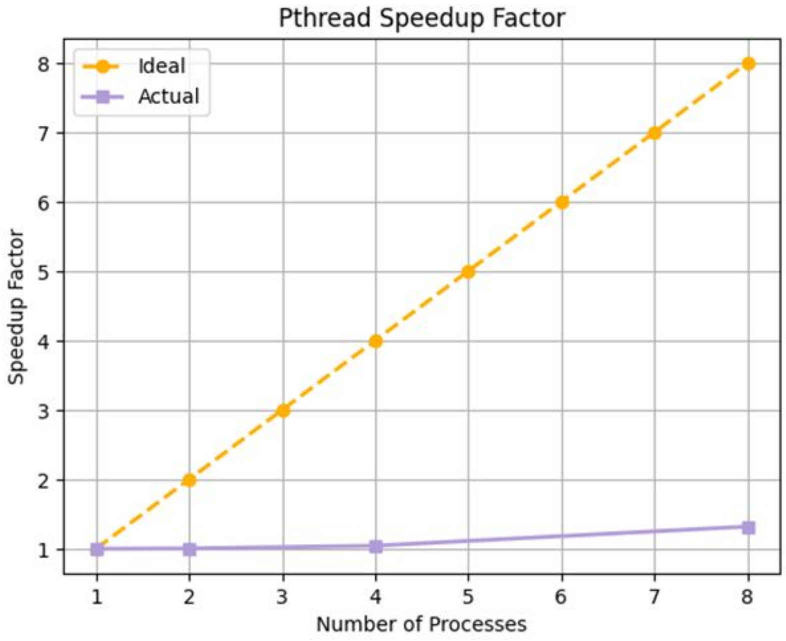




Experiment - Pthread

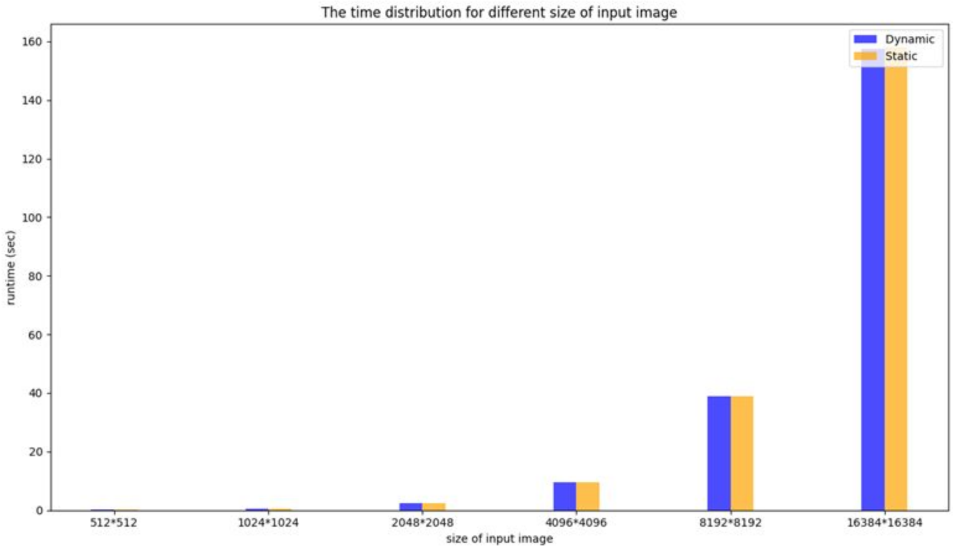
- static scheduling with SIMD-v1
    - testcase : input size = 1024*1024, kernel size = 96*96



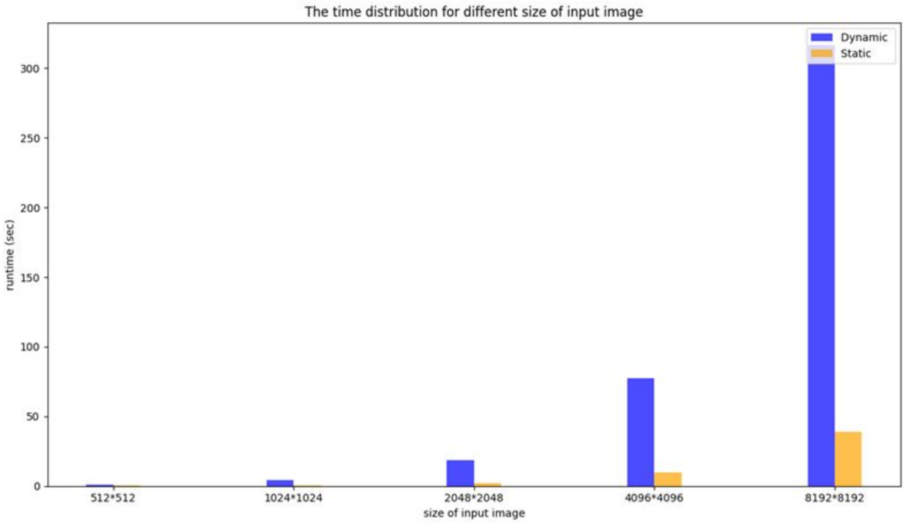


Experiment - OpenMP & Pthread

- Scheduling method : dynamic , static
  - kernel size = 96*96, 4 cores



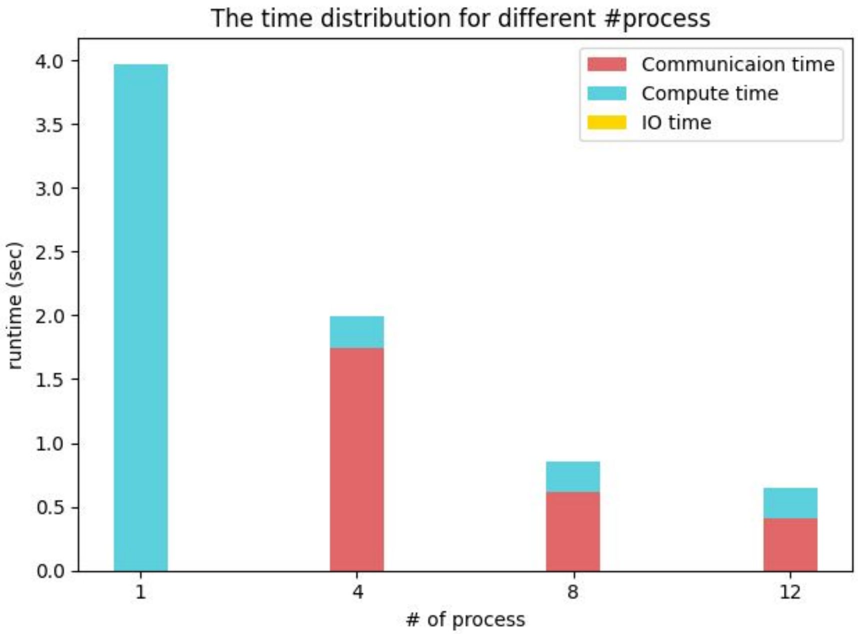
OpenMP

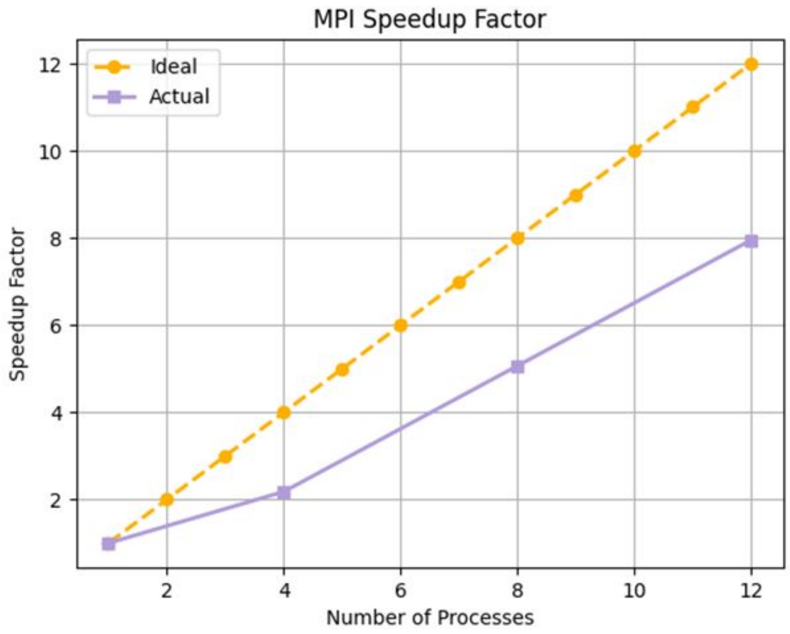


Pthread

Experiment - MPI

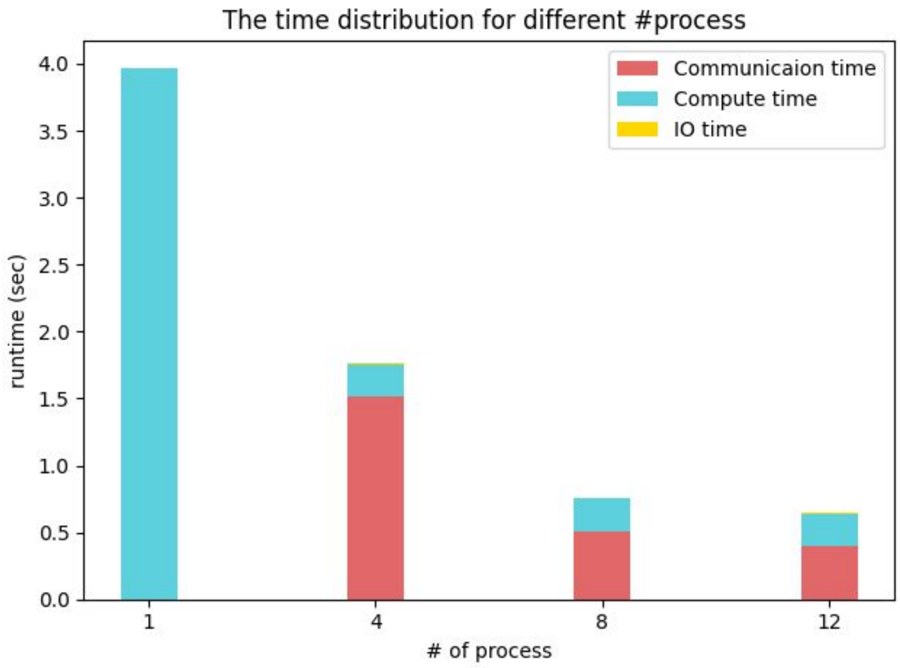
- 4 nodes
  - testcase : input size = 1024*1024, kernel size = 96*96

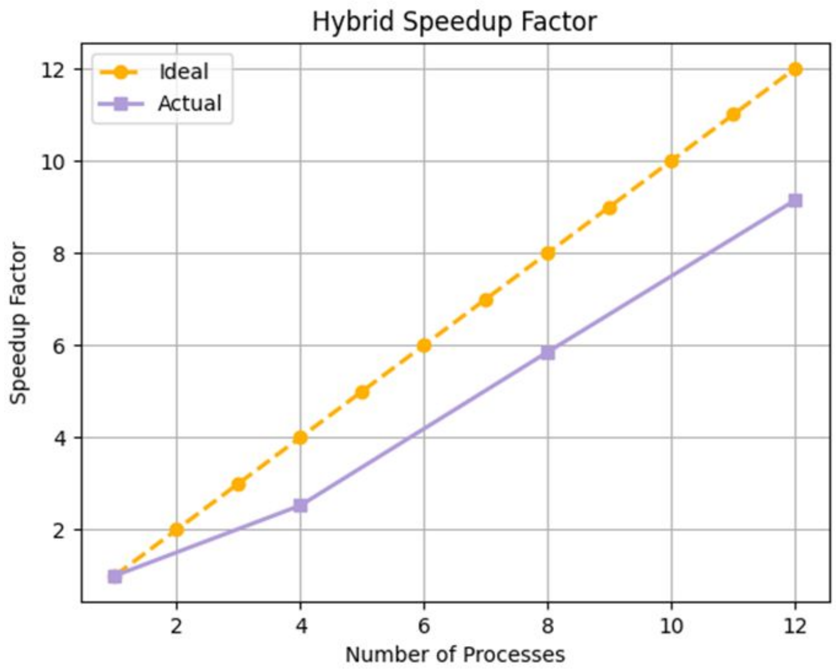




Experiment - Hybrid

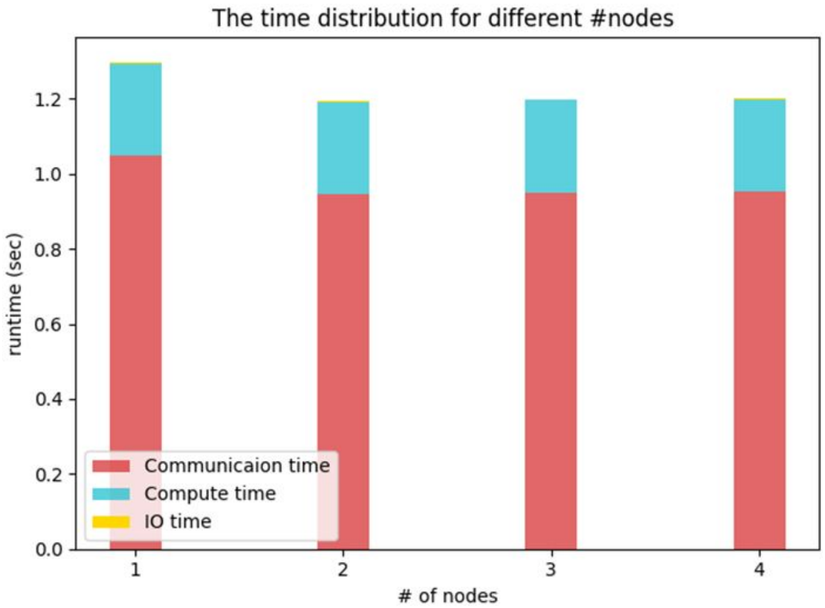
- 4 nodes
  - testcase : input size = 1024*1024, kernel size = 96*96



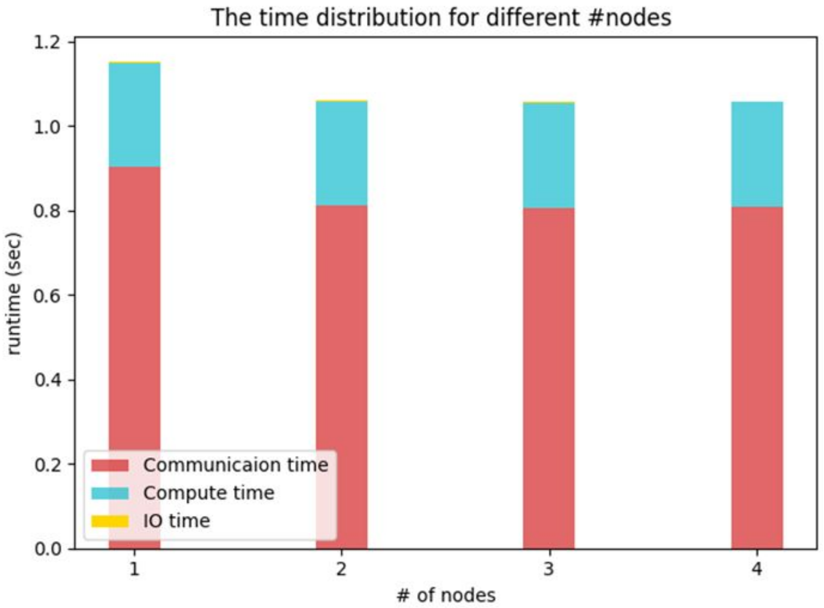


Experiment - MPI & Hybrid

- Comparison of different #nodes
    - testcase : input size = 1024*1024, kernel size = 96*96, ppn = 6



MPI

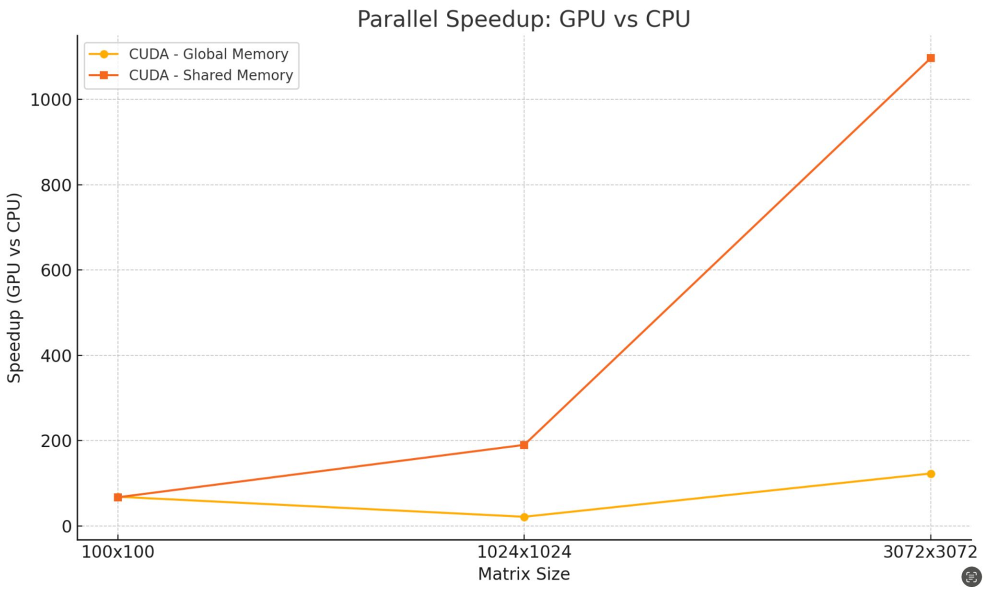


Hybrid

Experiment - Speedup GPU v.s sequential CPU

Compute Time (ms)	100 100 5	1024 1024 96	3072 3072 17
CPU - naïve sequential	0.200	3,932.212	3125.228
CUDA - global memory	0.104	56.326	39.733
CUDA - shared memory	0.107	0.110	4.4482
CPU - parallel version (MPI)	7.124	1197	4878.624

Speedup / against GPU	100 100 5	1024 1024 96	3072 3072 17
CUDA - global memory	1.915708812	69.812	78.656
CUDA - shared memory	1.877934272	35877.847	702.58
Speedup / against GPU -global	100 100 5	1024 1024 96	3072 3072 17
CUDA -speed up smem / global	0.98028169	513.923	8.932
Speedup / against CPU - parallel	100 100 5	1024 1024 96	3072 3072 17
CUDA - global memory	68.2375	21.2513	122.7852
CUDA - shared memory	66.8920	189.8240	1096.7636



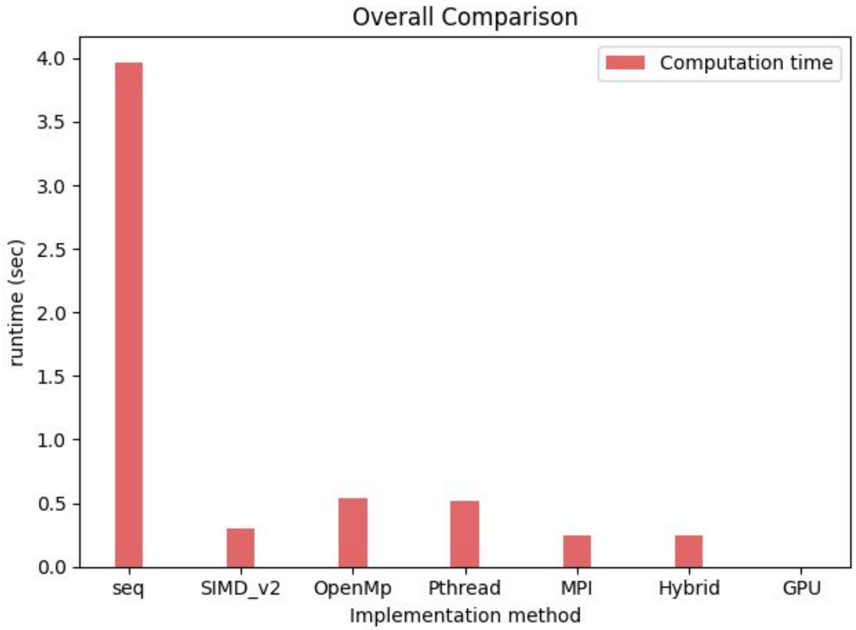
GPU Analysis

```
==1714625== Profiling application: ./conv_gpu_v2 ./testcases/1024_1024_96.in ./profile.out
==1714625== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   56.75%  802.19us         2  401.09us  4.3520us  797.83us  [CUDA memcpy HtoD]
                   43.25%  611.40us         1  611.40us  611.40us  611.40us  [CUDA memcpy DtoH]
      API calls:   97.32%  104.02ms         3  34.673ms  64.887us  103.88ms  cudaMalloc
                    1.89%  2.0173ms         3  672.44us  79.650us  1.0278ms  cudaMemcpy
                    0.40%  425.97us         3  141.99us  93.392us  187.82us  cudaFree
                    0.17%  180.51us       114  1.5830us      92ns  70.661us  cuDeviceGetAttribute
                    0.11%  120.28us         1  120.28us  120.28us  120.28us  cudaLaunchKernel
                    0.04%  40.553us         2  20.276us     620ns  39.933us  cudaEventCreate
                    0.03%  27.998us         2  13.999us  3.7630us  24.235us  cudaEventRecord
                    0.02%  17.822us         1  17.822us  17.822us  17.822us  cuDeviceGetName
                    0.02%  16.323us         1  16.323us  16.323us  16.323us  cuDeviceGetPCIBusId
                    0.01%  12.426us         1  12.426us  12.426us  12.426us  cudaEventSynchronize
                    0.00%  4.3800us         1  4.3800us  4.3800us  4.3800us  cudaEventElapsedTime
                    0.00%  1.1980us         3     399ns     131ns     867ns  cuDeviceGetCount
                    0.00%     554ns         2     277ns     119ns     435ns  cuDeviceGet
                    0.00%     356ns         1     356ns     356ns     356ns  cuDeviceTotalMem
                    0.00%     342ns         1     342ns     342ns     342ns  cuModuleGetLoadingMode
                    0.00%     226ns         1     226ns     226ns     226ns  cuDeviceGetUuid
```

- CGMA ratio
- align shared memory

Experiment - Overall Comparison

- testcases : input size = 1024*1024, kernel size = 96*96
- Only consider the computing time and communication time
  - Sequential
  - SIMD_v2
  - OpenMP + SIMD_v1 + 4 cores
  - Pthread + SIMD_v1 + 4 cores
  - MPI + 4 nodes + (ppn=12)
  - Hybrid + 4 nodes + (ppn=12)
  - GPU (shared memory)



Outline

- Problem formulation
- Implementation
- Expirement
- Future work

Future work: Separatable Convolution

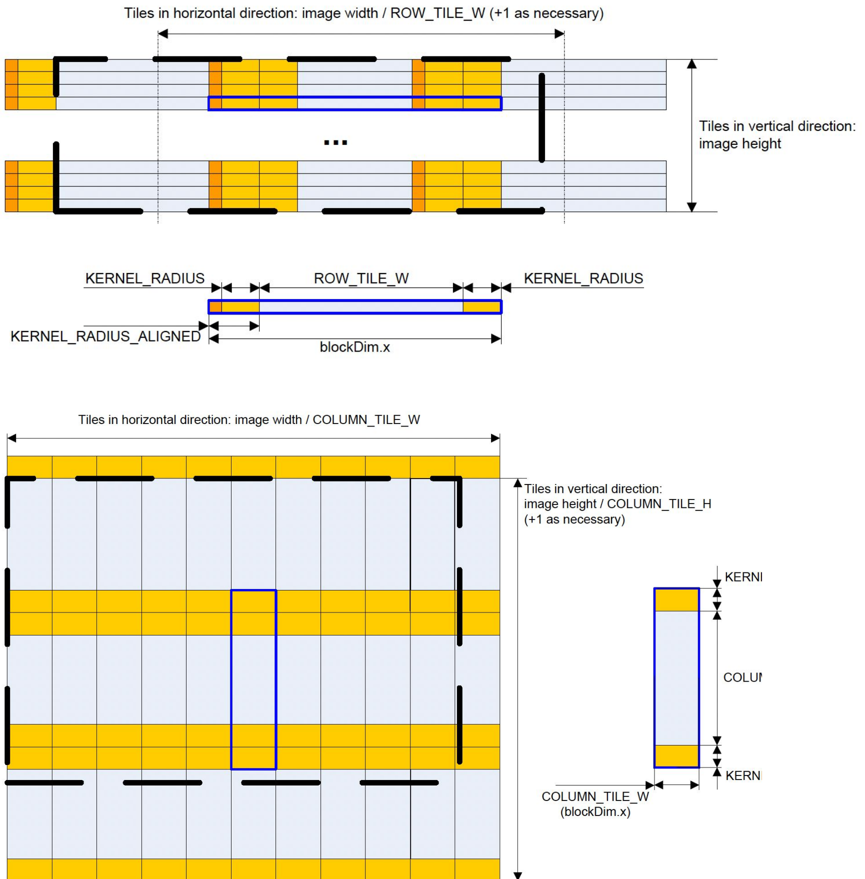
- Devide into two one dimensional filter (row convolution -> col convolution)
- Constraint: kernel matrix must be rank-1
  → so can be written as two rank-1 kernel matrix outer product

Applying $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ to the data is the same as applying $\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$ followed by $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$.

- Application example of kernel matrix are rank-1 : Gaussian Blur, Sobel Edge detect (lab practice)

https://github.com/NVIDIA/cuda-samples/tree/master/Samples/2_Concepts_and_Techniques/convolutionSeparable

How it works and Application



- Good for big kernel size
- Time complexity:

$$O(n^2 \cdot k^2) \text{ ---> } O(n^2 \cdot k)$$

- save share memory usage, store k, instead of k*k in each pass (can process the real data)
- Better scalability
- direct: growth quadratic k^2
- seperatable: growth lineary k

Thanks