# TA Session 2

조교 김진표, 박대영, 신준식, 정재훈

pp-ta@aces.snu.ac.kr

THUNDER Research Group
Seoul National University
서울대학교 천둥 연구실

# 구성

- 오전
  - 과제1 해설
  - FMA & Vector instruction 예제
  - 과제 2: Pthread matmul

- 오후
  - Cache optimization
  - 과제2 진행

# FMA 사용 예시 - Vector inner product

- Fused Multiply Add
  - $c = c + a \times b$
  - 실수 곱셈과 덧셈을 한번에 수행
  - 대부분의 프로세서에서 지원

- Vector inner product (벡터 내적)
  - $c_i = a_0 b_0 + a_1 b_1 + \ ... + a_{n-1} b_{n-1}$

# FMA 사용 예시 - Vector inner product

- 로그인 노드의 `/skeleton/fma-example`
  - `fma.c, run.sh`

- 벡터 내적을 네 가지 방식으로 구현
  - naive
  - loop unrolling
  - vector instruction
  - loop unroll + vector instruction

# FMA 사용 예시 - Vector inner product

- naive

```
for (unsigned int i = 0; i < NUM_ELEM; ++i) {
  c += a[i] * b[i];
}
```

- loop unrolling

```
for (unsigned int i = 0; i < NUM_ELEM / 8; ++i) {
  s0 += a[8 * i + 0] * b[8 * i + 0];
  s1 += a[8 * i + 1] * b[8 * i + 1];
  s2 += a[8 * i + 2] * b[8 * i + 2];
  s3 += a[8 * i + 3] * b[8 * i + 3];
  s4 += a[8 * i + 4] * b[8 * i + 4];
  s5 += a[8 * i + 5] * b[8 * i + 5];
  s6 += a[8 * i + 6] * b[8 * i + 6];
  s7 += a[8 * i + 7] * b[8 * i + 7];
}
float c = s0 + s1 + s2 + s3 + s4 + s5 + s6 + s7;
```

# FMA 사용 예시 - Vector inner product

- vector instruction (fma)

```
void vector(float *a, float *b) {
  __m256 a0;
  __m256 b0;
  __m256 s0;
  s0 = _mm256_set_ps(0., 0., 0., 0., 0., 0., 0., 0.);

  double st = get_time();
  for (unsigned int i = 0; i < NUM_ELEM / 8; ++i) {
    a0 = _mm256_load_ps(a + i * 8);
    b0 = _mm256_load_ps(b + i * 8);
    s0 = _mm256_fmadd_ps(a0, b0, s0);
  }

  float c = s0[0] + s0[1] + s0[2] + s0[3] + s0[4] + s0[5] + s0[6] + s0[7];
  double et = get_time();
  print_result(c, et - st);
}
```

- `#include <immintrin.h>` 및 `-mfma -mavx2` 옵션을 주어 컴파일
- load 주소는 32byte align 되어 있어야 함

# FMA 사용 예시 - Vector inner product

- loop unrolling + vector instruction

```
for (unsigned int i = 0; i < NUM_ELEM / (8 * 8); ++i) {
  a0 = _mm256_load_ps(a + i * 64);
  b0 = _mm256_load_ps(b + i * 64);
  a1 = _mm256_load_ps(a + i * 64 + 8);
  b1 = _mm256_load_ps(b + i * 64 + 8);
  a2 = _mm256_load_ps(a + i * 64 + 16);
  b2 = _mm256_load_ps(b + i * 64 + 16);
  a3 = _mm256_load_ps(a + i * 64 + 24);
  b3 = _mm256_load_ps(b + i * 64 + 24);
  a4 = _mm256_load_ps(a + i * 64 + 32);
  b4 = _mm256_load_ps(b + i * 64 + 32);
  a5 = _mm256_load_ps(a + i * 64 + 40);
  b5 = _mm256_load_ps(b + i * 64 + 40);
  a6 = _mm256_load_ps(a + i * 64 + 48);
  b6 = _mm256_load_ps(b + i * 64 + 48);
  a7 = _mm256_load_ps(a + i * 64 + 56);
  b7 = _mm256_load_ps(b + i * 64 + 56);

  s0 = _mm256_fmadd_ps(a0, b0, s0);
  s1 = _mm256_fmadd_ps(a1, b1, s1);
  s2 = _mm256_fmadd_ps(a2, b2, s2);
  s3 = _mm256_fmadd_ps(a3, b3, s3);
  s4 = _mm256_fmadd_ps(a4, b4, s4);
  s5 = _mm256_fmadd_ps(a5, b5, s5);
  s6 = _mm256_fmadd_ps(a6, b6, s6);
  s7 = _mm256_fmadd_ps(a7, b7, s7);
}

float c0 = s0[0] + s0[1] + s0[2] + s0[3] + s0[4] + s0[5] + s0[6] + s0[7];
float c1 = s1[0] + s1[1] + s1[2] + s1[3] + s1[4] + s1[5] + s1[6] + s1[7];
float c2 = s2[0] + s2[1] + s2[2] + s2[3] + s2[4] + s2[5] + s2[6] + s2[7];
```

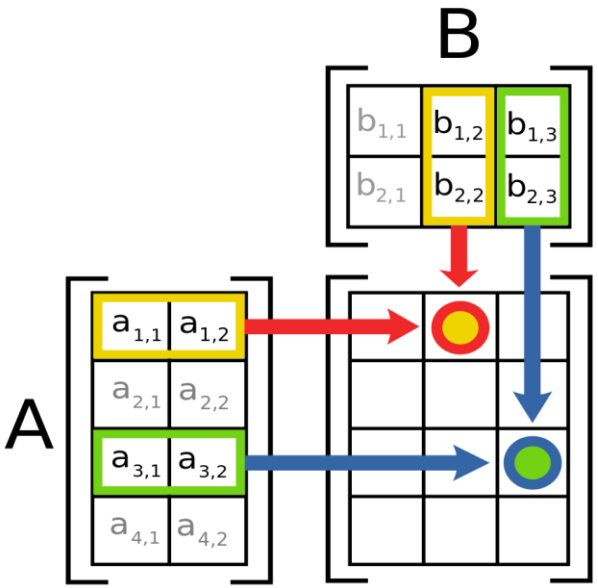
# FMA 사용 예시 - Vector inner product

- run.sh 으로 컴파일 및 실행 (성능 비교)

```
kjp4155@login:~/fma-example$ ./run.sh
========= Naive =========
Result : 32809.589844
N : 131072
Elapsed time : 0.000998 sec
Throughput : 0.26266 GFLOPS

========= Loop Unrolling =========
Result : 32809.957031
N : 131072
Elapsed time : 0.000610 sec
Throughput : 0.42966 GFLOPS

========= Vector Instruction(FMA) =========
Result : 32809.957031
N : 131072
Elapsed time : 0.000264 sec
Throughput : 0.99234 GFLOPS

========= Loop Unrolling + Vector Instruction(FMA) =========
Result : 32809.925781
N : 131072
Elapsed time : 0.000164 sec
Throughput : 1.59813 GFLOPS

kjp4155@login:~/fma-example$
```

# 과제2: Pthread matmul

- FP32 행렬 둘을 곱하는 프로그램을 최적화
- $A_{M\times K} \times B_{K\times N} = C_{M\times N}$
- $c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \ ... + a_{iK}b_{Kj} = \sum_{k=1}^{K} a_{ik}b_{kj}$






https://en.wikipedia.org/wiki/Matrix_multiplication#/media/File:Matrix_multiplication_diagram_2.svg

# Pthread matmul

- /skeleton/HW2 에 주어진 뼈대 코드를 각자의 홈 디렉토리로 복사해 작업
  - `cp -r /skeleton/HW2 ~/`
  - `Makefile, run.sh, main_ref, …`
  - `make` 로 컴파일이 가능하도록 `Makefile` 을 제공
  - 실행을 위한 `run.sh` 스크립트를 제공

- 뼈대 코드는 주어진 옵션에 상관없이 1개의 스레드만을 생성해 행렬 곱셈을 수행함

- 뼈대 코드를 수정해 행렬 곱셈을 최적화하는 것이 목표
  - `mat_mul.cpp` 만 수정 가능
  - `mat_mul` 함수의 인자로 넘어온 `_num_threads` 개의 스레드만을 생성할 것

# Pthread matmul - 실행 예시

- make 명령으로 빌드

```
kjp4155@login:~/HW2$ ls
main.cpp  Makefile  mat_mul.cpp  mat_mul.h  run.sh  util.cpp  util.h
kjp4155@login:~/HW2$ make
g++  -std=c++11 -O3 -Wall -march=native  -c -o util.o util.cpp
g++  -std=c++11 -O3 -Wall -march=native  -c -o mat_mul.o mat_mul.cpp
g++  -std=c++11 -O3 -Wall -march=native -lm -pthread  main.cpp util.o mat_mul.o   -o main
kjp4155@login:~/HW2$ ls
main  main.cpp  Makefile  mat_mul.cpp  mat_mul.h  mat_mul.o  run.sh  util.cpp  util.h  util.o
kjp4155@login:~/HW2$
```

- run.sh, run_performance.sh, run_validate.sh 으로 실행
  - srun 을 통해 계산 노드에서 실행하는 커맨드

```
kjp4155@login:~/HW2$ cat run.sh
#!/bin/bash

srun --nodes=1 --exclusive  numactl --physcpubind 0-39 ./main $@
kjp4155@login:~/HW2$
```

# Pthread matmul - 실행 예시

- ./run.sh 을 사용해 실행
  - 옵션을 주어 실행 가능
  - 각종 디버깅/성능 실험에 활용

```
kjp4155@login:~/HW2_answer$ ./run.sh -v -n 5 -t 3 512 512 512
Options:
  Problem size: M = 512, N = 512, K = 512
  Number of threads: 3
  Number of iterations: 5
  Print matrix: off
  Validation: on

Initializing... done!
Calculating...(iter=0) 0.018954 sec
Calculating...(iter=1) 0.016933 sec
Calculating...(iter=2) 0.016106 sec
Calculating...(iter=3) 0.016062 sec
Calculating...(iter=4) 0.015967 sec
Validating...
Result: VALID
Avg. time: 0.016804 sec
Avg. throughput: 15.974154 GFLOPS
kjp4155@login:~/HW2_answer$
```

# Pthread matmul - 실행 예시

- ./run_performance.sh 을 사용해 실행
  - srun 을 통해 **계산 노드에서 실행하는** 커맨드
  - 20 thread, 10 iteration, 4096 x 4096 x 4096 으로 실행
  - 성능 평가 시 사용할 방법

```
kjp4155@login:~/HW2$ ./run_performance.sh
Options:
  Problem size: M = 4096, N = 4096, K = 4096
  Number of threads: 20
  Number of iterations: 10
  Print matrix: off
  Validation: off

Initializing... done!
```

# Pthread matmul - 실행 예시

- ./run_validate.sh 을 사용해 실행
  - srun 을 통해 **계산 노드에서 실행하는** 커맨드
  - 10개의 test case
  - 정확성 평가 시 사용할 방법

```
kjp4155@login:~/HW2$ ./run_validate.sh
Options:
  Problem size: M = 831, N = 538, K = 2304
  Number of threads: 26
  Number of iterations: 1
  Print matrix: off
  Validation: on

Initializing... done!
Calculating...(iter=0) 1.167624 sec
Validating...
Result: VALID
Avg. time: 1.167624 sec
Avg. throughput: 1.764383 GFLOPS
Options:
  Problem size: M = 3305, N = 1864, K = 3494
  Number of threads: 9
  Number of iterations: 1
  Print matrix: off
  Validation: on
```

# Pthread matmul - 실행 예시

- 다양한 옵션 활용
  - -p : 디버깅을 위해 행렬 출력
  - -v : 행렬곱 결과 검증
  - -h : help
  - -t : 스레드 개수
  - -n : 반복 실험 횟수 (iteration)

# Pthread programming

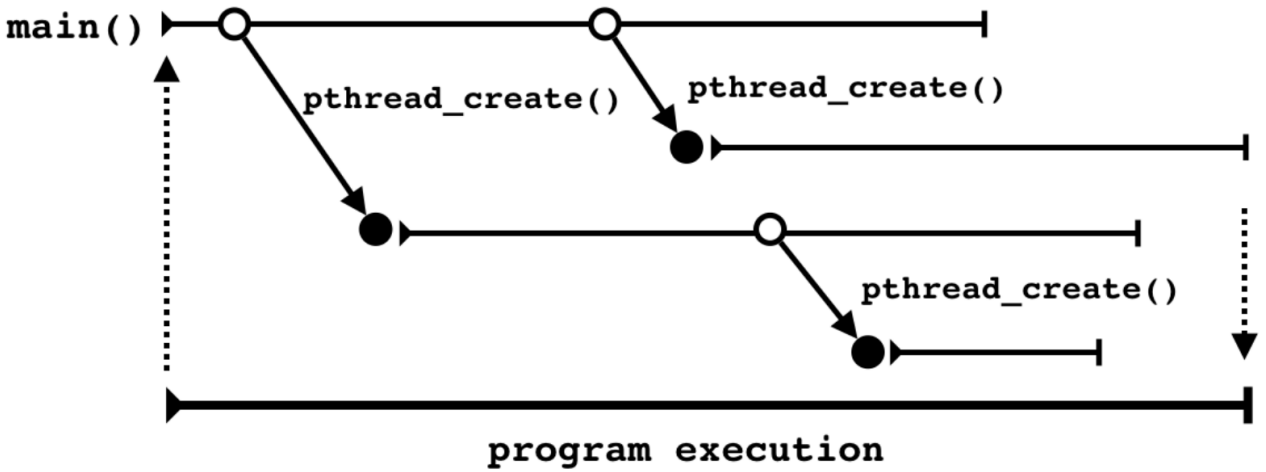
- 스레드 생성: pthread_create
  - https://man7.org/linux/man-pages/man3/pthread_create.3.html
- 다른 스레드 종료 대기: pthread_join
  - https://man7.org/linux/man-pages/man3/pthread_join.3.html

- 뼈대 코드 참고

```
void mat_mul(float *_A, float *_B, float *_C, int _M, int _N, int _K, int _num_threads) {
  A = _A, B = _B, C = _C;
  M = _M, N = _N, K = _K;
  num_threads = _num_threads;

  pthread_t thread;
  pthread_create(&thread, NULL, mat_mul_thread, NULL);
  pthread_join(thread, NULL);
}
```

# Pthread programming

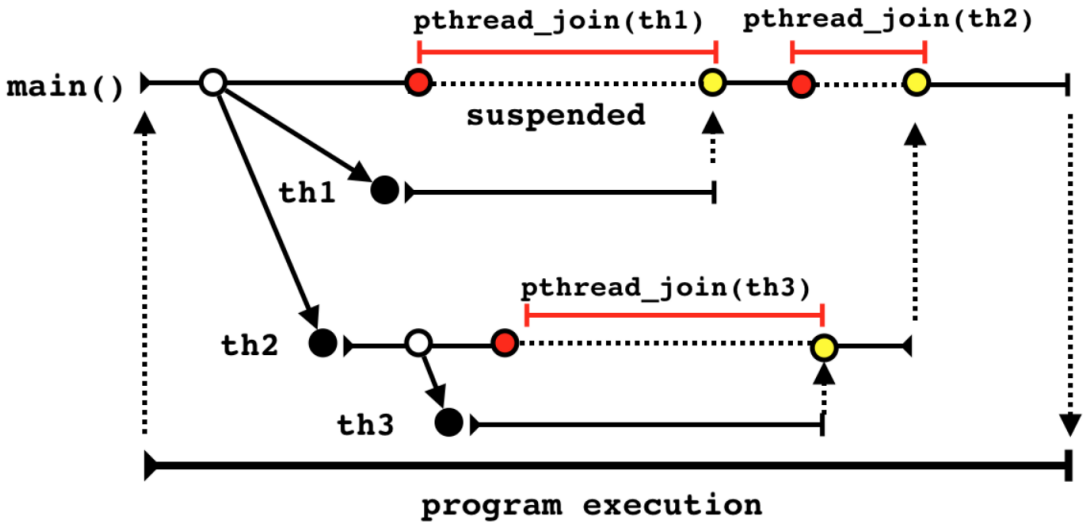
- 스레드 생성: pthread_create
  - https://man7.org/linux/man-pages/man3/pthread_create.3.html




https://www.dcc.fc.up.pt/~edrdo/CP1819/notes/slides/pthreads.pdf

# Pthread programming

- 다른 스레드 종료 대기: `pthread_join`
  - https://man7.org/linux/man-pages/man3/pthread_join.3.html




https://www.dcc.fc.up.pt/~edrdo/CP1819/notes/slides/pthreads.pdf

# Pthread programming

- Hint: 각 스레드가 결과 행렬 $C_{M \times N}$ 의 서로 다른 부분을 맡아서 계산
  - 스레드끼리 안 겹치도록 작업을 어떻게 나눌 것인지
  - 각 스레드의 속도를 높이는 것도 중요

# Pthread matmul - 평가 기준

- 정확성 점수 50%
  - 무작위 N M K 및 스레드 개수에 대해 -v 옵션을 주어 값 검증을 통과해야 함
  - 총 10개의 test case 로 채점할 예정
  - N, M, K 각각은 2048 이하.
  - 스레드 개수는 40개 이하.

- 성능 점수 50%
  - 1노드 20thread 사용, 10 iteration, 4096 x 4096 x 4096 기준 **200 GFLOPS** 달성 시 만점
  - 그 이하는 성능에 비례해 점수 부여
  - 답이 틀리면 0점 (-v 옵션)
  - ex. 100 GFLOPS 달성 시성능 점수의 절반 부여

# Pthread matmul - 생각해 볼 것

- CPU의 Theoretical peak FLOPS 와 달성한 성능 비교

- 스레드를 80개 사용하였을 때, 1 - 40개의 스레드를 사용하였을 때와 비슷한 추세로 성능이 증가하는가?
  - 실습 서버의 CPU 는 SMT가 적용되어 physical core는 총 40개이지만 logical core는 80개이다.
  - 80 스레드 실험을 위해선 run.sh 파일의 numactl --physbind 부분을 0-39 에서 0-79로 변경할 것

# Pthread matmul - 주의사항

- `mat_mul.cpp` 만 수정 가능
- 다양한 행렬 크기에 대해 답이 맞는지 체크를 할 것 (-v 옵션)
- 로그인 노드에서 성능 측정을 하지 말 것
  - 반드시 주어진 run.sh 혹은 srun 으로 계산 노드에서 실행

# Pthread matmul - 주의사항

- 사용한 스레드 개수를 셀 때, 계산을 수행한 스레드만 개수를 셈
  - 예제 코드는 메인 스레드 1개 + 생성된 스레드 1개, 총 2개의 스레드를 사용하지만 메인 스레드에서는 계산을 하지 않으므로 1개의 스레드로 취급
  - 제출한 코드가 메인 스레드에서 계산을 수행한다면 메인 스레드도 개수에 포함된다

- 반드시 주어진 num_threads 이하의 스레드를 사용하도록 구현해야 함
  - 몰래 더 많은 스레드를 사용해서 성능의 이득을 본 경우 0점 처리

- Pthread 이외의 병렬화 방식 (OpenMP, MPI, OpenCL 등) 사용 금지
  - Vector instruction 은 사용 가능

# 제출 방법

- 각자의 홈 디렉토리에 `~/submit/HW2` 디렉토리 생성
- 구현한 mat_mul.cpp 파일을 디렉토리 내에 복사

- **디렉토리명, 파일명이 정확히 일치하도록**
- 즉, `ls` 명령의 출력이 아래 스크린샷과 같이 나와야 함

```
kjp4155@login:~$ ls ~/submit/HW2
mat_mul.cpp
kjp4155@login:~$
```

- check-submit 유틸리티 활용

```
kjp4155@login:~$ check-submit
Username: kjp4155
------------------------------------
HW1 prob1.txt   : X file not found
HW1 prob2.txt   : OK
HW1 main.cpp    : X empty file
------------------------------------
HW2 mat_mul.cpp : OK
------------------------------------
kjp4155@login:~$
```

# 주의사항

- **제출 기한: 4월 27일 (수) 14:29**
  - 기한이 되면 프로그램이 자동으로 제출 파일을 복사해 감
  - 파일명, 디렉토리, 형식을 정확히 지켰는지 체크
  - 평가를 하는 과제

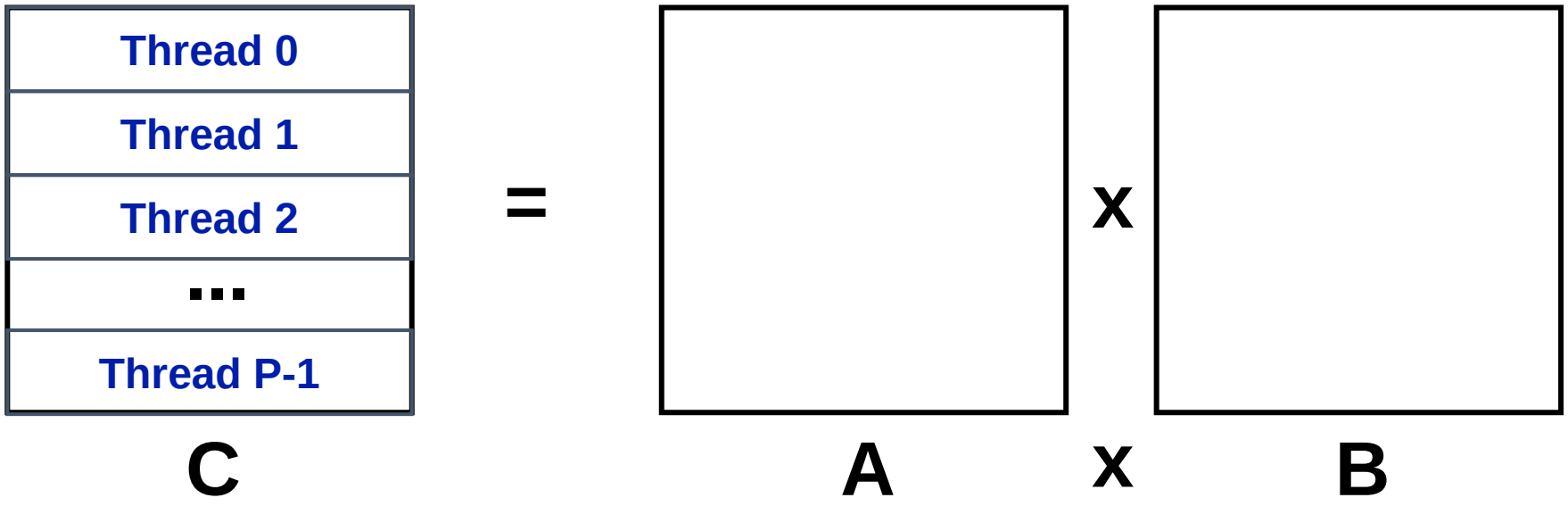
- **조교가 실수를 수정해줄 것이라 기대하지 않을 것**
  - 제출 디렉토리 확인
  - 제출 파일명 확인
  - 제출한 파일 내용 확인
  - 프로그램이 불필요한 출력을 하는 지 확인
  - 디버깅 코드 등 불필요한 코드를 모두 제거했는지 확인

# Pthread matmul 해설

- 예시 답안을 `/skeleton/HW2-ans` 에 준비해 둠
  - `mat_mul.cpp`
    - ~ 330 GFLOPS
  - `mat_mul_4096.cpp`
    - ~ 600 GFLOPS

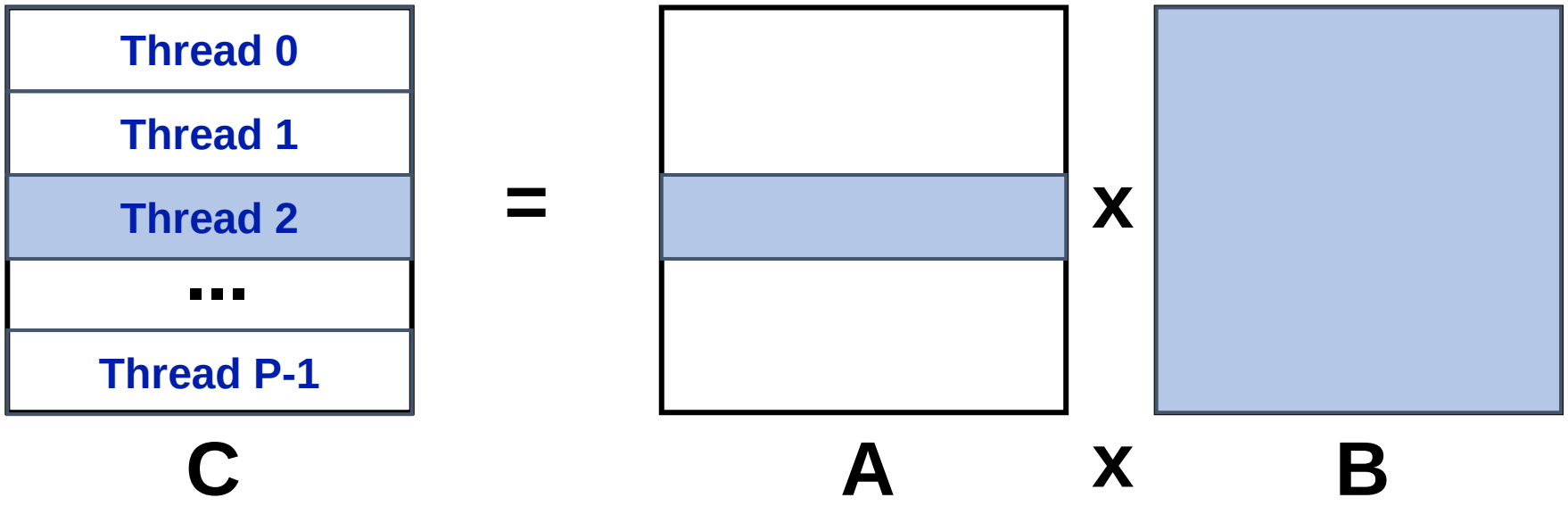
# Pthread matmul 해설

- mat_mul.cpp
  - 각 스레드가 결과 행렬 C 의 일부분을 나누어 계산
  - 행 단위로 고르게 나눔
  - 기본적인 타일링 & 루프 순서 변경



# Pthread matmul 해설

- mat_mul.cpp
  - 각 스레드가 결과 행렬 C 의 일부분을 나누어 계산
  - 행 단위로 고르게 나눔
  - 기본적인 타일링 & 루프 순서 변경



# Pthread matmul 해설

- mat_mul.cpp
  - 각 스레드가 결과 행렬 C 의 일부분을 나누어 계산
  - 행 단위로 고르게 나눔
  - 기본적인 타일링 & 루프 순서 변경

```
43  void mat_mul(float *_A, float *_B, float *_C, int _M, int _N, int _K, int _num_threads) {
44    A = _A, B = _B, C = _C;
45    M = _M, N = _N, K = _K;
46    num_threads = _num_threads;
47    pthread_t threads[num_threads];
48    for (long i = 0; i < num_threads; ++i) {
49      pthread_create(&threads[i], NULL, mat_mul_thread, (void*)i);
50    }
51    for (long i = 0; i < num_threads; ++i) {
52      pthread_join(threads[i], NULL);
53    }
54  }
```

# Pthread matmul 해설

- mat_mul.cpp

```cpp
static void* mat_mul_thread(void *data) {
  int tid = (long)data;
  int is = M / num_threads * tid + min(tid, M % num_threads);
  int ie = M / num_threads * (tid + 1) + min(tid + 1, M % num_threads);

  for (int ii = is; ii < ie; ii += ITILESIZE) {
    for (int jj = 0; jj < N; jj += JTILESIZE) {
      for (int kk = 0; kk < K; kk += KTILESIZE) {

        for (int k = kk; k < kk + KTILESIZE; k++) {
          for (int i = ii; i < ii + ITILESIZE; i++) {
            float ar = A[i * K + k];
            for (int j = jj; j < jj + JTILESIZE; j+=1) {
              C[i * N + j] += ar * B[k * N + j];
            }
          }
        }

      }
    }
  }

  return NULL;
}
```

# Pthread matmul 해설

- mat_mul_4096.cpp
  - loop unrolling + vectorization (AVX2) 적용
  - 행렬 크기가 2의 지수승인 경우에만 성능을 올린 것
  - validation 은 통과 못함
  - 행렬 크기가 보장되어 있는 경우에 사용 가능