

Name-Ketki Khati

Nu ID-002436886

Assignment

- Create a [modal.com](#) account **using your github account.**
- Join the [modal.com](#) course workspace by clicking on the invite link
- Read through these sections of the [modal.com](#) documentation and run the examples:
 - https://modal.com/docs/examples/hello_world
 - <https://modal.com/docs/guide/images>
 - <https://modal.com/docs/guide/gpu>
 - <https://modal.com/docs/guide/cuda>
 - <https://modal.com/docs/guide/resources>
 - Please note that there are limited funds in the shared workspace. Please monitor your usage carefully and avoid submitting unnecessary runs.
- Implement matrix multiplication
 - Implement a single-threaded version in C
 - Write test cases. Check a wide range of different dimensions for the input matrices to make sure your code has no errors in corner cases.
 - for example:
 - A = 1x1, B = 1x1
 - A = 1x1, B = 1x5
 - A = 2x1, B = 1x3
 - A = 2x2, B = 2x2
 - and others.
 - Implement a multi-threaded version in C using pthreads.
 - Check that your test cases pass for the multi threaded version

- Measure the speed up for different thread counts: 1, 4, 16, 32, 64, 128.
 - Use large matrices so that the speed up is measure-able.

CODE-

```
#define _GNU_SOURCE

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>

#include <string.h>

#include <time.h>

#include <pthread.h>

#include <math.h>

typedef struct {

    const double *A;

    const double *B;

    double *C;

    int M, K, N;      // A: MxK, B: KxN, C: MxN

    int row_start;

    int row_end;      // [start, end)

} WorkerArgs;

static inline double rand_unit(unsigned *seed) {

    // simple deterministic RNG

    *seed = (*seed * 1103515245u + 12345u);

    return (double)((*seed / 65536u) % 32768u) / 32768.0;

}
```

```

static double now_seconds(void) {
    struct timespec ts;
    clock_gettime(CLOCK_MONOTONIC, &ts);
    return (double)ts.tv_sec + (double)ts.tv_nsec * 1e-9;
}

static void matmul_single(const double *A, const double *B, double *C, int M, int K, int N) {
    // Row-major layout:
    // A[i*K + k], B[k*N + j], C[i*N + j]
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            double sum = 0.0;
            for (int k = 0; k < K; k++) {
                sum += A[i*K + k] * B[k*N + j];
            }
            C[i*N + j] = sum;
        }
    }
}

static void* worker_rows(void *arg) {
    WorkerArgs *w = (WorkerArgs*)arg;
    const double *A = w->A;
    const double *B = w->B;
    double *C = w->C;
}

```

```

for (int i = w->row_start; i < w->row_end; i++) {
    for (int j = 0; j < w->N; j++) {
        double sum = 0.0;
        for (int k = 0; k < w->K; k++) {
            sum += A[i*w->K + k] * B[k*w->N + j];
        }
        C[i*w->N + j] = sum;
    }
}
return NULL;
}

static int matmul_pthreads(const double *A, const double *B, double *C, int M, int K, int N,
int num_threads) {
    if (num_threads < 1) num_threads = 1;
    if (num_threads > M) num_threads = M; // more threads than rows doesn't help

    pthread_t *threads = (pthread_t*)malloc(sizeof(pthread_t) * (size_t)num_threads);
    WorkerArgs *args = (WorkerArgs*)malloc(sizeof(WorkerArgs) * (size_t)num_threads);

    if (!threads || !args) {
        free(threads); free(args);
        return -1;
    }

    int base = M / num_threads;

```

```
int rem = M % num_threads;

int row = 0;

for (int t = 0; t < num_threads; t++) {

    int take = base + (t < rem ? 1 : 0);

    args[t].A = A; args[t].B = B; args[t].C = C;
    args[t].M = M; args[t].K = K; args[t].N = N;
    args[t].row_start = row;
    args[t].row_end = row + take;
    row += take;

    int rc = pthread_create(&threads[t], NULL, worker_rows, &args[t]);
    if (rc != 0) {
        // join what started
        for (int j = 0; j < t; j++) pthread_join(threads[j], NULL);
        free(threads); free(args);
        return -2;
    }
}

for (int t = 0; t < num_threads; t++) pthread_join(threads[t], NULL);

free(threads);
free(args);
return 0;
}
```

```
static void fill_random(double *X, int len, unsigned seed0) {
    unsigned seed = seed0;
    for (int i = 0; i < len; i++) X[i] = rand_unit(&seed) - 0.5;
}
```

```
static int almost_equal(double a, double b, double tol) {
    double diff = fabs(a - b);
    double scale = fmax(1.0, fmax(fabs(a), fabs(b)));
    return diff <= tol * scale;
}
```

```
static int check_equal(const double *X, const double *Y, int len, double tol) {
    for (int i = 0; i < len; i++) {
        if (!almost_equal(X[i], Y[i], tol)) return 0;
    }
    return 1;
}
```

```
static void run_tests(void) {
    struct { int M, K, N; } cases[] = {
        {1,1,1},
        {1,1,5},
        {2,1,3},
        {2,2,2},
        {3,4,2},
    }
```

```

{4,3,5},
{7,8,9},
{16,16,16},
{31,17,29},
};

for (size_t idx = 0; idx < sizeof(cases)/sizeof(cases[0]); idx++) {
    int M = cases[idx].M, K = cases[idx].K, N = cases[idx].N;

    double *A = (double*)malloc(sizeof(double)*(size_t)(M*K));
    double *B = (double*)malloc(sizeof(double)*(size_t)(K*N));
    double *C1 = (double*)malloc(sizeof(double)*(size_t)(M*N));
    double *C2 = (double*)malloc(sizeof(double)*(size_t)(M*N));
    if (!A || !B || !C1 || !C2) { fprintf(stderr, "malloc failed\n"); exit(1); }

    fill_random(A, M*K, 1234u + (unsigned)idx);
    fill_random(B, K*N, 5678u + (unsigned)idx);

    matmul_single(A, B, C1, M, K, N);

    // test multiple thread counts
    int thread_counts[] = {1,2,4,8,16,32,64,128};
    for (size_t t = 0; t < sizeof(thread_counts)/sizeof(thread_counts[0]); t++) {
        memset(C2, 0, sizeof(double)*(size_t)(M*N));
        matmul_pthreads(A, B, C2, M, K, N, thread_counts[t]);
        if (!check_equal(C1, C2, M*N, 1e-9)) {

```

```

        fprintf(stderr, "TEST FAIL: M=%d K=%d N=%d threads=%d\n", M, K, N,
thread_counts[t]);

        exit(2);

    }

}

free(A); free(B); free(C1); free(C2);

}

printf("All tests passed ✅\n");

}

static double bench_matmul(int M, int K, int N, int threads, int iters) {

double *A = (double*)malloc(sizeof(double) * (size_t)(M*K));
double *B = (double*)malloc(sizeof(double) * (size_t)(K*N));
double *C = (double*)malloc(sizeof(double) * (size_t)(M*N));

if (!A || !B || !C) { fprintf(stderr, "alloc failed\n"); exit(1); }

fill_random(A, M*K, 111u);
fill_random(B, K*N, 222u);

// warmup

if (threads == 1) matmul_single(A, B, C, M, K, N);
else matmul_pthreads(A, B, C, M, K, N, threads);

```

```
double t0 = now_seconds();

for (int i = 0; i < iters; i++) {
    if (threads == 1) matmul_single(A, B, C, M, K, N);
    else matmul_pthreads(A, B, C, M, K, N, threads);
}

double t1 = now_seconds();

// prevent optimizing away
volatile double sink = C[0];
(void)sink;

free(A); free(B); free(C);

return (t1 - t0) / (double)iters;
}

int main(int argc, char **argv) {
    printf("Program started\n");

    if (argc >= 2 && strcmp(argv[1], "test") == 0) {
        run_tests();
        return 0;
    }

    // Default benchmark dimensions (big enough to see speedup on most machines)
    int M = 1024, K = 1024, N = 1024;
    int iters = 3;
```

```
if (argc >= 5) {  
    M = atoi(argv[1]);  
    K = atoi(argv[2]);  
    N = atoi(argv[3]);  
    iters = atoi(argv[4]);  
    if (iters < 1) iters = 1;  
}  
  
int thread_counts[] = {1,4,16,32,64,128};  
  
printf("Benchmark: A=%dx%d, B=%dx%d, iters=%d\n", M, K, K, N, iters);  
  
double t1 = bench_matmul(M, K, N, 1, iters);  
printf("threads=%3d time=%.6f s speedup=1.00x\n", 1, t1);  
  
for (size_t i = 1; i < sizeof(thread_counts)/sizeof(thread_counts[0]); i++) {  
    int th = thread_counts[i];  
    double tt = bench_matmul(M, K, N, th, iters);  
    printf("threads=%3d time=%.6f s speedup=%.2fx\n", th, tt, t1/tt);  
}  
  
return 0;  
}
```

OUTPUT-

```
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\khati\Documents\Ai> $env:Path = "$PWD\mingw64\bin;" + $env:Path
>>
PS C:\Users\khati\Documents\Ai> .\matmul.exe
Program started
Benchmark: A=1024x1024, B=1024x1024, iters=3
threads= 1 time=3.872762 s speedup=1.00x
threads= 4 time=0.839297 s speedup=4.61x
threads= 16 time=0.803383 s speedup=4.82x
threads= 32 time=0.733775 s speedup=5.28x
threads= 64 time=0.701492 s speedup=5.52x
threads=128 time=0.687492 s speedup=5.63x
PS C:\Users\khati\Documents\Ai> .\matmul.exe test
Program started
All tests passed !Fà
PS C:\Users\khati\Documents\Ai> .\matmul.exe 1024 1024 1024 2
>>
Program started
Benchmark: A=1024x1024, B=1024x1024, iters=2
threads= 1 time=3.820700 s speedup=1.00x
threads= 4 time=0.799839 s speedup=4.78x
threads= 16 time=0.784822 s speedup=4.87x
threads= 32 time=0.713022 s speedup=5.36x
threads= 64 time=0.733155 s speedup=5.21x
threads=128 time=0.735309 s speedup=5.20x
PS C:\Users\khati\Documents\Ai>
```

- Read through these sections of the [modal.com](#) documentation and run the examples:
 - https://modal.com/docs/examples/hello_world
 - <https://modal.com/docs/guide/images>
 - <https://modal.com/docs/guide/gpu>
 - <https://modal.com/docs/guide/cuda>
 - <https://modal.com/docs/guide/resources>
 - Please note that there are limited funds in the shared workspace. Please monitor your usage carefully and avoid submitting unnecessary runs.

Name	State	Created	Stopped
resources-demo	stopped	1 minute ago	1 minute ago
cuda-check	stopped	3 minutes ago	2 minutes ago
gpu-hello	stopped	4 minutes ago	4 minutes ago
modal-images-example	stopped	8 minutes ago	7 minutes ago
hello-world	stopped	13 minutes ago	13 minutes ago

1.hello-world

```
import modal
```

```
app = modal.App("hello-world")
```

```
@app.function()
```

```
def hello():
```

```
    print("Hello from Modal!")
```

```
if __name__ == "__main__":
```

```
    with app.run():
```

```
        hello.remote()
```

OUTPUT-

```

Token written to C:\Users\khati/.modal.toml in profile khatiketki.
PS C:\Users\khati\Documents\Ai> python -m modal run Helloworld.py
>>
✓ Initialized. View run at https://modal.com/apps/khatiketki/main/ap-cCdqVvoDjY4nkMnUsSSP8N
✓ Created objects.
└─ ↴ Created mount C:\Users\khati\Documents\Ai\Helloworld.py
└─ ↴ Created function hello.
Hello from Modal!
Stopping app - local entrypoint completed.
✓ App completed. View run at http://python -m modal token newkетки/main/ap-cCdqVvoDjY4nkMnUsSSP8N
>> C:\Users\khati\Documents\Ai>
The web browser should have opened for you to authenticate and get an API token.
If it didn't, please copy this URL into your web browser manually:

https://modal.com/token-flow/tf-maxSKAokQqQi5sgUfAKwiw

Web authentication finished successfully!
Token is connected to the khatiketki workspace.
Verifying token against https://api.modal.com
Token verified successfully!
Token written to C:\Users\khati/.modal.toml in profile khatiketki.

```

2.modal_images.py

```

import modal

# Define a custom Modal image

image = (
    modal.Image.debian_slim(python_version="3.11")
    .apt_install("git")      # install additional system packages
    .pip_install("numpy")    # example: add a pip package
    .run_commands(
        "echo 'This is running inside the Modal Image!'"
    )
)

```

```
app = modal.App("modal-images-example")
```

```

# Define a function that uses that image

@app.function(image=image)

```

```

def use_image():

    import numpy as np

    print("NumPy inside Modal:", np.__version__)

    print("Hello from Modal Image!")


if __name__ == "__main__":
    with app.run():

        use_image.remote()

```

OUTPUT-

```

This is running inside the Modal image.
Saving image...
Image saved, took 1.03s

Built image im-h2UtRZeJ8g2gCtuFPpiIoN in 3.04s

✓ Created objects.
└─ ↴ Created mount C:\Users\khati\Documents\Ai\modal_images.py
└─ ↴ Created function use_image.
NumPy inside Modal: 2.4.1
Hello from Modal Image!
Stopping app - local entrypoint completed.
✓ App completed. View run at https://modal.com/apps/khatiketki/main/ap-AofEmjanee1P1oYy6SoJpj

```

3.modal_gpy.py

```

import modal


app = modal.App("gpu-hello")



# Small image: only installs tiny package
image =
modal.Image.debian_slim(python_version="3.11").pip_install("platformdirs")

```

```

@app.function(image=image, gpu="any") # request any available GPU

def gpu_info():

    import os

    print("Hello from a GPU container on Modal!")

    # This won't guarantee CUDA libs, but proves GPU scheduling works.

    print("Environment variables (sample):")

    for k in ["NVIDIA_VISIBLE_DEVICES", "CUDA_VISIBLE_DEVICES"]:

        print(f" {k}={os.getenv(k)}")



if __name__ == "__main__":
    with app.run():
        gpu_info.remote()

```

OUTPUT-

```

PS C:\Users\khati\Documents\Ai> python -m modal run modal_gpu.py
✓ Initialized. View run at https://modal.com/apps/khatiketki/main/ap-IFsAvYiiXsZNaudWlivODw
Building image im-9a3w7EALggCKJekqlaThjF

=> Step 0: FROM base

=> Step 1: RUN python -m pip install platformdirs
Looking in indexes: http://pypi-mirror.modal.local:5555/simple
Collecting platformdirs
  Downloading http://pypi-mirror.modal.local:5555/simple/platformdirs/platformdirs-4.5.1-py3-none-any.whl.metadata (12 kB)
  Downloading http://pypi-mirror.modal.local:5555/simple/platformdirs/platformdirs-4.5.1-py3-none-any.whl (18 kB)
Installing collected packages: platformdirs
Successfully installed platformdirs-4.5.1

[notice] A new release of pip is available: 25.2 -> 25.3
[notice] To update, run: pip install --upgrade pip
Saving image...
Image saved, took 691.66ms

Built image im-9a3w7EALggCKJekqlaThjF in 3.81s

✓ Created objects.
└─ ↗ Created mount C:\Users\khati\Documents\Ai\modal_gpu.py
   ↗ Created function gpu_info.
Hello from a GPU container on Modal!
Environment variables (sample):
  NVIDIA_VISIBLE_DEVICES = GPU-6f4284c3-d866-d484-5414-f17b2c163746
  CUDA_VISIBLE_DEVICES = None
Stopping app - local entrypoint completed.
✓ App completed. View run at https://modal.com/apps/khatiketki/main/ap-IFsAvYiiXsZNaudWlivODw

```

4.modal_cuda.py

```
import modal

app = modal.App("cuda-check")

# Image with CUDA tools; this is the typical pattern.
# We keep it minimal and just run `nvidia-smi` and `nvcc --version` checks.

image = (
    modal.Image.debian_slim(python_version="3.11")
    .apt_install("pciutils") # lightweight; optional
)

@app.function(image=image, gpu="any")
def cuda_check():
    import subprocess

    print("Running CUDA checks...")

    # nvidia-smi is usually present on GPU containers; if not, we show the error
    for cmd in [["nvidia-smi"], ["bash", "-lc", "nvcc --version"]]:
        try:
            out = subprocess.check_output(cmd, stderr=subprocess.STDOUT, text=True)
            print(f"\n$ {' '.join(cmd)}\n{out}")
        except subprocess.CalledProcessError as e:
```

```
        print(f"\n$ {' '.join(cmd)}\nCommand failed:\n{e.output}")

    except FileNotFoundError:

        print(f"\n$ {' '.join(cmd)}\nNot found in this image/container.")
```

```
if __name__ == "__main__":
    with app.run():
        cuda_check.remote()
```

OUTPUT-

```
✓ Created objects.
└─ ↴ Created mount C:\Users\khati\Documents\Ai\modal_cuda.py
└─ ↴ Created function cuda_check.

Running CUDA checks...

$ nvidia-smi
Sat Jan 17 04:37:49 2026
+-----+
| NVIDIA-SMI 580.95.05      Driver Version: 580.95.05     CUDA Version: 13.0 |
+-----+
| GPU  Name     Persistence-M | Bus-Id     Disp.A  Volatile Uncorr. ECC |
| Fan  Temp     Perf            Pwr:Usage/Cap | Memory-Usage | GPU-Util  Compute M. |
|          |          |          |           |           |          |          MIG M. |
|-----|
|  0  Tesla T4           On     00000000:00:1D.0 Off |       0MiB / 15360MiB |      0%     Default |          N/A |
| N/A   23C     P8          9W / 70W |           0MiB / 15360MiB |      0%     Default |          N/A |
+-----+
+-----+
| Processes:
| GPU  GI  CI          PID  Type  Process name          GPU Memory |
|          ID  ID          ID   ID          Usage          Usage |
|-----|
| No running processes found
+-----+


$ bash -lc nvcc --version
Command failed:
bash: line 1: nvcc: command not found

Stopping app - local entrypoint completed.
✓ App completed. View run at https://modal.com/apps/khatiketki/main/ap-Vpk7sJDxOcQqM0zPlmfv0
```

5.modal_resources.py

```
import modal

import time


app = modal.App("resources-demo")

image = modal.Image.debian_slim(python_version="3.11")

# Configure resources: keep it small to avoid burning credits

@app.function(
    image=image,
    cpu=1,      # request 1 CPU
    memory=512,   # 512 MiB RAM
    timeout=60,    # 60 seconds max
)

def resource_demo():

    print("Resources demo running.")

    print("Doing a tiny bit of work...")

    t0 = time.time()

    s = 0

    for i in range(5_000_00): # small loop

        s += i % 7

    print("Done. checksum =", s, "elapsed =", round(time.time() - t0, 3), "sec")

if __name__ == "__main__":
```

```
with app.run():

    resource_demo.remote()
```

Output-

```
PS C:\Users\khati\Documents\Ai> python -m modal run modal_resources.py
>>
✓ Initialized. View run at https://modal.com/apps/khatiketki/main/ap-UeGjvSbwWt5fcRIGAeOskV
✓ Created objects.
└─ ↴ Created mount C:\Users\khati\Documents\Ai\modal_resources.py
    └─ ↴ Created function resource_demo.

Resources demo running.
Doing a tiny bit of work...
Done. checksum = 1499994 elapsed = 0.023 sec
Stopping app - local entrypoint completed.
✓ App completed. View run at https://modal.com/apps/khatiketki/main/ap-UeGjvSbwWt5fcRIGAeOskV
```