ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KĨ THUẬT MÁY TÍNH

BÁO CÁO LAB 3: WORK POOL

MÔN HỌC: TÍNH TOÁN SONG SONG

GVCĐ: La Quốc Nhựt Huân

—o0o—

SVTH: Nguyễn Phúc An (2210022)

TP. HỒ CHÍ MINH, THÁNG 10 NĂM 2025

Mục lục

1 Sequential (Single-thread)

1.1 Mục tiêu

Phiên bản tuần tự có mục tiêu mô phỏng sự lan truyền của sóng xung kích (shock wave) từ một vụ nổ được đặt tại tâm của lưới kích thước $N \times N$. Mỗi ô đại diện cho diện tích 10×10 (m). Ở mỗi thời điểm t, thuật toán xác định những ô mà sóng đã lan tới và tính giá trị *peak overpressure* P_{so} theo mô hình Kingery–Bulmash.

1.2 Mô tả thuật toán

Thuật toán gồm các bước chính:

1. Xác định tâm vụ nổ:
$$(center\_x, center\_y) = \left(\frac{N}{2}, \frac{N}{2}\right)$$
2. Lặp theo thời gian $t = 1 \ldots SECONDS$:
   - Tính bán kính lan truyền của sóng:
   $$R_{\max}(t) = 343 \cdot t$$
   (343 m/s là vận tốc âm thanh.)
   - Đổi sang đơn vị ô lưới:
   $$\text{max\_cells} = \frac{R_{\max}}{10}$$
   - Giới hạn chỉ duyệt trong một hình vuông xung quanh tâm:
   $$i \in [center\_x - \text{max\_cells},\ center\_x + \text{max\_cells}]$$
   $$j \in [center\_y - \text{max\_cells},\ center\_y + \text{max\_cells}]$$
   - Duyệt từng ô (i, j) trong vùng này.
   - Với mỗi ô:
     - Nếu đã tính trước đó ($res[iN + j] > 0$) thì bỏ qua.
     - Tính khoảng cách thực tế từ tâm tới ô:
     $$dx = (i - center\_x) \cdot 10, \qquad dy = (j - center\_y) \cdot 10$$
     $$R = \sqrt{dx^2 + dy^2}$$
     - Nếu $R \le R_{\max}(t)$ thì sóng đã đến ô đó, tiến hành tính:
       * Scaled distance:
       $$Z = R \cdot W^{-1/3}$$
       * Biến trung gian:
       $$U = -0.21436 + 1.35034 \log_{10}(Z)$$
       * Giá trị logarit của áp suất:
       $$\log_{10}(P_{so}) = \sum_{k=0}^{8} C_k U^k$$
       * Peak overpressure:
       $$P_{so} = 10^{\log_{10}(P_{so})}$$
     - Ghi kết quả vào:
     $$res[iN + j] = P_{so}, \quad A[iN + j] = P_{so}$$

1.3 Mã giả thuật toán

```
center_x = N / 2
   center_y = N / 2

   for t = 1 to SECONDS:

       R_max = 343.0 * t           // bán kính sóng nổ tại thời điểm t
       max_cells = R_max / 10.0

       // Giới hạn vùng ô cần tính trong ma trận
       min_i = max(0, center_x - max_cells)
       max_i = min(N - 1, center_x + max_cells)
       min_j = max(0, center_y - max_cells)
       max_j = min(N - 1, center_y + max_cells)

       for i = min_i to max_i:
           for j = min_j to max_j:

               // Bỏ qua ô đã tính rồi (tức là trogn trường hợp này sẽ là > 0)
               if res[i*N + j] > 0:
                   continue

               dx = (i - center_x) * 10.0
               dy = (j - center_y) * 10.0
               R = sqrt(dx*dx + dy*dy)

               // Tránh chia cho 0
               if R < 1e-9:
                   continue

               if R <= R_max:
                   Z = R * W^(-1/3)
                   U = -0.21436 + 1.35034 * log10(Z)

                   logPso = 0
                   Ui = 1.0

                   for k = 0 to 8:
                       logPso = logPso + C[k] * Ui
                       Ui = Ui * U

                   Pso = 10^logPso

                   res[i*N + j] = Pso
                   A[i*N + j]   = Pso
```

Algorithm 1: Sequential Shock Wave Simulation

1.4 Độ phức tạp tính toán

Thuật toán `Sequential Shock Wave Simulation` mô phỏng sự lan truyền của sóng nổ qua lưới $N \times N$ trong *SECONDS* bước thời gian. Tại mỗi thời điểm t, chỉ các ô nằm trong bán kính $R_{\max}(t) = 343t$ mới được tính toán; các ô khác được bỏ qua.

Độ phức tạp thời gian

Với:

$$R_{\max}(t) = 343t \quad (\text{m}), \qquad \text{mỗi ô} = 10\ \text{m}, \qquad \text{max\_cells}(t) = \frac{R_{\max}(t)}{10} = 34.3t.$$

Gọi $S = SECONDS$.

Số ô cần duyệt tại thời điểm t Thuật toán giới hạn vùng kiểm tra bằng một hình vuông có nửa cạnh $\text{max\_cells}(t)$, do đó số ô trong hình vuông xấp xỉ

$$\text{cells\_square}(t) \approx \big(2 \cdot \text{max\_cells}(t)\big)^2 = O(t^2).$$

Vì vậy chi phí duyệt tại thời điểm t là $O(t^2)$ (mỗi ô bị duyệt ít nhất 1 kiểm tra).

Tổng số lần duyệt ô (iteration checks) trong toàn mô phỏng Tổng số lần duyệt (tức số lần lặp qua các ô trong các bước t) là:

$$\sum_{t=1}^{S} O(t^2) = O\left(\sum_{t=1}^{S} t^2\right).$$

Sử dụng công thức tổng bình phương:

$$\sum_{t=1}^{S} t^2 = \frac{S(S+1)(2S+1)}{6} = O(S^3).$$

Vì vậy, nếu ta quan tâm đến *tổng số lần duyệt ô* (kể cả các lần kiểm tra nhanh 'if res[...] > 0'), thì

$$\boxed{\text{Total iterations} = O(S^3).}$$

Thời điểm "phủ kín toàn bộ lưới" (t_{full}) Khoảng cách lớn nhất từ tâm đến một góc lưới (theo mét) là:

$$R_{\text{corner}} = \frac{N}{2}\sqrt{2} \cdot 10.$$

Số ô *thực sự* tính (heavy computations) — mỗi ô chỉ cần tính P_{so} một lần Vì trong code có điều kiện 'if (res[i*N+j] > 0) continue;' (các ô đã được tính sẽ không bị tính lại), nên số lượng phép toán nặng (tính đa thức và luỹ thừa để ra P_{so}) bị giới hạn bởi tổng số ô trong lưới:

$$\boxed{\text{Total heavy computations} \le N^2.}$$

Thực tế mỗi ô sẽ được tính đúng một lần — tại thời điểm t đầu tiên sao cho $R_{\max}(t) \ge R(i,j)$ — vì sau đó res[i*N+j] > 0 và các bước sau bỏ qua.

Độ phức tạp không gian

- Thuật toán sử dụng hai mảng kích thước $N \times N$:

$$A,\ res \quad \Rightarrow \quad O(N^2).$$

- Vector hệ số C dài 9 và các biến trung gian là hằng số:

$$O(1).$$

$$S(N) = O(N^2).$$

Kết luận

- Thời gian:

$$O(SECONDS^3)$$

- Không gian:

$$O(N^2).$$

- Thuật toán có thể song song hóa trên miền (i,j) của từng bước t.

2 Parallel

2.1 Mục tiêu

Mục tiêu của phần hiện thực song song là tăng tốc quá trình mô phỏng sự lan truyền áp suất trên lưới kích thước $N \times N$ theo thời gian.

Bài toán yêu cầu tính toán giá trị áp suất tại mỗi ô (i, j) dựa trên khoảng cách đến tâm lưới và các tham số vật lý cho từng thời điểm t.

Giải pháp sử dụng mô hình thread work pool với hàng đợi tác vụ (Task Queue). Mỗi tác vụ (Task) phụ trách một dải hàng (row range) trên ma trận. Các worker thread liên tục lấy tác vụ từ hàng đợi và xử lý độc lập, giúp tận dụng tối đa tài nguyên CPU đa lõi.

2.2 Mô tả thuật toán

Thuật toán song song sử dụng mô hình thread pool gồm nhiều worker threads và một hàng đợi nhiệm vụ (TaskQueue). Quá trình tính toán được chia thành ba giai đoạn chính như sau.

1. **Sinh tác vụ (Task Generation)**
   Ở mỗi thời điểm t, sóng lan truyền ra xa với vận tốc 343 m/s nên bán kính cực đại đạt được là:

$$R_{\max} = 343 \cdot t$$

   Vì lưới được rời rạc hoá theo bước 10 cm giữa các ô, ta xác định những hàng có thể bị ảnh hưởng bằng cách tính:

$$\text{min\_i} = \max\left(0,\ center - \frac{R_{\max}}{10}\right)$$

$$\text{max\_i} = \min\left(N-1,\ center + \frac{R_{\max}}{10}\right)$$

   Tuy nhiên, không phải mọi hàng trong khoảng $[min_i, max_i]$ đều chứa điểm hợp lệ thuộc vòng tròn bán kính $R_{\max}$. Để kiểm tra, thuật toán quét từng hàng i và tính **mức lan rộng theo trục ngang**:

$$dy_{\max} = \frac{1}{10}\sqrt{R_{\max}^2 - (i - center)^2 \cdot 100}$$

   Nhờ đó ta xác định được:

$$min_j = center - dy_{\max}, \qquad max_j = center + dy_{\max}$$

   Nếu tồn tại ít nhất một hàng có $min_j \le max_j$, nghĩa là vùng sóng tại thời điểm t thật sự cắt qua ma trận, khi đó hệ thống tạo đúng **một tác vụ duy nhất**:

$$\texttt{Task}(task\_id,\ start = min_i,\ end = max_i)$$

   Tác vụ này bao gồm toàn bộ các hàng bị ảnh hưởng trong thời điểm t, và được đưa vào hàng đợi bằng:

```
TaskQueue::enqueue(new Task(...))
```

   Vì vậy, mỗi giây chỉ sinh đúng 1 task: Điều này là hợp lý vì sóng lan truyền theo dạng hình tròn, và tại mỗi thời điểm chỉ có duy nhất một dải hàng liên tục chứa điểm hợp lệ.

2. **Xử lý tác vụ trong Worker Thread**
   Mỗi worker chạy trong một vòng lặp vô hạn, liên tục lấy tác vụ từ hàng đợi:

   - Nếu hàng đợi có tác vụ, worker lấy ra và gọi hàm `process()`.
   - Nếu hàng đợi trống nhưng hệ thống chưa gửi tín hiệu kết thúc, worker tạm nghỉ 1 ms rồi thử lại.
   - Nếu hàng đợi trống và `finished = true`, worker dừng hoạt động.

Bên trong hàm `process()`, worker duyệt qua tất cả các ô (i, j) trong dải hàng:

$$i = \text{start\_row} \to \text{end\_row}, \qquad j = 0 \to N-1$$

Với mỗi ô, worker thực hiện:

(a) Nếu ô đã được tính từ thời điểm trước, bỏ qua ngay.

(b) Tính khoảng cách đến tâm:

$$R = \sqrt{(i-center)^2 + (j-center)^2} \cdot 10$$

(c) Nếu $R > R_{\max}$ hoặc gần bằng 0, bỏ qua ô.

(d) Tính các giá trị trung gian:

$$Z = R \cdot W^{-1/3}$$

$$U = -0.21436 + 1.35034 \log_{10}(Z)$$

(e) Tính giá trị áp suất:

$$\log P_{so} = \sum_{k=0}^{8} C_k U^k$$

$$P_{so} = 10^{\log P_{so}}$$

(f) Ghi kết quả vào hai ma trận `A` và `res`.

Do mỗi tác vụ xử lý một dải hàng không chồng lấp, các worker chạy độc lập, không cần đồng bộ hoá phức tạp.

3. **Kết thúc chương trình**

Sau khi vòng lặp thời gian kết thúc và tất cả tác vụ được sinh ra, hàm:

`TaskQueue::setFinished()`

được gọi để báo hiệu rằng không còn tác vụ mới nào.

Các worker xử lý nốt tác vụ còn lại trong hàng đợi, sau đó tự động thoát vòng lặp và kết thúc.

Cuối cùng, chương trình chờ (join) tất cả worker và giải phóng bộ nhớ.

2.3 Mã giả thuật toán

```
STRUCT Task:
    id          : integer
    start_row   : integer
    end_row     : integer
    t           : integer
    A           : pointer to array of double
    res         : pointer to array of double
    W           : double
    C           : pointer to array of double

    CONSTRUCTOR Task(id, s, e, A, res, W, C, t):
        self.id        = id
        self.start_row = s
        self.end_row   = e
        self.A         = A
        self.res       = res
        self.W         = W
        self.C         = C
        self.t         = t

CLASS TaskQueue:
    PRIVATE:
```

```
    mtx      : mutex
    queue    : queue of Task*
    instance : static TaskQueue*
    finished : boolean = false

PUBLIC:

    FUNCTION enqueue(task):
        lock(mtx)
        queue.push(task)
        unlock(mtx)

    FUNCTION dequeue() RETURNS Task*:
        lock(mtx)
        IF queue is empty THEN
            unlock(mtx)
            RETURN null
        ENDIF
        task = queue.front()
        queue.pop()
        unlock(mtx)
        RETURN task

    STATIC FUNCTION get() RETURNS TaskQueue*:
        IF instance is null THEN
            instance = new TaskQueue()
        ENDIF
        RETURN instance

    FUNCTION isEmpty() RETURNS boolean:
        lock(mtx)
        result = (queue is empty)
        unlock(mtx)
        RETURN result

    FUNCTION setFinished():
        lock(mtx)
        finished = true
        unlock(mtx)

    FUNCTION isFinished() RETURNS boolean:
        lock(mtx)
        result = finished
        unlock(mtx)
        RETURN result
```

```
for t = 1 → SECONDS:
   Rmax = 343 * t
   start = max(0, center - Rmax/10)
   end   = min(N-1, center + Rmax/10)

   if (start  end):
       task = new Task(start, end, A, res, W, C, t)
       TaskQueue.enqueue(task)
```

Algorithm 2: Phần thêm task

```
function process(task):
    for i = start_row → end_row:
        for j = 0 → N-1:
            if res[i*N + j] > 0: continue

            dx = (i - center) * 10
            dy = (j - center) * 10
            R  = sqrt(dx*dx + dy*dy)

            if R <= 0 or R > Rmax: continue

            Z = R * W^(-1/3)
            U = -0.21436 + 1.35034 * log10(Z)

            logPso =  (C[k] * U^k)
            Pso    = 10^(logPso)

            res[i*N + j] = Pso
            A[i*N + j]   = Pso
```

Algorithm 3: Hàm xử lý task được phân công từng thread

```
function Worker.run():
    loop:
        task = TaskQueue.dequeue()
        if task != nullptr:
            process(task)
            delete task
        else if TaskQueue.isFinished():
            break
        else:
            sleep(1 ms)
```

Algorithm 4: Phần chạy task

2.4 Độ phức tạp tính toán

Thuật toán song song mô phỏng lan truyền sóng nổ trên lưới $N \times N$ với *SECONDS* bước thời gian. Cơ chế song song dựa trên nhiều worker threads lấy tác vụ từ `TaskQueue` và xử lý dải hàng liên tục.

Độ phức tạp thời gian

- Mỗi thời điểm t, thuật toán chỉ tạo một tác vụ duy nhất bao gồm các hàng bị ảnh hưởng bởi bán kính

$$R_{\max}(t) = 343 \cdot t$$

- Mỗi worker duyệt tất cả các ô (i, j) trong dải hàng của tác vụ. Mỗi ô chỉ thực hiện các phép tính nặng một lần nhờ điều kiện:

`if(res[i*N+j] > 0) continue;`

- Giả sử ta có p worker, nếu bỏ qua chi phí đồng bộ, mỗi worker xử lý xấp xỉ $1/p$ tổng số ô, do đó chi phí tính toán giảm tương ứng:

$$O\left(\frac{S^3}{p}\right)$$

- Tuy nhiên, chi phí thực tế tăng thêm do:
  - Truy cập hàng đợi dùng `mutex` → overhead đồng bộ.

- Worker tạm nghỉ 1 ms khi hàng đợi rỗng (`sleep(1ms)`).
- Không sử dụng `condition_variable` → polling loop tốn thời gian.

Độ phức tạp không gian

- Hai ma trận A và `res` kích thước $N \times N$:
$$O(N^2)$$
- Vector hệ số C và các biến trung gian: $O(1)$
- Hàng đợi tác vụ (TaskQueue) chứa tối đa $SECONDS$ task: $O(S)$
- Mỗi worker có một thread riêng, overhead bộ nhớ cho thread stack là hằng số: $O(p)$

$$S(N) = O(N^2)$$

Kết luận

- Thời gian lý thuyết: $O(N^2/p)$ nếu p thread cân bằng tải hoàn hảo.
- Không gian: $O(N^2 + S + p) \approx O(N^2)$
- Thuật toán song song hóa dựa trên dải hàng trong Task và có thể mở rộng bằng cách tạo ít task lại để tăng khả năng cân bằng tải.

3 Mô phỏng và đánh giá kết quả

Cách sử dụng, mô phỏng toàn chương trình LAB3:

- make build → Biên dịch (build) chương trình.

- make run → Chạy chương trình.

```
Time to sequential: 2.050263 s
Worker 8 executes task 0
Worker 9 executes task 1
Worker 0 executes task 2
Worker 1 executes task 3
Worker 3 executes task 4
Worker 2 executes task 5
Worker 4 executes task 6
Worker 5 executes task 7
Worker 6 executes task 8
Worker 7 executes task 9
Worker 8 executes task 10
Worker 9 executes task 11
Worker 0 executes task 12
Worker 1 executes task 13
Worker 3 executes task 14
Worker 2 executes task 15
Worker 4 executes task 16
Worker 5 executes task 17
Worker 6 executes task 18
Worker 7 executes task 19
Worker 8 executes task 20
Worker 9 executes task 21
Worker 0 executes task 22
Worker 1 executes task 23
Worker 3 executes task 24
```

```
Worker 2 executes task 25
Worker 4 executes task 26
Worker 5 executes task 27
Worker 6 executes task 28
Worker 7 executes task 29
Worker 8 executes task 30
Worker 9 executes task 31
Worker 0 executes task 32
Worker 1 executes task 33
Worker 3 executes task 34
Worker 2 executes task 35
Worker 4 executes task 36
Worker 5 executes task 37
Worker 6 executes task 38
Worker 7 executes task 39
Worker 8 executes task 40
Worker 9 executes task 41
Worker 0 executes task 42
Worker 1 executes task 43
Worker 3 executes task 44
Worker 2 executes task 45
Worker 4 executes task 46
Worker 5 executes task 47
Worker 6 executes task 48
Worker 7 executes task 49
Worker 8 executes task 50
Worker 9 executes task 51
Worker 0 executes task 52
Worker 1 executes task 53
Worker 3 executes task 54
Worker 2 executes task 55
Worker 4 executes task 56
Worker 5 executes task 57
Worker 6 executes task 58
Worker 7 executes task 59
Worker 8 executes task 60
Worker 9 executes task 61
Worker 0 executes task 62
Worker 1 executes task 63
Worker 3 executes task 64
Worker 2 executes task 65
Worker 4 executes task 66
Worker 5 executes task 67
Worker 6 executes task 68
Worker 1 executes task 73
Worker 8 executes task 70
Worker 7 executes task 69
Worker 9 executes task 71
Worker 0 executes task 72
Worker 3 executes task 74
Worker 2 executes task 75
Worker 4 executes task 76
Worker 5 executes task 77
Worker 6 executes task 78
Worker 7 executes task 79
Worker 8 executes task 80
Worker 9 executes task 81
Worker 0 executes task 82
Worker 1 executes task 83
Worker 6 executes task 87
```

```
Worker 0 executes task 88
Worker 5 executes task 89
Worker 1 executes task 90
Worker 7 executes task 91
Worker 9 executes task 92
Worker 3 executes task 93
Worker 8 executes task 94
Worker 2 executes task 95
Worker 5 executes task 96
Worker 7 executes task 97
Worker 6 executes task 98
Worker 3 executes task 99
Time to parallel: 0.495748 s
Check matrix equal successfully
```

Kết quả chạy chương trình trên lưới $N \times N$ với 100 bước thời gian (100 task) và 10 worker threads:

Nhận thấy ma trận kết quả song song và tuần tự trùng khớp.

3.1 Đánh giá hiệu suất

Tốc độ tăng (Speedup):

$$S = \frac{T_{\text{seq}}}{T_{\text{par}}} = \frac{2.050263}{0.495748} \approx 4.14$$

Hiệu quả sử dụng luồng (Efficiency):

$$E = \frac{S}{P} = \frac{4.14}{10} \approx 0.414$$

với $P = 10$ là số worker threads.

3.2 Nhận xét

- Thuật toán song song chạy nhanh hơn so với phiên bản tuần tự nhờ chia các dải hàng của ma trận vào từng task và các worker xử lý đồng thời.
- Mặc dù có 10 worker, thời gian song song không giảm đúng T_{seq}/P do một số lý do:
  - Chi phí đồng bộ khi truy cập `TaskQueue` (mutex) làm worker đôi khi phải chờ.
  - Chi phí tạo và hủy worker.
  - Worker tạm nghỉ 1 ms khi hàng đợi rỗng (`sleep(1ms)`), gây overhead.
  - Số task (100) tương đối lớn so với số worker (10), nên khoảng thời gian tạo task và thêm task sẽ chiếm khá nhiều thời gian
  - Không cân bằng hoàn hảo: một số task nặng hơn (dải hàng lớn hơn) vì phải duyệt ở thời gian càng tăng gây chênh lệch thời gian giữa các worker.
- Tổng thể, thuật toán vẫn đạt được xấp xỉ 4.1 và efficiency xấp xỉ 41% — chấp nhận được cho số lượng task và phương pháp polling hiện tại.

3.3 Kết luận

Thuật toán song song giúp giảm đáng kể thời gian tính toán so với phiên bản tuần tự, nhưng để đạt hiệu quả gần tối đa cần:

- Giảm task nhỏ hơn để cân bằng tải giữa worker.
- Giảm chi phí đồng bộ (mutex hoặc polling) bằng các cơ chế thread-safe hiệu quả hơn.