# Lưu đồ thuật toán

| **Tuần tự** | **Song song hóa** |
| --- | --- |
|  |  |

# 

# Hướng tiếp cận: **Data-parallelism**

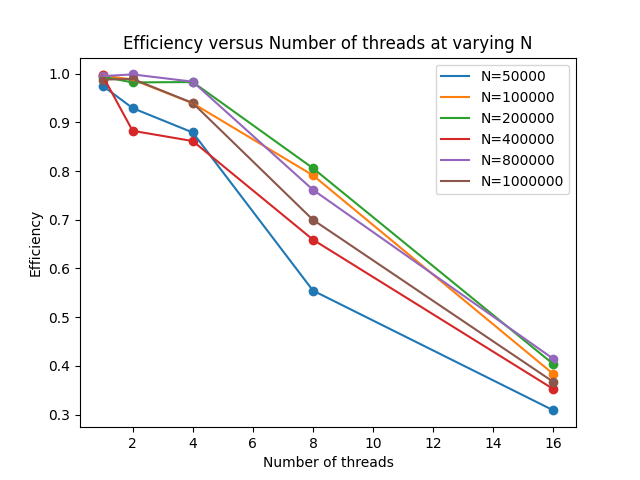
* N điểm được chia đều theo số lượng của threads (*num\_threads).* Trong trường hợp chia không hết, phần còn lại sẽ được phân bổ vào thread cuối cùng (1)
* **Điểm chính trong thuật toán song song hóa**
  + **Initialization:** K điểm đầu tiên sẽ được chọn làm centroids bắt đầu
  + **Data-parallelism:​** Mỗi thread sẽ được phân bổ N/num\_threads điểm
  + **Thread function:** Mỗi thread sẽ chạy một vòng lặp (giá trị max\_iter là 100). Trong mỗi vòng lặp, mỗi thread sẽ tìm các centroids gần nhất cho mỗi điểm được phân bố, sau đấy phân bố điểm vào các cụm. Sau khi tất cả các điểm được phân bố vào cụm, centroids\_global sẽ được cập nhật bằng cách lấy trung bình tọa độ các điểm trong từng centroids
  + **Stopping-condition: ​**L2-norm được dùng để tính tổng khoảng cách giữa các điểm với centroids của chúng để so sánh với giá trị ở lần iteration trước. Ngoài ra, cũng được so sánh với giá trị threshold (1e-6). Vòng lặp sẽ dừng khi giá trị delta thấp hơn giá trị threshold hoặc hết max\_iter.
  + ​#pragma omp critical: Được dùng cho “thread synchronization", đảm bảo tất cả các thread cập nhật cùng centroids
  + #pragma omp barrier: Được dùng cho “progress synchronization", đảm bảo các thread đều hoàn thành tính toán trước khi chuyển sang step tiếp theo.

## 

## Đánh giá

### Efficiency: |

* Kích thước data: sizes = [50000, 100000, 200000, 400000, 800000, 1000000]
* Số lượng threads: threads = [1, 2, 4, 8, 16]
* Đánh giá: Nhìn chung thì efficiency giảm khi số core càng tăng. Ngoài ra cũng có thể thấy rằng, dataset càng lớn thì efficiency nhìn chung sẽ cao hơn.



### 

### Speedup:

* Kích thước data: sizes = [100000]
* Số lượng threads: threads = [1, 2, 4, 8, 16]
* Đánh giá: Nhìn chung, số core càng nhiều, tỉ lệ tăng tốc càng lớn.

