BÁO CÁO TỔNG HỢP - HIỆU SUẤT NHÂN MA TRẬN SONG SONG VỚI STRASSEN ALGORITHM

CS401V - Distributed Systems Assignment 1
Nhóm: Phan Văn Tài (2202081) & Hà Minh Chiến (2202095)

TÓM TẮT DỰ ÁN

Mục tiêu nghiên cứu

Nghiên cứu và so sánh hiệu suất của **Strassen Algorithm** trong việc nhân ma trận song song, bao gồm: - Phân tích hiệu suất Strassen Algorithm O(n^log₂7) - Đánh giá tác động của song song hóa (Parallel Row vs Parallel Element) - Tối ưu hóa số lượng tiến trình cho từng kích thước ma trận - So sánh với lý thuyết và thực nghiệm

Phương pháp nghiên cứu

- **Thuật toán**: Strassen Algorithm với threshold cho ma trận nhỏ
- **Song song hóa**: Process-based parallelization với fork(), mmap(), semaphores
- **Benchmark (gốc)**: Ma trận 4×4 đến 1024×1024; processes 10, 100, 1000
- **Dữ liệu mở rộng**: Bổ sung các kích thước lớn (1536→6144) và dải processes đa dạng (32→2000) để so sánh thời gian tốt nhất giữa phương pháp Row/Element
- **Reproducibility**: Fixed seed (12345) cho kết quả nhất quán

Hệ thống thử nghiệm

- **OS**: Linux 6.8.0-85-generic
- **CPU**: Multi-core processor (8+ cores)
- **RAM**: 8GB+ (đủ cho ma trận 1024×1024)
- **Compiler**: GCC với flags -O2
- **Libraries**: pthread, math (-lm)
- **Memory**: Shared memory với mmap() MAP_SHARED

KẾT QUẢ CHÍNH (CẬP NHẬT THEO DỮ LIỆU MỞ RỘNG ĐẾN 6144)

Hiệu suất Strassen Algorithm (≤1024 có baseline tuần tự)

Matrix Size	Sequential (μs)	Best Parallel	Speedup	Optimal Processes
256×256	11,463	2,352 (Row)	4.87x	10
512×512	75,109	28,016 (Row)	2.68x	10
1024×1024	540,443	323,885 (Row)	1.67x	1000

Kết quả mở rộng (≥1536 không có baseline tuần tự − so sánh thời gian tốt nhất)

Matrix Size	Best Time (s)	Method	Processes
1536×1536	2.802	Parallel Row	1024

Matrix Size	Best Time (s)	Method	Processes
2048×2048	8.833	Parallel Element	32
2560×2560	18.607	Parallel Element	32
3072×3072	35.804	Parallel Element	128
3584×3584	63.007	Parallel Element	128
4096×4096	105.498	Parallel Element	128
5120×5120	299.282	Parallel Element	2000
6144×6144	547.510	Parallel Element	512

Phân tích quan trọng (cập nhật)

1. **Strassen hiệu quả**: Từ 256×256 trở lên, đạt lợi ích rõ rệt so với kích thước rất nhỏ.
2. **Tối ưu theo vùng kích thước**:
   - **≤1024**: Parallel Row thường tốt hơn Parallel Element và cho speedup đáng kể so với baseline tuần tự.
   - **≥1536**: Dữ liệu mở rộng cho thấy Parallel Element cho thời gian tốt hơn Parallel Row ở đa số kích thước lớn (trừ 1536, nơi Parallel Row tốt nhất).
3. **Số tiến trình tối ưu (theo dữ liệu)**:
   - **256–512**: ~10–32 processes (Row)
   - **1024**: ~100–1000 processes (Row)
   - **≥1536**: 32–256 processes cho Parallel Element (một số trường hợp tốt nhất ở 128), riêng 5120 tốt nhất ở 2000 processes
4. **Threshold quan trọng**: <64×64 nên dùng tuần tự do overhead.

BIỂU ĐỒ HIỆU SUẤT

1. Thời gian thực thi

- **Sequential**: Tuân theo $O(n^{\log_2 7}) \approx O(n^{2.81})$
- **Parallel Row**: Hiệu quả hơn ở ≤1024 (có speedup so với tuần tự)
- **Parallel Element**: Ở ≥1536 thường cho thời gian tốt hơn Row trong dữ liệu mở rộng

2. Speedup Analysis

- **Tối đa (≤1024)**: 4.87x với 256×256, 10 tiến trình (Row)
- **Cảnh báo**: Với kích thước ≥1536 không có số liệu tuần tự tương ứng → không báo cáo speedup, chỉ so sánh thời gian tốt nhất giữa các phương pháp.

3. Process Count Optimization (cập nhật từ dữ liệu)

- **Ma trận nhỏ (≤128×128)**: Overhead cao khi tăng tiến trình.
- **256×256–512×512**: ~10–32 tiến trình (Row) cho thời gian tốt nhất.
- **1024×1024**: 100–1000 tiến trình (Row) cho thời gian tốt nhất.
- **≥1536**: Parallel Element thường tối ưu ở 32–256 tiến trình (điển hình 128). Ngoại lệ: 5120×5120 tốt nhất ở 2000 tiến trình.

4. Memory Usage Analysis (chỉ báo tính xu hướng)

Matrix Size	Parallel Best (ms)	Ghi chú
256×256	2.352	Row, 10 tiến trình
512×512	28.016	Row, 10 tiến trình
1024×1024	323.885	Row, 1000 tiến trình
2048×2048	8832.631	Element, 32 tiến trình (không có baseline tuần tự)

5. Theoretical vs Practical Performance (≤1024)

Matrix Size	Theoretical Speedup	Practical Speedup	Efficiency
256×256	7.0x	4.87x	69.6%
512×512	5.0x	2.68x	53.6%
1024×1024	3.5x	1.67x	47.7%

Lưu ý: Không tính efficiency cho ≥1536 do thiếu baseline tuần tự tương ứng.

PHÂN TÍCH KỸ THUẬT

Strassen Algorithm Implementation

- **Recursive approach**: Chia ma trận thành 4 submatrices
- **7 multiplications**: Tối ưu hóa so với phương pháp truyền thống
- **Threshold**: 64×64 cho ma trận nhỏ (chuyển sang phương pháp khác)
- **Memory management**: Padding cho ma trận không phải lũy thừa của 2

Parallelization Strategy

- **Work-stealing**: Dynamic load balancing với shared indices
- **Memory sharing**: mmap() với MAP_SHARED
- **Synchronization**: Semaphores cho shared variables
- **Process management**: fork(), wait(), _exit()

Performance Bottlenecks

1. **Memory bandwidth**: Giới hạn với ma trận rất lớn
2. **Process overhead**: Nhiều tiến trình → context switching
3. **Cache efficiency**: Strassen có cache locality cần tối ưu hóa
4. **Synchronization cost**: Semaphore operations

KẾT LUẬN VÀ KHUYẾN NGHỊ

Kết luận chính (cập nhật)

1. **Strassen Algorithm hiệu quả**: Với ma trận ≥256×256.
2. **Chiến lược tối ưu phụ thuộc kích thước**: Parallel Row tốt hơn ở ≤1024; Parallel Element cho thời gian tốt hơn ở ≥1536 (trừ 1536).
3. **Số tiến trình tối ưu**: 10–32 (256–512, Row), 100–1000 (1024, Row), 32–256 (≥1536, Element; ngoại lệ 5120 cần 2000).

4. **Threshold**: <64×64 nên dùng tuần tự.

Khuyến nghị thực tế

- ➢ **Cho ma trận nhỏ (≤64×64):** - Sử dụng sequential Strassen - Tránh song song hóa do overhead
- ➢ **Cho ma trận trung bình (128×128-512×512):** - Sử dụng parallel row với 10-100 tiến trình - Tránh quá nhiều tiến trình
- ➢ **Cho ma trận lớn (≥1536×1536):** - Ưu tiên parallel element với 32–256 tiến trình (thử 128 trước); riêng 5120×5120 có thể cần cao hơn (≈2000). - Cân nhắc giới hạn băng thông bộ nhớ.

Hướng phát triển

1. **Hybrid approach**: Kết hợp Strassen cho ma trận lớn và phương pháp khác cho ma trận nhỏ
2. **Memory optimization**: Tối ưu hóa memory usage cho ma trận rất lớn
3. **Load balancing**: Cải thiện work distribution trong parallel element
4. **GPU acceleration**: Thử nghiệm Strassen trên GPU

TÀI LIỆU THAM KHẢO

- Strassen, V. (1969). “Gaussian elimination is not optimal”
- Cormen, T. H. et al. (2009). “Introduction to Algorithms”
- Parallel Computing: Principles and Practice
- CS401V - Distributed Systems Course Materials