# Parallel Execution Analysis: Matrix Multiplication vs. Factorial Computation

**Compare a two-dimensional matrix multiplication program with a factorial computation program in terms of their parallelization potential. In your answer, discuss the following points:**

1. **Task Decomposition**

**Explain whether the task division (decomposition) is partial or complete for each program.**

* 1. **Two-Dimensional Matrix Multiplication Program**

**Type: Complete Decomposition using Data Decomposition**



In matrix multiplication, the computation of each element (or block) of the result matrix **C = A × B** is independent. Each element C[i][j] can be computed as a separate task without waiting for other elements. This allows:

* **Fine-grain decomposition** where each iteration (element) runs independently
* **Maximum parallelism = n²** for an n×n matrix
* No dependencies between tasks computing different result elements

Matrix multiplication achieves **complete task decomposition** — all work units can execute concurrently with minimal dependencies.

* 1. **Factorial Computation Program**

**Type: Partial (Minimal) Decomposition - Sequential Chain**

Factorial computation follows the pattern: **n! = n × (n-1) × (n-2) × ... × 1**



Each multiplication in factorial depends on the previous result, creating a **serial dependency chain**:

* fact(n) requires fact(n-1)
* fact(n-1) requires fact(n-2)
* And so on...

This forms a **critical path equal to the entire computation**, with no independent branches for parallel execution.

Factorial has **partial (extremely limited) decomposition** — tasks cannot be divided into independent concurrent units due to strict sequential dependencies.

1. **Scalability and Efficiency**

**Discuss how much each program can benefit from parallel execution in terms of scalability and efficiency.**

**Note:**

**• Scalability refers to the program’s ability to achieve faster performance as more processors or threads are added.**

**• Efficiency refers to how effectively the program utilizes available processors or threads to achieve speedup during parallel execution.**

* 1. **Two-Dimensional Matrix Multiplication**

**Scalability: HIGH** ✓



With data decomposition into blocks:

* Adding more processors directly reduces execution time
* Can utilize hundreds or thousands of processors effectively
* Limited mainly by overhead costs (synchronization, communication)

**Efficiency: HIGH** ✓

Efficiency measures how well processors are utilized:



Matrix multiplication maintains high efficiency because:

* Independent tasks keep all processors busy
* Coarse-grain blocking minimizes overhead
* Little idle time or load imbalance
* High percentage of theoretical speedup achieved

With proper granularity, parallel programs can achieve near-linear speedup, maintaining efficiency close to 100% until communication overhead dominates.

* 1. **Factorial Computation**

**Scalability: VERY LOW** ✗



Factorial's sequential chain means:

* Adding processors provides **no benefit**
* Execution time remains constant (equal to critical path)
* Maximum concurrency = **1 processor at a time**
* Cannot reduce wall-clock time regardless of available processors

**Efficiency: VERY LOW** ✗



Factorial shows poor efficiency because:

* Most processors remain **idle** (unused)
* Efficiency approaches **0%** as processor count increases
* Overhead costs (thread creation, synchronization) exceed any benefit
* **Work inflation** — parallel version may be slower than serial

**Example:** With 8 processors attempting to compute factorial, only 1 can work at any time → Efficiency = (1/8) × 100% = **12.5%** maximum

3. **Why Factorial Shows Little or No Parallel Improvement**

Limited Parallel Improvement in Factorial Computation

Explain why the factorial computation program shows little or no performance improvement in parallel execution compared to the serial version.

factorial computation fails to benefit from parallelization due to:

* 1. **Critical Path Dominance**

Factorial's **entire computation is the critical path**:

* + - Every multiplication depends on the previous result
    - Length of critical path = total work
    - **Speedup upper bound = Total Work ÷ Critical Path ≈ 1.0**
    - No exploitable parallelism exists
  1. **Sequential Dependencies**



Factorial violates the independence requirement:

* + - Task dependency graph forms a **linear chain**
    - No concurrent branches possible
    - Maximum concurrency = **1 task at a time**
    - Falls under **sequential execution** pattern where only one task runs at any moment
  1. **Overhead Exceeds Benefit**



Parallel factorial incurs:

* + - **Work inflation** — duplicating code/data across threads
    - **Coordination costs** — barriers, locks, atomic operations
    - Thread management overhead
    - Context switching penalties

Since no actual parallel work occurs, **overhead dominates**, making the parallel version

**slower than serial**

* 1. **Amdahl's Law Limitation**



Factorial has a **serial fraction approaching 100%**:

* + - Even with infinite processors, speedup remains ≈ 1.0
    - The sequential chain cannot be parallelized
    - Fundamental algorithmic constraint, not implementation issue

1. **Comparison Table**

| **Aspect** | **Matrix Multiplication** | **Factorial Computation** |
| --- | --- | --- |
| **Decomposition Type** | Complete / Data Decomposition | Partial (Minimal) / Sequential Chain |
| **Independence** | High - independent blocks/elements | None - strict sequential dependencies |
| **Parallelism** | Maximum = n² (fine-grain) or blocks (coarse-grain) | Maximum = 1 (no concurrent tasks) |
| **Critical Path** | Short (few operations per block) | Long (entire computation) |
| **Scalability** | * HIGH - scales with processors | ✗ VERY LOW - no scaling benefit |
| **Efficiency** | * HIGH - processors well utilized | ✗ VERY LOW - most processors idle |
| **Speedup Potential** | Near-linear with proper granularity | ≈ 1.0 (no speedup) |
| **Overhead Impact** | Manageable with coarse-grain blocking | Dominates - makes parallel slower |
| **Practical Benefit** | * Significant performance improvement | ✗ No improvement, possibly degradation |