**Activity: Understanding Recursive and Data Decomposition**

**Course**: Parallel Computing (Final Year)  
**Level**: 7/10  
**Duration**: 30–40 minutes  
**Type**: Guided Questions + Short Exercise

**🔍 Learning Objectives**

By the end of this activity, students will be able to:

* Apply recursive decomposition to divide a problem into subproblems.
* Identify input, output, and intermediate data in a parallel algorithm.
* Apply data decomposition for efficient parallelization.

**📘 Scenario: Image Processing with Parallel Algorithms**

You are designing a parallel image processing system for edge detection using the Sobel filter. The image is represented as a 2D matrix of grayscale values (0–255).

Your goal is to:

1. **Apply recursive decomposition** to divide the problem.
2. **Apply data decomposition** to identify parallel tasks.
3. **Identify input, output, and intermediate data** in each method.

**Part 1: Recursive Decomposition**

**🧠 Q1. Recursive Strategy**

You decide to use a recursive algorithm that divides the image into smaller quadrants until the size reaches 64x64 pixels, which are then processed independently.

**Q1a.** Describe the recursive decomposition of a 1024x1024 image into 64x64 blocks.  
**Q1b.** How many levels of recursion are needed?  
**Q1c.** What are the input, output, and intermediate data at each level?

**✅ Answers**

**A1a.**  
At each recursive step, the image is split into 4 quadrants:

* Level 0: 1 image (1024x1024)
* Level 1: 4 images (512x512)
* Level 2: 16 images (256x256)
* Level 3: 64 images (128x128)
* Level 4: 256 images (64x64) – base case

**A1b.**  
4 levels of recursion are needed to reach 64x64 blocks.

**A1c.**

| **Level** | **Input Data** | **Intermediate Data** | **Output Data** |
| --- | --- | --- | --- |
| 0 | Full image (1024x1024) | Quadrants (512x512) | None |
| 1–3 | Sub-images from parent | Smaller sub-images | None |
| 4 | 64x64 image block | Sobel filter values | Edge-detected block |

**Part 2: Data Decomposition**

You now decide to process the image in parallel using **data decomposition**.

**🧠 Q2. Data Decomposition Strategy**

**Q2a.** Propose a 2D block decomposition of the 1024x1024 image using 16 processing elements.  
**Q2b.** What data does each processing element need to perform edge detection using the Sobel filter (3x3 kernel)?  
**Q2c.** Identify input, output, and intermediate data for each processing element.

**✅ Answers**

**A2a.**  
Divide the image into a 4x4 grid of blocks: each block is 256x256. Each of the 16 processors is assigned one block.

**A2b.**  
Each processor needs its 256x256 block **plus a 1-pixel border** from adjacent blocks for convolution (i.e., a **halo region**), making it 258x258 in total.

**A2c.**

| **Component** | **Description** |
| --- | --- |
| **Input Data** | Local block (256x256) + 1-pixel halo (from neighbors) |
| **Intermediate Data** | Gradient magnitude (Gx, Gy), convolution results |
| **Output Data** | Final 256x256 edge-detected image |

**Part 3: Reflective Questions**

**Q3a.** Compare the scalability of recursive vs. data decomposition in this problem.  
**Q3b.** Which decomposition would you choose for a real-time application, and why?

**✅ Sample Answer**

**A3a.**

* **Recursive Decomposition** introduces overhead due to recursion and merging sub-results. It’s scalable in divide-and-conquer problems but may be inefficient for large image operations.
* **Data Decomposition** is more efficient in this case because operations are local and can be done in parallel with minimal communication.

**A3b.**  
Data decomposition is better for real-time applications as it enables straightforward parallelism with lower overhead and better cache locality.

**🧠 Extension Task (Optional)**

Implement a simple recursive or data decomposition approach in Python using NumPy. Simulate dividing the image and calculating the edge detection.

Would you like a worksheet version (PDF or editable document) or coding task to pair with this?

Ask ChatGPT

**You said:**

some more activity

**ChatGPT said:**

Great! Here are **two more Level 7/10 activities** focused on **Recursive Decomposition** and **Data Decomposition**, building on different parallel computing concepts. Each includes clear questions and answers.

**🧩 Activity 2: Recursive Decomposition in Parallel Sorting**

**Scenario**: You are tasked with implementing a parallel version of **Merge Sort** using recursive decomposition.

**🔍 Part A: Understanding Recursive Decomposition in Merge Sort**

**Q1.** Describe how recursive decomposition works in Merge Sort.  
**Q2.** At what point should the recursion stop when implementing in parallel?  
**Q3.** Identify input, output, and intermediate data in the recursive merge sort.

**✅ Answers**

**A1.**

* The input array is divided recursively into halves until a single element remains.
* Each recursive step can be performed in parallel.
* Merging steps combine the results up the recursion tree.

**A2.**

* Recursion can stop at a base case, e.g., when subarray size ≤ 1000.
* At this point, a sequential sort (like insertion sort) can be more efficient.

**A3.**

| **Step** | **Input** | **Intermediate Data** | **Output** |
| --- | --- | --- | --- |
| Initial Split | Full array | Subarrays | None |
| Recursive Sorts | Subarrays | Sorted smaller arrays | None |
| Merge | Two sorted arrays | Temporary merge buffer | Merged array |

**🔍 Part B: Task Parallelism**

**Q4.** How would you implement task parallelism in this version of Merge Sort?  
**Q5.** If the array is of size 2^20 and you have 8 processors, what is the maximum parallelism level you should allow?

**✅ Answers**

**A4.**

* Assign recursive sort calls to parallel tasks.
* Merge operations can also be parallelized if implemented carefully.

**A5.**

* Max parallelism is achieved when 8 tasks run simultaneously.
* So, parallelize until the recursion tree reaches 8 leaf tasks (or levels).

**🧩 Activity 3: Data Decomposition in Matrix Multiplication**

**Scenario**: You're implementing parallel matrix multiplication (C = A × B), where all matrices are 1024×1024.

**🔍 Part A: Block Decomposition**

**Q1.** Describe how you can apply data decomposition to matrix multiplication.  
**Q2.** If you have 16 processors, how would you decompose the work?  
**Q3.** What data must each processor access?

**✅ Answers**

**A1.**  
Use **block decomposition**: split matrices into sub-blocks, assign each to a processor.

**A2.**  
Split each matrix into 4×4 blocks (each 256×256).  
Each processor computes one block of the output matrix.

**A3.**

| **Processor Task** | **Needs Access to** |
| --- | --- |
| Compute C(i,j) | Row i of A's sub-blocks and column j of B's sub-blocks |

**🔍 Part B: Input/Output/Intermediate**

**Q4.** Identify input, output, and intermediate data for each processor.  
**Q5.** How can you minimize communication between processors?

**✅ Answers**

**A4.**

| **Data Type** | **Description** |
| --- | --- |
| **Input** | Sub-blocks of A and B |
| **Intermediate** | Partial product accumulations |
| **Output** | Final sub-block of matrix C |

**A5.**  
Use **replicated broadcast** or **Cannon’s algorithm** to reduce data movement.  
Assign blocks smartly so that processors can reuse data.

**🧠 Reflective Question**

**Q6.** Which decomposition method (recursive or data) would be better for large-scale matrix multiplication on a distributed system?

**✅ Sample Answer:**  
**Data decomposition** is better because it minimizes recursive overhead and works well with large, regular datasets. Efficient mapping and memory access patterns also help performance.

**Want more?**

I can also provide:

* A **coding assignment** using OpenMP or MPI based on these.
* A **quiz-style activity** for revision.
* A **worksheet or editable template** for classroom use.

Let me know what format you prefer next!

Top of Form



Attach

Search

Voice

Bottom of Form