**Assignment: Exploratory & Speculative Decomposition in Parallel Computing**

**Objective:**  
To understand, compare, and practically explore *exploratory decomposition* and *speculative decomposition* techniques in parallel programming.

**Part A – Conceptual Understanding**

1. **Exploratory Decomposition**:
   * Define exploratory decomposition and explain its importance in problems with unknown search spaces.
   * Give two real-world problems (outside textbook examples) where this decomposition technique is applicable. Justify your choice.
2. **Speculative Decomposition**:
   * Define speculative decomposition and explain the concept of *speculation* in parallel execution.
   * Provide one advantage and one disadvantage of speculative decomposition, with brief reasoning.
3. **Comparison Table**:
   * Prepare a comparative table (minimum 4 parameters) between exploratory and speculative decomposition.

**Part B – Case Study Analysis**

1. Select a problem that can be solved using **exploratory decomposition** (e.g., pathfinding, puzzle solving, molecular simulation).
2. Select another problem that can benefit from **speculative decomposition** (e.g., branch prediction in compilers, speculative execution in databases).
3. For each problem:
   * Describe the problem.
   * Outline how you would decompose it into parallel tasks.
   * Discuss possible challenges (e.g., load balancing, wasted computation, synchronization).

**Part C – Implementation Task**

1. **Exploratory Decomposition Mini Project**:
   * Implement a parallel program (using OpenMP, MPI, or any parallel library) for a **search problem** such as solving a maze, N-Queens, or Sudoku, where different tasks explore different regions of the solution space concurrently.
2. **Speculative Decomposition Mini Project**:
   * Implement a program where multiple possible future paths are computed in parallel (e.g., evaluating both branches of an if statement in parallel) and the correct result is selected after a condition is resolved.
3. Record and compare:
   * Execution time (sequential vs. parallel).
   * Any wasted computation or rollback in speculative decomposition.

**Part D – Report Submission**

Prepare a **short technical report** (max 6 pages) including:

* Introduction to both techniques.
* Problem descriptions.
* Algorithm design with diagrams.
* Implementation details (code snippets allowed).
* Results (tables/graphs).
* Observations and conclusions.

**Evaluation Criteria:**

| **Criteria** | **Marks** |
| --- | --- |
| Conceptual clarity (Part A) | 15 |
| Case study analysis (Part B) | 20 |
| Implementation correctness & efficiency (Part C) | 30 |
| Report quality & depth (Part D) | 15 |
| Originality & extra insights | 10 |
| **Total** | **90** |

If you like, I can also prepare **a sample problem set with expected answers and performance metrics** so students can benchmark their implementations. This would help you evaluate faster. Would you like me to create that?

**You said:**

give solutions

**ChatGPT said:**

Got it — I’ll give you **model solutions** for the exploratory decomposition and speculative decomposition assignment so your students’ submissions can be checked quickly.  
I’ll keep the explanations compact but clear, and include **conceptual, case study, and sample code snippets** so it’s evaluation-ready.

**Model Solutions – Exploratory & Speculative Decomposition Assignment**

**Part A – Conceptual Understanding**

**1. Exploratory Decomposition**

* **Definition:**  
  Breaking down a problem into independent tasks, each exploring a different part of the problem space to discover a solution. Used when the complete solution space is unknown or too large to explore sequentially.
* **Applications:**
  1. **Protein structure prediction** – different threads explore different molecular folding configurations.
  2. **Game AI move search** – each thread searches a different set of possible moves in a game like Chess or Go.

**2. Speculative Decomposition**

* **Definition:**  
  Running multiple possible future execution paths in parallel before knowing which one will be needed, then discarding results from incorrect paths.
* **Advantage:** Can reduce waiting time for decision-making when branch outcome is predictable.
* **Disadvantage:** Wasted computation if many speculative tasks are discarded.

**3. Comparison Table:**

| **Parameter** | **Exploratory Decomposition** | **Speculative Decomposition** |
| --- | --- | --- |
| **Nature** | Searches unknown solution space | Executes possible future paths |
| **Task Independence** | Fully independent regions | May have dependencies resolved later |
| **Waste of Computation** | Minimal (unless pruning) | High if wrong speculations |
| **Examples** | Pathfinding, puzzle solving | Branch prediction, database query planning |

**Part B – Case Study Analysis**

**Case 1 – Exploratory Decomposition: N-Queens Problem**

* **Problem:** Place N queens on an N×N chessboard so no two queens threaten each other.
* **Decomposition:** Each thread explores a different subset of the search tree starting from a unique first row placement.
* **Challenges:** Load balancing (some branches resolve faster), duplicate checks.

**Case 2 – Speculative Decomposition: Conditional Computation**

* **Problem:** An algorithm chooses between two heavy computations based on late-arriving sensor data.
* **Decomposition:** Start both computations in parallel. Once the sensor data arrives, keep the correct computation’s result and discard the other.
* **Challenges:** High wasted CPU cycles if decision is heavily skewed.

**Part C – Implementation Samples**

**1. Exploratory Decomposition – OpenMP N-Queens Example**

#include <stdio.h>

#include <omp.h>

#define N 8

int isSafe(int board[N], int row, int col) {

for (int i = 0; i < row; i++)

if (board[i] == col || abs(board[i] - col) == row - i)

return 0;

return 1;

}

void solve(int row, int board[N], int \*count) {

if (row == N) {

#pragma omp atomic

(\*count)++;

return;

}

for (int col = 0; col < N; col++) {

if (isSafe(board, row, col)) {

board[row] = col;

solve(row + 1, board, count);

}

}

}

int main() {

int count = 0;

#pragma omp parallel

{

int localBoard[N];

#pragma omp for

for (int col = 0; col < N; col++) {

localBoard[0] = col;

solve(1, localBoard, &count);

}

}

printf("Solutions: %d\n", count);

return 0;

}

**2. Speculative Decomposition – Parallel Branch Execution**

#include <stdio.h>

#include <omp.h>

int taskA() { for(int i=0;i<100000000;i++); return 42; }

int taskB() { for(int i=0;i<150000000;i++); return 84; }

int main() {

int resultA, resultB;

int decision = 1; // This would normally come later

#pragma omp parallel sections

{

#pragma omp section

resultA = taskA();

#pragma omp section

resultB = taskB();

}

if (decision == 1)

printf("Chosen Result: %d\n", resultA);

else

printf("Chosen Result: %d\n", resultB);

return 0;

}

**Part D – Sample Results**

| **Problem** | **Sequential Time (ms)** | **Parallel Time (ms)** | **Speedup** | **Wasted Computation (%)** |
| --- | --- | --- | --- | --- |
| N-Queens (Exploratory) | 850 | 300 | 2.83× | ~0% |
| Branch Execution (Speculative) | 950 | 520 | 1.82× | ~48% |

**Observations:**

* Exploratory decomposition achieves near-linear speedup when tasks are independent and search space is large.
* Speculative decomposition can still improve performance but wastes cycles if speculation is wrong.

If you want, I can also prepare a **ready-to-distribute PDF with these solutions plus diagrams of task decomposition trees** so students have a visual reference. That would make the evaluation more straightforward.

Do you want me to prepare that visual PDF?

**Unlock more with Plus**

ChatGPT Plus gives you higher limits, smarter models, and Sora for video.

Get Plus

Top of Form



Bottom of Form