Define the following terms:

Complexity of a problem	Efficiency
Bounded parallelism	Anomalies in parallel processing
Unbounded parallelism	Time complexity
Speedup factor	Utilization
Amdahl's law	Gustafson's law
Synchronization primitives	Semaphore
Execution rate	Degree of parallelism
Mutual exclusion	Monitor

Complexity of a problem

Se refiere a la cantidad de recursos computacionales que se necesitan para resolver un problema, sin importar el algoritmo especifico que se use.

Bounded parallelism

Se refiere a un modelo de ejecución paralela en donde el número de tareas que pueden ejecutarse de manera simultánea es limitado por una cantidad fija

Unbounded parallelism

Esto ocurre cuando no existe el límite fijo al número de tareas que pueden ejecutarse en paralelo; en este caso se puede pensar que es posible aprovechar cualquier número de procesadores

Speedup factor

Es la relación entre el tiempo de ejecución secuencial y el tiempo de ejecución paralelo

$$S = \frac{Tsecuencial}{Tparalelo}$$

Amdahl's law

Es una formula usada para encontrar la mejora máxima posible al mejorar solo una parte específica de un sistema, es una fórmula que proporciona la aceleración teórica en la latencia de ejecución de una tarea con una carga de trabajo fija.

$$S(n) = \frac{1}{(1-p) + \frac{p}{n}}$$

Synchronization primitives

Son herramientas básicas que permiten coordinar la ejecución de hilos/procesos concurrentes.

Execution rate

Es la velocidad con la que una máquina ejecuta instrucciones o tareas por unidad de tiempo

Mutual exclusión

Mecanismo que me garantiza que solo un proceso acceda a una sección critica a la vez, evitando así las condiciones de carrera

Efficiency

Medida de qué tan bien se aprovechan los recursos de cómputo en paralelo

$$E = \frac{S}{P}$$

Anomalies in parallel processing

Se refiere así, a las situaciones inesperadas donde aumentar el número de procesadores no mejora, o incluso empeora, el rendimiento

Time complexity

Es el número de operaciones que un algoritmo necesita en función del tamaño de entrada (n), expresado con notación O

Utilization

Es el porcentaje de recursos computacionales que están siendo usados durante la ejecución de un programa.

Gustafson's law

Es una ley que afirma que, al aumentar el número de procesadores, se pueden resolver problemas más grandes en el mismo tiempo

$$S(p) = p - f(p-1)$$

Semaphore

Variable usada para controlar mecanismos de sincronización para controlar el acceso a recursos compartidos en sistemas concurrentes o paralelos

Degree of parallelism

Número promedio de operaciones independientes que pueden ejecutarse simultáneamente en un programa.

Monitor

Estructura de alto nivel que combina variables compartidas, mecanismos de sincronización y procedimientos, para garantizar exclusión mutua automática

Design a parallel algorithm to compute

$$F(x) = x^{32} + x^{16} + x^{8} + x^{4} + x^{2} + x^{1}$$

(a) Express your algorithm as a computational graph.

(b) Express the speedup of your parallel algorithm over the sequential algorithm.

```cpp
double F(double x) {
    double p1, p2, p4, p8, p16, p32;
    p1 = x;
    p2 = p1 * p1;
    p4 = p2 * p2;
    p8 = p4 * p4;
    p16 = p8 * p8;
    p32 = p16 * p16;
    double s1, s2, s3;
    #pragma omp parallel sections
    {
        #pragma omp section
        { s1 = p32 + p16; }

        #pragma omp section
        { s2 = p8 + p4; }

        #pragma omp section
        { s3 = p2 + p1; }
    }
    // reducción final
    double t1, result;
    #pragma omp parallel sections
    {
        #pragma omp section
        { t1 = s1 + s2; }
        #pragma omp section
        { result = t1 + s3; }
    }
    return result;
}

int main() {
    double x;
    cout << "Introduce el valor de x: ";
    cin >> x;
    double result = F(x);
    cout << "F(x) = " << result << endl;

    return 0;
}
```

Paralelo

```cpp
#include <iostream>
using namespace std;

double F(double x) {
    double p1 = x;
    double p2 = p1 * p1;
    double p4 = p2 * p2;
    double p8 = p4 * p4;
    double p16 = p8 * p8;
    double p32 = p16 * p16;

    double s1 = p32 + p16;
    double s2 = p8 + p4;
    double s3 = p2 + p1;

    double t1 = s1 + s2;
    double result = t1 + s3;

    return result;
}

int main() {
    double x;
    cout << "Introduce el valor de x: ";
    cin >> x;

    double result = F(x);
    cout << "F(x) = " << result << endl;

    return 0;
}
```

Secuencial

$$\text{SpeedUp} = \frac{T_{secuencial}}{T_{paralelo}}$$

$$T_{secuencial} = 5.082\,s, \quad T_{paralelo} = 4.374\,s$$

$$\text{SpeedUp} = \frac{5.082}{4.374} \approx 1.16$$

Explain the relationship between the computing speed and physical size of a serial computer. Is this relationship likely to hold for a parallel system?

En un computador serial, la velocidad de cómputo está ligada con el tamaño físico del procesador, en donde a mejor distancia de propagación de señales, mayor frecuencia, ósea , mayor velocidad, Sin embargo, en un sistema paralelo, esta relación no se mantiene de forma directa puesto que el rendimiento depende de cosas como la sincronización , interconexión y comunicación entre múltiples procesadores, de forma que el simple hecho de reducir el tamaño físico, no garantiza un aumento en la velocidad del sistema

Define the following terms:

Directed graph	Undirected graph
Adjacency matrix	Adjacency list
Weighted graph	Spanning tree
Connected component	Cycle in a graph
Connectivity matrix	Path in a graph

Directed graph

Grafo en el que las aristas tienen una dirección; es decir, cada arista va de un nodo origen a un nodo destino. Se presenta como un par *(u,v)*

Adjacency matrix

Matriz cuadrada que representa las conexiones entre los nodos de un grafo.

Weighted graph

Grafo donde cada arista tiene un peso asociado

Connected component

En un grafo no dirigido, es un subgrafo en el que cualquier par de nodos está conectado por un camino y no hay conexión con nodos fuera de un subgrafo

Connectivity matrix

Matriz que indica si existe o no un camino entre cada par de nodos. Puede derivarse de la matriz de adyacencia elevándola a potencias sucesivas.

Undirected graph

Grafo en el que las aristas no tienen dirección. Una arista (u,v) indica una conexión bidireccional entre los nodos u y v.

Adjacency list

Estructura que representa un grafo como un arreglo de listas, donde cada índice corresponde a un nodo y su lista contiene los nodos vecinos conectados por aristas.

Spanning tree

Subgrafo que incluye todos los nodos del grafo original

Cycle in a graph

Secuencia de nodos que comienza y termina en el mismo nodo, y en la que todas las aristas y nodos intermedios son distintos (excepto el primero y último).

Path in a graph

Secuencia de nodos donde cada par consecutivo está conectado por una arista.

Devise a PRAM algorithm to solve the graph-coloring problem that has lower time complexity than the algorithm presented in this chapter. You may use a more powerful PRAM model.

Algorithm 1 PRAM Graph Coloring (CRCW)

```
Input: Grafo G = (V, E), |V| = n, |E| = m, grado máximo Δ.
Output: Coloreo propio de G con a lo sumo Δ + 1 colores.
Inicialización en paralelo:
for all v ∈ V in parallel do
    color[v] ← Random(1, Δ + 1)
end for
repeat
    Detección de conflictos (en paralelo):
    for all (u, v) ∈ E in parallel do
        if color[u] = color[v] then
            if ID[u] < ID[v] then
                mark[v] ← TRUE
            else
                mark[u] ← TRUE
            end if
        end if
    end for
    Resolución de conflictos (en paralelo):
    for all v ∈ V in parallel do
        if mark[v] = TRUE then
            mark[v] ← FALSE
            forbidden ← {color[u] : (u, v) ∈ E}
            color[v] ← Random({1, ..., Δ + 1} \ forbidden)
        end if
    end for
until no haya conflictos
Return: arreglo color[v].
```

Define the following terms:

Depth-first search	Breadth-first search
Best-first search	Branch-and-bound
Alpha-beta search	Divide-and-conquer

Depth-first search

Estrategia que explora un grafo desde lo más profundo de un camino.

Best-first search

Método que busca primero el mejor valor según la función de evaluación

Alpha-beta search

Es una optimización del algoritmo minimax, y su objetivo es reducir el número de nodos evaluados en el árbol de decisiones

Breadth-first search

Es un algoritmo de recorrido de grafos que explora nivel por nivel, comenzando desde un nodo raíz y visitando todos sus vecinos antes de pasar al siguiente nivel.

Branch-and-bound

Es una técnica de optimización para problemas combinatorios, este divide el problema en subproblemas y usa cotas para descartar ramas que no pueden mejorar la solución actual

Divide-and-conquer

Es una estrategia de diseño algorítmico que consiste en dividir, resolver y combinar, este divide el problema en subproblemas más pequeños, resuelve cada uno de manera recursiva y combina los resultados para obtener la solución general

Explain how to calculate a factorial number N! using the divide-and-conquer method.

Para calcular N! con dividir y conquistar, se divide el intervalo [1,N] en mitades recursivamente hasta llegar a subintervalos de un solo número; luego se multiplican los resultados obteniendo finalmente N!.

Let n denote the number of vertices, d_i denote the degree of vertex i, and m denote the number of edges in a graph. Prove that an upper bound for a sequential algorithm to search a graph (depth-first or breadth-first) is

$$T = \sum_{i=1}^{n} (d_i + 1) = 2m + n$$

where d_i is the maximum number of times that vertex i can be chosen as the vertex from which searching is to be done.

Demostración

Sea $G = (V, E)$ un grafo no dirigido con $n = |V|$ vértices y $m = |E|$ aristas. Sea d_i el grado del vértice i, es decir, el número de aristas incidentes en i. Queremos probar que el tiempo de ejecución de un algoritmo secuencial de búsqueda en anchura (BFS) o en profundidad (DFS) está acotado por

$$T = \sum_{i=1}^{n}(d_i + 1) = 2m + n.$$

Costo de procesar un vértice

Cuando se procesa un vértice i:

- Se realiza una operación constante para marcarlo o visitarlo (+1).
- Se recorren todas sus aristas incidentes para inspeccionar a sus vecinos ($+d_i$).

Por lo tanto, el costo de procesar el vértice i es $d_i + 1$.

Suma total de operaciones

Sumando sobre todos los vértices se obtiene:

$$T = \sum_{i=1}^{n}(d_i + 1).$$

En un grafo no dirigido se cumple la identidad fundamental

$$\sum_{i=1}^{n} d_i = 2m.$$

Sustituyendo en la expresión de T:

$$T = \sum_{i=1}^{n} d_i + \sum_{i=1}^{n} 1 = 2m + n.$$

Conclusión

El tiempo de ejecución de un algoritmo secuencial de búsqueda tiene como cota superior

$$T = 2m + n.$$

Esto refleja que cada arista se examina a lo sumo dos veces y cada vértice se procesa exactamente una vez.