OPTIMIZACIÓN DE FUNCIONES MATEMATICAS

Actividad 2

PEDRO ALBERTO ESPARZA GARZA
Matrícula: 9041301
Alumno: 23230221

CONTENIDO

TAREA

Favor de optimizar 5 funciones matemáticas benchmark haciendo uso de hilos para que encuentren una solución en paralelo por medio de un algoritmo genético (Deben indicar que parámetros están utilizando, es decir generaciones, población, individuos, cruce, mutación, cuantos hilos tiene su computadora, etc). En el lenguaje C# o el que sea de su preferencia. Puede ser en consola o en interfaz gráfica de usuario(Gui).

PLAN DE TRABAJO

Funciones Matemáticas a Optimizar:

- Función de Ackley
- Función de Rosenbrock
- Función de Rastrigin
- Función Sphere
- Función Griewank

Algoritmo Genético (AG) para resolver estas funciones de manera eficiente en paralelo. Los componentes principales de un algoritmo genético son:

Población: Conjunto de soluciones candidatas.

Generaciones: Número de veces que la población evoluciona.

Individuos: Representaciones de las soluciones (en este caso, números reales o vectores de números).

Cruzamiento: Forma de combinar dos soluciones para generar nuevas soluciones.

Mutación: Forma de alterar ligeramente una solución para explorar nuevas áreas del espacio de soluciones.

Aptitud: Mide qué tan cerca está un individuo de resolver el problema.

Uso de Hilos en PHP: Con la extensión pthreads en PHP, vamos a distribuir el trabajo entre varios hilos. Cada hilo puede encargarse de evaluar la aptitud de un subconjunto de la población, cruzar individuos, o mutar soluciones.

REQUERIMIENTOS DE DESARROLLO

Optimizar cinco funciones matemáticas de benchmark usando un algoritmo genético implementado en PHP. Aunque PHP no es conocido por su soporte nativo de hilos, la extensión pthreads proporciona una capacidad adecuada para implementar concurrencia y aprovechar los núcleos de CPU para mejorar la eficiencia en la búsqueda de soluciones óptimas.

El uso de hilos permitirá que múltiples operaciones del algoritmo genético (como la evaluación de aptitud, cruzamiento y mutación) se ejecuten en paralelo, acelerando el proceso de optimización.

- ✓ Debe usar PHP y la extensión pthreads para gestionar hilos.
- ✓ Debe implementar un algoritmo genético (AG) con los siguientes parámetros:
  - Tamaño de la población: 100 individuos
  - Generaciones: 1000 generaciones
  - Probabilidad de cruzamiento: 0.7
  - Probabilidad de mutación: 0.01
- ✓ Debe optimizar cinco funciones matemáticas:
  - Ackley
  - Rosenbrock
  - Rastrigin
  - Sphere
  - Griewank
- ✓ Debe utilizar tantos hilos como núcleos tiene la computadora en uso (por ejemplo, si tiene una CPU con 4 núcleos, se deben usar 4 hilos).
- ✓ Debe poder ser ejecutado en sistemas Linux/Unix con PHP >= 7.4.
- ✓ Podría utilizar mecanismos de sincronización si es necesario (ej: Mutexes).
- ✓ Debe evaluar la aptitud de cada individuo en paralelo para mejorar la velocidad de convergencia.

DISEÑO DE LA SOLUCION

Lenguage PHP nativo, utilizando clases para encapsular las operaciones del algoritmo genético, y respetando las convenciones de POO, como:

- ✓ Separación de responsabilidades.
- ✓ Uso de clases y métodos bien definidos.
- ✓ Tipado estricto (disponible desde PHP 7.4 en adelante).
- ✓ Buenas prácticas de concurrencia para el uso de hilos.

El diseño del algoritmo genético (AG) en PHP nativo se basa en una arquitectura orientada a objetos, dividiendo las responsabilidades clave del algoritmo en las siguientes clases y módulos:

AlgoritmoGenetico: Clase principal que gestiona el flujo del algoritmo, incluyendo la inicialización de la población, el proceso de evolución a través de generaciones, y la ejecución paralela de los hilos.

Individuo: Clase que representa una solución candidata dentro de la población. Cada individuo tiene un conjunto de genes (valores de los parámetros) y una función para calcular su aptitud.

Poblacion: Clase que mantiene un conjunto de individuos y proporciona **métodos** para realizar cruzamientos y mutaciones.

Función: Clase abstracta que define la interfaz para las funciones matemáticas de benchmark. Las clases específicas (como Ackley, Rastrigin, etc.) heredan de esta clase para implementar la evaluación de aptitud de cada función.

Hilos: Se usan hilos para evaluar la aptitud de los individuos de manera concurrente utilizando la extensión `pthreads`.

IMPLEMENTACIÓN

Esquema de clases y cómo interactúan entre sí:

```
class AlgoritmoGenetico {
    +__construct(int $poblacion, int $generaciones)
    +iniciar() : void
}

class Individuo {
    +__construct(array $genes)
    +calcularAptitud(Funcion $funcion) : float
    +getGenes() : array
    +getAptitud() : float
}

class Poblacion {
    +__construct(int $tamano)
    +seleccionarPadres() : array
    +realizarCruzamiento(array $padres) : Individuo
    +realizarMutacion(Individuo $individuo) : Individuo
}
```

```
class Funcion {

    +calcular(array $genes) : float

}


class Ackley extends Funcion {}

class Rosenbrock extends Funcion {}

class Rastrigin extends Funcion {}

class Sphere extends Funcion {}

class Griewank extends Funcion {}


AlgoritmoGenetico -> Poblacion : "mantiene"

Poblacion -> Individuo : "contiene"

Individuo -> Funcion : "usa para calcular aptitud"


Funcion <|-- Ackley

Funcion <|-- Rosenbrock

Funcion <|-- Rastrigin

Funcion <|-- Sphere

Funcion <|-- Griewank
```

DESCRIPCIÓN DE LAS CLASES

AlgoritmoGenetico: Controla el ciclo de vida del algoritmo genético.

__construct(int $poblacion, int $generaciones): Inicializa el algoritmo con el tamaño de la población y el número de generaciones.

iniciar(): Ejecuta el ciclo de vida del algoritmo genético, distribuyendo el trabajo entre los hilos para evaluar aptitudes.

Individuo: Representa a cada miembro de la población.

__construct(array $genes): Inicializa un individuo con un conjunto de genes.

calcularAptitud(Funcion $funcion): Calcula la aptitud del individuo con la función de evaluación correspondiente.

getGenes(): Devuelve los genes del individuo.

getAptitud(): Devuelve la aptitud del individuo.

Poblacion: Gestiona la población de individuos y los procesos de cruzamiento y mutación.

__construct(int $tamano): Inicializa una nueva población de tamaño dado.

seleccionarPadres(): Selecciona individuos para cruzamiento.

realizarCruzamiento(array $padres): Combina los genes de dos individuos para crear uno nuevo.

realizarMutacion(Individuo $individuo): Aplica mutación a un individuo con cierta probabilidad.

Funcion: Interfaz para las funciones matemáticas de evaluación.

calcular(array $genes): Calcula el valor de la función con un conjunto de genes dado.

PARÁMETROS Y CONFIGURACIÓN

Los parámetros utilizados para este diseño son los siguientes:

Tamaño de la población: 100 individuos.

Generaciones: 1000 generaciones.

Cruce: Selección de dos padres aleatorios para generar un hijo.

Mutación: Probabilidad de mutación de un 1% para cada gen.

Número de hilos: Se utiliza el número de núcleos de la CPU disponible en el servidor donde se ejecuta el algoritmo.

USO DE HILOS

En PHP, vamos a usar pthreads para ejecutar la evaluación de la aptitud en paralelo. Cada hilo se encargará de un subconjunto de la población, evaluando la aptitud de esos individuos. Se sincroniza la escritura de los resultados utilizando Mutexes, si es necesario.

```
class HiloEvaluacion extends Thread {
    private array $individuos;
    private Funcion $funcion;

    public function __construct(array $individuos, Funcion $funcion) {
        $this->individuos = $individuos;
        $this->funcion = $funcion;
    }

    public function run() {
        foreach ($this->individuos as $individuo) {
            $individuo->calcularAptitud($this->funcion);
        }
    }
}
```

El algoritmo genético puede lanzar varios hilos de este tipo, distribuyendo la población entre ellos.

DIAGRAMA GENERAL DEL FLUJO

El algoritmo genético (AG) se implementará utilizando PHP nativo con un enfoque OOP, aplicando hilos (con pthreads) para la evaluación en paralelo de la población. Se seguirán las mejores prácticas de diseño, incluyendo alto tipado y separación de responsabilidades a través de clases bien definidas.

actor User

participant "Main Algorithm" as AG

participant "Thread Pool" as Threads

participant "Fitness Calculation" as Fitness

participant "Crossover/Mutation" as Evolution

User -> AG: Ejecuta el Algoritmo Genético

AG -> Threads: Inicia Evaluación en Paralelo

Threads -> Fitness: Calcula Aptitud

AG -> Evolution: Selecciona, Cruza y Muta

Evolution -> AG: Devuelve Nueva Generación

AG -> User: Devuelve la Mejor Solución

Arquitectura de Clases

Algorithm: Controla el flujo general del algoritmo genético.

Population: Gestiona la población de individuos.

Individual: Representa un individuo en la población.

FitnessFunction: Calcula la aptitud de los individuos.

GeneticOperators: Define los operadores de cruzamiento y mutación.

ThreadedFitnessEvaluator: Implementa la evaluación de aptitud en paralelo usando hilos.

CÓDIGO FUENTE

Repositorio del del proyecto:

https://github.com/pedro-esparza/MCC/tree/main/TecnologiasdelaprogramacionMCCOM-0106-MCC16/Actividad2AlgoritmoGenetico

PATRONES DE DISEÑO UTILIZADOS

1. **Patrón Factory**
   Se utilizó para la creación de instancias de funciones matemáticas (Ackley, Rosenbrock, Rastrigin, etc.). El patrón Factory permite la creación de objetos sin especificar la clase exacta de los objetos que se crearán, mejorando la escalabilidad del sistema.

2. **Patrón Singleton**
   Aplicado en el diseño del router para garantizar que haya una única instancia de este componente en todo el ciclo de vida del algoritmo, gestionando el flujo general y el acceso a recursos compartidos.

3. **Patrón de Hilos (Thread Pool)**
   Implementado utilizando **pthreads** en PHP. Permite la evaluación concurrente de la aptitud de los individuos, creando hilos para distribuir el procesamiento de la población entre los núcleos disponibles de la CPU.

CONCLUSIÓN

La implementación del algoritmo genético para la optimización de funciones matemáticas en Ubuntu fue exitosa utilizando PHP con alto tipado, programación orientada a objetos (OOP), y la extensión pthreads para la ejecución en paralelo. A través de la creación de una máquina virtual con Ubuntu 24 sobre Windows utilizando Oracle VirtualBox, logramos ejecutar las pruebas con las siguientes características:

Funciones matemáticas de benchmark implementadas correctamente, incluyendo Ackley, Rosenbrock, Rastrigin, Sphere y Griewank.

Uso de concurrencia mediante la extensión `pthreads`, la cual permitió la evaluación en paralelo de la población en cada generación del algoritmo genético, mejorando el rendimiento global de la solución.

Nginx configurado como servidor web para las pruebas en un entorno de servidor, y uso de PHP CLI para ejecutar el algoritmo desde la línea de comandos en Ubuntu.

Recursos compartidos en la máquina virtual configurados correctamente para soportar la ejecución de múltiples hilos en paralelo, lo que permitió una optimización eficiente de los recursos de CPU asignados.

En resumen, la implementación en Ubuntu montado sobre Windows usando VirtualBox fue una solución viable y eficiente para el desarrollo y la prueba de algoritmos genéticos con concurrencia en PHP, aprovechando las capacidades nativas de Linux para manejar hilos y procesos en paralelo. El rendimiento obtenido es satisfactorio y la configuración ofrece un entorno flexible para pruebas futuras.