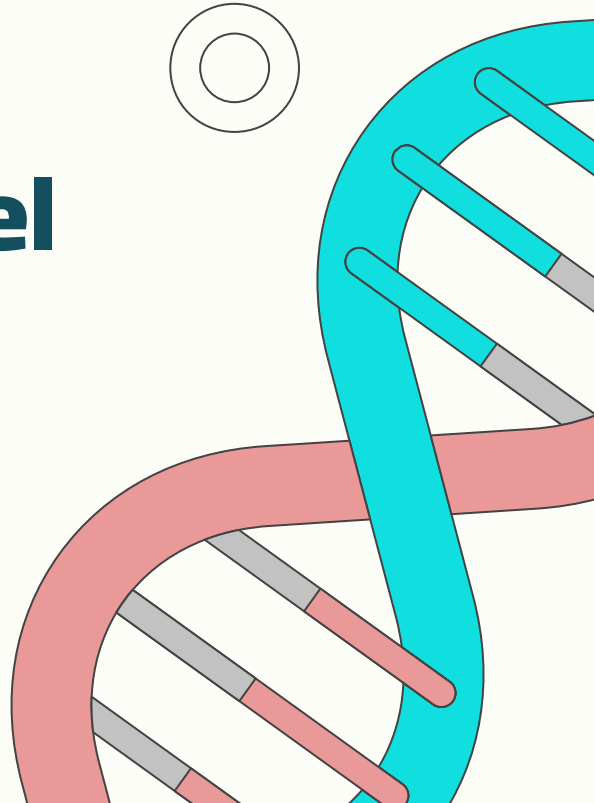
UNAM-FC

Paralelización del Algoritmo Genético para el TSP

Santiago Romero Palacios & Mariana Nava Córdova
Complejidad Computacional



Contenidos

El problema

Agente viajero euclidiano

Algoritmos para CPU

Algoritmos genéticos y paralelización con OpenMP

Paralelización para GPU

Con CUDA

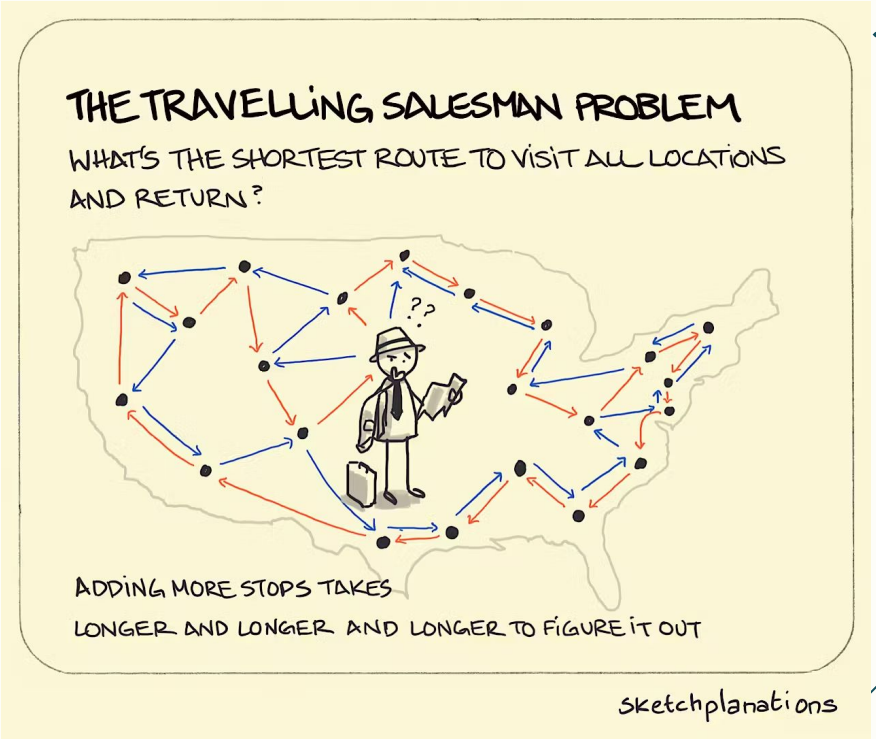
Resultados

Análisis y conclusiones

01

El problema

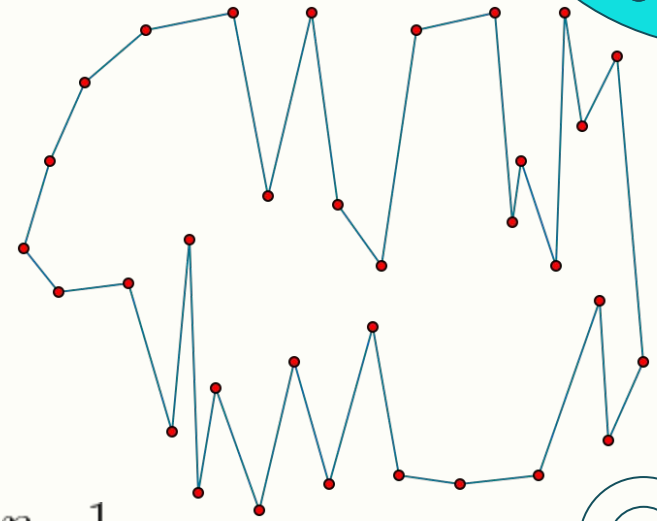
Agente viajero euclidiano



TSP EUCLIDIANO (OPTIMIZACIÓN)

Ejemplar: Un conjunto de ciudades, cada una con coordenadas en un plano.

Pregunta: ¿Cuál es la secuencia de ciudades que dan lugar al ciclo hamiltoniano de menor peso, es decir, tal que la distancia euclidiana recorrida es la menor?

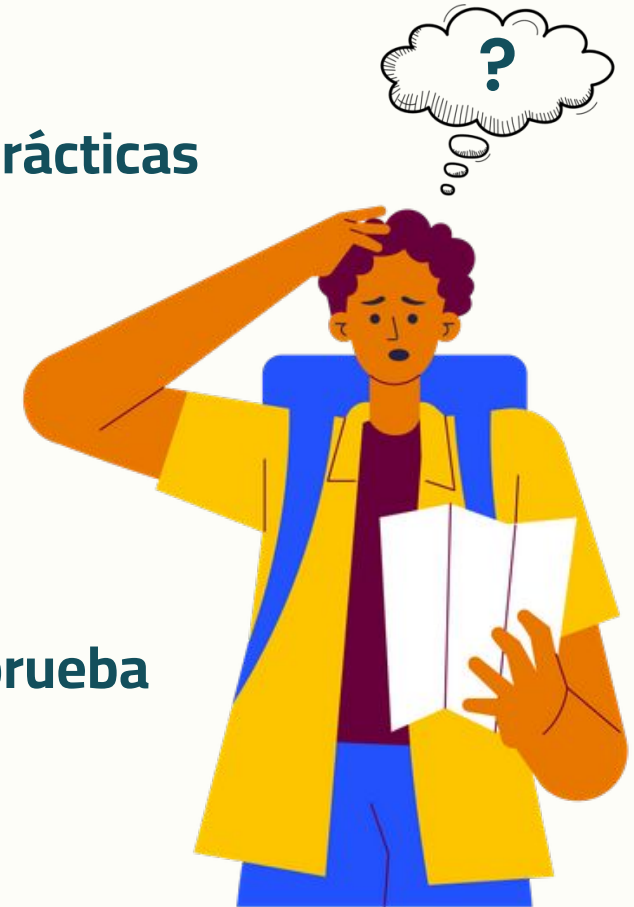


$$\sum_{i=0}^{n-1} \|x_i - x_{i+1 \bmod n}\|$$

¿Por qué este problema?

Aplicaciones prácticas

Fácil de entender



Fácil de generalizar

Fácil de evaluar

Buenos datos de prueba

Bien estudiado

02

Algoritmos para CPU

Algoritmos genéticos y paralelización OpenMP

Primero, ¿Qué es un algoritmo genético?

Un algoritmo genético es una metaheurística, la cual es aplicada a un problema en específico, que está inspirada en la combinación de los códigos genéticos y la producción de individuos mejor adaptados conforme se producen más generaciones.

¿En qué consiste un algoritmo genético?

Población

Individuos con alelos, genes y cromosomas

Generación

Estado de la población en un momento dado en el tiempo (entre iteraciones)

Selección

Para producir una nueva generación se deben determinar los individuos que se reproducirán

Cruza

Se combinan dos soluciones de tal forma que se tienen características de ambas al producir al nuevo individuo

Mutación

Se introducen pequeños cambios en algunos individuos con el propósito de tener variación genética

Elitismo

Se selecciona un conjunto de individuos con mejores puntajes o características

Nuestra implementación...

Población

Generamos una población inicial aleatoria de tamaño máx(2*N, 1000), con N el número de ciudades. Cada individuo dentro de la población es una permutación de ciudades.

Generación

Para hacer una nueva generación reemplazamos a un individuo dentro de la población sólo si el nuevo individuo generado tiene un mejor valor de aptitud; o con una baja probabilidad se puede reemplazar con uno que tenga una peor aptitud.

Selección

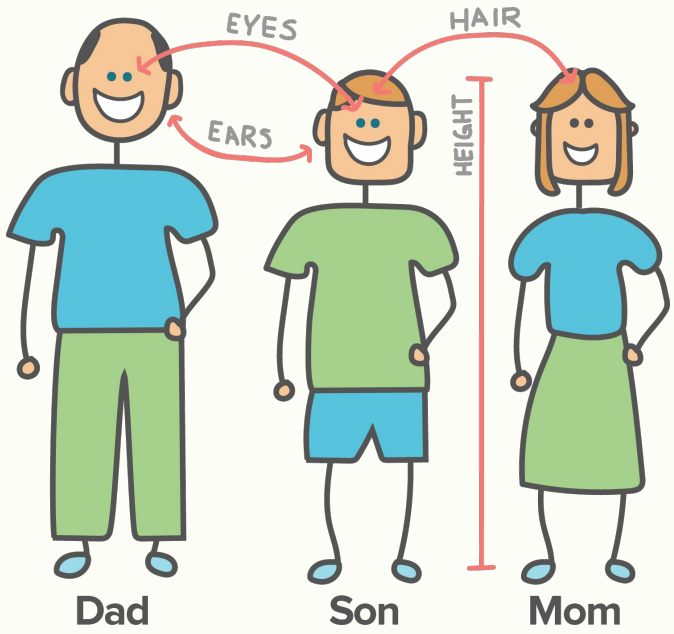
Hacemos la selección de los padres en tres pasos, para un individuo dentro de la población en la posición i:

- Uno de los padres para el nuevo individuo en la posición i de la nueva generación será el individuo original en la posición i.
- Para elegir el segundo padre, mediante un torneo entre 5 individuos elegidos aleatoriamente, elegimos al de mejor aptitud.

  Con una probabilidad baja podemos intercambiar al segundo padre por un individuo en la élite, para balancear convergencia más rápida con diversidad genética.

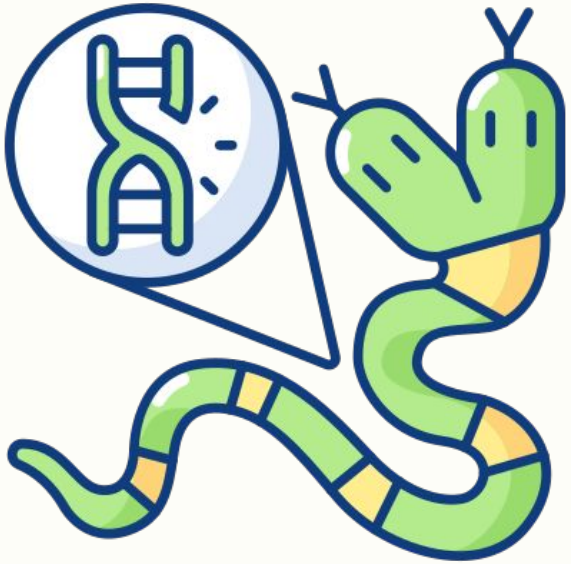
Cruza

1. Primero elegimos dos índices aleatorios i y j.
2. Encontramos las ciudades contenidas en el segundo padre en el intervalo [i,j].
3. Introducimos dichas ciudades a un conjunto.
4. Revisamos las ciudades del primer padre en el intervalo [0,j]; si no están contenidas en el conjunto, las agregamos.
5. Introducimos las ciudades restantes del segundo padre al conjunto.
6. Introducimos las ciudades restantes del segundo padre al conjunto.



Mutación

Se le hace un cambio a un individuo con una probabilidad baja para explorar el espacio de búsqueda de mejor manera. La mutación aplicada es la selección de dos índices aleatorios, para invertir el intervalo de ciudades que delimitan.



Elitismo

Elegimos los mejores 5 individuos que se han producido a lo largo de toda la ejecución del programa y los usamos en la selección para converger a soluciones mejores.

Cambios en la implementación en paralelo con OpenMP

- Creación de generadores de números aleatorios correspondientes a cada hilo
- Creación de poblaciones aisladas por hilo
- Comunicación entre hilos por grupo de élite

```
/**
 * Estructura para obtener números aleatorios con estado autocontenido.
 */
struct rng
{
    std::random_device RD_;
    std::mt19937 Generator_;

    /** Constructor */
    rng()
      : Generator_(RD_())
    { }

    /**
     * Da un i32 de 0 hasta N-1 inclusivo.
     */
    i32
    Rng_i32(i32 N)
    {
        std::uniform_int_distribution<i32> Distribution(0, N-1);
        return Distribution(Generator_);
    }

    /**
     * Da un r32 de 0 hasta 1 inclusivo.
     */
    r32
    Rng_r32()
    {
        std::uniform_real_distribution<r32> Distribution(0.0f, 1.0f);
        return Distribution(Generator_);
    }
};
```

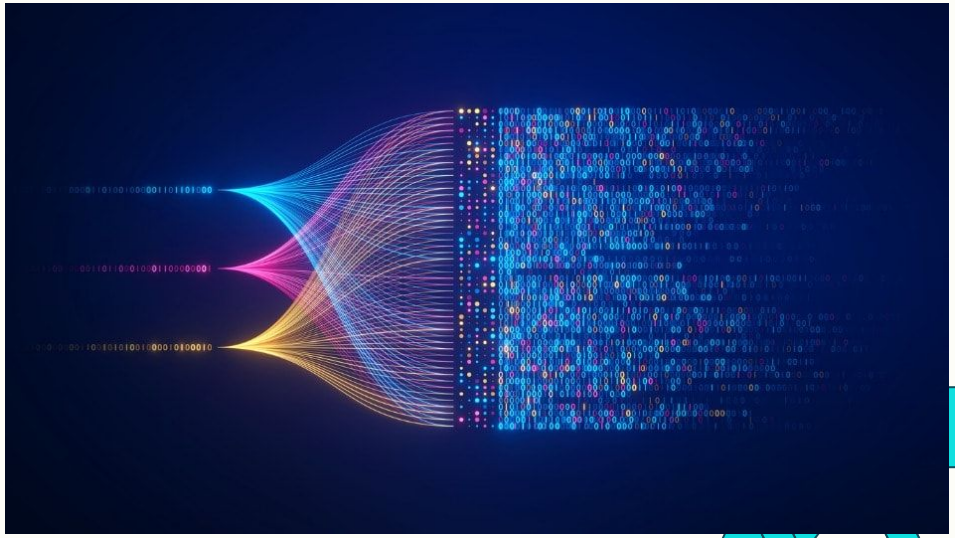
```
size_t RngLen = sizeof(rng)*MaxThreads;
size_t IslasLen = sizeof(i32) * N*TamPoblacion*(size_t)MaxThreads;
size_t PuntuacionesTLen = sizeof(r32) * TamPoblacion*(size_t)MaxThreads;
size_t IslasEliteLen = sizeof(i32) * NUM_ELITE*MaxThreads*(size_t)N;
size_t PuntuacionEliteLen = sizeof(i32) * NUM_ELITE*(size_t)MaxThreads;
size_t SolsPorHiloLen = sizeof(i32) * N*(size_t)MaxThreads;
size_t FitPorHiloLen = sizeof(r32) * MaxThreads;

size_t ArenaLen = RngLen + IslasLen +  PuntuacionesTLen +
    IslasLen + PuntuacionesTLen + IslasEliteLen + IslasEliteLen +
    PuntuacionEliteLen + SolsPorHiloLen + FitPorHiloLen;

void *Arena = mmap(0, ArenaLen,
        PROT_READ | PROT_WRITE,
        MAP_PRIVATE | MAP_ANONYMOUS,
        -1, 0);

if (Arena == MAP_FAILED)
{
    IGNORE_RESULT(write(1, ERR_MALLOC, sizeof(ERR_MALLOC)));
    return 0;
}

rng *Rngs = (rng *)Arena;
i32 *Islas = (i32*)(((u8*)Rngs) + RngLen);
r32 *PuntuacionesT = (r32*)(((u8*)Islas) + IslasLen);
i32 *NuevasIslas = (i32*)(((u8*)PuntuacionesT) + PuntuacionesTLen);
r32 *NuevasPuntuacionesT = (r32*)(((u8*)NuevasIslas) + IslasLen);
i32 *IslasElite = (i32*)(((u8*)NuevasPuntuacionesT) + PuntuacionesTLen);
i32 *IslasEliteNuevas = (i32*)(((u8*)IslasElite) + IslasEliteLen);
r32 *PuntuacionesEliteT = (r32*)(((u8*)IslasEliteNuevas) + IslasEliteLen);
i32 *MejoresSols = (i32*)(((u8*)PuntuacionesEliteT) + PuntuacionEliteLen);
r32 *MejoresCostos = (r32*)(((u8*)MejoresSols) + SolsPorHiloLen);
```

```
/**
 * Realiza un shuffle de toda la élite, incluso entre islas, para que se
 * comuniquen los resultados de las diferentes poblaciones.
 *
 * @param N Número de ciudades.
 * @param Elite Toda la élite.
 * @param PuntuacionesElite Todas las puntuaciones de la élite.
 */
internal void
RemezclaElite(i32 N, i32 *Elite, r32 *PuntuacionesElite)
{
    i32 Buf[N];
    i32 M = NUM_ELITE * MaxThreads;
    for (i32 R = M-1; R > 0; R--)
    {
        i32 L = rand() % (R+1);
        memcpy(Buf, GetIndividuo(Elite, N, L), sizeof(i32)*N);
        memcpy(GetIndividuo(Elite, N, L), GetIndividuo(Elite, N, R),
               sizeof(i32)*N);
        memcpy(GetIndividuo(Elite, N, R), Buf, sizeof(i32)*N);
        r32 Tmpf = PuntuacionesElite[L];
        PuntuacionesElite[L] = PuntuacionesElite[R];
        PuntuacionesElite[R] = Tmpf;
    }
}
```

03

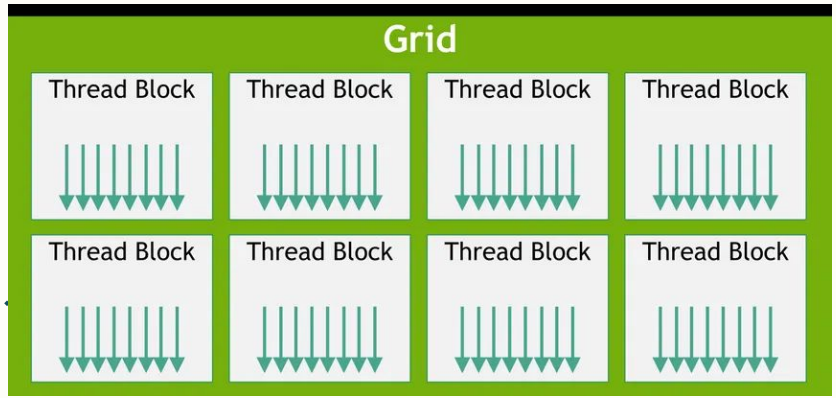
Paralelización para GPU

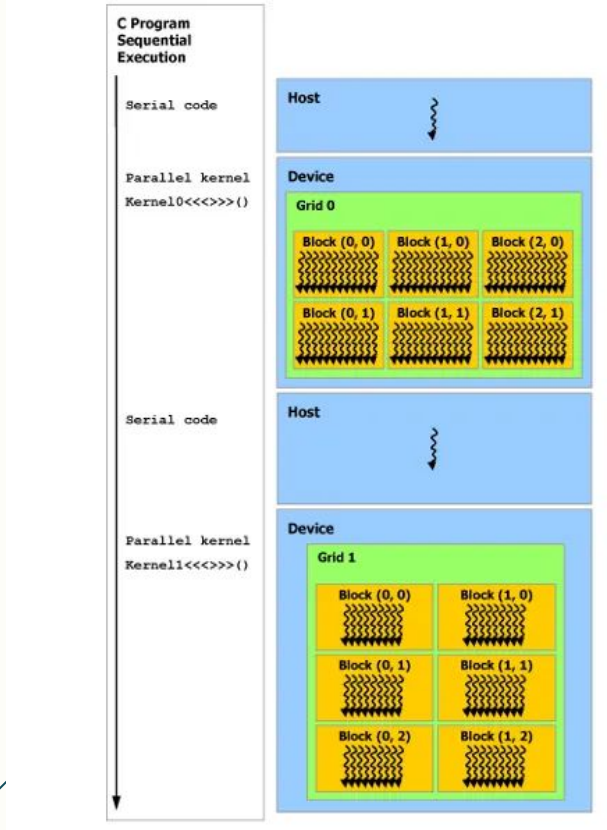
Con CUDA

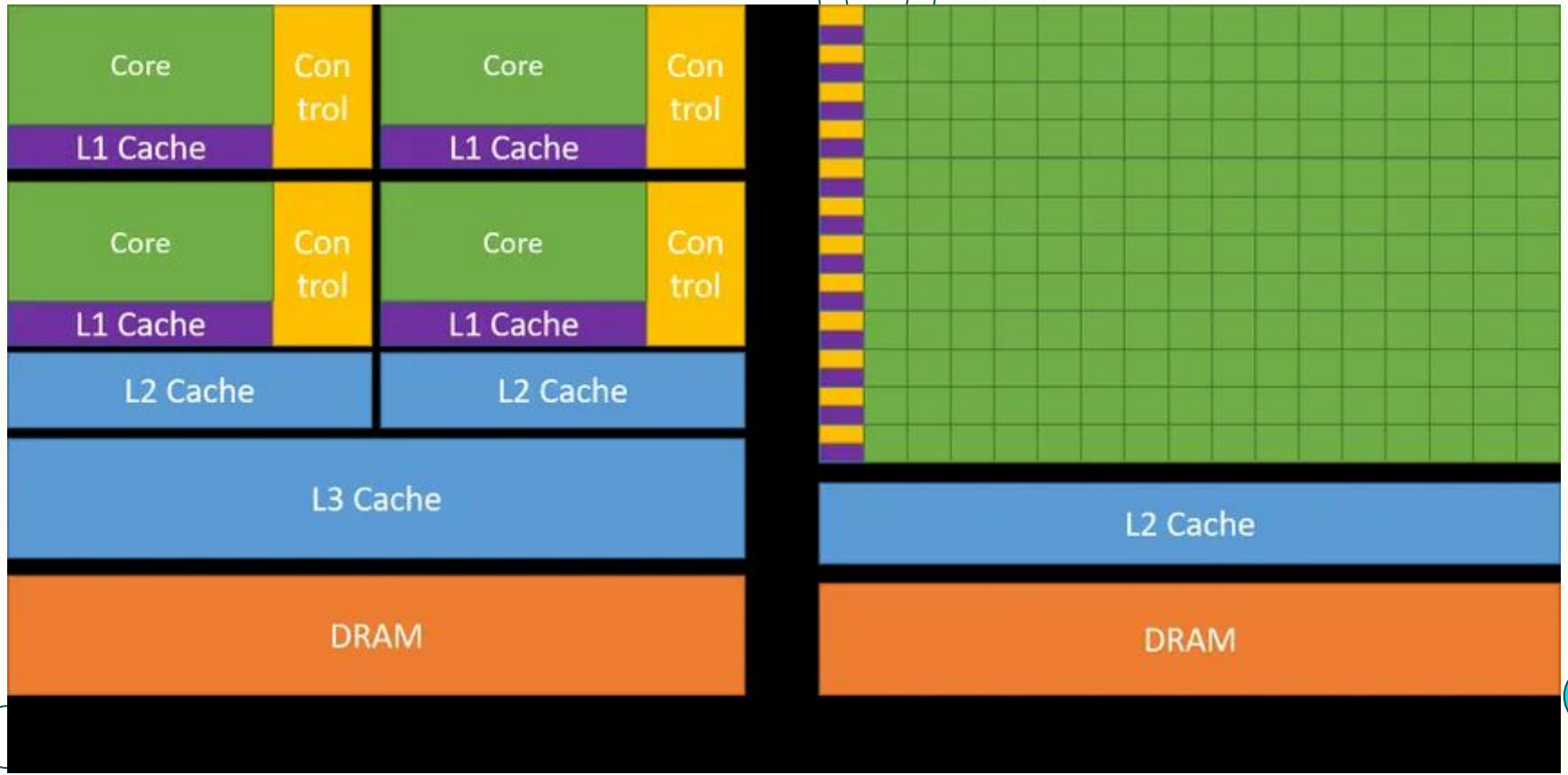


Cambios realizados entre el algoritmo para CPU y GPU

- Las poblaciones son del tamaño del número de hilos por bloque.
- El tamaño de hilos por bloque se decide por la cantidad de memoria compartida que requieren nuestros algoritmos.
- Encuentra la élite con reducciones paralelas de cada isla.






Core
Con trol
L1 Cache
Core
Con trol
L1 Cache
Core
Con trol
L1 Cache
Core
Con trol
L1 Cache
L2 Cache
L2 Cache
L3 Cache
DRAM
L2 Cache
DRAM


GAMERSNEXUS
PCI Express 3.0 Host Interface
GigaThread Engine
GPC
Raster Engine
TPC
SM
RT CORE
Memory Controller
L2 Cache
High-Speed Hub
NVLink – Two x8 Links

Realizamos esto directamente en el GPU con una técnica llamada *tiling*.

Algorithm 2 Pseudocode of PGAEI

```
main(){
  memory allocation on the CPU and the GPU
  calculate cost matrix
  copy of memory from CPU to GPU
  threads = IND_NUM
  blocks = ISLA_NUM
  IniPop<<<blocks,threads>>>()
  while condition_end==FALSE do
    PGAEI<<<blocks, threads>>>(parameters)
  end while
  copy of memory Elite Island from GPU to CPU
  get best solution
  free memory on GPU and CPU
}
__global__IniPop(){
  PobGen(I)
  Evalutate(I)
  set the best indiviaul in each island
  EliteMigrate(E)
}
__global__ PGAEI(){
  SetParents(I, E)
  I_new ← ParentsCross(I, E)
  Evaluate(I_new)
  if Fitness(I_new) < Fitness(I) then
    I = I_new
  end if
  I_new ← Mutation(I)
  set the best indiviaul in each island
  EliteMigrate(E)
}
```

Veamos una ejecución

Resultados

04

Análisis y conclusiones

Equipos de cómputo

Fedora

Parameter	Value
Hostname	fedora
OS Info	Fedora Linux 37 (KDE Plasma), Kernel: 6.5.12-100.fc37.x86_64
glibc Version	2.36
CPU Model	AMD Ryzen 7 PRO 2700U w/ Radeon Vega Mobile Gfx
CPU Vendor	AuthenticAMD
CPU Flags	3dnowprefetch abm adx aes aperfmperf apic arat avic avx avx2 bmi1 bmi2 bpext clflush clflushopt clzero cmov cmp_legacy constant_tsc cpb cpuid cr8_legacy cx16 cx8 de decodeassists extapic extd_apicid f16c flushbyasid fma fpu fsgsbase fxsr fxsr_opt ht hw_pstate ibpb irperf lahf_lm lbrv lm mca mce misalignsse mmx mmxext monitor movbe msr mtrr mwaitx nonstop_tsc nopl npt nrip_save nx osvw overflow_recov pae pat pausefilter pclmulqdq pdpe1gb perfctr_core perfctr_llc perfctr_nb pfthreshold pge pni popcnt pse pse36 rapl rdrand rdseed rdtscp rep_good sep sev sev_es sha_ni skinit smap smca smep ssbd sse sse2 sse4_1 sse4_2 sse4a ssse3 succor svm svm_lock syscall tce topoext tsc tsc_scale v_vmsave_vmload vgif vmcb_clean vme vmmcall wdt xgetbv1 xsave xsavec xsaveerptr xsaveopt
Logical Cores	8
Physical Cores	4
Base Frequency (MHz)	1600
Boost Frequency (MHz)	2200
Chipset	83DA
L1d Cache	128 KiB (4 instances)
L1i Cache	256 KiB (4 instances)
L2 Cache	2 MiB (4 instances)
L3 Cache	4 MiB (1 instance)
OpenMP Version	4.5
Total RAM (GB)	14.55
CUDA Capable Device	N/A
CUDA Version	N/A
AVX Supported	Yes
SSE Supported	Yes

GPU

Parameter	Value
Hostname	WallE
OS Info	Ubuntu 22.04.5 LTS, Kernel: 5.15.167.4-microsoft-standard-WSL2
glibc Version	2.35
CPU Model	AMD Ryzen 5 2600 Six-Core Processor
CPU Vendor	AuthenticAMD
CPU Flags	3dnowprefetch abm adx aes apic arat avx avx2 bmi1 bmi2 clflush clflushopt clzero cmov cmp_legacy constant_tsc cpuid cr8_legacy cx16 cx8 de decodeassists extd_apicid f16c flushbyasid fma fpu fsgsbase fxsr fxsr_opt ht hypervisor ibpb lahf_lm lm mca mce misalignsse mmx mmxext movbe msr mtrr nonstop_tsc nopl npt nrip_save nx osvw osxsave pae pat pausefilter pclmulqdq pdpe1gb perfctr_core pfthreshold pge pni popcnt pse pse36 rdrand rdrnd rdseed rdtscp rep_good sep sha sha_ni smap smep ssbd sse sse2 sse4_1 sse4_2 sse4a ssse3 svm syscall topoext tsc tsc_reliable tsc_scale v_vmsave_vmload virt_ssbd vmcb_clean vme vmmcall xgetbv1 xsave xsavec xsaveerptr xsaveopt
Logical Cores	12
Physical Cores	6
Base Frequency (MHz)	3.2
Boost Frequency (MHz)	3.7
Chipset	N/A
L1d Cache	192 KiB (6 instances)
L1i Cache	384 KiB (6 instances)
L2 Cache	3 MiB (6 instances)
L3 Cache	8 MiB (1 instance)
OpenMP Version	4.5
Total RAM (GB)	15.6
CUDA Capable Device	Yes
NVIDIA GPU Model	NVIDIA GeForce GTX 1060 6GB
Global Memory	6.00 GB
Shared Memory per Block	49152 bytes
Warp Size	32
Streaming Multiprocessor Count	20
Compute Capability	6.1
CUDA Core Count	1280
PCIe Bus Bandwidth	<15.76GB
GPU Memory Bus Width	192-bit
CUDA Version	12.5
AVX Supported	Yes
SSE Supported	Yes

Veamos el análisis en Jupyter

¡Gracias!

- Alba, E., & Luque, G. (2005). Measuring the Performance of Parallel Metaheuristics. En Parallel Metaheuristics (pp. 43–62). John Wiley & Sons, Ltd. https://doi.org/https://doi.org/10.1002/0471739383.ch2
- Gaxiola Sánchez, L. N., Tapia Armenta, J. J., & Díaz Ramírez, V. H. (2014). Parallel Genetic Algorithms on a GPU to Solve the Travelling Salesman Problem. Difu100ci@, Revista de difusión científica, ingeniería y tecnologías, 8(2), 84–90. http://difu100cia.uaz.edu.mx/index.php/difuciencia/article/view/145
- Merino Trejo, A. O. (2023). Algunas heurísticas aplicadas al problema del agente viajero [Licenciatura]. Universidad Nacional Autónoma de México [Facultad de Ciencias, UNAM]. http://132.248.9.195/ptd2023/septiembre/0846778/Index.html

Repositorio: https://github.com/srp-mx/parallel-tsp/