
# Corto 5
# Computación Paralela y Distribuida

**Hugo Rivas - 22500**
**Mauricio Lemus - 22461**
**Alexis Mesias - 22562**

**1. Diseño de la idea — Simulación de una cafetería (s**

Se simula la operación de una cafetería durante un intervalo T (p. ej., 180 min) con llegadas de clientes, toma de pedido y preparación de bebidas calientes. Cada cliente:

- llega según una tasa de arribo por minuto (puede variar por hora pico),
- hace un pedido (espresso/americano/latte/tea) con tiempos de servicio distintos,
- pasa por Caja (toma de orden/pago) y luego por Barra caliente (baristas),
- finaliza al recibir la bebida o abandona si su espera excede un umbral.

Indicadores (KPIs): pedidos completados, tiempo promedio de espera (caja/barra), utilización de recursos (cajeros/baristas), abandonos e ingresos estimados.

¿Qué va en paralelo y cómo se aplican las directivas?

Réplicas Monte Carlo (parallel for)

- Ejecutamos R réplicas independientes de la misma configuración para estimar promedios/variabilidad de KPIs.
- Se usa parallel for para distribuir las réplicas entre hilos.
- firstprivate: semilla del RNG y parámetros del escenario por hilo/réplica.
- reduction: agregación de KPIs globales (ventas, tiempos acumulados, completados, abandonos).
- shared: parámetros del sistema (tiempos medios, mezcla de pedidos, precios, T, $\Delta t$).

Subsistemas en cada réplica (sections)

Dentro de cada réplica y en cada tick de tiempo:

- sections separa el trabajo en subsistemas que avanzan en paralelo:
    - Llegadas/Abandono
    - Cajas
    - Barra caliente

Paralelismo dentro de cada sección

- Si hay varios recursos homogéneos (p. ej., 2 cajeros, 3 baristas), dentro de la sección puede usarse un parallel for adicional para repartir clientes entre esos servidores con schedule(dynamic) por la variabilidad del servicio.

Sincronización y utilidades

- barrier al final de cada tick para avanzar el reloj con estado consistente.
- single para el avance del reloj o logging ligero.
- schedule(dynamic) en bucles de servidores para balancear carga por tiempos variables.

Variables shared vs private/firstprivate

Compartidas (shared)

- Parámetros del sistema:
  N_CAJA, N_BARISTA, precios[], tiempos_servicio , mezcla_pedidos, tasa_llegadas[t], UMBRAL_ABANDONO.
- Estructuras de configuración (constantes o de solo lectura).
- Acumuladores globales (si no van por reduction):
  ventas_totales, espera_total, completados_total, abandonos_total, utilizacion_caja[], utilizacion_barista[].

Privadas / de réplica (private / firstprivate)

- RNG y semilla (con firstprivate para que cada réplica sea independiente).
- Estructuras de estado de la réplica (no compartidas):
  cola_caja, cola_barra, servidores_caja[], servidores_barista[].
- Métricas locales (que luego se reducen):
  ventas_local, espera_local, completados_local, abandonos_local, util_local_*.
- Variables temporales de iteración en los bucles (i, j, k), tiempos de servicio muestreados, cliente actual, etc.

**2. Pseudocódigo**

// ===== SHARED (solo lectura) =====

const T=180, DT=0.25, R=24, N_CAJA=2, N_HOT=2, N_COLD=1, UMBRAL=15

shared precios[Tipo], mu_caja, mu_hot[Tipo], mu_cold[Tipo], mezcla[Tipo]

shared lambda_por_min[tick]

```
// ===== REDUCCIONES GLOBALES =====
shared ventas_tot=0.0, espera_tot=0.0, compl_tot=0, aband_tot=0
// ===== SIMULACIÓN (réplicas en paralelo) =====
#pragma omp parallel for \
shared(precios,mu_caja,mu_hot,mu_cold,mezcla,lambda_por_min,T,DT,N_CAJA,N_HOT,N_COLD,
UMBRAL) \
  reduction(+:ventas_tot,espera_tot,compl_tot,aband_tot) \
  firstprivate(/* semilla_base, escenario */)
for r in 0..R-1 {
  RNG rng = init(semilla_base+r)
  queue<Cliente> q_caja, q_hot, q_cold
  Servidor caja[N_CAJA]={libres}, hot[N_HOT]={libres}, cold[N_COLD]={libres}
  double ventas=0.0, espera=0.0; int compl=0, aband=0
  for t in 0..T step DT {
    #pragma omp parallel sections
    {
      // A) Llegadas / Abandono
      #pragma omp section
      {
        int k = Poisson(lambda_por_min[t]*DT, rng)
        repetir k: enqueue(q_caja, Cliente{t_llegada:t, tipo:sortear(mezcla,rng)})
        while len(q_caja)>UMBRAL: dequeue(q_caja), aband++
        while len(q_hot)>UMBRAL:  dequeue(q_hot),  aband++
        while len(q_cold)>UMBRAL: dequeue(q_cold), aband++
      }
      // B) Cajas
      #pragma omp section
      {
        for i in 0..N_CAJA-1 {
```

```
    if caja[i].ocupado {
      caja[i].rest -= DT
      if caja[i].rest<=0 {
        c = caja[i].c; c.t_fin_caja = t
        if es_fria(c.tipo) enqueue(q_cold,c) else enqueue(q_hot,c)
        caja[i].libre()
      }
    }
    if caja[i].libre() and not empty(q_caja) {
      c = dequeue(q_caja); espera += (t - c.t_llegada)
      caja[i].ocupar(c, Exp(mu_caja, rng))
    }
  }
}
// C) Barra caliente
#pragma omp section
{
  for j in 0..N_HOT-1 {
    if hot[j].ocupado {
      hot[j].rest -= DT
      if hot[j].rest<=0 { ventas+=precios[hot[j].c.tipo]; compl++; hot[j].libre() }
    }
    if hot[j].libre() and not empty(q_hot) {
      c = dequeue(q_hot); espera += (t - c.t_fin_caja)
      hot[j].ocupar(c, Exp(mu_hot[c.tipo], rng))
    }
  }
}
```

```
      // D) Barra fría
      #pragma omp section
      {
        for k in 0..N_COLD-1 {
          if cold[k].ocupado {
            cold[k].rest -= DT
            if cold[k].rest<=0 { ventas+=precios[cold[k].c.tipo]; compl++; cold[k].libre() }
          }
          if cold[k].libre() and not empty(q_cold) {
            c = dequeue(q_cold); espera += (t - c.t_fin_caja)
            cold[k].ocupar(c, Exp(mu_cold[c.tipo], rng))
          }
        }
      }
    } // sections
  }
  // Reducción (ya aplicada por reduction)
  ventas_tot+=ventas; espera_tot+=espera; compl_tot+=compl; aband_tot+=aband
}
// ===== Resultados =====
prom_ventas   = ventas_tot / R
prom_espera   = espera_tot / max(1, compl_tot)
throughput    = compl_tot / (R*T)
tasa_abandono = aband_tot / max(1, (aband_tot+compl_tot))
```