

# **Documentación Técnica: Paralelización de KNN con MPI**

9 de noviembre de 2025

# Índice general

<b>1. Estrategia de Paralelización</b>	<b>2</b>
1.1. Paradigma Utilizado: Data Parallelism . . . . .	2
1.2. Técnicas MPI Implementadas . . . . .	2
1.3. Flujo de Ejecución Paralelo . . . . .	3
1.4. Región Paralelizable . . . . .	3
<b>2. Normalización de la Expresión Teórica</b>	<b>4</b>
2.1. Modelo Teórico Base . . . . .	4
2.2. Componente de Cómputo . . . . .	4
2.3. Componente de Comunicación . . . . .	4
2.4. Normalización y Ajuste de Parámetros . . . . .	5
<b>3. Resultados Experimentales</b>	<b>6</b>
3.1. Tabla de Resultados . . . . .	6
3.2. Análisis del Modelo y Escalabilidad . . . . .	6
3.3. Cantidad Óptima de Procesos . . . . .	6
<b>4. Observaciones Críticas</b>	<b>7</b>
4.1. Limitación del Modelo . . . . .	7
4.2. Recomendaciones . . . . .	7
4.3. Cálculo de FLOPs . . . . .	7
<b>5. Conclusiones</b>	<b>8</b>

# Capítulo 1

## Estrategia de Paralelización

### 1.1. Paradigma Utilizado: Data Parallelism

Se implementó un enfoque de **paralelismo de datos** donde el conjunto de test se divide equitativamente entre los procesos disponibles. Cada proceso ejecuta el mismo algoritmo KNN sobre su porción local de datos.

**Justificación:** KNN es un problema *embarrassingly parallel*, ya que cada predicción es completamente independiente de las demás, sin dependencias de datos entre iteraciones.

### 1.2. Técnicas MPI Implementadas

#### a) MPI\_Bcast (Broadcast)

```
X_train = comm.bcast(X_train, root=0)
y_train = comm.bcast(y_train, root=0)
```

**Propósito:** Distribuir el conjunto de entrenamiento completo a todos los procesos. **Complejidad:**  $O(\log p)$  con algoritmo de árbol binomial. **Razón:** Todos los procesos necesitan acceso al conjunto completo de entrenamiento para calcular distancias.

#### b) MPI\_Scatter (Dispersión)

```
local_X_test = comm.scatter(X_test_chunks, root=0)
local_y_test = comm.scatter(y_test_chunks, root=0)
```

**Propósito:** Distribuir el conjunto de test dividiéndolo en chunks de tamaño  $m/p$ . **Complejidad:**  $O(m/p)$ , donde  $m$  es el número de puntos de test. **Implementación:** Se utilizó `np.array_split()` para manejar divisiones no exactas automáticamente.

#### c) MPI\_Gather (Recolección)

```
all_predictions = comm.gather(local_predictions, root=0)
all_y_test = comm.gather(local_y_test, root=0)
```

**Propósito:** Recolectar todas las predicciones locales en el proceso raíz para evaluación final. **Complejidad:**  $O(m/p)$  por proceso.

## 1.3. Flujo de Ejecución Paralelo

Rank 0:

1. Cargar dataset (load\_digits)
2. Dividir en train/test
3. Particionar X\_test en p chunks

Todos los ranks:

4. Recibir X\_train, y\_train (Broadcast)
5. Recibir chunk local de X\_test (Scatter)
6. CÓMPUTO LOCAL: KNN sobre chunk local
7. Enviar predicciones locales (Gather)

Rank 0:

8. Ensamblar predicciones
9. Calcular accuracy
10. Reportar métricas

## 1.4. Región Paralelizable

```
# SECUENCIAL
y_pred = [knn_predict(x, X_train, y_train, k) for x in X_test]
# Complejidad: O(m      n      d)

# PARALELO
local_predictions = [knn_predict(x, X_train, y_train, k) for x in
                     ↪ local_X_test]
# Complejidad por proceso: O((m/p)      n      d)
```

# Capítulo 2

## Normalización de la Expresión Teórica

### 2.1. Modelo Teórico Base

Complejidad secuencial:

$$T_{seq} = m \times n \times d \times t_{op}$$

Donde:

- $m = 360$  (muestras de test)
- $n = 1437$  (muestras de entrenamiento)
- $d = 64$  (características)
- $t_{op}$  = tiempo por operación elemental

Complejidad paralela:

$$T_{par}(p) = T_{compute}(p) + T_{comm}(p)$$

### 2.2. Componente de Cómputo

$$T_{compute}(p) = \frac{m}{p} \times n \times d \times t_{op}$$

Normalizando  $t_{op}$  con el tiempo secuencial medido:

$$t_{op} = \frac{1,5669}{360 \times 1437 \times 64} = 4,73 \times 10^{-8} \text{ s/operación}$$

### 2.3. Componente de Comunicación

$$T_{comm}(p) = \alpha \log_2(p) + \beta(n \times d + m/p)$$

Donde:

- $\alpha$ : latencia base de comunicación colectiva.
- $\beta$ : tiempo por byte transferido.

## 2.4. Normalización y Ajuste de Parámetros

```
def modelo_teorico(p, alpha, beta):
    t_compute = (m * n * d / p) * t_op
    t_comm = alpha * log2(p) + beta * (n*d + m/p)
    return t_compute + t_comm
```

Parámetros ajustados:

$$\alpha = 1,00 \times 10^{-3} \text{ s}, \quad \beta = 1,00 \times 10^{-6} \text{ s/byte}$$

Modelo final:

$$T_{par}(p) = \frac{1,5669}{p} + 0,001 \log_2(p) + 0,092$$

# Capítulo 3

## Resultados Experimentales

### 3.1. Tabla de Resultados

Procesos	T_medido (s)	T_teórico (s)	Error (%)	Speedup	Eficiencia (%)
1	1.583	1.659	4.8	1.02	101.7
2	0.849	0.877	3.3	1.90	94.9
4	0.467	0.486	4.1	3.45	86.2
8	0.628	0.291	53.7	2.56	32.0

Cuadro 3.1: Comparación entre tiempos medidos y teóricos.

### 3.2. Análisis del Modelo y Escalabilidad

El modelo teórico se ajusta bien para  $p \leq 4$ , con un error promedio de 4 %. Para  $p = 8$ , el error crece significativamente (53.7 %) debido a contención de red, sincronización costosa y baja granularidad (solo 45 muestras por proceso).

El **speedup** y la **eficiencia** muestran un excelente desempeño hasta  $p = 4$ , donde:

$$S(4) = 3.45, \quad E(4) = 86.2 \%$$

Más allá de ese punto, el overhead de comunicación domina.

### 3.3. Cantidad Óptima de Procesos

La cantidad óptima de procesos se determinó en  $p_{óptimo} = 4$ , maximizando el balance entre velocidad, eficiencia y costo computacional.

# Capítulo 4

## Observaciones Críticas

### 4.1. Limitación del Modelo

El modelo actual no captura correctamente la discontinuidad observada en  $T_{comm}(8)$ . Una extensión sugerida es:

$$T_{comm}(p) = \alpha \log_2(p) + \beta(n \times d + m/p) + \gamma \max(0, p - p_{crítico})^2$$

donde el término cuadrático modela la contención a partir de un número crítico de procesos.

### 4.2. Recomendaciones

- Usar  $p = 4$  procesos para este dataset.
- Para datasets más grandes, reevaluar  $p_{óptimo}$ .
- Implementar estrategias de overlapping comunicación/cómputo para mejorar  $p=8$ .

### 4.3. Cálculo de FLOPs

$$FLOPs_{total} = m \times n \times (3d) = 360 \times 1437 \times 192 = 9,93 \times 10^7$$

Rendimiento:

$$p = 4 \Rightarrow 0,213 \text{ GFLOPS/s (máximo)}$$

# Capítulo 5

## Conclusiones

1. La paralelización fue exitosa hasta  $p = 4$ , alcanzando un speedup de  $3,45 \times$  y una eficiencia del 86 %.
2. El modelo teórico ajustado predice correctamente el comportamiento para  $p \leq 4$ .
3. El cuello de botella para  $p > 4$  es la comunicación, representando hasta el 72 % del tiempo total.
4. La accuracy del modelo se mantuvo constante en 98,33 % en todos los casos.