Universidad de Costa Rica
IE0521 - Estructuras de Computadoras Digitales II

DISEÑO DE CACHE EN PYTHON

Kevin Delgado Rojas
Marvin Castro
Andrés Chaves Vargas
Gabriel Briceño

QUÉ ES LA CACHE?

Memoria de rápido acceso. Que se encuentra entre la CPU y la RAM.

1. Reduce el tiempo de acceso a datos que se encuentran en memoria principal.

2. Es posible dividir la memoria cache en distintos ways. Lo que genera una mayor eficiencia al utilizar los espacios en memoria.

3. Puede tener hasta tres niveles de memoria. Cada nivel de diferente tamaño y con diferente tiempo de acceso.

OPTIMIZACIÓN AVANZADA

Existen distintas optimizaciones para mejorar aspectos específicos de la memoria cache. En este caso se escogió el método **Predicción de Sentido (Way).**

1. **Predice el Way a revisar en el siguiente acceso a cache**
2. **Reducción del Hit Time.**
3. **Precisión de 90 % para 2 Way.**

CÓDIGO GENERADO

```
def buildCache(data):
    linea, index, ways, cache_size = data
    cache = np.zeros((index, linea*ways))
    return cache
```

FUNCIÓN: TAGBLOGBITS

```python
def tagBlockBits(cache):
    linea, index, ways, cache_size = cache
    block_offset_bits = np.log2(linea)
    index_bits = np.log2(index)
    tag_bits = 32 - block_offset_bits - index_bits
    print(block_offset_bits, index_bits, tag_bits)
    return block_offset_bits, index_bits, tag_bits
```

FUNCIÓN: PROCESSTRACE

```
def processTrace(cache, data, address_bits, way_size, optimization=False):
    initial_time = time.time()
    linea, index, ways, cache_size = data
    # creación de máscaras para obtener tag, index y block offset dada la dirección
    block_offset_bits, index_bits, tag_bits = int(address_bits[0]), int(address_bits[1]), int(address_bits[2])
    mask_block_offset = (1 << block_offset_bits) - 1
    mask_index_bits   = ((1 << index_bits) - 1) << block_offset_bits
    mask_tag_bits     = ((1 << tag_bits) - 1) << (block_offset_bits + index_bits)
    i = 0
    way_predictor = 0
    with open('trace.out', 'r') as file:
        with open("logfile.txt", "w") as logfile:
            queue_LRU = []          # Lista que guarda los primeros elementos, para el reemplazo de LRU
            misses = 0              # Variable que guarda la cantidad de misses
            hits = 0                # Variable que guarda la cantidad de hits
            reemplazos = 0          # Variable que guarda la cantidad de reemplazos
            seed, [])
```

CÓDIGO DE PRUEBA

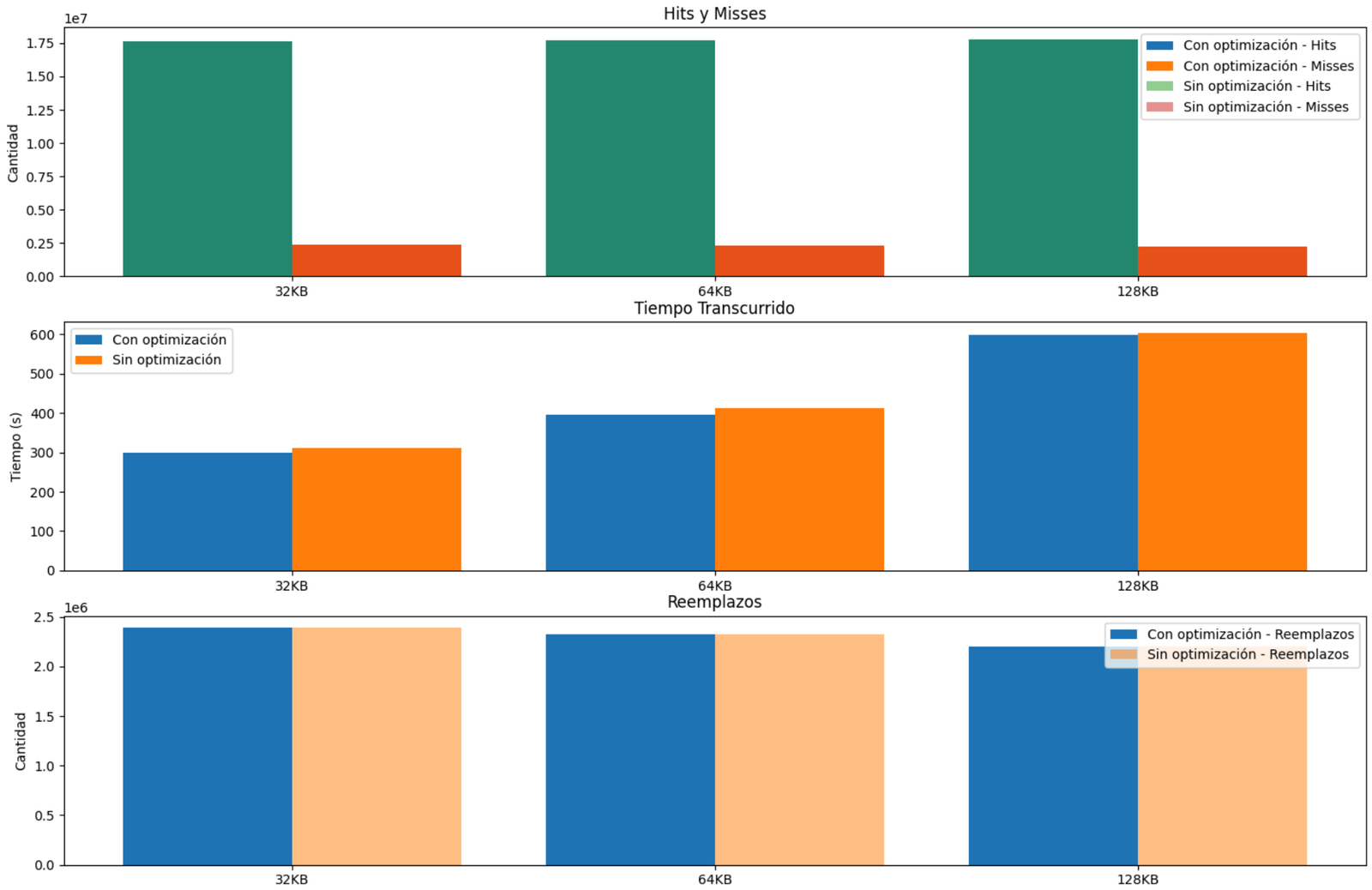
```python
if __name__ == '__main__':
    data = 32, 256, 4, 32768
    cache = buildCache(data)
    address_bits = tagBlockBits(data)
    way_size = 4
    HMR, total_time_funct = processTrace(cache,data, address_bits, way_size, True)
    print("Se tuvieron ", HMR[0], "hits")
    print("Se tuvieron ", HMR[1], "misses")
    print("Se tuvieron ", HMR[2], "reemplazos")
    print("Tiempo Transcurrido: {:.2f} s".format(total_time_funct))
    print(np.shape(cache))
```

RESULTADOS

Prueba: Barrido de Tamaño

Cantidad de Hits, Misses, Remplazos

Tiempo Transcurrido

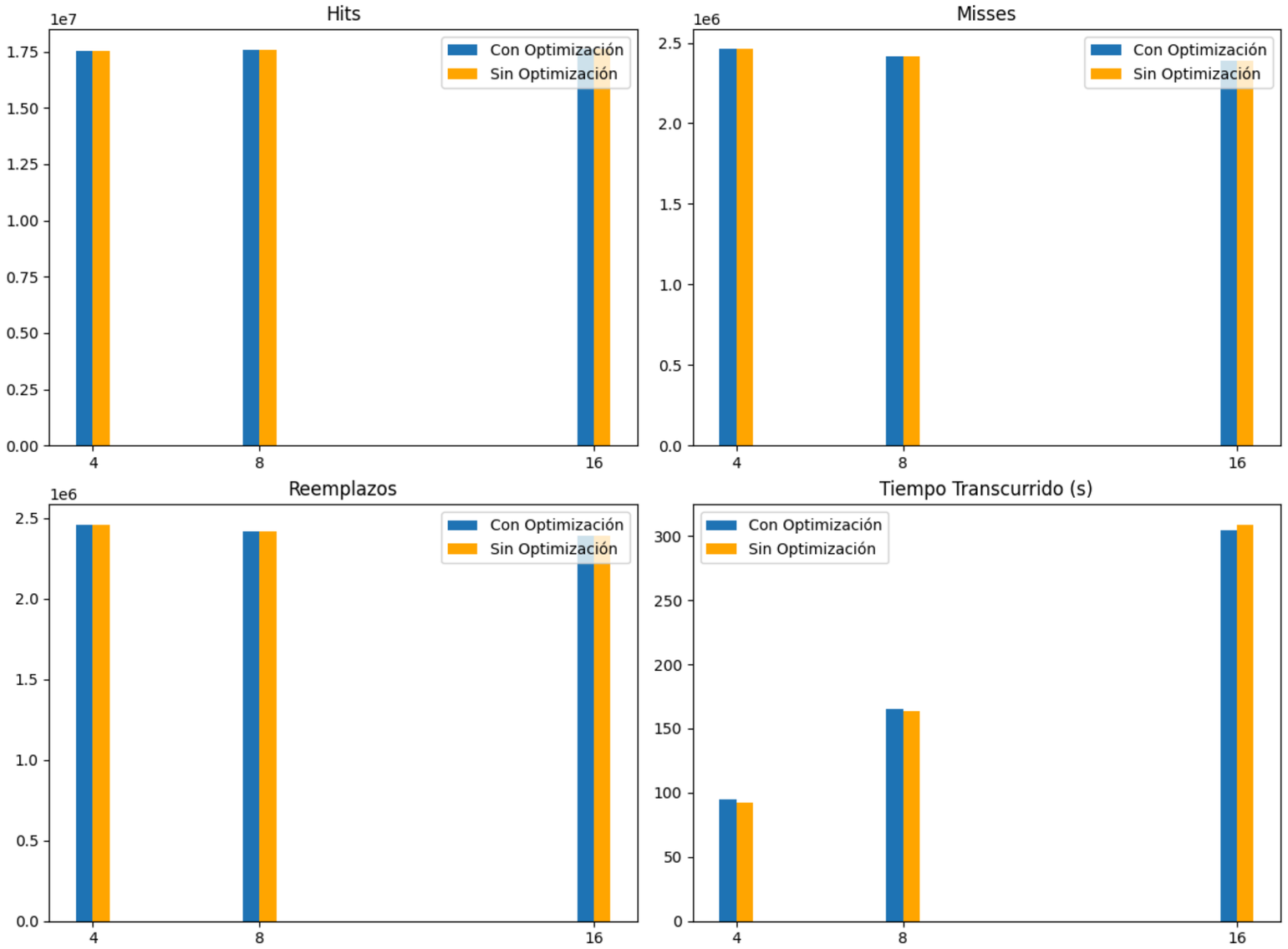


RESULTADOS

Prueba: Barrido de Asociatividad

Cantidad de Hits, Misses, Remplazos

Tiempo Transcurrido

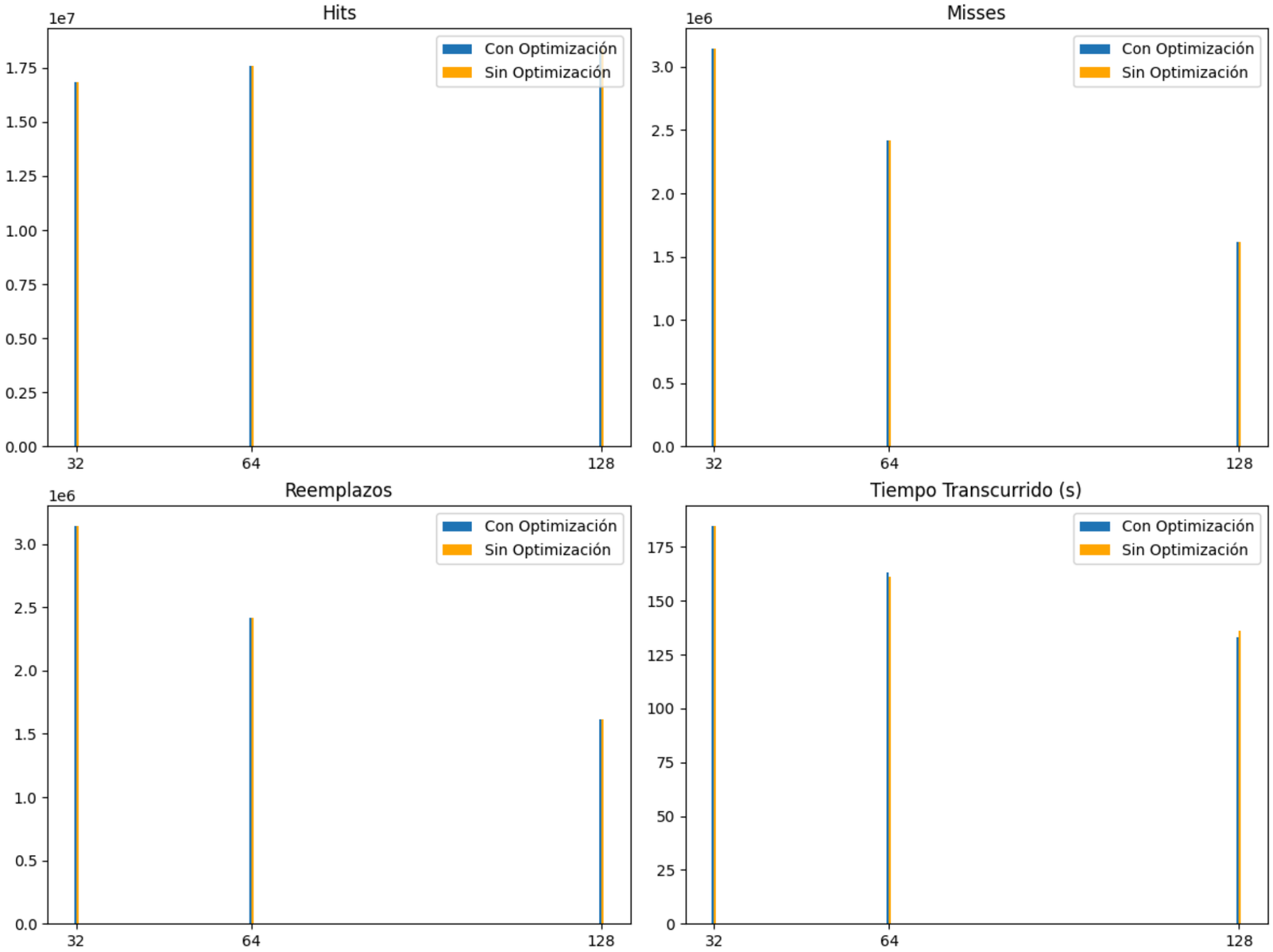


RESULTADOS

Prueba: Barrido de Línea de Cache

Cantidad de Hits, Misses, Remplazos

Tiempo Transcurrido



CONCLUSIONES

A continuación se muestran las conclusiones generadas a partir del proyecto realizado:

- **Versatilidad**

  La memoria cache puede implementarse de muchas formas, dependiendo del proyecto requerido.

- **Rapidez**

  Permite la obtención de datos de forma rápida. A diferencia de otras memorias.

- **Comprensión de Funcionamiento**

  Se obtuvo una mejor comprensión del funcionamiento de este tipo de memoria.

- **Mejoras**

  Se debe mejorar o incluso cambiar el método de optimización avanzada utilizado.