Parallel Computing Systems (CS4171-KP12)

Exercise Sheet 1 Submission

Md Norul Gofran
md.gofran@student.uni-luebeck.de
Matriculation Number: 814021

November 3, 2025

1 Task 1: Think Parallel (Prefix Sum)

Given Array (N=8): `x = [4, 9, 2, 7, 1, 6, 3, 8]`

1.1 Task 1.1: Sequential and Parallel Algorithm Design

1.1.1 Sequential Algorithm

The sequential algorithm for an inclusive prefix sum:

```
y[0] = x[0]
FOR i = 1 to N-1
    y[i] = y[i-1] + x[i]
END FOR
```

Execution Trace:

- `y[0] = 4`
- `y[1] = 4 + 9 = 13`
- `y[2] = 13 + 2 = 15`
- `y[3] = 15 + 7 = 22`
- `y[4] = 22 + 1 = 23`
- `y[5] = 23 + 6 = 29`
- `y[6] = 29 + 3 = 32`
- `y[7] = 32 + 8 = 40`

Final Prefix Sum: `[4, 13, 15, 22, 23, 29, 32, 40]`

1.1.2 Parallel Algorithm (Blelloch Scan)

The parallel algorithm consists of two passes (up-sweep and down-sweep).

1. **Reduce (Up-Sweep) Phase:** Builds a binary tree of partial sums.
2. **Down-Sweep Phase:** Traverses the tree from the root down to build the exclusive prefix sum.
3. **Final Step (Exclusive to Inclusive):** A final parallel operation where `y[i] = y_exclusive[i] + x[i]`.

1.2 Task 1.2: Reduce Operation (Up-Sweep Phase)

Input: `[4, 9, 2, 7, 1, 6, 3, 8]`

Stride	Active Indices (i)	Array State x
Initial	-	`[4, 9, 2, 7, 1, 6, 3, 8]`
1	1, 3, 5, 7	`[4, 13, 2, 9, 1, 7, 3, 11]`
2	3, 7	`[4, 13, 2, 22, 1, 7, 3, 18]`
4	7	`[4, 13, 2, 22, 1, 7, 3, 40]`

Outcome of Reduce Operation: The total sum, **40**, is in `x[7]`.

1.3 Task 1.3: Down-Sweep Phase Output

This phase constructs the *exclusive* prefix sum. We set the last element to 0.
Start: `[4, 13, 2, 22, 1, 7, 3, 40]` → `[4, 13, 2, 22, 1, 7, 3, 0]`

Stride	Active Indices (i)	Array State x
4	3, 7	`[4, 13, 2, 0, 1, 7, 3, 22]`
2	1, 3, 5, 7	`[4, 0, 2, 13, 1, 22, 3, 29]`
1	1, 3, 5, 7	`[0, 4, 13, 15, 22, 23, 29, 32]`

Final Output of Down-Sweep (Exclusive Sum): `[0, 4, 13, 15, 22, 23, 29, 32]`

1.4 Task 1.4: Evaluation and Complexity

Metric	Sequential Strategy	Parallel Strategy (Blelloch)
Time	$O(N)$ (One loop of N-1 steps)	$O(\log N)$ (Two tree traversals of $\log N$ steps)
Operation Count	$O(N)$ ($N-1$ additions)	$O(N)$ ($2(N-1)$ additions for up/down)
Required CPUs	1 (Inherently serial)	$O(N)$ (specifically $N/2$) (To achieve $O(\log N)$ time)

2 Task 2: Parallel Algorithms (Scalar Product)

Given: Two vectors, A and B, with $N = 200$ elements. $p = 8$ processors.

2.1 Task 2.1: Sequential Algorithm

```
sum = 0
FOR i = 0 to N-1
    product = A[i] * B[i]
    sum = sum + product
END FOR
RETURN sum
```

2.2 Task 2.2: Parallel Algorithm (8 Processors)

1. **Distribute Work:** Each processor is responsible for $N/p = 200/8 = 25$ elements.
2. **Local Computation (Parallel):** Each processor k calculates its own local scalar product (a partial sum) for its 25 elements.
3. **Global Reduction (Parallel):** The 8 partial sums are combined into a final sum using a parallel reduction.

Scheme:

```
// --- Phase 1 & 2: Local Computation (in parallel by all 8) ---
// Processor 'k' (where k is 0 to 7)
local_sum_k = 0
my_start = k * 25
my_end = my_start + 25
FOR i = my_start to my_end-1
    local_sum_k = local_sum_k + (A[i] * B[i])
END FOR

// --- Phase 3: Global Reduction (in log_2(p) steps) ---
global_sum = REDUCE(local_sum_k, operation=SUM)
RETURN global_sum
```

2.3 Task 2.3: Time Steps

(Assuming 1 time step per operation)

- **Sequential Time (T_1):**
  - N multiplications + $N - 1$ additions
  - $T_1 = 200 + 199 = \mathbf{399}$ time steps.
- **Parallel Time (T_8):**
  - **Local Compute:** (Done by all processors in parallel)
  - $N/p = 25$ multiplications
  - $(N/p - 1) = 24$ additions
  - Time $= 25 + 24 = 49$ steps.
  - **Global Reduction:** Add 8 partial sums.
  - Time $= \log_2(p) = \log_2(8) = 3$ addition steps.
  - $T_8 = 49 + 3 = \mathbf{52}$ time steps.

2.4 Task 2.4: Speed-up and Efficiency ($p = 8$)

- **Speed-up (S_8):** $S_p = T_1/T_p$

$$S_8 = \frac{399}{52} \approx \mathbf{7.67}$$

- **Efficiency (E_8):** $E_p = S_p/p$

$$E_8 = \frac{7.67}{8} \approx 0.959 \text{ (or } \mathbf{95.9}\%\text{)}$$

2.5 Task 2.5: Generalized Functions (N and p)

- **Time (Sequential):**
$$T_1(n) = n + (n-1) = 2n - 1$$

- **Time (Parallel):**
$$T_p(n,p) = \underbrace{\left(\frac{n}{p} + \left(\frac{n}{p} - 1\right)\right)}_{\text{Local Compute}} + \underbrace{(\log_2 p)}_{\text{Global Reduce}}$$
$$T_p(n,p) = \frac{2n}{p} - 1 + \log_2 p$$

- **Speed-up (S_p):**
$$S_p(n,p) = \frac{T_1}{T_p} = \frac{2n-1}{\frac{2n}{p} - 1 + \log_2 p}$$

- **Efficiency (E_p):**
$$E_p(n,p) = \frac{S_p}{p} = \frac{2n-1}{p\left(\frac{2n}{p} - 1 + \log_2 p\right)} = \frac{2n-1}{2n - p + p\log_2 p}$$

3 Task 3: PRAM - Matrix Multiplication

Given $N \times N$ matrices A, B, and C. Sequential time $T_1 = O(n^3)$.

3.1 Task 3.1: EREW PRAM Algorithm

An EREW algorithm requires explicit broadcast phases.

Algorithm (using $p = n^3$ processors $P_{i,j,k}$):

1. **Phase 1: Broadcast Data (EREW)**
   - In parallel: Broadcast $A[i,k]$ to all n processors $P_{i,j,k}$ (for $j = 1..n$).
   - In parallel: Broadcast $B[k,j]$ to all n processors $P_{i,j,k}$ (for $i = 1..n$).
   - EREW broadcast takes $O(\log n)$ time.
2. **Phase 2: Compute Products (Parallel)**
   - In parallel, all n^3 processors $P_{i,j,k}$ compute:
   - $Temp[i,j,k] = A[i,k] * B[k,j]$
3. **Phase 3: Reduce Sums (EREW)**
   - In parallel for all i,j: Perform a parallel sum (reduction) of the n values $Temp[i,j,k]$ (for $k = 1..n$).
   - EREW summation takes $O(\log n)$ time.
   - $C[i,j] = \sum_{k=1}^{n} Temp[i,j,k]$

- **Processors Used:** $O(n^3)$
- **Time Complexity:** $O(\log n) + O(1) + O(\log n) = \mathbf{O}(\log \mathbf{n})$.

3.2 Task 3.2: $O(n^2)$ Processors Algorithm (CREW)

We assign one processor $P_{i,j}$ to compute one element $C[i,j]$. This requires a **CREW PRAM model** because multiple processors read the same rows of A and columns of B.

Algorithm (using $p = n^2$ processors):

1. **Phase 1: Compute Products (Sequential per Processor)**
   - In parallel, each processor $P_{i,j}$ does:
   - `FOR k = 1 to n:`
   - `Temp[k] = A[i,k] * B[k,j]` // *Concurrent Read*
2. **Phase 2: Compute Sum (Sequential per Processor)**
   - In parallel, each processor $P_{i,j}$ does:
   - `C[i,j] = 0`
   - `FOR k = 1 to n:`
   - `C[i,j] = C[i,j] + Temp[k]` // *Exclusive Write*

- **Time Complexity:** $T_p = O(n) + O(n) = \mathbf{O}(\mathbf{n})$.
- **Speed-up (S_p):**

$$S_p = \frac{T_1}{T_p} = \frac{O(n^3)}{O(n)} = \mathbf{O}(\mathbf{n^2})$$

- **Efficiency (E_p):**

$$E_p = \frac{S_p}{p} = \frac{O(n^2)}{O(n^2)} = \mathbf{O}(\mathbf{1})$$

3.2.1 Comparison

The $O(n^2)$ processor algorithm is slower ($O(n)$) than the $O(n^3)$ processor algorithm ($O(\log n)$), but it is **work-optimal** and has an efficiency of $O(1)$, making it a more practical use of resources.

4 Task 4: Distributed Search

Given: A string of length n with unique characters. p processors. Find index of 'c'.

4.1 Task 4.1: Parallel Search Algorithms

All models share the same first step:

1. **Phase 1: Local Search (All Models)**
   - Divide the string into p chunks of size n/p.
   - In parallel, each processor k sequentially searches its local chunk.
   - This takes $O(n/p)$ time (worst case).
   - Each processor k stores its result: `local_index = (found_index or -1)`.
2. **Phase 2: Global Reduction (Model-Dependent)** This phase finds the single non-negative index from the p `local_index` variables. A `global_result` is initialized to -1.

4.1.1 EREW-PRAM Algorithm

- **Phase 2 (EREW Reduction):** A reduction tree is used to find the one valid index. In $\log_2 p$ steps, processors compare results in pairs and pass up the non-negative index.

4.1.2 CREW-PRAM Algorithm

- **Phase 2 (CREW Reduction):** The $O(\log p)$ EREW reduction algorithm is the most efficient method and works on a CREW machine.

4.1.3 CRCW-PRAM Algorithm

- **Phase 2 (CRCW Reduction):** We use the **Arbitrary Write** CRCW model.
  - If processor k finds the character (`local_index` $\neq$ `-1`), it writes its result to the shared `global_result`.
  - `IF local_index` $\neq$ `-1 THEN global_result = i`
  - Since the character is unique, only one processor writes, completing in $O(1)$ time.

4.2 Task 4.2: Time Complexity Analysis

- **EREW-PRAM:**

$$T_p = \underbrace{O(n/p)}_{\text{Local Search}} + \underbrace{O(\log p)}_{\text{EREW Reduce}} = \mathbf{O(n/p + \log p)}$$

- **CREW-PRAM:**

$$T_p = \underbrace{O(n/p)}_{\text{Local Search}} + \underbrace{O(\log p)}_{\text{Best Reduce}} = \mathbf{O(n/p + \log p)}$$

- **CRCW-PRAM (Arbitrary):**

$$T_p = \underbrace{O(n/p)}_{\text{Local Search}} + \underbrace{O(1)}_{\text{CRCW Write}} = \mathbf{O(n/p)}$$