Laborator 5-6

Programare paralela si concurenta

Ciprian Paduraru

1. Pipelines (theory: pipeline latency, cycle, complexity)

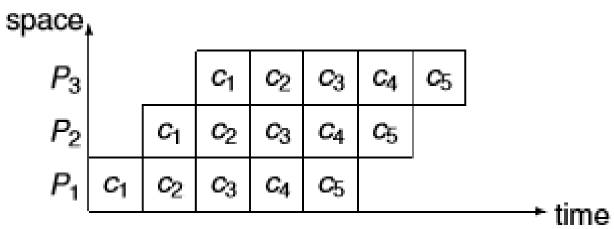


Figure 1. Linear pipeline with 3 stages.

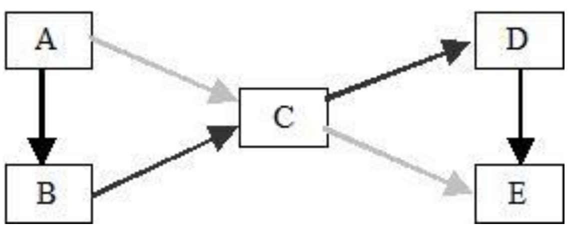


Figure 2. Non-linear pipeline.

Implement: evaluation of a given polynom (with coefficients) for multiple X values in parallel.

Hint: consider Horner evaluation metod f(x) = (((a4x +a3)x +a2)x +a1)x +a0, then use a pipeline for multiplications.

Start with the skeleton code provided in the solution of this lab (1.cpp).

2. Reduce

- Check Course to see the pseudocode of the "Reduce" backend implementation in MPI, and https://www.cs.fsu.edu/~engelen/courses/HPC-adv/PRAM.pdf for PRAM model.

Implement: Given an array of numbers between $[0, 2^{10} - 1]$ on master process, compute the histogram considering 2^4 equal intervals (bins).

Hints:

- Divide the array of numbers among processes using a scatter
- Compute in parallel the histogram for each part.

➢ Use MPI_Reduce with a custom operation to gather the results back.

3. **Map-Reduce** operation in Hadoop / Spark.

A) Pi estimation Monte-Carlo method

sc = spark context

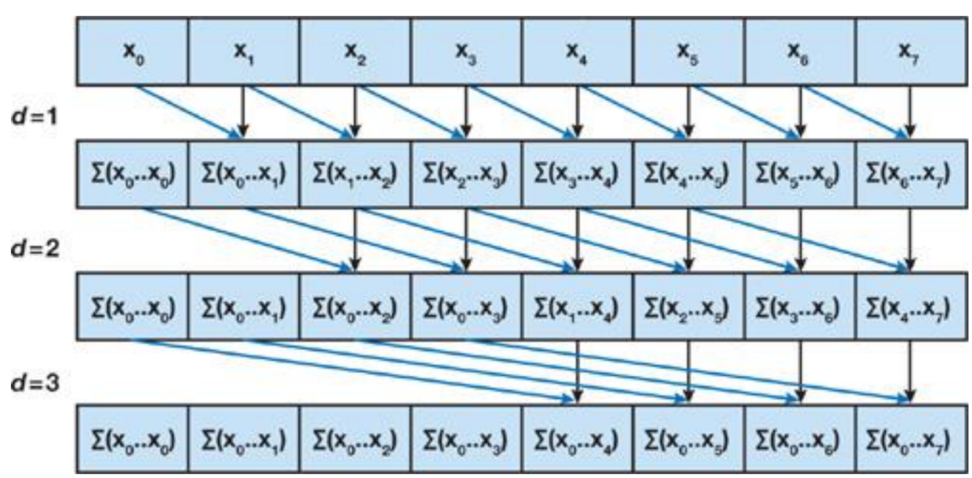
```
def sample():
    x, y = random(), random()
    return 1 if x*x + y*y < 1 else 0

count = sc.parallelize(xrange(0, NUM_SAMPLES)).map(sample) \
             .reduce(lambda a, b: a + b)
print "Pi is roughly %f" % (4.0 * count / NUM_SAMPLES)
```

Word count in a distributed file

```
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

4. **Scan / ExclusiveScan (prefix sum)**

A) Algorithm 1 (Hillis and Steele 1986) – not work efficient, complexity O(NlogN) operations;



```
for d = 1 to log2 n do
    for all k in parallel do
        if k ≥ 2^d then
            x[k] = x [k – 2^(d-1)] + x[k]
```

Note: Needs double buffering when a thread handles more items:

```
for d = 1 to log2 n do
    for all k in parallel do
        if k ≥ 2^d then
            x[out][k] = x[in][k – 2^(d-1)] + x[in][k]
        else
            x[out][k] = x[in][k]
```

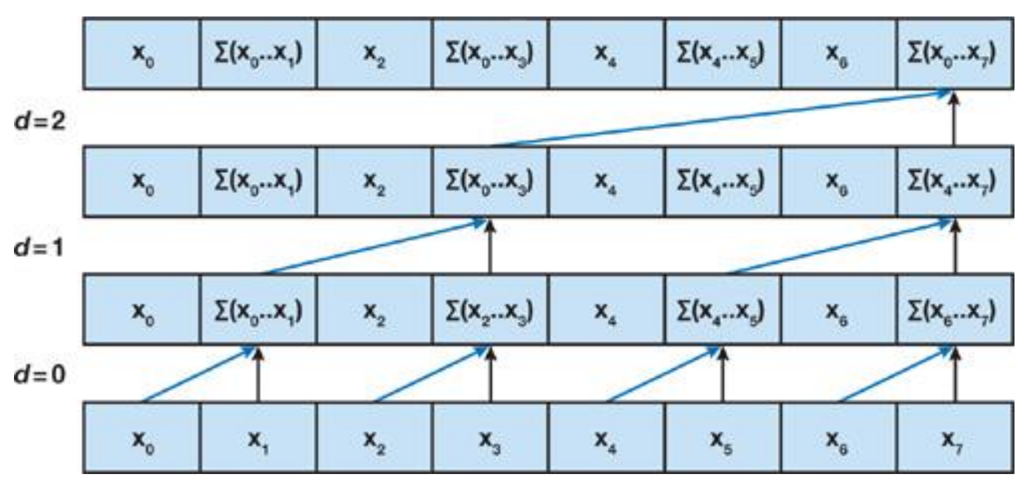
Number of operations: n/2 + n/4 +…. + 1 ~ nlogn.

B) Algorithm 2 (Blelloch 1990) – work efficient, complexity O(N) operations

Main ideas to reduce number of operations to O(N):

- Use a common pattern in parallel computing: computation using a binary tree. Binary tree has logN levels and 2^d nodes at each level
- If each node does a single operation then we have N operations overall
- Do the computation in two steps: (root node of the tree is X[n-1], last element)
  - Step 1: (Up-Sweep) Do a reduce operation that brings data to X[n-1]
  - Step 2: (Down -Sweep): Insert 0 at root then each node running at each level sends its current value to the left child, while the node's value is updated with the old value of the left child + current value at node

Step 1: 2(n-1) operations

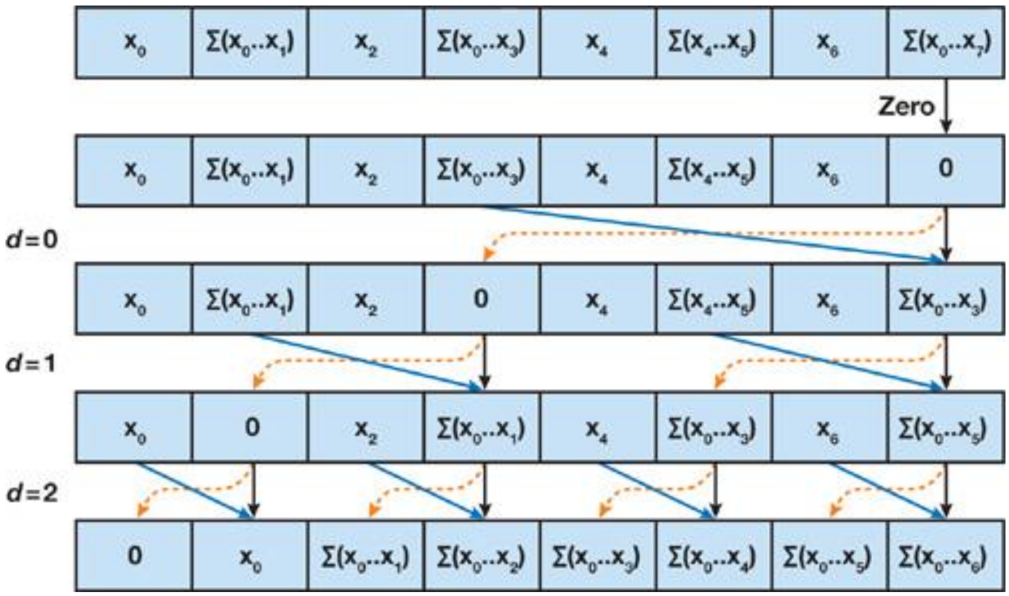


for $d = 0$ to $\log_2 n - 1$ **do**

    for all $k = 0$ to $n - 1$ by 2^{d+1} in parallel **do**

        $x[k + 2^{d+1} - 1] = x[k + 2^{d} - 1] + x[k + 2^{d+1} - 1]$

Step 2: 2(n-1) operations



$x[n - 1] \leftarrow 0$

for $d = \log_2 n - 1$ down to 0 **do**

    for all $k = 0$ to $n - 1$ by $2^{d} +1$ in parallel **do**

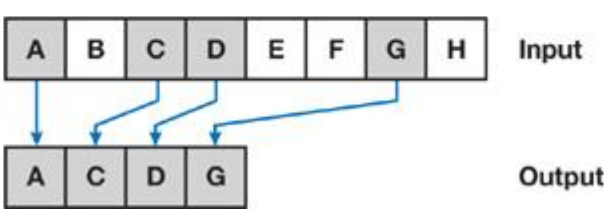
        $t = x[k + 2^{d} - 1]$

        $x[k + 2^{d} - 1] = x[k + 2^{d+1} - 1]$

        $x[k + 2^{d} +1 - 1] = t + x[k + 2^{d+1} - 1]$

5. **Compact pattern**

How do we filter an array of items in parallel?



Reference: http://www.cse.chalmers.se/~uffe/streamcompaction.pdf

6. Radix sort