Chapter 5: Performance Analysis

Elements of Parallel Computing

Eric Aubanel

Work-Depth Analysis

- **Depth**: parallel execution time
- **Work**: sequential execution time to complete all tasks
- **Parallelism**: work/depth. Upper bound on performance gain.

Depth

- Consider all possible paths in task graph
- Depth given as longest time of all paths
- If all tasks take the same time then depth is time of longest path: *critical path*
- No matter how many computational resources, solution cannot be computed in less time than depth

Work Efficiency

Parallel algorithm is **work-efficient** if its work has the same complexity as the best sequential algorithm

Reduction

- work: $O(n)$
  - work efficient
- depth: $O(\log n)$
- parallelism: $O(n/\log n)$

Naive Merge Sort

- work: $O(n \log n)$
- depth: from recurrence relation $d(n) = d(n/2) + O(n)$, gives depth of $O(n)$
- parallelism: $O(\log n)$

Merge Sort with Parallel Merge:

- parallel merge: $O(n)$ work and $O(\log^2 n)$ depth
- work of merge sort still $O(n \log n)$
- depth: recurrence relation for $d(n) = d(n/2) + O(\log^2 n)$, yields $O(\log^3 n)$
- parallelism: $O(n/\log^2 n)$

Hillis and Steele Scan

for $k \leftarrow 0$ *to* $\log n - 1$ **do**
  $j \leftarrow 2^k$
  $\{a[i] \leftarrow a[i-j] + a[i] : i \in [0..n) \mid i \geq j\}$
end

- $n - 2^k$ independent tasks (each doing an addition) at each level
- tasks in one level complete before next level starts
- depth: $O(\log n)$, work: $O(n \log n)$ (not work efficient), parallelism: $O(n)$

Blelloch Scan

Two traversals of reduction tree, so same complexity as reduction: $O(n)$ work, $O(\log n)$ depth, and $O(n/\log n)$ parallelism

Performance Metrics

- Speedup
- Cost
- Efficiency
- Throughput

Speedup

- *relative speedup*: ratio of the execution time of the parallel program on one core to the time on p cores
- *absolute speedup*: ratio of the execution time of the best known sequential algorithm to the execution time of the parallel algorithm on p cores.

Cost

Cost: product of the asymptotic execution time and the number of cores.

- Not the same as work!
- Reduction: $O(n)$ work. If $p = n/2$ cores used, cost is $O(p \log p)$. Reduction is work efficient but is not cost efficient
- Reduction: if fewer cores used, then each core first sums n/p before parallel reduction. This takes $O(n/p + \log p)$ time, so cost is $O(n + p \log p)$. If n is greater than $p \log p$ then this algorithm is cost efficient and its speedup is $O(n/(n/p)) = O(p)$, which is optimal.

Efficiency

- *Efficiency*: ratio of the speedup to the number of cores.
- Also given by ratio of the execution time of the sequential algorithm to the cost

Throughput Metrics

- When sequential time not possible or meaningful, such as for GPU execution
- E.g.: floating point operations per second (FLOPS), cell updates per second, ...
- *Operation count must be based on best known sequential algorithm*

Comparing Scan Algorithms

- Hillis and Steele with $p = n$ cores: $\log n$ time and $n/\log n$ speedup
- Blelloch with $p = n/2$ cores: $n/(2\log n)$ speedup.
- Both algorithms cost-inefficient, at $O(n\log n)$
- Better to use fewer cores, as for reduction

SPMD Shared Memory Scan

```
// assumes n divisible by p
shared a, b
start ← id * n/p
sum ← 0
for j ← 0 to n/p − 1 do
    sum ← sum + a[start + j]
    a[start + j] ← sum
end
b[id] ← a[(id + 1) * n/p − 1] // sum of all values in
   sub-array
scan(b)
if id > 0 then
    for j ← 0 to n/p − 1 do
        a[start + j] ← a[start + j] + b[id − 1]
    end
end
```

Analysis of SPMD Scan

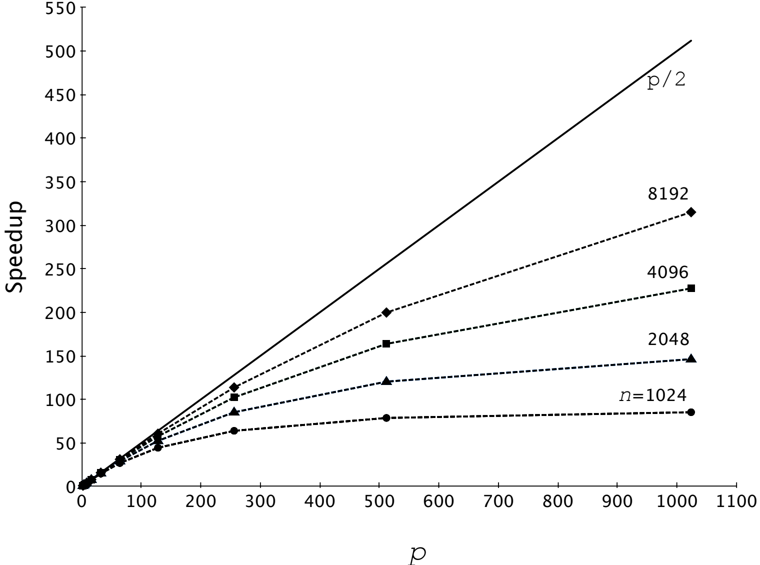
Parallel execution time:

- Hillis and Steele: $2\lceil n/p \rceil + \log p$
- Blelloch: $2\lceil n/p \rceil + 2\log p$

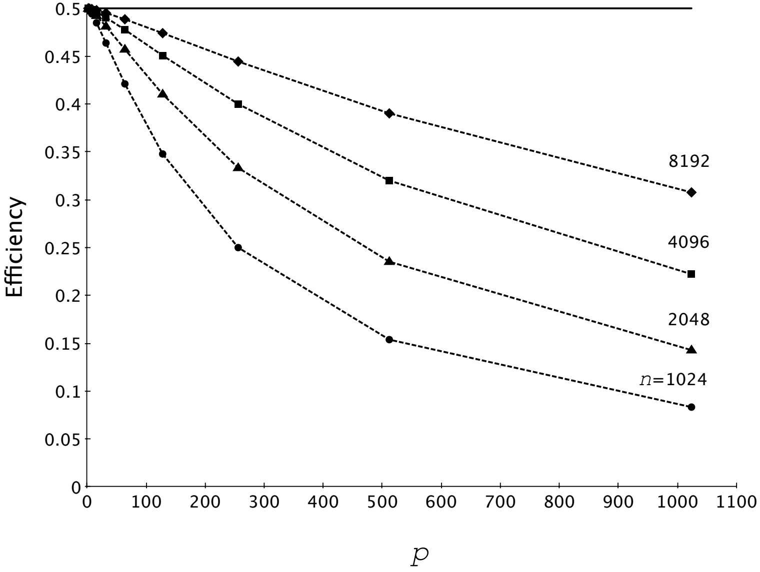
Cost is $O(n + p\log p)$, so cost efficient if n is greater than $p\log p$

Speedup of SPMD Hillis and Steele scan

$$n/(2\lceil n/p \rceil + \log p)$$



Efficiency of SPMD Hillis and Steele scan

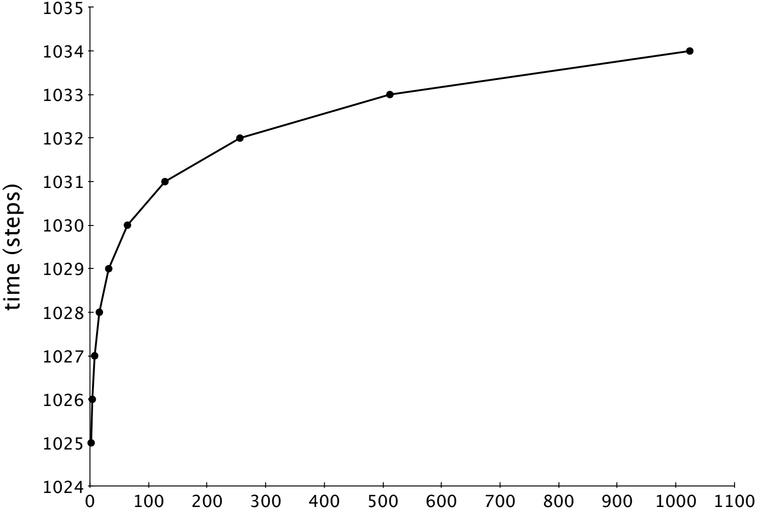


Strong and Weak Scaling

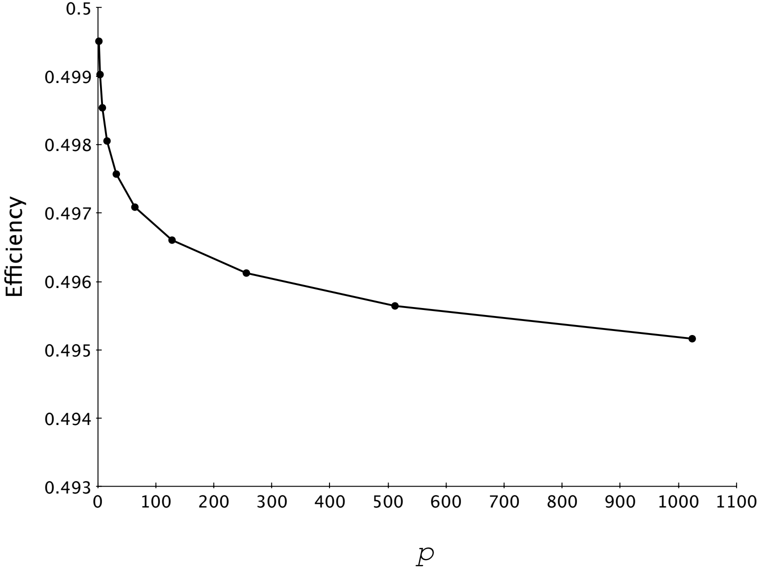
- *Strong scaling*: problem size fixed as number of cores increases
- *Weak scaling*: amount of work per core is kept fixed as number of cores increases

Weak Scaling of SPMD Hillis and Steele

Problem size scaled as $n = 512p$



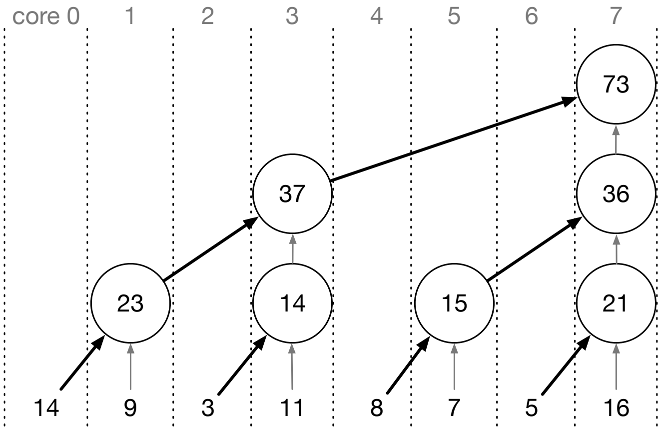
p


Efficiency
0.5
0.499
0.498
0.497
0.496
0.495
0.494
0.493
0
100
200
300
400
500
600
700
800
900
1000
1100
p

Communication Analysis

- Communication over a link: $t_{comm} = \lambda + m/\beta$, for HW/SW latency λ and bandwidth β.
- Communication topology of application: *virtual topology*
- At runtime, virtual topology embedded on the physical topology

Communication Analysis: Reduction



$t_{\text{reduction}} = (\sigma + \lambda + m/\beta)\log p = O(m \log p)$, σ is time to add two values and m is size of a value.

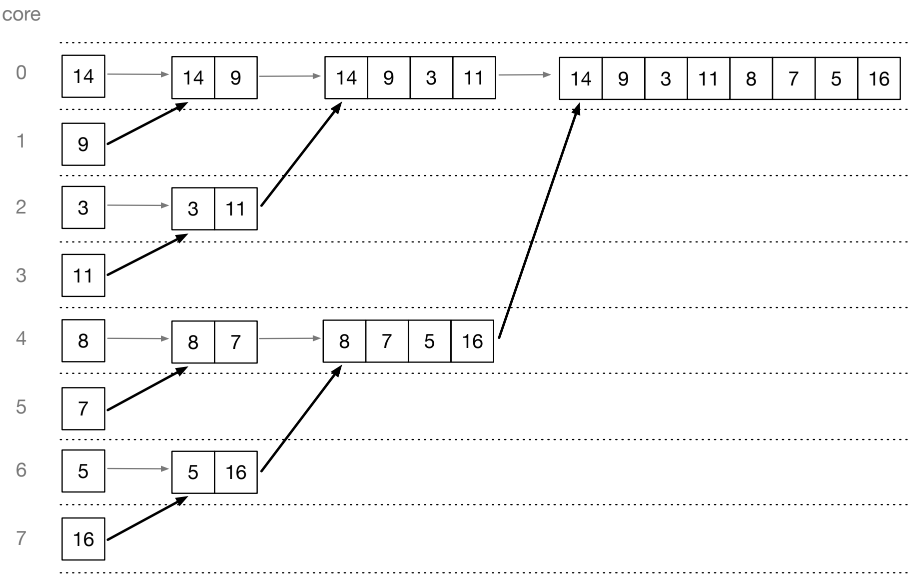
Analysis of Game of Life

- 1D decomposition: each core sends 2 messages of size n, $t_{\mathrm{1D}} = O(n^2/p + n)$
- 2D decomposition: each core sends 4 messages of size $n/\sqrt{p}$, $t_{\mathrm{2D}} = O(n^2/p + n/\sqrt{p})$
- Both decompositions are cost efficient
- 2D decomposition has lower time and better weak scaling (increasing n as a function of $\sqrt{p}$), with an efficiency of $O(1)$

Analysis of 1D Matrix-Vector Multiplication

- communication: gather result vectors and broadcast gathered vector
- Broadcast has same topology as reduction, $t_{\text{broadcast}} = (\lambda + n/\beta)\log p = O(n \log p)$
- Gather is similar, but message size increases at each step

Gather


core
0
1
2
3
4
5
6
7
14
9
3
11
8
7
5
16
14 9
3 11
8 7
5 16
14 9 3 11
8 7 5 16
14 9 3 11 8 7 5 16

Gather Analysis

// `Assumes` $p = 2^i$ `and` $n \bmod p = 0$

for $i \leftarrow 0$ *to* $\log p - 1$ **do**

  if $id \bmod 2^i = 0$ **then**

    if $id/2^i \bmod 2 = 1$ **then**

      send array of length $2^i n/p$ to $id - 2^i$

    else

      receive array starting at position $(id + 2^i)n/p$ from $id + 2^i$

    end

  end

end

- $\log p$ steps and the message size at step i is $2^i n/p$
- $t_{\text{gather}} = \lambda \log p + n(p-1)/(p\beta) = O(n + \log p)$

1D vs. 2D Matrix-Vector Multiplication

$$t_{\mathrm{1D}} = O(n^2/p + n\log p)$$

$$\begin{aligned} t_{\mathrm{2D}} &= O(n^2/p + (n/\sqrt{p})\log\sqrt{p} + (n/\sqrt{p})\log\sqrt{p}) \\ &= O(n^2/p + (n/\sqrt{p})\log p) \end{aligned}$$