CS 498
Hot Topics in High Performance Computing

Networks and Fault Tolerance

7. Network Topologies

Intro

- What did we learn in the last lecture
  - Optimal 1-item scatter in LogGP
    - k-item scatter is an open problem
  - Showed benefits over LogP model
  - Measuring LogGP parameters (not in exam)
- What will we learn today
  - Introduction to network topologies
  - Parallel sorting

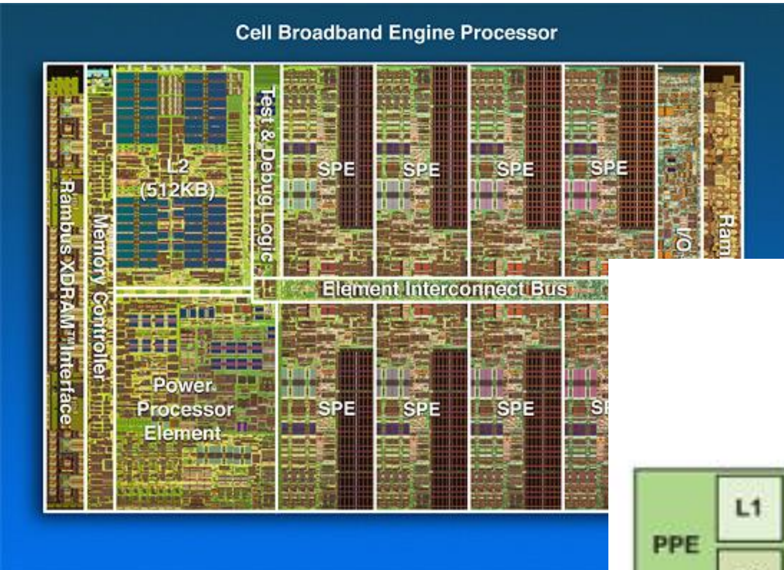
Section IV: Topology

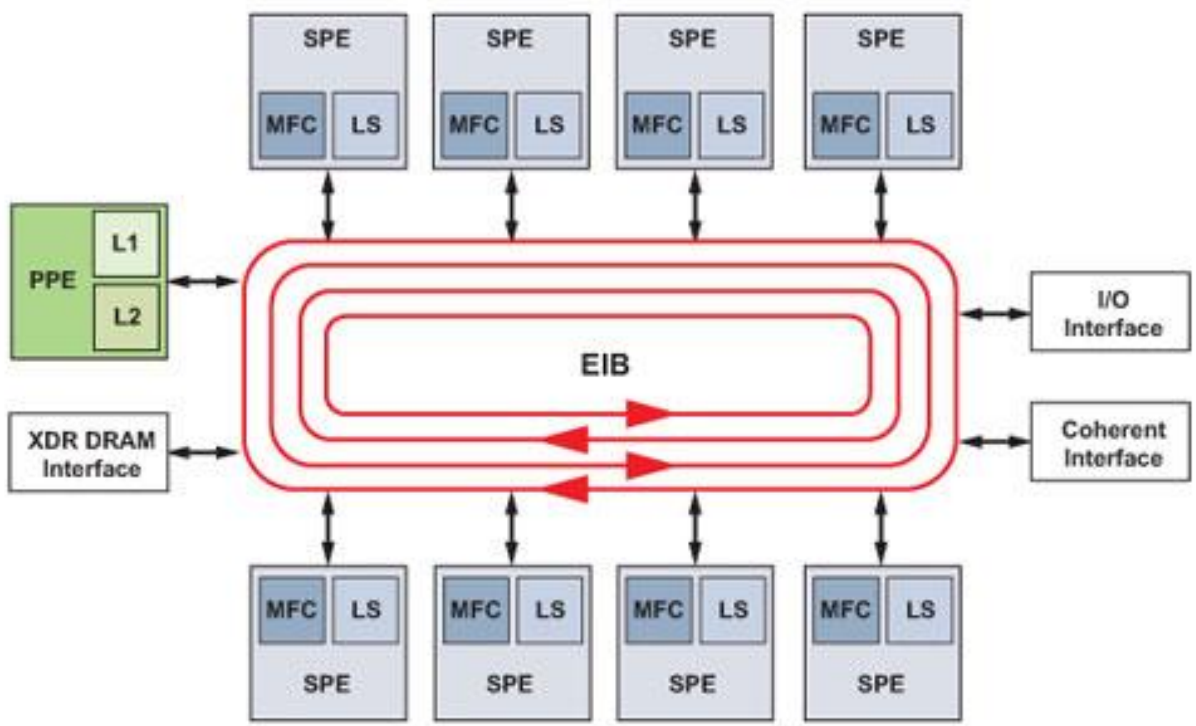
- The structure in which PEs are connected is called network topology

- Examples: Array, Mesh, Torus, Hypercube, cube-connected-cycles (ccc), tree, fat-tree, k-ary n-cube, k-ary n-tree, de Bruijn network, Kautz graph, random ☺

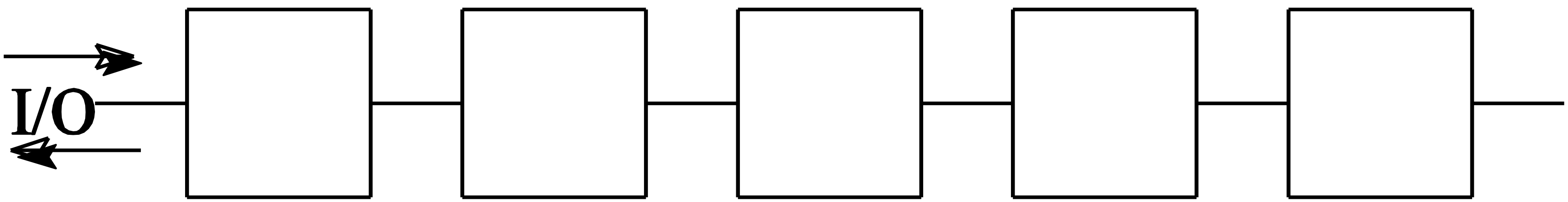
… but let's start slowly …

Practical Example: Cell B.E.





Linear Array



- Each PE has left and right neighbor
  - Leftmost PE is assumed to manage I/O
- Each PE has a simple control program and small local storage
  - Receive; Read local; Process; Send; Write local
  - PEs are assumed to operate synchronously
- Sometimes called “systolic computation”

Simple Sorting on a Linear Array

- Sort N elements on N-PEs
- Algorithm:
  1. Read input from left neighbor
  2. Compare input with stored value
  3. Output larger value to right neighbor
  4. Store smaller value locally

- Example: sort {3,1,4,2} on a 4-PE Array

Linear Array Sort Runtime

- Leftmost PE holds on to the smallest element and passed N-1 on
  - Class Question: After how many steps does the algorithm terminate?

Linear Array Sort Runtime

- Leftmost PE holds on to the smallest element and passed N-1 on
  - PE 1 passes N-1, PE I passes N-i on
  - All elements are in place after 2N-1 steps
    - $\Theta(N)$
- Parallel Speedup?
  - Class Question: What is the best serial (comparison-based) algorithm and what is the parallel speedup of the proposed algorithm?

Linear Array Sort Runtime

- Leftmost PE holds on to the smallest element and passed N-1 on
  - PE 1 passes N-1, PE I passes N-i on
  - All elements are in place after 2N-1 steps
    - $\Theta(N)$
- Parallel Speedup?
  - Best serial (comparison-based) algorithm: $\Theta(N \log N)$
  - Speedup S= $\Theta(\log N)$

Maximum Speedup?

- Class Question: ”What is the maximum achievable speedup with P processing elements and why?”

Maximum Speedup?

- Class Question: ”What is the maximum achievable speedup with P processing elements and why?”

- Answer: “P, since a serial computer can emulate a single step on a parallel computer with P PEs in P steps”

Performed Work

- Work is the product of runtime and the number of processors used W=TP
  - Accounts for parallel inefficiency

- Class Question: "What is the work performed by our sorting algorithm?"

Performed Work

- Work is the product of runtime and the number of processors used W=TP
  - Accounts for parallel inefficiency

- Class Question: “What is the work performed by our sorting algorithm?”

- Answer: “$W = \Theta(N^2)$, N PEs compute for time $\Theta(N)$“

Parallel Efficiency

- The efficiency is the ratio of the speedup to the number of PEs E=S/P
  - How effectively is the parallel machine used?

- Class Question: “What is the efficiency of our linear array sort?”

Parallel Efficiency

- The efficiency is the ratio of the speedup to the number of PEs E=S/P
  - How effectively is the parallel machine used?
  - Should be close to 1!

- Class Question: “What is the efficiency of our linear array sort?”

- Answer: “$\Theta\left(\frac{\log N}{N}\right)$, very poor for large N”

Sorting with less PEs

- Sorting N elements with P=N PEs is impractical
  - Usually P<N
- General solution: simulating N PEs with P<N PEs
  - Each processor simulates N/P original processors
  - Induces slowdown of N/P but same efficiency
- Sorting N elements on P PEs
  - In time $\mathcal{O}(N^2/P)$ (serial Bubblesort)

Lower Bounds

- Is our $\Theta(N)$ sorting algorithm optimal?

- Argument 1: Yes, it needs at least N steps to input or output the N elements!
  - Well, this could be changed if each PE had an input
- Class Question: More arguments for it?

Lower Bounds

- Is our $\Theta(N)$ sorting algorithm optimal?
- Argument 1: Yes, it needs at least N steps to input or output the N elements!
  - Well, this could be changed if each PE had an input
- Argument 2: Yes, a number might need to travel N steps to get to its right position.
  - This is called "network diameter" and a common lower bound

Lower Bounds

- Class Question: More arguments for it?

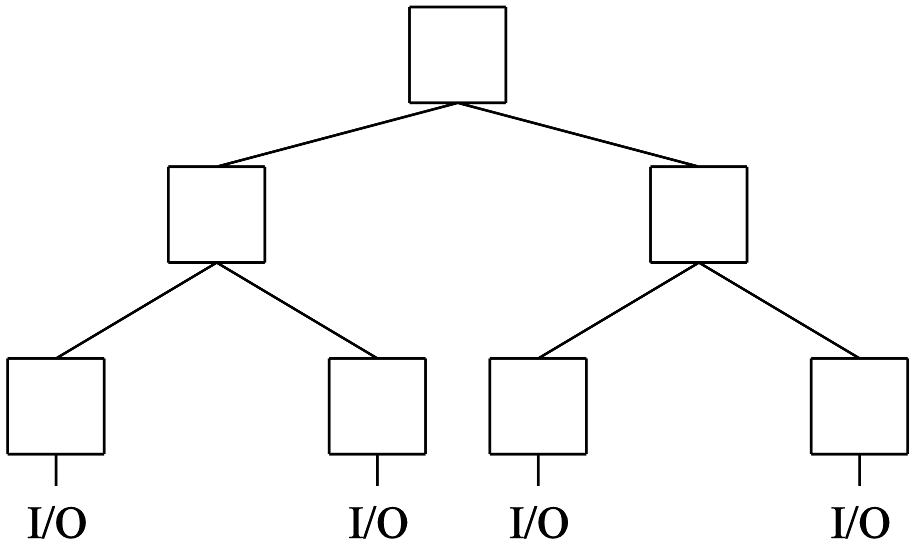
Lower Bounds

- Argument 3: Half of the elements might need to “switch sides”.
  - This is called “bisection width”, i.e., the number of cables that need to be removed in order to cut the network
- Class Question: “What is the bisection width of a linear array?”

Lower Bounds

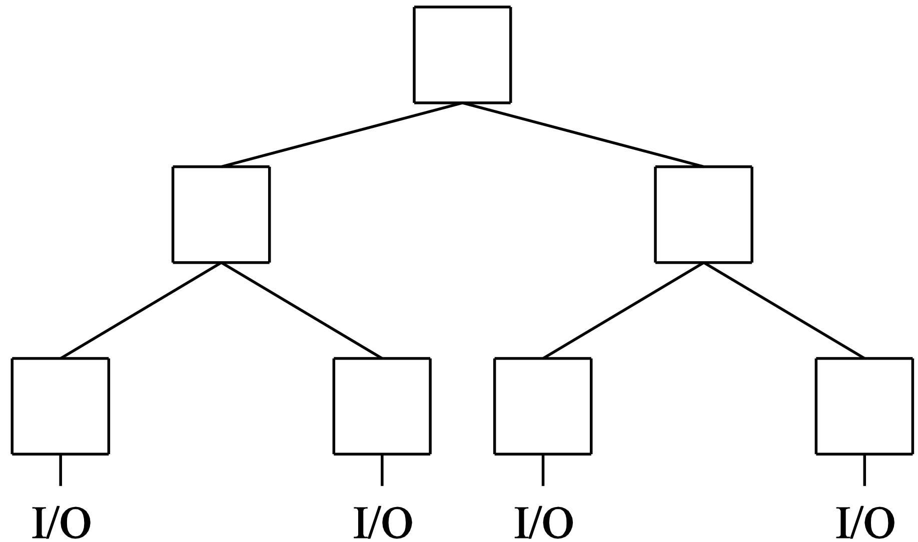
- Argument 3: Half of the elements might need to "switch sides".
  - This is called "bisection width", i.e., the number of cables that need to be removed in order to cut the network
- Class Question: "What is the bisection width of a linear array?"
  - Yes, 1!
- Conclusion: $\Theta(N)$ sorting is optimal on linear arrays

Sorting on a Binary Tree



- N inputs at leaves of the tree
- Class Question: “What are diameter and bisection width of the tree with regards to N?”

Sorting on a Binary Tree



- N inputs at leaves of the tree
- Class Question: “What are diameter and bisection width of the tree with regards to N?”
  – Yes, 2*log(N) and 1!

Lower Bounds on a Tree

- Class Question: “What is the minimal time needed to sort N values on a tree?”

Lower Bounds on a Tree

- Class Question: “What is the minimal time needed to sort N values on a tree?”
  - Yes, $\Theta(N)$
- Counter-example: 1-Bit Sort on a Tree.
  - Assuming 1-bit input
  - Runtime: $\mathcal{O}(\log N)$
  - How? Number of 1-bits are counted upwards the tree and broadcast to all leaves. This requires only $\mathcal{O}(\log N)$ bits to cross the bisection!
  - It’s not comparison-based though!

Revisited: Sorting on a Linear Array

- Values are initially distributed to all N PEs
- Odd/Even Transposition Sort (aka Bubble Sort):
    1. Odd steps: compare values in 1,2; 3,4; … and switch
    2. Even steps: compare values in 2,3; 4,5; … and switch

- Example: sort {3,1,4,2}

- Class Question: “What is the worst-case input? And how many iterations does it take?”

The 0-1 Sorting Lemma

- *The 0-1 Sorting Lemma: "If an oblivious comparison-exchange algorithm sorts all input sets consisting solely of 0s and 1s, then it sorts all input sets with arbitrary values"*
  - Oblivious: output of comparisons cannot depend on other comparisons!
- Proof by contradiction:
  - Assume oblivious comparison-sort algorithm which fails to sort some inputs $x_1, x_2, x_3, \ldots, x_n$. Let π be the correctly sorted permutation.
  - Let σ be the permutation returned by the sorting algorithm.
  - Let k be the smallest value such that $x_{\sigma(k)} \neq x_{\pi(k)}$
    - This means: $x_{\sigma(i)} = x_{\pi(i)}$ for i<k and $x_{\sigma(k)} > x_{\pi(k)}$ and there will be an r>k with $x_{\sigma(r)} = x_{\pi(k)}$
  - Let $x_i^* = \begin{cases} 0 \ if \ x_i \leq x_{\pi(k)}, \\ 1 \ if \ x_i > x_{\pi(k)} \end{cases}$

0-1 Sorting Lemma Cont.

- Apply sort to x^*
  - Since $x_i \geq x_j \Rightarrow x_i^* \geq x_j^*$ for all i,j, the algorithm performs the same comparison/exchange operations as for x
  - Thus, the output permutation will be

$$x^*_{\sigma(1)}, x^*_{\sigma(2)}, \ldots, x^*_{\sigma(k-1)}, x^*_{\sigma(k)}, \ldots, x^*_{\sigma(r)}, \ldots = 0, 0, \ldots, 0, 1, \ldots, 0, \ldots$$

  and is thus also incorrect! q.e.d.

- The 0-1 sorting lemma is very simple and powerful
  - Can be used to proof correctness of sorting algorithms!

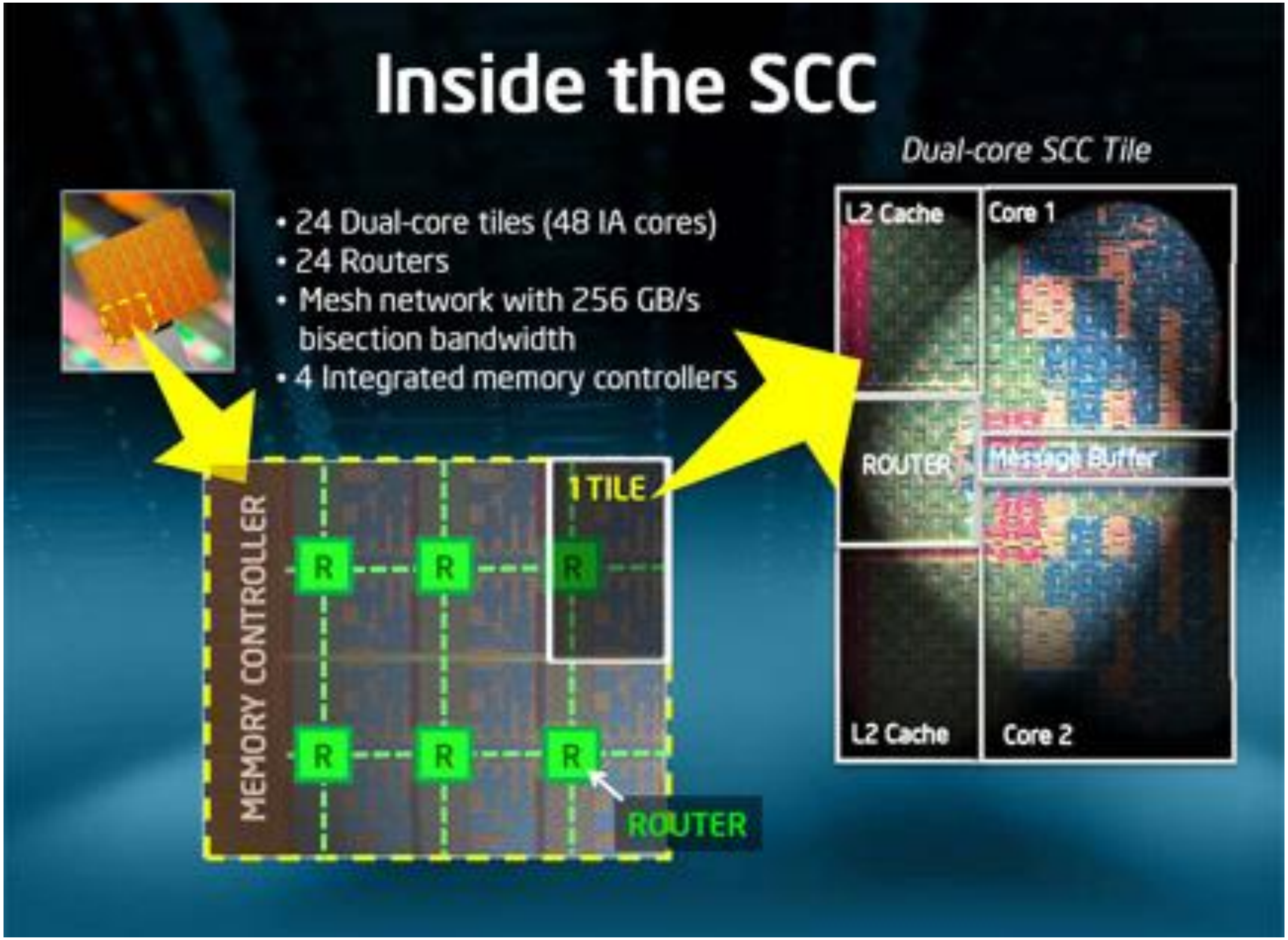
Back to Odd/Even Transposition Sort

- Prove correctness and time-bound with 0-1 sorting lemma
  - Assuming arbitrary string of N-k 0s and k 1s
  - Need to show that after N steps, all k 1s are moved into cells N-k+1, N-k+2,…,N.
- Example: sort {1,1,1,0,0}
  - Observation: 1s move only right and 0s only left
  - Rightmost 1 moves right at each step until it reaches position N

Odd/Even Transposition Sort

- 2nd rightmost 1 moves each step to the right after step 2
  - Its movement can never be blocked by the rightmost 1 since it's moving too at each step!
- General: ith rightmost 1 begins moving right after step i
- → kth rightmost 1 starts moving at step k and reaches position N-k+1 after at most N-k steps
- → the array is sorted after at most N steps!

Example: Intel SCC



Shearsort - Sorting on an Array

- Assuming $\sqrt{N} \times \sqrt{N}$ array
- Sorts in $\sqrt{N}(\log N + 1)$ phases
  - Sort all rows in phases 1,3,…,$2\log(\sqrt{N}) + 1$
  - Sort all columns in phases 2,4,…, $2\log(\sqrt{N})$
  - Column sort moves smaller numbers upwards
  - Odd rows move smaller numbers left, even rows right
- Example: 4x4 array sort!

Shearsort – Runtime & Correctness

- Runtime: $\sqrt{N}(2\log(\sqrt{N})+1) = \sqrt{N}(\log N + 1)$
  - Class Question: "What are speedup and efficiency for Shearsort?"
- Correctness:
  - Apply 0-1 sorting lemma
  - Example: 4x4 0-1 Shearsort
  - Each step has "clean" rows (either all 0 or all 1) and "dirty" rows (0s and 1s in one row)
  - Look at row- and column-sort step of the algorithm

Shearsort – Runtime & Correctness

- Outcomes after row-sort:

```
0....01..1 0..01....1  0...01...1
1..10....0 1....10..0  1...10...0
```

more 0s | more 1s | equal

- Outcomes after column sort:

```
0........0 0.01..10.0  0........0
1.10..01.1 1........1  1........1
```

more 0s | more 1s | equal

Shearsort – Runtime & Correctness

- At least one of two columns becomes “clean”
  - Reduces number of unclean columns to ½
- After $2\log(\sqrt{N})$ phases, only one unclean column is left which is sorted in the additional column-sort phase
  - need to sort all columns because we don’t know which
- In each phase, columns or rows can be sorted with odd/even transposition sort in time $\sqrt{N}$
  - Overall time can be improved by recognizing that columns/rows are approximately sorted!

Lower Bound for Sorting on a Mesh

- Simple lower bound: $2\sqrt{N} - 2$
  - Element might move from (1,1) to ($\sqrt{N}, \sqrt{N}$)
  - Needs, $2\sqrt{N} - 2$ steps to go there
- Class Question: “What about the bisection?”

Lower Bound for Sorting on a Mesh

- Simple lower bound: $2\sqrt{N} - 2$
  - Element might move from (1,1) to ($\sqrt{N}, \sqrt{N}$)
  - Needs, $2\sqrt{N} - 2$ steps to go there
- Class Question: "What about the bisection?"
  - Bisection width: $\sqrt{N}$
  - Communication volume: $N/2$ $\rightarrow \sqrt{N}/2$ rounds!
- Relatively tight bound: $3\sqrt{N} - o(\sqrt{N})$
  - Detailed proof omitted