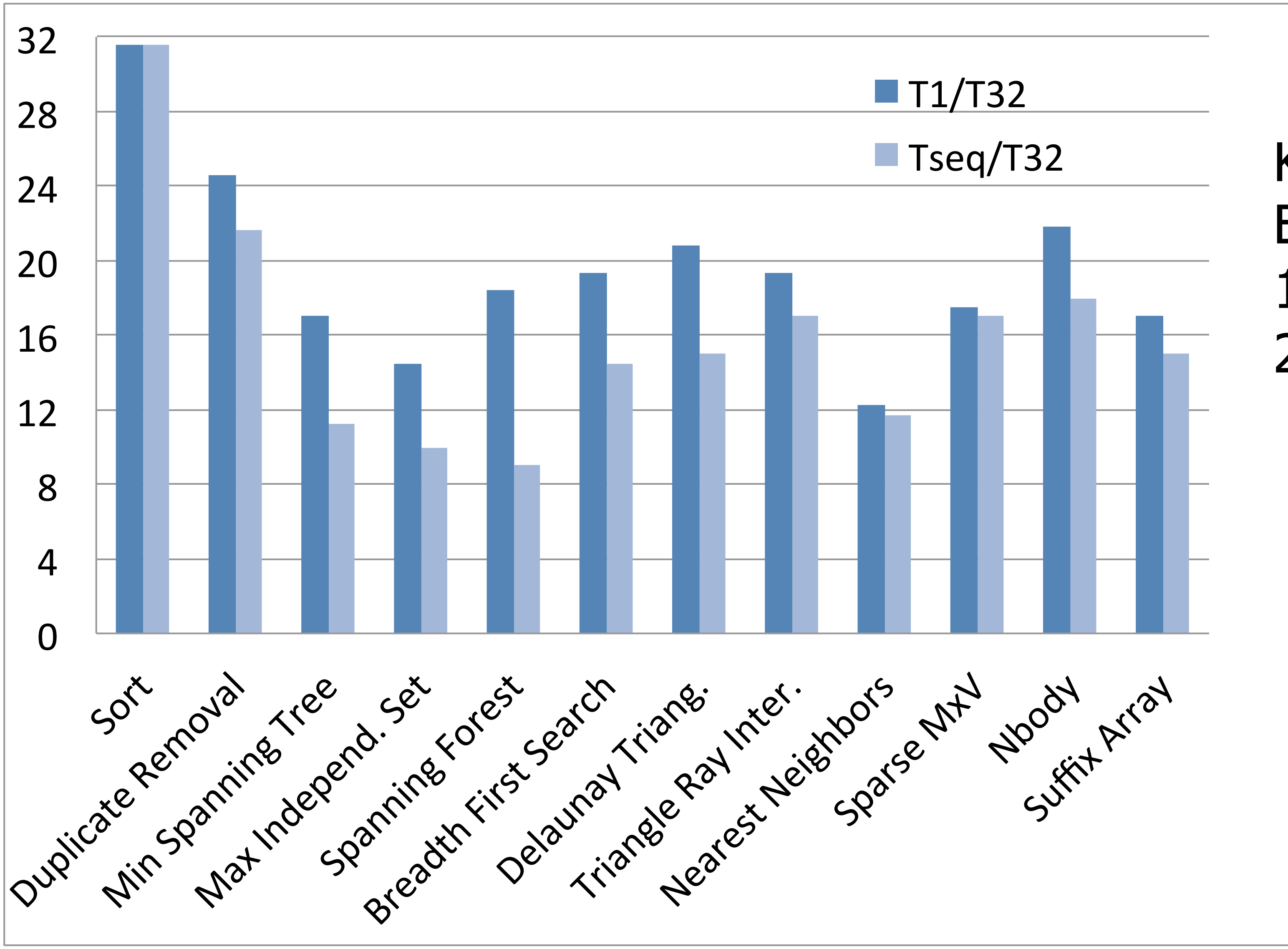
Research Directions in Parallel Algorithms

Three of many:

1. Locality aware parallel algorithms
2. Interaction of languages and algorithms
3. Work efficient algorithms

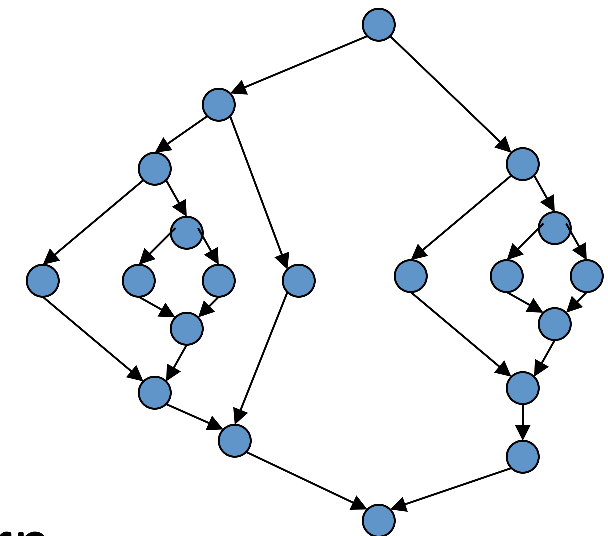
Problem-Based Benchmark Suite



Key to good Efficiency

1. Work
2. Locality

“Efficient” Parallel Algorithms



NC Algorithms

- Polylogarithmic depth/span
- Polynomial work/size

Not a good model because

- **Firstly**: Work is primary concern
- **Secondly**: parallelism (work/depth) is what is important. Polylogarithmic depth is not necessary.

Why is work important

- Energy is proportional to work*
- Rental cost is proportional to resources used (e.g. Amazon EC2)
- Importance of scaling down

*Note : “work” can be a variety of types of operations: e.g. cache misses

What is a good definition?

An algorithm is efficient if:

- Polynomial parallelism
- Optimal work?
  Or no more work than best sequential algorithm?

Example:
Single Source Shortest Paths

In NC:

– Can use Matrix multiply
– O(M(n) log n) work, O(log n) depth

Best work-efficient algorithm

– O(m + n log n) work, O(n) depth
– For sparse graphs only O(log n) parallelism

Is there an algorithm with O(m + n log n) work and O(n^{ε}) parallelism ($\varepsilon > 0$)?

Other Examples

Constant factors in work, e.g. $(1+\varepsilon)$ factor

Worked hard to get MIS, MST and BFS to do little more work than sequential algorithm.

Matrix Inversion:

Is there an algorithm that can invert in $O(n^{1-\varepsilon})$ depth and $O(M(n))$ work?

Integer Sort: sort n integers in range $[0..n^k)$

– Work = O(1/epsilon n), Depth(n^epsilon)