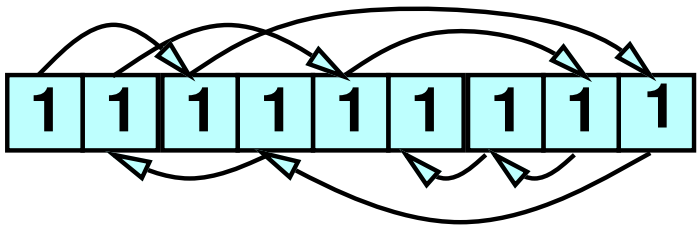
List Ranking and List Scan on the Cray Y-MP/C90

Margaret Reid-Miller
School of Computer Science
Carnegie Mellon University
mrmiller@cs.cmu.edu

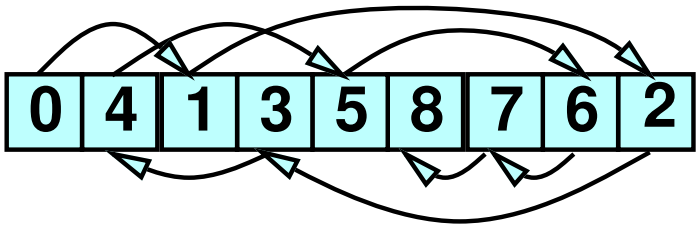
SPAA'94

Carnegie Mellon

List Ranking



Input



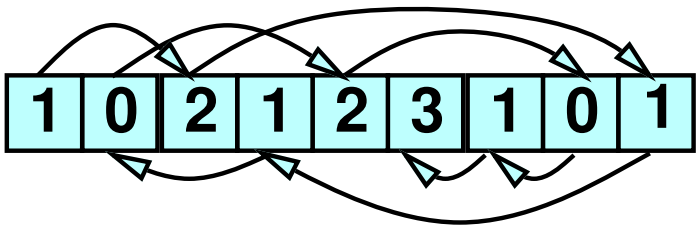
List Rank

Primitive for parallel algorithms:

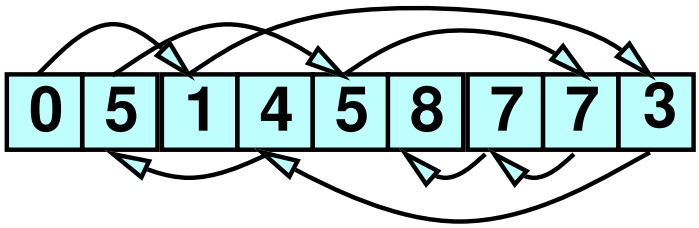
- Ordering the elements of a linked list,
- Finding the Euler tour of a tree,
- Load balancing,
- Contention avoidance,
- Parallel tree contraction,

List Scan

List Scan (parallel prefix) computes for each vertex the “sum” of prior vertices in a linked list. where “sum” is a binary associative operator.



Input



List Scan

Can we beat an Alpha workstation?

Yes!

($>$ 300 times faster on 8 processor C-90)

But it is not easy.

A Good Implementation is Hard

It is hard for a parallel algorithm to compete with serial list ranking:

- Serial algorithm trivial
- Low computation / communication ratio
- Irregular
- Break symmetry
- Load balancing

List Ranking Algorithms

Alg.	Time	Work	Const.
Serial	$O(n)$	$O(n)$	small
Wyllie	$O(\frac{n \log n}{p})$	$O(n \log n)$	small
Ours	$O(\frac{n}{p} + (\log n)^2)$	$O(n)$	medium
Random	$O(\frac{n}{p} + \log n)$	$O(n)$	large
Optimal	$O(\frac{n}{p} + \log n)$	$O(n)$	huge

Why use a Cray C-90?

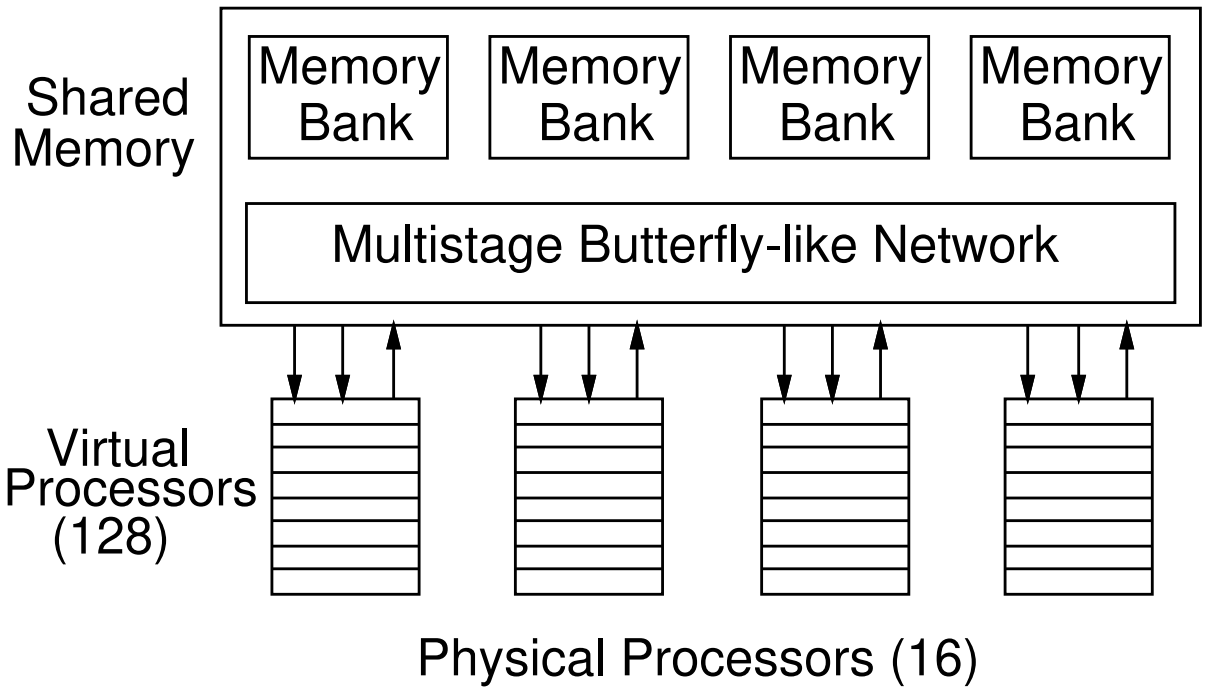
Because it is fast and has extremely high memory bandwidth!

Claim: Of existing multiprocessors Cray vector multiprocessors most closely resemble PRAMs.

Terminology

Parallel	Vector
virtual processor	vector element
bulk synchronous	vector pipeline
direct addressing	load/store
indirect addressing	gather/scatter

Vector Multiprocessors as PRAMs



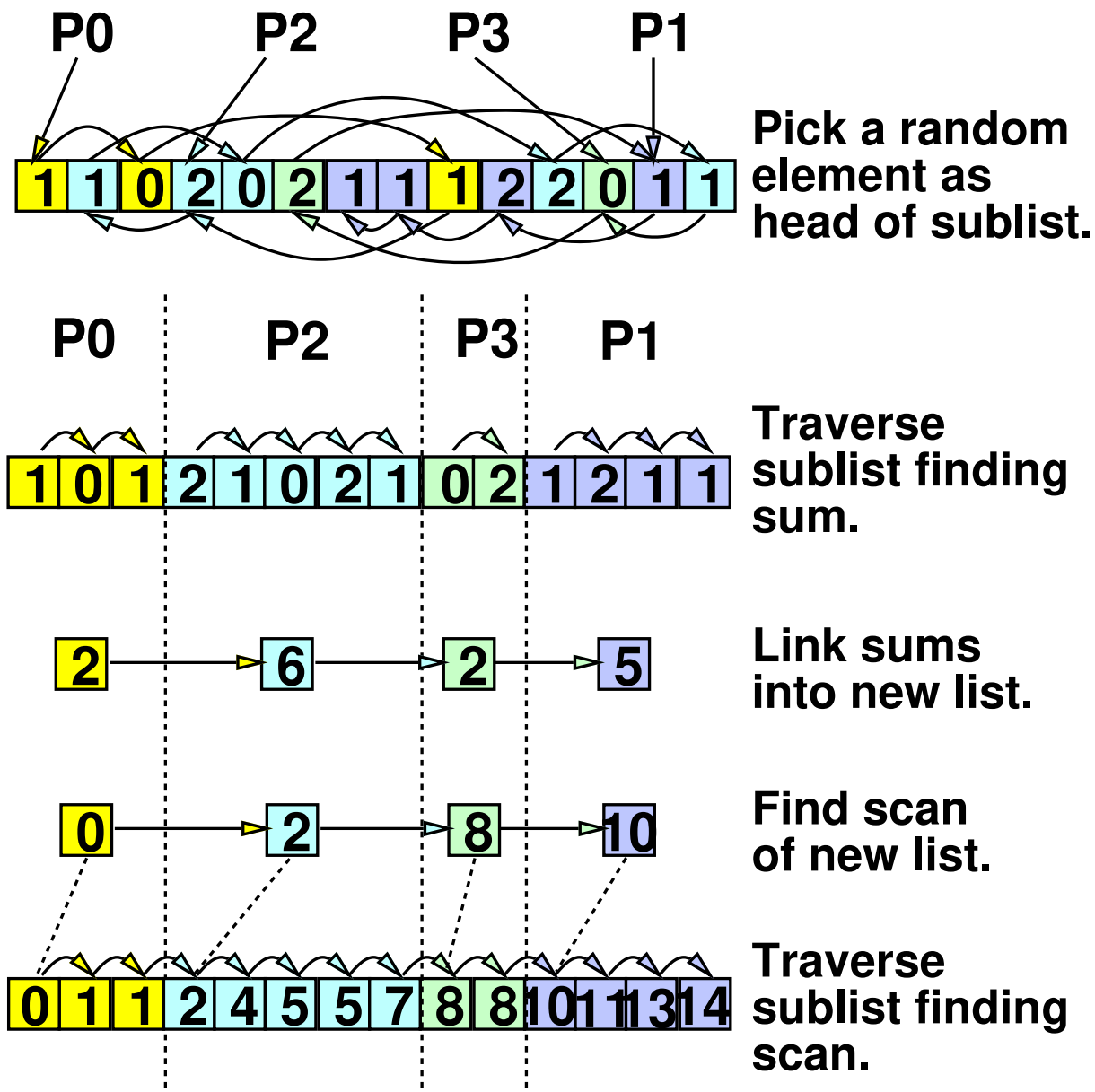
16 physical processors x 128 vector elements
= 2048 virtual processors

If we can

- avoid memory bank conflicts
- hide latencies through parallel slackness

then vector multiprocessors are like PRAMs.

Blelloch/Reid-Miller List Scan



Main Subroutines are Simple

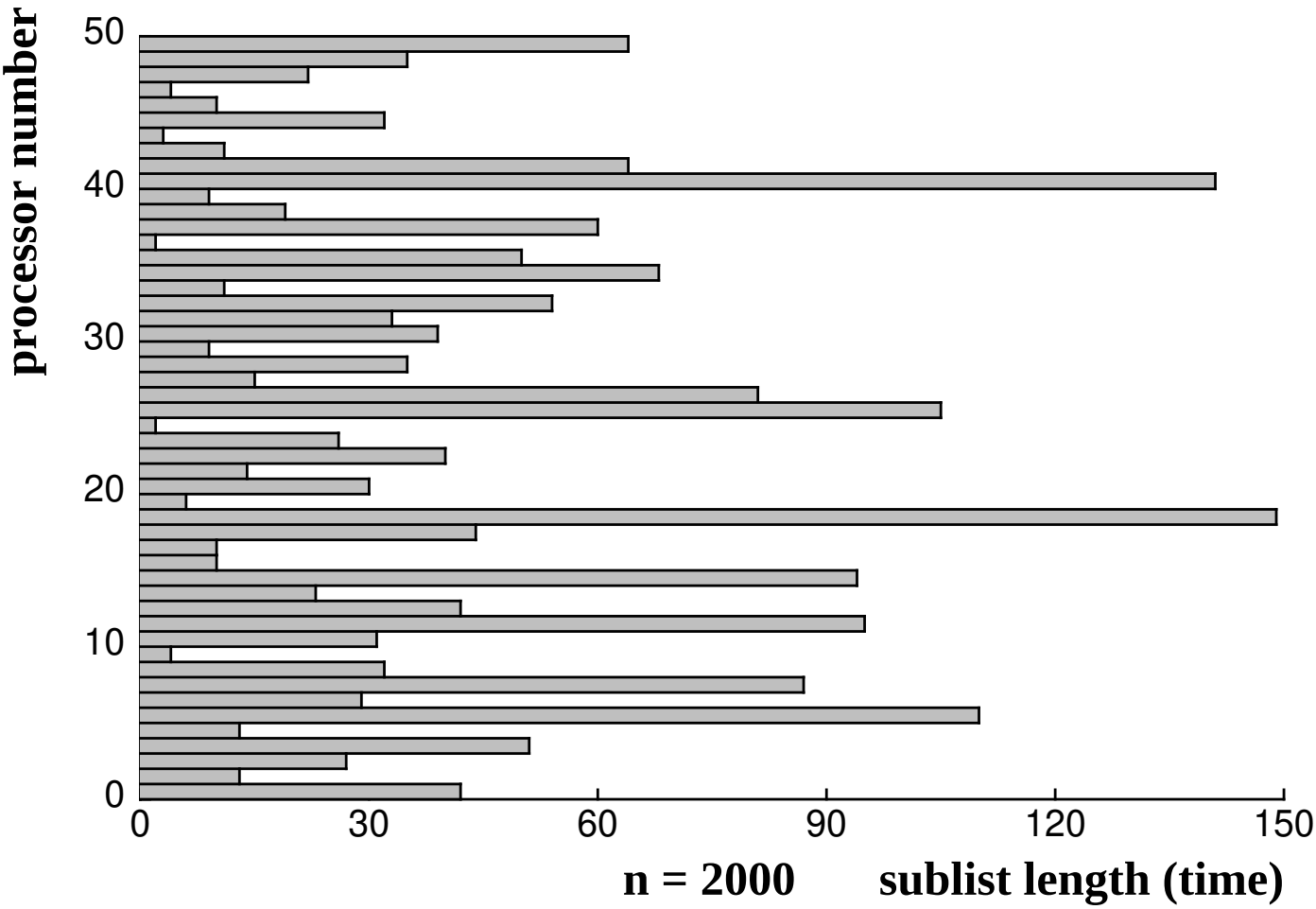
Initial_Rank $(list, sum, next, m, n_links)$ $\{$

```
  for (j = 0; j < n_links; j++)
    for (i = 0; i < m; i++) {      /* Vectorized */
      sum[i] += list.value[next[i]];
      next[i] = list.next[next[i]];
    }
}
```

- Same operations as serial algorithm and no more,
- Applied to every vertex,
- Similar loop for final phase,

But need to load balance periodically.

Vectorized List Rank



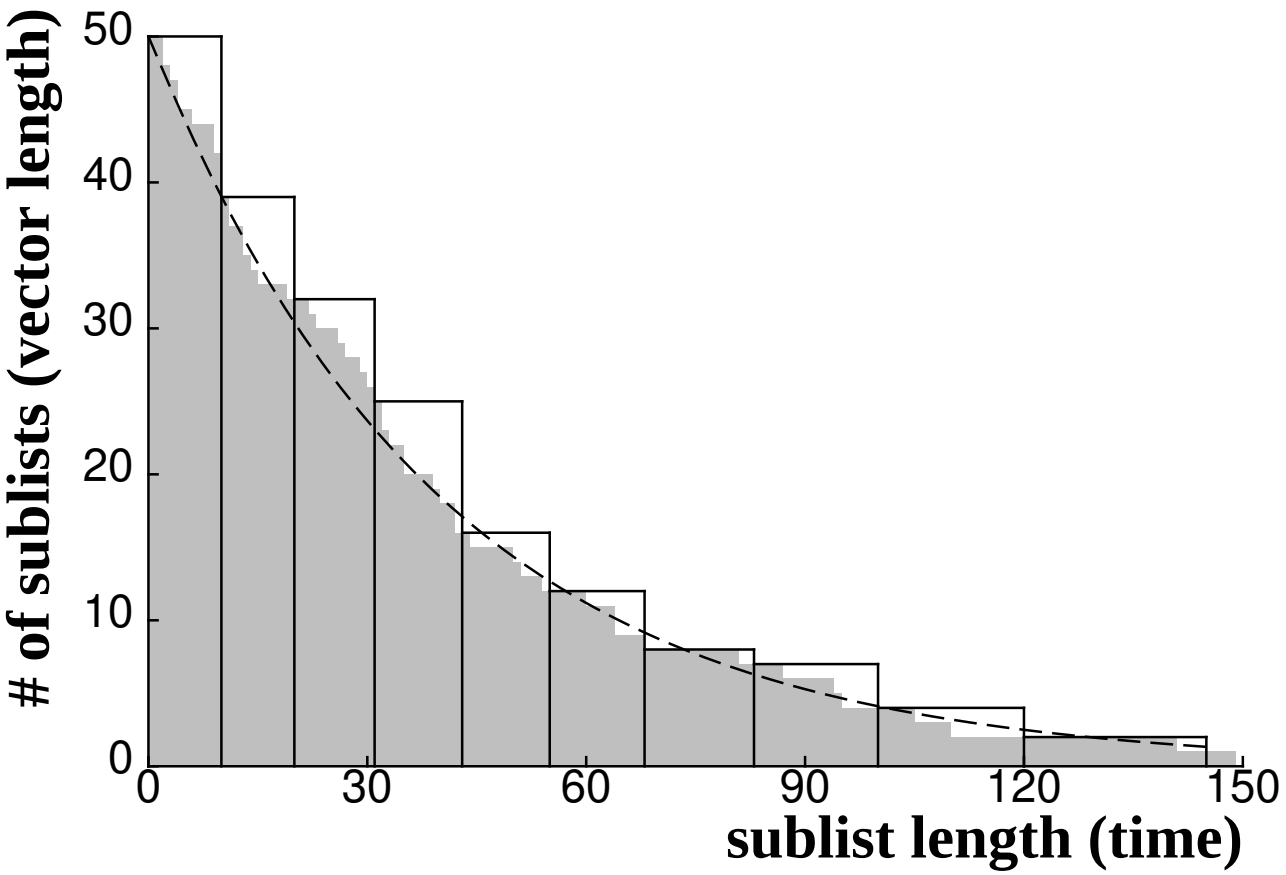
Tuning the parameters

1. How many sublists should we use?
2. How many times should we load balance?
3. How should load balancing frequency change with time?

Depends on:

- expected distribution of the sublist lengths (exponential)
- relative constants of ranking and load balancing

When should we load balance?



The step function:

number of sublists currently active.

The dotted function:

expected sublists length.

When to Load Balance

$$T(n) = \sum_{0}^{l-1}(S_{i+1} - S_i)(ag(S_i)/p + b) + \sum_{0}^{l-1}(cg(S_i)/p + d) + em/p + f$$

where:

m, the number of sublists,
$S_i, i = 1, \ldots, l$ are load balancing iterations,
$g(x) = me^{-mx/n}$.

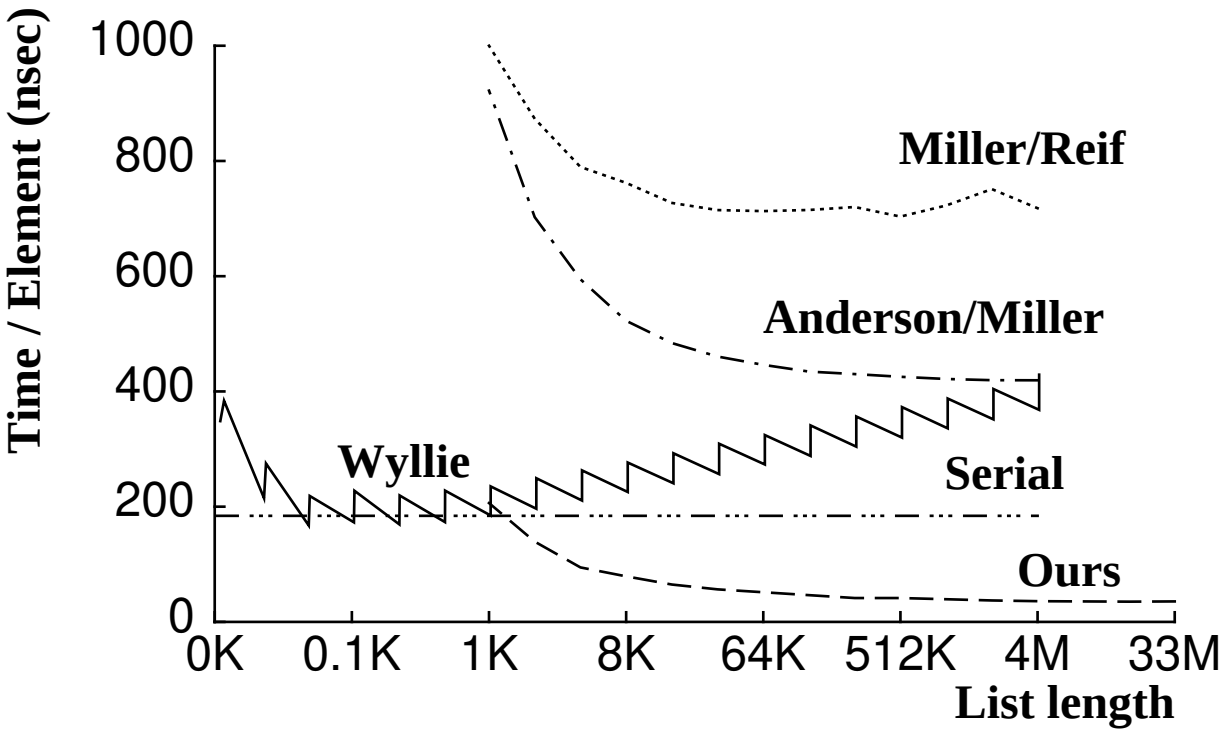
Minimize $T(n)$ with respect to $S_1, S_2, \ldots, S_l$:

$$S_{i+1} = S_i + \frac{g(S_{i-1}) - g(S_i)}{mg(S_i)/n} - \frac{c}{a}$$

If know S_1 can determine remaining S_i.

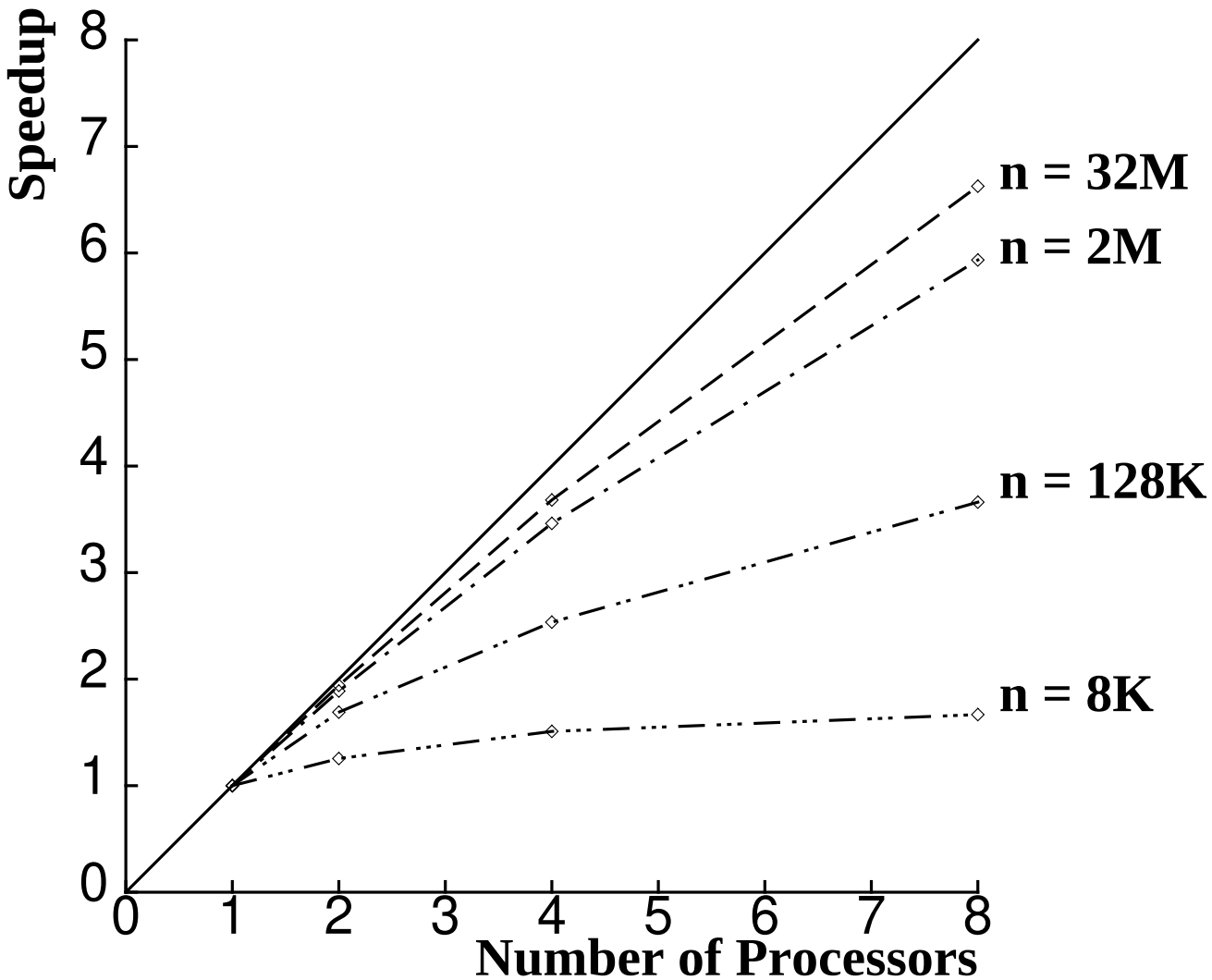
$$T(n) = O(\frac{n}{p} + \frac{n}{m}\log m)$$

List Scan on Cray C-90 (1 processor)



- Fastest implementation of List Scan
- Significantly faster than a fast workstation!

Vector Multiprocessor Performance



Relative speedup.

Conclusions

- Vector Multiprocessors are great PRAMs
- In practice $p << n$
- Work efficiency and small constants are key
- Randomization and oversampling important
- Parallel list ranking is fast on a machine with good memory performance!