Lab: FCDS

Ruochun Tzeng

Environment

OpenMP fork-join framework

Techniques

3-SAT

- wait-free parallelization

Bucketsort

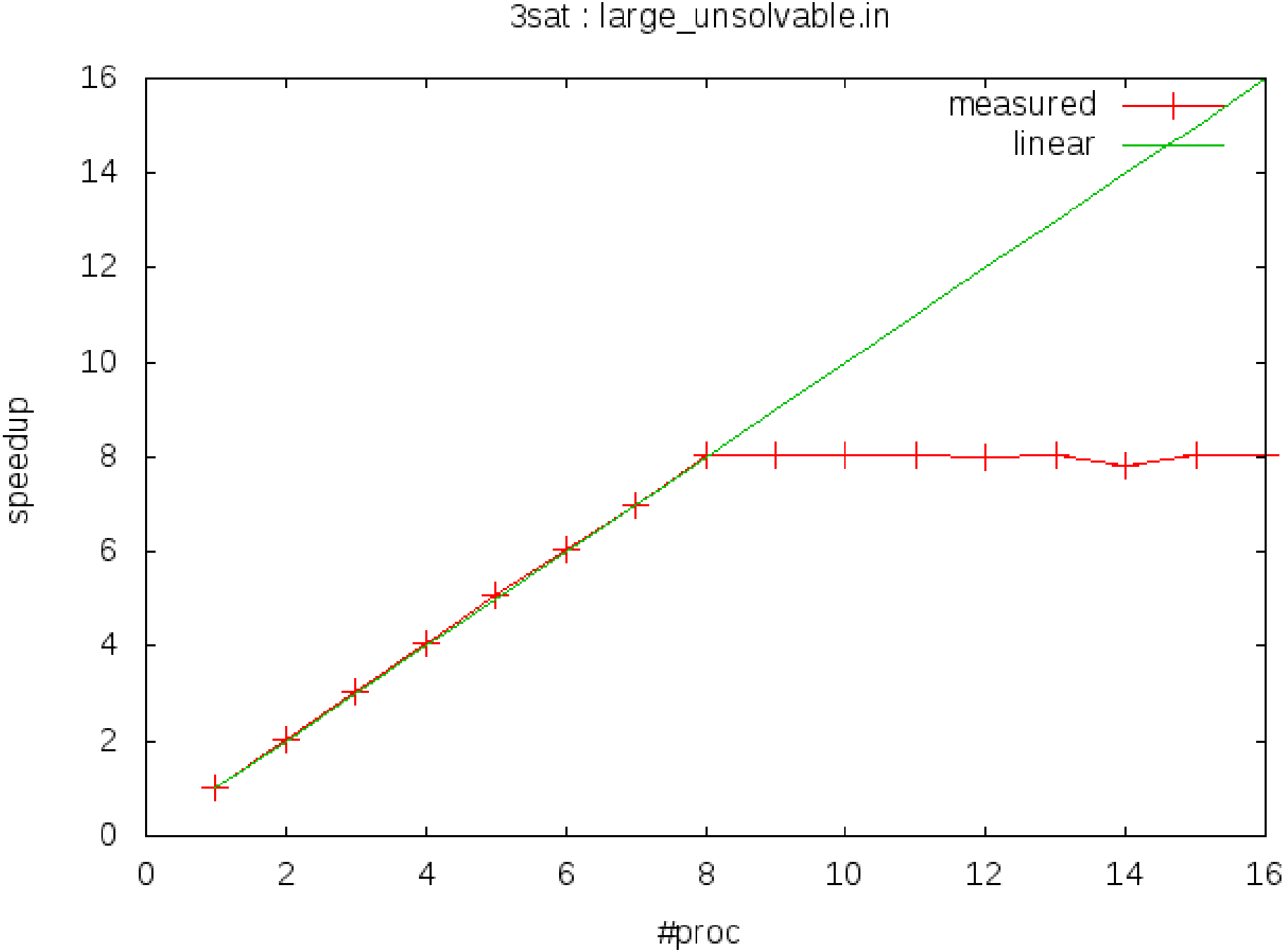
- Wait-free sort
- Fine-grained lock

Unbounded knapsack Problem

- Fine-grained lock

3-SAT

- **Sequential version**
  - Calculate all possible combination
  - $O\left(2^{literal} * clauses\right)$
- **Concurrent version**
  - Parallel the sequential program
  - $O\left(\frac{2^{literal} * clauses}{proc}\right)$
- **Expected speedup**
  - At most linear speedup


3sat : large_unsolvable.in
measured
linear
speedup
#proc
0
2
4
6
8
10
12
14
16

Bucketsort (1/2)

- **Sequential version**
  - Put strings to bucket $O(S)$
  - Sort each bucket $O(94*(2-\frac{1}{S}))$

$O(S)$

- **Concurrent version**
  - Each thread has its own 94 buckets
    - put strings to its local buckets
    - sort buckets, less items to sort
  - Merge to 94 global buckets $O(94*P)$
    - Problem: concatenation on same bucket
    - Approach: each thread start from different buckets

$O(\frac{S}{P}+P)$

$O(\frac{S}{P}+94*(2-\frac{P}{S}))$

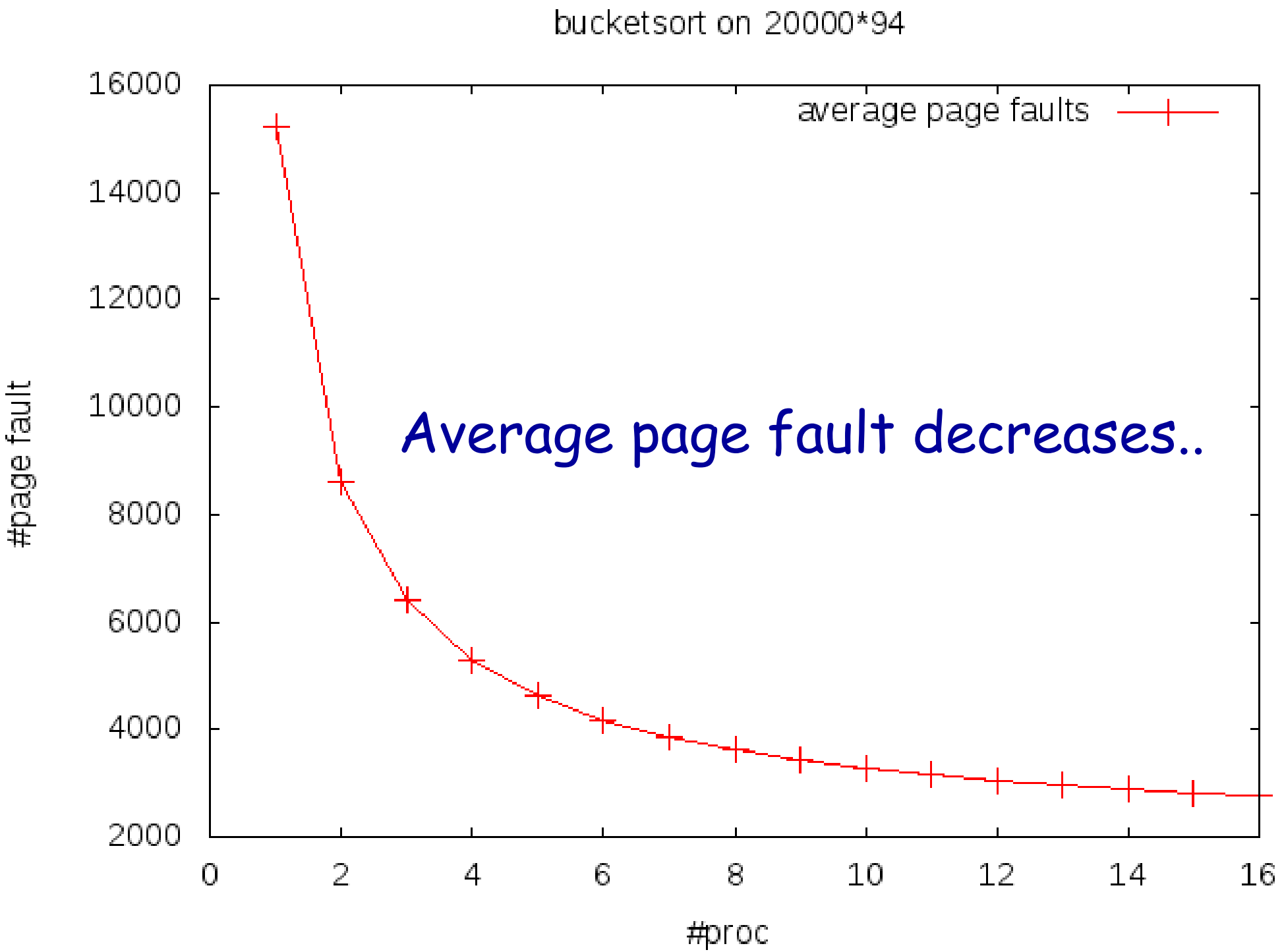
- **Expected linear-speedup**

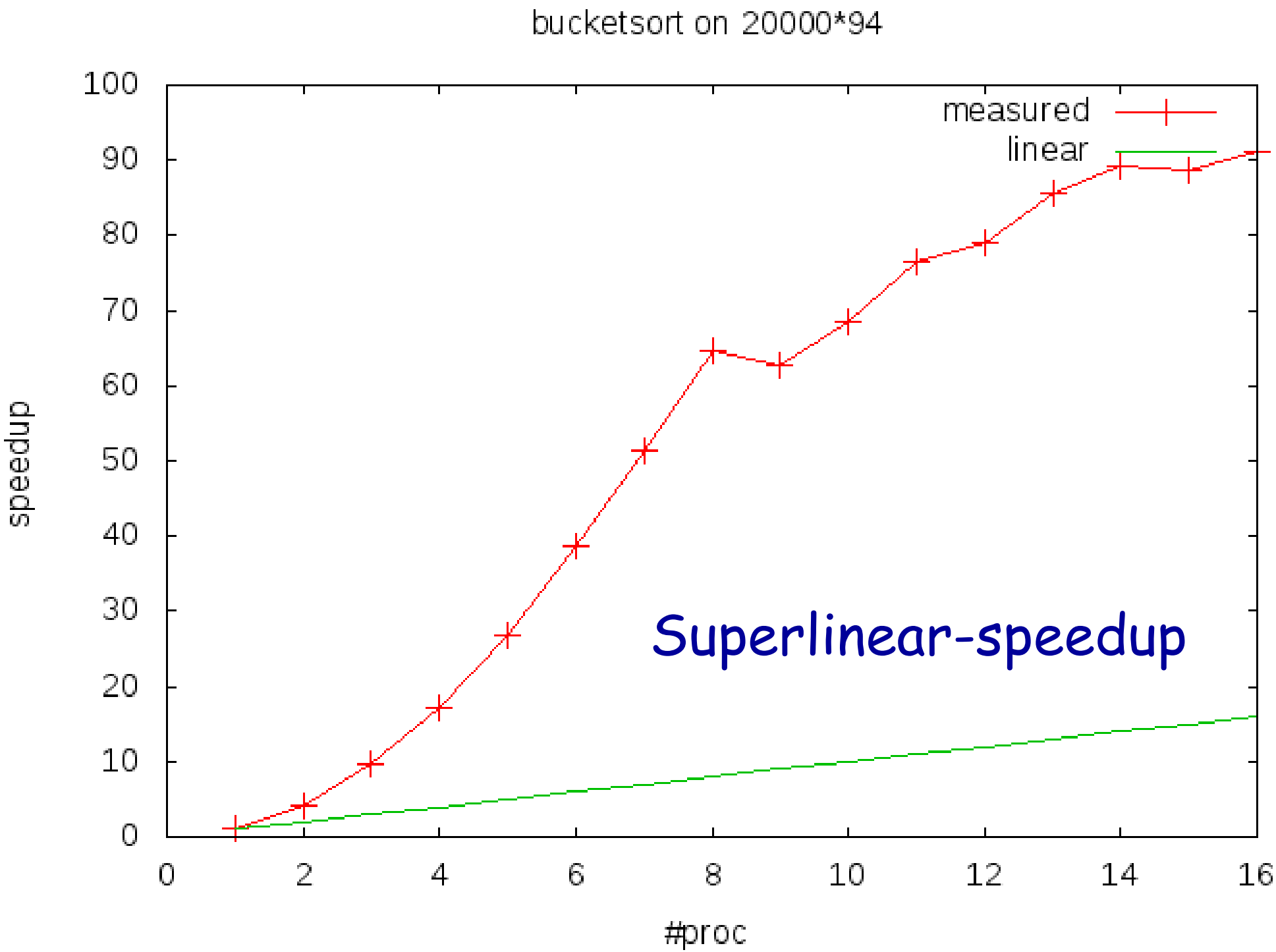
Bucketsort (2/2)

- **Measured** Superlinear-speedup

#proc	input size = 20000*94	50000*94
2x	4x	10x
…	…	...
16x	91x	307x

- Against Amdahl's Law?
  - Heavy I/O task
  - Runtime environment changed!
    - Cache hit rate increases
    - **Average page fault on each processor is reduced**


bucketsort on 20000*94
average page faults
Average page fault decreases..
#page fault
#proc
16000
14000
12000
10000
8000
6000
4000
2000
0
2
4
6
8
10
12
14
16


bucketsort on 20000*94
measured
linear
speedup
#proc
0
10
20
30
40
50
60
70
80
90
100
0
2
4
6
8
10
12
14
16
Superlinear-speedup

Knapsack Problem (1/3)

- **Goal** – Find the largest value with capacity M
- **Dynamic programming**

Identify subproblem

dp[x]: largest value using x capacity

now at x capacity

All case of item i

wasn't taken — largest value using x capacity
=> dp[x]

was taken — largest value using (x-weight[i]) capacity + value[i]
=> dp[x-weight[i]] + value[i]

Knapsack Problem (2/3)

Find a nice order to fill-out dp table

```
for(int i=0;i<n;i++)
    for(int j=weight[i];j<=M;j++)
        dp[j] = max(dp[j], dp[j-weight[i]+value[i])
```

- **Question**
  - How could I parallelize it?
- What's the necessary condition for dp[x] to be correct?

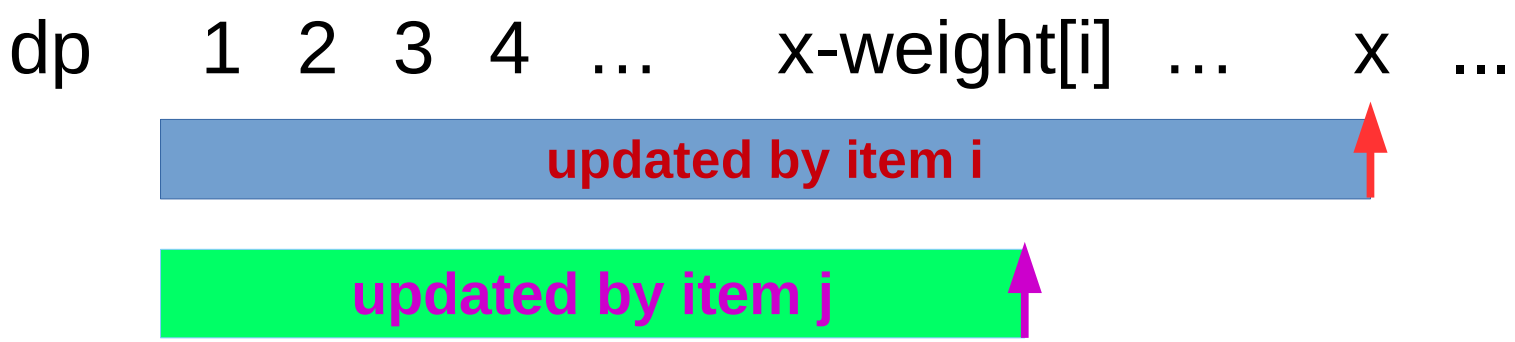
dp[x] = max(dp[x], dp[x-weight[i]]+value[i])

=> as long as

dp[x-weight[i]] is updated with item i

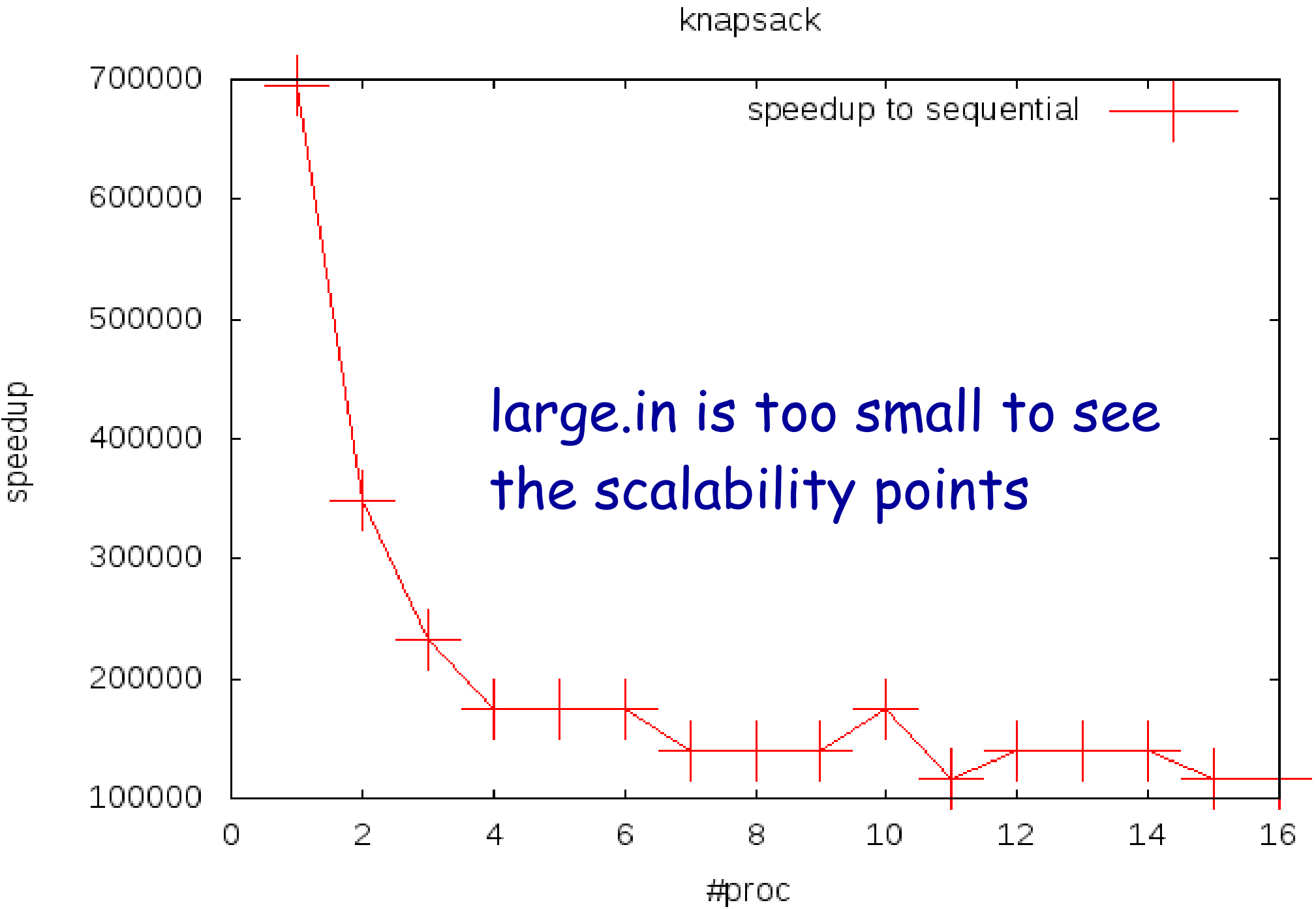
Knapsack Problem (3/3)

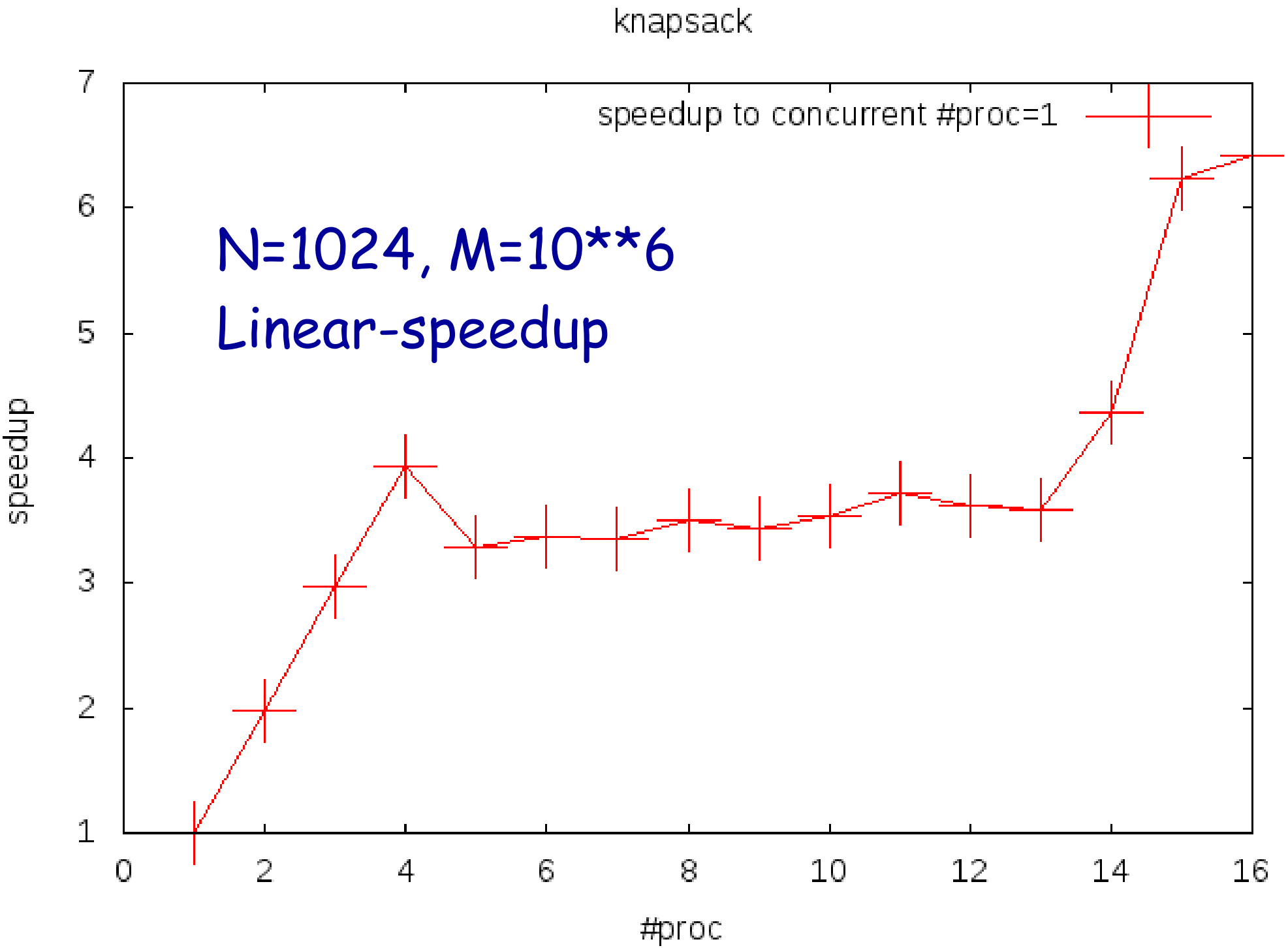
- What if dp[x-weight[i]] is updated by other item j?
  - This would only lead us even quicker to the final answer



Eventually, each dp element must be update by all items

- Parallel between items, sequential on each item
- Fine-grained lock to update dp[x]
- **Expected speedup – at most to linear-speedup**


knapsack
speedup to sequential
700000
600000
500000
400000
300000
200000
100000
speedup
0
2
4
6
8
10
12
14
16
#proc
large.in is too small to see
the scalability points


knapsack
speedup to concurrent #proc=1
N=1024, M=10**6
Linear-speedup
speedup
#proc
1
2
3
4
5
6
7
0
2
4
6
8
10
12
14
16