Nested Parallelism and Hierarchical Locality

Guy Blelloch
Carnegie Mellon University

(Fine Grained) Nested Parallelism =

- Nested parallel loops and fork joins
- Desirably : built in "collective operations"
- NESL, Cilk+, X10, Open MP (perhaps)
  - Support for collective operations differ

Quicksort

```
function quicksort(S) =
if (#S <= 1) then S
else let
  a = S[rand(#S)];
  S1 = {e in S | e < a};
  S2 = {e in S | e = a};
  S3 = {e in S | e > a};
  R = {quicksort(v) : v in [S1, S3]};
in R[0] ++ S2 ++ R[1];
```

Work = $O(n \log n)$
Span = $O(\log^2 n)$

{ … } – means parallelism

Fourier Transform

```
function fft(a,w) =
if #a == 1 then a
else
  let r = {fft(b, even_elts(w)):
           b in [even_elts(a),odd_elts(a)]}
  in {a + b * w : a in r[0] ++ r[0];
                  b in r[1] ++ r[1];
                  w in w};
```

Sparse Matrix Vector Multiply

```
function spmv(A, x) =
   {sum({v * x[i] :(i,v) in row} : row in A}

e.g. A = [[(3, 7.9), (11, 2.2), (14, -2.0)],
          [(4, -1.0), (6, 1.5)],
          [(0, .1), (14,.9), (22, -2,3), … ]
          …]
```

Matrix Multiplication

```
Fun A*B {
  if #A < k then baseCase..
  C11 = A11*B11 + A12*B21
  C12 = A11*B12 + A12*B22
  C21 = A21*B11 + A22*B21
  C22 = A21*B12 + A22*B22
  return C
}
```

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

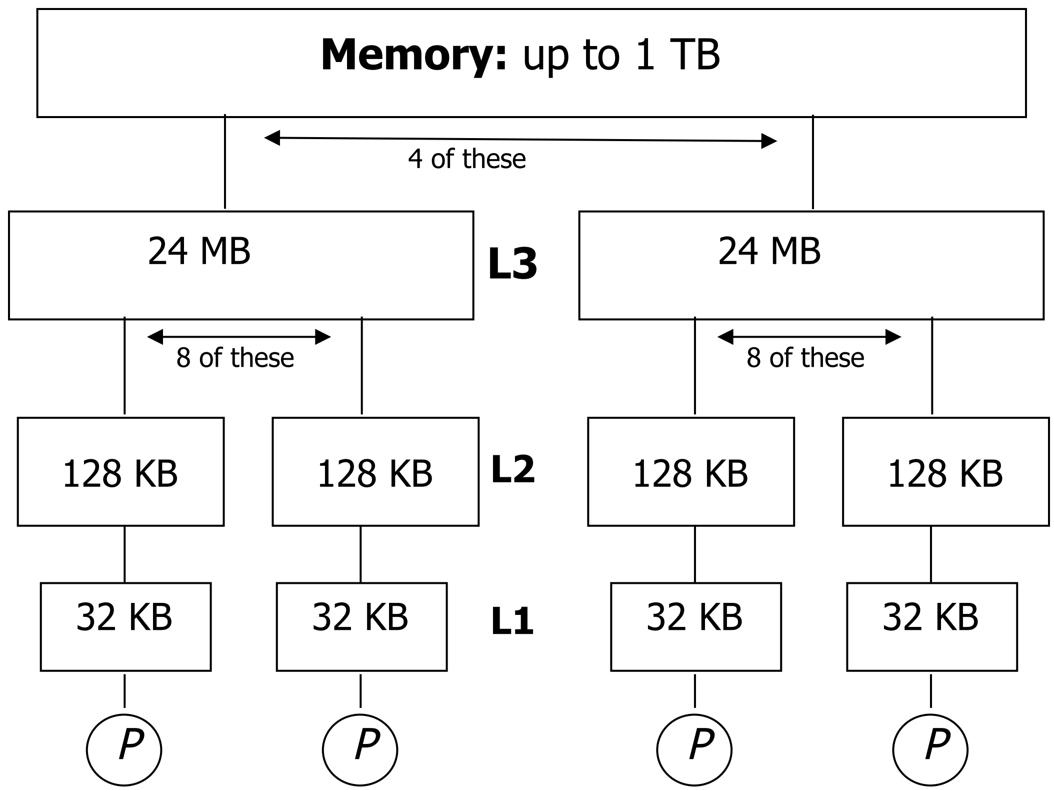
$D = O(\log^2 n)$
$W = O(n^3)$

Advantages of Nested Parallelism

- Lots of parallelism
- Flexibility in scheduling…good for both vector/ SIMD and asynchronous computing
- Easy to reason about
- Broadly applicable
- Reasonably easy to make deterministic
- Simple formal cost model (Work and Span)
- **Good for (hierarchical) locality**

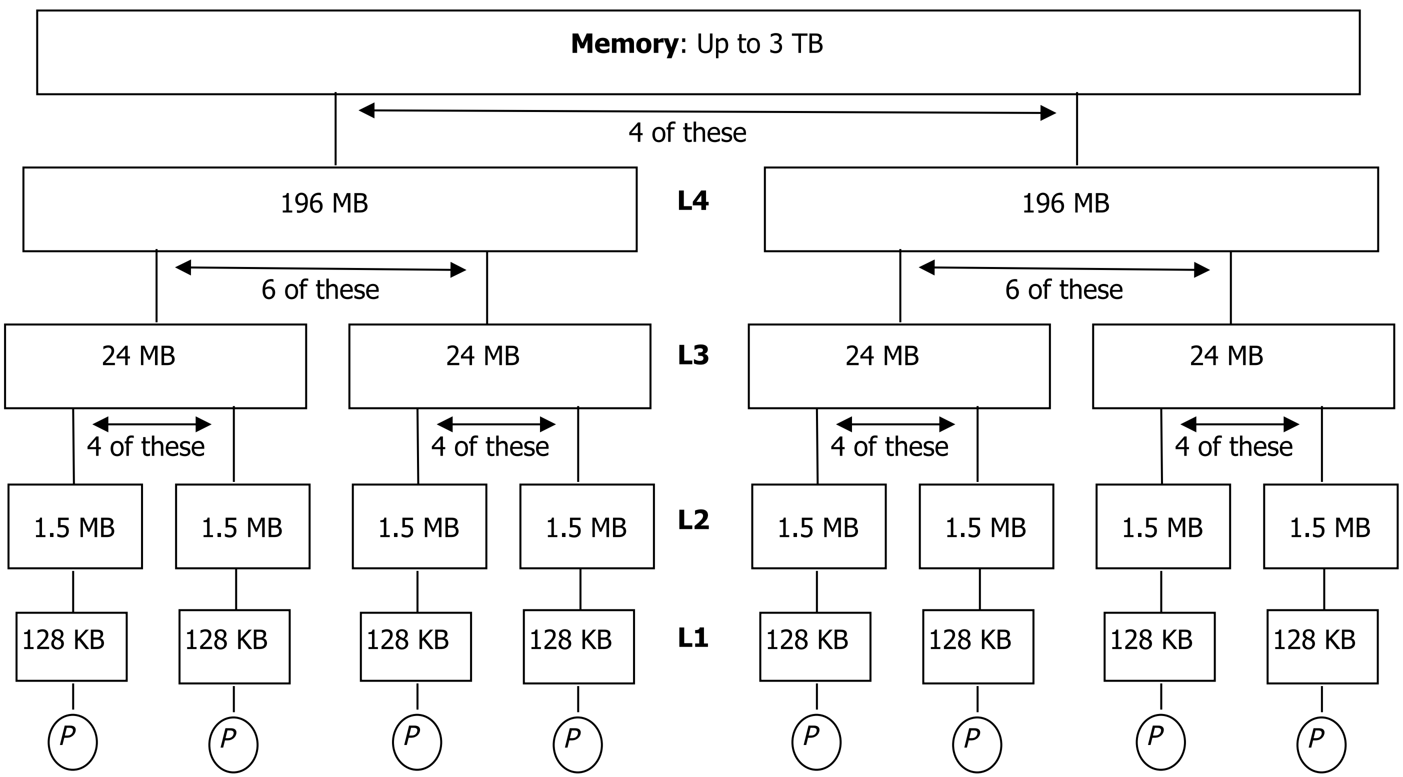
Current machines already have deep hierarchies

- **Xeon**: 3 levels of cache + Memory, 32 cores



…and deeper

- **IBM z196**: 4 levels of cache + Memory



Problem

- Trying to write portable code to take advantage of all levels of cache is near impossible. Possibly more true on exascale machines.
- Assuming two levels is unlikely to work.

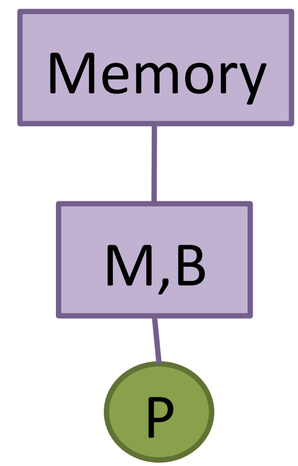
Goal

- Give the user a **high-level** dynamically parallel **programming model**.
- Give them a way to **reason about the locality**/ communication costs in their program that is independent of details of the machine.
- Supply **schedulers** that take advantage of locality on a wide variety of machines (including exascale?).

Ideal Cache Model

Sequentially assume a machine with two cache parameters

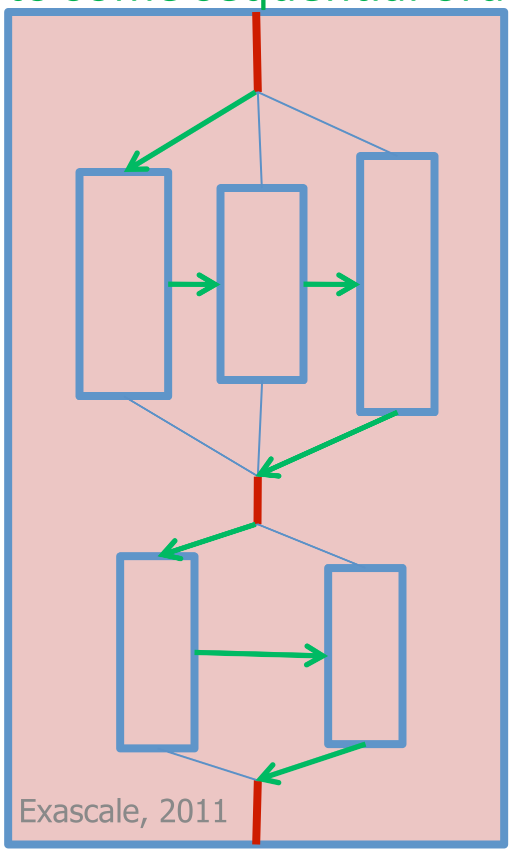
- Cache size
- Block size

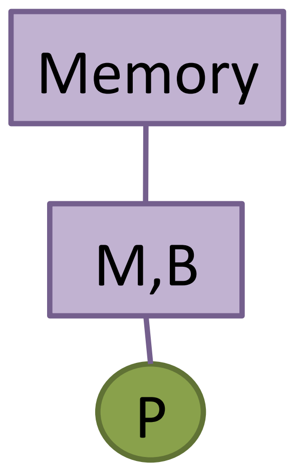


If program does not use parameters then it will be reasonably efficient across all levels of the cache (the **Cache Oblivious Model**)

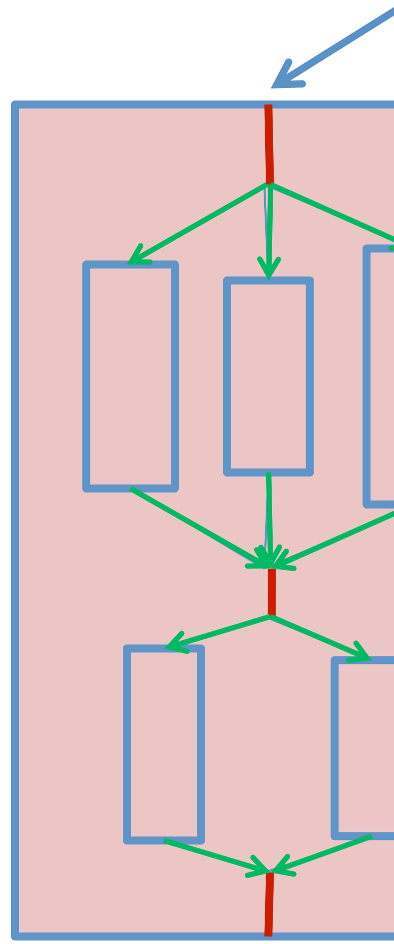
Parallel Cache Oblivious Model (PCO)

Carry forward cache state according to some sequential order





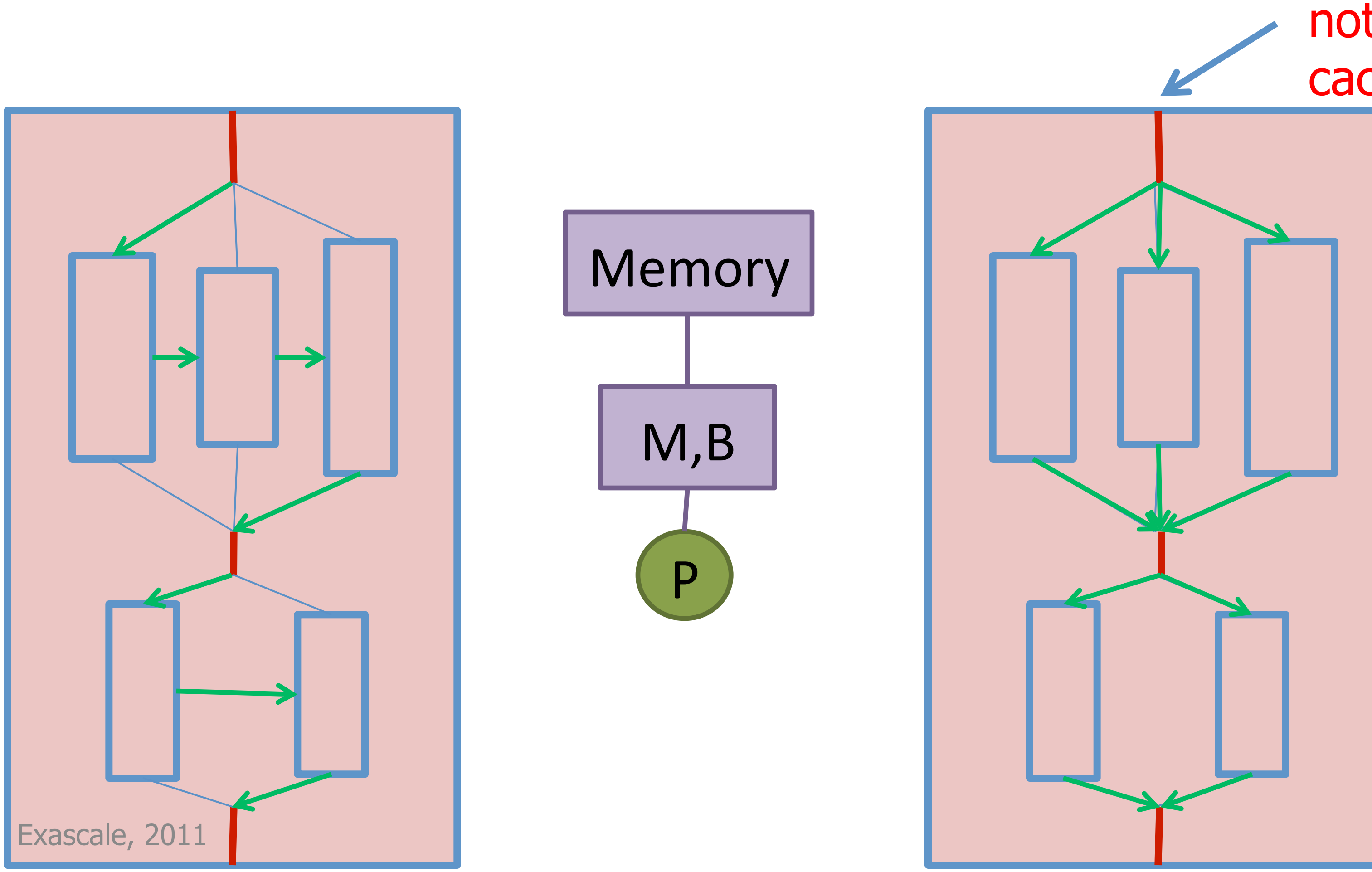
Assuming this task fits in cache



All three subtasks start with same state

Merge state and carry forward

Parallel Cache Oblivious Model (PCO)



Task does not fit in cache

All three tasks start with empty state

Strand starts with empty state

Summary of Bounds

$$Q(n) =$$

Scan Memory, prefix sums, merge, median, matrix transpose: $O\left(\frac{n}{B}\right)$

Matrix Multiply

Matrix Inversion: $O\left(\frac{n^{1.5}}{BM^{.5}}\right)$

FFT: $O\left(\frac{n}{B}\log_Z n\right)$

Mergesort, Quicksort, NNs, KD-trees: $O\left(\frac{n}{B}\log_2(n/M)\right)$

Sample Sort: $O\left(\frac{n}{B}\log_M n\right)$

Better Sort

```
Function sort(A) =
n = |A|
if n <= 1 return a
else
  Pivots = sort sample of size sqrt n
  For each B in partition(A,sqrt(n))
    C = split(sort(B),Pivots)
  D = transpose(C)
  For each B in D
    R = sort(flatten(B))
  Return flatten(R)
```

$Q = O(n/B \log_M n)$
Instead of
$Q = O(n/B \log (n/M)0$

Why?

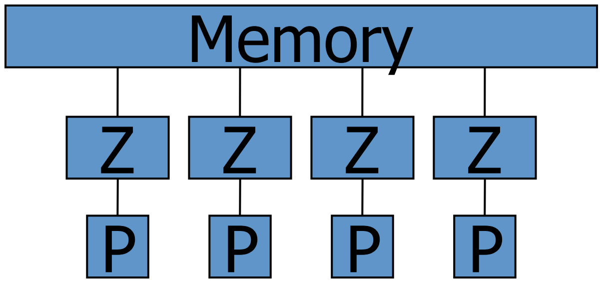
How is the cost model useful

General Bounds

On a private cache [ABB00]

$Q_P(C) = Q(C) + O(PDM/B)$

Using work stealing

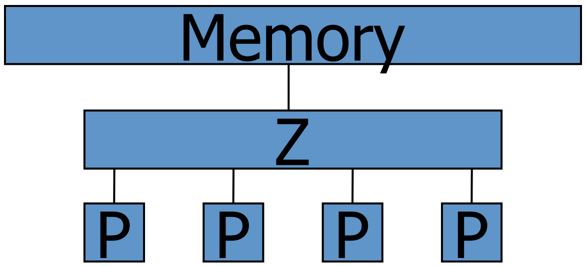


On shared caches [BG04]

$Q_P(C) = Q(C)$

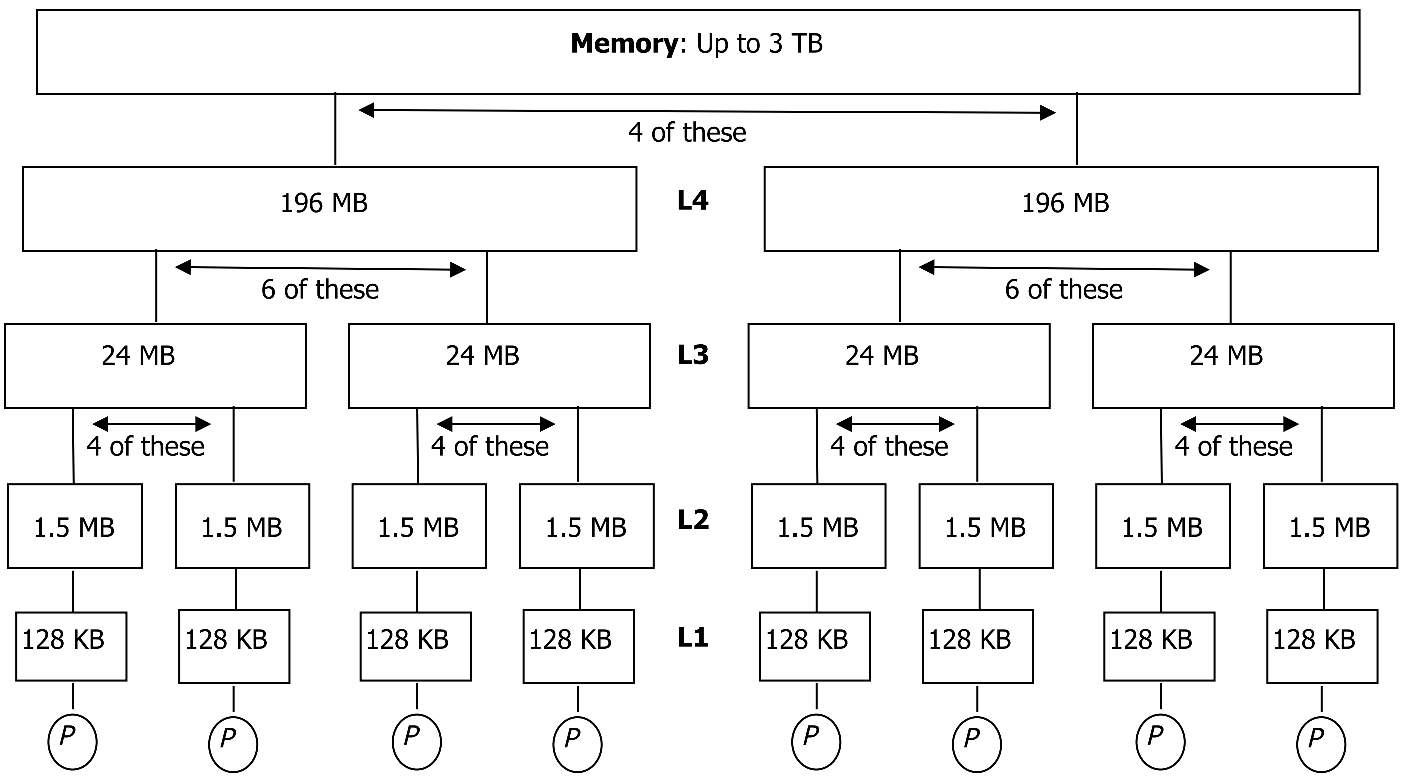
for $M_P = M_1 + O(PD)$

Using parallel depth first



…but what about

- **IBM z196**: 4 levels of cache + Memory



General Bounds (informal)

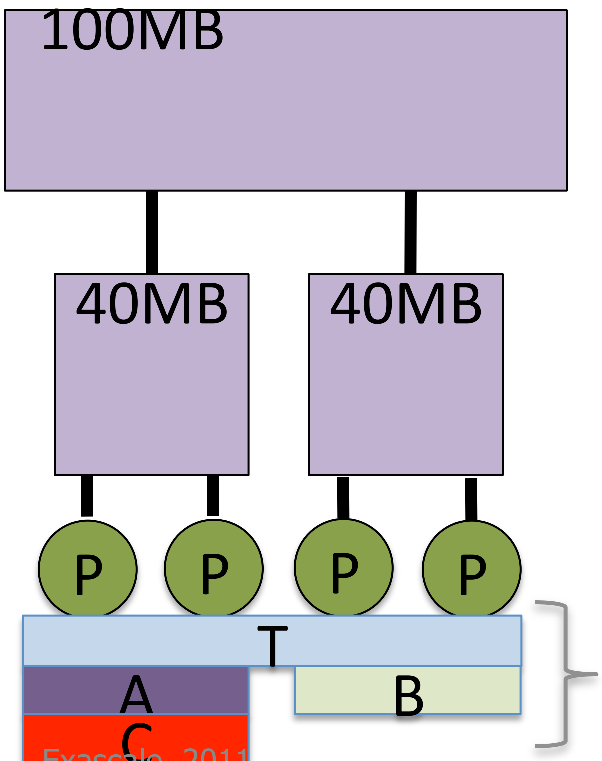
- Under some assumptions, can show with an appropriate scheduler something like the following can be shown

$$\text{Time} = \frac{\sum_{i=0}^{h-1} Q_\alpha^*(t; M_i/3, B_i)C_i}{\#procs} \times overhead$$

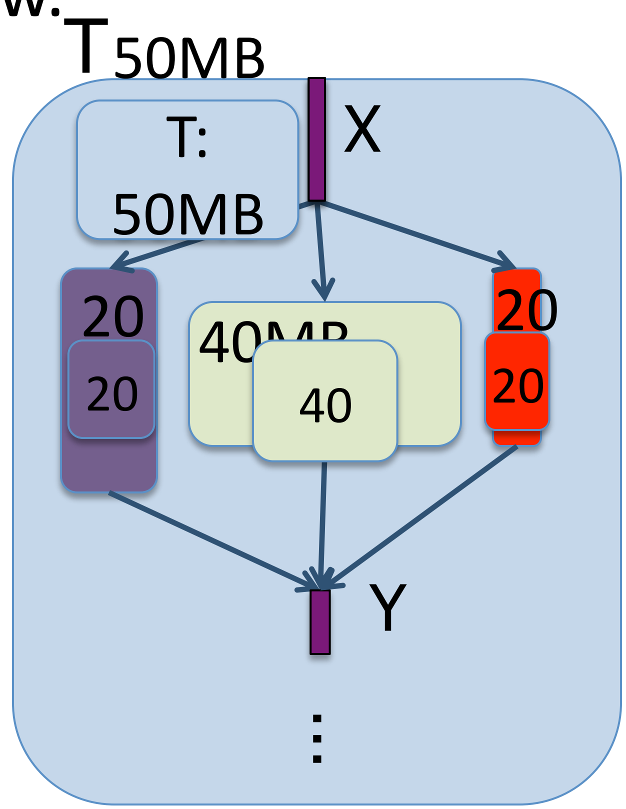
Space-Bounded (SB) Scheduler

Assign tasks to caches that fit them.

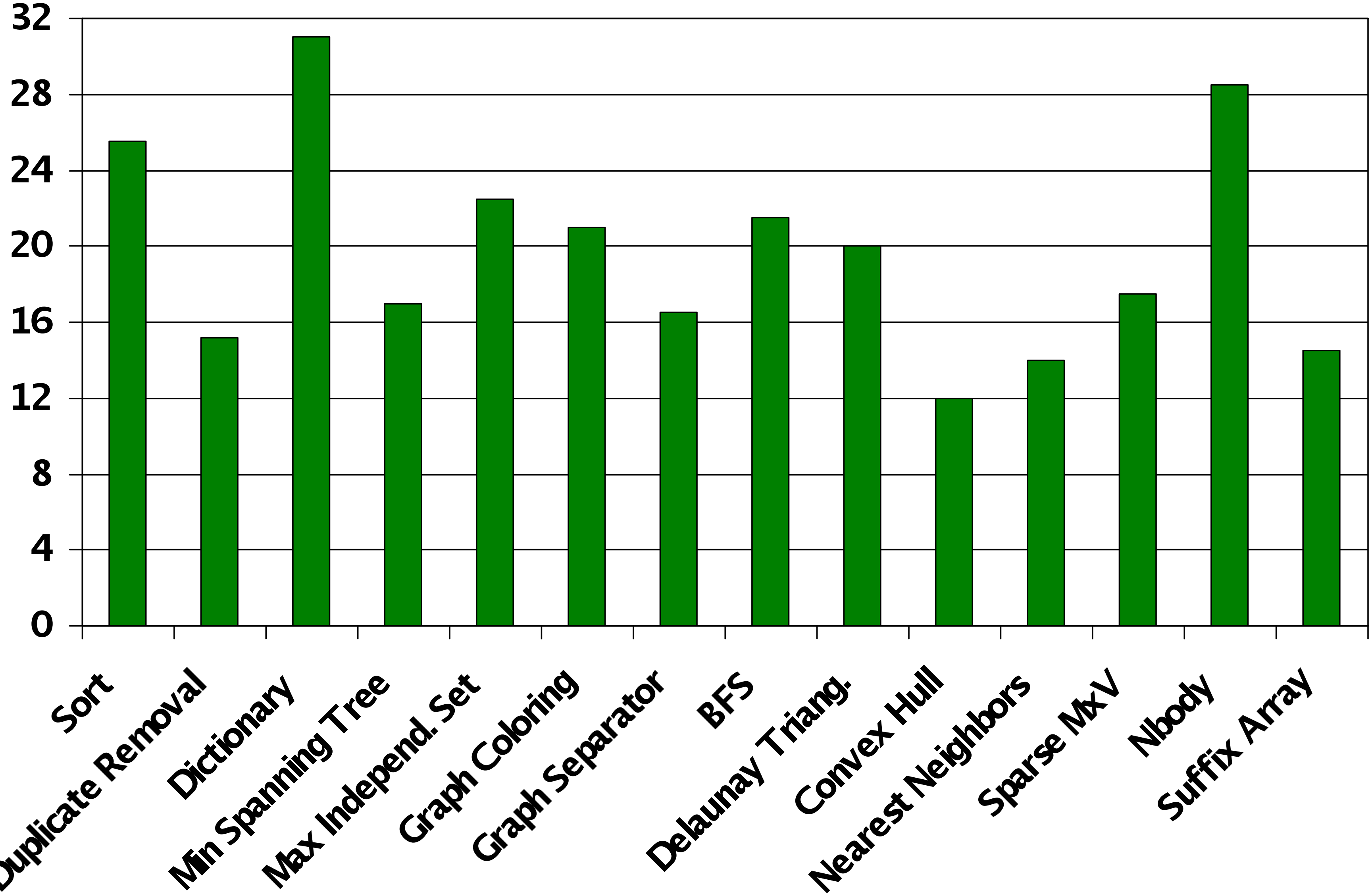
- Do not allow tasks to move
- Do not allow caches to overflow.



Permitted processors per



Preliminary Numbers



Conclusion

Reasoning about locality in exascale machines is likely to be very difficult.

In addition to other important properties for exascale computing:

– Lots of fine grained parallelism

– Various choices in scheduling

– …

Nested parallelism can be good for taking advantage of **hierarchical locality**