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

Exascale, 2011
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];

{ ... } – means parallelism

Work = O(n log n)
Span = O(log² n)
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, 1.0), (14, 0.9), (22, -2.3), ... ]
         ...]
Matrix Multiplication

Fun A*B {
    if #A < k then baseCase..
    C_{11} = A_{11} * B_{11} + A_{12} * B_{21}
    C_{12} = A_{11} * B_{12} + A_{12} * B_{22}
    C_{21} = A_{21} * B_{11} + A_{22} * B_{21}
    C_{22} = A_{21} * B_{12} + A_{22} * B_{22}
    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

```
Memory: up to 1 TB
```
```
L3  24 MB
     8 of these
    /    \\
L2 128 KB
     8 of these
    /    \\
L1 32 KB
     P
```
```
L3  24 MB
     8 of these
    /    \\
L2 128 KB
     8 of these
    /    \\
L1 32 KB
     P
```
...and deeper

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

![Diagram showing cache levels and memory capacity](image)
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)

Exascale, 2011
Summary of Bounds

\[ Q(n) = \]

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

matrix transpose:

Matrix Multiply \( O\left(\frac{n^{1.5}}{BM^5}\right) \)

Matrix Inversion:

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))
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

Exascale, 2011
...but what about

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

Memory: Up to 3 TB

- L1: 128 KB, 4 of these, 128 KB, 4 of these, 128 KB, 4 of these, 128 KB, 4 of these.
- L2: 1.5 MB, 4 of these, 1.5 MB, 4 of these, 1.5 MB, 4 of these, 1.5 MB, 4 of these.
- L3: 24 MB, 6 of these, 24 MB, 6 of these, 24 MB, 6 of these, 24 MB, 6 of these.
- L4: 196 MB, 4 of these, 196 MB, 4 of these, 196 MB, 4 of these, 196 MB, 4 of these.
General Bounds (informal)

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

\[
\text{Time} = \sum_{i=0}^{h-1} \frac{Q^*(t; M_i/3, B_i)C_i}{\#\text{procs}} \times \text{overhead}
\]
Space-Bounded (SB) Scheduler

Assign tasks to caches that fit them.

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

Exascale, 2011
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**