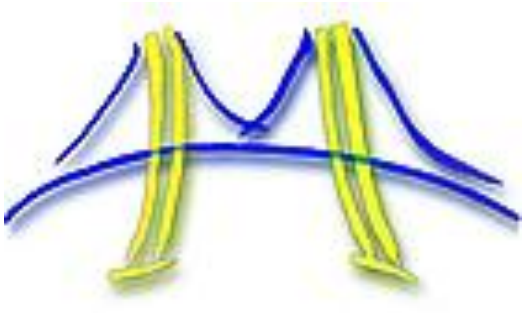
Update on Angelic Programming

synthesizing GPU friendly parallel scans

Shaon Barman, Sagar Jain, Nicholas Tung, Ras Bodik

University of California, Berkeley



Angelic programming and synthesis

Angelic programming:

- An oracle invents an example execution
- Programmer generalizes example to an algorithm

Results:

- Entanglement: a tool to help with algorithm discovery

Synthesis with sketching:

- Sketch is a template synthesized into an implementation
- The resulting program meets a functional specification

Results:

- Sketching generates algorithm variants (for autotuning)

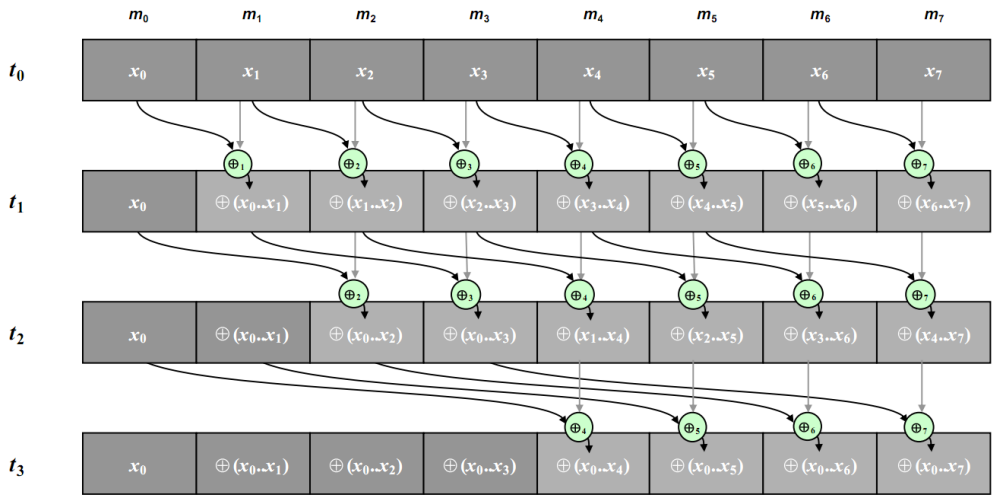
Why scans?

Many practical algorithms use scans [Blelloch '90]

- lexically compare string of characters; lexical analysis
- evaluate polynomials
- radix sort, quicksort
- solving tridiagonal linear systems
- delete marked elements from an array
- search for regular expressions
- tree operations
- label components in two dimensional images
- dynamic programming (see Evan Pu's poster)

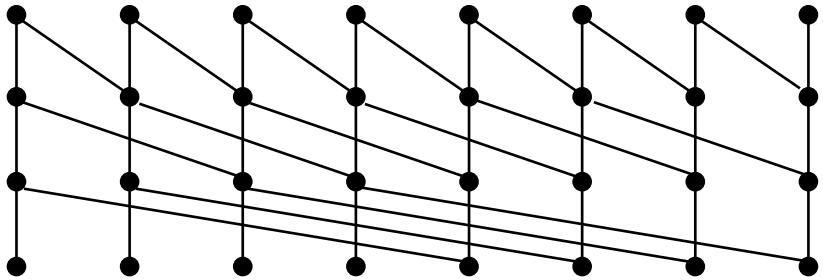
Many problems are sums with some assoc operator

Implementing scans



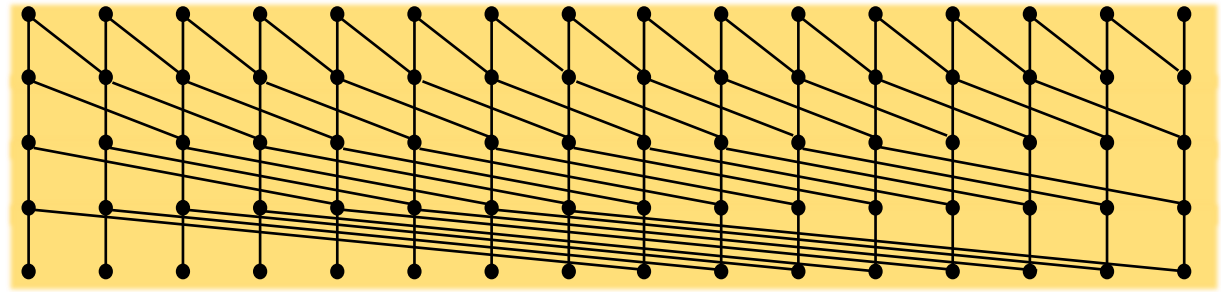
N = 8

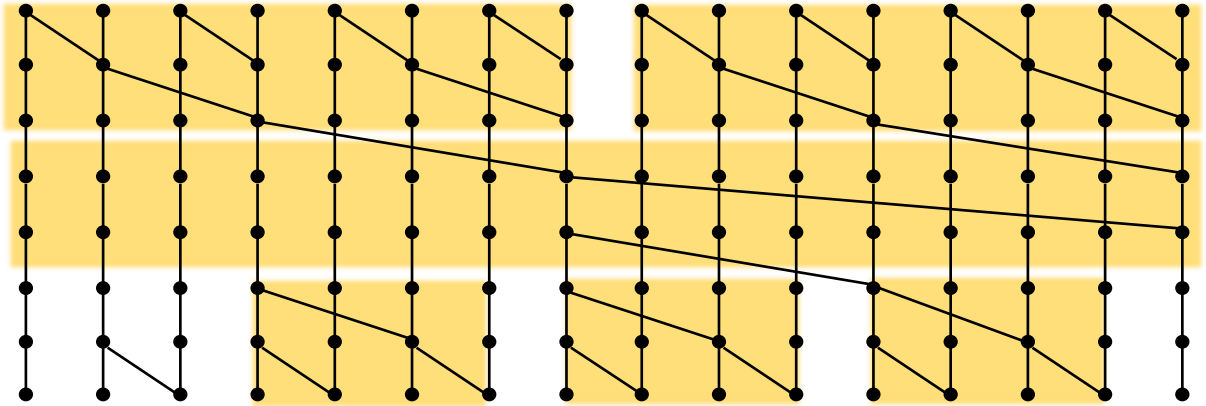
instance of parallel scan algorithm



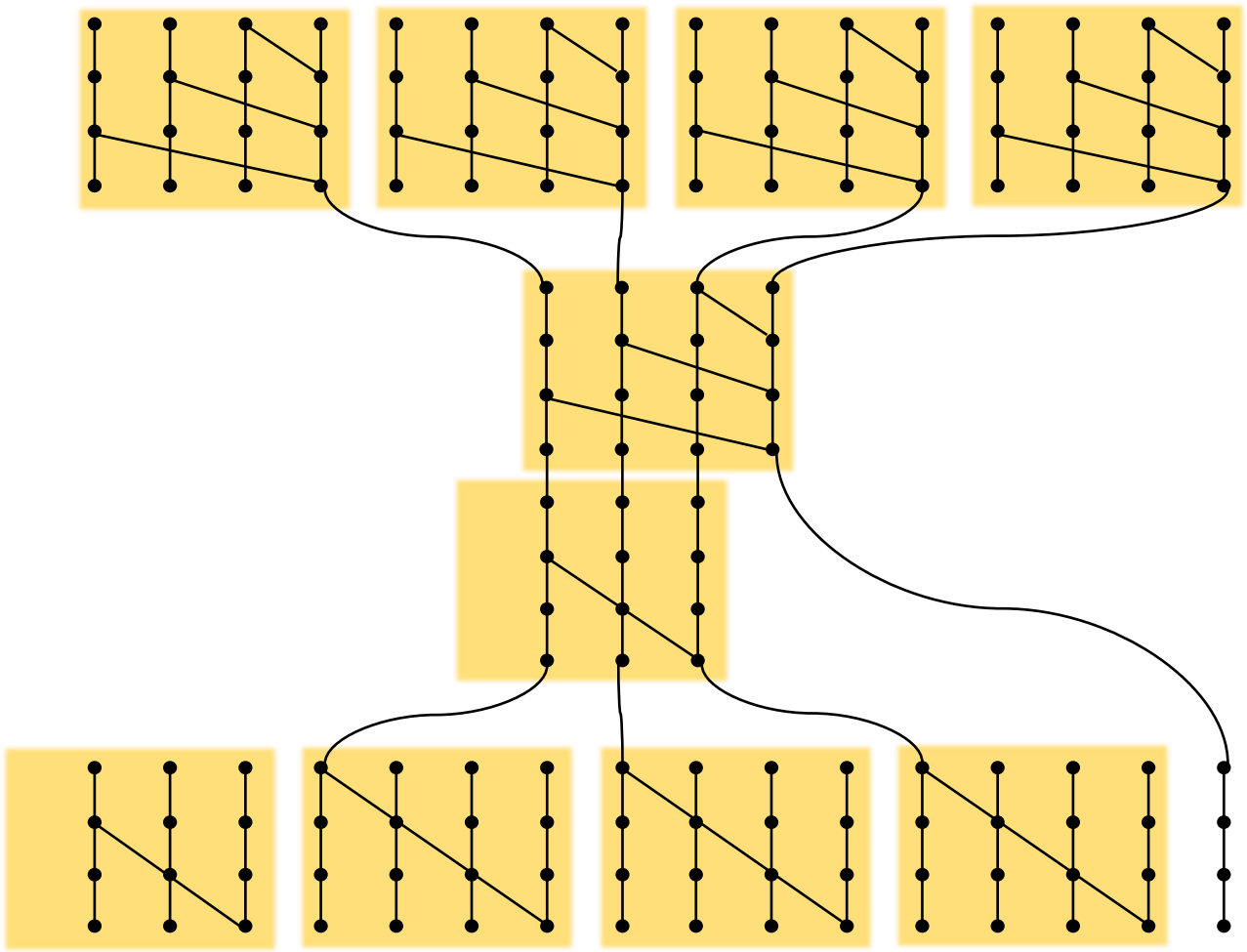
its abstract visualization

SIMD execution of scan algorithms

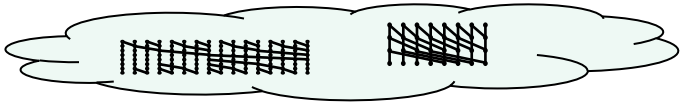




Hierarchical execution of scans algorithms

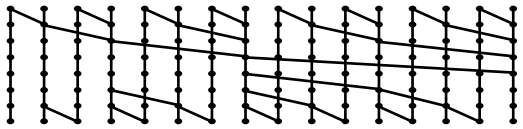


The workflow (talk outline)



space of all algorithms

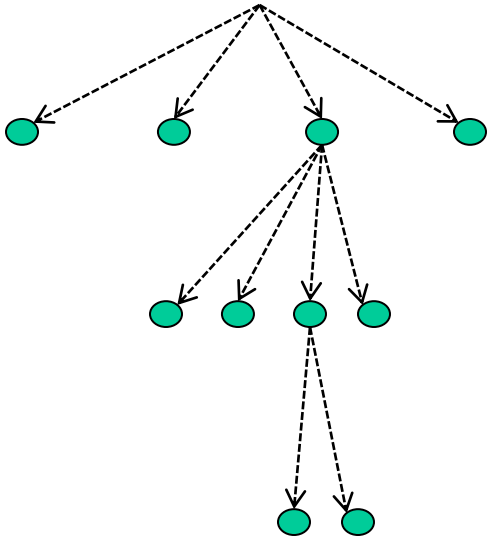
all networks that meet some constraints



instance of the algorithm

a specific network for a fixed input size

```
for d in 1 to log N
    forall i in 1 to N
        A[i] = A[i]+A[i-step]
```



program of the algorithm

works for arbitrary input size

functional variants of the algorithm

forward/backward x inclusive/exclusive

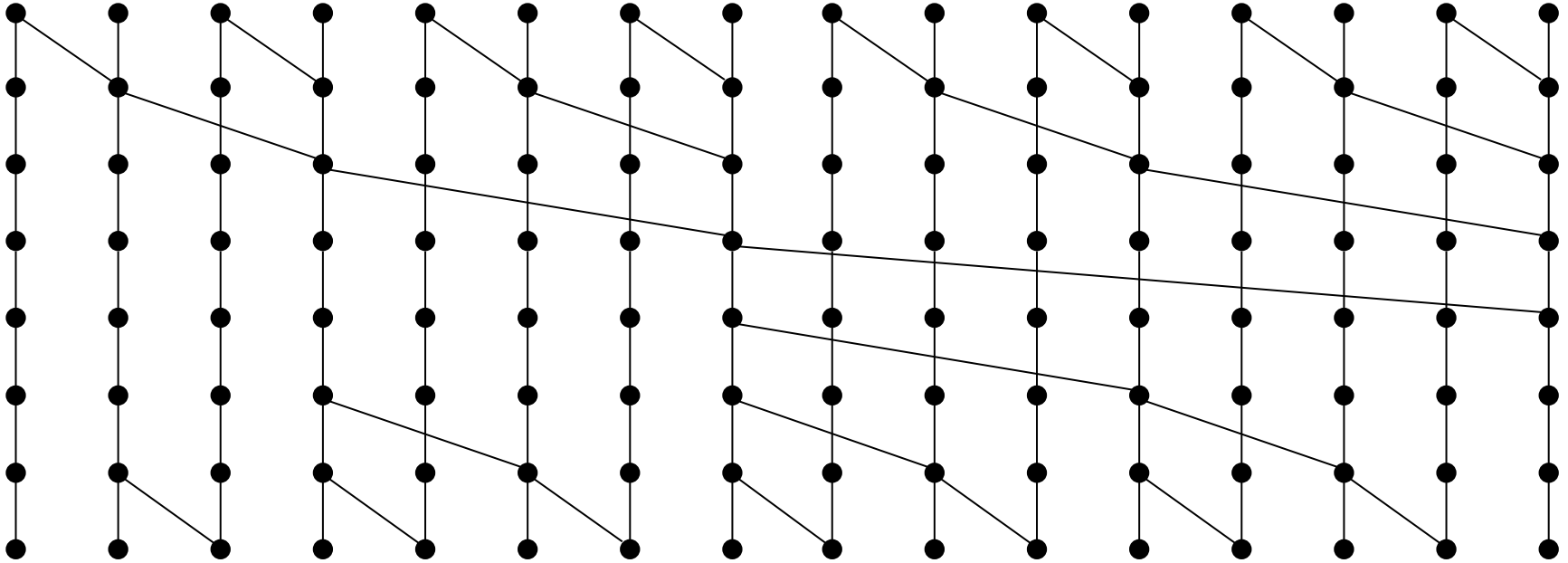
segmented scan

for various segment representations

optimized scan

bank conflict avoidance

Finding an example network



Brent-Kung algorithm

A textbook (clean, regular) instance of the algorithm is all we need to define the algorithm.

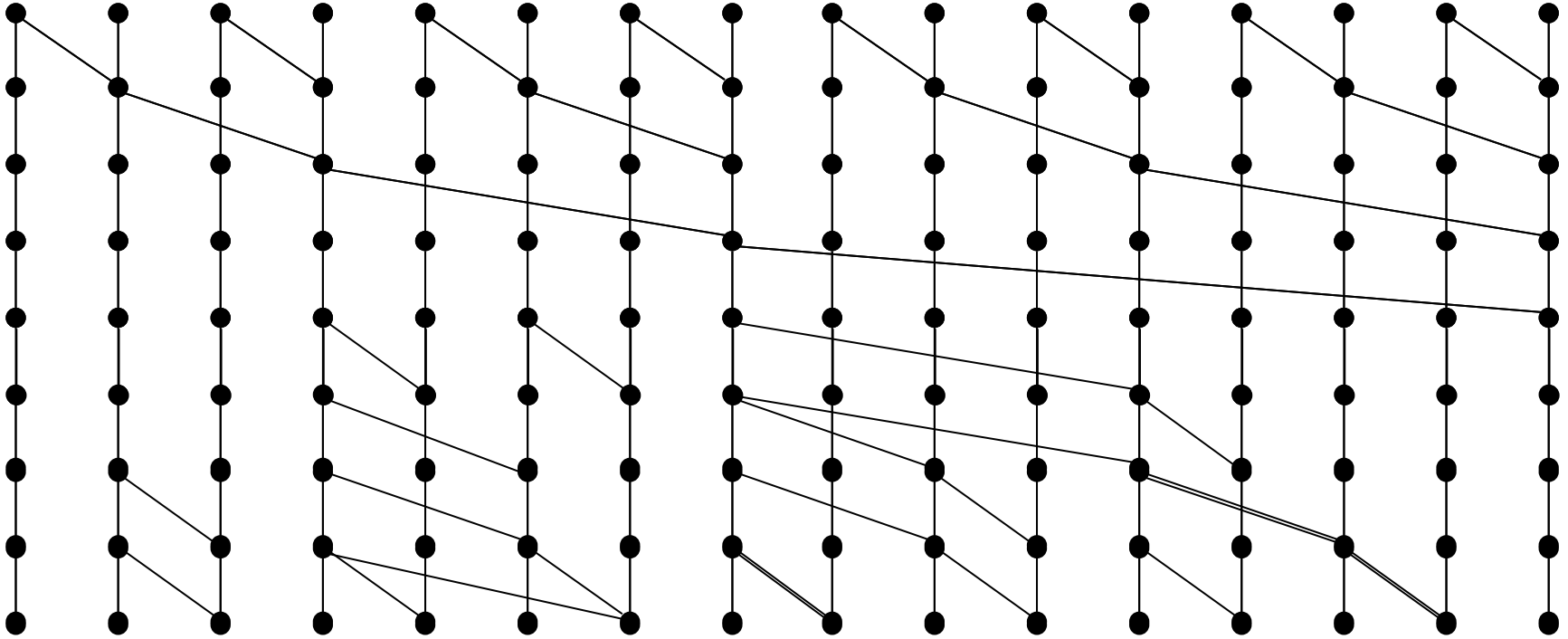
How do we find such networks?

Ask angels how to do parallel scan in O(log n) time

```
N = 16
ops = 0
for step = 1 .. 2*log(N)
    for r from 0 to N-1 in parallel
        if (!!)
            x[r] = x[r] + x[r-!!]
            ops += 1

assert x is a prefix sum
assert ops <= 2N
```

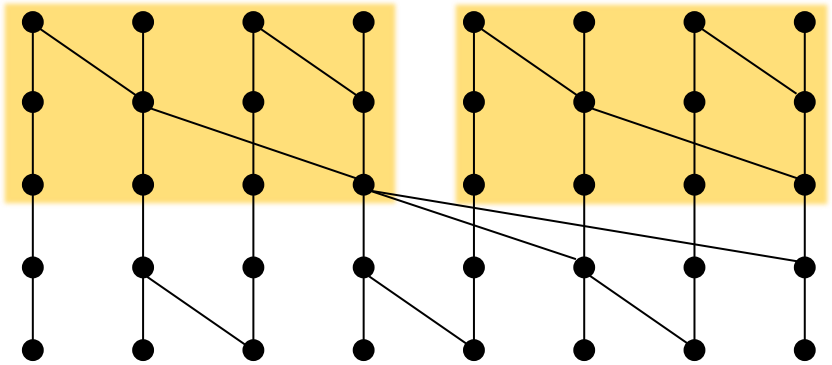
Problem with angelic programming



Synthesizer creates irregular networks

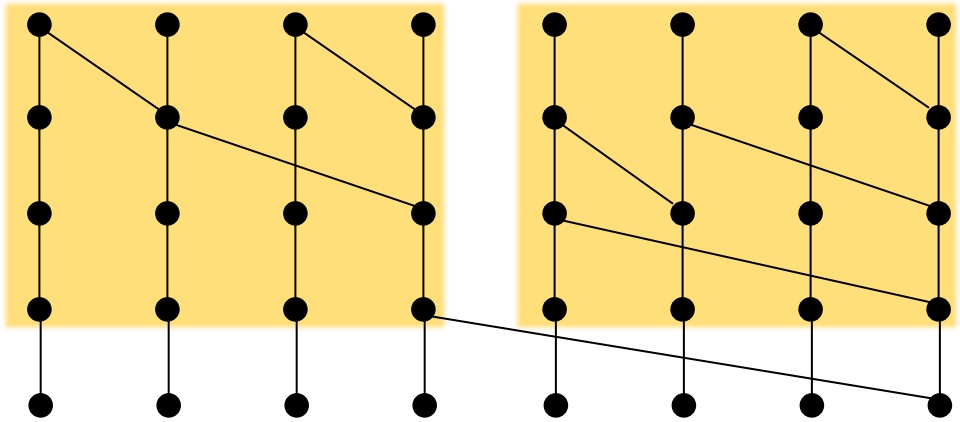
Attempt 1: find a regular pattern

- Ask angels to generate all networks
- Find one that is regular
- Manually examine each network for regularity
- Too many networks (possibly thousands)

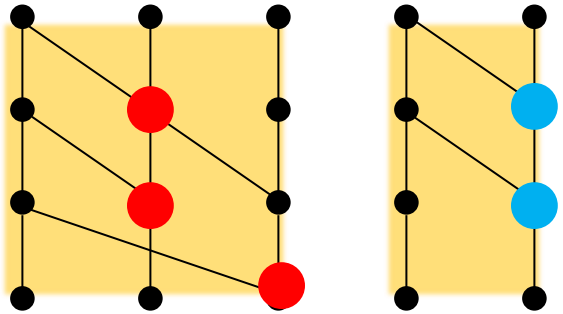


Attempt 2: Functional decomposition

- Look at all networks
- Find in them the function they compute
- These are the subcomputations
- Example network:
  - The right functional view make them the same

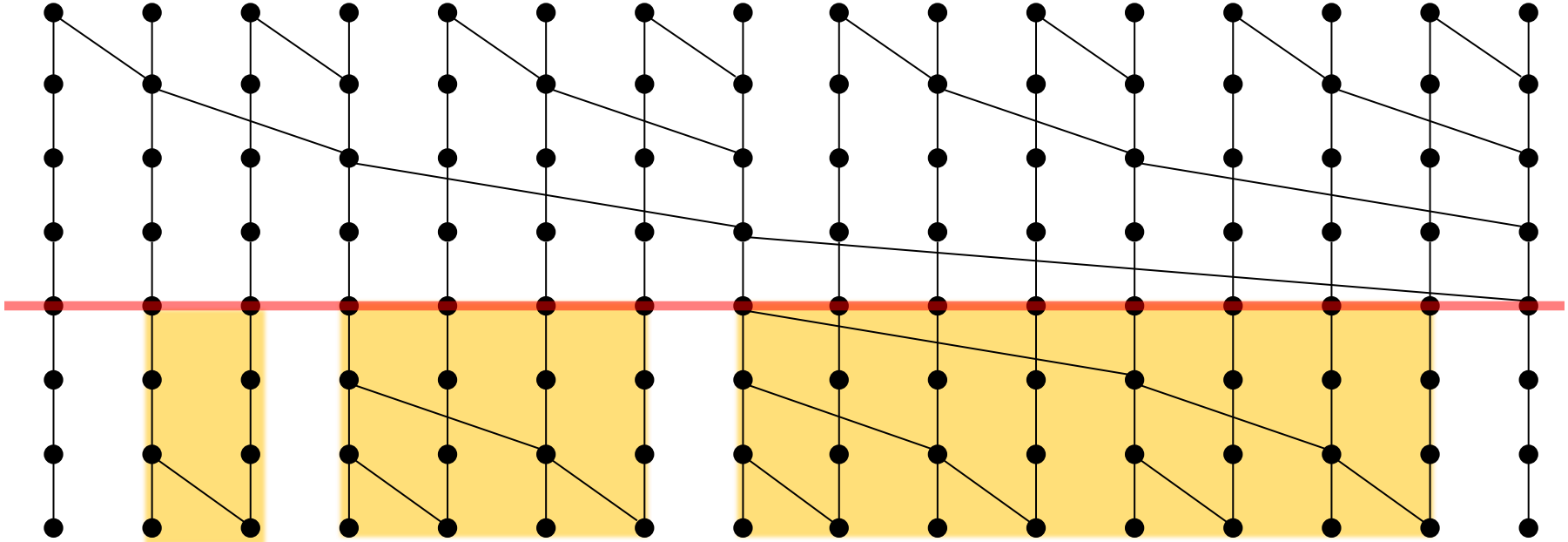


Entanglement



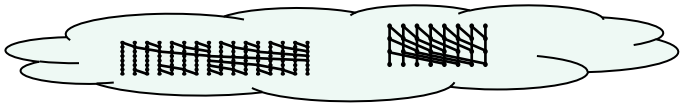
- Entanglement occurs when angels coordinate
- We use this coordination to partition the angels
  - No coordination across partitions
- Unengtangled angels = subcomputations

Building a regular pattern



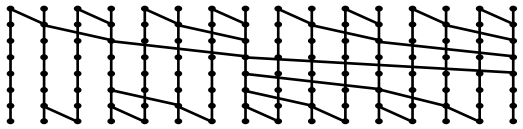
Examining each partition leads to the BK pattern

The workflow (talk outline)



space of all algorithms

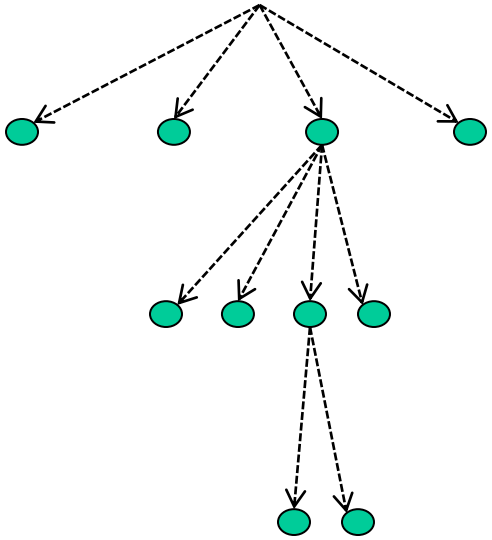
all networks that meet some constraints



instance of the algorithm

a specific network for a fixed input size

```
for d in 1 to log N
    forall i in 1 to N
        A[i] = A[i]+A[i-step]
```



program of the algorithm

works for arbitrary input size

functional variants of the algorithm

forward/backward x inclusive/exclusive

segmented scan

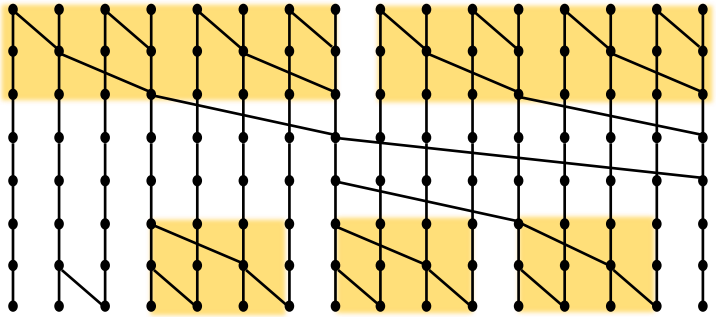
for various segment representations

optimized scan

bank conflict avoidance

Generalize Examples to Programs

Input

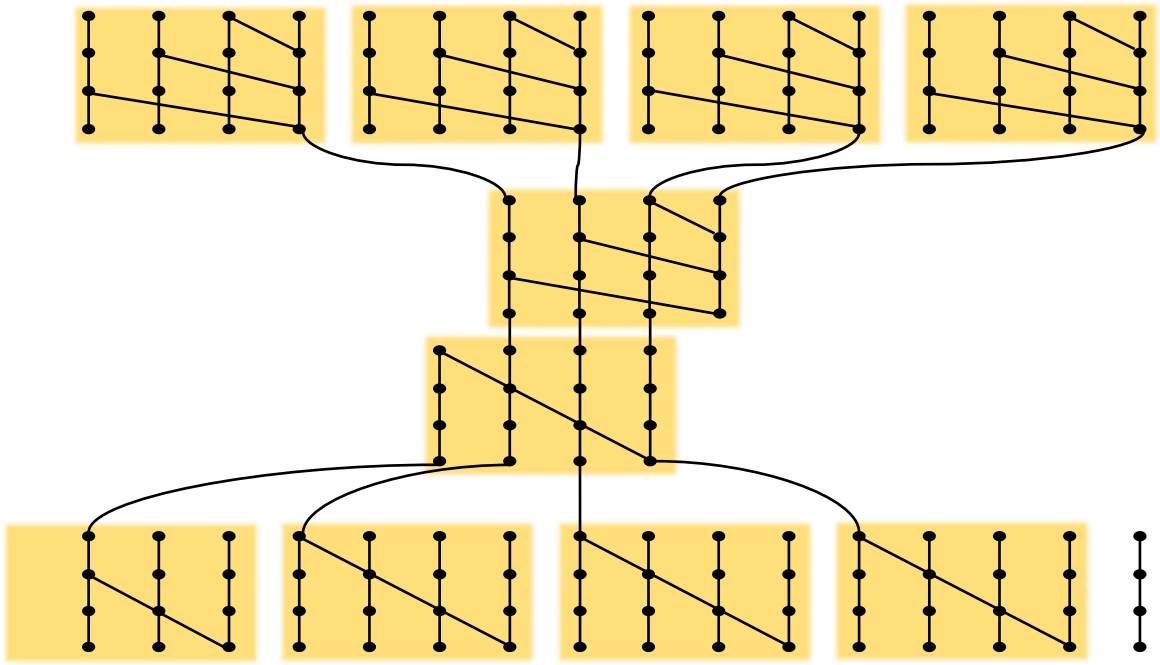


Output

```
offset = 1
for step = 1 .. log(N)
  for i from 0 to N-1 in parallel
     if((i+1)%2*offset)
        a[i] = a[i] + a[i - offset]
  offset = offset * 2
for step = 1 .. log(N)
  offset = offset / 2
  for i from 0 to N-1 in parallel
     if((i+1)%2*offset && (i + offset) < N)
        a[i + offset] = a[i] + a[i + offset]
```

Generalize Examples to Programs

Generalize into recursive program



Sketching

Input:

- Functional specification (e.g., compute the scan function)
- Sketch (a program template)
- Constraints (e.g. assertions)

Output:

- Program that meets the spec and the constraints

Hello world example of sketching:

Spec : x + x

Sketch: □ * x

Output: 2 * x

From BK instance to BK code

Input:

- Functional specification (Sequential Scan)
- Sketch

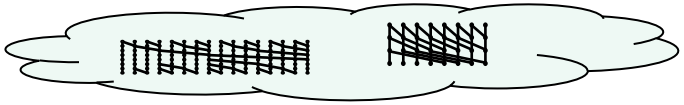
```
for step = 1 .. log(N)
  for i from 0 to N-1 in parallel
     if(□)
        a[□] = a[i] + a[□]
```

- Constraints -- Execute like the BK(16) example

Output:

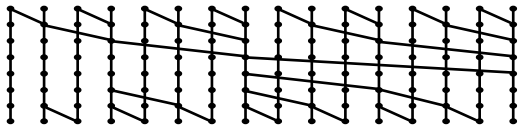
- SIMD program for BK on any (or many) N

The workflow (talk outline)



space of all algorithms

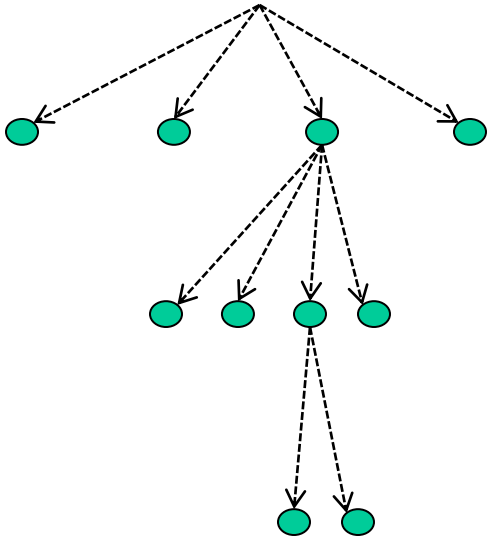
all networks that meet some constraints



instance of the algorithm

a specific network for a fixed input size

```
for d in 1 to log N
    forall i in 1 to N
        A[i] = A[i]+A[i-step]
```



program of the algorithm

works for arbitrary input size

functional variants of the algorithm

forward/backward x inclusive/exclusive

segmented scan

for various segment representations

optimized scan

bank conflict avoidance

Forward to Backward

Input:

- Forw Spec: `a[i] = a[i] + a[i-1] + .. + a[0]`
- **Back Spec**: `a[i] = a[n-1] + a[n-2] + .. a[i]`

Output:

```
for step = 1 .. log(N)
  offset = offset / 2
  for i from 0 to N-1 in parallel
     if((i+1)%2*offset == 0 && (i + offset) < N)
        a[i + offset] = a[i] + a[i + offset]
```

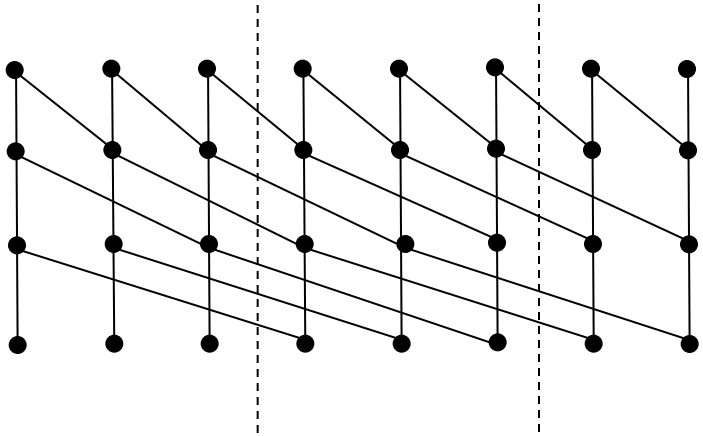
```
for step = 1 .. log(N)
  offset = offset / 2
  for i from 0 to N-1 in parallel
     if(i%2*offset == 0 && (i - offset) >= 0)
        a[i - offset] = a[i] + a[i - offset]
```

Segmented Scans

Input is divided into segments and scan is performed for each of the segments.

Segments

Original Scan Network

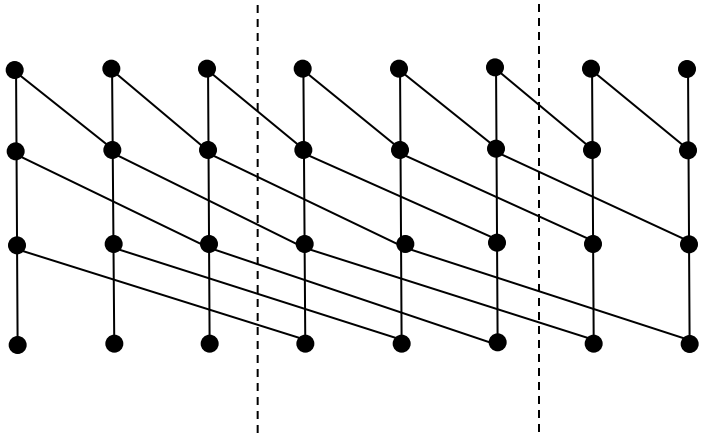


Segmented Scans

Input is divided into segments and scan is performed for each of the segments.

Segments

Segmented Scan Network



Segmented Scans

- Segments can be represented in various ways:
  - Bit mask – {1, 0, 0, 1, 0, 0, 1, 0}
    - 1 denotes starting point of each segment.
  - Header Pointers – {0,3,6}
    - Each entry denotes index corresponding to start of each segment.
  - And many more.
- We can synthesize‘bit mask’ program from an unsegmented implementation.
- Further, we would like to derive implementations from one representation to other representations.

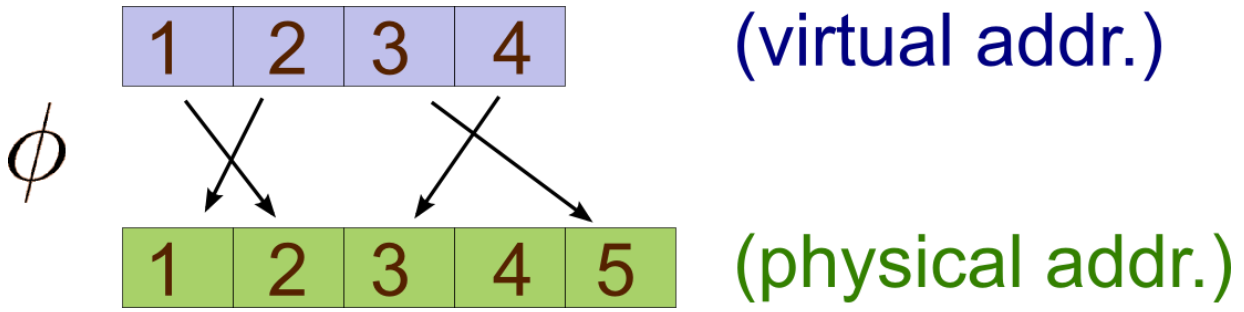
Bank conflict avoidance

Input:

- Functional specification (i.e., compute scan)
- **Sketch** (deterministic BK_2 with array re-indexing)
- Constraint – minimize bank conflicts

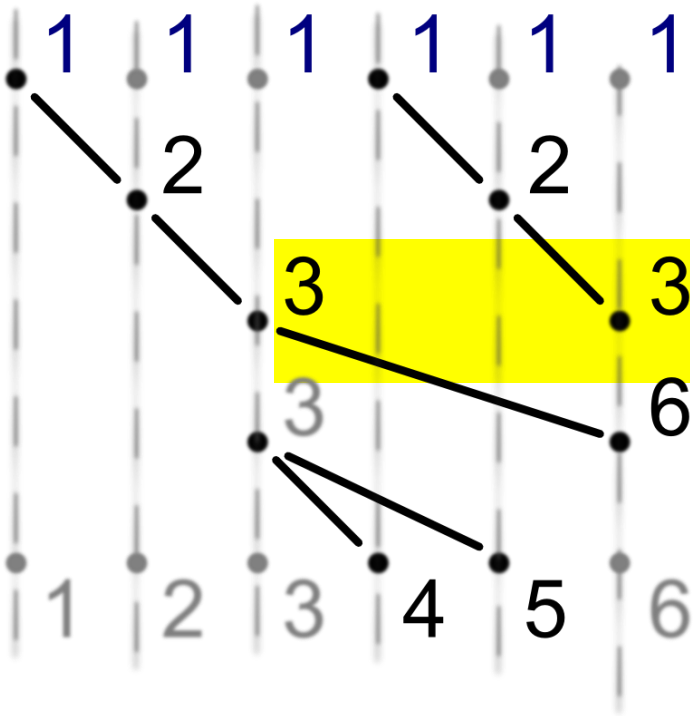
One approach to bank conflict optimization for BK_2 involves remapping array indices, so distinct memory accesses are actually handled in parallel.

We have synthesized [injective] reordering functions as shown below. Synthesis takes approximately 2 minutes.



Bank conflict avoidance

We were able to sketch another algorithm, the BK_3 circuit, in which each thread processes 3 elements [additions] instead of 2, and the recurrence operates on every 3 elements from the previous step.



(input array)

The recurrence operates on every 3 elements, not every 2

(output array)

Bank conflict avoidance

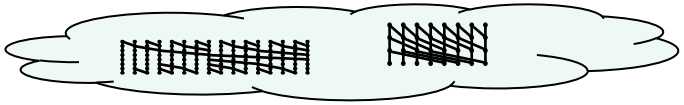
The BK_3 circuit showed a 14% performance improvement over the BK_2 circuit on a GTX 260, though it is currently outperformed [on that chip] by nVidia's non-work-efficient scan kernel.

Experimentation has shown that padding the array does not help much, since we have to calculate the new indices.

There's more…

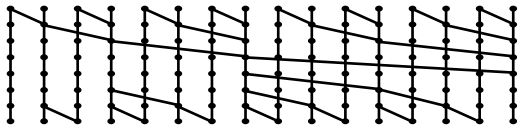
- Come visit my poster!
  - Low level CUDA model that can help synthesize programs to avoid typical RW hazards and syncthreads misuse
    - The BK_3 sketch was written with this model
  - Language support ("instrumentation") to facilitate counting bank conflicts, or checking RW hazards
  - Histogramming work
  - Ongoing Scala work for a nicer frontend language

The workflow (talk outline)



space of all algorithms

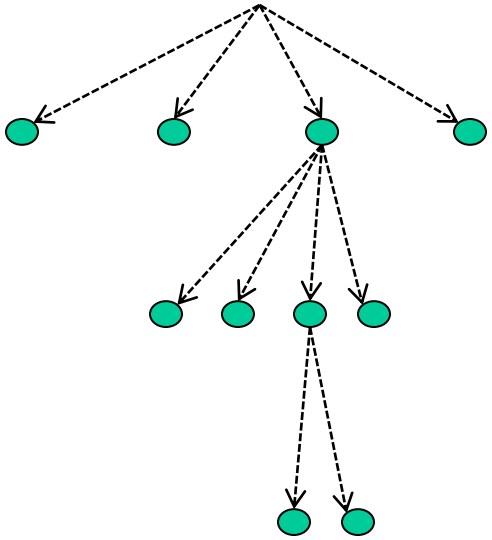
all networks that meet some constraints



instance of the algorithm

a specific network for a fixed input size

```
for d in 1 to log N
    forall i in 1 to N
        A[i] = A[i]+A[i-step]
```



program of the algorithm

works for arbitrary input size

functional variants of the algorithm

forward/backward x inclusive/exclusive

segmented scan

for various segment representations

optimized scan

bank conflict avoidance