Can Tensor Programming Be Liberated from the Fortran Data Paradigm?

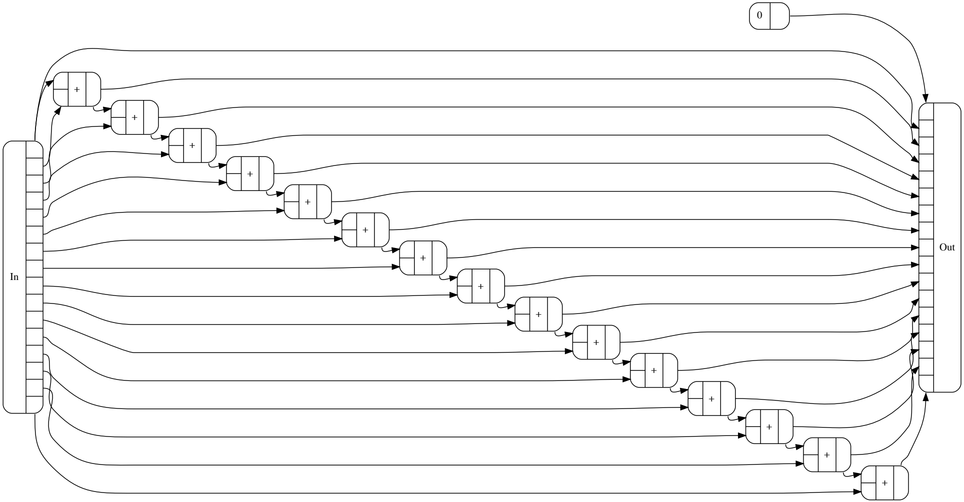
Conal Elliott

October 2021

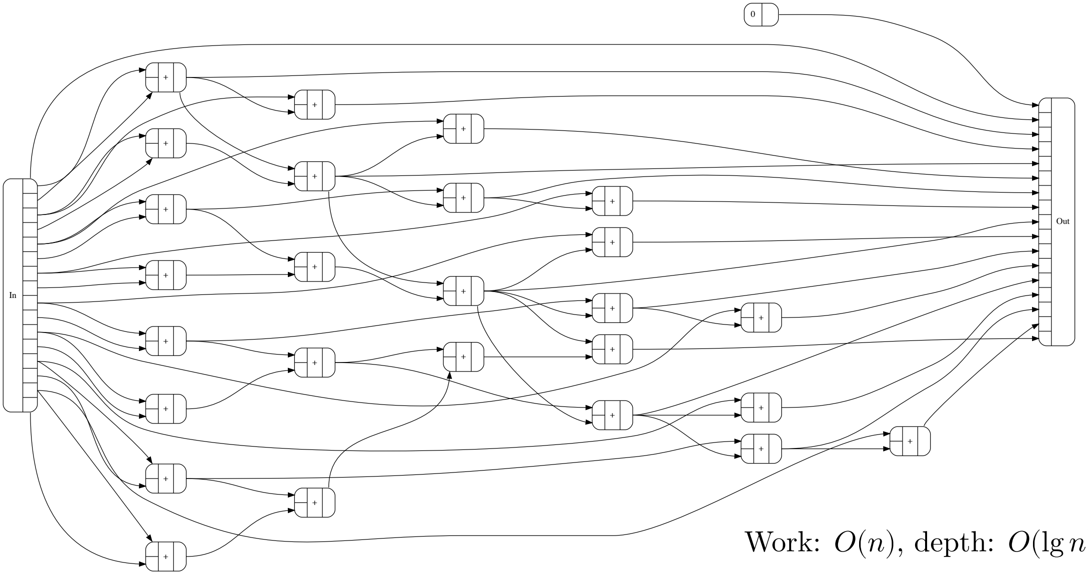
Prefix sum (left scan)

$$b_k = \sum_{i<k} a_i$$

Efficient *sequential* scan



Efficient *parallel* scan



Work: $O(n)$, depth: $O(\lg n)$.

An efficient array program (CUDA C)

```
__global__ void prescan(float *g_odata, float *g_idata, int n) {
    extern __shared__ float temp[];  // allocated on invocation
    int thid = threadIdx.x;
    int offset = 1;
    // load input into shared memory
    temp[2*thid] = g_idata[2*thid];
    temp[2*thid+1] = g_idata[2*thid+1];
    // build sum in place up the tree
    for (int d = n>>1; d > 0; d >>= 1) {
        __syncthreads();
        if (thid < d) {
            int ai = offset*(2*thid+1)-1;
            int bi = offset*(2*thid+2)-1;
            temp[bi] += temp[ai]; }
        offset *= 2; }
    // clear the last element
    if (thid == 0) { temp[n - 1] = 0; }
    // traverse down tree & build scan
    for (int d = 1; d < n; d *= 2) {
        offset >>= 1;
        __syncthreads();
        if (thid < d) {
            int ai = offset*(2*thid+1)-1;
            int bi = offset*(2*thid+2)-1;
            float t = temp[ai];
            temp[ai] = temp[bi];
            temp[bi] += t; } }
    __syncthreads();
    // write results to device memory
    g_odata[2*thid] = temp[2*thid];
    g_odata[2*thid+1] = temp[2*thid+1]; }
```

WAT

Source: Harris, Sengupta, and Owens in *GPU Gems 3*, Chapter 39

In NESL

```
function scan(a) =
  if #a == 1 then [0]
  else
    let es = even_elts(a);
        os = odd_elts(a);
        ss = scan({e+o: e in es; o in os})
    in interleave(ss,{s+e: s in ss; e in es})
```

Source: Guy Blelloch in *Programming parallel algorithms, 1990*

Still, why does it work, and how does it (correctly) generalize?

Realization

It's not naturally an array algorithm.

What is it?

Clarifying the question

$$Arr\ 2^d\ a \xrightarrow{\quad scanA \quad} Arr\ 2^d\ a$$

Clarifying the question

$$Z\ d\ a \xrightarrow{\qquad scanZ \qquad} Z\ d\ a$$

$$Arr\ 2^d\ a \xrightarrow{\qquad scanA \qquad} Arr\ 2^d\ a$$

Clarifying the question

$$\begin{tikzcd}
Z\ d\ a \arrow[r, "scanZ"] & Z\ d\ a \\
Arr\ 2^d\ a \arrow[u, "parseZ"] \arrow[r, "scanA"] & Arr\ 2^d\ a \arrow[u, "parseZ"]
\end{tikzcd}$$

Where $scanZ$ is simple to state, prove, and generalize; and $parseZ$ is formulaic in Z.

A compositional refinement

$$
\begin{tikzcd}
Z\ d\ a \arrow[rr, "scanZ"] & & Z\ d\ a \times a \\
& & \\
Arr\ 2^d\ a \arrow[uu, "parseZ"] \arrow[rr, "scanA"] & & Arr\ 2^d\ a \times a \arrow[uu, "parseZ \otimes \mathrm{id}"']
\end{tikzcd}
$$

Where $scanZ$ is simple to state, prove, and generalize; and $parseZ$ is formulaic in Z.

Wrong guess: top-down, binary trees

$$
\begin{array}{l}
\textbf{data}\ T_{\downarrow} :: * \to * \ \textbf{where} \\
\quad L :: a \to T_{\downarrow}\ a \\
\quad B :: T_{\downarrow}\ a \to T_{\downarrow}\ a \to T_{\downarrow}\ a
\end{array}
$$

$$\textbf{deriving instance}\ \mathit{Functor}$$

$$
\begin{array}{l}
\mathit{scanT}_{\downarrow} :: \mathit{Monoid}\ a \Rightarrow T_{\downarrow}\ a \to T_{\downarrow}\ a \times a \\
\mathit{scanT}_{\downarrow}\ (L\ x) \qquad = (L\ \varepsilon, x) \\
\mathit{scanT}_{\downarrow}\ (B\ u\ v) = (B\ u'\ (\mathit{fmap}\ (u_{tot} \oplus)\ v'), u_{tot} \oplus v_{tot}) \\
\quad \textbf{where} \\
\qquad (u', u_{tot}) = \mathit{scanT}_{\downarrow}\ u \\
\qquad (v', v_{tot}) = \mathit{scanT}_{\downarrow}\ v
\end{array}
$$

Work: $O(n \lg n)$, depth: $O(\lg n)$.

Refined wrong guess: top-down, binary, *perfect* trees

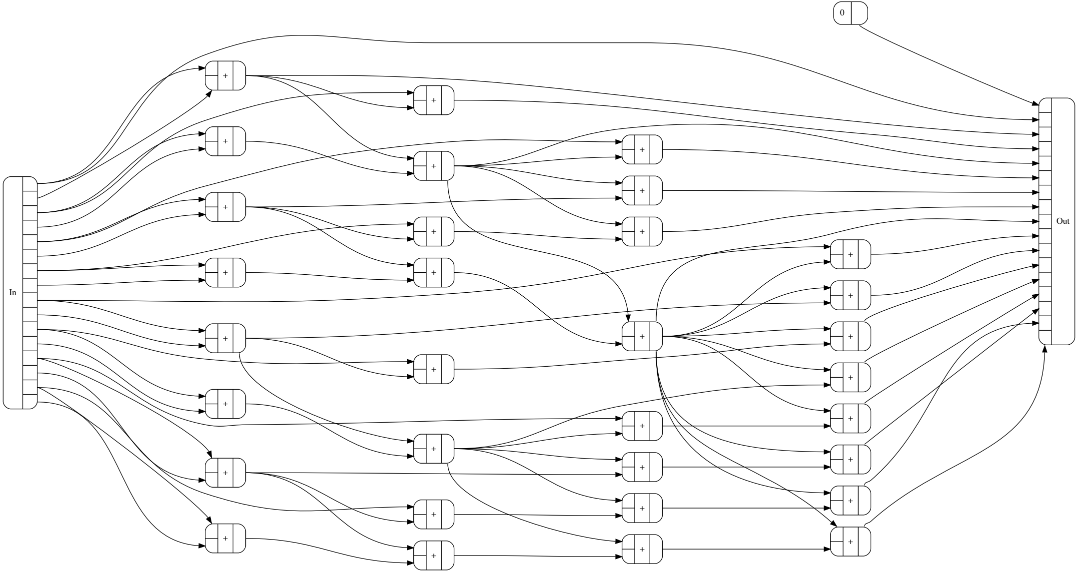
$$
\begin{array}{l}
\mathbf{data}\ T_{\downarrow} :: \mathit{Nat} \to * \to *\ \mathbf{where} \\
\quad L :: a \to T_{\downarrow}\ 0\ a \\
\quad B :: T_{\downarrow}\ d\ a \to T_{\downarrow}\ d\ a \to T_{\downarrow}\ (d+1)\ a
\end{array}
$$

$$\mathbf{deriving\ instance}\ \mathit{Functor}\ (T_{\downarrow}\ d)$$

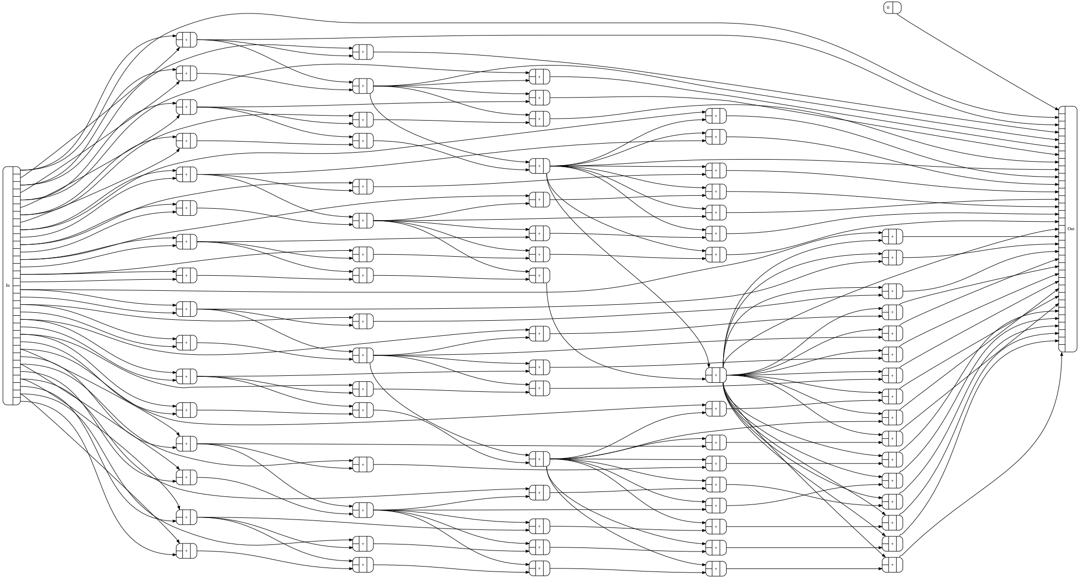
$$
\begin{array}{l}
\mathit{scanT}_{\downarrow} :: \mathit{Monoid}\ a \Rightarrow T_{\downarrow}\ d\ a \to T_{\downarrow}\ d\ a \times a \\
\mathit{scanT}_{\downarrow}\ (L\ x) \quad\ \ = (L\ \varepsilon, x) \\
\mathit{scanT}_{\downarrow}\ (B\ u\ v) = (B\ u'\ (\mathit{fmap}\ (u_{tot} \oplus)\ v'), u_{tot} \oplus v_{tot}) \\
\quad \mathbf{where} \\
\qquad (u', u_{tot}) = \mathit{scanT}_{\downarrow}\ u \\
\qquad (v', v_{tot}) = \mathit{scanT}_{\downarrow}\ v
\end{array}
$$

Work: $O(n \lg n)$, depth: $O(\lg n)$.

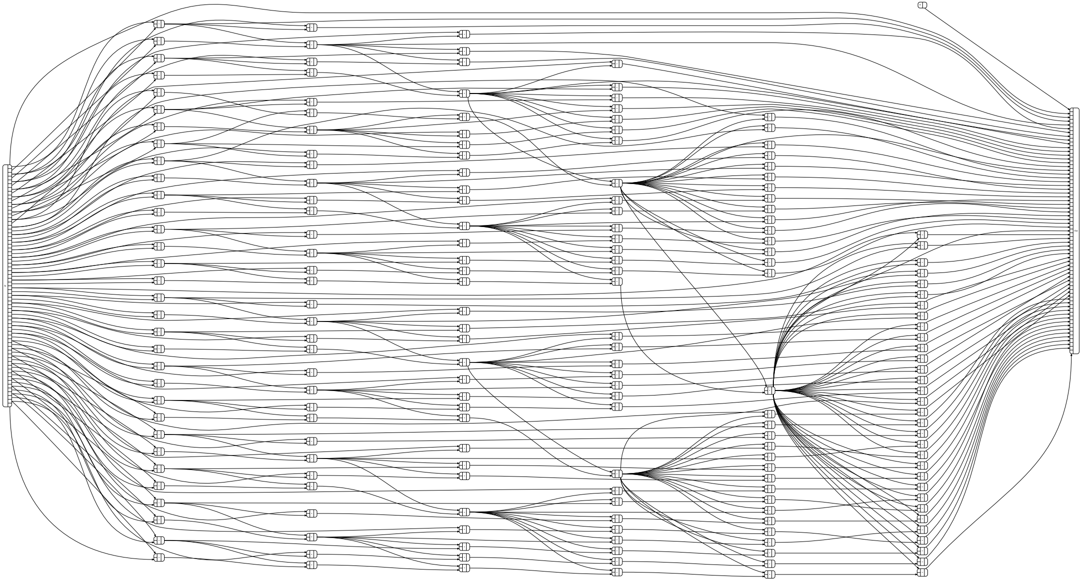
Top-down tree scan



Top-down tree scan



Top-down tree scan



But right answer

$$\begin{tikzcd}
T_{\downarrow}\ d\ a \arrow[r, "scanT_{\downarrow}"] & T_{\downarrow}\ d\ a \times a \\
Arr\ 2^d\ a \arrow[u, "parseT_{\downarrow}"] \arrow[r, "scanA"] & Arr\ 2^d\ a \times a \arrow[u, "parseT_{\downarrow} \otimes \mathrm{id}"']
\end{tikzcd}$$

Refined wrong guess: top-down, binary, *perfect* trees

$$
\begin{array}{l}
\mathbf{data}\ T_{\downarrow} :: \mathit{Nat} \to * \to *\ \mathbf{where} \\
\quad L :: a \to T_{\downarrow}\ 0\ a \\
\quad B :: T_{\downarrow}\ d\ a \to T_{\downarrow}\ d\ a \to T_{\downarrow}\ (d+1)\ a \\
\mathbf{deriving\ instance}\ \mathit{Functor}\ (T_{\downarrow}\ d)
\end{array}
$$

$$
\begin{array}{l}
\mathit{scanT}_{\downarrow} :: \mathit{Monoid}\ a \Rightarrow T_{\downarrow}\ d\ a \to T_{\downarrow}\ d\ a \times a \\
\mathit{scanT}_{\downarrow}\ (L\ x) = (L\ \varepsilon, x) \\
\mathit{scanT}_{\downarrow}\ (B\ u\ v) = (B\ u'\ (\mathit{fmap}\ (u_{tot} \oplus)\ v'), u_{tot} \oplus v_{tot}) \\
\quad \mathbf{where} \\
\qquad (u', u_{tot}) = \mathit{scanT}_{\downarrow}\ u \\
\qquad (v', v_{tot}) = \mathit{scanT}_{\downarrow}\ v
\end{array}
$$

Wrong guess refactored

$$\textbf{data}\ P\ a = a \mathbin{:\#} a\ \textbf{deriving}\ \mathit{Functor}$$

$$\begin{aligned}
&\textbf{data}\ T_{\downarrow} :: \mathit{Nat} \to * \to *\ \textbf{where} \\
&\quad L :: a \to T_{\downarrow}\ 0\ a \\
&\quad B :: P\ (T_{\downarrow}\ d\ a) \to T_{\downarrow}\ (d+1)\ a \\
&\textbf{deriving instance}\ \mathit{Functor}\ (T_{\downarrow}\ d)
\end{aligned}$$

$$\begin{aligned}
&\mathit{scanT}_{\downarrow} :: \mathit{Monoid}\ a \Rightarrow T_{\downarrow}\ d\ a \to T_{\downarrow}\ d\ a \times a \\
&\mathit{scanT}_{\downarrow}\ (L\ x) = (L\ \varepsilon, x) \\
&\mathit{scanT}_{\downarrow}\ (B\ (u \mathbin{:\#} v)) = (B\ (u' \mathbin{:\#} \mathit{fmap}\ (u_{tot}\oplus)\ v'), u_{tot} \oplus v_{tot}) \\
&\quad \textbf{where} \\
&\qquad (u', u_{tot}) = \mathit{scanT}_{\downarrow}\ u \\
&\qquad (v', v_{tot}) = \mathit{scanT}_{\downarrow}\ v
\end{aligned}$$

Wrong guess: top-down, binary, perfect trees

$$\textbf{data}\ P\ a = a :\# \ a\ \textbf{deriving}\ \mathit{Functor}$$

$$\begin{array}{l}
\textbf{data}\ T_{\downarrow} :: \mathit{Nat} \to * \to *\ \textbf{where} \\
\quad L :: a \to T_{\downarrow}\ 0\ a \\
\quad B :: P\ (T_{\downarrow}\ d\ a) \to T_{\downarrow}\ (d+1)\ a \\
\textbf{deriving instance}\ \mathit{Functor}\ (T_{\downarrow}\ d)
\end{array}$$

$$\begin{array}{l}
\mathit{scanP} :: \mathit{Monoid}\ a \Rightarrow P\ a \to P\ a \times a \\
\mathit{scanP}\ (x :\#\ y) = (\varepsilon :\#\ x, x \oplus y)
\end{array}$$

$$\begin{array}{l}
\mathit{scanT}_{\downarrow} :: \mathit{Monoid}\ a \Rightarrow T_{\downarrow}\ d\ a \to T_{\downarrow}\ d\ a \times a \\
\mathit{scanT}_{\downarrow}\ (L\ x) = (L\ \varepsilon, x) \\
\mathit{scanT}_{\downarrow}\ (B\ ts) = (B\ (\mathit{zipWithP}\ \mathit{tweak}\ \mathit{tots}'\ \mathit{ts}'), \mathit{tot}) \\
\quad \textbf{where} \\
\qquad (\mathit{ts}', \mathit{tots}) \ \ = \mathit{unzipP}\ (\mathit{fmap}\ \mathit{scanT}_{\downarrow}\ \mathit{ts}) \\
\qquad (\mathit{tots}', \mathit{tot}) = \mathit{scanP}\ \mathit{tots} \\
\qquad \mathit{tweak}\ x \qquad = \mathit{fmap}\ (x \oplus)
\end{array}$$

Work: $O(n \lg n)$, depth: $O(\lg n)$.

Right guess: *bottom-up*, perfect, binary trees

$$\textbf{data}\ P\ a = a :\#\ a\ \textbf{deriving}\ \mathit{Functor}$$

$$\begin{array}{l}
\textbf{data}\ T_{\uparrow} :: \mathit{Nat} \to * \to *\ \textbf{where} \\
\quad L :: a \to T_{\uparrow}\ 0\ a \\
\quad B :: T_{\uparrow}\ d\ (P\ a) \to T_{\uparrow}\ (d+1)\ a \\
\textbf{deriving instance}\ \mathit{Functor}\ (T_{\uparrow}\ d)
\end{array}$$

$$\begin{array}{l}
\mathit{scanP} :: \mathit{Monoid}\ a \Rightarrow P\ a \to P\ a \times a \\
\mathit{scanP}\ (x :\#\ y) = (\varepsilon :\#\ x, x \oplus y)
\end{array}$$

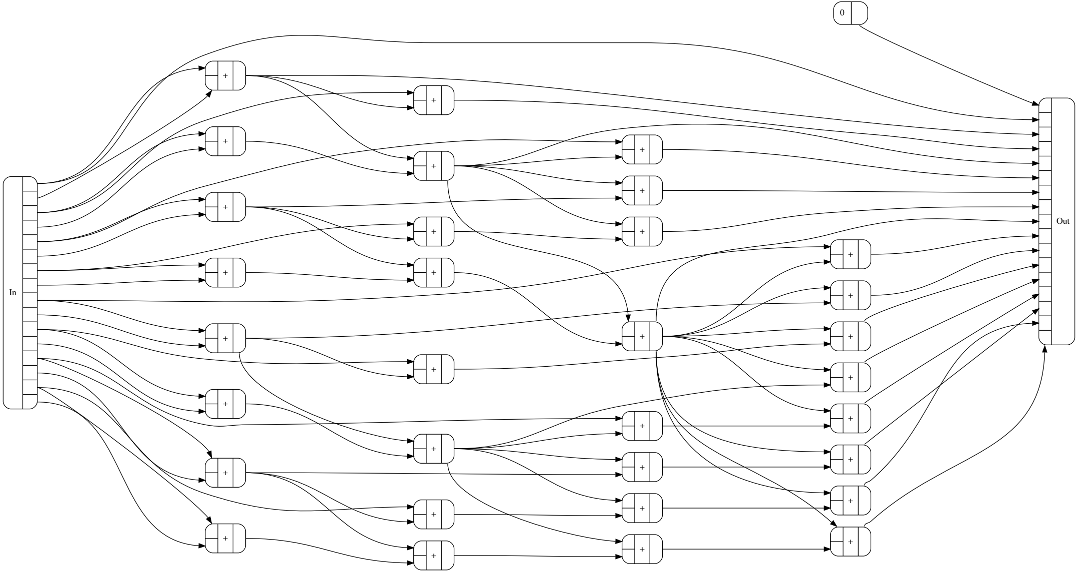
$$\begin{array}{l}
\mathit{scanT}_{\uparrow} :: \mathit{Monoid}\ a \Rightarrow T_{\uparrow}\ d\ a \to T_{\uparrow}\ d\ a \times a \\
\mathit{scanT}_{\uparrow}\ (L\ x) = (L\ \varepsilon, x) \\
\mathit{scanT}_{\uparrow}\ (B\ \mathit{ps}) = (B\ (\mathit{zipWithT}_{\uparrow}\ \mathit{tweak}\ \mathit{tots}'\ \mathit{ps}'), \mathit{tot}) \\
\quad \textbf{where} \\
\qquad (\mathit{ps}', \mathit{tots}) \ = \mathit{unzipT}_{\uparrow}\ (\mathit{fmap}\ \mathit{scanP}\ \mathit{ps}) \\
\qquad (\mathit{tots}', \mathit{tot}) = \mathit{scanT}_{\uparrow}\ \mathit{tots} \\
\qquad \mathit{tweak}\ x \qquad = \mathit{fmap}\ (x \oplus)
\end{array}$$

Work: $O(n)$, depth: $O(\lg n)$.

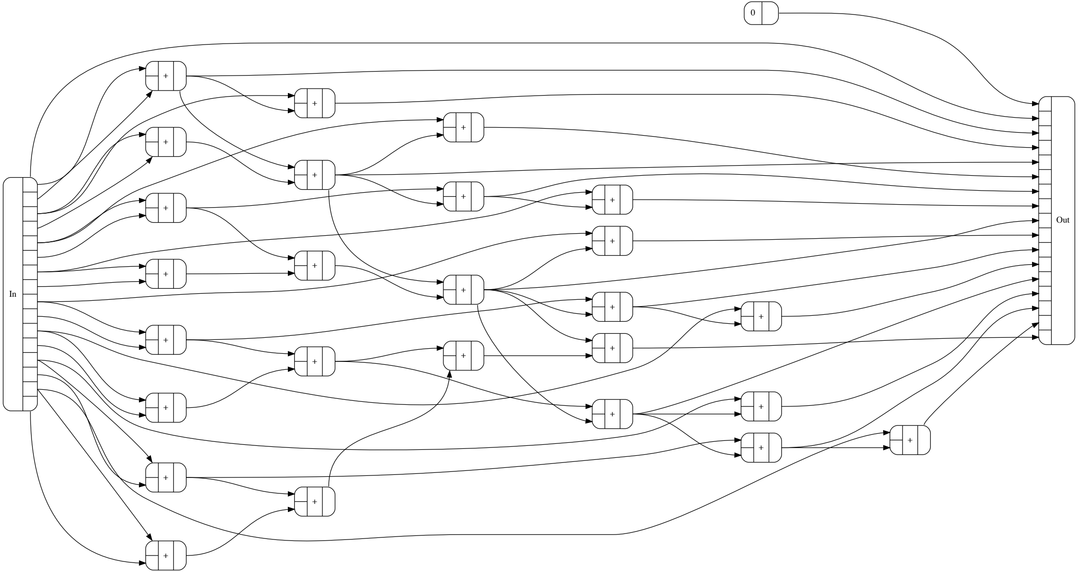
And right answer

$$
\begin{tikzcd}
T_{\uparrow}\ d\ a \arrow[r, "scanT_{\uparrow}"] & T_{\uparrow}\ d\ a \times a \\
Arr\ 2^d\ a \arrow[u, "parseT_{\uparrow}"] \arrow[r, "scanA"] & Arr\ 2^d\ a \times a \arrow[u, "parseT_{\uparrow} \otimes \mathrm{id}"']
\end{tikzcd}
$$

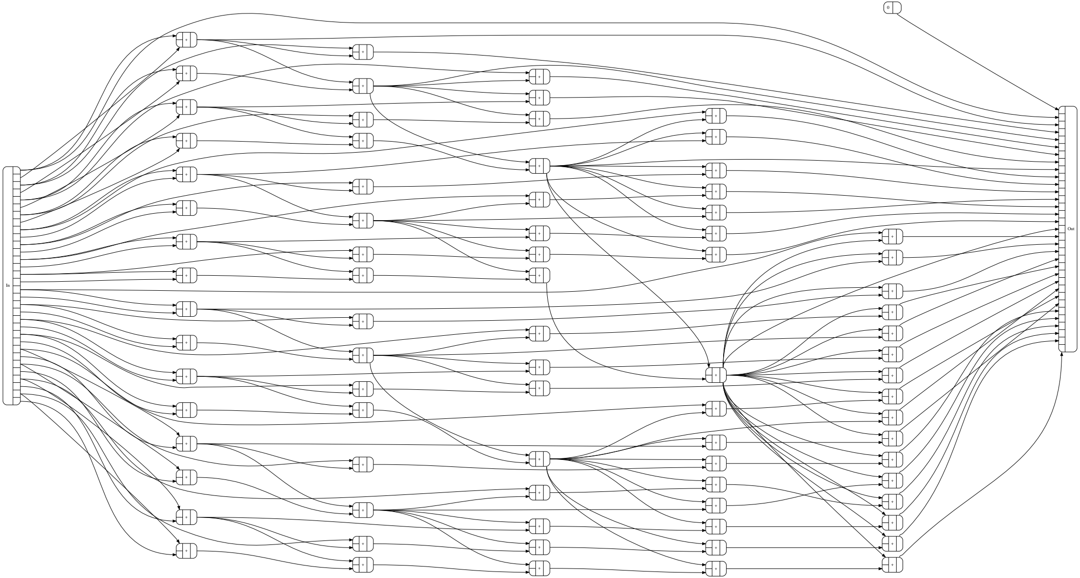
Top-down tree scan



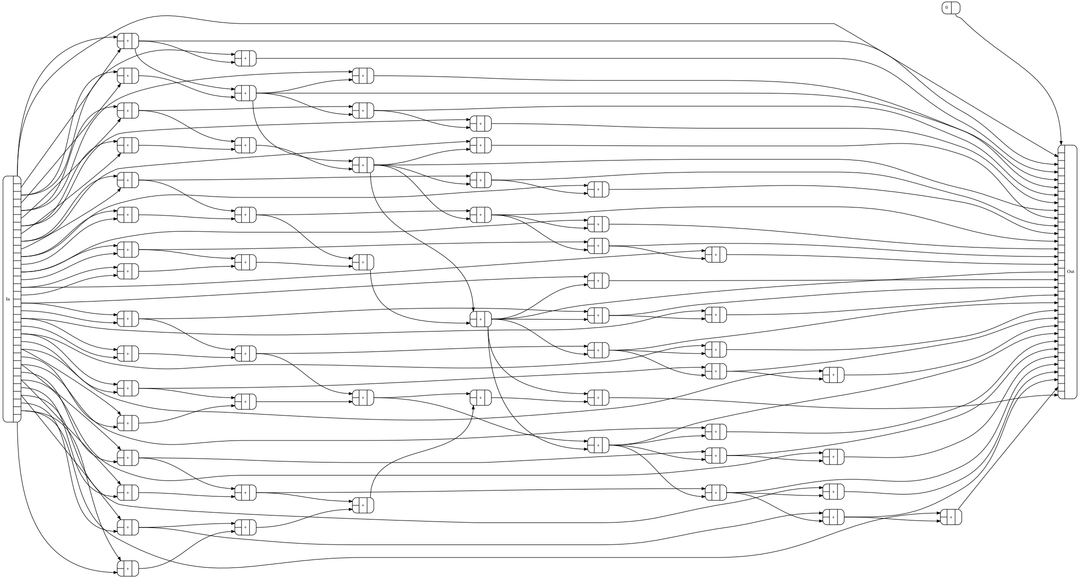
Bottom-up tree scan



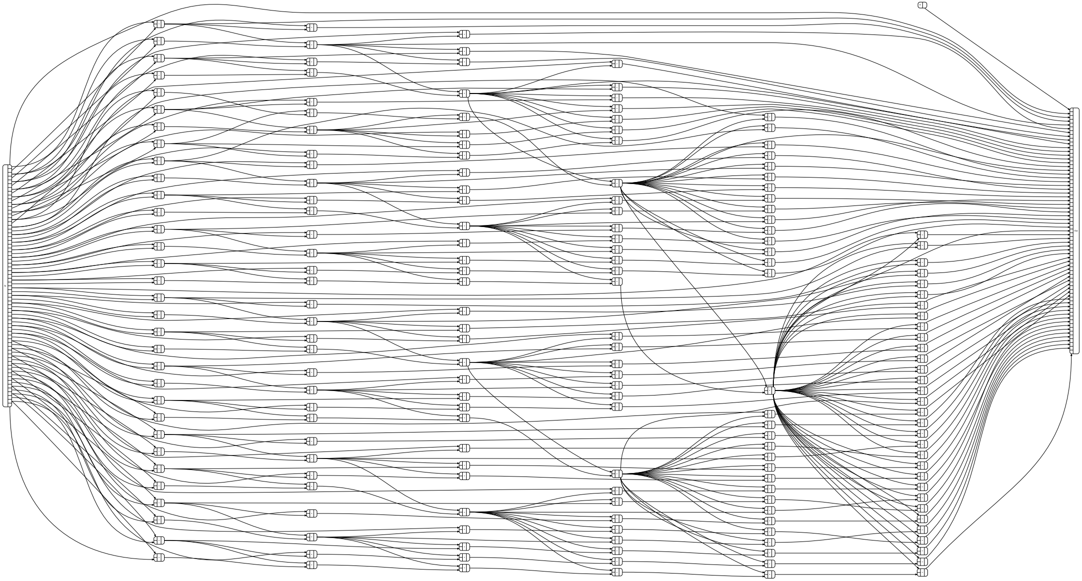
Top-down tree scan



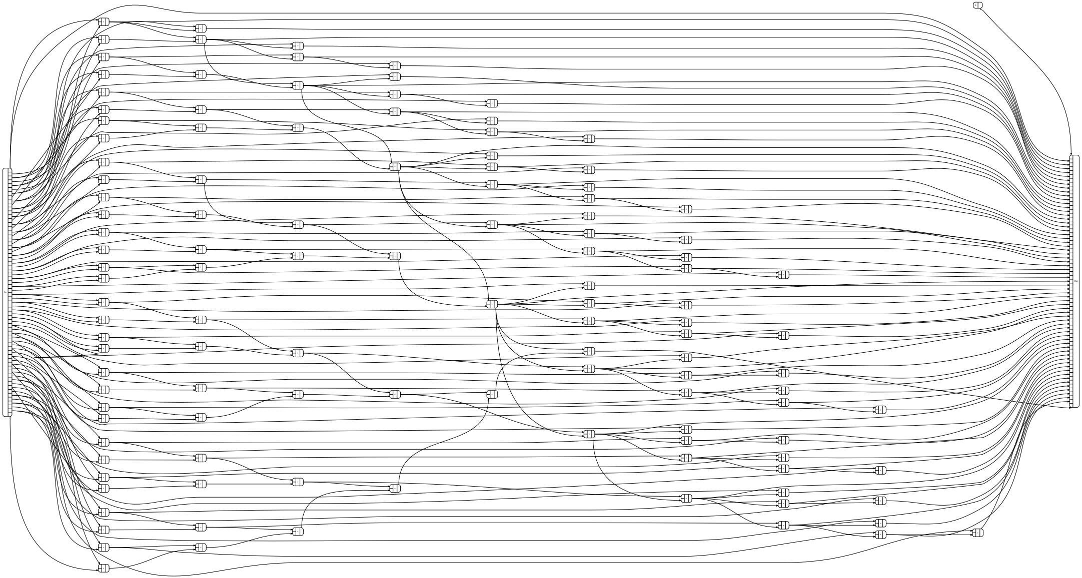
Bottom-up tree scan



Top-down tree scan



Bottom-up tree scan

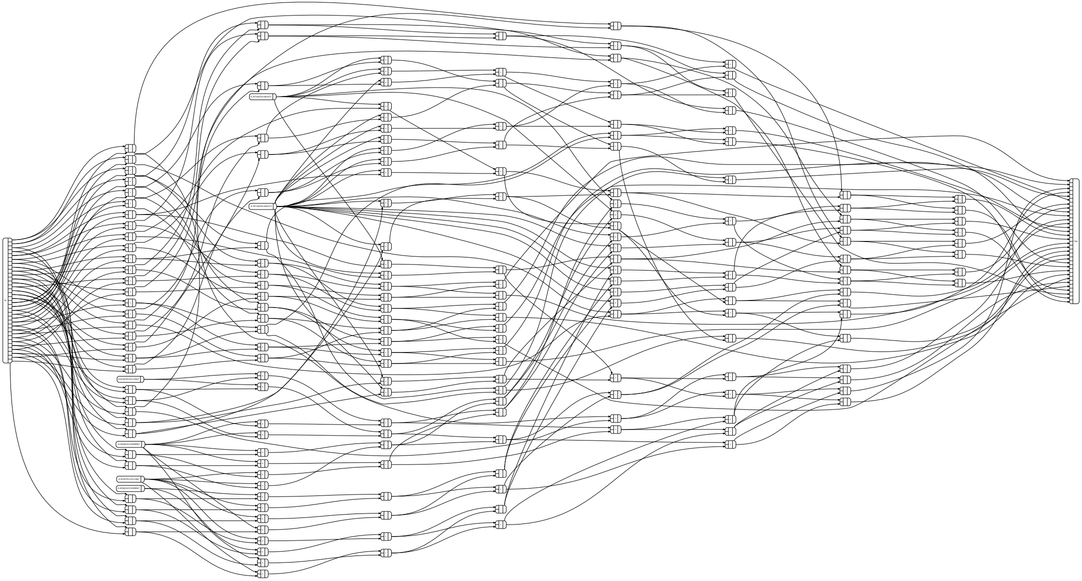


FFT

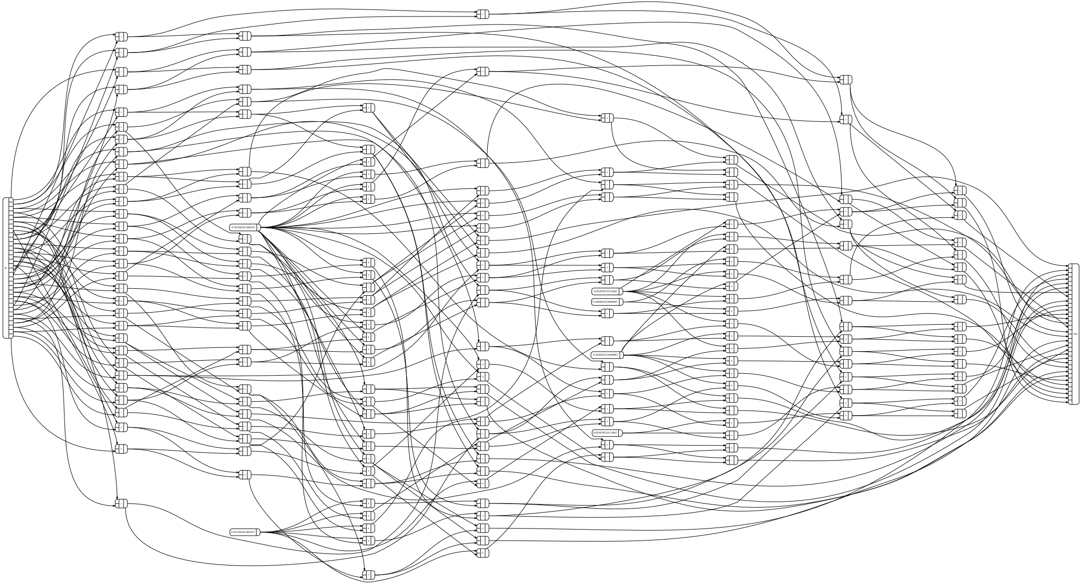
FFT decomposes similarly, yielding classic DIT & DIF algorithms.

See *Generic functional parallel algorithms: Scan and FFT* (ICFP 2017).

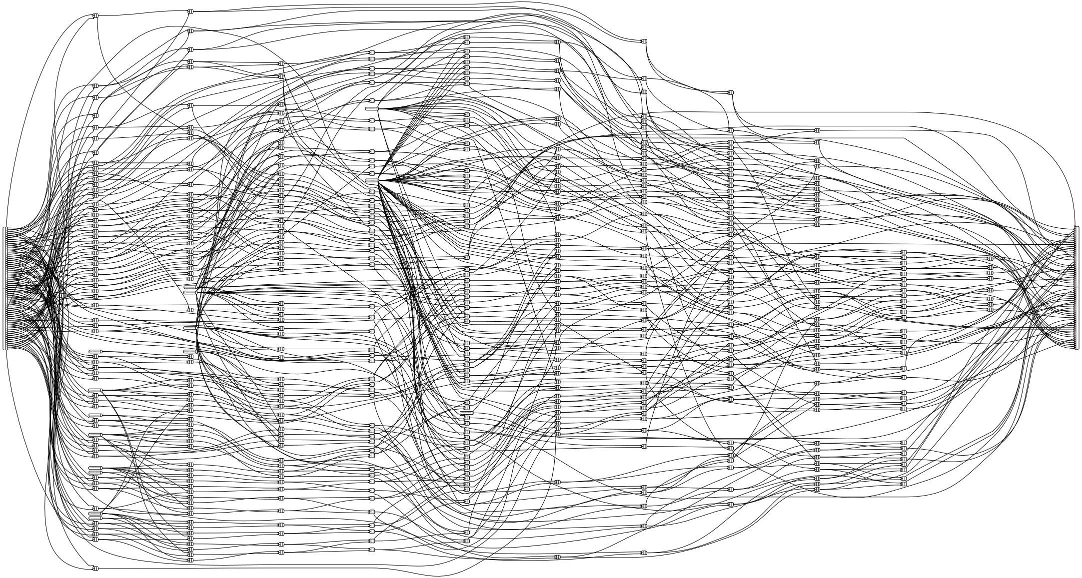
Top-down tree FFT (DIT)



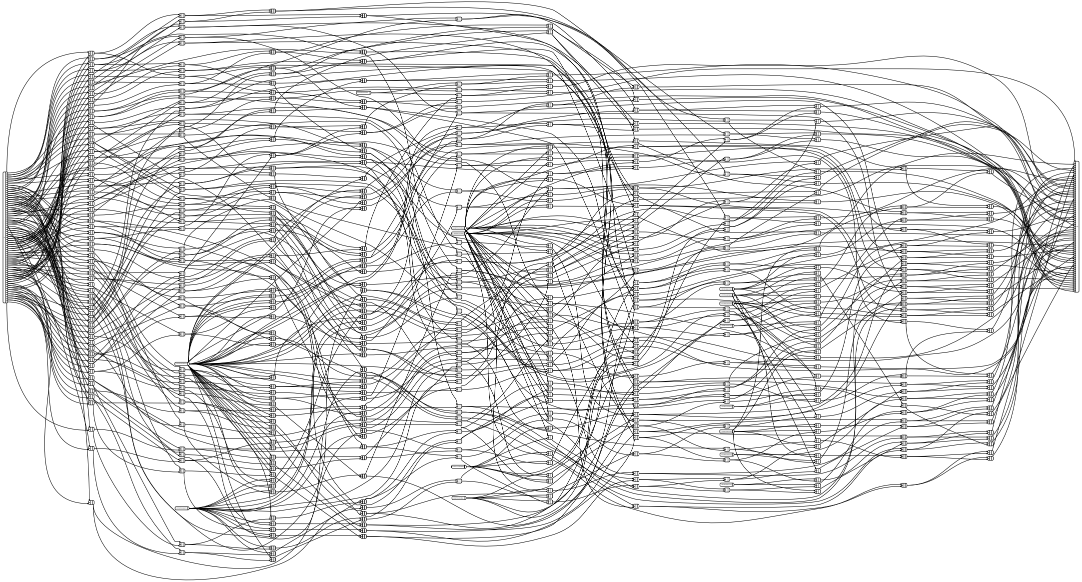
Bottom-up tree FFT (DIF)



Top-down tree FFT (DIT)



Bottom-up tree FFT (DIF)



Generalizing

Re-express parallel algorithm via singletons, products, and *compositions*.

Recomposing yields infinite family of *correct* parallel algorithms on tries.

All such tries are isomorphic to arrays (“parsing/unparsing”).

Arrays are numeric exponentials

$$
\begin{aligned}
a^0 &= 1 \\
a^1 &= a \\
a^{m+n} &= a^m \times a^n \\
a^{m \times n} &= (a^m)^n
\end{aligned}
$$

Tries are general exponentials

$$
\begin{array}{llcl}
\textbf{data}\ U & a & = & U \\
\textbf{data}\ I & a & = & I\ a \\
\textbf{data}\ (f \times g) & a & = & X\ (f\ a)\ (g\ a) \\
\textbf{data}\ (g \circ f) & a & = & O\ (g\ (f\ a))
\end{array}
$$

$$
\begin{array}{ll}
\mathit{Arr}\ 0 & \cong U \\
\mathit{Arr}\ 1 & \cong I \\
\mathit{Arr}\ (m + n) & \cong \mathit{Arr}\ m \times \mathit{Arr}\ n \\
\mathit{Arr}\ (m \times n) & \cong \mathit{Arr}\ n \circ \mathit{Arr}\ m
\end{array}
$$

Tries are general exponentials

$$
\begin{array}{ll}
\textbf{data}\ U\ \ a = U & \tilde{0}\ :: Arr\,0 \cong U \\
\textbf{data}\ I\ \ a = I\ a & \tilde{1}\ :: Arr\,1 \cong I \\
\textbf{data}\ (f \times g)\ a = X\ (f\ a)\ (g\ a) & (\tilde{+}) :: Arr\ m \cong f \to Arr\ n \cong g \to Arr\ (m + n) \cong f \times g \\
\textbf{data}\ (g \circ f)\ a = O\ (g\ (f\ a)) & (\tilde{*}) :: Arr\ m \cong f \to Arr\ n \cong g \to Arr\ (m * n) \cong g \circ f
\end{array}
$$

$$
\begin{array}{l}
\quad\text{-- ``Parse/unparse''} \\
\textbf{data}\ (\cong) :: (* \to *) \to (* \to *) \to *\ \textbf{where} \\
\quad Iso :: (\forall a. f\ a \to g\ a) \to (\forall a. g\ a \to f\ a) \to f \cong g \\
\qquad \text{-- Plus isomorphism proof}
\end{array}
$$

Vectors

$$\bar{n} = \overbrace{I \times \cdots \times I}^{n \text{ times}}$$

Right-associated ("cons"):

$$\begin{aligned}
&\textbf{type family } \overrightarrow{d} \textbf{ where} \\
&\quad \overrightarrow{0} \quad\;\;\, = U \\
&\quad \overrightarrow{d+1} = I \times \overrightarrow{d}
\end{aligned}$$

Left-associated ("snoc"):

$$\begin{aligned}
&\textbf{type family } \overleftarrow{d} \textbf{ where} \\
&\quad \overleftarrow{Z} \quad\;\;\, = U \\
&\quad \overleftarrow{d+1} = \overleftarrow{d} \times I
\end{aligned}$$

Perfect trees

$$h^d = \overbrace{h \circ \cdots \circ h}^{d \text{ times}}$$

Right-associated ("top-down"):

$$\begin{array}{l} \textbf{type family } h^{\downarrow d} \textbf{ where} \\ \quad h^{\downarrow 0} \quad\;\;\, = I \\ \quad h^{\downarrow d+1} = h \circ h^{\downarrow d} \end{array}$$

Left-associated ("bottom-up"):

$$\begin{array}{l} \textbf{type family } h^{\uparrow d} \textbf{ where} \\ \quad h^{\uparrow 0} \quad\;\;\, = I \\ \quad h^{\uparrow d+1} = h^{\uparrow d} \circ h \end{array}$$

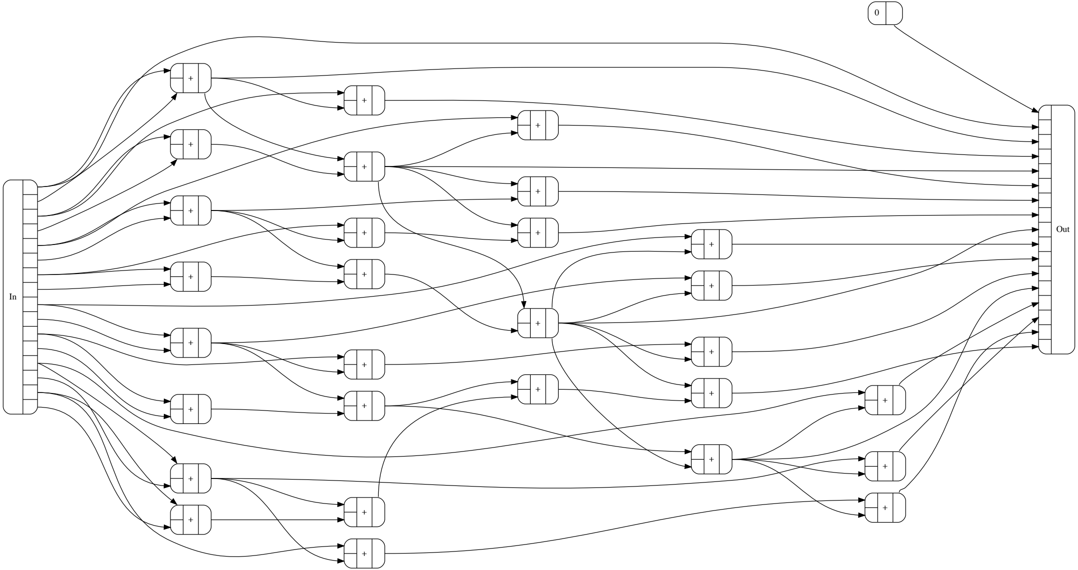
Bushes

type family 2^{2^d} **where**
$2^{2^0} = 2$
$2^{2^{d+1}} = 2^{2^d} \circ 2^{2^d}$

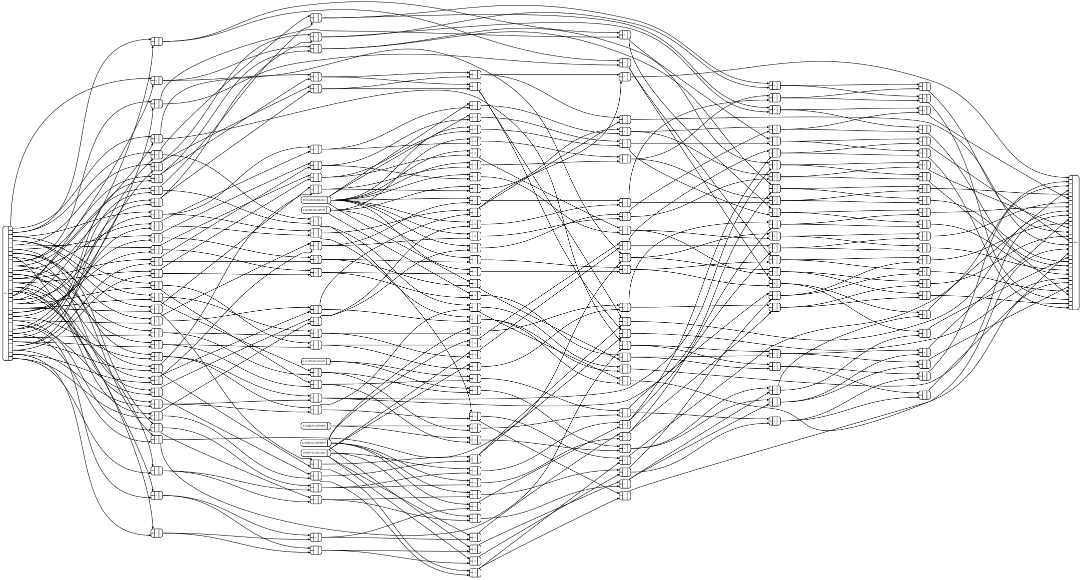
Notes:

- Variation of *Bush* type in *Nested Datatypes* by Bird & Meertens.
- Size 2^{2^n}, i.e., $2, 4, 16, 256, 65536, \ldots$.
- Composition-balanced counterpart to $2^{\uparrow 2^d}$ and $2^{\downarrow 2^d}$.
- Easily generalizes beyond pairing and squaring.

Bush scan



Bush FFT



Scan comparison

Size 16:

Type	work	depth
$2^{\downarrow 4}$	32	4
$2^{\uparrow 4}$	26	6
2^{2^2}	29	5

Size 256:

Type	work	depth
$2^{\downarrow 8}$	1024	8
$2^{\uparrow 8}$	502	14
2^{2^3}	718	10

FFT comparison

Size 16:

Type	$+$	$\times$	$-$	work	depth
$2^{\downarrow 4}$	74	40	74	188	8
$2^{\uparrow 4}$	74	40	74	188	8
2^{2^2}	72	32	72	176	6

Size 256:

Type	$+$	$\times$	$-$	work	depth
$2^{\downarrow 8}$	2690	2582	2690	7692	20
$2^{\uparrow 8}$	2690	2582	2690	7692	20
2^{2^3}	2528	1922	2528	6978	14

“But computer memory is arrays!”

- Systematically convert natural algorithms to accommodate.

- It’s really $2^{\uparrow d}$ or $2^{\downarrow d}$.

Conclusions

- Alternative to array programming:
  - Reveals algorithm essence, connections, and generalizations.
  - Free of index computations (safe and uncluttered).
  - Translates to array program safely and systematically.
- Four well-known parallel algorithms: $h^{\downarrow d}$, $h^{\uparrow d}$.
- Two new and possibly useful algorithms: 2^{2^d}.
- Other examples: arithmetic, linear algebra, polynomials, bitonic sort.
- *Optimization matters but harms clarity and composability, so do it late.*