Powerlist

COMS W4995 002: Parallel Functional Programming Fall 2021

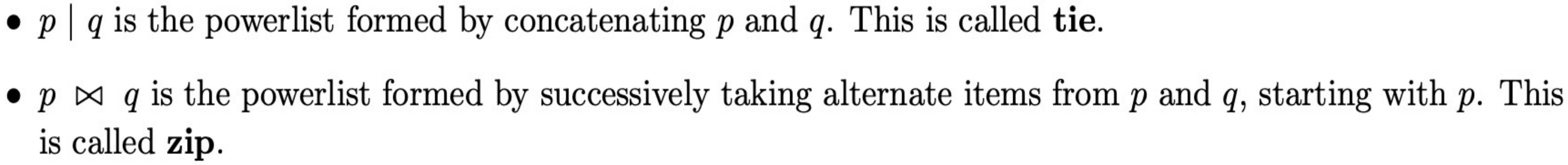
Yash Datta (yd2590)

21/12/2021

Powerlist

- A new recursive DS for data parallel algorithms

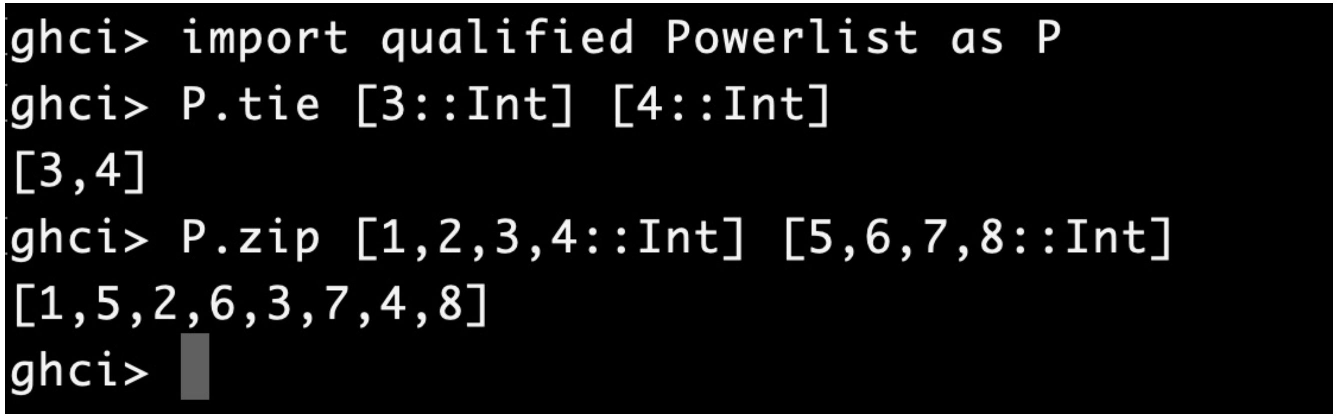
- Base case :   A list of 1 element
- Longer power lists constructed from 2 powerlist of same length and having similar elements using 2 operators



  - $p \mid q$ is the powerlist formed by concatenating p and q. This is called **tie**.
  - $p \bowtie q$ is the powerlist formed by successively taking alternate items from p and q, starting with p. This is called **zip**.

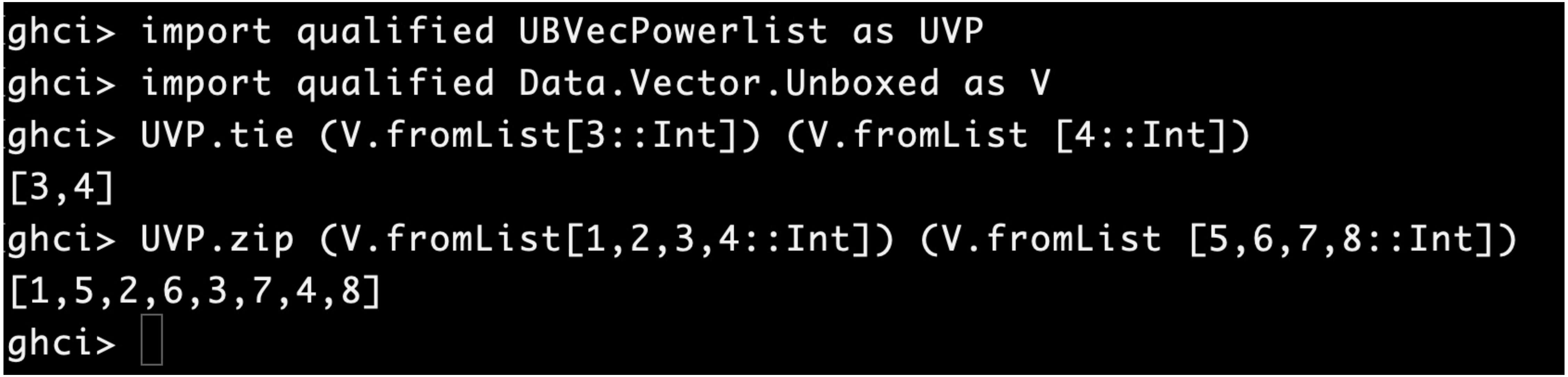
2 implementations

- Using List : Powerlist



```
ghci> import qualified Powerlist as P
ghci> P.tie [3::Int] [4::Int]
[3,4]
ghci> P.zip [1,2,3,4::Int] [5,6,7,8::Int]
[1,5,2,6,3,7,4,8]
ghci> 
```

- Using Unboxed Vectors: UBVecPowerlist

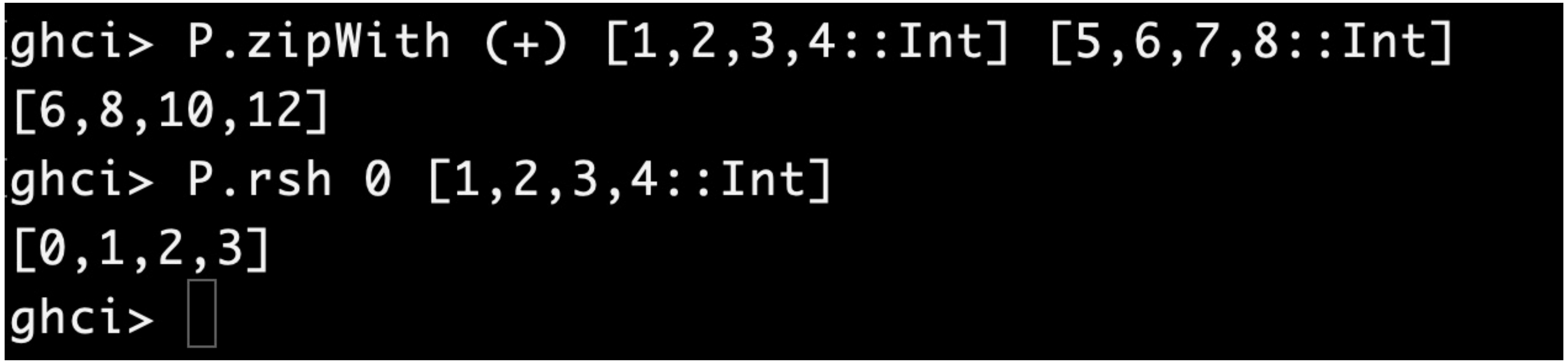


```
ghci> import qualified UBVecPowerlist as UVP
ghci> import qualified Data.Vector.Unboxed as V
ghci> UVP.tie (V.fromList[3::Int]) (V.fromList [4::Int])
[3,4]
ghci> UVP.zip (V.fromList[1,2,3,4::Int]) (V.fromList [5,6,7,8::Int])
[1,5,2,6,3,7,4,8]
ghci> 
```

Powerlist Operators

- $p \oplus q$ is the powerlist obtained by applying the binary scalar operator $\bigoplus$ on the elements of p and q at the same position in the 2 lists.
- L^* is the powerlist obtained by shifting the powerlist L by one. the effect of shifting is to append a 0 to the left and discard the rightmost element.

Note that 0 is considered the left identity element of $\oplus$, i.e. $0 \bigoplus x = x$.

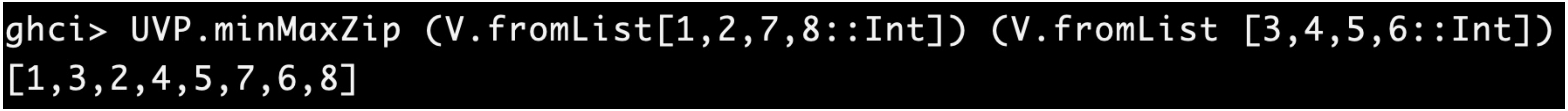


```
ghci> P.zipWith (+) [1,2,3,4::Int] [5,6,7,8::Int]
[6,8,10,12]
ghci> P.rsh 0 [1,2,3,4::Int]
[0,1,2,3]
ghci> 
```

Powerlist Operators

- Another operator for sorting

$$p \updownarrow q = (p\ min\ q) \bowtie (p\ max\ q)$$



```
ghci> UVP.minMaxZip (V.fromList[1,2,7,8::Int]) (V.fromList [3,4,5,6::Int])
[1,3,2,4,5,7,6,8]
```

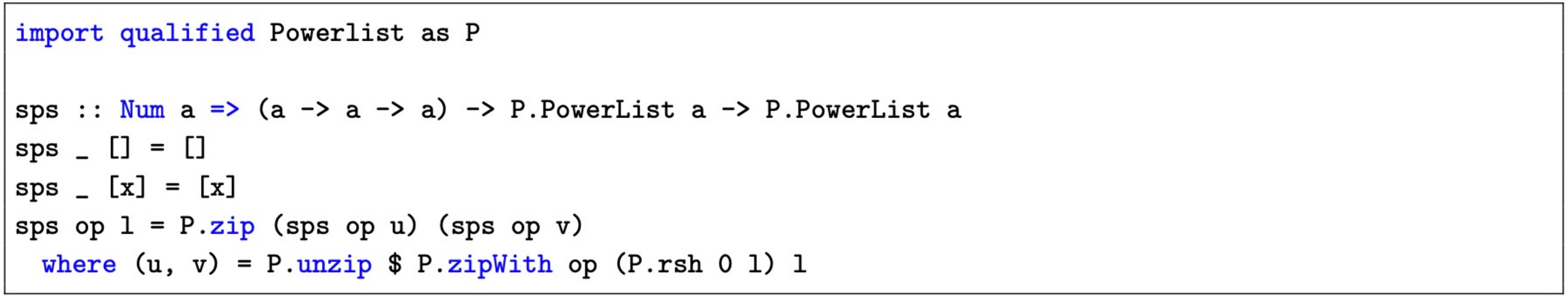
Algorithms

- Demonstrate use of powerlist in
  - Scan
    - Simple Prefix Sum
      - SPSPL
      - SPSPLPar1
      - SPSPLPar2
      - SPSPLPar3
      - SPSUBVecPLPar
    - Ladner Fischer Scheme
      - LDFPar
      - LDFUBVecPLPar
      - LDFChunkUBVecPLPar
  - Sort
    - Batcher Merge Sort

Simple Prefix Sum

$$
\begin{aligned}
&sps\ \langle x \rangle = \langle x \rangle \\
&sps\ L = (sps\ u)\ \bowtie\ (sps\ v) \\
&\qquad where\ u\ \bowtie\ v = L^{*}\ \oplus\ L
\end{aligned}
$$

In Haskell:

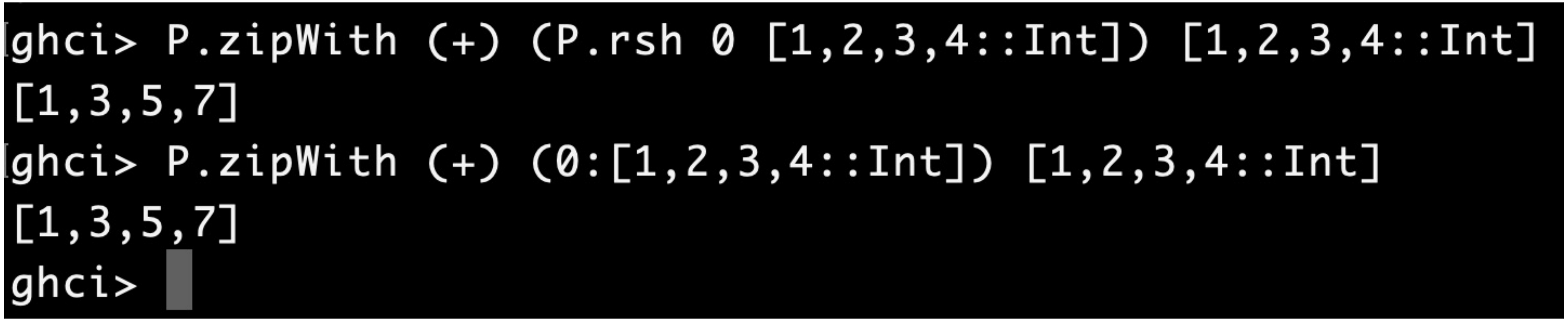


```haskell
import qualified Powerlist as P

sps :: Num a => (a -> a -> a) -> P.PowerList a -> P.PowerList a
sps _ [] = []
sps _ [x] = [x]
sps op l = P.zip (sps op u) (sps op v)
  where (u, v) = P.unzip $ P.zipWith op (P.rsh 0 l) l
```

Parallelizing SPS

- Algorithm divides the input into 2 halves, calls recursively
- Parallelize the “unzip” operation to deconstruct the list
- Parallelize “zipWith” by breaking input into chunks
- P.zipWith op (P.rsh 0 t) t can be rewritten as P.zipWith op (0:t) t (since zipWith only considers intersection of 2 lists)



```
[ghci> P.zipWith (+) (P.rsh 0 [1,2,3,4::Int]) [1,2,3,4::Int]
[1,3,5,7]
[ghci> P.zipWith (+) (0:[1,2,3,4::Int]) [1,2,3,4::Int]
[1,3,5,7]
ghci>
```

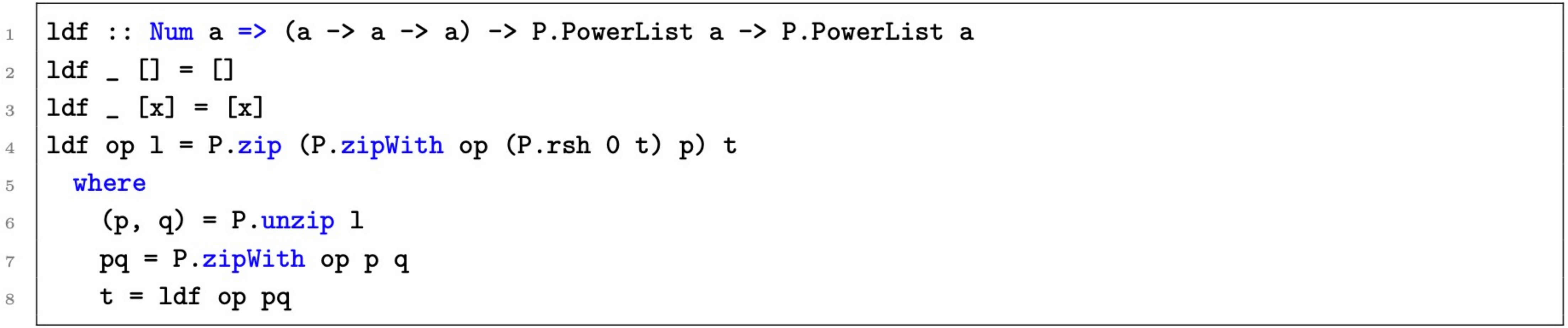
Parallelizing SPS

- Use Unboxed Vector implementation to reduce GC
- Introduce “shiftAdd” and “filterUsing” methods to directly execute certain operations over mutable vectors.

Ladner Fischer

$$
\begin{aligned}
&\mathit{ldf}\ \langle x\rangle = \langle x\rangle\\
&\mathit{ldf}(p \bowtie q) = (t^* \oplus p) \bowtie t\\
&\qquad \mathit{where}\ t = \mathit{ldf}(p \oplus q)
\end{aligned}
$$

In Haskell:

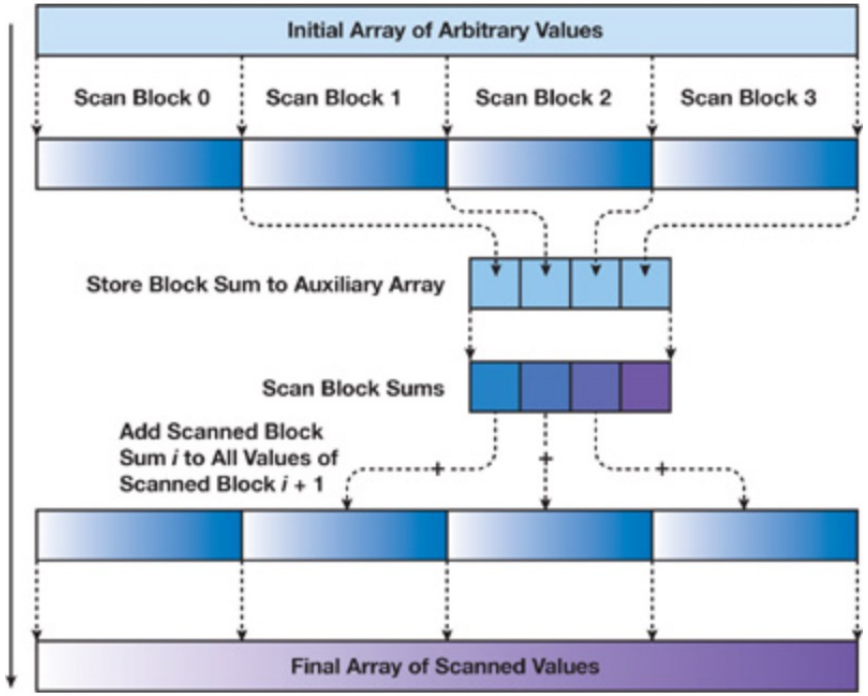


```haskell
ldf :: Num a => (a -> a -> a) -> P.PowerList a -> P.PowerList a
ldf _ [] = []
ldf _ [x] = [x]
ldf op l = P.zip (P.zipWith op (P.rsh 0 t) p) t
  where
    (p, q) = P.unzip l
    pq = P.zipWith op p q
    t = ldf op pq
```

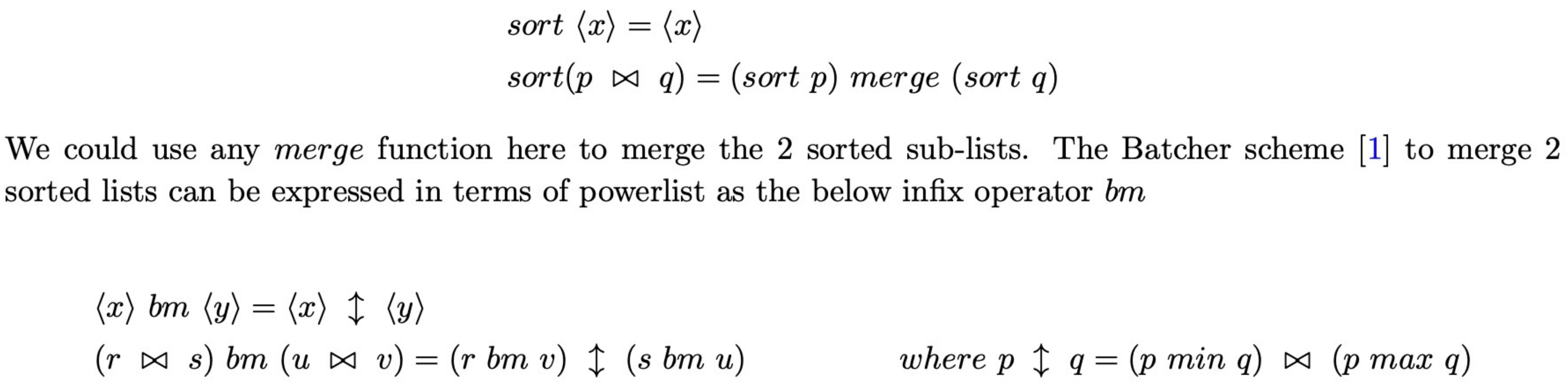
Similar techniques were used to parallelize LDF

LDFChunkUBVecPLPar

- A hybrid approach, where input is split into chunks first, then LDF is applied in parallel to all chunks.
- Bleloch style merge is used to combine the results


Initial Array of Arbitrary Values
Scan Block 0
Scan Block 1
Scan Block 2
Scan Block 3
Store Block Sum to Auxiliary Array
Scan Block Sums
Add Scanned Block Sum *i* to All Values of Scanned Block *i* + 1
+
+
+
Final Array of Scanned Values

Batcher merge sort

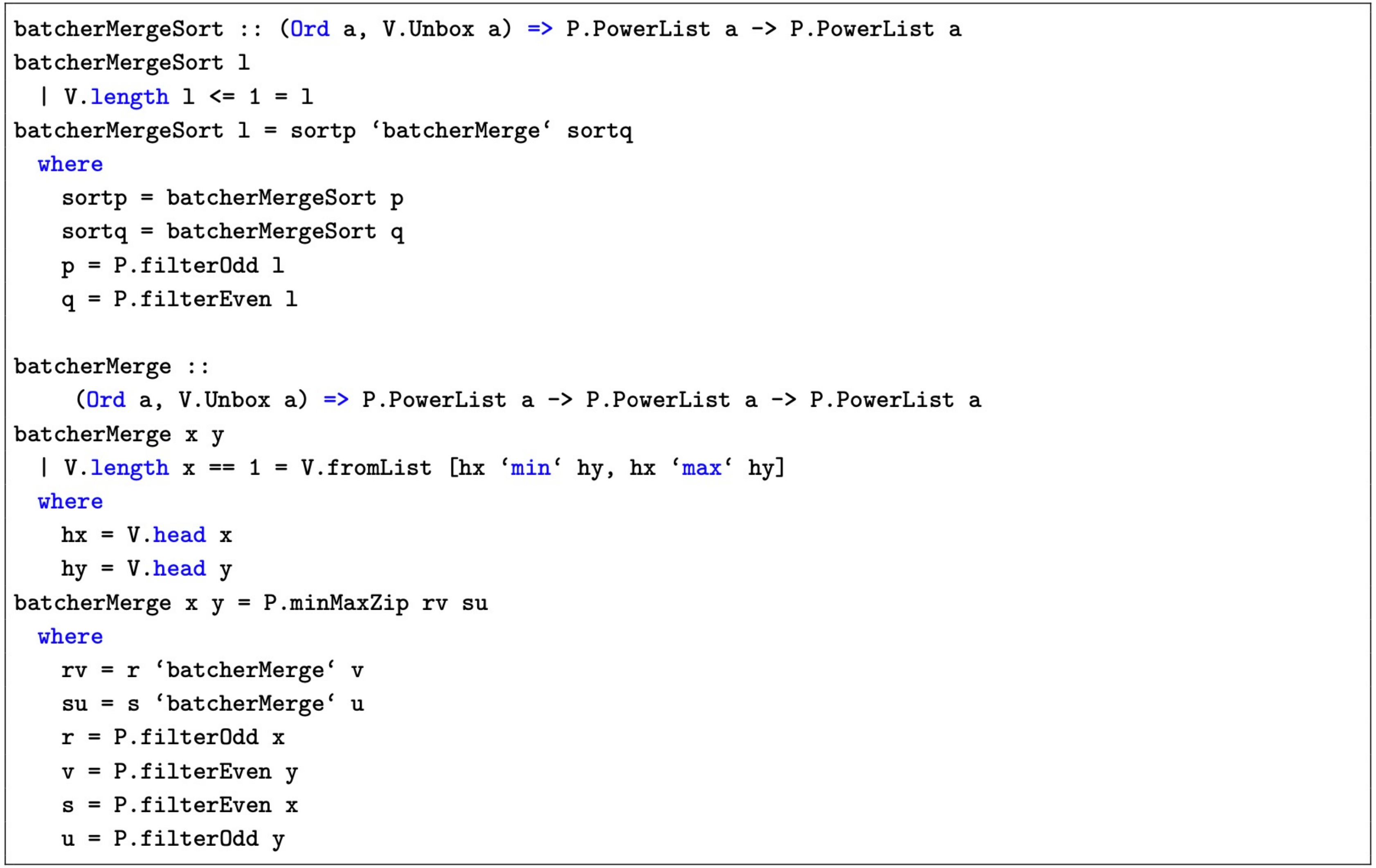


$$sort\ \langle x\rangle = \langle x\rangle$$
$$sort(p \bowtie q) = (sort\ p)\ merge\ (sort\ q)$$

We could use any *merge* function here to merge the 2 sorted sub-lists. The Batcher scheme [1] to merge 2 sorted lists can be expressed in terms of powerlist as the below infix operator *bm*

$$\langle x\rangle\ bm\ \langle y\rangle = \langle x\rangle \updownarrow \langle y\rangle$$
$$(r \bowtie s)\ bm\ (u \bowtie v) = (r\ bm\ v) \updownarrow (s\ bm\ u) \qquad where\ p \updownarrow q = (p\ min\ q) \bowtie (p\ max\ q)$$

Batcher merge sort

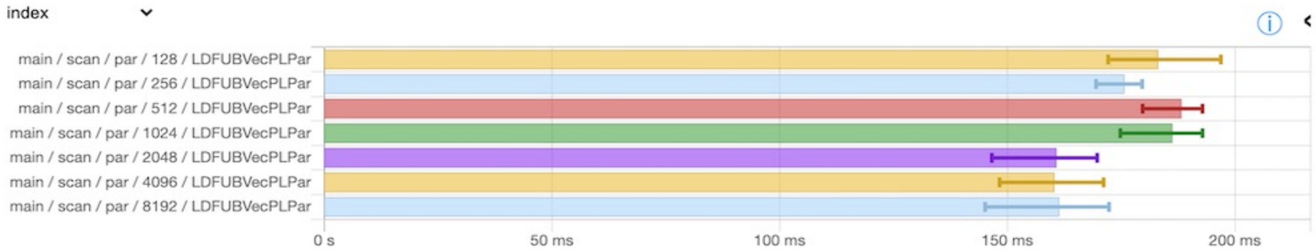


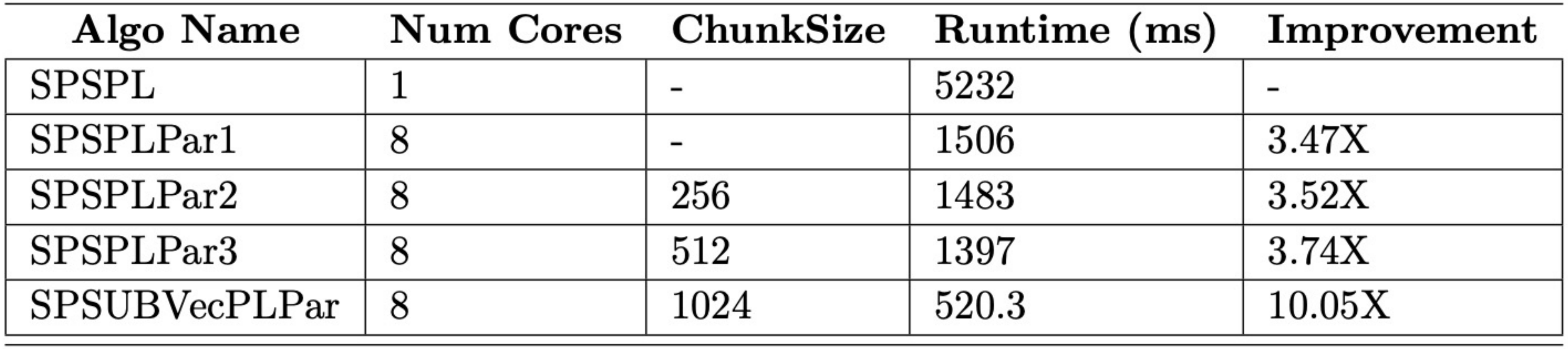
```haskell
batcherMergeSort :: (Ord a, V.Unbox a) => P.PowerList a -> P.PowerList a
batcherMergeSort l
  | V.length l <= 1 = l
batcherMergeSort l = sortp ‘batcherMerge‘ sortq
  where
    sortp = batcherMergeSort p
    sortq = batcherMergeSort q
    p = P.filterOdd l
    q = P.filterEven l

batcherMerge ::
     (Ord a, V.Unbox a) => P.PowerList a -> P.PowerList a -> P.PowerList a
batcherMerge x y
  | V.length x == 1 = V.fromList [hx ‘min‘ hy, hx ‘max‘ hy]
  where
    hx = V.head x
    hy = V.head y
batcherMerge x y = P.minMaxZip rv su
  where
    rv = r ‘batcherMerge‘ v
    su = s ‘batcherMerge‘ u
    r = P.filterOdd x
    v = P.filterEven y
    s = P.filterEven x
    u = P.filterOdd y
```

Results

- Run benchmarks on Intel 8 core Intel i9-9900K CPU @ 3.60 GHZ (32G memory) on Debian 11 (bullseye)
- Use criterion package to benchmark algorithms over arrays of length 2^20
- Different chunk sizes tried

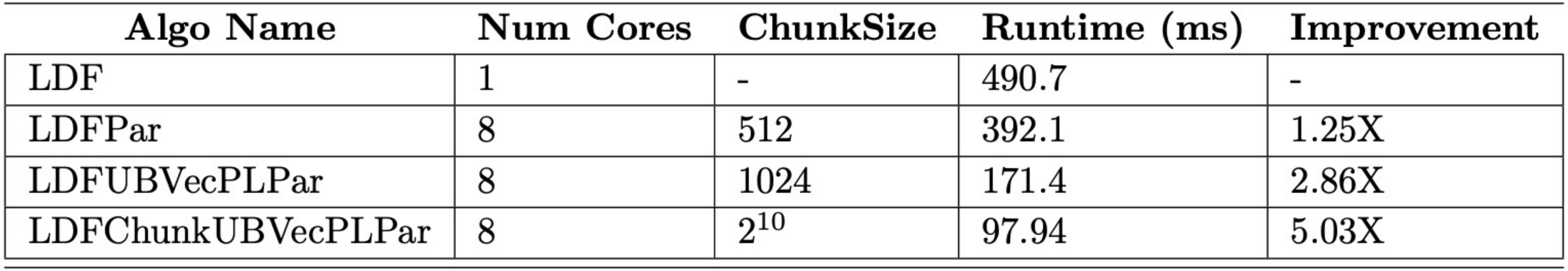




Algo Name	Num Cores	ChunkSize	Runtime (ms)	Improvement
SPSPL	1	-	5232	-
SPSPLPar1	8	-	1506	3.47X
SPSPLPar2	8	256	1483	3.52X
SPSPLPar3	8	512	1397	3.74X
SPSUBVecPLPar	8	1024	520.3	10.05X

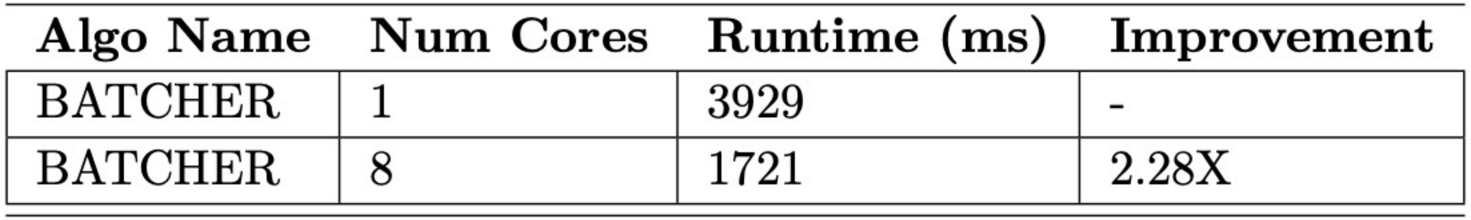
Scan results

Results



Algo Name	Num Cores	ChunkSize	Runtime (ms)	Improvement
LDF	1	-	490.7	-
LDFPar	8	512	392.1	1.25X
LDFUBVecPLPar	8	1024	171.4	2.86X
LDFChunkUBVecPLPar	8	2^{10}	97.94	5.03X

LDF scan



Algo Name	Num Cores	Runtime (ms)	Improvement
BATCHER	1	3929	-
BATCHER	8	1721	2.28X

Sort results

Project Materials

- Everything accessible at github: https://github.com/saucam/powerlist
- Extensive benchmarks: https://github.com/saucam/powerlist/blob/main/docs/Benchmark.md
- Project report: https://github.com/saucam/powerlist/blob/main/docs/project_report.pdf