Unidirectional Input/Output Streaming
Complexity of Reversal and Sorting

Nathanaël François  U. Paris Diderot
Rahul Jain       National U. Singapore
Frédéric Magniez  U. Paris Diderot
Multi-stream computation

Constraints
- Access to data stream
  - ReadWrite / ReadOnly / WriteOnly / BurnOnly (WriteOnce)
- Unidirectional heads

Complexity
- Memory space $s(n)$
- $p(n)$ unidirectional passes
- Expansion $\lambda: \# \text{ written cells} / n$
Basic problems

Problems
- \( Y = \text{Reverse}(X), \ Y = \text{Sort}(X) \)

Constraints
- No auxiliary streams
- Sublinear \( p(n) \) & \( s(n) \)
- Constant \( \lambda \)

Background
- Sorting is easy (\( p(n) \) & \( s(n) = O(\log n) \)) with
  - 3 ReadWrite streams and \( \lambda = 1 \)
  - 2 ReadWrite streams and \( \lambda = n \)
- Sorting primitives are very powerful for streaming algorithms [FOCS'04]
- Bidirectional streaming algorithms require exponentially less memory
  [STOC'11,FOCS'11,ICDT'12,STACS'13] (1 single ReadOnly stream)
  Example for checking well-parenthesized expressions
    - \( p \) unidirectional passes require memory \( \sqrt{n/p} \)
    - 2 bidirectional passes uses memory polylog \( n \)
Multi-stream problems

1970
- T. Kameda and R. Vollmar. Note on tape reversal complexity of languages. IC’70

1991

2004-09
- M. Grohe, A. Hernich, and N. Schweikardt. Lower bounds for processing data with few random accesses to external memory. JACM’09
- P. Beame, T. S. Jayram, and A. Rudra. Lower bounds for randomized read/write stream algorithms. STOC’07
- P. Beame and T. Huynh. The value of multiple read/write streams for approximating frequency moments. FOCS’08

2012
- C. Konrad and F. Magniez. Validating XML documents in the streaming model with external memory. ICDT’12
Multi-stream problems

1970
- T. Kameda and R. Vollmar. Note on tape reversal complexity of languages. IC’70

1991

2004-09
- M. Grohe, A. Hernich, and N. Schweikardt. Lower bounds for processing data with few random accesses to external memory. JACM’09
- P. Beame, T. S. Jayram, and A. Rudra. Lower bounds for randomized read/write stream algorithms. STOC’07
- P. Beame and T. Huynh. The value of multiple read/write streams for approximating frequency moments. FOCS’08

2012
- C. Konrad and F. Magniez. Validating XML documents in the streaming model with external memory. ICDT’12

All lower bounds were for \( o(\log n) \)-pass algorithms
Communication problem

- Input: Agathe gets X1 and Juliette X2
- Output: Agathe learns X2 and Juliette X1
- Fact: $\Omega(n)$ bits of communication are needed
Simple case

**Simplification**
- Heads are always synchronized

**Reduction**
- Agathe generates stream $X_1$ and Juliette stream $X_2$
- Agathe/Juliette simulates algorithm while reading $X_1/X_2$
- At each pass, Agathe and Juliette exchange $2s(n)$ bits
  Communication protocol with $2p(n)s(n)$ exchanged bits
- At the end, Agathe learns $X_2$ and Juliette $X_1$
  Therefore $n$ bits must be exchanged: $p(n)s(n) = \Omega(n)$
**Simple case**

**Simplification**
- Heads are always synchronized

**Reduction**
- Agathe generates stream $X_1$ and Juliette stream $X_2$
- Agathe/Juliette simulates algorithm while reading $X_1/X_2$
- At each pass, Agathe and Juliette exchange $2s(n)$ bits
  Communication protocol with $2p(n)s(n)$ exchanged bits
- At the end, Agathe learns $X_2$ and Juliette $X_1$
  Therefore $n$ bits must be exchanged: $p(n)s(n) = \Omega(n)$
**Simplification**

- Heads are always synchronized

**Reduction**

- Agathe generates stream $X_1$ and Juliette stream $X_2$
- Agathe/Juliette simulates algorithm while reading $X_1/X_2$
- At each pass, Agathe and Juliette exchange $2s(n)$ bits
  
  Communication protocol with $2p(n)s(n)$ exchanged bits
- At the end, Agathe learns $X_2$ and Juliette $X_1$

Therefore $n$ bits must be exchanged: $p(n)s(n) = \Omega(n)$
**Simplification**

- Heads are always synchronized

**Reduction**

- Agathe generates stream $X_1$ and Juliette stream $X_2$
- Agathe/Juliette simulates algorithm while reading $X_1/X_2$
- At each pass, Agathe and Juliette exchange $2s(n)$ bits
  - Communication protocol with $2p(n)s(n)$ exchanged bits
- At the end, Agathe learns $X_2$ and Juliette $X_1$
  - Therefore $n$ bits must be exchanged: $p(n)s(n) = \Omega(n)$
Another simple case

**Weaker simplification**
- Heads always *synchronize* at the *same* position $(L, n-L)$

![Diagram showing synchronization between Heads]

**Reduction**
- Similar to previous one
- At each pass, Agathe and Juliette exchange $s(n)$ bits
  - Communication protocol with $2p(n)s(n)$ exchanged bits
- Same conclusion: $p(n)s(n) = \Omega(n)$

**Remark**
- Tradeoff remains valid for ReadWrite streams
Another simple case

**Weaker simplification**

- Heads always **synchronize** at the **same** position \((L,n-L)\)

![Diagram showing synchronization](image)

**Reduction**

- Similar to previous one
- At each pass, Agathe and Juliette exchange **s(n)** bits
  
  Communication protocol with \(2p(n)s(n)\) exchanged bits
- Same conclusion: \(p(n)s(n) = \Omega(n)\)

**Remark**

- Tradeoff remains valid for ReadWrite streams
Another simple case

Weaker simplification

- Heads always synchronize at the same position \((L, n-L)\)

Reduction

- Similar to previous one
- At each pass, Agathe and Juliette exchange \(s(n)\) bits
  
  Communication protocol with \(2p(n)s(n)\) exchanged bits
- Same conclusion: \(p(n)s(n) = \Omega(n)\)

Remark

- Tradeoff remains valid for ReadWrite streams
Our results (general case)

<table>
<thead>
<tr>
<th>Input Output</th>
<th>ReadOnly</th>
<th>ReadWrite</th>
</tr>
</thead>
<tbody>
<tr>
<td>WriteOnly</td>
<td>ps=θ(n)</td>
<td>p^2s=θ(n)</td>
</tr>
<tr>
<td>ReadWrite</td>
<td>p^2s=θ(n)</td>
<td>p,s=θ(log n)</td>
</tr>
</tbody>
</table>

- Deterministic algorithms with expansion 1
  but lower bounds also hold for randomized ones with any expansion
- ReadOnly/WriteOnly
  lower bound only proved in the BurnOnly model

Remarks

- Previous simplification is too strong
- No hope to prove tight lower bounds using communication complexity
Our results (general case)

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>ReadOnly</th>
<th>ReadWrite</th>
</tr>
</thead>
<tbody>
<tr>
<td>WriteOnly</td>
<td>$ps = \theta(n)$</td>
<td>$p^2s = \theta(n)$</td>
</tr>
<tr>
<td>ReadWrite</td>
<td>$p^2s = \theta(n)$</td>
<td>$p, s = \theta(\log n)$</td>
</tr>
</tbody>
</table>

- Deterministic algorithms with expansion 1
  but lower bounds also hold for randomized ones with any expansion
- ReadOnly/WriteOnly
  lower bound only proved in the BurnOnly model

Remarks
- Previous simplification is too strong
- No hope to prove tight lower bounds using communication complexity
RO-RW algorithm with $\sqrt{n}$ passes and log $n$ memory

Stage 1: RO-WO
- Partition the input in blocks of size $\sqrt{n}$
- In each pass
  copy one block in place (but with same order in each block)

Stage 2: ⊥-RW
- Reverse the order inside each block (and move elts to the previous block)
  one symbol per block at each pass
RO-RW algorithm with $\sqrt{n}$ passes and $\log n$ memory

**Stage 1: RO-WO**
- Partition the input in blocks of size $\sqrt{n}$
- In each pass
  - copy one block in place (but with same order in each block)

```
X1  X2  X3  ...  X\sqrt{n}
```

```
X\sqrt{n}  ...  X3  X2  X1
```

**Stage 2: ⊥-RW**
- Reverse the order inside each block (and move els to the previous block)
  - one symbol per block at each pass

```
X\sqrt{n}  ...  X3  X2  X1
```
RO-RW algorithm with $\sqrt{n}$ passes and log $n$ memory

Stage 1: RO-WO
- Partition the input in blocks of size $\sqrt{n}$
- In each pass
  copy one block in place (but with same order in each block)

Stage 2: $\bot$-RW
- Reverse the order inside each block (and move elts to the previous block)
  one symbol per block at each pass
Stage 1: RO-WO
- Partition the input in blocks of size $\sqrt{n}$
- In each pass
  - copy one block in place (but with same order in each block)

Stage 2: RW
- Reverse the order inside each block (and move elts to the previous block)
  - one symbol per block at each pass
RO-RW algorithm with $\sqrt{n}$ passes and $\log n$ memory

**Stage 1: RO-WO**
- Partition the input in blocks of size $\sqrt{n}$
- In each pass copy one block in place (but with same order in each block)

**Stage 2: ⊥-RW**
- Reverse the order inside each block (and move elts to the previous block)
  one symbol per block at each pass
RO-RW algorithm with $\sqrt{n}$ passes and log $n$ memory

**Stage 1: RO-WO**
- Partition the input in blocks of size $\sqrt{n}$
- In each pass
  
  copy one block in place (but with same order in each block)

**Stage 2: ⊥-RW**
- Reverse the order inside each block (and move els to the previous block)
  one symbol per block at each pass
**Stage 1: RO-WO**

- Partition the input in blocks of size $\sqrt{n}$
- In each pass
  
  copy one block in place (but with same order in each block)

**Stage 2: $\perp$-RW**

- Reverse the order inside each block (and move elts to the previous block)
  
  one symbol per block at each pass
RO-RW algorithm with $\sqrt{n}$ passes and log $n$ memory

**Stage 1: RO-WO**

- Partition the input in blocks of size $\sqrt{n}$
- In each pass
  
  copy one block in place (but with same order in each block)

**Stage 2: ⊥-RW**

- Reverse the order inside each block (and move els to the previous block)
  
  one symbol per block at each pass
Tradeoff for RO-RW algorithms

Notations
- Reverse head direction on $Y$
- Reverse becomes Copy ($Y = X$)

Measure of progress after pass $t$
- $X$: uniform random $n$-bit string
  \[ W^t = I(X[1]:Y^t[1]) + I(X[2]:Y^t[2]) + \ldots + I(X[n]:Y^t[n]) \]
- Before first pass: $W^0 = 0$
- After last pass
  \[ W^p = n, \text{ when zero error} \]
  \[ W^p \geq (1 - H_2(\varepsilon)) n, \text{ when bounded error } \varepsilon \quad (\text{Fano}) \]

Conclusion
- Need to bound \[ \Delta^t[i] = I(X[i]:Y^t[i]) - I(X[i]:Y^{t-1}[i]) \]
Tradeoff for RO-RW algorithms

Notations
- Reverse head direction on $Y$
- Reverse becomes Copy ($Y=X$)

Measure of progress after pass $t$
- $X$: uniform random $n$-bit string
  $$W^t = I(X[1]:Y^t[1]) + I(X[2]:Y^t[2]) + \ldots + I(X[n]:Y^t[n])$$
- Before first pass: $W^0 = 0$
- After last pass
  $$W^p = n, \text{ when zero error}$$
  $$W^p \geq (1-H_2(\varepsilon)) n, \text{ when bounded error } \varepsilon \quad (\text{Fano})$$

Conclusion
- Need to bound $\Delta^t[i] = I(X[i]:Y^t[i]) - I(X[i]:Y^{t-1}[i])$
Notations

- Reverse head direction on $Y$
- Reverse becomes Copy ($Y=X$)

Measure of progress after pass $t$

- $X$: uniform random $n$-bit string
  $$W^t = I(X[1]:Y^t[1]) + I(X[2]:Y^t[2]) + \ldots + I(X[n]:Y^t[n])$$
- Before first pass: $W^0 = 0$
- After last pass
  $$W^p = n, \text{ when zero error}$$
  $$W^p \geq (1-H_2(\epsilon)) n, \text{ when bounded error } \epsilon \quad (\text{Fano})$$

Conclusion

- Need to bound $\Delta^t[i] = I(X[i]:Y^t[i]) - I(X[i]:Y^{t-1}[i])$
Tradeoff for RO-RW algorithms

**Notations**
- Reverse head direction on Y
- Reverse becomes Copy (Y=X)

**Measure of progress after pass t**
- X: uniform random n-bit string
  \[ W^t = I(X[1]:Y_t^t[1]) + I(X[2]:Y_t^t[2]) + \ldots + I(X[n]:Y_t^t[n]) \]
- Before first pass: \( W^0 = 0 \)
- After last pass
  \[ W^p = n, \text{ when zero error} \]
  \[ W^p \geq (1-H_2(\varepsilon)) n, \text{ when bounded error } \varepsilon \quad \text{(Fano)} \]

**Conclusion**
- Need to bound \( \Delta^i[t] = I(X[i]:Y_t^t[i]) - I(X[i]:Y_{t-1}[i]) \)
Bounding progress $\Delta^t[i]$

**Meeting position**
- $L^t$ = position where heads meet at pass $t$

**Analysis**
- Case $i = L^t$ (heads meet)
  $$I(X[i]:Y^t[i] \mid L^t=i) - I(X[i]:Y^{t-1}[i] \mid L^t=i) \leq 1$$
- Case $i < L^t$ (heads have already met)
  $$Y^t[i] = f(Y^{t-1}[i], M^{t,i}, X[L^t,n])$$
  where $M^{t,i}$: memory when output head enters cell $i$
  $X[L^t,n]$: input head may read many cells at positions $\geq L^t$
  while output head remains on cell $i$
- Case $i > L^t$ (heads have not yet met)
  $$Y^t[i] = f(Y^{t-1}[i], M^{t,i}, X[1,L^t])$$
Bounding progress $\Delta^t[i]$

Meeting position
- $L^t = $ position where heads meet at pass $t$

![Diagram of meeting position]

Analysis
- Case $i = L^t$ (heads meet)
  \[ I(X[i]:Y^t[i] \mid L^t=i) - I(X[i]:Y^{t-1}[i] \mid L^t=i) \leq 1 \]
- Case $i < L^t$ (heads have already met)
  \[ Y^t[i] = f(Y^{t-1}[i], M^{t,i}, X[L^t,n]) \]
  where $M^{t,i}$: memory when output head enters cell $i$
  $X[L^t,n]$: input head may read many cells at positions $\geq L^t$
  while output head remains on cell $i$
- Case $i > L^t$ (heads have not yet met)
  \[ Y^t[i] = f(Y^{t-1}[i], M^{t,i}, X[1,L^t]) \]
Bounding progress $\Delta^t[i]$

**Meeting position**
- $L^t = \text{position where heads meet at pass } t$

**Analysis**
- Case $i=L^t$ (heads meet)
  \[ I(X[i]:Y^t[i] \mid L^t=i) - I(X[i]:Y^{t-1}[i] \mid L^t=i) \leq 1 \]
- Case $i<L^t$ (heads have already met)
  \[ Y^t[i] = f(Y^{t-1}[i], M^{t,i}, X[L^t,n]) \]
  where $M^{t,i}$: memory when output head enters cell $i$
  $X[L^t,n]$: input head may read many cells at positions $\geq L^t$
  while output head remains on cell $i$
- Case $i>L^t$ (heads have not yet met)
  \[ Y^t[i] = f(Y^{t-1}[i], M^{t,i}, X[1,L^t]) \]
Bounding progress $\Delta^t[i]$  

Meeting position  
- $L^t$ = position where heads meet at pass $t$

Analysis
- Case $i=L^t$ (heads meet)
  
  \[ I(X[i]:Y^t[i] \mid L^t=i) - I(X[i]:Y^{t-1}[i] \mid L^t=i) \leq 1 \]

- Case $i<L^t$ (heads have already met)
  
  \[ Y^t[i] = f(Y^{t-1}[i], M^{t,i}, X[L^t,n]) \]
  
  where $M^{t,i}$: memory when output head enters cell $i$
  
  $X[L^t,n]$: input head may read many cells at positions $\geq L^t$
  while output head remains on cell $i$

- Case $i>L^t$ (heads have not yet met)
  
  \[ Y^t[i] = f(Y^{t-1}[i], M^{t,i}, X[1,L^t]) \]
Bounding progress $\Delta^t[i]$

Meeting position

- $L^t = \text{position where heads meet at pass } t$

Analysis

- Case $i = L^t$ (heads meet)
  
  \[ I(X[i]:Y^t[i] \mid L^t = i) - I(X[i]:Y^{t-1}[i] \mid L^t = i) \leq 1 \]

- Case $i < L^t$ (heads have already met)
  
  \[ Y^t[i] = f(Y^{t-1}[i], M^{t,i}, X[L^t,n]) \]

  where $M^{t,i}$: memory when output head enters cell $i$

  $X[L^t,n]$: input head may read many cells at positions $\geq L^t$ while output head remains on cell $i$

- Case $i > L^t$ (heads have not yet met)
  
  \[ Y^t[i] = f(Y^{t-1}[i], M^{t,i}, X[1,L^t]) \]
Bounding progress $\Delta^t[i]$

Meeting position
- $L^t = \text{position where heads meet at pass } t$

Analysis
- Case $i = L^t$
  \[ I(X[i]:Y^t[i] \mid L^t=i) - I(X[i]:Y^{t-1}[i] \mid L^t=i) \leq 1 \]
- Case $i \neq L^t$
  \[ Y^t[i] = f(Y^{t-1}[i], M^{t,i}, X[\neq i]) \]
  \[ M^{t,i}: \text{at most } s \text{ bits} \quad X[\neq i]: \text{fixed from start} \quad (\text{ReadOnly}) \]
  \[ I(X[i]: f(Y^{t-1}[i], M^{t,i}, X[\neq i]) \mid L^t=i, X[\neq i]) \]
  \[ \leq I(X[i]:Y^{t-1}[i], M^{t,i}, X[\neq i] \mid L^t=i, X[\neq i]) \quad (\text{Data processing}) \]
  \[ \leq I(X[i]:Y^{t-1}[i] \mid L^t=i, X[\neq i]) + s \quad (\text{Sub-additivity}) \]
Bounding progress $\Delta^t[i]$

### Meeting position
- $L^t = \text{position where heads meet at pass } t$

![Diagram showing meeting position]

### Analysis
- Modified measure
  \[
  \Delta^t[i] = I(X[i]:Y^t[i] \mid L^t=i,X[\neq i]) - I(X[i]:Y^{t-1}[i] \mid L^t=i,X[\neq i])
  \]
- Case $i=L^t$
  \[
  \Delta^t[i] \leq 1
  \]
- Case $i \neq L^t$
  \[
  \Delta^t[i] \leq s \quad \text{but } s \geq 1 \ldots
  \]

### Idea
- Group cells in k blocks of size n/k
Bounding progress $\Delta^t[i]$

Meeting position
- $L^t = \text{position where heads meet at pass } t$

Analysis
- Modified measure
  \[ \Delta^t[i] = I(X_i:Y_i^t \mid L^t=i,X_{\neq i}) - I(X_i:Y_{i-1}^t \mid L^t=i,X_{\neq i}) \]
  - Case $i=L^t$: $\Delta^t[i] \leq n/k$
  - Case $i \neq L^t$: $\Delta^t[i] \leq s$
- Total at each pass: $ks+n/k$
- Initial and final conditions: $W^0=0$ and $W^p \approx n$

Conclusion
- $p(ks+n/k) \geq n$
  Therefore $s \geq n/p^2$
**Analysis at pass t**

- Before heads meet
  
  Only memory $M^1$ at the beginning of the pass is useful

- After heads meet
  
  Only memory $M^L$ at the meeting point is useful

- Conclusion: Only 2s correct bits can be written at each pass

**Lemma**

- Let $Z \in \{0, 1, \bot\}^n$ be the string written on $Y$ at pass $t$
  
  Then $\sum_i I(X_i : Z_i L) \leq 2(s + \log n)$
Tradeoff for RO-WO algorithms

Obstacle
- Y cannot be read, but could possibly be overwritten

Restriction
- **BurnOnly** model: avoid overwrites

Let \( q_i = \Pr(Z_i \neq \bot) \) and \( \varepsilon_i = \Pr(Z_i \neq X_i, \bot) \) at pass \( t \)

Then \( I(X_i : Z_i L) \geq q_i \left(1 - H_2\left(\varepsilon_i/q_i\right)\right)\)

Conclusion
- At pass \( t \)
  \[ 2(s + \log n) \geq \sum_i q_i \left(1 - H_2\left(\varepsilon_i/q_i\right)\right) \]
- Summing over all passes
  \( Y[i] \) has been written with probability \( \sum_t q_i(t) \geq 1 - \varepsilon \)
  Overall error on \( Y[i] \) with probability \( \sum_t \varepsilon_i(t) \leq \varepsilon \)
  Concavity of entropy leads to
  \[ 2p(s + \log n) \geq n(1 - \varepsilon) \left(1 - H_2\left(\varepsilon/(1-\varepsilon)\right)\right) \]
**Reverse**

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>ReadOnly</th>
<th>ReadWrite</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WriteOnly</strong></td>
<td>$ps=\theta(n)$</td>
<td>$p^2s=\theta(n)$</td>
</tr>
<tr>
<td><strong>ReadWrite</strong></td>
<td>$p^2s=\theta(n)$</td>
<td>$p,s=\theta(\log n)$</td>
</tr>
</tbody>
</table>

- Deterministic algorithms with expansion 1
  but lower bounds also hold for randomized ones with any expansion
- ReadOnly/WriteOnly
  lower bound only proved in the BurnOnly model

**Sort**

<table>
<thead>
<tr>
<th>Expansion</th>
<th>Deterministic</th>
<th>Randomized</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Expansion</strong></td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

- $p(n) \& s(n) = O(\log n)$
MergeSort with 3 streams

Iteration k
- Hypothesis: \(X\) is partitioned in \(n/k\) sorted blocs of size \(k\)
- 2 passes
  1st pass: Duplicate even blocks of \(X\) on 2nd stream
  2nd pass: Merge 2nd stream with odd blocks of \(X\) on 3rd stream

With only 2 streams
- Issue: Merging (2nd pass) requires a 3rd stream
- Idea:
  Label elements with their positions in the merged block
  Move them in 2 extra passes
  Require expansion \(\log n\)
MergeSort with 3 streams

**Iteration k**

- Hypothesis: $X$ is partitioned in $n/k$ sorted blocs of size $k$

  ![Partitioned Blocks](image)

- 2 passes
  - 1st pass: Duplicate even blocks of $X$ on 2nd stream
    
    ![Duplicate Even Blocks](image)

  - 2nd pass: Merge 2nd stream with odd blocks of $X$ on 3rd stream
    
    ![Merge Odd Blocks](image)

**With only 2 streams**

- Issue: Merging (2nd pass) requires a 3rd stream
- Idea:
  - Label elements with their positions in the merged block
  - Move them in 2 extra passes
  - Require expansion $\log n$
QuickSort with 2 streams

Iteration k
- Hypothesis: $X$ is partitioned in $n/k$ \textit{consecutive} blocs of size $k$

| XI | X2 | X3 | X4 | ... |

- 2 passes
  1st pass
  Take a random pivot $P_i$ for each block $X_i$
  Copy elements smaller than $P_i$ to 2nd stream
  reserving place for other elements

| XI | X2 | X3 | X4 | ... |

2nd pass: Copy other elements

Analysis
- Constant expansion
- In expectation, log $n$ iterations