Energy-Efficient Algorithms

Erik Demaine, Jayson Lynch,
Geronimo Mirano, Nirvan Tyagi
MIT CSAIL

Why energy-efficient? Cheaper, Greener, Faster, Longer

- Cheaper and Greener
- Longer battery life
- Faster processors

Computation represents 5% of worldwide energy use, growing 4-10% annually compared with 3% growth in total energy use [Heddeghem 2014]

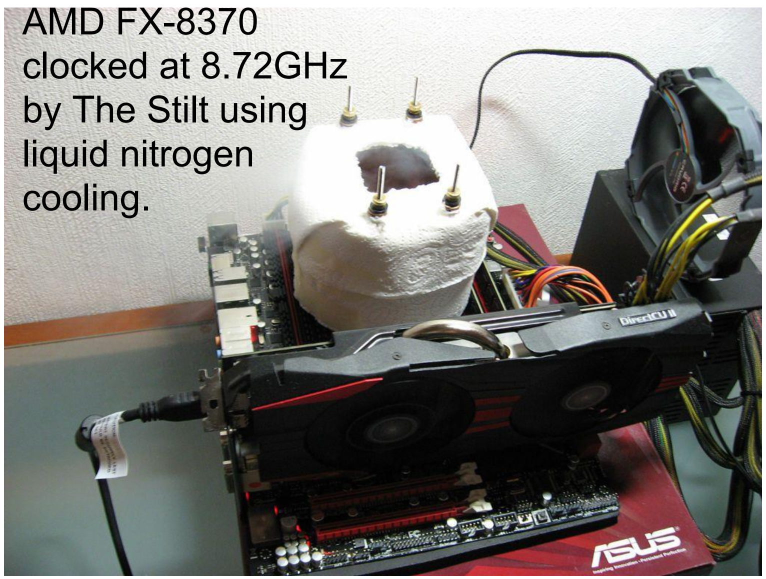
Why energy-efficient? Cheaper, Greener, Faster, Longer

- Cheaper and Greener
- Longer battery life
- Faster processors

Computation represents 5% of worldwide energy use, growing 4-10% annually compared with 3% growth in total energy use [Heddeghem 2014]

Why energy-efficient? Cheaper, Greener, Faster, Longer

- Cheaper and Greener
- Longer battery life
- Faster processors

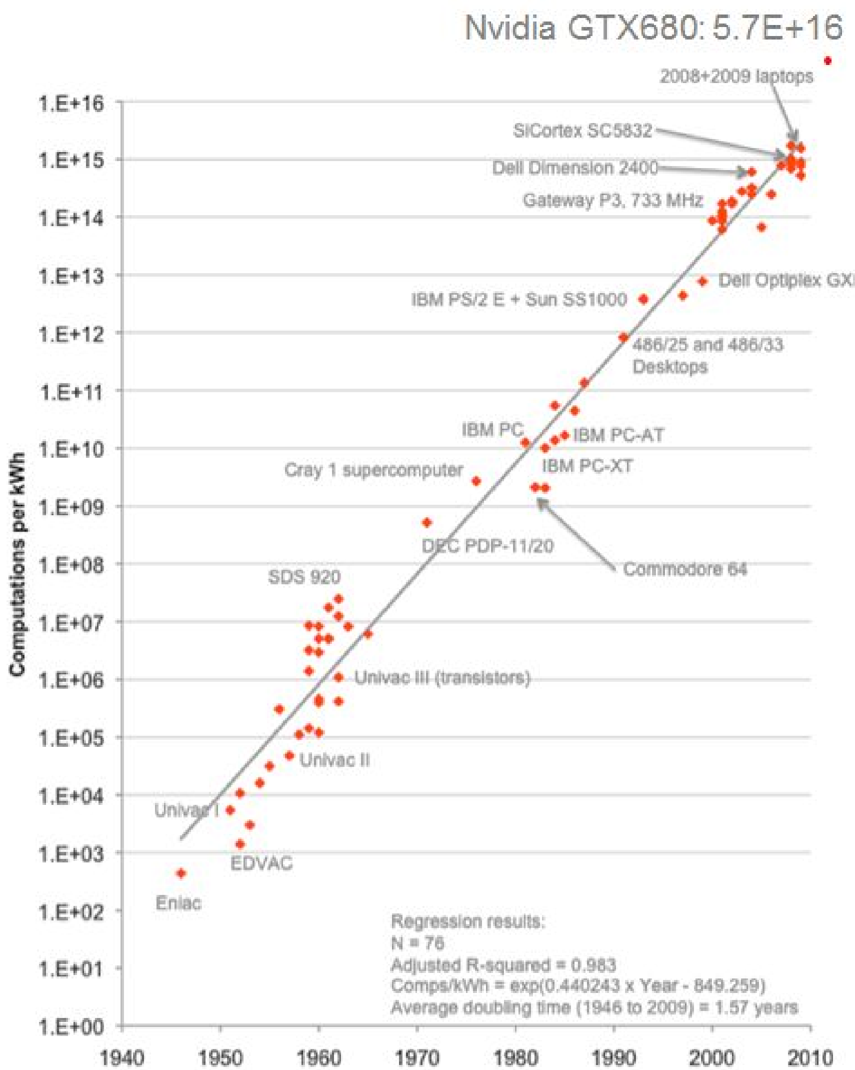


AMD FX-8370 clocked at 8.72GHz by The Stilt using liquid nitrogen cooling.

Computation represents 5% of worldwide energy use, growing 4-10% annually compared with 3% growth in total energy use [Heddeghem 2014]

Koomey’s Law

- Energy efficiency of computation increases exponentially
- Computations per kWh doubles every 1.57 years.



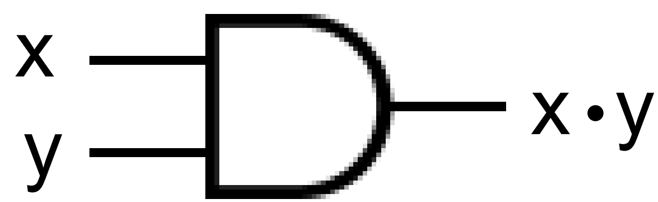
[Koomey, Berard, Sanchez, Wong ‘09]

Landauer's Principle
[Landauer '61]

- Erasing bits has a **minimum energy cost**
- 1 bit = *k T ln 2* Joules
  - k is Boltzman's constant
  - T is the temperature
- 1 bit = 7.6*10^-28 kWh at room temperature
- Experimental support [BAPCDL '12]

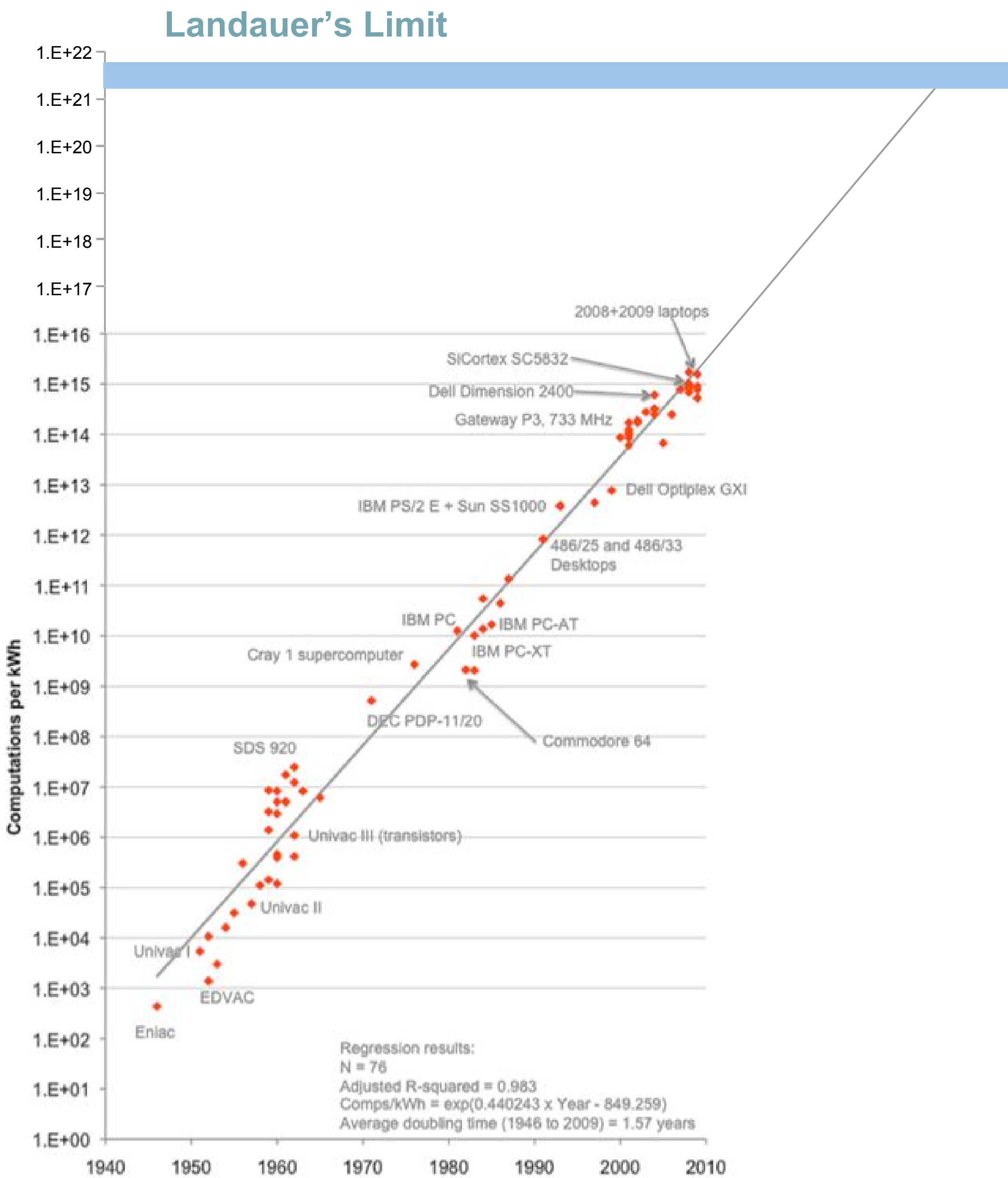
$$x = 0$$

$$x \cdot= y$$



Landauer’s Limit

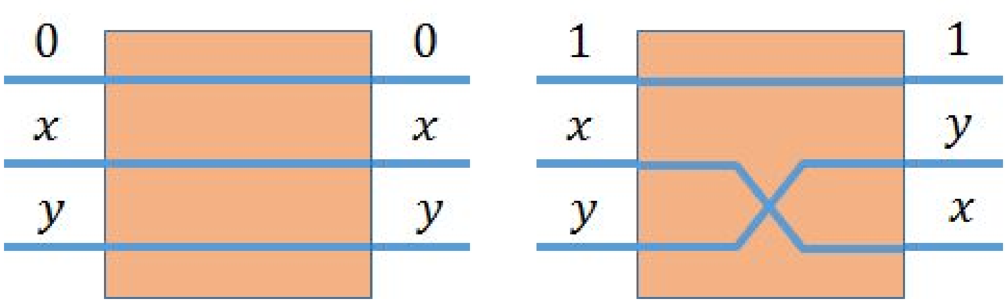
- Koomey’s Law: energy efficiency of computation doubles every 1.57 years
- Landauer’s Principle:
  - 1 bit = 7.6*10^-28 kWh
- ≈ Five orders of magnitude away [Center for Energy Efficient Electronic Science]
- At this rate we will hit a ‘ceiling’ in a few decades.



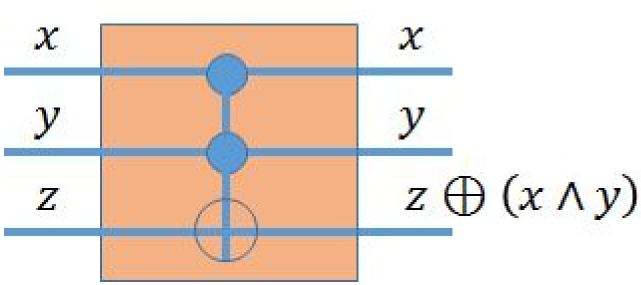
[Koomey, Berard, Sanchez, Wong ‘09]

Reversible Computing

- Circumvents Landauer’s Limit - no information destroyed
- Requires that all gates/functions are bijective
- Reversible computing is still universal (given extra ‘garbage’ space) [Lecerf ‘63, Bennett ‘73, FT ‘82]
    - Only a constant number of ancilla bits needed for circuits [AGS ‘15]



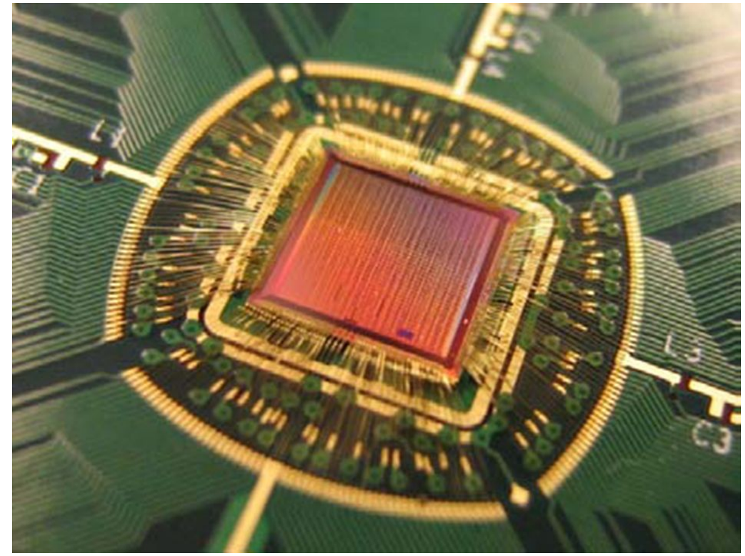
Fredkin Gate



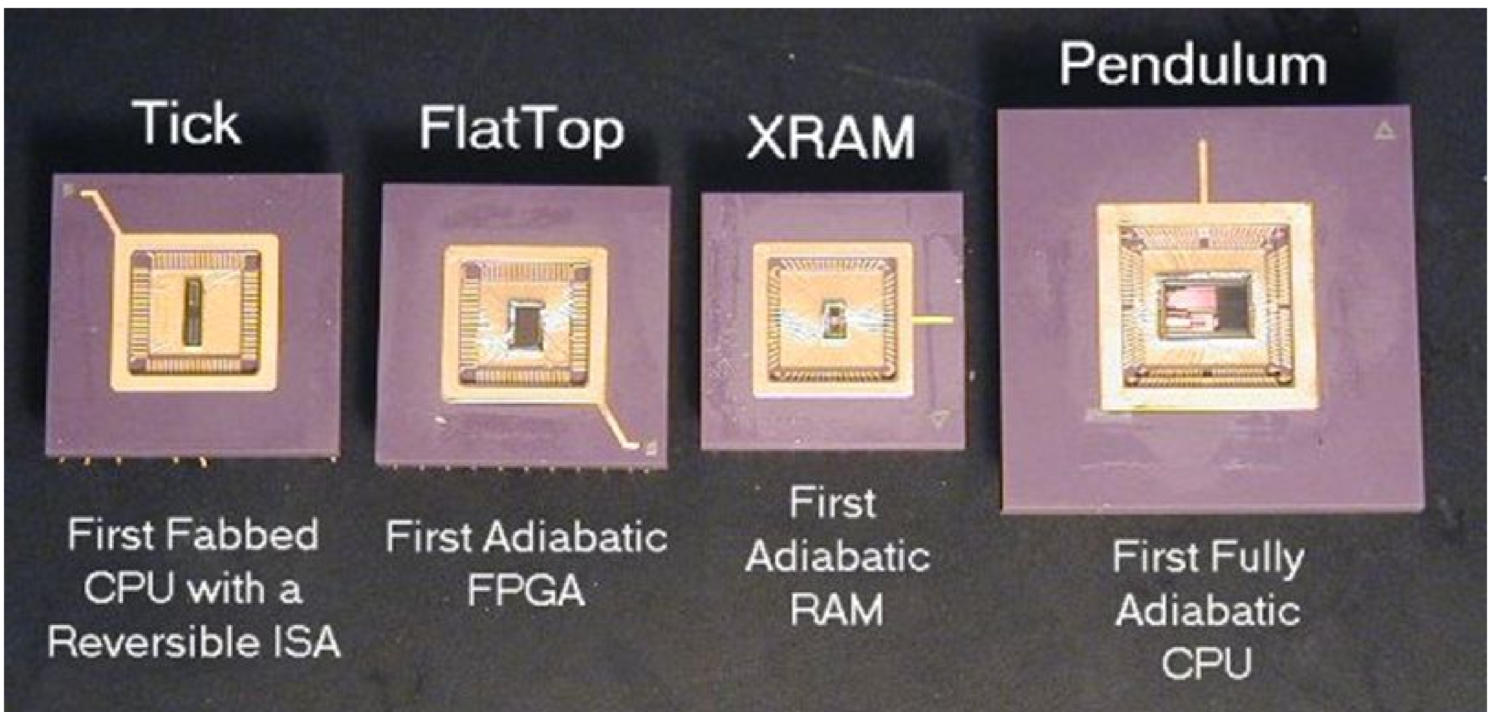
Toffoli Gate

Building Reversible Computers

- Split Level Charge Recovery Logic
- Resonant Circuits
- Nanomagnetic Circuits
- Superconducting Circuits



Cyclos Semiconductor ‘12



MIT ‘99

Reversible Computing

- Circumvents Landauer's Limit - no information destroyed
- Requires that all gates/functions are bijective
- Reversible computing is still universal [Lecerf '63, Bennett '73, FT '82]
    - Only a constant number of ancilla bits needed for circuits [AGS '15]

- Existing general results for simulating all algorithms reversibly require significantly more computational resources
    - Quadratic space [Bennett '79] or
    - Exponential time [Bennett '89] or
    - Trade-off between those extremes [Williams '00][BTV '01]

Landauer Energy Cost
[this paper]

- Establish RAM model of computation
- Charge one unit of energy whenever a bit is destroyed.
  - Li and Vitany also pose information-energy model [LV ‘92]
- Some operations are cheap (reversible), others are expensive
  - Cost of a function is: $$\lg \frac{|\text{input space}|}{|\text{output space}|}$$
- Examples:

x += y	*x >> 1*	*x = 0*
Energy Cost: 0	Energy Cost: 1	Energy Cost: *w*

Semi-Reversible Computing
[this paper]

- Analyze the energy complexity $E(n)$ of algorithms
  - $0 \le E(n) \le wT(n)$
- Create new (semi-)reversible algorithms to minimize the energy cost without large time/space overhead
- Understand time/space/energy tradeoff

Algorithms
[this paper]

Algorithm	Time	Space (words)	Energy (bits)
Sorting Algorithms			
Comparison Sort	$\Theta(n \lg n)$	$\Theta(n)$	$\Theta(n \lg n)$
Reversible Comparison Sort	$\Theta(n \lg n)$	$\Theta(n)$	0
Reversible Insertion Sort	$\Theta(n^2)$	$\Theta(n)$	0
Counting Sort	$\Theta(n+k)$	$\Theta(n+k)$	$\Theta(n+k)$
Reversible Counting Sort	$\Theta(n+k)$	$\Theta(n+k)$	0
Graph Algorithms			
Breadth-first Search	$\Theta(V+E)$	$\Theta(V+E)$	$\Theta(wV+E)$
Reversible BFS [Fra99]	$\Theta(V+E)$	$\Theta(V+E)$	0
Bellman-Ford	$\Theta(VE)$	$\Theta(V)$	$\Theta(VEw)$
Reversible Bellman-Ford	$\Theta(VE)$	$\Theta(VE)$	0
Floyd-Warshall	$\Theta(V^3)$	$\Theta(V^2)$	$\Theta(V^3 w)$
Reversible Floyd-Warshall [Fra99]	$\Theta(V^3)$	$\Theta(V^3)$	0
Matrix APSP	$\Theta(V^3 \lg V)$	$\Theta(V^2)$	$\Theta(wV^3 \lg V)$
Reversible Matrix APSP [Fra99]	$\Theta(V^3 \lg V)$	$\Theta(V^2 \lg V)$	0
Semi-reversible Matrix APSP	$\Theta(V^3 \lg V)$	$\Theta(V^2)$	$wV^2 \lg V$

Data Structures

[this paper]

Algorithm	Time	Space (words)	Energy (bits)
Data Structures			
Standard AVL Trees (build)	$O(n \lg n)$	$O(n)$	$O(w \cdot n \lg n)$
(search)	$O(\lg n)$	$O(1)$	$O(\lg n)$
(insert)	$O(\lg n)$	$O(1)$	$O(w \lg n)$
(k deletes)	$O(k \lg n)$	$O(1)$	$O(w \lg n)$
Reversible AVL Trees (build)	$O(n \lg n)$	$O(n)$	0
(search)	$O(\lg n)$	$O(1)$	0
(insert)	$O(\lg n)$	$O(1)$	0
(k deletes)	$O(k \lg n)$	$O(k)$	0
Standard Binary Heap (insert)	$O(\lg n)$	$O(1)$	$O(\lg n)$
(delete max)	$O(\lg n)$	$O(\lg n)$	$O(w \lg n)$
Reversible Binary Heap (insert)	$O(\lg n)$	$O(1)$	0
(delete max)	$O(\lg n)$	$O(\lg n)$	0
Dynamic Array (build)	$O(n)$	$O(n)$	0
(query)	$O(1)$	$O(1)$	0
(add)	$O(1)$	$O(1)$	0
(delete)	$O(1)$	$O(1)$	0

Basic Building Blocks
[this paper]

- Languages and compiler for semi-reversible computing [DLT '16]
- Costs and energy efficient versions for many computer primitives
- Protected vs. General

Primitive	Time (ops)	Space in Log (bits)	Energy (bits)
Control Logic			
Paired Jump	$\Theta(1)$	1	0
Variable Jump	$\Theta(1)$	$1 + w$	0
Protected If	$\Theta(1)$	0	0
General If	$\Theta(1)$	1	0
Simple For loop	$\Theta(l)$	0	0
Protected For loop	$\Theta(l)$	0	0
General For loop	$\Theta(l)$	$\lg l$	0
Function call	$\Theta(1)$	0	0
Memory Management			
Free lists	$\Theta(N)$	$\Theta(wN)$	0
Reference Counting	$\Theta(N)$	$\Theta(wN)$	0
Mark & Sweep	$\Theta(N)$	$\Theta(wN)$	0

General `if` example:

```
if (a > 2) {
    a -= 4;
}
```

Basic Building Blocks
[this paper]

- Languages and compiler for semi-reversible computing [DLT '16]
- Costs and energy efficient versions for many computer primitives
- Protected vs. General

Primitive	Time (ops)	Space in Log (bits)	Energy (bits)
Control Logic			
Paired Jump	$\Theta(1)$	1	0
Variable Jump	$\Theta(1)$	$1 + w$	0
Protected If	$\Theta(1)$	0	0
General If	$\Theta(1)$	1	0
Simple For loop	$\Theta(l)$	0	0
Protected For loop	$\Theta(l)$	0	0
General For loop	$\Theta(l)$	$\lg l$	0
Function call	$\Theta(1)$	0	0
Memory Management			
Free lists	$\Theta(N)$	$\Theta(wN)$	0
Reference Counting	$\Theta(N)$	$\Theta(wN)$	0
Mark & Sweep	$\Theta(N)$	$\Theta(wN)$	0

Protected `if`:

```
if (condition) {
    … condition not
            modified …
} else {
    … condition not
            modified …
}
```

Basic Building Blocks
[this paper]

- Languages and compiler for semi-reversible computing [DLT '16]
- Costs and energy efficient versions for many computer primitives
- Protected vs. General

Primitive	Time (ops)	Space in Log (bits)	Energy (bits)
Control Logic			
Paired Jump	$\Theta(1)$	1	0
Variable Jump	$\Theta(1)$	$1 + w$	0
Protected If	$\Theta(1)$	0	0
General If	$\Theta(1)$	1	0
Simple For loop	$\Theta(l)$	0	0
Protected For loop	$\Theta(l)$	0	0
General For loop	$\Theta(l)$	$\lg l$	0
Function call	$\Theta(1)$	0	0
Memory Management			
Free lists	$\Theta(N)$	$\Theta(wN)$	0
Reference Counting	$\Theta(N)$	$\Theta(wN)$	0
Mark & Sweep	$\Theta(N)$	$\Theta(wN)$	0

Protected `if` example:

```
if (a > 2) {
    b -= 4;
}
```

Basic Building Blocks
[this paper]

- Languages and compiler for semi-reversible computing [DLT '16]
- Costs and energy efficient versions for many computer primitives
- Protected vs. General

Primitive	Time (ops)	Space in Log (bits)	Energy (bits)
Control Logic			
Paired Jump	$\Theta(1)$	1	0
Variable Jump	$\Theta(1)$	$1 + w$	0
Protected If	$\Theta(1)$	0	0
General If	$\Theta(1)$	1	0
Simple For loop	$\Theta(l)$	0	0
Protected For loop	$\Theta(l)$	0	0
General For loop	$\Theta(l)$	$\lg l$	0
Function call	$\Theta(1)$	0	0
Memory Management			
Free lists	$\Theta(N)$	$\Theta(wN)$	0
Reference Counting	$\Theta(N)$	$\Theta(wN)$	0
Mark & Sweep	$\Theta(N)$	$\Theta(wN)$	0

Protected `for`:

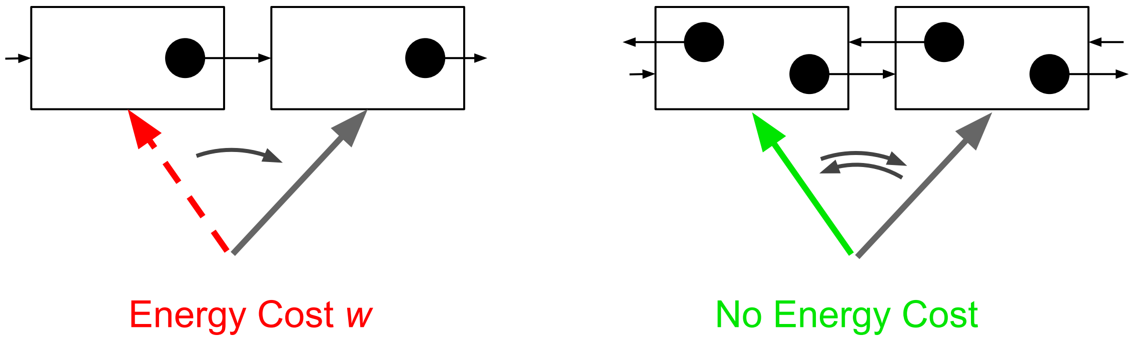
```
for (init; cond;
reversible update) {
   … cond not
            affected …
}
```

Algorithmic Techniques for Semi-Reversibility

- Pointer Swapping

Irreversible:

```
p = p.next;
```



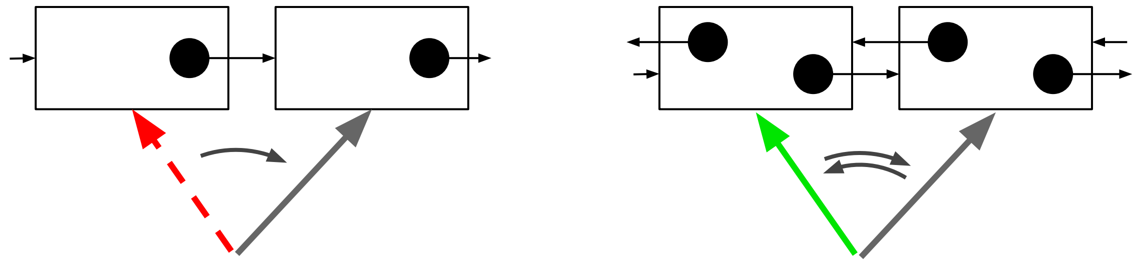
- Logging
  - energy cost → space cost
- Copy-out trick, unrolling and reverse-subroutines

Algorithmic Techniques for Semi-Reversibility

- Pointer Swapping

Reversible, Doubly-linked:

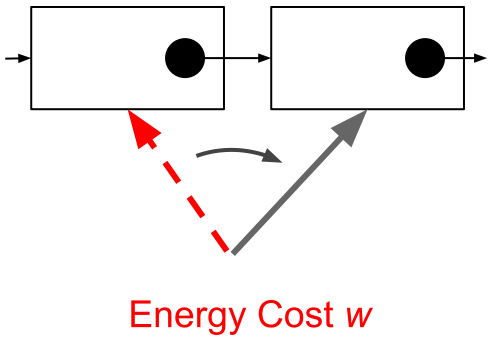
```
q += p // q was 0
p -= q
p += q.next // p was 0
q -= p.prev
```

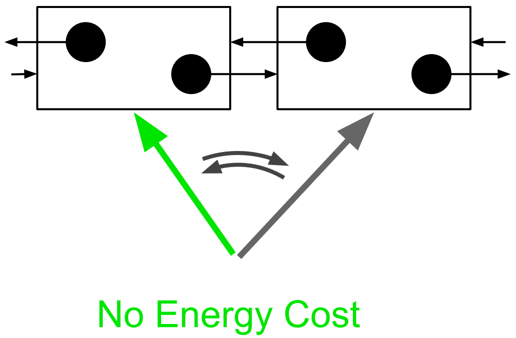


- Logging
  - energy cost → space cost
- Copy-out trick, unrolling and reverse-subroutines

Algorithmic Techniques for Semi-Reversibility

- Pointer Swapping





Reversible, Doubly-linked:

```
q += p // q was 0
p -= q
p += q.next // p was 0
q -= p.prev
```

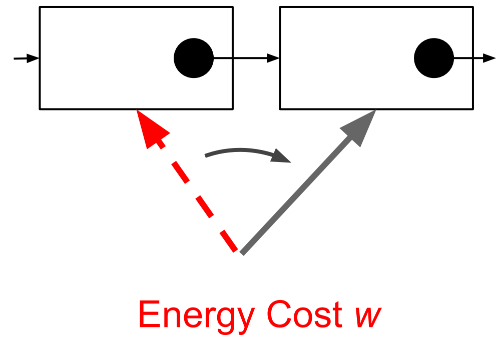
- Logging
    - energy cost → space cost
- Copy-out trick, unrolling and reverse-subroutines

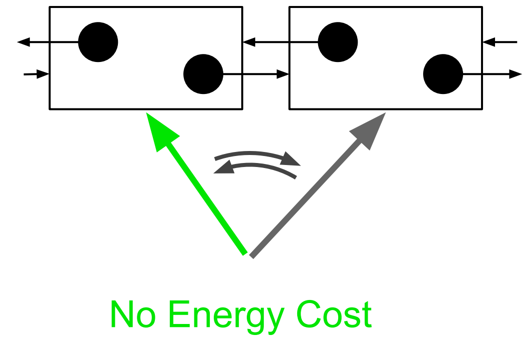
Algorithmic Techniques for Semi-Reversibility

- Pointer Swapping

Reversible, Doubly-linked:

```
q += p // q was 0
p -= q
p += q.next // p was 0
q -= p.prev
```





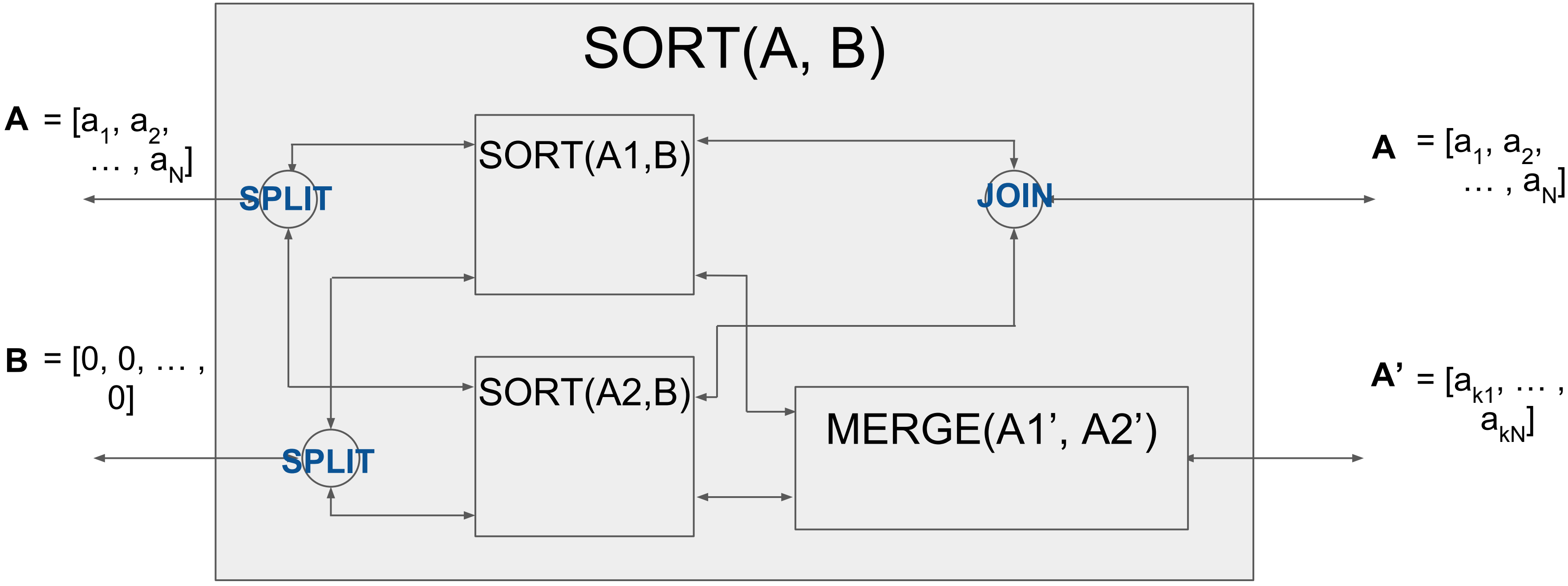
- Logging
  - energy cost → space cost
- Copy-out trick, unrolling and reverse-subroutines

Sorting Algorithms
[this paper]

Algorithm	Time	Space (words)	Energy (bits)
Sorting Algorithms			
Comparison Sort	$\Theta(n \lg n)$	$\Theta(n)$	$\Theta(n \lg n)$
Reversible Comparison Sort	$\Theta(n \lg n)$	$\Theta(n)$	0
Reversible Insertion Sort	$\Theta(n^2)$	$\Theta(n)$	0
Counting Sort	$\Theta(n + k)$	$\Theta(n + k)$	$\Theta(n + k)$
Reversible Counting Sort	$\Theta(n + k)$	$\Theta(n + k)$	0

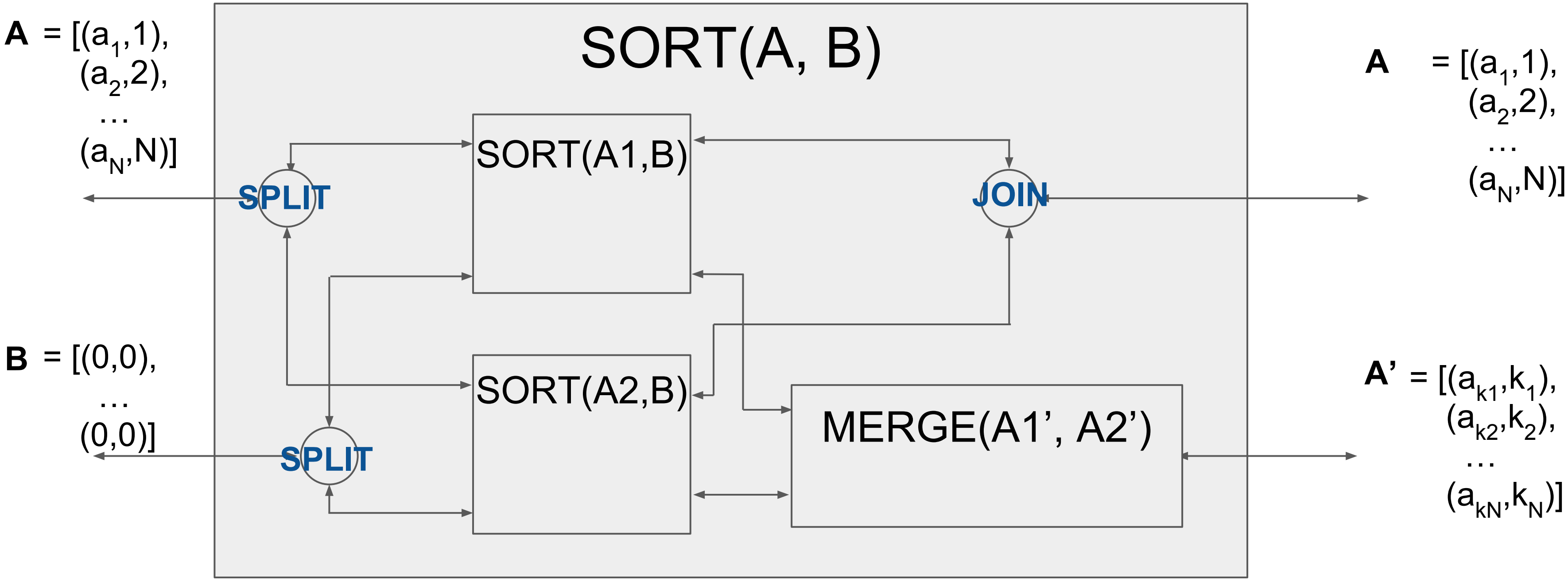
Reversible Merge Sort
[this paper]

- Preserve a copy of the input; if not preserving input, would necessarily pay *Ω (n lg n)* energy.
- Attains theoretical irreversible lower bound, *O(n lg n)* time + *O(n)* space



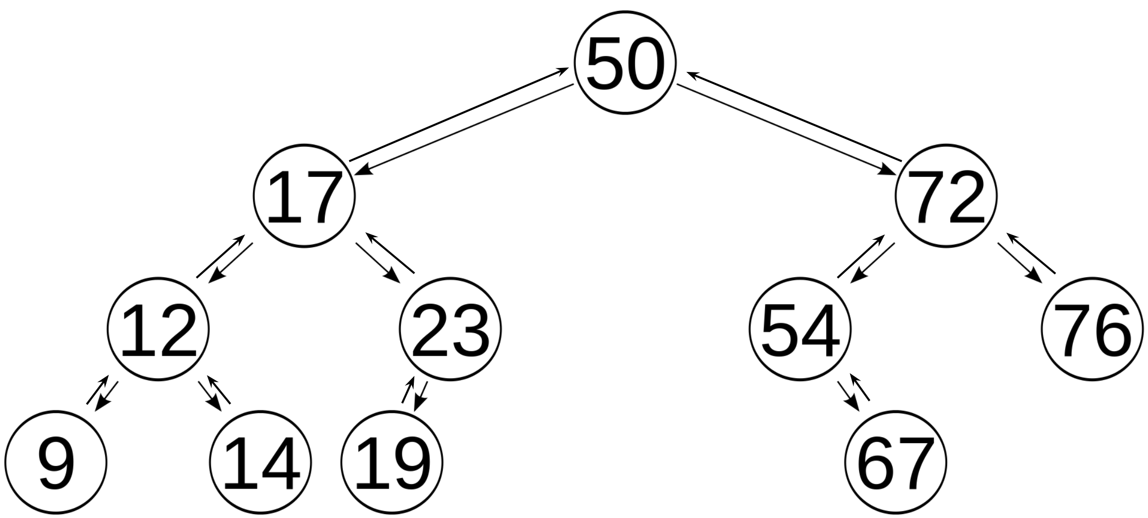
Reversible Merge Sort
[this paper]

- Preserve a copy of the input; if not preserving input, would necessarily pay *Ω (n lg n)* energy.
- Attains theoretical irreversible lower bound, *O(n lg n)* time + *O(n)* space



Data Structure Techniques for Semi-Reversibility

- In general, data structures will accumulate logging space with every operation
- Partially solved by periodic rebuilding



+

Log:

1. Rots: 010
2. Rots: 001
3. Rots: 0101
4. Rots: 1001
5. Rots: 1100

Data Structures!
[this paper]

Algorithm	Time	Space (words)	Energy (bits)
Data Structures			
Standard AVL Trees (build)	$O(n \lg n)$	$O(n)$	$O(w \cdot n \lg n)$
(search)	$O(\lg n)$	$O(1)$	$O(\lg n)$
(insert)	$O(\lg n)$	$O(1)$	$O(w \lg n)$
(k deletes)	$O(k \lg n)$	$O(1)$	$O(w \lg n)$
Reversible AVL Trees (build)	$O(n \lg n)$	$O(n)$	0
(search)	$O(\lg n)$	$O(1)$	0
(insert)	$O(\lg n)$	$O(1)$	0
(k deletes)	$O(k \lg n)$	$O(k)$	0
Standard Binary Heap (insert)	$O(\lg n)$	$O(1)$	$O(\lg n)$
(delete max)	$O(\lg n)$	$O(\lg n)$	$O(w \lg n)$
Reversible Binary Heap (insert)	$O(\lg n)$	$O(1)$	0
(delete max)	$O(\lg n)$	$O(\lg n)$	0
Dynamic Array (build)	$O(n)$	$O(n)$	0
(query)	$O(1)$	$O(1)$	0
(add)	$O(1)$	$O(1)$	0
(delete)	$O(1)$	$O(1)$	0

Graph Algorithms

Algorithm	Time	Space (words)	Energy (bits)
Graph Algorithms			
Breadth-first Search	$\Theta(V+E)$	$\Theta(V+E)$	$\Theta(wV+E)$
Reversible BFS [Fra99]	$\Theta(V+E)$	$\Theta(V+E)$	0
Bellman-Ford	$\Theta(VE)$	$\Theta(V)$	$\Theta(VEw)$
Reversible Bellman-Ford	$\Theta(VE)$	$\Theta(VE)$	0
Floyd-Warshall	$\Theta(V^3)$	$\Theta(V^2)$	$\Theta(wV^3)$
Reversible Floyd-Warshall [Fra99]	$\Theta(V^3)$	$\Theta(V^3)$	0
Matrix APSP	$\Theta(V^3 \lg V)$	$\Theta(V^2)$	$\Theta(wV^3 \lg V)$
Reversible Matrix APSP [Fra99]	$\Theta(V^3 \lg V)$	$\Theta(V^2 \lg V)$	0
Semi-reversible Matrix APSP	$\Theta(V^3 \lg V)$	$\Theta(V^2)$	$\Theta(wV^2 \lg V)$

All Pairs Shortest Path

- Floyd-Warshall Algorithm
  - Potentially deletes path lengths in adjacency matrix many times

```
FloydWarshall():
for k = 1 to n:
  for i = 1 to n:
    for j = 1 to n :
      path[i][j] = ...
       min(path[i][j]; path[i][k] + path[k][j])
```

Algorithm	Time	Space (words)	Energy (bits)
Graph Algorithms			
Breadth-first Search	$\Theta(V+E)$	$\Theta(V+E)$	$\Theta(wV+E)$
Reversible BFS [Fra99]	$\Theta(V+E)$	$\Theta(V+E)$	0
Bellman-Ford	$\Theta(VE)$	$\Theta(V)$	$\Theta(VEw)$
Reversible Bellman-Ford	$\Theta(VE)$	$\Theta(VE)$	0
Floyd-Warshall	$\Theta(V^3)$	$\Theta(V^2)$	$\Theta(wV^3)$
Reversible Floyd-Warshall [Fra99]	$\Theta(V^3)$	$\Theta(V^3)$	0
Matrix APSP	$\Theta(V^3 \lg V)$	$\Theta(V^2)$	$\Theta(wV^3 \lg V)$
Reversible Matrix APSP [Fra99]	$\Theta(V^3 \lg V)$	$\Theta(V^2 \lg V)$	0
Semi-reversible Matrix APSP	$\Theta(V^3 \lg V)$	$\Theta(V^2)$	$\Theta(wV^2 \lg V)$

All Pairs Shortest Path

- Reversible Floyd-Warshall [Frank '99]
  - Must recover the state of all the erased distances.
  - Can be seen immediately from full logging technique.

```
FloydWarshall():
for k = 1 to n:
    for i = 1 to n:
        for j = 1 to n :
            path[i][j] = ...
              min(path[i][j]; path[i][k] + path[k][j])
```

Algorithm	Time	Space (words)	Energy (bits)
Graph Algorithms			
Breadth-first Search	$\Theta(V+E)$	$\Theta(V+E)$	$\Theta(wV+E)$
Reversible BFS [Fra99]	$\Theta(V+E)$	$\Theta(V+E)$	0
Bellman-Ford	$\Theta(VE)$	$\Theta(V)$	$\Theta(VEw)$
Reversible Bellman-Ford	$\Theta(VE)$	$\Theta(VE)$	0
Floyd-Warshall	$\Theta(V^3)$	$\Theta(V^2)$	$\Theta(wV^3)$
Reversible Floyd-Warshall [Fra99]	$\Theta(V^3)$	$\Theta(V^3)$	0
Matrix APSP	$\Theta(V^3 \lg V)$	$\Theta(V^2)$	$\Theta(wV^3 \lg V)$
Reversible Matrix APSP [Fra99]	$\Theta(V^3 \lg V)$	$\Theta(V^2 \lg V)$	0
Semi-reversible Matrix APSP	$\Theta(V^3 \lg V)$	$\Theta(V^2)$	$\Theta(wV^2 \lg V)$

All Pairs Shortest Path

- (min, +) Matrix Multiplication
  - Still deleting many entries in the adjacency matrix
  - Algorithm runs *O(lg V)* matrix multiplications

APSPMM(W):
//Given adjacency matrix W
$W^{(1)} = W$
while m < n-1:
    $W^{(2m)} = W^{(m)} \oplus W^{(m)}$
    m = 2m
return $W^{(m)c}$

Algorithm	Time	Space (words)	Energy (bits)
Graph Algorithms			
Breadth-first Search	$\Theta(V+E)$	$\Theta(V+E)$	$\Theta(wV+E)$
Reversible BFS [Fra99]	$\Theta(V+E)$	$\Theta(V+E)$	0
Bellman-Ford	$\Theta(VE)$	$\Theta(V)$	$\Theta(VEw)$
Reversible Bellman-Ford	$\Theta(VE)$	$\Theta(VE)$	0
Floyd-Warshall	$\Theta(V^3)$	$\Theta(V^2)$	$\Theta(wV^3)$
Reversible Floyd-Warshall [Fra99]	$\Theta(V^3)$	$\Theta(V^3)$	0
Matrix APSP	$\Theta(V^3 \lg V)$	$\Theta(V^2)$	$\Theta(wV^3 \lg V)$
Reversible Matrix APSP [Fra99]	$\Theta(V^3 \lg V)$	$\Theta(V^2 \lg V)$	0
Semi-reversible Matrix APSP	$\Theta(V^3 \lg V)$	$\Theta(V^2)$	$\Theta(wV^2 \lg V)$

All Pairs Shortest Path

- Reversible (min, +) Matrix Multiplication [Leighton]
  - Save space by only storing each intermediate matrix.
  - Each new matrix can be recomputed from the prior two.

APSPMM(W):
//Given adjacency matrix W
$W^{(1)} = W$
while m < n-1:
    $W^{(2m)} = W^{(m)} \oplus W^{(m)}$
    m = 2m
return $W^{(m)c}$

Algorithm	Time	Space (words)	Energy (bits)
Graph Algorithms			
Breadth-first Search	$\Theta(V+E)$	$\Theta(V+E)$	$\Theta(wV+E)$
Reversible BFS [Fra99]	$\Theta(V+E)$	$\Theta(V+E)$	0
Bellman-Ford	$\Theta(VE)$	$\Theta(V)$	$\Theta(VEw)$
Reversible Bellman-Ford	$\Theta(VE)$	$\Theta(VE)$	0
Floyd-Warshall	$\Theta(V^3)$	$\Theta(V^2)$	$\Theta(wV^3)$
Reversible Floyd-Warshall [Fra99]	$\Theta(V^3)$	$\Theta(V^3)$	0
Matrix APSP	$\Theta(V^3 \lg V)$	$\Theta(V^2)$	$\Theta(wV^3 \lg V)$
Reversible Matrix APSP [Fra99]	$\Theta(V^3 \lg V)$	$\Theta(V^2 \lg V)$	0
Semi-reversible Matrix APSP	$\Theta(V^3 \lg V)$	$\Theta(V^2)$	$\Theta(wV^2 \lg V)$

All Pairs Shortest Path

[this paper]

- Reduced Energy (min, +) Matrix Multiplication
  - Each matrix element can be calculated reversibly. We now only erase $O(V^2)$ bits per matrix multiplication.

APSPMM(W):
//Given adjacency matrix W
$W^{(1)} = W$
while m < n-1:
  $W^{(2m)} = W^{(m)} \oplus W^{(m)}$
  m = 2m
return $W^{(m)c}$

Algorithm	Time	Space (words)	Energy (bits)
Graph Algorithms			
Breadth-first Search	$\Theta(V+E)$	$\Theta(V+E)$	$\Theta(wV+E)$
Reversible BFS [Fra99]	$\Theta(V+E)$	$\Theta(V+E)$	0
Bellman-Ford	$\Theta(VE)$	$\Theta(V)$	$\Theta(VEw)$
Reversible Bellman-Ford	$\Theta(VE)$	$\Theta(VE)$	0
Floyd-Warshall	$\Theta(V^3)$	$\Theta(V^2)$	$\Theta(wV^3)$
Reversible Floyd-Warshall [Fra99]	$\Theta(V^3)$	$\Theta(V^3)$	0
Matrix APSP	$\Theta(V^3 \lg V)$	$\Theta(V^2)$	$\Theta(wV^3 \lg V)$
Reversible Matrix APSP [Fra99]	$\Theta(V^3 \lg V)$	$\Theta(V^2 \lg V)$	0
Semi-reversible Matrix APSP	$\Theta(V^3 \lg V)$	$\Theta(V^2)$	$\Theta(wV^2 \lg V)$

All Pairs Shortest Path

- Non-trivial tradeoff between time, space, and energy in the APSP algorithms.

Algorithm	Time	Space (words)	Energy (bits)
Graph Algorithms			
Breadth-first Search	$\Theta(V+E)$	$\Theta(V+E)$	$\Theta(wV+E)$
Reversible BFS [Fra99]	$\Theta(V+E)$	$\Theta(V+E)$	0
Bellman-Ford	$\Theta(VE)$	$\Theta(V)$	$\Theta(VEw)$
Reversible Bellman-Ford	$\Theta(VE)$	$\Theta(VE)$	0
Floyd-Warshall	$\Theta(V^3)$	$\Theta(V^2)$	$\Theta(wV^3)$
Reversible Floyd-Warshall [Fra99]	$\Theta(V^3)$	$\Theta(V^3)$	0
Matrix APSP	$\Theta(V^3 \lg V)$	$\Theta(V^2)$	$\Theta(wV^3 \lg V)$
Reversible Matrix APSP [Fra99]	$\Theta(V^3 \lg V)$	$\Theta(V^2 \lg V)$	0
Semi-reversible Matrix APSP	$\Theta(V^3 \lg V)$	$\Theta(V^2)$	$\Theta(wV^2 \lg V)$

Open Problems - New Way of Analyzing Algorithms

Any algorithms you want!

- Shortest Path and APSP
- Machine Learning Algorithms
- Dynamic Programming
- Linear Programming
- vEB Trees
- Fibonacci Heaps
- FFT
- String Search
- Geometric Algorithms
- Cryptography

Open Problems - Model Extensions

- Streaming and Sub-Linear Algorithms
    - typically, space-heavy algorithms are easiest to make reversible; thus, these present a challenge.
- Succinct Data Structures
- Randomized algorithms
    - Motivation for minimizing randomness needed.
- Modeling memory and cache
- New hardware
- Lower bounds on time/space/energy complexity

Acknowledgments

Erik Demaine
Jayson Lynch
Geronimo Mirano
Nirvan Tyagi

Martin Demaine
Kevin Kelly
Maria L. Messick
Licheng Rao

CSAIL

MADALGO
CENTER FOR MASSIVE DATA ALGORITHMICS

MITei