CSE 633 Fall 2012

Polynomials and Coding

Jimmy Dobler

Definitions

- Zero Norm
  - For a polynomial p, $||p||_0$:= the number of non-zero coefficients in p
  - E.g., if $p = 1 + 3x^2 + 4x^7$, $||p||_0 = 3$.

Definitions

- Problem: Given a maximal degree n, find a polynomial p of degree less than or equal to n such that for all polynomials q of degree less than or equal to n, $||p * q||_0$ or $||q||_0$ is large.

Definitions

- Problem: Given a maximal degree n, find a polynomial p of degree less than or equal to n such that for all polynomials q of degree less than or equal to n, $||p * q||_0$ or $||q||_0$ is large.
- Large?
  - Let m = $\min_q(\max(||p * q||_0, ||q||_0))$.
  - Ideally, m = Θ(n).
  - However, m = $\Theta\left(\frac{n}{\log(n)}\right)$ or even m = $\Theta(n^{.95})$ would be acceptable.

Simplifications

- Restrict to coefficients of 0 and 1 only, and perform all math mod 2
  - XOR
- Can encode as binary strings
  - Multiplication becomes the XOR of binary shifts of the original polynomial

Example

- Let p = $1 + x + x^3 + x^4 + x^6 + x^7 + \cdots + x^{21} + x^{22}$
- In binary, p = 1101101101101101101101101
- Suppose q = $1 + x^3$ = 1001

Example

- Let p = $1 + x + x^3 + x^4 + x^6 + x^7 + \cdots + x^{21} + x^{22}$
- In binary, p = 11011011011011011011011
- Suppose q = $1 + x^3$ = 1001
- $p * q = 11011011011011011011011$
  $XOR \quad 11011011011011011011011$
  $= 11000000000000000000011$

Example

- Let p = $1 + x + x^3 + x^4 + x^6 + x^7 + \cdots + x^{21} + x^{22}$
- In binary, p = 11011011011011011011011
- Suppose q = $1 + x^3$ = 1001
- $p * q = 11011011011011011011011$
  $XOR \quad 11011011011011011011011$
  $= 11000000000000000000000011$
  - $||p * q||_0$ = 4… not very good

Why useful?

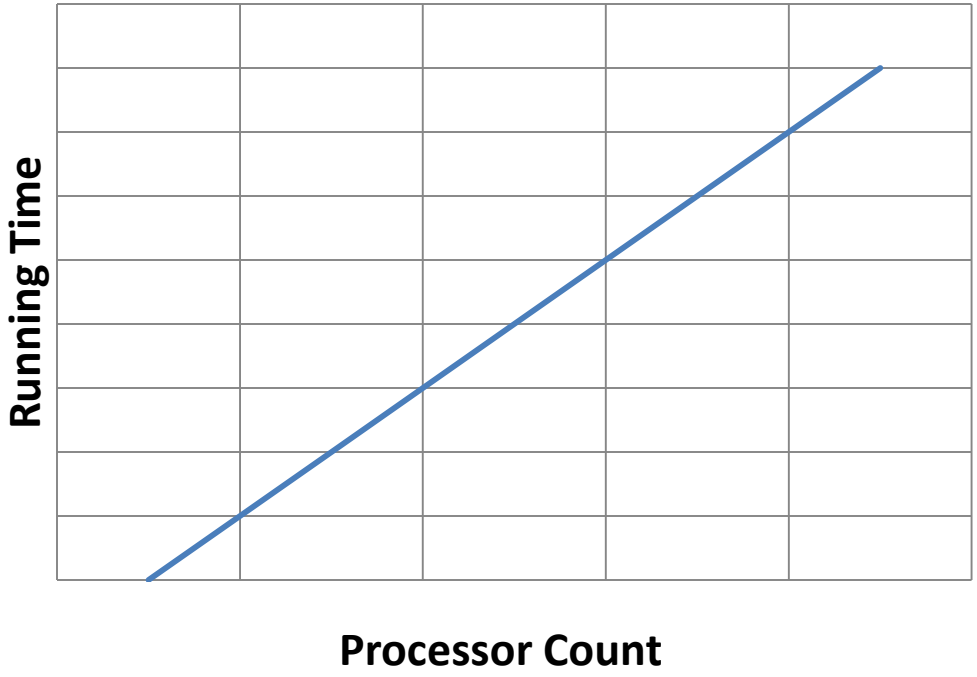
- Coding theory
  - Equivalent to having large distance between codewords
  - Allows for easy decoding, error correction
  - Existing methods all sacrifice something

Why do in parallel?

- Too many values of p to start trying randomly in serial
- Even given a promising p:
  - Number of possible values of q is proportional to 2^n
  - Takes too long to calculate

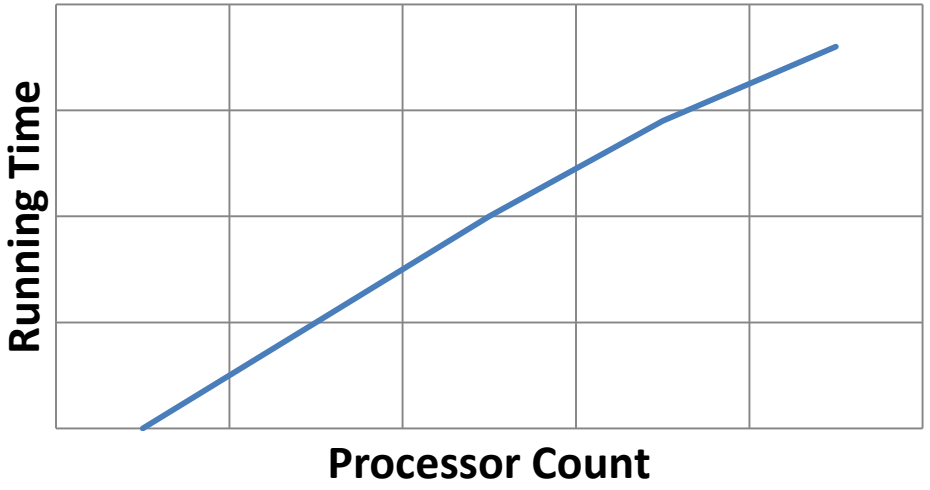
Implementation

- Method 1: Given n, check all possible p
- Almost 100% in parallel
- Using MPI, C/C++



Implementation

- Method 2: Given a promising p, find m for increasing values of n
- We need to calculate the shifts on each processor and spend some time merging data, but this takes very little time.



Algorithm

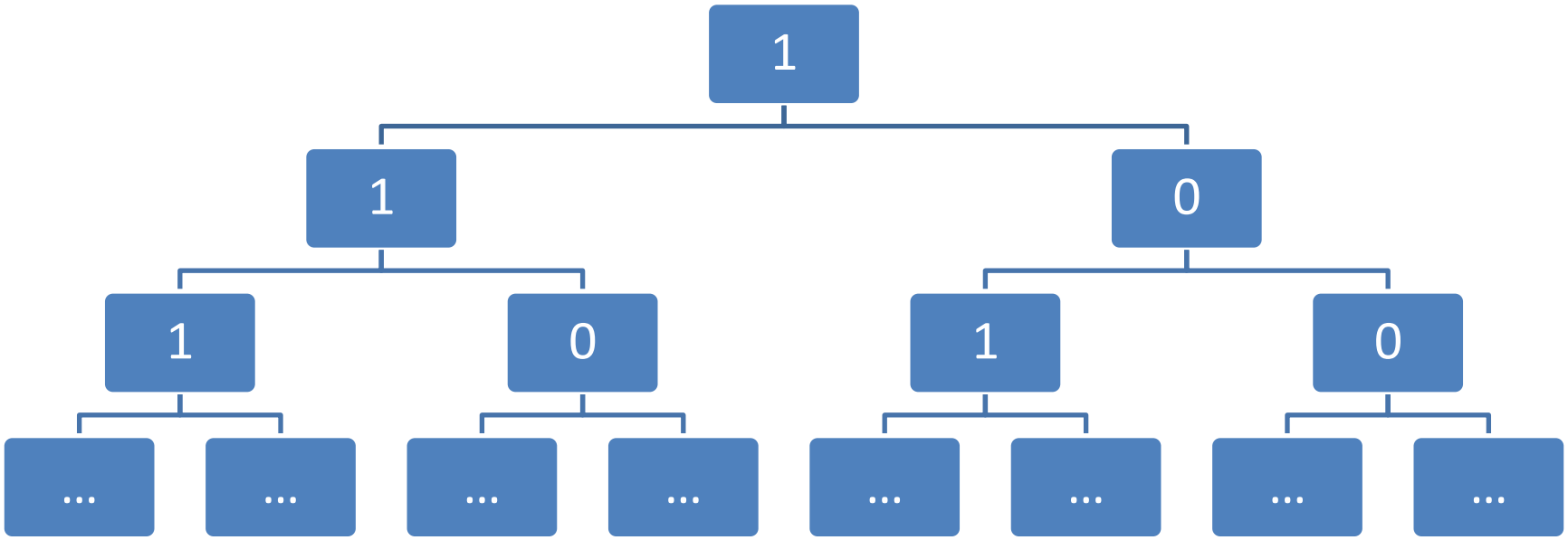
```
for (int i=1; i < chosen.size()+1; i++)
        shiftSet <<=1;
        if(chosen[i]==1) {
                sumSet^=shiftSet;
        }
}
```

Algorithm

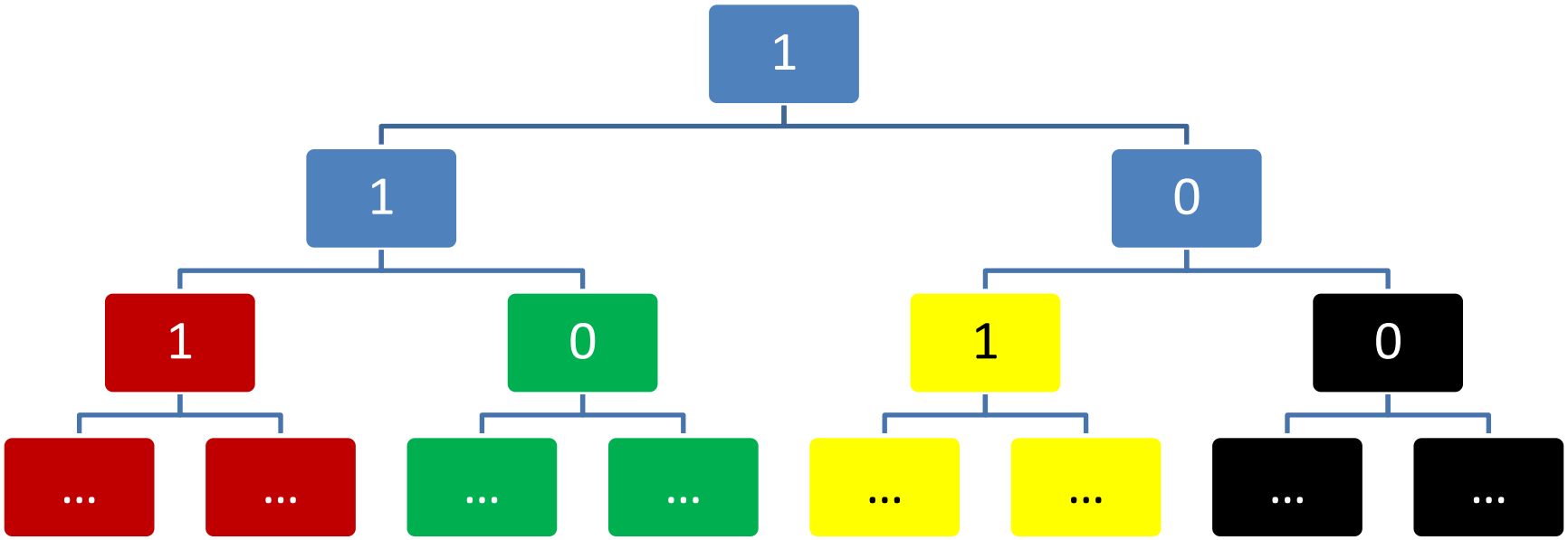
```
for (int i=1; i < chosen.size()+1; i++)
        shiftSet <<=1;
        if(chosen[i]==1) {
                sumSet^=shiftSet;
        }
}
```

- But how do we generate all the different sets of shifts to use?

Algorithm



Algorithm



Technical Details

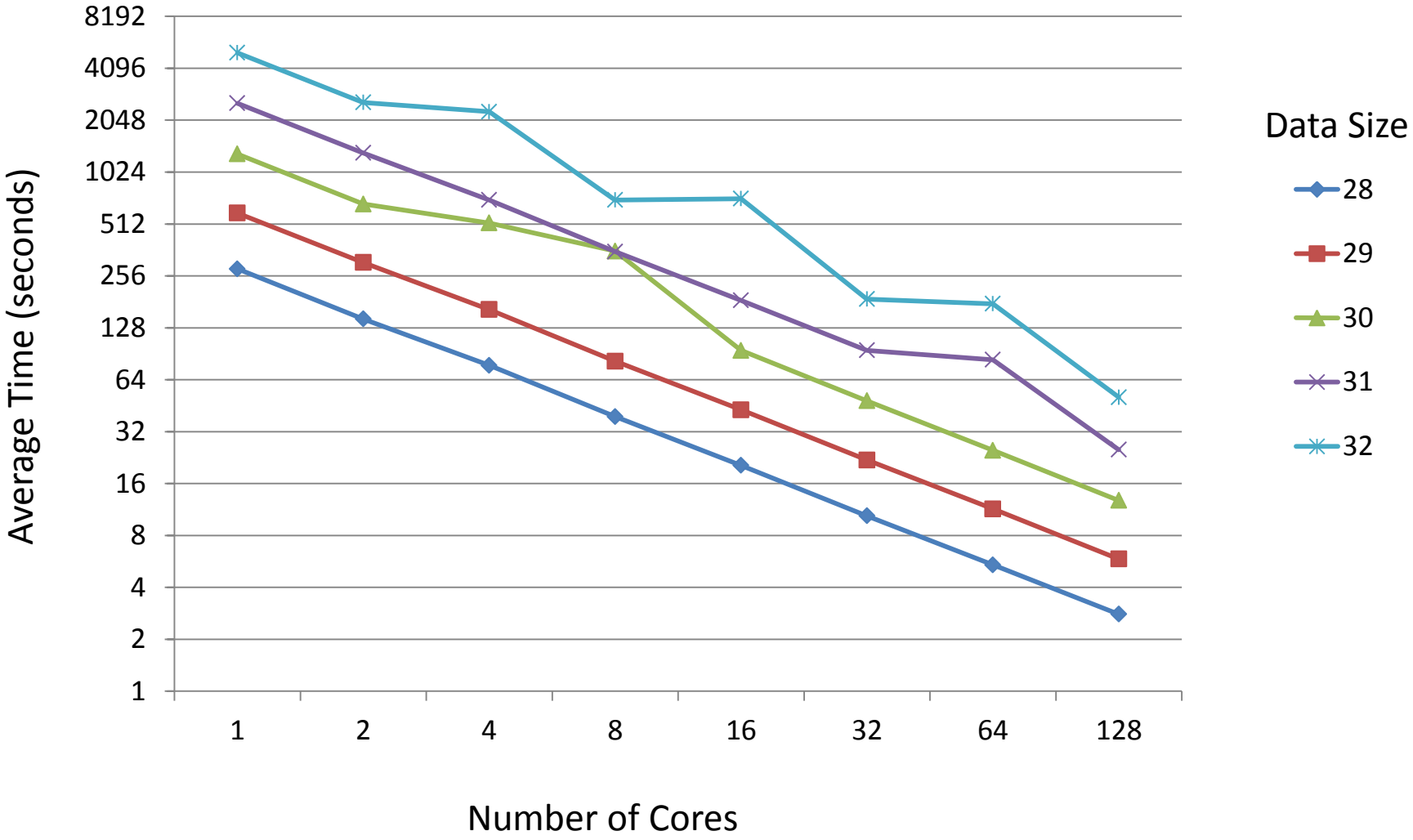
- Experiments performed on 12-core nodes using MPI
- Average runtime of 10 randomly chosen bitstrings (generated using random.org)
- For testing lengths <32, use substring of above strings

Results (Full)

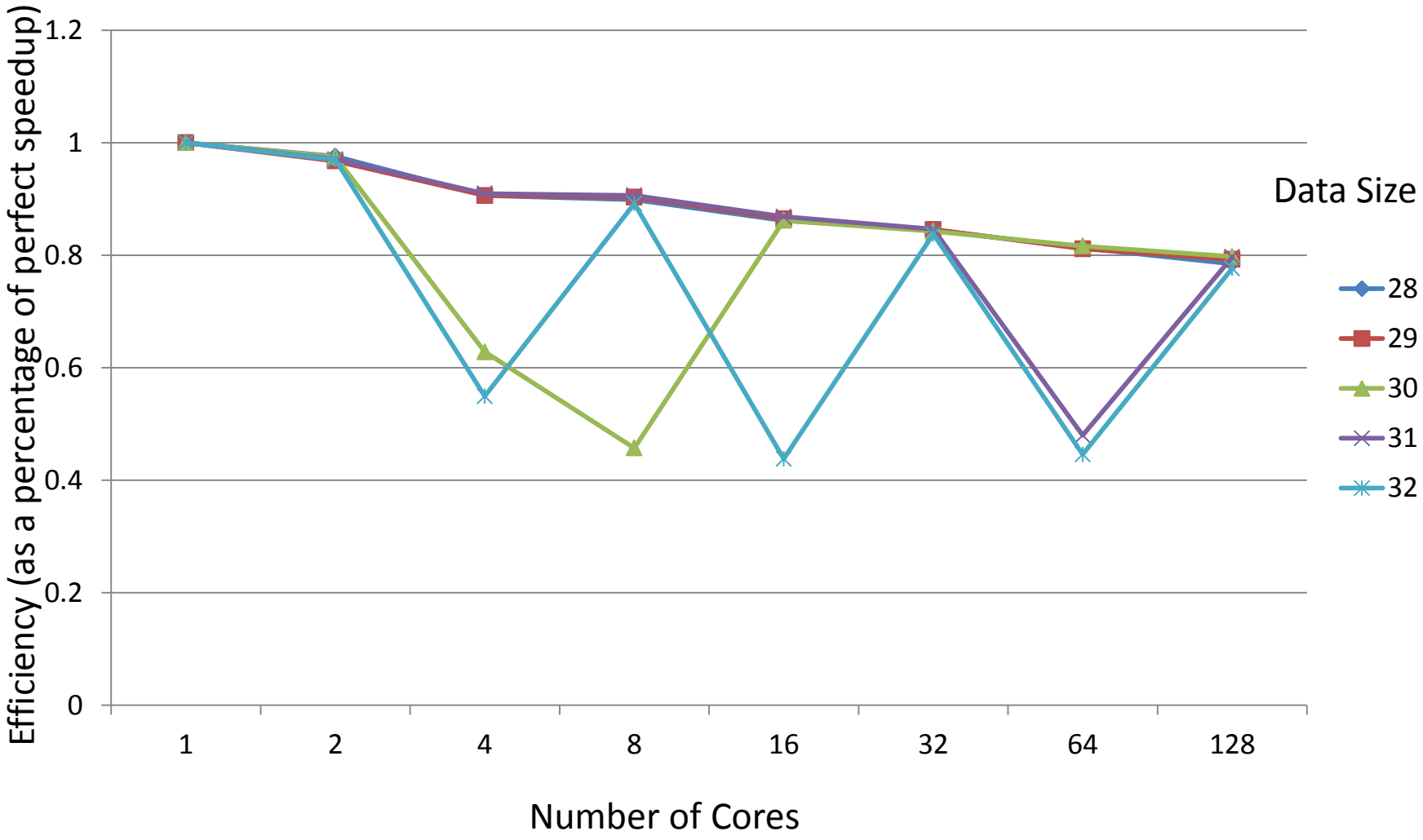
Core Count

Data Size	1	2	4	8	16	32	64	128
28	281.13	144.03	77.56	39.10	20.39	10.40	5.40	2.80
29	592.65	306.25	163.49	82.00	42.83	21.90	11.41	5.84
30	1303.51	668.63	518.77	356.37	94.54	48.35	24.95	12.77
31	2566.96	1322.49	705.58	354.04	184.72	94.77	83.62	25.17
32	5032.04	2594.25	2290.09	705.24	718.00	187.75	176.32	50.62

Results (Full)



Results -- Efficiency

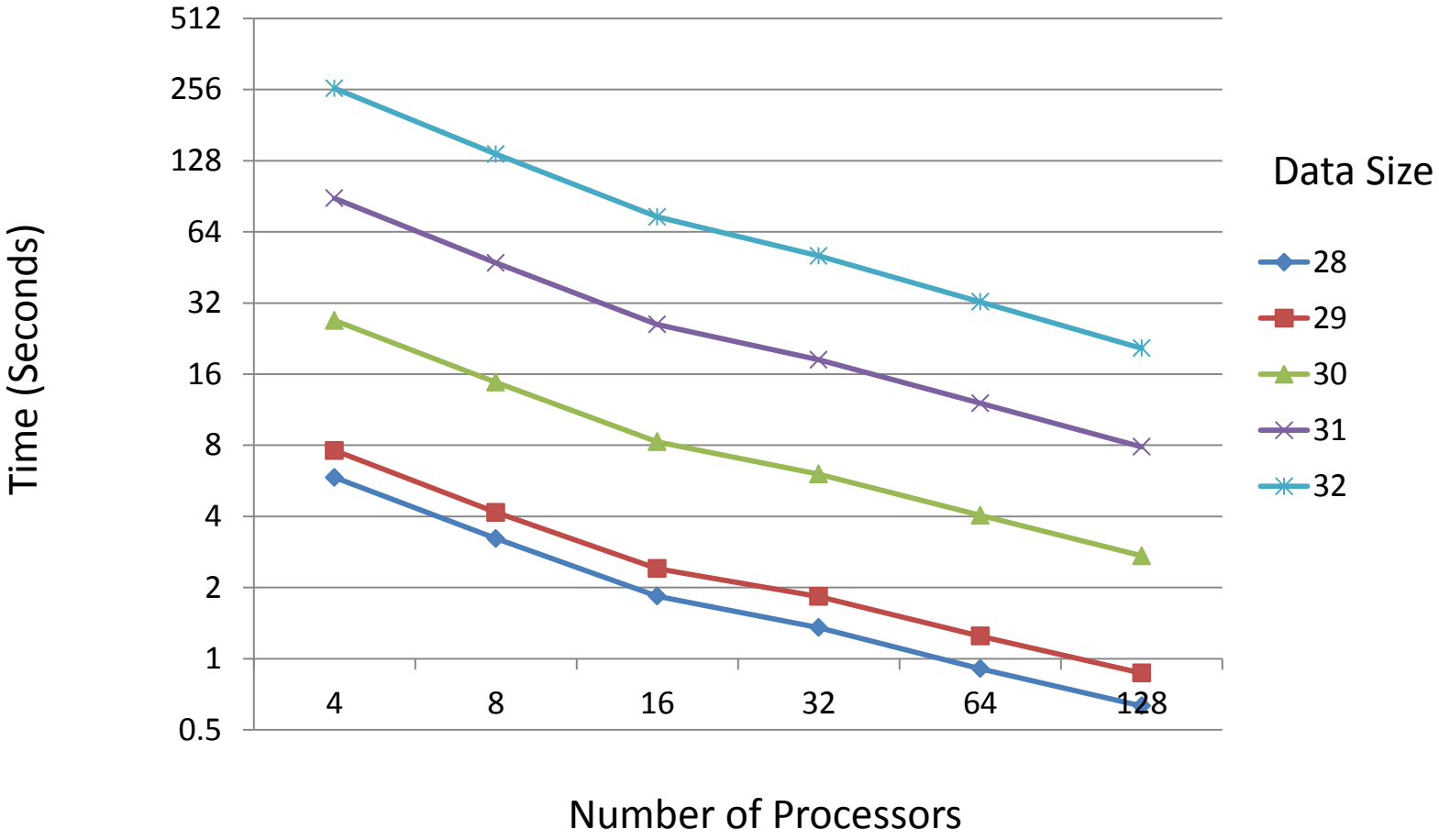


Results -- 10011011000000010101000001100111

Processor Count

Data Size	4	8	16	32	64	128
28	5.85	3.23	1.84	1.35	0.91	0.63
29	7.61	4.16	2.41	1.83	1.25	0.87
30	26.97	14.78	8.27	6.04	4.04	2.72
31	88.81	47.28	25.95	18.40	12.05	7.88
32	259.03	136.44	73.98	50.60	32.36	20.60

Results -- 1001101100000001010100000110011 1



Other Causes

- Load imbalance

Other Causes

- Load imbalance

- Highly varying times for same length input

Other Causes

- Load imbalance
- Highly varying times for same length input
- Word Size (?)

However…

Processor Count

Data Size	1	2	4	8	16	32	64	128
28	281.13	144.03	77.56	39.10	20.39	10.40	5.40	2.80
29	592.65	306.25	163.49	82.00	42.83	21.90	11.41	5.84
30	1303.51	668.63	518.77	356.37	94.54	48.35	24.95	12.77
31	2566.96	1322.49	705.58	354.04	184.72	94.77	83.62	25.17
32	5032.03	2594.25	2290.09	705.24	718.00	187.75	176.32	50.62

^what?

Future Goals

- Improve load balance without heavily increasing communication
- Optimization of math
- Implement in OpenMP

Questions/Comments?