CS257 Introduction to Nanocomputing

Coded Computation I

John E Savage

Lecture Outline

- Coded Computation: Replace the repetition code with something more efficient.
- First we review work from the 50s and 60s on coded computation, highlighting some negative results.
- Next we look at Dan Spielman's paper **Highly Fault-Tolerant Parallel Computation** *Procs 37th Annl IEEE Conf. Foundations of Computer Science*, pp. 154-163, 1996.
- Spielman's approach realizes reliable circuits with unreliable gates more efficiently than the "von Neumann" method.

Coded Computation

- **Goal:** Reliably compute function $g_k(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{y}$ where $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{y}$ are of length k.
- **Method:**
  - **Encode** $\mathbf{x}_1$, $\mathbf{x}_2$. Compute $G_n(E(\mathbf{x}_1),E(\mathbf{x}_2)) = E(\mathbf{y})$. $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{y}$ are encoded in the same code C (prevents cheating).
  - Result of computation is $\mathbf{z}$, a noisy version of $\mathbf{y}$.
  - **Decode** $\mathbf{z}$ to a codeword in C.
- The challenge is to choose codes to ensure that $\mathbf{z}$ can be decoded correctly in the presence of errors.

Local Coded Computation

- We explore the idea that G_n, like g_k, is **local**, *i.e.* operates component-wise on $E(\mathbf{x}_1)$, $E(\mathbf{x}_2)$.

- [Winograd 62] has shown that when G_n is local and input and output codes are the same, $n \geq 2e+1$ to cope with e errors (See following.) Can't beat repetition!

- Here the amount of redundancy required to cope with faults is n/k. As with communication, the "rate" is k/n.

A Negative Result

Let $\mathbf{y} = g_k(\mathbf{x}_1, \mathbf{x}_2) = \text{AND}(\mathbf{x}_1, \mathbf{x}_2)$. Assume G_n = AND.

$$E(y) = \text{AND}(E(x_1), E(\mathbf{x}_2)) = E(\text{AND}(\mathbf{x}_1, \mathbf{x}_2))$$

Note: $a \Rightarrow b$ if and only if AND(a,b) = a.

If $\mathbf{x}_1 \Rightarrow \mathbf{x}_2$ ($x_{1i} \Rightarrow x_{2i}$ for each *i*), $E(\mathbf{x}_1) \Rightarrow E(\mathbf{x}_2)$. Follows ‘cause $\text{AND}(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1$ implies $\text{AND}(E(\mathbf{x}_1), E(\mathbf{x}_2)) = E(\mathbf{x}_1)$

Since $g_k(0^k, \mathbf{x}_2) = 0^k$, $E(0^k) = \text{AND}(E(0^k), E(\mathbf{x}_2))$ must be 0^n because 0^n implies all $E(\mathbf{x})$. Similarly $E(1^k) = 1^n$.

A Negative Result (cont.)

If $\mathbf{x}_1$ is all 1s, E($\mathbf{y}$) = E($\mathbf{x}_2$). If we set $\mathbf{x}_2$ to all 0s, then set k bits to 1 one at a time, some bits in E($\mathbf{x}_2$) increase from 0 to 1. Each change in $\mathbf{x}_2$ generates a new codeword.

To tolerate e errors, at least (2e + 1) 1s must be added at each step. Thus, $\mathbf{n \geq (2e + 1)k}$.

Same for OR, $n \geq (2e+1)k$. ♦

Non-Local Coded Computation

- Coded computation cannot be local without paying a high price.
- Must consider non-local coded computation.

Spielman's Computational Model

- *d*-dim hypercube executing *normal algorithms*
  - *Normal algorithms* exchange info between processors along the same dimension on a step.
  - Other parallel models can be mapped to hypercube with poly-log time loss.
  - Each hypercube node is a processor
  - Processors exchange info with neighbors and compute in synchronism.
  - Each processor has own instruction stream.

Spielman's Encoding Model

- Encodes tuple of processor states (σ_j) using a RS code $p(x)$.
- Encodes tuple ($w_{j,t}$) of instructions on tth step for processors using a RS code $q(x)$.

Spielman's Implementation of Computation

- The function $\Phi(\sigma_j, \beta, w_{j,t})$ combining processor state σ_j for the jth processor with neighbor state σ_s and instruction word $w_{j,t}$ on tth step is implemented using interpolation polynomial ϕ(u,v,w).
- States encoded with polyn. *p(x),* permuted states with *p'*(x), and instructions with *q(x)*.
- Output is $\phi(p(x),p'(x),q(x))$ evaluated at $x \ni F$ Output is word in different RS code!

Interpolation Polynomials

- A 1D interpolation polynomial

$$m(x) \ = \sum_{i=1}^{|H|} \tau(h_i) \frac{\prod_{j \neq i}(x - h_j)}{\prod_{j \neq i}(h_i - h_j)}$$

- A 2D interpolation polynomial

$$m(x) \ = \sum_{i=1}^{|H|} \sum_{j=1}^{|H|} \tau(h_{i,j}) \frac{\prod_{r \neq i}(x - h_r)}{\prod_{r \neq i}(h_i - h_r)} \frac{\prod_{s \neq j}(y - h_s)}{\prod_{s \neq j}(h_j - h_s)}$$

- $\phi(\sigma_j, \beta, w_{j,t})$ represented by