CSCI 2570 Introduction to Nanocomputing

Coded Computation II

John E Savage

Lecture Topic

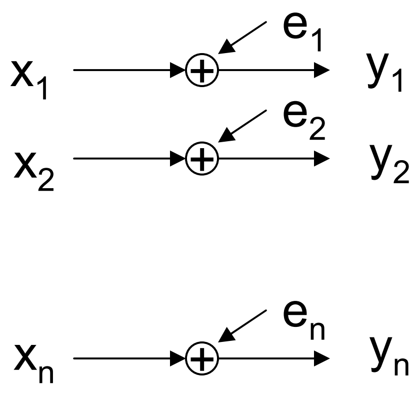
- This talk is based on Dan Spielman’s paper **Highly Fault-Tolerant Parallel Computation** *Procs 37th Annl IEEE Conf. Foundations of Computer Science*, pp. 154-163, 1996.
- **Spielman’s goal:** To realize circuits with unreliable gates more efficiently than the “von Neumann” method.
- **The approach:** To replace the repetition code with a more efficient one.

Coded Computation

- **Goal:** To make parallel computation reliable.
- **Model:** Gates fail (compute incorrectly) with statistical independence and probability ε.
- **Approach:**
  - Encode step inputs and outputs with same code.
  - Design step operations so that a fraction ≤ θ of outputs are in error for each step, $\theta = O(\varepsilon)$, with probability p.
- **Result:**
  - T step computation fails with probability $\leq Tp$.
  - Reliable decoder produces corrected result with probability $\leq Tp$ if the code can correct a fraction θ of the errors.

Relationship to Coded Communications Model

- *k* characters are encoded into *n* inputs
- *n* inputs sent through channel
- Output is correctable with probability *p* if the probability that ≤ e errors occurs is ≤ *p* where e is the error correcting capability of the code.



Interpretation of the Coded Computation Model

- Model assumes gates fail independently.
- Each step
  - Operates on encoded input and produces encoded output in the absence of errors.
  - Assumes the fraction of inputs in error is ≤ θ as is the fraction of outputs.
  - Fails to meet above requirement with small probability p.
    - Steps must not compound errors.
- Computation is reliable with probability $\leq Tp$ if inputs and outputs are reliably encoded and decoded.

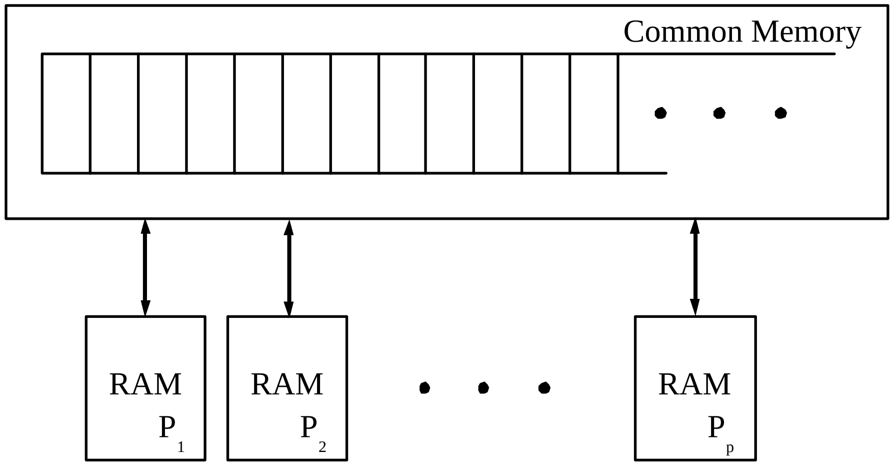
Interpretation of the Coded Computation Model

- Application to circuits:
  - Compute circuit with parallel machine.

Three Parallel Models of Computation

- Parallel Random Access Model (PRAM)
- Hypercube
- Mesh

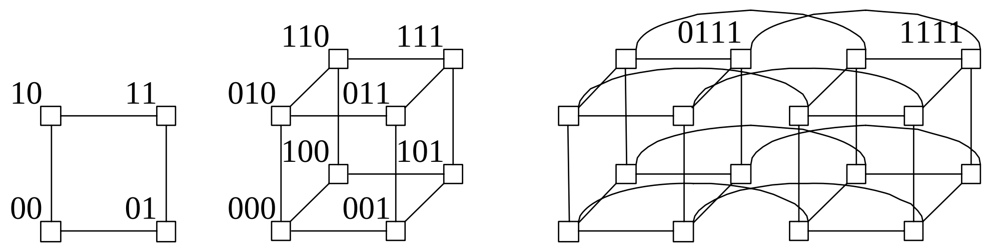
The PRAM Model



- The PRAM is an abstract programming model
- Four types: EREW, ERCW, CREW, CRCW
- Can Boolean functions be computed quickly?
  - How to represent a function?
  - Can we use concurrency to good advantage?
  - Is this use of concurrency realistic?

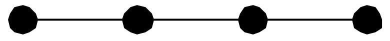
Hypercube-Based Machines

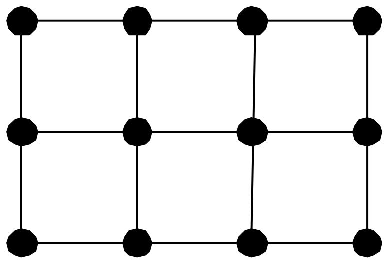
- Has 2^d vertices labeled by binary d-tuples

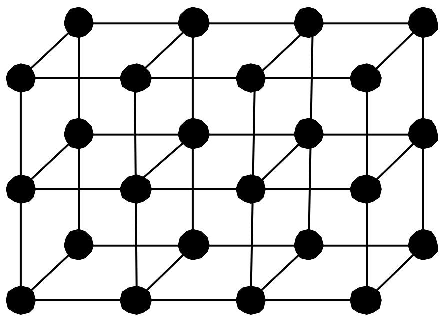


- Has $d2^d$ edges. Can be formed by joining two $(d-1)$-dimensional hypercubes by edges at corresponding vertices.

Mesh-Based Machines


(4)-mesh


(4,3)-mesh


(4,3,2)-mesh

Application to Circuits

- Realization of a circuit on EREW PRAM
  - Reduce fan-out to two without increasing circuit size by more than a constant factor.
  - Assign one processor to each gate.
  - Read left input which is first fan-out.
  - Read left input which is second fan-out.
  - Read right input which is first fan-out.
  - Read right input which is second fan-out.
  - Write to location reserved for gate output.

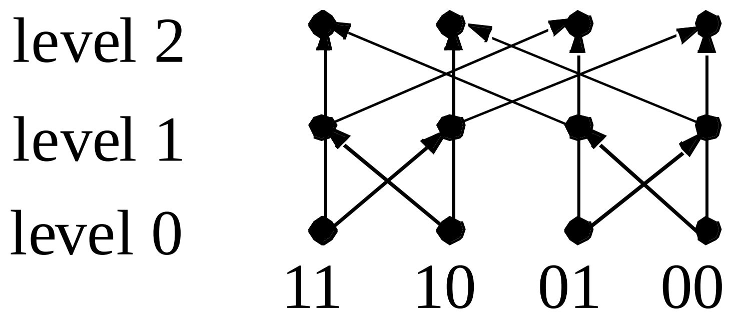
Simulation of EREW PRAM on Hypercube

Theorem Each computation cycle of an EREW PRAM can be simulated on a p-vertex hypercube with high probability in O(log p) steps.

Proof See Karlin and Upfal "Parallel Hashing: An Efficient Implementation of Shared Memory" JACM vol. 35, no. 4, pp. 876-892, 1988.

Normal Algorithms

- A **normal algorithm** on a hypercube is one in which data moves synchronously across one dimension at a time.
- FFT is example



- Input data on vertices at level 0. Computations at level 1 use exchanges with neighbors differing in least significant bit. Computations at level 2 are done after exchanges between neighbors differing on most significant bit.

Hypercube (HC) Computation

- $N = 2^n$ nodes indexed by binary n-tuples in $H = GF(2^n)$
- Processors at nodes are identical but each processor has its own instruction sequence.

Hypercube (HC) Computation

- The i^{th} processor has state $\sigma_{i,j} \in S$ and instruction sequence $w_{i,j} \in S$ on j^{th} step.
  - States and instructions are in S.

- Successor node state $\sigma_{i,j}^{*} = \phi(\sigma_{i,j-1}, \sigma_{i+d,j-1}, w_{i,j})$ on j^{th} step where $\sigma_{i,j-1}$, $\sigma_{i+d,j-1}$ and $w_{i,j}$ are i^{th} node state, state of neighbor in current dimension, & i^{th} node instruction.

Encoding Data on Hypercube

- Encode processor states and instructions with Reed Solomon code over *F*.
  - Let $S \subseteq F$, S contains states & instructions, $s = |S|$
  - Index HC processor nodes by elements in $H \subseteq F$, $|H| = N = 2^n$.

RS Encoding of Data on Hypercube

- $\sigma_{ij} \in S$ for $i \in H$. Form $m_j(x)$ so that $m_j(i) = \sigma_{ij}$. Deg(m_j)=N-1 Encode as $\Sigma_j = (m_j(1), m_j(2), \ldots, m_j(|F|))$. (The set of j^{th} states used to form one codeword.)
- Let Σ_j^d denote permutation of Σ_j so that element in ith position is the neighbor of ith element across the dth dimension.
- $w_{ij} \in S$ for $i \in H$. Form $n_j(x)$ so that $n_j(i) = w_{ij}$. Deg(m_j)=N-1 Encode as $W_j = (n_j(1), n_j(2), \ldots, n_j(|F|))$. (The set of j^{th} instructions used to form one codeword.)

Computing with Encoded Data

- Recall $\sigma_{i,j}^{*} = \phi(\sigma_{i,j-1}, \sigma_{i+d,j-1}, w_{i,j})$ on jth step where $\phi{:}S^3 \rightarrow S$ is next-state function of a processor.

- The codewords Σ_j, Σ_j^{d} and W_j contain current state of a node, its neighbor and its instruction. We can apply ϕ to components in S, not those in F.

Computing with Encoded Data

- To handle values in *F* not *S*, extend ϕ to the interpolation polynomial $\Phi(r,s,t)$, where *r*,*s*,*t* in *F* such that for h_i,h_j,h_k in *H*, $\Phi(h_i,h_j,h_k) = \phi(\sigma^i,\sigma^j,\sigma^k)$ where $\sigma^i,\sigma^j,\sigma^k$ are corresponding elements of *S*.

- Form

$$\Phi(r,s,t) = \sum_{i,j,k} \phi(\sigma^i,\sigma^j,\sigma^k) \frac{\prod_{u\neq i}(r-h_t)}{\prod_{u\neq i}(h_i-h_t)} \frac{\prod_{u\neq j}(s-h_t)}{\prod_{u\neq j}(h_i-h_t)} \frac{\prod_{u\neq i}(r-h_t)}{\prod_{u\neq i}(h_i-h_t)}$$