Optimizing compute graphs

author

April 8, 2015

1 Enumeration

Assumption: Only care about functions f that are multivariate polynomials, e.g.,

$$f = f(X_1, X_2, \cdots, X_n).$$

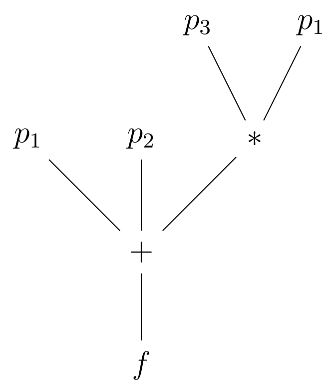
I am going to ignore division and minus for now. I also don't really know how to handle the coefficients, I am somehow magically assuming they don't exist (you will see later).

Definition: Compute graph $\mathcal{G}$ is a "representation" of some function f. Each function f is "represented" by multiple compute graphs. Consider the following "recursive" definition of $\mathcal{G}$: A compute graph $\mathcal{G}$ is said to be k-computation of a multivariate polynomial f, if f satisifes

$$f = g(p_1, p_2, \cdots, p_k)$$

where i) g is a k-variate polynomial, and ii) p_i are polynomials in $X_1, \cdots, X_n$ that have k-computation representations by "children" graphs of $\mathcal{G}$. The motivation for k-computations from practical standpoints, is that the maximum number of mults/adder is k. You can also limit the number of adds but that is a simple generalization. Since I ignore division the polynomials p_i must have degree at most that of f.

The below computation graph is a 2-computation representation of f.



I am interested enumerating all possible k-computations of some multivariate polynomial f. If we can enumerate, we can pick the one with some smallest cost.

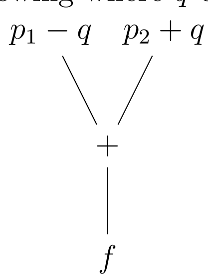
General idea: Maintain a pool of multivariate polynomials. Build larger degree polynomials by "combining" them via k-variate polynomials. We have to limit the pool by quickly eliminating polynomials that we know cannot build f. We can use the following observation to eliminate candidates.

Observation: If a polynomial p can be used to build f, then any monomial of p must divide some monomial of f.

Example: A trite one, $f = X_1^n$. In this case, any polynomial p that is a function of any $X_2, \cdots$ is eliminated.

Example: Consider $f = X_1X_2 + X_2X_3 + X_1X_3$. In this case, any polynomial p that has multiple variates, e.g. $X_2^2X_3$, is eliminated.

Example: Why this fails with minus. Very simple. We could have the following where q could be an arbitrary large degree polynomial.



Similar bad things will happen with divide.

Example: Hardamard transform. This is the Kronecker power of the matrix

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

The 2×2 transform is given by the polynomials $X_1 + X_2$ and $X_1 - X_2$. The

4×4 transform is

$$X_1 + X_2 + X_3 + X_4 \tag{1}$$
$$X_1 - X_2 + X_3 - X_4 \tag{2}$$
$$X_1 + X_2 - (X_3 + X_4) \tag{3}$$
$$X_1 - X_2 - (X_3 - X_4) \tag{4}$$

The following 1-computation graph represents (all rows of the) Kronecker transform (or actually any Kronecker matrix product).

$$\begin{tikzcd} f_{n-1} \arrow[dr, dash] & & f_{n-1} \arrow[dl, dash] \\ & + \arrow[d, dash] & \\ & f_n & \end{tikzcd}$$

where f_n is a row of the $2^n \times 2^n$ Hardamard transform. We simply need to adjust for the coefficients $1/-1$. This particular computation graph will achieve the smallest depth (i.e., $\log(n)$ for each row). There are of course other compute graphs, but the polynomial that represents all rows of the $2^n \times 2^n$ transform has the general form

$$f_n = \sum_{i=1}^{2^n} \prod_{j=1}^{m} a_{ij} X_i,$$

where the coefficients a_{ij} come from the Kronecker product. To achieve $n \log(n)$ total operations, there must be term reuse (recall the butterfly structures) so we would have to optimize 2^n compute graphs simulataneously. In this case we only need to maintain the pool of univariate polynomials in $X_1, X_2, \cdots, X_n$.