

Parallel Fast Fourier Transform

10/10/07

Review of last class

- We previously discussed breaking the evaluation of a polynomial into even and odd components.
- Based on this formulation, we developed a “divide and conquer” approach that required $O(\log n)$ time.

The trick (part deux)

- The n^{th} roots of unity are all the complex numbers that yield 1 when raised to a given power n .
- One is primitive if the powers from 1 to $n - 1$ do not return a value of 1
 - Example: $e^{2\pi i/n}$ denoted by ω^n

Goal

- We wanted like to reduce the runtime complexity from n^2 to $n \log n$.
- Today, we will discuss how to solve these subproblems on a parallel computer in $O(\log n)$ computation time using n processors.

Example

- As an example, consider a single channel of CD-quality sound: 44k samples per second.
- To compute a Fourier transform on a subset of 1000 would require 2 million operations and 88 million floating point operations.
 - 1 multiply and 1 add for n^2
- FFT, however, only requires ~20,000 operations. This is a 100X improvement.

Algorithm

- Calculating each component of the Fourier transform separately involves a substantial number of repeated calculations.
- Instead, lets start with n complex numbers and produce a new set of n numbers using our butterfly network.
- Because each stage involves $O(n/2)$ operations each stage can be done in $O(n)$ time.

Parallelization

- Parallel FFT can best be described using a static butterfly network.
- Remember that a butterfly is a modified hypercube such that each processor has a constant number of links.
 - As such, only hypercubic communication permutations are required.

Basics

- We will only use hypercubic permutations in our algorithm.
- Further, we will consider solving a FFT problem of size n on a butterfly with n rows and $\log n + 1$ columns.

Approach

- We draw the butterfly such that the lower dimension cross links are to the left and the higher dimension ones are to the right.
- Input to the network are the coefficients of the zero degree polynomials at the leaves of the divide and conquer tree.
- Polynomial P_i is the input to row i from the left and level i of the tree is mapped to column i in the butterfly network.

Approach

- At level i of the tree, there are 2^i polynomials to be evaluated at $p/2^i$ distinct values.
- Each processor in column i of the butterfly evaluates one of these polynomials at one value.
- To do so, we divide the p processes in the column into 2^i groups, each with $p/2^i$ consecutive processors.

Computation

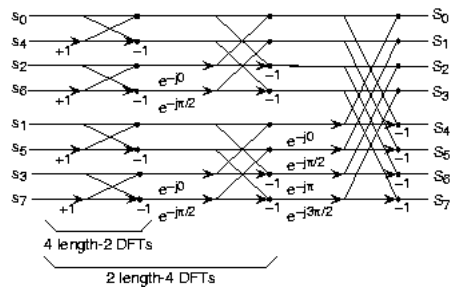
- Suppose we label a processor with an id $\langle r, 0 \rangle$ corresponding to the 0th column.
- This processor should compute $P(\omega^r)$
- In general, all processors evaluate the same polynomial, but using different values of x .

Finding relative positions

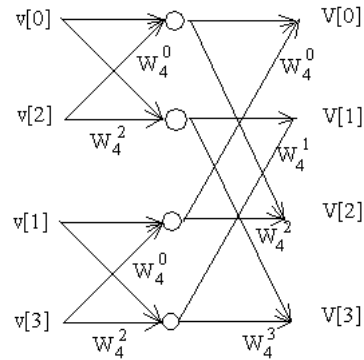
- Note that the power of ω to be used by a processor only depends on its relative position in its group.
- This can be obtained by ignoring the most significant bit of the row id.
- In addition, the power is twice the number represented by the remaining bits.
 - It follows that the power can be directly determined by shifting the bit representation of r
 - Equivalent to ignoring and multiplying the rest by 2

Algorithm

- Let $\langle r, i \rangle$ be the id of a processor.
- Receive u from $\langle r, i + 1 \rangle$
- Receive v from $\langle r \text{ with } (i + 1)^{\text{th}} \text{ bit flipped}, i + 1 \rangle$
- $m = \omega^{r \ll i}$
- if i^{th} bit of r is a 0
 - $y = u + mv$
- else
 - $y = v + mu$
- send y to $\langle r, i - 1 \rangle$
- send y from $\langle r \text{ with } i^{\text{th}} \text{ bit flipped}, i - 1 \rangle$



<http://cnx.org/content/m10250/latest/>



<http://www.relisoft.com/Science/Physics/fft.html>

Mapping

- Because this algorithm only uses one column of the butterfly at a time, it only uses hypercubic permutations.
- It runs in $\log(p)$ time.

Case where $n > p$

- For this case, suppose we map the divide and conquer tree to our p processors.
- To do so, we assign p subtrees to level $\log p$ to, one to each processor.
- Note each subtree can be processed without additional communication.

Differences in algorithm

- Once the subtrees are computed, the remaining tree is computed in $\log p$ steps.
 - It follows the computation time is the same
- Each stage, however, now involves evaluating a polynomial at n/p values.
 - Communication is $O(\tau + \mu n/p)$

Conclusion

- FFT can be done in $\log(n)$ parallel time with $O(n \log n)$ work
 - Efficiency is optimal