Manh, Huynh
Dawson, Rohan
Parallel Distributed System
Fall 2016

A Summary on Prefix Algorithms and Their Applications

1. Summary of prefix problem and algorithms

Prefix Problem Definition: The sum prefix computation aims to find the sum (at index K) of every elements on an array from index 0 to K. The different between sum prefix and binary sum is that for binary sum, a single result is calculated at the end by summing all elements in an array while in prefix computation, not only the final result (sum of all elements) but results at intermediate indexes (at position k in the array) is also calculated.

As we know the depth of tree for finding summation of n elements in parallel is $\log_2 n$ and size is still n-1. Sum prefix problem appears to be a much more complex because of data dependence. The prefix algorithms such as upper/lower, odd/even, and Ladner and Fishcher's algorithms are invented to solve this problem so that we can achieve the minimum depth and sizes.

It is interesting that upper/lower can solve prefix problem follow the construction of the tree with depth k and adding much more operations $\frac{kN}{2}$. This means that given the enough processors so that we can have a full parallelization every layer of the tree, the speed of solving prefix sum is the same as solving one single sum.

The odd/even algorithm is designed to have a more depth, which is $2k \quad 2$, but reducing the number of operations compared to upper/lower algorithms. From these, we can see that if the system having large number of processors, choosing upper/lower algorithms help solve the problem faster. However, if the number of processors is small, choosing odd/even algorithm may perform better since it retains less number of operations to be done in some sequential parts.

So far we have seen that the Upper-Lower and Odd-Even algorithms take slightly different approaches in ordering data while calculating the solution to the prefix problem. It is possible to combine them to create P_j, a new algorithm that gives a smaller size (for sufficiently large N), than either algorithm by itself.

P0 starts by splitting the input into an upper and lower half, just as in the Upper-Lower algorithm. However, it then treats the lower half as P_1, an Odd-Even implementation, and the remaining upper half as yet another P_0 Upper-Lower implementation. As such, both P_0 and P_1 are defined recursively on systematically smaller sized problems until the base case is reached.

What is remarkable about this is that the Odd-Even algorithm is generally used when the processor/solution depth trade-off favours fewer processors and a larger solution depth. The fact that this can be combined with the Upper-Lower algorithm which favours using more processors to minimise the solution depth in a relatively simple manner to harness the best of both worlds (processors and solution depth) is very unintuitive.

This is especially apparent when we take a closer look at search depth. Upper-Lower gives a solution depth of k, Odd-even gives a solution depth of 2k-2, and so we would expect an algorithm that is entirely made up of half of each to have a search depth of somewhere in the middle. However, P_j manages to give a solution depth of k even though it is impossible for the Upper-Lower part's search depth to offset that of the Odd-Even. In fact, the algorithms work together to create more parallelism than either one could on its own.

2. The applications of prefix algorithms

Application 1: Integral Image:

Integral image is used as a quick and efficient way to calculate the sum of values within a selected region (e.g rectangle) in an image. Integral image is a core implementation for Harr-like feature, which is commonly used in object detections [1].

Integral image at pixel (i,j), is the sum of all elements inside the rectangle (red part in figure 1) having the width $i + 1$ and height $j + 1$.

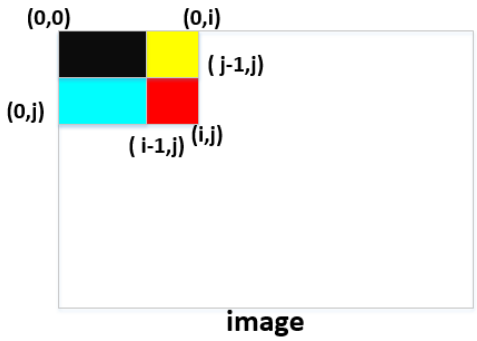

Figure 1. Integral Image

Finding all integral image at every pixel on the image is considered to be a prefix problem as follows [2]:

1) Solving prefix problem for each row of image (can be done through several prefix algorithms we studied).
2) Transpose the result from 1.
3) Solving prefix problem for each row of transposed result in 2.
4) Transposing the result matrix in 3, to get the final result.

The sequential method to find integral image can be done efficiently using following equation:

$$s(i,j) = I(i,j) + s(i \quad 1,j) + s(i,j \quad 1) + s(i,j)$$

Where $s(i,j)$ is integral image at pixel (i,j), $I(i,j)$ is the value of pixel (i,j) with gray-scale image assumption. Results from [2] shows that it achieves 9x , 7x faster than sequential method on a single

precision floating point matrix and double precision at image size ~ 10^6 pixels on NVIDIA GeForce GTX 295.

Application 2: Prefix problem application: Line of sight problem

A more general form of the prefix problem can be applied to calculate the line of sight from a given start location looking over a series of points of varying altitudes. For example, in a video game, the program only needs to render graphics for objects that the player can currently see. Hence it can run this algorithm and only render areas of the map that currently need to be rendered.

The algorithm works by first taking the series of points currently in the line of sight, fetching the altitude of them, and putting them into a vector. Next, we divide each element by the distance to it, and then take the arctangent of the result. This gives us the angle between the horizontal and the respective point of altitude. Finally, we can use a modified version of the prefix sum by carrying out a maximisation operation instead of an addition. Of course, this maximisation version can be parallelised in the exact same way as the original prefix sum could. The result of this maximisation version of the prefix sum (scan) is that we have a vector that tells us the maximum altitude of the point we have seen so far. As such, the program can simply read this vector to find out exactly which points are currently in the player's line of sight.

References:

[1] Lienhart, Rainer, and Jochen Maydt. "An extended set of haar-like features for rapid object detection." In Procs. of International Conference on Image Processing. Vol. 1. IEEE, 2002.

[2] Bilgic, B, B KP Horn, and I Masaki, "Efficient Integral Image Computation on the GPU." IEEE, 2010. 528-553.

[3] Blelloch, Guy E. "Prefix sums and their applications." (1990).