**Meeting 6/6/19:**

Nirmal is working on her PhD. Her dissertation is on the importance of the optimization of these algorithms such as ordering. On Friday, she will defend the need for this kind of ordering related to the research problem. Ordering appears in neural networks, AI, image processing and machine learning.

Typically, small matrices occur in application, so the frames used are small, (10x100). This is not a large problem size for a GPU, so they do not order in the interest of precision. Ordering is also not always desirable when there is additional processing happening between multiplications.

Matt answered the roofline question. The goal of optimization is to balance the work done on data and access to data. Once under the flat part of the roofline model, one can use vectorization, parallelism and the functionality of the chip to move it up. Left and right movement does not help under the flat part. In the computational, flat roof case, the next real life consideration energy consumption. More operations, operational intensity, means fewer communications to slow memory, so less energy consumption and heat.

Moving up in the roofline model is *attained* performance – optimization is sought above the roof.

If you do not attain the gain in roof you want computationally, you may have not considered the proper memory roofs. Is there a bandwidth bottleneck? Are you reaching your expected operational intensity? Is the assumed prefetching happening? These are common misinterpretations of a roof when implementing optimization.

Paul read the floating point answer. (4 flops / 6 accesses) is interpreted as correct under stipulation of the following:

Be specific what is being counted, as accesses can be interpreted in multiple ways. The product p[i-1]p[k]p[j] over k in the ordering algorithm is invariant with k with the terms of j and k. If the multiplication of these terms is done before the loop as in compiler optimization, the the number of floating point multiplications can be reduced. The compiler may or may not do it automatically – it usually does not when multiplying floats as it is tuned for speed and not strict optimization. Multiplication is neither communicative or associative in the compiler. This is another reason it does not optimize floating point operations invariant of loops.

Our Eddy diagram is too complicated. It should encompass an arbitrarily large problem without being the form of an essentially infinite sum. Our model shows too many options, essentially all the options is too many to show.

A visualization is given from the whiteboard. There are no colors because they are not needed. This algorithm is not as complex as RNA folding where colors were added to show the depth of complexity.

**New Rule Set for MCP Eddy Diagrams (Shown in the PDF “Correct MCP Eddy Diagram”):**

* The chain can take the form of a single matrix, the base case, or a system of 2 to 3 or more points: i, k, k+1, j.
* k and k+1 are distinct points, i may equal k, j may equal k+1.
* The parenthesization of F(i,j) is thus 0 if i=j and F(i,k),F(k+1,j) if i<j choosing the minimum k in i less than or equal to k less than j.
* Problem specific modularization yields F(i,j) equals 0 if i=j, and the minimum of (F(I,k) + F(k+1,j) + Pi-1\*Pk\*Pj) for i less than or equal to k less than j.
* Dots on the horizontal axis represent matrices in the chain.
* A dotted line represents a possible parenthesization.
* A solid line represents a choice of parenthesization.

Sean Eddy invented the Eddy diagram. His work inspired the use of eddy diagrams in this project. Bioinformatics researchers understand this diagram well, so it helps programmers communicate with them by implementing from it. It is an interesting pursuit for this project.

It is part of the research process to make mistakes and learn new things.

Matt: is this used in GPU’s? Find it ourselves. There is no library for MCP or ordering. Typically, architects will implement this algorithm for MCP problems. It is useful to do this when the matrices have different sizes.

**2 Tasks this week:**

1. AlphaZ wiki - read http://www.cs.colostate.edu/AlphaZ/wiki/doku.php then do the LUD tutorial.
2. Read about caches and processors from the point of view of the hardware. How are they built? Why do we have them? <http://ac.aua.am/arm/public/2017-Spring-Computer-Organization/Textbooks/ComputerOrganizationAndDesign5thEdition2014.pdf> - Read chapter 5; understand direct map caching (first 3 sections) and understand the terminology. (page 397 of pdf). Work out the first few examples in 5.4 on paper (up to 409 of the paper itself).

Working together is encouraged. Make sure you understand the material. We can turn in one assignment for both of us.

**Additional Questions:**

How do we get class credit for this research? Register for CS498 or CS495. Send a message to Albert Lionelle, (4 credits is about 12 hours of work per week).