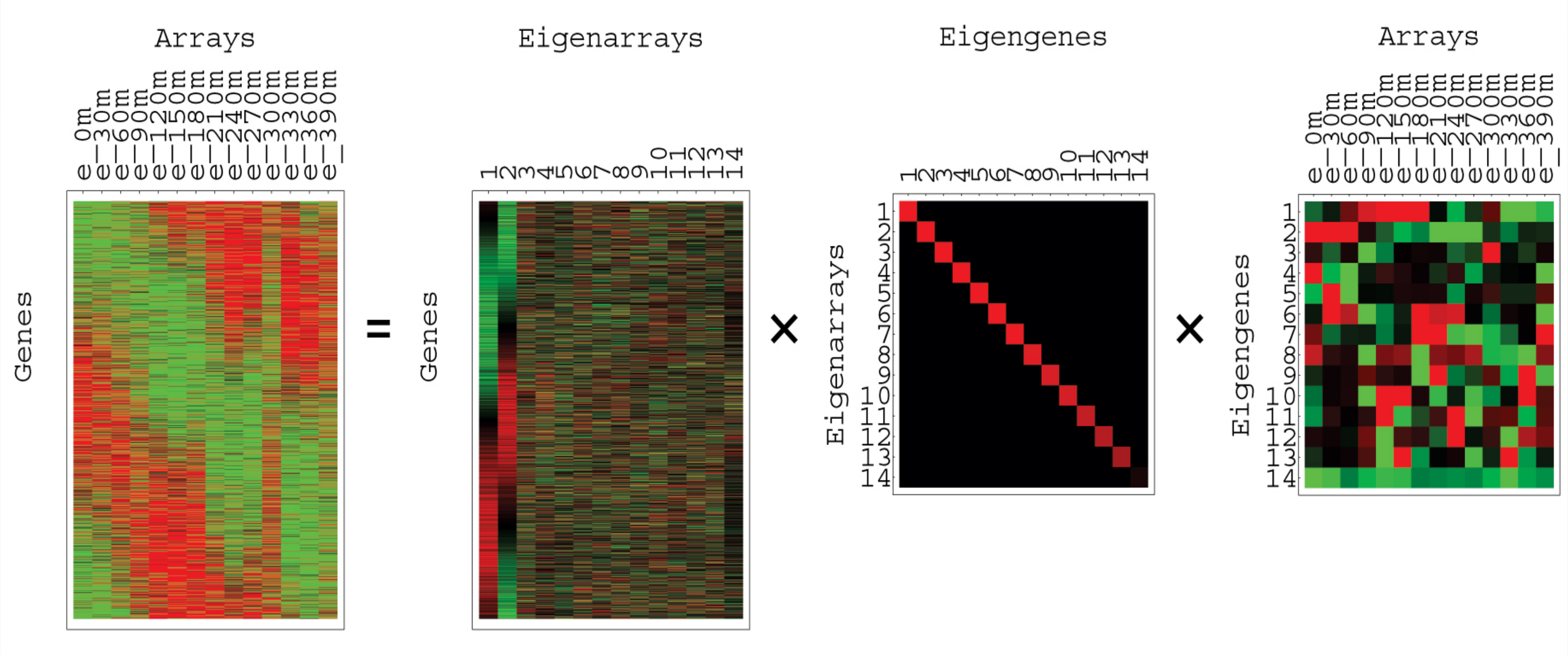
Single Value Decomposition

B649 DataFlow SuperComputing: Assignment 1

Submitted By Rishabh Monga

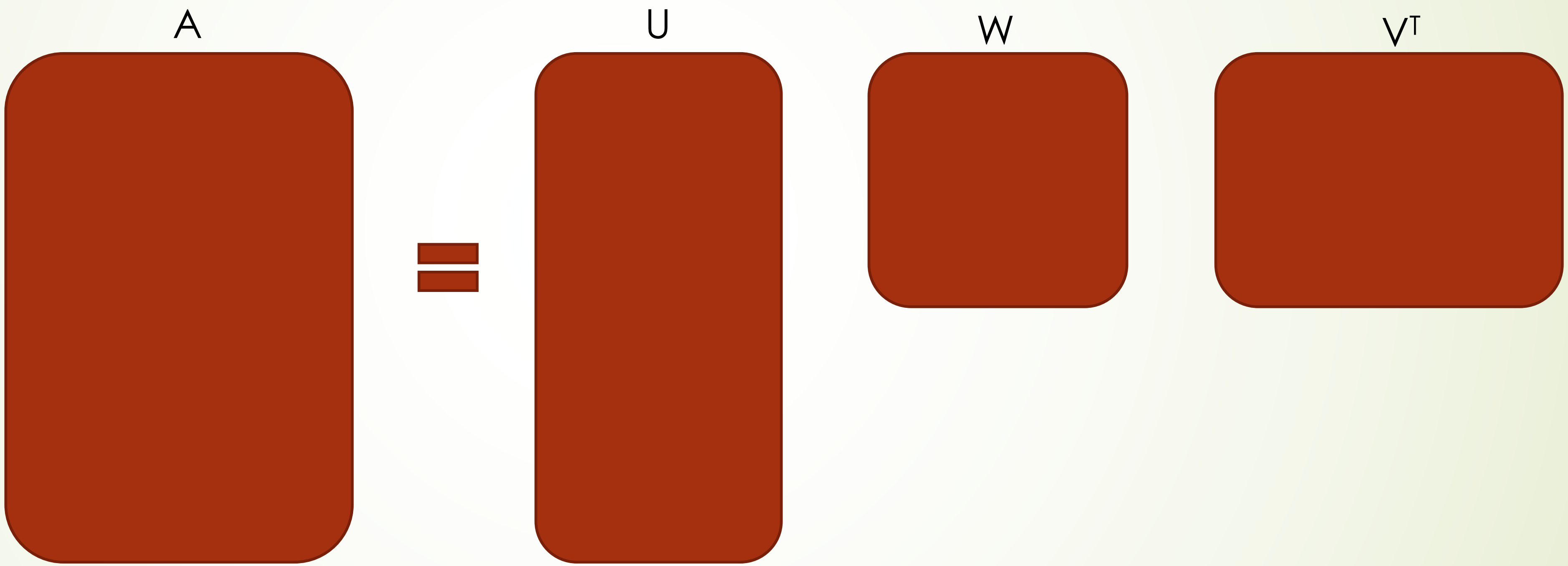
Introduction



- A singular value decomposition
  - provides a convenient way for breaking a matrix
  - which perhaps contains some data we are interested in
  - into simpler, meaningful pieces.

Example

SVD is the method of choice for solving most of the linear least-square problems



Data/ word occurrence matrix

Embeddings of tags with words they occur with

Embeddings of words with tags they occur with

- Any M x N matrix *A*
  - With $M \geq N$
- Can be written as product of 3 matrices *U*, *W*, *V*
- *U* is column orthogonal M x N
- *W* is N x N diagonal matrix
  - With elements being positive or 0 (*singular values*)
- *V* is N x N transpose matrix

$$\begin{pmatrix} \\ \\ & \mathbf{A} & \\ \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \\ & \mathbf{U} & \\ \\ \\ \end{pmatrix} \cdot \begin{pmatrix} w_1 & & & \\ & w_2 & & \\ & & \cdots & \\ & & \cdots & \\ & & & w_N \end{pmatrix} \cdot \begin{pmatrix} & \mathbf{V}^T & \end{pmatrix}$$

Extracting Kernels

- We need to identify key loops in the algorithm
- That can be transformed into Maxeler kernels
- The c language implementation of Single Value Decomposition has 23 loops
  - of which 16 are nested
- So extraction of key loops was challenging.

Extracted Loop

- I was able to identify a loop that was repeated 4 times inside the nested loops
- The loop was used to find the normal distance between elements of the matrix
- Following is the original code of the loop:

```
double svd(double a, double b) {
    double absa,absb;

    absa = fabs(a);
    absb = fabs(b);

    if(absa > absb)
        return(absa * sqrt(1.0 + SQR(absb/absa)));
    else
        return(absb == 0.0 ? 0.0 : absb * sqrt(1.0 + SQR(absa / absb)));
}
```

```
class SVDKernel extends Kernel {

  SVDKernel(KernelParameters parameters) {
    super(parameters);

    DFEFloat singleType = dfeFloat(8, 24);

    DFEVar a = io.input("a", singleType);
    DFEVar b = io.input("b", singleType);

    DFEVar at = KernelMath.abs(a);
    DFEVar bt = KernelMath.abs(b);

        DFEVar ct = constant.var(dfeFloat(8, 24), 0);
        DFEVar result = constant.var(dfeFloat(8, 24), 0);

        result = (at == KernelMath.max(at, bt)) ? at * KernelMath.sqrt(1.0 + (bt/at)*(bt/at)):ct;
        result = (bt == KernelMath.max(bt, ct)) ? bt * KernelMath.sqrt(1.0 + (at/bt)*(at/bt)):ct;

        io.output("y", result, dfeFloat(8, 24));

  }

}
```

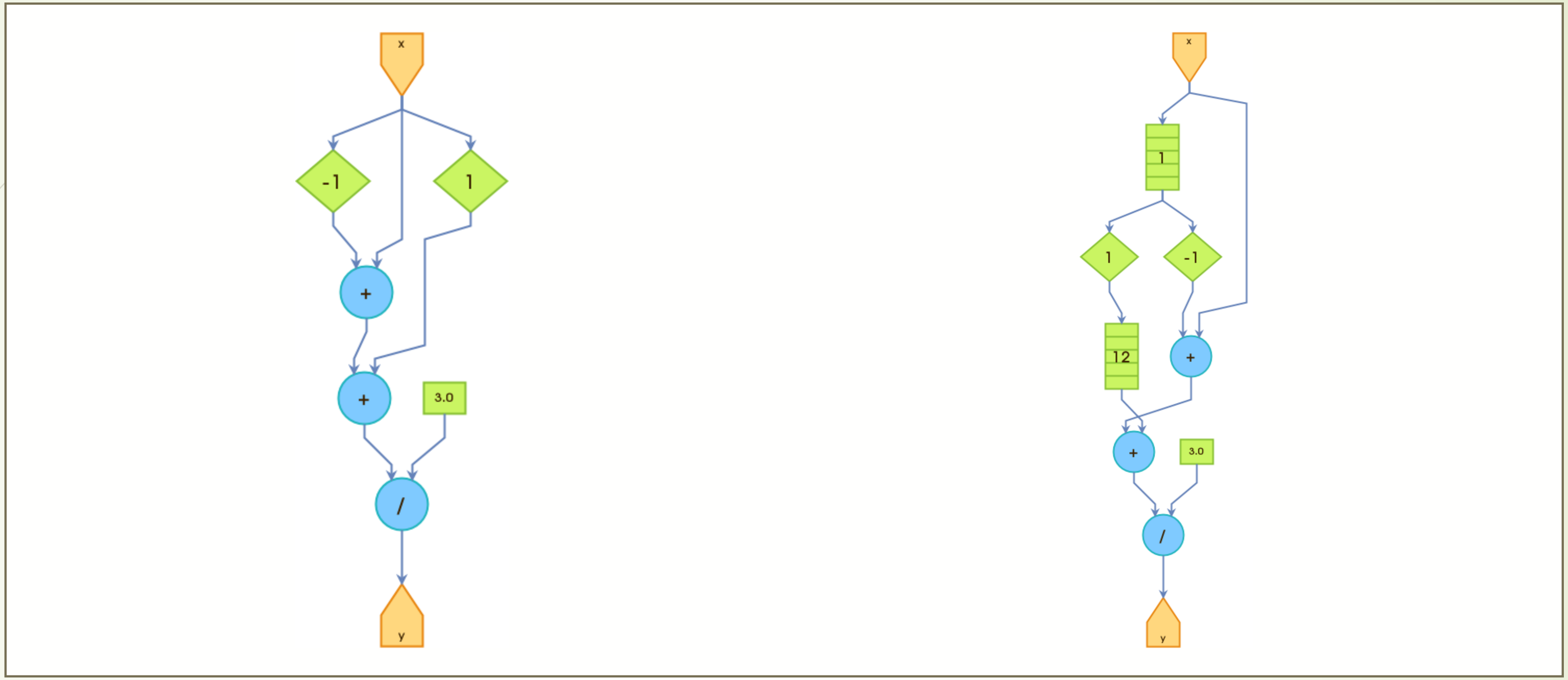
Maxeler Kernel for the extracted loop

```
class SVDManager {
  public static void main(String[] args) {
    EngineParameters params = new EngineParameters(args);
    Manager manager = new Manager(params);

    // Instantiate the kernel
    Kernel kernel = new SVDKernel(manager.makeKernelParameters());

    manager.setKernel(kernel);
    manager.setIO(IOType.ALL_CPU); // Connect all kernel ports to the CPU
    manager.createSLiCinterface();
    manager.build();
  }
}
```


Kernel Manager



Kernel Graphs

Conclusion

- SVD plays a crucial role in the fields of analysis, signal processing, patter recognition and machine learning.
- Various modelling and distributed algorithms have been applied
  - to reduce the order of execution of the SVD algorithm
- Dataflow techniques can introduce critical improvement in execution of this algorithm
  - As it has numerous loops and many of them are nested