TensorFlow For Scientific and Engineering HPC Computations: It's Not Just for Machine Learning

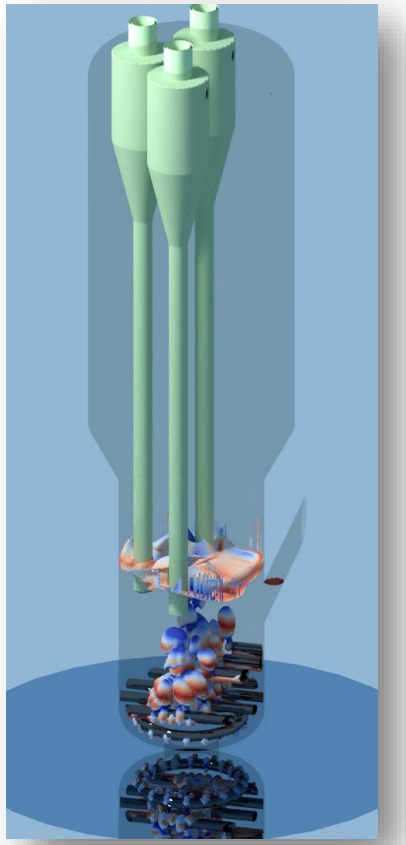
Opeoluwa Owoyele, Terry Jordan, Mino Woo, Dirk Van Essendelft*

dirk.vanessendelft@netl.doe.gov, 304-285-5231

November 20, 2019

Research Motivation

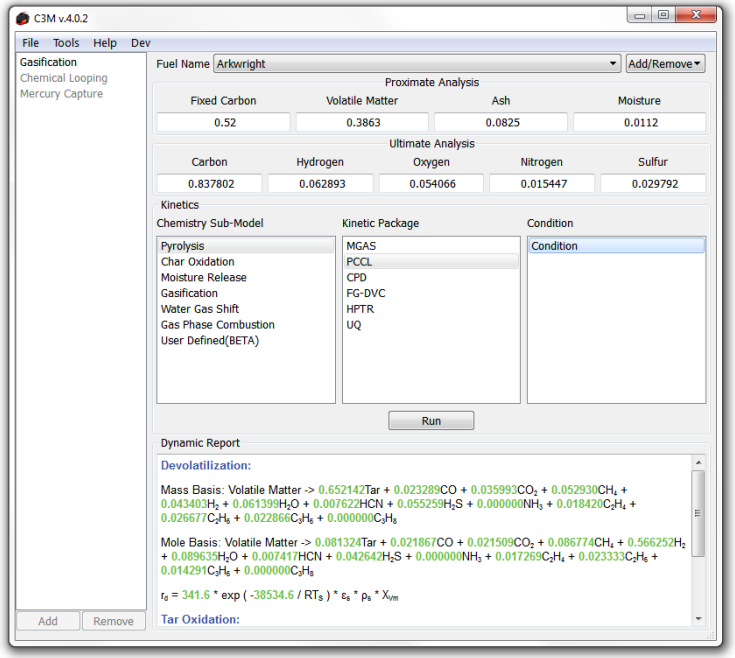
Rapid, Scalable Design and Engineering



MFiX TFM

MFiX PIC

MFiX DEM



C3M
Carbonaceous Chemistry for Computational Modeling

https://mfix.netl.doe.gov/

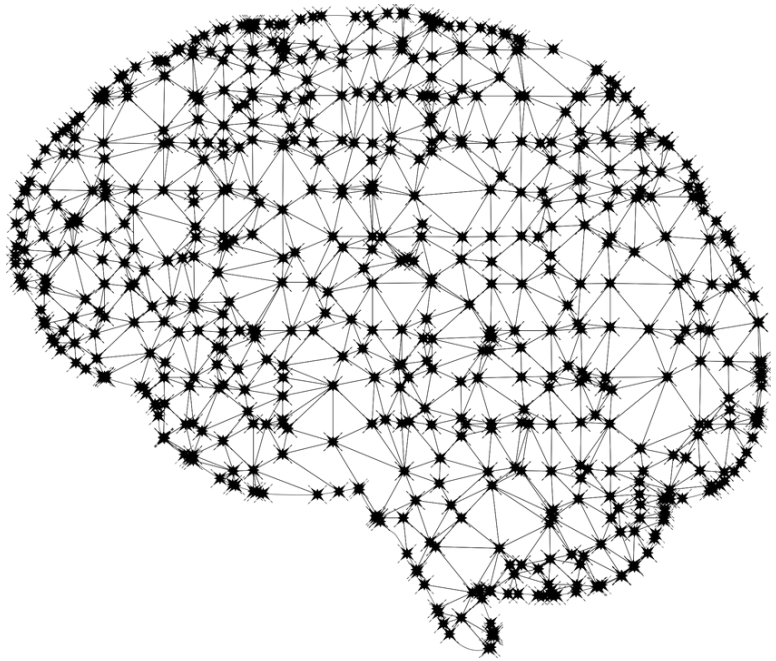
A new Paradigm

Three Pillars of Future Computation

Hardware

Software

Machine Learning



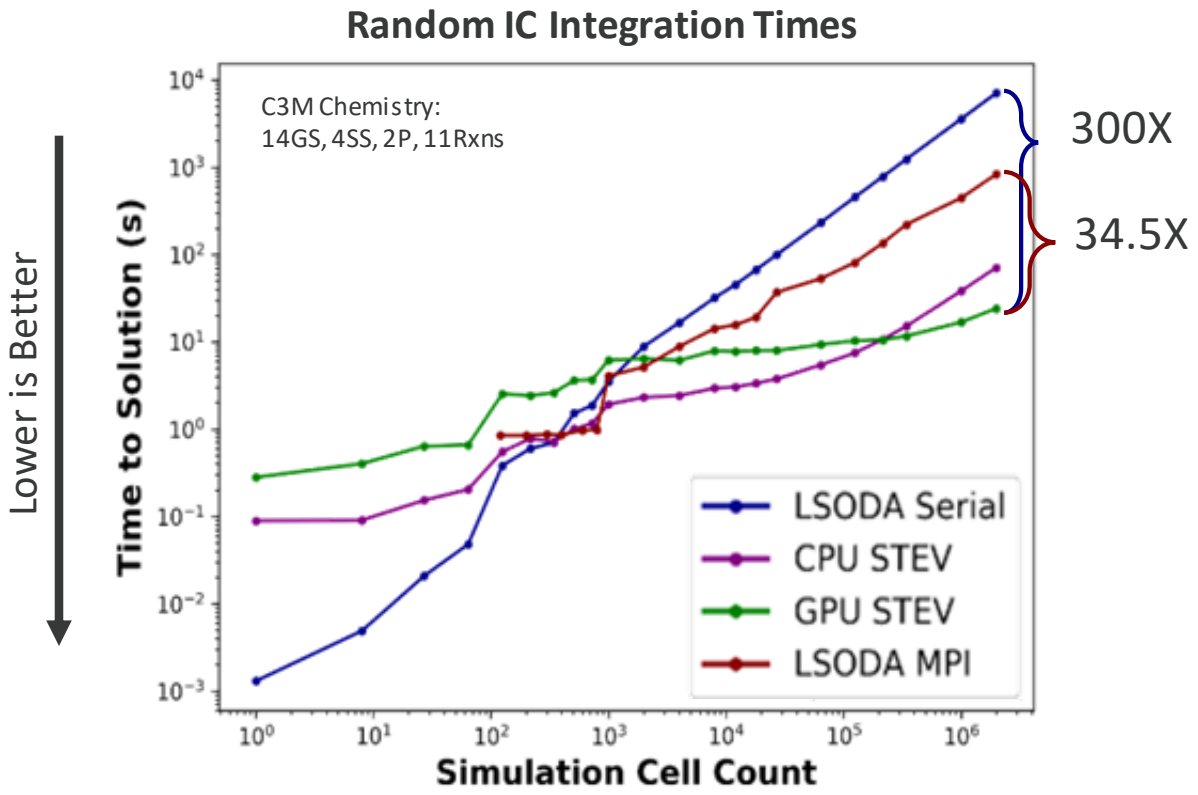
TensorFlow within HPC and AI/ML

- **TensorFlow For Scientific Computing**
  - TF is a tensor algebra library, not an AI/ML library
    - Memory management, communication, data operations, optimization, and parallelization handled "automatically"
    - Focus just on the algorithm!!
    - Written in Python and very similar to Numpy
    - Creates efficient JIT compiled code
    - Many tools to improve calculation efficiency (ex: tensorRT)
    - Easy to interface with existing compiled codes
    - Multi device/node operation is extremely easy
    - No more worries about rewriting code for specific hardware
    - Extremely large user and support base (largest AI/ML framework)
    - Always among the first to be supported by new hardware
  - A practical and easy "On Ramp" to the best hardware available

TensorFlow within HPC and AI/ML

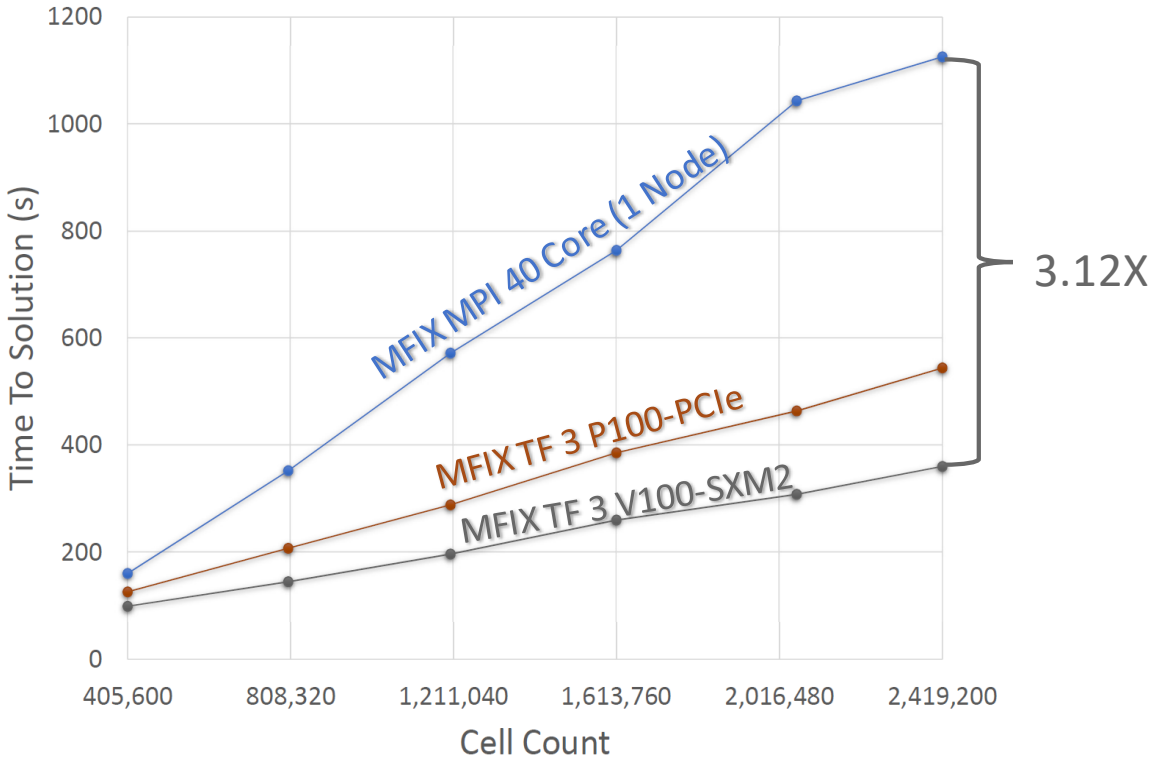
TensorFlow within HPC and AI/ML

MLA-STEV Solver



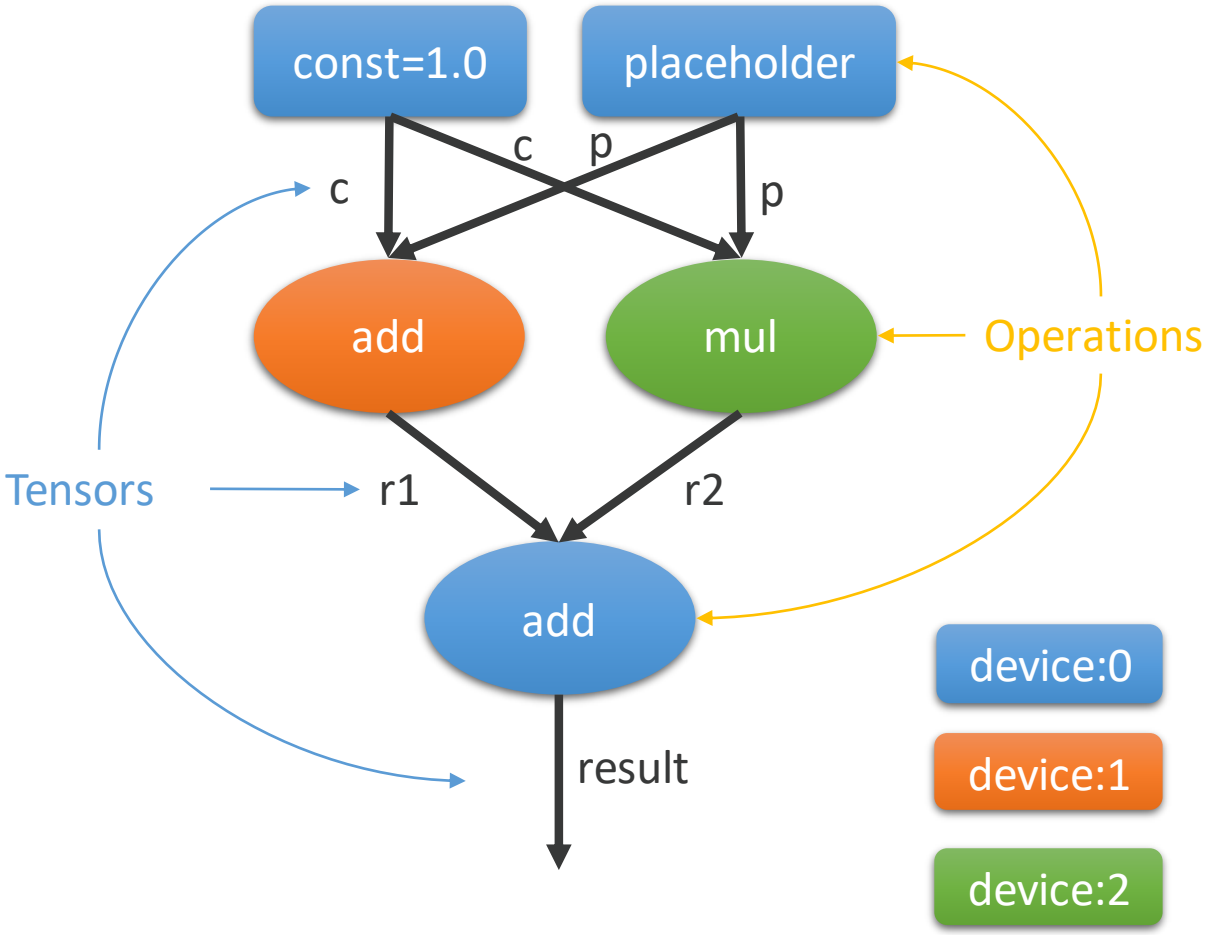
https://arxiv.org/abs/1905.09395

TensorFlow Fluid Solver



TensorFlow: Not what We are Used To

Static graph Computation Methodology




```python
import tensorflow as tf

with tf.device("device:0"):
  c = tf.constant(1.0, dtype=tf.float32)
  p = tf.placeholder(tf.float32, shape=(None,1))

with tf.device("device:1"):
  r1 = c + p

with tf.device("device:2"):
  r2 = c * p

with tf.device("device:0"):
  result = r1 + r2
```

```python
sess = tf.Session()
fd={p:[[2.0]]}
r = sess.run(result, feed_dict = fd)
```

TensorFlow: Not what We are Used To

Feeding Data In

NATIONAL ENERGY TECHNOLOGY LABORATORY

tf.constant

- Data defined at graph creation
- Allocated at session creation
- Immutable
- Shape fixed at graph creation
- Persistent until session closed

General Notes

- Not good for large data sets as data is saved in the graph file and it can make the session creation unmanageable and slow

tf.placeholder

- Data defined at run
- Sent in and allocated at run
- Immutable
- Shape not fixed until run, unless fully specified in graph
- Persistent during single run

General Notes

- Good for data placement
- Easy gate to pass data in from third party applications as they can be populated by reference

tf.Variable

- Allocated during session creation
- Populated from tensors during run
- Mutable
- Shape can be fixed or undefined
- Persistent until session closed

General Notes

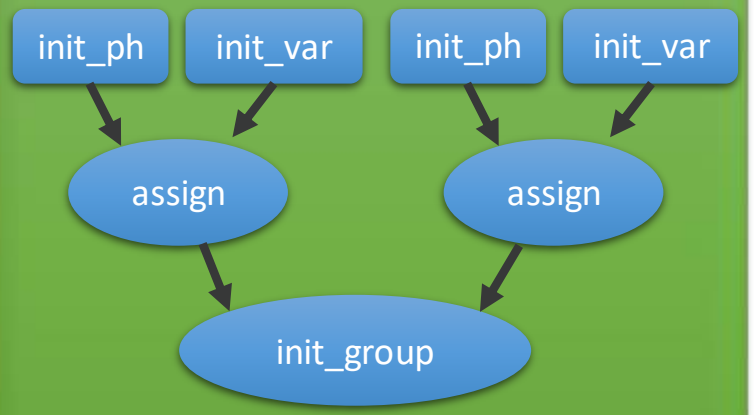
- Only object that can be populated at run time and persistent across runs
- Use heavily for “constant” and variable storage

TensorFlow: Computation Strategy

Break the problem up into sub-graphs which "talk" using variables

Initialization

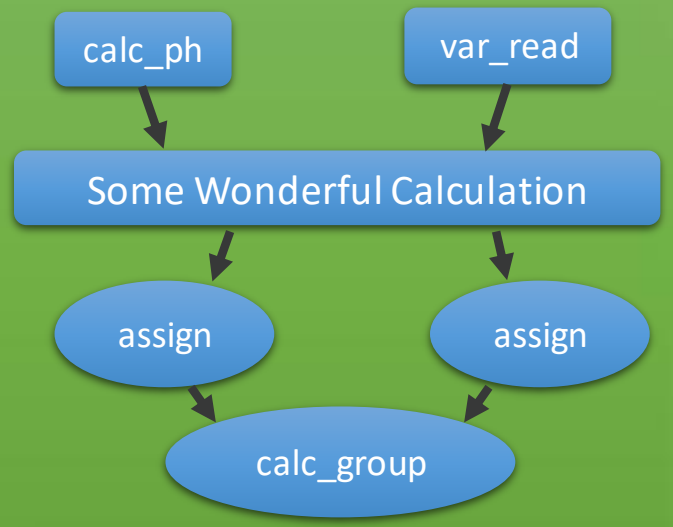
Initialize all data that needs to be persistent as variables




```
sess.run(init_group, feed_dict=init_feed)
```

Calculation

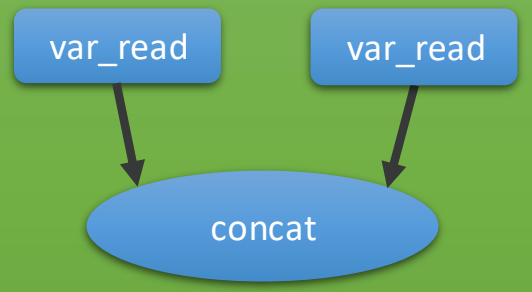
Calculate, but end in an assignment to a variable and call assign op(s)




```
sess.run(calc_group, feed_dict=calc_feed)
```

Data Retrieval

Use Data Retrieval as sparingly as possible, especially on accelerators




```
output = sess.run(concat)
```

```
outputList = sess.run([var1, var2])
```

TensorFlow: Interfacing With TensorFlow

Interfacing with TensorFlow from an Application

Application Code

1. Gather Application Data
2. Package into data type that can be understood by C
3. Call C function and pass data

C Interface

1. Package Data into python objects (dictionaries and/or numpy arrays)
2. Call python function

TF Python

Form feed dictionary if necessary

```
sess.run(calc_group, feed_dict=calc_feed)
output = sess.run(concat)
```

Return any Output

C Interface

1. Package Python data into data the Application can understand
2. Return Packaged Data

Application Code

1. Unpack data to Application
2. Continue Application execution

TensorFlow: Interfacing With TensorFlow

Calling an Application from TensorFlow: Example Calling AMGx

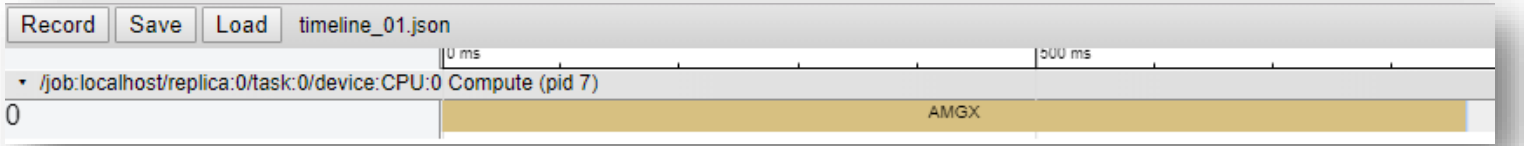
- **Make use of Custom Op**
  - Have to create graph operation to be able to make data calls at run time
  - Here we see an example of creating a sparse matrix and calling AMGx to solve the matrix with an initial guess of zeros and a source of unity

```python
import tensorflow as tf

amgx_module = tf.load_op_library('./amgx.so')
indices = tf.constant([[0,0,1],[0,1,2],[0,3,3],[0,8,4],[1,0,5],[1,1,6],[1,2,7],[1,3,8],
        [2,1,9],[2,2,10],[2,3,11],[2,4,12],[2,5,13],[3,0,14],[3,1,15],
        [3,2,16],[3,3,17],[3,4,18],[3,5,19],[3,8,20],[3,10,21],[4,2,22],
        [4,4,23],[4,5,24],[4,6,25],[5,2,26],[5,3,27],[5,4,28],[5,5,29],
        [5,6,30],[5,7,31],[5,10,32],[6,4,33],[6,5,34],[6,6,35],[6,7,36],
        [7,5,37],[7,6,38],[7,7,39],[7,9,40],[7,10,41],[8,0,42],[8,3,43],
        [8,8,44],[8,10,45],[8,11,46],[9,7,47],[9,9,48],[9,10,49],[9,11,50],
        [10,3,51],[10,5,52],[10,7,53],[10,8,54],[10,9,55],[10,10,56],
        [10,11,57],[11,8,58],[11,9,59],[11,10,60],[11,11,61]])

result = amgx_module.amgx(dimension=12, matrix=indices, config='/path/to/JACOBI.json')

sess = tf.Session()
sess.run(result)
```



TensorFlow: Interfacing With TensorFlow

Calling an Application from TensorFlow: Example Calling AMGx

Forming the .so library that is loaded in python

```
1   #include "tensorflow/core/framework/op.h"
2   #include "tensorflow/core/framework/shape_inference.h"
3   #include "tensorflow/core/framework/op_kernel.h"
4   #include "amgx_c.h"
5   #include <vector>
6
7   using namespace tensorflow;
8
9   REGISTER_OP("AMGX")
10      .Attr("dimension : int")
11      .Attr("config : string")
12      .Input("matrix: int32")
13      .Output("result: double");
14
15  class AMGXOp : public OpKernel {
16   public:
17    int dimension;
18    std::string config;
19    explicit AMGXOp(OpKernelConstruction* context) : OpKernel(context) {
20      OP_REQUIRES_OK(context, context->GetAttr("dimension", &dimension));
21      OP_REQUIRES_OK(context, context->GetAttr("config", &config));
22    }
```

```
24    void Compute(OpKernelContext* context) override {
25      // Grab the matrix
26      const Tensor& input_tensor = context->input(0);
27      auto input = input_tensor.flat<int32>();
28
29      //Grab the result
30      Tensor* output_tensor = NULL;
31      OP_REQUIRES_OK(context, context->allocate_output(0, {dimension,1}, &output_tensor));
32      auto output_flat = output_tensor->flat<double>();
33
34      //AMGX - variables
35      AMGX_Mode mode;
36      AMGX_config_handle cfg;
37      AMGX_resources_handle rsrc;
```

```
120     //AMGX - shutdown and exit
121     AMGX_SAFE_CALL(AMGX_finalize_plugins());
122     AMGX_SAFE_CALL(AMGX_finalize());
123
124     //assign result
125     for (int i = 0; i < dimension; i++) {
126       output_flat(i) = xdata[i];
127     }
128   }
129 };
130
131 REGISTER_KERNEL_BUILDER(Name("AMGX").Device(DEVICE_CPU), AMGXOp);
```

https://www.tensorflow.org/guide/create_op

TensorFlow: Incorporating Trained Models

Make your workflow smart with DNN's

- **Very Easy ML Integration**
  - Load Trained Model
  - Connect Inputs and outputs
  - Done
  - Example: Specific Heat Calcs
    - Almost twice as fast as evaluating polynomials (even without tensor cores)
    - All species properties predicted at once with two matrix multiplications

```
# read protobuf binary for a trained network into a graph_def
allSpCpnn = os.path.join(dataDir, 'Cp_all.pb')
with tf.gfile.GFile(allSpCpnn, "rb") as f:
    graph_def = tf.GraphDef()
    graph_def.ParseFromString(f.read())

# import graph_def to main graph, connect inputs and outputs
all_cp=tf.import_graph_def(graph_def, input_map={'Temp_input:0':Temperature},
                           return_elements=['final_output:0'],
                           name='cp_nn_0')[0]
```

TensorFlow: Pitfalls

Dataflow vs Sequential Programming

How Many Additions Performed?

Dataflow Programming

```python
import tensorflow as tf

constant = [tf.constant(i) for i in range(50)]

output = []

for c in constant:
    output.append(c+1)

sess = tf.Session()

print(sess.run(output[25]))
```

Sequential Programming

```python
constant = [i for i in range(50)]

output = []

for c in constant:
    output.append(c+1)

print(c[25])
```

TensorFlow: Pitfalls

Dataflow vs Sequential Programming

- **Avoiding Undetermined Behavior with Variables**
  - Variables do not live in graph, read ops and assign ops can if done correctly
  - Avoid multiple assign statements per run
  - Can result in undetermined behavior
  - Do calculation, end in assignment, call assignment, call var to retrieve

Undetermined Behavior

```python
import tensorflow as tf

var = tf.Variable(tf.constant(1.0))

tf.assign(var, var+1.0)

output = var*5.0

sess = tf.Session()

print(sess.run(output))
```

Determined Behavior

```python
import tensorflow as tf

var = tf.Variable(tf.constant(1.0))

output = (var+1.0)*5.0

asignment_op = tf.assign(var, output)

sess = tf.Session()

sess.run(asignment_op)
print(sess.run(var))
```

TensorFlow: Pitfalls

Conditionals

- **Use Truth Tables and vector merging whenever possible**
  - Whenever possible precalculate/store truth tables that represent traditional logic in loops
  - Use tf.where or z = TT*x+(1.0 – TT)*y
  - NAN's can only be eliminated with tf.where
  - Used heavily in forming coefficient matrices for linear solver
  - We favor storing TT as float32's on GPU's of 0.0 and 1.0 values and using math merge
  - Very much want to see binary variable support on GPU's

Custom Vector Merge

```
def our_where(tt, *args, **kwargs):
    aDType = args[0].dtype
    one = tf.constant(1,dtype=aDType)
    tt_c = tt
    if tt.dtype != aDType:
        tt_c = tf.cast(tt, aDType)
    if kwargs.pop('useMul', False) and len(args) >= 2:
        return tt_c*args[0] + (one - tt_c)*args[1]
    if kwargs.pop('zeros1', False):
        return (one - tt_c)*args[1]
    if kwargs.pop('zeros2', False):
        return tt_c*args[0]
    if tt.dtype != tf.bool:
        tt = tf.cast(tt, 'bool')
    return tf.where(tt,*args, **kwargs)
```

Only sure way to stop NAN propagation

```
NAN_free = our_where(tf.is_nan(data), tf.zeros_like(data), data)
```

TensorFlow: Pitfalls

Debugging

- **Debugging is Complicated by run time evaluation**
  - For simple graphs use TF Command Line Debugger
    - Can be used when launching a program which calls into TF
    - Tends to hang with complex graphs with while loops
    - Search for "TensorFlow Debugger Screencast" in YouTube
  - End in Var Assignment
    - Create a debug flag which creates extra variables
    - Assign tensors into vars within code and keep a debug_assign list of assign ops
    - Concat normal assign list with debug_assign using debug flag
    - Retrieve debug variables with separate run call after run assign call
  - Make use of tf.debugging assert statements
    - Create separate debug flag for turning on and off assertions as they create extra computational load
    - Keep in mind that order is not based on placement in code so it is usually not possible to guarantee which assert statement is triggered first
    - Make sure to return the tensor and use it in following calcs to ensure it is placed in graph
  - tf.print
    - Useful for the quick and dirty
    - Clogs up screen
    - Cant use many at same time

```python
import tensorflow as tf

DEBUG_VARS = True
DEBUG_ASSERT = True
finite_check = tf.debugging.assert_all_finite

var = tf.Variable(tf.constant(1.0))
if DEBUG_VARS:
    var_dbg = tf.Variable(tf.constant(1.0))

inpt_ph = tf.placeholder(tf.float32, shape=(None,))
if DEBUG_ASSERT: inpt_ph = finite_check(inpt_ph, 'inpt check')

num = inpt_ph+1.0
denom = inpt_ph-1.0
output = num/denom

assign_list = [tf.assign(var, output)]
if DEBUG_VARS: assign_list += [tf.assign(var_dbg, denom)]
assign_group = tf.group(assign_list)

sess = tf.Session()

sess.run(assign_group)
print(sess.run(var, feed_dict={inpt_ph:some_data}))
if DEBUG_VARS: print(sess.run(var_dbg, feed_dict={inpt_ph:some_data}))
```

Acknowledgements

A BIG thankyou!

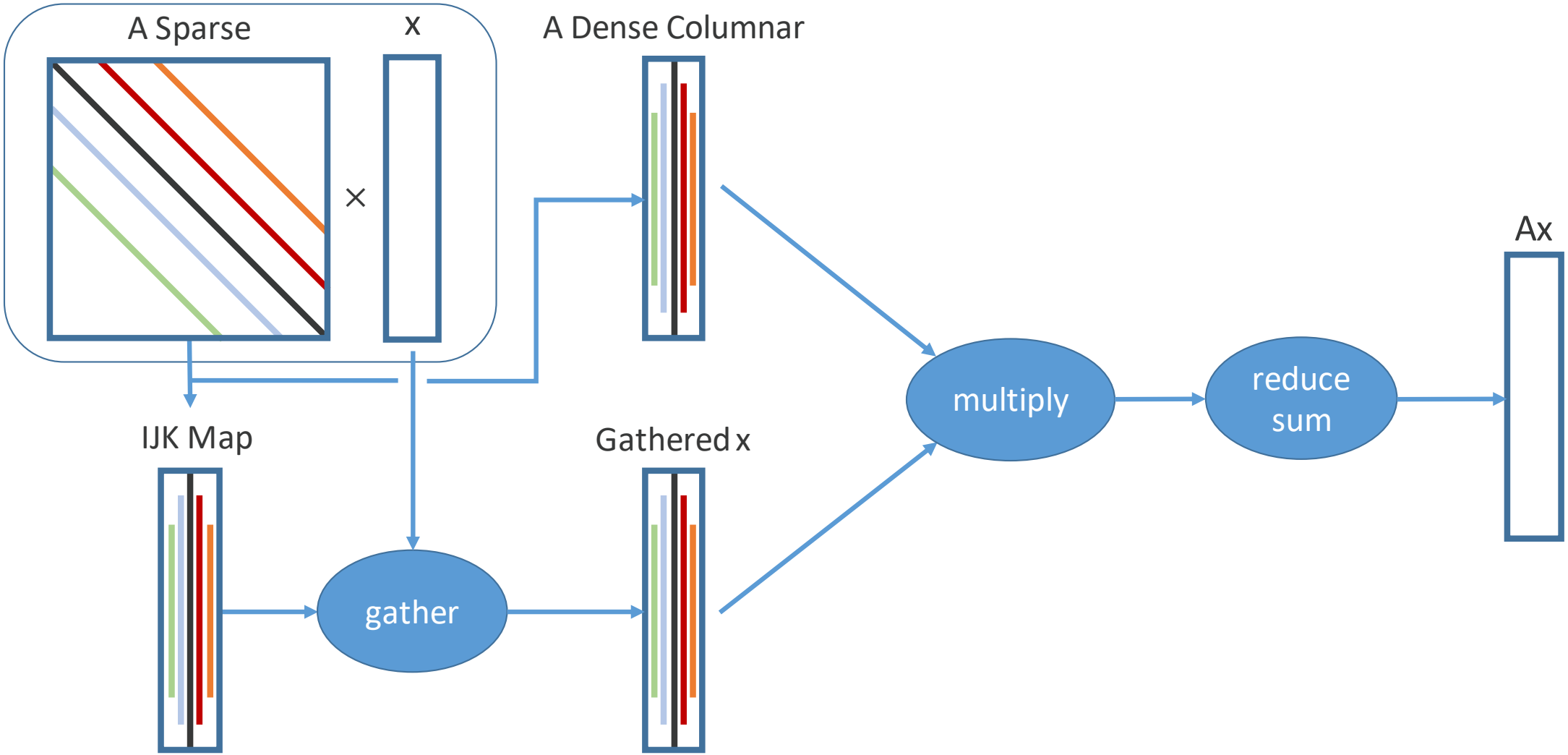
Jim Hooks, Craig Tierney, Griffin Lacey, and Stan Posey
Technical Guidance, Advice, and Consulting
The Generous Invitation to Participate in GTC

Art Mann and Eliot Eshelman
Providing Access to a DGX-1 for Development and Benchmarking

Questions??

TensorFlow: Linear Solver

N-diagonal, Sparse, (Semi)Structured CFD Linear Solver



TensorFlow: Linear Solver

N-diagonal, Sparse, (Semi)Structured CFD Linear Solver

```
def bicgstab(A_list, b, x0, indices, tol, maxIters=5):
    A_max = tf.abs(A_list[3])
    A_norm = [tf.divide(A, A_max) for A in A_list]
    b_norm = tf.divide(b, A_max)

    x0g = tf.gather(x0, indices)
    A_cc = tf.concat([aCol[:,None] for aCol in A_norm], axis=1)
    r0 = b_norm - tf.reduce_sum(A_cc*x0g, axis=1)

    Rnorm0 = tf.sqrt(vDot(r0,r0))
    tolRnorm0 = tf.multiply(tol,Rnorm0)

    def body(*args):
        x,p,r,_ = args
        x_n, p_n, r_n = bicgstab_main(A_cc, indices, x, r, r0, p)
        normR = tf.sqrt(vDot(r_n,r_n))
        return [x_n, p_n, r_n, normR]

    def conditional(*args):
        return tf.greater_equal(args[3], tolRnorm0)

    while_init = [x0, r0, r0, tf.constant(1e6, dtype=x0.dtype)]
    result = tf.while_loop(conditional, body, while_init,
                           maximum_iterations=maxIters)
    return result[0]
```

```
def vDot(a,b):
    return tf.reduce_sum(a*b)

def bicgstab_main(A_cc, indices_cat, x, r, r0, p):
    pg = tf.gather(p, indices_cat)
    Ap = tf.reduce_sum(A_cc*pg, axis=1)
    alpha = vDot(r,r0) / vDot(Ap,r0)
    s = r - alpha * Ap
    sg = tf.gather(s, indices_cat)
    As = tf.reduce_sum(A_cc*sg, axis=1)
    omega = vDot(As,s) / vDot(As,As)
    alphap = alpha*p
    omegas = omega*s
    omegaAs = omega*As
    omegaAp = omega*Ap
    x_n = tf.add_n([x, alphap, omegas])
    r_n = s - omegaAs
    beta = alpha/omega*(vDot(r_n, r0)/vDot(r,r0))
    p_n = r_n + beta*(p - omegaAp)
    return x_n, p_n, r_n
```

TensorFlow: Distributed Computing

Septadiagonal Distributed Matrix Multiplication

```python
def distr_mul(A_list, X_list, IJK_list, n_ghost_list):
    # initialize storage lists and detect devices
    Product_list=[]
    Xp_list = []
    devices = [x.device for x in X_list]

    # Extract vector pieces related to ghost cells and store in list
    for i,dev in enumerate(devices):
        with tf.device(dev):
            n_ghost = n_ghost_list[i]
            x_size = tf.size(X_list[i], out_type='int64')
            if i== 0:
                splitList_next = [x_size-n_ghost[1], n_ghost[1]]
                _, Xp_next = tf.split(X_list[i], splitList_next,axis=0,
                                      num=2, name='Xp_0_n')
                Xp_prev = []
            elif i==len(devices)-1:
                splitList_prev = [n_ghost[0], x_size-n_ghost[0]]
                Xp_prev,_ = tf.split(X_list[i], splitList_prev, axis=0,
                                     num=2, name='Xp_'+str(i)+'_p')
                Xp_next = []
            else:
                splitList = [n_ghost[0], x_size-n_ghost[1]-n_ghost[0], n_ghost[1]]
                Xp_prev, _, Xp_next = tf.split(X_list[i], splitList,axis=0,
                                               num=3, name='Xp_'+str(i)+'_np')
            Xp_list.append([Xp_prev, Xp_next])
```

```python
    # Concatenate to form full vector on each device, gather Indicies, do matmul
    for i,dev in enumerate(devices):
        with tf.device(dev):
            if i== 0:
                X_feg = tf.concat((X_list[i], Xp_list[1][0]),0)
            elif i==len(devices)-1:
                X_feg = tf.concat((Xp_list[i-1][1], X_list[i]),0)
            else:
                X_feg = tf.concat((Xp_list[i-1][1], X_list[i], Xp_list[i+1][0]),0)
            X_g = tf.gather(X_feg, IJK_list[i], name = 'Xg_gather_'+str(i))
            sept_mul = tf.multiply(A_list[i],X_g, name='P_mult_'+str(i))
            Prod_i = tf.reduce_sum(sept_mul, axis=1, name='P_reduce_sum_'+str(i))
            Product_list.append(Prod_i)
    return Product_list
```

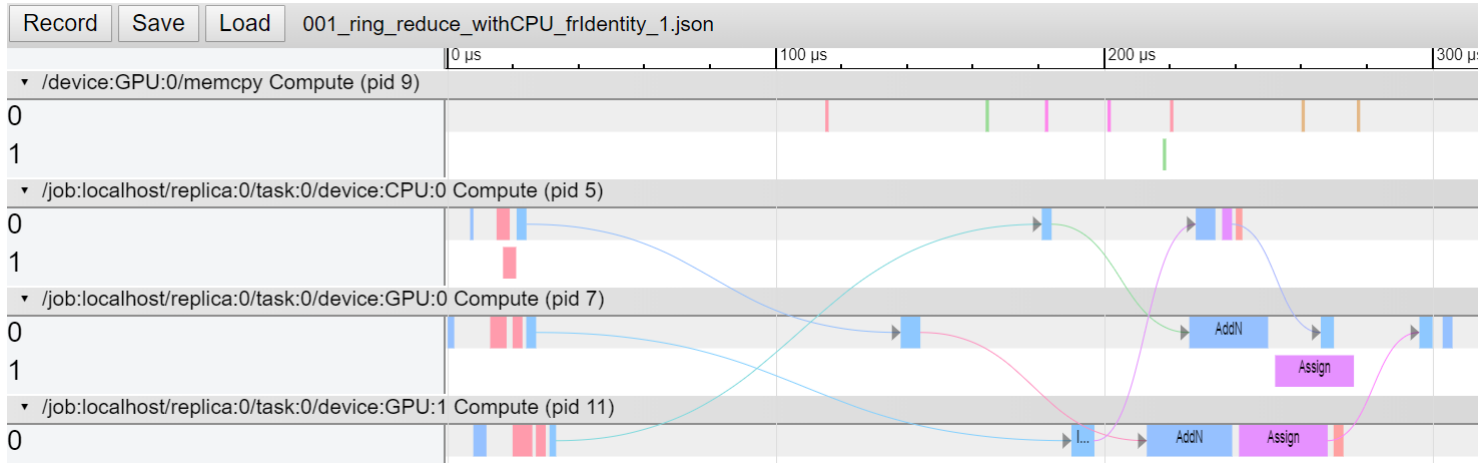
TensorFlow: Distributed Computing

Ring Reduction for Distributed Dot Products in Native TensorFlow

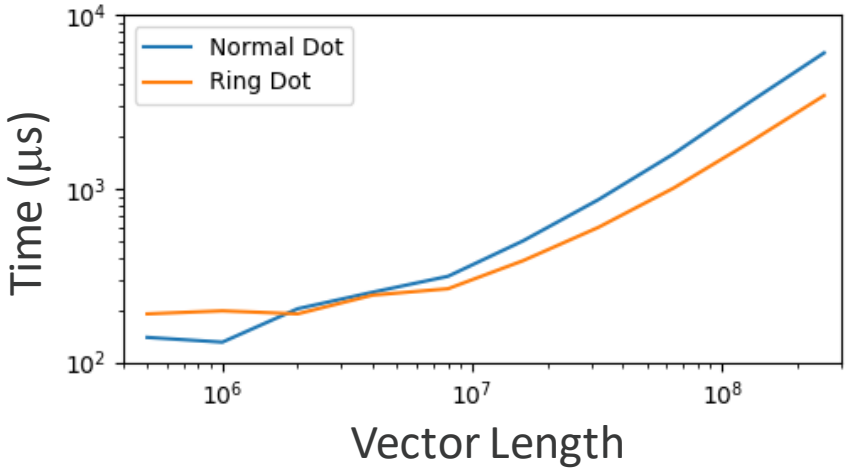
- **Ring Reduction**
  - Eliminates Communication Pairs
  - Distributed dot product calcs
  - Example
    - Average Run time ~200 µs
    - Vector Length: 6 – float32
    - Representative of reduction in iBICGSTAB method

```python
def ring_reduce(dataToReduce, tfOp, *args, **kwargs):
    dlist = [x.device for x in dataToReduce]
    reduce_list = [[x] for x in dataToReduce]
    reducedData = []
    for j in range(len(dataToReduce)-1):
        for i in range(len(dataToReduce)):
            with tf.device(dlist[i]):
                ldataCpy = tf.identity(reduce_list[i-1][j], name='lnd_'+str(j)+'_'+str(i))
                reduce_list[i].append(ldataCpy)
    for i,dev in enumerate(dlist):
        with tf.device(dev):
            reducedData.append(tfOp(reduce_list[i], *args, **kwargs))
    return reducedData
```

3 Device Trace: CPU-GPU-GPU

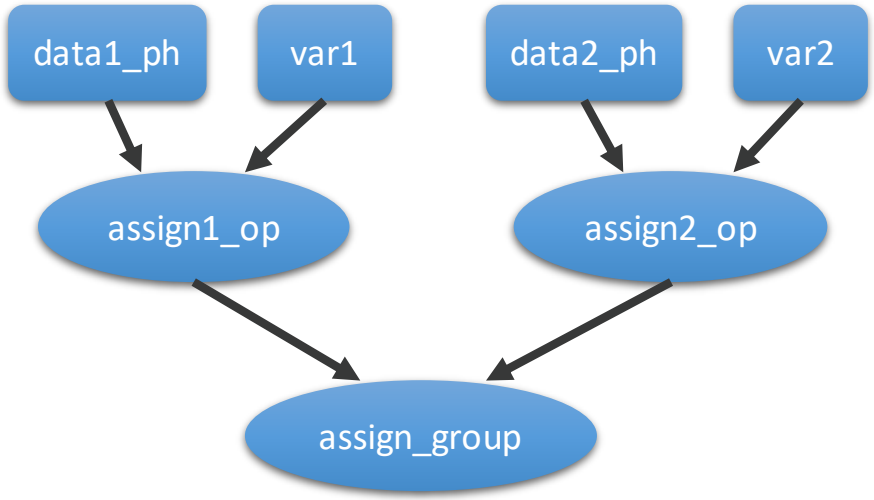


GPU-GPU Dot Timings



TensorFlow: Populating Variables

Storing data in variables without knowing shape




```python
import tensorflow as tf

tf.compat.v1.disable_eager_execution()

#create simple graph
data1_ph = tf.compat.v1.placeholder(dtype=tf.float32, shape=(None, None))
data2_ph = tf.compat.v1.placeholder(dtype=tf.float32, shape=())
var1 = tf.Variable(initial_value=data1_ph)
var2 = tf.Variable(initial_value=data2_ph)
assign1_op = var1.assign(data1_ph)
assign2_op = var2.assign(data2_ph)
assign_group = tf.group([assign1_op, assign2_op])

#create session
sess = tf.compat.v1.Session()
#run the assignment op
sess.run(assign_group, feed_dict={data1_ph:[[3.0,4.0,5.0]], data2_ph:6.0})
#retrieve and print the variable
print('vars: ', sess.run([var1, var2]))
```

```
vars:  [array([[3., 4., 5.]], dtype=float32), 6.0]
>
```