Understanding the Design Space of Sparse/Dense Multiphase Dataflows for Mapping Graph Neural Networks on Spatial Accelerators

Raveesh Garg

*This work is accepted for publication to IPDPS 2022. Preprint- https://arxiv.org/abs/2103.07977

Acknowledgements

Tushar Krishna

Eric Qin

Sivasankaran Rajamanickam

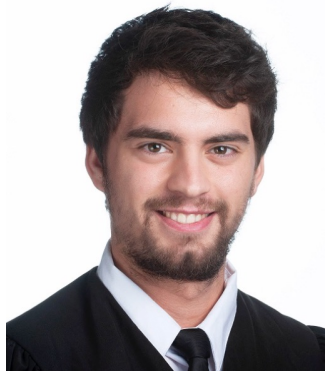
Francisco Muñoz-Martínez

José L. Abellán

Manuel E. Acacio

Sergi Abadal

Eduard Alarcón



Robert Guirado

Akshay Jain

Design-space of Dataflows

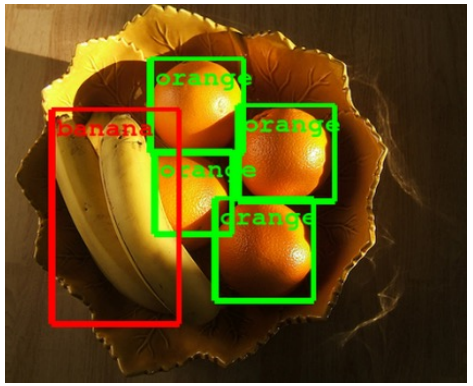
- **Complex design-space** with million possible dataflows.
- Prior works have explored the space of dataflows and Mappings for **DNNs** (CONVs and Matmul) (Eg. Timeloop (ISPASS 2020), MAESTRO (MICRO 2019), Interstellar(ASPLOS 2020)).
- This goal of this work is to navigate the **design-space of dataflows for Graph Neural Networks**.

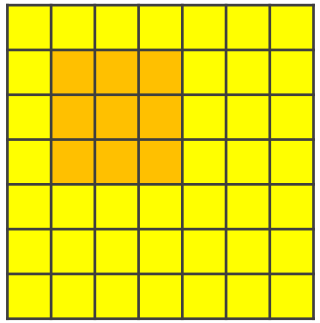
Outline

- Graph Neural Networks (GNNs)
- GNN Dataflows – Taxonomy and modeling
- Hands-on Exercises

Learning on Irregular Data

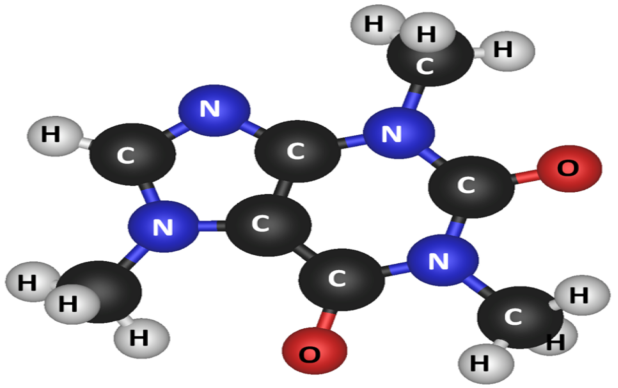
Regular Data





Irregular Data

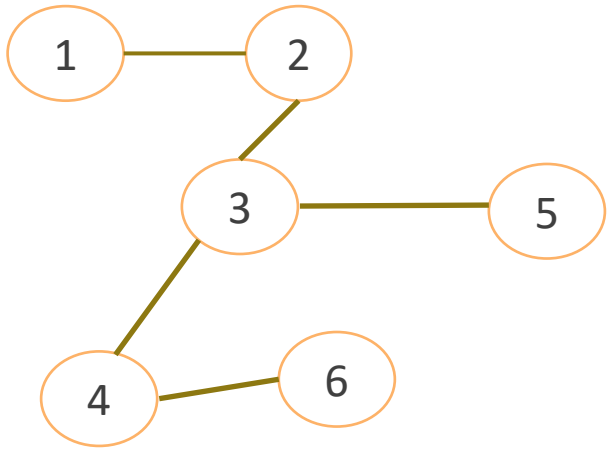
Social Networks



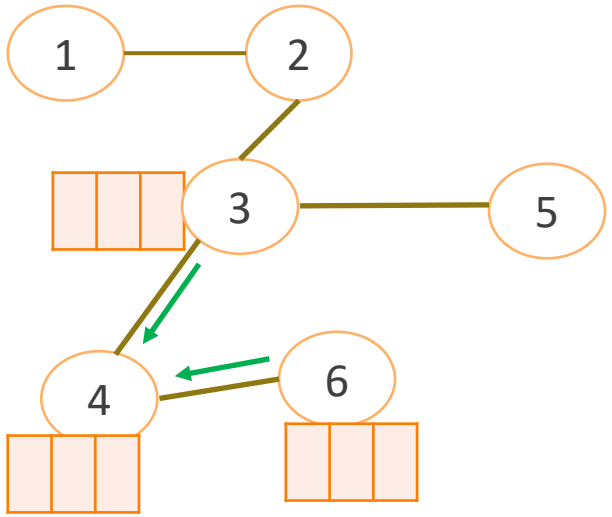
Biochemistry

Graph Neural Networks

Graph

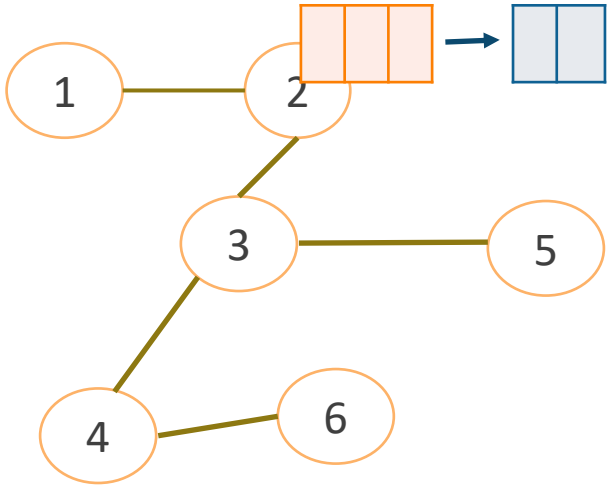


Aggregation phase



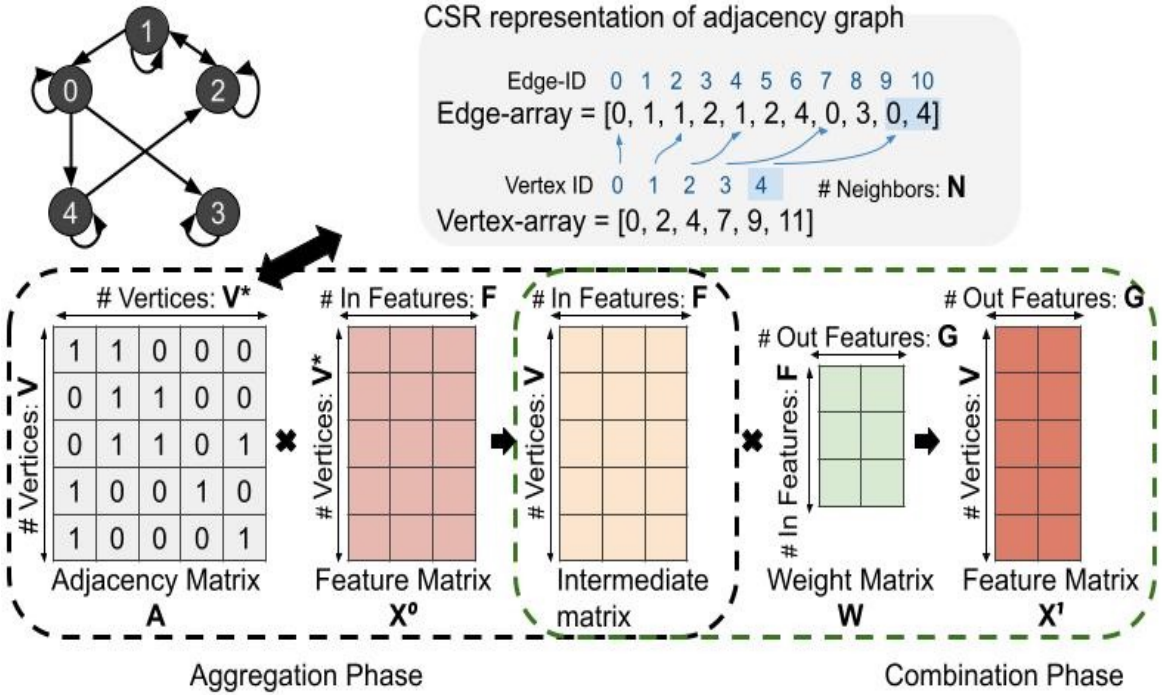
Aggregation of neighboring feature vectors, for example, **v4=v3+v6**

Combination phase



This is a feature transformation phase feature vector of each vertex is transformed using trained weights similar to NN.

GNN Computations



Graphs can also be represented as adjacency matrix with non-zeros representing the connection as shown.

Thus, Aggregation phase becomes an SpMM problem and Combination involves DenseGEMMs

V: Vertices; N: Neighbors; F: Input Features, G: Output Features

Key characteristics of GNNs

- Aggregation and Combination phases
- Interplay of sparse and dense computations.
- Reuse opportunities between the two phases.
- Squares the design-space.
- Complicates it further due to dependency between the phases.

Key characteristics of GNNs

- Aggregation and Combination phases
- Interplay of sparse and dense computations

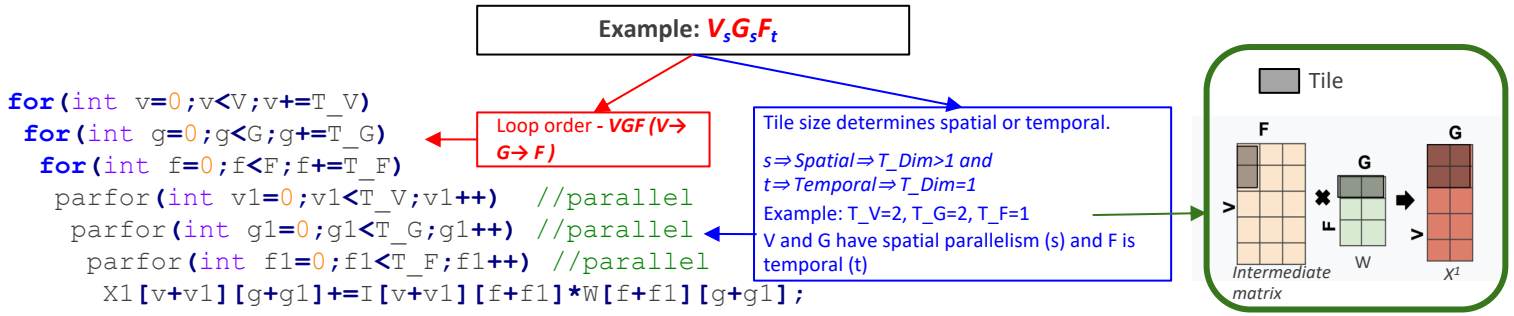
GNNs offer reuse opportunities between two phases which makes the design-space of dataflows and mappings more complex.

Outline

- Graph Neural Networks (GNNs)
- GNN Dataflows – Taxonomy and modeling
- Hands-on Exercises

Intra-phase dataflows

V: Vertices; N: Neighbors; F: Input Features, G: Output Features



- Intra-phase dataflows represent dataflows within a phase
- As discussed before, individual dataflows consist of loop orders (VGF) and parallelization strategies (V and G parallel)

Inter-phase Dataflows

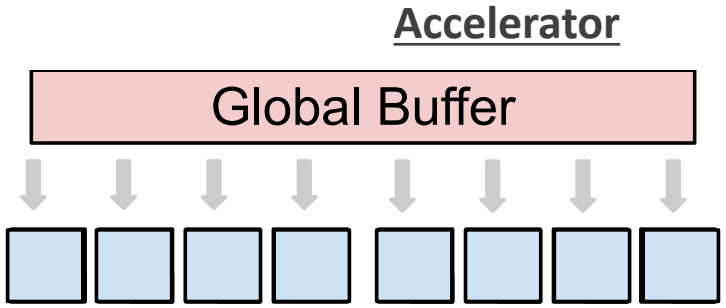
Inter-phase dataflows describe the data reuse strategies between the two phases. We classify them into 3 categories

- Sequential (SEQ)
- Sequential Pipeline (SP)
- Parallel Pipeline (PP)

Execution of SEQ dataflows

PE running Aggregation

PE running Combination

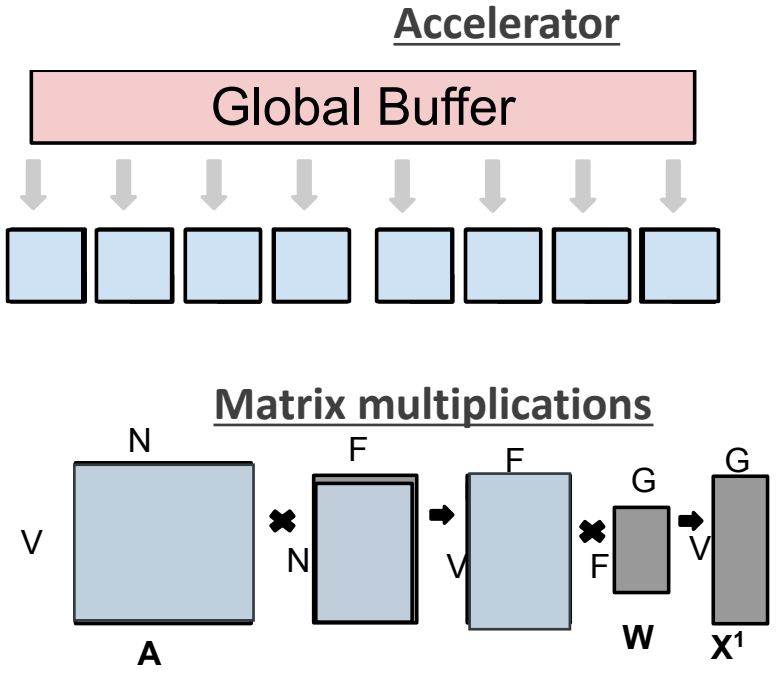


Execution of SEQ dataflows

V: Vertices; N: Neighbors; F: Input Features, G: Output Features

PE running Aggregation

PE running Combination

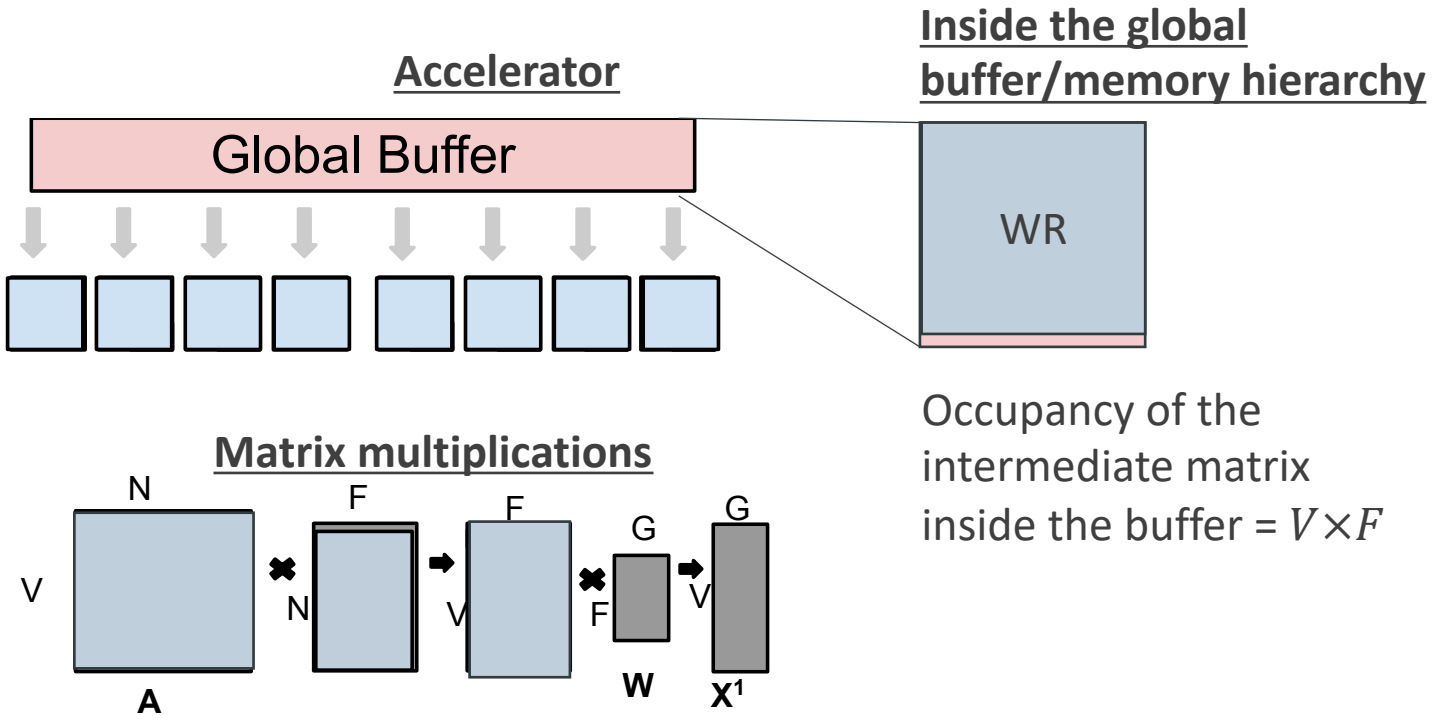


Execution of SEQ dataflows

V: Vertices; N: Neighbors; F: Input Features, G: Output Features

PE running Aggregation

PE running Combination

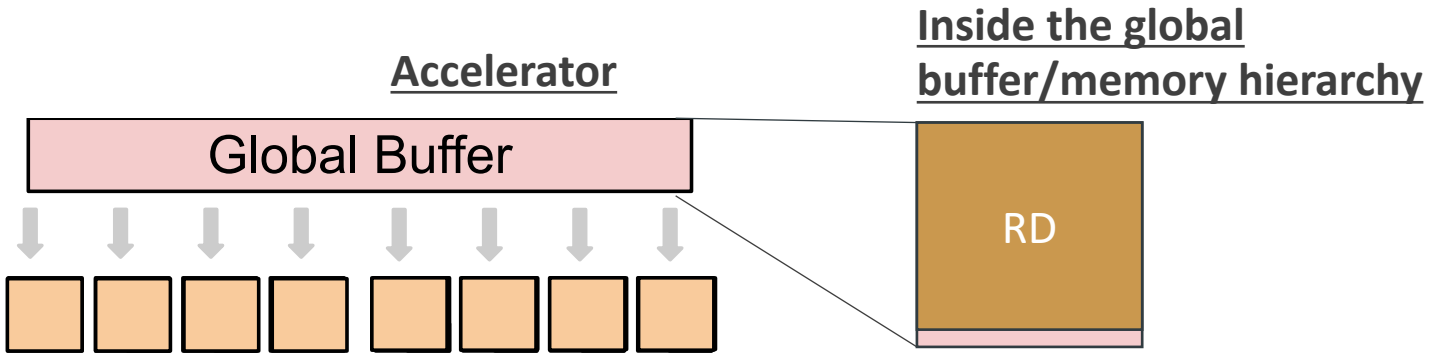


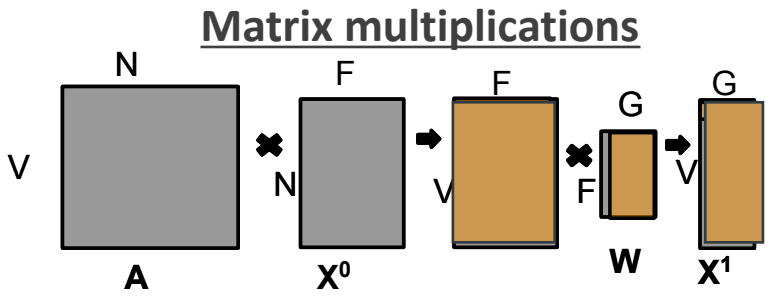
Execution of SEQ dataflow

V: Vertices; N: Neighbors; F: Input Features, G: Output Features

PE running Aggregation

PE running Combination



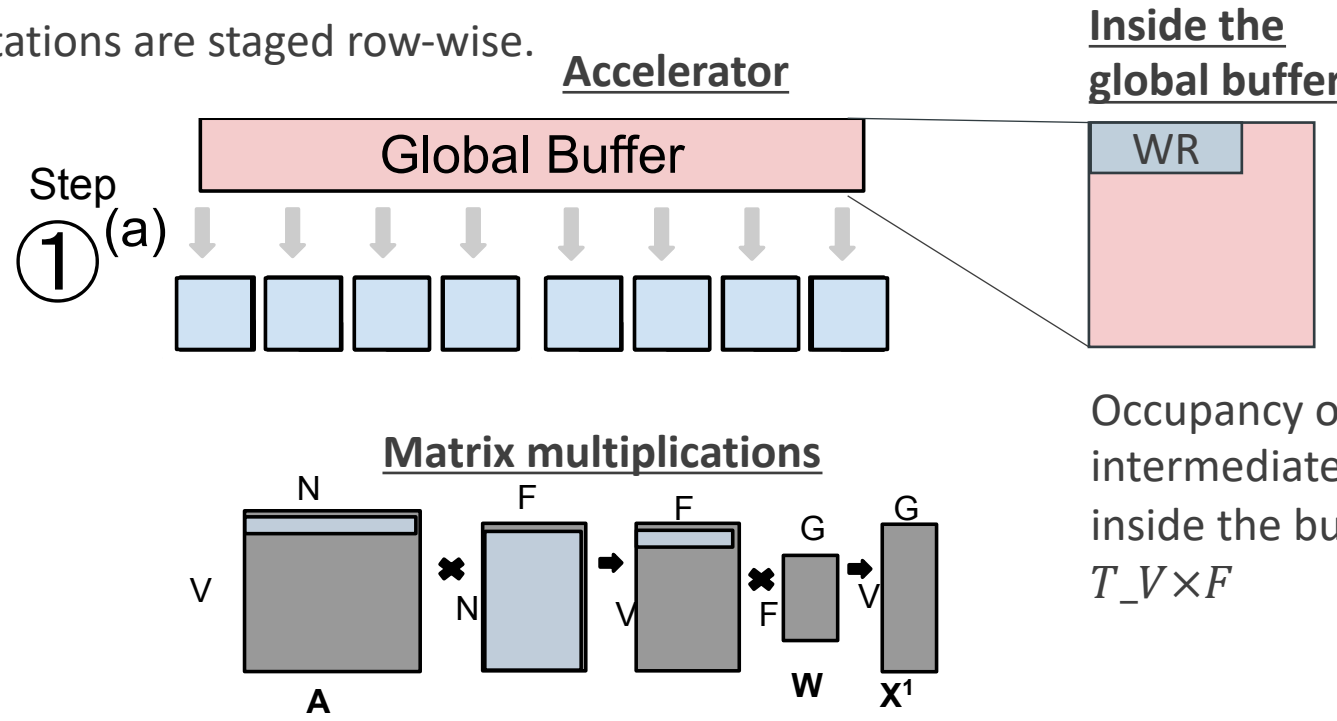


Occupancy of the intermediate matrix inside the buffer = $V \times F$

Execution example of SP dataflow

V: Vertices; N: Neighbors; F: Input Features, G: Output Features

Assuming that the computations are staged row-wise.

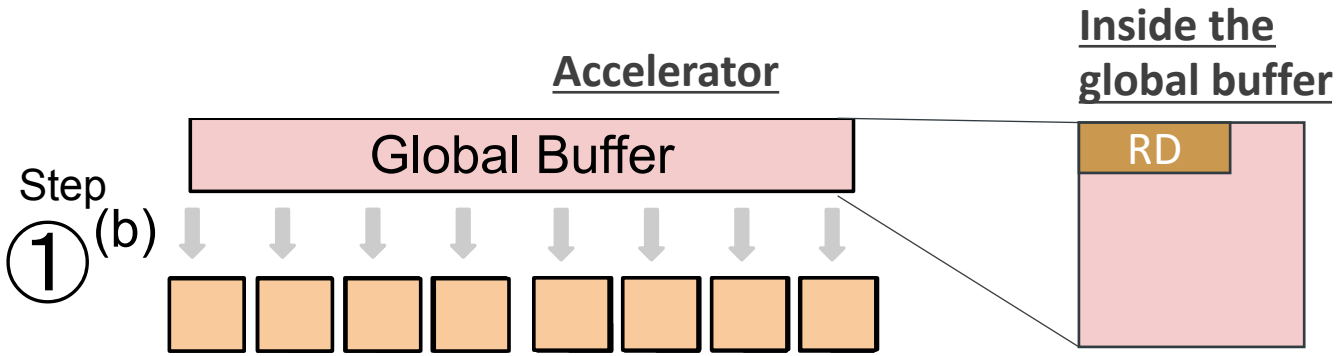


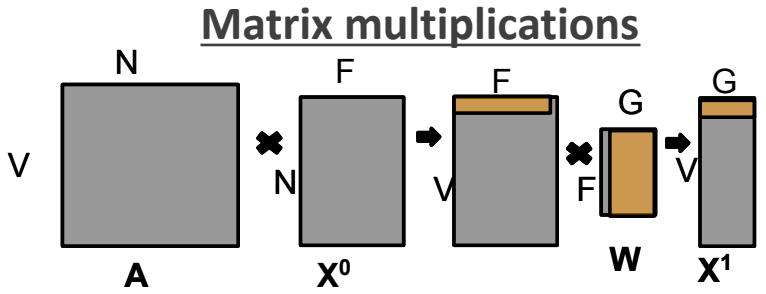
Occupancy of the intermediate matrix inside the buffer = $T\_V \times F$

Here, step refers to the duration in which producer writes a part of the intermediate matrix and consumer reads a part of the intermediate matrix

Execution example of SP dataflow

V: Vertices; N: Neighbors; F: Input Features, G: Output Features

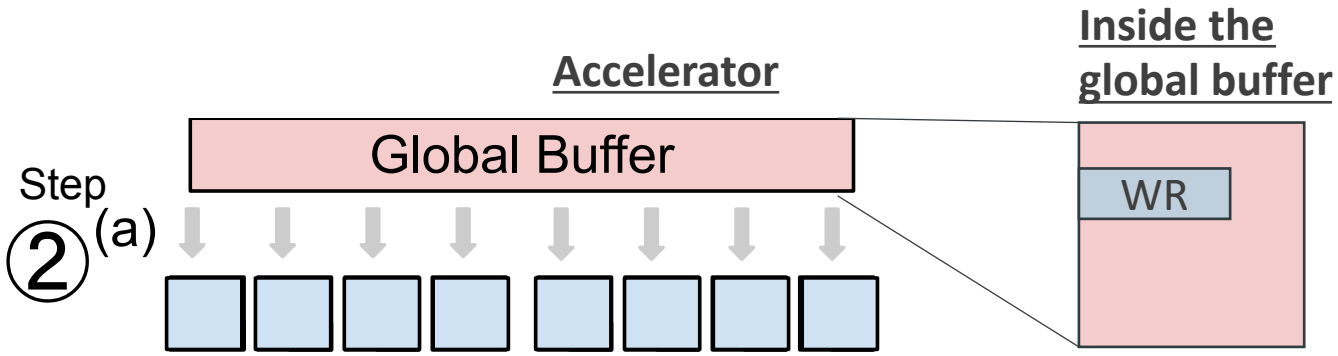


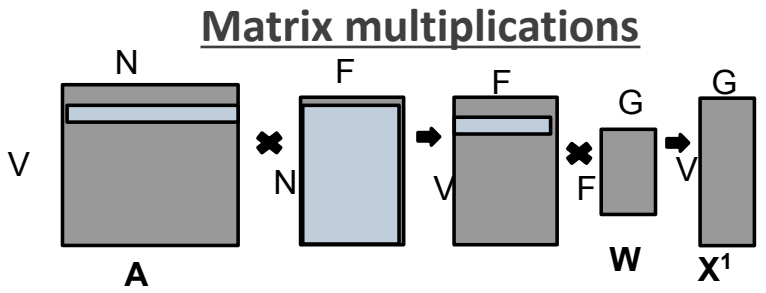


Occupancy of the intermediate matrix inside the buffer = $T\_V \times F$

Execution example of SP dataflow

V: Vertices; N: Neighbors; F: Input Features, G: Output Features

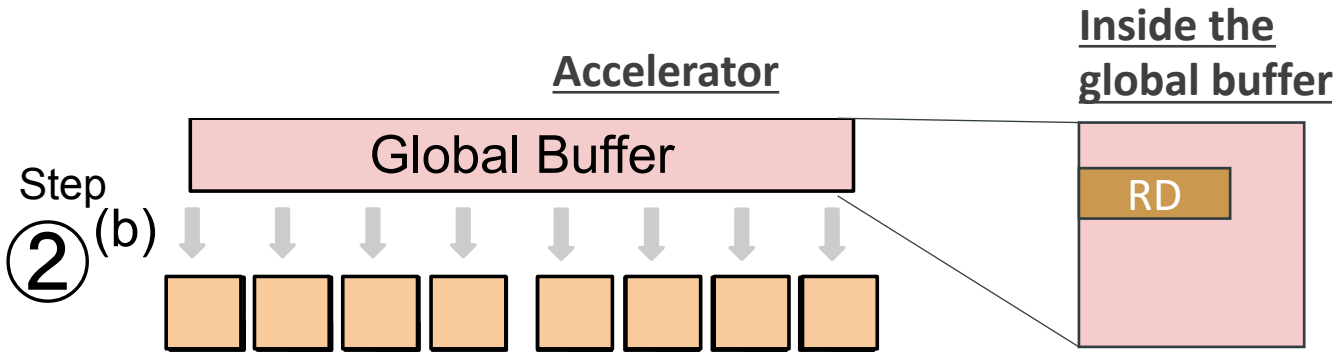


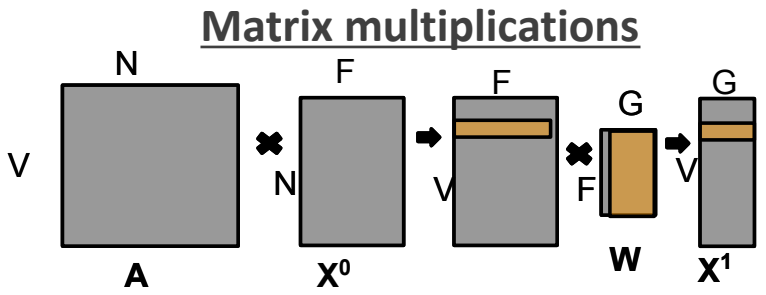


Occupancy of the intermediate matrix inside the buffer = $T\_V \times F$

Execution example of SP dataflow

V: Vertices; N: Neighbors; F: Input Features, G: Output Features



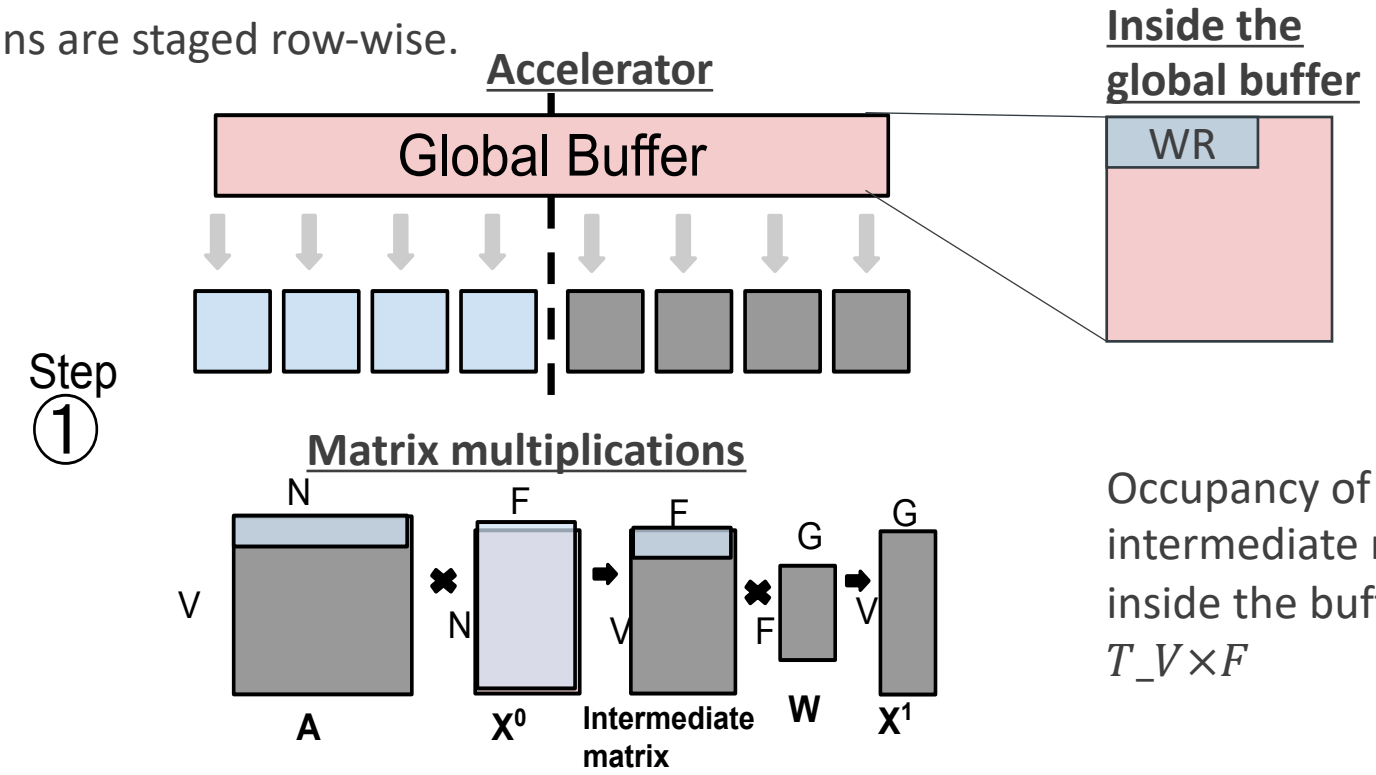


Occupancy of the intermediate matrix inside the buffer = $T\_V \times F$

Execution example of PP dataflow

V: Vertices; N: Neighbors; F: Input Features, G: Output Features

Assuming that the computations are staged row-wise.

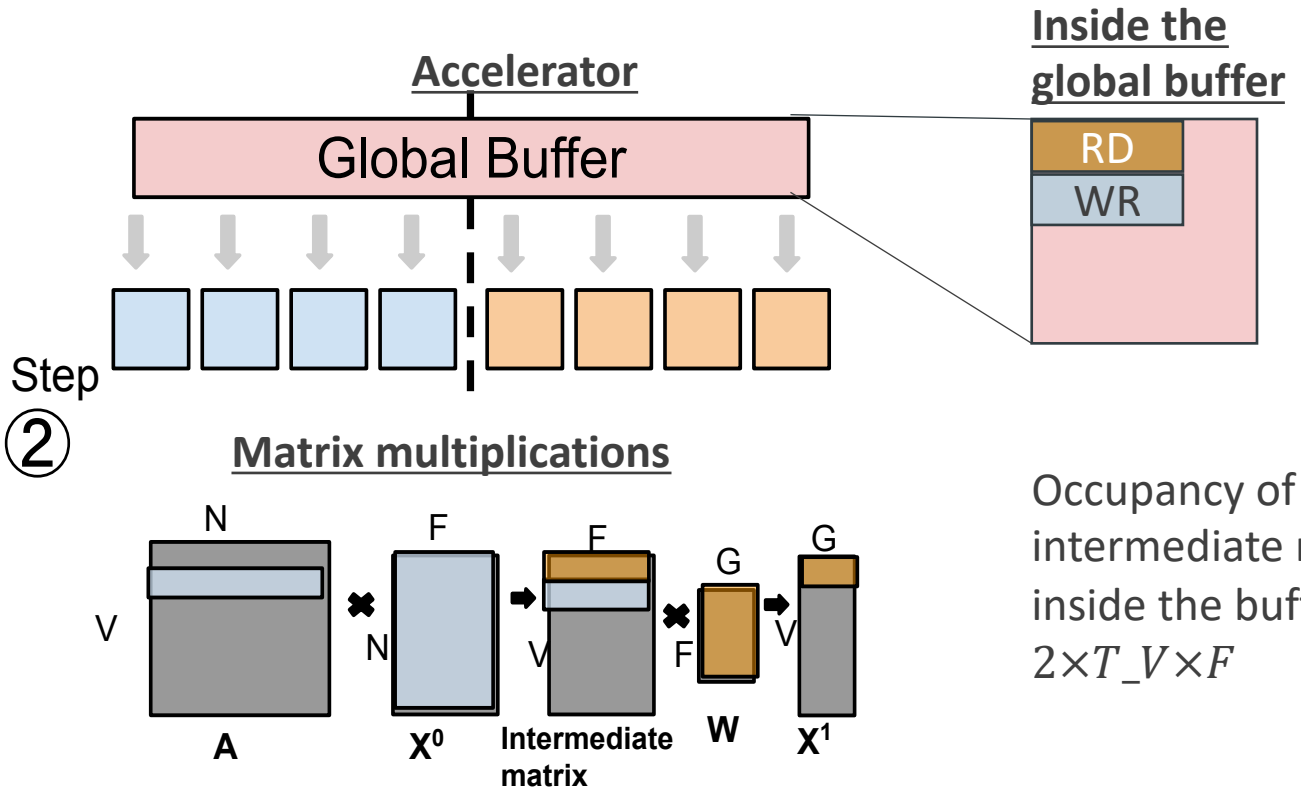


Occupancy of the intermediate matrix inside the buffer = $T\_V \times F$

Here, step refers to the duration in which producer writes a part of the intermediate matrix and consumer reads a part of the intermediate matrix

Execution example of PP dataflow

V: Vertices; N: Neighbors; F: Input Features, G: Output Features

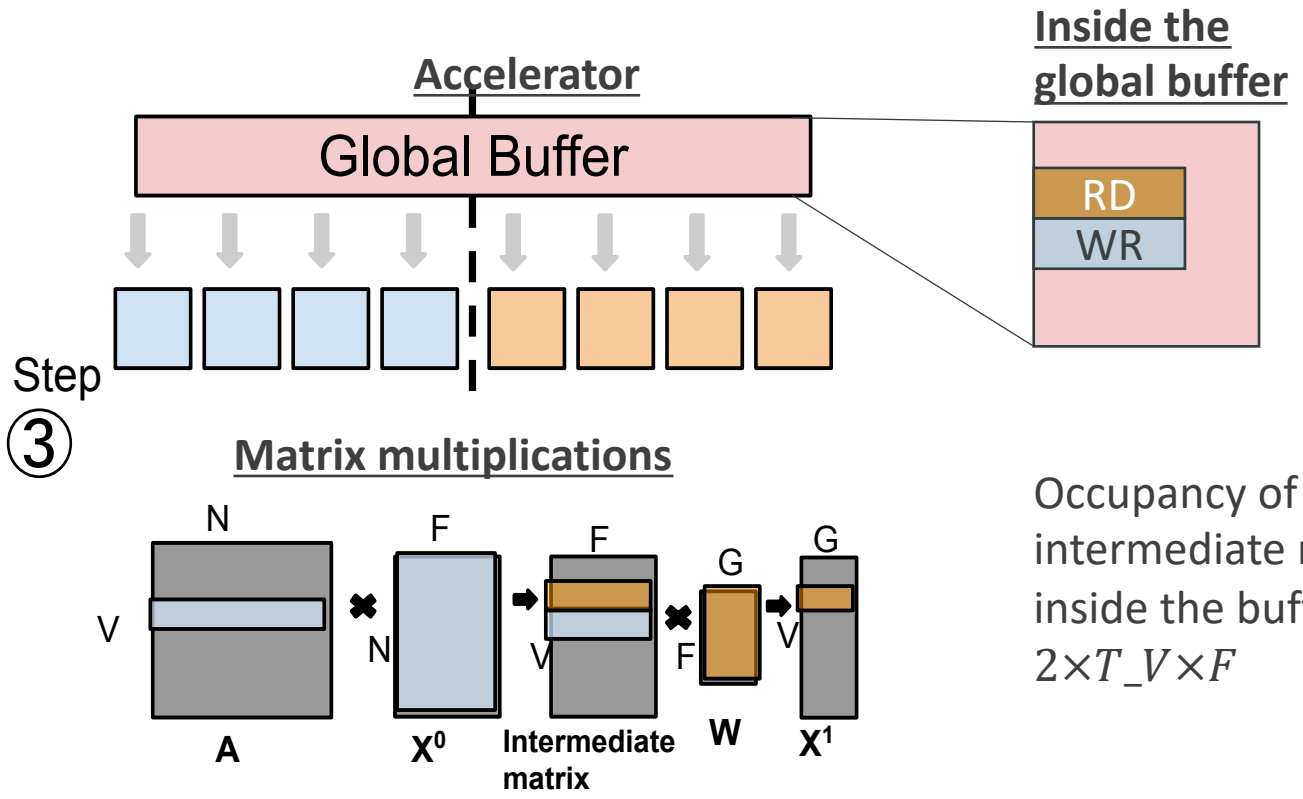


Occupancy of the intermediate matrix inside the buffer = $2 \times T\_V \times F$

Here, step refers to the duration in which producer writes a part of the intermediate matrix and consumer reads a part of the intermediate matrix

Execution example of PP dataflow

V: Vertices; N: Neighbors; F: Input Features, G: Output Features

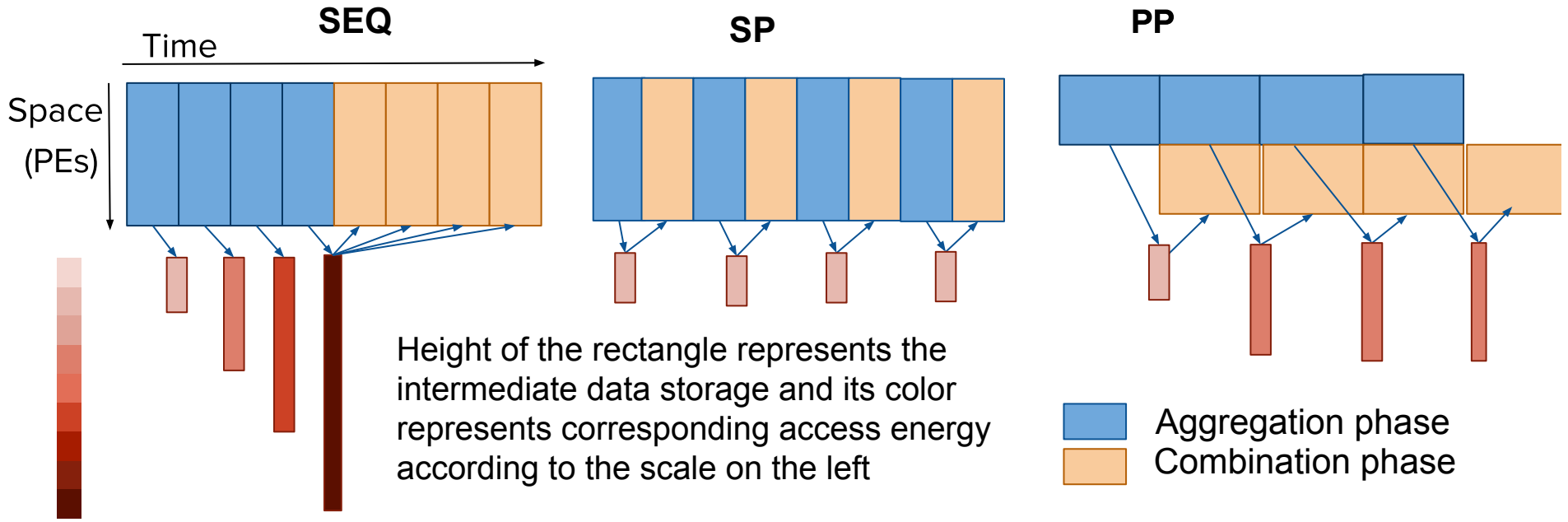


Occupancy of the intermediate matrix inside the buffer = $2 \times T\_V \times F$

Here, step refers to the duration in which producer writes a part of the intermediate matrix and consumer reads a part of the intermediate matrix

Data movement in Inter-phase Dataflows

Inter-phase dataflows describe the data reuse strategies between the two phases.



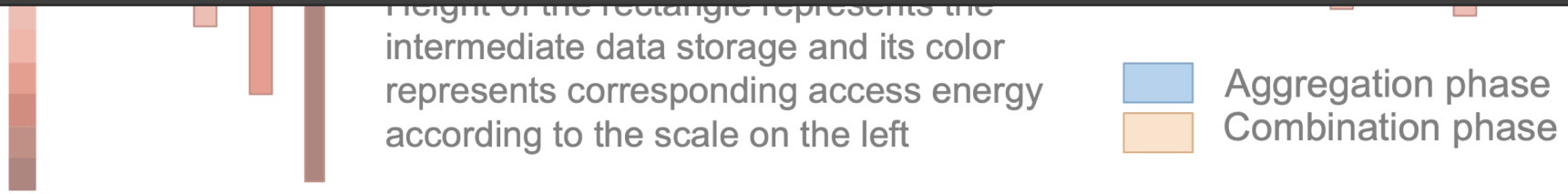
SEQ dataflow writes the complete intermediate matrix to the memory in Aggregation and then reads it in the Combination phase.

SP and PP dataflows reuse the data between the two phases, thus avoiding writing the entire intermediate matrix to the DRAM and reading it.

Data movement in Inter-phase Dataflows

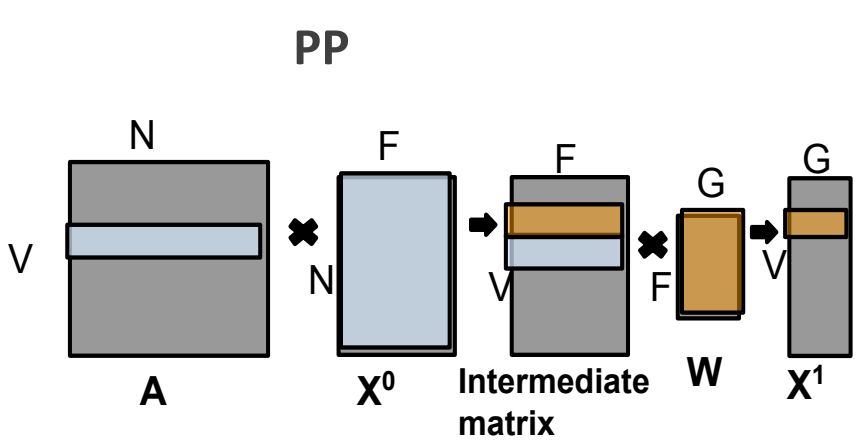
Inter-phase dataflows describe the data reuse strategies between the two phases.

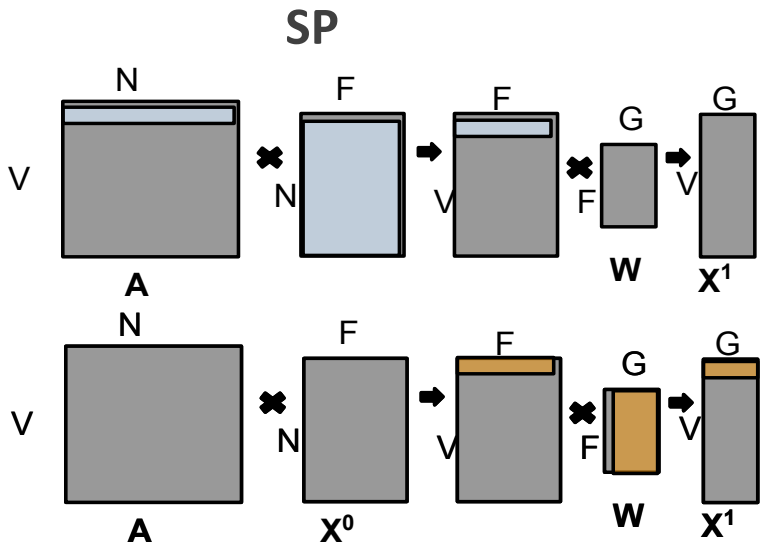
SP and PP dataflow reduce the data movement of the intermediate matrix by consuming right after producing



SEQ dataflow writes the complete intermediate matrix to the memory in Aggregation and then reads it in the Combination phase.
SP and PP dataflows reuse the data between the two phases, thus avoiding writing the entire intermediate matrix to the DRAM and reading it.

Interdependence between dataflows





Here, the production and consumption of data are in the row-major order. Therefore, the outermost loop is V in both the cases and inner loops can be re-ordered.

AGG

```
for v=1:V
  for f=1:F
    for n=1:N
```

CMB

```
for v=1:V
  for g=1:G
    for f=1:F
```

For more details on dependencies between the loop orders and the tile sizes, and for other ways to stage the computations please refer to the pre-print of the paper https://arxiv.org/pdf/2103.07977.pdf

Interdependence between dataflows

SP and PP dataflows are interdependent.

Here, the production and consumption of data are in the row major order. Therefore, the outermost loop is V in both the cases and inner loops can be re-ordered.

AGG

```
for v=1:V
  for f=1:F
    for n=1:N
```

CMB

```
for v=1:V
  for g=1:G
    for f=1:F
```

For more details on dependencies between the loop orders and the tile sizes, and for other ways to stage the computations please refer to the pre-print of the paper https://arxiv.org/pdf/2103.07977.pdf

GNN Dataflow Taxonomy

V: Vertices; N: Neighbors; F: Input Features, G: Output Features

<Inter Phase>$_{<Phase\ order>}$(<Intra Phase Agg>, <Intra Phase Cmb>)

- Inter-Phase dataflows - Seq, SP, PP
- Phase Order - AC, CA
- Intra-Phase dataflow *(Examples - $V_sG_sF_t$ for Combination)*.

GNN dataflow DSE

TABLE II

CHARACTERIZING THE DESIGN-SPACE OF GNN DATAFLOWS. NOTE THAT ANY DIRECTION WILL AFFECT THE AGGREGATION AND COMBINATION DIMENSION VARIABLES, BUT SIMILAR CONCEPTS APPLY. SUBSCRIPTS *s*, *t*, *x* MEAN SPATIAL, TEMPORAL, EITHER SPATIAL OR TEMPORAL RESPECTIVELY.

Row	Inter Phase	Order - Aggregation, Combination	Intermediate Global Buffer	NoC/PE support	Example & Order	Remarks
1	**Sequential (Seq)**	ANY-All pairs	✓	Intra-phase	TPU [31] Eyeriss [7] (any direction)	Similar to running one DNN layer at a time. The outputs of one phase get stored in the memory and rescheduled back onto the PEs.
2	**Sequential Pipeline (SP)**	AC - $V_xF_xN_t$, $V_xF_xG_t$ AC - $F_xV_xN_t$, $F_xV_xG_t$ CA - $N_xF_xV_t$, $V_xG_xF_t$ CA - $F_xN_xV_t$, $G_xV_xF_t$	✗	Local buffer inside PEs to accumulate data	EnGN [21] (any direction)	**SP-Optimized**: Avoids GB accesses, as output data of one phase is stationary in the PE RF, and can be used as input for the next phase. For agg->cmb, $T\_V_{AGG}=T\_V_{CMB}$ and $T\_F_{AGG}=T\_F_{CMB}$ and reduction is temporal as the data is always inside in-place buffers.
3		Same as rows 4-9	✓	Intermediate outputs stored in GB.	(any direction)	**SP-Generic**: There will be a buffer/ setup delay between aggregation and combination phases to remap the output data to a new location.
4	**Parallel Pipeline (PP)**	AC - $V_xF_xN_x$, $V_xF_xG_x$ AC - $F_xV_xN_x$, $F_xV_xG_x$	✓	NoC connecting Agg and Cmb units to intermediate buffer.	(agg ->cmb)	**Element(s) wise granularity**: Element(s) of the intermediate matrix indexed by V,F can be pipelined.
5		AC - $V_xF_xN_x$, $V_xG_xF_x$ AC - $V_xN_xF_x$, $V_xG_xF_x$ AC - $V_xN_xF_x$, $V_xF_xG_x$	✓	NoC connecting Agg and Cmb units to intermediate buffer.	HyGCN [19] Auten et al. [24] (agg ->cmb)	**Row(s) wise granularity**:Row(s) of the intermediate matrix indexed by V can be pipelined. HyGCN allocates a fixed number of PEs, which may lead to stalls. (combination engine idle waiting). HyGCN dataflow-$PP_{AC}(V_xF_sN_t, V_sG_sF_t)$
6		AC - $F_xV_xN_x$, $F_xG_xV_x$ AC - $F_xN_xV_x$, $F_xG_xV_x$ AC - $F_xN_xV_x$, $F_xV_xG_x$	✓	NoC connecting Agg and Cmb units to intermediate buffer.	(agg ->cmb)	**Column(s) wise granularity**: Column(s) of the intermediate matrix indexed by F can be pipelined.
7		CA - $N_xF_xV_t$, $V_xG_xF_t$ CA - $F_xN_xV_t$, $G_xV_xF_t$	✓	NoC connecting Agg and Cmb units to intermediate buffer.	(cmb ->agg)	**Element(s) wise granularity**: The order should be (NFV, VGF) or (FNV, GVF). V×G matrix after Cmb becomes N × F for Agg.
8		CA - $N_xV_xF_x$, $V_xG_xF_x$ CA - $N_xV_xF_x$, $V_xF_xG_x$ CA - $N_xF_xV_x$, $V_xF_xG_x$	✓	NoC connecting Agg and Cmb units to intermediate buffer.	(cmb ->agg)	**Row(s) wise granularity**
9		CA - $F_xV_xN_x$, $G_xV_xF_x$ CA - $F_xV_xN_x$, $G_xF_xV_x$ CA - $F_xN_xV_x$, $G_xF_xV_x$	✓	NoC connecting Agg and Cmb units to intermediate buffer.	AWB-GCN [20] (cmb ->agg)	**Column(s) wise granularity**: AWB-GCN enables a flexible allocation of PEs for different phases to match production and consumption rates. AWB-GCN dataflow- $PP_{CA}(F_sN_tV_s, G_tF_tV_s)$

GNN dataflow DSE

TABLE II

CHARACTERIZING THE DESIGN-SPACE OF GNN DATAFLOWS. NOTE THAT ANY DIRECTION WILL AFFECT THE AGGREGATION AND COMBINATION DIMENSION VARIABLES, BUT SIMILAR CONCEPTS APPLY. SUBSCRIPTS *s*, *t*, *x* MEAN SPATIAL, TEMPORAL, EITHER SPATIAL OR TEMPORAL RESPECTIVELY.

Row	Inter Phase	Order - Aggregation, Combination	Intermediate Global Buffer	NoC/PE support	Example & Order	Remarks
5		AC - $V_xF_xN_x$, $V_xG_xF_x$ AC - $V_xN_xF_x$, $V_xG_xF_x$ AC - $V_xN_xF_x$, $V_xF_xG_x$	✓	NoC connecting Agg and Cmb units to intermediate buffer.	HyGCN [19] Auten et al. [24] (agg ->cmb)	matrix indexed by V can be pipelined. HyGCN allocates a fixed number of PEs, which may lead to stalls. (combination engine idle waiting). HyGCN dataflow-$PP_{AC}(V_xF_sN_t, V_sG_sF_t)$
6		AC - $F_xV_xN_x$, $F_xG_xV_x$ AC - $F_xN_xV_x$, $F_xG_xV_x$ AC - $F_xN_xV_x$, $F_xV_xG_x$	✓	NoC connecting Agg and Cmb units to intermediate buffer.	(agg ->cmb)	**Column(s) wise granularity**: Column(s) of the intermediate matrix indexed by F can be pipelined.
7		CA - $N_xF_xV_t$, $V_xG_xF_t$ CA - $F_xN_xV_t$, $G_xV_xF_t$	✓	NoC connecting Agg and Cmb units to intermediate buffer.	(cmb ->agg)	**Element(s) wise granularity**: The order should be (NFV, VGF) or (FNV, GVF). V×G matrix after Cmb becomes N × F for Agg.
8		CA - $N_xV_xF_x$, $V_xG_xF_x$ CA - $N_xV_xF_x$, $V_xF_xG_x$ CA - $N_xF_xV_x$, $V_xF_xG_x$	✓	NoC connecting Agg and Cmb units to intermediate buffer.	(cmb ->agg)	**Row(s) wise granularity**
9		CA - $F_xV_xN_x$, $G_xV_xF_x$ CA - $F_xV_xN_x$, $G_xF_xV_x$ CA - $F_xN_xV_x$, $G_xF_xV_x$	✓	NoC connecting Agg and Cmb units to intermediate buffer.	AWB-GCN [20] (cmb ->agg)	**Column(s) wise granularity**: AWB-GCN enables a flexible allocation of PEs for different phases to match production and consumption rates. AWB-GCN dataflow- $PP_{CA}(F_sN_tV_s, G_tF_tV_s)$

Our taxonomy can capture the dataflows used by the state-of-the-art GNN accelerators. For more details, please refer to the paper.

Qualitative observations in Inter-phase dataflows

- SEQ – Any intra-phase dataflow ; SP and PP constrained.
- Within SP, for intermediate matrix to stay in the register file, T_V should be equal for both phases and so does T_F. T_N=1. So, higher data movement reduction puts more constraints on the dataflow choices. We call this SP-Optimized and we refer to other SP datafloes as SP-Generic.
- PP requires balancing Aggregation and Combination.

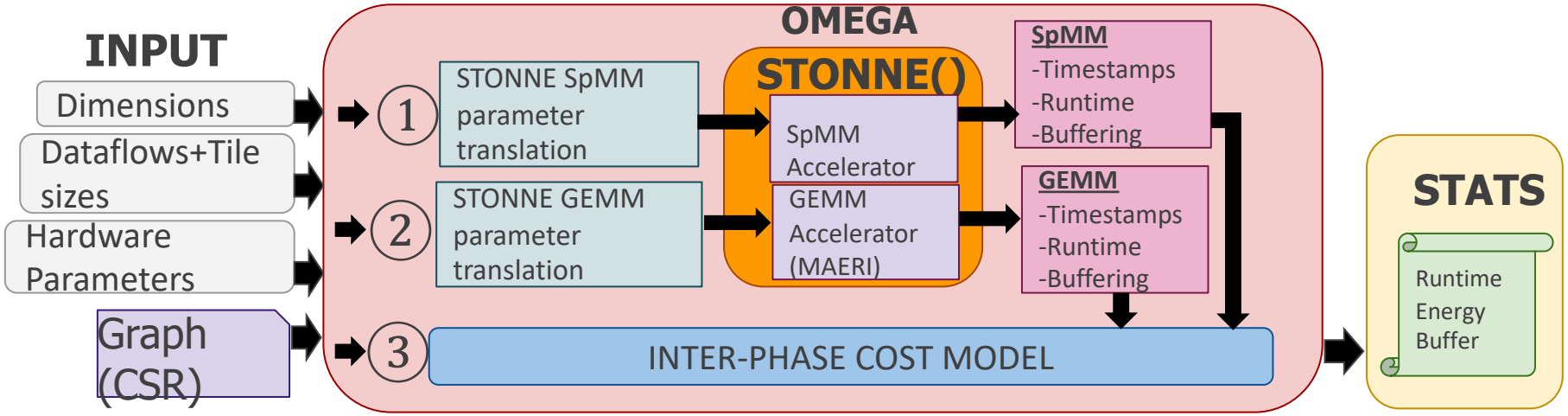
Qualitative observations in Inter-phase dataflows

- SEQ – Any intra-phase dataflow ; SP and PP constrained.
- Within SP, for intermediate matrix to stay in the register file, T_V should be equal for both phases and so does T_F. T_N=1. So, higher

These dataflows have numerous tradeoffs, thus detailed quantitative modeling of these phases is required.

OMEGA Cost Model

- To this end, OMEGA encodes the analytical model of the inter-phase dataflows
- OMEGA builds upon STONNE (Munoz-Martinez et al., IISWC 2021), which accurately models the hardware for MAERI which is a flexible accelerator.
- We also add support for indexing logic and memory controllers for SpMM which is in addition to MAERI (CONV/GEMM) and SIGMA(SpGEMM) and we use the MAERI accelerator for DenseGEMMs.
- `vim stonne/src/SparseDenseSDMemory.cpp`



Some key Analytical Equations

- For more detailed explanation, please refer to the pre-print of the paper section 4

Inter-phase Dataflows	Intermediate buffer storage required	Runtime	Global Buffer Accesses
Seq	$V \times F$	$T_{AGG} + T_{CMB}$	$Access_{AGG} + AccessCMB$
SP-Generic-rows	$T\_Vmax \times F$	$T_{AGG} + T_{CMB}$	$Access_{AGG} + AccessCMB$
SP-Optimized	0	$T_{AGG} + T_{CMB} - T_{redistribution}$	$Access_{AGG} + Access_{CMB} - (Int\_acessesWR_{,A} + Int\_accessesRD_{,CMB})$
PP-rows	$2 \times T\_Vmax \times F$	$sum(\max(t_{AGG}, tCMB)_{stages})$	$Access_{AGG} + AccessCMB$

Outline

- Graph Neural Networks (GNNs)
- GNN Dataflows – Taxonomy and modeling
- Hands-on Exercises

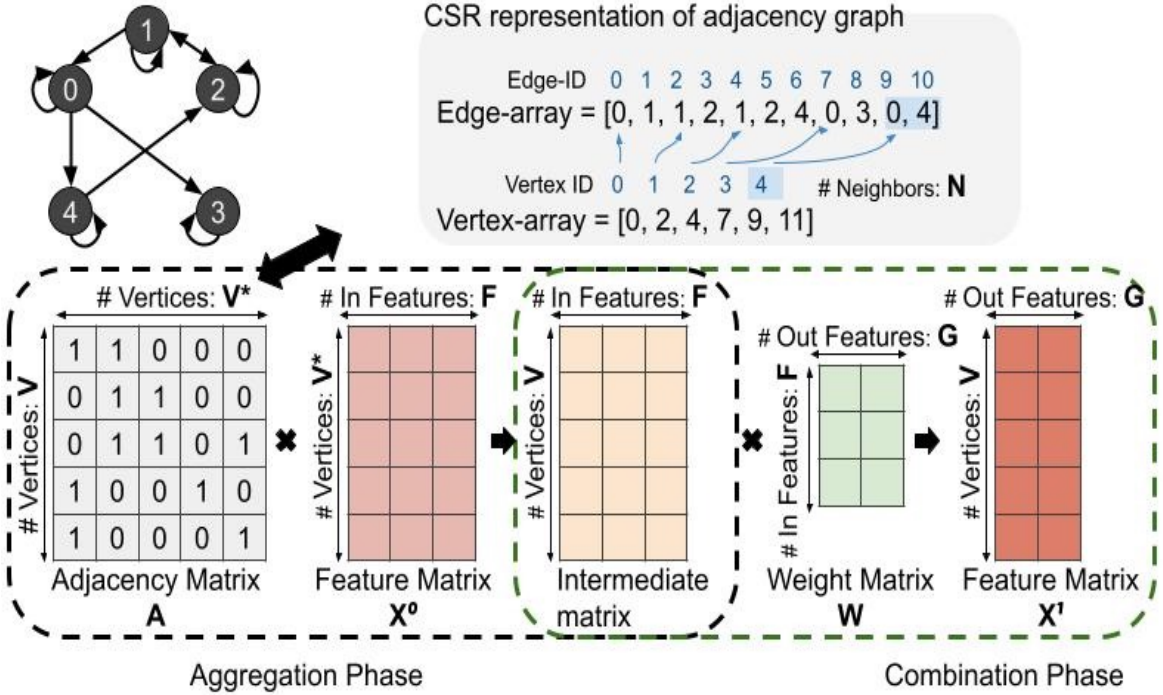
Inputs: OMEGA command line interface

```
vim example_simulation.sh
./omega -V=1168 -F=28 -G=2 -E=2590 -T_Va=18 -T_N=1 -T_Fa=28 -T_Vc=18 -T_G=1 -
T_Fc=28 -pe_agg=512 -pe_cmb=512 -dn_bw_agg=512 -rn_bw_agg=512 -dn_bw_cmb=512 -
rn_bw_cmb=512 -vertex_path="sample_graphs/vertex_mutag_batch64.txt" -
edge_path="sample_graphs/edge_mutag_batch64.txt"
cd sample_graphs
vim edge_mutag_batch64.txt
vim vertex_mutag_batch64.txt
```

- Dimensions
  - -V, -F, -G, -E (Edges, required for parsing)
- Tile sizes for both phases
  - -T_Va, -T_N, -T_Fa, -T_Vc, -T_G, -T_Fc
- Hardware Parameters
  - -Pe_agg, -Pe_cmb, -dn_bw_agg, -dn_bw_cmb, -rn_bw_agg, -rn_bw,cmb
- Path to Input files for the adjacency matrix (CSR representation)
  - -vertex_path, edge_path

Parameter translation to STONNE

- $V_{AGG, GNN} \rightarrow M_{SpMM}$
- $F_{AGG, GNN} \rightarrow N_{SpMM}$
- $N_{GNN} \rightarrow K\_non\text{-}zero_{SpMM}$
- $V_{CMB, GNN} \rightarrow M_{GEMM}$
- $F_{CMB, GNN} \rightarrow K_{GEMM}$
- $G_{GNN} \rightarrow N_{GEMM}$



OMEGA wrapper

`vim stonne/src/omega.cpp`

The OMEGA wrapper is **`omega.cpp`** and consists of the following parts-

1. Parsing the command line parameters
2. Reading the matrices
3. Instantiating and running STONNE for SpMM
4. Repeating 2. and 3. for GEMM.
5. Inter-phase analytical equations. Please note that currently there are additional equations to enable $T\_V_{AGG}=1$.

Configuring STONNE for SpMM

```
Stonne* stonne_instance = new Stonne(stonne_cfg1);


    stonne_instance->loadSparseDense(layer_name, F, V, V, MK_sparse_matrix, KN_dense_matrix, (unsigned int*)MK_col_id, (unsigned int*) MK_row_pointer, acc_output, T_Fa, T_N); //Loading Sparse Dense
stonne_instance->loadClocking(clocked_op);
stonne_instance->run(); //Running the simulator to get the intra-phase stats and the timestamps
unsigned int a_cycles = stonne_instance->getCycles(); //Cycles returned by the simulator for the SpMM phase
    std::cout<<"\n\n------Timestamps for simulated SpMM----------------------------------------------------------------------------------------------------\n";
for(int i=F-1;i<V*F;i+=F)
    std::cout<<clocked_op[i]<<"\t"; //Timestamps are saved in this variable when the simulator is running

std::cout<<"\n\n------Statistics for simulated SpMM----------------------------------------------------------------------------------------------------\n";
////////////// Other stats
MSNetworkStats mulstatssp = stonne_instance->getMultiplierNetworkStats();
std::cout<<"\n\nRF Stats:\n";
std::cout<<"RF weight reads = "<<mulstatssp.n_l1_weight_reads<<"\n";
std::cout<<"RF weight writes = "<<mulstatssp.n_l1_weight_writes<<"\n";
std::cout<<"RF input reads = "<< mulstatssp.n_l1_input_reads<<"\n";
std::cout<<"RF input writes = "<< mulstatssp.n_l1_input_writes<<"\n";
std::cout<<"RF psum reads = "<<mulstatssp.n_l1_psum_reads<<"\n";
std::cout<<"RF psum writes = "<<mulstatssp.n_l1_psum_writes<<"\n";
//std::cout<<"Number of multiplications = "<<mulstatssp.n_multiplications<<"\n";
//std::cout<<"Number of link traversals = "<<mulstatssp.n_local_network_traversals<<"\n";

std::cout<<"\n\nAdder stats: \n";
if(T_N==1)
    std::cout<<"Reduction network is linear since reduction is temporal\n";
else
{
    ASNetworkStats adderstatssp = stonne_instance->getASNetworkStats();
    std::cout<<"Number of link traversals = "<<adderstatssp.n_total_traversals<<"\n";
}

DSNetworkStats diststatssp = stonne_instance->getDSNetworkStats();
std::cout<<"\n\nDistribution stats: \n";
std::cout<<"Number of link traversals = "<<diststatssp.n_total_traversals<<"\n";

SDMemoryStats memstatssp = stonne_instance->getMemoryStats();
std::cout<<"\n\nGlobal Buffer stats: \n";
std::cout << "Global SRAM weight reads = " << memstatssp.n_SRAM_weight_reads <<"\n";
std::cout << "Global SRAM input reads = " << memstatssp.n_SRAM_input_reads <<"\n";
std::cout << "Global SRAM psum reads = " << memstatssp.n_SRAM_psum_reads << "\n";
std::cout << "Global SRAM psum writes = " << memstatssp.n_SRAM_psum_writes << "\n";
```

Inter-phase equations

```
        else
        n_cycles+=clocked_op2[V*G-1]-clocked_op2[(V-T_Vp)*G-1];
        std::cout<<"\n\n----------------------------------------------------Inter-phase stats----------------------------------------------------\n";
        std::cout<<"\n\n-----------------------------------------------PP_AC(VFN, VGF), numPEs=pe_agg+pe_cmb-----------------------------\n";
        std::cout<<"\n\n\nNumber of PP cycles = "<<n_cycles<<"\n\nNumber of PP buffers utilized = "<<n_buffers<<"\n\n";
}

        //////// Modifying SP. SP-Optimized requires (VFN,VFG). We also estimate the psum overhead and add it to the stats computed by (VFN,VGF)
        std::cout<<"\n\n-----------------------------------------------Seq_AC(VFN, VGF), numPEs=pe_agg=pe_cmb-----------------------------\n";
        std::cout<<"\n\nSimulated Aggregation Cycles (T_Va=1) = "<<a_cycles<<"\n\n";
        std::cout<<"\n\nCombination Cycles = "<<c_cycles<<"\n\n";
        std::cout<<"\n\nAggregation Cycles = "<<m_cycles<<"\n\n";
        int s_cycles=m_cycles+c_cycles;
        int s_buffers=V*F;
        std::cout<<"\n\nSeqential cycles = "<<s_cycles<<"\t Intermediate buffering for sequential = "<<s_buffers<<"\n\n";
        int l_cycles=(V/T_Vc+(V%T_Vc!=0))*(F/T_Fc+(F%T_Fc!=0))*pe_cmb/dn_bw_cmb;

        std::cout<<"\n\n--------------------------------SP_AC(VFN, VFG)(Psum overhead of VFG considered), numPEs=pe_agg=pe_cmb-----------------------------\n";
        std::cout<<"\n\nRedistribution Cycles saved= "<<l_cycles<<"\n\n"; ///////////////// Redistribution cycles saved by SP /////////////////
        int sp_buffers = G*T_Vc*(F/T_Fc+(F%T_Fc!=0));
        int p_cycles = ((sp_buffers/pe_cmb)+(sp_buffers%pe_cmb!=0))*(V/T_Vc+(V%T_Vc!=0));       // Adding the partial sum overhead of VFN,VFG
        if(F==T_Fc)
        p_cycles=0;
        std::cout<<"\n\nPsum overhead Cycles = "<<p_cycles<<"\n\n";
        int sp_cycles = s_cycles-l_cycles+p_cycles;
        std::cout<<"\n\nSP dataflow cycles = "<<sp_cycles<<"\t Intermediate buffering for interleaving = "<<sp_buffers<<"\n\n";

        ////////////Printing buffer stats and analytical model here

        std::cout<<"-----------------------------------Global Buffer stats-------------------------------------------------------------------\n";
        int sp_psum_accesses=2*G*V*(F/T_Fc+(F%T_Fc!=0));
        if(F==T_Fc)
                sp_psum_accesses=0;
        std::cout<<"\n\nSP Psum accesses = \n\n"<<sp_psum_accesses;
        int pp_global_accesses=memstats.n_SRAM_weight_reads+memstats.n_SRAM_input_reads+memstats.n_SRAM_psum_reads+memstats.n_SRAM_psum_writes+
                memstatssp.n_SRAM_weight_reads+memstatssp.n_SRAM_input_reads+memstatssp.n_SRAM_psum_reads+memstatssp.n_SRAM_psum_writes;
        std::cout<<"\n\nPP and SEQ Global Buffer Accesses = \n\n"<<pp_global_accesses;
        int sp_global_accesses=memstats.n_SRAM_input_reads+memstats.n_SRAM_psum_reads+memstats.n_SRAM_psum_writes+
                memstatssp.n_SRAM_weight_reads+memstatssp.n_SRAM_input_reads+sp_psum_accesses;

        std::cout<<"\n\nSP Global Buffer Accesses = \n\n"<<sp_global_accesses;
        std::cout<<"\n\n\n\n";
```

Loop and phase orders currently

SEQ – (VFN, VGF)

SP – (VFN, VFG)

PP - (VFN, VGF)

Phase order – AGG to CMB

Running example simulation

```
make all
source example_simulation.sh
```

Output: Printed Results

- SpMM RF statistics
- SpMM Global Buffer (GB) statistics
- GEMM RF statistics
- GEMM Global Buffer Accesses
- Aggregation and Combination runtimes
- PP cycles (nPEs=AGG+CMB)
- SP cycles (These make sense only if PE_agg=PEcmb and then nPEs=PE_AGG)
- Intermediate GB occupancies
- GB accesses

Intra-phase stats

```
------Statistics for simulated SpMM-------------------------


RF Stats:
RF weight reads = 145012
RF weight writes = 72520
RF input reads = 72520
RF input writes = 72520
RF psum reads = 72520
RF psum writes = 0


Adder stats:
Reduction network is linear since reduction is temporal


Distribution stats:
Number of link traversals = 145040


Global Buffer stats:
Global SRAM weight reads = 2590
Global SRAM input reads = 72520
Global SRAM psum reads = 0
Global SRAM psum writes = 32704
Running CPU version to compare results
Test passed correctly
```

```
------Statistics for simulated GEMM---------------


RF Stats:
RF weight reads = 97776
RF weight writes = 32760
RF input reads = 65520
RF input writes = 65520
RF psum reads = 65520
RF psum writes = 0


Adder stats:
Number of link traversals = 122192


Distribution stats:
Number of link traversals = 98280


Global Buffer stats:
Global SRAM weight reads = 32704
Global SRAM input reads = 3640
Global SRAM psum reads = 0
Global SRAM psum writes = 2336
```

```
----------------------------------------------Inter-phase stats-----------------------------------------------

----------------------------------------------PP_AC(VFN, VGF), numPEs=pe_agg+pe_cmb----------------------------

Number of PP cycles = 1055

Number of PP buffers utilized = 1008

----------------------------------------------Seq_AC(VFN, VGF), numPEs=pe_agg=pe_cmb----------------------------

Simulated Aggregation Cycles (T_Va=1) = 14524

Combination Cycles = 1040

Aggregation Cycles = 908

Seqential cycles = 1948  Intermediate buffering for sequential = 32704

-------------------------------SP_AC(VFN, VFG)(Psum overhead of VFG considered), numPEs=pe_agg=pe_cmb-------------

Redistribution Cycles saved= 65

Psum overhead Cycles = 0

SP dataflow cycles = 1883        Intermediate buffering for interleaving = 36

-----------------------------------Global Buffer stats-------------------------------------------------------

SP Psum accesses =

0

PP and SEQ Global Buffer Accesses =

146494

SP Global Buffer Accesses =
```

Inter-phase stats

Changing the tile sizes

```
cp example_simulation.sh example2_simulation.sh
vim example2_simulation.sh
```

```
./omega -V=1168 -F=28 -G=2 -E=2590 -T_Va=170 -T_N=1 -T_Fa=3 -T_Vc=170 -T_G=1 -
T_Fc=3 -pe_agg=512 -pe_cmb=512 -dn_bw_agg=512 -rn_bw_agg=512 -dn_bw_cmb=512 -
rn_bw_cmb=512 -vertex_path="sample_graphs/vertex_mutag_batch64.txt" -
edge_path="sample_graphs/edge_mutag_batch64.txt"
```

```
source example2_simulation.sh
```

We observe that performance improves with these tile sizes

Changing the bandwidth

```
cp example_simulation.sh example3_simulation.sh
vim example3_simulation.sh
```

```
./omega -V=1168 -F=28 -G=2 -E=2590 -T_Va=18 -T_N=1 -T_Fa=28 -T_Vc=18 -T_G=1 -
T_Fc=28 -pe_agg=512 -pe_cmb=512 -dn_bw_agg=128 -rn_bw_agg=128 -dn_bw_cmb=128 -
rn_bw_cmb=128 -vertex_path="sample_graphs/vertex_mutag_batch64.txt" -
edge_path="sample_graphs/edge_mutag_batch64.txt"
```

```
source example3_simulation.sh
```

As expected, performance deteriorates with decrease in BW

For details on other experiments, please look at the preprint section 5

Summary

- Several important workloads across ML and HPC have multiphase matrix computations and the dataflows that can exploit reuse between the phases are crucial.
- We capture the design-space of these dataflows using a succinct taxonomy. We contrast these dataflow choices and highlight the trade-offs in these dataflows.
- We propose an analytical framework for cost-modelling dataflows.