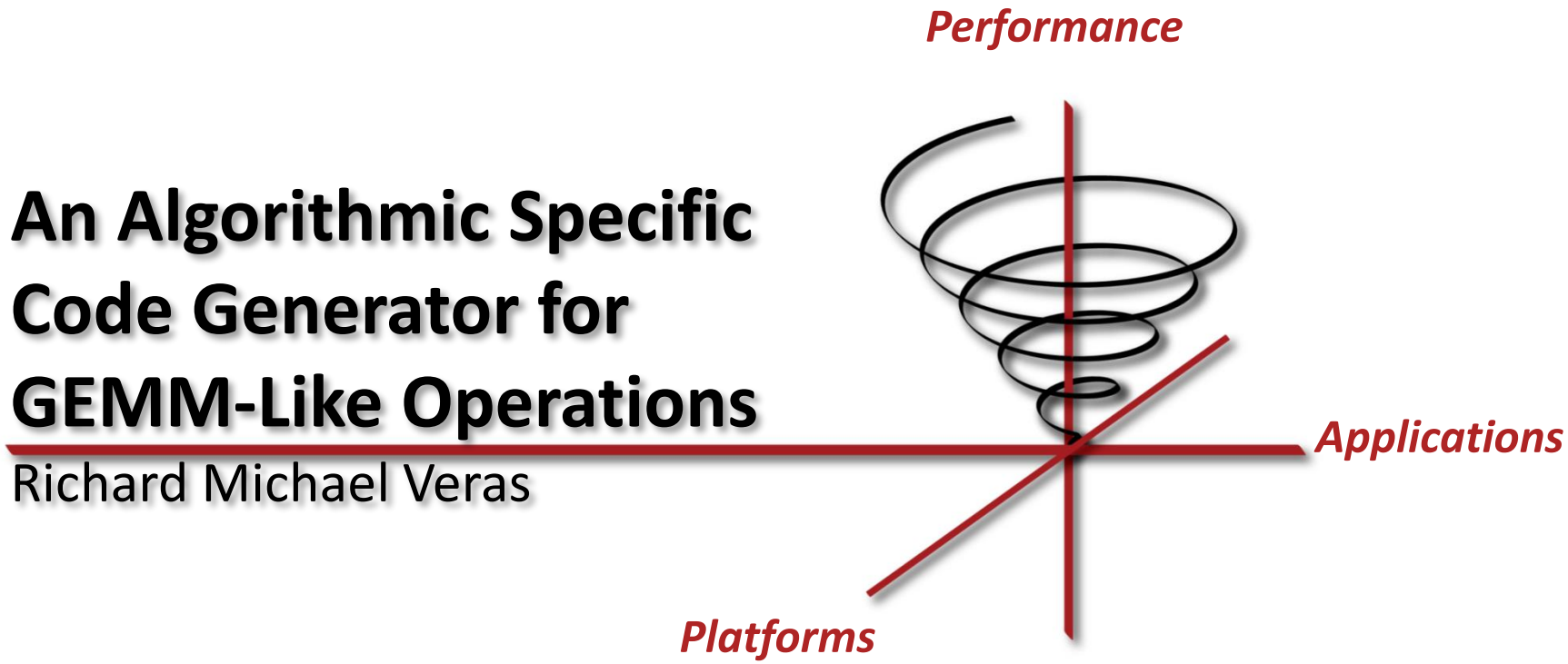
SPIRAL
www.spiral.net

An Algorithmic Specific Code Generator for GEMM-Like Operations

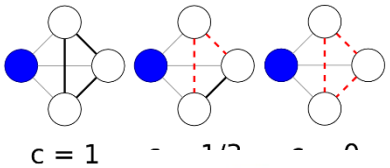
Richard Michael Veras

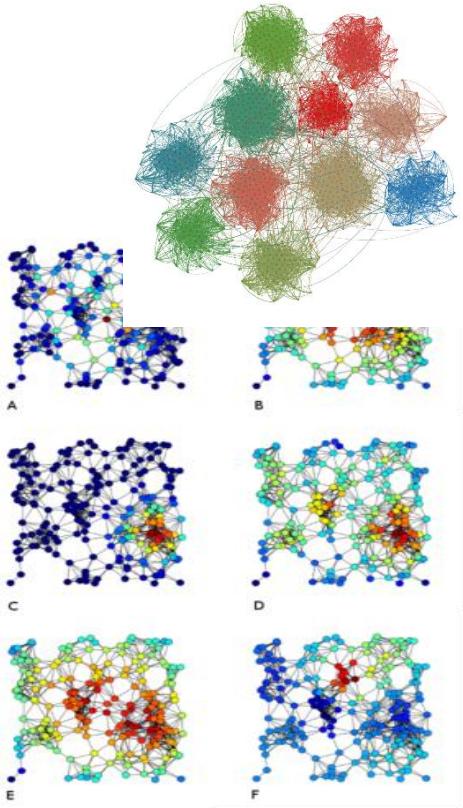


Want Automatic High Performance

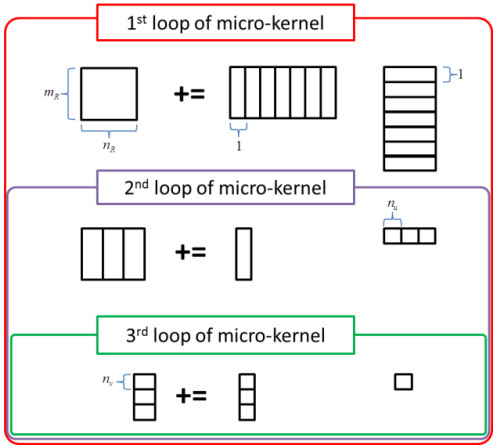
SPIRAL
www.spiral.net

GEMM-Like Operations

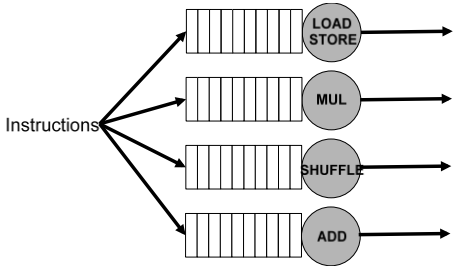




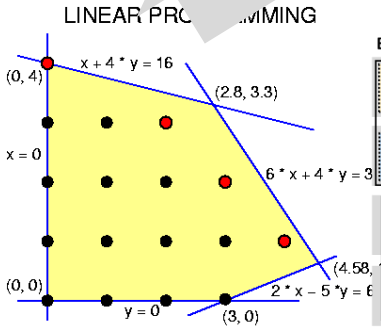
Model Driven approach for generating DGEMM:

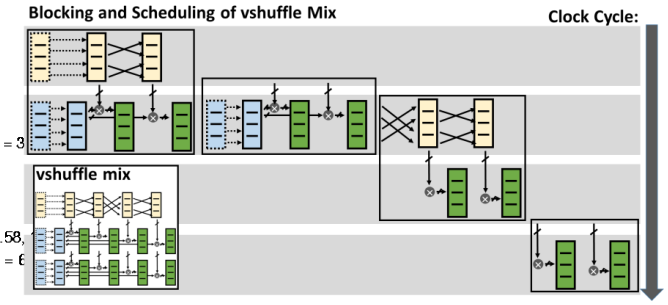


High Perf.



Compiler Techniques:



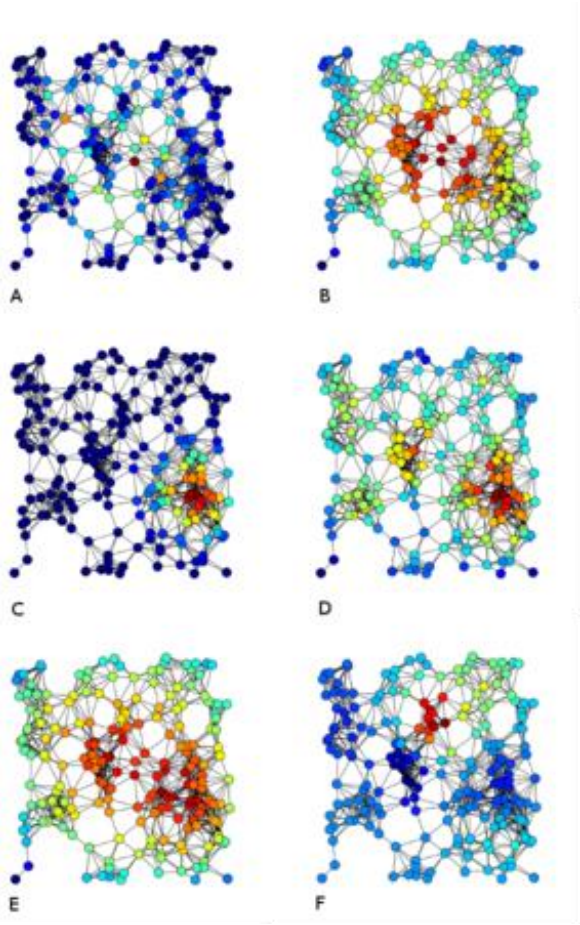


Function to maximize: f(x, y) = 6 * x + 5 * y
Optimum LP solution (x, y) = (2.4, 3.4)
Pareto optima: (0, 4), (2, 3), (3, 2), (4, 1)
Optimum ILP solution (x, y) = (4, 1)

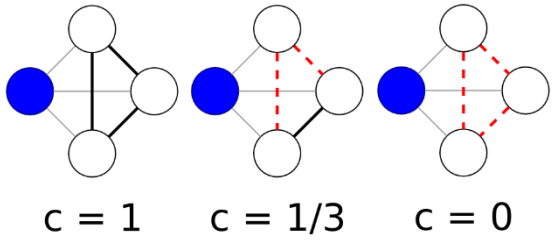
GEMM Like Operations

SPIRAL www.spiral.net

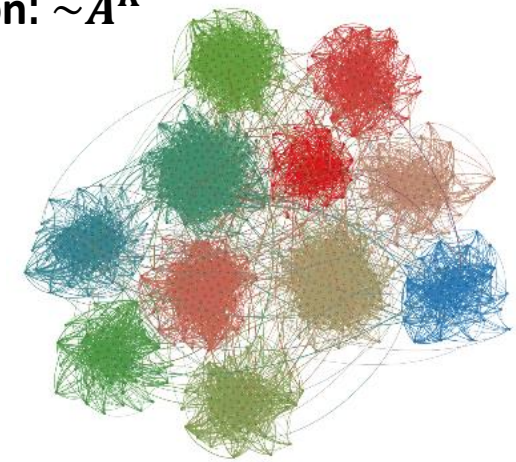
Centrality: $\sim A^T A$
[betweeness]
$(Z, +, MIN, 0, 1)$



Clustering: $\sim AA^T A$
(triangle)
$(Z, \wedge, +, 1, 0)$



Community Detection: $\sim A^K$

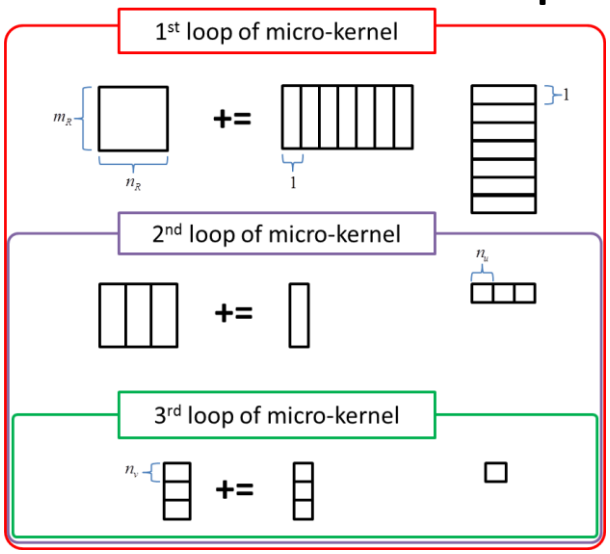


Check out GraphBLAS

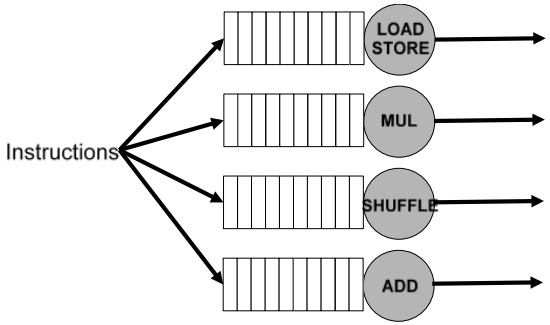
High Performance Micro-Kernels

SPIRAL www.spiral.net

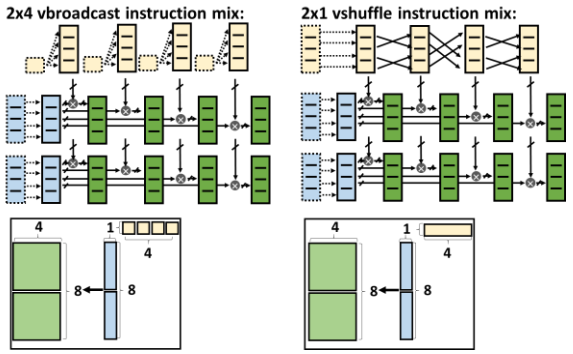
Cast micro kernel as outer product:



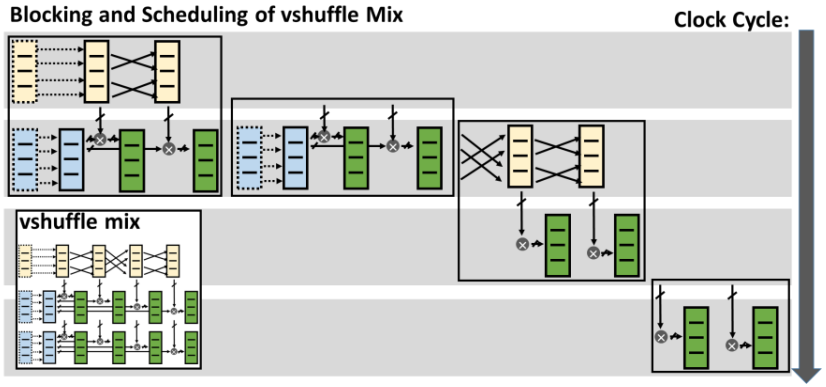
Use Models to Select from Design Space:



Enumerate all possible tilings given ISA



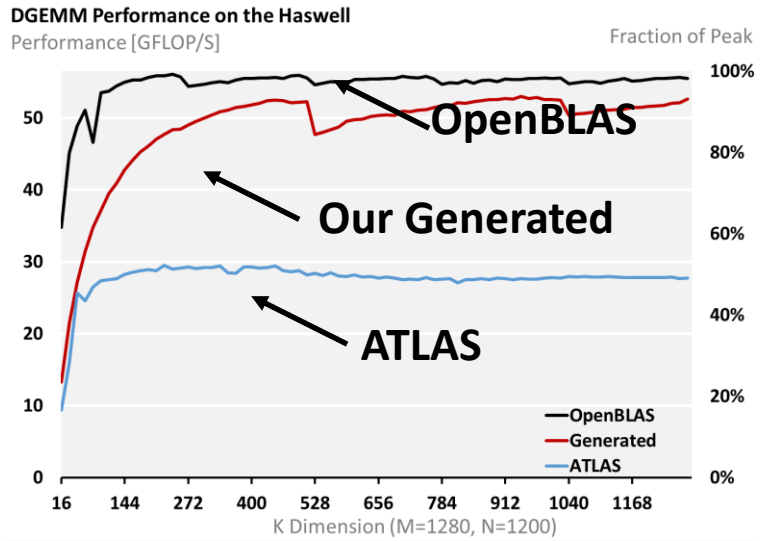
Aggressively Schedule and Optimize:

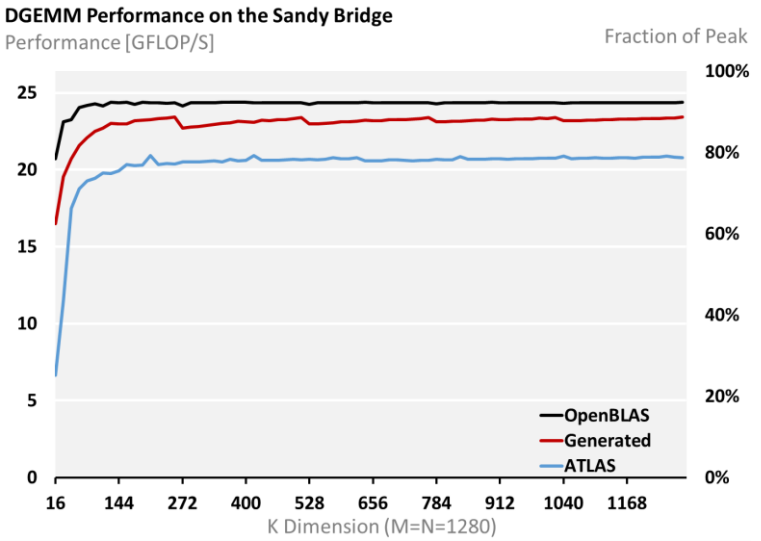


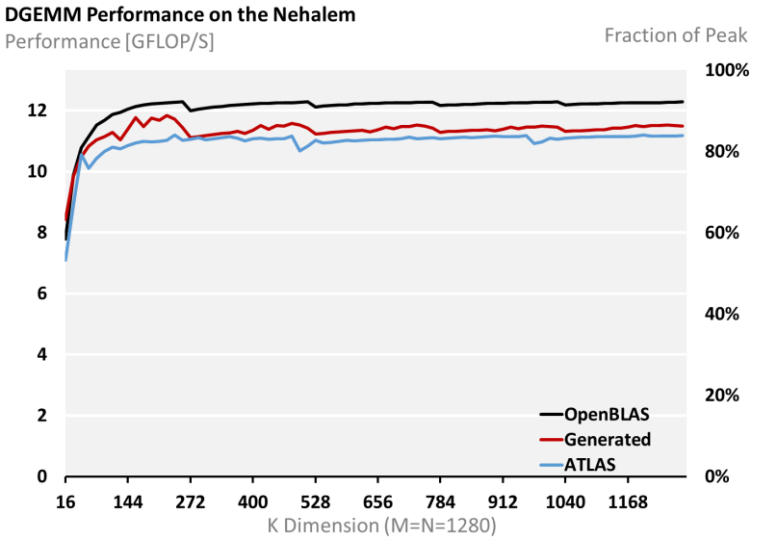
Veras, R. , Smith T., Low T.M., Franchetti, F. van de Geijn, R. [CGO 2017 Submitted]

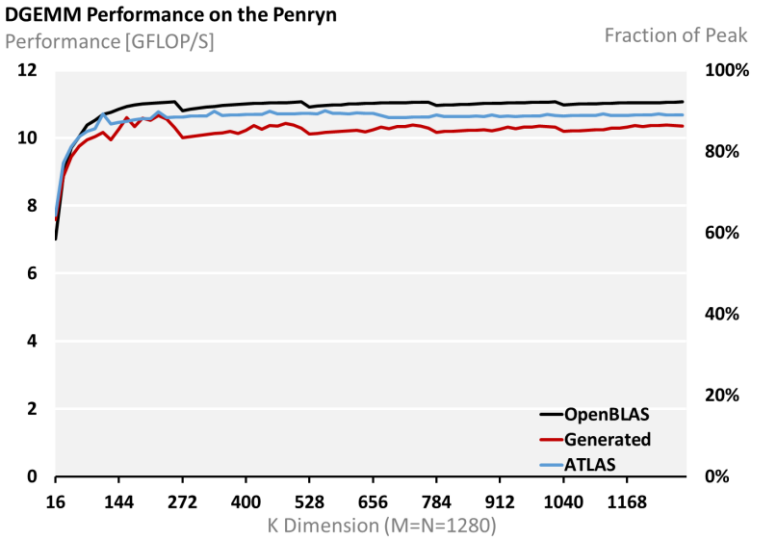
High Performance Micro-Kernels

SPIRAL
www.spiral.net







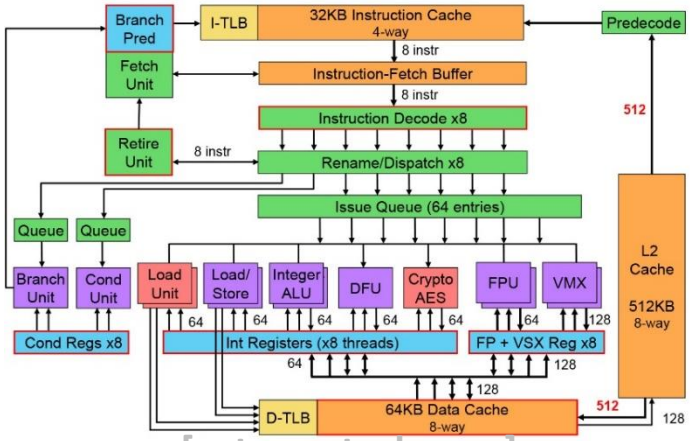


Veras, R. , Smith T., Low T.M., Franchetti, F. van de Geijn, R. [CGO 2017 Submitted]

Automating with Compiler Techniques

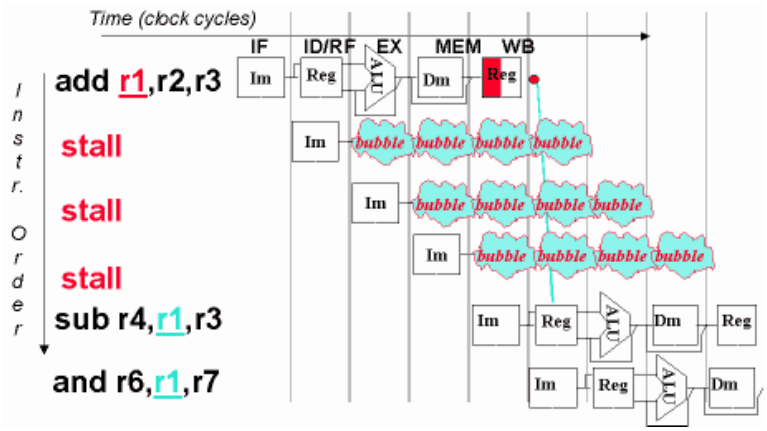
SPIRAL www.spiral.net

Subtle Semiring Changes Impact Performance:



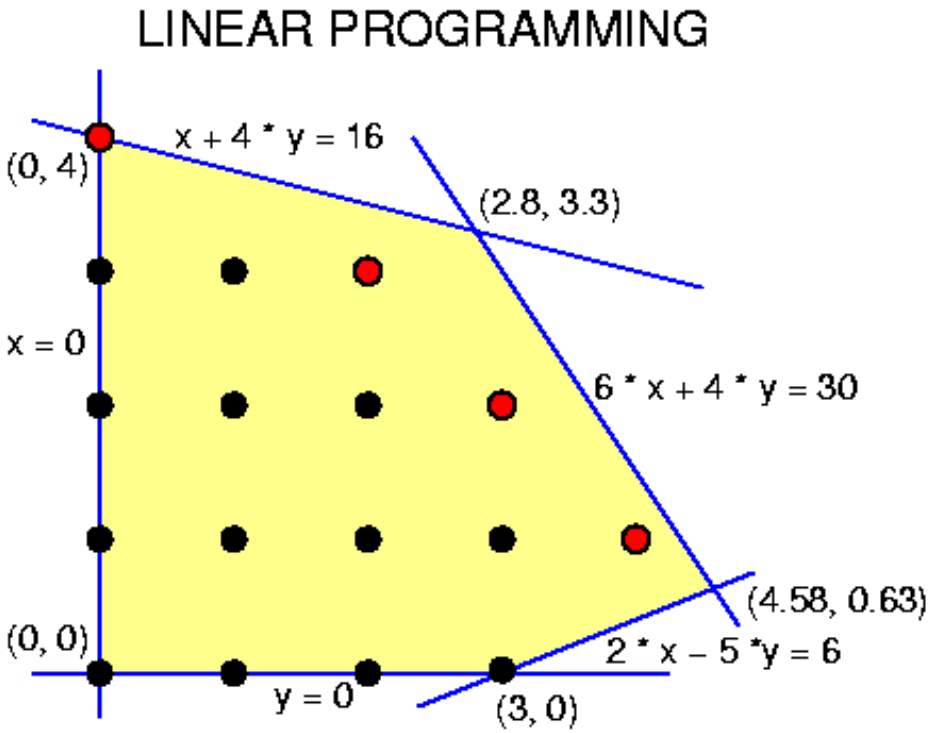
[extremetech.com]

Minimizing Stalls



[massey.ac.nz]

Cast as ILP Problem



Function to maximize: f(x, y) = 6 * x + 5 * y
Optimum LP solution (x, y) = (2.4, 3.4)
Pareto optima: (0, 4), (2, 3), (3, 2), (4, 1)
Optimum ILP solution (x, y) = (4, 1)

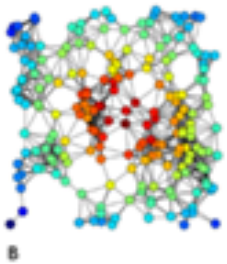
[cs.duke.edu]

A Generator for GEMM-Like Kernels

SPIRAL www.spiral.net

Input:

Betweeness Centrality (Floyd-Warshall)



GEMM Like:

$C \leftarrow AB + C$

$C_{acc} \leftarrow 0$ Initialize

$C_{acc} \leftarrow \sum_p ab^T$ Compute

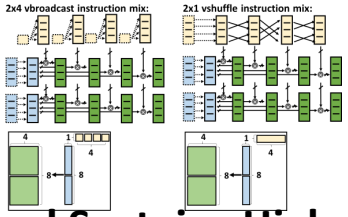
$C \leftarrow f(C_{acc})$ Accumulate

Semiring

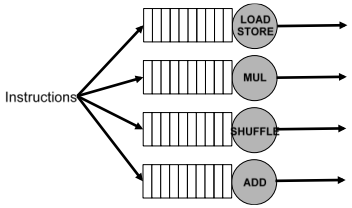
$(Z, +, MIN, 0, 1)$

Ouput:

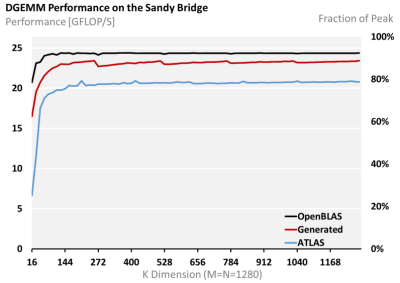
Kernel Algorithm from our design space:



Kernel Sustains High Throughput:

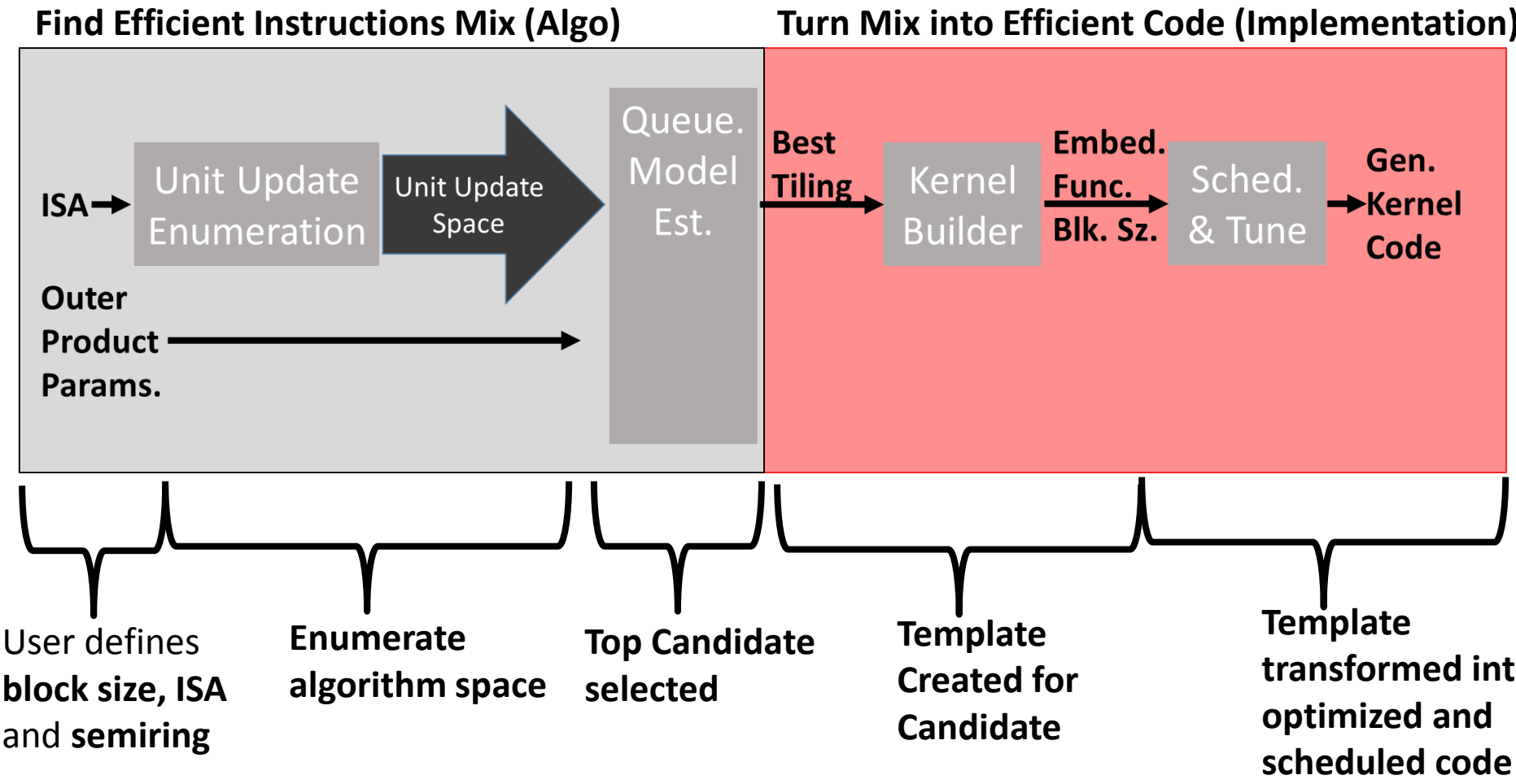


Tuned to the Target Architecture



Our GEMM Generator Pipeline

SPIRAL www.spiral.net



From Math to Tiling

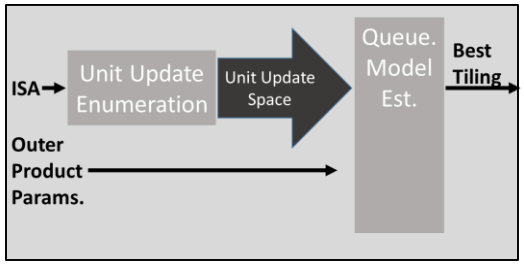
$C \leftarrow AB + C$

$C_{acc} \leftarrow 0$ Initialize

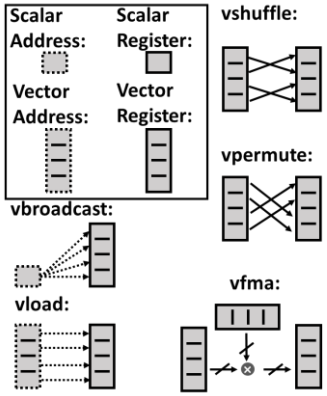
$C_{acc} \leftarrow \sum_p ab^T$ Compute

$C \leftarrow f(C_{acc})$ Accumulate

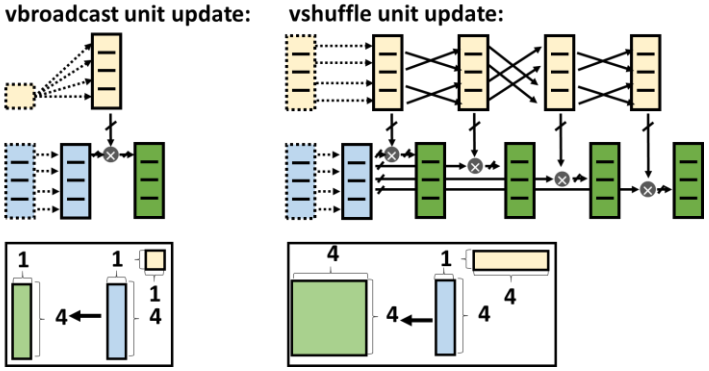
A High Throughput Mix



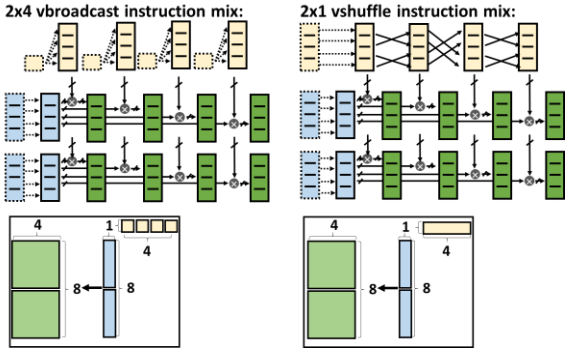
Start with ISA



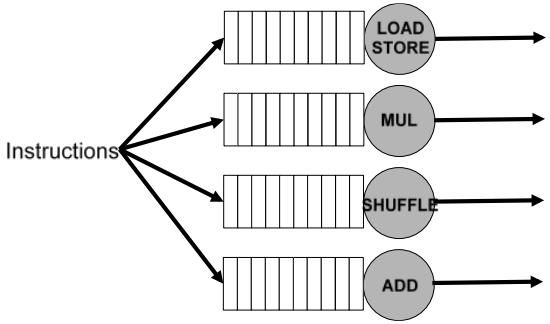
Identify Small Outer Products from ISA



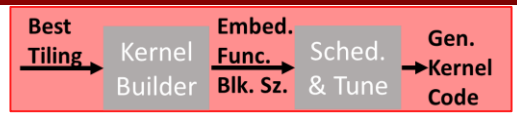
Enumerate Space of Outer Products



Select Best Mix with Queueing Model

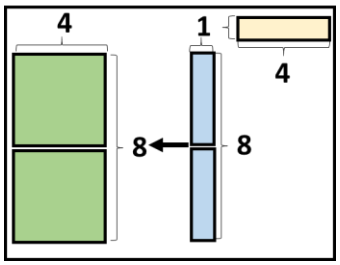


From Tiling to Template

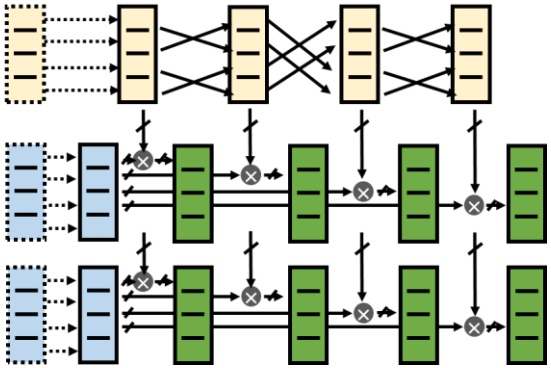


SPIRAL www.spiral.net

Selected Outer Product:



2x1 vshuffle instruction mix:



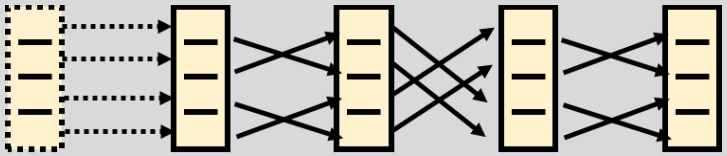
$C_{acc} \leftarrow 0$

```
for( i = 0; i <  m_r; i++ )
 for( j = 0; j <  n_r; j++ )
  init(c_reg, ii,jj )
```

$C_{acc} \leftarrow \sum_{p} ab^T$

```
for( pp = 0; pp < k_b; pp++ )
 /* perform the outer products */
 for( i = 0; i <  m_r; i+=m_s )
  for( j = 0; j <  n_r; j+=n_s )
   for( ii = i; ii < i+m_s; ii++ )
    get_a_elem(a_reg, ii,j );
    for( jj = j; jj < j+n_s; jj++ )
     get_b_elem(b_reg, ii,jj );
     apply(c_reg,a_reg,b_reg,ii,jj,pp);
```

Embedding Function (get_b):



```
def get_b_element( var array b_reg[][],
                   ptr B, ii, jj )
  opts = {
    0: assign( b_reg[jj], vload(B,jj)),
    1: assign( b_reg[jj], shuffle(b_reg[jj-1]) ),
    2: assign( b_reg[jj], permute(b_reg[jj-1]) ),
    3: assign( b_reg[jj], shuffle(b_reg[jj-1]) )}
  if ii mod v = 0
    return opts[jj]
```

$C \leftarrow f(C_{acc})$

```
for( i = 0; i <  m_r; i++ )
 for( j = 0; j <  n_r; j++ )
    accumulate(C, c_reg, ii, jj );
```

Floyd-Warshall Embedded in GEMM

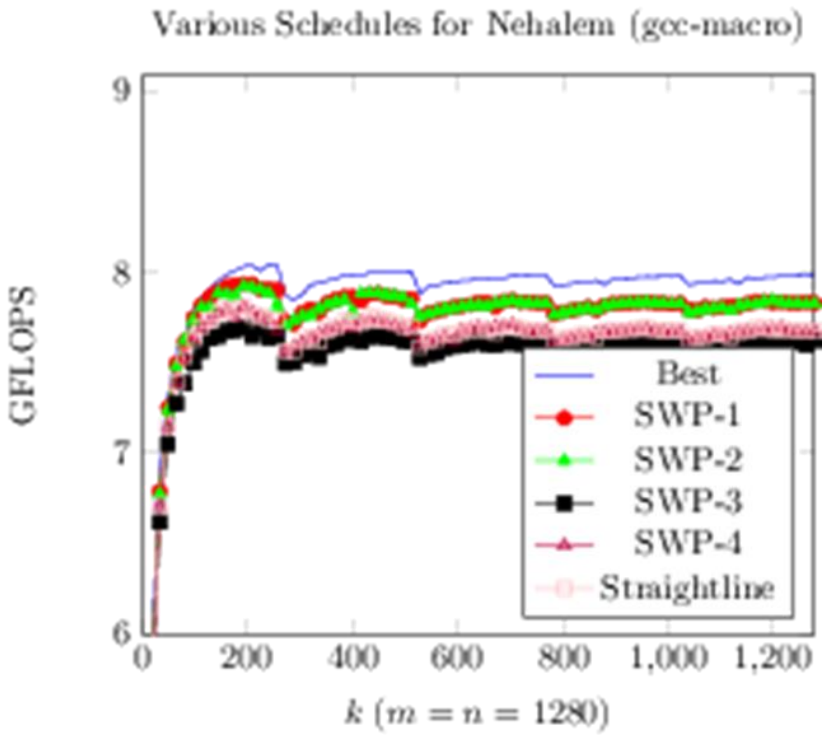
SPIRAL www.spiral.net

	DGEMM	Floyd-Warshall
Semiring:	$(R, *, +, 1, 0)$	$(Z, +, MIN, 0, \infty)$
Initialize: $C_{acc} \leftarrow 0$	`assign( c_reg[ii,jj], 0)`	`assign( c_reg[ii,jj], (ii==jj)? 0 : INFINITY)`
Compute: $C_{acc} \leftarrow \sum_p ab^T$	`assign( c_reg[ii,jj], a_reg[ii,pp]* b_reg[pp,jj], +c_reg[ii,jj]))`	`assign( c_reg[ii,jj], MIN(a_reg[ii,pp]+ b_reg[pp,jj], c_reg[ii,jj]))`
Accumulate: $C \leftarrow f(C_{acc})$	`assign( C[(ii,jj)], c_reg[ii,jj]+ C[(ii,jj)]`	`assign( C[(ii,jj)], MIN(c_reg[ii,jj], C[(ii,jj)] )`

Scheduling the Problem

We have built the kernel code,
Now we need to schedule:

Static Scheduling still matters on OOO Processors:



Pipeline for Scheduling:
Built Kernel +
Embedding Func.

↓

Express as Decision Vars

↓

Formulate Constraints over Vars

↓

Minimize with ILP Solver

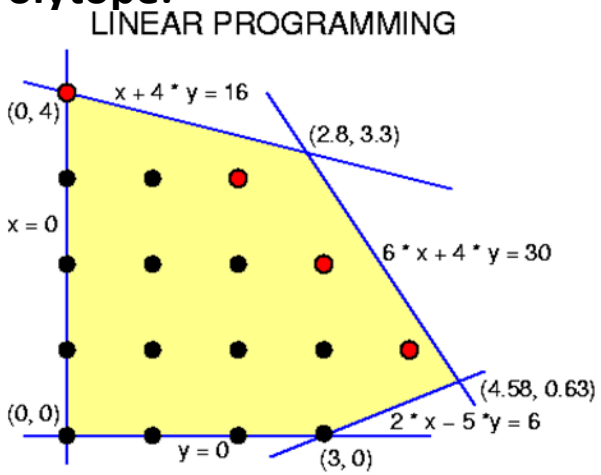
↓

Scheduled Kernel Code

OASIC approach for ILP Scheduling

SPIRAL www.spiral.net

Representing Design Space as Polytope:



Decision Variable: instruction n is executed on functional unit k at time step t

x_{nt}^{k}

Functional unit

cycle

Instruction label

Expressing Constraints in terms of X:

Every instruction **n is executed once**

$$\sum_k \sum_t x_{nt}^k = 1$$

At any timestep t, functional unit k is used no more than it can

$$\sum_k x_{nt}^k \leq R_k$$

If t_m depends on t_n, then t_m will not execute until l cycles after t_n

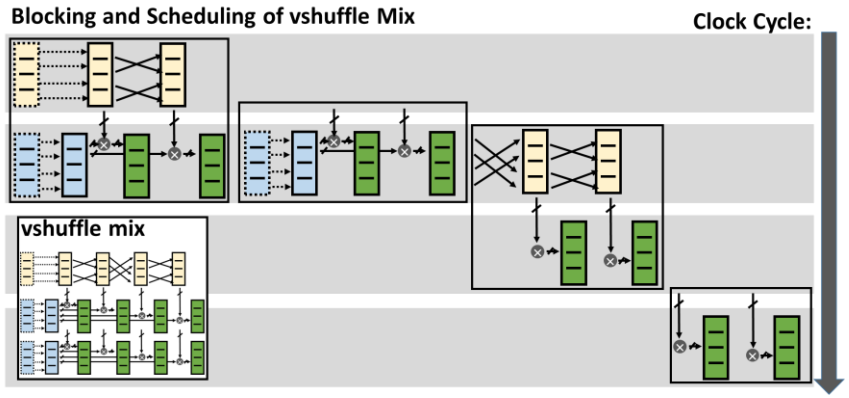
$$\sum_k \sum_t x_{nt_m}^k + \sum_k \sum_t x_{nt_m}^k \leq 1$$

But wait, there's more!

Emitting The Code

SPIRAL www.spiral.net

Code is now Scheduled



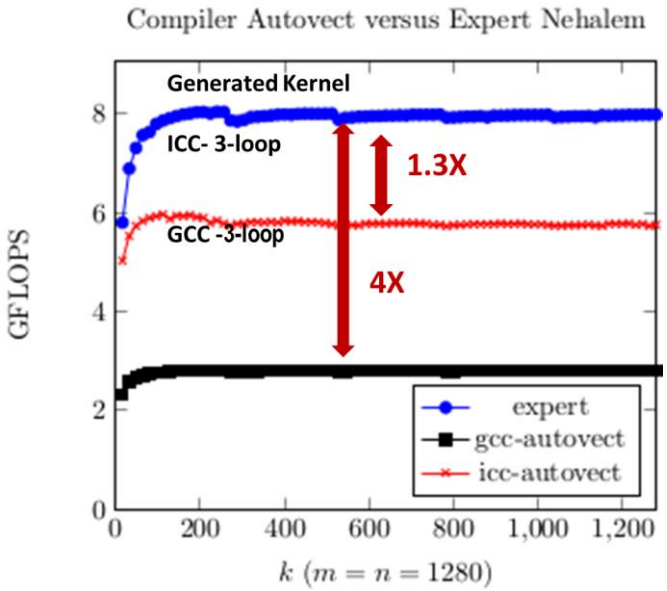
Code is emitted:

```
for( pp = 0; pp < k_b; pp+=KUNR )
 /* STEADY STATE CODE */
 VLOAD_IA(GET_A_ADDR(0),GET_A_REG(0))
 VLOAD_IA(GET_A_ADDR(1),GET_A_REG(1))
 VLOAD_IA(GET_B_ADDR(0),GET_B_REG(0))
 VSHUFFLE_IA(GET_B_REG(0),GET_B_REG(1))
 VFMA(GET_A_REG(0),
  GET_B_REG(0),GET_C_REG(0,0))
 VFMA(GET_A_REG(0),GET_B_REG(1),GET_C_REG(0,1))
 VPERM2F128_IA(0x01,GET_B_REG(1),GET_B_REG(2))
 VSHUFFLE_IA(0x05,GET_B_REG(2),GET_B_REG(3))
 VFMA(GET_A_REG(1),GET_B_REG(0),GET_C_REG(0,0))
 VFMA(GET_A_REG(1),GET_B_REG(1),GET_C_REG(0,1))
```

Need Custom ANSI C compliant SIMD wrappers to schedule in Compiler:

```
#define VADD(srca,srcb,dest)
  asm volatile(
  "vaddpd %[vsrca],%[vsrcb],
          %[vdest]"
  : [vdest] "=x"(dest)
  : [vsrca] "x"(srca),
    [vsrcb] "x"(srcb));
```

Is this necessary?



Putting it All Together

SPIRAL www.spiral.net

Have Operation that we can express like GEMM:

$$C \leftarrow AB + C$$

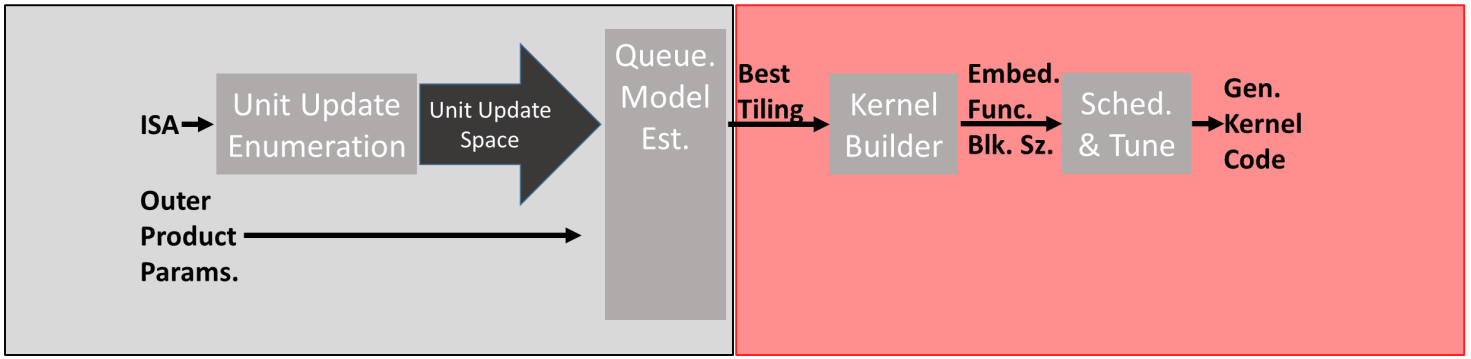
$C_{acc} \leftarrow 0$ Initialize

$C_{acc} \leftarrow \sum_p ab^T$ Compute

$C \leftarrow f(C_{acc})$ Accumulate

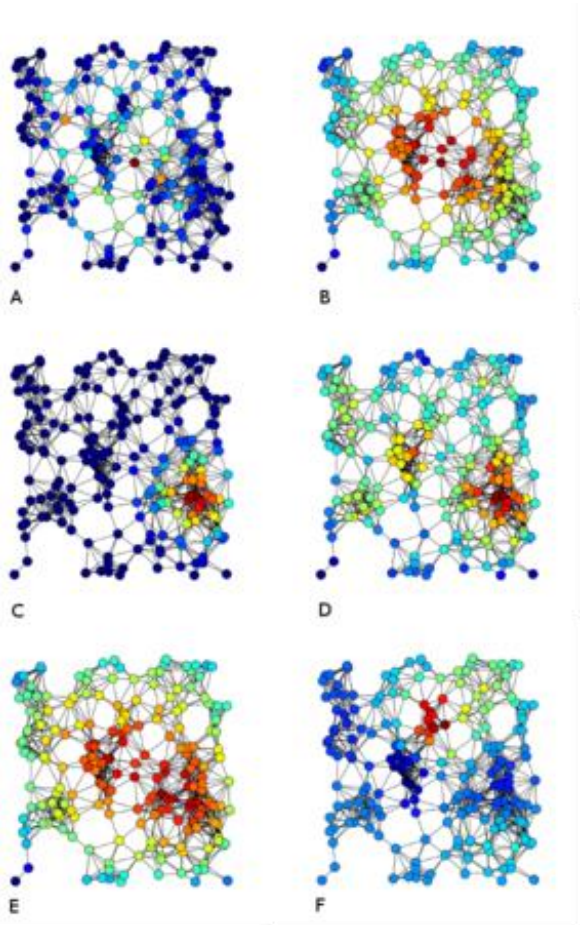
	DGEMM	Floyd-Warshall
Semi-ring:	$(Double, *, +, 1, 0)$	$(Z, +, MIN, 0, 1)$
Initialize: $C_{acc} \leftarrow 0$	`assign( c_reg[ii,jj], 0)`	`assign( c_reg[ii,jj], (ii==jj)? 0 : INFINITY)`
Compute: $C_{acc} \leftarrow \sum_p ab^T$	`assign( c_reg[ii,jj], a_reg[ii,pp]* b_reg[pp,jj], +c_reg[ii,jj]))`	`assign( c_reg[ii,jj], MIN(a_reg[ii,pp]+ b_reg[pp,jj], c_reg[ii,jj]))`
Accumulate: $C \leftarrow f(C_{acc})$	`assign( C[(ii,jj)], c_reg[ii,jj]+ C[(ii,jj)]`	`assign( C[(ii,jj)], MIN(c_reg[ii,jj], C[(ii,jj)] )`

Run through our Pipeline:

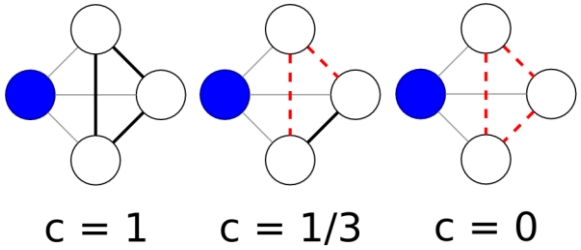


Moving Forward

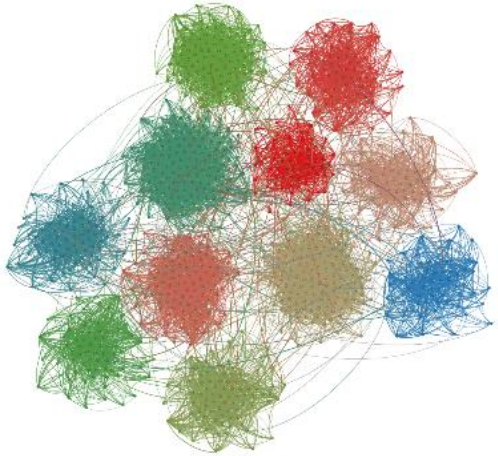
Centrality: $\sim A^T A$ (betweeness)



Clustering: $\sim AA^T A$ (triangle)



Community Detection: $\sim A^K$



SPIRAL www.spiral.net

Summary

- There exists a **large class** of **GEMM-like Operations**
- Obtaining DGEMM level **performance** for each of these operations **requires automation**
- We have a **systematic approach** for automatically **generating DGEMM**
- We are extending it by allowing the user to **define a semi-ring** with an initialize and accumulate function