An Analytical Model for a GPU
Overview
SVM Kernel Behavior: Need for other metrics
Degree of Parallelism

- **GPU Architecture**
  - Each SM executes multiple warps in a time-sharing fashion while one or more are waiting for memory values
    - Hiding the execution cost of warps that are executed concurrently.

- How many memory requests can be serviced and how many warps can be executed together while one warp is waiting for memory values.
MWP and CWP

- **Memory Warp:**
  - The warp that is waiting for memory values

- **Memory Warp waiting period:**
  - The time period from right after one warp sent memory requests until all the memory requests from the same warp are serviced.

- **CWP (Computation Warp Parallelism)**
  - Represents the number of warps that the SM processor can execute during one memory warp waiting period plus one.

- **MWP (Memory Warp Parallelism)**
  - Represents the maximum number of warps per SM that can access the memory simultaneously during memory warp waiting period
Relationship between MWP and CWP: CWP > MWP

What is getting hidden?

Total execution time (a) 8 warps (b) 4 warps

What is getting hidden?
What is going on here?

\[ \text{Exec\_cycles} = \text{Mem\_cycles} \times \frac{N}{\text{MWP}} + \text{Comp\_p} \times \text{MWP} \] (1)

\[ \text{Comp\_p} = \frac{\text{Comp\_cycles}}{\#\text{Mem\_insts}} \] (2)

\textit{Mem\_cycles}: Memory waiting cycles per each warp (see Equation (18))

\textit{Comp\_cycles}: Computation cycles per each warp (see Equation (19))

\textit{Comp\_p}: execution cycles of one computation period

\textit{\#Mem\_insts}: Number of memory instructions

\textit{N}: Number of active running warps per SM
Relationship between MWP and CWP: MWP > CWP

Case3: CWP = 4
MWP = 8
8 Computation + 1 Memory

Case4: 8 Computation + 1 Memory
1st Memory period
2nd Memory period
1st Computation period
2nd Computation period

\[ \text{Exec\_cycles} = \text{Mem\_p} + \text{Comp\_cycles} \times N \]

Total execution time (a) 8 warps (b) 4 warps
Relationship between MWP and CWP: MWP > CWP

Case 5:

CWP < 2
MWP = 8

8 Computation + 1 Memory

Total execution time (8 warps)
Not Enough Warps Running

Case 6:

\[ \text{1 1 1 1 1 1 1 1} \]

8 Computation + 8 Memory

(a)

Case 7:

\[ \text{1 1 1 1 1 1 1 1} \]

\[ \text{2 2 2 2 2 2 2 2} \]

5 Computation + 4 Memory

(b)

\[ \text{Memory period} \quad \text{Computation period} \]

\[ \text{Exec\_cycles} = \text{Mem\_cycles} \times \frac{N}{MWP} + \text{Comp\_cycles} \times \frac{N}{MWP} + \text{Comp\_p}(MWP - 1) \]
CPI

\[ CPI = \frac{\text{Exec}_{\text{cycles}}_{\text{app}}}{\text{#Total}_{\text{insts}} \times \frac{\text{#Threads}_{\text{per block}}}{\text{#Threads}_{\text{per warp}}} \times \frac{\text{#Blocks}}{\text{#Active}_{\text{SMs}}}} \]

PTX instruction set
# Model

\[
\#Rep = \frac{\#Blocks}{\#Active\_blocks\_per\_SM \times \#Active\_SMs}
\]

If (MWP is N warps per SM) and (CWP is N warps per SM)

\[
Exec\_cycles\_app = (\text{Mem\_cycles} + \text{Comp\_cycles} + \frac{\text{Comp\_cycles}}{\#\text{Mem\_insts}} \times (MWP - 1)) \times \#Rep
\]

If (CWP \geq MWP) or (\text{Comp\_cycles} > \text{Mem\_cycles})

\[
Exec\_cycles\_app = (\text{Mem\_cycles} \times \frac{N}{MWP} + \frac{\text{Comp\_cycles}}{\#\text{Mem\_insts}} \times (MWP - 1)) \times \#Rep
\]

If (MWP \gt CWP)

\[
Exec\_cycles\_app = (\text{Mem\_L} + \text{Comp\_cycles} \times N) \times \#Rep
\]
Example

Blocks: 80
Threads: 128
5 blocks per SM

```c
1: MatrixMulKernel<<<80, 128>>> (M, N, P);
2: ....
3: MatrixMulKernel(Matrix M, Matrix N, Matrix P)
4: {
5:     // init code ...
6:     
7:     for (int a=starta, b=startb, iter=0; a<=enda;
8:         a+=steapa, b+=stepb, iter++)
9:         {
10:             __shared__ float Msub[BLOCKSIZE][BLOCKSIZE];
11:             __shared__ float Nsub[BLOCKSIZE][BLOCKSIZE];
12:             
13:             Msub[ty][tx] = M.elements[a + wM * ty + tx];
14:             Nsub[ty][tx] = N.elements[b + wN * ty + tx];
15:             
16:             __syncthreads();
17:             
18:             for (int k=0; k < BLOCKSIZE; ++k)
19:                 subsum += Msub[ty][k] * Nsub[k][tx];
20:             
21:             __syncthreads();
22:         }
23:         
24:         int index = wN * BLOCKSIZE * by + BLOCKSIZE
25:         P.elements[index + wN * ty + tx] = subsum;
26: }
```

1: ...
2:     // Init Code
3:         
4:         $OUTERLOOP:
5:         ld.global.f32 %f2, [%rd23+0]; //
6:         st.shared.f32 [%rd14+0], %f2; //
7:         ld.global.f32 %f3, [%rd19+0]; //
8:         st.shared.f32 [%rd15+0], %f3; //
9:         bar.sync 0; // Synchronization
10:        ld.shared.f32 %f4, [%rd8+0]; // Innerloop unrolling
11:        ld.shared.f32 %f5, [%rd6+0]; //
12:        mad.f32 %f1, %f4, %f5, %f1; //
13:        // the code of unrolled loop is omitted
14:        bar.sync 0; // synchronization
15:        setp.le.s32 %p2, %r21, %r24; //
16:        @%p2 bra $OUTERLOOP; // Branch
17:        ...
18:         // Index calculation
19:        st.global.f32 [%rd27+0], %f1; // Store in P.elements
```

```c
```