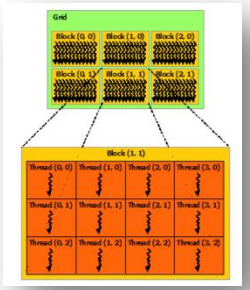
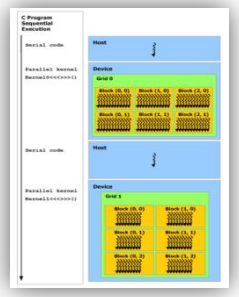
WC: A New GPU Programming Model

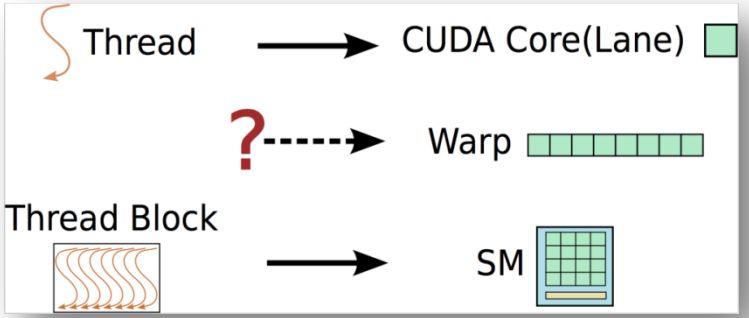
Ang Li, Pacific Northwest National Laboratory (PNNL)
Weifeng Liu (Norwegian University of Science and Technology)
Linnan Wang (Brown University)
Kevin Barker and Shuaiwen Leon Song (PNNL)

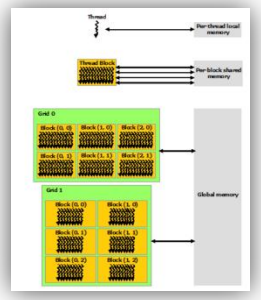
Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle *Since 1965*

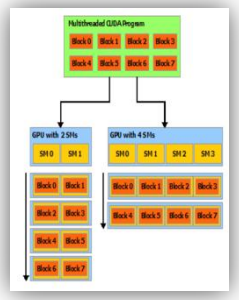
The Missing Warp in CUDA





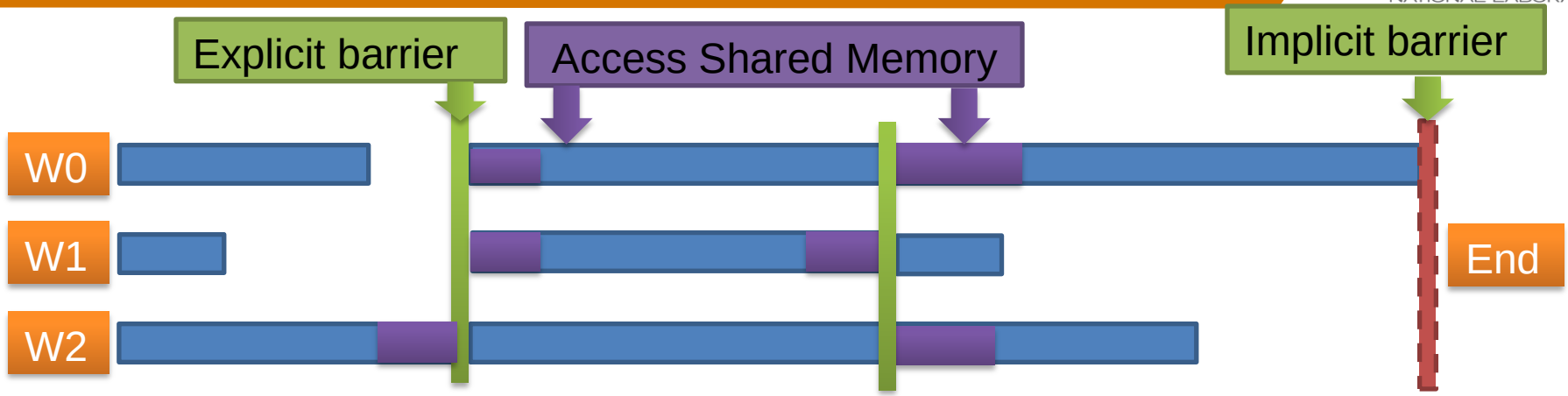




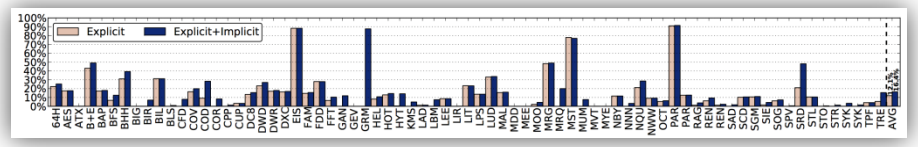


- CUDA's 2-level programming model
  - Thread-block ➔ SM, Thread ➔ CUDA core
- GPU Hardware's 3-level execution model
  - SM, warp, SIMD-lane
- CUDA programming/execution model mismatch
- Warp cannot be simply ignored
  - Warp divergence
  - Inter-warp synchronization

Inter-warp Synchronization



- Explicit Synchronization
- Implicit Synchronization



12% (up to 90%) explicit stalls, 4.4% implicit stalls for 80 GPGPUs applications on P100

Issues with CUDA Programming Model

Large & correlated design space

- Difficult to program
  - Setting kernel configuration <<<grid(x,y,z), block(x,y,z)>>>
  - Related to resources allocation: register, shared-mem, warp-slots
- Difficult to optimize
  - Without knowing “warp” it’s hard to achieve optimal performance
  - Thinking about warp, program about warp, but never express warp
- Synchronization overhead
  - Warp divergence/unbalancing
  - Explicit warp synchronization
  - Implicit warp syncrhonization

CUDA’s programming/execution model mismatch

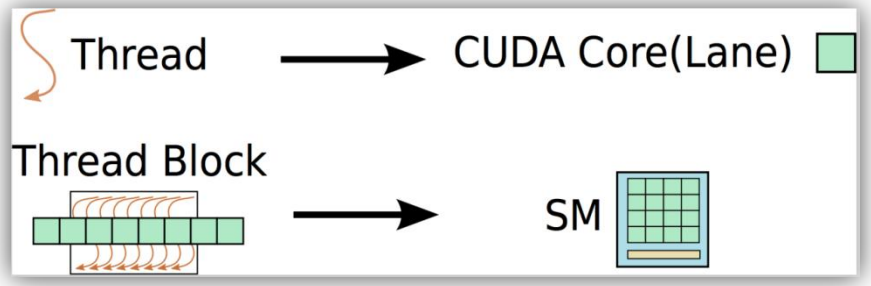
How to resolve the “mismatch” issue?

Pacific Northwest NATIONAL LABORATORY

- Warp-Consolidation Model
  - 5 advantages over CUDA model
  - 3 code examples
  - 2 drawbacks & solution
  - 2 comparisons with other techniques
- Evaluation
  - Settings
  - Results
- Potential Extension
  - Active context switch
  - Volta

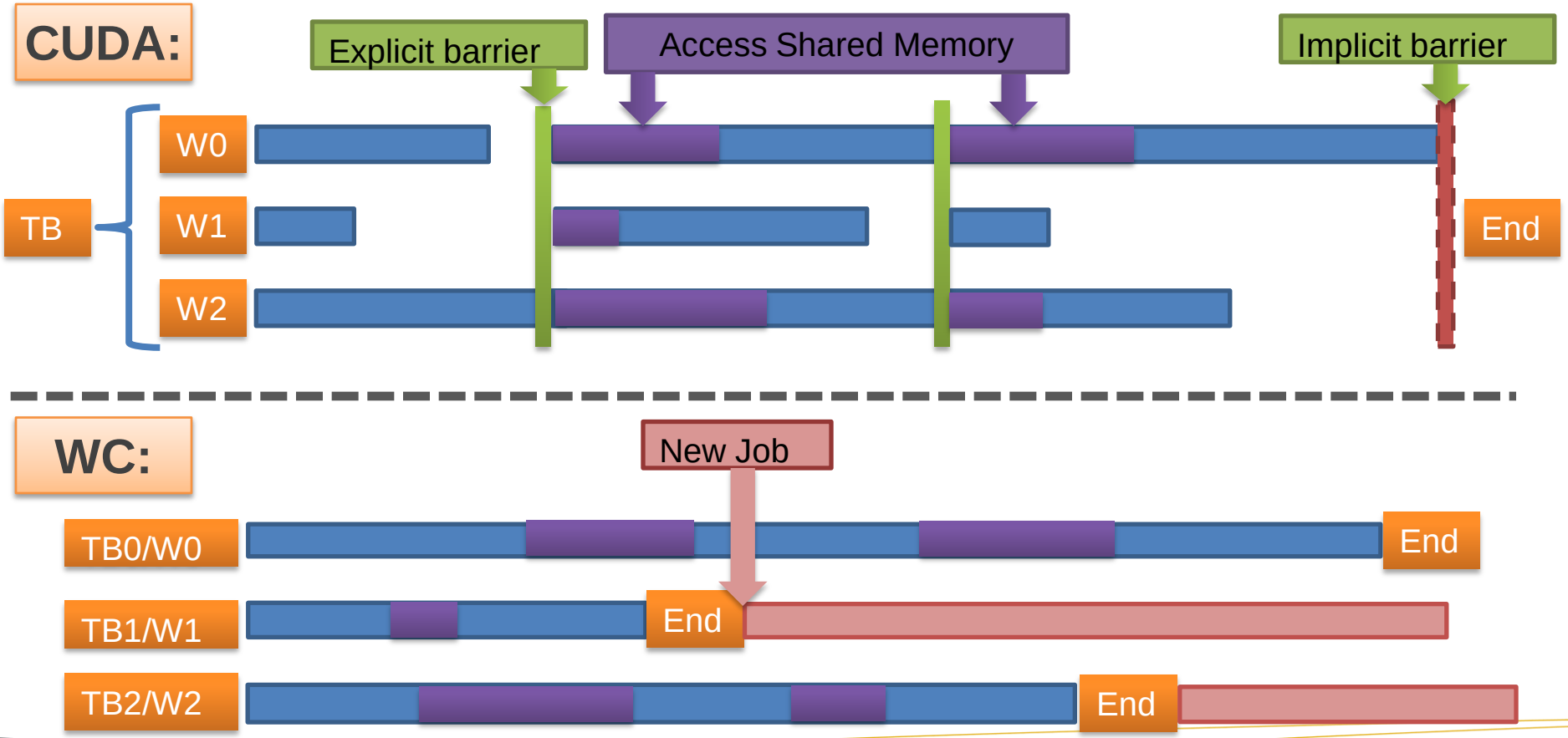
Warp Consolidation Model

- WC Model
  - Unify TB and warp: 1-warp/thread-block
  - 2-level programming/execution model



- Combine the advantages
  - thread-block(free resource allocation)
  - warp (fast communication, synchronization and cooperation)
- Motivates novel SMT-like programming concept for GPU
  - Partition computation space into multiple independent warp based jobs. A job is handled by a warp
  - Threads in a TB/warp can efficiently sync, cooperate and communicate

Adv 1: No synchronization & barrier



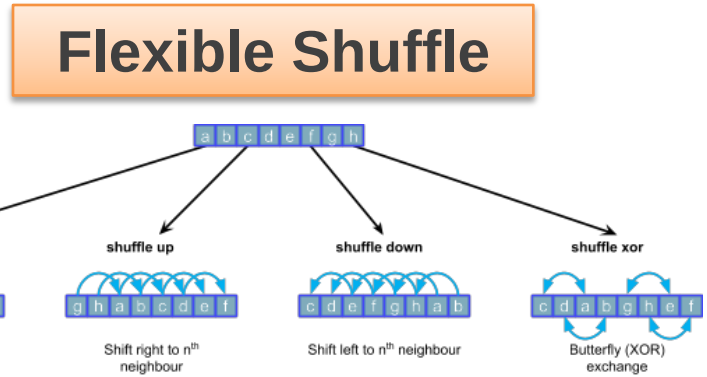
Adv 2: Simplified Design Space

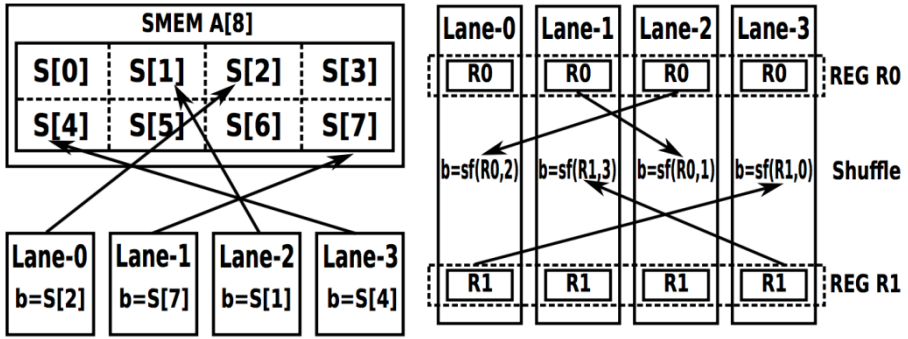
Pacific Northwest NATIONAL LABORATORY

- *CUDA Programming Model* **Hard to program**
  - *Large design space <<<grid(x,y,z), block(x,y,z)>>>*
  - *Correlated resources usage: register, shared-mem, warp-slots, etc.*
- WC Programming Model **Easy to program**
  - Fixed block configuration <<<grid(x,y,z), 32>>>
  - Fixed per-warp resources ➔ No correlated resource allocation
  - Significantly reduces design space
  - CPU-like programming: simultaneous multi-threaded SSE code

Adv 3: Register fast communication

- Shared memory communication:
  - reg➔write➔sync➔read➔reg
- Register shuffle communication:
  - reg➔shuffle➔reg
  - Very flexible
- Communication
  - CUDA: collective shared memory communication
  - WC: distributive message passing communication



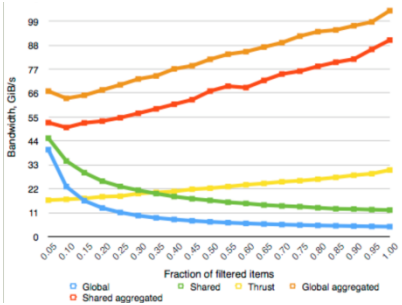


Adv 4: Fine-grained cooperation

Pacific Northwest NATIONAL LABORATORY

- __any()
- __all()
- __ballot()
- __ffs()
- __clz()
- __popc()

Warp-aggregated Atomics

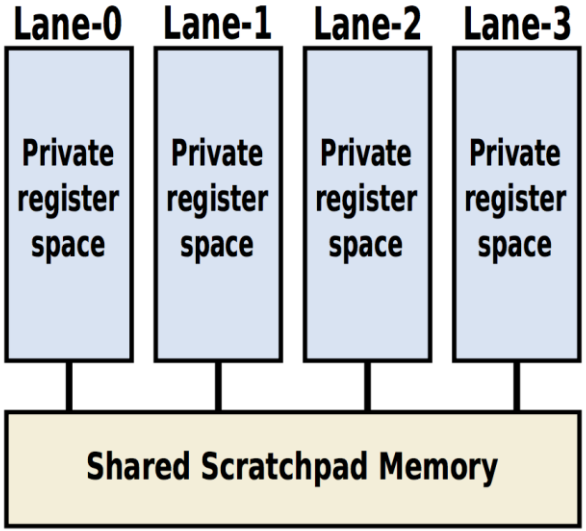


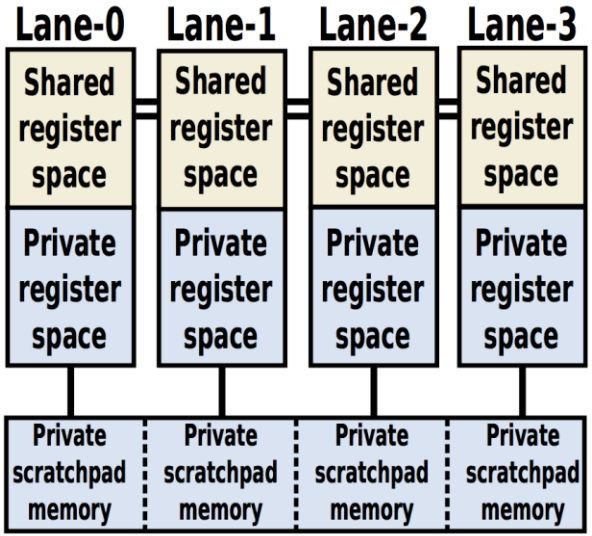

```
1 __device__
2 int atomicAggInc(int *ctr) {
3   int mask = __ballot(1);
4   int leader = __ffs(mask) - 1;  // select the leader
5   int res;
6   if(lane_id() == leader)  // leader does the update
7     res = atomicAdd(ctr, __popc(mask));
8   res = __shfl(res, leader);
9   // each thread computes its own value
10  return res + __popc(mask & ((1 << lane_id()) - 1));
11 }
```


[1] M. Bauer et al, *Singe: Leveraging warp specialization for high performance on GPUs,* PPoPP-14

[2] Andy Adinets, *Optimizing Filtering with Warp-Aggregated Atomics*, https://devblogs.nvidia.com/cuda-pro-tip-optimized-filtering-warp-aggregated-atomics/

Adv 5: Extended Register Space



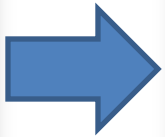


- Register spilling in shared memory
  - Register shuffling relax the usage of shared memory as comm buffer
  - Shared memory allocation is now private a warp
  - No shared memory bank conflict

Exp 1: How to program in WC

Pacific Northwest NATIONAL LABORATORY

```
__global__ void Kernel(...){
  __shared__ sh[N];
  for(...){
    __syncthreads();
    __threadfence_block();
    ...}
}
num_thds = 512;//16 warps
num_blks = (int)ceil(num_nodes/
        (double)num_thds;
dim3 grid(num_blks,1,1);//grid config
dim3 thds(num_thds,1,1);//blk config
do{
  Kernel<<<grid,thds>>>(...)
while(stop);
```



```
__global__ void Kernel(...){
  __shared__ volatile sh[N];//volatile smem
  for(...){
    //sync is avoided
    //blockwise memory fence is avoided
    ...}
}
num_thds = 32;//reduce to one warp
num_blks = (int)ceil(num_nodes/
        (double)num_thds;//num of blks x16
dim3 grid(num_blks,1,1);//grid config
dim3 thds(num_thds,1,1);//blk config
do{
  Kernel<<<grid,thds>>>(...)
while(stop);
```

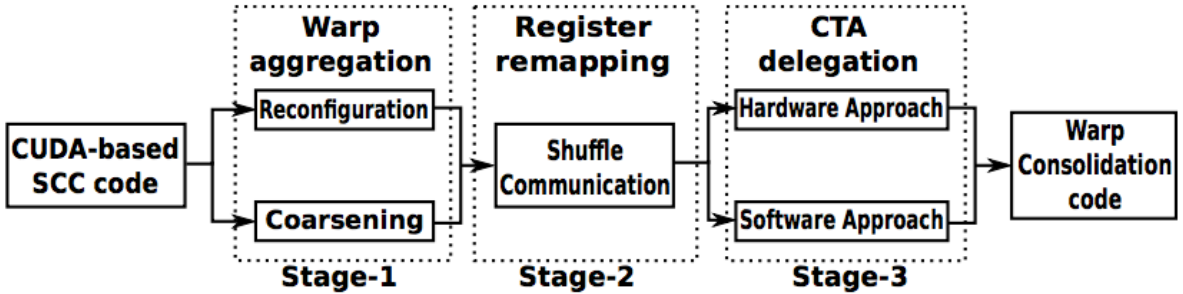
- Synchronization is removed
- Shared memory allocate as volatile
- Thread block configuration == 32

Exp 2: How to transform legacy code

Pacific Northwest NATIONAL LABORATORY

```
if(i0<n_vectors) //replicate or use "for"
 d_output[i0]=(float)X*k_2powneg32;
unsigned v_log2stridem1=v[__ffs(stride)-2];
unsigned v_stridemask=stride-1;
//"for" statement need to replicated,
//can be fused later if possible
for(int i=i0+stride;i<n_vectors;i+=stride){
 X^=v_log2stridem1^v[__ffs(~((i-stride)|
     v_stridemask))-1];
 d_output[i]=(float)X*k_2powneg32;
}}
```

```
if(i0_0<n_vectors)
   d_output[i0_0]=(float)X_0*k_2powneg32;
if(i0_1<n_vectors)
   d_output[i0_1]=(float)X_1*k_2powneg32;
unsigned v_log2stridem1=v[__ffs(stride)-2];
unsigned v_stridemask=stride-1;
for(int i=i0_0+stride;i<n_vectors;i+=stride){
    X_0^=v_log2stridem1^v[__ffs(~((i-stride)|
        v_stridemask))-1];
    d_output[i]=(float)X_0*k_2powneg32;
}
for(int i=i1_0+stride;i<n_vectors;i+=stride){
    X_1^=v_log2stridem1^v[__ffs(~((i-stride)|
       v_stridemask))-1];
    d_output[i+32]=(float)X_1*k_2powneg32;
}}
```



Input : Original Kernel Code
Output : Warp-Consolidation based Optimized Kernel Code
Stage-1 `Warp Aggregation(`*kernel code*`)`:
- **if** *CTA-size can be adjusted to 32 threads* **then**
  - Adjust CTA-size to 32 threads;
- **else**
  - Perform aggressive warp coarsening;
  - **for** *each code block separated by __syncthreads()* **do**
    - Replicate each var reused across the code block;
    - Replicate statements depending on *threadIdx.x*;
    - Convert *"if"* statements to *"for"* statements;
    - Replicate *"for"* and *"while"* block;
    - Perform loop fusion if possible;
  - **end**
- **end**
- Remove all sync statements;
- **return** optimized code after warp-aggregation;

Stage-2 `Register Remapping(`*kernel code*`)`:
- **if** *Fixed s-mem access pattern and no addr calculation* **then**
  - Remove all shared memory allocation;
  - Partition smem space into warp-based chunks;
  - Allocate these chunks in different registers;
  - Replace smem references by shuffle instructions;
- **end**
- **return** optimized code after register-remapping;

Stage-3 `Warp Delegation(`*kernel code*`)`:
- Include the *Warp_Delegation.cuh* header file;
- Pack kernel body into the *WARP_DELEGATION* macro;
- Set partition strategy;
- Convert kernel invokation;
- **return** optimized code after warp-delegation;

Exp 3: How to communicate via reg

```
for(int k=1;k<nz-1;k++){
 float a_left_right,a_up,a_down;
 if((i<nx)&&(j<ny))
   top=A0[Index3D(nx,ny,i,j,k+1)];
 if(w_region){
  ...
  a_left_right=x_l_bound?A0[Index3D(
       nx,ny,i-1,j,k)]:sh_A0[sh_id-1];
  Anext[Index3D(nx,ny,i,j,k)]=(top+bottom
     +a_up+a_down+sh_A0[sh_id+1]
     +a_left_right)*c1-sh_A0[sh_id]*c0;
 }
 ...
  __syncthreads();
  bottom=sh_A0[sh_id];
  sh_A0[sh_id]=top;
  __syncthreads();
```

```
//shuffle is outside branch
a_left=__shfl_up(r0_A0,1);//left neighbor
a_right=__shfl_down(r0_A0,1);//right neighbor
a_other=__shfl(r1_A0,0);//2nd reg space
if(w_region){
 ...
 a_left=x_l_bound?A0[Index3D(nx,ny,i-1,j,k)]
        :a_left;
 a_right=(threadIdx.x==31)?a_other:a_right;
 Anext[Index3D(nx,ny,i,j,k)]=(top+bottom
    +a_up+a_down+a_left
    +a_right)*c1-r0_A0*c0;
}
...
 bottom=r0_A0;
 r0_A0=top;
```

- Shuffle is very flexible (up, down, shfl)
- Be careful when shuffle with warp divergence
- No synchronization required

Stencil from Parboil Benchmark Suite.

Drawback 1: Occupancy degradation

Pacific Northwest NATIONAL LABORATORY

GPU	Architecture	SMs	Thread-Blocks/SM	Warps/SM
Tesla-K80	Kepler	15	16	64
Tesla-M40	Maxwell	24	32	64
Tesla-P100	Pascal	56	32	64
Tesla-V100	Volta	80	32	64

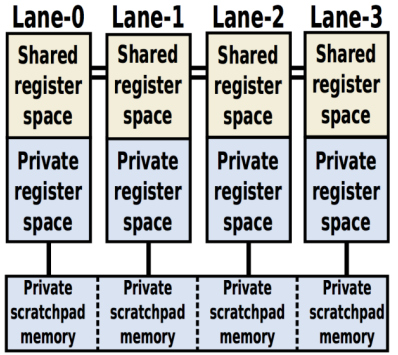
- Occupancy is at maximum 0.5
  - Resolved by warp-delegation (a variant of [1])
  - CUDA Best Practice Guide (50% occupancy is sufficient)
  - We hope TBs/SM == Warps/SM in future GPUs

[1] A. Li et al. "**Locality-Aware CTA Clustering for Modern GPUs**." in *ASPLOS-17, ACM*

Drawback 2: Shared Mem/Reg usage

Pacific Northwest NATIONAL LABORATORY

- Extra shared mem usage
  - Ideally can be mitigated & released by register shuffling
  - Otherwise, multiplexing can be an alternative solution[1]
- Extra register usage
  - Our experiment show that reg usage not increase much
  - Can be resolved by shared memory spilling



[1] Y. Yang et al. ”**Shared memory multiplexing**: a novel way to improve GPGPU throughput" in *PACT-12, ACM*

Comparison 1: Thread Coarsening

Pacific Northwest NATIONAL LABORATORY

- Fuse multi-threads so per-thread workload increases but thread number decrease

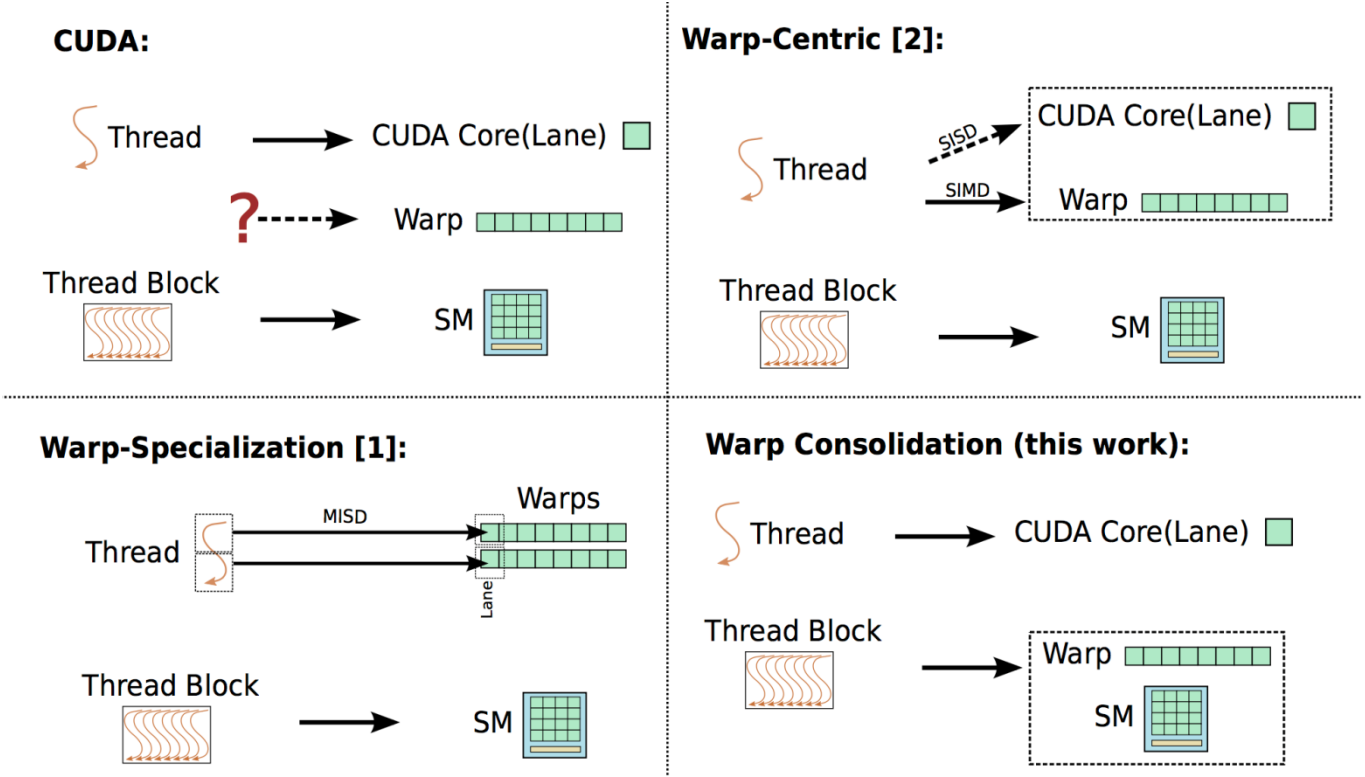
Thread Coarsening

- Increase ILP
- Register reuse
- Reduce auxiliary inst
- Coarsening factor

Warp Consolidation

- An approach to reduce warps/TB to 1 for legacy CUDA code
- Coarsening === 1

Comparison 2: Warp Specialization



[1] M. Bauer et al. "**Singe: leveraging warp specialization for high performance on GPUs**." in *PPoPP-14, ACM*

[2] S. Hong et al. "**Accelerating CUDA graph algorithms at maximum warp**" in *PPoPP-11, ACM.*

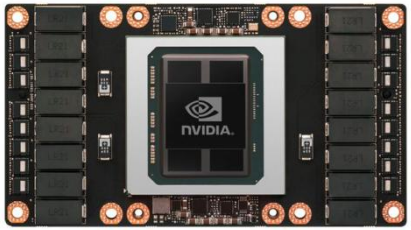
How to resolve the “mismatch” issue?

Pacific Northwest NATIONAL LABORATORY

- Warp-Consolidation Model
  - 5 advantages over CUDA
  - 3 examples
  - 2 drawbacks & solution
  - 2 comparisons with other techniques
- Evaluation
  - Settings
  - Results
- Potential Extension
  - Active context switch
  - Volta

Evaluation Settings

Pacific Northwest NATIONAL LABORATORY



Tesla-K80 Kepler

Tesla-M40 Maxwell

Tesla-P100 Pascal

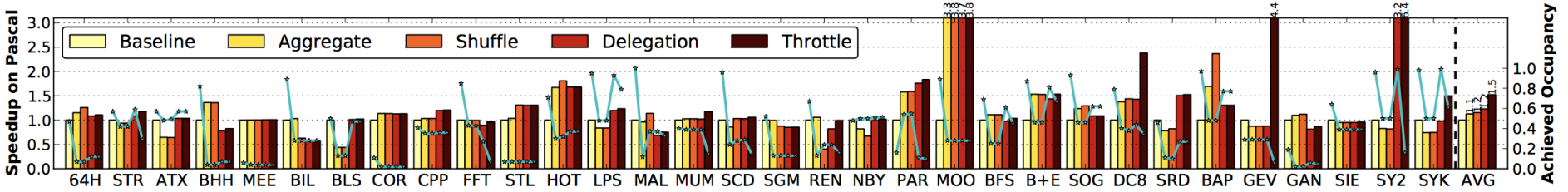
Tesla-V100 Volta

Application	Description	abbr.	Kernel Name	CTAs	WPs	Regs	SMem	Sync	Comm	Ref
64H	64 and 256 bin histogram calculation	64H	*mergeHistogram256Kernel()*	256	8	15	1000B	Y	Y	[32]
streamcluster	Assigning points to nearest centers	STR	*kernel_compute_cost()*	128	16	25	0B	N	N	[33]
atax	Matrix transpose and vector multiply	ATX	*atax_kernel2()*	256	8	18	0B	N	N	[34]
bh	Gravitational forces via Barnes-Hut method	BHH	*BoundingBoxKernel()*	168	32	32	24000B	Y	Y	[35]
mersenne	Mersenne Twister random generator	MEE	*BoxMullerGPU()*	32	4	18	0B	N	N	[32]
binomial	Option call price via binomial model	BIL	*binomialOptionsKernel()*	1024	8	17	2007B	Y	Y	[32]
BlackScholes	Option call price via Black-Scholes model	BLS	*BlackScholesGPU()*	480	4	21	0B	N	N	[32]
corr	Correlation computation	COR	*corr_kernel()*	8	8	22	0B	N	N	[34]
cutcp	Compute Coulombic potential for 3D grid	CPP	*cenergy()*	512	4	22	0B	N	N	[36]
fft	Fast Fourier transform	FFT	*IFFT512_device()*	32768	2	56	4500B	Y	Y	[37]
stencil	Jacobi stencil operation on regular 3D grid	STL	*block2D_hybrid_coarsen_x()*	64	4	29	1000B	Y	Y	[36]
hotspot	Estimate processor temperature	HOT	*calculate_temp()*	7396	8	38	3000B	Y	Y	[33]
lps	3D Laplace solver	LPS	*GPU_laplace3d()*	5000	4	16	2390B	Y	Y	[38]
matrixMul	Matrix multiplication	MAL	*matrixMulCUDA()*	16384	32	27	8000B	Y	Y	[32]
mum	Pairwise local sequence alignment for DNA	MUM	*mummergpuKernel()*	196	8	23	0B	N	N	[38]
scalarProd	Scalar products of input vector pairs	SCD	*scalarProdGPU()*	4096	8	24	4000B	Y	Y	[32]
sgemm	Single precision general matrix multiply	SGM	*mysgemmNT()*	496	4	46	512B	Y	Y	[36]
recursiveGaussian	Recursive Gaussian filter	REN	*d_transpose()*	1024	8	10	1062B	Y	Y	[32]
nbody	All-pairs gravitational N-body simulation	NBY	*integrateBodies()*	224	8	36	4000B	Y	Y	[32]
pathfinder	Dynamically finding a path in 2D grid	PAR	*dynproc_kernel()*	3334	1	13	256B	Y	Y	[33]
MonteCarlo	Option call price via Monte-Carlo method	MOO	*MonteCarloReduce()*	1024	8	29	4000B	Y	Y	[32]
bfs	Breadth first search	BFS	*Kernel()*	1954	16	16	0B	Y	N	[33]
b+tree	B+tree Operation	B+E	*findK()*	10000	8	27	0B	Y	N	[33]
SobolQRNG	Sobel edge detection filter for images	SOG	*sobolGPU_kernel()*	25600	2	19	128B	Y	Y	[32]
dct8x8	Discreite cosine transform for 8x8 block	DCB	*CUDAkernel1DCT()*	4096	2	18	500B	Y	Y	[32]
srad	Speckle reducing anisotropic diffusion	SRD	*reduce()*	2048	16	25	4000B	Y	Y	[33]
backprop	Perceptron back propagation	BAP	*bpnn_layerforward_CUDA()*	32768	8	18	1062B	Y	Y	[33]
gesummv	Scalar vector and matrix multiplication	GEV	*gesummv_kernel()*	128	8	23	0B	N	N	[34]
gaussian	Solving variables in a linear system	GAN	*Fan1()*	2	16	12	0B	N	N	[33]
single	Monte Carlo single Asian option	SIE	*initRNG()*	782	4	32	0B	Y	Y	[32]
syr2k	Symmetric rank-2k operations	SY2	*syr2k_kernel()*	16384	8	19	0B	N	N	[34]
syrk	Symmetric rank-k ooperations	SYK	*syrk_kernel()*	4096	8	28	0B	N	N	[34]

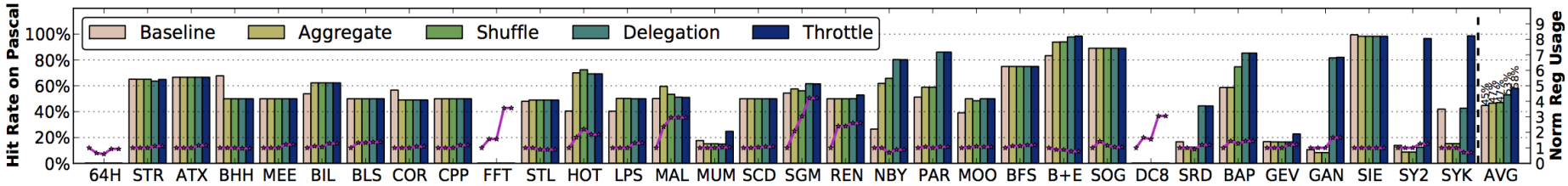
32 Applications from commonly used GPGPU benchmark suits

Evaluation Results

Pacific Northwest NATIONAL LABORATORY



On average 1.7x, 2.3x, 1.5x and 1.2x over Tesla K80 (Kepler), M40 (Maxwell), P100 (Pascal) and V100 (Volta)



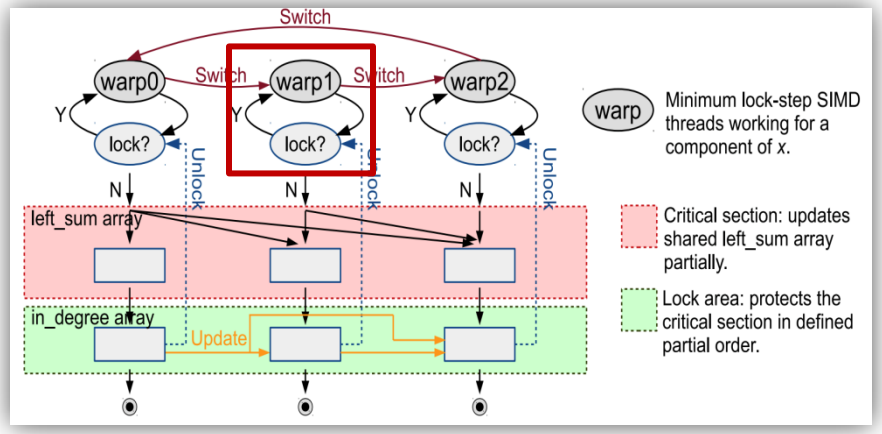
Performance gain is not directly from cache by reducing sync, communication and cooperation (SCC) overhead

How to resolve the “mismatch” issue?

Pacific Northwest NATIONAL LABORATORY

- Warp-Consolidation Model
  - 5 advantages over CUDA
  - 3 examples
  - 2 drawbacks & solution
  - 2 comparisons with other techniques
- Evaluation
  - Settings
  - Results
- Potential Extension
  - Active context switch
  - Volta

Extension 1: Active context switch



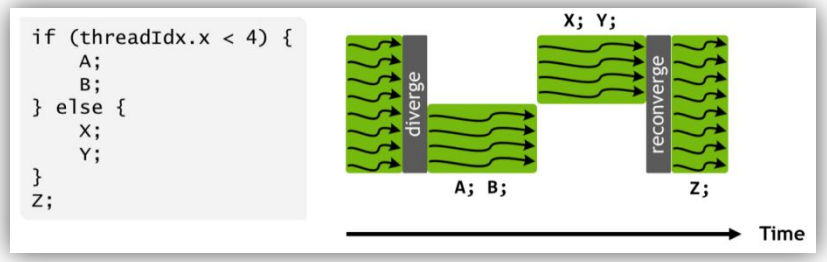
while $\texttt{s\_in\_degree}[i]+1 \neq \texttt{d\_in\_degree}[i]$ **do**
//busy wait
end while

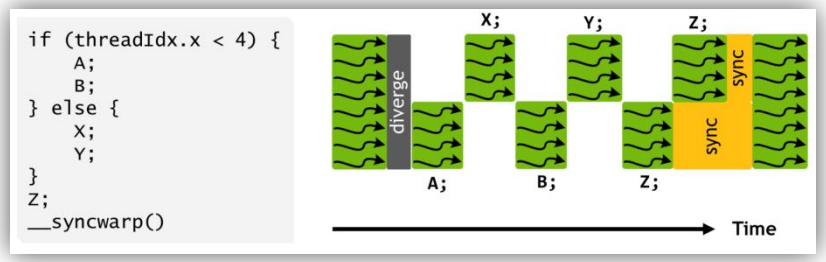
Not a deadlock if contains "clock()" which appear to signal warp context switch

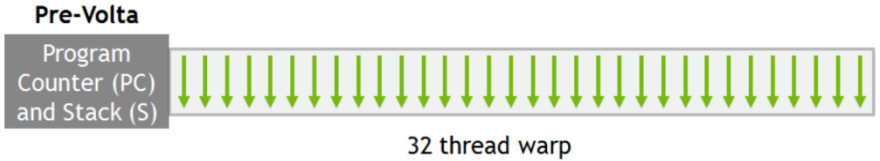
- Exploit intra-TB and inter-TB data reuse
  - Invoke clock() after memory fetch when there is more inter-TB locality

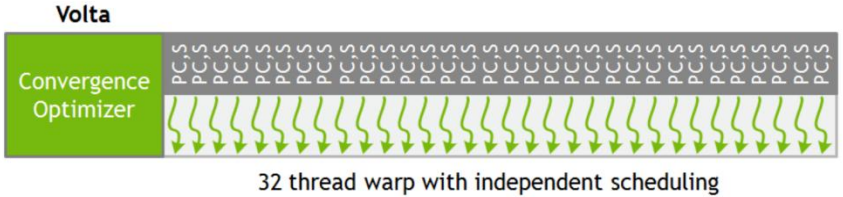
Weifeng Liu et al. **Fast synchronization-free algorithms for parallel sparse triangular solves with multiple right-hand sides**. *Concurrency and Computation: Practice and Experience*, 2017.

Extension 2: Volta opportunities



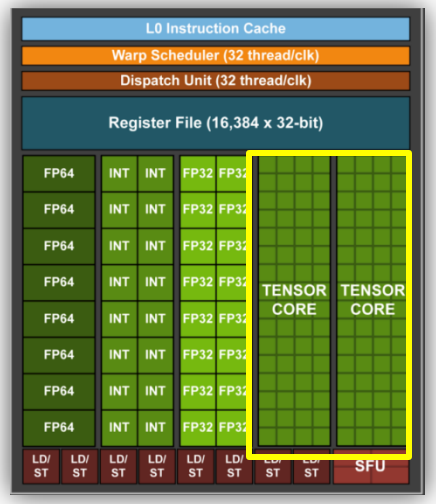






- Threads in a warp can proceed in sub-warp granularity
  - Benefit: more flexible warp sync, communication and cooperation
  - Issue: threads in a warp has to synchronize

Extension 2: Volta opportunities




```
// Calculate AB with NVIDIA Tensor Cores
// Kernel executed by 1 Warp (32 Threads)
__global__ void tensorOp(float *D, half *A, half *B) {
        // 1. Declare the fragments
        wmma::fragment<wmma::matrix_a, M, N, K, half, wmma::col_major> Amat;
        wmma::fragment<wmma::matrix_b, M, N, K, half, wmma::col_major> Bmat;
        wmma::fragment<wmma::accumulator, M, N, K, float, void> Cmat;
        // 2. Initialize the output to zero
        wmma::fill_fragment(Cmat, 0.0f);
        // 3. Load the inputs into the fragments
        wmma::load_matrix_sync(Amat, A, M);
        wmma::load_matrix_sync(Bmat, B, K);
        // 4. Perform the matrix multiplication
        wmma::mma_sync(Cmat, Amat, Bmat, Cmat);
        // 5. Store the result from fragment to global
        wmma::store_matrix_sync(D, Cmat, M, wmma::mem_col_major);
}
```

How can the WC model behaves to be more efficiently operates the Tensor Core!

Summary

Pacific Northwest NATIONAL LABORATORY

- **Warp-Consolidation**: a GPU Programming and Execution model that
  - Unifies warp and thread block (no explicit & implicit sync)
  - Communicates via register while cooperates via warp voting
- **Applicability**:
  - Simplified programming model than CUDA
  - SCC (sync, communication, cooperation) applications
- **1.7x, 2.3x, 1.5x and 1.2x** average speedups across 32 GPGPU applications on Kepler, Maxwell, Pascal and Volta GPUs, respectively

WC: A New GPGPU Programming Model

Ang Li, Pacific Northwest National Laboratory (PNNL)
Weifeng Liu (Norwegian University of Science and Technology)
Linnan Wang (Brown University)
Kevin Barker and Shuaiwen Leon Song (PNNL)

Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle *Since 1965*