GPU TECHNOLOGY CONFERENCE

NVIDIA

RESOLVING FALSE DEPENDENCE ON SHARED MEMORY

Patric Zhao

patricz@nvidia.com

Agenda

- Background
- Programming Skeleton by Shared Memory
- False Dependence Issue
- Solutions to a real case
- Conclusions

1. Background

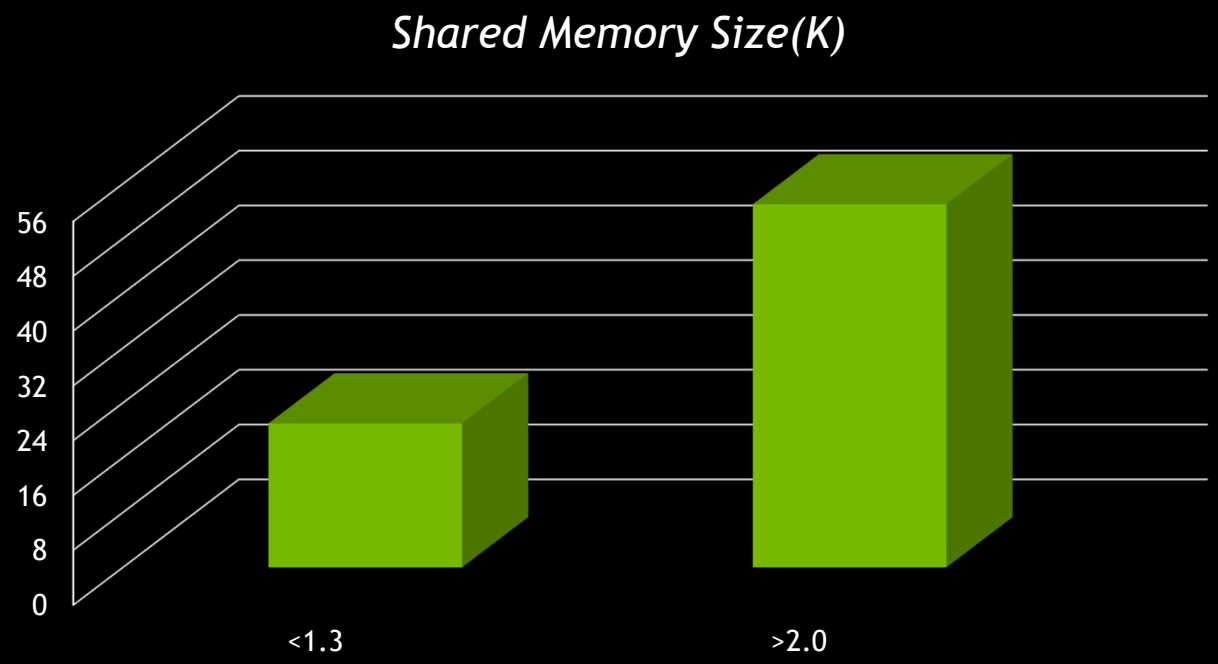
Shared Memory Capabilities

- Much faster than global memory
- Lower latency

  roughly 100x lower than un-cached global memory latency

Shared Memory Size

- Increase to 48K from 16K



Usages of Shared Memory

- user-managed data caches
- high-performance cooperative parallel algorithms
- facilitate global memory coalescing

2.Programming Skeleton by Shared Memory

CUDA source code example

```
Loop:
    load data ← global memory
    computations
    write data → global memory
End loop
```

2.Programming Skeleton by Shared Memory

CUDA source code example

What's problem ?

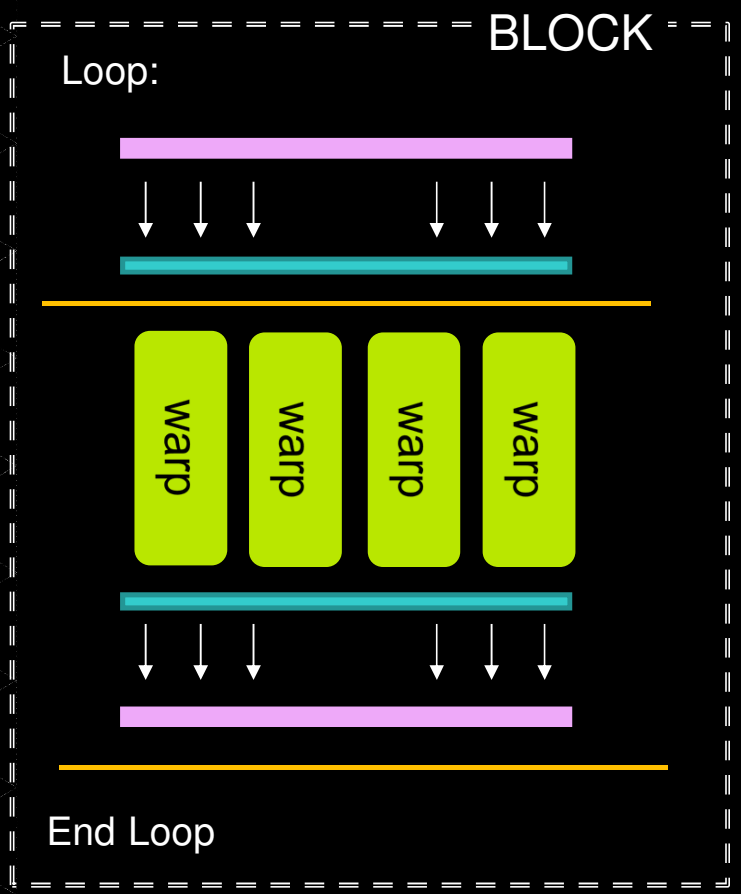
Low efficiency on GMEM if same location is hit many times in a BLOCK.

```
Loop:
    load data ←  global memory
    computations
    write data →  global memory
End loop
```

CUDA source code example by shared memory

```
Loop :

    //BLOCK level load  and computation
    SMEM ← GMEM
    __syncthreads( )

    Computation

    Results → GMEM
    __syncthreads()

End Loop
```

Execution Mode



read data from GMEM to SMEM

guarantee all threads done

4 warps execute in parallel

write results to GMEM

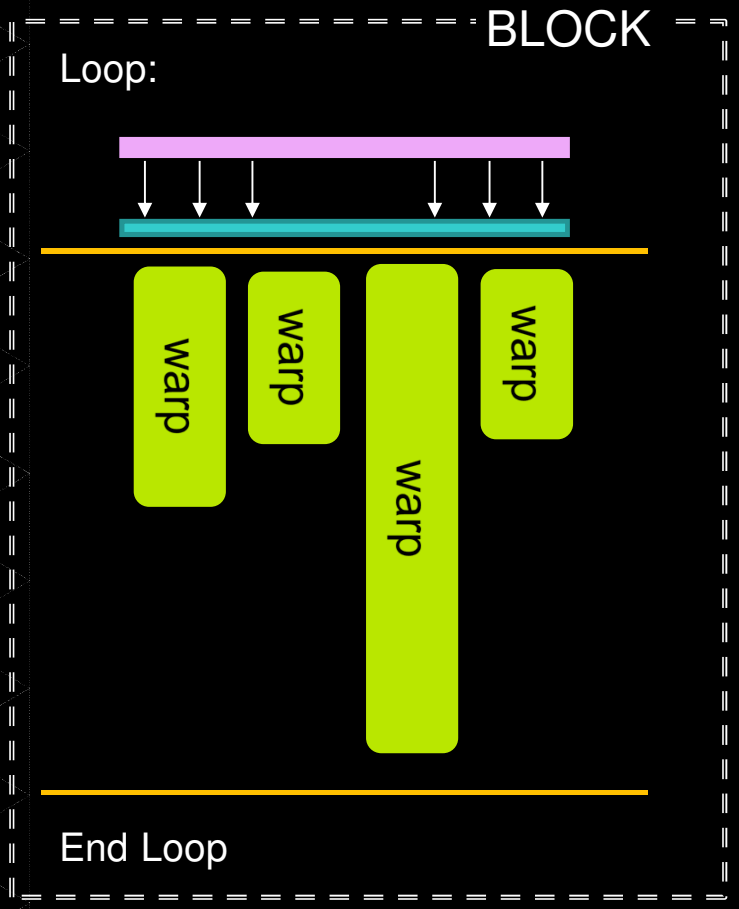
guarantee all threads done

3.False Dependence Issue

What's false dependence ?

There is no data dependence between each WARP in a BLOCK but they have to wait for each other to complete to update only because they are sharing shared memory.

False Dependence on Shared Memory



In case

- Each warp can perform independently
- Each warp needs to get data from SMEM
- A warp is longer than others

What happens ?

Fast warps are blocked by barrier !
This is what we call false dependence.

Why do longer warps occur in this case?

Two major reasons:

- Unbalanced Workload
- Divergence code

4. Solutions to a real case

What is NAMD?

It is a popular molecular dynamic program.

- Comprehensive, industrial-quality software
- Simulation of large bio-molecular systems
- Large user base – 51,000 users

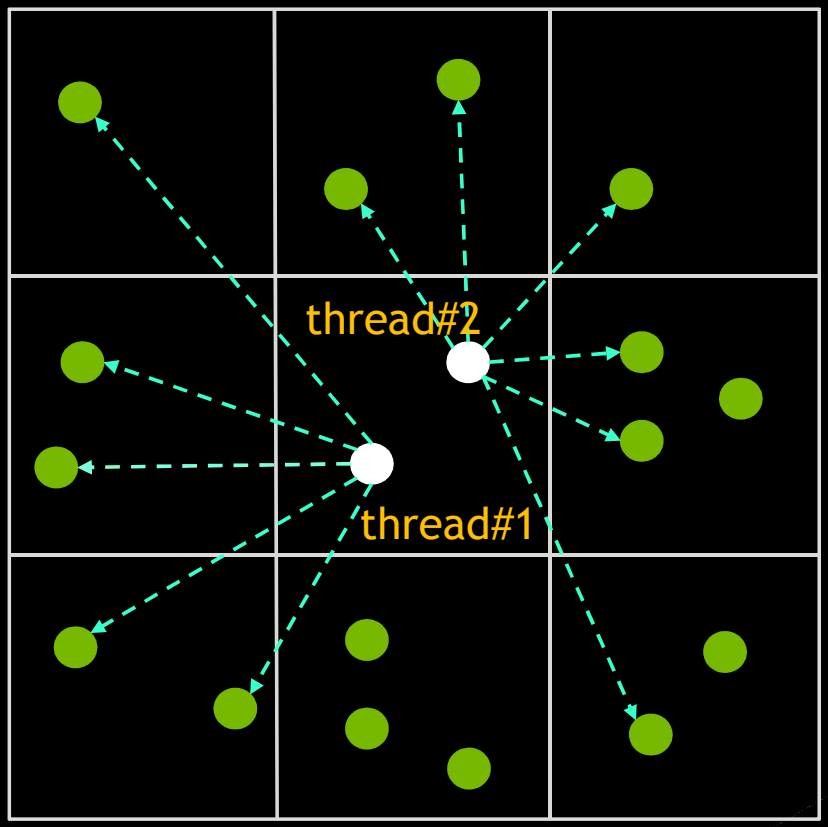
Short-range interaction

- **Computation**

BLOCK → all atoms on a patch

Thread → compute interactive force
in one atom in each time

Each BLOCK needs to load
surrounding cells (patches)

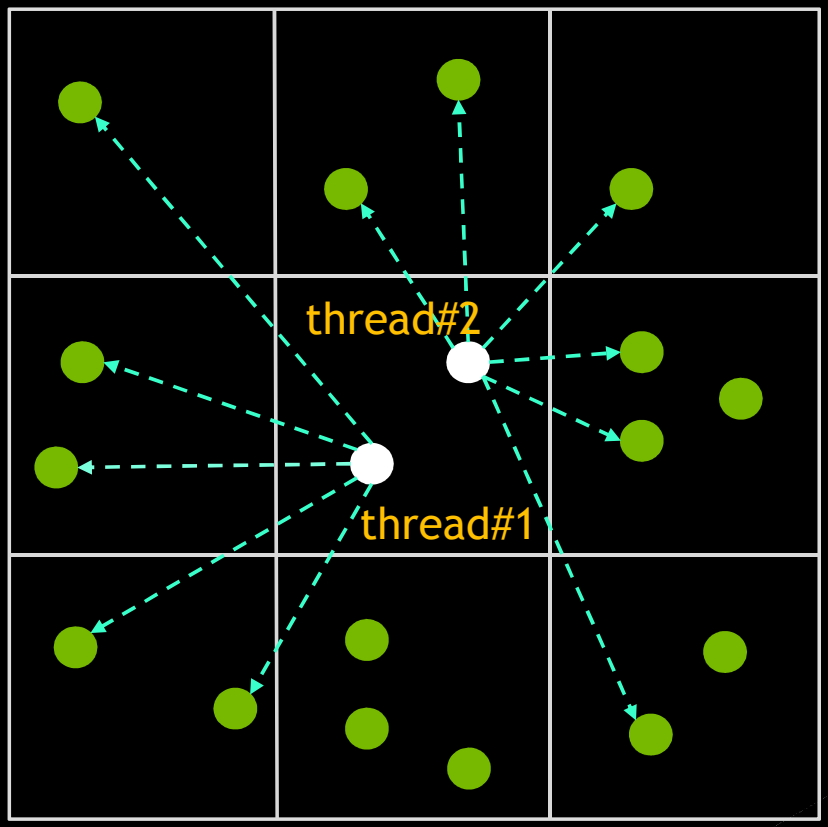


- **Memory usage**

register ← keep an atom info

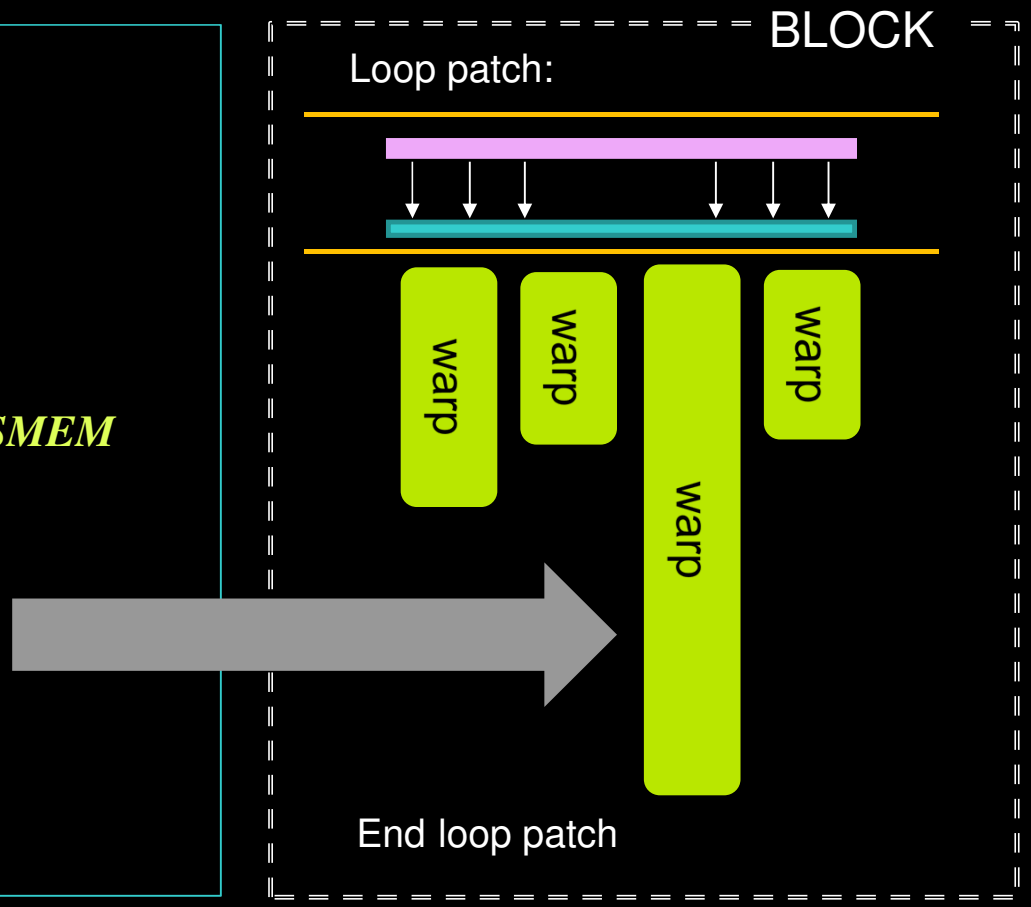
SMEM ← as atoms cache to store neighbor atoms

GMEM ← all atoms data



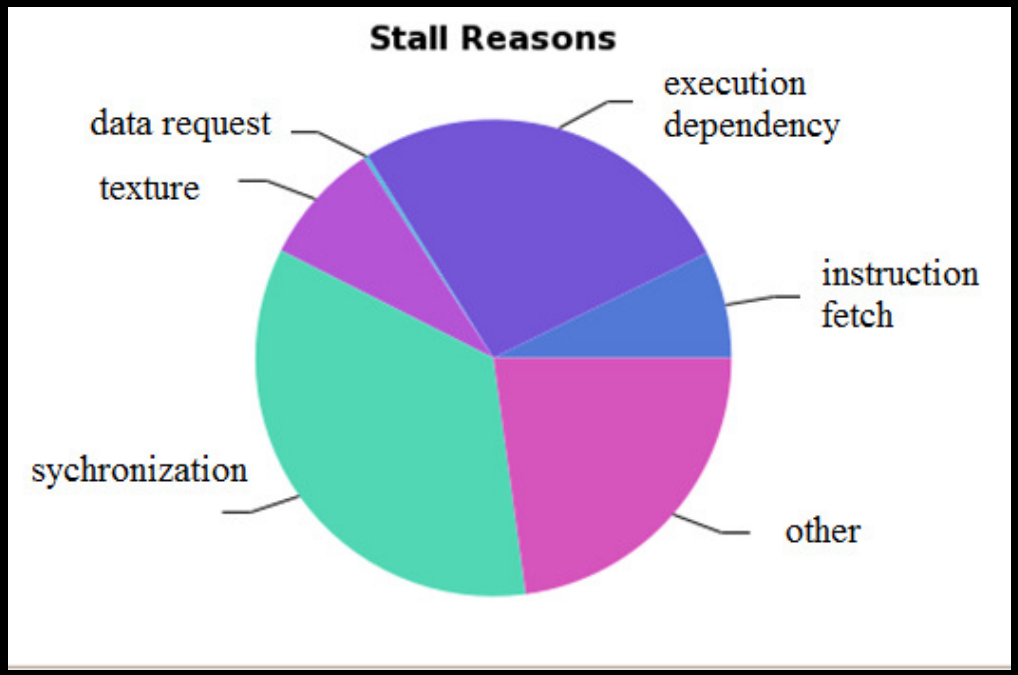
Algorithm Structure

```
RF ← GMEM per thread
Loop patch:
        //load neighbor atoms
        __syncthreads()
        SMEM ← GMEM
        __syncthreads()

        iterate each atom in SMEM
        computations
        if ( r < cutoff )
        {
          divergence code
        }

End loop patch
```



Performance Monitor

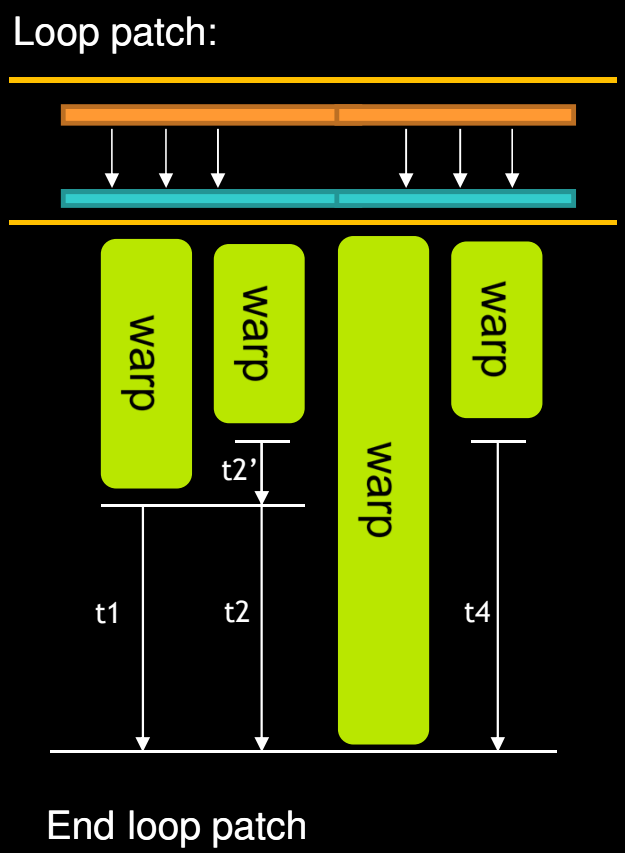
- Identify the performance issue
  - visual profiler (NVVP)
  - command profiler (nvprof)



Solution : Small Block

Small block size can reduce the synchronization time for every warp in a block.

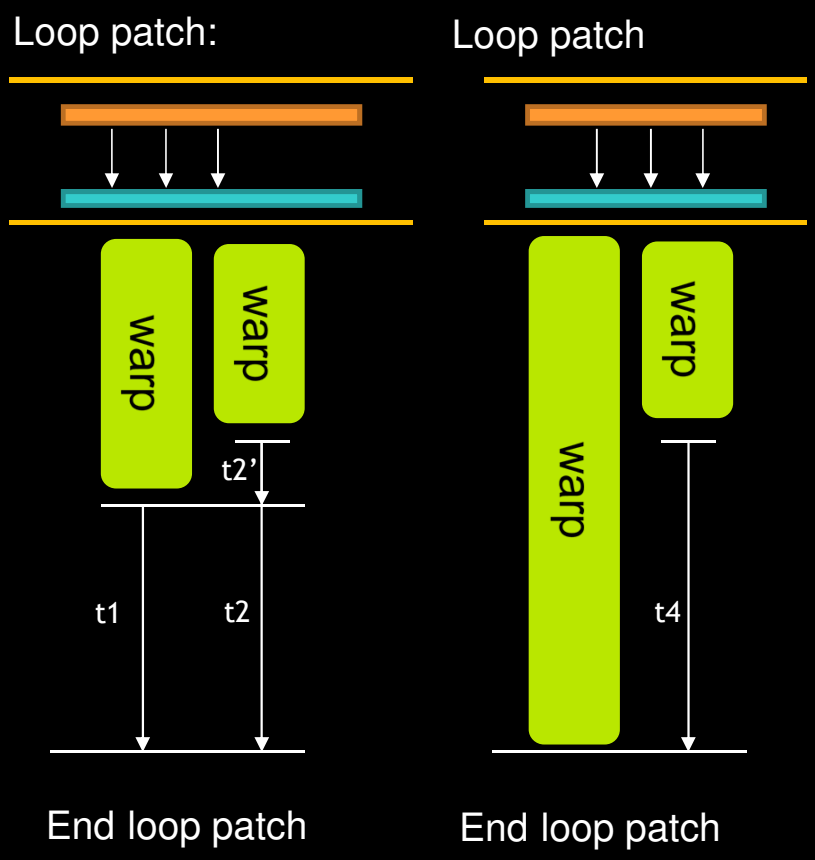
We suggest developers use small block size for these applications with unbalanced workload / divergence code.



Total Waiting Time:

4 warps block:

t1 + t2 + t4



Total Waiting Time:

4 warps block:

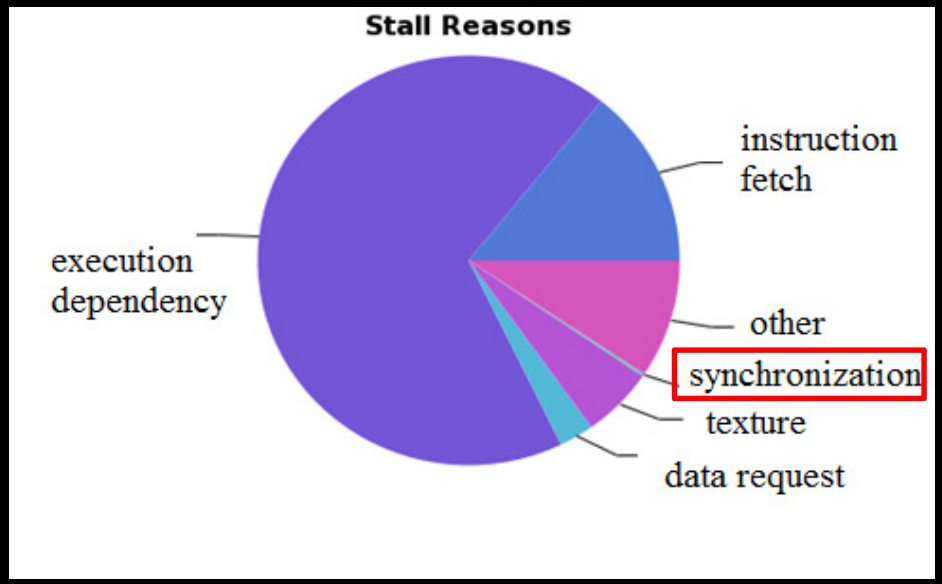
t1 + t2 + t4

2 warps block:

t2 - t1 + t4

we save 2*t1 time.

Use 32 threads (1 warp) per block



Kernel (ms)	Original	Small Block	Speedup
dev_nonbonded	20.65	17.49	~20%

Solution : Prefetch data

- move data to register/cache/SMEM earlier
- avoid delays for mathematical operations

```
Loop :
    // load patch2
     __sync();
     SMEM ← GMEM
     __sync();
     computation;
     {
         divergence code
      }
End loop
```

```
Loop :
    // load patch2
    {
         RF ← GMEM
         __sync();
         SMEM ← RF
         __sync();
     }
     computation;
     {
         divergence code
      }
End loop
```

Results

kernel	Original (ms)	Prefetch (ms)	Speedup
dev_nonbonded	20.65	19.39	~10%

Tips :

- General approach to improve performance
- Large benefits in some cases depending on data access pattern

Final results for NAMD case

method	stall_sync (%)	Improvement (%)
original	32	
prefetch	32	10 %
small block	~0	20 %
Total	**~0**	**32 %**

5.Conclusions

- Shared Memory can highly improve performance

  But false dependence can reduce performance, due to
  - workload unbalance
  - divergence code

- Identify the issue by visual profile or nvprof

- Resolve issue by small block size or data prefetch

Thanks for your attention