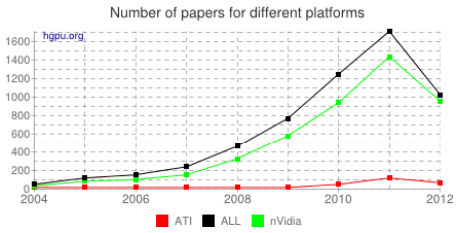
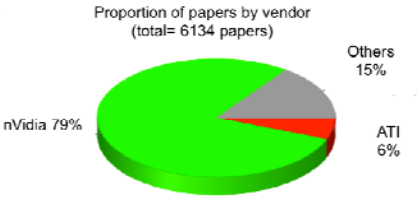
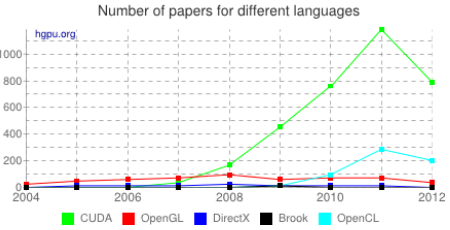
Performance Bugs in Heterogeneous Programming

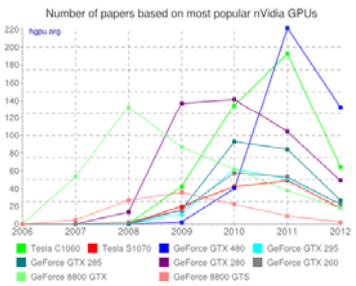
Fahad A. Arshad

Publication Statistics from hgpu.org





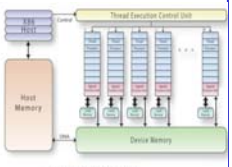




Recap

- Heterogeneous Architecture
  - GPU (integrated and discrete)
- Programming
  - Language (CUDA, OpenCL, OpenACC)
- Performance Bugs
  - General
    - synchronization, skippable function, wrong data-structures, …
  - GPU-specific
    - memory data-access patterns, architecture, code-portability

Integrated GPU


Discreted GPU

Model for Heterogeneous Systems

- "normal system" + coprocessor
  - Intel x86 host + Nvidia GPU
  - AMD Opteron + AMD GPU
  - Intel core + Intel MIC
- Similarities
  - Asynchronous execution
  - Internal parallelism

Programming Heterogeneous Systems is HARD

- Performance
  - Parallel activities
  - Synchronization
  - Data Locality
  - CPU-GPU communication
- Programming Languages
  - CUDA (Nvidia only)
  - OpenCL
  - OpenACC (Directive-based)

PURDUE UNIVERSITY

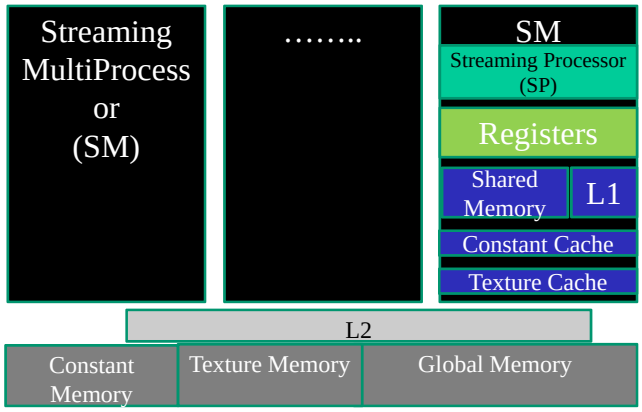
Fixing Performance Bugs: An Empirical Study of Open-Source GPGPU Programs

Yi Yang, Ping Xiang, Huiyang Zhou (NCSU), Mike Mantor (AMD)
International Conference on Parallel Processing (ICPP 2012)

- Background
- Study open source projects
  - Categorized performance bugs
  - Proposed solutions
  - Performance and energy evaluation
- Conclusions

PURDUE UNIVERSITY

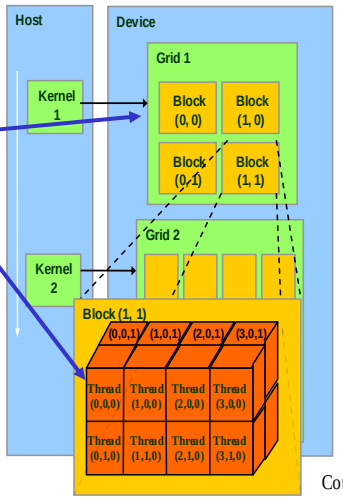
GPU architecture



- How to access the global memory efficiently?
- The constant and texture memories are overlooked

GPU Thread Hierarchy: Block IDs and Thread IDs

- Each thread uses IDs to decide what data to work on
  - Block ID: 1D or 2D
  - Thread ID: 1D, 2D, or 3D
- Simplifies memory addressing when processing multidimensional data
  - Image processing
  - Solving PDEs on volumes
  - …



Courtesy: NDVIA

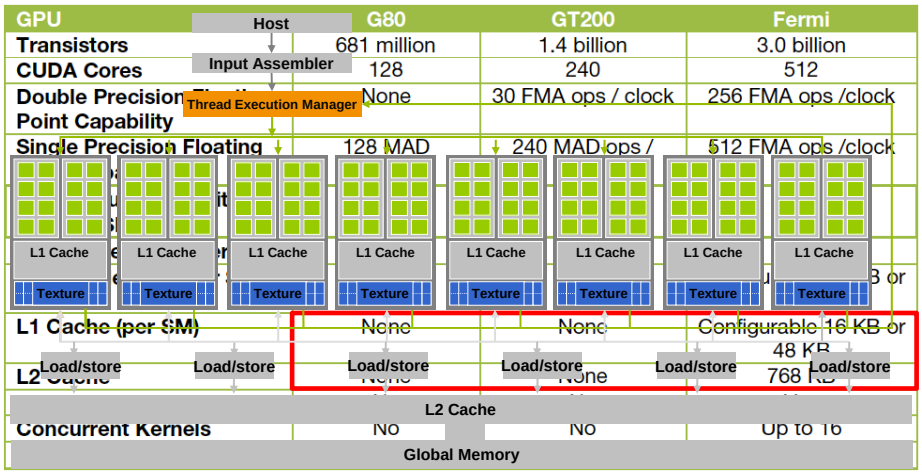
CUDA and OpenCL programming language

- How well programmers utilize the GPUs hardware?
- Application developers need to specify the thread block dimension, and most applications choose 16x16 or 256x1

- OpenCL is supported by both AMD and NVIDIA GPUs
  - How to find the optimal thread block dimension?
  - How to achieve the high performance on different GPUs using same code?

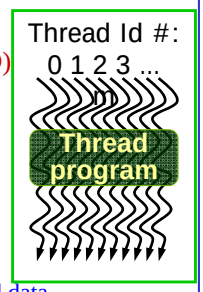
GPU Questions from Last Session

- Is there even a L1/L2 Cache?
  - Yes/No



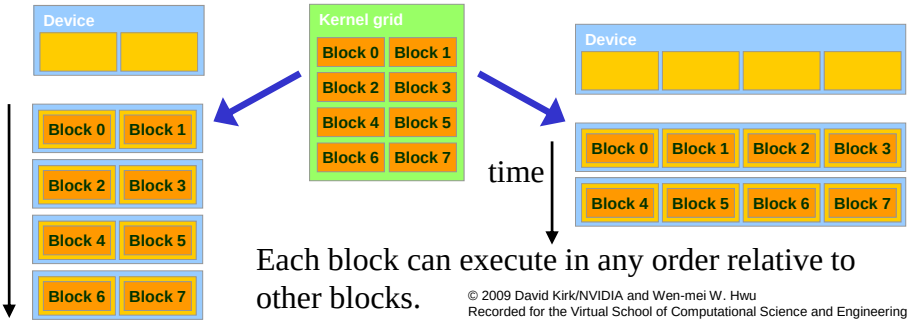
GPU Questions from Last Session

- Is shared memory per block or per SM?
  - A programmer sees thread blocks and not SM
- CUDA Thread Block
- All threads in a block execute the same kernel program (SPMD)
- Programmer declares block:
  - Block size 1 to **512** concurrent threads on G80, G200
  - Up to 1024 on **GF100**
  - Block shape 1D, 2D, or 3D
  - Block dimensions in threads
- Threads have thread id numbers within block
  - Thread program uses thread id to select work and address shared data
- Threads in the same block can synchronize while doing their share of the work
- Threads in different blocks cannot cooperate
  - Each block can execute in any order relative to other blocks!



Transparent Scalability

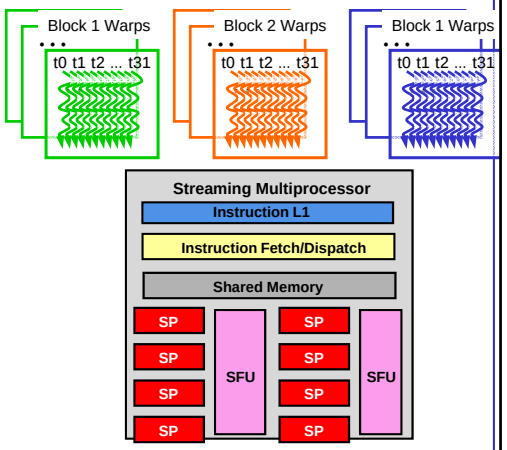
- Hardware is free to assigns blocks to any processor at any time
  - A kernel scales across any number of parallel processors



Each block can execute in any order relative to other blocks.

G80 Example: Thread Scheduling and Warp Concept

- Each Block is executed as 32-thread Warps
  - Warps are scheduling units in SM
- If 3 blocks are assigned to an SM and each block has 256 threads, how many Warps are there in an SM?
  - Each Block is divided into 256/32 = 8 Warps
  - There are 8 * 3 = 24 Warps



Studied GPGPU Projects

Project	Description
DecGPU	An error correction algorithm implemented in NVIDIA CUDA and MPI, and runs on a GPU cluster.
FLAGON	A library for programming NVIDIA CUDA from Fortran 9x. It provides multiple primitive functions and an interface to CUBLAS and CUFFT library.
GPUMLib	A GPGPU code library for machine learning algorithms.
Ising GPU	A project uses GPUs to accelerate Monte Carlo simulation of the 2D and 3D Ising models. Up to 35X speedups are achieved over the CPU implementation.
MUMmerGPU	A high-throughput parallel pair-wise local sequence alignment program; 13X faster than the CPU version.
nDust	A set of GPGPU programs to calculate dust-plasma charge equilibrium of dust-plasma systems in protoplanetary disc environments.
OpenCurrent	A C++ library for solving Partial Differential Equations (PDEs) over regular grids.
Qymsym	A GPU accelerated parallel hybrid symplectic integrator for planetary system integration.
ViennaCL	An OpenCL code library of common linear algebra operations and the solution of large sparse systems of equations by means of iterative methods.
CUBLAS & CUDA SDK	Although CUBLAS 3.1 and 3.2 are not open source, their matrix multiplication implementations are available. The matrix multiplication in CUDA SDK is open source.

Include computational physics, biology, mathematics, machine learning, and etc.

Performance Bug Patterns

- Classify performance bugs

1) Thread block dimension
2) Constant and texture memory
3) Off-chip memory bandwidth

- Code-segments leading to inefficient use of GPU hardware
- Propose solutions for these performance bugs

PURDUE UNIVERSITY

Methodology

- Intel Core 2 Quad Q9650 CPU
  - NVIDIA GTX285
  - GTX480
  - AMD HD5870

- CUDA SDK 3.1 and ATI Stream SDK v2.2

- Evaluation
  - Performance
  - Energy efficiency
    - Gflops/Joule (computational workload)
    - Gbytes/Joule (transmission benchmarks)

Energy = Power * Time
Power = Dynamic_power + Static_power

PURDUE UNIVERSITY

1) Thread block dimension

Buggy Code

```
int main() {
   dim3blkDim(16, 16); // Kernel invocation
   dim3gridDim(N / blkDim.x, N / blkDim.y);
   myKernel<<<gridDim, blkDim>>>(…);
}
```

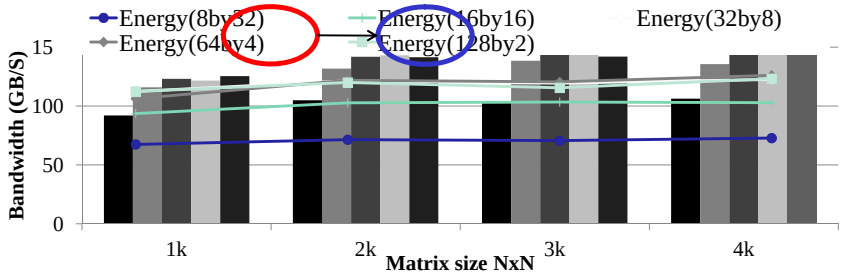
Kernel invocation

- Many applications choose 16x16 or 256x1
- The search space of the optimal thread block dimension is large
- Examine the thread block dimension in three cases
  - No data reuse
  - Data reuse through shared memory
  - Data reuse through the hardware cache

Case 1: no data reuse (GTX 480)

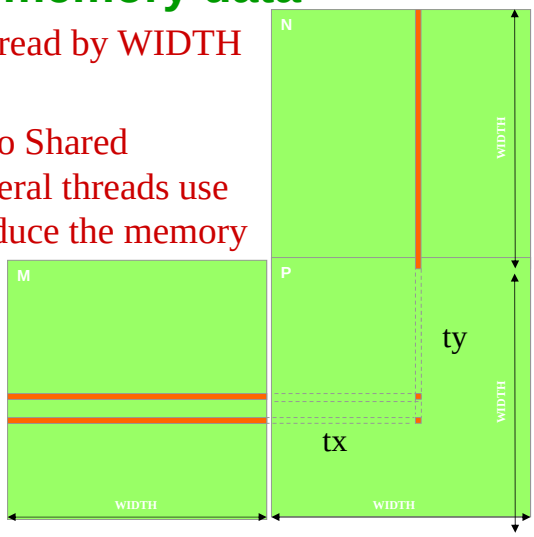
```
__global__ void kernel(float* out, float* in, int
w){
   int idx = threadIdx.x+blockIdx.x*blockDim.x;
   int idy = threadIdx.y+blockIdx.y*blockDim.y;
   out[idy*w+idx]  = in[idy*w+idx];
}
```

Memory copy in 2D domain



Idea: Use Shared Memory to reuse global memory data

- Each input element is read by WIDTH threads.
- Load each element into Shared Memory and have several threads use the local version to reduce the memory bandwidth
  - Tiled algorithms

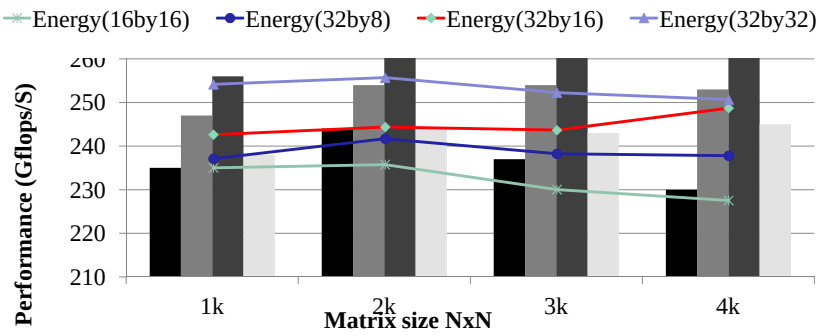


Case 2: data reuse through shared memory

```
__global__ void matrixMul( output* C, input* A, input* B,
intwA, in wB){
    int tx = threadIdx.x; int ty = threadIdx.y;
    ….//variable declaration and definition
    for (a = aBegin, b = bBegin;a <= aEnd;a+=aStep, b+=bStep){
          __shared__ float As[blocky][Step], Bs[Step][blockx];
          …//load a tile of A into shared mem As
          …//load a tile of B into shared mem Bs
          for(i= 0; i<Step; i++)
         Csub += As[ty][i]* Bs[i][tx];
    }
    …//store Csub to matrix C
}
```

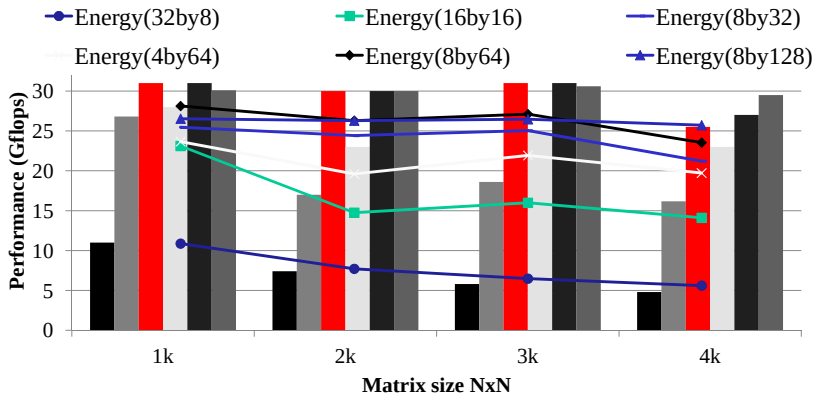
Matrix multiplication from NVIDIA SDK

Matrix multiplication (SDK) on GTX 480



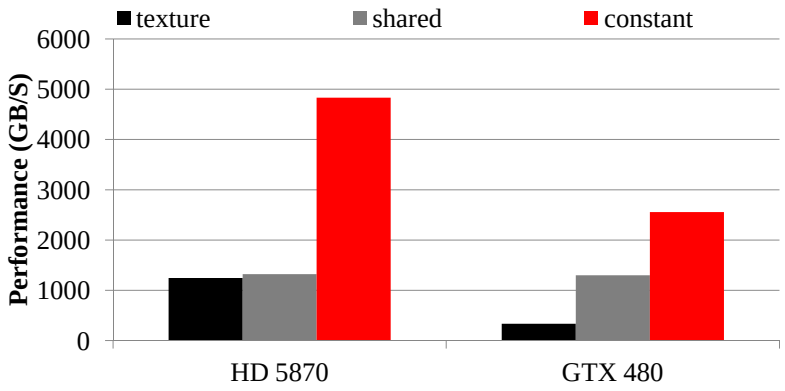
- 32by16 achieves the best performance
  - More data reuse in shared memory with larger thread block
  - 3 thread blocks per SM (one SM can have up 1.5K threads)
- 32by32 has best energy consumption
  - Best data reuse with 1 thread block per SM

Case 3: data reuse through hardware cache



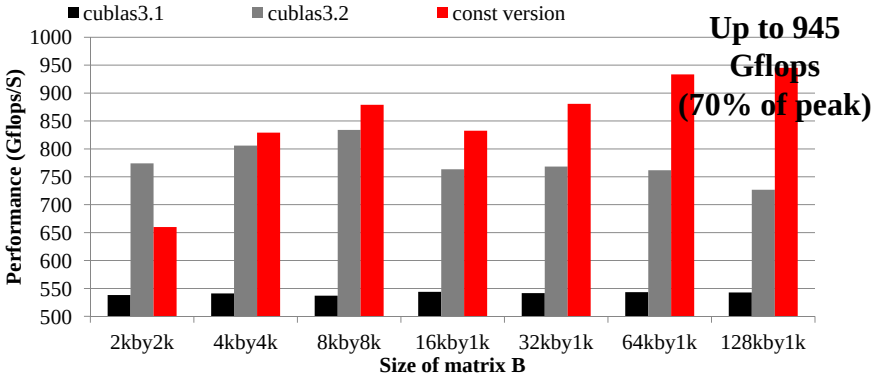
- 8x32 is up to 77% speedup over 16x16
- Reduce the inter-warp reuse and increase the intra-warp reuse
- The detail can be found in the paper

2) Constant and texture memory (cache)



The bandwidth of different caches (assume all data are in the cache)

Performance result comparison on GTX 480



- cublas3.1: A is tiled in shared memory and B is tiled in register
- cublas3.2: A,B are first tiled in shared memory and further tiled in register
- Speedup: Up to 74% speedup on cublas3.1 and 30% speedup on cublas3.2

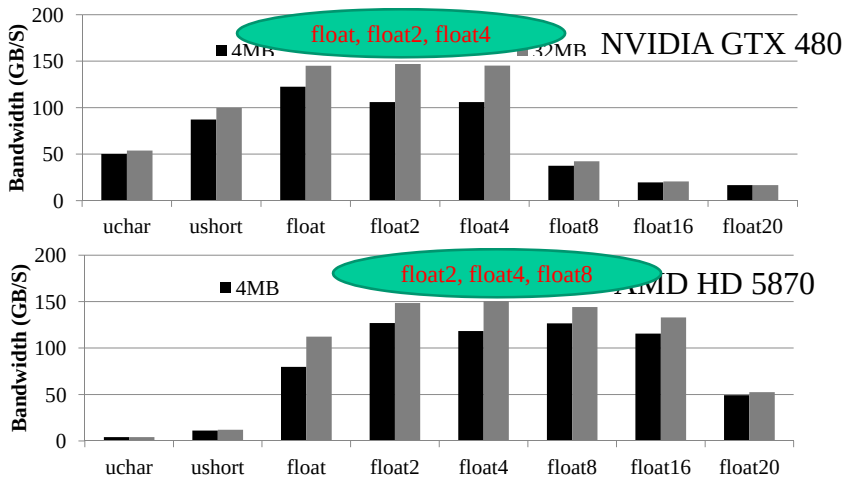
3) Global memory datatypes

Buggy Code

```
template <class DT>
__global__ void kernel(DT* out, DT* in){
   int idx = threadIdx.x+blockIdx.x*blockDim.x;
   out[idx]  = in[idx];
}
```

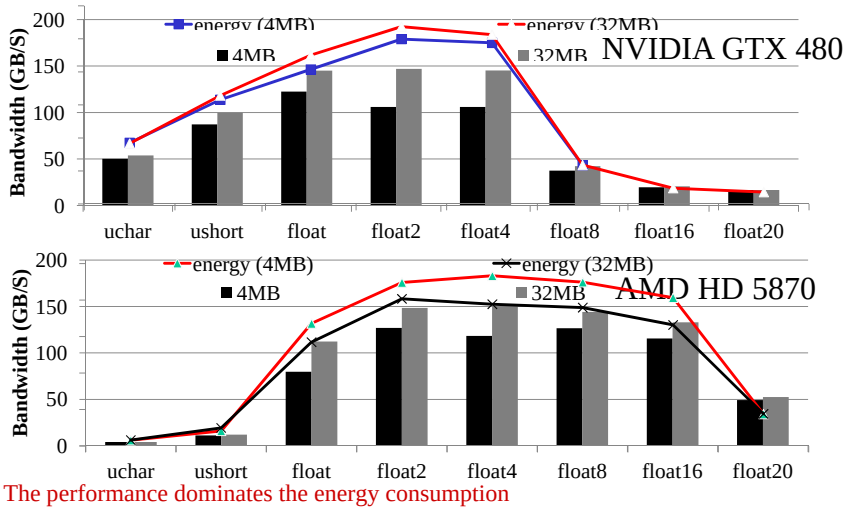
Accessing global memory using different data types

Global memory data types



- Observation: Only some data types can deliver optimal bandwidth

Global memory data types: energy consumption



- The performance dominates the energy consumption
- float2 has optimal energy consumption on both GTX 480 and HD5870

Impact of bugs

Bug type	Affected projects	Fixed kernels	Speedup GTX285	Speedup GTX480	SpeedupHD5870
Global Mem.	7	1	11.14X	2.33X	31.30X
Thread block Dim.	10	4	N/A	1.07X-1.77X	N/A
Portability	1	1	1.82X-2.38X	1.61X-5.00X	3.80X-6.89X
Constant and texture	2	2	2.42X	1.1X-4.03X	9.30X
Function special.	3	3	N/A	1.93X-4.72X	N/A
Floating-point Num.	2	2	N/A	1.14X-1.50X	N/A

The proposed fixes achieve significant improvements

Conclusions

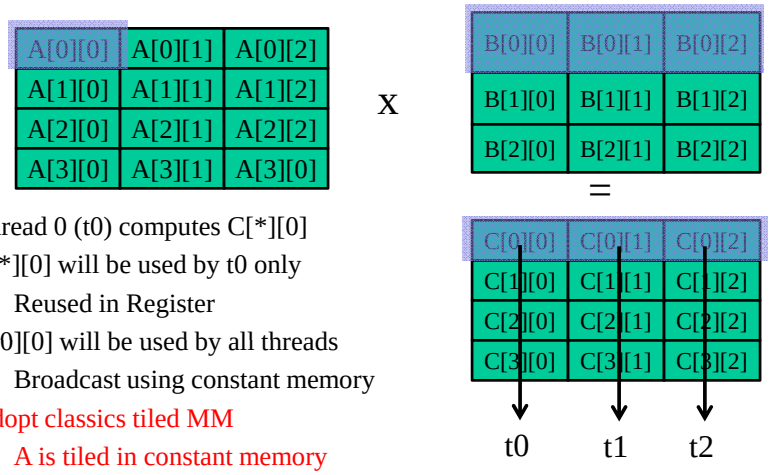
- Investigated ten open source projects and characterized the common performance bugs issues.
- Proposed a set of new optimization techniques to fix the performance bugs of these open source projects.
- Studied the energy effect of performance bugs and show that proposed fixes achieve both high performance and energy efficiency.

PURDUE UNIVERSITY

Backup Slides

PURDUE UNIVERSITY

Matrix multiplication using constant memory



- Thread 0 (t0) computes C[*][0]
- B[*][0] will be used by t0 only
  - Reused in Register
- A[0][0] will be used by all threads
  - Broadcast using constant memory
- Adopt classics tiled MM
  - A is tiled in constant memory
  - B is tiled in register

Tiled Multiply

- Break up the execution of the kernel into phases so that the data accesses in each phase is focused on one subset (tile) of Md and Nd