グリッドコンピューティング

2012/1/23

小嶋秀徳 (11M37123 松岡研究室)

紹介論文

GROPHECY: GPU Performance Projection from CPU Code Skeletons

SC 2011
(Super Computing)

Jiayuan Meng, Vitali A. Morozov, Kalyan Kumaran, Venkatram Vishwanath, Thomas D. Uram

Argonne National Laboratory

Abstract

- GROPHECY
  - GPU performance projection framework
  - Without actual GPU programming or hardware
  - Code skeletonization
  - Automatic transformation from code skeletons to mimic tuned GPU codes

Agenda

Introduction

GROPHECY

* CPU code skelton → GPU Performance Projection
  - Developers use this projection to determine whether they should port the CPU code to the GPU code or not.

* Without GPU code and without accessible GPUs
  - Only hardware specifications and application statistics are needed.

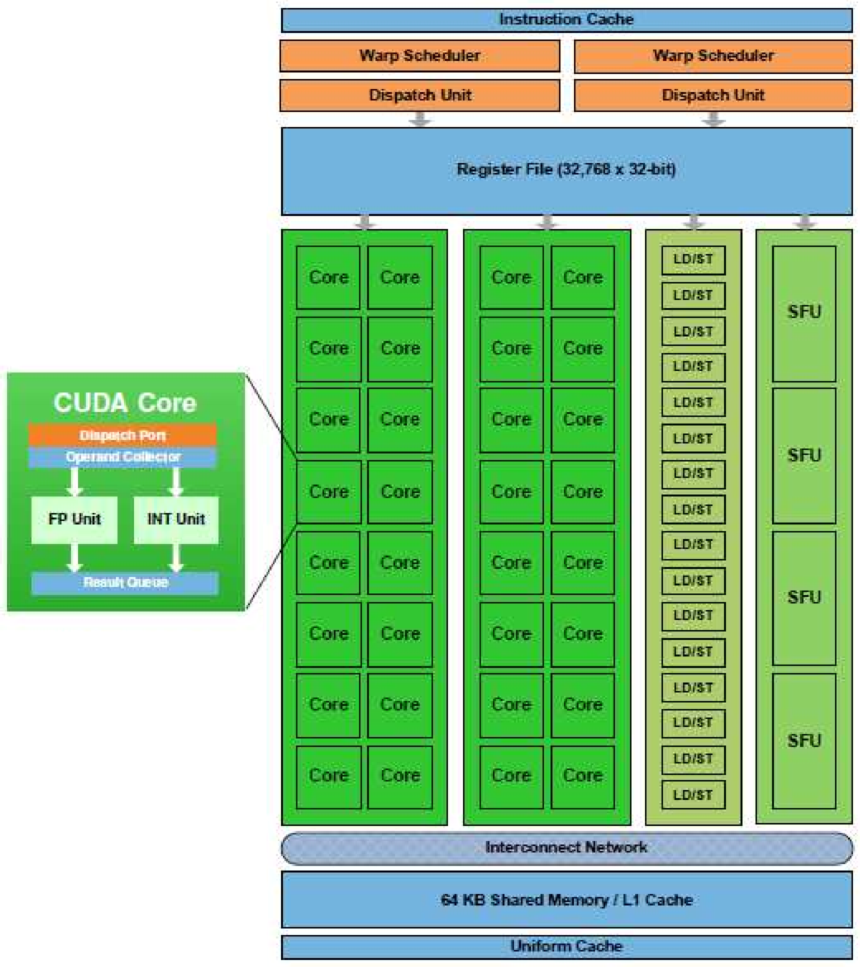
Agenda

Related Work

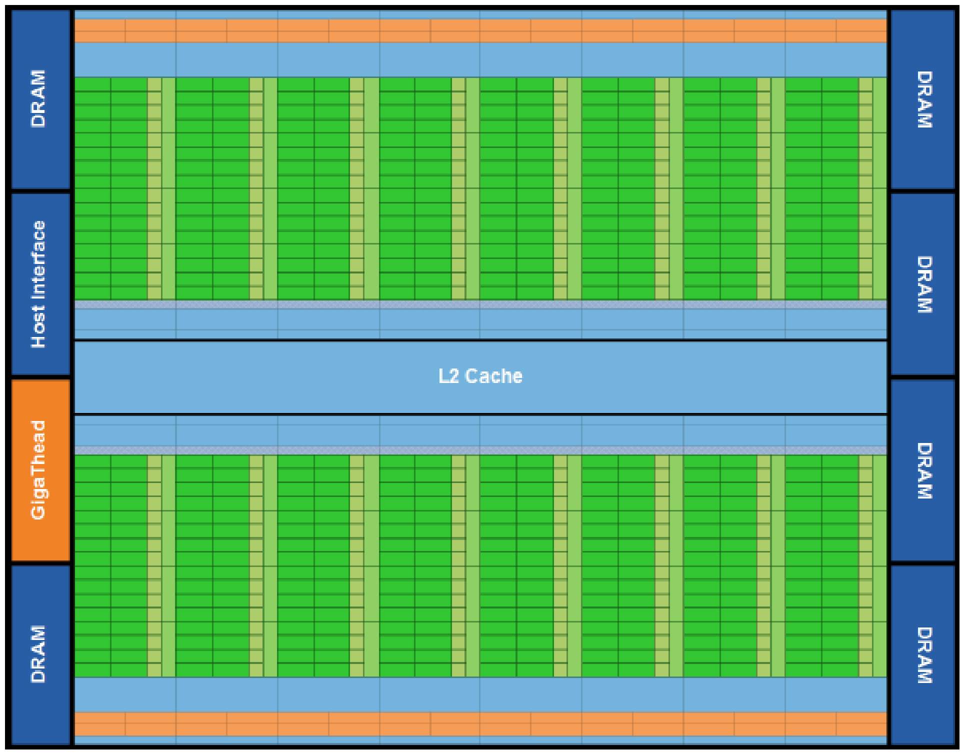
- Translation tools based on an annotated legacy code
- Metaprogramming tools
- Model-driven auto tuning framework
- GPU performance model
- Performance models for application tuning over complex or large scale systems
- Cross platform performance prediction

Agenda

GPU Architecture

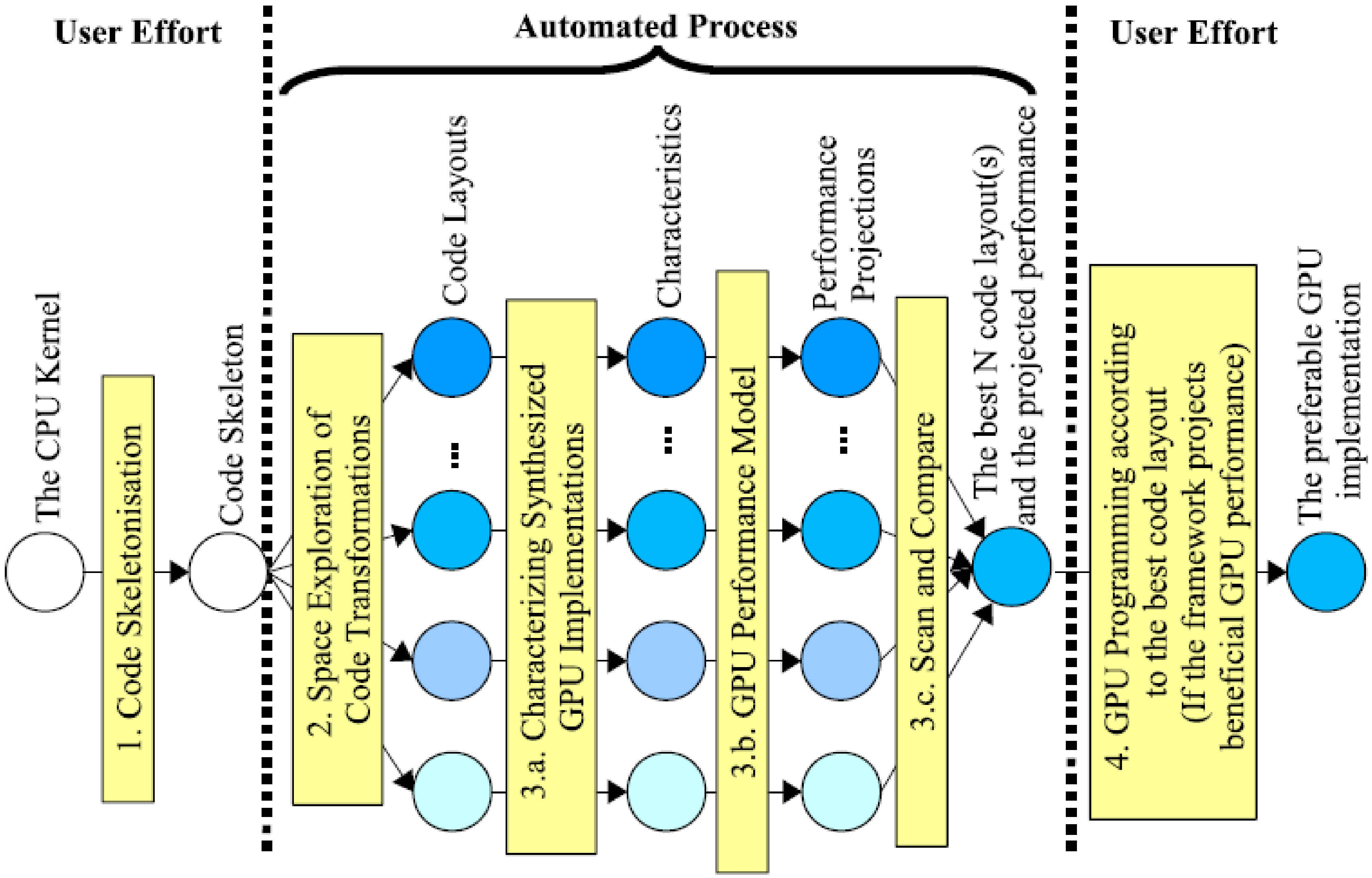


Fermi Streaming Multiprocessor (SM)



Agenda

An Overview of GROPHECY


User Effort
Automated Process
User Effort
The CPU Kernel
1. Code Skeletonisation
Code Skeleton
2. Space Exploration of Code Transformations
Code Layouts
3.a. Characterizing Synthesized GPU Implementations
Characteristics
3.b. GPU Performance Model
Performance Projections
3.c. Scan and Compare
The best N code layout(s) and the projected performance
4. GPU Programming according to the best code layout (If the framework projects beneficial GPU performance)
The preferable GPU implementation

Agenda

Code Skeletonization

- Code skeleton
  - Abstract of the CPU code structure
  - Starting point for code transformation

Listing 1: MatMul's CPU code

```
float A[N][K], B[K][M];
float C[N][M];
int i, j, k;
for(i=0; i<N; ++i){
 for(j=0; j<M; ++j){
   float sum = 0;
   for(k=0; k<K; ++k){
     sum+=A[i][k]*B[k][j];
   }
 C[i][j] = sum;
}
```

Code Skeletonization

Listing 2: MatMul's code skeleton

```
float A[N][K]
float B[K][M]
float C[N][M]
/* the loop space */
parallel_for(N, M)
: i,j
{
  /* computation w/t
   * instruction count
   */
  comp 1
  /* streaming loop */
  stream k = 0:K {
    /* load */
    ld A[i][k]
    ld B[k][j]
    comp 3
  }
  comp 5
  /* store */
  st C[i][j]
}
```

Code skeleton elements

- Data parallelism
- A task
- Data Accesses
- Computation instructions
- Branch instructions
- For loops
- Streaming loops
- Macros

Code Skeletonization

Listing 2: MatMul's code skeleton

```
float A[N][K]
float B[K][M]
float C[N][M]
/* the loop space */
parallel_for(N, M)
: i,j
{
  /* computation w/t
   * instruction count
   */
  comp 1
  /* streaming loop */
  stream k = 0:K {
    /* load */
    ld A[i][k]
    ld B[k][j]
    comp 3
  }
  comp 5
  /* store */
  st C[i][j]
}
```

Code transformation based on code skeleton information

Listing 3: MatMul's optimized GPU code

```
float A[N][K], B[K][M], C[N][M];
dim3 block(BlkSize, BlkSize);
dim3 grid(N/BlkSize, M/BlkSize);
MatrixMul<<<grid, block>(A, B, C);

__global__ MatrixMul(A, B, C)
{
  __shared__ a[BlkSize][BlkSize];
  __shared__ b[BlkSize][BlkSize];
  int ty = threadIdx.y;
  int tx = threadIdx.x;
  int y = blockIdx.y*blockDim.y+ty;
  int x = blockIdx.x*blockDim.x+tx;
  float sum = 0.f;
  for(int n=0; n<K; n+=BlkSize){
    a[ty][tx]=A[y][n+tx];
    b[ty][tx] = B[n+ty][x];
    __syncthreads();
    for(int k=0; k<BlkSize; ++k){
      sum += a[ty][k]*b[k][tx];
    }
    __syncthreads();
  }
  C[y][x] = sum;
}
```

Agenda

Code Layout Parameterization

$$\{\underbrace{\mathbb{B}}_{①},\underbrace{\mathbb{S},\mathbb{O}}_{②},\underbrace{\mathbb{F}}_{③},\underbrace{\{\dot{\mathbb{A}}\},\{\bar{\mathbb{A}}\},\{ShMem(D_i)\}}_{④},\underbrace{\mathbb{L}}_{⑤}\}$$

① Tread block size $\mathbb{B} = \{b_1, ..., b_n\}$

② Staging $\mathbb{S} = \{s_1, ..., s_n\}$, $\mathbb{O} = \{o_1, ..., o_n\}$

③ Folding $\mathbb{F} = \{f_1, ..., f_n\}$

④ Caching Strategy $\mathbb{A}(D, \Theta, I)$

⑤ Loop Unrolling $\mathbb{L} = \{l_i, ..., l_n\}$

The Search Space

Dependency

$$\mathbb{B} \rightarrow \mathbb{F} \rightarrow [\mathbb{S}, \mathbb{O}] \rightarrow [\{\dot{\mathbb{A}}\}, \{\bar{\mathbb{A}}\}, \{ShMem(D_i)\}, \mathbb{L}]$$

The pseudocode for space exploration of code layouts

```
for all $\mathbb{B}$ such that $size(\mathbb{B}) \leq Max\_block\_size$:
    for all $\mathbb{F}$ such that $size(\mathbb{B}) \times size(\mathbb{F}) \leq size(Loop\_space)$:
        for all $\mathbb{S}$:
         for all $\mathbb{O}$:
            for all $\{\{\dot{\mathbb{A}}\}, \{\bar{\mathbb{A}}\}, \{ShMem(D\_i)\}\}$:
               maximize $\mathbb{L}$ for any applicable loops
               emit $\{\mathbb{B}, \mathbb{S}, \mathbb{O}, \mathbb{F}, \{\dot{\mathbb{A}}\}, \{\bar{\mathbb{A}}\}, \{ShMem(D\_i)\}, \mathbb{L}\}$
```

Identifying Cacheable Data

$$ShrDegree(D) = \frac{size\left(\mathbb{H}(D, \hat{\mathbb{F}})\right) \times size(\mathbb{B})}{size\left(\mathbb{H}(D, \hat{\mathbb{B}})\right)} \quad (1)$$

Footprint

$$\mathbb{H}(D, \mathbb{T}) = \cup\{\mathbb{P}(\mathbb{A}(D), \mathbb{T}) | \forall \mathbb{A}(D)\}$$

Determining Stating Sizes

$$StageShrDegree(D, k) = \frac{size\left(\mathbb{H}(D, \hat{\mathbb{B}} \wedge [k : \langle 0, 1, 1 \rangle])\right) \times size\left([k : \langle K_l, K_u, K_s \rangle]\right)}{size\left(\mathbb{H}(D, \hat{\mathbb{B}} \wedge [k : \langle K_l, K_u, K_s \rangle])\right)} \quad (2)$$

$$StageSize(D, k) = \frac{size(\mathbb{B}) \times StageShrDegree(D, k)}{size\left(\mathbb{H}(D, \hat{\mathbb{B}} \wedge [k : \langle 0, 1, 1 \rangle])\right)} \quad (3)$$

$$NumStages(k) = \begin{pmatrix} \left\lceil \frac{size([k:\langle K_l, K_u, K_s \rangle])}{StageSize(k)} \right\rceil, & if\ k > 0 \\ 1, & if\ k = 0 \end{pmatrix} \quad (4)$$

Estimating Shared Memory Usage

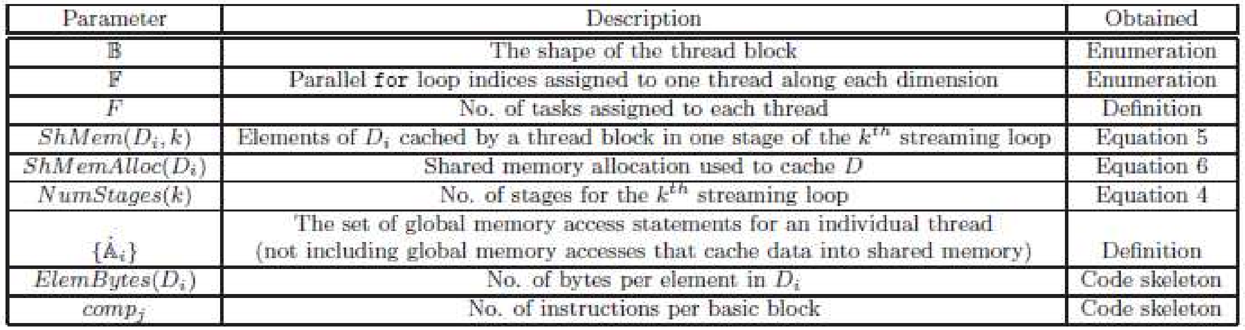
$$ShMem(D, k) = \begin{pmatrix} \mathbb{H}(D, \hat{\mathbb{B}} \wedge [k : \langle 0, StageSize(k), 1 \rangle]), & if\ k > 0 \\ \mathbb{H}(D, \hat{\mathbb{B}}), & if\ k = 0 \end{pmatrix} \quad (5)$$

$$ShMemAlloc(D) = ElemBytes(D) \times \sum_k size\left(ShMem(D, k)\right) \quad (6)$$

Agenda

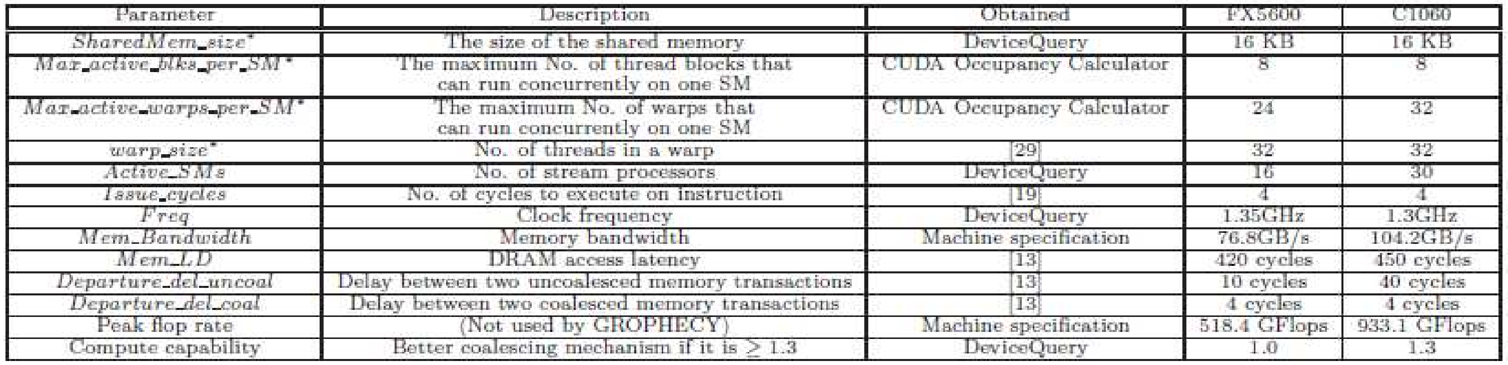
Characterizing Code Layouts

Table 1: Parameters describing a code layout



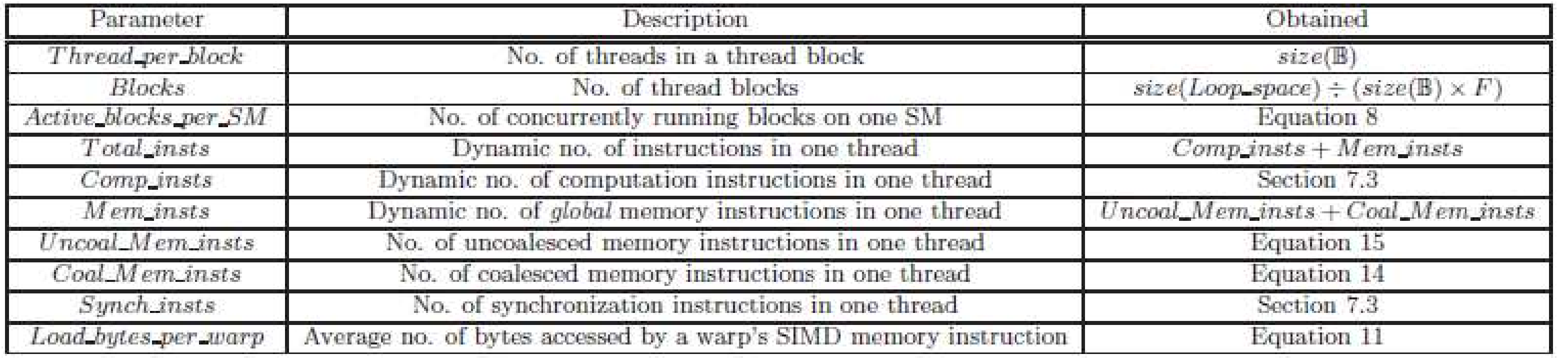
Parameter	Description	Obtained
$\mathbb{B}$	The shape of the thread block	Enumeration
$\mathbb{F}$	Parallel `for` loop indices assigned to one thread along each dimension	Enumeration
F	No. of tasks assigned to each thread	Definition
$ShMem(D_i, k)$	Elements of D_i cached by a thread block in one stage of the k^{th} streaming loop	Equation 5
$ShMemAlloc(D_i)$	Shared memory allocation used to cache D	Equation 6
$NumStages(k)$	No. of stages for the k^{th} streaming loop	Equation 4
$\{\dot{\mathbb{A}}_i\}$	The set of global memory access statements for an individual thread (not including global memory accesses that cache data into shared memory)	Definition
$ElemBytes(D_i)$	No. of bytes per element in D_i	Code skeleton
$comp_j$	No. of instructions per basic block	Code skeleton

Table 5: Hardware parameters. Parameters marked with "*" are used in modeling code layouts and generating workload statistics. All other parameters are used by the underlying GPU performance model.



Parameter	Description	Obtained	FX5600	C1060
$SharedMem\_size^*$	The size of the shared memory	DeviceQuery	16 KB	16 KB
$Max\_active\_blks\_per\_SM^*$	The maximum No. of thread blocks that can run concurrently on one SM	CUDA Occupancy Calculator	8	8
$Max\_active\_warps\_per\_SM^*$	The maximum No. of warps that can run concurrently on one SM	CUDA Occupancy Calculator	24	32
$warp\_size^*$	No. of threads in a warp	[29]	32	32
$Active\_SMs$	No. of stream processors	DeviceQuery	16	30
$Issue\_cycles$	No. of cycles to execute on instruction	[19]	4	4
$Freq$	Clock frequency	DeviceQuery	1.35GHz	1.3GHz
$Mem\_Bandwidth$	Memory bandwidth	Machine specification	76.8GB/s	104.2GB/s
Mem_LD	DRAM access latency	[13]	420 cycles	450 cycles
$Departure\_del\_uncoal$	Delay between two uncoalesced memory transactions	[13]	10 cycles	40 cycles
$Departure\_del\_coal$	Delay between two coalesced memory transactions	[13]	4 cycles	4 cycles
Peak flop rate	(Not used by GROPHECY)	Machine specification	518.4 GFlops	933.1 GFlops
Compute capability	Better coalescing mechanism if it is ≥ 1.3	DeviceQuery	1.0	1.3

Table 2: Workload characteristics serving as inputs to the GPU performance model



Parameter	Description	Obtained
$Thread\_per\_block$	No. of threads in a thread block	$size(\mathbb{B})$
$Blocks$	No. of thread blocks	$size(Loop\_space) \div (size(\mathbb{B}) \times F)$
$Active\_blocks\_per\_SM$	No. of concurrently running blocks on one SM	Equation 8
$Total\_insts$	Dynamic no. of instructions in one thread	$Comp\_insts + Mem\_insts$
$Comp\_insts$	Dynamic no. of computation instructions in one thread	Section 7.3
Mem_insts	Dynamic no. of *global* memory instructions in one thread	$Uncoal\_Mem\_insts + Coal\_Mem\_insts$
$Uncoal\_Mem\_insts$	No. of uncoalesced memory instructions in one thread	Equation 15
$Coal\_Mem\_insts$	No. of coalesced memory instructions in one thread	Equation 14
$Synch\_insts$	No. of synchronization instructions in one thread	Section 7.3
$Load\_bytes\_per\_warp$	Average no. of bytes accessed by a warp's SIMD memory instruction	Equation 11

Active Thread Blocks per SM

$$SharedMem\_bytes\_per\_block = \sum_i ShMemAlloc(D_i) \tag{7}$$

$$Blks\_per\_ShrM = \left\lfloor \frac{SharedMem\_size}{SharedMem\_bytes\_per\_block} \right\rfloor$$

$$warps\_per\_block = \left\lceil \frac{Thread\_per\_block}{warp\_size} \right\rceil$$

$$Active\_blocks\_per\_SM = \min(\frac{Max\_active\_warps\_per\_SM}{warps\_per\_block},$$

$$Blks\_per\_ShrM, Max\_active\_blks\_per\_SM, \frac{Blocks}{Active\_SMs}) \tag{8}$$

Global Memory Accesses

$$data\_reqs(D_i) = \sum_k size\left(ShMem(D_i, k)\right) + \sum_j size\left(\mathbb{P}(\dot{\mathbb{A}}_j(D_i), \hat{\mathbb{B}})\right) \tag{9}$$

$$Avg\_elem\_bytes = \frac{\sum_i \left(ElemBytes(D_i) \times data\_reqs(D_i)\right)}{\sum_i data\_reqs(D_i)} \tag{10}$$

$$Load\_bytes\_per\_warp = Avg\_elem\_bytes \times warp\_size \tag{11}$$

$$caching\_mem\_insts(D) = \sum_k \left(\left\lceil \frac{size(Shmem(D, k))}{Thread\_per\_block} \right\rceil \times NumStages(k) \right) \tag{12}$$

$$direct\_mem\_insts\left(\dot{\mathbb{A}}_j(D)\right) = size\left(\mathbb{P}(\dot{\mathbb{A}}_j(D), \hat{\mathbb{F}})\right) \tag{13}$$

$$Coal\_Mem\_insts = \sum_j direct\_mem\_insts(\dot{\mathbb{A}}_j^\dagger) + \sum_i caching\_mem\_insts^\dagger(D_i) \tag{14}$$

$$Uncoal\_Mem\_insts = \sum_j direct\_mem\_insts(\dot{\mathbb{A}}_j') + \sum_i caching\_mem\_insts'(D_i) \tag{15}$$

$$Uncoal\_per\_mw = \frac{\sum_j Mem\_trans(\dot{\mathbb{A}}_j') + \sum_{i,k} Mem\_trans\left(ShMem'(D_i, k)\right)}{Uncoal\_Mem\_insts} \tag{16}$$

Computation Instructions

$$Func\_insts = F \times \sum_{i=1}^{N} (Ins_i \times LP_i) \tag{17}$$

Adopting GPU Performance Model

Hon and Kim’s GPU performance model

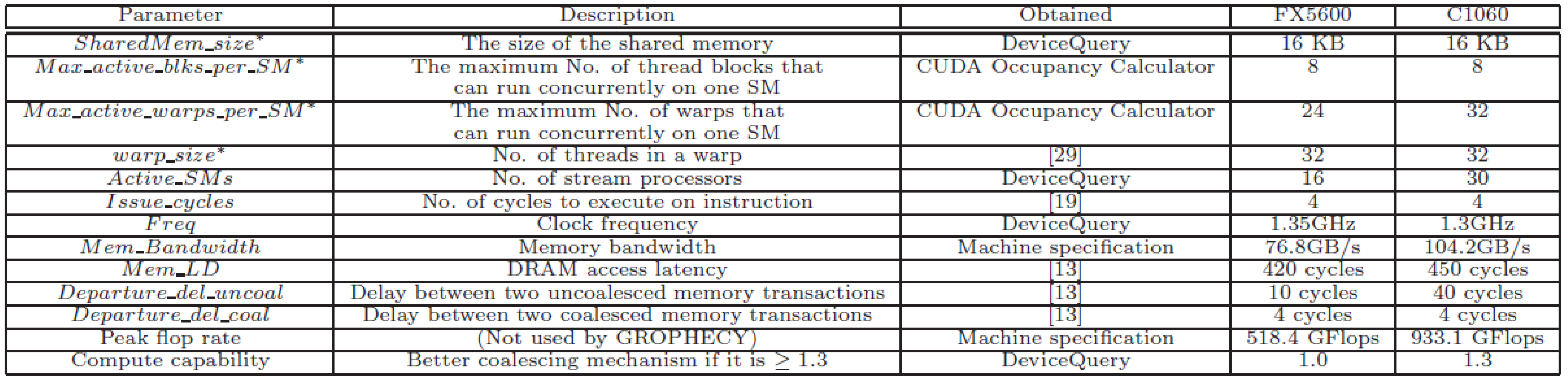
Agenda

Methodology

2 different generation GPUs

Quadro FX5600 Tesla C1060

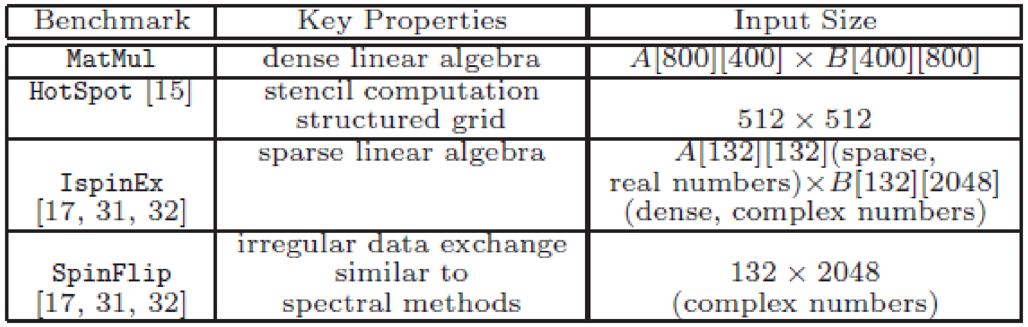
Table 5: Hardware parameters. Parameters marked with "*" are used in modeling code layouts and generating workload statistics. All other parameters are used by the underlying GPU performance model.



Parameter	Description	Obtained	FX5600	C1060
*SharedMem_size**	The size of the shared memory	DeviceQuery	16 KB	16 KB
*Max_active_blks_per_SM**	The maximum No. of thread blocks that can run concurrently on one SM	CUDA Occupancy Calculator	8	8
*Max_active_warps_per_SM**	The maximum No. of warps that can run concurrently on one SM	CUDA Occupancy Calculator	24	32
*warp_size**	No. of threads in a warp	[29]	32	32
Active_SMs	No. of stream processors	DeviceQuery	16	30
Issue_cycles	No. of cycles to execute on instruction	[19]	4	4
Freq	Clock frequency	DeviceQuery	1.35GHz	1.3GHz
Mem_Bandwidth	Memory bandwidth	Machine specification	76.8GB/s	104.2GB/s
Mem_LD	DRAM access latency	[13]	420 cycles	450 cycles
Departure_del_uncoal	Delay between two uncoalesced memory transactions	[13]	10 cycles	40 cycles
Departure_del_coal	Delay between two coalesced memory transactions	[13]	4 cycles	4 cycles
Peak flop rate	(Not used by GROPHECY)	Machine specification	518.4 GFlops	933.1 GFlops
Compute capability	Better coalescing mechanism if it is ≥ 1.3	DeviceQuery	1.0	1.3

Benchmarks

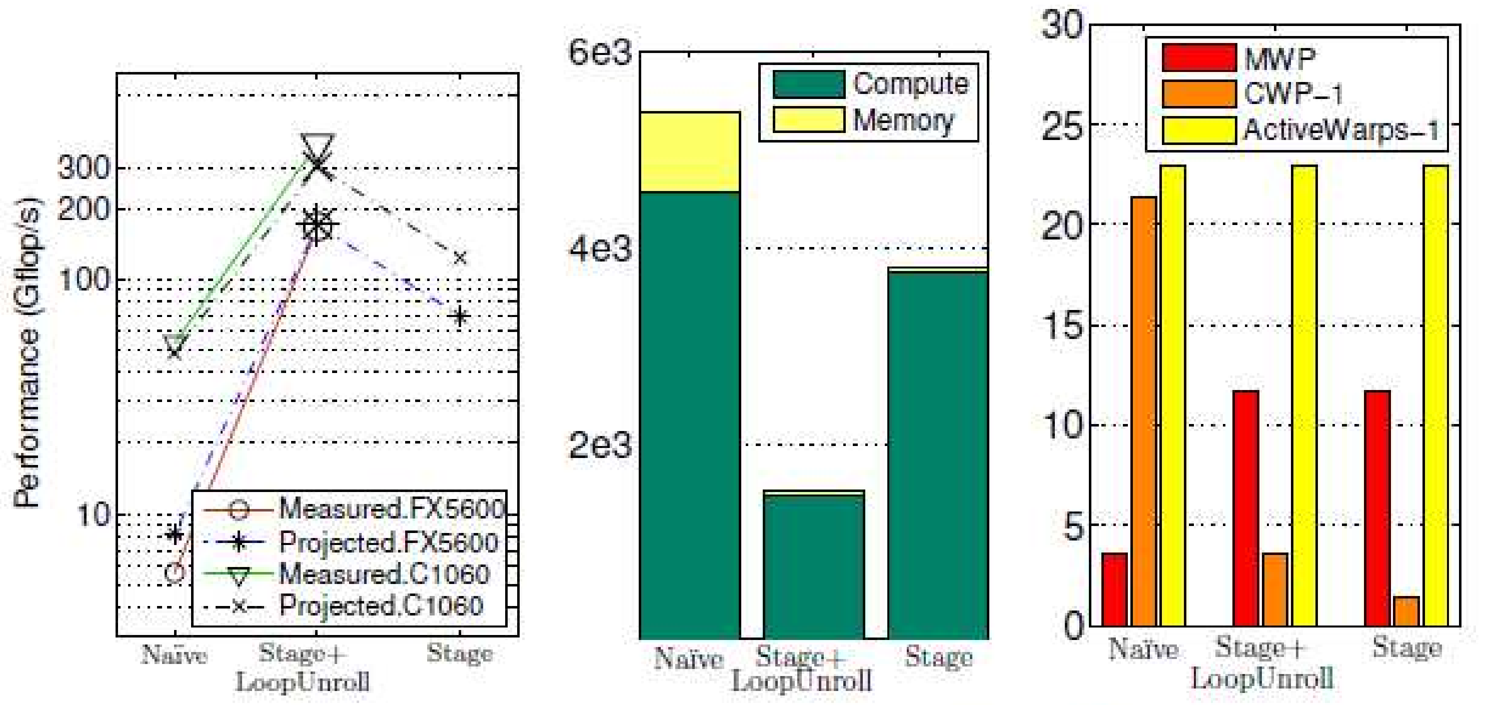
Table 4: Workload properties



Benchmark	Key Properties	Input Size
`MatMul`	dense linear algebra	$A[800][400] \times B[400][800]$
`HotSpot` [15]	stencil computation structured grid	512×512
`IspinEx` [17, 31, 32]	sparse linear algebra	$A[132][132]$(sparse, real numbers)$\times B[132][2048]$ (dense, complex numbers)
`SpinFlip` [17, 31, 32]	irregular data exchange similar to spectral methods	132×2048 (complex numbers)

Agenda

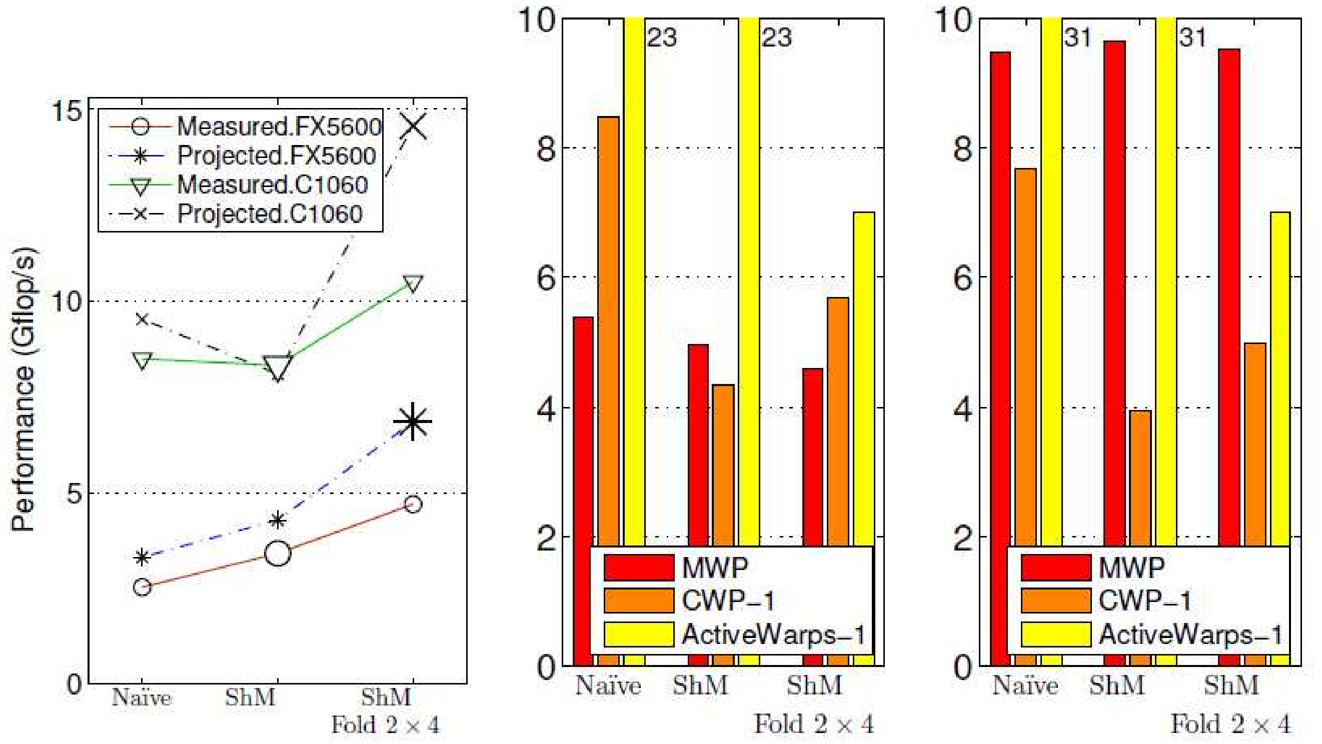
MatMul: Stating and Loop Unrolling



(a) Performance (b) Inst.Count(Fx5600) (c) Bottleneck(FX5600)

Figure 2: Validating projections of MatMul. In all cases, the thread block size is 16×16. Enlarged markers for measured data correspond to manually tuned implementations. Enlarged markers for projected data correspond to code layouts suggested by GROPHECY. Staging reduces memory instructions and transforms the GPU kernel from memory bounded ($MWP < CWP - 1$) to computation bounded ($MWP > CWP - 1$). Without loop unrolling, the number of computation instructions almost doubles.

HotSpot: Folding and Coalescing



(a) Performance (b) Bottleneck(FX5600) (c) Bottleneck(C1060)

Figure 3: Validating projections of HotSpot. In all cases, the thread block size is 16×16. Enlarged markers for measured data correspond to manually tuned implementations. Enlarged markers for projected data correspond to code layouts suggested by GROPHECY. Naïve is memory bounded ($MWP < CWP - 1$) on FX5600, but it is already computation bounded ($MWP > CWP - 1$) on C1060 because of better coalescing. Therefore caching has different effects on the two GPU hardware.

IspinEx: Sparsity and Code Restructuring

Listing 4: Baseline code skeleton of IspinEx. Array B is in the form of complex numbers

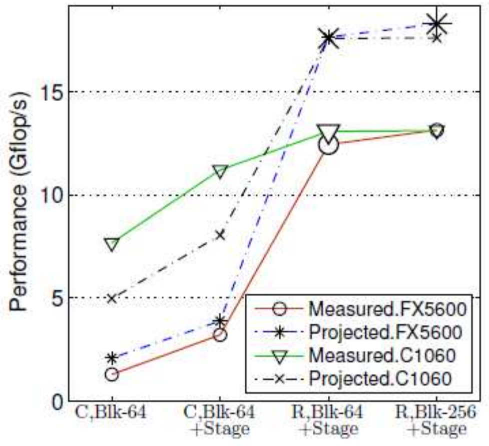
```
/* compute data as
 * complex numbers
 */
#define ELEMS 1848
#define ROWS 132
#define COLS 2048

int J[ROWS+1]
int I[ELEMS]
float T[ELEMS]
float B[ROWS][COLS][2]
float C[ROWS][COLS][2]
parallel_for(ROWS, COLS)
: j, i
{
  ld J[j]
  ld J[j+1]
  begin = J[j]
  end = J[j+1]
  comp 4
  /* The No. of nonzero
   * elements depends
   * on data in array J.
   * Non-constant boundary
   * disables unrolling.
   * Hint the average loop
   * size.
   */
  stream n = begin:end
  (hint:14)
  {
    ld T[n]
    ld I[n]
    r = I[n]
    /* indirect accesses to
     * the complex number
     */
    /* real part */
    ld B[r][i][0]
    /* imaginal part */
    ld B[r][i][1]
    /* sum+=T[n]*B[r][i] */
    comp 22
  }
  /* C[j][i] = sum */
  comp 2
  st C[j][i][0]
  st C[j][i][1]
}
```

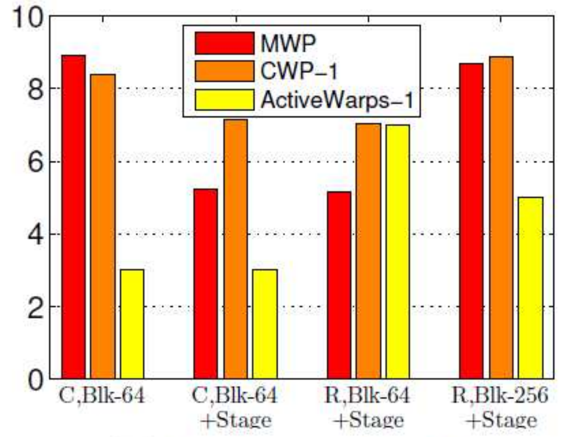
Listing 5: Modified code skeleton of IspinEx. Array B is in the form of real numbers

```
/* compute data as
 * complex numbers
 */
#define ELEMS 1848
#define ROWS 132
#define COLS 4096

int J[ROWS+1]
int I[ELEMS]
float T[ELEMS]
float B[ROWS][COLS]
float C[ROWS][COLS]
parallel_for(ROWS, COLS)
: j, i
{
  ld J[j]
  ld J[j+1]
  begin = J[j]
  end = J[j+1]
  comp 4
  /* The No. of nonzero
   * elements depends
   * on data in array J.
   * Non-constant boundary
   * disables unrolling.
   * Hint the average loop
   * size.
   */
  stream n = begin:end
  (hint:14)
  {
    ld T[n]
    ld I[n]
    r = I[n]
    /* indirect accesses */
    ld B[r][i]
    /* sum+=T[n]*B[r][i] */
    comp 11
  }
  /* C[j][i] = sum */
  comp 2
  st C[j][i]
}
```



(a) Single-precision performance



(b) Bottleneck on C1060

Figure 4: Validating projections of IspinEx. Enlarged markers for measured data correspond to manually tuned implementations. Enlarged markers for projected data correspond to code layouts suggested by GROPHECY. Although C1060 has more SMs, each SM has an insufficient number of warps to hide latency ($ActiveWarps - 1 < MWP$ and $ActiveWarps - 1 < CWP - 1$) because of limited input size of real data set. Therefore C1060 does not gain much performance compared with FX5600.

SpinFlap: Indirect Accesses and Thread Block sizes

Listing 6: Code skeleton of SpinFlap

```
#define ROWS 132
#define COLS 2048
float A[ROWS][COLS][2]
float B[ROWS][COLS][2]
float C[ROWS][COLS][2]
/* M: index array for indirect accesses.
 *    A sample data array is provided as a hint to better assess coalescing.
 */
int M[COLS/4][4] : hints<sample="./M.txt">
parallel_for(ROWS, COLS/4) : j, i
{
  for n = 0:4
  {
    ld M[i][n]
    /* load the complex number in A */
    ld A[j][M[i][n]][0]
    ld A[j][M[i][n]][1]
    /* load the complex number in B */
    ld B[j][M[i][n]][0]
    ld B[j][M[i][n]][1]
    comp 56
  }
  comp 228
  /* produce the complex numbers */
  for n = 0:4
  {
    /* store the computed complex number */
    st C[j][M[i][n]][0]
    st C[j][M[i][n]][1]
  }
}
```

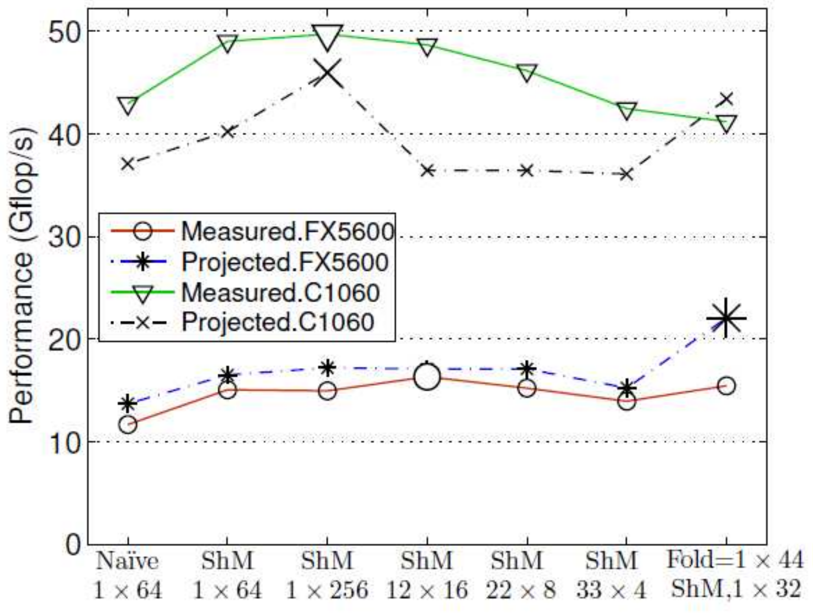


Figure 5: Validating projections of SpinFlap in single precision. Enlarged markers for measured data correspond to manually tuned implementations. Enlarged markers for projected data correspond to code layouts suggested by GROPHECY. GROPHECY can project the performance for workloads with indirect accesses as well.

Agenda

Limitations

- Modifying algorithm
- Restructuring data
- Automatic parallelization
- Non representative control path
- Data-independent control flow
- Texture memory and constant memory
- Model instruction level parallelism
- Double precision kernels
- Data transfer time between CPU and GPU

Agenda

Conclusion

- **GROPHECY**
  - Fast GPU performance projection framework
  - Without actual GPU hardware or GPU programming
  - Significantly reduces the performance evaluation process
  - Projected performance vs manually tuned code:
    - 17 % (geometric mean), 31% (maximum)
    - Worst case: 95% performance compared with manually tuned code
    - Best case: 1.37x performance compared with manually tuned code

Comments

- There are still tedious and error-prone processes depended on users of GROPHECY.
- There are still many limitations on GROPHECY.
- The name, “GROPHECY” is not familiar and difficult to remember and pronounce…