A Script-Based Autotuning Compiler System to Generate High-Performance CUDA code

Malik Khan, **Protonu Basu**, Gabe Rudy, Mary Hall, Chun Chen, Jacqueline Chame

USC UNIVERSITY OF SOUTHERN CALIFORNIA

THE UNIVERSITY OF UTAH

Motivation

Challenges to programming the GPU

- **Parallel Computation Partitioning**
- **Data placement in memory hierarchy**
- **Memory Bandwidth optimizations**

Best solution depends on architecture and input data set

Target Devices

	GTX-280	C2050
#SMs	30	14
Cores/SM	8	32
Total cores	240	448
Peak (SP)	933 GF/s	1.03 TF/s
Peak (DP)	87 GF/s	515 GF/s
Global memory	1 GB	3 GB
Bandwidth	142 GB/s	144 GB/s
Shared memory/ SM	16 KB	(up to) 48 KB
Registers/ SM	16K	32K
"Texture" accesses	Yes	Yes
Data cache	0	(up to) 48 KB

C2050: more cores / SM, but fewer SMs

C2050: less registers and shared memory per core

THE UNIVERSITY OF UTAH

Our Approach

Autotuning

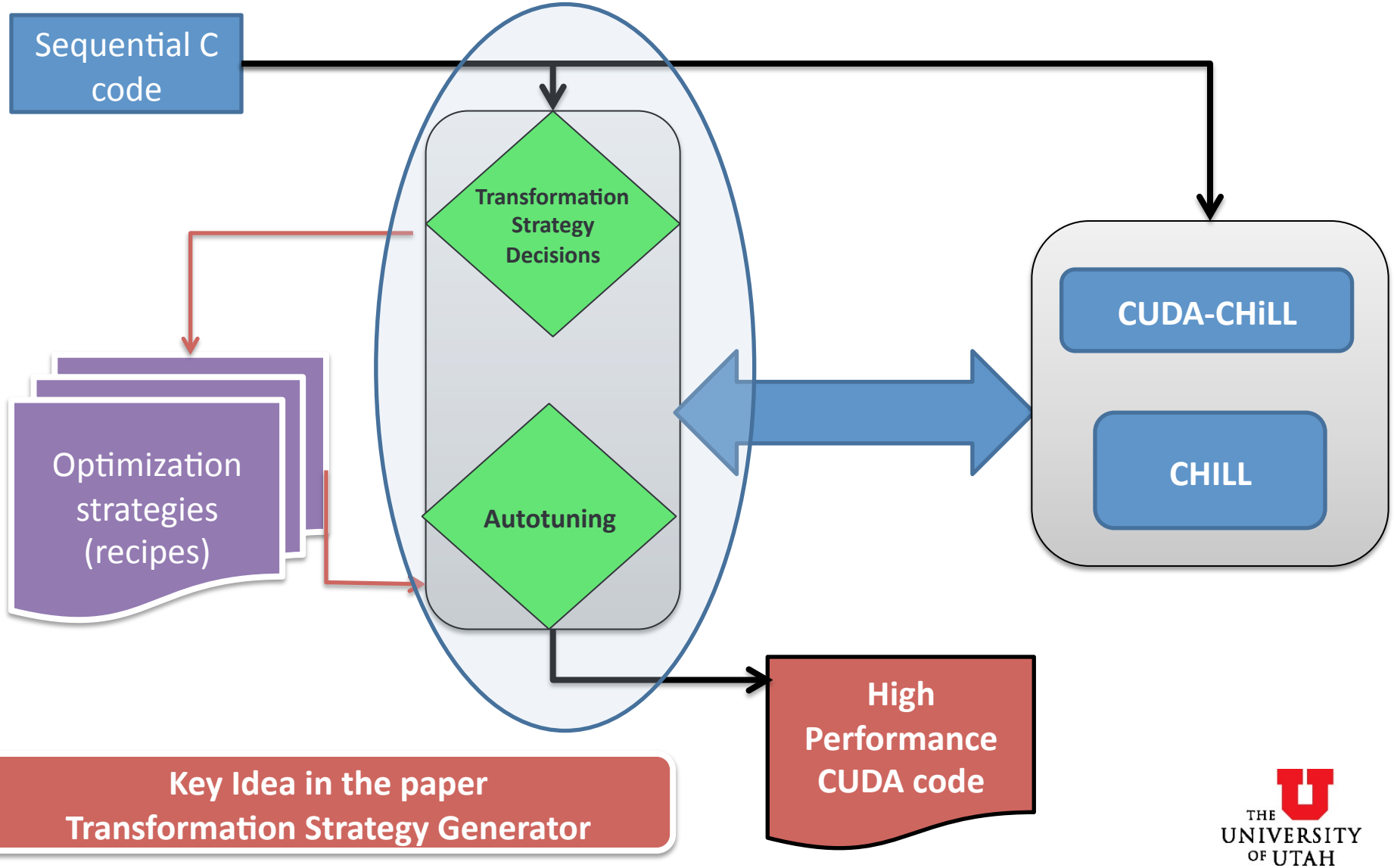
- Automatically generate a "search space" of possible implementations
- A code variant represents a unique implementation amongst many
- A parameter represents a discrete set of values that govern code generation of a variant

A layered autotuning compiler framework

- Enhances productivity of naïve and expert programmers
- Separates code generation from optimization strategy
- Facilitates exploration of a search space of strategies

THE UNIVERSITY OF UTAH

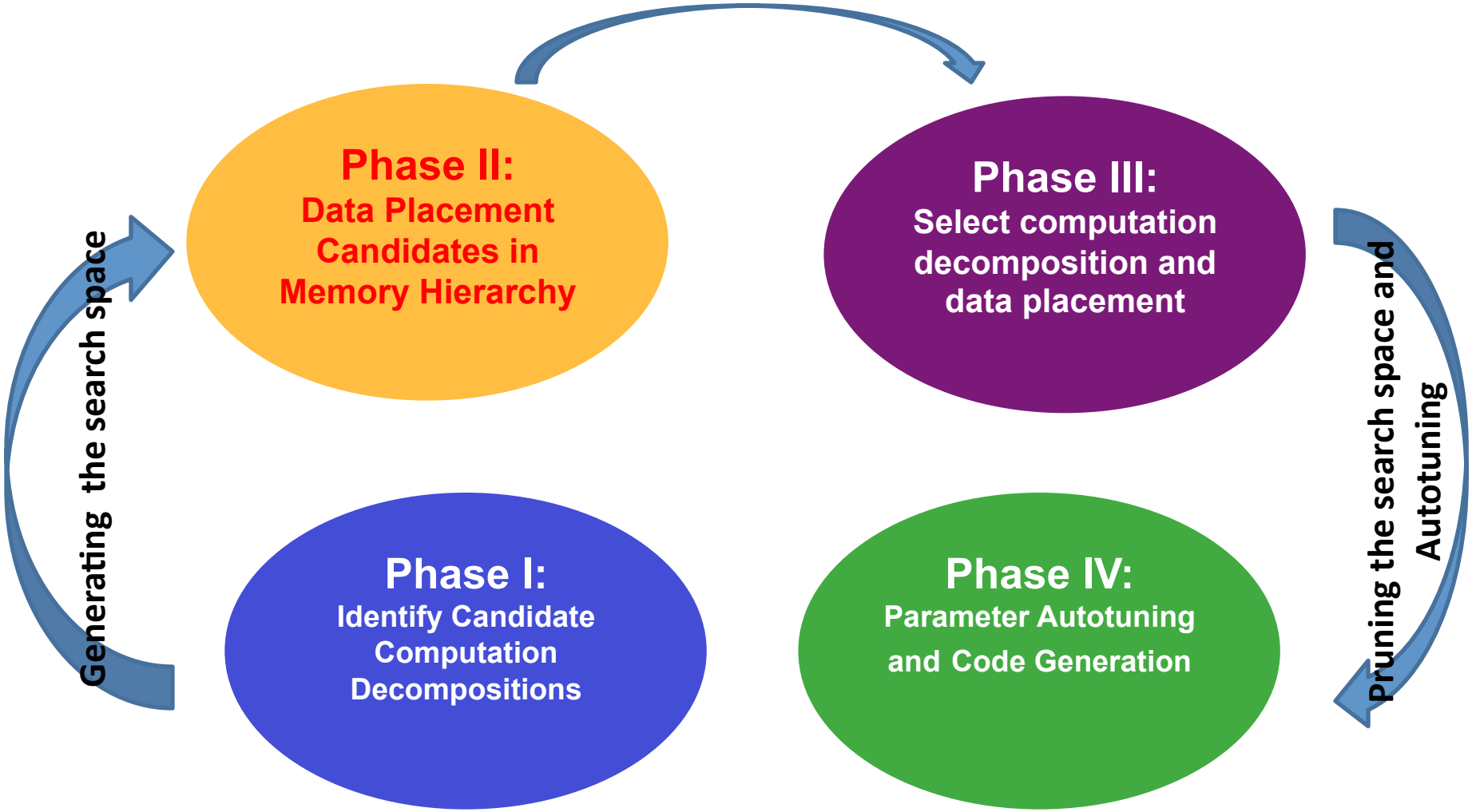
Compiler Framework for Autotuning



Key Idea in the paper
Transformation Strategy Generator

THE UNIVERSITY OF UTAH

TSG: Four Phase Approach



THE UNIVERSITY OF UTAH

TSG: Identifying Strategy

Parallel Mapping:

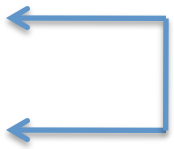
- Constrained by dependences
- Global memory coalescing in x thread dimension
- Evaluate data partitioning options

Manage Heterogeneous Memory Hierarchy:

- **Register Candidate:** Reuse in a thread
- **Shared Memory:** Reuse across thread or non-coalesced global memory access
- **Texture Memory:** Read-only data already mapped to registers or shared memory
- **Constant Memory:** Read-only, with reuse across threads (and blocks) and short reuse distance

Example : Matrix-Matrix Multiply

```
for I
  for j
    for k
      c[j][i] += a[k][i] * b[j][k];
```



I and j loops: tile to create block and thread dimensions
2-d data structures ➔ 2-d blocks and threads

Dependence carried by k loop
Move inside thread

Global memory coalescing – i loop as x thread dimension

Data placement candidates:

Registers: c
Shared memory: a, b
Texture memory: a, b
Global memory: a
Constant memory: <empty>

Data partitioning options

C: cyclic x, block y
C: cyclic x, cyclic y

THE UNIVERSITY OF UTAH

Example : Matrix-Matrix Multiply

TC2050 Fermi implementation
Mostly corresponds to CUBLAS 3.2 and MAGMA

```
1 tile_by_index({"i","j"},{TI,TJ},{l1_control="ii",l2_control="jj"},
                {"ii","jj","i","j"})
2 tile_by_index({"k"},{TK},{l1_control="kk"},
                {"ii","jj","kk","i","j","k"},strided)
3 tile_by_index({"i"},{TJ},{l1_control="tt",l1_tile="t"},
                {"ii","jj","kk","t","tt","j","k"})
4 cudaize("mm_GPU",{a=N*N,b=N*N,c=N*N},
          {block={"ii","jj"},thread={"t","tt"}})
5 copy_to_shared("tx","b",-16)
6 copy_to_registers("kk","c")
7 copy_to_texture("b")
8 unroll_to_depth(2)
```

GTX-280 implementation
Mostly corresponds to CUBLAS 2.x and Volkov's SC08 paper

```
1  tile_by_index({"i","j"},{TI,TJ},{l1_control="ii",l2_control="jj"},
                 {"ii","jj","i","j"})
2  tile_by_index({"k"},{TK},{l1_control="kk"},
                 {"ii","jj","kk","i","j","k"},strided)
3  tile_by_index({"i"},{TK},{l1_control="t",l1_tile="tt"},
                 {"ii","jj","kk","tt","t","j","k"})
4  tile_by_index({"j"},{TK},{l1_control="s",l1_tile="ss"},
                 {"ii","jj","kk","tt","t","ss","s","k"})
5  cudaize("mm_GPU",{a=N*N,b=N*N,c=N*N},
           {block={"ii","jj"},thread={"tt","ss"}})
6  copy_to_shared("tx","b",-16)
7  copy_to_texture("b")
8  copy_to_shared("tx","a",-16)
9  copy_to_texture("a")
10 copy_to_registers("kk","c")
11 unroll_to_depth(2)
```

Different computation decomposition leads to additional tile command

Now a also in (larger) shared memory, both a and b are read through texture memory

THE UNIVERSITY OF UTAH

Pruning the Search Space

Parameterized Code Variant Selection

- Heuristics limit variants to promising ones
- Based on architecture features and data access properties (coalescing, reuse within/across threads)

Optimization Parameters

- Architectural knowledge (warp size, memory capacities)
- Load balance model

TSG autotunes parameterized variant selection and optimization parameters independently to further help prune the search space

Benchmarks

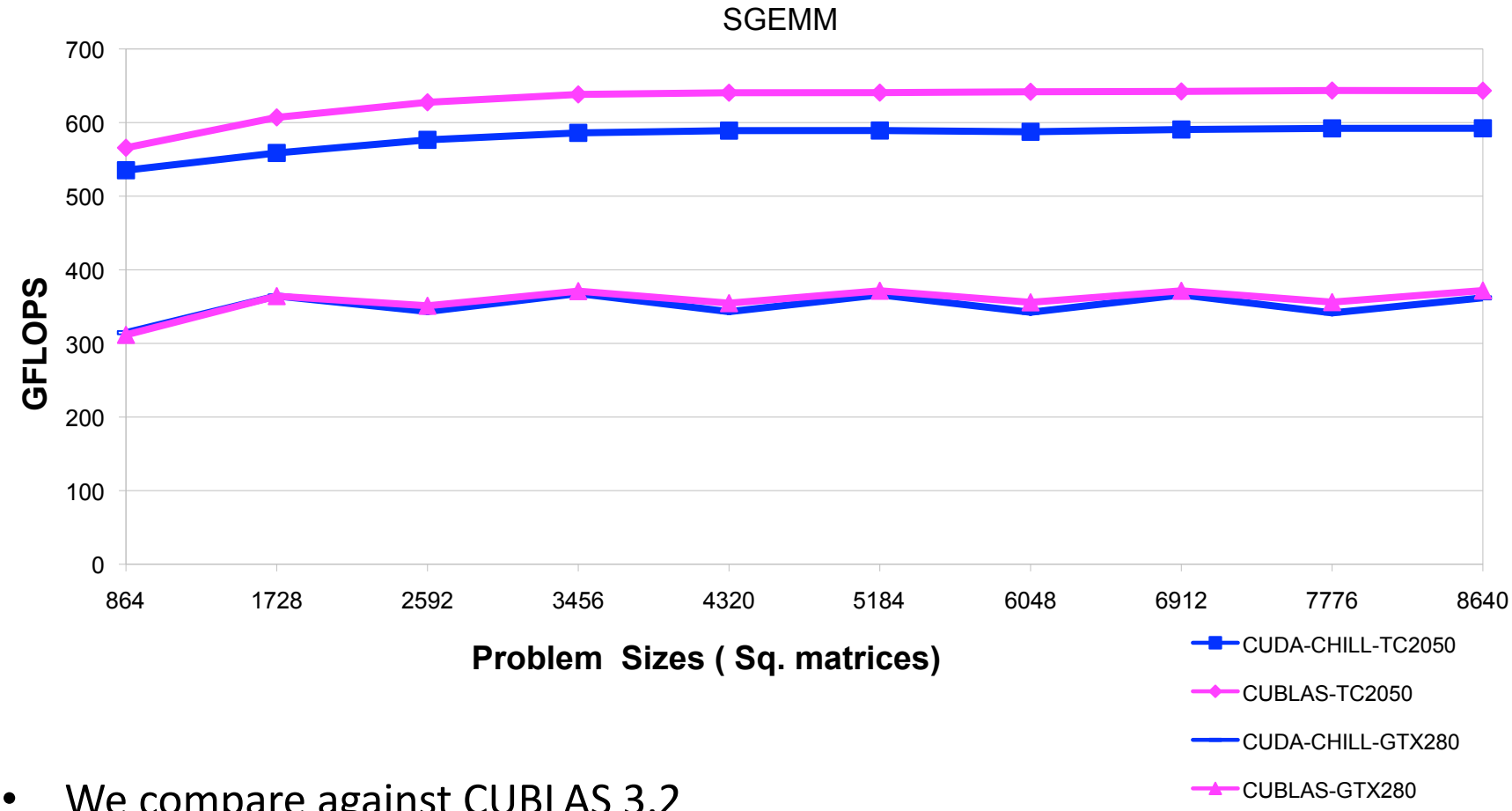
	Domain	Source
MM	**Linear Algebra**	**CUBLAS**
MV	**Linear Algebra**	**CUBLAS**
TMV	**Linear Algebra**	**CUBLAS**
CP	**Scientific Computation**	**PLUTO**
NBody	**Scientific Computation**	**PLUTO**
MPEG4	**Multimedia**	**PLUTO**
MRI-FH	**Imaging**	**PLUTO**
MRIQ	**Imaging**	**PLUTO/ PARBOIL**

THE UNIVERSITY OF UTAH

Search Space Size

	Computation Partitioning Phase						Pruning and Autotuning Phase				
	I/P	Loop Perms. Pruned		O/P Comb.	Decomp. Map.	Total Var.	Potential	Evaluation Versions			Total
Kernel	Comb.	Data Dep.	Mem. Coal.				Variants	Opt.	parms	Eval.	%Pruned
MM	6	4	1	1	4	5	7936	124	64	188	97.63
MV	2	1	0	1	1	2	560	35	16	51	90.80
TMV	2	1	0	1	1	2	560	35	16	51	90.80
MRIQ	2	1	0	1	1	2	176	11	16	27	84.65
Conv	24	22	1	1	4	5	9984	156	64	220	97.79
MRIFH	2	1	0	1	1	2	5104	319	16	335	93.43
MPEG 4	24	22	1	1	4	5	9984	156	64	220	97.79
CP	6	4	1	1	4	5	2816	44	64	108	96.16
Nbody	2	1	0	1	1	2	304	19	16	35	88.48

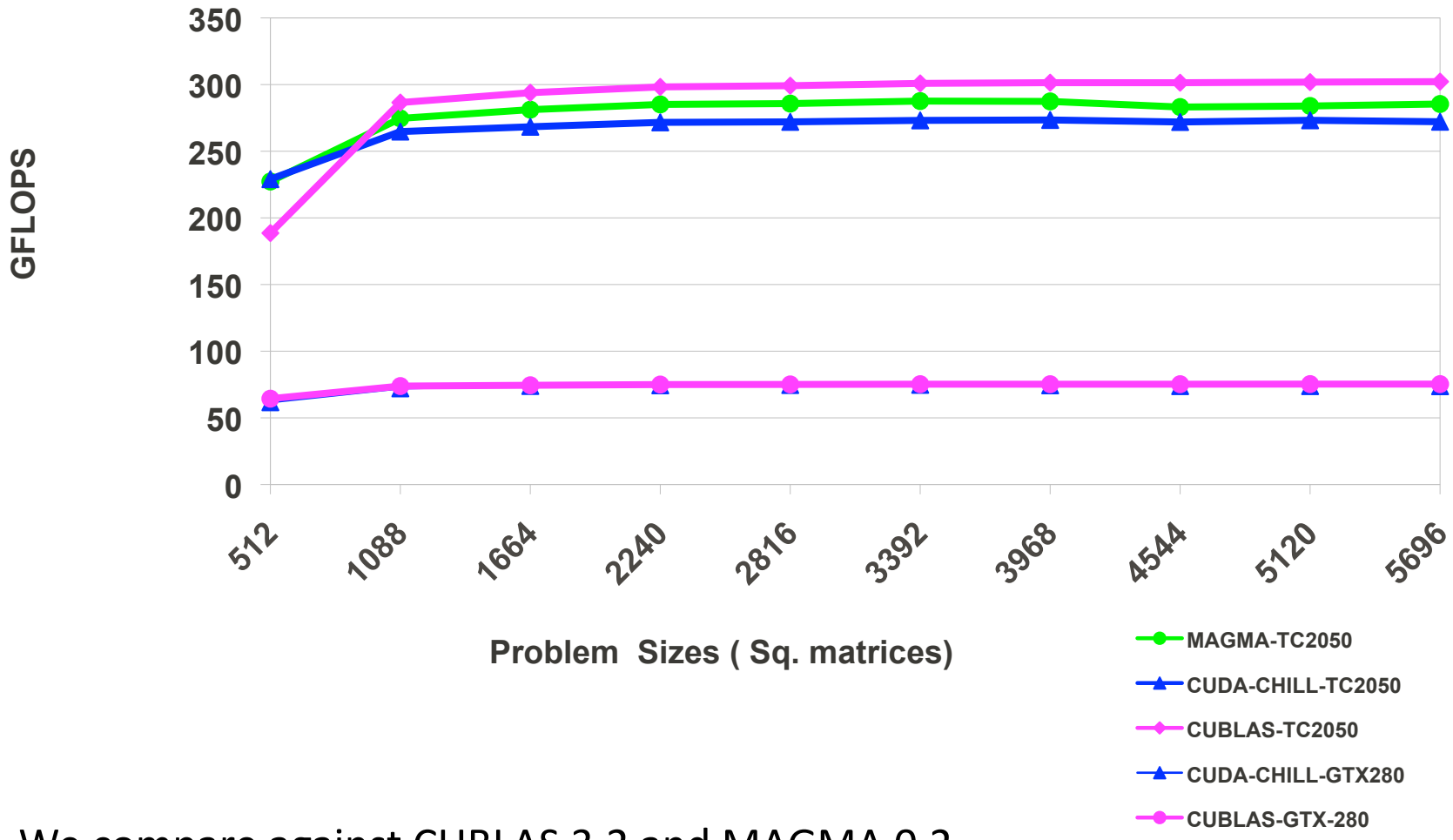
Results: SGEMM



- We compare against CUBLAS 3.2
- On GTX-280 we are within 2%
- On C2050 we are within 8%

THE UNIVERSITY OF UTAH

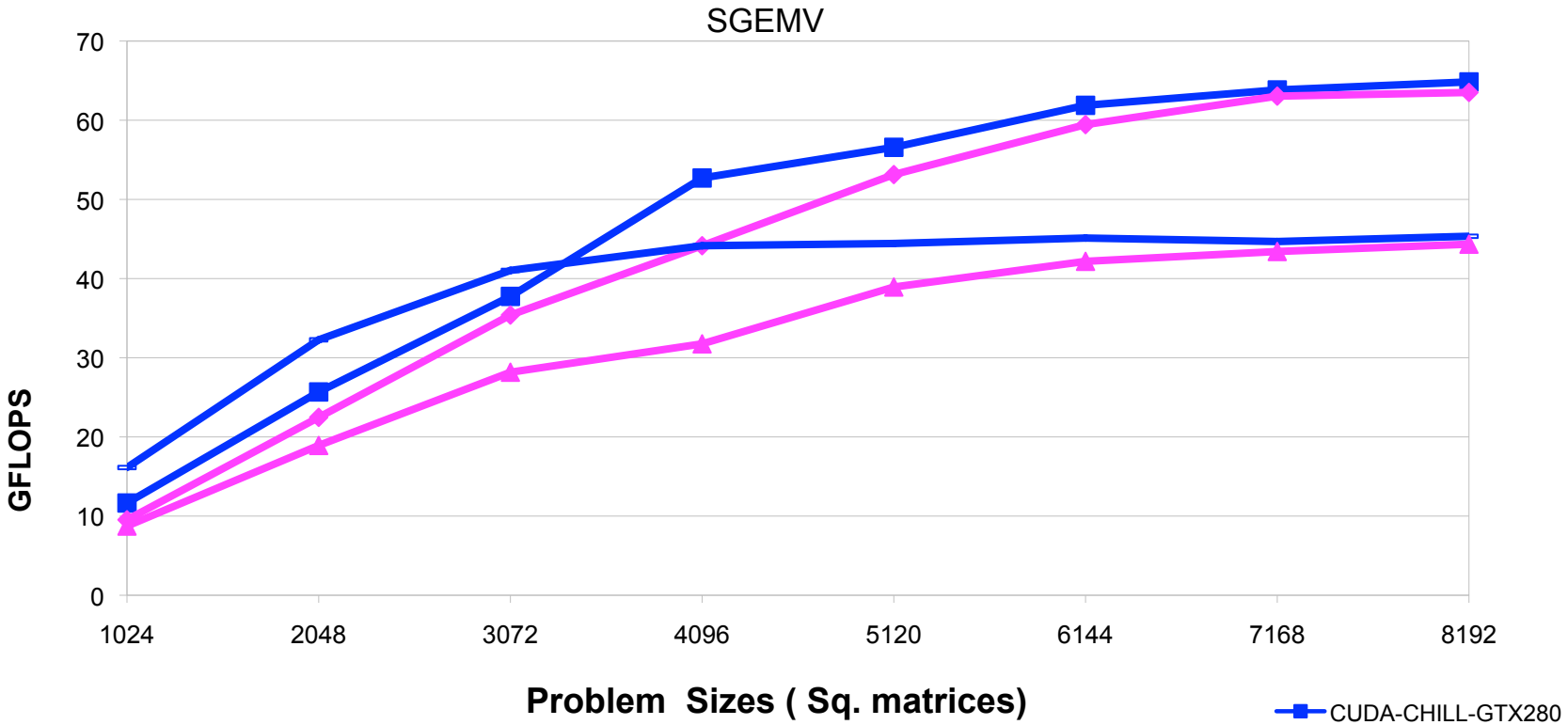
Results: DGEMM



- We compare against CUBLAS 3.2 and MAGMA 0.2
- On GTX-280 we are within 2% of CUBLAS
- On C2050 we are within 10% of CUBLAS

THE UNIVERSITY OF UTAH

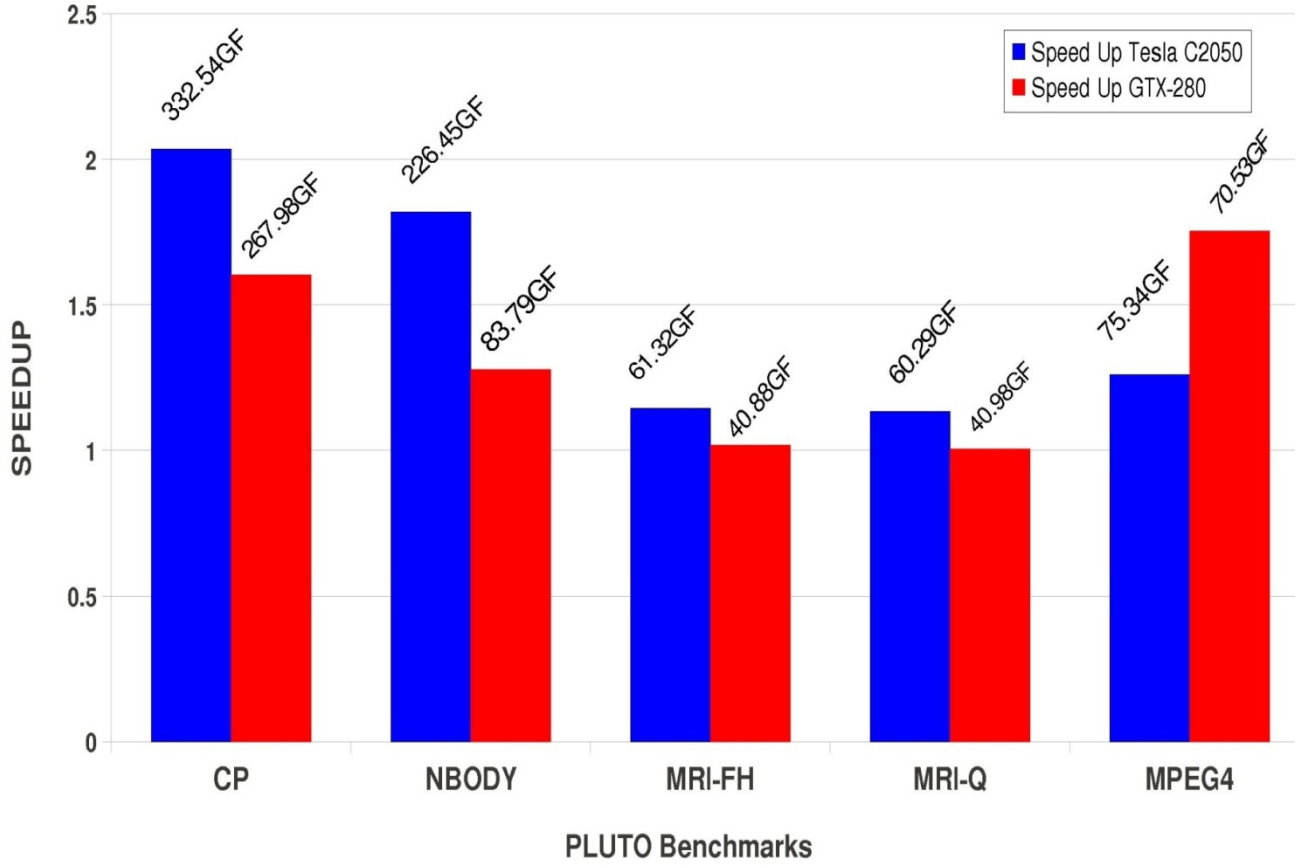
Results: SGEMV



- We compare against CUBLAS 3.2
- Compiler generated code outperforms CUBLAS
- Speedup of up to 1.22x of GTX-280, 1.84x on C2050
- Raw performance of SP MV higher of GTX-280
- Similar computation partitioning and data placement strategy used
- Thread/block larger on C2050

THE UNIVERSITY OF UTAH

Comparison against a state-of-the-art GPU compiler based on PLUTO



CP shows the highest speedup of 2.03x

Average speedup of 1.48X (C2050) and 1.33X (GTX-280).

THE UNIVERSITY OF UTAH

Summary and Contributions

Transformation Strategy Generator (TSG) algorithm

Meta-optimizer that generates a collection of transformation recipes for parallelization targeting GPUs

Demonstrate performance portable code generation across execution contexts

High performance on two devices and across input problem sizes

Achieves high performance

Comparison with manually-tuned and state-of-the-art GPU compiler-generated code

THE UNIVERSITY OF UTAH