

OpenMPC: Extended OpenMP Programming and Tuning for GPUS

Yajaira Gonzalez Gonzalez
May 09 2011

Background

- Based on previous work:
 - OpenMP to GPGPU: A Compiler Framework for Automatic Translation and Optimization. [2]
- Contributions:
 - Modified version of previous translator.
 - Introduces the OpenMPC programming interface.

OpenMP to Cuda Translation

OpenMP-To-CUDA Translation System

- Baseline Translation

1. Interpretation of OpenMP semantics under the CUDA API.
2. Transform eligible regions into CUDA Kernel functions.
(Includes memory transfer code from CPU to GPU.)

OpenMP-To-CUDA Translation System

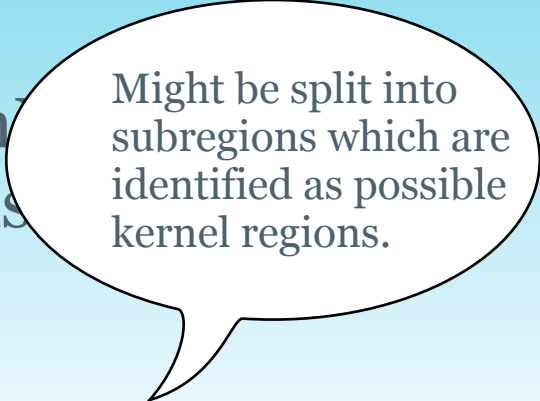
1. Interpretation of OpenMP semantics

OpenMP directives classification:

- Parallel construct:
 - ie. (omp parallel)
- Work Sharing constructs:
 - ie. (omp for, omp sections)
- Synchronization Constructs
 - ie. (omp barrier, omp critical)
- Directives for data properties
 - ie. (omp shared, omp private)

OpenMP-To-CUDA Translation System

1. Interpretation of OpenMP OpenMP directives classification



Might be split into subregions which are identified as possible kernel regions.

- Parallel construct:
 - ie. (omp parallel)
- Work Sharing constructs:
 - ie. (omp for, omp sections)
- Synchronization Constructs
 - ie. (omp barrier, omp critical)
- Directives for data properties
 - ie. (omp shared, omp private)

OpenMP-To-CUDA Translation System

1. Interpretation of OpenMP semantics

OpenMP directives classification:

- Parallel construct:
 - ie. (omp parallel)
- Work Sharing constructs:
 - ie. (omp for, omp sections)
- Synchronization Constructs
 - ie. (omp barrier, omp critical)
- Directives for data properties
 - ie. (omp shared, omp private)



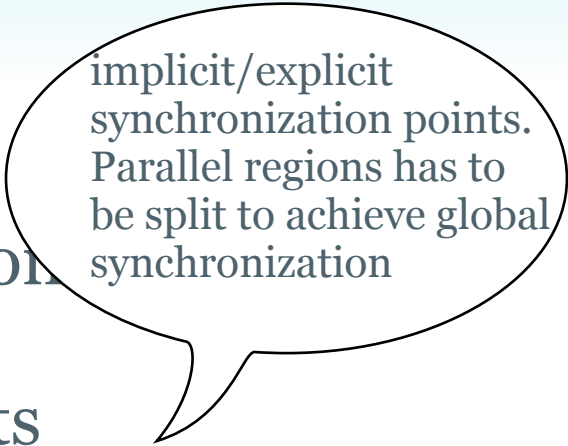
True parallel regions,
other regions are
executed by one thread,
serialized or redundantly
by all the threads

OpenMP-To-CUDA Translation System

1. Interpretation of OpenMP semantics

OpenMP directives classification:

- Parallel construct:
 - ie. (omp parallel)
- Work Sharing constructs:
 - ie. (omp for, omp section)
- Synchronization Constructs
 - ie. (omp barrier, omp critical)
- Directives for data properties
 - ie. (omp shared, omp private)



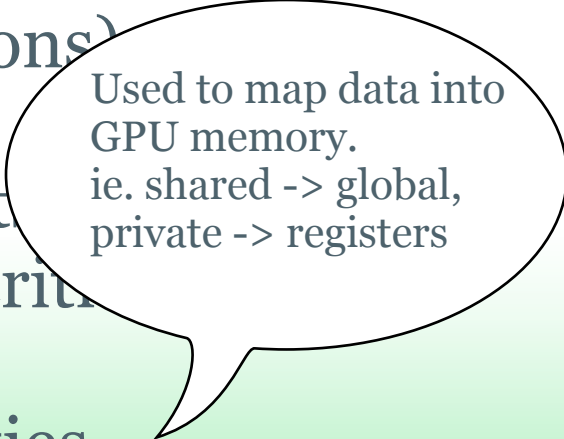
implicit/explicit
synchronization points.
Parallel regions has to
be split to achieve global
synchronization

OpenMP-To-CUDA Translation System

1. Interpretation of OpenMP semantics

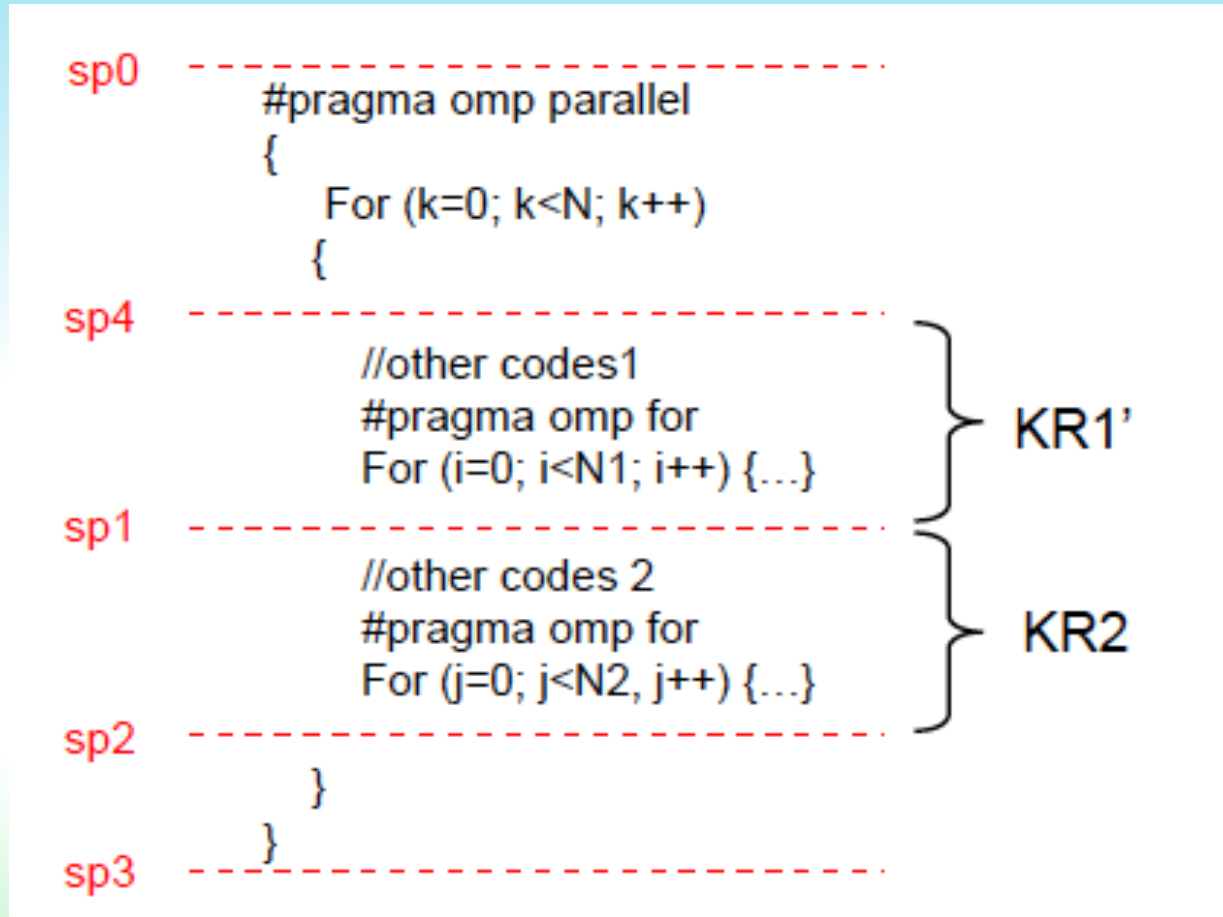
OpenMP directives classification:

- Parallel construct:
 - ie. (omp parallel)
- Work Sharing constructs:
 - ie. (omp for, omp sections)
- Synchronization Construct
 - ie. (omp barrier, omp critical)
- Directives for data properties
 - ie. (omp shared, omp private)



Used to map data into GPU memory.
ie. shared -> global,
private -> registers

OpenMP-To-CUDA Translation System



OpenMP-To-CUDA Translation System

2. Transformation of kernel regions into Kernel Functions.

Kernel regions transformations:

- Work Partitioning
- Data Mapping
- Work Partitioning
 - Each iteration of for loop -> 1 thread
 - Each section -> 1 thread
 - Remaining code sections -> all threads

OpenMP-To-CUDA Translation System

Kernel Region Transformations:

- Data Mapping
 - Data referenced in kernel region but not in construct translator can determine sharing attributes using OpenMP default sharing rules.
 - Default data mapping follows "Directive Specifying data properties" mentioned before.
 - To exploit temporal locality ->
 - Assigned Read Only shared data to constant / texture memory.
 - Frequently reused shared data -> register and shared memory as cache.

Compiler Optimizations

- Compiler Optimizations
 - CPU and GPU data movements.
 - GPU global memory access optimizations.
 - Exploit GPU on chip memory.

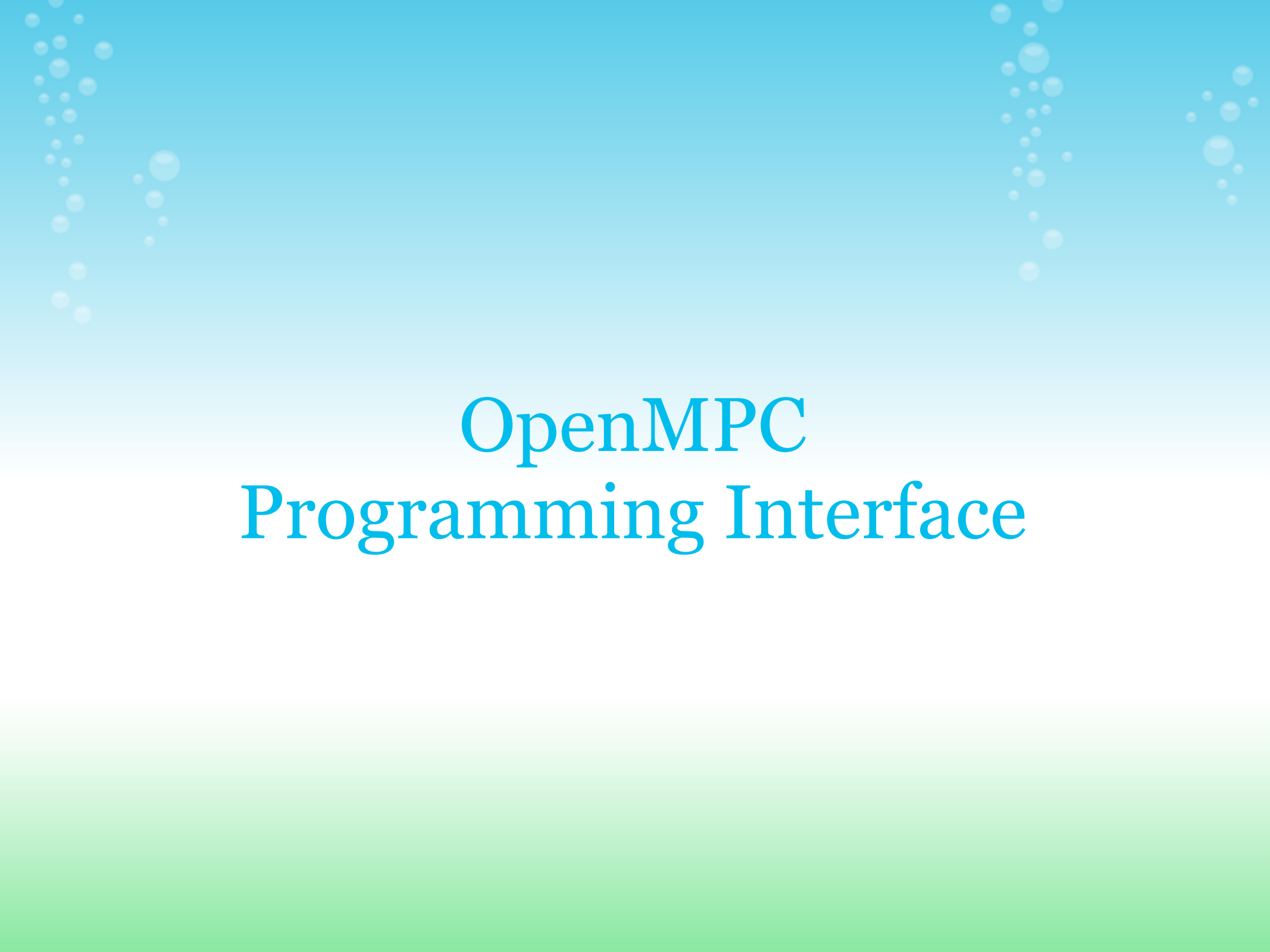
Compiler Memory Optimizations

- CPU to GPU data movement.
 - Detects whether variable in GPU has same content as variable in the CPU.
 - Detects GPU variables that are not read by the CPU before a write.
- GPU to global memory and on chip memory exploitation.
 - Using SM, Registers, Constant and Texture Memory

TABLE V

CACHING STRATEGIES. *Reg* DENOTES *Registers*, *CM* MEANS *Constant Memory*, *SM* IS *Shared Memory*, AND *TM* REPRESENTS *Texture Memory*.

Variable Type	Caching Strategy
R/O shared scalar w/o locality	SM
R/O shared scalar w/ locality	SM, CM, Reg
R/W shared scalar w/ locality	Reg, SM
R/W shared array element w/ locality	Reg
R/O 1-dimensional shared array	TM
R/W private array w/ locality	SM



OpenMPC Programming Interface

OpenMPC - Extended OpenMP for CUDA

Set of directives and environment variables to apply CUDA - specific optimization.

- Directive Extension

TABLE I
OPENMPC DIRECTIVE FORMAT

```
#pragma cuda gpurun [clause [,] clause ]...]  
#pragma cuda cpurun [clause [,] clause ]...]  
#pragma cuda nogpurun  
#pragma cuda ainfo procname(pName) kernelid(kID)
```

OpenMPC - Extended OpenMP for CUDA

TABLE I
OPENMPC DIRECTIVE FORMAT

```
#pragma cuda gpurun [clause [,] clause ]...]  
#pragma cuda cpurun [clause [,] clause ]...]  
#pragma cuda nogpurun  
#pragma cuda ainfo procname(pName) kernelid(kID)
```

gpurun: attached parallel region is eligible for kernel-region transformation.

cpurun: associated parallel region executed in the CPU.

nogpurun: dont transform attached kernel region, dont covert to kernel function.

ainfo:allows user to assign unique ids to each kernel region. Users can later refer to them in a separate directive file.

OpenMPC - Extended OpenMP for CUDA

- Environment Variables
 - Control program behavior of various optimizations or execution configurations for an output CUDA program.
 - Directives have higher priority.

OpenMPC - Extended OpenMP for CUDA

TABLE IV

BRIEF DESCRIPTION OF OPENMPC ENVIRONMENT VARIABLES, WHICH CONTROL PROGRAM-LEVEL BEHAVIORS OF VARIOUS OPTIMIZATIONS, THREAD BATCHINGS, DATA MAPPING STRATEGIES, AND TUNING LEVEL.

Parameter	Description	Category
maxNumOfCudaThreadBlocks=N	Set the maximum number of CUDA thread blocks	CUDA Thread Batching
cudaThreadBlockSize=N	Set the default CUDA thread block size	CUDA Thread Batching
shrdSclrCachingOnReg	Cache shared scalar variables onto GPU registers	OpenMP-to-CUDA Data Mapping
shrdArryElmtCachingOnReg	Cache shared array elements onto GPU registers	OpenMP-to-CUDA Data Mapping
shrdSclrCachingOnSM	Cache shared scalar variables onto GPU shared memory	OpenMP-to-CUDA Data Mapping
prvtArryCachingOnSM	Cache private array variables onto GPU shared memory	OpenMP-to-CUDA Data Mapping
shrdArryCachingOnTM	Cache 1-dimensional, R/O shared array variables onto GPU texture memory	OpenMP-to-CUDA Data Mapping
shrdCachingOnConst	Cache R/O shared variables onto GPU constant memory	OpenMP-to-CUDA Data Mapping
useMatrixTranspose	Apply Matrix Transpose optimization	OpenMP Stream Optimization
useLoopCollapse	Apply LoopCollapse optimization	OpenMP Stream Optimization
useParallelLoopSwap	Apply Parallel Loop-Swap optimization	OpenMP Stream Optimization
useUnrollingOnReduction	Apply loop unrolling for in-block reduction	CUDA Optimization
useMallocPitch	Use cudaMallocPitch() for 2-dimensional arrays	CUDA Optimization
useGlobalGMalloc	Allocate GPU variables as global variables	CUDA Optimization
globalGMallocOpt	Apply CUDA malloc optimization for globally allocated GPU variables	CUDA Optimization
cudaMallocOptLevel=N	Set CUDA malloc optimization level for locally allocated GPU variables	CUDA Optimization
cudaMemTrOptLevel=N	Set CUDA CPU-GPU memory transfer optimization level	CUDA Optimization
assumeNonZeroTripLoops	Assume that all loops have non-zero iterations	Optimization Configuration
tuningLevel=N	Set tuning level (0: Program-level tuning 1: Kernel-level tuning)	Tuning Configuration

Compilation for OpenMPC

Overall Compilation Flow

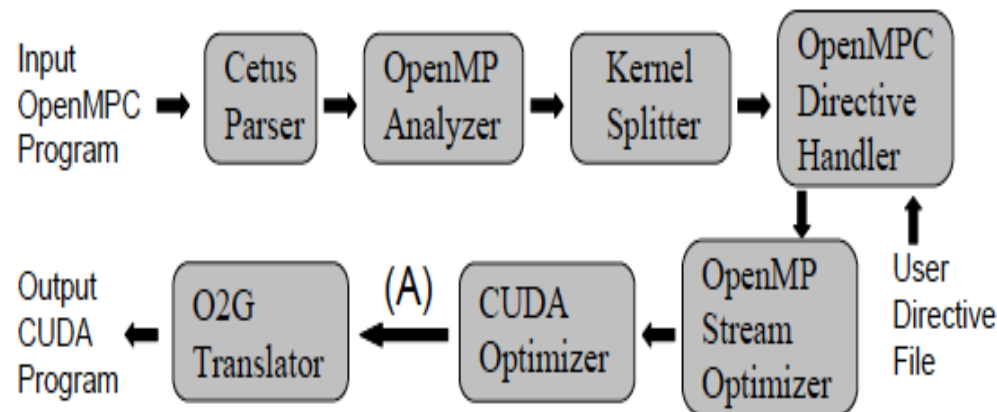


Fig. 3. Overall Compilation Flow. When the compilation system is used for automatic tuning, additional passes are invoked between *CUDA Optimizer* and *O2G Translator*, marked as (A) in the figure (See Figure 4)

Compilation for OpenMPC

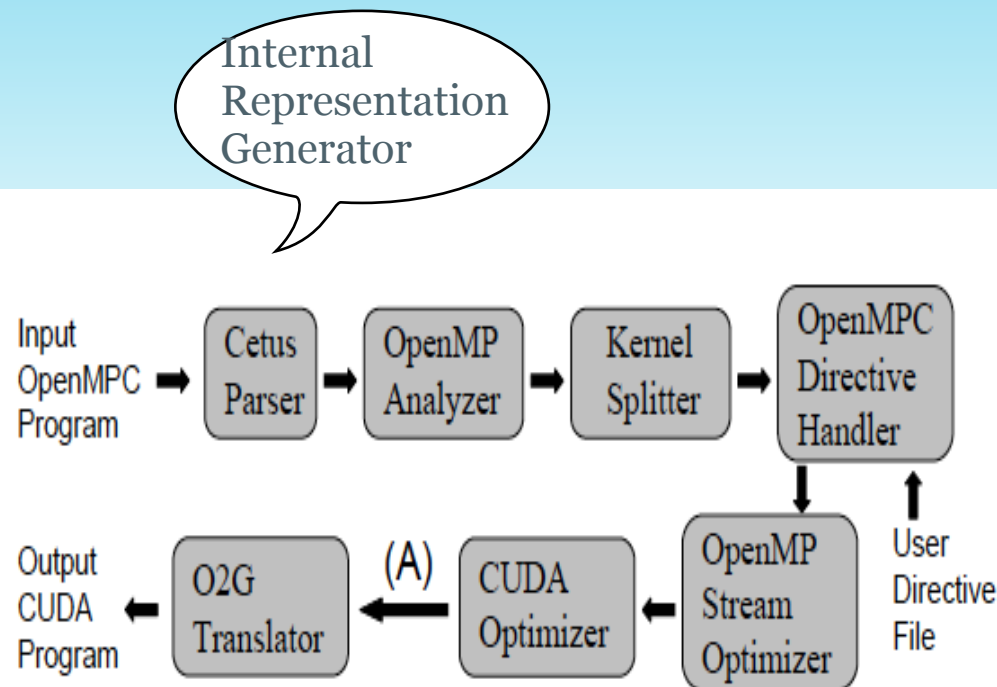


Fig. 3. Overall Compilation Flow. When the compilation system is used for automatic tuning, additional passes are invoked between *CUDA Optimizer* and *O2G Translator*, marked as (A) in the figure (See Figure 4)

Compilation for OpenMPC

Find OpenMP, shared, threadprivate and private variables, implicit and explicit barriers

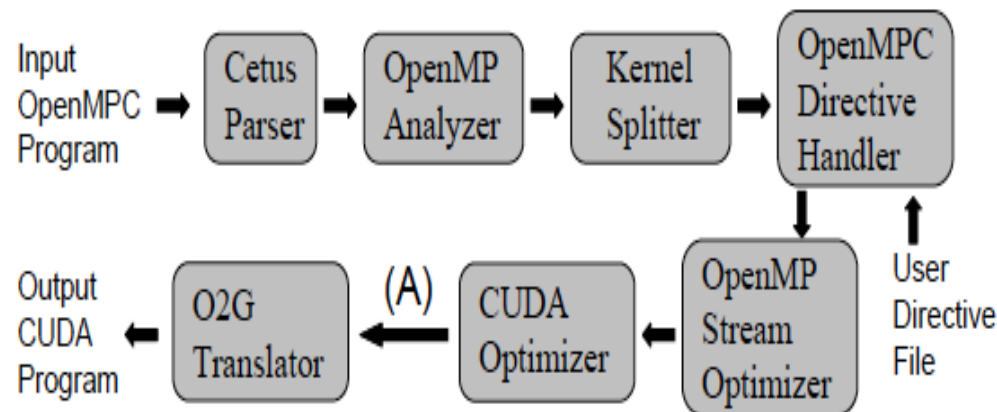


Fig. 3. Overall Compilation Flow. When the compilation system is used for automatic tuning, additional passes are invoked between *CUDA Optimizer* and *O2G Translator*, marked as (A) in the figure (See Figure 4)

Compilation for OpenMPC

Divides Parallel
Regions at
synchronization
points

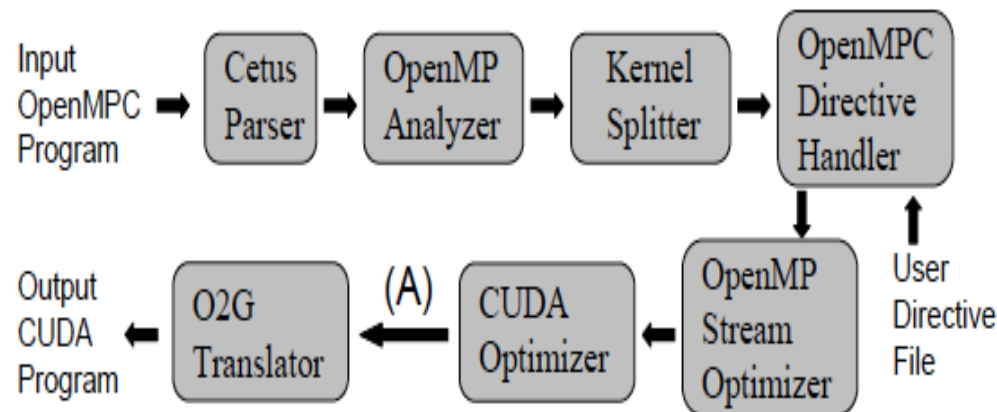


Fig. 3. Overall Compilation Flow. When the compilation system is used for automatic tuning, additional passes are invoked between *CUDA Optimizer* and *O2G Translator*, marked as (A) in the figure (See Figure 4)

Compilation for OpenMPC

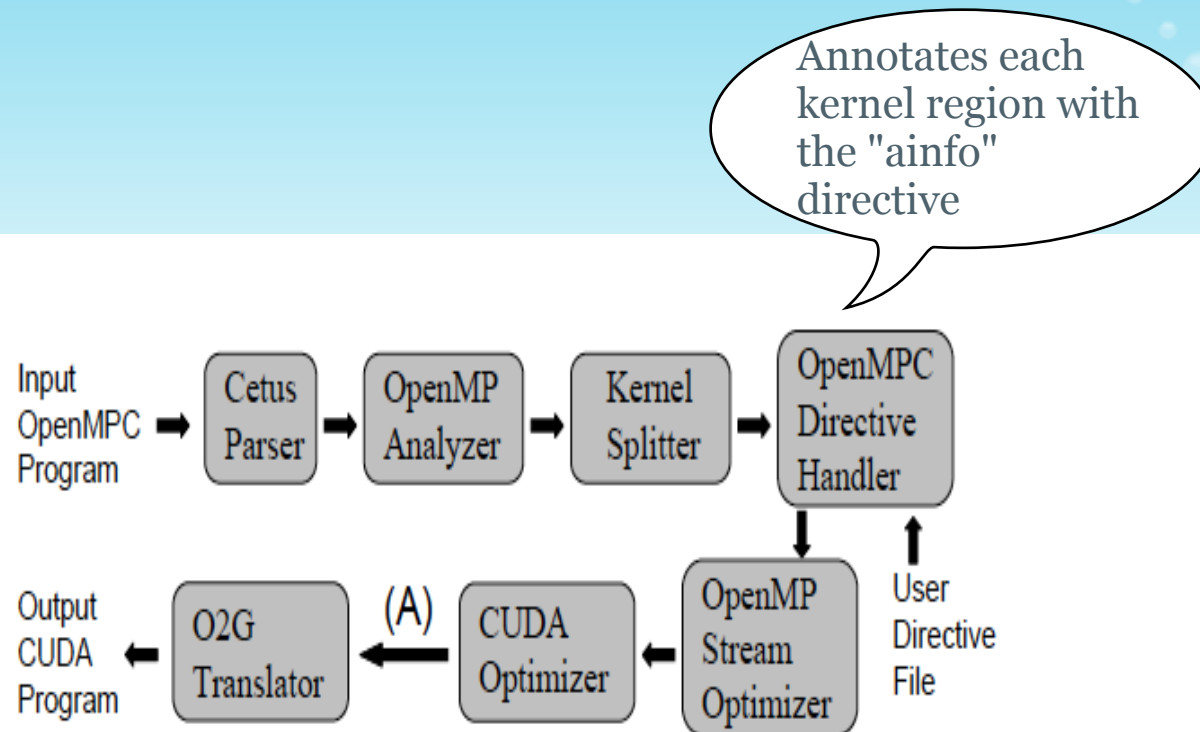


Fig. 3. Overall Compilation Flow. When the compilation system is used for automatic tuning, additional passes are invoked between *CUDA Optimizer* and *O2G Translator*, marked as (A) in the figure (See Figure 4)

Compilation for OpenMPC

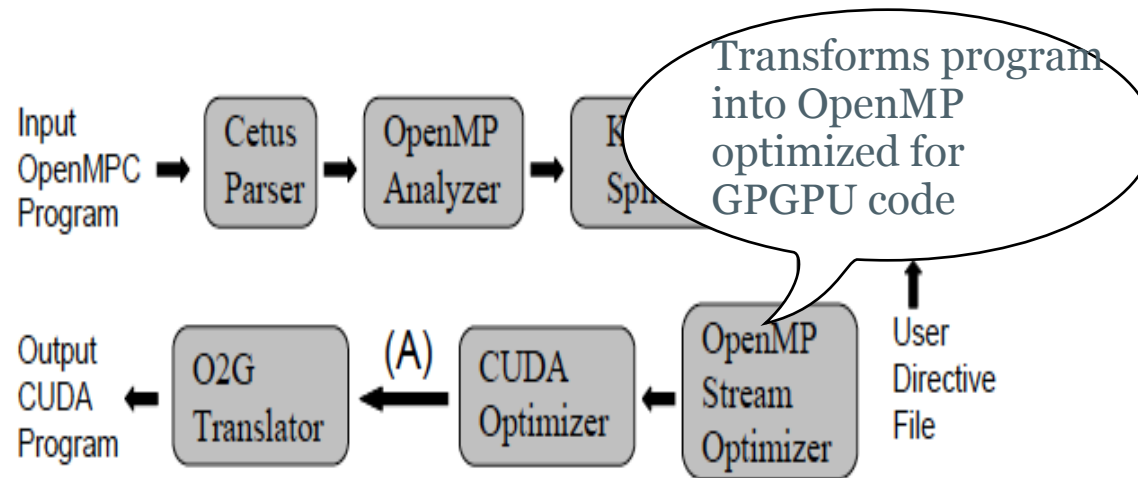


Fig. 3. Overall Compilation Flow. When the compilation system is used for automatic tuning, additional passes are invoked between *CUDA Optimizer* and *O2G Translator*, marked as (A) in the figure (See Figure 4)

Compilation for OpenMPC

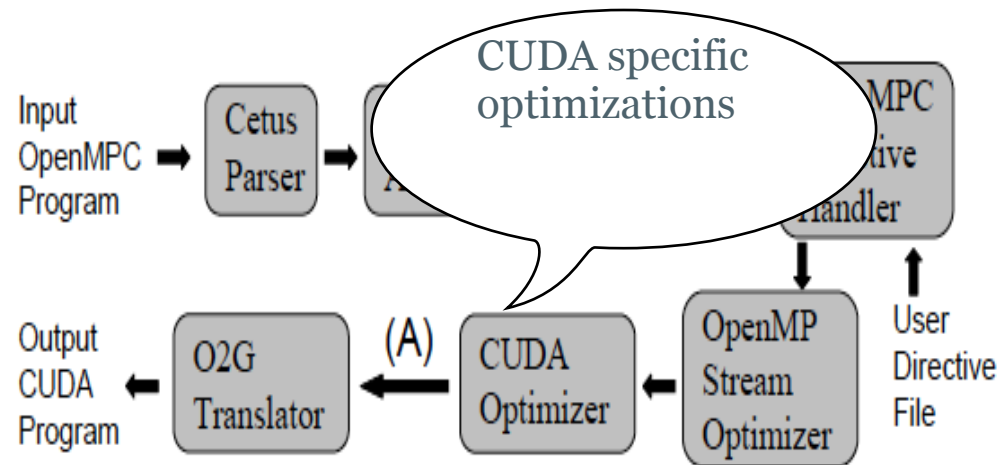


Fig. 3. Overall Compilation Flow. When the compilation system is used for automatic tuning, additional passes are invoked between *CUDA Optimizer* and *O2G Translator*, marked as (A) in the figure (See Figure 4)

Compilation for OpenMPC

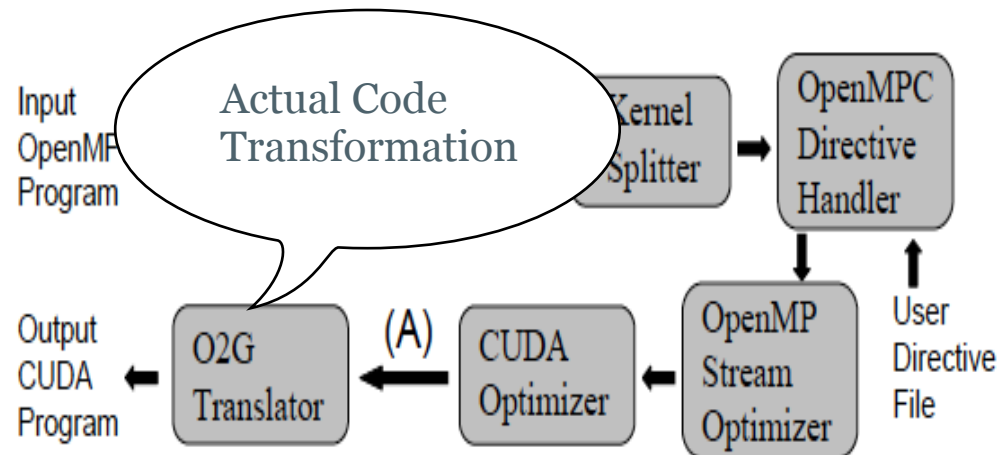


Fig. 3. Overall Compilation Flow. When the compilation system is used for automatic tuning, additional passes are invoked between *CUDA Optimizer* and *O2G Translator*, marked as (A) in the figure (See Figure 4)

Compiler Support For Tunning

- Drawback:
 - Complete optimization search space consist of all combinations of directives and environment variables.
- Solution:
 - Search Space Pruning
 - Removes optimizations for which conditions are not satisfied in the code.
 - Suggests caching strategies for each data type.

Prototyping Tuning System

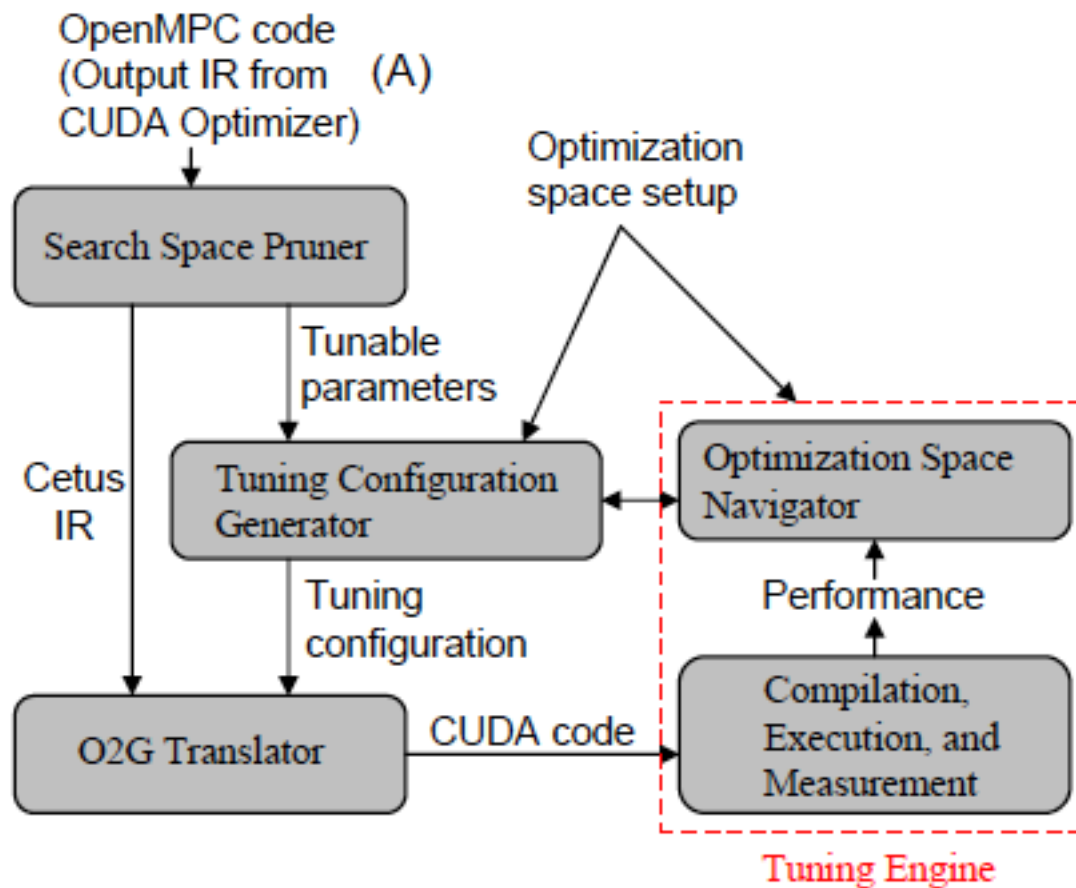


Fig. 4. Overall Tuning Framework. In the figure, input OpenMPC code is an output IR from CUDA Optimizer in the compilation system (See Figure 3)

Prototyping Tuning System

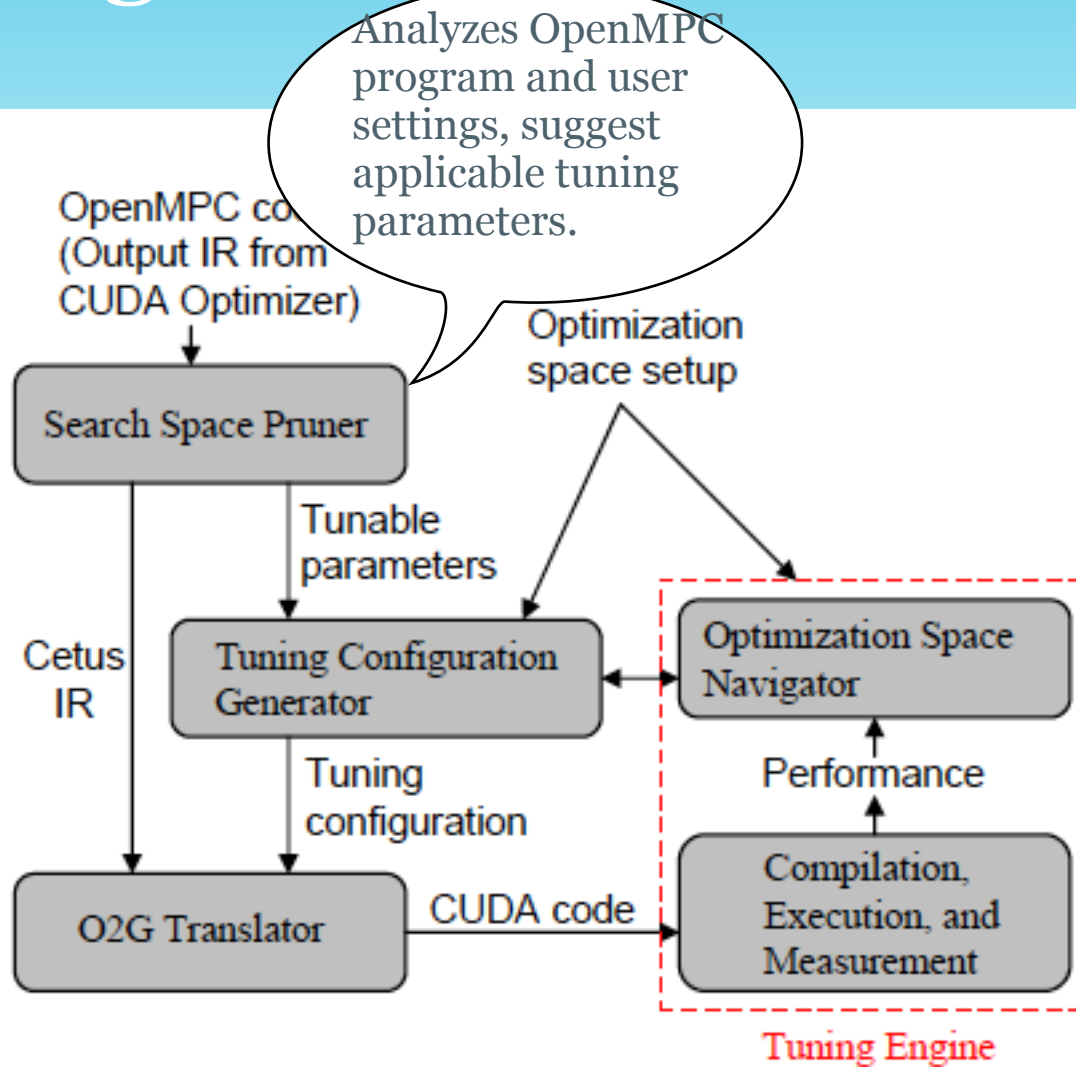


Fig. 4. Overall Tuning Framework. In the figure, input OpenMPC code is an output IR from CUDA Optimizer in the compilation system (See Figure 3)

Prototyping Tuning System

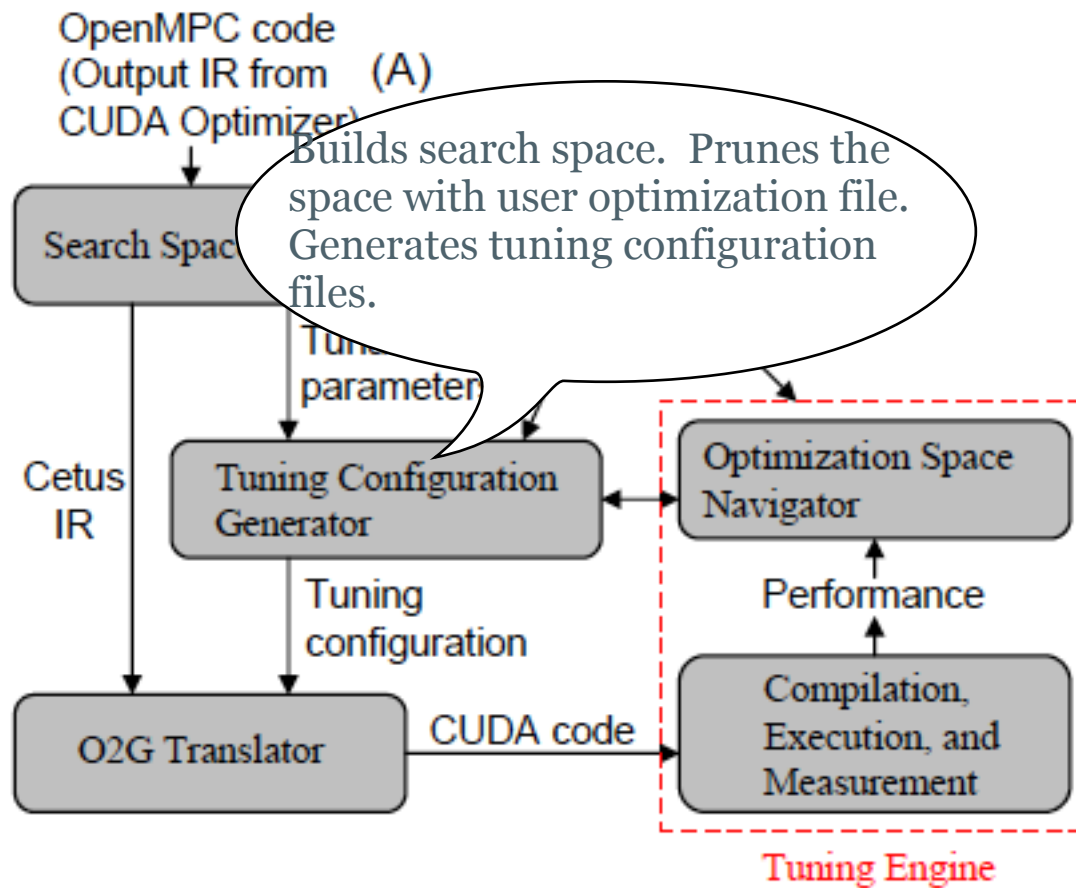


Fig. 4. Overall Tuning Framework. In the figure, input OpenMPC code is an output IR from CUDA Optimizer in the compilation system (See Figure 3)

Prototyping Tuning System

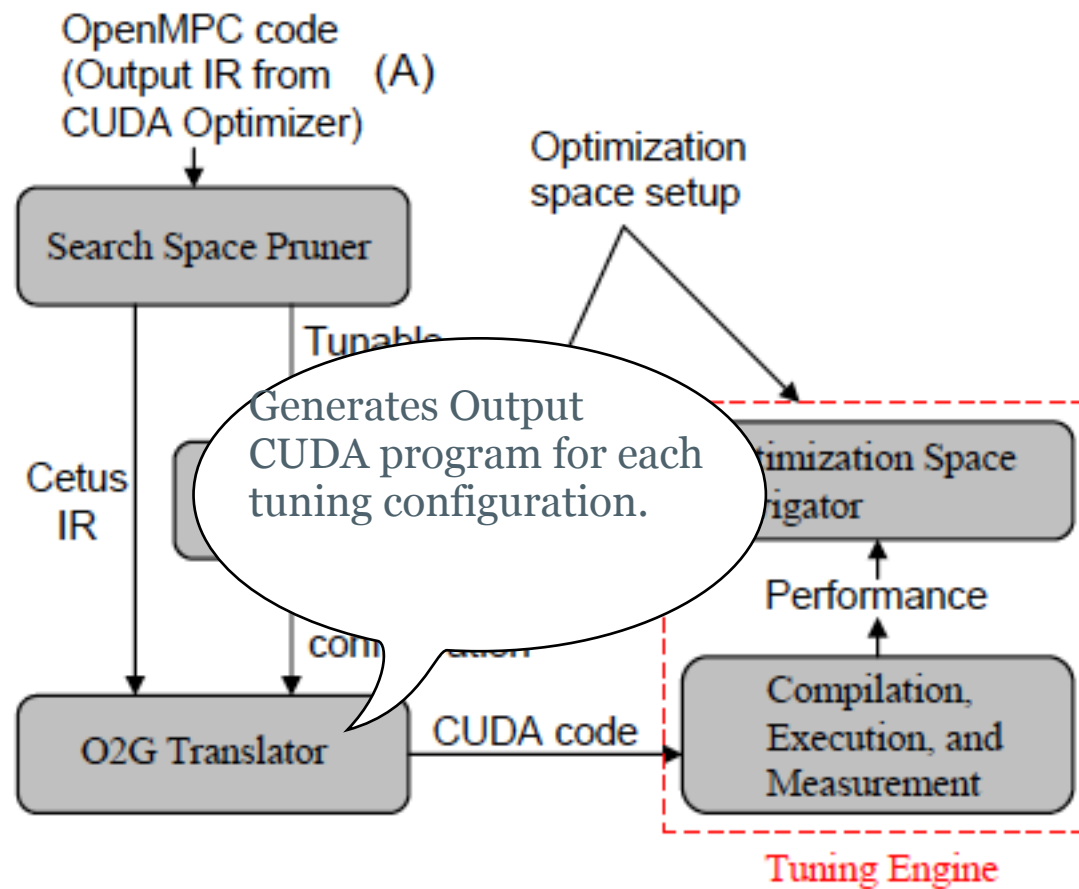
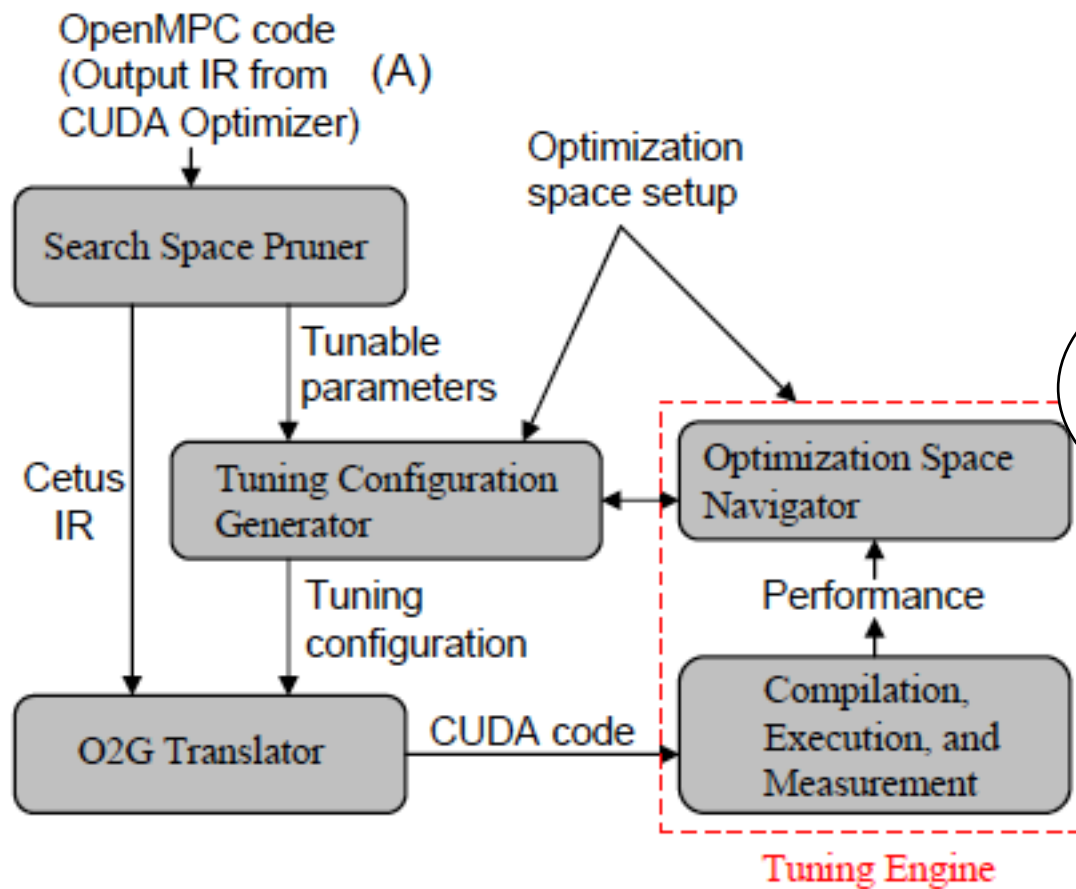


Fig. 4. Overall Tuning Framework. In the figure, input OpenMPC code is an output IR from CUDA Optimizer in the compilation system (See Figure 3)

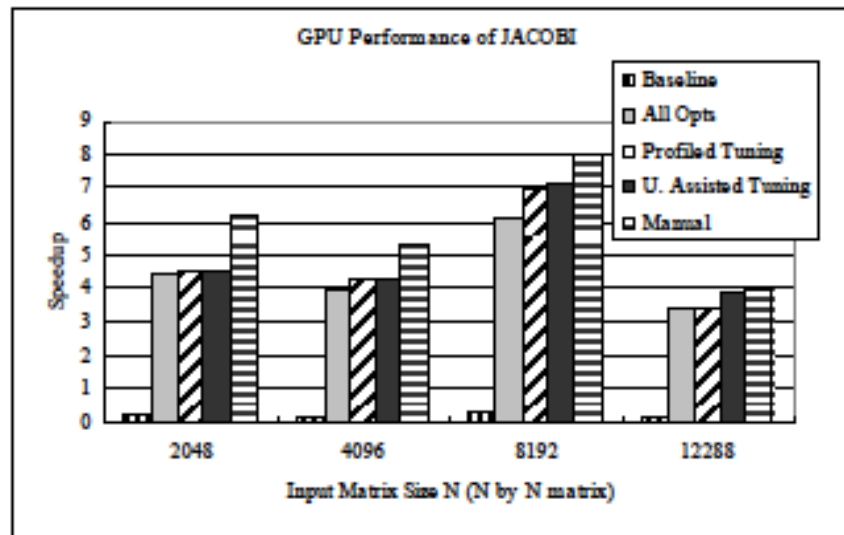
Prototyping Tuning System



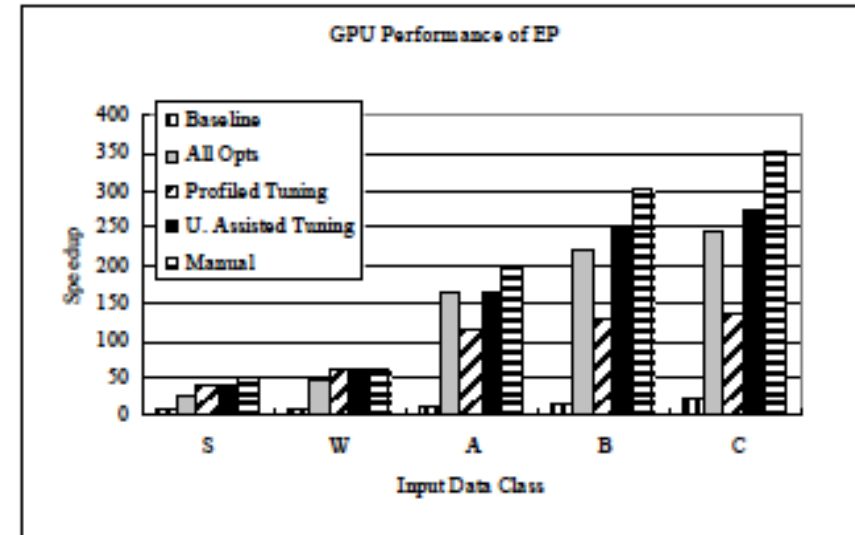
Creates executables and measures performance. Decides direction of the next search.

Fig. 4. Overall Tuning Framework. In the figure, input OpenMPC code is an output IR from CUDA Optimizer in the compilation system (See Figure 3)

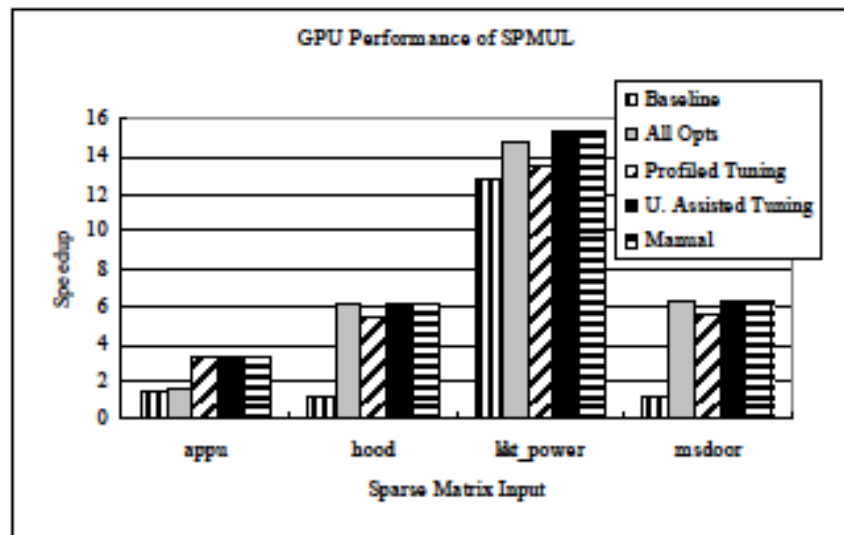
Evaluation



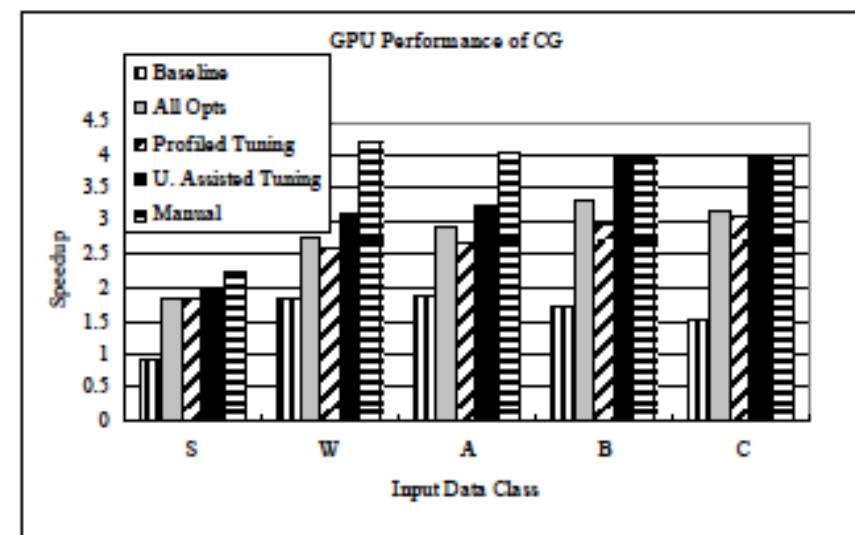
(a) JACOBI Kernel



(b) NAS Parallel Benchmark EP

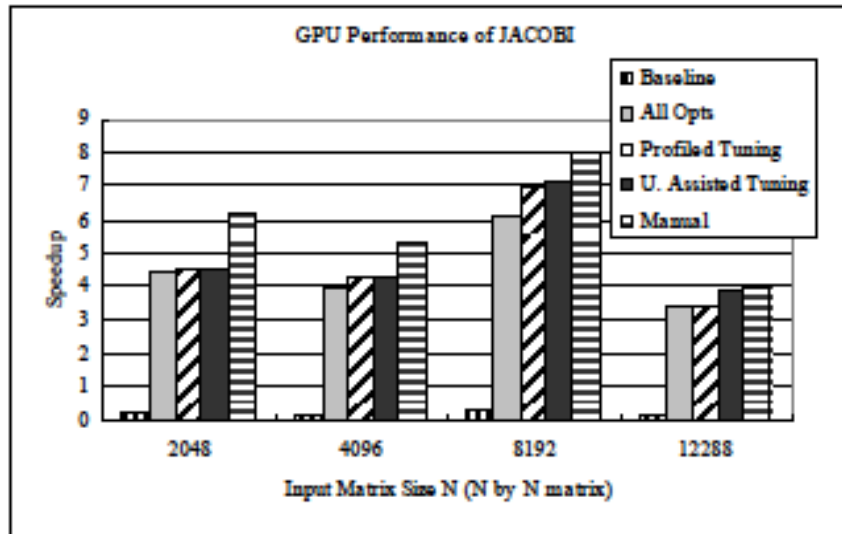


(c) SPMUL Kernel

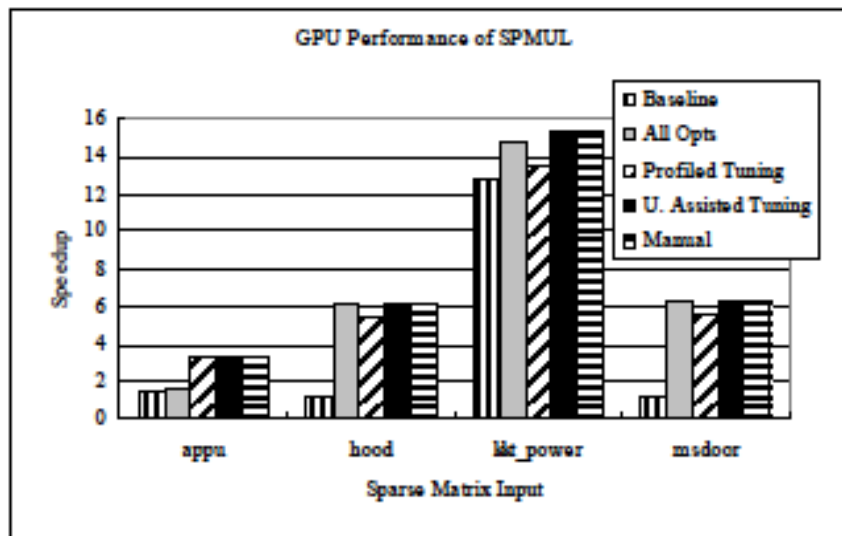


(d) NAS Parallel Benchmark CG

Evaluation



(a) JACOBI Kernel



(c) SPMUL Kernel

Baseline: Translated by system, with no optimizations.

All opt: All safe optimizations have been applied.

Manual: Baseline output optimized manually

Profiled Tuning: Training data set used to choose tuning parameters.

User Assisted Tuning: After a profiled based tuning, user confirms applicability of optimizations.

(d) NAS Parallel Benchmark CG

References:

- [1]S.Lee, R.Eigenmann."OpenMPC: Extended OpenMP Programming and Tuning for GPUs". IEEE 2010.
- [2]S.Lee, S.J.Min, R.Eigenmann. "OpenMP to GPUGPU: A Compiler Framework for Automatic Translation and Optimizations". ACM 2009.



THANKS!