Domain-Specific Languages for Stencil Computations

Azamat Mametjanov
Boyana Norris

Mathematics and Computer Science Division
Argonne National Laboratory
Motivation

- Finite-difference stencils are very common in numerical modeling. They exhibit high degree of data parallelism and regular structure. However, their memory requirements hinder the performance.

- Our approach consists of
  - Exploitation of a stencil’s data access pattern
  - Automatic conversion of C loops to CUDA C host+kernel code
  - Automatic tuning of CUDA C performance parameters
  - Raising the programming model to domain abstractions
Outline

- Introduction
- **Stencil data structures**
- Transformation and tuning framework of Orio
- Our approach
- Results
- DSLs for stencils
Stencils

- Sets of neighboring discrete points in a structured grid
- Stencil pattern determines the interaction among points
  - Domain dimension: 1D, 2D, 3D
  - Stencil shape: star, box
  - Stencil width: distance from stencil center
  - Boundary condition: Dirichlet, periodic etc.
Grid, adjacency matrix and its compression

![Grid, Adjacency Matrix and Its Compression](image-url)
Outline

- Introduction
- Stencil data structures
- Transformation and tuning framework of Orio
- Overview of the approach
- Results
- DSLs for stencils
Method: Code Transformation

- **Motivation**
  - Compilation: HL source code into LL portable executable code
  - Optimization: performance, energy
  - Refactoring: resiliency, maintainability, readability

- **Workflow**
  - Parse: any structured source text into abstract syntax tree
  - Analyze: common intermediate representation
  - Transform: compositions of reusable transforms
  - Generate: any structured target text

- **Challenges**
  - Create source and target domains
  - Create analysis and transformation rules
Method: Code Tuning

- **Motivation**
  - Deep component stacks
  - Each component is adjustable

- **Workflow**
  - System model: pre-specified, learned
  - Application profile: memory-/compute-bound
  - Configure: create a valid configuration of parameters
  - Select: the best performing parameter configuration

- **Challenges**
  - Auto-profile
  - Auto-modify
  - Search
  - Whole-app autotuning
Orio autotuning framework

- Code with DSL Annotations
- DSL Parser
- Sequence of (Nested) Annotated Regions
- Tuning Specification
- Code Transformations
- Code Generator
- Transfomed Code
- Empirical Performance Evaluation
- Search Engine
- best performing version
- Optimized Code
- CUDA
- Fortran
- C
Outline

- Introduction
- Stencil data structures
- Transformation and tuning framework of Orio
- **Overview of the approach**
- Results
- DSLs for stencils
Begin with reference C code

```c
for(i=0; i<=nrows-1; i++) {
    for(j=0; j<=ndiags-1; j++){
        col = i+offsets[j];
        if(col>=0&&col<nrows)
            y[i] += A[i+j*nrows] * x[col];
    }
}
```
Add a DSL annotation

/*@ begin Loop(...

for(i=0; i<=nrows-1; i++) {
  for(j=0; j<=ndiags-1; j++){
    col = i+offsets[j];
    if(col>=0&&col<nrows)
      y[i] += A[i+j*nrows] * x[col];
  }
}
) @*/

for ...

/*@ end @*/
Specify performance parameters (optional)

/*@ begin Loop(transform CUDA(  
  threadCount=TC,  
  blockCount=BC,  
  streamCount=SC,  
  preferL1Size=PL,  
  unrollInner=UIF, ...  
))  
for ...  
) */
for ...
/*@ end @*/
Specify parameter search ranges

/*@ begin PerfTuning(
  def performance_params{
    param TC[] = range(32,1025,32);
    param BC[] = range(14,113,14);
    param SC[] = range(1,17);
    param PL[] = [16,48];
    param UIF[] = range(1,8); ...
  }
) @*/
/*@ begin Loop(transform CUDA( ...
/*@ end */
Define empirical experiment inputs

/*@ begin PerfTuning(
def input_params {
    param M[] = [16,32,64,128,256]; ...
}
def input_vars {
    decl static double A[M*N*P*NOS*DOF] = random;
    decl static double x[M*N*P*DOF]    = random;
    decl static double y[M*N*P*DOF]    = 0; ...
}
...
) @*/
Define build and search parameters

/*@ begin PerfTuning(
  def build {
    arg build_command = 'nvcc -arch=sm_20 @CFLAGS';
  }
  def performance_counter {
    arg repetitions = 10;
  }
  ...
) @*/
Launch

./orcuda matVec3D.c

... 

Search_Space = 1.024e+04
Number_of_Parameters = 05
Numeric_Parameters = 05
Binary_Parameters = 00
['TC', 'BC', 'UIF', 'PL', 'CFLAGS']
[[32, 64, 96, 128, 160, 192, 224, 256, 288, 320, 352, 384, 416, 448, 480, 512, 544, 576, 608, 640, 672, 704, 736, 768, 800, 832, 864, 896, 928, 960, 992, 1024], [14, 28, 42, 56, 70, 84, 98, 112], [1, 2, 3, 4, 5], [16, 48], [' ', '-O1', '-O2', '-O3']]
Outline

- Introduction
- Stencil data structures
- Transformation and tuning framework of Orio
- Overview of the approach
- Results
- DSLs for stencils
Reduction kernels

Intel Xeon (dual quad-core E5462 processors), 2.8GHz; GPU: NVIDIA Fermi C2070
Pointwise kernels

Intel Xeon (dual quad-core E5462 processors), 2.8GHz; GPU: NVIDIA Fermi C2070

Normalized Execution Time

- Orio
- CUSP
- cuBLAS
Example: Sparse matrix-vector product (5- and 7-point stencil) on a GPU

Intel Xeon (dual quad-core E5462 processors), 2.8GHz; GPU: NVIDIA Fermi C2070

![Graph showing execution time comparison between Orio and cuSPARSE (CSR) for different grid sizes. Lower is better.](image)
Application: Bratu solid fuel ignition problem

![Graph showing normalized time for different problem sizes and libraries (GPU-OrCuda, GPU-Cusp, MKL). The x-axis represents problem size (64x64x64, 75x75x75, 100x100x100, 128x128x128), and the y-axis represents normalized time. The graph indicates up to 1.5x improvement. Lower is better.](image-url)
Outline

- Introduction
- Stencil data structures
- Transformation and tuning framework of Orio
- Overview of the approach
- Results
- DSLs for stencils
DSL for Stencils

/*@ begin Loop(...

for i,j in [0:M]x[0:N] {
    F[i][j] = (A[i][j] +
               A[i-1][j]+A[i][j-1]
               A[i+1][j]+A[i][j+1])/5
}
) @*/

for ...

/*@ end @*/
Conclusion

- Workflow:
  - Functionality
  - Performance

- Application stack is complex
  - Dependency depth
  - Heterogeneity at each level

- Autotuning provides end-to-end integration
  - Hardware and components will continue to change
  - Application only needs to be written once
  - Programmability through portability

- DSLs improve
  - Performance portability
  - Parallelization
Thank you

http://tinyurl.com/OrioTool