Chapel With Polyhedral Transformation Using Autotuning

Tuowen Zhao and Mary Hall

The 3rd Annual Chapel Implementers and Users Workshop, 2016
Loop Transformation

• Manipulation of loop nest
  • Structure
  • Schedule

• Prior work: manually apply loop transformations in Chapel
  • I. J. Bertolacci et al. Parameterized diamond tiling for stencil computations with Chapel parallel iterators. ICS 2015
  • A. Sharma et al. Affine loop optimization based on modulo unrolling in Chapel. PGAS 2014

• We: Automatically applied loop transformations using recipes from script which enables integration with autotuning framework
Contribution

• Uses C code to capture sequential computation
• Generates Chapel programs by composing polyhedral transformations on the sequential computation and mapping from iteration spaces to Chapel domains and iterator
• Demonstrates with a simple example in Chapel the benefits of applying such transformations in conjunction with autotuning
Chapel Language

```chapel
proc mm(A:[]real,B:[]real,
an:int,ambn:int,bm:int){
  const D = {0..an-1, 0..bm-1};  // Domain
  var C : [D] real;              // Domain mapped array
  forall (i,j) in D do {        // Iterator
    C[i,j] = 0;
    for k in {0..ambn-1} do
      C[i, j] += A[i, k] * B[k,j];
  }
  return C;
}
```
Polyhedral Framework

• Iteration Spaces
  • A set of iteration vectors represented as integer tuples
  • Direct mapping from Chapel domain

• Transformation done by linear mapping
  • Affine loop bounds, conditional expressions, array subscripts
Dependence analysis

• Ensure validity of transformation and correctness of program
• Have to know the order of references to each array elements
• Cannot be applied to Chapel iterator without programmer intervention or runtime information
CHiLL

• Composable High-Level Loop transformation framework

• A polyhedral transformation and code generation framework
  • Relies on autotuning to generate highly-tuned implementations for a specific target architecture
  • Uses a transformation recipe to express optimization strategy (recipe may be generated by a compiler)
Architecture Overview

- Transformation Recipes
  - CHiLL
    - Generated C code
      - Translator
        - Chapel Code
          - Execution Environment
          - Final Optimized Code
Experiment – matrix multiply

• Input in C
  for(i = 0; i < an; i++)
  for(j = 0; j < bm; j++)
  {
    C[i][j]=0.0f;
    for(n = 0; n < ambn; n++)
      C[i][j] += A[i][n] * B[n][j];
  }

• Tile sizes {8; 16; 32; 64; 128; 256}
• Distribution of the initialization code
• Tile sizes
  • Chapel’s configuration variable
  • Literal constant
• Intel Haswell i7-4790K
• 16GB DDR3 RAM
Result

![Graph showing speedup for Chapel Config, Chapel Config + Distribute, Chapel Const, and Chapel Const + Distribute. The x-axis represents tile size (8, 16, 32, 64, 128, 256), and the y-axis represents speedup ranging from 0 to 5. The graph compares the performance of different configurations across various tile sizes.]
Stencil Computations

- Operations on structured grids
- MiniGMG
  - Geometric multigrid benchmark
  - Uses stencil computations extensively especially in smooth and residual operators
- CHiLL on MiniGMG
Stencil Optimizations

- Communication avoiding optimizations
  - Wavefront (loop fusing)
  - Deeper ghost zones with redundant computation
User-defined library

• StencilDist library

• Problems
  • Can’t guarantee correctness (dependence)
  • Handwrite optimized code
  • Generality concern
Multi-locale Stencil
Multi-locale Stencil
Multi-locale Stencil

• Programmer writes simple serial code fragments
• Recipes provided by programmer or generated by autotuner
• Behind-the-scene generation of distributed computation and distributed data
• Produce fine-tuned code without programmer’s rewriting
Conclusion

• Integrating Chapel with CHiLL
  • Instantly enables a lot of different optimization techniques that can be composed in complex sequences
  • Autotuning can be used to find the best performing combination of transformations under target architecture

Future work

• Expanding the domain of autotuning by generating and tuning domain maps and iterators
• Relaxing the transformation requirements by generalize to non-affine loop bounds and subscripts that employ indirection through an index array
Questions?