Beyond Shared Memory Loop Parallelism in the Polyhedral Model

Tomofumi Yuki
Ph.D Dissertation
10/30 2012
The Problem

Evolution of Intel Platforms

Floating point peak performance [Mflop/s]
CPU frequency [MHz]

100,000
10,000
1,000
100
10

work required

free speedup

1993 1995 1997 1999 2001 2003 2005 2007

Year

data: www.sandpile.org

Figure from www.spiral.net/problem.html
Parallel Processing

- A small niche in the past, hot topic today
- **Ultimate Solution: Automatic Parallelization**
  - Extremely difficult problem
  - After decades of research, limited success
- **Other solutions: Programming Models**
  - Libraries (MPI, OpenMP, CnC, TBB, etc.)
  - Parallel languages (UPC, Chapel, X10, etc.)
  - Domain Specific Languages (stencils, etc.)
Contributions

MPI Code Generation

AlphaZ

MDE

40+ years of research linear algebra, ILP

CLooG, ISL, Omega, PLuTo

X10

Polyhedral X10

Colorado State University
Polyhedral State-of-the-art

- Tiling based parallelization
- Extensions to parameterized tile sizes
  - First step [Renganarayana2007]
  - Parallelization + Imperfectly nested loops [Hartono2010, Kim2010]
- PLuTo approach is now used by many people
  - Wave-front of tiles: better strategy than maximum parallelism [Bondhugula2008]
- Many advances in shared memory context
How far can shared memory go?

- The Memory Wall is still there
- Does it make sense for 1000 cores to share memory? [Berkley View, Shalf 07, Kumar 05]
  - **Power**
  - Coherency overhead
  - False sharing
  - Hierarchy?
  - Data volume (tera- peta-bytes)
Distributed Memory Parallelization

- Problems implicitly handled by the shared memory now need explicit treatment

- Communication
  - Which processors need to send/receive?
  - Which data to send/receive?
  - How to manage communication buffers?

- Data partitioning
  - How do you allocate memory across nodes?
MPI Code Generator

- Distributed Memory Parallelization
  - Tiling based
  - Parameterized tile sizes
  - C+MPI implementation
- Uniform dependences as key enabler
  - Many affine dependences can be uniformized
- Shared memory performance carried over to distributed memory
  - Scales as well as PLuTo but to multiple nodes
Related Work (Polyhedral)

- Polyhedral Approaches
  - Initial idea [Amarasinghe1993]
  - Analysis for fixed sized tiling [Cläßen2006]
  - Further optimization [Bondhugula2011]
- “Brute Force” polyhedral analysis for handling communication
  - No hope of handling parametric tile size
  - Can handle arbitrarily affine programs
Outline

- Introduction
- “Uniform-ness” of Affine Programs
  - Uniformization
  - Uniform-ness of PolyBench
- MPI Code Generation
  - Tiling
  - Uniform-ness simplifies everything
  - Comparison against PLuTo with PolyBench
- Conclusions and Future Work
Affine vs Uniform

Affine Dependences: \( f = Ax + b \)
- Examples
  - \((i,j \rightarrow j,i)\)
  - \((i,j \rightarrow i,i)\)
  - \((i \rightarrow 0)\)

Uniform Dependences: \( f = lx + b \)
- Examples
  - \((i,j \rightarrow i-1,j)\)
  - \((i \rightarrow i-1)\)
Uniformization

- (i->0)

- (i->i-1)
Uniformization

- Uniformization is a classic technique
  - “solved” in the 1980’s
  - has been “forgotten” in the multi-core era
- Any affine dependence can be uniformized
  - by adding a dimension [Roychowdhury1988]
- Nullspace pipelining
  - simple technique for uniformization
  - many dependences are uniformized
Uniformization and Tiling

- Uniformization does not influence tilability
PolyBench [Pouchet2010]

- Collection of 30 polyhedral kernels
  - Proposed by Pouchet as a benchmark for polyhedral compilation
  - Goal: Small enough benchmark so that individual results are reported; no averages

- Kernels from:
  - data mining
  - linear algebra kernels, solvers
  - dynamic programming
  - stencil computations
Uniform-ness of PolyBench

- 5 of them are “incorrect” and are excluded

<table>
<thead>
<tr>
<th>Stage</th>
<th>Uniform at Start</th>
<th>After Embedding</th>
<th>After Pipelining</th>
<th>After Phase Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Fully Uniform Programs</td>
<td>8/25 (32%)</td>
<td>13/25 (52%)</td>
<td>21/25 (84%)</td>
<td>24/25 (96%)</td>
</tr>
</tbody>
</table>

- Embedding: Match dimensions of statements
- Phase Detection: Separate program into phases
  - Output of a phase is used as inputs to the other
Outline

- Introduction
- Uniform-ness of Affine Programs
  - Uniformization
  - Uniform-ness of PolyBench
- MPI Code Generation
  - Tiling
  - Uniform-ness simplifies everything
  - Comparison against PLuTo with PolyBench
- Conclusions and Future Work
Basic Strategy: Tiling

- We focus on tilable programs
Dependences in Tilable Space

- All in the non-positive direction
Wave-front Parallelization

- All tiles with the same color can run in parallel
Assumptions

- Uniform in at least **one** of the dimensions
- The uniform dimension is made outermost
  - Tilable space is fully permutable
- One-dimensional processor allocation
- Large enough tile sizes
  - Dependences do not span multiple tiles
- Then, **communication is extremely simplified**
Processor Allocation

- Outermost tile loop is distributed
Values to be Communicated

- Faces of the tiles (may be thicker than 1)
Naïve Placement of Send and Receive Codes

- Receiver is the consumer tile of the values
Problems in Naïve Placement

- Receiver is in the **next** wave-front time.
Problems in Naïve Placement

- Receiver is in the **next** wave-front time
- Number of communications “in-flight” = amount of parallelism
- **MPI_Send** will **deadlock**
  - May not return control if system buffer is full
- Asynchronous communication is required
  - Must manage your own buffer
  - required buffer = amount of parallelism
    - i.e., number of *virtual* processors
Proposed Placement of Send and Receive codes

- Receiver is one tile below the consumer
Naïve Placement:
- Receive -> Compute -> Send

Proposed Placement:
- Issue asynchronous receive (MPI_Irecv)
- Compute
- Issue asynchronous send
- Wait for values to arrive

Overlap of computation and communication
- Only two buffers per physical processor
Evaluation

- Compare performance with PLuTo
  - Shared memory version with same strategy
- Cray: 24 cores per node, up to 96 cores
- Goal: Similar scaling as PLuTo
- Tile sizes are searched with educated guesses
- PolyBench
  - 7 are too small
  - 3 cannot be tiled or have limited parallelism
  - 9 cannot be used due to PLuTo/PolyBench issue
Performance Results

Summary of AlphaZ Performance Comparison with PLuTo

- Linear extrapolation from speed up of 24 cores
- Broadcast cost at most 2.5 seconds
AlphaZ System

- System for polyhedral design space exploration
- Key features not explored by other tools:
  - Memory allocation
  - Reductions
- Case studies to illustrate the importance of unexplored design space [LCPC2012]
- Polyhedral Equational Model [WOLFHPC2012]
- MDE applied to compilers [MODELS2011]
Polyhedral X10 [PPoPP2013?]

- Work with Vijay Saraswat and Paul Feautrier
- Extension of array data flow analysis to X10
  - supports finish/async but not clocks
- finish/async can express more than doall
  - Focus of polyhedral model so far: doall
- Dataflow result is used to detect races
  - With polyhedral precision, we can guarantee program regions to be race-free
Conclusions

- Polyhedral Compilation has lots of potential
  - Memory/reductions are not explored
  - Successes in automatic parallelization
  - Race-free guarantee
- Handling arbitrary affine may be an overkill
  - Uniformization makes a lot of sense
  - Distributed memory parallelization made easy
  - Can handle most of PolyBench
Future Work

- Many direct extensions
  - Hybrid MPI+OpenMP with multi-level tiling
  - Partial uniformization to satisfy pre-condition
  - Handling clocks in Polyhedral X10

- More broad applications of polyhedral model
  - Approximations
  - Larger granularity: blocks of computations instead of statements
  - Abstract interpretations [Alias2010]
Acknowledgements

- Advisor: Sanjay Rajopadhye
- Committee members:
  - Wim Böhm
  - Michelle Strout
  - Edwin Chong
- Unofficial Co-advisor: Steven Derrien
- Members of
  - Mélange, HPCM, CAIRN
  - Dave Wonnacott, Haverford students
Backup Slides
Uniformization and Tiling

- Tilability is preserved
D-Tiling Review [Kim2011]

- Parametric tiling for shared memory
- Uses non-polyhedral skewing of tiles
  - Required for wave-front execution of tiles
- The key equation:
  - \[ \text{time} = \sum_{i=1}^{d} \frac{t_i}{t_s_i} \]
  - where
    - d: number of tiled dimensions
    - ti: tile origins
    - ts: tile sizes
D-Tiling Review cont.

- The equation enables skewing of tiles
  - If one of time or tile origins are unknown, can be computed from the others

- Generated Code: (tix is d-1th tile origin)

```c
for (time=start:end)
  for (til=tilLB:tilUB)
    ...
    for (tix=tixLB:tixUB) {
      tid = f(time, til, ..., tix);
      //compute tile til,ti2,...,tix,tid
    }
```
Placement of Receive Code using D-Tiling

- Slight modification to the use of the equation
- Visit tiles in the next wave-front time

```c
for (time=start:end)
    for (til=tilLB:tilUB)
        ...
        for (tix=tixLB:tixUB) {
            tidNext = f(time+1, til, ..., tix);
            //receive and unpack buffer for
            //tile til,ti2,...,tix,tidNext
        }
```
Proposed Placement of Send and Receive codes

- Receiver is one tile below the consumer
Extensions to Schedule Independent Mapping

- Schedule Independent Mapping [Strout1998]
  - Universal Occupancy Vectors (UOVs)
  - Legal storage mapping for any legal execution
  - Uniform dependence programs only
- Universality of UOVs can be restricted
  - e.g., to tiled execution
- For tiled execution, shortest UOV can be found without any search
LU Decomposition

![Graph showing speed up with respect to PLuTo with 1 core for different numbers of cores. The graph compares PLuTo, AlphaZ, and AlphaZ (No Bcast).]
seidel-2d

Number of Cores

Speed Up with respect to PLuTo with 1 core

- PLuTo
- AlphaZ
- AlphaZ (No Bcast)

Number of Cores
seidel-2d (no 8x8x8)

seidel-2d (without 8x8x8 tiles)

Number of Cores

Speed Up with respect to PLuTo with 1 core

- PLuTo
- AlphaZ
- AlphaZ (No Bcast)
jacobi-2d-imper

Number of Cores

Speed Up with respect to PLuTo with 1 core

- PLuTo
- AlphaZ
- AlphaZ (No Bcast)

Number of Cores
Related Work (Non-Polyhedral)

- Global communications [Li1990]
  - Translation from shared memory programs
  - Pattern matching for global communications
- Paradigm [Banerjee1995]
  - No loop transformations
  - Finds parallel loops and inserts necessary communications
- Tiling based [Goumas2006]
  - Perfectly nested uniform dependences
adi.c: Performance

- PLuTo does not scale because the outer loop is not tiled
UNAfold: Performance

Complexity reduction is empirically confirmed
Contributions

- The AlphaZ System
  - Polyhedral compiler with full control to the user
  - Equational view of the polyhedral model
- MPI Code Generator
  - The first code generator with parametric tiling
  - Double buffering
- Polyhedral X10
  - Extension to the polyhedral model
  - Race-free guarantee of X10 programs