An Optimizing Code Generator for a Class of Lattice-Boltzmann Computations

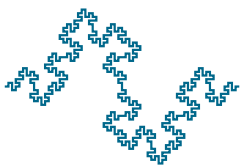
Irshad Pananilath, Aravind Acharya, Vinay Vasista, **Uday Bondhugula**

uday@csa.iisc.ernet.in

Dept of CSA

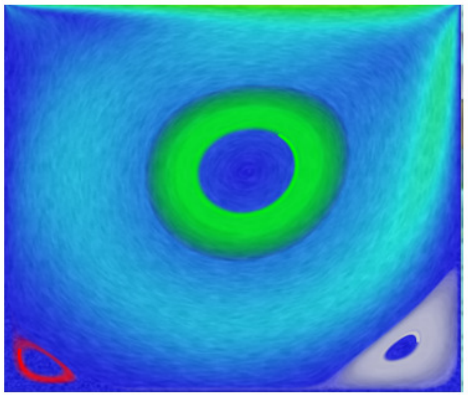
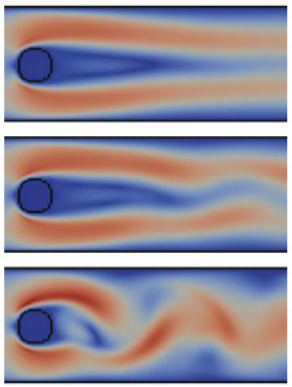
Indian Institute of Science

Bangalore 560012 India



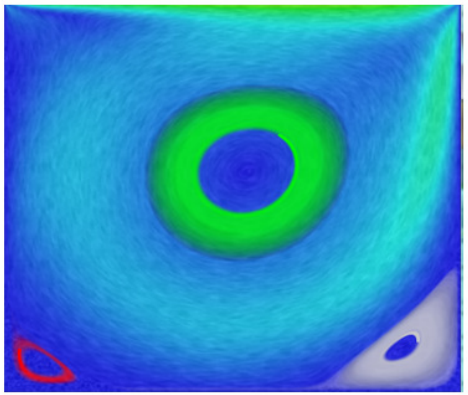
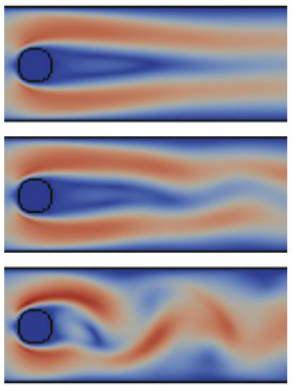
February 4, 2016

INTRODUCTION

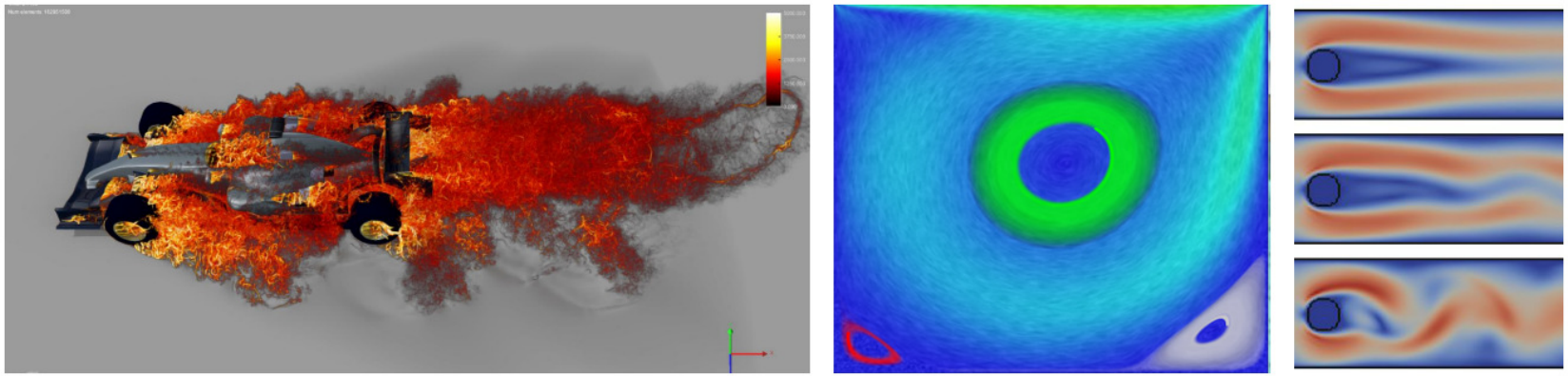
- Lattice–Boltzmann method (LBM) is used for simulation of complex fluid flows in Computational Fluid Dynamics (CFD)

INTRODUCTION

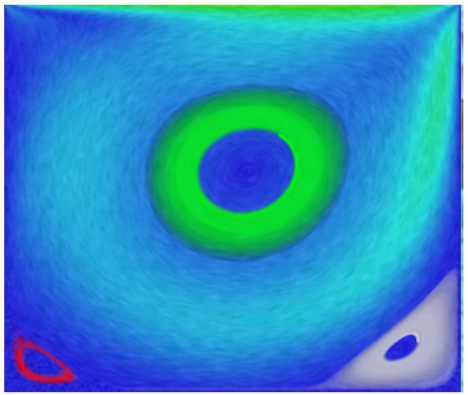
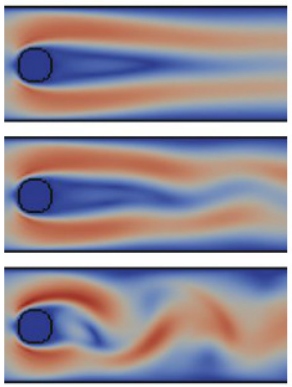
- Lattice–Boltzmann method (LBM) is used for simulation of complex fluid flows in Computational Fluid Dynamics (CFD)
- The simplicity of formulation and its versatility explain the rapid expansion of LBM to applications in complex and multiscale flows

INTRODUCTION



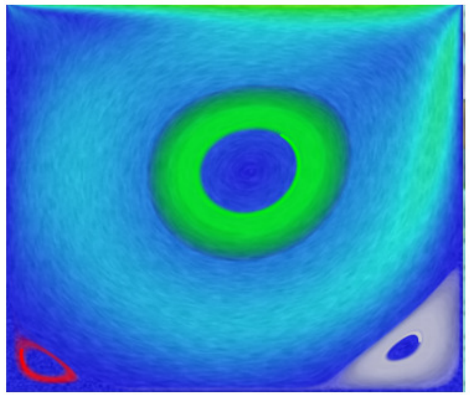
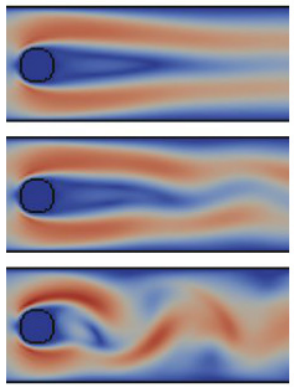
- Lattice–Boltzmann method (LBM) is used for simulation of complex fluid flows in Computational Fluid Dynamics (CFD)
- The simplicity of formulation and its versatility explain the rapid expansion of LBM to applications in complex and multiscale flows
- Particularly suited for parallel and high performance implementations

INTRODUCTION

- Lattice–Boltzmann method (LBM) is used for simulation of complex fluid flows in Computational Fluid Dynamics (CFD)
- The simplicity of formulation and its versatility explain the rapid expansion of LBM to applications in complex and multiscale flows
- Particularly suited for parallel and high performance implementations
- In spite of tremendous advances in its application, several fundamental opportunities for optimization remain

INTRODUCTION

- Lattice–Boltzmann method (LBM) is used for simulation of complex fluid flows in Computational Fluid Dynamics (CFD)
- The simplicity of formulation and its versatility explain the rapid expansion of LBM to applications in complex and multiscale flows
- Particularly suited for parallel and high performance implementations
- In spite of tremendous advances in its application, several fundamental opportunities for optimization remain
- We explore one such opportunity through this work

LATTICE–BOLTZMANN METHOD

- Fluid flows are modelled as hypothetical particles
  - moving in a lattice domain (discretized space)
  - with different lattice velocities (discretized momentum)
  - over different time steps (discretized time)

LATTICE–BOLTZMANN METHOD

- Fluid flows are modelled as hypothetical particles
  - moving in a lattice domain (discretized space)
  - with different lattice velocities (discretized momentum)
  - over different time steps (discretized time)
- Solves the discrete Boltzmann equation for the particle distribution function (a probability density function)

LBM - LATTICE ARRANGEMENTS

- Lattice arrangements are represented as *DmQn*
  - *m* is the space dimensionality of the lattice
  - *n* is the number of PDFs (or speeds) involved

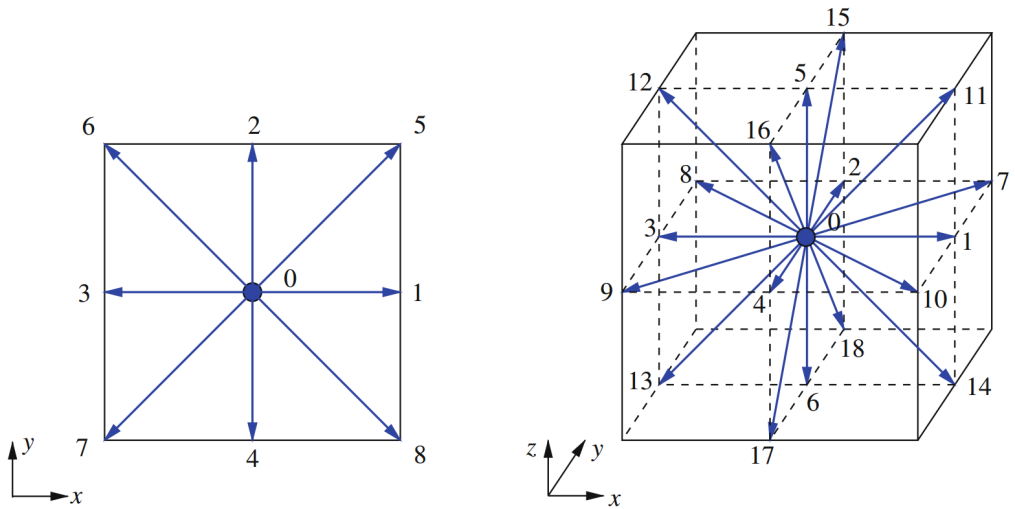


Figure: *D*2*Q*9 (left) & *D*3*Q*19 (right) lattice arrangements

LBM - LATTICE ARRANGEMENTS

- Lattice arrangements are represented as *DmQn*
  - *m* is the space dimensionality of the lattice
  - *n* is the number of PDFs (or speeds) involved
- Choice of lattice affects precision and duration of simulation

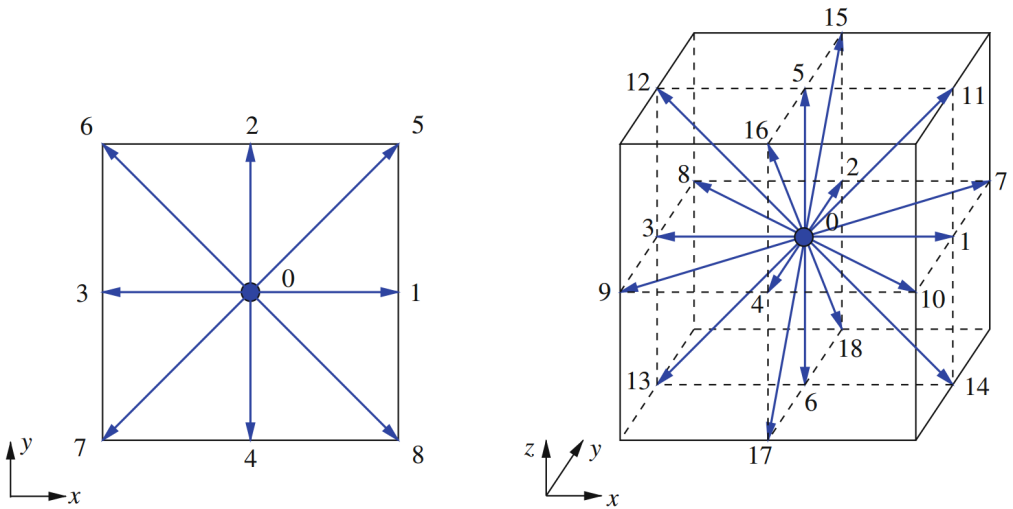


Figure: *D*2*Q*9 (left) & *D*3*Q*19 (right) lattice arrangements

LATTICE–BOLTZMANN METHOD

- The discretized form of Lattice–Boltzmann Equation forms the basis of all LBM models

$$f_i(\mathbf{x} + \mathbf{c_i}\Delta t, t + \Delta t) = f_i(\mathbf{x}, t) + \Omega_i(f_i(\mathbf{x}, t)),\ i = 1, \ldots n. \quad (1)$$

LATTICE–BOLTZMANN METHOD

- The discretized form of Lattice–Boltzmann Equation forms the basis of all LBM models

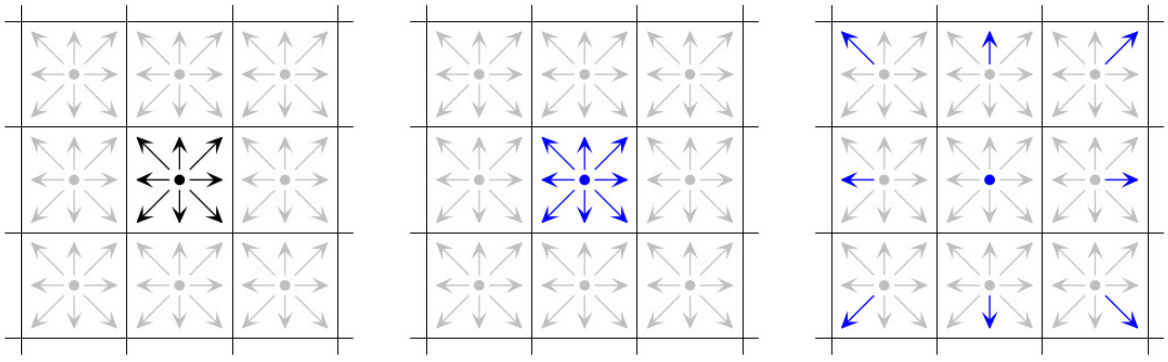
$$f_i(\mathbf{x} + \mathbf{c_i}\Delta t, t + \Delta t) = f_i(\mathbf{x}, t) + \Omega_i(f_i(\mathbf{x}, t)),\ i = 1, \ldots n. \quad (1)$$

- Eqn. 1 is solved in two steps, the **collision** step (Eqn. 2) & the **advection** step (Eqn. 3)

$$f_i^*(\mathbf{x}, t + \Delta t) = f_i(\mathbf{x}, t) + \Omega_i(f_i(\mathbf{x}, t)) \quad (2)$$

$$f_i(\mathbf{x} + \mathbf{c_i}\Delta t, t + \Delta t) = f_i^*(\mathbf{x}, t + \Delta t) \quad (3)$$

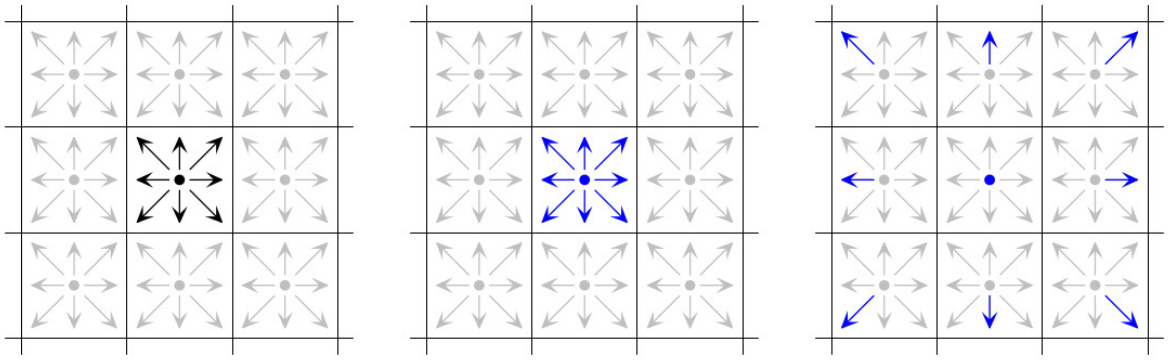
LBM - IMPLEMENTATION STRATEGIES



(a) Pre-collision (b) Post-collision (c) Post-propagation

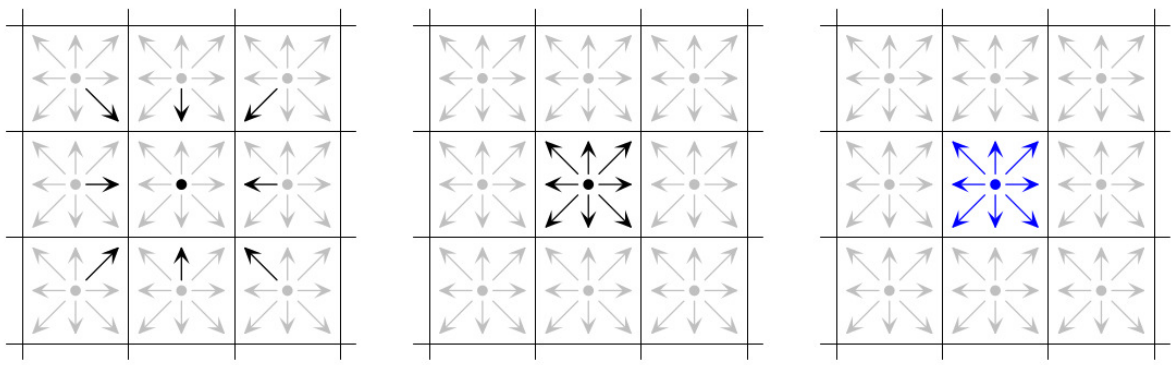
Figure: Push Scheme

LBM - IMPLEMENTATION STRATEGIES



(a) Pre-collision (b) Post-collision (c) Post-propagation

Figure: Push Scheme



(a) Initial state (b) Pre-collision (c) Post-collision

Figure: Pull Scheme

MOTIVATION

To build a comprehensive framework that can address shortcomings of previous works and automatically generate high performance code for LBM simulations

- Related work
  - Wilke et al. - *Euro-Par 2003* - 1D & 2D blocking
  - Nitsure et al. - *FIS 2006* - Cache oblivious algorithm for LBM
  - Nguyen et al. - *SC 2010* - 3.5D Blocking
  - Palabos - an open-source CFD solver based on the Lattice Boltzmann method (library-based)

Is it possible to optimize LBM using time tiling?

TILING (BLOCKING)

- Partition and execute iteration space in blocks

```
for(i=1; i<T; i++)
  for(j=1; j<N-1; j++)
    S(i,j)
```

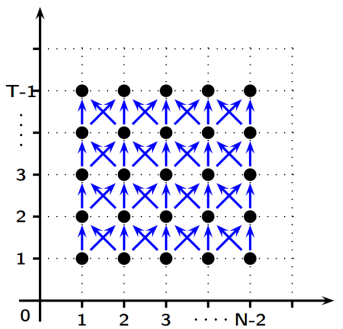


Figure: Iteration space

TILING (BLOCKING)

- Partition and execute iteration space in blocks
- Benefits - *cache locality* & *parallelism*

```
for(i=1; i<T; i++)
  for(j=1; j<N-1; j++)
    S(i,j)
```

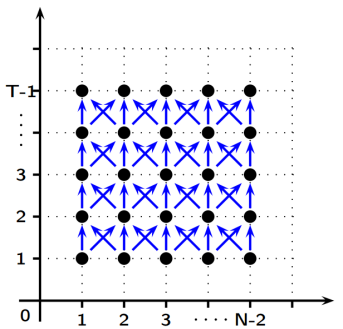


Figure: Iteration space

TILING (BLOCKING)

- Partition and execute iteration space in blocks
- Benefits - *cache locality* & *parallelism*
- Validity of tiling

```
for(i=1; i<T; i++)
  for(j=1; j<N-1; j++)
    S(i,j)
```

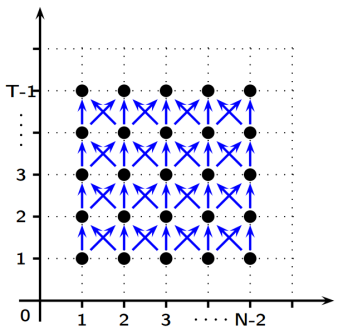


Figure: Iteration space

TILING (BLOCKING)

- Partition and execute iteration space in blocks
- Benefits - *cache locality* & *parallelism*
- Validity of tiling
  - Executed atomically

```
for(i=1; i<T; i++)
  for(j=1; j<N-1; j++)
    S(i,j)
```

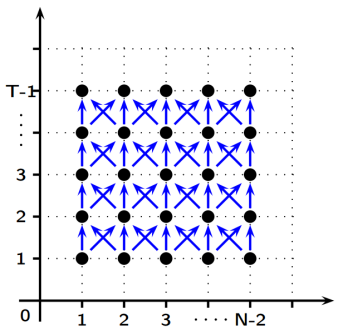


Figure: Iteration space

TILING (BLOCKING)

- Partition and execute iteration space in blocks
- Benefits - *cache locality* & *parallelism*
- Validity of tiling
  - Executed atomically
  - Total order possible on the set of all tiles

```
for(i=1; i<T; i++)
  for(j=1; j<N-1; j++)
    S(i,j)
```

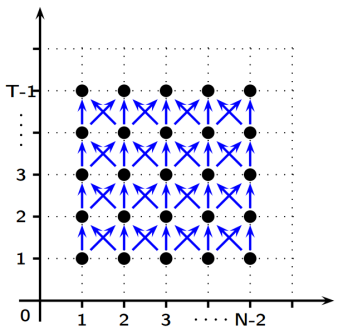


Figure: Iteration space

TILING (BLOCKING)

- Partition and execute iteration space in blocks
- Benefits - *cache locality* & *parallelism*
- Validity of tiling
  - Executed atomically
  - Total order possible on the set of all tiles
- Time tiling

```
for(i=1; i<T; i++)
  for(j=1; j<N-1; j++)
    S(i,j)
```

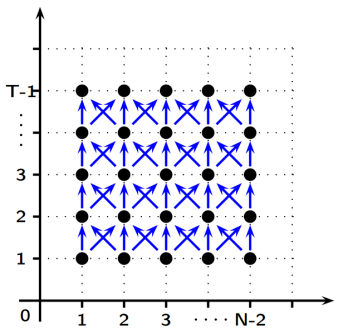


Figure: Iteration space

TILING (BLOCKING)

- Partition and execute iteration space in blocks
- Benefits - *cache locality* & *parallelism*
- Validity of tiling
  - Executed atomically
  - Total order possible on the set of all tiles
- Time tiling

```
for(i=1; i<T; i++)
  for(j=1; j<N-1; j++)
    S(i,j)
```

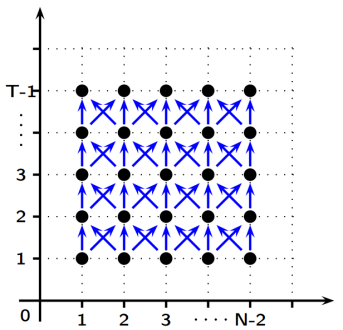


Figure: Iteration space

TILING (BLOCKING)

- Partition and execute iteration space in blocks
- Benefits - *cache locality* & *parallelism*
- Validity of tiling
  - Executed atomically
  - Total order possible on the set of all tiles
- Time tiling

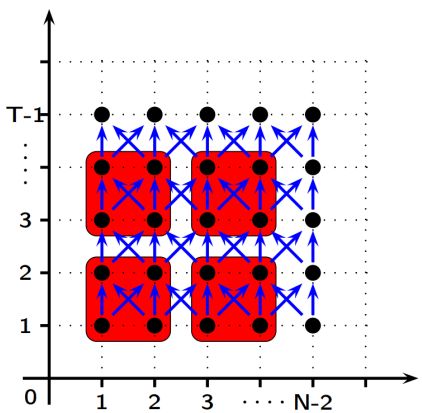


Figure: Invalid tiling

```
for(i=1; i<T; i++)
  for(j=1; j<N-1; j++)
    S(i,j)
```

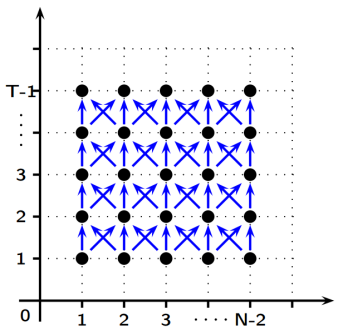


Figure: Iteration space

TILING (BLOCKING)

- Partition and execute iteration space in blocks
- Benefits - *cache locality* & *parallelism*
- Validity of tiling
  - Executed atomically
  - Total order possible on the set of all tiles
- Time tiling

```
for(i=1; i<T; i++)
  for(j=1; j<N-1; j++)
    S(i,j)
```

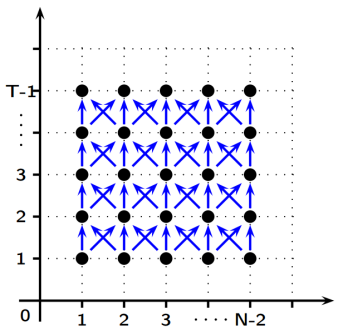


Figure: Iteration space

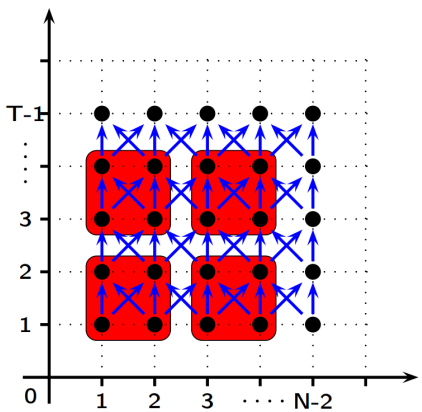


Figure: Invalid tiling

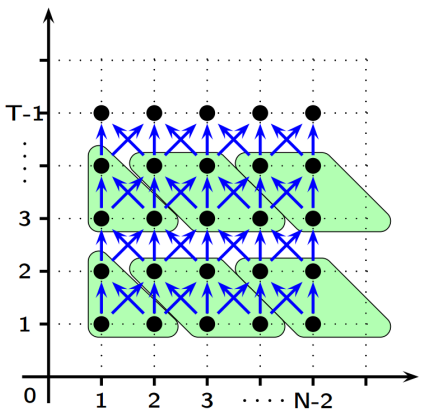


Figure: Valid tiling

CONCURRENT START AND DIAMOND TILING[1]

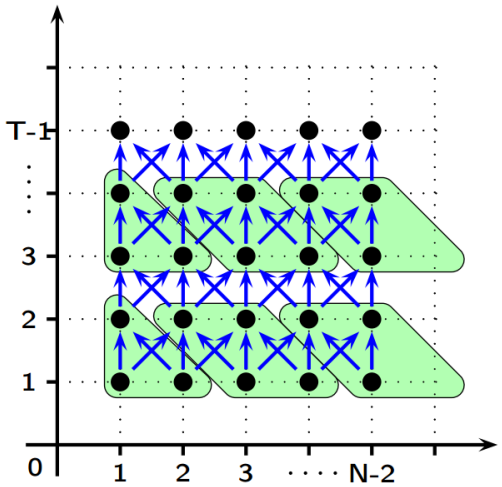


Figure: Parallelogram tiling

[1] Tiling Stencil Computations to Maximize Parallelism, *Vinayaka Bandishti, Irshad Pananilath, and Uday Bondhugula*, ACM/IEEE Supercomputing (SC) 2012

CONCURRENT START AND DIAMOND TILING[1]

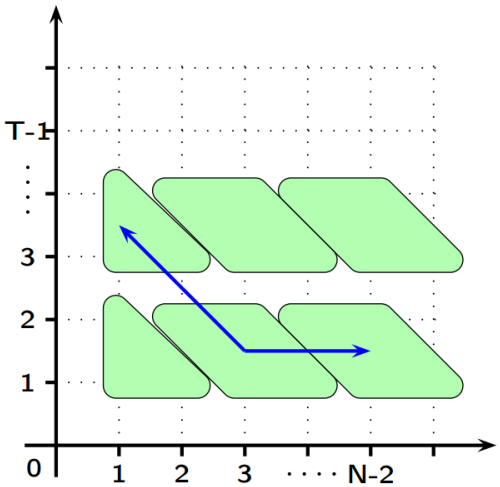


Figure: Pipelined start

[1] Tiling Stencil Computations to Maximize Parallelism, *Vinayaka Bandishti, Irshad Pananilath, and Uday Bondhugula*, ACM/IEEE Supercomputing (SC) 2012

CONCURRENT START AND DIAMOND TILING[1]

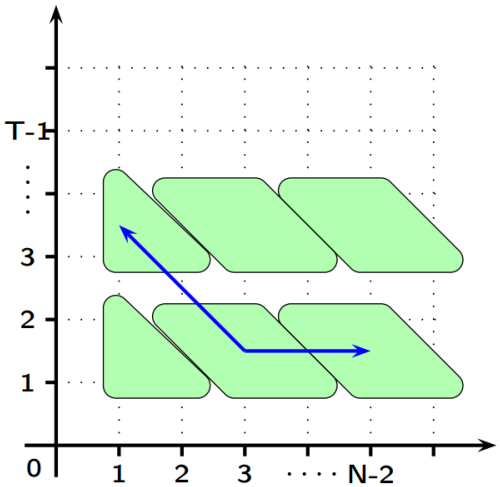


Figure: Pipelined start

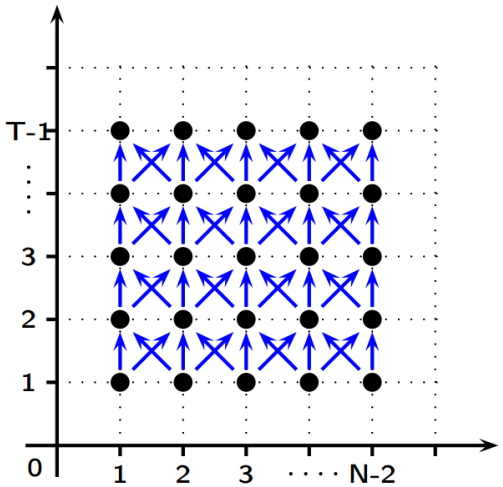


Figure: Original iteration space

[1] Tiling Stencil Computations to Maximize Parallelism, *Vinayaka Bandishti, Irshad Pananilath, and Uday Bondhugula*, ACM/IEEE Supercomputing (SC) 2012

CONCURRENT START AND DIAMOND TILING[1]

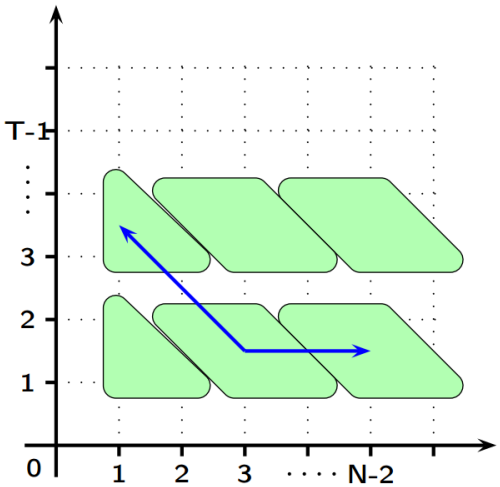


Figure: Pipelined start

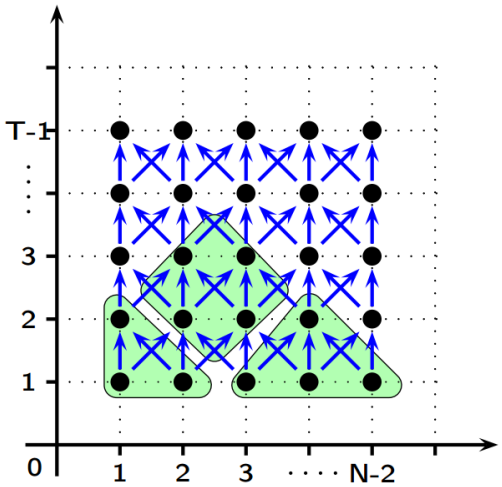


Figure: Group as diamonds

[1] Tiling Stencil Computations to Maximize Parallelism, *Vinayaka Bandishti, Irshad Pananilath, and Uday Bondhugula*, ACM/IEEE Supercomputing (SC) 2012

CONCURRENT START AND DIAMOND TILING[1]

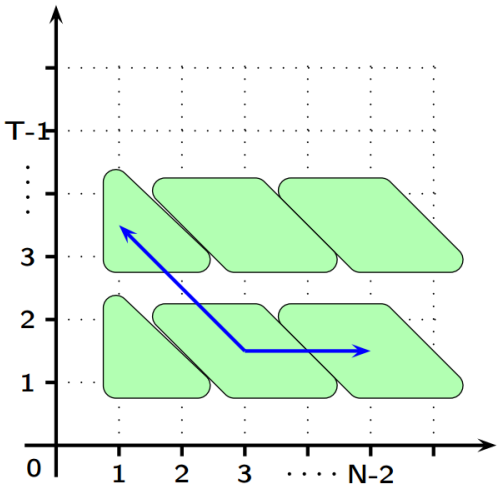


Figure: Pipelined start

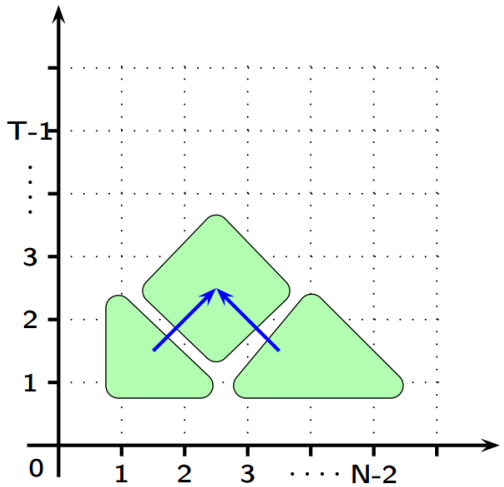


Figure: Concurrent start possible

[1] Tiling Stencil Computations to Maximize Parallelism, *Vinayaka Bandishti, Irshad Pananilath, and Uday Bondhugula*, ACM/IEEE Supercomputing (SC) 2012

CONCURRENT START AND DIAMOND TILING[1]

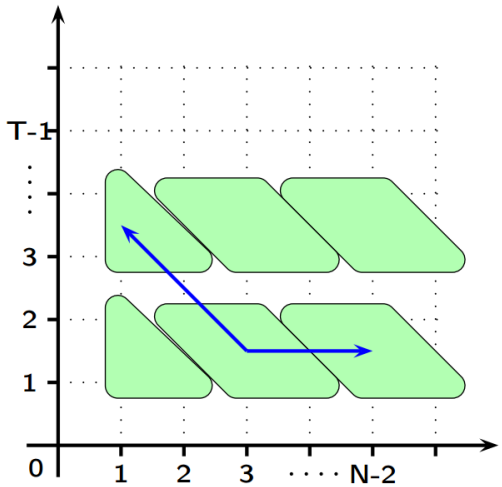


Figure: Pipelined start

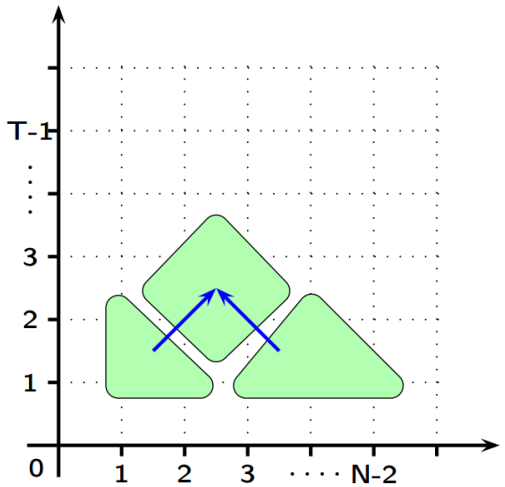


Figure: Concurrent start possible

- Validity conditions
  - Face allowing concurrent should be strictly within the cone of the tiling hyperplanes
  - Tile schedule should be parallel to that face

[1] Tiling Stencil Computations to Maximize Parallelism, *Vinayaka Bandishti, Irshad Pananilath, and Uday Bondhugula*, ACM/IEEE Supercomputing (SC) 2012

CONCURRENT START AND DIAMOND TILING[1]

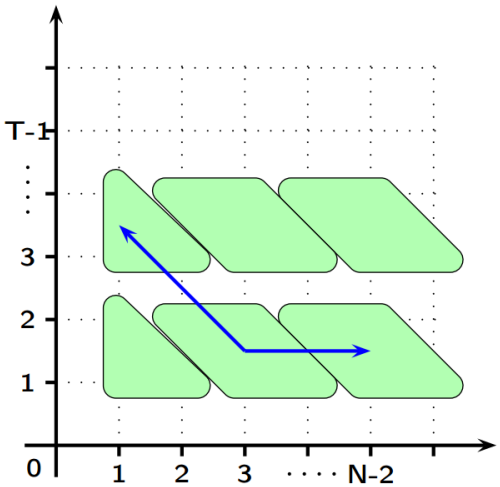


Figure: Pipelined start

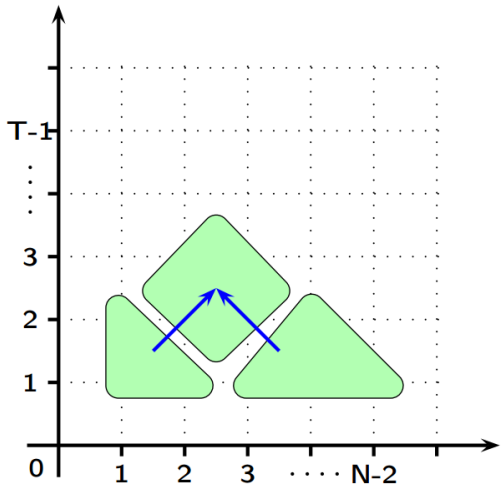


Figure: Concurrent start possible

- Validity conditions
  - Face allowing concurrent should be strictly within the cone of the tiling hyperplanes
  - Tile schedule should be parallel to that face
- Factor of 2× to 15× improvement over previous compiler techniques on a set of benchmarks on 12 cores

[1] Tiling Stencil Computations to Maximize Parallelism, *Vinayaka Bandishti, Irshad Pananilath, and Uday Bondhugula*, ACM/IEEE Supercomputing (SC) 2012

CONCURRENT START AND DIAMOND TILING[1]

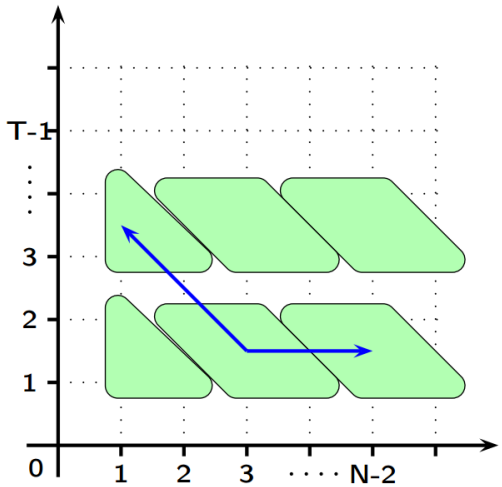


Figure: Pipelined start

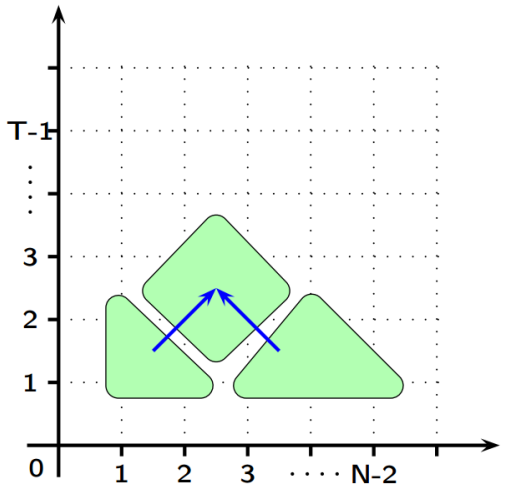


Figure: Concurrent start possible

- Validity conditions
  - Face allowing concurrent should be strictly within the cone of the tiling hyperplanes
  - Tile schedule should be parallel to that face
- Factor of 2× to 15× improvement over previous compiler techniques on a set of benchmarks on 12 cores
- Improvement of 4% to 27% over a domain-specific compiler (Pochoir) on 12 cores

[1] Tiling Stencil Computations to Maximize Parallelism, *Vinayaka Bandishti, Irshad Pananilath, and Uday Bondhugula*, ACM/IEEE Supercomputing (SC) 2012

POLYHEDRAL OPTIMIZATIONS - TIME TILING

- LBM can be written using storage for either one grid or two grids

POLYHEDRAL OPTIMIZATIONS - TIME TILING

- LBM can be written using storage for either one grid or two grids
- One grid ⇒ Separate collision and advection
- Fused Collision + Advection ⇒ Two grids

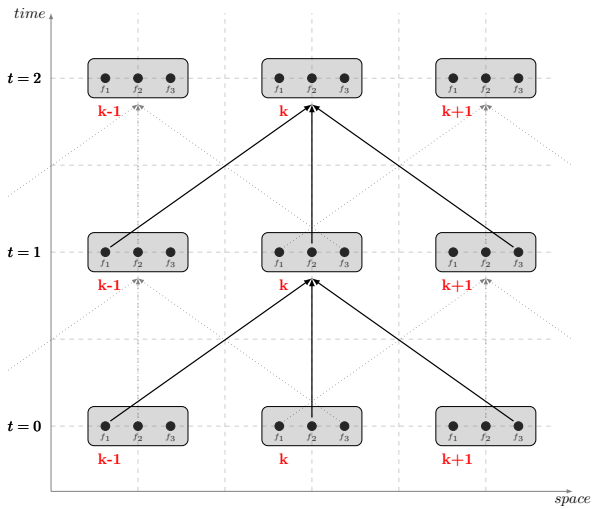


Figure: 1D LBM with single grid

POLYHEDRAL OPTIMIZATIONS - TIME TILING

- LBM can be written using storage for either one grid or two grids
- One grid ⇒ Separate collision and advection
- Fused Collision + Advection ⇒ Two grids

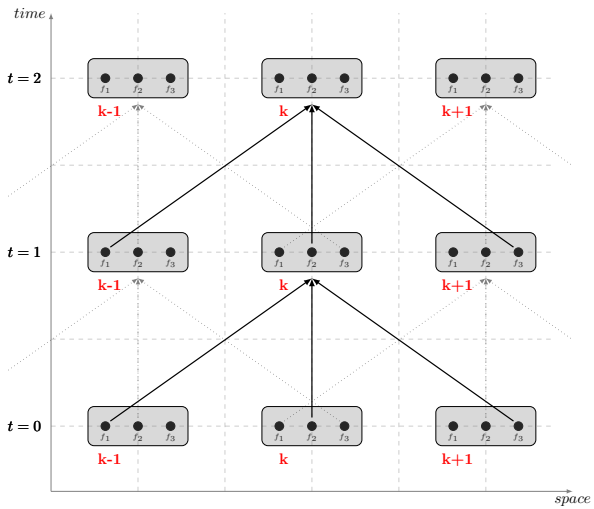


Figure: 1D LBM with single grid

- **Not possible to "*time tile*" LBM with single grid**

POLYHEDRAL OPTIMIZATIONS - TIME TILING

- LBM can be written using storage for either one grid or two grids
- One grid ⇒ Separate collision and advection
- Fused Collision + Advection ⇒ Two grids

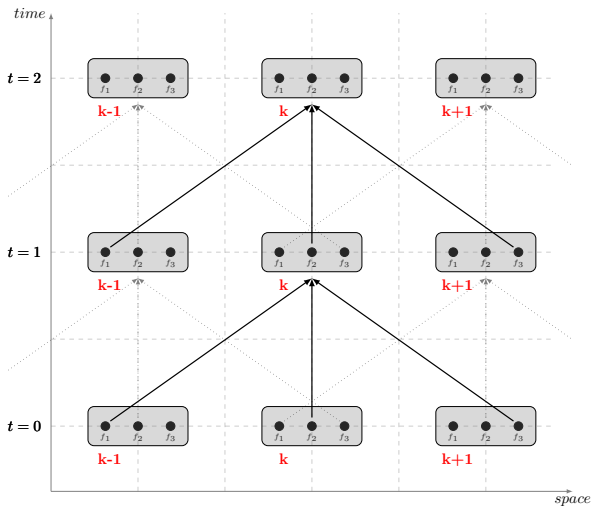


Figure: 1D LBM with single grid

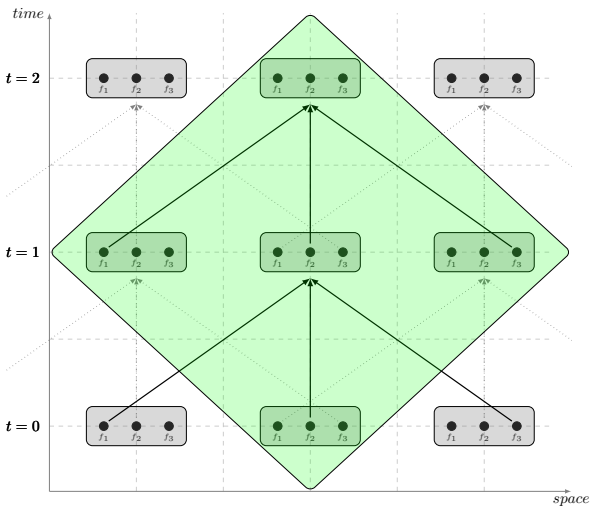


Figure: Pull scheme on a 1D LBM

- **Not possible to "*time tile*" LBM with single grid**

POLYHEDRAL OPTIMIZATIONS - TIME TILING

- LBM can be written using storage for either one grid or two grids
- One grid ⇒ Separate collision and advection
- Fused Collision + Advection ⇒ Two grids

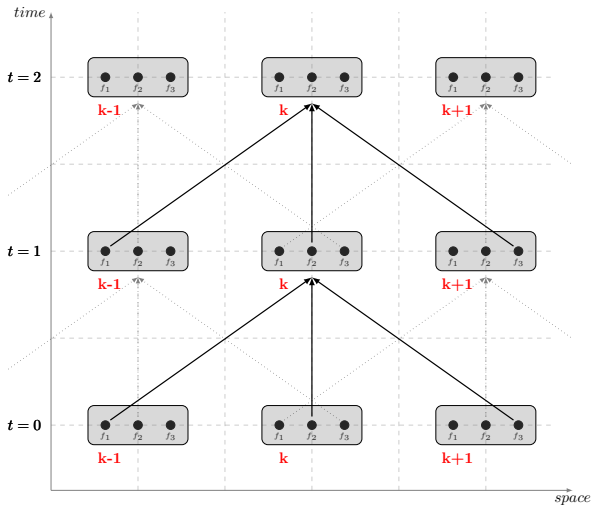


Figure: 1D LBM with single grid

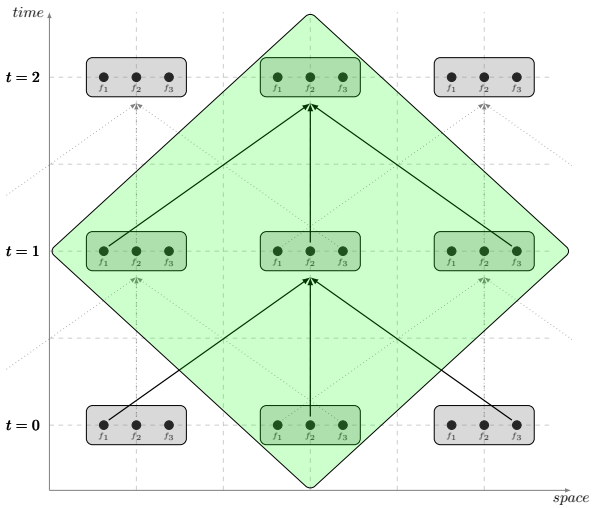


Figure: Pull scheme on a 1D LBM

- **Not possible to "*time tile*" LBM with single grid**
- **Time tiling is possible with two grids**

msLBM - LBM Optimization Framework

- We utilize a fused version of the LBM kernel
  - No explicit *advection* phase
  - 2 data grids with a pull scheme to read data
  - Array-of-Structures (*AoS*) layout for data

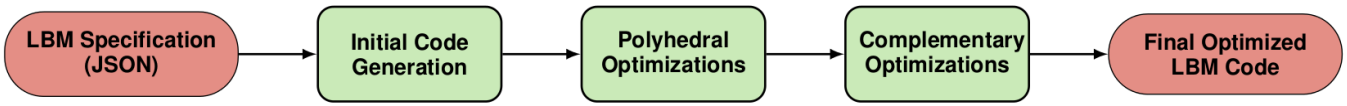


Figure: System Overview

[1] JSON (**J**ava**S**cript **O**bject **N**otation) is a lightweight data-interchange format.

INPUT TO POLYHEDRAL TILER

```
#pragma scop
for (t = 0; t < _nTimesteps; t++)
    for (y = 2; y < _nY; y++)
        for (x = 1; x < _nX; x++)
            lbm_kernel(grid[t % 2][y][x][C],
                grid[t % 2][y - 1][x + 0][N],
                grid[t % 2][y + 1][x + 0][S],
                grid[t % 2][y + 0][x - 1][E],
                grid[t % 2][y + 0][x + 1][W],
                grid[t % 2][y - 1][x - 1][NE],
                grid[t % 2][y - 1][x + 1][NW],
                grid[t % 2][y + 1][x - 1][SE],
                grid[t % 2][y + 1][x + 1][SW],

                &grid[(t + 1) % 2][y][x][C],
                &grid[(t + 1) % 2][y][x][N],
                &grid[(t + 1) % 2][y][x][S],
                &grid[(t + 1) % 2][y][x][E],
                &grid[(t + 1) % 2][y][x][W],
                &grid[(t + 1) % 2][y][x][NE],
                &grid[(t + 1) % 2][y][x][NW],
                &grid[(t + 1) % 2][y][x][SE],
                &grid[(t + 1) % 2][y][x][SW], t, y, x);
```

- Use the **PET polyhedral frontend [Verdoolaege and Grosser 2012]**: the LBM collision is treated as a blackbox (abstracted as a single function)
- Dependence structure is now similar to "toy time-iterated stencils"
- **All time tiling strategies can now be applied!**

COMPLEMENTARY OPTIMIZATIONS

Simplification of the LBM Kernel

- In most cases, the compiler is not able to simplify the LBM kernel to remove common expressions

COMPLEMENTARY OPTIMIZATIONS

Simplification of the LBM Kernel

- In most cases, the compiler is not able to simplify the LBM kernel to remove common expressions
- We perform common sub-expression elimination using **Python SimPy** on the composed LBM kernel
  - reduces the number of FP-ops required for LBM kernel execution

COMPLEMENTARY OPTIMIZATIONS

Alignment, Vectorization, Inlining & Unrolling

- For the polyhedral optimization part, the LBM compute kernel was treated as a blackbox function
  - prevents the compiler from performing certain optimizations like vectorization

COMPLEMENTARY OPTIMIZATIONS

Alignment, Vectorization, Inlining & Unrolling

- For the polyhedral optimization part, the LBM compute kernel was treated as a blackbox function
  - prevents the compiler from performing certain optimizations like vectorization
- Perform structuring to improve vectorizability
  - Generate data alignment directives
  - Unroll the LBM kernel by a suitable factor

COMPLEMENTARY OPTIMIZATIONS

Boundary Tile Separation

- **Boundary conditions**: *in-domain* tiles and *boundary* tiles
- LBM requires special treatment for the boundary tiles of the domain
  - results in *IF* conditions inside the LBM kernel to identify and handle boundaries separately
  - results in degraded performance for the vast number of in-domain tiles

COMPLEMENTARY OPTIMIZATIONS

Boundary Tile Separation

- **Boundary conditions**: *in-domain* tiles and *boundary* tiles
- LBM requires special treatment for the boundary tiles of the domain
  - results in *IF* conditions inside the LBM kernel to identify and handle boundaries separately
  - results in degraded performance for the vast number of in-domain tiles
- We identify and generate different LBM kernels for boundary and in-domain tiles
- In-domain LBM kernel can be both unrolled and inlined for better performance

EXPERIMENTAL SETUP

Intel Xeon E5-2680 (SandyBridge)	
Clock	2.7 GHz
Cores / socket	8
Total cores	16
L1 cache / core	32 KB
L2 cache / core	512 KB
L3 cache / socket	20 MB
Peak GFLOPs	172.8
Compiler	Intel C compiler (icc) 14.0.1
Compiler flags	-O3 -xHost -ipo -fno-alias -fno-fnalias -restrict -fp-model precise -fast-transcendentals
Linux kernel	3.8.0-38

Table: Architecture details

- We compare the performance of our framework on 7 benchmarks against
  - Palabos - an open-source CFD solver based on LBM
  - Compiler auto-parallelization [*icc-auto-par*]
  - Naive manual parallelization using OpenMP [*icc-omp-par*]

BENCHMARKS

- Lid Driven Cavity - `d2q9`, `d3q19` and `d3q27`

BENCHMARKS

- Lid Driven Cavity - `d2q9`, `d3q19` and `d3q27`
- SPEC LBM [*470.lbm* from SPEC2006] - `d3q19`

BENCHMARKS

- Lid Driven Cavity - `d2q9`, `d3q19` and `d3q27`
- SPEC LBM [*470.lbm* from SPEC2006] - `d3q19`
- Poiseuille Flow - `d2q9`

BENCHMARKS

- Lid Driven Cavity - `d2q9`, `d3q19` and `d3q27`
- SPEC LBM [*470.lbm* from SPEC2006] - `d3q19`
- Poiseuille Flow - `d2q9`
- Flow Past Cylinder - `d2q9`

BENCHMARKS

- Lid Driven Cavity - `d2q9`, `d3q19` and `d3q27`
- SPEC LBM [*470.lbm* from SPEC2006] - `d3q19`
- Poiseuille Flow - `d2q9`
- Flow Past Cylinder - `d2q9`
- MRT - GLBM - `d2q9`

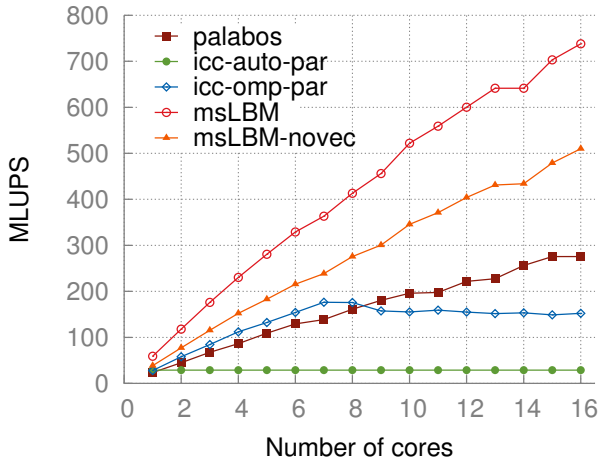
BENCHMARKS

- Lid Driven Cavity - `d2q9`, `d3q19` and `d3q27`
- SPEC LBM [*470.lbm* from SPEC2006] - `d3q19`
- Poiseuille Flow - `d2q9`
- Flow Past Cylinder - `d2q9`
- MRT - GLBM - `d2q9`

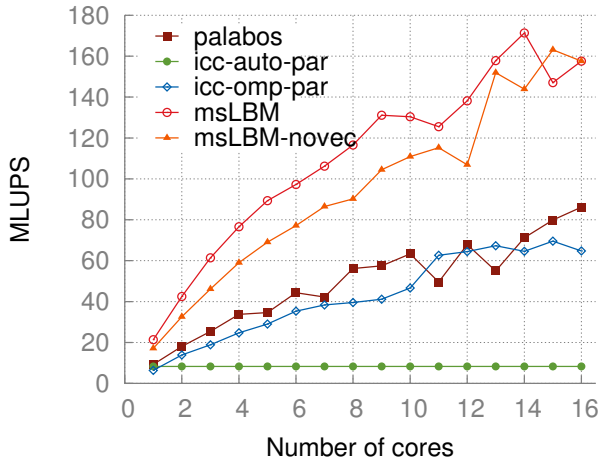
Performance Metrics

- MLUPS - Million Lattice site Updates Per Second
- MEUPS - Million Element Updates Per Second

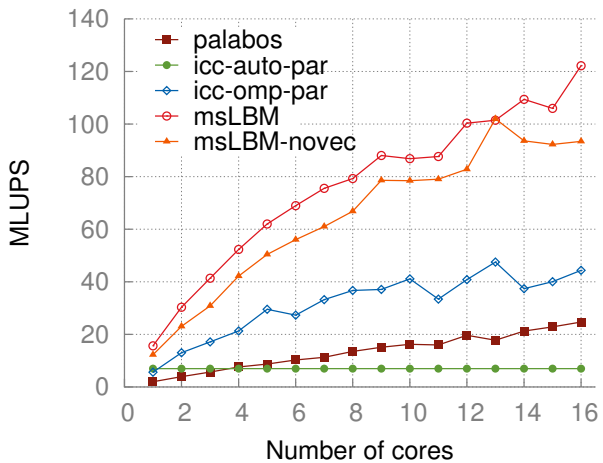
PERFORMANCE - LDC



(a) `ldc-d2q9`

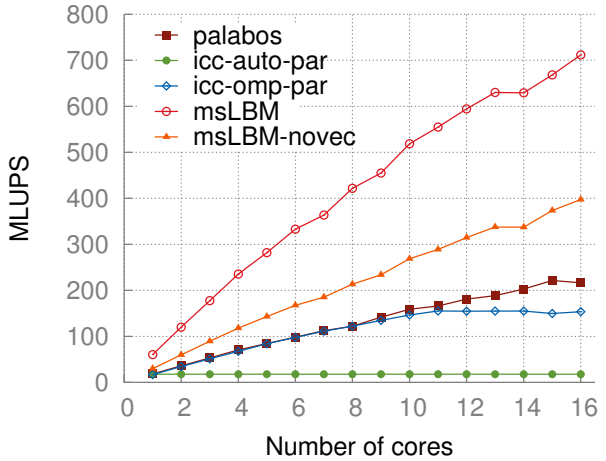


(b) `ldc-d3q19`

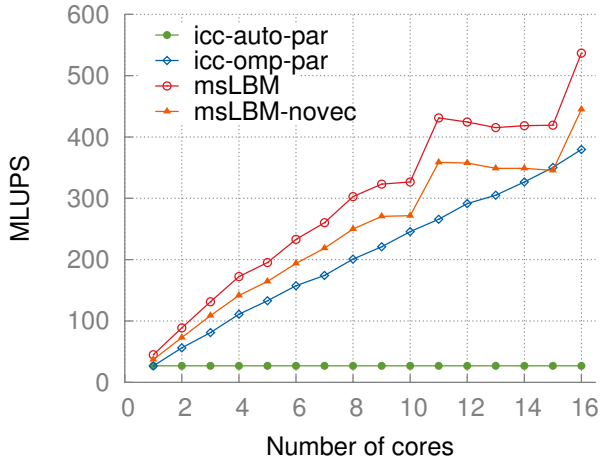


(c) `ldc-d3q27`

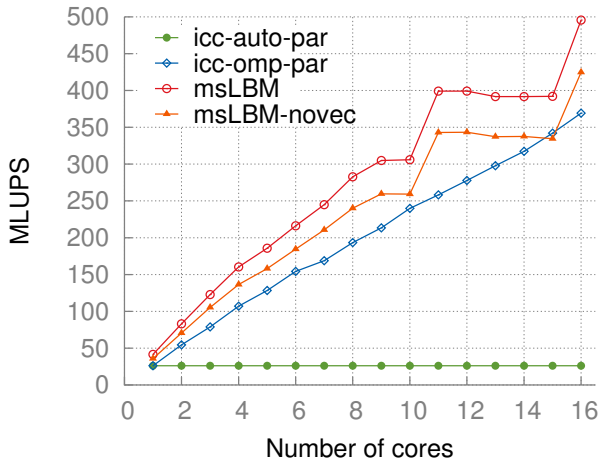
PERFORMANCE - MRT,POISEUILLE,FPC



(a) `mrt-d2q9`

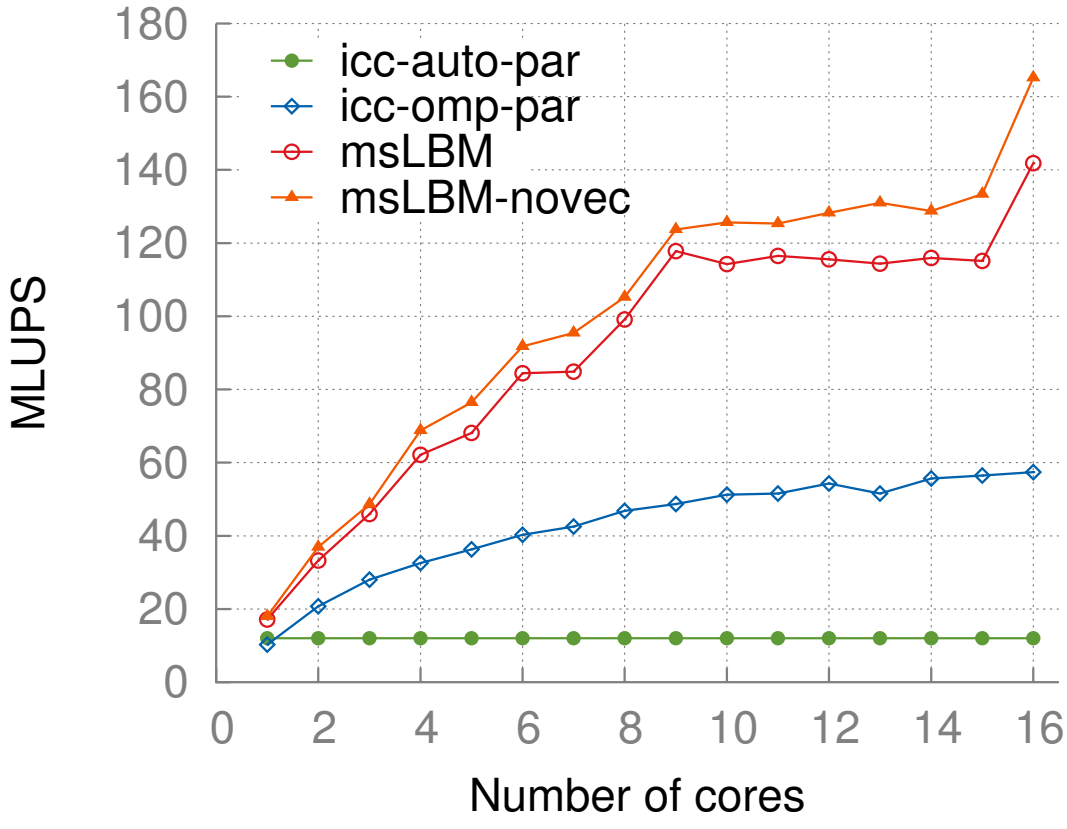


(b) `poiseuille`



(c) `fpc-d2q9`

PERFORMANCE - SPEC LBM



(a) `spec-lbm` performance

- D3Q19 lattice with an ellipsoid obstacle in the grid
- An improvement of 2.47x

ROOFLINE PERFORMANCE MODEL

- Computational characteristics of numerical methods can vary dramatically

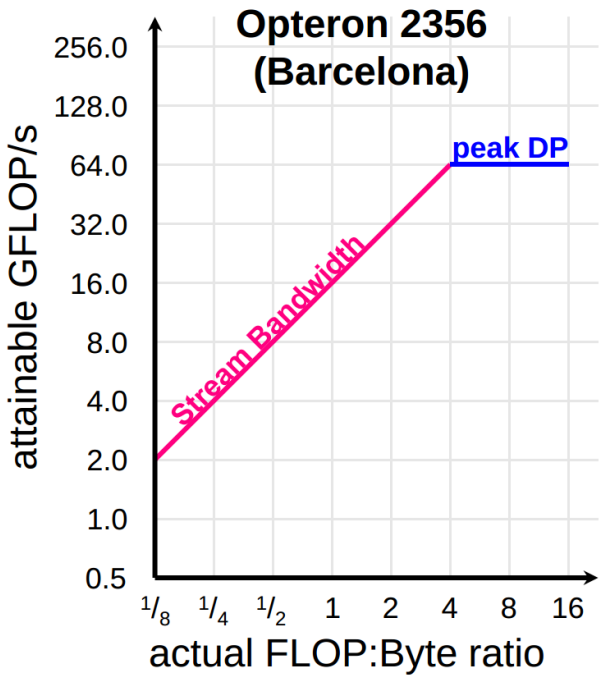


Figure: Basic roofline plot

ROOFLINE PERFORMANCE MODEL

- Computational characteristics of numerical methods can vary dramatically
- Goal: integrate in-core performance, memory bandwidth, and locality into a single readily understandable performance figure

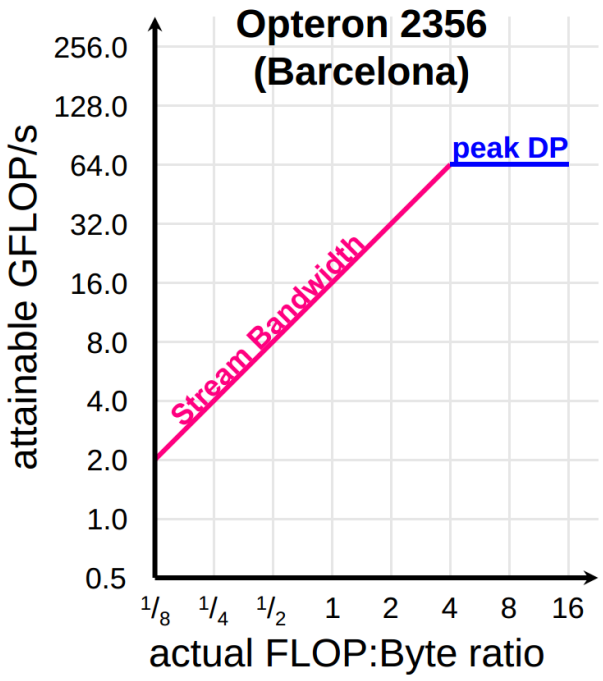


Figure: Basic roofline plot

ROOFLINE PERFORMANCE MODEL - CONTD.

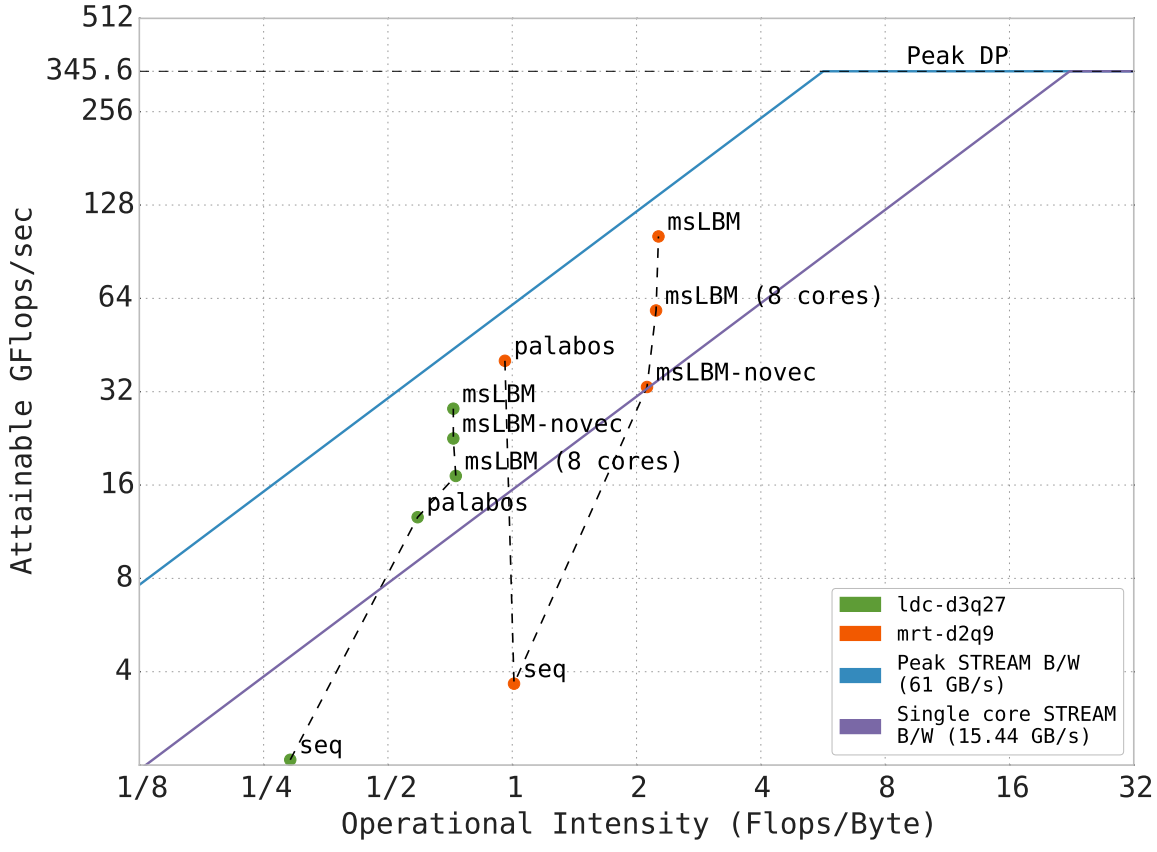


Figure: Roofline model for `mrt-d2q9` & `ldc-d3q27`

ROOFLINE PERFORMANCE MODEL - CONTD.

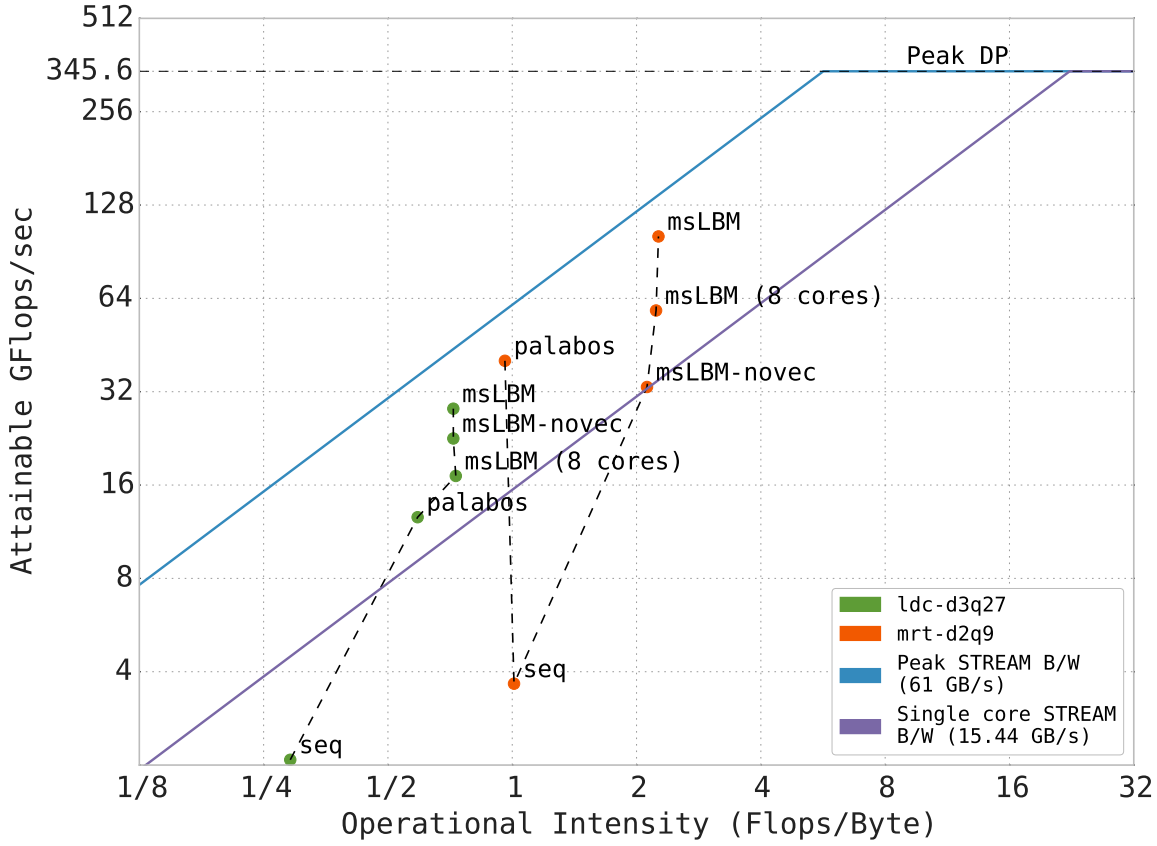


Figure: Roofline model for `mrt-d2q9` & `ldc-d3q27`

- `msLBM` obtains further improvement over Palabos in both operational intensity and peak achievable performance

RESULTS - SUMMARY

- Our scheme consistently outperformed Palabos on average by 3× on 16 cores of an Intel Sandybridge system
- A significant improvement of 2.47× over the best native production compiler for SPEC LBM benchmark
- `msLBM` obtains improvement over Palabos in both operational intensity and sustained performance

CONCLUSIONS AND FUTURE WORK

- Proposed a domain-specific framework to generate optimized code for LBM computations
- Reasoned about automatic time tiling for LBM kernels

CONCLUSIONS AND FUTURE WORK

- Proposed a domain-specific framework to generate optimized code for LBM computations
- Reasoned about automatic time tiling for LBM kernels
- Significant improvements in various cases
- Characterized LBM performance with the Roofline model
- Implemented the system using PET, Pluto and Python Sym

CONCLUSIONS AND FUTURE WORK

- Proposed a domain-specific framework to generate optimized code for LBM computations
- Reasoned about automatic time tiling for LBM kernels
- Significant improvements in various cases
- Characterized LBM performance with the Roofline model
- Implemented the system using PET, Pluto and Python Sym
- **Future:** Integrate with DSL frontend of PolyMage [Mullapudi et al ASPLOS 2015]

ACKNOWLEDGEMENTS

- ***Anand Deshpande, Aniruddha Shet, Bharat Kaul, and Sunil Sherlekar*** from **Intel Labs, Bangalore** for discussions
- **Intel Labs, Bangalore** for equipment and software donation

Thank You!