Lattice Boltzmann -
Efficient Implementation and Parallelization
VIS 2008 Tutorial

Klaus Iglberger, Ulrich Rüde
University Erlangen-Nuremberg – System Simulation

October 21st 2008
The Lattice Boltzmann Method

The Need for Optimization
- Moore’s Law
- The Memory Gap
- Multi-Core CPUs

Lattice Boltzmann Optimization Techniques
- Blocking Techniques
- Grid Compression

OpenMP and MPI Parallelization

Conclusion
Outline

- The Lattice Boltzmann Method
- The Need for Optimization
  - Moore’s Law
  - The Memory Gap
  - Multi-Core CPUs
- Lattice Boltzmann Optimization Techniques
  - Blocking Techniques
  - Grid Compression
- OpenMP and MPI Parallelization
- Conclusion
The Lattice Boltzmann Method
The Lattice Boltzmann Method

The Boltzmann Equation

\[ \frac{\partial f}{\partial t} + \xi \frac{\partial f}{\partial x} + K \frac{\partial f}{\partial \xi} = Q(f, f), \quad f = f(t, x, \xi) \]
The Lattice Boltzmann Method

**The Boltzmann Equation**

\[
\frac{\partial f}{\partial t} + \xi \frac{\partial f}{\partial x} + K \frac{\partial f}{\partial \xi} = Q(f, f), \quad f = f(t, x, \xi)
\]

**The Discretized Boltzmann Equation (BGK-scheme)**

\[
f_i(x + c_i \Delta t, t + \Delta t) - f_i(x, t) = \frac{1}{\tau_i}(f_i - f_i^{eq}).
\]
The Lattice Boltzmann Method

The Boltzmann Equation
\[
\frac{\partial f}{\partial t} + \xi \frac{\partial f}{\partial x} + K \frac{\partial f}{\partial \xi} = Q(f, f), \quad f = f(t, x, \xi)
\]

The Discretized Boltzmann Equation (BGK-scheme)
\[
f_i(x + c_i \Delta t, t + \Delta t) - f_i(x, t) = -\frac{1}{\tau} (f_i - f_i^{eq}).
\]

Collide and Stream Step

The Collide Step: \( \tilde{f}_i(x, t + \Delta t) = f_i(x, t) - \frac{1}{\tau} (f_i - f_i^{eq}) \)

The Stream Step: \( f_i(x + c_i \Delta t, t + \Delta t) = \tilde{f}_i(x, t + \Delta t) \)
The Lattice Boltzmann Method

2-dimensional D2Q9 model
- 9 directions
- ⇒ 9-point stencil
The Lattice Boltzmann Method

2-dimensional D2Q9 model
- 9 directions
- \( \Rightarrow \) 9-point stencil

3-dimensional D3Q19 model
- 19 directions
- \( \Rightarrow \) 19-point stencil
Outline

- The Lattice Boltzmann Method
- The Need for Optimization
  - Moore’s Law
  - The Memory Gap
  - Multi-Core CPUs
- Lattice Boltzmann Optimization Techniques
  - Blocking Techniques
  - Grid Compression
- OpenMP and MPI Parallelization
- Conclusion
1965 G. Moore (co-founder of Intel) claimed that the number of transistors on processor chip doubles every 12-24 months.

Processor speed grew roughly at the same rate (approx. 43% p.a.).

Why worry about performance?
Memory (DRAM) Gap

- Memory bandwidth grows only at a speed of 7% p.a.
- Memory latency remains constant/increases in terms of CPU speed
- Loading a single data item from main memory can cost 100s of cycles

**Memory Bottleneck**

![Graph showing the gap between CPU frequency and DRAM speeds](image)
Memory (DRAM) Gap

- Memory bandwidth grows only at a speed of 7% p.a.
- Memory latency remains constant/increases in terms of CPU speed
- Loading a single data item from main memory can cost 100s of cycles

Optimization of main memory access is mandatory for most applications in order to improve performance
Multi-Core CPUs

By courtesy of D. Vrsalovic, Intel

1.00x

Max Frequency

Performance
Power
Multi-Core CPUs

By courtesy of D. Vrsalovic, Intel
Multi-Core CPUs

By courtesy of D. Vrsalovic, Intel

![Bar chart showing performance comparisons between Dual-Core and other categories.](chart.png)
Outline

- The Lattice Boltzmann Method
- The Need for Optimization
  - Moore’s Law
  - The Memory Gap
  - Multi-Core CPUs
- Lattice Boltzmann Optimization Techniques
  - Blocking Techniques
  - Grid Compression
- OpenMP and MPI Parallelization
- Conclusion
Data Layout

- D2Q9-model:

- Collision-optimized data layout:

- Propagation-optimized data layout:
Fusion of the Stream and Collide Steps

```cpp
for( int t=0; t<timeSteps; ++t ) {
    stream_step();
    collide_step();
}
```
Fusion of the Stream and Collide Steps

```c
for( int t=0; t<timesteps; ++t ) {
    stream_step();
    collide_step();
}
```

```c
for( int t=0; t<timesteps; ++t ) {
    stream_collide_step();
}
```
Basic Performance

Definition

Our basic LBM code has a relative performance of 1.0 on our target architecture with our main compiler and compiler flags.
General Optimization Techniques

- Compiler optimizations
- Loop unrolling
- Loop fusion
- Arithmetic optimizations
- Reduction of common subexpressions
- ...

Klaus Iglberger, Ulrich Rüde (LSS Erlangen)
General Optimization Techniques

- Performance improvement by 100% is easily possible
Blocking Techniques

- Increase of spatial (and temporal) locality
- Improved cache usage
- 3-way blocking
  - Very easy to implement
  - Good to very good performance improvements
  - Applicable nearly everywhere
- 4-way blocking
  - Slightly more complex to implement
  - Even faster than 3-way blocking
  - Unfortunately only rarely applicable
3-way Blocking
3-way Blocking
3-way Blocking

- Standard implementation

```c
for( int i=0; i<Z; ++i ) {
    for( int j=0; j<Y; ++j ) {
        for( int k=0; k<X; ++k ) {
            // Update of cell (i,j,k)
        }
    }
}
```
3-way Blocking

- 3way-blocked implementation

```c
for( int ii=0; ii<Z; ii+=BLOCKSIZE ) {
    for( int jj=0; jj<Y; jj+=BLOCKSIZE ) {
        for( int kk=0; kk<X; kk+=BLOCKSIZE ) {
            for( int i=ii; i<ii+BLOCKSIZE; ++i ) {
                for( int j=jj; j<jj+BLOCKSIZE; ++j ) {
                    for( int k=kk; k<kk+BLOCKSIZE; ++k ) {
                        // Update of cell (i,j,k)
                    }
                }
            }
        }
    }
}
```
3-way Blocking

- Additional performance improvement by 40% is possible
4-way Blocking
4-way Blocking
Another performance improvement by 12% is possible.
Grid Compression

- Reduces the memory requirement by nearly 50%
- Requires some additional (tricky) implementation
- Implementation can be nicely hidden in wrapper classes/functions
Grid Compression

- Reduces the memory requirement by nearly 50%
- Requires some additional (tricky) implementation
- Implementation can be nicely hidden in wrapper classes/functions
Grid Compression

- Unburdens the memory bus ⇒ about 10% performance improvement
Outline

- The Lattice Boltzmann Method
- The Need for Optimization
  - Moore’s Law
  - The Memory Gap
  - Multi-Core CPUs
- Lattice Boltzmann Optimization Techniques
  - Blocking Techniques
  - Grid Compression
- OpenMP and MPI Parallelization
- Conclusion
OpenMP Parallelization

- Shared memory parallelization
- Available with the Intel compilers and the GCC (from version 4.2)
- Thread setup based on preprocessor directives...
OpenMP Parallelization

- Shared memory parallelization
- Available with the Intel compilers and the GCC (from version 4.2)
- Thread setup based on preprocessor directives...

```c
int i;

#pragma omp parallel for private(i)
for (i = 0; i < 1000; ++i) {
    ...
}
```
OpenMP Parallelization

Partitioning for two OpenMP Threads: Split along Y Axis

Full Simulation Domain

Thread 1

Thread 2

Thread Barrier

Thread Barrier

Thread Barrier
MPI Parallelization

- MPI = Message passing interface
- Distributed memory parallelization
- More complex than a shared memory parallelization
MPI Parallelization
MPI Parallelization
MPI Parallelization

- Gather all boundary distribution functions in a send buffer
- Send all distribution functions to the neighboring process
- Write the distribution function into the according memory locations
Conclusion

- Choice of a suitable data layout is imperative for good performance
- Fusion of stream and collide step is significant for the performance
- Arithmetic optimizations, blocking and grid compression boost the performance by 328%
- Nearly perfect parallelization is possible via OpenMP or MPI
Choice of a suitable data layout is imperative for good performance.
Fusion of stream and collide step is significant for the performance.
Arithmetic optimizations, blocking and grid compression boost the performance by 328%.
Nearly perfect parallelization is possible via OpenMP or MPI.

Thank you for your attention!