Optimization of Cache Oblivious Lattice Boltzmann Method in 2D and 3D

Aditya Nitsure, Klaus Iglberger, Ulrich Rüde, Christian Feichtinger*

Gerhard Wellein, Georg Hager†
Friedrich Alexander Universität Erlangen-Nürnberg
naditya@rediffmail.com

Abstract

The Lattice Boltzmann Method (LBM) is one of the modern techniques to simulate fluids. A lot of work has already been done to speed up the LBM on cache-based microprocessor architecture. In order to achieve high performance on these architectures, not only the clock rate of the processor plays an important role, but also the significant influence of the caches have to be taken into account. A common method to exploit spatial and temporal data locality by reusing the cache is blocking (a cache aware algorithm). In this paper, a " *Cache Oblivious Lattice Boltzmann Algorithm (COLBA)* " and its optimization techniques are presented. Therefore Frigo's stencil based cache oblivious algorithm [FS05] has been coupled to the LBM. Main aim of this work has been the comparison of cache aware/blocked LBM and COLBA on different architectures. The results of the COLBA have shown equivalent performance to the fastest LBM implementations available at the Regional Computing Center at Erlangen (RRZE) [Don04] also less cache misses and less buss access than the iterative LBM implementation.

1 Introduction

Since the past decade, cache-based microprocessors and hierarchy based memory structures are very common in the area of High Performance Computing. Despite improvements in technology, microprocessors are still much faster than the main memory. Therefore memory access time is increasingly the bottleneck in overall application performance. To close the gap between processor speed and memory bandwidth, modern architectures use hierarchies of small but fast caches. Blocking is one of the cache based approaches of performance optimization which reduces the data transfer from/to main memory by increasing spatial and temporal locality i.e. reusing the cached data again and again. In the recent computational development of the Lattice Boltzmann Method, blocking has been used to improve the computational performance [Igl03]. Normally, for effective blocking, the cache parameters like cache size, cache line length etc. are considered. In this paper, a *Cache Oblivious Lattice Boltzmann Algorithm (COLBA)* for cache based architectures is

*Lehrstuhl für Systemsimulation (Informatik 10), D-91058 Erlangen, Germany
†Regionales RechenZentrum Erlangen, D-91058 Erlangen, Germany

introduced, which is independent of the cache parameters.

Previously, for problems like matrix multiplication, FFT, sorting (merge sort) and matrix transpose, it was observed by Prokop [Pro99] that the cache oblivious algorithm (COA) executes in less than 70% of the time than the iterative algorithm for problem sizes that do not fit in L2-cache. The idea of a stencil based COA for solving stencil based finite difference problems has been presented by Frigo. [FS05]. This algorithm is flexible to adjust with any point stencil calculation, which makes the LBM a promising candidate. The COA assumes an ideal cache model, which contains an optimal replacement policy, exactly two levels of memory, automatic replacement and full associativity. These points are theoretically proved by Frigo and Prokop. [FS05] [Pro99].

2 The Lattice Boltzmann Method

The lattice Boltzmann Method is a discrete computational method for fluid simulation based on the Boltzmann equation. The kinetic model using the Boltzmann equation with the single relaxation time approximation (BGK model) [MSY02], after discretization in space x and time t is

$$f_\alpha\left(\vec{x}+\vec{e_\alpha}\Delta t, t+\Delta t\right) = f_\alpha\left(\vec{x},t\right) - \frac{\Delta t}{\tau}\Big[f_\alpha\left(\vec{x},t\right) - f_\alpha^{(0)}\left(\vec{x},t\right)\Big], \tag{1}$$

where $\vec{x}$ is a cell in the discretized simulation domain (square cells in 2D and cubes in 3D), t is the current time step and $t+\Delta t$ the next time step. The difference between the old particle distribution function (PDF) $f_\alpha\left(\vec{x},t\right)$ and the new PDF $f_\alpha\left(\vec{x}+\vec{e_\alpha}\Delta t, t+\Delta t\right)$, is the weighted summation of the equilibrium distribution $f_\alpha^{(0)}\left(\vec{x},t\right)$ and the old PDF. To facilitate the implementation of the LBM, Eq. 1 can be separated into two steps, known as the collide step and the stream step, shown in Eq. 2 and 3 :

$$\tilde{f_\alpha}\left(\vec{x},t+\Delta t\right) = f_\alpha\left(\vec{x},t\right) - \frac{\Delta t}{\tau}\Big[f_\alpha\left(\vec{x},t\right) - f_\alpha^{(0)}\left(\vec{x},t\right)\Big] \tag{2}$$

$$f_\alpha\left(\vec{x}+\vec{e_\alpha}\Delta t, t+\Delta t\right) = \tilde{f_\alpha}\left(\vec{x},t+\Delta t\right) \tag{3}$$

where $\tilde{f}\left(t,\vec{x}\right)$ denotes the post-collision state of the distribution function. Eq. 2 can be interpreted as the collide step : the particles that are currently at lattice site $\vec{x}$ at time step t interact with one another, affecting the particle distribution in the next time step $t+\Delta t$. This interaction is modeled by the right hand side of Eq. 2. Eq. 3 can be interpreted as the stream step : the flow of the particles from one lattice site to the adjacent sites. The particles described by the value of the PDF $\tilde{f_N}\left(\vec{x},t+\Delta t\right)$ will flow to the site in the north of the current site, described by $f_N\left(\vec{x}+\vec{e_N}\Delta t, t+\Delta t\right)$ and so on for each direction except center C, which remain at the current lattice site. For this work a typical discretization scheme the D2Q9 model has been used in 2D case and D3Q19 in 3D case [Igl03].

3 Cache Oblivious Lattice Boltzmann Algorithm

The basic principle behind Frigo's COA [FS05] is traversal of a space-time domain of unknowns by optimally exploring spacial and temporal locality. During the traversal each unknown is updated by the evaluation of finite difference stencil. As the LBM is a stencil based method, it is suitable to integrate it into the COA. Thus the COLBA traverses the discretized space-time domain in a specific order optimized for cache reusage and calls the LBM kernel at appropriate locations. Inside the LBM kernel all the computations necessary for the LBM are executed. For the explanation of the traversal order here a 1D space, a 1D time domain (2D space-time domain) and with a 3 point stencil calculation i.e. the new updated value at each point x_i at time t depends on x_{i-1}, x_i, x_{i+1} at time $t-1$. Thus in the following example, a 2D discretized space-time rectangular domain has been used. The algorithm is based on a divide and conquer strategy. The complete rectangular space-time domain is thereby divided into triangles, parallelograms and trapezoids. Although the interest is in traversing a rectangular space-time domain, the algorithm can be explained with the help of trapezoidal region as shown in Fig. 1. The algorithm visits all points of the trapezoid in an order that respects the stencil dependencies. It decomposes the trapezoid recursively into smaller trapezoids either by a space or time cuts. The space cut is done, if the width is at least twice the height of the trapezoid. Then the trapezoid is cut along the line with slope -1 through the center of the trapezoid in order to preserve the dependencies in the stencil calculation. If a space cut is not possible, a time cut is done along the horizontal line through the center of trapezoid. Thus in both cases, two smaller trapezoids T1 and T2 are created as shown in Fig 1. The same procedure is applied on these newly created trapezoids until the problem is solvable inside the cache. The algorithm always first solves T1 and then T2. This traversal order is valid because no point in T1 depends on the updated (new) value of any point in T2. Thus by changing the slope parameter, the same algorithm can be applied to domains of different shapes like rectangular, triangular etc.

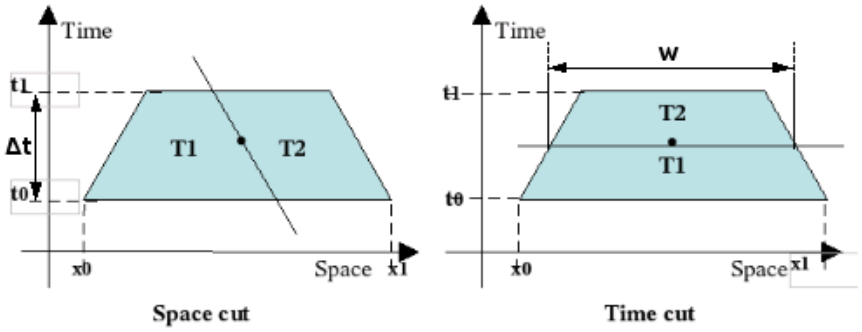

Figure 1: Illustration of space and time cuts, in a two dimensional trapezoid, consisting of a one dimensional space region and a one dimensional time region.

4 Optimization Strategies for COLBA

A number of optimization strategies to improve the performance of the COLBA have been implemented. Thereby, the main emphasis lay on the optimization of LBM kernel as it consists of more than 250 FLOPs [Igl03] and 5 (in case of 2D) or 7 (in case of 3D) *if*-conditions. A more detailed discussion can be found in [Nit06]. The applied strategies are shown in Tab. 1

Optimization Strategy	**Description**	**Performance**
Influence of space cut (SPC) factor	Different combinations of SPCs have been tried within the range of 0.25 to 4. The best performance has been observed for a SPC of value 2 in x, y and z direction for both the 2D and 3D case.	Improved
FLOP reduction	Floating point operations were cut down by rearranging the instructions inside the LBM kernel.	Improved
Small time steps	Dividing total time steps into a number of small chunks and solving the COLBA for each small chunk successively over the complete space domain	Not improved
Splitting LBM kernel in two functions	Two different functions. a) with boundary checks b) without boundary checks. COLBA divides bigger domains in a number of small regions. Few of these regions have boundary cells. Therefore boundary conditions do not have to be checked for regions without boundaries.	Substantially improved
#Pragma ivdep and #Pragma vector always	Improvement of 'for-loop' vectorization;	Substantially improved
Prefetching	Prefetches data into the cache. The function *_mm_prefetch()* [Int] has been used, which loads data from the main memory to a location closer to the processor (L2 cache) in advance.	Not improved

Table 1: Different optimization strategies.

5 Results

In this section the performance of the COLBA is measured and compared to the different LBM implementations. As test case the *Lid-Driven Cavity* problem has been used, which describes a box with a moving lid. In Fig. 2 the results for L2 cache misses, performance

(MLUps), Buss Access and DTLB misses are shown.

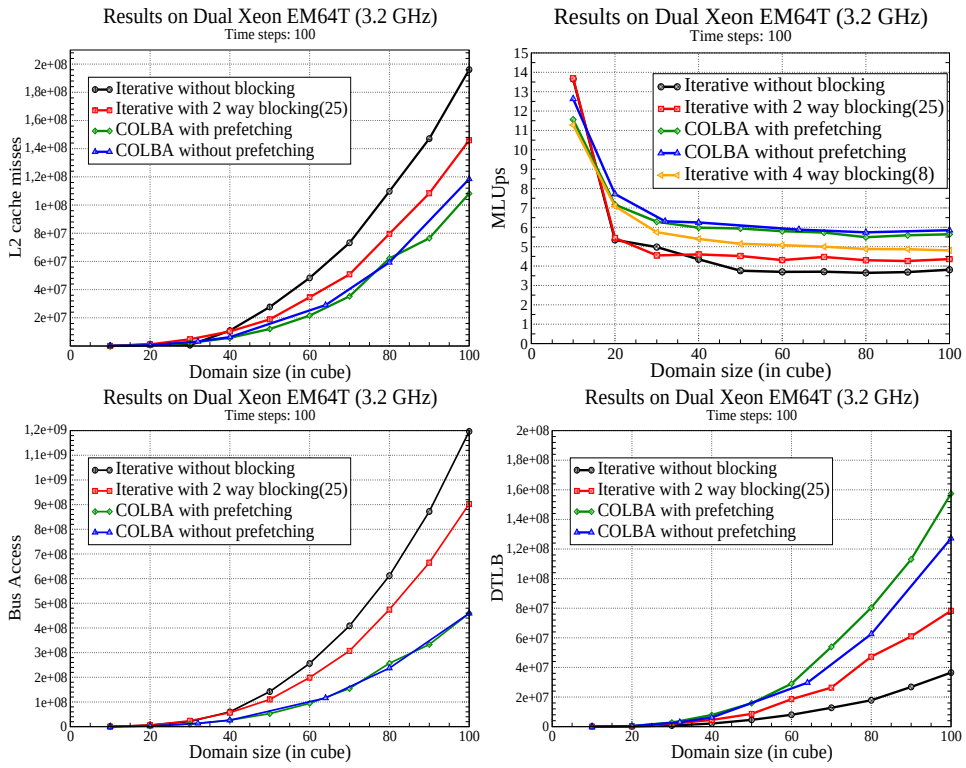

Figure 2: Comparison between COLBA and iterative algorithm for the D3Q19 model. a) Cache misses (top left) b) Mega Lattice site Updates per second. (MLUps) (top right) c) Buss access (bottom left) d) DTLB misses (bottom right).

For the cache misses Fig. 2 (a), it can be seen that, at a domain size of 10^3 the complete problem fits inside the cache so all the algorithms show less and almost equal cache misses which result from compulsory cache misses. As the domain size increases, the L2 cache misses increase. In Fig. 2 (b), the overall performance is depicted. Here it can be seen that the performance drop for the iterative algorithm is larger compared to the COLBA. The COLBA out performs the iterative 4-way blocked algorithm [Igl03]. In Fig. 2 (c) and (d), the COLBA shows less buss access and more DTLB misses compared to iterative algorithm. As the COLBA explores temporal and spacial locality, most of the data can be found in the cache which reduces the traffic between the main memory and the caches. This is the reason why the COLBA shows less cache misses, less bus access and higher performance compared to the iterative versions.

6 Conclusion

In this paper, the feasibility of a cache oblivious algorithm in the context of the lattice boltzmann method was shown. The COLBA behaved as expected as the implementation showed substantially less cache misses compared to the iterative implementations of the LBM. Additionally, the performance was equivalent to the fastest LBM implementation at the RRZE. The performance of the COLBA could be increased by splitting the LBM kernel into two functions and by the usage of '#pragma ivdep' and '#pragma vector always'. Along with these good features, some pit falls like high DTLB misses have been observed in the case of the COLBA. Also, the ability of the COLBA to explore the cache makes it useless on vector machines. For this work the COLBA has only been tested for simple geometry like Lid-Driven Cavity. There is still a wide scope remaining for future implementation, e.g. for handling complex geometries and for various boundary conditions.

References

[Don04] Stefan Donath. On Optimized Implementations of the Lattice Boltzmann Method on Contemporary High Performance Architectures, 2004. Lehrstuhl für Systemsimulation, Universität Erlangen-Nürnberg.

[FS05] Matteo Frigo and Volker Strumpen. Cache Oblivious Stencil Computations. *IBM Austin Research Laboratory, 11501 Burnet Road, Austin, TX 78758*, 2005. Available at : http://www.fftw.org/ athena/papers/ics05.pdf.

[Igl03] Klaus Iglberger. Cache Optimizations for the Lattice Boltzmann Method in 3D, 2003. Lehrstuhl für Systemsimulation, Universität Erlangen-Nürnberg.

[Int] IntelÂ®. IntelÂ® C++ Compiler for Linux* Systems User's Guide.

[MSY02] Renwei Mei, Wei Shyy, and Dazhi Yu. Lattice Boltzmann Method for 3D Flows with Curved Boundary. *Technical Report NASA/CR-2002-211657, ICASE, NASA Langley Research Center, Hampton, Virginia*, June 2002.

[Nit06] Aditya Nitsure. Implementation and Optimization of a Cache Oblivious Lattice Boltzmann Algorithm. Master's thesis, Lehrstuhl für Systemsimulation, Universität Erlangen-Nürnberg, 2006.

[Pro99] Herald Prokop. Cache-Oblivious Algorithm. Master's thesis, Department of Electrical Engineering and Computer Science, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 1999. Available at : http://supertech.lcs.mit.edu/cilk/papers/index.html.