Procedia Computer Science

Volume 51, 2015, Pages 1403–1412

ICCS 2015 International Conference On Computational Science

On the Use of a Many-core Processor for Computational Fluid Dynamics Simulations

Sebastian Raase, Tomas Nordström

Halmstad University, Halmstad, Sweden
{sebastian.raase,tomas.nordstrom}@hh.se

Abstract

The increased availability of modern embedded many-core architectures supporting floating-point operations in hardware makes them interesting targets in traditional high performance computing areas as well. In this paper, the Lattice Boltzmann Method (LBM) from the domain of Computational Fluid Dynamics (CFD) is evaluated on Adapteva's Epiphany many-core architecture.

Although the LBM implementation shows very good scalability and high floating-point efficiency in the lattice computations, current Epiphany hardware does not provide adequate amounts of either local memory or external memory bandwidth to provide a good foundation for simulation of the large problems commonly encountered in real CFD applications.

Keywords: Many-core, Epiphany, Computational Fluid Dynamics, Lattice Boltzmann

1 Introduction

In many scientific problems, from designing a boat engine to predicting tomorrow's weather, there is a need to compute the dynamics of liquids and gases. These complex simulations, known as computational fluid dynamics (CFD), amount to huge amounts of computing time if good accuracy needs to be achieved. There are examples where the simulation of a few turns of a propeller can take thousands of CPU hours on powerful workstations. In order to significantly reduce the simulation time for CFD, utilizing parallel processing will be of paramount importance.

A relatively recent method in CFD is the Lattice Boltzmann Method (LBM), which is well suited for parallelization, due to its inherently local structure [8].

Adapteva's Epiphany platform [5] is a modern, low-power many-core architecture supporting floating-point operations in hardware. While the current generation of chips contain 16 or 64 cores, the architecture is designed to scale up to thousands of cores on a single chip. Its hardware support for floating-point calculations makes it a possible choice for many scientific settings, and recent performance studies are promising (cf. [9]).

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2015

doi:10.1016/j.procs.2015.05.348

In this paper we will evaluate the potential of using the Epiphany architecture for CFD simulations using the LBM. The lid cavity test case [1] will be used as a test case and the LBM has been implemented [6] and run to the Epiphany E16G3 chip, as found on the Parallella board [5].

The organization of this paper is that we first will describe the LBM and the Epiphany Architecture. Then follows a description of the implementation and the benchmark we have used to evaluate the suitability of the architecture for LBM. In section 6 we then describe our evaluation and results. After this we conclude the paper.

2 Lattice Boltzmann Method

The Lattice Boltzmann Method (LBM) has become an alternative to traditional algorithms for fluid flow simulations, and is able to simulate a wide range of behaviours in both single and multiphase fluids [8]. Its particle-based nature makes it easily parallelizable and stems from the Boltzmann equation

$$\left(\frac{\partial f}{\partial t}\right) = \left.\frac{\partial f}{\partial t}\right|_{\text{coll.}} + \left.\frac{\partial f}{\partial t}\right|_{\text{stream}} + \left.\frac{\partial f}{\partial t}\right|_{\text{extern}}, \tag{1}$$

in which the right-side terms denote the effects caused by collision, streaming or external forces on the particle density function $f = f(\vec{x}, \vec{v}, t)$. While the collision and external terms account for internal (particle-particle) and external (particle-environment) forces, respectively, the streaming term accounts for the diffusion of particles.

The BGK (Bhatnagar-Gross-Krook) operator treats collisions as relaxations towards a local equilibrium f^{eq} with a single relaxation time τ, which depends on the viscosity of the simulated fluid. After discretization in both time and space, and neglecting external forces, the fluid node update function

$$f_i(\vec{x} + \vec{e}_i\delta t, t + \delta t) = f_i(\vec{x}, t) - \frac{1}{\tau}\left[f_i(\vec{x}, t) - f_i^{eq}(\rho, \vec{u})\right], i = 0..n, \tag{2}$$

in which f_i describes the fraction of particles at time t found at position $\vec{x}$ moving with the microscopic velocity $\vec{e}_i$, can be formulated. The variables ρ and $\vec{u}$ denote the local macroscopic density of the fluid and its velocity, respectively.

Different lattice models can be employed, which follow the scheme $DmQn$, with m describing the number of dimensions, and n the number of discrete velocities e_i. For this work, the common $D2Q9$ and $D3Q19$ models have been chosen.

Once per iteration, all fluid nodes are updated in a two-step process:

- *Collision*, i.e. calculating the local macroscopic flow quantities ρ and $\vec{u}$, and then executing the relaxation, and
- *Streaming*, or propagating the new values to the neighboring cells according to the direction of e_i.

While the first step is computationally intensive, it only involves the local fluid node. The second step is just a data transfer between a fluid node and all of its logical n neighbors. Boundary conditions may be treated as a modified collision and/or streaming step.

3 Adapteva Epiphany

The Epiphany architecture [5] is a scalable, two dimensional mesh of computing nodes (Fig. 1). It operates in a shared, flat 32-bit address space, in which each node uses a continuous 1 MiB large block. Nodes are addressed by their row and column numbers (6 bits each), although the special node address (0,0) always refers to the local node.

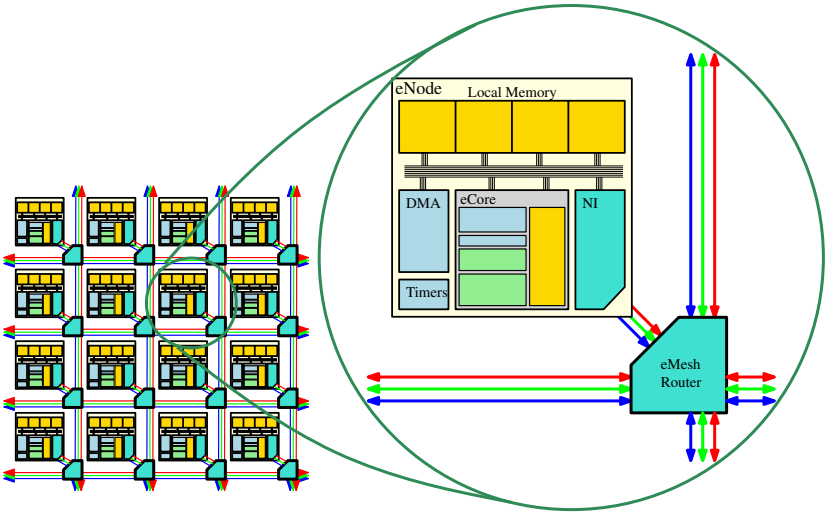


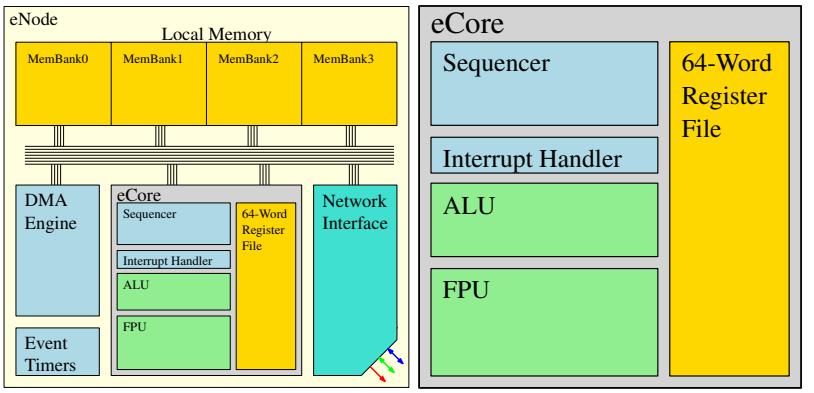
Figure 1: Epiphany mesh fabric

The mesh itself is made of three meshes with different purposes, called *rMesh*, *cMesh* and *xMesh*. The former is used exclusively for read requests, while the latter two carry write transactions destined for on-chip and off-chip nodes, respectively. To the application, these are indistinguishable, apart from bandwidth and latency differences. Routing follows static rules and happens along the axes, first by row, then by column address. For multicast traffic, a different routing scheme can be selected.

Writes are heavily favoured over reads, since reading a foreign address involves sending a read request and waiting for the answer to arrive. On the other hand, writes are of a fire-and-forget type, allowing the node to continue processing while the data will eventually reach its destination. For performance reasons, the memory-order model is weakened, so that the order of memory transactions is non-deterministic for write-after-read or write-after-write transactions when foreign nodes are involved. Transactions inside the local node always follow a strong-order memory model, however.

Each node (called eNode) contains an eCore processor, 32 KiB of high-speed local memory, a two-channel DMA controller, and a mesh controller (Fig. 2a), and every eCore consists of an in-order, dual-issue 32-bit RISC CPU including both an IEEE754 compatible floating-point unit (FPU), an integer arithmetic-logic unit (IALU), a 64-word register file and an interrupt controller (Fig. 2b). Two event timers allow measuring execution times with clock-cycle granularity as well as different kinds of pipeline stalls.

The 32 KiB of local memory in each node are divided into four independent banks of 8 KiB each, of which the first bank contains the interrupt vector table and is therefore suitable for the code. In this study we will use the remaining three banks (24 KiB total) for data.



(a) eNode (b) eCore

Figure 2: Epiphany components

This work used the open Parallella platform, which was started in 2012 as a Kickstarter project and is commercially available from Adapteva. The Parallella board itself is a fully open-source credit-card sized computer containing a Xilinx Zynq 7010 or 7020, an Epiphany E16G3 and 1 GiB of RAM. The Xilinx Zynq is a System-on-Chip with two ARM Cortex-A9 processor cores and some reconfigurable logic, and is fully supported by Linux. The board also contains 1 GBit-Ethernet, USB and HDMI interfaces, can boot from a MicroSD card and is able to run the Epiphany SDK. The E16G3 chip is a 16-core implementation of the Epiphany-III architecture running at 600 MHz on the Parallella.

Adapteva's e-Link interface is implemented using FPGA logic and is used to exchange data between the ARM cores and the Epiphany. By default, a 32 MiB block of memory is shared between both systems. On the Parallella board, the 4x4 grid of processor nodes uses the coordinates between (32,8) and (35,11) inclusive. Shared memory starts at address `0x8e000000`, which translates to mesh coordinates (35,32) and (36,0). Accesses to shared memory are routed to these coordinates and leave the chip on the EAST e-Link interface, which is connected to the Zynq's FPGA logic and thus, indirectly, to the host memory. However, using this interface is significantly slower than the internal on-chip communication pathes.

4 Implementation

Central to the Lattice Boltzmann Method is the lattice, which contains the simulation state of the fluid in the domain. Only a single copy of the lattice is stored in local memory. Once per time-step, the whole lattice is updated in-place by utilizing the "swap trick" [4]. For some or all iterations, the whole lattice is then copied to shared memory.

The only way to get data in or out of the Epiphany chip on the Parallella board is via the shared memory. Since accessing it is at least one order of magnitude slower than accessing local memory, we have decided to map the lattice directly to the local memory of the cores. However, as we only have 24 KiB of data memory available in each core, this major decision severely restricts the lattice sizes we will be able to simulate.

The lattice itself will be divided into *blocks*, where each block is stored in and processed by a single processing node only. In a two-dimensional simulation, blocks are rectangular, and mapping them to the two-dimensional Epiphany mesh is straight-forward. A similar approach was chosen for three-dimensional simulations, where the then cuboid-shaped blocks are distributed in two dimensions only (Fig. 3).

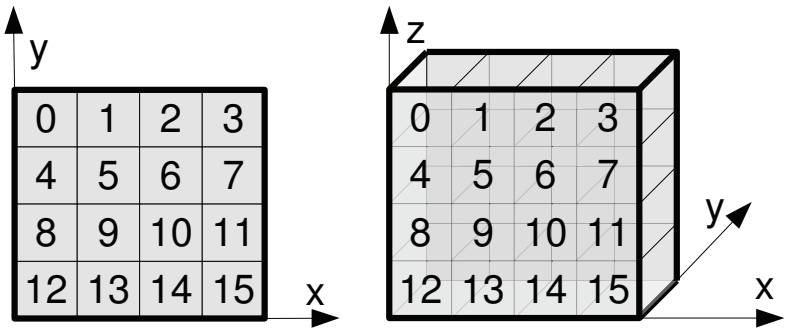


Figure 3: Mapping of blocks in 2D (left) and 3D (right), outer boundaries drawn bold

Each block contains multiple lattice nodes, which are classified as *bulk* nodes, if all of their neighbors are part of the same block. Otherwise, nodes are either part of the *inner boundary*, if their neighbors are still part of the simulation domain, or else, *outer boundary*. Even if the latter requires special treatment, it is only the nodes of the inner boundary that require communication between different mesh nodes. Fortunately, this communication will be next-neighbor only.

The outer boundaries describe the behaviour of the fluid on the simulation domain boundary. By specifying the behaviour of these fluid nodes (e.g. imprinting a velocity), solving a linear equation system allows simulating physically reasonable boundaries. This work uses the boundary conditions described by Zou/He [11].

Due to the limited amount of local memory, the achieveable lattice sizes are very small (see Table 1). In 3D, two block sizes were studied: While 7x6x7 blocks are almost cube-shaped, the 3x35x3 extend maximally in the non-distributed dimension. Still, due to the small block sizes, there is little advantage in using more complicated mapping schemes. For the 2D benchmarks, block sizes of 24x24 (instead of the maximum 26x26) have been used.

lattice model	**node size (bytes)**	**max. nodes per block**	**maximum block size**	**lattice size (16 cores)**
$D2Q9$	36	682	26x26	104x104
$D3Q19$	76	323	7x6x7	28x6x28
$D3Q19$	76	323	3x35x3	12x35x12

Table 1: Achieveable lattice sizes (E16G3)

While it is possible to achieve larger lattices by storing it in shared memory and update it in a streaming fashion, the limited external memory bandwidth makes this approach infeasible. An alternative solution would be to combine multiple Epiphany chips to increase the number of cores in a single system. Doing so would still require using slower off-chip communication, but keep the advantage of using the fast local memory.

The *collision-streaming* approach is a straight implementation of the two-step process described earlier. Each core needs to access all nodes in its block twice (once per step), since the computation and communication steps of the algorithm are separated.

It is possible to combine both steps, as long as all nodes are processed in a specific order (cf. [4]). However, due to the parallel processing of different sub-lattices, this order can only be guaranteed inside each block, requiring the inner boundaries to be handled separately.

The *boundary-bulk* approach handles the boundary nodes similarly to the collision-streaming approach, but uses additional code to combine both steps for the bulk of each block. Consequently, this optimization benefits from larger block sizes, trading iteration throughput for lattice node throughput.

All active cores in the Epiphany mesh always run in lockstep, being synchronized with barriers when needed. In local memory, only a single lattice version can be stored, requiring the cores to wait while the lattice is copied to shared memory in order to get a consistent iteration snapshot. Floating-point calculations are done with single-precision.

To provide intermediate simulation states to the host, the lattice is copied to shared memory after some (or all) iterations.

5 Benchmark

To validate the implementation, the lid cavity test case (see e.g. [1]) has been simulated in two and three dimensions. The chosen boundary conditions force fluid nodes attached to a wall to always move at the same speed as the wall itself (no-slip walls), which is required for the lid cavity test case.

The simulation domain consists of a box with zero-speed walls on all sides except one (top) and starts with a fluid of constant density. The top wall moves at a constant velocity, dragging the fluid along. Since the fluid cannot leave the box through the right-side wall, it starts to move downwards, forming a circular stream over time, finally reaching a steady state. Fig. 4 shows the normalized absolute velocity of a two-dimensional 104x104 simulation after 150 (left) and 1000 (right) time steps. Higher velocities are shown with higher intensity.

Figure 4: Velocity field of an example 2D lid cavity simulation, iterations 150 (left) and 1000 (right)

For real applications, much larger lattice sizes are required. While our implementation achieves three-dimensional lattice sizes of 12x35x12, studies on porous media have shown to require at least 128x128x1024 lattices [3]. One of the SPEC2006 floating-point benchmarks (470.lbm) uses the Lattice Boltzmann Method on a 150x150x150 lattice (cf. [7]).

6 Evaluation and results

In the 2D case, the boundary-bulk approach is more than twice as fast as the collision-streaming approach, reaching 45 MLU/s (Millions of Lattice-node Updates per second) at a 700 MHz clock rate.

On the other hand, the 3D case does not benefit from it at all, due to the much smaller block sizes (Fig. 5) possible. Additionally, it was not possible to compile the 3D code at the highest optimization level without breaking the 8 KiB code size limit. Together with the more complicated boundary conditions and more involved calculations, the 3D performance reaches at most 5.4 MLU/s with a block size of 7x6x7 using the boundary-bulk approach.

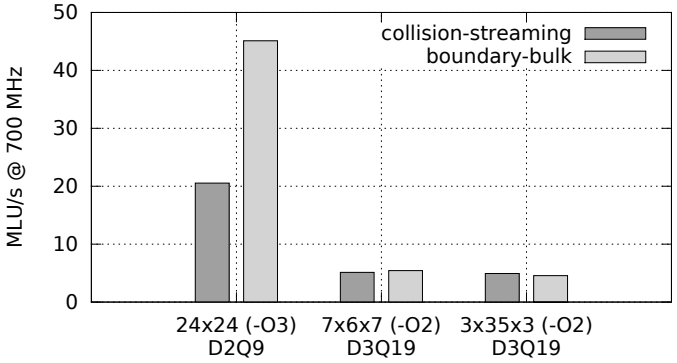


Figure 5: Node update rates (16 cores)

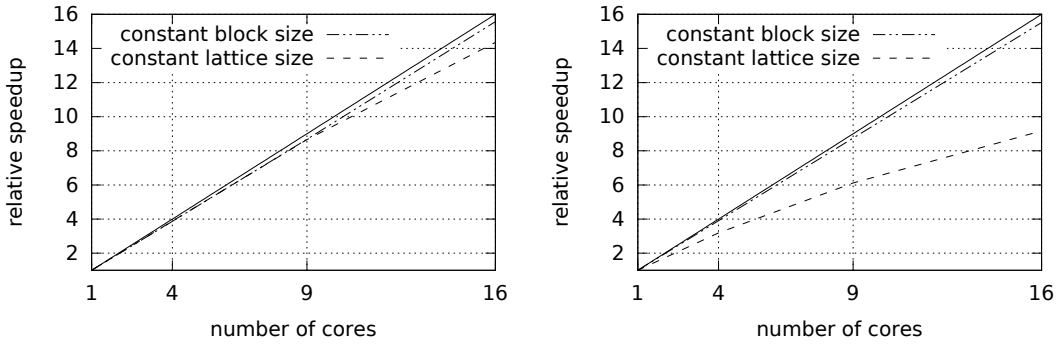
Current CPU generations have been shown to perform more than 110 MLU/s on larger, three-dimensional lattices using double-precision arithmetic. Also, using single-precision arithmetic, GPUs have been able to achieve up to 650 MLU/s. Since these system differ widely, simply comparing the throughput does not lead far. As table 2 shows, the Epiphany architecture is very much competitive in terms of power efficiency, even though the Lattice Boltzmann Method by itself is a bad fit to the architecture due to its large working set.

system type	**clock freq in MHz**	**number of FPUs**	**power in W**	**MLU/s total**	**MLU/s per FPU**	**MLU/s per W**
Epiphany-III	700	16	2	5.4	0.34	2.7
Tesla C2070 [2]	1150	448	238	650	1.45	2.73
Xeon E5-2660 [10]	2200	10	95	110	11.0	1.16

Table 2: Throughput comparison of different systems

The scalability was analyzed in 2D both for a constant 24x24 lattice size (using block sizes of 24x24, 12x12, 8x8, and 6x6 as the number of cores increased), or for a constant 24x24 block size (leading to lattice sizes of 24x24, 48x48, 72x72, and 96x96 respectively).

The collision-streaming approach scales almost linearly with the number of cores used (Fig. 6a). Since smaller block sizes contain a larger percentage of boundary nodes, the increased communication requirements lead to a slightly decreased speedup in that case. In contrast, the boundary-bulk approach suffers much more from smaller block sizes (Fig. 6b), losing its speed advantage as blocks start to only consist of boundary nodes.



(a) collision-streaming (b) boundary-bulk

Figure 6: Relative speedup, 2D case

The two-dimensional collision step by itself requires in average 285 clock cycles per node, in which 135 IALU- and 73 FPU-instructions are executed [6]. Thus, only 26% of collision clock cycles are spent executing FPU-instructions (73% for both IALU- and FPU-instructions). Assuming perfect scheduling while simultaneously hiding the streaming step in dual-issue cycles, the node update rate could at most be quadrupled.

Since we put the code to the first memory bank only, code size is limited to slightly below 8 KiB. Compiling the three-dimensional boundary-bulk code at the highest optimization level exceeds that limit. For this reason, the 3D code was compiled at the `-O2` level only.

For most CFD applications, at least some intermediate results of a simulation are important, requiring the lattice to be sent to the host application. This is done through shared memory, and unfortunately, accesses to it are very slow. For a two-dimensional 24x24 lattice (20,736 bytes), the time to copy the lattice accounts for more than 90% of the iteration clock cycles. In average, it takes about 8 clock cycles per byte to write the lattice to shared memory, which translates to about 75 MiB/s, or one eighth of the theoretical maximum throughput of 600 MiB/s. It has, however, been shown that at least 150 MiB/s are achieveable on the Parallella platform (cf. [9]).

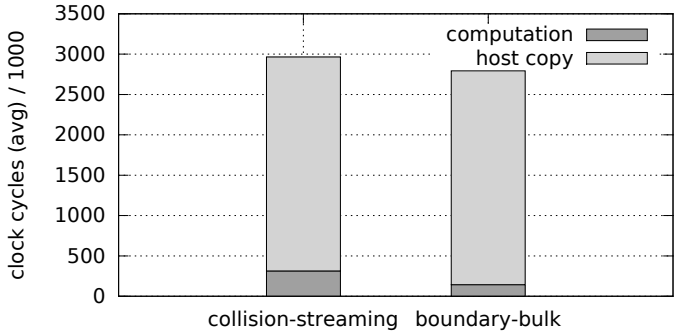


Figure 7: Host copy overhead

Copying the lattice to shared memory happens concurrently on all cores, and blocks until all cores have finished to provide a consistent iteration result. As shown in Fig. 8, network congestion leads to a non-uniform distribution of the shared memory bandwidth. While the total bandwidth does not change, most rows suffer from starvation (cf. [9]). In our case, the bottom row, which is connected with the shortest distance to shared memory, is least affected.

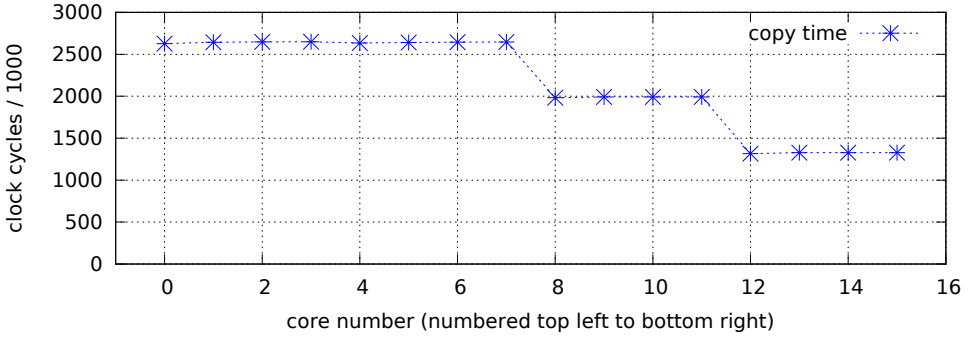


Figure 8: Per-core host copy time

In all tests, the size of the lattice is hard limited by the assigned 24 KiB of local per-core memory, which is too small for almost all CFD applications. Since it is impossible to store larger lattices inside the Epiphany chip, extending the lattice is only possible through the external interface, or by extending the mesh fabric with additional Epiphany chips.

Although a multi-chip solution has yet to be studied, adding additional chips requires some of the inner boundary data to use the slower off-chip communication pathes. Since barriers are used to synchronize all cores, the resulting slowdown would be felt on all cores. On the other hand, due to the next-neighbor communications, the slowdown should also be limited such that adding chips to a multi-chip solution should not slow the system down any further.

However, extending the lattice with additional chips does not increase the external memory bandwidth, increasing the overhead for copying the lattice to the host.

7 Conclusion

In this paper we have studied to feasibility to run Computational Fluid Dynamics (CFD) simulations on Adapteva's many-core architecture Epiphany. The easily parallelizable Lattice Boltzmann Method (LBM) has been implemented and evaluated using a basic lid cavity benchmark.

Due to the low bandwidth to external memory the only reasonable mapping of LBM lattices is to the local memory in the cores. However due to the small local memory (32 KiB) we can not fit more than a very small number of lattice nodes per core (for the 2D case we managed to have 24x24 lattice nodes per core). As we have only used one chip, with 16 cores, the overall area or volume one currently can simulate is very restricted.

Still, the potential of the Epiphany core for doing LBM calculations can be seen in the respectable 45 MLU/s (Millions of Lattice-node Updates per second) while consuming less than 2 W for the 2D case. Unfortunately, for the 3D case we find a significant reduction in performance due to the contained local memory. Nevertheless, as the LBM algorithm only exhibits

nearest-neighbor communication we see an almost linear scaling in performance with the number of cores.

However, as soon as we need to simulate a problem than is larger than what can be fit into one chip (which is true for any real world simulation), the performance will be reduced when using a multi-chip solution. Using the external memory to store the lattice will reduce the performance significantly.

That is, in order for the Epiphany architecture to efficiently run LBM on real world sized problems one needs to both increase the local memory size as well as introduce a more efficient external memory access scheme.

Acknowledgement

This work was supported by the ESCHER project funded by the Swedish Knowledge foundation, and Volvo Penta AB, Gothenburg, Sweden. Additionally, the authors would like to thank Jonas Latt from the University of Geneva, and Lars Johansson from Volvo Penta AB for their very helpful discussions and comments.

References

[1] Charles-Henri Bruneau and Mazen Saad. The 2D lid-driven cavity problem revisited. *Computers & Fluids*, 35(3):326–348, 2006.

[2] Johannes Habich, Christian Feichtinger, Harald Köstler, Georg Hager, and Gerhard Wellein. Performance engineering for the lattice boltzmann method on gpgpus: Architectural requirements and performance results. *Computers & Fluids*, 80:276–282, 2013.

[3] Jens Harting, Maddalena Venturoli, and Peter V Coveney. Large-scale grid-enabled lattice boltzmann simulations of complex fluid flow in porous media and under shear. *PHILOSOPHICAL TRANSACTIONS-ROYAL SOCIETY OF LONDON SERIES A MATHEMATICAL PHYSICAL AND ENGINEERING SCIENCES*, 362:1703–1722, 2004.

[4] Jonas Latt. Technical report: How to implement your DdQq dynamics with only q variables per node (instead of 2q). Technical report, Tufts University, 2007.

[5] Andreas Olofsson, Tomas Nordström, and Zain Ul-Abdin. Kickstarting High-performance Energy-efficient Manycore Architectures with Epiphany. 48th Asilomar Conference on Signals, Pacific Grove, CA, Nov. 2-5, 2014.

[6] Sebastian Raase. *Exploring the Epiphany manycore architecture for the Lattice Boltzmann algorithm.* Master thesis, Halmstad University, 2014.

[7] Standard Performance Evaluation Corporation. CFP2006.

[8] Michael Sukop. *Lattice Boltzmann modeling an introduction for geoscientists and engineers.* Springer, Berlin New York, 2006.

[9] Anish Varghese, Bob Edwards, Gaurav Mitra, and Alistair P. Rendell. Programming the Adapteva Epiphany 64-Core Network-on-Chip Coprocessor. In *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pages 984–992, 2014.

[10] M Wittmann, T Zeiser, G Hager, and G Wellein. Modeling and analyzing performance for highly optimized propagation steps of the lattice boltzmann method on sparse lattices. *arXiv preprint arXiv:1410.0412*, 2014.

[11] Qisu Zou and Xiaoyi He. On pressure and velocity boundary conditions for the lattice Boltzmann BGK model. *Physics of Fluids (1994-present)*, 9(6):1591–1598, 1997.