Accelerating LBM and LQCD Application Kernels by In-Memory Processing

Paul F. Baumeister[1], Hans Boettiger[2], José R. Brunheroto[3], Thorsten Hater[1](✉), Thilo Maurer[2], Andrea Nobile[4], and Dirk Pleiter[1]

[1] Jülich Supercomputing Centre, Forschungszentrum Jülich, 52425 Jülich, Germany
t.hater@fz-juelich.de
[2] IBM Deutschland Research and Development GmbH, 71032 Böblingen, Germany
[3] IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA
[4] Institute for Advanced Simulation, Forschungszentrum Jülich, 52425 Jülich, Germany

Abstract. Processing-in-memory architectures promise increased computing performance at decreased costs in energy, as the physical proximity of the compute pipelines to the data store eliminates overheads for data transport. We assess the overall performance impact using a recently introduced architecture of that type, called the Active Memory Cube, for two representative scientific applications. Precise performance results for performance critical kernels are obtained using cycle-accurate simulations. We provide an overall performance estimate using performance models.

1 Introduction

With increasing complexity the performance of modern HPC architectures becomes increasingly communication limited. The resulting need for improving efficiency of data transport and eliminating the associated overheads in terms of time and energy can be addressed by integrating compute pipelines close to the data stores, which is achieved, e.g., by processing-in-memory (PIM) designs. For various reasons none of the proposed PIM designs have resulted in products which could be integrated into HPC architectures. There are good arguments for expecting PIMs to enter system roadmaps, see [2] for an extended discussion. Besides a growing need for such approaches there are new opportunities for realizing such architectures due to novel 3-dimensional stacking technologies.

In this paper we assume that PIM technologies will become part of future massively-parallel HPC architectures to address the following question: How efficiently could such an architecture be exploited by relevant scientific applications?

For our work we selected two representative applications, namely computational fluid-dynamics calculations based on the Lattice Boltzmann Method and simulations of the theory of strong interactions, i.e. Lattice Quantum Chromodynamics. We have implemented performance relevant kernels on the IBM Research Active Memory Cube (AMC) architecture [14] to allow for cycle-accurate simulations. Performance models are used to assess the overall performance for different proxy architectures. This approach allows us to make the following contributions:

J.M. Kunkel and T. Ludwig (Eds.): ISC High Performance 2015, LNCS 9137, pp. 96–112, 2015.
DOI: 10.1007/978-3-319-20119-1_8

- We present and discuss our implementation of applications on AMC and analyse the observed performance.
- Based on this performance analysis we conduct an architectural evaluation of AMC for the considered application kernels.
- To assess the overall performance based on performance models for different variants of a proxy architecture.
- Finally, we report on feedback on future applications' roadmaps which we initiated to assess future application needs.

In Sects. 2 and 3 we provide an overview on the AMC architecture and the applications considered for this paper. We continue with a presentation of details of application kernels on AMC and performance analysis results in Sects. 4 and 5, respectively. In Sect. 6 we explore features and parameters of the AMC architecture and its suitability for the given application kernels. Finally in Sect. 7, we introduce a performance model approach for assessing the overall performance for HPC architectures that integrate AMCs.

2 Active Memory Cubes

Recently a new, 3-dimensional packaged memory architecture, Hybrid Memory Cube (HMC) [11], was introduced, where a set of memory dies are stacked on top of a logic die. In the HMC architecture the logic die implements a memory controller and a network interface via which the processor can access the memory. In the IBM Research Active Memory Cube (AMC) [14] architecture 32 computational lanes are added to the logic die, which also have access to the memory. The most important hardware performance parameters are listed in Table 1.

Each lane is composed of a control unit and 4 computation slices, which each comprise a memory-access and an arithmetic pipeline. Each slice contains multiple register files which include 32 scalar plus 16 vector computation registers. The latter are composed of 32 elements. All registers are 64-bit wide. Overall, AMC enabled nodes offer a significant amount of parallelism, comparable to GPU-accelerated architectures. Nodes are expected to contain multiple AMCs, in the order of sixteen devices. Each AMC possesses 32 lanes, multiplied by four slices.

The computational lanes are micro-coded and each can process in every clock cycle one Very Long Instruction Word (VLIW), mapping to 1 Control + $4 \cdot (1 \text{ ALU} + 1 \text{ LSU})$ instructions for a total of nine. To reduce the number of VLIWs a static or dynamic repeat counter can be specified following a temporal single-instruction-multiple-data (SIMD) paradigm. Instructions can be repeated up to 32 times and access, e.g., during each iteration consecutive elements of vector registers. The VLIWs are read from a buffer which can hold up to 512 entries.

The arithmetic pipelines take 64-bit input operands and can complete in each clock cycle one double-precision or a 2-way SIMD single-precision multiply-add.

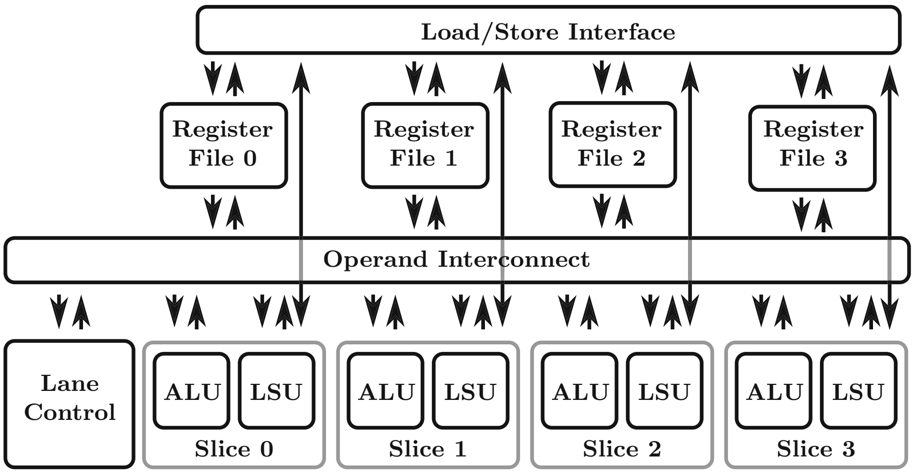

Fig. 1. Sketch of the AMC lane architecture. For more details see [14].

Table 1. Summary of AMC performance characteristics.

Clock frequency	$f = 1.25\,\text{GHz}$
Floating-point performance per lane	$B_{\text{fp}} = 8\,\text{Flop/cycle} = 10\,\text{GFlop/s}$
Memory bandwidth per lane	$B_{\text{mem}} = 8\,\text{Byte/cycle} = 10\,\text{GByte/s}$
Memory capacity per lane	$C_{\text{mem}} = 0.25\,\text{GByte}$
Number of lanes per AMC	$N_{\text{lane}} = 32$

Thus up to 8 double-precision floating-point operations can be completed per clock cycle and lane. The peak performance of one AMC running at a clock speed of 1.25 GHz is 320 GFlop/s. As one slice can read the vector registers of the other slices, double-precision complex arithmetic can be implemented without the need for data re-ordering instructions. The same holds for single-precision complex arithmetic thanks to special 2-way SIMD instructions.

Like in the HMC architecture the memory is organized into vaults. Each of the 32 vaults in the AMC architecture is functionally and operationally independent. Each vault has a memory controller (also called vault controller) which like the lanes is attached to the internal interconnect. Load/store requests issued by a lane are buffered in the lane's load-store queue which has 192 entries. A lane can load or store 8 Byte/cycle from or to the internal interconnect, i.e. the ratio of floating-point performance versus memory bandwidth is 1 and thus significantly larger than in typical processor architectures. The memory access latency depends on the distance between lane and vault as well as on whether the data is already buffered by the vault controller. The minimum access latency is 24 cycles, i.e. load operations have to be carefully scheduled to avoid stalls due to data dependencies.

Each AMC features a network interface with a bandwidth of 32 GByte/s to connect it to a processor and multiple devices may be chained to share a link to the processor. The execution model foresees main programs to be executed on the processor with computational lanes being used for off-loading small kernels. Assuming a design based on 14 nm technology, the power consumption is expected to be around 10 W.

Architectures based on AMC would feature multiple levels of parallelism with 4 slices per lane, 32 lanes per AMC, N_{AMC} AMCs per processor and possibly multiple processors per node and many nodes per system.

3 Applications and Application Roadmaps

3.1 Lattice Boltzmann Method

The Lattice Boltzmann method (LBM) is an approach to computational fluid dynamics based on a discrete approximation to the Boltzmann equation in terms of the distribution functions f_α in phase space

$$f_\alpha(\boldsymbol{x} + \boldsymbol{c}_\alpha \Delta t, t + \Delta t) = f_\alpha(\boldsymbol{x}, t) + \omega\,(f_\alpha(\boldsymbol{x}, t) - f_\alpha^{\mathrm{eq}}). \tag{1}$$

Here, spatial position $\boldsymbol{x}$, velocity $\boldsymbol{c_\alpha}$ and time t only take discrete values. The update process can be implemented in two separate steps: First, the local update on the right hand side, describing changes in the distribution due to particle interactions (*collide*). Second, distributing the update to adjacent sites according to a stencil with a range of three sites, equivalent to the fluid advection (*propagate*).

There are various LBM models which differ in number of spatial dimensions and velocities. In this work we considered an application which implements the *D2Q37* model , utilizes 37 populations in two spatial dimensions [17]. Parallelisation of the application is based on a domain decomposition in 1 or 2 dimensions. For typical runs and architectures the time spent in the collide and propagate kernel corresponds to about 90 % and 10 % of the overall execution time, respectively. The propagate step involves communication between tasks processing adjacent domains to update a halo. Communication can typically be overlapped with updating the bulk region. The collide step requires 5784 floating-point operations per site at $37 \cdot 8$ Bytes of input and output, whereas propagate does not involve any computation. Given this high arithmetic intensity we expect the collide step to be compute performance limited on the AMC architecture, while the propagate step will be limited by the memory bandwidth on any architecture. This split into two central tasks with distinct performance characteristics makes D2Q37 an interesting choice for evaluating the AMC architecture.

Future Requirements. The D2Q37 model is used to study fluid turbulence using lattices of size L^2 with $L = \mathcal{O}(1000 \ldots 10000)$. These studies will have to be extended to 3 dimensions. Studies on lattices of size $L^3 = 4000^3$ are expected to require 64 days of computing time on a machine with 100 PFlop/s peak performance assuming an efficiency of about 20 % [18]. The number of floating-point operations, which have to be executed per site and update, would increase from about 6000 to about 50000, while the memory footprint per site increases by a factor of about 3.

3.2 Lattice QCD

Quantum Chromodynamics (QCD) is a theory for describing strong interactions, e.g. the interaction between key constituents of matter like quarks. Lattice QCD (LQCD) refers to the formulation of QCD on a 4-dimensional discrete lattice. LQCD opens the path for numerical simulations of QCD.

The computational challenge in LQCD computations consists of repeatedly solving very large sparse linear systems. Iterative solver algorithms result in a very large number of multiplications of a matrix D and a vector ψ, which thus becomes the most performance critical computational tasks. In this paper we focus on a particular formulation of LQCD with so-called Wilson-Dirac fermions where this task takes the following form:

$$D\psi(\boldsymbol{x}) = m\psi(\boldsymbol{x}) + \sum_{\pm\mu}(1 \pm \hat{\gamma}^{\mu}) \otimes U_{\mu}^{\pm\dagger}(\boldsymbol{x})\psi(\boldsymbol{x} \pm \hat{\mu}). \qquad (2)$$

The index μ runs over all four dimensions of space-time.

Modern solvers (see, e.g., [8]) allow for a mixed precision approach where majority of the floating-point operations can be performed in single-precision, without compromising on the double-precision solution.

The operator D consists of a real, scalar on-site term m and eight links towards the Cartesian nearest-neighbour lattice sites in the four-dimensional discretised space-time. Spinor fields ψ are represented by one element per lattice site, where each element lives in the product space of 3 colours and 2×2 spins and thus comprises 12 complex numbers. Gauge fields U are represented by vectors with one element per lattice link, where each element acts as a SU(3)-rotation onto the colour-space. These are typically represented by a 3×3 complex matrix, however, the full information can be encoded in just 8 real numbers. The spin space is invariant with respect to the link rotations, but sensitive to the link-direction.

The spin projection operators $(1 \pm \hat{\gamma}^{\mu})$ commute with the link matrices U due to operating on different dimensions. A closer look at the 4×4 operators $(1 \pm \hat{\gamma}^{\mu})$ reveals that their matrix representation is of rank 2 rather than rank 4. This enables us to separate the operators into two parts, henceforth called spin deflation $P_{\pm\mu} \in \mathbb{C}^{2\times4}$ and spin inflation $J_{\pm\mu} \in \mathbb{C}^{4\times2}$. This reduces the number of necessary operations by performing the SU(3)-rotations in the deflated subspace. The computational task now takes the following form:

$$D\psi(\boldsymbol{x}) = m\psi(\boldsymbol{x}) + \sum_{\pm\mu} J_{\pm\mu}U_{\mu}^{\pm\dagger}(\boldsymbol{x})P_{\pm\mu}\psi(\boldsymbol{x} \pm \hat{\mu}) \qquad (3)$$

Usually, data parallelism is exploited by decomposing the lattice into cubic subdomains in a one to four dimensional scheme. The main source of communication

Table 2. Implementation characteristics of the three kernels per lattice site.Shown are the volumes of data loaded and stored, the number of required floating point operations (FP) and the percentage of operations which could be mapped to multiply-add type instructions (MA). We further give the resulting arithmetic intensity (AI) and the number of VLIWs in the kernel.

	Load (Byte)	Store (Byte)	FP (% MA)	AI	VLIWs
LQCD	1152	480	1368 (89 %)	0.8	63
LBM collide	765	296	5784 (80 %)	5.5	163
LBM propagate	296	296	-	0.0	26

is the surface exchange with the nearest neighbours in the respective dimensions. Therefore an appropriate torus topology is preferred. In order to minimise the impact of the communication, the operator computation is split into the forward and backward directions. This allows for transferring partial results of the latter and meanwhile computing the former, thus overlapping computation and communication. Then the procedure is reversed and the final result is computed.

Future Requirements. The performance of future algorithms are expected to be dominated by similar computational tasks as considered in this paper. With more computational resources becoming available, the lattice sizes are expected to grow during the next couple of years to $L^3 \cdot T = 128^3 \cdot 256$ [13]. Currently, even larger lattices are only expected to be required for relatively special cases.

4 Implementation on AMC

We describe our implementations of the collide and propagate kernels of the D2Q37 LBM code and the Dirac operator from the LQCD code. As the experiments were performed at an early stage of the development of this technology, neither compilers for high-level languages nor assemblers were available. Consequently, the implementation was done directly at the level of AMC micro-code supported by a set of ad-hoc tools. Thus programming involved scheduling of instruction and their allocation to slices as well as the allocation of registers.

The focus of the dicussion lies on the features of the instruction set and AMC architecture that support the implementation, rather than providing a full overview on our implementations.

4.1 LBM D2Q37 Model

Collide. For this kernel, we implemented a vectorised version over 32 elements, using the temporal SIMD paradigm, but neglected the traversal of the lattice. For a full version, this infrastructure would have to be added. An efficient, static work sharing scheme, based on partitioning the lattice, can be applied. In this case, we did not use the four slices for additional parallelisation, in order to avoid further constraint on the input. The operation is divided in four sub-steps:

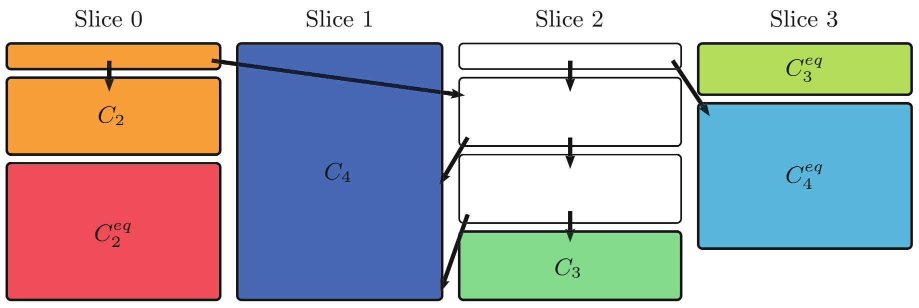

Fig. 2. Distribution of collide central loop body over the slices. Similar colours match coefficients of f and f^{eq}, white signifies auxiliary terms.

computing bulk properties, applying a non-linear transformation, computing the update on the transform and, finally, reverting to the original representation. The last three steps require the computation of six terms of differing length. These computations are distributed carefully over the four slices, resulting in a very dense packing where 92 % of the instructions are floating point computations. The six coefficients are computed in pairs matched for even length in three of the slices, while the fourth supplies intermediate terms. Common sub-terms are preferably computed in this auxiliary slice, which further increases efficiency.

Propagate. We implemented the full kernel, including traversal of the lattice, loading from the input array and storing to a different output array. Exploiting our freedom in choosing an optimal data layout, we sub-divide the lattice into rows and distribute those evenly over the participating lanes. Each lane iterates over the assigned rows handing one quarter of the row to each of its slices.

The capabilities of the LSU instructions are exploited to map the requirements to efficient code. We load the local site in linear fashion, using two VLIWs to capture the 37 populations. The LSU is capable of automatically incrementing the source address. Further, the store instruction is supplied with a vector register holding the stencil and applies this to the relevant elements of the stored data. Thus, the loop body of the kernel consist of just five VLIWs.

4.2 LQCD

In order to hide communication required when parallelising the matrix-vector multiplication, the computation of the forward and backward part is overlapped as described in Sect. 3.2. We re-produce this structure to keep our implementation close to the original memory access pattern, but do not implement inter-task communication. Further, we simplify the kernel by only mocking up the indirection infrastructure for storing the results. Therefore, the simulation results have to be understood as being approximative.

We took advantage of the temporal SIMD capabilities of the AMC by vectorising over eight lattice sites, utilising 24 out of 32 vector register elements.

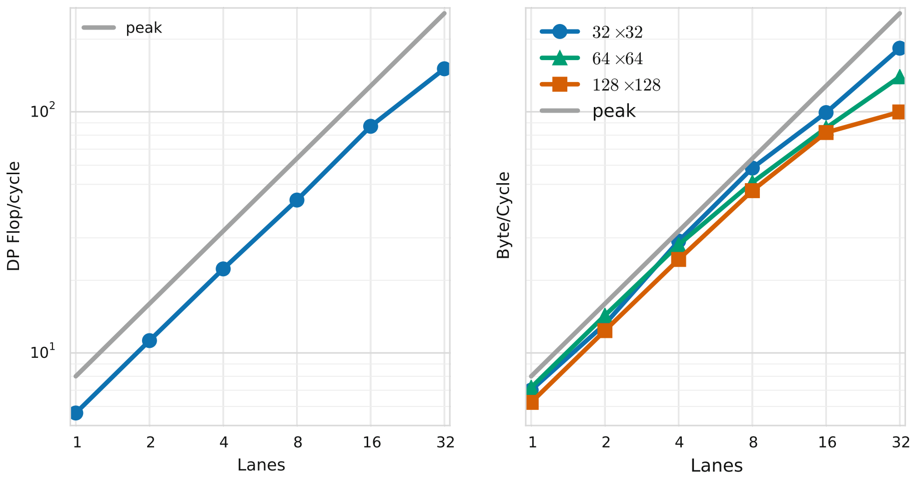

Fig. 3. Weak-scaling of the collide kernel performance for 32×32 lattice sites per lane as a function of the number of used lanes (left pane) and strong-scaling of the propagate kernel (right pane). The lines indicate the theoretical peak performance of 8 Flop/cycle/lane (left) and 8 Byte/cycle/lane (right).

By this we ensure that load latencies are mostly hidden. All necessary operations are performed on complex numbers in single precision. Specialised instructions have been added to the AMC ISA to efficiently implement complex multiply-add operations using 8-Byte complex numbers and only 2 instructions. This corresponds to a throughput of 1 single-precision complex multiply-add, i.e. 8 floating-point operations, per 2 cycles per slice. Each of the four slices operates on one of the four link dimensions. The requirements of the kernel are summarised in Table 2. Ultimately, the performance is limited by the memory bandwidth. However, while the first part executes one arithmetic operation per Byte loaded and stored, the second is imbalanced by a larger initially loaded segment. Thus, it incurs an additional start-up overhead.

5 Performance Analysis

We now proceed by presenting performance results for the kernels, for which implementation was discussed in the previous section. The results have been obtained on a cycle-accurate simulator for a full AMC including memories.

5.1 LBM

Collide. We start with a weak-scaling analysis of the collide kernel. Each instance of the kernel is executed on a single lane and operates on an individual set of 32 lattice sites. From the analysis of the kernel in terms of floating-point operations per memory access, we expect the kernel to operate close to the maximum floating-point performance. We present the relevant metric, the number of

Table 3. Median time per lattice site spent in the collide (t_{coll}, weak scaling: $l = 32$ sites per lane), propagate (t_{prop}, strong scaling: L^2 sites per AMC) and Wilson-Dirac (t_{comp}, weak scaling: l^4 sites per lane) kernel.

Lanes	t_{coll} (ns)		t_{prop} (ns)				t_{comp} (ns)				
	l	32	L^2	32^2	64^2	128^2	l^4	2^4	4^4	8^4	16^4
1		821.2		33.7	32.9	37.9		158.5	141.0	142.9	143.8
2		411.4		17.9	16.5	19.1		90.2	76.5	81.1	81.4
4		207.5		8.1	8.4	9.6		64.0	41.7	41.7	42.9
8		107.5		4.0	4.6	5.0		58.9	25.9	23.3	23.7
16		53.2		2.3	2.7	2.8		47.3	20.0	15.2	14.4
32		30.7		1.2	1.6	2.3		41.7	13.2	10.7	-

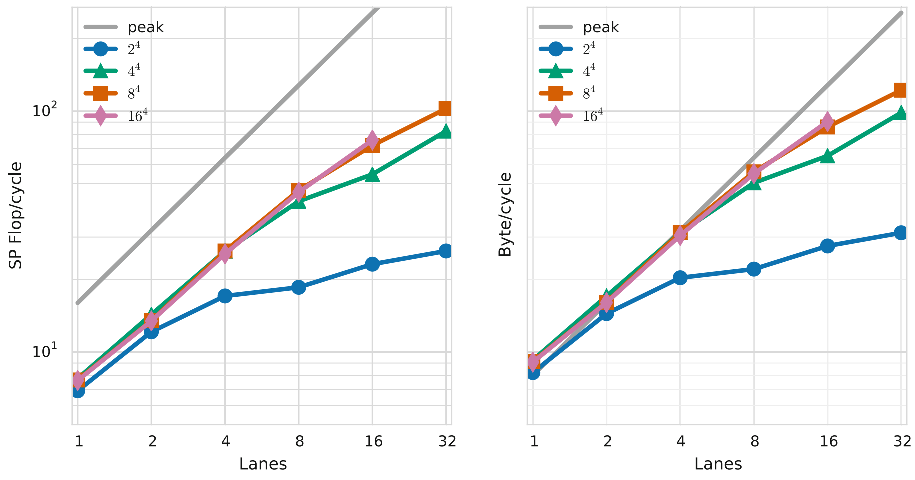

Fig. 4. LQCD operator weak scaling on AMC over the number of lanes in use.

floating-point operations performed per cycle, along with the speed-up over a single lane in Fig. 3. Across all numbers of lanes up to 32, we can exploit above 60 % of the peak floating-point performance. On a single lane, the implementation reaches 70 % of peak. We observe a small negative impact of utilising more than a single lane, despite operating on disjunct blocks of memory.

Propagate. The performance of the propagate kernel completely depends on memory access performance. In Fig. 3 (right pane) we show strong scaling results.

Out of the theoretical maximum of 320 GByte/s bandwidth, propagate can sustain 125 GByte/s on a lattice of 128^2, or 39 % of peak. The simulations show that for larger number of lanes memory accesses take longer to complete, which could be result of congestion or non-optimal access patterns.

5.2 LQCD

We investigated the performance of the partial implementation of the Wilson-Dirac operator D in a weak scaling study. The extracted kernel is run on multiple lanes, each processing a private lattice of extent L^4 where $L = 2, 4, 8, 16$. The results are summarised in Fig. 4 as performance in floating-point operations and memory bandwidth. For all lattice sizes, the best performance was achieved at the maximum of 32 lanes. From $L = 2$, the overall performance increases up to the case $L = 8$, where it is virtually identical to $L = 16$. The optimum sustained bandwidth on 32 lanes is 107 GByte/s, or 33 % of the peak value.

From Fig. 4 we can derive the optimum local lattice size per lane, namely 2^4 for two lanes, 8^4 for eight lanes and 16^4 for sixteen or more lanes.

6 Discussion of AMC Architecture

Based on our analysis of different application kernels we are now able to analyse selected hardware aspects and parameters and explore possible impact on the kernel's performance.

Instruction Set. The given set of application kernels could be mapped well on the instruction set architecture (ISA) of the Active Memory Cube (AMC). Specific ISA features which have been exploited include flexible load/store instructions and the availability of specialised single precision complex operations. The architecture allows for an efficient implementation of complex arithmetics. For the Dirac kernel the upper bound of the floating point efficiency as determined only by instruction selection and dependence lies at 94 %.

Temporal SIMD and Register File. For all kernels the temporal SIMD could be extensively exploited. The penalty of not using all elements of a vector register was found to be small. The collide kernel utilises near the maximum of scalar and vector registers on all slices. The propagate kernel uses three vectors per slice and about ten scalars. The Wilson-Dirac operator makes use of 14 out of 16 vector registers and 23 out of 32 scalar registers. The register file size was in all three cases found to be adequate for holding data or staging data loaded from memory. Sharing of vector registers between slices proved to be a crucial feature for efficient mapping of the application kernels on this architecture.

Instruction Buffer Size. The overall number of VLIWs needed to implement the (most important parts) of the kernels was found to be small (see Table 2) such that at most 32 % of the instruction buffer was filled.

Memory Bandwidth. Although the memory bandwidth is high compared to the compute performance, the Dirac and propagate kernels are nevertheless limited by the memory bandwidth on the AMC.

Memory Access Latency. Due to the size of the register file, load operations can be scheduled such that memory access latencies are largely hidden. For a strictly in-order design timely scheduling of such instructions can have significant performance impact. From our simulations we find memory access latencies to grow significantly when increasing the number of used lanes, resulting in a loss of efficiency.

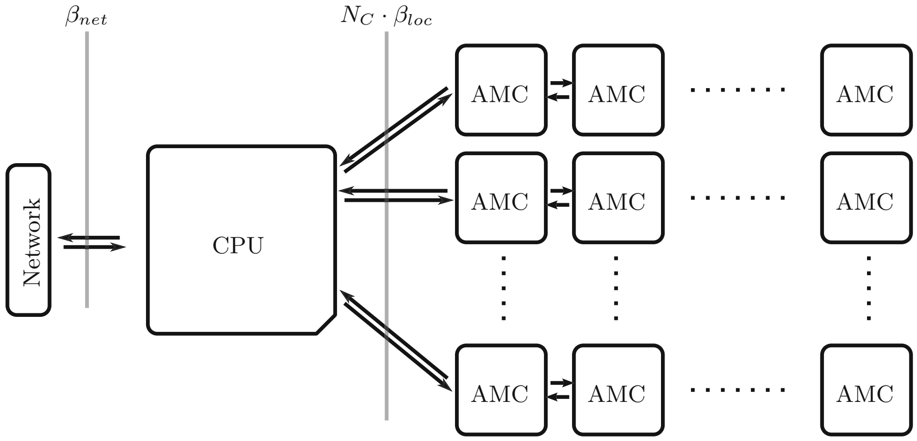

Fig. 5. Proxy architecture for AMC enabled nodes.

Table 4. Key parameters of our node proxy architecture.

Description	Parameter	Default value
Number of AMC	N_{AMC}	16
Number of chains	N_{C}	8
AMC memory capacity	C_{mem}	8 GByte
Inter-AMC link performance	$(\lambda_{\mathrm{mem}}, \beta_{\mathrm{mem}})$	$(1\,\mu$s, 32 GByte/s)
Network link performance	$(\lambda_{\mathrm{net}}, \beta_{\mathrm{net}})$	$(1\,\mu$s, 100 GByte/s)

Load/Store Queue (LSQ). The LSQ size is limited to 192 entries and holds both load and store requests. Its particular size has not been critical for neither the propagate nor the collide kernel. For the Wilson-Dirac operator, however, the majority of stalled cycles are due to a filled LSQ. Based on the memory access pattern we do not expect a larger LSQ to improve performance.

7 System Performance Modelling

For assessing the characteristics of the full applications on an AMC enabled node architecture, we define a proxy node architecture [1]. The basic structure of this architecture, as shown Fig. 5, comprises a CPU with a set of AMCs, which are organised in chains of equal length, plus a network interface attached. We assume that various parameters of this architecture as listed in Table 4 can be varied. To obtain quantitative results we use a specific choice guided by the node architecture proposed in [14].

To model the system performance for the kernels considered in this paper we assume that all calculations are performed on the AMC, for which we can use the cycle accurate simulator results presented in the previous sections.

To assess the overall performance we have to model the time needed to perform the necessary data exchange when application kernels are parallelised over multiple AMCs per node and multiple nodes. We assume that time for point-to-point communications can be described by a latency-bandwidth model, i.e.

$$T_{\text{comm}} = \lambda + \frac{I}{\beta}, \tag{4}$$

where I is the amount of communicated data and (λ, β) are the start-up latency and asymptotic bandwidth, respectively. We, furthermore, make the following assumptions:

– All links between components are bi-directional and can be described by the same parameters in both directions.
– All data transfers can be overlapped, therefore, the simulator results are valid for node level implementations.
– One task is executed per AMC and care is taken to achieve optimal placement.

7.1 LBM

The LBM application has, apart from collide and propagate, only one relevant step in performing a lattice update, the communication between different tasks. We model this part to estimate the performance on an AMC-enabled system.

As mentioned before, the lattice is distributed over multiple tasks using a one dimensional decomposition. Each task, and equivalently each AMC, holds a tranche with extent $L_x \cdot L_y$. A layer of three halo columns is introduced to exchange boundaries, necessitating a transfer of

$$I = 3 \cdot 2 \cdot L_x \cdot 37 \cdot 8\,\text{Byte} = L_x \cdot 1776\,\text{Byte} \tag{5}$$

per sub-domain. We assume L_x to be sufficiently large to allow execution of the propagate kernel and the halo exchange to be perfectly overlapped. We can therefore estimate the time required for an update step as follows:

$$T_{\text{step}} = \max(L_x \cdot L_y \cdot t_{\text{prop}}, T_{\text{comm}}) + L_x \cdot L_y \cdot t_{\text{coll}}, \tag{6}$$

where t_{coll} and t_{prop} are the measured execution time for collide and propagate kernel, respectively. They depend on the problem size as well as the number of used lanes. In the following we will consider only the case $N_{\text{lane}} = 32$.

Assuming that data communication performance is limited either by the link connecting the CPU and the first AMC within a chain as well as the network bandwidth, we obtain the following estimate for the time required for data communication:

$$T_{\text{comm}} = \max\left(T_{\text{ex}}^{\text{mem}}, T_{\text{ex}}^{\text{net}}\right) = \max\left(\lambda_{\text{mem}} + \frac{I \cdot N_{\text{AMC}}}{N_{\text{C}} \cdot \beta_{\text{mem}}}, \lambda_{\text{net}} + \frac{I \cdot N_{\text{AMC}}^{\frac{3}{4}}}{\beta_{\text{net}}}\right), \tag{7}$$

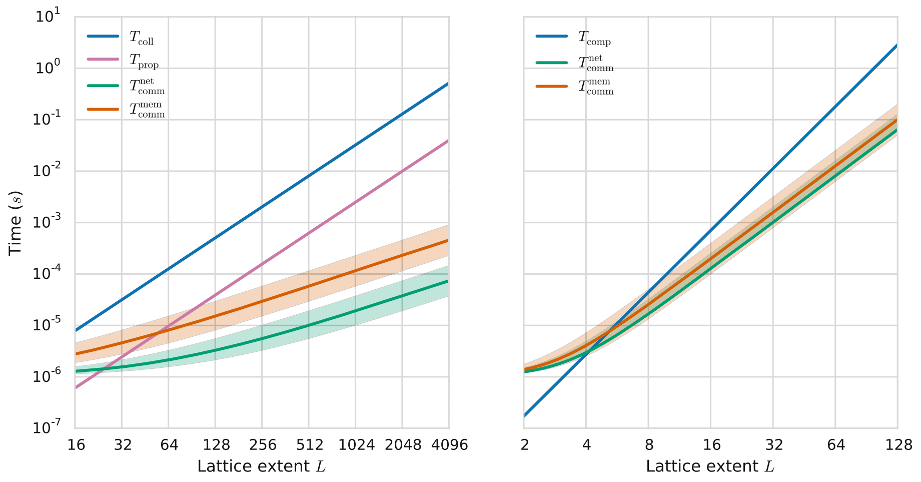

Fig. 6. Performance model for LBM (**left**) and LQCD (**right**), using the selected parameters for the proxy architecture as a function of the lattice size per AMC. Shaded areas indicate the effect of halving/doubling the respective bandwidth.

Considering square sub-domains per AMC, the minimal local size at which the kernels can utilise the AMC efficiently is about 32^2 from our experiments and maximally 3500^2 due to memory constraints.

For exploring execution time as a function of the lattice size per AMC we use the maximum execution times per lattice site observed when using $N_{\text{lane}} = 32$ lanes (see Table 3): $t_{\text{coll}} = 30.7\,\text{ns}$, $t_{\text{prop}} = 2.37\,\text{ns}$. The results are shown in Fig. 6. We find that lattice size beyond 128^2 yield performance numbers that are dominated by the computations as opposed to boundary exchange. Specifically for $L_x = L_y = 128$ we obtain the following results: $T_{\text{coll}} = 502\,\mu s$ and $T_{\text{step}} = 542\,\mu s$.

7.2 LQCD

To model the performance of a parallelised matrix-vector multiplication we apply the same methodology as in the previous section. We assume that computation and communication can be perfectly overlapped such that

$$T^{\text{Dirac}} = \max(T_{\text{comp}}, T_{\text{comm}}).$$

We assume a four dimensional domain decomposition into one sub-domain per task. Let L^4 be the lattice size per AMC, where the assumed memory capacity C_{mem} limits us to $L \leq 64$. The performance results shown in Table 3 indicate that a lattice with $L \geq 8$ should be used. Time for computation $T_{\text{comp}} = L^4 \cdot t_{\text{comp}}$ with $t_{\text{comp}} = 10.7\,\text{ns}$ is obtained from simulations (see Table 3).

A halo of thickness one, containing matrices comprising 12 single-precision complex numbers is exchanged with each neighbour, resulting in a data volume of

$$I = L^3 \cdot 2 \cdot 4 \cdot 12 \cdot 8\,\text{Byte} = L^3 \cdot 768\,\text{Byte} \tag{8}$$

per task. On the grounds of optimal task placement, we can argue that the amount of data exchanged by a node over the network is $I \cdot N_{\text{AMC}}^{\frac{3}{4}}$. The time consumed by the communication is then given by

$$T_{\text{comm}} = \max\left(T_{\text{ex}}^{\text{mem}}, T_{\text{ex}}^{\text{net}}\right) = \max\left(\lambda_{\text{mem}} + \frac{I \cdot N_{\text{AMC}}}{N_{\text{C}} \cdot \beta_{\text{mem}}}, \lambda_{\text{net}} + \frac{I \cdot N_{\text{AMC}}^{\frac{3}{4}}}{\beta_{\text{net}}}\right), \tag{9}$$

where the local exchange has an extra factor N_{AMC}/N_C since all AMCs in a chain share the bandwidth to the CPU.

We investigate the resulting timing in Fig. 6. We conclude that even with half the available bandwidth, a lattice extent of $L = 16$ yields timings that are determined solely by the computation. At this setting we expect $T_{\text{Dirac}} = T_{\text{op}} = 0.7\,\text{ms}$ for a lattice update step, using 8192 tasks on 512 nodes. The total peak performance of such a partition is 2.6 PFlop/s. Halving the local extent to $L = 8$ is the point at which the time for local exchange (at slightly more than the half effective bandwidth) is equal to T_{op}. At this point we are using 8192 nodes (42 PFlop/s) and expect an operator evaluation every $T_{\text{Dirac}} \simeq 44\,\mu s$.

8 Related Work

Performance analysis of Lattice Boltzmann Methods have been investigated in detail in various papers. For instance, the 2-dimensional D2Q37 model has been explored extensively for different processor and accelerator architectures including IBM PowerXCell 8i [3], Intel Xeon [4], GPUs and Xeon Phi [6]. Recently an increasing number of publications focus on the 3-dimensional D3Q19 model, concentrating on optimising memory hierarchies [15,20].

With advances in research on Lattice Quantum Chromodynamics (LQCD) strongly linked to progress in available compute resources, there exists a large number of publications where performance characteristics of LQCD applications are studied and results from performance analysis for different architectures are presented. Recent examples include the performance analysis for LQCD solvers on Blue Gene/Q [5] and architectures comprising GPUs and Xeon Phi [10,21].

Starting in the 90 s different processing-in-memory (PIM) architectures have been proposed and explored, including Computational RAM [7], Intelligent RAM [16], DIVA [9], and FlexRAM [12]. The initial enthusiasm decreased lacking a perspective of PIM architectures being realised. For a brief but more comprehensive overview on different projects see [19]. Different application kernels have been mapped to these architectures to explore their performance, with focus on kernels that feature irregular memory access patterns. To the best of our knowledge no results from a performance analysis for large-scale scientific applications on massively-parallel architectures comprising PIM modules has been published.

9 Summary and Conclusions

Studying two applications on the IBM Research Active Memory Cube (AMC) processing-in-memory architecture yielded promising results. Both applications are representative for many simulation codes, which are similarly structured, so conclusions can be expected to be transferable to those applications. We could demonstrate that compared to peak floating-point performance efficiencies could be reached which are mostly larger than those for processors and accelerators available today. By keeping most of the data transfer local within the memory, the need for data transport outside the memory could be significantly reduced.

Our results for kernel execution times allows to estimate energy efficiency. Assuming a power consumption of 10 W the costs of the D2Q37 LBM collide kernel per lattice site is $0.3\,\mu J$. This compares favourably to NVIDIA K20x GPUs with an energy consumption of $2.6\,\mu J$ per lattice site [6], even if one takes into account that these measurements were done on a technology which became available in 2012.

While optimal exploitation of the AMC architecture requires the programmer to take data locality and access patterns into consideration, the resulting benefit to the user is a power efficiency distinctly better than alternative platforms.

Acknowledgements. We thank the AMC team at IBM Research, in particular J. Moreno, for sharing their knowledge on the AMC and continued help on this project including many fruitful discussions. Furthermore, we gratefully acknowledge F.S. Schifano and R. Tripiccione (INFN/University of Ferrara) for making a mini-application version of their D2Q37 code available and for discussing their future roadmaps [18]. We also thank G. Koutsou, S. Krieg, and H. Simma from the Simulation Lab LQCD at Cyprus Institute/DESY/JSC for discussing the future requirements of LQCD [13]. Finally, we thank A. Frommer and S. Krieg for making their implementation of their AMG solver [8] available.

References

1. Ang, J.A., Barrett, R.F., Benner, R.E., Burke, D., Chan, C., Cook, J., Donofrio, D., Hammond, S.D., Hemmert, K.S., Kelly, S.M., Le, H., Leung, V.J., Resnick, D.R., Rodrigues, A.F., Shalf, J., Stark, D., Unat, D., Wright, N.J.: Abstract machine models and proxy architectures for exascale computing. In: Proceedings of the 1st International Workshop on Hardware-Software Co-Design for High Performance Computing (Co-HPC 2014), pp. 25–32. IEEE Press, Piscataway (2014). http://dx.doi.org/10.1109/Co-HPC.2014.4
2. Balasubramonian, R., Chang, J., Manning, T., Moreno, J.H., Murphy, R., Nair, R., Swanson, S.: Near-data processing: insights from a MICRO-46 workshop. IEEE Micro **34**(4), 36–42 (2014)
3. Biferale, L., Mantovani, F., Pivanti, M., Sbragaglia, A., Schifano, S., Toschi, F., Tripiccione, R.: Lattice Boltzmann fluid-dynamics on the QPACE supercomputer. Procedia Comput. Sci. **1**(1), 1075–1082 (2010). http://www.sciencedirect.com/science/article/pii/S1877050910001201, ICCS 2010

4. Biferale, L., Mantovani, F., Pivanti, M., Pozzati, F., Sbragaglia, M., Scagliarini, A., Schifano, S.F., Toschi, F., Tripiccione, R.: Optimization of multi-phase compressible lattice Boltzmann codes on massively parallel multi-core systems. Procedia Comput. Sci. **4**, 994–1003 (2011). http://www.sciencedirect.com/science/article/pii/S1877050911001633, Proceedings of the International Conference on Computational Science, ICCS 2011
5. Boyle, P.A., Christ, N.H., Kim, C.: Co-design of the IBM BlueGene/q level 1 prefetch engine with QCD. IBM J. Res. Dev. **57**(1/2), 13:1–13:10 (2013)
6. Calore, E., Schifano, S.F., Tripiccione, R.: A portable OpenCL lattice Boltzmann code for multi- and many-core processor architectures. Procedia Comput. Sci. **29**, 40–49 (2014). http://www.sciencedirect.com/science/article/pii/S1877050914001811, 2014 International Conference on Computational Science
7. Elliott, D., Snelgrove, W., Stumm, M.: Computational ram: a memory-simd hybrid and its application to dsp. In: Proceedings of the IEEE 1992 on Custom Integrated Circuits Conference, pp. 30.6.1–30.6.4, May 1992
8. Frommer, A., Kahl, K., Krieg, S., Leder, B., Rottmann, M.: Adaptive aggregation based domain decomposition multigrid for the lattice Wilson Dirac operator. SIAM J. Sci. Comput. **36**, A1581–A1608 (2014)
9. Hall, M., Kogge, P., Koller, J., Diniz, P., Chame, J., Draper, J., LaCoss, J., Granacki, J., Brockman, J., Srivastava, A., Athas, W., Freeh, V., Shin, J., Park, J.: Mapping irregular applications to DIVA, a PIM-based data-intensive architecture. In: ACM/IEEE 1999 Conference on Supercomputing, pp. 57–57, November 1999
10. Heybrock, S., Joó, B., Kalamkar, D.D., Smelyanskiy, M., Vaidyanathan, K., Wettig, T., Dubey, P.: Lattice QCD with domain decomposition on intel xeon phi coprocessors. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2014), pp. 69–80. IEEE Press, Piscataway (2014). http://dx.doi.org/10.1109/SC.2014.11
11. Hybrid Memory Cube Consortium: Hybrid Memory Cube Specification (2013)
12. Kang, Y., Huang, W., Yoo, S.M., Keen, D., Ge, Z., Lam, V., Pattnaik, P., Torrellas, J.: FlexRAM: toward an advanced intelligent memory system. In: International Conference on Computer Design (ICCD 1999), pp. 192–201 (1999)
13. Koutsou, G., Krieg, S., Pleiter, D., Simma, H.: EIC co-design questionnaire: lattice QCD (unpublished, 2013)
14. Nair, R., Antao, S.F., Bertolli, C., Bose, P., Brunheroto, J.R., Chen, T., Cher, C.-Y., Costa, C.H.A., Evangelinos, C., Fleischer, B.M., Fox, T.W., Gallo, D.S., Grinberg, L., Gunnels, J.A., Jacob, A.C., Jacob, P., Jacobson, H.M., Karkhanis, T., Kim, C., Moreno, J.H., O'Brien, J.K., Ohmacht, M., Park, Y., Prener, D.A., Rosenburg, B.S., Ryu, K.D., Sallenave, O., Serrano, M.J., Siegl, P.D.M., Sugavanam, K., Sura, Z.: Active memory cube: a processing-in-memory architecture for exascale systems. IBM J. Res. Dev. **59**(2/3), 17:1–17:14 (2015)
15. Nguyen, A., Satish, N., Chhugani, J., Kim, C., Dubey, P.: 3.5-d blocking optimization for stencil computations on modern cpus and gpus. In: International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2010), pp. 1–13, November 2010
16. Patterson, D., Anderson, T., Cardwell, N., Fromm, R., Keeton, K., Kozyrakis, C., Thomas, R., Yelick, K.: A case for intelligent RAM. IEEE Micro **17**(2), 34–44 (1997)
17. Scagliarini, A., Biferale, L., Sbragaglia, M., Sugiyama, K., Toschi, F.: Lattice Boltzmann methods for thermal flows: continuum limit and applications to compressible Rayleigh-Taylor systems. Phys. Fluids **22**(5), 055101 (2010)

18. Schifano, S.F., Tripiccione, R.: EIC co-design questionnaire: LBM (unpublished, 2013)
19. Torrellas, J.: Flexram: toward an advanced intelligent memory system: a retrospective paper. In: IEEE 30th International Conference on Computer Design (ICCD 2012), pp. 3–4, September 2012
20. Williams, S., Oliker, L., Carter, J., Shalf, J.: Extracting ultra-scale lattice Boltzmann performance via hierarchical and distributed auto-tuning. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2011), pp. 55:1–55:12. ACM, New York (2011). http://doi.acm.org/10.1145/2063384.2063458
21. Winter, F., Clark, M., Edwards, R., Joo, B.: A framework for lattice QCD calculations on GPUs. In: 2014 IEEE 28th International Parallel and Distributed Processing Symposium, pp. 1073–1082, May 2014