Exploiting In-Memory Processing Capabilities for Density Functional Theory Applications

Paul F. Baumeister[1], Thorsten Hater[1], Dirk Pleiter[1],
Hans Boettiger[2], Thilo Maurer[2], and José R. Brunheroto[3]

[1] Jülich Supercomputing Centre, Forschungszentrum Jülich, 52425 Jülich, Germany
[2] IBM Deutschland Research & Development GmbH, 71032 Böblingen, Germany
[3] IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, US

Abstract. Processing-in-memory (PIM) is an approach to address the data transport challenge in future HPC architectures and various designs have been explored in the past. Despite, it remains unclear how scientific applications could efficiently exploit massively-parallel HPC architectures integrating PIM modules. In this paper we address this question for material science applications for which we ported relevant kernels to the Active Memory Cube architecture developed by IBM Research.

1 Introduction

Over at least two decades exponential growth of arithmetic performance of HPC architectures could be sustained. Exploiting this performance becomes more challenging as with growing complexity massively-parallel architectures will become limited by data transport. One approach to mitigate this problem is to move processing pipelines closer to the locations where data is stored, as it is done in processing-in-memory (PIM) architectures. Such architectures could be particularly attractive for future, power-constrained supercomputers, because potentially energy consuming data movements may be avoided. A recent example of PIM architectures is IBM Research's Active Memory Cube (AMC) [18].

While there are architectural arguments in favour of PIM-based HPC architectures, it remains unclear how efficiently such architectures could be exploited by relevant scientific applications. The goal of this paper is to explore this question for two materials science applications on the basis of the AMC architecture.

As of today, materials science applications consume a significant fraction of the available HPC resources. It is one of the areas in science and engineering that will significantly benefit from further growth of computational resources and is expected to require exascale computing capabilities in the future.

A key technique in materials sciences is Density Functional Theory (DFT). DFT simulations give access to an accurate prediction of the electronic ground state structure, equilibrium geometries and thermodynamic properties of most classes of materials, see [5] for an overview. The approach has grown from fundamental research to wide application in the field of materials research and design.

In this paper we consider two selected DFT-based applications, which differ significantly in terms of application performance characteristics. Both have

in common that they are highly scalable on current supercomputers. We have ported performance relevant parts of these codes to the AMC architecture to evaluate the performance in cycle-accurate simulations and assess the overall benefits from performance profiles obtained on existing systems.

This paper makes the following contributions:

- We show results from implementations of relevant kernels of selected DFT applications on a future processing-in-memory architecture and provide a performance analysis based on cycle accurate simulations.
- To better understand the opportunities of such future technology, we provide an assessment of future requirements of DFT applications.
- Based on results from implementation and performance analysis we explore the features of the AMC architecture as well as its hardware parameters.

In the next section we provide background information on DFT-based methods and future developments in this application area, which is followed by details of the AMC architecture in Sec. 3. After discussing details and performance characteristics of the specific DFT applications considered in this paper (Sec. 4) we discuss their implementation on AMC (Sec. 5). Based on an analysis of the obtained performance, which is presented in Sec. 6, we discuss the suitability of the AMC architecture in Sec. 7. Before concluding we provide a short overview on related work in Sec. 8.

2 Application Background

In this investigation, we analyse kernels of two different implementations of Density Functional Theory (DFT), juRS [4, 23] and KKRnano [23]. Both applications are optimised for high scalability and to address problems with a large number of atoms, $N_{\text{atom}} \gg 1,000$, on massively-parallel machines. Their approach to the problem differs as juRS solves for eigenstates of the DFT Hamiltonian whereas KKRnano finds the electronic structure by operator inversion. KKRnano even allows a truncation of very long-ranged interactions and, thus, transits into an $\mathcal{O}(N_{\text{atom}})$ scaling behaviour. The linear scaling mode makes a million atoms feasible as computer systems grow. So far, more than $200,000$ atomic sites could already be processed during a pioneering run on an IBM Blue Gene/Q$^{\text{TM}}$ system providing a peak performance of 5.9 PFlop/s [25].

Future application requirements While we will use today's applications to evaluate a future technology, i.e. the Active Memory Cube, we also analyse future requirements of these applications. Based on a questionnaire we analysed together with domain experts on how, e.g., the application domain, the used methods and algorithms, problem size and the resource requirements are expected to evolve.

According to [6] the development of methods for ab initio studies exhibits various trends, one being the development towards a more precise methodology overcoming the drawbacks of approximations made in current applications [17]. More computing resources will be required to either facilitate high throughput for

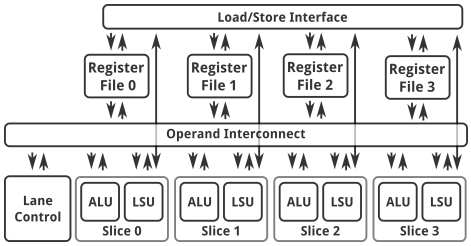

Fig. 1. Sketch of the AMC lane architecture. For more details see [18].

medium-sized problems as well as to address large-scale challenges. The former will, e.g., be required to scan parameter spaces and evaluate high-dimensional phase diagrams. The latter involves problems where a large number of atoms, N_{atom}, are required. Challenging problems are related to broken symmetries, i.e. crystals with impurities, random alloys or amorphous materials. To address these questions $N_{\mathrm{atom}} \gg 10^5$ atoms are often necessary.

The aforementioned applications, juRS and KKRnano, target such large problem sizes. With the $\mathcal{O}(N_{\mathrm{atom}})$-mode of KKRnano exascale compute resources allow to determine the electronic structure of a million atoms within less than an hour and structural relaxation within a single day.

3 Active Memory Cubes

Recently, several new high-bandwidth memory technologies have been introduced. Both, Hybrid Memory Cube (HMC) [14] as well as High Bandwidth Memory (HBM) [15] have in common that they foresee a stack of DRAM dies on top of a logical die. This logic die is currently mainly foreseen to facilitate data transport, e.g. in the HMC architecture the logic die implements the memory controller and a network interface via which the processor can access the memory. But in principle also processing of data could be supported at this level. This approach is explored in a recent architectural proposal by IBM Research: the Active Memory Cube (AMC) [18]. In this architecture 32 computational lanes are added to the logic die, which also have access to the memory, i.e. the memory becomes dual-ported as it continues to be accessible from the CPU.

Each lane is composed of four computation slices, which comprise a load-store unit (LSU) as well as an arithmetic-logic unit (ALU), plus a control unit. Each slice has a register file, which includes 32 scalar plus 16 vector computation registers. All registers are 64-bit wide and each vector register has 32 elements. See Fig. 1 for a sketch of the AMC lane architecture.

The computational lanes are micro-coded. In each clock cycle a lane can process one Very Long Instruction Word (VLIW) composed of nine sub-instructions. The instructions are read from a buffer which can hold 512 VLIW instructions. Due to the length of a VLIW instructions it is important to reduce the required number of these instructions. In this architecture this is facilitated through a temporal single-instruction-multiple-data (SIMD) paradigm. Instructions can be repeated up to 32 times, matching the length of the vector registers.

The arithmetic pipelines take 64-bit input operands and can complete in each clock cycle one double-precision Fused Multiply-Add (FMA) or a two-way SIMD single-precision FMA. Thus, up to 8 double-precision floating-point (FP) operations can be completed per clock cycle in one lane. The peak performance of one AMC running at a clock speed of 1.25 GHz is thus 320 GFlop/s in double-precision. A slice can read the vector registers of the other slices, which allows to distribute data over multiple register files. Furthermore, double-precision complex arithmetic with real and imaginary part distributed over different slices can be implemented without the need for data re-ordering instructions.

Load/store requests are buffered in a load-store queue with 192 entries. A lane can load or store 8 Byte/cycle from or to the internal interconnect, i.e. the ratio of memory bandwidth vs. FP performance is 1 Byte/Flop and thus significantly larger than in typical processor architectures. Additional non-exposed arithmetic units are given inside the memory controllers allowing to issue atomic update operations onto memory locations. Here, the instruction set architecture foresees integer and also 64-bit FP addition operations.

Each AMC features a network interface with a bandwidth of 32 GByte/s. It can be used to connect to a processor or to chain multiple AMC devices in a similar way as HMC devices.

The execution model foresees main programs to be executed on a general-purpose CPU with computational lanes being used for off-loading small kernels. It is planned to have VLIW instructions for the off-loaded kernels being generated by a compiler controlled through directives, e.g. OpenMP-4.0, see [18] for details. Such a compiler is not yet available and therefore all sequences of VLIW instructions have been implemented manually.

The maximum power envelope for dies within a 3D stack is small, since layers cannot be cooled individually, yet. Assuming a design based on 14 nm technology, the power consumption for an AMC device is expected to be around 10 W.

4 Applications and Performance Characteristics

Real-space grid DFT: juRS represent the DFT Hamiltonian on a uniform Cartesian real-space grid and follows the approach of iterative diagonalisation, see [4] for details. The application of the grid Hamiltonian to wave functions reads

$$\hat{H}\,|\Psi_k\rangle = \left[-\frac{\hbar^2}{2m}\left(\partial_{xx}+\partial_{yy}+\partial_{zz}\right)+V_{\mathrm{loc}}(x,y,z)\right]|\Psi_k\rangle\,. \tag{1}$$

The 3D Laplacian represents the kinetic energy operator in real-space representation. In juRS, it is approximated by an 8^{th}-order finite-differences (FD) scheme which leads to a 3D stencil operation on a uniform lattice of grid points. This allows for a controllable accuracy and avoids FFTs and the related parallel scalability issues completely. The selected FD approximation is symmetric around the central coefficient, c_0, with legs of 4 constant coefficients reaching into both directions of each of the three spatial dimensions, see left side of Fig. 2. On most architectures, the decomposition into three 1D FD stencils is benefitial. Then,

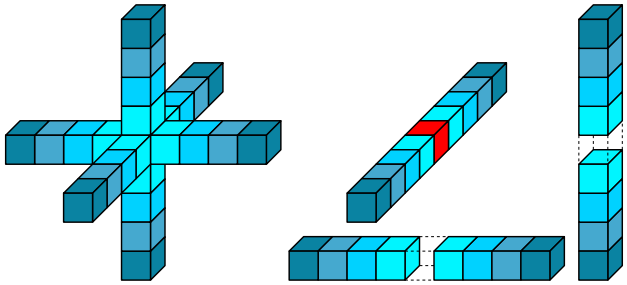

Fig. 2. Decomposition of a 3D finite-difference stencil (left) into three 1D stencils (right). The central coefficient of the y and z-direction (horizontal and vertical) are merged into that of x (red) leaving gaps.

Table 1. Requirements for the relevant DFT kernels. The arithmetic intensity (AI), given in the limit of $n_{x|y|z} \to \infty$, represents the ratio between compute and data movement in Flop/Byte.

Kernel	**Flops**	**Loads** (8 Byte)	**Stores**	**AI**
`fdd-Vx`	$32 \cdot 17\, n_x$	$32 \cdot (8{+}n_x){+}n_x$	$32\, n_x$	1.1
`fdd-yz`	$32 \cdot 16\, n_{y\|z}$	$32 \cdot (8{+}2\, n_{y\|z})$	$32\, n_{y\|z}$	0.7
`zgemm-16`	32768	1536	512	2.0

only one stencil (x, kernel `fdd-Vx`) carries the central coefficients and the local potential $V_{\mathrm{loc}}(x, y, z)$, see right side of Fig. 2, and the kernel `fdd-yz` with its eight non-zero coefficients is called twice with different array strides. The grid Hamiltonian may be applied to several wave functions Ψ with index k at once in order to bundle communication of grid-halos. Details about the requirements of the juRS finite-difference kernel are summarized in Table 1.

Green function DFT: KKRnano directly inverts the the DFT Hamiltonian matrix, H. Instead of finding eigenstates, we search for columns of the Green function, x, i.e. a linear equation with multiple right-hand sides is solved. The so-called tight-binding or screened formulation of the Green function formalism allows for representing the Hamiltonian as short-ranged in real-space [26], i.e. its application to a trial vector only couples elements that are associated to basis functions localised on neighbouring atomic sites.

The parallelisation strategy foresees one atom per MPI process and we typically deal with 16 basis functions per atom and energy, resolving states with different angular momentum. The solutions are found using the transpose-free quasi-minimal residual technique [8] for 16 right hand sides at a time. Here, the performance of the application depends almost exclusively on that of applying the matrix H to vector x as

$$y_i = \sum_j H_{ij} x_j, \qquad y_i,\, H_{ij},\, x_j \in \mathbb{C}^{16\times 16}\,. \tag{2}$$

All elements of this equation are complex matrices of dimension 16, which thus leads to a large number of multiplications of (double-precision) complex matrices of dimension 16. These are implemented in a kernel called `zgemm-16`.

For this kernel we have an arithmetic intensity AI=2 (see Table 1). As each AMC lane features a compute performance versus memory bandwidth ratio of 1 Flop/Byte we expect the performance of this kernel to be limited by the compute capability.

5 Implementation on AMC

KKRnano The most important kernel of KKRnano, `zgemm-16`, can be considered as a specialised version of the BLAS routine `zgemm` which implements the operation

$$C \leftarrow C + A \cdot B, \qquad A, B, C \in \mathbb{C}^{16\times 16}. \tag{3}$$

Eq. (2) is evaluated many times to solve the linear set of equations iteratively and the kernel `zgemm-16` is invoked even more often. Within each application of the Hamiltonian to a vector, `zgemm-16` is executed about 16 000 times per atom in KKRnano's $\mathcal{O}(N_{\text{atom}})$-mode and even more times without truncation for linear scaling.

Our implementation in microcode makes use of the fact that the AMC's vector registers of each lane can hold up to 16 kiByte of data. Therefore, all elements of A can be kept in the vector registers once they are loaded. Register spills can be avoided completely. The elements of B are loaded successively into scalar registers. The loops can be fully unrolled resulting in a kernel comprising 16 384 multiply-add operations. Exploiting the temporal (SIMD) paradigm of pipelining we only need 384 VLIW instructions for implementing this kernel (neglecting some entry and exit code). Due to the organisation of the vector register hardware, random access is to vector register elements not possible. This necessitates the reorganisation of the matrix-matrix product algorithm to accumulate multiple results simultaneously, in our case 16 real and 16 imaginary values. In other words, we compute one column of the solution matrix leveraging the SIMD-in-time model

$$C_{0\ldots15,j} \leftarrow C_{0\ldots15,j} + \sum_k A_{0\ldots15,k} \cdot B_{k,j} \tag{4}$$

juRS finite-differences As the Hamiltonian is always applied to a larger set of independent states at a time, we tile the set with index k into coherent subsets of length 32 to match to AMC's vector length.

A single-pass implementation of the finite-difference Laplacian stencil on a 3D array with $n_x \times n_y \times n_z$ lattice sites can only exploit data re-use in the direction of the traversal of the 3D stencil. This corresponds to an arithmetic intensity of 49 Flop/144 Byte = 0.34 Flop/Byte. On AMC such an implementation would be memory-bandwidth-bound. As no caches are present on the AMC, it is advantageous to decompose the 3D stencil and perform three passes of a 1D FD stencil with index strides 1, n_x and $n_x n_y$ for the x-, y-, and z-direction, respectively, as described in Sec. 4 and Fig. 2. An overview of the characteristic numbers of the FD kernels `fdd-Vx` and `fdd-yz` is given in Table 1.

Fig. 3 explains the slice-parallelisation strategy of the 1D FD derivative for the implementation on AMC. Elements of the source array `A` holding the set of wave functions to be derived are distributed in a cyclic fashion over the four slices. All slices process the same sequence of instructions except for a phase shift by one VLIW (32 cycles due to the vectorisation) between adjacent slices. Therefore, all slices access the same lattice element at the same time exploiting that the

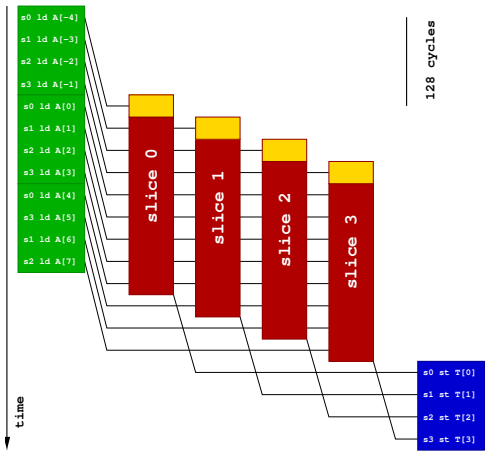

Fig. 3. Slice parallelisation scheme for finite-differences. Data flows from left to right while time propagates from top to bottom in the diagram.

read access to a vector register is shared across the slices of a lane. We schedule the load instruction on elements of `A` seven VLIWs (although Fig. 3 shows only a distance of four VLIWs) before all four slices access the shared vector register holding elements of `A` for reading. This is equivalent to 7×32 cycles between the issuing of the load and the usage of the element if no stalls are encountered. This hides the typical memory access latencies of the AMC. Furthermore, a four-fold loop unrolling allows for an efficient register allocation for this microcode so that 4×4 elements on 32 independent grids are processed per iteration.

The kernel `fdd-Vx` utilises the memory bandwidth mostly for loading elements of `A` and for storing the target array `T`. All arithmetic instructions are double-precision FMAs except for the first element. Therefore, 8 Byte are loaded, 8 Byte stored and 17 Flop performed per lattice site. Hence, the AI is about 1 Flop/Byte. The kernel `fdd-yz` for the other two derivatives ∂_{yy} and ∂_{zz} consists of update operations, i.e. we need to load the source array `A` and the target array `T` before updating `T`, which increases the amount of data to be loaded by 50 % compared to `fdd-Vx`. Due to the missing central FD coefficient, we perform 8 FMAs per 24 Byte, i.e. an AI of 0.7. This is below the specifications of the AMC with a ratio of at least 1 Flop/Byte, therefore, we expect this kernel to be memory-bound. As an alternative to this load-update-store scheme, *atomic* update operations can reduce the pressure on the memory interface.

6 Performance Analysis

The relevant performance metric for the investigate compute kernels is GFlop/s. During the analysis, we define the floating-point efficiency ϵ_{FP} as the ratio of achieved FP performance over the maximum of the AMC of 256 Flop/cycle or 320 GFlop/s when using all 32 lanes.

KKRnano The fully unrolled implementation of the double-precision $\mathbb{C}^{16\times16}$-matrix-matrix multiplication requires only 4886 AMC cycles to finish on a single lane. This is equivalent to a floating-point efficiency $\epsilon_{\mathrm{FP}} = 84\,\%$. As this kernel

has a high theoretical arithmetic intensity (AI) (see Table 1), we expect it to be compute bound. There are two potential causes for a lower effective performance. First, the time required to setup the registers and read the corresponding values from the stack memory. Second, the time overhead for offloading and returning control to the CPU.

The data layout for the complex arrays was tuned to allow for load combines, i.e. bundled memory requests of 16 or 32 Byte of data with adjacent memory addresses. This reduces the total number of memory requests which is important to sustain the performance also in multi-lane execution.

The resulting implementation is highly efficient, the instructions issued to the four ALUs are almost exclusively FMAs (98%) and only very few slots remain empty (2%). About 600 cycles are spent to setup the kernel and preload the initial values of the first matrix, A, which is then kept in the vector registers during kernel execution. Values of B are streamed through the scalar registers individually. Accordingly, the kernel utilises the memory interface efficiently (LSU instruction mix: 8% load, 5% store, 88% nop).

Most rows of H in KKRnano contain around 14 non-zero matrices. By introducing an index list into the kernel that contains the start addresses of the next pair of small matrices we can pipeline `zgemm-16` and, hence, distribute the start-up latency over the execution time for all elements in a row. Furthermore, we could save the storing and loading of C increasing the AI to 3.5. Asymptotically, we expect $\epsilon_{\mathrm{FP}} \simeq 98\,\%$ for a fully pipelined implementation of the block sparse matrix product, with a single kernel per row of 14 blocks.

We investigate the behaviour of the kernel when scaling to multiple processing elements. Each is issuing a separate instance of the problem, simulating the final implementation, where each lane concurrently computes one block of y and traverses a row of block in H. Results of multi-lane experiments can be found in Table 2. Here, excellent scaling is observed when increasing the number of active lanes. The sustained performance on all 32 lanes of an AMC is 262 GFlop/s which corresponds to $\epsilon_{\mathrm{FP}} = 82\,\%$. With more lanes working in parallel the number of stall cycles relative to those spent on executing instructions grows significantly. We suspect congestion on the memory system to be the cause. The effect was larger for allocation strategies controlling the placement in memory vaults other than the one used for these measurements. This indicates that the balancing of the memory requests to the different vault controllers is a crucial compontent to multi-lane efficiency. As each lane processes a disjoint problem set, the amount of memory requests in flight increases proportionally puts more load on the internal interconnect. We presented the optimal result from high level tuning of the memory locality. While more fine grained control on the actual location of memory allocations could alleviate the issue for elements of H, the access into y is hard to optimise.

juRS The AI of the FD kernel `fdd-Vx` ranges in a field where small changes in terms of requested memory traffic lead to a transition from being FP performance limited to memory-bandwidth-bound, compare Table 1. Both `fdd`-kernels process the grid in rows of the length of one of the domain dimensions. Each row

Table 2. Results for the execution of the juRS finite-difference derivative `fdd-Vx` and KKRnano `zgemm-16` kernels on **L** AMC lanes.

	fdd-Vx 16^3			fdd-Vx 32^3			zgemm-16			
L	**Cycles**	**Stalls**	ϵ_{FP}	**Cycles**	**Stalls**	ϵ_{FP}	**Cycles**	**Stalls**	ϵ_{FP}	**GFlop/s**
1	324 k	45 k	0.86	2.6 M	366 k	0.85	4886	682	0.83	8.4
2	186 k	47 k	0.74	1.4 M	293 k	0.79	4807	603	0.85	17.0
4	126 k	56 k	0.55	904 k	347 k	0.61	4893	689	0.83	33.5
8	63 k	28 k	0.55	454 k	176 k	0.61	4955	751	0.82	66.1
16	29 k	11 k	0.60	216 k	77 k	0.64	5007	803	0.81	130.9
32	20 k	11 k	0.43	160 k	90 k	0.43	4991	787	0.82	262.6

starts and ends with a halo region of four grid elements that need to be loaded but do not exhibit an 8 or 9-fold data re-use as it is in the bulk of the row. Therefore, shorter rows have a reduced average AI and, consequently, larger domain sizes lead to larger ϵ_{FP}. In addition to the halo-related overhead, the loads experience congestion effects when executed on multiple lanes as shown in Table 2 for `fdd-Vx`. Here, the number of lattice sites in one domain was 16×16^2 or 32×32^2 where the row length n_x is 16 and 32, respectively. The number of rows that are processed independently, 16^2 and 32^2, were distributed evenly among the number of lanes. Taking the halo-related overhead into account, the corresponding AIs are 0.84 and 0.93 Flop/Byte, respectively, compare Table 1. These translate into a maximum efficiency that is achieved only on a single lane in the smaller case. All multi-lane runs exhibit memory congestion effects that infer additional stall cycles and, hence, lower the total FP efficiency. Nevertheless, a sustained efficiency $\epsilon_{\mathrm{FP}} = 43\,\%$ can be measured.

For the `fdd-yz` kernel, the amount of memory accesses can be reduced by one third using the `AtomicAdd` instruction rather than a usual `LoadStore` scheme. Then, the *atomic* store operation only sends the numerical difference to be added to the content of the (64-bit) memory location. This allows to improve the single-lane efficiency from $\epsilon_{\mathrm{FP}} = 60\,\%$ to $80\,\%$ for a domain of 32×32 grid points.

Application performance It is difficult to make predictions of the overall speed-up of the juRS Hamiltonian action as the balance between the FD operations and other kernels depends on the input. This includes the species of atoms, their density, the grid spacing, and the required accuracy of the projector representation. All these can change significantly for different types of runs. Typically, the application spends between 30 to $60\,\%$ in FD operations. However, on standard CPU architectures we found the FP efficiency for these kernels to be typically $\epsilon_{\mathrm{FP}} \lesssim 10\,\%$. For instance, on BG/Q [4] ϵ_{FP} could be as low as $2\,\%$. For all kernels, which we have ported to AMC, we observe an efficiency $\epsilon_{\mathrm{FP}} \geq 43\,\%$. We thus expect the execution for the juRS Hamiltonian to be at least $7\times$ faster on a single AMC device compared to a BG/Q processor, that features a peak performance of 204.8 GFlop/s.

Table 3. Usage of AMC hardware resources and NOP-metric for the microcode.

Kernel	ALU-NOP	LSU-NOP	VLIW	VectorReg.
`fdd-Vx`	163	89 %	216 (42 %)	6
`fdd-yz`	194	75 %	206 (40 %)	8
`zgemm-16`	78	88 %	398 (78 %)	16

As the fraction of the work executed within the kernels considered in this paper reduces for larger N_{atom}, we refrain from make predictions on the overall application performance of juRS in the presence of AMC devices.

7 Discussion of AMC Architecture

For all kernels investigated here, the arithmetic intensities are high which results in a good usage of the ALU pipeline. The implementations exhibit a constant flow of arithmetic instructions inside the bulk of a kernel execution. Merely during startup and finalisation some VLIWs are found that carry only LSU instructions. Table 3 shows a measure of the NOP instructions. For example the `zgemm-16` kernel that runs 4886 cycles is implemented with only 1.6 % pure LSU instructions for all slices. Also for the juRS-kernels the number of ALU-NOP instructions is independent of the problem size. In contrast, the number of LSU-NOP instructions is large and scales with the problem size, therefore, the fraction of LSU-NOP instructions over possible LSU instruction slots is given here. The largest usage of LSU instructions is 25 % found at `fdd-yz` in the `LoadStore` scheme. Using the `AtomicAdd` scheme, this fraction is halfed.

The AMC architecture exhibits also other useful features for processing linear algebra tasks within DFT applications. Of particular interest are variants of real and complex double-precision matrix-matrix multiplications where a complete set of variants of multiply-add instructions, strong vectorisation and the overlap of load latencies with computations allows to achieve high efficiencies for sparse operator arithmetics. For the investigated kernels, 32 elements in each vector and four slices per lane can be employed to a full extent. In all kernels, load latencies are hidden by unrolling loops and the possibility to issue load or store instructions in the same cycle with arithmetic operations allows to run at floating-point efficiencies close to 100 % given that the AI is sufficiently high. The manually assembled micro-code implementations of the kernels make use of most of the 32 scalar registers per slice. The number of used vector registers per slice and the filling of the instruction buffer are listed in the right columns of Table 3 for the different kernels. Only for the `zgemm-16` kernel all 16 vector registers have been used. The latter indicates that the size of the register file is suitable for these application kernels. Despite completely unrolled loops in `zgemm-16` the instruction buffer with up to 512 VLIWs is large enough to host the instructions for the investigated operations without reloading.

All kernels make extensive use of overlapping load latencies with computations. When scheduling the loads in the sequence of VLIWs we have to balance between too early which would stall the lane due to overfilling of the load queue and too late which leads to stall cycles waiting for data to arrive. A shorter load

queue than 192 items is expected to increase the pressure on this trade-off and the dependence of the total runtime on memory congestion effects.

8 Related Work

Over the last years numerous DFT simulation codes have been developed for high-end HPC systems. For some of these codes performance analysis results for different architectures have been published. The authors of [2, 4] focus on the performance of the CP2K and juRS codes on the Blue Gene architecture. An overview of performance evaluations for Quantum Espresso on different high-end HPC systems is given in [10]. Recently, there has been increased interest in exploiting massively-parallel compute devices like GPUs for this type of applications [9, 11, 24, 21, 12, 20].

Extensive research on PIM architectures has been performed in the 90s resulting in various architectures being proposed and explored, including Computational RAM [7], Intelligent RAM [19], DIVA [13], Gilgamesh [22], and FlexRAM [16]. Recently, a reviving interest can be observed (see, e.g., [18, 1]). Different application kernels have been mapped to these architectures to explore their performance, with focus on kernels that feature irregular memory access patterns. Similar to the approach taken for this work, we analysed the relevant kernels of a fluid dynamics code using the Lattice Boltzmann method and the Dirac operator from a Lattice Quantum Chromodynamics application and implemented these for the AMC architecture [3].

9 Conclusions

By porting relevant kernels of high-scalable density functional theory applications to the Active Memory Cube (AMC) we could demonstrate the potential of this architecture to be efficiently used for such scientific applications, where regular linear algebra and stencil operations dominate.

In particular, matrix-matrix multiplications can be executed efficiently even if it involves many tasks with small matrix dimensions. Using a suitable data layout for complex numbers a floating-point efficiency $\epsilon_{\mathrm{FP}} \gtrsim 80\,\%$ could be achieved, which is significantly above the efficiency of about 15 % observed on Blue Gene/Q$^{\mathrm{TM}}$ for the same kernel. When using KKRnano in its linear scaling mode with 2229 atoms in the interaction region on BG/Q, about 91 % of the time is spent in this kernel. Therefore, a single AMC has the potential to speed-up the overall application by a factor 5, while a single AMC is expected to consume about 5 times less power compared to a BG/Q processor. For this energy efficiency assessment the power consumed by the CPU is, however, not taken into account.

In addition to variants of matrix-matrix operations, we investigated a stencil operation that arises from 3D finite-difference derivatives and can be mapped to 1D stencils. Here, a floating-point efficiency $\epsilon_{\mathrm{FP}} = 43\,\%$ could be measured.

A speed-up of the juRS Hamiltonian depends on the balance between finite-difference kernel and other tasks, which in return depend on several input quantities. Thus a typical overall speed-up of the application is difficult to assess.

For any of the relevant kernels we observed a floating-point efficiency $\epsilon_{\mathrm{FP}} > 40\,\%$, which corresponds to a double-precision floating-point performance of at least 128 GFlop/s within an estimated power-envelope of 10 W. In particular the implementation of matrix-matrix multiplications with $\epsilon_{\mathrm{FP}} \simeq 80\,\%$ translates into 25 GFlop/s per Watt. This significantly exceeds power efficiencies on current architectures, it takes, however, only the power consumed by the AMC into account.

In regard of the hand-written microcode implementations generated for this investigation it remains unclear if compilers (once fully functional) will achieve similar performance numbers.

Acknowledgments

We thank the AMC team at IBM Research, in particular Jaime Moreno, for sharing their knowledge on the AMC, continued help and many fruitful discussions. We also acknowledge the collaboration of Stefan Blügel and his group.

References

1. Junwhan Ahn et al. PIM-enabled instructions. In *ISCA '15 Proceedings*, page 336.
2. S. Alam, C. Bekas, H. Boettiger, et al. *IBM J. Res. Dev.*, 57(1):161–169, Jan 2013.
3. P. F. Baumeister, H. Boettiger, J. R. Brunheroto, T. Hater, T. Maurer, A. Nobile, and D. Pleiter. In *High Performance Computing*, volume 9137 of *LNCS*. 2015.
4. Paul F. Baumeister. PhD thesis, RWTH Aachen University, 2012.
5. A. D. Becke. *J. Chem. Phys.*, 140(18), 2014 and references therein.
6. S. Blügel, D. Wortmann, et al. EIC Co-design Questionnaire for DFT. unpublished.
7. D. G. Elliott et al. Computational RAM. In *Proceedings IEEE 1992*, page 30.6.1.
8. R. W. Freund and N. Nachtigal. QMR. *Numerische Mathematik*, 60(1):315, 1991.
9. L. Genovese, M. Ospici, T. Deutsch, et al. *J. Chem. Phys.*, 131(3), 2009.
10. I. Girotto et al. *Enabling of Quantum ESPRESSO ...* PRACE, 2012.
11. M. Hacene et al. Accelerating VASP. *J. Comp. Chem.*, 33(32):2581–2589, 2012.
12. S. Hakala et al. volume 7782 of *LNCS*, pages 63–76. Springer, 2013.
13. M. Hall et al. Mapping irregular applications to DIVA. In *SC '99 Conf.*, page 57.
14. Hybrid Memory Cube Consortium. Hybrid Memory Cube specification, 2013.
15. JEDEC. JEDEC Standard High Bandwidth Memory(HBM) DRAM Spec., 2013.
16. Yi Kang, Wei Huang, et al. FlexRAM. In *ICCD '99 Conf.*, pages 192–201.
17. S. Kümmel and L. Kronik. *Rev. Mod. Phys.*, 80:3–60, Jan 2008.
18. R. Nair, S.F. Antao, et al. *IBM J. Res. Dev.*, 59(2/3):17:1–17:14, Mar 2015.
19. D. Patterson et al. A case for intelligent RAM. *Micro, IEEE*, 17(2), Mar 1997.
20. R. Solcà, A. Kozhevnikov, et al. In *SC '15 Proceedings*, pages 10:1–10:12.
21. F. Spiga and I. Girotto. phiGEMM. In *Euromicro 2012 Proceedings*, page 368.
22. T. L. Sterling and H. P. Zima. Gilgamesh. In *SC '02 Proceedings*, page 48.
23. A. Thiess, R. Zeller, et al. KKRnano. *Phys. Rev. B*, 85:235103, Jun 2012.
24. L. Wang, Yue Wu, Weile Jia, et al. In *SC '11 Proceedings*, pages 71:1–71:10.
25. R. Zeller. KKRnano. VSR Seminar (Oct 2014), JSC, FZ Jülich(Germany), 2014.
26. R. Zeller et al. *Phys. Rev. B*, 52:8807–8812, Sep 1995.