Exascale Scientific Applications: Programming Approaches for Scalability, Performance, and Portability: KKRnano

Paul F. Baumeister[1], Marcel Bornemann[2], Dirk Pleiter[1], and Rudolf Zeller[3]

[1] Jülich Supercomputing Centre, Forschungszentrum Jülich, 52425 Jülich, Germany
[2] Peter-Grünberg Institut, Forschungszentrum Jülich, 52425 Jülich, Germany
[3] Institute for Advanced Simulation, Forschungszentrum Jülich, 52425 Jülich, Germany

Abstract. Addressing certain materials science problems, e.g. inhomogeneous materials, using Density Functional Theory will require exascale compute capabilities as a sufficiently large number of atoms need to be simulated. In this chapter we consider a particular approach and an implementation that is optimized for extreme scale parallelism. We provide an overview on the Kohn-Sham approach as well as its application areas and discuss in detail the application KKRnano. Here we focus on recent efforts to port this application to GPU-accelerated architectures.

1 Introduction

Materials development for advanced 21st century applications will be supported more and more by an atom by atom understanding of the nanoscale origin of their mesoscopic and macroscopic properties. For this goal, beyond empirical concepts and experimental data, quantum mechanical calculations within Density Functional Theory (DFT) have proved to be an increasingly powerful tool. DFT [17, 19] treats the many-electron system by single-particle equations using the electron density instead of the many-electron wavefunction as the basic quantity. Although this represents an obvious simplification, calculations for systems with many atoms still represent a serious computational challenge since the computational effort in conventional density functional calculations increases with the third power of the number of atoms in the system. Systems with a few hundred atoms can be treated routinely today, but larger systems require enormous computer resources. Therefore, considerable effort has been spent in recent years to reduce the effort by novel techniques, which exploit the so-called nearsightedness of electronic matter [21] by utilizing the exponential decay of the density matrix. These techniques are continuously improved and with the advent of the next generation of supercomputers it will become possible to simulate the quantum effects, e.g., in entire semiconductor devices. These devices are currently scaled down below the ten-nanometer range and thus contain about 10^5 atoms.

One of the new computer codes, which avoids the unfavourable cubic scaling of the computational effort with system size, is KKRnano [23] which was designed from the outset to run efficiently on supercomputers with the IBM Blue Gene/P and Blue Gene/Q architectures. KKRnano, an all-electron code, which makes no use of pseudopotentials, is based on the full-potential Korringa-Kohn-Rostoker (KKR) multiple-scattering Green-function method as described in Ref. [20]. It avoids the cubic increase of computing time by adopting the techniques explained in Ref. [25] and has been used so far for systems containing as many as 10^5 atoms. Such calculations could be processed with up to 1.8 million parallel tasks on the 5.9-Petaflop installation of the Blue Gene/Q architecture in Jülich [1].

In the following section we provide background information on the methodology of KKRnano and explain differences compared to other approaches in the field. This is followed by a description of the algorithmic details of KKRnano and an overview on performance characteristics in Sec. 3. We continue with explaining our implementation (Sec. 4) and software practices (Sec. 5). Additionally, we present in Sec. 6 various performance results with particular focus on a scalability analysis on Blue Gene/Q and on GPU accelerated POWER 8 nodes. Conclusively, in Sec. 7 we report an analysis of energy efficiency, before we provide a summary and outlook towards exascale in Sec. 8.

2 Scientific Methodology

Contrary to most other DFT codes, which determine the Kohn-Sham wavefunctions by solving the differential Kohn-Sham wave equation, in KKRnano the Green function $G(\mathbf{r}, \mathbf{r}'; \epsilon)$ for the Kohn-Sham equation is obtained by solving the integral equation

$$G(\mathbf{r}, \mathbf{r}'; \epsilon) = \mathbf{G}^{\mathrm{ref}}(\mathbf{r}, \mathbf{r}'; \epsilon) + \int \mathrm{d}\mathbf{r}'' \mathbf{G}^{\mathrm{ref}}(\mathbf{r}, \mathbf{r}''; \epsilon) \left[\mathbf{v}_{\mathrm{eff}}(\mathbf{r}'') - \mathbf{v}^{\mathrm{ref}}(\mathbf{r}'')\right] \mathbf{G}(\mathbf{r}'', \mathbf{r}'; \epsilon) \tag{1}$$

where the integral extends over the space covered by the system. Here, $v_{\mathrm{eff}}(\mathbf{r})$ is the Kohn-Sham effective potential and $G^{\mathrm{ref}}(\mathbf{r}, \mathbf{r}'; \epsilon)$ the known Green function of a suitable reference system with potential $v^{\mathrm{ref}}(\mathbf{r})$. In KKRnano the reference system consists of constant repulsive potentials within non-overlapping spheres around the atoms. With this choice the reference Green function has two important properties: It decays exponentially with the distance $\mathbf{r} - \mathbf{r}'$ for ϵ values relevant in DFT calculations and secondly its angular-momentum representation around the atoms is easily obtained from the analytically known Green function for the system with zero potential.

By dividing space into non-overlapping cells around the atoms, the angular-momentum representation of the Green functions is used to separate the solution of (1) into intracell and intercell parts. The intracell part requires the solution of single-site integral equations where the integral extends only over the volume of a single cell. Since these equations are independent of each other, the computational effort for this part scales linearly with system size and can be parallelized in an efficient way. The intercell part requires to solve a large linear matrix equation

$$G^{nn'}_{LL'}(\epsilon_i) = G^{\mathrm{ref},nn'}_{LL'}(\epsilon_i) + \sum_{n''}\sum_{L''} G^{\mathrm{ref},nn''}_{LL''}(\epsilon_i) \sum_{L'''} \Delta t^{n''}_{L''L'''}(\epsilon_i) G^{n''n'}_{L'''L'}(\epsilon_i) \qquad (2)$$

for the Green function matrix elements $G^{nn'}_{LL'}(\epsilon_i)$. Here $\Delta t^{n}_{L''L'''}$ is the difference between single-cell scattering matrices in the system and the reference system. $G^{\mathrm{ref},nn'}_{LL'}$ are the known matrix elements of the reference Green function. The indices n and $L = (\ell, m)$ label cells and angular-momentum components while ϵ_i denotes integration mesh points used in the complex ϵ plane to obtain the fundamental quantity in DFT, the electronic density $n(\mathbf{r})$. In KKRnano $n(\mathbf{r})$ can be calculated very precisely, if the effective potential around each atom is understood as a non-local angular-projection potential [27] since for such potentials the dependence of $n(\mathbf{r})$ on the angular variables $\hat{\mathbf{r}} = \mathbf{r}/|\mathbf{r}|$ can be represented exactly [28] so that only the dependence on the radial variable $|\mathbf{r}|$ must be calculated numerically.

Massively parallel calculations with KKRnano have been used so far to investigate a variety of large disordered multicomponent alloys. Among others, these are dilute magnetic semiconductors, phase change materials and high-entropy alloys.

Dilute magnetic semiconductors are semiconductors doped with magnetic impurities. These systems are technologically interesting because they can be used for novel electronic devices, which utilize not only the electron charge but also the electron spin. For such so-called spintronic devices it is important that the materials are magnetic with Curie temperatures T_{C} as high as room temperature or above. Gadolinium-doped gallium-nitride is a system that has attracted significant attention because magnetic moments of 4000 μ_{B} per Gd atom and ferromagnetism above room temperature were claimed in experiments (μ_{B} is the Bohr magneton). KKRnano was used to study realistic models of Gd-doped GaN with co-doping by nitrogen or oxygen interstitials or Ga vacancies. It was found that only Ga vacancies provide a robust path to magnetic percolated clusters which can explain the ferromagnetism with rather large moments.

Phase-change materials, in particular alloys of germanium, antimony and tellurium, are basic materials for DVD and BluRay technology because laser heating can be used to switch their structure between crystalline and amorphous phases which have considerably different optical properties. KKRnano was used to investigate how vacancies and vacancy clusters can explain the experimentally observed metal-insulator transition in $\mathrm{GeSb_2Te_4}$. It was found that vacancy cluster, which appear in the insulating phase and are dissolved in the metallic phase, likely play an important role for the electronic resistance because they can act as strong scattering centers. Other interesting systems are phase-change materials doped by magnetic impurities because not only the electronic resistivity but also the magnetic state can be changed by switching between the two phases, which could be exploited for new multivalued memory devices. In calculations with KKRnano it was found that doping of $\mathrm{Ge_2Sb_2Te_5}$, particularly with chromium impurities, leads to a strong tendency for ferromagnetism with T_C values close to room temperature for large impurity concentrations.

High-entropy alloys consisting of four or more metallic elements, which crystallize in simple face-centered cubic lattices, constitute a relatively new class of materials with favourable properties like high hardness, wear resistance and corrosion resistance. With KKRnano energetics and magnetism in CrFeCoNi alloys were investigated. From the calculated local energies and their static fluctuations it can be concluded that completely random solutions are not stable. Instead, an $\mathrm{L}1_2$ structure is energetically preferred where the Cr atoms, which show the largest environmental effects with magnetic moments varying between $-1.7\mu_\mathrm{B}$ to $+0.8\mu_\mathrm{B}$, are at the cube corners and Fe, Co and Ni atoms are randomly distributed at the other sites.

Sparse Dyson Equation Formally, without indices, the Dyson equation (1) is equivalent to

$$X = \Delta t + \Delta t G^\mathrm{ref} X \tag{3}$$

where the matrix $X = \Delta t + \Delta t G \Delta t$ is known as the scattering path operator in the KKR method. Contrary to conventional KKR programs, in KKRnano this matrix equation is not solved by Gaussian elimination, but iteratively as

$$X^{(i+1)} = \Delta t + \Delta t G^\mathrm{ref} X^{(i)} \tag{4}$$

using the transpose-free quasi-minimal residual (TFQMR) method of Freund and Nachtigal [11]. The use of an iterative solution method is associated with several advantages. The first advantage is that the matrix elements $X^{nn'}_{LL'}$ can be determined separately for each atom n' and for each angular-momentum component L' which makes straightforward parallelization possible. The second advantage is that the exponential decay of the reference Green function $G^\mathrm{ref}(\mathbf{r}, \mathbf{r}', \epsilon)$ can be utilized easily. By neglecting exponentially small matrix elements of G^ref in (4) this matrix is turned into a sparse matrix where the number of non-zero matrix elements is proportional to $N_\mathrm{cl} N_\mathrm{at}$ instead of proportional to N^2_at as for a dense matrix. Here N_cl is the number of atoms in a cluster in the vicinity of each atom. Usually clusters consisting of 20 to 50 atoms are sufficient. [24] The sparse matrix multiplication in (4) requires order N^2_at operations which leads to a computational effort that increases only quadratically with the number of atoms and not cubically as in conventional DFT codes. The third advantage is that one can terminate the iterations, if the desired precision of the results is achieved. For instance, in DFT-based total-energy calculations the energy error is a few meV per atom, if the bound for the QMR residual norm is set to $||r|| = 10^{-3}$, and a few tenths of meV for $||r|| = 10^{-4}$ as shown in Ref. [25] for periodic model systems consisting of supercells with 16384 copper or palladium atoms.

Linear-scaling $\mathcal{O}(N)$ mode The computational effort in KKRnano, which scales quadratically with system size, if no compromise on precision is made, can be reduced to a linear scaling by exploiting that the Green function $G(\mathbf{r}, \mathbf{r}', \epsilon)$ decays with the distance $|\mathbf{r} - \mathbf{r}'|$. This decay is exponential for the complex values of

ϵ used in KKRnano, but considerably slower than the decay of the reference Green function $G^{\mathrm{ref}}(\mathbf{r},\mathbf{r}',\epsilon)$, so that matrix elements $G^{nn'}_{LL'}$ in (2) or $X^{nn'}_{LL'}$ in (4) are more important for larger distances between atom n and n' than matrix elements $G^{\mathrm{ref},nn'}_{LL'}$. Nevertheless, by trading some precision for computational speed, the matrix elements $X^{nn'}_{LL'}$ can be neglected beyond truncation regions of about $N_{\mathrm{tr}} > 1000$ atoms. This reduces the overall computational complexity in KKRnano to $N_{\mathrm{it}}N_{\mathrm{cl}}N_{\mathrm{tr}}N_{\mathrm{at}}$, where N_{it} is the number of TFQMR iterations, and makes calculations possible for large systems involving up to 100000 atoms. In order to assess which values of N_{tr} are reasonable, the dependence of the ground-state total energy on the number N_{tr} of atoms in the truncation region around each atom was studied with KKRnano. Since highly ordered pristine metallic systems are more demanding than insulators or disordered systems, the model systems used were constructed by a 32-times-repetition of simple cubic unit cells with four copper or palladium atoms in all three space directions. Using N_{tr} values between 55 and 34251 for the constructed periodic supercells with 131072 atoms, it was found that for small truncation regions with 55 atoms the error in the total energy can be as large as 0.1 eV per atom. While this accuracy can be acceptable for certain applications, the usual goal in DFT calculations is to determine the total energy with an accuracy of a few meV per atom. In KKRnano this can be achieved even for the difficult ordered systems with truncation regions containing a few thousand atoms [25, 26].

3 Algorithmic Details and Performance Characteristics

The following listing gives an overview on the sequence of tasks, which is executed during one run of KKRnano:

```
.1 read input.
.1 read potentials.
.2   compute general atom quantities.
.3     compute E-dependent atom quantities.
.3     setup reference system.
.4       setup scattering path operator.
.4       invert operator using TFQMR.
.5         invoke sparse operator times block vector.
.4       use diagonal elements of inverse.
.3     .
.2   compute density.
.2   solve Poisson equation for new potential.
.1 store potential.
```

In the Kohn-Sham DFT scheme the effective potential and the density must be determined self-consistently since they depend on each other in a non-linear manner. Usually they are calculated alternately in about 50 to 200 self-consistency steps. A typical run of KKRnano begins with reading some control parameters, the coordinates of the cell centers, the radial integration mesh in each cell and an initial guess for the effective potential. Before the self-consistency steps are started, it is necessary to calculate the so-called shape functions, which describe the geometric shape of the cells in an angular-momentum representation, and the Madelung coefficients, which are used to determine the electrostatic part of the effective potential from the charge multipole moments in each cell. These parts of the calculation require no communication and are easily parallelized over the cells.

During the self-consistency steps, the intracell part requires to solve systems of $(\ell_{\max}+1)^2$ coupled radial integral equations, as given in Ref. [27], in order to obtain $\Delta t^n_{LL'}(\epsilon_i)$ and to solve systems of linear equations of dimension $N_{\mathrm{cl}}(\ell_{\max}+1)^2$ to obtain $G^{r,nn'}_{LL'}(\epsilon_i)$. Here, $\ell_{\max}$ is a cutoff parameter which essentially determines the angular resolution of the projection potential in each cell by restricting the projection to a subspace of spherical harmonics with $\ell \leq \ell_{\max}$. The standard choice in KKRnano is $\ell_{\max} = 3$ leading to $(\ell_{\max}+1)^2 = 16$ since common experience with the KKR method has shown that this is enough for most purposes. The intracell part requires no communication and is easily parallelized over the cells.

For the subsequent intercell part the matrices $\Delta t^n_{LL'}(\epsilon_i)$ and $G^{\mathrm{ref},nn'}_{LL'}(\epsilon_i)$ with N_{cl} values of n' must be communicated to those other processors where they are needed to solve the Dyson equation (4) by the TFQMR method. The matrices X, Δt and G^{ref} in (4) are sparse matrices with X containing $N_{\mathrm{at}}N_{\mathrm{tr}}(\ell_{\max}+1)^4$ non-zero elements and G^{ref} containing $N_{\mathrm{at}}N_{\mathrm{cl}}(\ell_{\max}+1)^4$ non-zero elements while Δt contains non-zero blocks of size $(\ell_{\max}+1)^4$ only on the diagonal. During the TFQMR iterations, which require no communication and are easily parallelized over the cells, the sparsity structure of the matrices is exploited so that multiplication with zeros is avoided (see Sec. 4). After the TFQMR iterations the density and the charge multipole moments in each cell are calculated by simply summing up the Green-function contributions at the mesh points ϵ_i with appropriate integration weights.

For the calculation of the effective potential the charge multipole moments of each cell, a data set of $(2\ell_{\max}+1)^2 = 49$ numbers, must be communicated to all other cells. This small amount of data together with the $256(N_{\mathrm{cl}}+1)$ numbers, which must be transferred before the TFQMR iterations but only to the participating cells, indicates that the principle communication time in KKRnano is, in many situations, of not much importance.

In order to enable the use of more processors than atoms at the expense of additional communication, KKRnano optionally employs two other levels of parallelization besides the natural parallelization over the cells. One is over the two magnetic spin directions in ferromagnetic systems and the other is over the energy points ϵ_i. Since for physical reasons quite different numbers of TFQMR iterations are needed for different values of ϵ_i, a simple distribution to parallel tasks is inefficient. Instead, the ϵ_i are pooled in two or three groups where each group is treated by one thread. Load balancing is achieved by dynamically updating this grouping during the self-consistency steps.

Besides the coarse-grained parallelism described above, KKRnano can exploit additional parallelization levels, particularly for the most time-consuming part, the TFQMR iterations. For the Blue Gene/P JUGENE, which was installed at the Jülich Supercomputing Centre from 2008 to 2012, an additional parallelization over the 16 Ł components in the Dyson equation was implemented which allowed to utilize all 294912 processors available on this machine for a ferromagnetic nickel-palladium system containing 3072 atoms. For the Blue Gene/Q JUQUEEN, which has been operational at the Jülich Supercomputing Centre since 2012, the parallelization strategy for the iterative calculation of the matrix elements $X_{LL'}^{nn'}$ by using (4) was changed. Instead of calculating the columns labelled by L' by using several MPI processes, the block rows labelled by n are calculated by several OpenMP threads. The advantages of the OpenMP implementation are simple use of the hybrid programming model of the Blue Gene/Q, increased flexibility, because the number of block rows, which equals the number of atoms, is much larger than the number of columns per atom, reduced memory requirements, because the shared memory on the nodes is exploited, and increased flop-rate, because matrix-vector operations are replaced by matrix-matrix operations. This enabled efficient use of all 1835008 parallel threads available on Blue Gene/Q for a phosphorus-silicon system with 57344 atoms and 57344 empty cells.

In typical runs most of the time is spent in the TFQMR algorithm. Each iteration involves two matrix multiplications. The amount of floating point operations as well as the amount of data, which needs to be loaded and stored, are denoted by I_{fp}, I_{ld} and I_{st}, respectively:

$$I_{\mathrm{fp}} = 2N_{\mathrm{it}} \cdot \frac{N_{\mathrm{at}}}{N_{\mathrm{MPI}}} \cdot N_{\mathrm{tr}} N_{\mathrm{cl}} \cdot b^3 \cdot 8\,\mathrm{Flop}\,, \tag{5}$$

$$I_{\mathrm{ld}} = 4N_{\mathrm{it}} \cdot \frac{N_{\mathrm{at}}}{N_{\mathrm{MPI}}} \cdot N_{\mathrm{tr}} N_{\mathrm{cl}} \cdot b^2 \cdot 16\,\mathrm{Byte}\,, \tag{6}$$

$$I_{\mathrm{st}} = 2N_{\mathrm{it}} \cdot \frac{N_{\mathrm{at}}}{N_{\mathrm{MPI}}} \cdot N_{\mathrm{cl}} \cdot b^2 \cdot 16\,\mathrm{Byte} \tag{7}$$

where $b = (\ell_{\max} + 1)^2$. For more details see [9]. From these quantities the Arithmetic Intensity (AI) can easily be determined as $\mathrm{AI} = I_{\mathrm{fp}}/(I_{\mathrm{ld}} + I_{\mathrm{st}})$.

The original implementation supporting the $\mathcal{O}(N)$-mode by truncation was restricted to exactly one source atom per MPI process. In terms of programming, this involved much less bookkeeping than the current version that supports $n_{\mathrm{at}} = N_{\mathrm{at}}/N_{\mathrm{MPI}}$ source atoms per MPI process. One MPI process per atom also infers a large memory overhead related to global arrays, the MPI library requirements and, in particular, the scattering path operator. However, a lower total memory consumption can be achieved with the flexibility of several source atoms per MPI process since elements of the scattering path operator can be shared to a large extent. This also yields an improved performance of the sparse matrix multiplication as the arithmetic intensity (AI) of multiplying two square complex matrices of dimension 16 is 4 Flop/Byte. The multiplication of a square matrix and a rectangular matrix with the long dimension $16\,n_{\mathrm{at}}$ leads to an $\mathrm{AI}(n_{\mathrm{at}})$ that converges towards 5.32 Flop/Byte and is as high as 5.0 for five source atoms, c.f. Figure 1.

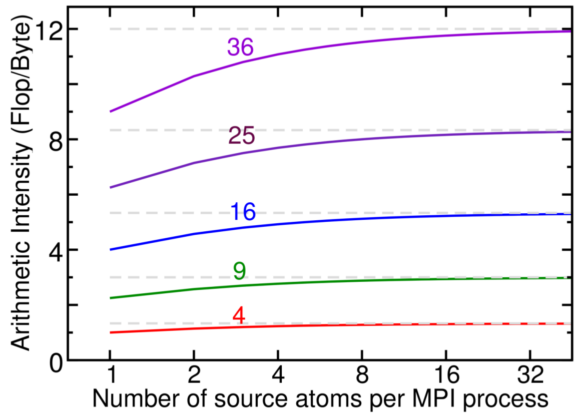


Fig. 1. The nominal arithmetic intensity (AI) for a matrix multiplication $A \times X$ increases for larger matrix dimensions. Here, $A \in \mathbb{C}^{b\times b}$ and $X \in \mathbb{C}^{b\times nb}$. The solid lines show the AI as a function of n for $b \in \{4, 9, 16, 25, 36\}$.

4 Programming Approach

The main programming language for KKRnano is Fortran and here, the usage of Fortran90 or Fortran95 is encouraged over Fortran77, in particular when adding new functionalities or restructuring old parts of the code. The character of Fortran as a domain specific language is particularly useful to avoid many explicit loops (Fortran90 `(:)`-syntax), natural handling of matrix data layout and matrix operations and intrinsic support of complex numbers. Furthermore, Fortran implies reasonable assumptions beneficial for compiler optimization, as e.g. assuming no pointer-aliasing between arguments. In KKRnano, derived data types assist to encapsulate data that belong to one module. This leads to relatively short argument lists and, hence, readable code. Especially for the less performance critical parts, this infers a structure that is easy to maintain. Each derived data type is defined in a Fortran module that exposes a creation and a destruction routine. By keeping memory management statements at well defined locations, it becomes easier to maintain the code and adapt data layout to new architectures.

The following code block gives an example of a simple derived data type:

```
module AtomicCoreData_mod
  private
  public :: AtomicCoreData, create, destroy

  type AtomicCoreData
    integer                        :: ellcore
    double precision               :: Ecore
    double precision               :: corecharge
    integer                        :: irmd
    double precision, allocatable :: rhocat(:,:)
  endtype

  interface create
    module procedure createAtomicCoreData
  endinterface

  interface destroy
    module procedure destroyAtomicCoreData
  endinterface

  contains

  subroutine createAtomicCoreData(self, irmd)
    type(AtomicCoreData), intent(inout) :: self
    integer,              intent(in)    :: irmd

    self%irmd = irmd

    self%ellcore = -1
    self%Ecore = 1.d9

    allocate(self%rhocat(irmd,2)) ! both spin directions
    self%rhocat = 0.d0
  endsubroutine

  elemental subroutine destroyAtomicCoreData(self)
    type(AtomicCoreData), intent(inout) :: self

    integer :: ist
    deallocate(self%rhocat, stat=ist)
  endsubroutine

endmodule
```

KKRnano exploits the Fortran90 feature of overloading names of routines (generic names) using the `interface` statement. With that, any derived data type in KKRnano can, for example, always be destructed by

```
  call destroy(core_state)
```

which causes all dynamically sized member fields to be deallocated. The `elemental` keyword even adds more comfort since that allows a single call onto an array of core state data items:

```
  call destroy(core_states(:))
```

In general, name overloading, encapsulation and the Fortran-module syntax for namespacing with the `only`-syntax allow for a well-structured approach towards large software packets written in Fortran90 that can be maintained with relatively low efforts and facilitate porting to new architectures.

The truncation approach which is made in KKRnano asks for a matrix storage format that reflects the resulting block sparsity pattern. To this aim we employ the Variable Block Row format (VBR) where only the elements in non-zero blocks alongside four pointer arrays for structure description are stored.[22] The pointer array KVST contains the global row index and simultaneously the global column index of each block row since in KKRnano only square blocks occur. The block-wise column index of each block is stored in JA. KA indexes into the beginning of each block in the array A where all elements from non-zero blocks are saved. Elements in IA point to the beginning of each block row in both JA and KA. KVST, IA and KA are supplemented by one concluding element to indicate the end of the matrix structure. In the example in Fig. 2 KVST has three elements plus the concluding element. The first three elements point to the beginning of each block row while the fourth element points behind the last block row to indicate where the matrix structure ends. As the example matrix is composed of five non-zero blocks, JA contains five entries and each entry gives the block column index of the corresponding block in row-major order, e.g. first block can be found in 1st block column, second block in 3rd block column, third block in 2nd block column etc.. KA also holds as many entries as non-zero blocks plus one concluding entry similar to that in KVST. Since in this example all blocks have equal size (2x2), there is a constant increment of 4 for the entries in KA. Finally, IA accomodates four pointer entries (three block rows plus concluding element) and each, apart from the last one, points to the start of a new block row in JA and KA.

Scalability KKRnano makes use of the Message Passing Interface (MPI) for exploiting the distributed memory parallelism. After each operator inversion information related to atoms, which belong to the same truncation cluster but are processed by different MPI ranks, needs to be exchanged. Therefore, point-to-point communication operations dominate. For typical work-loads the time spent in the inverter is significantly larger than the time required for communication.

For the implementation of more than one source atom per MPI process ($n_{\mathrm{at}} > 1$), the bookkeeping during communication becomes difficult, in particular using the truncation mode. Therefore, the communication of the ingredients required to construct the scattering path operator makes use of one-sided MPI communication routines. A call to `MPI_Win_create` allows the other MPI processes to access the locally stored quantities.

For testing the scalability of the $\mathcal{O}(N)$-mode, we selected an input deck with 2197 atoms. It features a supercell of 13^3 unit cells of pristine face centered cubic (FCC) crystal. This is large enough to perform truncation with a truncation cluster size N_{tr} of 1289, i.e. there are six shells of target atoms around each source atom.

The test indicated that about 32 % of the runtime for an energy point was spent in communication for the reference Green function. Here, 2197 MPI processes exchanged $13 \cdot 16 \cdot 16$ `double complex` with 1288 other processes,

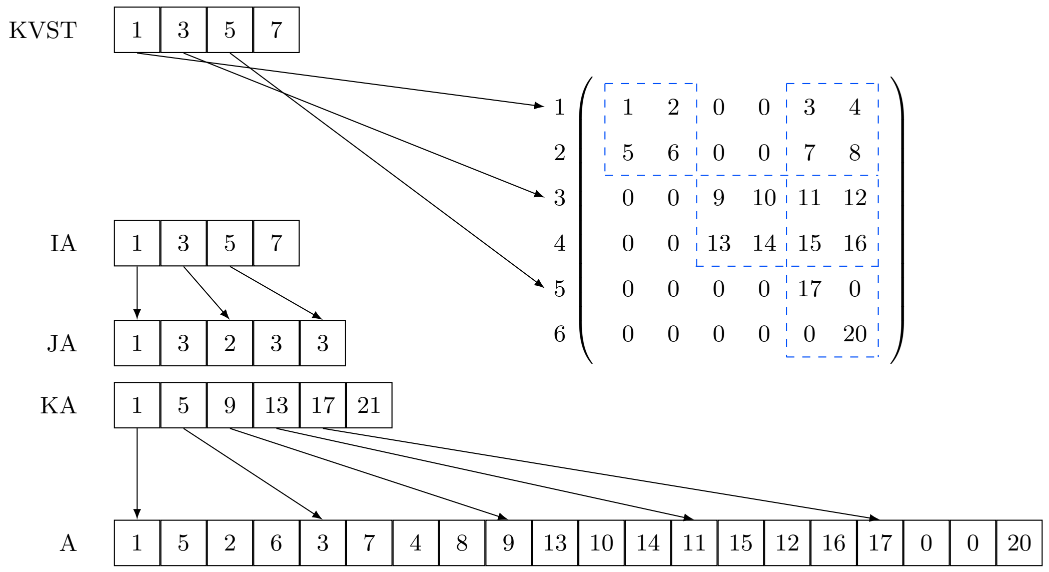


Fig. 2. A 6x6 sparse matrix consisting of nine 2x2-blocks stored in VBR format. The five non-zero blocks are highlighted in blue.

i.e. 280.66 GiByte per energy point were transferred over the network. The high percentage means that other ways of communication should be explored.

An experiment with direct point-to-point, non-blocking communication showed that the communication time can be reduced significantly. While the memory-saving one-sided communication routine took 2.2 sec, direct MPI messages were faster by 5×, so that only about 8.4 % of the iteration time were spent for communication.

In order to analyze both runs, an instrumentation with ScoreP [18, 7] generates profile summaries (or traces) that can be analyzed using Scalasca [6, 14, 13], see Figure 4. The number of visits to each function, the time spent in it and, very important for the analysis of the communication pattern, the number of bytes sent and received. However, the latter numbers are not given in the case of one-sided communication. Scalasca was used to extract the time spent in the communication functions. As each process delivers a timing result for each function call that has not been filtered out, the display of the distribution of time values is particularly useful, see rightmost panel in Figure 4.

I/O It is often underestimated how I/O can limit scalability of an application. KKRnano performs tasking-local POSIX I/O operations, which on currently available parallel file systems do not scale. To mitigate this problem we use SIONlib [12], which is a library that combines accesses to one file per MPI task into accesses to shared files. It requires only minimal changes to the application as it allows to continue using the most popular functions for reading and writing.

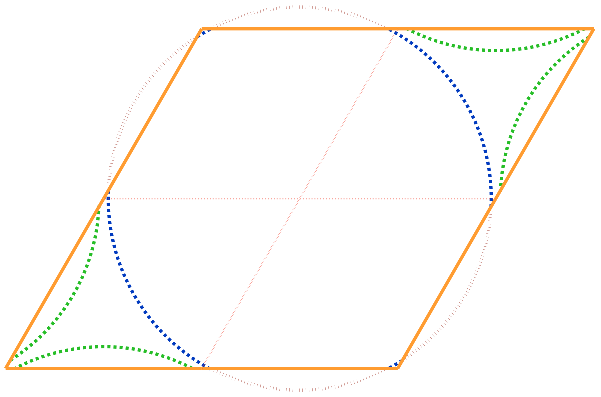


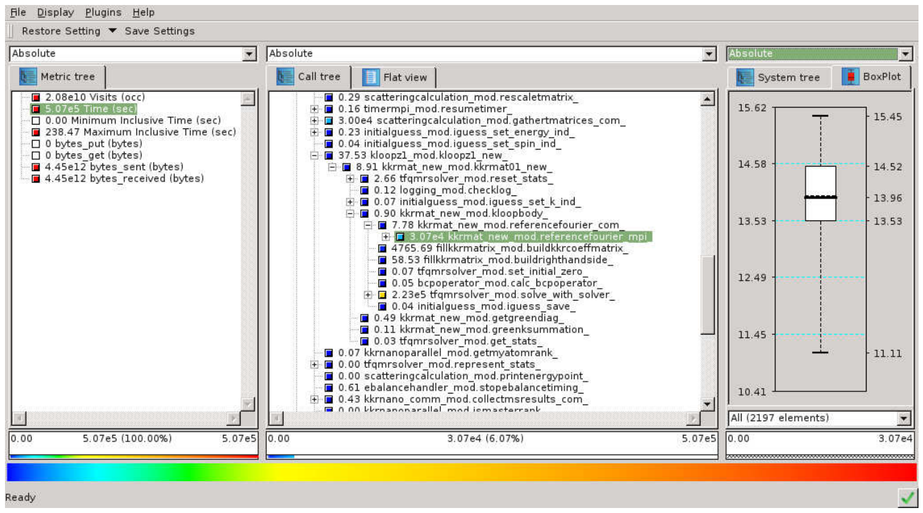
Fig. 3. The truncation sphere with a radius of six lattice constants contains 1289 target atoms in 3D using a supercell of 13^3 FCC unit cells, i.e. 2197 source atoms. This input deck is designed so that truncation spheres do not overlap with their periodic images and is used for profiling the application in $\mathcal{O}(N)$-mode.

GPU Acceleration One of the most promising approaches for realizing exascale computers is the use of compute devices, which are operated at relatively low clock frequency but are capable to perform an extremely larger number of floating-point operations each clock cycle. One increasingly popular example of such devices are Graphics Processing Units (GPU). Due to the low clock frequency they feature a throughput of floating-point operation versus power consumption ratio that is much higher compared to standard CPUs. Efficient exploitation of such devices requires applications with a high level of parallelism and large Arithmetic Intensity. These are exactly the features of the TFQMR solver in KKRnano. For this reason this solver has been ported to GPUs [9] in order to be able to exploit the performance of GPU-accelerated HPC systems.

This custom-tailored solver is written in C++ employing the CUDA toolkit provided by NVIDIA. The basic structure of the existent TFQMR can be adopted but data transfer from host to device and vice versa has to be taken care of and matrix operations must be implemented in a CUDA-conform manner. The latter can be easily achieved by using cuBLAS routines as they are included in the CUDA toolkit. However, tests showed that in our area of application where sparse matrices are essential self-written algorithms can outperform those routines. Therefore, KKRnano does not rely on library function calls but is equipped with its own custom-built linear algebra routines that handle matrices stored in a block sparse format.

Portability As discussed above, a typical work-load based on KKRnano spends most of it's time in the TFQMR solver. The default implementation of this solver comprises 400 lines-of-code. For this reason we can apply the following portability strategy:

– The bulk of the code is kept Fortran 95 compliant.
– Parallelization of the application is compliant to MPI version 3 and OpenMP version 3.
– Optionally, KKRnano can be linked to specialised versions of the TFQMR solver.



```
2.08e10 Visits (occ)
5.07e5 Time (sec)
0.00 Minimum Inclusive Time (sec)
238.47 Maximum Inclusive Time (sec)
0 bytes_put (bytes)
0 bytes_get (bytes)
4.45e12 bytes_sent (bytes)
4.45e12 bytes_received (bytes)
```

```
   0.29 scatteringcalculation_mod.rescaletmatrix_
 + 0.16 timermpi_mod.resumetimer_
 + 3.00e4 scatteringcalculation_mod.gathertmatrices_com_
 + 0.23 initialguess_mod.iguess_set_energy_ind_
   0.04 initialguess_mod.iguess_set_spin_ind_
 - 37.53 kloopz1_mod.kloopz1_new_
   - 8.91 kkrmat_new_mod.kkrmat01_new_
     + 2.66 tfqmrsolver_mod.reset_stats_
       0.12 logging_mod.checklog_
     + 0.07 initialguess_mod.iguess_set_k_ind_
     - 0.90 kkrmat_new_mod.kloopbody_
       - 7.78 kkrmat_new_mod.referencefourier_com_
         + 3.07e4 kkrmat_new_mod.referencefourier_mpi
         4765.69 fillkkrmatrix_mod.buildkkrcoeffmatrix_
         58.53 fillkkrmatrix_mod.buildrighthandside_
         0.07 tfqmrsolver_mod.set_initial_zero_
         0.05 bcpoperator_mod.calc_bcpoperator_
       + 2.23e5 tfqmrsolver_mod.solve_with_solver_
         0.04 initialguess_mod.iguess_save_
       0.49 kkrmat_new_mod.getgreendiag_
       0.11 kkrmat_new_mod.greenksummation_
       0.03 tfqmrsolver_mod.get_stats_
   0.07 kkrnanoparallel_mod.getmyatomrank_
 + 0.00 tfqmrsolver_mod.represent_stats_
   0.00 scatteringcalculation_mod.printenergypoint_
   0.61 ebalancehandler_mod.stopebalancetiming_
 + 0.43 kkrnano_comm_mod.collectmsresults_com_
   0.00 kkrnanoparallel_mod.ismasterrank
```

Fig. 4. Cube viewer for profiles analyzed by the Scalasca tool. Metrics are selected in the left panel, the investigated function is shown embedded into its call tree in the center panel and statistics about the time per call or other quantities are shown on the right.

The application can thus easily be ported to any platform supporting Fortran 95, MPI and OpenMP. These requirements are sufficiently easy to fulfill such that KKRnano runs both, on almost any HPC system as well as Linux laptop computers. In fact, it has been ported to a variety of HPC systems, including IBM Blue Gene, different clusters based on Intel Xeon processors or IBM POWER processors (with or without GPUs). As this does not guarantee the required high level of performance for HPC systems, we do foresee specialised versions of the solver, which might be implemented in a non-portable way. One example for this approach is our GPU-accelerated version of this solver, which is implemented in CUDA (see section 4). The efforts for creating such specialised code versions is typically moderate and thus a good balance between additional efforts and performance gains can be obtained.

External Libraries Wherever possible, standard linear algebra is performed in BLAS [3] and LAPACK [5] routines. Here, vendor-tuned libraries, where available (MKL/ESSL), deliver good performance. In particular, matrix-matrix multiplications for complex matrices (`zgemm`) are used intensively for small and mid-sized matrices. Another essential ingredient are inversion routines like `zgetrf`, `zgetrs` or `zgetri`, which make use of efficient vendor-provided implementations on matrices of dimension $b \cdot N_{\mathrm{cl}}$.

5 Software Practices

The development of KKRnano is facilitated by a software repository running the version control system `git`. The corresponding `git`-server [4] allows distributed development and a simple management of source code versions and documentation. The software project management system `TRAC` [8] is used as a web front end for meeting minutes and ticketing besides uncomplicated access to single files from different uploaded branches.

Conditional Compilation and Metaprogramming KKRnano makes use of the preprocessor. This means that most source files carry the suffix `.F90` rather than `.f90`. This envokes the Fortran-internal preprocessor. It is used here to deactivate code that was included for testing purposes with `-D NDEBUG` and activates machine specific solutions in selected routines.

As mentioned earlier, Fortran90 supports generic functions that allow for well-readable high-level routines. Nevertheless, C++-like template programming is not supported. Some KKRnano code structure, however, would benefit from such language capabilities. The workaround for this missing feature of the programming language is code generation with small Python scripts or text replacement with the Linux command line tool `sed`. The latter is used for communication routines. Besides all the advantages to the ease of programming, the drawback of an explicit routine interface that is being checked during compilation is that interfaces for many data types need to be generated. Due to the absence of template programming, KKRnano generates communication routines for `double precision`, `double complex`, `integer` using text replacement with `sed`. Although solutions using the preprocessor are viable, `sed` has the advantage that the other preprocessor directives survive the manipulation.

Furthermore, a Python script is used to generate the routine that reads the input file. Since various data types and combinations (with or without default value) are required for the user control of KKRnano, the input file reader is regenerated by the Python code when an input file keyword is added or changed or default values are adjusted.

Automatic Documentation The KKRnano documentation is twofold: Most features are described for practical reasons in the source code following the Doxygen convention [16]. However, some issues which should be addressed in more detail as well as small tutorials are available in a 'DokuWiki' [2]. The importance of a good documentation cannot be overestimated since in KKRnano a lot of parameters have to be introduced whose meaning and significance in the bigger context is not necessarily clear at first sight. Therefore, parameters that are in KKRnano commonly defined within F90 types are extensively described and the purposes of subroutines are explained if instructive.

Verification and Validation The correctness of the code is verified using a set of regression tests collected in `tests.py`. Here, the test cases inherit from `unittest` and compare the total energy results of serial runs on small input decks (up to

16 source atoms) with precalculated values. Furthermore, verification of the MPI parallelisation can be switched on checking if the results stay unchanged when source atoms are distributed to up to a single source atom per process.

The script `tests.py` also provides validation: The convergence with respect to $\ell_{\max}$ is checked for a single atom of copper in face centered cubic symmetry. At the same time, this test is used to reveal implementation errors when compiled with the `-checkbounds` option.

The full set of regression tests runs about 20 minutes on a workstation. Single subtests can be as fast as 2 seconds.

Distribution and Licensing Access to the KKRnano-repository is currently available in the context of scientific collaboration with the Institute for Advanced Simulation since the code has not yet been released under a public license.

Build System KKRnano uses traditional makefiles. A single `Makefile` can take various configuration options to specify the flavor of executable one wants to build. The options control the machine type to compile for, the compilers used and the compiler flags. In particular the preprocessor macros are controlled here, whereas some of them will be set depending on the machine type. The dependency table of the source files is explicitly expressed in the makefile. This is necessary to ensure that Fortran modules are already refreshed and present in the module directory before compiling a source file that includes them via `use`-statement. The dependency table can be constructed using a tool envoking `grep` on all source files. A prerequisite here is that developers have to stick to the rule that only one module is defined inside one source file and the name of the module matches the name of file (except for the `.F90` suffix and case sensitivity). Parallel compilation (`make -j`) allows a full recompilation plus linking in about 15 seconds (`-O3`) on a workstation with 4 cores.

6 Performance Results

Scaling on Blue Gene/Q A weak scaling analysis for KKRnano has been conducted on the Blue Gene/Q installation JUQUEEN at Jülich Supercomputing Center. Its peak performance is 5.9 PetaFlop/s. Here, the code has been applied to a host crystal of silicon with a single impurity atom of phosphorous that adds a shallow donor state into the band gap of the semiconductor. Meaningful input decks can easily be generated for any size of the host material. The lattice structure of Si is the diamond structure, which exhibits relatively large void spaces in between the atoms. In order to describe it properly, a body-centered structure is set up so that there are two Voronoi cells per atom, one containing an atomic core and one for the void. The results of the measurements with four OpenMP threads per MPI process can be seen in Figure 5. As the characteristic scaling behaviour of the order(N)-mode starts when the system is larger than the truncation cluster, the measurements begin at two thousand cells. Despite some deviations from the linear behaviour that might stem from suboptimal job partitions on the machine, the solver time per cell is around 645 seconds for $1k$, $8k$ and $65k$ atoms.

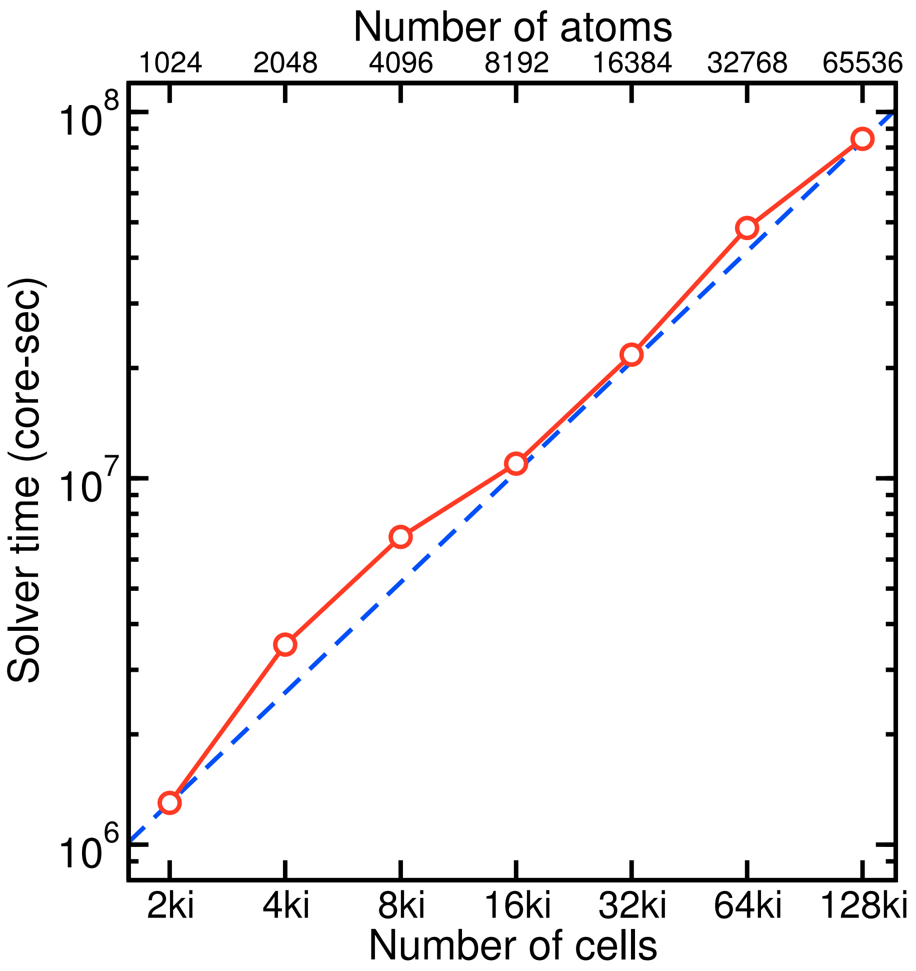


Fig. 5. Blue Gene/Q weak scaling of a the multiple scattering solver for a single atom of phosphorous in a host crystal of silicon. Two cells per atom have been used to describe the diamond structure with bcc Voronoi cells, one MPI process per cell and four OpenMP threads per MPI process. Around 2000 cells were inside the truncation cluster, 1.7 % for the largest calculation.

Performance on GPU-accelerated POWER8 nodes An in-depth performance analysis of the GPU-accelerated TFMQR solver on server based on POWER8 processors and NVIDIA Tesla K40m GPUs has been published in [9]. To improve utilization of the GPU it is required to use the multi-process service (MPS) feature to run multiple solvers concurrently. Benchmark data shows that at least

Table 1. Times t_{POSIX} and t_{SIONlib} (in seconds) on the Blue Gene/Q JUQUEEN used to write the effective potential task locally to non-shared files and to a shared SIONlib file for a system with 114688 cells using 114688 MPI tasks and a varying number of OpenMP threads per MPI task.

N_{OpenMP}	N_{tasks}	t_{direct}	t_{sionlib}
2	229376	101	10.1
4	458752	166	11.1
8	917504	434	7.3
16	1835008	831	8.1

four GPU tasks are required to maximize performance on a K40m GPU for a typical problem size (see Fig. 6). It can also be observed that performance is best when the number of atoms in the truncation cluster N_tr is 1000 or more. This can be ascribed to the GPU architecture which favours large matrices as they appear when working with a big truncation cluster.

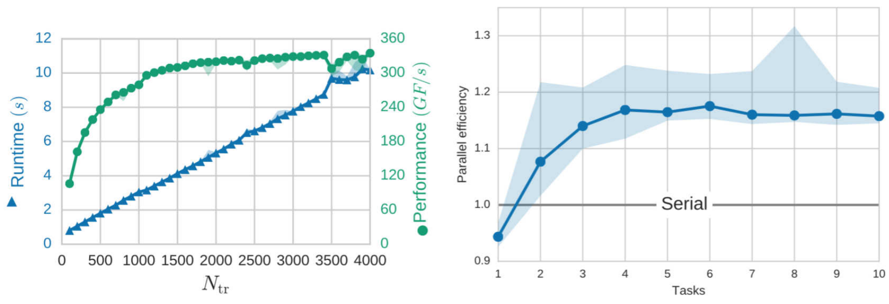


Fig. 6. a) Performance of GPU-TFQMR solver using 1000 iterations with varying truncation cluster size N_{tr}. b) Efficiency of the multi-process service (`mps`) on an NVIDIA K40m

I/O performance using SIONlib The standard output of KKRnano consists of the effective potential, the charges (in angular-momentum decomposition), the total-energy contributions and the forces on the atoms. In the original version of KKRnano the output to the file system was implemented using task-local, unformatted Fortran writes to non-shared files. The individual record for each cell was written by the MPI process responsible for that cell. While charges, total-energies and forces are needed only after the last self-consistency step, the situation is different for the effective potential. Usually, for monitoring and improving convergence of the self-consistency steps about 15 steps are done in one production job which requires that the potential is written to the file system at least once per job. It is, however, desirable to write the potential after every self-consistency step to obtain checkpoint data. They can be used to restart the calculation, for instance, after an unexpected crash of a production job or if the self-consistency process was manually terminated because its behaviour changed from convergent to divergent.

Although the potential comprises only a few hundred kBytes per cell, for large systems the use of separate files for each MPI rank severely limits the scalability. This is obvious in Table 1 where the third column shows the time which was used to write about 10 GByte for a system with 114688 cells using between two and sixteen OpenMP threads per MPI process. The time for output approximately doubles if the number of processing units is doubled unless SIONlib is used.

To alleviate this problem we use SIONlib [12], which is a library that prevents file-system overhead arising from hundreds of thousands of task-local files by using a small number of shared files. SIONlib employs a communication layer to aggregate metadata among the tasks and exploits the I/O infrastructure and the file-system properties. The use of SIONlib in KKRnano, which required only minimal changes to the code by replacing the standard functions for reading and writing by the appropriate SIONlib functions, gave impressive improvements for the output time as the fourth column of Table 1 shows. The time for output remains almost constant at a level far below the total time used in a self-consistency step.

The current version of KKRnano is expected to scale to a significantly larger number of tasks without being limited by I/O performance. There is more room for improvements by better tuning of the SIONlib parameters. No fine-tuning was attempted for Table 1 where only a single, shared SIONlib file was used. Tests using eight SIONlib files gave reduced times of 3.6 and 2.7 seconds for 229376 and 458752 tasks.

7 Energy Considerations

The combination of a throughput-optimized compute accelerator with a general-purpose CPU makes the choice of clock speeds of both chips an important task. As shown by Hater *et al.* [15] on a heterogeneous compute node with two POWER8 CPUs and two K40m GPUs, dynamic voltage and frequency scaling (DVFS) can be tuned to minimize the energy to solution. Here, the POWER8 processors are featuring a rich cache hierarchy and also offer a substantial floating point performance, about 20 % that of the GPUs. The energy to solution becomes a function of the CPU frequency. Despite the high base power consumed by the main memory, the best choice of CPU frequency is not the highest possible. Measurements of the total power consumption of the compute node (dots in Figure 7) infer that the POWER CPUs should solve the iterative inversion problem at 90 % of the max. CPU frequency to achieve the highest energy efficiency. The findings can be explained by an energy model (solid line in Figure 7) that is derived from a performance model extended by prefactors for the power consumption that are extracted from device-resolved power measurements. For the GPU solver, the optimal frequency is the maximum of 875 MHz and the energy to solution for the input parameters used in the investigation is lower by about 40 % compared to the CPU solver.

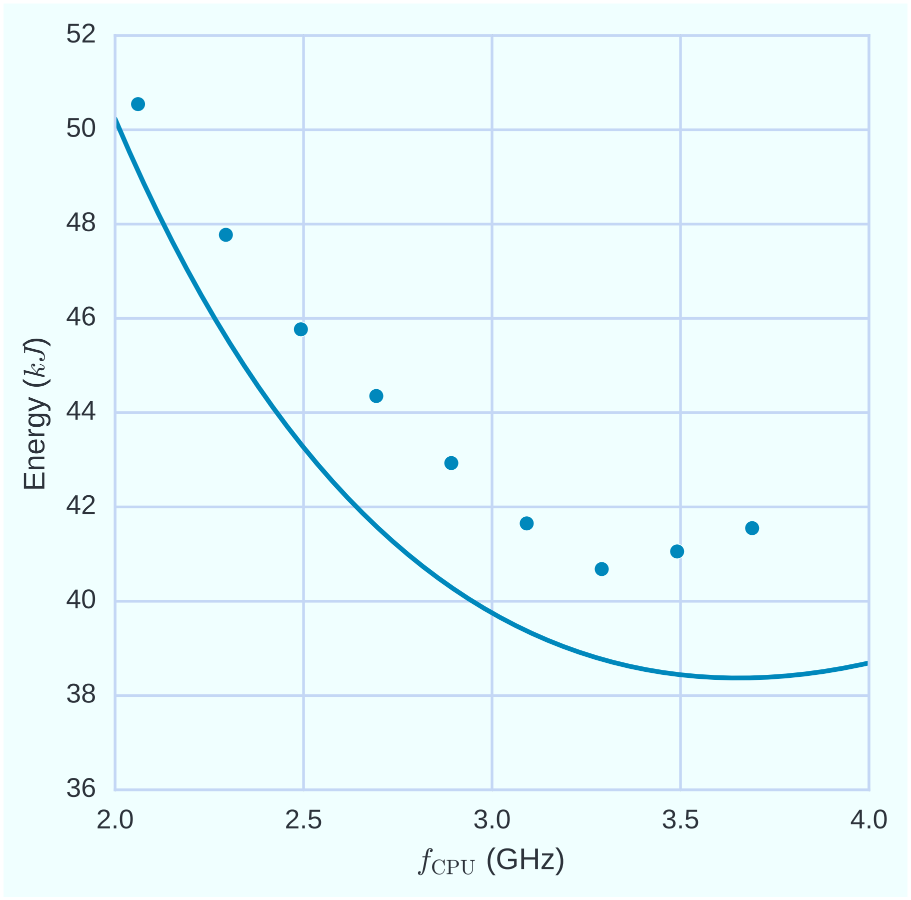


Fig. 7. Energy to solution for the QMR solver running on POWER8 processors as a function of the CPU frequency. Dots show the results of power measurements while the solid line represents the energy model. Figure from Hater *et al.* [15].

8 Summary and Outlook

In this chapter we provided an introduction into the Korringa-Kohn-Rostoker method and described its features, which allows for addressing challenging problems in materials science. One important feature is the extreme scalability that can be achieved on massively-parallel computers, which was demonstrated with one specific implementation of this method, namely KKRnano. We consider it as a showcase that a systematic approach to increase the parallelism of an application pays off.

For KKRnano we presented details of the implementation and, in particular, aspects on how it is parallelized. Furthermore, we discuss our programming approach aiming for both, maintainability as well as portability of the code. This allows us to run the code on desktop systems for code development and on HPC systems of the highest performance class when using KKRnano for research. On the latter systems a very high level of scalability could be demonstrated, e.g. on an IBM Blue Gene/Q system at Jülich Supercomputer Centre with 458,752 cores.

Based on an analysis of the properties of the key algorithms and their implementation in KKRnano we expect that the level of parallelism can be further significantly increased. This will be crucial to exploit future architectures, e.g. architectures including GPUs as compute accelerators. These devices feature an extremely large throughput of floating-point operations per clock cycle. As it can be expected that this hardware level parallelism will further increase, KKRnano is in a good position to exploit these future compute technologies efficiently. This could already be demonstrated using node architectures, which are similar to those of the Summit supercomputer, which will be installed at Oak Ridge National Lab. The simple properties of the most performance critical kernel, namely the operator solver based on the TFMQR algorithm, makes this application also suitable for unconventional architectures. An interesting example are processing-in-memory architectures, where data transport is avoided by integrating compute capabilities into the memory, which can help to significantly reduce energy-to-solution [10]. As power consumption is expected to become the major limiting factor for further increase of HPC system performance, it will become more and more important to explore such unconventional architectures towards exascale computing.

Acknowledgments

We would like to thank all contributors to the application, in particular Alexander Thiess and Elias Rabel, who proved the linear scaling and brought the code into shape. For support on the Blue Gene systems, especially focussed to I/O, we thank Wolfgang Frings and Kay Thust from the Jülich Supercomputer Centre. We would also like to acknowledge the efforts towards the GPU implementation by Thorsten Hater in the context of the Exascale Innovation Centre and the POWER Acceleration and Design Center. Finally, we thank Jiri Kraus (NVIDIA) for consultancy.

References

1. http://www.fz-juelich.de/ias/jsc/EN/Expertise/High-Q-Club/node.html/, accessed: 2016-08-21
2. https://www.dokuwiki.org/
3. BLAS Basic Linear Algebra Subprograms. http://www.netlib.org/blas/, accessed: 2016-07-20
4. git. https://git-scm.com/, accessed: 2016-07-20
5. LAPACK – Linear Algebra PACKage. http://www.netlib.org/lapack/, accessed: 2016-07-20
6. Scalasca. http://www.scalasca.org/, accessed: 2016-07-21
7. Score-P. http://www.vi-hps.org/projects/score-p/, accessed: 2016-07-21
8. The Trac project. https://trac.edgewall.org/, accessed: 2016-07-20

9. Baumeister, P.F., Bornemann, M., Bühler, M., Hater, T., Krill, B., Pleiter, D., Zeller, R.: Addressing Materials Science Challenges Using GPU-accelerated POWER8 Nodes, pp. 77–89. Springer International Publishing, Cham (2016)
10. Baumeister, P.F., Hater, T., Pleiter, D., Boettiger, H., Maurer, T., Brunheroto, J.R.: Exploiting In-Memory Processing Capabilities for Density Functional Theory Applications. Springer International Publishing, Cham (2016)
11. Freund, R.W., Nachtigal, N.: QMR: a quasi-minimal residual method for non-Hermitian linear systems. Numerische Mathematik 60(1), 315 (1991)
12. Frings, W., Wolf, F., Petkov, V.: Scalable Massively Parallel I/O to Task-Local Files. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis. pp. 17:1–17:11. SC '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.114/1654059.1654077
13. Geimer, M., Hermanns, M.A., Siebert, C., Wolf, F., Wylie, B.J.N.: Scaling Performance Tool MPI Communicator Management, pp. 178–187. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
14. Geimer, M., Wolf, F., Wylie, B.J.N., Ábrahám, E., Becker, D., Mohr, B.: The scalasca performance toolset architecture. Concurrency and Computation: Practice and Experience 22(6), 702–719 (2010), http://dx.doi.org/10.1002/cpe.1556
15. Hater, T., Anlauf, B., Baumeister, P., Bühler, M., Kraus, J., Pleiter, D.: Exploring Energy Efficiency for GPU-Accelerated POWER Servers. Springer International Publishing, Cham (2016)
16. van Heesch, D.: git. http://www.doxygen.org/, accessed: 2016-07-20
17. Hohenberg, P., Kohn, W.: Inhomogeneous electron gas. Phys. Rev. 136, B864–B871 (Nov 1964), http://link.aps.org/doi/10.1103/PhysRev.136.B864
18. Knüpfer, A., Rössel, C., Mey, D.a., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A., Nagel, W.E., Oleynik, Y., Philippen, P., Saviankou, P., Schmidl, D., Shende, S., Tschüter, R., Wagner, M., Wesarg, B., Wolf, F.: Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope,Scalasca, TAU, and Vampir, pp. 79–91. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
19. Kohn, W., Sham, L.J.: Self-consistent equations including exchange and correlation effects. Phys. Rev. 140, A1133–A1138 (Nov 1965), http://link.aps.org/doi/10.1103/PhysRev.140.A1133
20. Papanikolaou, N., Zeller, R., Dederichs, P.H.: Conceptual improvements of the KKR method. J.Phys.: Condens. Matter 14(11), 2799–2824 (2002)
21. Prodan, E., Kohn, W.: Nearsightedness of electronic matter. Proc. Natl. Acad. Sci. USA 102, 11635–11638 (2005)
22. Saad, Y.: Sparskit: a basic tool kit for sparse matrix computations - version 2 (1994)
23. Thiess, A., Zeller, R., Bolten, M., Dederichs, P.H., Blügel, S.: Massively parallel density functional calculations for thousands of atoms: KKRnano. Phys. Rev. B 85, 235103 (Jun 2012)
24. Zeller, R.: Evaluation of the screened Korringa-Kohn-Rostoker method for accurate and large-scale electronic-structure calculations. Phys. Rev. B 55, 9400–9408 (April 1997)
25. Zeller, R.: Towards a linear-scaling algorithm for electronic structure calculations with the tight-binding Korringa-Kohn-Rostoker Green function method. J. Phys.: Condens. Matter 20(29), 294215 (2008)
26. Zeller, R.: Linear scaling for metallic systems by the Korringa-Kohn-Rostoker multiple-scattering method. In: Papadopoulos, M.G., Zalesny, R., Mezey, P.G.,

Leszczynski, J. (eds.) Linear-Scaling Techniques in Computational Chemistry and Physics: Methods and Applications, pp. 475–505. Challenges and Advances in Computational Chemistry and Physics, Springer Netherlands, Dordrecht (2011)

27. Zeller, R.: Projection potentials and angular momentum convergence of total energies in the full-potential Korringa-Kohn-Rostoker method. J. Phys.: Condens. Matter 25(10), 105505 (2013)
28. Zeller, R.: The Korringa-Kohn-Rostoker method with projection potentials: exact result for the density. J. Phys.: Condens. Matter 27(30), 306301 (2015)