Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Optimizing the post-Wannier Berry-phase Code for Optical and Anomalous Hall Conductivities and Orbital Magnetization

Thomas Ponweiser[a,*], Małgorzata Wierzbowska[b,†]

[a]*Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Altenberger Straße 69, 4040 Linz, Austria*
[b]*Institute of Physics, Polish Academy of Science, Al. Lotników 32/46, 02-668 Warsaw, Poland*

Abstract

WANNIER90 is a quantum-mechanics code for the computation of maximally localized Wannier functions, ballistic transport, thermoelectrics and Berry-phase derived properties – such as optical conductivity, orbital magnetization and anomalous Hall conductivity. In this whitepaper, we report on optimizations for WANNIER90 carried out in the course of the PRACE preparatory access project PA2231. Through performance tuning based on the integrated tool suite *HPCToolkit* and further parallelisation of runtime-critical calculations, significant improvements in performance and scalability have been achieved. Previously unfeasible computations with more than 64 atoms are now possible and the code exhibits almost perfect strong scaling behaviour up to 2048 processes for sufficiently large problem settings.

Keywords: wannier90, maximally-localized Wannier functions, Berry phase, anomalous Hall conductivity, orbital magnetization, performance optimization, HPCToolkit

1. Introduction

For the discovery and design of new, improved materials – e.g. for transparent electrodes of solar cells or for non-volatile magnetoresistive random access memories (MRAM) – simulation codes, capable of predicting desired and undesired material properties, are an inevitable complement to practical experimentation. One of such simulation codes is WANNIER90, which is based centrally on the theory of maximally localized Wannier functions (MLWFs). In this whitepaper, we report on several optimizations for WANNIER90, which have been carried out in the course of the PRACE preparatory access project PA2231 on the PRACE Tier-0 System *SuperMUC* at the Leibniz Supercomputing Centre (LRZ) located in Garching near Munich, Germany.

This report is structured in the following way: The remainder of this section is devoted to a short introduction to the theory of MLWFs and its applications. In Section 2, we introduce the software package WANNIER90 and briefly discuss its main characteristics from a purely software-engineering point of view. Moreover, the main objectives of this project are defined in that section. Section 3 discusses the most important code changes carried out in the course of this project as well as their impact on the performance of WANNIER90. Section 4 complements this discussion with scalability analyses and a detailed comparison of the original and new code version.

1.1. Maximally Localized Wannier Functions

Wannier functions (WF) give real space representation of Bloch functions characterizing a crystal in the momentum space (so-called reciprocal space). WFs were introduced in 1937 [1] for the very special case of

* Principal PRACE expert, E-mail address: Thomas.Ponweiser@risc-software.at
† Principal investigator, E-mail address: wierzbowska@ifpan.edu.pl

isolated impurity states, and later were extended to all semiconductor or insulator states in 1997 [2], and metallic systems as well in 2001 [3]. The recipe on how to find WFs rests on a certain localization criterion, therefore they are called *maximally-localized Wannier functions* (MLWFs).

There are many applications of MLWFs. Most important, various observables can be calculated quickly using very accurate interpolation in the MLWFs basis. Examples for such calculations include band structures and Fermi surfaces (including topological insulators), phonon spectra and electron-phonon couplings, excited-state properties by means of the so-called GW-method, van der Waals corrections, the thermoelectric properties, and the properties derived from the Berry phase, such as: anomalous Hall effect (AHE), orbital magnetization and optical conductivity [7,8]. The latter three effects are subject of our investigations and the focus of the optimization work in this project.

Optical conductivity can for example tell as whether a certain material of interest is suitable for transparent electrodes of solar cells. As an example, one might consider using elastic graphene or doped graphene (for better metallicity and elasticity) instead of popular tin oxide doped with Zn [10].

Orbital magnetization can cause a material to be magnetic although it does not contain any magnetic elements (such as Fe, Mn, Ni or Co). When electrons move along a closed loop, magnetic moments arise. This effect is for example observable for graphene nanoribbons, depending on the width and chirality (i.e. the cutting direction through graphene chosen to define the nanoribbon) [11].

Anomalous Hall conductivity (AHC) is the conductivity similar to the ordinary Hall effect – where the electrons and holes move in a crystal with some component perpendicular to the applied voltage and the magnetic field. But in the case of AHC, instead of the external magnetic field, the intrinsic magnetization of the measured sample acts as the driving force [12]. The technical application of this effect can be in magnetic sensors and non-volatile magnetic random access memories (MRAM) [13]. It is known for decades but the search for strong AHE materials is still an open research field [14].

For a very comprehensive review on the theory and all applications of MLWFs see also Mostofi et al. [9].

2. The WANNIER90 code

The open-source software package WANNIER90 [4,5] is available since 2007 to the material science community. It is entirely written in Fortran and released under GNU GPL.

WANNIER90 consists of two main executables: 1) a serial tool named *wannier90.x* and 2) a purely MPI-parallelized application, called *postw90.x*. Based on an initial electronic structure calculation employing density-functional theory [6] - using for example Quantum ESPRESSO [15], or other codes such as: SIESTA, Wien2k, Octopus, Fleur etc. – *wannier90.x* computes MLWFs which are needed by *postw90.x* for computing the thermoelectric (the BoltzWann code) and Berry-phase derived properties of interest.

Within this workflow, originally the calculations of *postw90.x* for Berry-phase derived properties were most problematic regarding wall-clock runtime and total resource consumption. For this reason, the main focus of this project has been the optimization of *postw90.x* with respect to performance and scalability.

The computations of *postw90.x* essentially boil down to the evaluation of certain quantities associated to the vertices of a regular grid within a 3-dimensional space, the so-called reciprocal space (or k-space) whose points are often referred to as k-points. Calculations for each k-point are in turn dominated by dense matrix operations (double precision complex), in particular by matrix-matrix multiplication and diagonalization. It is an important feature of the approach using MLWFs that calculations can be done independently for each k-point, which renders the overall computation an embarrassingly parallel problem.

From a software-architectural point of view, the Berry part of *postw90.x* is organized in three main computation modules, which accomplish slightly different tasks but share the same core functionality for evaluating k-point quantities: 1) *berry_main* samples the reciprocal space along a finite 3-dimensional regular grid and calculates some aggregated values (such as optical conductivity, anomalous Hall conductivity or orbital magnetization), 2) *kslice* produces 2-dimensional plot data with respect to a certain 2d-slice through the reciprocal space, finally, 3) *kpath* produces plot data according to a polyline (a one-dimensional path) through the reciprocal space. Depending on the use-case, each module can either be disabled or configured with an individual sampling resolution. Compared to the other modules, usually the calculations of *berry_main* involve the largest number of k-points and are therefore most computationally intensive. For this reason, originally *berry_main* was the only module which was parallelized using MPI. For *kpath* and *kslice*, only serial implementations existed at the beginning of this project.

3. Performance optimization

All optimizations to *postw90.x* implemented in the course of this project are based on performance analyses using the integrated tool suite *HPCToolkit* [16]. In this section, we list the most important and successful of these optimizations and discuss their impact on program performance. All reported runtimes and speedups in this section refer to a small test case with 32 atoms running at 64 processes.

Additional information on all changes to the code-base of WANNIER90 can be found online in the GitHub repository of this project: https://github.com/ponweist/Wannier90.

3.1. BLAS integration

The first target for performance optimization was the computation for anomalous hall conductivity, which happens in the program module *berry_main*.

Through profiling runs with *HPCToolkit* (modules *kpath* and *kslice* disabled), we identified a hotspot in the routine `utility_rotate` (utility.F90): The calculation of matrix products of the form $B = R^{H} \cdot A \cdot R$, accounted for almost half of the code's total runtime. Note that the matrices involved are dense double-precision complex matrices of size $N \times N$, where N=133 is the number of Wannier-functions (N depends essentially linearly on the number of atoms in the test case). The matrix product was formulated in one line of Fortran code, as follows:

```
B = matmul(matmul(transpose(conjg(R)), A), R)
```

Clearly, this implementation yields sub-optimal performance for two main reasons: First, several (avoidable) temporary matrices are created during result evaluation. Second and much more important, the default behavior of our compiler (Intel Fortran Compiler 14.0.3) is to translate the Fortran's built-in `matmul` function to primitive loop constructs, which in turn are not vectorized because auto-vectorization does not seem to be supported for loops involving complex data types.

As a first simple solution, we tried using the compilation flag `-opt-matmul`, which should instruct the compiler to use matrix multiplication routines from Intel's Math Kernel Library (MKL) whenever a call to `matmul` is encountered. However, neither an improvement in runtime, nor any evidence that MKL is actually used could be seen from the profiles.

For this reason, we implemented a new version of `utility_rotate`, where we explicitly used BLAS for matrix products instead of Fortran's built-in `matmul` function. Additional analysis of the code showed that at least in all performance-critical calls to `utility_rotate`, the input matrix A is actually not needed any more and its storage can be reused as target for the result matrix B. The new version of `utility_rotate` incorporates this observation and uses only one temporary matrix object. The observed speedup factor for `utility_rotate` through these changes is 5.7 – the total speedup of the entire computation (*berry_main*) is 55%.

Note that the original implementation of `utility_rotate` is still used in several non-performance critical sections of the code. We advise the code developers to entirely remove it and to use exclusively the new implementation instead.

3.2. Optimization of core calculation routines

After introducing BLAS for matrix multiplications in `utility_rotate`, the next performance critical code section was identified in the routine `get_imfgh_k_list` (berry.F90). When enabling the computation of orbital magnetization, this routine contributes to a significant amount of the total runtime of *berry_main* (>50%).

The routine `get_imfgh_k_list` computes three quantities, $F_{\alpha\beta}$, $G_{\alpha\beta}$ and $H_{\alpha\beta}$, for each of a given list of energies close to the Fermi-energy level. Evaluation of these quantities is based on equations (51), (66) and (56) of [14] and involves traces (sums of diagonal elements) of certain matrix products.

It is important to note that in the original code these matrix products were calculated exhaustively, although only their diagonal elements were actually needed for the final result. This observation led to a first straight-forward optimization, where we replaced the final matrix multiplication of each of the involved products with a "partial" multiplication, deriving only the product's diagonal elements. Additionally, we replaced all of the remaining 15 (originally 25) calls to `matmul` with explicit calls to BLAS.

The second step in optimizing `get_imfgh_k_list` involved a closer dependency analysis of all necessary matrix products. By exploiting associativity and by rearranging and re-using intermediate results, the number of matrix multiplications could be further reduced from 15 to 7. Moreover, two of these seven intermediate

products are actually not energy-dependent and could thus be pulled out of the innermost loop of `get_imfgh_k_list`.

	Original code	Trace optimization	Reusing intermediate results
$F_{\alpha\beta}$	4	0	0
$G_{\alpha\beta}$	9	6	2
$H_{\alpha\beta}$	12	9	3
Overall	25	15	5 (+2)*

* Two energy-independent matrix multiplications can be calculated outside the innermost loop.

Table 1: Number of (full) matrix multiplications for computing $F_{\alpha\beta}$, $G_{\alpha\beta}$ and $H_{\alpha\beta}$ for a given energy close to the Fermi-energy level

The above optimizations for the evaluation of $F_{\alpha\beta}$, $G_{\alpha\beta}$ and $H_{\alpha\beta}$ came at the cost of using five temporary matrix objects within `get_imfgh_k_list` (instead of originally one), but brought a significant improvement in runtime – the achieved speedup for *berry_main* (including the effect of the optimization for `utility_rotate` from the previous section) corresponds to a factor of 2.5.

	Runtime (in CPU cycles)		Speedup factor
	Original	New	
berry_main	2.6×10^{14}	1.0×10^{14}	≈ 2.5
get_imfgh_k_list	1.9×10^{14}	3.6×10^{13}	≈ 5.3
[matrix multiplications for $F_{\alpha\beta}$, $G_{\alpha\beta}$ and $H_{\alpha\beta}$]	9.8×10^{13}	8.6×10^{12}	≈ 11.5

Table 2: Perfomance improvement for computing anomalous hall conductivity and orbital magnetization (test case with 32 atoms and 64 processes)

Note that also the computation for anomalous hall conductivity (AHC) benefited from the above changes: AHC is derived from the quantity $F_{\alpha\beta}$, which is now evaluated using only 4 "partial" matrix multiplications, yielding a computational effort of the order $O(N^2)$ (N is again the number of Wannier functions), instead of originnally $O(N^3)$ when computing unneeded off-diagonal elements of the 4 involved products.

3.3. Bottleneck elimination in the initialization phase

After successfully optimizing the most performance-critical program module of *postw90.x*, *berry_main*, we put our focus on the program's initialization phase.

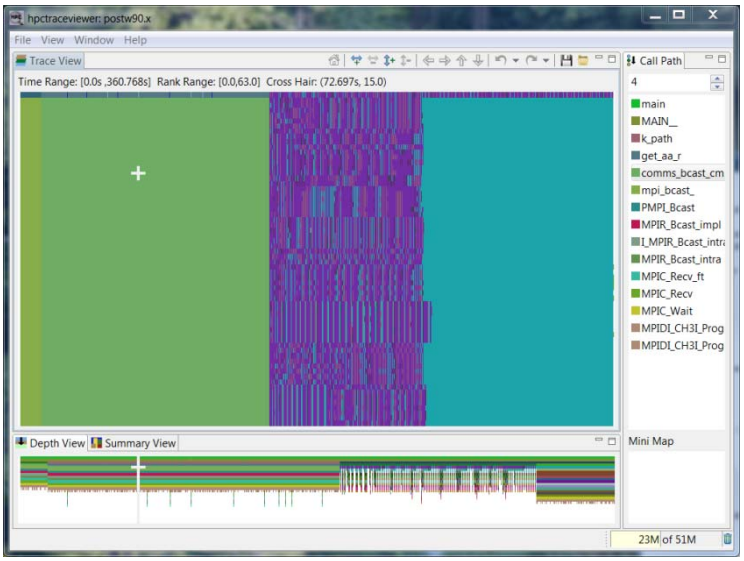

Figure 1: Performance problem in initialization phase: Non-master ranks wait for more than 2 minutes (green region with cross-hair) for receiving calculation parameters from the master rank.

Figure 1 shows a visualization of the activity of 64 processes, executing a test case with 32 atoms (*berry_main* and *kpath* enabled). Time flows from left to right; colors encode program routines which are currently being executed. As can be seen from the large green area (where the white cross-hair is positioned), MPI ranks 1 to 63 are waiting for more than two minutes (cross-hair position is at ≈ 73 seconds) in `MPI_Bcast` for the master rank to broadcast the parameters needed for the actual computations. Investigations with *HPCToolkit* showed that the master rank (whose activity can be seen in the topmost timeline Figure 1), spends most of the initialization time in a loop of the following structure:

```
! Wannier-gauge overlap matrix S in the projected subspace
!
call get_win_min(ik,winmin_q)
call get_win_min(nnlist(ik,nn),winmin_qb)
S=cmplx_0
do m=1,num_wann
   do n=1,num_wann
      do i=1,num_states(ik)
         ii=winmin_q+i-1
         do j=1,num_states(nnlist(ik,nn))
            jj=winmin_qb+j-1
            S(n,m)=S(n,m)&
                  +conjg(v_matrix(i,n,ik))*S_o(ii,jj)&
                  *v_matrix(j,m,nnlist(ik,nn))
         end do
      end do
   end do
end do
```

For the elimination of this bottleneck, it is a key step to recognize that the above loop is just an alternative formulation for a matrix product of the form $S = (V_1)^{H} \cdot S_0 \cdot V_2$. By introducing a temporary matrix object and by applying the variable substitutions `ik_a` ← `ik`; `ik_b` ← `nnlist(ik,nn)`; `ns_a` ← `num_states(ik)`; `ns_b` ← `num_states(nnlist(ik,nn))` and `wm_a` ← `winmin_q`; `wm_b` ← `winmin_qb`, this product can be formulated equivalently as:

```
allocate(tmp(ns_b,num_wann))

call gemm(S_o(wm_a:wm_a+ns_a-1, wm_b:wm_b+ns_b-1), &
          v_matrix(1:ns_a, 1:num_wann, ik_a), &
          tmp, 'C', 'N')
call gemm(tmp, &
          v_matrix(1:ns_b,1:num_wann,ik_b), &
          S, 'C', 'N')
```

Through this change, the computational complexity for the matrix product dropped from essentially $O(N^4)$ (N = `num_wan` ≈ `num_states`) to $O(N^3)$. For the concrete test case ($N = 133$) the matrix product is now calculated 610 times faster than before (additional performance comes from the usage of BLAS). As can be seen in Figure 2, the total initialization time dropped in this way from originally more than 2 minutes to less than 15 seconds (cross-hair position is at ≈ 6.6 seconds).

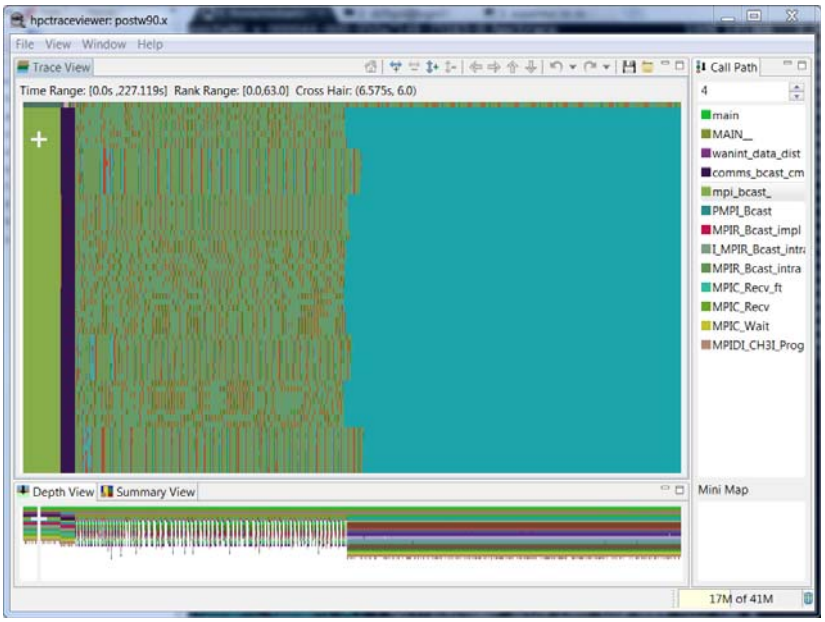

Figure 2: Improved initialization performance: Waiting time for non-master ranks (green region with cross-hair) has been reduced from originally more than 2 minutes to less than 15 seconds.

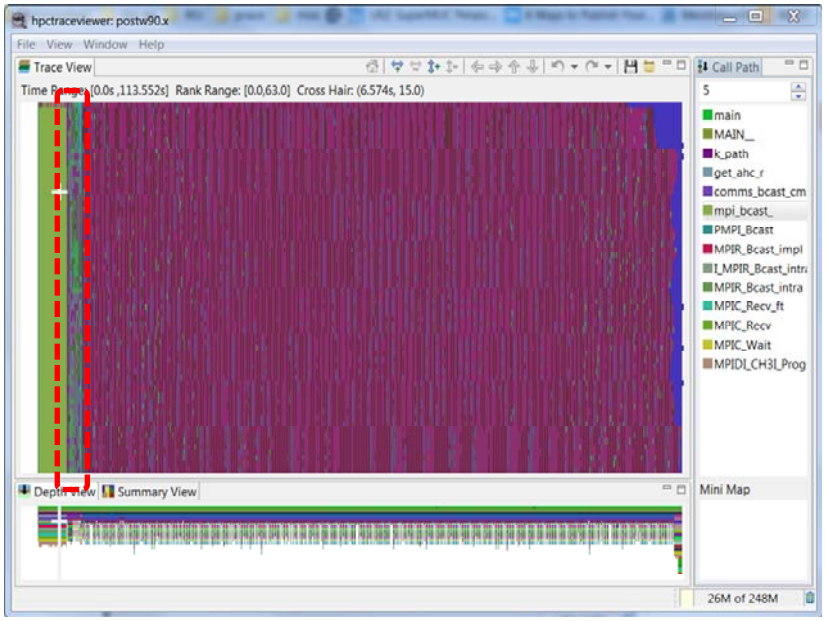

Figure 3: Parallelized *kpath* module: Workload is distributed evenly among all processes. The computation phase corresponding to *kpath* is highlighted.

3.4. Parallelization of program modules 'kpath' and 'kslice'

Looking at Figure 1 and Figure 2, another problem is quite obvious: From the large turquoise region at the end of the program execution it can be seen that MPI ranks 1 to 63 have to wait quite long in `MPI_Finalize` for the master rank to finish its computations. This load imbalance is simply explained by the fact that originally only serial implementations for the program modules *kpath* and *kslice* existed, i.e. all the work was done exclusively on the master rank (note that the master rank executes *kpath* before *berry_main*, while other ranks directly start their computations for *berry_main*).

For this reason, our last goal in the frame of this project was the parallelization of *kpath* and *kslice*. Separating the result data output from the actual computation loops and distributing the workload among all processes was quite straight-forward and gave very satisfactory results.

In Figure 3, we see the final version of the code performing the same computation as in Figure 1 and Figure 2. It can be seen clearly that computational workload is balanced pretty evenly among the processes. Again the green region with the cross-hair corresponds to waiting times in the initialization phase. It is followed by a short computation phase for *kpath* (see red dashed rectangle), and later by a significantly longer phase for *berry_main*. The runtime for the particular computation could be further reduced from more than 3.5 minutes to less than 2 minutes. For the *kslice* module, which has not been enabled for the specific test case, the results with respect to load balance look very similar.

4. Results

For discussing the finally achieved performance and scalability of *postw90.x*, we use a series of physically meaningful computations for Berry curvature and anomalous Hall conductivities of (Ga,Mn)As dilute magnetic semiconductors. The elementary cell sizes were 8, 16, 32, 64 and 128 which corresponds to the following dopings: 25%, 12.5%, 6.25%, 3.125%, and 1.5625%, respectively (we substituted one Ga atom in the cell by Mn). The symmetry of GaAs is face-centered cubic (fcc). However, after doping and building larger cells, we obtain simple cubic cells with 8 and 64 atoms, fcc cells with 16 and 128 atoms and body-centered cubic (bcc) cells with 32 and 256 atoms. All calculations were done for the bulk 3D perfect crystal, without disorder. The experimental finding that the AHC exhibits the maximum of its value at the doping of about 3% is reproduced in our approach, which was not the case when tight-binding methods were applied (compare Fig. 23 in [11]). These calculations are preliminary for more demanding project concerned with the surface doping, and necessity to explain the experimental data which show strong dependence of the AHE on the thin film thickness, temperature and doping; the AHC changes even its signum [17]. It is important, because magnetism in thin films of (Ga,Mn)As can be controlled by the electric field, and therefore be utilized in future spintronic devices [18].

4.1. Strong scalability analysis

In Figure 4, we give a detailed comparison of the scalability behaviour of the original and new version of *postw90.x* for different test cases. All program modules, *berry_main*, *kpath* and *kslice* have been enabled and representative sampling resolutions have been chosen for each of them. Improvements in performance are mainly due to the usage of BLAS for all performance-critical matrix multiplications and due to the optimizations discussed in Section 3.2. Increased scalability – which is almost optimal up to 2048 processes for large problems – has been achieved mainly through parallelization of *kpath* and *kslice* as well as optimizing the initialization phase (see Section 3.3). A different view on the achieved scalability improvements is provided by Figure 5 and Figure 6 in terms of computational cost (CPU hours) and relative scaling overhead.

At the time of writing, we do not completely understand the sudden scalability breakdown between 2048 and 4096 processes. Analysis of the generated *HPCToolkit* profiles indicated a problem related to the result data collection within *kslice* which is done using `MPI_Gatherv`. While the run-time of all other routines behave as expected (near-to-optimal scaling), the calls to `MPI_Gatherv` take approximately 70 times longer for 4096 processes than they do for 2048 processes, although the same amount of data is transferred to the master process (this phenomenon is observable for both test-cases, 64 and 128 atoms). Note that for our tests we were using Intel MPI version 4.1. In the course of further investigations the problem with `MPI_Gatherv` could be reproduced in a small test program outside the WANNIER90 code and an according issue has been reported to Intel.

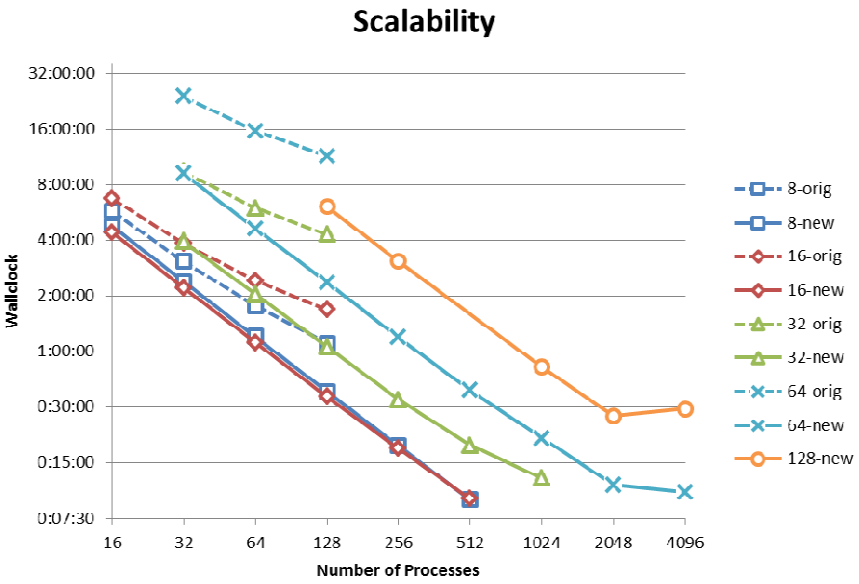

Figure 4: Strong scalability comparison for the original (dashed lines) and the new (solid lines) code version for computations with 8 up to 128 atoms. Performance has been increased significantly; scalability is now optimal up to 2048 processes for large problems.

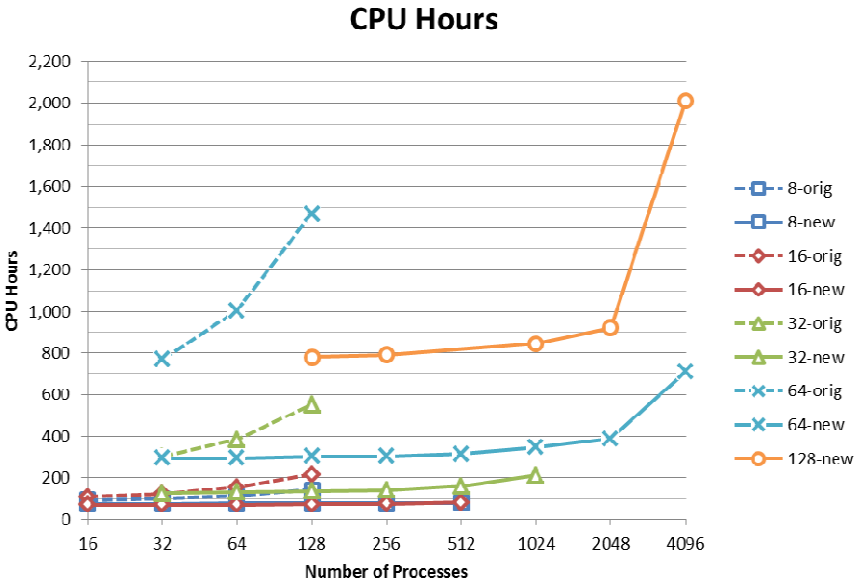

Figure 5: Computational cost for the original (dashed lines) and the new (solid lines) code version for computations with 8 up to 128 atoms. A scalability breakdown between 2048 and 4096 processes is observed for the new code version.

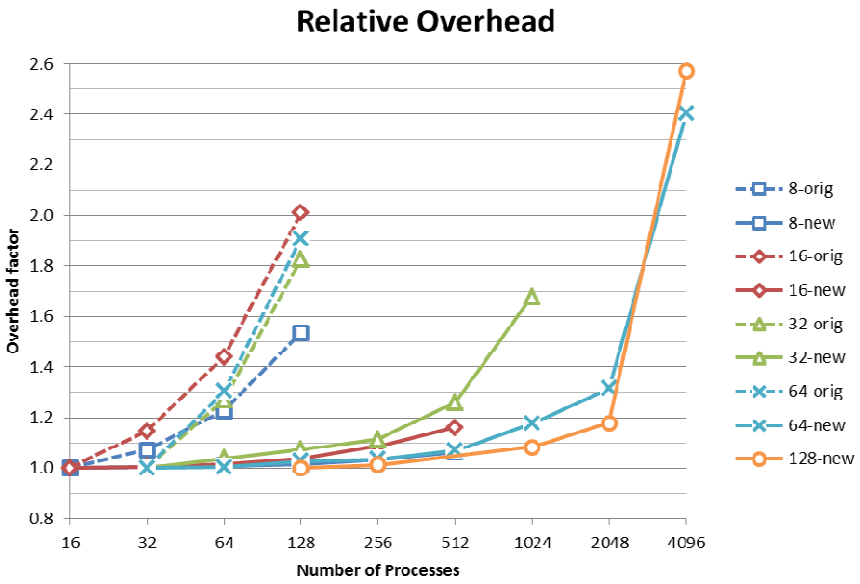

Figure 6: Increase of computational cost relative to the run with smallest number of processes in the test series. The scalability improvement from the original (dashed lines) to the new (solid lines) code version can clearly be seen.

4.2. Weak scalability analysis

In order to demonstrate the scalability of the most computationally intensive program module *berry_main* beyond 2048 processes, we carried out a weak scalability analysis for the 64 atoms test case. The program modules *kpath* and *kslice* have been disabled and the sampling resolution (i.e. grid size) has been chosen such that the number of grid vertices is approximately proportional to the number of processes. Table 1 shows that in this setup, weak scalability is nearly optimal (i.e. wall-clock runtime remains almost constant) up to 16-thousand processes.

Processes	Grid size	Runtime
512	100^3	0:36:47
1024	126^3	0:37:05
2048	159^3	0:37:03
4096	200^3	0:37:10
8192	252^3	0:37:15
16384	317^3	0:37:46

Table 3: Weak scalability analysis for the 64 atoms case. Only computations of *berry_main* are enabled. Weak scalability is almost perfect up to 16-thousand processes.

4.3. Performance analysis using threaded BLAS version

While performance optimization was the original focus of this project, also considerations regarding the memory requirements of *postw90.x* became more and more important when trying to perform computations with higher and higher number of atoms.

Due to the resource constraints of this project, we have not been able to set up a test case with 256 atoms. However, we expect such a computation to be problematic with respect to memory consumption: The originally planned test case for 256 atoms would involve N=1029 Wannier functions. In this scenario, the required memory for most of the matrix objects would already exceed 16MB (double precision complex, dimension $N \times N$). Assuming 1.5GB of available memory per core (SuperMUC thin nodes), each MPI process could hold only approximately 90 such matrices, which is most likely not enough.

A first attempt to overcome this problem could be to use a hybrid version of *postw90.x*, i.e. reducing the number of MPI processes per node and thereby increasing the available memory per MPI process. In order to evaluate how viable such an approach would be, we performed a preliminary series of test runs, where we linked

postw90.x against the threaded version of Intel's math kernel library. As can be seen in Table 4, the decrease in efficiency (due to thread idle times while non-BLAS routines are executed) may be tolerable when using up to 4 threads per MPI process. Clearly, efficiency in this respect could be further improved by additionally parallelizing performance-critical code sections of *postw90.x* with OpenMP. However, this work has been out of scope of this project.

Processes	Threads/Process for BLAS	Runtime
2048	1	0:26:56
1024	2	0:33:27
512	4	0:49:17
256	8	1:17:46

Table 4: Performance comparison for the 128 atoms case when using a fixed number of 2048 processor cores. Using a threaded BLAS version might be a viable option in order to increase the available memory per MPI process.

5. Conclusion

In this whitepaper, we reported on a series of very successful optimizations for the post-Wannier Berry-phase program *postw90.x* of the Quantum mechanics software package WANNIER90.

The performance of core computations, dominated by dense matrix operations, could be increased by a factor of 5 and higher by using BLAS instead of the Fortran built-in `matmul` function or explicit loop constructs for performing runtime-critical matrix products. A further speedup could be achieved by algebraically rearranging matrix multiplications (exploiting associativity) and reusing intermediate results.

Another important improvement was the elimination of a severe bottleneck in the initialization phase. This now makes possible previously unfeasible computations with more than 64 atoms. Besides that, also the scalability of *postw90.x* benefited significantly from this optimization. An additional improvement in scalability has been achieved through the parallelization of two previously serial program modules, *kpath* and *kslice*.

As a result of our work, we demonstrated near-to-perfect strong scalability of *postw90.x* up to 2048 processes for sufficiently large problem settings. When restricting the computations to the program module *berry_main*, we were able to demonstrate optimal weak-scalability up to 16-thousand processes.

5.1. Outlook

When it comes to computations with large number of atoms (more than 128), the memory requirements of *postw90.x* are getting problematic. A hybrid version of *postw90.x* (employing OpenMP and/or a threaded BLAS library for intra-node parallelism) may be a first solution to this problem. In the long term, also a proper distribution of large common data objects may be necessary. To give an example, in the current implementation every MPI rank holds its own copy of the matrix array `v_matrix`, which consumes several hundred megabytes for large problem settings. Clearly the distribution of such data objects will imply additional MPI communications during the computation. Although first ideas have been developed and discussed within this project, a precise data distribution concept which best maintains the current performance of *postw90.x* is subject of future investigations.

Another more straight-forward task for a future improvement of the WANNIER90 package is the parallelization of currently serial computations within *wannier90.x*. Here it is especially important to parallelize the Wannierization code, since for large systems (more than 100 atoms) the number of needed bands may excess 1000 and the run-times for metallic systems approach 5 hours and more. Also the run-time of the serial code for transport is more than 1 hour for systems with 100 atoms and 1D structure. We expect the computational effort to jump dramatically for 2D structures. For this reason, parallelization of that part is of great interest too.

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-312763. This work has been achieved using the PRACE Research Infrastructure resource *SuperMUC* at the Leibniz Supercomputing Centre (LRZ) located in Garching near Munich, Germany.

References

[1] G. H. Wannier, “The structure of electronic excitation levels in insulating crystals”, Physical Review, 52, 191 (1937).

[2] N. Marzari and D. Vanderbilt, “Maximally localized generalized Wannier functions for composite energy bands”, Physical Review B, 56, 12 847 (1997).

[3] I. Souza et al., “Maximally localized Wannier functions for entangled energy bands”, Physical Review B, 65, 035109 (2001).

[4] A. A. Mostofi, “WANNIER90: A tool for obtaining maximally-localised Wannier functions”, Computer Physics Communications, 178, 685 (2008).

[5] http://www.wannier.org

[6] W. Kohn and L. J. Sham, “Self-Consistent Equations Including Exchange and Correlation Effects“, Physical Review, 140, A1133 (1965).

[7] X. Wang et al., “Ab initio calculation of the anomalous Hall conductivity by Wannier interpolation”, Physical Review B, 74, 195118 (2006).

[8] X. Wang et al., “Fermi-surface calculation of the anomalous Hall conductivity”, Physical Review B, 76, 195109 (2007).

[9] Guang-Xin Ni et al., “Tuning Optical Conductivity of Large-Scale CVD Graphene by Strain Engineering”, Advanced Materials, 26, 1081 (2014).

[10] Junfeng Liu et al., “Orbital magnetization of graphene and graphene nanoribbons”, Journal of Applied Physics, 103, 103711 (2008).

[11] N. Nagaosa et al., “Anomalous Hall effect”, Review of Modern Physics, 82, 1539 (2010).

[12] R.J. Gambino and T.R. McGuire, IBM Tech. Disclosure Bull. 18, 4214 (1976).

[13] L.D. Alegria et al., “Large anomalous Hall effect in ferromagnetic insulator-topological insulator heterostructures”, Applied Physics Letters, 105, 053512 (2014).

[14] M.G. Lopez, D. Vanderbilt, T. Thonhauser, I. Souza “Wannier-based calculation of the orbital magnetization in crystals”, Physical Review B, 85, 014435 (2012).

[15] P. Giannozzi, S. Baroni, N. Bonini, et al, “Quantum ESPRESSO: a modular and open-source software project for quantum simulations of materials”, J.Phys.:Condens.Matter 21, 395502 (2009), http://arxiv.org/abs/0906.2569

[16] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, N. R. Tallent, “*HPCToolkit*: Tools for performance analysis of optimized parallel programs”, Concurrency and Computation: Practice and Experience 22 (2010) 685–701.

[17] D. Chiba, A. Werpachowska, M. Endo, Y. Nishitani, F. Matsukura, T. Dietl, and H. Ohno, “Anomalous Hall Effect in Field-Effect Structures of (Ga,Mn)As”, Physical Review Letters, 104, 106601 (2010).

[18] Prof. Hideo Ohno presentation in Keio Topical Workshop on Semiconductor Spintronics (2013); about 22-nd minute of the talk http://www.youtube.com/watch?v=YrXCfUUM_vc

[19] V. Weinberg, N. Anastopoulos, P. Nikunen, “Best Practice Guide SuperMUC v1.0”, PRACE Best Practice Guides, http://www.prace-ri.eu/best-practice-guides