Optimizing Parallel Electronic Structure Calculations Through Customized Software Environment

Study of GPAW on Small HPC Clusters

STEFAN GABRIEL SORIGA[1], PETRICA IANCU[1]*, ISABELA COSTINELA MAN[2,3], GEORGE MARTON[1], VALENTIN PLESU[1]

[1]University Politehnica of Bucharest, Centre for Technology Transfer in Process Industries (CTTPI), 1, Gh. Polizu Str., Bldg A, Room A056, 011061, Bucharest, Romania

[2] Romanian Academy, 'C.D. Nenitescu', Center of Organic Chemistry, 202 Splaiul Independentei 060023, Bucharest, Romania

[2] University of Bucharest, Faculty of Chemistry, 3-7 Polizu Str., 011061, Bucharest, Romania

In this work focus on optimization the density functional theory (DFT) code GPAW on two high performance computing (HPC) clusters, for office use, for large solid system, namely MgO(001) solid surface is described. In this respect, different ways of building GPAW in HPC environments for best integration with the hardware architecture, aimed to optimize the overall performance, and some practical solutions are presented. Two computational platforms with different hardware architecture are considered. For each of them, in order to obtain the best performing solution, some software environments on which to build GPAW application are investigated. Another relevant factor taken into account configurability easiness, keeping an eye on reducing the development effort while achieving the same performance. The tests with MgO(001) surface, modeled using (4X4) unit cells with four atomic layers thickness, and 36 atoms per layer, revealed the best integration of GPAW application with hardware architecture considered.

Keywords - HPC small-clusters; GPAW; molecular modeling optimization; customized environment

The recent advances in hardware and their availability as cheap and commodity components together with the availability of right software tools encourage the development of small-sized high performance computing (HPC) clusters designed for office use. Academic institutions use small clusters in search for efficient hardware resource usage and lower energy consumption. This kind of cluster is a cheap alternative to traditional, specialized supercomputing platforms, capable of supporting the needs of a small research group. Generally, it has two to four computing nodes, one of them being the head node too. Power and cooling requirements do not exceed facilities found in a typical office environment. Standard power receptacle and residential-grade air conditioning are easily sufficient for optimal operation. Custom build chassis with straight airflow design reduce the heat and noise to levels lower than many workstations [1].

On top of the hardware is the cluster system software. The cluster system software describes the collection of programs used in configuring and maintaining individual nodes, together with the software involved in submission, scheduling, monitoring, and termination of parallel jobs [2]. A second software layer is the system software. The system software includes per node software such as operating system, system libraries and compilers. Together, the cluster and system software may present very different software environments on which to build parallel numerical libraries and applications, and require a certain level of customization if they are to achieve an adequate level of performance.

Extensive studies were carried out on performance of GPAW [3] software on massively parallel supercomputers [4]. In our study the main goal of this paper is to present an exhaustive analysis of several sets of system software, compiler suites, and numerical libraries used to set up the GPAW software environment for a large solid surface system, exemplified on a large MgO (001) unit surface, for maximizing the performance on small HPC clusters.

Another goal is to provide the reader with estimation on the performance - in terms of maximum speedup, and to give an idea on the way the components of the running environment are glued together particularly for this large solid surface and which can be extended to other similar systems.

The steps taken into consideration for implementation and evaluation of several application build environments aim to improve the performance of parallel structure calculations on clusters designed for office use. Special attention is given to the computing platform description, underlying hardware and software configuration of the clusters. The investigation of performance is largely developed, revealing the experimental results obtained for each architecture set-up. The conclusions underline that the best solution for AMD cluster improves the performance with 35% compared to base case, and for Intel cluster the improvement is 15% compared to base case.

Grid-Based Projector-Augmented Wave Application

GPAW [3] is a popular, open-source program for ab-initio simulations of nanostructures based on Density-Functional and Time-Dependent Density-Functional Theory. The code is based on Projector-Augmented-Wave method and uses a real-space grid representation of the electronic wave functions. GPAW is coded in Python with extensions written in C for performance critical parts, and uses MPI programming model for parallel execution. MPI is used both within and between nodes. OpenMP is not supported, only threading. The code is build upon the Atomic Simulation Environment (ASE), a set of modules to facilitate setting up, running and analyzing atomic calculations, and NumPy, a Python library for manipulating

* *email: p_iancu@chim.upb.ro*

multidimensional arrays of numbers. For linear algebra operations it uses BLAS, LAPACK, ScaLAPACK, BLACS, FFTW (for fast plane-wave based calculations). GPAW (version 0.10. 0.11364) has the following requirements: Python 2.6 - 2.7; NumPy 1.3.0 or later (multithreading in Numpy is not supported by GPAW); Atomic Simulation Environment (ASE); C compiler; Libxc (> 2.0.1); BLAS and LAPACK/ScaLAPACK libraries (multi-threading is not supported); An MPI library (required for parallel calculations); HDF5 (> 1.8.0) library for parallel I/O and for saving files in HDF5 format; SciPy 0.7.0 or later (required for transport of response calculations); Atomic PAW-setups package.

BLAS, LAPACK, and ScaLAPACK are numerical libraries. BLAS (Basic Linear Algebra Subprograms) routines are a de facto standard API for linear algebra libraries. These libraries contain common mathematical operations such as root finding, matrix inversion, and solving systems of equations. Netlib [5] provides a reference implementation of BLAS written in Fortran 77 but without any attempt at optimizing performance. ATLAS (Automatically Tuned Linear Algebra Software) and OpenBLAS are two highly efficient machine-specific implementations of BLAS library. Netlib CBLAS is the reference C interface to the BLAS, although it is also possible to call the Fortran BLAS from C. BLACS (Basic Linear Algebra Communication Subprograms) is a set of routines that implement low-level matrix and vector operations on distributed memory platforms. It can use different communication interfaces including MPI. BLAS exploit vector and hierarchical memory architectures to the greatest extent. For this reason it is widely used by LAPACK (Linear Algebra Package), a well-known linear algebra package. LAPACK provides routines for solving systems of linear equations and linear least squares, eigenvalue problems, and singular value decomposition. In also includes routines to implement the associated matrix factorization such as LU, QR, Cholesky and Schur decomposition. LAPACK was designed to effectively exploit the caches on modern architectures, and thus can run very fast given a well-tuned BLAS implementation. LAPACK has also been extended to run on distributed-memory systems in packages such as ScaLAPACK (Scalable LAPACK). ScaLAPACK provides high performance linear algebra routines specifically designed for parallel-distributed memory platforms. It uses explicit message passing for interprocessor communication, so it is portable on any computer that supports MPI. ScaLAPACK solves dense and banded linear systems, least squares problems, eigenvalue problems, and singular value problems. ScaLAPACK depends on PBLAS operations. PBLAS (Parallel BLAS) is an implementation of BLAS intended for distributed memory architectures. As of version 2.0 the ScaLAPACK code base directly includes PBLAS and BLACS.

HDF5 is a data model, library, and file format for storing and managing data. It supports a large variety of data types, and is designed for flexible and efficient I/O and for high volume and complex data.

Libxc is a library of exchange-correlation functionals for density-functional theory written in C and has Fortran bindings, released under the LGPL license. It can be a performance bottleneck for small systems.

Compilers and libraries

When compiling programs under Linux x86_64 platforms we have the choice between several solutions:

·GNU toolchain: the standard compiler and libraries of Linux world: GCC, G++, gfortran, libc;

·Intel toolchain: the Intel compilers and libraries from Intel Parallel Studio XE non-commercial: ICC, ICPC, ifort, Math Kernel Library (MKL);

·AMD toolchain: the AMD compilers and libraries: opencc, openCC, openf95, AMD Core Math Library (ACML);

The GNU toolchain offers robust portability of code intended for compilation in Linux, while the Intel and AMD toolchains should offer a substantial performance increase over GNU when used on their corresponding vendor hardware. Depending on the code and the hardware architecture, each of these toolchains may offer better performance than the other.

MKL and ACML are numerical libraries. Processors vendors develop highly tuned implementations of numerical libraries. These libraries are design and optimized for particular processor type. Intel Math Kernel Library (MKL) for Intel processors and AMD Core Math Library (ACML) for AMD processors are examples of such implementations. The Intel MKL is a library of highly optimized, extensively threaded math routines. It contains an implementation of BLAS, BLACS, LAPACK and ScaLAPACK, Fast Fourier Transforms (FFT) complete with FFTW interfaces, Sparse Solvers, Vector Math Library and Vector Random Number Generators. The ACML incorporates BLAS (including Sparse Level 1), LAPACK, FFT, and a set of Random Number Generators (RNG) routines that are designed for performance on AMD platforms. Intel MKL has a proprietary license, but you can get one for free for non-commercial use. Although ACML has a proprietary license, the library is distributed in binary form free of charge.

Computing platforms description

The following two systems, located at the Center of Organic Chemistry C.D. Nenitescu in Bucharest, are used for the experiments presented in the following sections.

2-node Intel Nehalem cluster

The first cluster designed for office use has two dual-processor motherboards, each holding two Intel Xeon E5520 (Nehalem) Quad-core processors running at 2.26 GHz. Both motherboards are mounted in the same case, and essentially there are two machines inside. One being the master and the other the slave, connected together using a short gigabit Ethernet network cable. The master node has: 2 Intel Xeon E5520 quad-core @ 2.26 GHz processors; 16 GB DDR3 Registered ECC RAM @ 1333 MHz; 2 x 500 GB Serial-ATA2 hard drive, 7200 RPM disks connected with a RAID controller equipped with 64 MB RAM cache memory (disks configured with RAID mirroring and striping for additional security and faster I/O); 1 Gb/s gigabit Ethernet connection used as the internal connection for the machines; 1 Gb/s Ethernet connection for external communication. The slave node has the same features except that it lacks the network card for the external communication and the video card. When bought, this cluster was running a version of CentOS 5 as operating system. As a way to improve performance we replaced the original operating system and cluster system software. Now, the cluster is running CentOS 7 operating system and the job management (HPC scheduling) is performed by open-source Torque resource manager with Maui as scheduler.

4-node AMD Interlagos cluster

The second cluster designed for office use is composed of four dual-processor motherboards, each holding two AMD Opteron (Interlagos) 12-core processors running at 2.4 GHz. The motherboards are mounted in the same case, and essentially there are four machines inside. One master and three compute nodes; all connected together using an in-case gigabit Ethernet switch. The overall configuration has: 96 cores: 8 x 2.4 GHz 12-core Opteron processors; 192 GB DDR3 Registered ECC RAM @ 1600 MHz (2 GB/core); 8 TB Serial-ATA2 hard disk storage (7200 RPM disks, 64 MB

cache, RAID mirroring and striping); 1 Gb/s gigabit Ethernet for internal communication; 1 Gb/s external Ethernet connection. This cluster is using the original operating system (Centos 6.3) and cluster system software (Sun Grid Engine parallel job queue and scheduler manager).

Performance evaluation

To select the most suitable implementation of GPAW for the cluster architecture, in the following paragraphs we compare instances of GPAW build with open-source tools and various numerical libraries against versions build with tuned vendor specific ones for the large solid surface system MgO (001) shown in figure 1.

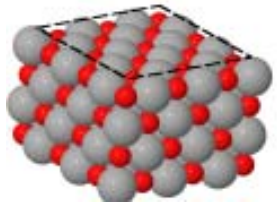


Fig. 1. Side view of MgO (001) surface cell used in calculations

The MgO (001) surface is modeled using a (4x4) surface unit cell with a thickness of four atomic layers, corresponding to a cell side length of 12.930 A° with 36 atoms per layer. A° vacuum region of 14 A° separates the periodic repeated images. In the calculation we used a grid spacing of h = 0.2 A° (64x64x102 grid points) and Γ point sampling of the Brillouin zone. A single point total energy calculation of the optimized cell is done using RPBE [7] exchange correlation functional for 44 SCF iterations using Davidson eigensolver. It was chosen the default parallelization and because we perform a gamma point calculation the processors are used only for domain decomposition (the decomposition is done with number of cores).

This application was run on both clusters. The problem size was kept constant between all runs. We selected only one application to test the different versions of GPAW. This job is illustrative enough for the applications we usually run on these clusters. The results presented in this section focused on the execution time of GPAW as reported by the "Total" field in the text report output by the code.

2-node Intel Nehalem cluster

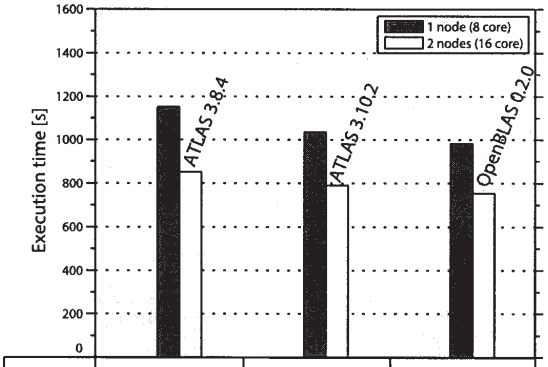
On Nehalem cluster we start from a clean install of CentOS 7. GPAW was not ported on this distribution yet and we have to build GPAW manually. Manually building GPAW requires that you first set up the build environment and install the required development tools and libraries. We first compiled GPAW using the software environment provided by the distribution. The versions of the packages available in the repositories meet GPAW requirements. Using software associated with a particular distribution is the easiest way to keep your system up-to-date with latest patches, but usually you do not get the most efficient configuration. This was our case, too. If someone just downloads the GPAW sources and wants to build them, he will have to disable ScaLAPACK and HDF5 support, because these packages are not part of the CentOS 7 distribution. In order to have a complete installation of GPAW, besides what is included in the distribution, you have to compile and build Libxc, HDF5, and to set up ASE and GPAW-Setups. Moreover, if you want to run GPAW in parallel you have to install a message passing interface implementation. In Linux community, open-source versions of MPI are well known and primarily used instead of proprietary libraries. OpenMPI has become the first free choice for MPI on Linux. After all these dependencies are met, the default configuration of GPAW will use Netlib versions of BLAS and LAPACK mathematical libraries. Although it is known that Netlib reference implementations should not be used for production because will not usually perform at maximum on many computing environments. A step toward improving performance is to build GPAW with the BLAS and LAPACK versions provided by the ATLAS (version 3.8.4) library included in the distribution. This library is not optimized for any specific platform. So we took another step and rebuilt ATLAS with the particular settings of the processors found in the system. We did this step, and in the same time we used the latest version of ATLAS (3.10.2). New versions of packages are consistently being improved. The drawback is that to use the latest version of ATLAS you need to build it from source. Once taken on "building from source" road, we compiled from source another high performance open-source library: OpenBLAS. Table 1 shows the versions of GPAW obtained using mainly packages within repositories (ATLAS 3.10.2 and OpenBLAS are the exceptions) and compiled with the default GNU tool chain. Figure 2 shows the performance comparison between the different custom versions of GPAW. The best performance is obtained when running the OpenBLAS version of the code. Compare these numbers with those obtained with Netlib reference implementation of BLAS and LAPACK: 7096.362 seconds for 1 node and 4391.082 seconds for 2 nodes. The original BLAS and LAPACK implementations give very poor performance compared to optimized implementations. We outline the fact that in these configurations the software stack - Python, NumPy, SciPy, is part of the CentOS 7 distribution.

We compiled from source using the default GNU toolchain ATLAS 2.10.2, OpenBLAS, OpenMPI, Libxc, and HDF5. ASE and Gpaw-Setups do not need compilation, only the declaration of two environment variables. These versions of GPAW are very easy to obtain and you keep maximum compatibility with the rest of the system.

Another step toward improved execution time is to rebuild the whole software stack required by GPAW or to use vendor-optimized numerical libraries. We followed both directions. We built everything starting with numerical libs, NumPy, SciPy, and ScaLAPACK using different compiler suites. There are a

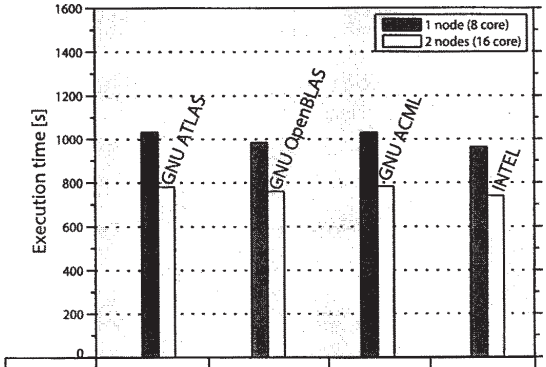
GPAW requirements	Package version			
GNU C compiler	4.8.2			
Python	2.7.5			
BLAS & LAPACK	Netlib (ref.) 3.2.1	ATLAS 3.8.4	ATLAS 3.10.2	OpenBLAS 0.2.0
Numpy	1.7.1 / 1.8.2			
Scipy	0.12.1 / 0.14.0			
OpenMPI	1.8.1			
ASE	3.8.1.3440			
Libxc	2.2.0			
HDF5	1.8.13			
GPAW-Setups	0.8.7929			

Table 1
GPAW WITH GNU TOOL CHAIN PROVIDED BY CENTOS 7 DISTRIBUTION



1 node	1151.924	1035.074	982.677
2 nodes	851.617	790.116	752.916

Fig. 2. Performance of GPAW versions build with default GNU tool chain and open-source mathematical libraries



1 node	1032.308	984.215	1028.963	962.363
2 nodes	780.636	759.647	782.994	739.606

Fig. 3. Performance of GPAW versions built from scratch with different toolchains on Intel Nehalem cluster

number of possible combinations of compilers and libraries, from which we choose the following:

·GNU ATLAS - meaning the GNU toolchain and ATLAS mathematical library. This is a complete open-source software stack.

·GNU ACML. We intended to build a version of GPAW using AMD opencc compiler and ACML library, but the opencc failed to build NumPy, so we used GCC.

·INTEL. This means we built the whole software stack with the Intel compilers and libraries. More specifically, we used the Intel Parallel Studio 2013 SP1 Update 3 Suite.

Table 2 shows the new four versions of GPAW. We compiled every package required by GPAW except Python. To be able to use the ACML with NumPy, you need to have the CBLAS module, so we build CBLAS from source. For GNU ATLAS, NumPy and SciPy were compiled with UMFPACK and FFTW (besides BLAS and LAPACK libraries provided by ATLAS). UMFPACK is a collection of C functions for solving unsymmetric sparse linear systems, and it requires AMD (Approximate Minimum Degree ordering), a set of routines for ordering a sparse matrix prior to Cholesky factorization. Each version of GPAW was tested for correctness by running the GPAW self-tests. The self-test suite comprises 239 tests, sampling most features of the code. We run the test in parallel, using four cores. Because -O3 optimization level breaks a number of GPAW self-tests, we used the less aggressive -O2 optimization level for GPAW and -O3 for every other package. However, in our tests the lower optimization level didn't have any impact on performance.

After rebuilding GPAW with the configurations shown in table 2 we re-ran our application using the different versions of GPAW. Figure 3 shows the results. There is only a small difference between performance of vendor made and open-source implementations of the numerical libraries. The GNU ATLAS execution times, although better than the initial versions are still greater than those obtained with the OpenBLAS version. The speedup of the version based on ATLAS have increased only slightly such that the GNU ATLAS version is now up to 10% faster than the GPAW build with ATLAS 3.10.2 and CentOS 7 software environment. For OpenBLAS version the execution time is almost identical to our original results (fig. 3), and it remains the best performing open-source library. In fact, looking at the results of ten iterations of the application, we observed that the version of GPAW based exclusively on OpenBLAS is a little slower than the initial version. Remember that the initial OpenBLAS version was based on the packages provided by distribution and only GPAW was built with OpenBLAS. In that initial case NumPy package was using ATLAS library for BLAS. It looks like a combination of NumPy-ATLAS and GPAW-OpenBLAS is the best performer for open-source based versions, and even overall. It is true that the best performance on the Intel Nehalem cluster was obtained with INTEL version, with an execution time of approximately 16 minutes for 1 node and 12 minutes for 2 nodes. But compare these execution times with 16.4 and 12.65 min for OpenBLAS version. We may say the performance it is identical.

To investigate how Python influences the performance of GPAW, we rebuilt Python from sources using the ICC compiler and obtained a complete version of INTEL. Although in our Python benchmarks there was a speedup of 12%, there is little to no difference in performance between version which used Python and that who not. In conclusion, the performance that can be attained for BLAS operations is the key factor that determines the global performance of applications.

The best performance on the Intel cluster was obtained with the INTEL version, but OpenBLAS can be a good candidate when you don't want to use a proprietary toolchain or need a faster and easy to build version.

OpenBLAS is the easiest way of compiling GPAW on CentOS 7. INTEL version requires manually building the whole software stack. On CentOS 7, at a cost of 3% performance loss you can build from source only OpenBLAS, ScaLAPACK, and GPAW.

GPAW versions			
GNU ATLAS	GNU OpenBLAS	GNU ACML	INTEL
GCC 4.8.2			ICC 14.0.3
Python 2.7.5			
ATLAS 3.10.2, AMD 2.4.0, UMFPACK 5.7.0, FFTW 3.3.4	OpenBLAS 0.2.0	ACML 5.3.1, CBLAS 3.5.0	MKL
NumPy 1.8.2			
SciPy 0.14.0			
ScaLAPACK 2.0.2			

Table 2
HEAVY CUSTOMIZED GPAW VERSIONS

4-node AMD Interlagos cluster

We had a more difficult task in building GPAW on the AMD Interlagos cluster. AMD platform runs CentOS 6.3 distribution and we cannot upgrade it because of other computational chemistry applications. We first installed GPAW from dtufys [8] repository using the software package manager. This is the preferred way to install GPAW on a supported Linux system. But the performance of this version is much slower than any of the versions of GPAW tested on the Intel cluster. See table 3 for a quick view of the actual timings of dtufys version on 1 node on AMD cluster and INTEL version on Intel cluster. The execution time is much longer on AMD one node than on Intel one node even if the number of cores on node is much larger than on Intel. By decreasing the number of cores to eight the execution time it will increase as it will be shown later.

Table 3

THE EXECUTION TIME (IN SECONDS) OF DTUFY VERSION OF GPAW ON INTERLAGOS AND OF THE INTEL VERSION ON NEHALEM

4-node AMD Interlagos cluster		2-node Intel Nehalem cluster	
Nr. of nodes	Exec. time	Nr. of nodes	Exec. time
1 node (24 core)	4393.292	1 node (8 core)	962.363

We decided to do a developer installation. There is no easy way of building GPAW on AMD cluster. All the packages required by GPAW were outdated, and we had to build the whole stack: Python, BLAS, LAPACK, NumPy, SciPy, and ScaLAPACK with their latest versions. We choose to implement four custom versions: GNU ATLAS; GNU ACML; INTEL; GNU MKL. Table 4 shows the software environment. We had to compile each of those packages from scratch using two different compilers, GNU and Intel. Like in Nehalem cluster case, we used -O3 optimization level for every other package except GPAW. Because ACML libraries cannot be built with the GNU toolchain found in CentOS 6.3 (C and Fortran compilers version 4.4.7), we had to use Developer Toolset release 2 (devtoolset-2) [6]. The devtoolset-2 includes new versions of C, C++ and FORTRAN compilers and associated runtimes (GCC and gfortran version 4.8.2, the same version CentOS 7 provides). Besides the new features needed by ACML libraries, newer GCC versions can include better support for the processor instruction set, which might influence the performance in a positive way. Although it is expected that this improvement is generally only marginal. But we still encountered a problem, that is, devtoolset-2 does not ship the required version of FORTRAN library.

So, in the end we had to rebuild GCC 4.8.2 from source; a long and tedious task. Figure 4 gives the corresponding performance results obtained on the AMD Interlagos cluster. It is obvious that all of these GPAW versions outperform dtufys version by a great margin and that GNU ACML and INTEL versions are the best performers. Still, the best performance on the AMD cluster was obtained with the GNU ACML version. The execution time for 1 node was 561.206 seconds. Comparing this with the best performance obtained with the dtufys version (4393.292 seconds) we find that the GNU ACML is around 8 times faster than dtufys version. Even the GNU ATLAS version is around 7 times faster than dtufys version. This made us to check the dependencies on which the dtufys version was build. We found that dtufys version, although it uses the LAPACK library from ATLAS, is built on the Netlib version of BLAS. This clearly illustrates the importance of BLAS in GPAW calculations.

AMD Interlagos cluster has four nodes, thus up to 96 cores can be involved in calculation. The data from table 5 show that the speedup increase considerably until it gets to 24 cores (1 node). The execution time is calculated relative to the longest execution time achieved on two cores. At 32 cores the speedup remains the same while for 2 nodes (48 cores) increase slightly than keeps being constant for 64 and 72 cores. On 96 cores the execution time decrease and is the same as on 16 cores. On 24 cores is obtained the best ratio between the number of cores and execution time, which is an indication that the domain decomposition is a reliable one. Definitely

GPAW versions			
GNU ATLAS	GNU ACML	GNU MKL	INTEL
devtoolset-2 and a complete rebuild of gcc 4.8.2			ICC 14.0.3
GCC 4.8.2			
Python 2.7.8			
ATLAS 3.10.2, AMD 2.4.0, UMFPACK 5.7.0, FFTW 3.3.4	ACML 5.3.1, CBLAS 3.5.0		MKL 11.1.3
NumPy 1.8.2			
SciPy 0.14.0			
ScaLAPACK 2.0.2			

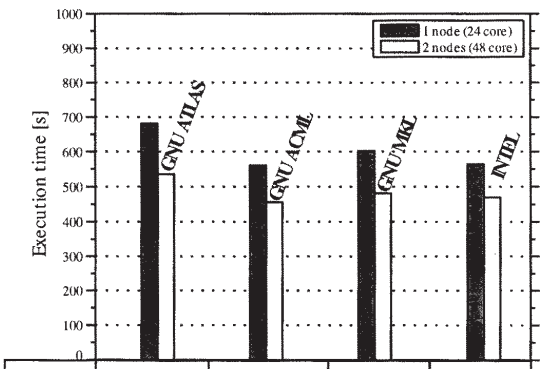
Table 4

HEAVY CUSTOMIZED GPAW VERSIONS

Table 5

EXECUTION TIME DEPENDING ON THE NUMBER OF CORES

4-node AMD, GNU ACML version	
Nr. of cores	Exec. Time/relative to the time obtained on 2 cores
2 cores	1
4 cores	1.469791
8 cores	5.908504
16 cores	9.084398
1 node (24 cores)	12.6051
32 cores	12.1944
2 nodes (48 cores)	15.50482
64 cores	15.55829
3 nodes (72 cores)	15.56866
4 nodes (96 cores)	9.541253



1 node	681.155	561.206	601.689	564.26
2 nodes	535.974	456.249	481.184	469.361

Fig. 4. Performance of GPAW versions built from scratch with different toolchains on AMD Interlagos cluster

more testing is required to identify all aspects related to different contributions and operations that can affect the performance characteristics such as the use of different eigen-solvers or how different operations are affected by libraries.

The performance of a similar HPC cluster was optimized with different software application [9], [10].

Conclusions

In summary, we discussed the influence of software building environment on the performance of a molecular modelling program. Our work in optimizing the performance of GPAW for small HPC clusters designed for office use through customized build environment has resulted in a total speedup of 15% compared with the initial version of the application for Intel Nehalem cluster (we consider ATLAS 3.8.4 as the default GPAW version for Intel cluster), and 35% for AMD Interlagos for our system used for testing, namely a large surface MgO(001). For Intel processors we have expected the optimization capabilities of GNU compilers to be much worse than these of Intel compilers. But the Intel compilers do not offer a substantial performance increase over GCC. For these versions of compilers, plus the level of optimization used and this kind of platforms, the difference is not so high. On the other side, the optimizations added to this version of OpenBLAS are obvious. At least for Intel Nehalem architecture, the performance is comparable with that of Intel MKL. For AMD platform, there is no significant difference between using GNU and Intel toolchains. Overall it is true that vendor optimized software tools have the best performance when used on their correspondent platform. In our case, this is more obvious on the AMD cluster, although we used only the vendor mathematical libraries because the AMD compiler does not build some essential programs required by GPAW. But on the Intel cluster, the OpenBLAS version of GPAW is a lot easier to build at almost the same performance with the version based exclusively on Intel suite.

Acknowledgments: Ştefan Gabriel ŞORIGA' work has been funded by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Ministry of European Funds through the Financial Agreement POSDRU/159/1.5/S/134398. Isabela Costinela MAN acknowledges for the support of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI under project number PN-II-RU-PD-2012-28/26.04.2013.

References

1. ***Turnkey Computational Chemistry, http://www.pqs-chem.com/machines.php, consulted 3.07.2015.
2. DESAI, N.L., BRADSHAW, R., LUSK, A., LUSK, E.L., MPI Cluster system Software, Recent advances in Parallel Virtual Machine and Message Passing Interface, Springer, 2004, p. 277.
3. ENKOVAARA, J., ROSTGAARD, C., MORTENSEN, J.J., CHEN, J., J. Phys.: Condens. Matter., **22,** no. 25, 2010, p. 253202.
4. ROMERO, N. A., GLINSVAD, C., LARSEN, A. H., ENKOVAARA, J., SHENDE, S., MOROZOV, V. A., MORTENSEN, J. J., Concurrency and Computation: Practice and Experience, **27**, no. 1, 2015, p.69.
5. ***Netlib Repository, http://www.netlib.org, consulted 3.07.2015.
6. ***Developer Toolset 2, http://people.centos.org/tru/devtools-2/readme, consulted 3.07.2015.
7. HAMMER, B., HANSEN, L.B., NORSKOV, J.K., Physical Review B Condensed Matter, **59**, no. 11, 1999, p. 7413.
8. ***Installation with package manager on Linux, https://wiki.fysik.dtu.dk/ase/download.html#installation –with-package-manager-on-linux, consulted 3.07.2015.
9. ŞORIGA, Ş.G., PLEŞU, V., MARTON, A., BONET-RUIZ, J., MARTON, G.I., IANCU, P., Small computer cluster for Molecular Modelling, Rev.Chim.(Bucharest), **65**, no. 8, 2014, p.960.
10. MARTON, G.I., MARTON, A.I., PLEŞU, V., IANCU, P., ŞORIGA, Ş.G., TAME – A Quantum mechanics study of the reaction mechanism for methoxylation of isoamylenes, Rev. Chim.(Bucharest), **66**, no.10, 2015, p. 1711

Manuscript received: 10.08.2015