Hands-on with OWL: the Oak–Ridge Wang–Landau Monte Carlo software suite

Ying Wai Li[1, 2], Krishna Chaitanya Pitike[3], Markus Eisenbach[1] and Valentino R. Cooper[3] ∗

[1] National Center for Computational Sciences, Oak Ridge National Laboratory, Oak Ridge, TN 37831, U.S.
[2] Computer, Computational, and Statistical Sciences Division, Los Alamos National Laboratory, Los Alamos, NM 87544, U.S.
[3] Condensed Matter Theory Group, Oak Ridge National Laboratory, Oak Ridge, TN 37831, U.S.

E-mail: `yingwaili@lanl.gov`

Abstract.
The Oak–Ridge Wang–Landau (OWL) package is an open-source scientific software specialized for large-scale, Monte Carlo simulations for the study of materials properties at finite temperature. In this paper, we discuss the main features and capabilities of OWL, followed by detailed descriptions of building and running the code. The readers will be guided through the usage and functionality of the code with a few hands-on examples. This paper is based on a tutorial on OWL given at the 32$^{\text{nd}}$ Center for Simulational Physics Workshop on Recent Developments in Computer Simulation Studies in Condensed Matter Physics.

1. Open source repository

OWL is open-sourced under the BSD 3-clause license. The latest version of the code can be found at: `https://github.com/owl-suite/OWL`. The discussion in this paper refers to Version `v0.0-alpha` of the code released on February 18, 2019.

2. What is OWL?

OWL is an acronym for "**O**ak–Ridge **W**ang–**L**andau" (or "**O**pen–source **W**ang–**L**andau"). It is a scientific software to perform large-scale, first-principles based statistical mechanics simulations for the study of finite temperature materials properties. Originally developed at the Oak Ridge National Laboratory, OWL was first intended for the studies of real-world materials and complex systems of which the physics cannot be represented by simple model Hamiltonians. It is also intended for performing simulations that are more readily and directly comparable

∗This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

with experimental observations. For example, to account for the electronic degrees of freedom and correlations in materials more accurately, first principles methods such as density functional theory (DFT) are employed for the calculation of total energies and physical properties. Monte Carlo methods, on the other hand, are used to obtain thermodynamic and finite temperature properties, which involve the sampling of a large number of configurations to obtain reasonable statistics and survey of phase space. The combination of these two components, *ab initio* and Monte Carlo methods, requires tremendous amount of computing resources due to the multiplicative effect - the computation time needed for a single *ab initio* computation multiplied by the number of Monte Carlo samples, which can easily be on the order of 10^4 and beyond. From a technical point of view, this simulation paradigm introduces multiple levels of computational parallelism that is a perfect fit for modern computer architectures with massive, heterogeneous, parallel computing units.

An important aspect of a software that will be widely applicable is its modularity. Different Monte-Carlo methods have behaviors that can be better suited for exploring different aspects of the statistical mechanics of physical systems and a well described interface will allow for the exploration of new or improved algorithms as they become available. Furthermore, our goal of facilitating the computational statistical mechanics of complex Hamiltonians, including *ab initio* calculations and recent developments in machine learning derived models, requires the flexible compatibility of the approach taken in OWL which affords the exploration of new physical systems on a wide range of computer architectures ranging from single workstations to the largest high performance computer (HPC) systems available.

As the name implies, Wang–Landau sampling [1] was the central component of OWL and was the original motivation for the development of the code. As time and research needs evolved, OWL was extended to include a collection of commonly used classical, modern and parallel Monte Carlo (MC) algorithms.

2.1. Two modes of simulations: standalone mode v.s. driver mode

OWL provides two modes of simulations: the standalone mode and the driver mode.

- The standalone mode contains the basic functionalities that encapsulate all the Monte Carlo algorithms implemented in the code, but without any interface to external codes. It is run as an independent application for simulating user-implemented model Hamiltonians.
- The driver mode interfaces with an external package. For every Monte Carlo step, OWL sends the trial configurations to the external code to calculate the total energy and other physical properties, which are then passed back to OWL to determine the acceptance of the trial move.

To date, OWL interfaces with two open-source DFT codes, Quantum Espresso (QE) [2, 3] and Locally Self-Consistent Multiple Scattering (LSMS) [4–6]. In principle, there is no restriction on the methods that the external library employs for the energy calculations. Packages other than DFT codes, such as molecular dynamics codes, can also be used. The interfaces with FERAM [7, 8] and LAMMPS [9, 10], are two such examples that are under development.

2.2. Programming model and software design

OWL is written in C++ with an object-oriented, modular software architecture. This brings several advantages in the software development. It disentangles the Monte Carlo algorithms from the physical systems to be studied. Every algorithm only needs to be implemented once-and-for-all; the same applies to the definition of the physical model Hamiltonians. This allows for flexible combinations of algorithms and physical models. As the modular design enables code reuse and avoids duplicate implementations, it also hides the implementation details from the algorithm logic, making the code cleaner and easier to extend. For example, developers do not

Figure 1. The logo for the OWL code.

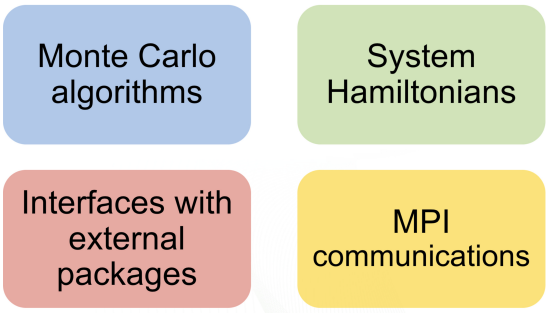

Figure 2. OWL has an object-oriented, modular design.

need to know the details of the interfaces to external codes in order to implement a new MC algorithm. Adding new features and functionality thus requires minimal effort.

OWL adopts the "MPI+X" programming model for parallelization: Message Passing Interface (MPI) [11] is a communication protocol for parallel programming. MPI provides a powerful way to easily express parallel programs through libraries. Through MPI parallelization, OWL can run efficiently on multiple computing nodes with distributed memory. We use MPI to take care of parallelization at the random walker level - each MPI rank corresponds to one random walker. By default, traditional serial code will be executed using one MPI rank assuming one random walker. The "X" component refers to additional levels of parallelism such as C++ threads or OpenMP (Open Multi-Processing) for shared memory multiprocessing [12], CUDA [13] or the like for Graphics Processing Unit (GPU) acceleration. With the MPI+X hybrid parallel programming model, MPI is used to parallelize across nodes and C++threads/OpenMP can be used to parallelize within the (multi-core) node.

In OWL, we reserve the "X" level of parallelization for physical model related computations. The external packages that OWL interfaces with usually take advantage of the "MPI+X" programming model too to perform calculations for one configuration. In this case, OWL would perform the MC simulations with one or several random walkers (depending on the algorithm), and each random walker would span several MPI ranks. OWL would serve as a computing resource manager and distribute the MPI ranks along with the "X" parallelization level to the external code for execution.

OWL uses Git for version control and code development, and the code is hosted as a GitHub repository. The "master branch" contains the most updated, development version of the code. Other branches are "feature branches" where new features and capabilities are being actively developed and are often work-in-progress. When a new implementation is completed in a feature branch, it will be tested and then merged into the master branch through a "pull request" process. This rigorous software engineering practice reduces the possibility of introducing programming bugs or incompatibilities into the master branch. Similar to other open-source code projects, we encourage users to report problems and request for features by submitting a "new issue" through GitHub.

3. Code usage

In this section, we will demonstrate the use of OWL through a few selected examples. Readers will learn about package building, how to perform serial and parallel Monte Carlo simulations for simple spin systems, as well as first-principles based Monte Carlo simulations using OWL. Updated documentation can be found on the Wiki page of the code repository: `https://github.com/owl-suite/OWL/wiki`.

3.1. Prerequisites
OWL is very light-weight and is designed with minimal dependence on external libraries and maximal portability in mind. A C++ compiler with MPI support are the only requirements to compile the standalone version of the code. OWL can be readily compiled on a variety of computing systems ranging from laptops, workstations or computer clusters, to leadership-class supercomputers.

To compile the driver mode of OWL where an external code will be used to calculate the energy and other physical quantities, the external code will need to be downloaded, compiled, and linked to OWL. We will describe the procedure in Section 3.3.2.

3.2. Downloading the code

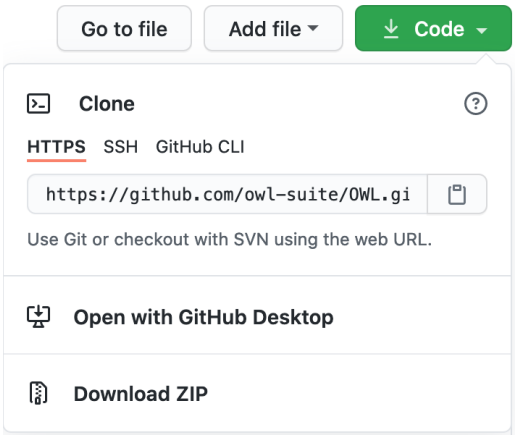

Figure 3. Downloading OWL from its GitHub site: click the green button "Code" on the OWL repository website and explore the different options. Common ways include downloading the code as a zip file (not recommended), or cloning its repository using "git" (recommended).

OWL's repository is hosted on GitHub at `https://github.com/owl-suite/OWL`. While it is possible to download the source code "as is" as a zip file (click the "Download ZIP" option under the green "Code" button on the repository site (Figure 3)), we recommend using Git to clone the repository onto your own computer by typing the following in a terminal:

```
git clone https://github.com/owl-suite/OWL.git
```

For code developers, Git provides a version control system that systematically keeps track of all the development history and versions of a software. For users, the benefit of using Git is that one may obtain the latest updates, new features, and bug fixes of the code readily using the command:

```
git pull
```

3.3. Compiling the code and linking to external codes
3.3.1. Standalone mode In the standalone mode, OWL performs the simulations using built-in simple models defined and implemented in the code. Before compiling, ensure that you have a working copy of an MPI-enabled C++ compiler (the MPI-wrapped compilers are usually named `mpicxx` or `mpic++`, with exceptions on some HPC systems).

When you first download OWL from the GitHub repository, you will see a symbolic link `architecture.inc` pointing to a string `path_to_architecture_file`. It is to remind the users to specify an appropriate architecture file for their computers. Some common computer architectures are provided in the `architecture` directory. Inside an architecture file, the MPI-enabled C++ compiler, as well as the compiler flags and link flags that would be used to compile the code, are specified. For special computer architectures or customized compiling options, the users should create their own architecture file.

Before linking to the desired architecture file, remove any previous `architecture.inc` file:

```
rm architecture.inc
```

Then recreate the symbolic link `architecture.inc` to point to the desired architecture file. For example, on a Linux machine, you may use the `generic_linux` architecture file:

```
ln -s architecture/generic_linux architecture.inc
```

For a Mac computer, the `mac_osx` architecture file is provided:

```
ln -s architecture/mac_osx architecture.inc
```

Finally, to compile the standalone version of OWL:

```
make owl
```

If the compilation finishes successfully, the executable `owl` should be created inside the `bin` directory.

3.3.2. Driver mode A powerful feature of OWL is the capability to drive an external atomistic or electronic structure code as a library for energy and physical properties calculations. This is advantageous when the system of interest cannot be defined as a simple model, especially when electronic degrees of freedom enter the Hamiltonian. In this case, OWL controls the Monte Carlo algorithms and related parameters the same way as in the standalone mode. The systems of interest can be specified using the external package's mechanism, the same way as when the calculation is carried out by that package alone.

To build OWL to drive an external code, the latter will need to be compiled first, and its source code location must be linked to OWL. In the OWL repository, we provide an example bash script, `build_QE_5.4.sh`, to illustrate how to download, compile and link the Quantum Espresso package (Version 5.4.0) to OWL as an example. This script is located in the `external_codes` directory. Assuming that working copies of the MPI-enabled C and Fortran compilers (`mpicc`, `mpif77` and `mpif90`) required to build QE are properly installed and configured, the following commands will build QE inside a directory called `q-e-qe-5.4` under `external_codes`:

```
cd external_codes
./build_QE_5.4.sh
```

It will also create a symbolic link `quantum_espresso` in OWL's top directory pointing to the top directory of QE, which is `q-e-qe-5.4` in this case.

Once QE is built, we can now build OWL in its driver mode by executing the following command from OWL's top directory:

```
make owl-qe
```

Again, if the compilation is successful, the resulting executable, `owl-qe` this time, should be created inside the `bin` directory.

3.4. Running example simulations

In the OWL repository, we provide a few example input files in the `examples` directory to illustrate the usage of different modes of the code. To run a serial Wang-Landau sampling of a 2D Ising model, we would use the standalone version of OWL using one processor (i.e., one MPI rank). From OWL's top directory, execute:

```
cd examples
cd Ising2D
mpirun -np 1 ../../bin/owl owl.input
```

More information and details about OWL's input file options are documented on the Wiki page of OWL's GitHub repository.

When running OWL in driver mode, users will have to specify additional input parameters in OWL's input file (`owl.input`) using the `PhysicalSystem` tag to indicate which external code to use. The external codes usually have their own input files to specify the physical systems, which need to be provided along with OWL's input file when OWL's driver mode is used. External packages will also have their own command line options to control a simulation/calculation; these can be specified in OWL's input file using the `PhysicalSystemCommandLine` tag.

In the `examples/CuZn` directory, we provide the input files to perform a Wang-Landau sampling for a binary alloy Copper-Zinc (CuZn), using QE to calculate the total energy using DFT. In addition to the OWL input file `owl.input`, three additional input files, `qe.input`, `Cu_pbesol_v1.2.uspp.f.upf` and `Zn_pbesol_v1.uspp.f.upf`, are required to run QE. The three QE input files should be prepared in exactly the same way as one would do for calculating this system using QE only: `qe.input` is for controlling the DFT calculations and specifying the system setup; while the `.upf` files define the pseudopotentials for the Cu and Zn atoms in the Unified Pseudopotential Format. To run the simulation, we would use the driver version of OWL, and multiple MPI ranks for QE to perform parallel DFT calculations:

```
cd examples/CuZn
mpirun -np 8 ../../bin/owl-qe owl.input
```

Interested readers are directed to consult Quantum Espresso's documentation (`https://www.quantum-espresso.org/resources/users-manual`) on the preparation of QE input files.

4. Summary and outlook

In summary, the Oak-Ridge Wang-Landau (OWL) code represents a scalable, efficient, modular approach to stochastic sampling. The goal of this project is to build a platform for a community (open source) code for exploring the properties of a material that depend on statistical averaging. The aim is that such a vehicle may not only afford more advanced materials simulations but may facilitate algorithm advancements and dissemination.

OWL contains a number of standard Monte Carlo techniques including the Wang-Landau method incorporated in a seamless fashion with the capability for easily swapping a range of potentials/Hamiltonians for the exploration of materials systems. Fundamental to the development of this code is the transferability and scalability of the code over multiple high performance computing architectures.

In essence, OWL offers a promising solution for the examination of temperature-dependent properties defined by ensemble averages using atomistic or electronic structure codes, a feat which has largely been applied to lattice models with discrete basis sets, e.g. Ising or Potts models. Initial efforts include interfaces to standard MD and electronic structure codes such as LAMMPS[9, 10], LSMS[4, 5] and Quantum Espresso[2, 3], with easily adaptable modules that could be applied to other electronic structure codes. Such developments may afford the ability to use atomistic stochastic sampling to explore temperature-dependent phase transitions as well as reactive surfaces and processes. We anticipate that further community development and application will open the door to a new era of computational research that affords direct simulation of materials properties and behavior under realistic conditions.

The features and capabilities of OWL grow steadily over time. We encourage interested users to try it out and we welcome comments, suggestions and contributions from the scientific community.

Acknowledgments

This work was sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U.S. Department of Energy, together with the Oak Ridge Leadership Computing Facility, which is supported by the Office of Science of the U.S. Department of Energy under contract No. DE-AC05-00OR22725.

References

[1] Wang F and Landau D P 2001 *Phys. Rev. Lett.* **86** 2050

[2] Giannozzi P, Baroni S, Bonini N, Calandra M, Car R, Cavazzoni C, Ceresoli D, Chiarotti G L, Cococcioni M, Dabo I, Dal Corso A, de Gironcoli S, Fabris S, Fratesi G, Gebauer R, Gerstmann U, Gougoussis C, Kokalj A, Lazzeri M, Martin-Samos L, Marzari N, Mauri F, Mazzarello R, Paolini S, Pasquarello A, Paulatto L, Sbraccia C, Scandolo S, Sclauzero G, Seitsonen A P, Smogunov A, Umari P and Wentzcovitch R M 2009 *Journal of Physics: Condensed Matter* **21** 395502 (19pp) URL `http://www.quantum-espresso.org`

[3] Giannozzi P, Andreussi O, Brumme T, Bunau O, Nardelli M B, Calandra M, Car R, Cavazzoni C, Ceresoli D, Cococcioni M, Colonna N, Carnimeo I, Corso A D, de Gironcoli S, Delugas P, Jr R A D, Ferretti A, Floris A, Fratesi G, Fugallo G, Gebauer R, Gerstmann U, Giustino F, Gorni T, Jia J, Kawamura M, Ko H Y, Kokalj A, Kkbenli E, Lazzeri M, Marsili M, Marzari N, Mauri F, Nguyen N L, Nguyen H V, de-la Roza A O, Paulatto L, Ponc S, Rocca D, Sabatini R, Santra B, Schlipf M, Seitsonen A P, Smogunov A, Timrov I, Thonhauser T, Umari P, Vast N, Wu X and Baroni S 2017 *Journal of Physics: Condensed Matter* **29** 465901 URL `http://stacks.iop.org/0953-8984/29/i=46/a=465901`

[4] Wang Y, Stocks G M, Shelton W A, Nicholson D M C, Szotek Z and Temmerman W M 1995 *Phys. Rev. Lett.* **75**(15) 2867–2870 URL `https://link.aps.org/doi/10.1103/PhysRevLett.75.2867`

[5] Eisenbach M, Li Y W, Liu X, Odbadrakh O K, Pei Z, Stocks G M and Yin J 2017 Locally self-consistent multiple scattering (LSMS) URL `https://www.osti.gov//servlets/purl/1420087;https://github.com/mstsuite/lsms`

[6] Eisenbach M, Larkin J, Lutjens J, Rennich S and Rogers J H 2017 *Computer Physics Communications* **211** 2–7

[7] Nishimatsu T, Waghmare U V, Kawazoe Y and Vanderbilt D 2008 *Phys. Rev. B* **78**(10) 104104 URL `https://link.aps.org/doi/10.1103/PhysRevB.78.104104`

[8] Nishimatsu T Feram: Md simulator for bulk and thin-film ferroelectrics URL `http://loto.sourceforge.net/feram/`

[9] Plimpton S 1995 *Journal of Computational Physics* **117** 1 – 19 ISSN 0021-9991 URL `http://www.sciencedirect.com/science/article/pii/S002199918571039X`

[10] Plimpton S LAMMPS URL `http://lammps.sandia.gov`

[11] MPI official website URL `https://www.mpi-forum.org/`

[12] OpenMP official website URL `https://www.openmp.org/`

[13] CUDA official website URL `https://developer.nvidia.com/CUDA-zone`