# Programming PHREEQC Calculations with C++ and Python
# A Comparative Study

Mike Müller[1], David L. Parkhurst[2], Scott R. Charlton[2]
[1]*hydrocomputing, mmueller@hydrocomputing.com, Leipzig, Germany*
[2]*U.S. Geological Survey, dlpark@usgs.gov, charlton@usgs.gov, Denver, CO, USA*

## ABSTRACT

The new IPhreeqc module provides an application programming interface (API) to facilitate coupling of other codes with the U.S. Geological Survey geochemical model PHREEQC. Traditionally, loose coupling of PHREEQC with other applications required methods to create PHREEQC input files, start external PHREEQC processes, and process PHREEQC output files. IPhreeqc eliminates most of this effort by providing direct access to PHREEQC capabilities through a component object model (COM), a library, or a dynamically linked library (DLL). Input and calculations can be specified through internally programmed strings, and all data exchange between an application and the module can occur in computer memory.

This study compares simulations programmed in C++ and Python that are tightly coupled with IPhreeqc modules to the traditional simulations that are loosely coupled to PHREEQC. The study compares performance, quantifies effort, and evaluates lines of code and the complexity of the design. The comparisons show that IPhreeqc offers a more powerful and simpler approach for incorporating PHREEQC calculations into transport models and other applications that need to perform PHREEQC calculations. The IPhreeqc module facilitates the design of coupled applications and significantly reduces run times. Even a moderate knowledge of one of the supported programming languages allows more efficient use of PHREEQC than the traditional loosely coupled approach.

## THE IPHREEQC MODULE

The widely used PHREEQC model (Parkhurst & Appelo, 1999) simulates a variety of geochemical processes. The new IPhreeqc module (Charlton & Parkhurst, 2011) facilitates the use of PHREEQC with multiple programming languages. The IPhreeqc module may be used to automate geochemical calculations, to couple PHREEQC with transport models, or to integrate geochemical calculations into other applications. There are several couplings of transport codes with PHREEQC, including models for the unsaturated zone (Wissmeier & Barry 2010), the saturated zone (Prommer et.al, 1999; Parkhurst et. al 2004, 2010), and pit lakes (Müller, 2011a). Each of these models had its own unique interface with PHREEQC. Instead, IPhreeqc provides a simple and consistent API for C, C++ and FORTRAN as well as a COM interface that is useable with a variety of different programming environments including Visual Basic®, the Python® programming language, and MATLAB®. While the COM module is restricted to the Windows operating system, Phreeqpy (Müller, 2011b) provides an implementation for Python based on ctypes (part of Python's standard library) and the IPhreeqc DLL or shared library. First tests of the implementations of Phreeqpy were successful on computers running Windows® and Linux operating systems (OS) and are expected to execute on Mac OS X® with only minor modifications.

In the past, incorporating PHREEQC calculations into an application has usually taken one of two approaches: (1) tight coupling, which involves custom modification of the PHREEQC's source code or (2) loose coupling, which requires writing files and starting a new PHREEQC process for each transport time step. The first approach requires a great familiarity with the PHREEQC source code and is very labor intensive, especially if the coupled code requires updating to each new PHREEQC release. Loose coupling, while much easier to implement, often results in long run times because of considerable overhead for starting a new process, reading input and output files, and repeated initialization calculations for each time step.

IPhreeqc uses a few new PHREEQC keywords, such as MODIFY, DUMP, DELETE, and COPY that allow easier manipulation of PHREEQC input and output data. The main advantage is that PHREEQC can run without reading and writing files and that data can be exchanged between IPhreeqc and an application by using the above-mentioned programming languages. All data exchange relies on input strings that use the PHREEQC input format, including the new keywords. This internal data exchange has two advantages: (1) typically execution times are faster, and (2) programming is simpler because IPhreeqc can preserve computed chemical states between time steps instead of reinitializing PHREEQC for each time step.

## EXAMPLE PROBLEM

This study uses example problem 11 from Parkhurst & Appelo (1999) featuring one-dimensional advection in a column (with no dispersion). Figure 1 shows a schematic of the model setup. A column with 40 cells is filled with a sodium-potassium-nitrate solution that equilibrates with the exchanger. Then, a calcium chloride solution flushes three pore volumes through the column, which requires 120 shifts. Figure 2 shows the concentrations at the outlet of the column. Figure 3 shows the PHREEQC input file for this problem.

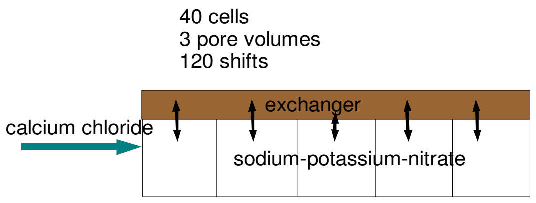

**Figure 1. Schematic setup of example problem: The sodium-potassium-nitrate solution is in equilibrium with exchanger and three pore volumes of calcium chloride flush the column.**

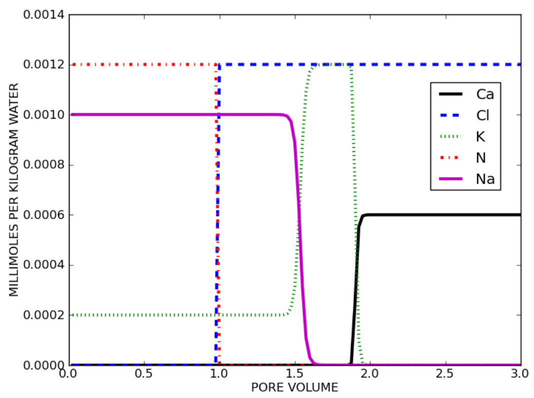

**Figure 2. Result of advection modeling with exchange. The concentrations are shown at the outlet of the column.**

```
TITLE Example 11.--Transport and ion exchange.
SOLUTION 0  CaCl2
        units            mmol/kgw
        temp             25.0
        pH               7.0      charge
        pe               12.5     O2(g)    -0.68
        Ca               0.6
        Cl               1.2
SOLUTION 1-40  Initial solution for column
        units            mmol/kgw
        temp             25.0
        pH               7.0      charge
        pe               12.5     O2(g)    -0.68
        Na               1.0
        K                0.2
        N(5)             1.2
EXCHANGE 1-40
        equilibrate 1
        X                0.0011
ADVECTION
        -cells           40
        -shifts          120
        -punch_cells     40
        -punch_frequency 1
        -print_cells     40
        -print_frequency 20
SELECTED_OUTPUT
        -file            ex11adv.sel
        -reset           false
        -step
        -totals          Na Cl K Ca N(5)
USER_PUNCH
  -heading  Pore_vol
  10 PUNCH (STEP_NO + .5) / 40.
END
```

**Figure 3. PHREEQC input file for the example problem.**

While PHREEQC can solve this simple problem in one run, we will use it as a test case for coupled applications. Figure 4 shows the conceptual approach for calculating advection in a transport model coupled with calculating chemical reactions with PHREEQC. Instead of the simple one-dimensional advection model, we could use a more sophisticated multi-dimensional advection-dispersion transport model, but the principle coupling approach would be the same.

## SELECTION OF PROGRAMMING LANGUAGES

IPhreeqc can interface with several programming languages. This study uses C++ (Stroustrup, 2000) and Python (Beazley, 2009; PSF, 2011). C++ is a powerful, statically-typed programming language. It is widely used for different tasks, including large software projects. C++ offers lots of capabilities, but requires considerable effort to learn. Python is a dynamically-typed programming language that is continually increasing in popularity. It is widely used in the scientific community because it is relatively easy to learn, yet powerful, and it provides many libraries applicable to common problems in different technical and scientific fields. Python is well suited for rapid prototyping and fast development. Therefore, the example problem was first programmed and tested with Python and later ported to C++.

## LOOSELY COUPLED MODEL USING PYTHON AND PHREEQC

Development of a loosely coupled model, which follows the simulation sequence shown in Figure 5, was the first step in the comparison of programming approaches. A Python program performed the shifting, generation of input files, and reading and interpreting of PHREEQC output files. In this approach, the PHREEQC executable was initiated 120 times. In addition, each PHREEQC run starts with a new input file and cannot access information from previous shifts. Therefore, each PHREEQC run saved all species concentrations and exchanger compositions in a selected-output file. The Python program used these values, after shifting the input, as initial conditions for the next PHREEQC run. The output after all time steps produced the same results shown in Figure 2. The Python program contains 158 lines of code, excluding comments and blank lines (Müller, 2011b).

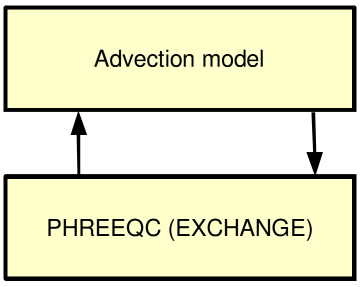

**Figure 4. Conceptual coupling of an advection model with PHREEQC.**

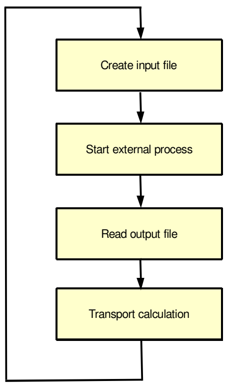

**Figure 5. Use of PHREEQC as external process.**

## TIGHTLY COUPLED MODEL USING PYTHON AND COM

In the next step, we used the IPhreeqc COM interface with another Python program. Specifying the PHREEQC keyword SOLUTION_MODIFY, we updated the species concentration for each time step after calculating the shifting. Unlike the loosely coupled approach, we did not read or write exchanger compositions because PHREEQC retained the values in memory during the course of the simulation. Figure 6 shows a sample of the Python code using the IPhreeqc API. The output after all time steps produced the same results shown in Figure 2. The program contains 152 lines of code, excluding comments and blank lines (Müller, 2011b).

## TIGHTLY COUPLED MODEL USING PYTHON AND THE DLL

Because the COM interface is only available on the Windows operating system, we wrapped the DLL/shared library containing IPhreeqc by using the Python programming language feature *ctypes* (part of the Python standard library). The resulting IPhreeqc API is the same as for the COM interface. We successfully tested this approach on a computer running a Linux OS. In principal, it will work on other non-Windows systems. The output after all time steps produced the same results shown in Figure 2. Since the wrapped-DLL/Shared-library approach uses the same API as the COM, the Python program also contains 152 lines of code, excluding comments and blank lines (Müller, 2011b).

```
...
# create IPhreeqc object
phreeqc = phreeqc_mod.IPhreeqc()
# load database
phreeqc.load_database(r"phreeqc.dat")
# create initial conditions
initial_conditions = make_initial_conditions()
phreeqc.run_string(initial_conditions)
# get components
components = phreeqc.get_component_list()
# create selected output and run it
selected_output = make_selected_output(components)
phreeqc.run_string(selected_output)
# run initial conditions
phc_string = "RUN_CELLS; -cells 0-1\n"
phreeqc.run_string(phc_string)
conc = get_selected_output(phreeqc)
...
```

**Figure 6. Initialization of IPhreeqc, loading of database and setup of initial conditions using Python. (Calls to IPhreeqc API are in bold).**

```
...
// create IPhreeqc object and load database
int id = CreateIPhreeqc();
if (LoadDatabase(id, "phreeqc.dat") != 0)
EHandler(id);
SetOutputFileOn(id, 1);
// run initial conditions, copy to column
initial_conditions(id, ncells);
// Define SELECTED_OUTPUT
std::vector<std::string> components, headings;
make_selected_output(id, components, headings);
// Run initial conditions
std::ostringstream run_stream;
run_stream << "RUN_CELLS" << std::endl << "-cells
1-" << ncells << std::endl;
if (RunString(id, run_stream.str().c_str()) != 0)
EHandler(id);
// conc has all selected output values
std::vector<std::vector<double>> conc;
extract_selected_output(id, conc);
...
```

**Figure 7. Initialization of IPhreeqc, loading of database and setup of initial conditions using C++. (Calls to IPhreeqc API are in bold).**

## TIGHTLY COUPLED MODEL USING C++ AND THE DLL

In the next step of our study, we used the IPhreeqc API with a C++ program. We implemented the same functions used in the Python programs. Figure 7 shows a sample of the code. The output after all time steps produced the same results shown in Figure 2. The program contains 197 statement lines, without comments or blank lines (Müller, 2011b).

## COMPARISION OF PROGRAMMING APPROACHES

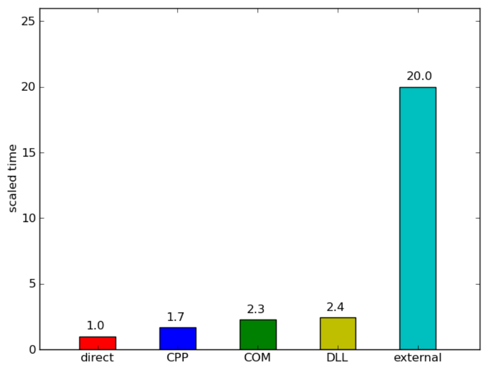

**Figure 8. Comparison of scaled run times for the direct solution and the different coupling methods.**

Figure 8 shows the scaled run times of all approaches. The run time for the direct solution is about 1.2 seconds. It serves as reference and all other values are multiples of it. The direct, non-coupled solution uses the ADVECTION keyword in PHREEQC. The model using C++ and the DLL (CPP, in Figure 8) takes about 70% longer. The Python models, COM and DLL, take 35% and 45% longer than the C++ model. The loosely coupled model (external, in Figure 8) takes about 20 times longer than the direct PHREEQC model, and more than 8 times longer than the Python-DLL model. In general, the run times for the IPhreeqc models are about an order of magnitude faster than the loosely-coupled model.

The effort to program the approaches varies considerably among programming languages and depends on programming skill. Programming in Python is typically easier and faster than in C++, in part because

Python is an interpreted language that does not require a separate compile step. Programming in C++ generally requires a more skilled programmer. In general, Python is well suited to fast prototyping of different approaches.

## CONCLUSIONS

The new IPhreeqc API allows for efficient development of coupled reactive transport models. Tests using IPhreeqc in a simple coupled reactive advection model showed an order of magnitude decrease in run times compared to a loosely-coupled model, which required starting PHREEQC as an external process for each transport time step. Furthermore, programming of a tightly-coupled model is facilitated because chemical states are retained in the IPhreeqc module between model time steps. While the run times of the Python models are somewhat longer than that of the C++ model, programming using Python can be considerably simpler, and Python code can be deployed without modification on multiple computer platforms.

## REFERENCES

Beazley, D.M., 2009 Python Essential Reference, 4th edition, Addison-Wesley Professional; ISBN-13: 978-0672329784.

Charlton, S.R., and Parkhurst, D.L., 2011. Modules based on the geochemical model PHREEQC for use in scripting and programming languages. Computers and Geosciences, doi:10.1016/j.cageo.2011.02.005

Müller, M., 2011a. PITLAKQ - The Pit Lake Hydrodynamic and Water Quality Model. http://www.pitlakq.com

Müller, M., 2011b. Phreeqpy - Python Tools for Working with PHREEQC. http://www.phreeqpy.com

Parkhurst, D.L., and Appelo, C.A.J., 1999. User's guide to PHREEQC (Version 2)—A computer program for speciation, batch-reaction, one-dimensional transport, and inverse geochemical calculations: U.S. Geological Survey Water-Resources Investigations Report 99–4259, 312 pp.

Parkhurst, D.L., Kipp, K.L., and Engesgaard, P., and Charlton, S.R., 2004. PHAST—A program for simulating ground-water flow, solute transport, and multicomponent geochemical reactions. U. S. Geological Survey Techniques and Methods 6—A8, 154 pp.

Parkhurst, D.L., Kipp, K.L., and Charlton, S.R., 2010. PHAST version 2 —A program for simulating groundwater flow, solute transport, and multicomponent geochemical reactions. U. S. Geological Survey Techniques and Methods 6—A35, 235 pp.

PSF (Python Software Foundation), 2011. The Python Website, http://www.python.org.

Prommer, H., Davis, G.B., Barry, D.A., 1999. PHT3D—A three-dimensional biogeochemical transport model for modelling natural and enhanced remediation, in: Johnston, C.D. (Ed.), Contaminated Site Remediation: Challenges Posed by Urban and Industrial Contaminants. Centre for Groundwater Studies, Fremantle, Western Australia, pp. 351-358.

Stroustrup, B., 2000. The C++ Programming Language. Addison-Wesley Professional; third edition, ISBN-13: 978-0201700732.

Wissmeier, L., Barry, D.A., 2010. Implementation of variably saturated flow into PHREEQC for the simulation of biogeochemical reactions in the vadose zone. Environmental Modelling & Software, 25(4), 526-538.