

Quantum Chemistry on Clusters: A Guide to Installing and Using MPQC, the Massively Parallel Quantum Chemistry Program*

Curtis L. Janssen

Contents

1	Background	2
2	Quantum Chemistry	2
3	The Massively Parallel Quantum Chemistry Program	3
4	Parallelism Models	4
5	Installing MPQC	5
6	MPQC Input Format	7
7	Running MPQC	8
8	Conclusion	10
9	Biography	11
10	Acknowledgments	11
A	Methods of Quantum Chemistry	11
B	What is a basis set?	12
C	Resources	14
D	Further Reading	14

*Preprint of C. Janssen, "Quantum Chemistry on Clusters", *ClusterWorld*, **2**(8), p. 18, August 2004.

1 Background

We often hear about major problems facing our society that have a technological solution. Energy provides a good example: while our civilization needs ever more energy to provide improvements in our standard of living, the production of that energy gives rise to problems such as pollution, oil dependence, global warming, etc. Hydrogen has been advocated as a part of the solution to these problems, and the US government is funding research to provide the technology needed for using hydrogen as fuel. The technological barriers are broad, and scientists from many disciplines will be required to bootstrap a viable hydrogen economy. Let us examine just one component of the solution. At a very basic level hydrogen is a chemical, and chemistry will be used to generate, store, and consume hydrogen. The chemicals involved may be relatively simple synthetic compounds, or they may be complex enzymes that use elaborate mechanisms to manipulate hydrogen. For all of the chemical systems involved, learning the properties of molecules and exactly how various molecules interact is a key part of understanding, refining, and improving the technologies. Two of the most important properties of the interaction between molecules are the energy released when joining them into one molecule and the energy required to overcome the initial repulsion that sometimes occurs when two molecules are joined together. For some molecular systems, a direct measurement of these quantities and other important properties is either too costly or too difficult, and computational modeling is therefore used instead. This requires modeling of the electrons and nuclei that comprise a molecule, and for this purpose quantum mechanics must be used.

2 Quantum Chemistry

Quantum chemistry is the discipline of computing the properties of atoms and molecules using quantum mechanics. Quantum chemists today are involved in both developing and applying these methods. Solving the equations governing the properties of molecules is extremely complex, and it cannot be done exactly. A hierarchy of approximate methods has therefore been developed that allow quantum chemists to select the method most appropriate for a given problem. Which methods are practicable depends on the computational expense of applying each method to a particular problem. The least computationally expensive methods also tend to be the least accurate methods. Such methods require an amount of computer time that formally grows like the fourth power of the size of the molecule being studied. That is, doubling the size of the molecule will require sixteen times as much computing power. The most expensive methods require processing time that grows exponentially in the size of the molecule. Between these extremes are a variety of methods whose computational expense grows like the fifth, sixth, or higher power of the size of the molecule.

Clearly, the expense of quantum chemistry calculations is significant. Two approaches are being used to deal with this cost. The first is the ongoing effort

to reformulate existing methods and develop new methods that greatly reduce the scaling of the computational cost with respect to the size of the molecule. The second approach is to apply more computing power to the problem, ideally in addition to using reformulated methods. This is the motivation for developing quantum chemistry programs that use clusters and massively parallel machines.

3 The Massively Parallel Quantum Chemistry Program

The Massively Parallel Quantum Chemistry program (MPQC) provides implementations of several of the most important methods of quantum chemistry. These include Hartree-Fock theory (HF), second-order Møller-Plesset perturbation theory (MP2), and Density Functional Theory (DFT). Also implemented is a new and very promising method: Auxiliary Basis Set MP2 with explicit inter-electron distances, (ABS) MP2-R12. MPQC has the only parallel implementation of (ABS) MP2-R12.

MPQC is unique among quantum chemistry programs in possessing all of the following properties: it is open-source, its design is object-oriented with an implementation in an object-oriented language, and it was designed from the beginning to run in parallel. Each of these properties provides important benefits.

The benefits of open-source software have been widely discussed, and these advantages are amplified in a scientific computing context. Scientific software is exceptionally complex and difficult to get right. While, in principle, journal publications contain all of the information necessary to reproduce results, inadvertent errors, omissions, and subtle numerical details make it difficult and time consuming to reproduce results from scratch. Open-source licensing arrangements encourage researchers to reuse existing code because they are granted the right to redistribute their works based on that code. This permits greater return on public investments in scientific research. And it allows researchers to examine in detail what makes a code work (or not work) well.

The object-oriented programming approach employed by MPQC has proven to be an extremely useful technique for managing the complexity of the program and enabling its extensibility. The C++ language is used to implement most of MPQC which, in the hands of experienced programmers, can be used to implement, in less time, programs with an efficiency equaling that of more traditional scientific programming languages. The downside to C++ is the greater amount of training required and the lack of good introductions to C++ in college curricula, requiring in-house training of programmers. A small portion of the diagram showing the inheritance and usage relationships between several of MPQC's C++ classes is given in Figure 1.

As discussed above, parallelism is important to reducing the turnaround time for jobs, but large scale machines are important for other reasons as well. They provide the aggregate memory of all the nodes to a single executing job,

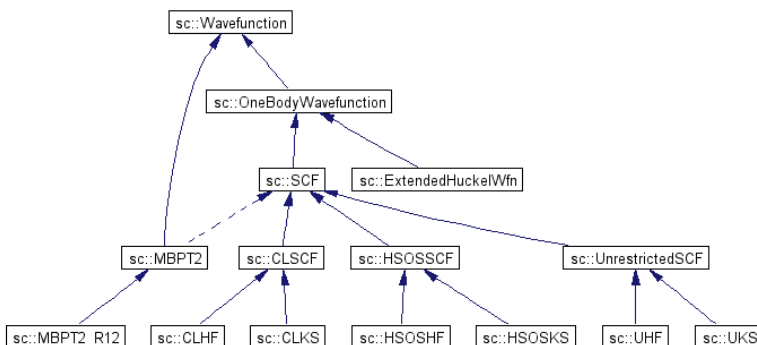


Figure 1: A portion of the MPQC class collaboration diagram.

and this large pool of available memory considerably extends the size of the calculations that can be run. For example, a large MP2-R12 calculation on a benzene dimer with 2004 basis functions is capable of utilizing as much as 106 GBytes of memory. This much memory would typically be available on a 128 node cluster permitting the calculation to be done without disk space and with fewer floating point operations than if less memory were available, dramatically reducing the time to solution.

However, to obtain the full benefit from parallel machines it is important to parallelize a code from its conception because parallelization is such an invasive process. Doing a piecemeal parallelization of a serial program often results in a code that has significant computations running in a serial mode, and these code sections will severely limit the performance of the code as the number of processors increases.

4 Parallelism Models

Before you begin to install and run MPQC, it helps to understand a little about the way MPQC does parallel processing. MPQC uses three abstractions to provide parallelism: Message Groups, Thread Groups, and Memory Groups. The Message Group provides a message passing interface, and the only implementation that is currently of interest is MPIMessageGrp, which uses the Message Passing Interface (MPI). MPIMessageGrp normally provides parallelism between different nodes in a cluster, but it can provide parallelism across multiple processors in a symmetric multi-processor SMP node as well. This SMP parallelism can also be accomplished by using a Thread Group. The POSIX Threads implementation of Thread Group, PthreadThreadGrp, provides this functionality. The MPIMessageGrp and PthreadThreadGrp parallelization models can be used together in a hybrid parallelization scheme. In this case, MPI must support coexistence with threads in the funneled mode of operation (that is,

message passing is done through the main thread). Finally, MP2 and MP2-R12 calculations require the Memory Group communication abstraction. This permits a task running on one node to read, write, and sum data into memory that is resident on another node. The one-sided communication feature that is a part of version 2 of the MPI standard is inadequate for these operations due to the lack of completion guarantees until a synchronization is done. Two specializations of Memory Group are provided that use other means to obtain this functionality. The first is `MTMPIMemoryGrp`. This requires an MPI implementation that supports fully multi-threaded operation. Furthermore, if the MPI implementation uses polling to check for incoming messages, then it will be necessary to dedicate one processor on each SMP node to poll for memory requests from other nodes. Otherwise, the performance of `MTMPIMemoryGrp` may be severely degraded. Future implementations of MPI will likely resolve these issues. Another Memory Group implementation, `ARMCIMemoryGrp`, is based on the Aggregate Remote Memory Copy Interface (ARMCI). This package specifically focuses on getting good performance for the needed memory operations and requires neither a thread-safe MPI implementation nor a processor dedicated to handle memory requests.

The cluster that we currently utilize the most is a 128 node Linux cluster with dual 3.06GHz Pentium 4 processor nodes. The interconnect is a 4X InfiniBand network with two switches: a 96 port switch and a 32 port switch. Thus, the largest job we can run on this cluster is limited to 96 nodes. We recently compared the speed-ups of an MP2-R12 calculation run on this machine to those obtained on an IBM SP with a Colony switch, and a Linux cluster using Gigabit Ethernet. Both Dynamic Load Balancing (DLB) and Static Load Balancing (SLB) were employed. The InfiniBand Linux cluster gives very good speed-ups compared to the other machines as shown in Figure 2.

5 Installing MPQC

We here outline the basic requirements and give tips for configuring MPQC for parallel use. The complete compilation instructions for MPQC are available at <http://www.mpqc.org>.

First C++, C, and FORTRAN 77 compilers are needed. Recent versions of the GNU compiler suite will work fine. If you wish to use MP2-R12 methods, then Edward Valeev's `libint` package is required. Other prerequisite software packages include the Basic Linear Algebra Subprograms (BLAS), the Linear Algebra Package (LAPACK), a recent version of `flex` (the version in most Linux/GNU distributions does not generate valid C++ code), GNU `bison`, GNU `make`, `perl`, and basic Unix or GNU file utilities. For parallel runs you will need MPI and possibly ARMCI as described above. Potential MPI implementations include Argonne's `MPICH` and Los Alamos' `LA-MPI`.

After the necessary software has been installed, MPQC can be configured for your machine by running the `configure` script. There are several configure options that you should be aware of:

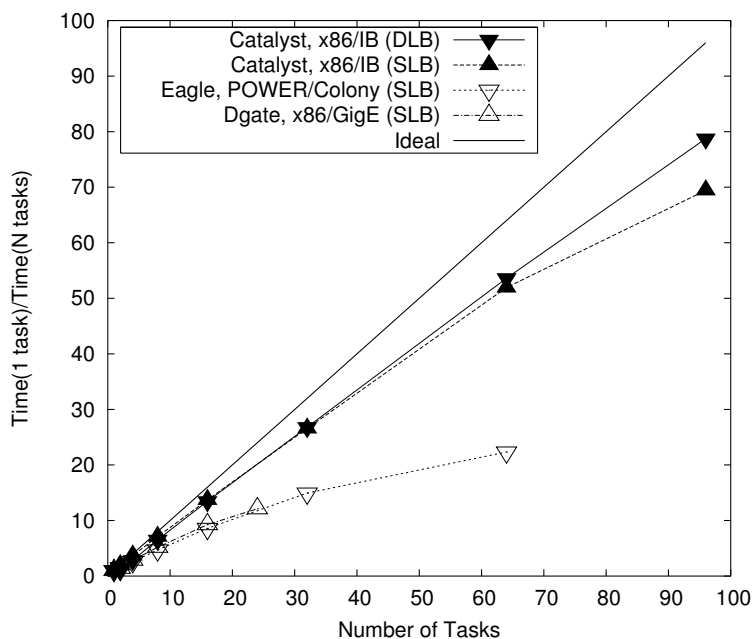


Figure 2: Scaling of the MP2-R12 method on: "Catalyst", a Linux cluster with a 4X InfiniBand network; "Eagle", an IBM SP with a Colony switch; and "Dgate" a Linux cluster using Gigabit Ethernet.

```
--with-cxx=C++ compiler
--with-f77=FORTRAN 77 compiler
--with-cc=C compiler
```

These three options specify the compilers to use. MPI implementations frequently provide scripts that run compilers with the correct options for finding MPI include files and linking in the MPI libraries. These can be specified here. Note, however, that it is very important to use the same FORTRAN compiler for BLAS, LAPACK, and MPQC. Also, it is not recommended to utilize multiple C++ compilers, which is an issue if a C++-based MPI implementation, such as LA-MPI, is used.

```
--enable-always-use-mpi
```

This ensures that MPI.Init is called early in MPQC's initialization. Some MPI implementations require this, and this option is highly recommended.

`--with-default-parallel=choice of parallel model`

This option gives the default Group specializations if more than one possibility is available. These models can be overridden at runtime, although it is more convenient to specify the model at compile time. Specifying `mtmpi` here will cause `MPIMessageGrp`, `PthreadThreadGrp`, and `MTMPIMemoryGrp` to be used for parallelization. An argument of `armcimpi` will cause the `ARMCIMemoryGrp` to be used instead of `MTMPIMemoryGrp`.

`--with-include=-I/inc_dir1 -I/inc_dir2 ...` Specifies the include flags given to the compilers. For example, if flex was installed in `/usr/local/flex`, then give `-I/usr/local/flex/include`.

`--with-libdirs=-L/lib_dir1 -L/lib_dir2 ...` Specifies the directories which will be searched for libraries. This might be needed if `libint`, `BLAS`, or `LAPACK` were not installed in a standard system library directory.

`--with-libs=-llib1 -llib2 ...` Specifies any extra libraries to be linked in. This would be needed if libraries such as `BLAS` and `LAPACK` were called something other than `libblas.a` and `liblapack.a`.

`--prefix=/mpqc_installation_directory` Gives the directory where MPQC will be installed.

MPQC is configured, compiled, and installed by executing something like the following in the top level MPQC directory:

```
% ./configure --enable-always-use-mpi --prefix=/usr/local/mpqc \
               --with-cxx=mpiCC --with-cc=mpicc --with-f77=mpif77
% make
% su
# make install
```

There should now be an `mpqc` executable in `/usr/local/mpqc/bin`. If you encountered difficulties in these steps, carefully examining the output to the screen as well as the contents of the `config.log` file may help in locating the problem.

6 MPQC Input Format

Before we can run an MPQC job, we have to know a little about writing an MPQC input. The original MPQC input format was carefully designed to be user friendly, while still allowing a high degree of flexibility. However, this format requires some experience, and a more basic input format has since been added. These two input formats are referred to as "object-oriented" and "simple". MPQC will auto-detect which input format to use at runtime. If you plan to do anything serious with MPQC you will need to learn the object-oriented

input. This input format is a direct reflection of MPQC's internal design. It allows the user to specify exactly what objects are constructed when MPQC starts and also has a few additional keywords that tells MPQC what to do with these objects. The simple input format, on the other hand, is just that: simple. It does not give access to all of MPQC's features, but it does let a user get a quick start. We will use the simple input format here. Full details on both formats can be found in the MPQC manual, which is available on the MPQC web-site.

For our simple example, we will do an optimization of the geometry of the water molecule using the B3LYP variant of DFT. Below is a line-by-line discussion of the input, `water.in`:

```
1 % B3LYP optimization of water
2 optimize: yes
3 method: KS (xc = B3LYP)
4 basis: 3-21G*
5 molecule:
6 O    0.172  0.000  0.000
7 H    0.745  0.000  0.754
8 H    0.745  0.000 -0.754
```

The first line is a comment; the % and everything following it are ignored by MPQC. Line 2 specifies that the energy of the water molecule should be minimized with respect to the nuclear coordinates. Line 3 tells MPQC that the molecular energy is to be computed using the B3LYP variation of DFT. Line 4 gives the name of the basis set to use. The remainder of the file, lines 5-8, specify the molecule in terms of the atom types and the atomic coordinates given in Angstroms.

This is all you need for a working input file. Many options have been omitted. More information can be found in the online documentation.

7 Running MPQC

As an example of running MPQC, let us run our `water.in` input file on a cluster. Note, that this calculation is so small that there will not be a speedup relative to running on one processor. The precise method for starting the parallel calculation can vary between different versions of MPI. Here is how to start the job using LA-MPI running on a machine with the `bproc` cluster extensions:

```
% mpirun -n 2 -H 0-1 /usr/local/mpqc/bin/mpqc -o water.out \
water.in
```

The `-n` argument specifies the number of MPI tasks, and the `-H` argument gives the node numbers to use. Also, if you have more than one version of MPI installed on your system, make sure that you are using the `mpirun` that corresponds to the version of MPI used to compile MPQC.

Your output file, `water.out`, should now contain several iterations of a geometry optimization. First, let us make sure the calculation was really running the way we wanted. Near the top of the output file there are three lines that give which Group implementations are used to provide parallelism:

```
Using MPIMessageGrp for message passing (number of nodes = 2).
Using PthreadThreadGrp for threading (number of threads = 1).
Using ARMCIMemoryGrp for distributed shared memory.
```

The first line gives the Message Group used, here an `MPIMessageGrp` that detected 2 nodes. If you see `ProcMessageGrp` here, MPQC was not configured correctly, and, instead of using MPI, two serial calculations were run, one on each node. The second line gives the Thread Group used, here a `PthreadThreadGrp` that used one thread per node. The third line is a Memory Group, in this case `ARMCIMemoryGrp`. If you didn't compile and install the ARMCI library, as well as configure MPQC to find the ARMCI library, then an `MTMPIMemoryGrp` would be used instead. In any case, as discussed above, the Memory Group is not used for this type of calculation.

If we have two processors per node, then one of those processors would not have been used in the above computation. However, there are two ways to utilize this extra processor: two MPI tasks can be run on each node, or each MPI task can run two threads. The first method can be accomplished by instructing `mpirun` to use four tasks while specifying only two nodes:

```
% /usr/local/lampi/bin/mpirun -n 4 -H 0-1 \
  /usr/local/mpqc/bin/mpqc -o water.out water.in
```

To use the second approach, MPQC must be given an extra argument that specifies exactly how the Thread Group object is to be constructed:

```
% /usr/local/lampi/bin/mpirun -n 2 -H 0-1 \
  /usr/local/mpqc/bin/mpqc \
  -threadgrp '<PthreadThreadGrp>:(num_threads=2)' \
  -o water.out water.in
```

Here, `num_threads=2` gives the number of threads to be used in each MPI task.

If everything worked, then the following line should appear in the output file:

```
The optimization has converged.
```

A few lines above this, you will find the energy:

```
total scf energy = -75.9739631648
```

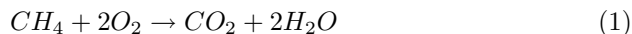
This gives the B3LYP energy of the water molecule in Hartrees at the optimized geometry. Some distance below this line is the optimized molecular geometry:

```

molecule<Molecule>: (
  symmetry = c2v
  unit = "angstrom"
  { n atoms          geometry          }={
    1   O [  -0.0914980466   0.0000000000   0.0000000000]
    2   H [   0.5225581894  -0.0000000000   0.7849930856]
    3   H [   0.5225581894  -0.0000000000  -0.7849930856]
  }
)

```

Much of the application of quantum chemistry is a variation on this theme. However, the energy of a single molecule is usually not interesting in its own right, and reactions involving several molecules are frequently the target of a study. For example, consider the combustion of methane:



Computing the energy released by this reaction involves selecting an appropriate level of theory, optimizing the geometry of each molecule, and obtaining the energy of each component. Also, it is usually necessary to do a vibrational frequency calculation to estimate the contribution to the reaction energy from nuclear motion. This procedure will provide enough information to compute the energy of the reaction. The results can be checked by varying the method and basis set to check for convergence of the energies and geometries.

8 Conclusion

We have given a brief introduction to the MPQC program including how to install and run it. Application of quantum chemistry programs for investigation of chemical systems, however, requires experience, and care must be taken to ensure that appropriate methods are applied for the problem at hand. Fortunately, there are many good books in this area that provide valuable guidance.

Quantum chemistry is still a dynamic field; new methods are being developed, and efforts are underway to reduce the computational cost for existing methods. Recently developed “local” methods eliminate distant interactions that contribute little to the energy, thereby making it possible to study much larger systems. The maximum system size is still bounded, however, and eventually it will be necessary to switch to other models for the system, perhaps a mechanical “ball and spring” model, and perhaps a continuum model where individual atoms are not even considered. Coupling all of these length scales together into a “multi-scale” application is a promising area of research. Such multi-scale software will be much more complex and even better ways of handling the complexity of scientific software will be needed. The Common Component Architecture (CCA) is an attempt to address the issues of complexity and code reuse. The CCA provides a standard mechanism for constructing a scientific

application out of existing code, each constituent code possibly developed by different researchers.

Projects are currently underway to push MPQC into these areas. More accurate (ABS) MP2-R12 methods are being developed, local coupled cluster methods are in the works, and CCA adaptors to MPQC objects already exist. And, importantly, MPQC is designed to be a community effort. SourceForge is now used for MPQC project management, and we encourage others to become involved to make MPQC meet even wider needs and solve even more problems.

9 Biography

Curtis Janssen is the lead developer of MPQC and a Distinguished Member of Technical Staff at Sandia National Laboratories, Livermore, CA. He can be reached at cljanss@sandia.gov.

10 Acknowledgments

The author thanks Ida Nielsen for proofreading this manuscript and providing helpful suggestions.

Most of the work on MPQC has been done at Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the U. S. Department of Energy under contract DE-AC04-94AL85000.

A Methods of Quantum Chemistry

One of the most basic quantum chemistry methods is Hartree-Fock theory (HF). In this method each electron sees the average effect of all the other electrons. The resulting equations are solved iteratively until the entire solution is self-consistent. HF is not very accurate, but it serves as the foundation for better approximations, known as correlated methods. In correlated methods, electrons do not feel merely the average influence of other electrons, they instead respond to the dynamic motion of other electrons, and, hence, the electrons' motions are correlated. Møller-Plesset perturbation theory, coupled cluster theory, and configuration interaction theory are all examples of correlated methods.

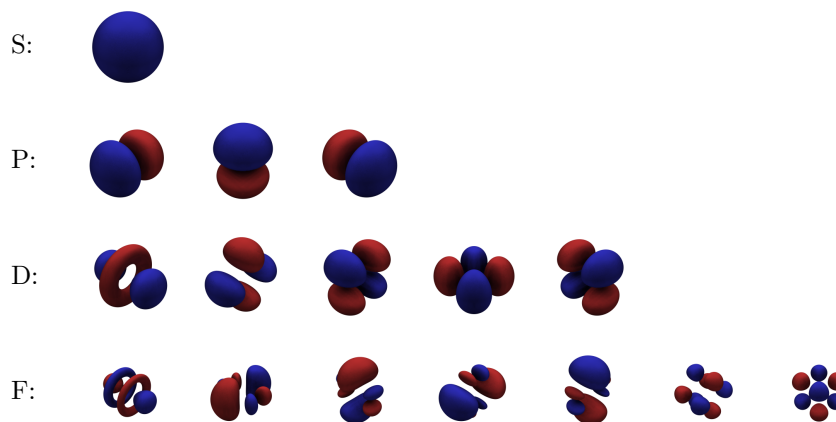
For certain molecules the HF approximation is so poor that it cannot serve as an acceptable starting point for correlated methods. For these molecules it is not possible to separate the self-consistent HF solution from the inclusion of electron correlation, and more expensive and difficult to use multi-reference methods must be used.

An entirely different quantum chemistry method is Density Functional Theory (DFT). This method is based on the Hohenberg-Kohn theorem, which, in effect, states that the exact electron density can be determined by finding the density that minimizes a density functional. The exact form of the functional is

unknown, but there are many functionals available that approximate it. In the Kohn-Sham approach to DFT, equations similar to the Hartree-Fock equations, but with an extra term, are solved. DFT generally gives good answers with less processing time than other methods, but it does not offer extremely high accuracy, and it is not possible to vary the DFT method in a systematic way to check for convergence of the solution, as can be done with the correlated methods discussed above.

B What is a basis set?

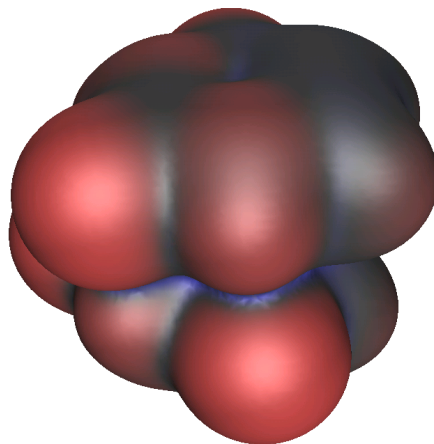
When employing classical mechanics for modeling, the location of all the particles is known. This is not the case, however, when using quantum mechanics. Because of the large mass of the nuclei it is usually a good approximation to pinpoint the nuclear locations, but the electrons are smeared out over space and must be described by a wave function. This wave function is typically constructed of orbitals, each of which describes a single electron. Each orbital is described by a linear combination of basis functions. The form of the basis functions are motivated by the exact quantum mechanical descriptions of simple atomic systems. These basis functions naturally fall into shells labeled S, P, D, F, and so on. In one real-valued representation, they look like the following:



Below are listed some well-known basis sets to illustrate the available range. In all, there are hundreds of basis sets. Also given below are the number of basis functions on the first, second, and third row elements of the periodic table. The more basis functions there are, the better the description of the orbitals, and the more costly the calculation.

Basis Name	n_1	n_2	n_3	Description
STO-3G	1	5	9	Minimum sized basis. Used as initial guess for better calculations.
3-21G*	2	9	19	Better than STO-3G, but still small.
6-311G**	6	18	26	Usually acceptable for HF and DFT.
cc-pVTZ	14	30	34	Designed for correlated methods such as MP2 and CCSD(T).
aug-cc-pVTZ	23	46	50	Supplements cc-pVTZ for molecules with loosely bound electrons.
aug-cc-pCV5Z		181		Very high quality basis including functions for describing effects involving low energy electrons.

After we finally combine all of the basis functions together into orbitals using a procedure such as Hartree-Fock theory, we can form the wave function and the electron density. The resulting electron density iso-surface for the uracil dimer is shown below.



C Resources

MPQC	http://www.mpqc.org
GCC	http://gcc.gnu.org
libint	http://www.ccmst.gatech.edu/evaleev/libint
BLAS	http://www.netlib.org/blas
LAPACK	http://www.netlib.org/lapack
flex	http://lex.sourceforge.net
GNU Bison	http://www.gnu.org/software/bison/bison.html
GNU Make	http://www.gnu.org/software/make
perl	http://www.cpan.org
MPICH	http://www-unix.mcs.anl.gov/mpi/mpich
LA-MPI	http://public.lanl.gov/lampi
ARMCi	http://www.emsl.pnl.gov/docs/parsoft/armci
bproc	http://bproc.sourceforge.net
CCA	http://www.cca-forum.org
SourceForge	http://www.sourceforge.net

D Further Reading

The number of good books that describe quantum chemistry methods and how to use them has grown rapidly in recent years. Here is a sampling of what is available:

- “Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory” by Attila Szabo and Neil S. Ostlund has been a classic reference for graduate students of quantum chemistry for years.
- “Introduction to Computational Chemistry” by Frank Jensen provides an introductory overview of quantum chemistry.
- “Molecular Electronic-Structure Theory” by Trygve Helgaker, Poul Jørgensen, and Jeppe Olsen provides an advanced and thorough discussion of the methods of quantum chemistry.