CRYSTAL09 - Parallel implementation

Overview

The parallel version of CRYSTAL uses a replicated data algorithm. Each host runs the same code and performs a number of independent *tasks* which are distributed at run time. One host is chosen as the master. The master host spawns the program onto other hosts (slaves) and operates dynamical load balancing of the task execution via a shared atomic counter.
During integral generation a task is defined as the calculation of a block of integrals. Thus each node computes a number of integrals which are stored to its local disk.
During a SCF cycle, a partial Hamiltonian matrix (**F**) is built on each node from those integrals which have been stored locally. The matrices are then passed between nodes so that each has a complete copy. The diagonalization of **F** at each k-point is treated as an independent task which is distributed. After diagonalization the eigenvalues are communicated to all nodes.
This strategy is comparatively easy to implement and successful on architectures where each node has access to fast disk storage and sufficient memory to run a complete copy of CRYSTAL. Low speed communication hardware (such as Ethernet) is usually sufficient. Performance depends critically on the system considered.
The integral generation step is performed efficiently when the number of integrals to be generated is much larger than the number of nodes. This condition is satisfied in most applications. Machines with up to 64 nodes have been used effectively on large cases. In the SCF process the construction of **F** is also efficient. Diagonalization of **F** is performed efficiently if the number of k-points is much larger than the number of nodes. This condition is usually not satisfied for large systems and thus diagonalization may be the most costly phase.
The parallel version of CRYSTAL requires a mechanism for initiating processes on remote machines and a library of routines to provide inter-process communication. There are many implementations of this functionality available and CRYSTAL has been modified to take advantage of the MPI message-passing library.

MPI Parallel version of CRYSTAL09

Running the MPI parallel version of CRYSTAL under Linux

The CRYSTAL09 parallel executables for Linux are based on either **MPICH2** or **OpenMPI** implementations of the MPI message-passing library and have been generated with the following features:

1. Fortran compiler: Intel Fortran Compiler 11.1
2. MPI libraries:
   - **MPICH2-1.2.1** (see http://www.mcs.anl.gov/research/projects/mpich2/index.php)
   - **Openmpi-1.4.1** (see http://www.open-mpi.org/)
3. Processor communication: TCP/IP (ch_p4)
4. Processor connection: ssh

The CRYSTAL09 parallel version is supposed to run on homogeneous workstation networks, Beowulf cluster and individual workstations.

To run the MPI parallel version of CRYSTAL09 under Linux special attention must be paid to set the proper environment:

1. MPICH2 or OpenMPI must be installed according to the adopted processor connection remote shell (either rsh or ssh) The `mpirun` load module may then be used to initiate parallel execution of CRYSTAL09 from the master host.
   For MPICH2 the `mpiexec` load module should be used.

2. each node must allow access via a remote shell, either rsh or ssh, to the master host. Note that the CRYSTAL09 parallel executable to be used will depend on the adopted remote shell.

Before a parallel job can be submitted:

1. there must be a consistent set of CRYSTAL09 parallel modules (**Pcrystal**) available on each node (e.g. through a NFS filesystem)

2. the CRYSTAL input deck must be provided on each node of the cluster in a file named INPUT.

Workstation clusters require each process in a parallel job be started individually.
The procedure to run CRYSTAL09 can then be summarized as:

1) create a temporary directory on each node (workstation)

2) either copy **Pcrystal** in the temporary directory of each node (workstation) or make **Pcrystal** available to each node through a NFS filesystem

3) copy the CRYSTAL09 input deck as INPUT in the temporary directory of each node (workstation)

4) MPICH2
   - Note that MPICH2 provides a separation of process management and communication. The default runtime environment consists of a set of daemons, called mpd's, that establish communication among the machines.
     Then, prepare a file with the list of nodes (workstations) to be used in the parallel run. The file is usually indicated as `mpd.hosts` and is located in the working directory or in the "/home/user" directory. The format is one hostname per line, with either `hostname` or `hostname:n`, where `n` is the number of processors in a cluster of symmetric multiprocessors. The `hostname` should be the same as the result from the command `"hostname"`.
     A sample file for a cluster of 6 nodes with a processor each will look like:

     ```
     #
     node9
     node10
     node11
     node12
     node13
     node14
     ```

     You can change this file to contain the machines that you want to use to run MPI jobs on.
   - before application process start up, deamons must be running on each node (workstation). So, start the set of mpd's on the machines in the list with the command `mpdboot`. For instance:

```
mpdboot -f mpd.hosts
```

- according to the `mpd.hosts` file, prepare a file with the list of nodes (workstations) to be used in the parallel run.
  The file is usually indicated as `machines.arch`, where `arch` is the architecture of the system (e.g. LINUX) and it can be located in the working directory. The format is one hostname per line, with either `hostname` or `hostname:n`, where `n` is the number of processors in a cluster of symmetric multiprocessors. The `hostname` should be the same as the result from the command `"hostname"`.
  According to the example above for `mpd.hosts` file, a sample for `machines.arch` will look like:

  ```
  #
  node9
  node10
  node11
  node12
  node13
  node14
  ```

  You can change this file to contain the machines that you want to use to run MPI jobs on.
- connect to the node which will be the master host (e.g. `node9`).
- move to the temporary directory of the master host and run `mpirun` as:

  ```
  mpirun -np nprocs -machinefile machines.arch Pcrystal
  ```

  This will run **Pcrystal** on the first `nprocs` processors in the `machines.arch`, located in the working directory. According to the list of nodes above, if `nprocs=4`, the program will run on: `node9`, `node10`, `node11` and `node12`.
- after execution of the run, deamons should be killed from each node (workstation) using the command `mpdexitall`.
- For more details please refer to the *MPICH2 User's Guide*

4) OpenMPI

- prepare a file with the list of nodes (workstations) to be used in the parallel run. The file is usually indicated as `machines.arch`, where `arch` is the architecture of the system (e.g. LINUX) and it can be located in the working directory or in the directory. The format is one hostname per line, with either `hostname` or `hostname:n`, where `n` is the number of processors in a cluster of symmetric multiprocessors. The `hostname` should be the same as the result from the command `"hostname"`.
  A sample file for a cluster of 6 nodes with a processor each will look like:

  ```
  #
  node9
  node10
  node11
  node12
  node13
  node14
  ```

  You can change this file to contain the machines that you want to use to run MPI jobs on.

- connect to the node which will be the master host (e.g. `node9`).
- move to the temporary directory of the master host and run `mpirun` as:

  ```
  mpirun -np nprocs -machinefile machines.arch Pcrystal
  ```

  This will run **Pcrystal** on the first `nprocs` processors in the `machines.arch`, located in the working directory. According to the list of nodes above, if `nprocs=4`, the program will run on: `node9`, `node10`, `node11` and `node12`.
- For more details please refer to the *Open MPI Software Documentation*

The output file will be displayed on the standard error. Use common Unix commands for redirecting `stderr` to a file.