MPQUEUE LIBRARY INTERFACE

JOHN M. NEUBERGER, NÁNDOR SIEBEN, AND JAMES W. SWIFT

MPQueue is a high level, STL friendly C++ library that implements easy to use job queues on an MPI cluster. It uses serialization to transfer data between nodes. Workers have the ability to submit new jobs into the current job queue.

- **Job data type.** Every job has one of several possible job types stored in the *type* field. The *data* field contains the serialized input for unfinished jobs and the serialized output for finished jobs.

```cpp
struct Tjob {
    int type;
    string data;
};
```

There is a template constructor that takes care of the automatic serialization of the data.

```cpp
template <class T>
Tjob(int type, T data);
```

- **Jobqueue data type.** This type is used to declare the input queue of unassigned jobs, as well as the output queue. The output queue might remain empty if the workers produce empty outputs.

```cpp
typedef queue<Tjob> Tjobqueue;
```

- **Initialize MPI.** This is usually the first function called by the main function. The global variables MPIrank and MPIsize are set. Note that MPIrank ranges from 0 to MPIsize -1; the boss node always has MPIrank set to 0.

```cpp
void MPQinit (int argc, char *argv[]);
```

- **Split the workers from the boss process.** Only the boss process returns from a call to MPQstart; the workers start waiting for jobs.

```cpp
void MPQstart ();
```

- **Submit a job into the currently running job queue.** Only a worker can call this function.

```cpp
void inline MPQsubmit (const Tjob &job);
```

- **Ask the boss to run a task.** A task is a short job that is immediately run by the boss and not placed on the job queue. Only a worker can call this function. The *job.data* variable contains the serialized input at invocation and the serialized output upon completion. The function MPQtask can be used, for example, to ask for the value of a counter to create a unique file name. It can also be used to return the result of a job.

```cpp
template <class T>
void inline MPQtask (Tjob & job);
```
• Get the number of jobs in the running job queue and the number of workers currently working from the boss. The workers execute this function, for example, when they need to decide if a new job should be submitted to the currently running job queue.

```c
void MPQinfo ( int &queue size, int &sent );
```

• Assign the jobs in inqueue to the workers and collect the results in outqueue. Only the nonempty results are collected in the output queue. During the execution of this function, the boss acts as a supervisor. It sends out jobs, collects the results, accepts new jobs submitted by workers via MPQsubmit, and executes tasks requested by the workers via MPQtask.

```c
void MPQrunjobs ( Tjobqueue & inqueue, Tjobqueue & outqueue );
```

• Send data to all the workers. The implementation uses a broadcast mechanism as an efficient way to share a large amount of data, enlisting workers to help in the distribution after they receive the data.

```c
template < class T >
void inline MPQsharedata ( const Tjob & job );
```

• Execute a job. This function must be implemented by every program. The workers execute this function when they receive a job to run. The boss executes this function when it receives a job request via MPQtask. The job.data variable contains the serialized input at invocation and the serialized output upon completion.

```c
void MPQswitch ( Tjob & job );
```

• Release the workers and stop MPI gracefully. This is usually the last line of the program.

```c
void MPQstop ();
```

• Serialize any variable. That is, turn it into a string. The above listed functions which take as input data structures use this function behind the scene. Standard STL variables are automatically serialized. Serializing a class requires a bit more effort; the creation of a simple template function is required. Consult the BSL for details.

```c
template < class T >
string to_string ( const T & in );
```

• Deserialize a variable. That is, decode the serialized variable out stored in the string str.

```c
template < class T >
void from_string ( T & out, const string & str );
```

• Generate load data. Turning on the compiler directive results in the creation of a data file used to produce a load diagram.

```c
#define LOAD_DIAGRAM
```