

Distributed Parallel Methods in Psi4

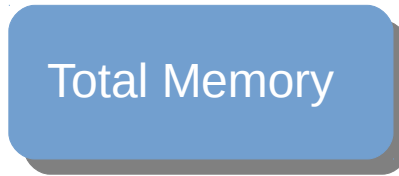
Ryan M. Richard
Sherrill Group
Georgia Institute of Technology

Psi 4 Developer's Meeting
November 14, 2014

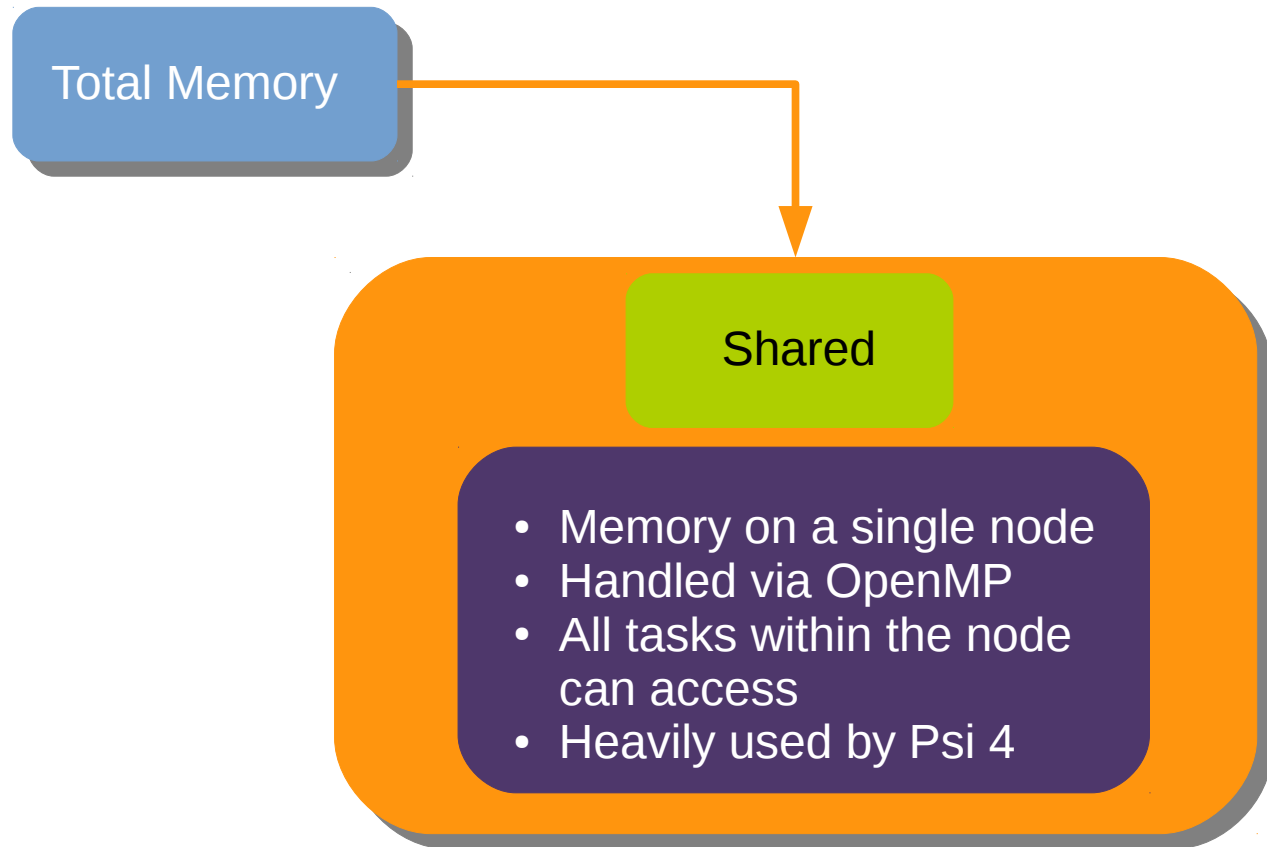


Distributed vs. Shared Memory

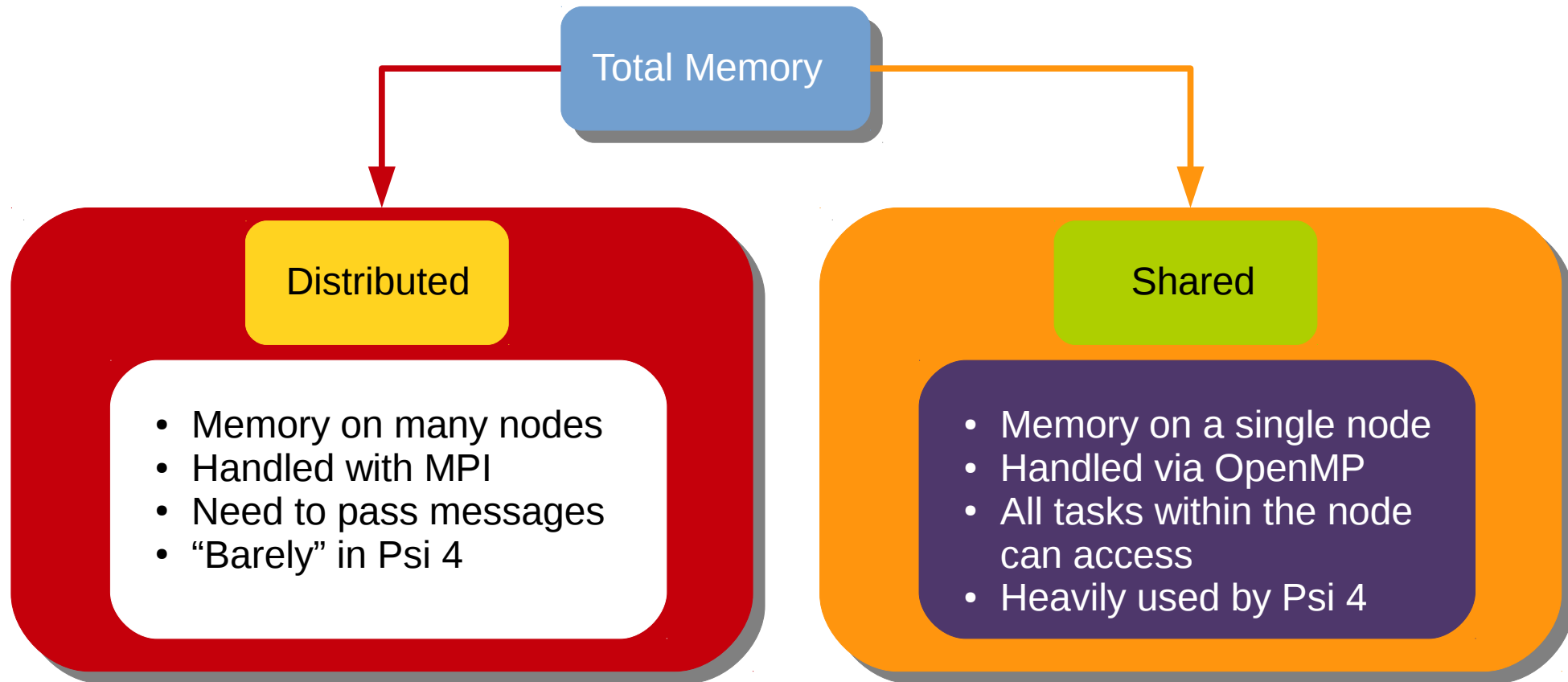
Total Memory



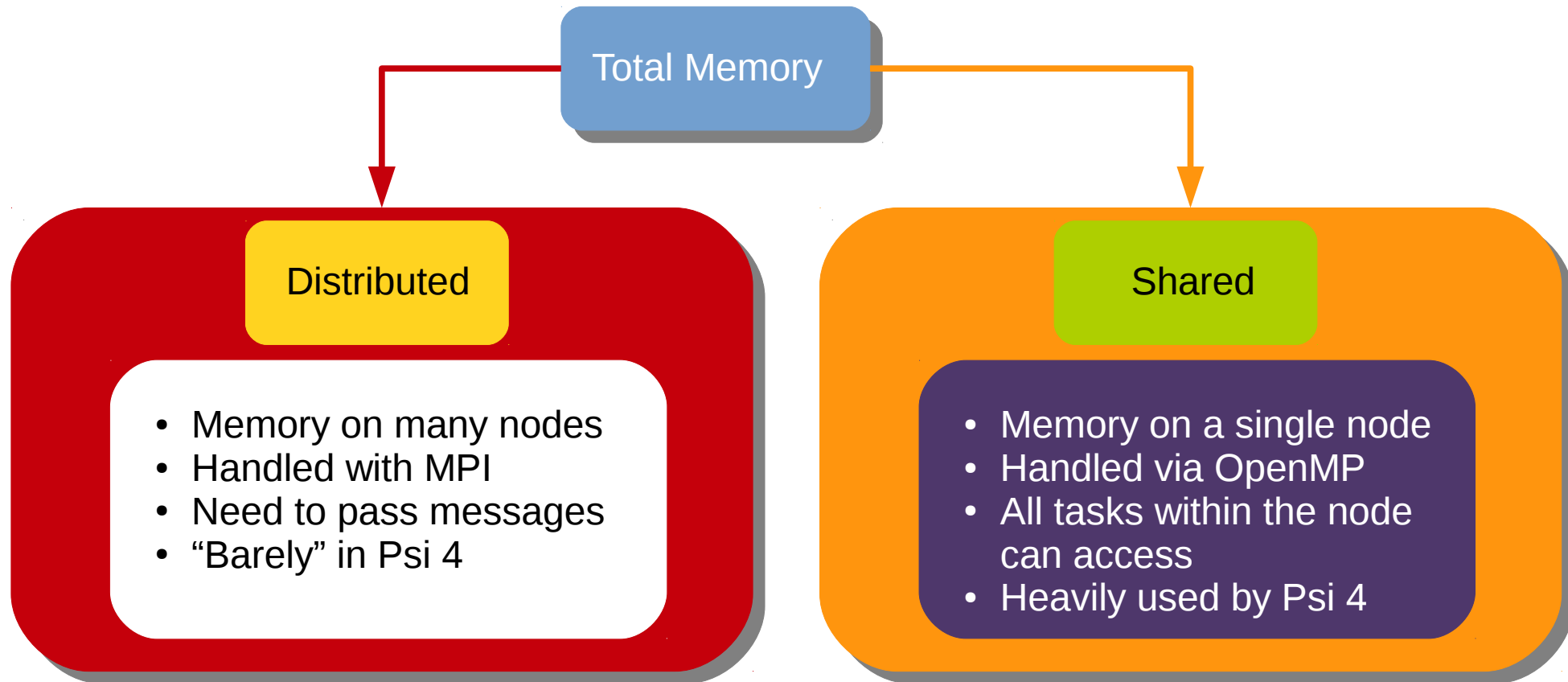
Distributed vs. Shared Memory



Distributed vs. Shared Memory



Distributed vs. Shared Memory



All my projects relate to introducing MPI
Into Psi 4

GT Fock Interface: LibJKFactory

- Designed to compute J and K matrices
 - Uses:
 - MPI
 - MKL
 - Global Arrays
 - Modified ERD
 - Demonstrated scaling to 156,000 cores
 - Tianhe-2
 - Intended to be used throughout:
 - HF
 - DFT
 - SAPT

Collaboration with:

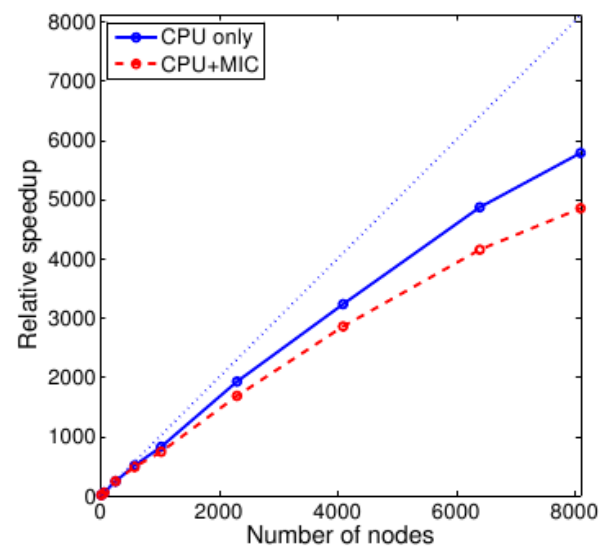
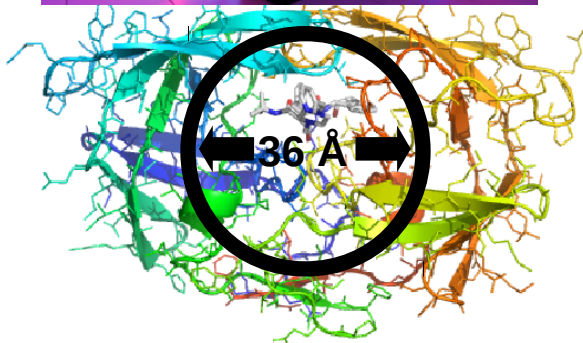


Edmond
Chow



Xing Liu

School of Computational
Science and Engineering
at GT



GTFOck Interface: LibJKFactory

- Status:
 - Working with spherical basis sets
 - Normalization issue with Cartesian
 - Timings underway
 - OptERD vs. LibInt
 - Cost of scatter/gather

LibFrag

Generalized Many Body Expansion (MBE)

$$E \approx \mathcal{E}^{(1)} + \Delta\mathcal{E}^{(2)} + \dots + \Delta\mathcal{E}^{(n)}$$

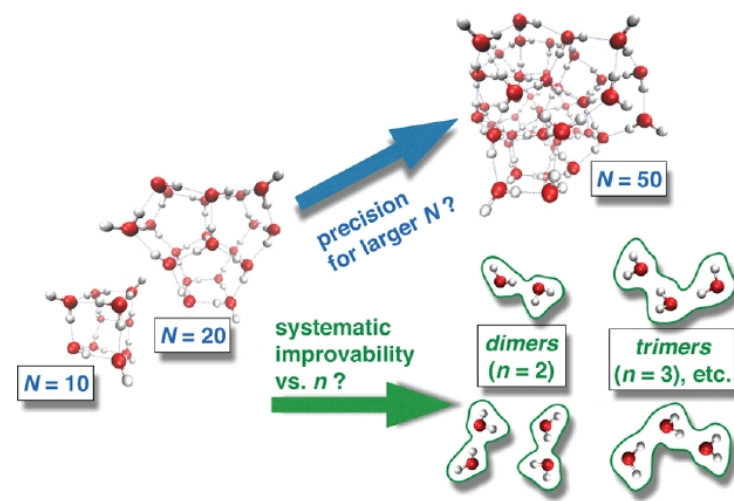
$$\Delta\mathcal{E}^{(n)} = \mathcal{E}^{(n)} - \mathcal{E}^{(n-1)}$$

$$\mathcal{E}^{(n)} = \sum_{I=1}^{\binom{N}{n}} E_I^{(n)} - \sum_{I=1}^{\binom{N}{n}} \sum_{J>I}^{\binom{N}{n}} E_{I \cap J}^{(n)} + \dots + (-1)^{\binom{N}{n}+1} E_{I \cap J \cap \dots \cap I}^{(n)}$$

- (G)MBE super-molecular alternative to SAPT
 - Property decomposition expansion
 - Applicable to any extensive property
- GMBE is a universal energy equation
 - Can put other fragment methods in terms of it
- Basis of current library

Reduces to MBE for disjoint fragments

$$E^{(n)} = \sum_{k=1}^n (-1)^{n-k} \binom{N-k-1}{n-k} \sum_{K=1}^{\binom{N}{k}} E_K^{(k)}$$



Shameless self-promotion:

Richard, Lao, Herbert. Acc. Chem. Res. 47 (2014) 2828-2836.

LibFrag

- Distinguishing components of a fragment method:
 - How are the fragments formed?
 - How are severed bonds dealt with?
 - How are orders $> n$ accounted for?
 - How is BSSE treated?

LibFrag

- Distinguishing components of a fragment method:
 - How are the fragments formed?
 - How are severed bonds dealt with?
 - How are orders $> n$ accounted for?
 - How is BSSE treated?

- Fragmentation Methods:
 - User defined
 - Bond based
 - Distance based

LibFrag

- Distinguishing components of a fragment method:
 - How are the fragments formed?
 - How are severed bonds dealt with?
 - How are orders $> n$ accounted for?
 - How is BSSE treated?

- Fragmentation Methods:
 - User defined
 - Bond based
 - Distance based

- Capping Methods:
 - Static/Dynamic Hydrogen Caps

LibFrag

- Distinguishing components of a fragment method:
 - How are the fragments formed?
 - How are severed bonds dealt with?
 - How are orders $> n$ accounted for?
 - How is BSSE treated?

- Fragmentation Methods:
 - User defined
 - Bond based
 - Distance based

- Capping Methods:
 - Static/Dynamic Hydrogen Caps

- Embedding Methods:
 - Multi-level (trivially supported)
 - Static point charges
 - Iterative point charges
 - Static density embedding (eventually)
 - Iterative density embedding (eventually)

LibFrag

- Distinguishing components of a fragment method:
 - How are the fragments formed?
 - How are severed bonds dealt with?
 - How are orders $> n$ accounted for?
 - How is BSSE treated?

- Fragmentation Methods:
 - User defined
 - Bond based
 - Distance based

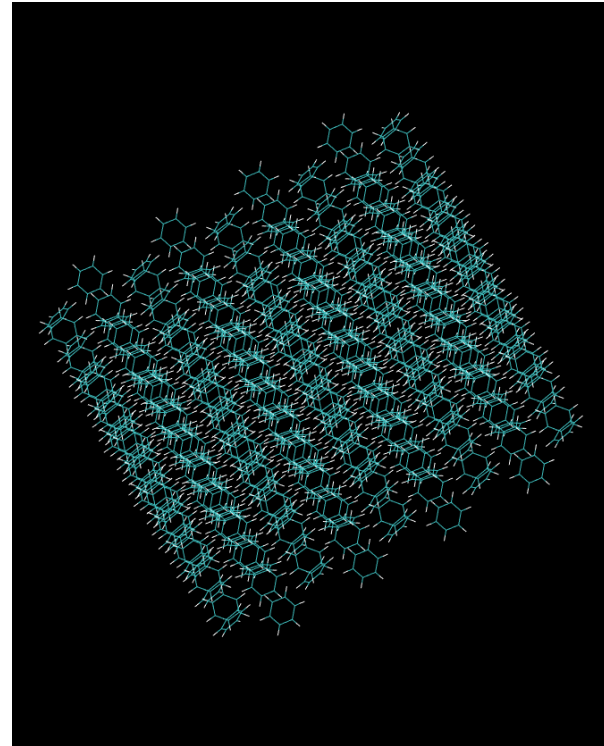
- Capping Methods:
 - Static/Dynamic Hydrogen Caps

- BSSE Methods:
 - Full
 - VMFC(n) [eventually]
 - MBCP(n) [eventually]

- Embedding Methods:
 - Multi-level (trivially supported)
 - Static point charges
 - Iterative point charges
 - Static density embedding (eventually)
 - Iterative density embedding (eventually)

LibFrag

- Highlights:
 - Hybrid MPI/OpenMP
 - Flexible and extensible
 - Highly automated
 - Re-use of SCF guesses
 - Capable of handling very large systems (~5,000 atoms)
 - Space group symmetry exploiting
 - Capable of creating periodic systems from unit cells



MPI Utilities

- Goal: Provide a universal, simple interface to Psi 4's MPI Capabilities
- Automatically:
 - Handles data synchronization
 - Load balancing
 - Communicator allocation

- Plans for:
 - Python Interface
 - Dynamic addition of tasks
 - Prolonged proliferation of distributed data

- As far as user is concerned:
 - MPITask Class
 - Holds details of what you want to parallelize
 - MPIJob Class
 - Gives you your instructions

MPI Utilities

- Label: input for a job. Purely for your convenience
- Could be:
 - Shell quartet for integrals
 - Element number for vector addition
 - Calculation name for databases
 - Atom and component for finite difference

```
//Declare an array of Tasks, with labels of type
//T. T can be anything that is convenient for you
std::vector<MPITask<T> > MyTasks;
for (int task=0; task<NTasks; ++task){
    T Label=GetLabel(task);//Get this task's label
    int Priority=GetPriority(task);//Get this task's priority
}

MPIJob<T> Job(MyTasks);//Make a job manager

std::vector<T2> Data;//Our generated data, of type T2

for(T task=Job.Begin(); !Job.Done(); task=Job.Next())
    Data.push_back(NewData(task)); //Collect the data

//Give everyone a copy of the data point
std::vector<T2> AllData=Job.Synch(Data,1);

//Multiply all the data together and return a value
T2 product=Job.Reduce(Data,1,MULTIPLY);
```


MPI Utilities

- Label: input for a job. Purely for your convenience
- Could be:
 - Shell quartet for integrals
 - Element number for vector addition
 - Calculation name for databases
 - Atom and component for finite difference

- Priority: An integer describing a task's complexity
 - 0 is highest priority
 - Negative is unknown priority

```
//Declare an array of Tasks, with labels of type
//T. T can be anything that is convenient for you
std::vector<MPITask<T> > MyTasks;
for (int task=0; task<NTasks; ++task){
    T Label=GetLabel(task);//Get this task's label
    int Priority=GetPriority(task);//Get this task's priority
}

MPIJob<T> Job(MyTasks);//Make a job manager

std::vector<T2> Data;//Our generated data, of type T2

for(T task=Job.Begin(); !Job.Done(); task=Job.Next())
    Data.push_back(NewData(task)); //Collect the data

//Give everyone a copy of the data point
std::vector<T2> AllData=Job.Synch(Data,1);

//Multiply all the data together and return a value
T2 product=Job.Reduce(Data,1,MULTIPLY);
```

MPI Utilities

- Label: input for a job. Purely for your convenience
- Could be:
 - Shell quartet for integrals
 - Element number for vector addition
 - Calculation name for databases
 - Atom and component for finite difference

- Priority: An integer describing a task's complexity
 - 0 is highest priority
 - Negative is unknown priority

- Call the job for tasks.
- Do whatever the label tells you.
 - Collect your data in the order you are given tasks.

```
//Declare an array of Tasks, with labels of type
//T. T can be anything that is convenient for you
std::vector<MPITask<T> > MyTasks;
for (int task=0; task<NTasks; ++task){
    T Label=GetLabel(task);//Get this task's label
    int Priority=GetPriority(task);//Get this task's priority
}

MPIJob<T> Job(MyTasks);//Make a job manager

std::vector<T2> Data;//Our generated data, of type T2

for(T task=Job.Begin(); !Job.Done(); task=Job.Next())
    Data.push_back(NewData(task)); //Collect the data

//Give everyone a copy of the data point
std::vector<T2> AllData=Job.Synch(Data,1);

//Multiply all the data together and return a value
T2 product=Job.Reduce(Data,1,MULTIPLY);
```

MPI Utilities

- Label: input for a job. Purely for your convenience
- Could be:
 - Shell quartet for integrals
 - Element number for vector addition
 - Calculation name for databases
 - Atom and component for finite difference

- Priority: An integer describing a task's complexity
 - 0 is highest priority
 - Negative is unknown priority

- Call the job for tasks.
- Do whatever the label tells you.
 - Collect your data in the order you are given tasks.

A more advanced Job interface exists for those who know MPI and want to Avoid scatter/gathers

```
//Declare an array of Tasks, with labels of type
//T. T can be anything that is convenient for you
std::vector<MPITask<T> > MyTasks;
for (int task=0; task<NTasks; ++task){
    T Label=GetLabel(task);//Get this task's label
    int Priority=GetPriority(task);//Get this task's priority
}

MPIJob<T> Job(MyTasks);//Make a job manager

std::vector<T2> Data;//Our generated data, of type T2

for(T task=Job.Begin(); !Job.Done(); task=Job.Next())
    Data.push_back(NewData(task)); //Collect the data

//Give everyone a copy of the data point
std::vector<T2> AllData=Job.Synch(Data,1);

//Multiply all the data together and return a value
T2 product=Job.Reduce(Data,1,MULTIPLY);
```

Acknowledgments

- Sherrill Group
- Edmond Chow and Xing Liu
- Georgia Tech
- NSF

