Combining QC codes

Alexander Gaenko

Ames Laboratory
Iowa State University

July 27, 2009

Advisors and collaborators

Advisors:

- Professor Mark S. Gordon
- Professor Theresa L. Windus

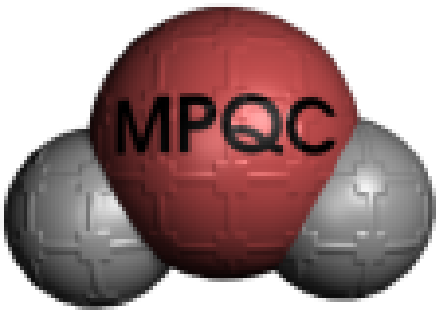
Main collaborators:

- Dr. Ajitha Devarajan
- Dr. Heather Netzloff
- Dr. Meng-Shiou Wu

General directions

Quantum Chemistry packages integration and extension to petascale.
Challenges: the packages usually

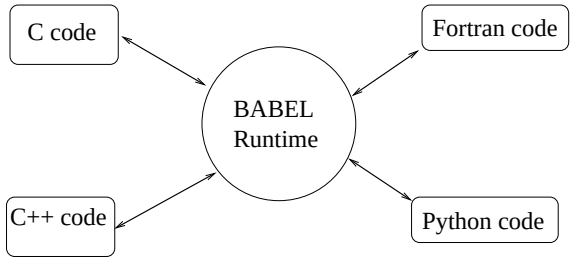
- Were never meant to work together
- Use different means of interprocess communication and memory management
- Have different policies and practices
- Are sometimes written in different languages
- Have different scalability issues and bottlenecks



Approach: CCA Framework

Common Component Architecture (CCA):

- Components — separate pluggable program units, with well-defined access interface
- The framework establishes connection, then the components interact directly
- The framework runs using BABEL runtime: language mix is possible (C/C++, Fortran 77/90, Python, Java are supported)
- Interfaces are defined using SIDL language, BABEL compiler generates glue and skeleton codes



CCA
Common Component Architecture

http://www.cca-forum.org/

https://computation.llnl.gov/casc/components/

Main project: QM/MM or MC with EFP potentials

- Effective Fragment Potential (EFP): almost-quantum potential, so far implemented only in GAMESS
- We want to use MM or MC code from NWChem

Subprojects:

- Make EFP a component — almost done
- Make MM/MC a component — is being done
- Bring DDI to multilevel
- Design Distributed Arrays component
- Address RTDB [lack of] scalability

DDI: general information

- The one and the only interprocess communication used by GAMESS.
- Sync and async message passing, distributed memory paradigm.
- Design focus: light weight, portability, no "unnecessary" functionality.
- Existing implementations run on top of:
  - TCP/IP sockets
  - MPI-1 + sockets
  - Cray's SHMEM
  - PNNL's ARMCI
  - TCP/IP sockets + shared memory
  - Pure MPI-1
  - IBM's LAPI

"Unfair comparison": GA vs. DDI:

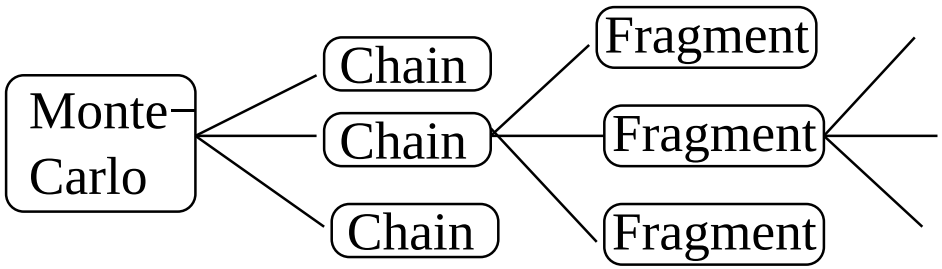
GA	DDI
`$ cd ga-4-0-8`	`$ cd Gamess/ddi`
`$ find . -type f -name '*.c' -o -name '*.h' | wc -l | tail -n 1`	
430	77
`$ find . -type f -name '*.c' -o -name '*.h' | xargs wc -l | tail -n 1`	
151762 total	16315 total

DDI and petascale

- *Grouping* is advantageous in certain QM methods (e.g. FMO)
- DDI supports a model "cluster of SMPs": 2 (and only 2!) level grouping
  - Distributed, replicated, node-replicated memory objects
  - "World", "Group"(e.g. SMP box) and "Masters" process groups

However, DDI needs to be extended — preferably without loss of portability, gain of complexity!

- *URGENT*: Have more than 2 level of grouping



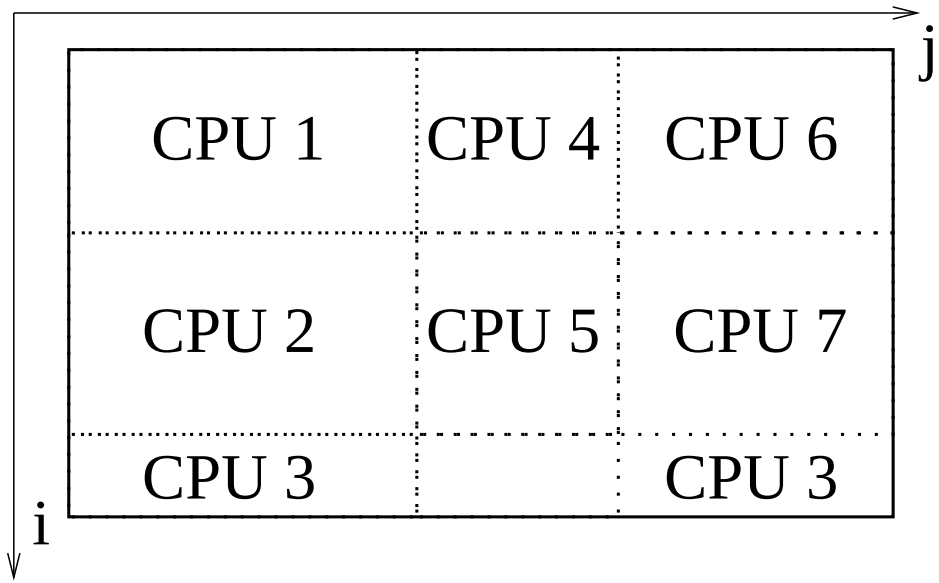
- Accommodate more transports: MPI-2 RMA? InfiniBand? ...?
- Add more reduction operations? Augment with I/O capabilities?

DA: The problem

- Large matrices in QM needs to be distributed
- There are a few competing incompatible interfaces (e.g. GA and DDI)
- Distributed data produced by one component should be consumed by another component — How?

The idea: Lightweight CCA component to deal with this.
Design decisions and limitations:

- Data is a matrix of dimension up to 4.
- Distribution of the matrix is "rectangular patches"
- The component can be built as a layer around an existing library (e.g. DDI or GA)



DA: Open questions

- Memory model: which one?
  - Treat all memory as "shared local" (like MPI-2 RMA): `lock()`–`access`–`unlock()`
  - Treat all memory as remote (like GA, DDI): `put()`/`get()` calls
- Adaptation by existing codes:
  - Make an adapter (wrapper) library to imitate GA or DDI?
  - Have a handle of the implementing (DDI or GA) library user accessible?
- Choice of level of implementation:
  - High: GA, DDI, ...?
  - Middle: MPI-2 RMA, ARMCI, ...?
  - Low: shared memory, InfiniBand, ...?
- Functionality:
  - Are matrix operations needed?
  - Are I/O operations needed?

RTDB: General information

RTDB (Run-Time DataBase) is used in NWChem. It is:

- Mean of persistent information storage.
- Mean of communication between parts of the program.
- Logically – a "type map": a collection of key-value pairs; keys are strings; values are typed arrays.
- Implemented as database files:
  - Replicated: each node has its own file
  - Centralized (seems to be not implemented fully): only a master performs file I/O, results are broadcast.

Example:

```
int handle; char* filename="myfile.db"; int ilen, dlen;
int iarray(ilen); double darray(dlen);
char* ikey, * dkey;
/* .....  */
rtdb_open(filename,"old",handle);
rtdb_get(handle,MA_INT,ikey,iarray,ilen);
rtdb_put(handle,MA_DBL,dkey,darray,dlen);
rtdb_close(handle,"keep");
```

RTDB: Scalability & interoperability challenges

- Mixed data: large arrays and small scalars
- Replication between nodes is done by close–copy–open:
  - Conceptually simple, but:
  - Slow: relies on underlying FS performance
  - What if we don't have FS, or if I/O is expensive?
- Alternative: keeping all data in memory
  (a pure F77 implementation of this exists!)
  - Fast to replicate & operate, but:
  - No persistence, unless backed by a file
  - Large, rarely used values unnecessary consume memory

QUESTION: Can we make use of parallel I/O along with caching of frequently used values?

Topics of interest

The topics of interest would be:

- What are the *efficient* means of interprocess communications on high-end machines — with respect to:
  - Run time?
  - Programming time?
- Remote Memory Access vs. Message Passing approach
- Parallel I/O: efficiency and technique of use.

Thank you!