Common Component Infrastructure

Concrete Implementation and Microkernel Support

A Straw Man

Satish Balay, Bill Gropp, Lois Curfman McInnes, Barry Smith

Mathematics and Computer Science Division

Argonne National Laboratory

Derived using ideas and software tools from PETSc and discussions with members of the CCA Forum

Objective

Define the minimal software infrastructure needed to enable development of numerical application codes using a variety of software components that are not necessarily available as source code.

Requirements

- Application code and components could be a combination of C, C++, Fortran, Java.
- Application codes could be written in a variety of styles (e.g., object-oriented, procedural)

Decisions

- Peer-to-peer relationship among components
- Objects encapsulate all data and functions
  - no free-standing functions or data
- Objects may, internally, use any consistent mechanism to handle parallel communication
- MPI is available as part of the software infrastructure
- Threads may be available

Microkernel Strategy

- Define universal object memory layout
- Define basic kernel objects (always available)
  - Object loader
  - Error handler
  - Profiler
  - Memory allocator

Universal Object Memory Layout

- Representation in memory of object methods
- C/Fortran represented as group of structures
- C++ represented as abstract base classes
- Two issues
  - Common methods to support
  - Details of memory layout, i.e., structure of classes

Common Methods to Support

- Get MPI Communicator from object
- View - visualize, serialize etc
- Reference - increase reference count
- Destroy
- Compose - attach another object (interface) by name
- Query - get attached object (interface) by name
- Compose function - attach method to object
- Query function - get a method from object
- Query languge - get representation of object (interface) in a different programming language

Details of Memory Layout: C

All objects are pointers to

```
struct _CCA {
   int              cookie;
   CCA_Ops *bops;
   void             *ops;
}
```

Details of Memory Layout: C

```
typedef struct {
   int   (*getcomm)(CCA,MPI_Comm*),
         (*view)(CCA,CCAViewer),
         (*reference)(CCA), (*destroy)(CCA);
         (*compose)(CCA,char *,CCA),
         (*query)(CCA,char *,CCA *),
         (*composefunction)(CCA,char *,char *,void *),
         (*queryfunction)(CCA,char *,void **),
         (*composelanguage)(CCA,CCALanguage,void *),
         (*querylanguage)(CCA,CCALanguage,void **);
} CCA_Ops;
```

Details of Memory Layout: C

void *ops is a pointer to the function table for that specific type of object, for example, for vectors:

```
typedef struct {
  int   (*dot)(CCAVector,CCAVector,double *),
        (*axpy)(double, CCAVector,CCAVector),
        (*norm)(CCAVector,double *),
        ….
}
```

Details of Memory Layout: C

void *ops is a pointer to the function table for that specific type of object, for example, for vectors:

```
typedef struct {
   int   (*dot)(CCAVector,CCAVector,double *),
         (*axpy)(double, CCAVector,CCAVector),
         (*norm)(CCAVector,double *),
        ....
}
```

Sample Use of Query Function Interface

```
int KSPGMRESSetRestart(KSP ksp,int max_k )
{
  int ierr, (*f)(KSP,int);

QueryFunction(ksp, "KSPGMRESSetRestart_C",&f);
  if (f) {
    (*f)(ksp,max_k);
  }
  return 0;
}
```

Sample Use of Query Language Interface

```
CCA_Cvector     cvec;           /* C interface to object */
CCA_CPPvector *cppvec;          /* C++ interface to object */

double                  cnorm,cppnorm;

VecNorm(cvec,&cnorm);

QueryLanguage(cvec,CPP,&cppvec);
cppvec->norm(&cppnorm);

/* The two norms should be the same */
```

Details of Memory Layout: C++

```
Class CCA {
  public:
  int cookie;
  virtual int getcomm(MPI_Comm *)                          = 0;
  virtual int view(class CCAViewer *)                      = 0;
  virtual int reference()                                  = 0;
  virtual int compose(char *,class CCA *)                  = 0;
  virtual int query(char *,class CCA **)                   = 0;
  virtual int composefunction(char *,char *,void *)        = 0;
  virtual int queryfunction(char *,void **)                = 0;
  virtual int querylanguage(char *,CCALanguage,void **) = 0;
}
```

Microkernel Strategy

- Define universal object memory layout
- Define basic kernel objects

Basic Kernel Objects

- No time for details here; they'll be presented elsewhere …
- Some issues to ponder ...
  - Are error handler object, profiler object, malloc object etc. really needed? Could these instead be treated as basic, "built-in", "framework" services?
  - Though we cannot fully justify the existence of such objects yet, we think that once we have them, we won't know how we lived without them.

Conclusions

- Objects are pointers to “standardized” function tables
- Basic system requires a minimal number of fundamental objects to be “available”
- Micro-kernel could be, in theory, a few hundreds lines of code