Introduction to cgm*lib*

School of Computer Science
Carleton University
Ottawa

Presenter: Albert Chan
(achan@scs.carleton.ca)
(http://www.scs.carleton.ca/~achan)

2003-01-31

Agenda

- ☺ Introduction
- ☺ Architecture
- ☺ Current Status
- ☺ Features
- ☺ The cgm*lib* in Details
- ☺ Extension
- ☺ Installation
- ☺ Using cgm*lib*
- ☺ Demo
- ☺ Q & A

What is CGM?

- ☺ CGM stands for Coarse Grained Multi-computers
- ☺ Invented by Dr. Frank Dehne et al. (Carleton University)
- ☺ Number of processors significantly less than input data size.
- ☺ Two parameters:
  - ✈ n = size of data;
  - ✈ p = number of processors.
- ☺ CGM algorithms consist alternating local computation and global communication rounds.
- ☺ In each communication rounds, an h-relation can be sent and received by each processor, here $h=O(n/p)$
- ☺ Efficient CGM algorithms minimize both the local computation time and the number of global communication rounds.

Why CGM?

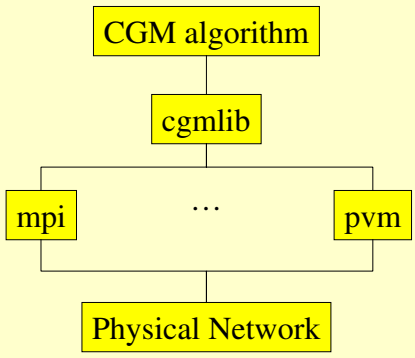
- ☺ Why coarse grained?
  - ✈ It is difficult to built fine grained machine.
  - ✈ We have to use existing machines to simulate fine grained algorithms
  - ✈ The simulation usually results a large amount of small messages to be sent and received from time to time.
  - ✈ Communication is still the bottleneck of the parallel algorithms, so we prefer a small (or fix) amount of large messages to a large amount of small messages.
- ☺ Why CGM?
  - ✈ Although equivalently powerful, other models are usually more difficult to analyze.
  - ✈ All messages in a global communication round are grouped into a fix amount of h-relations, thus minimizing the overhead of communication.

Why cgm*lib*?

- ☺ Most CGM machines are based on the message passing mechanism to exchange data.
- ☺ Although the message passing platforms have been settled to only a few different standards, they are still incompatible to each other. This forces the CGM algorithm authors to modify their algorithms when they move the algorithms to a different system (or when the system administer decides to switch to another platform).
- ☺ The cgm*lib* acts as a "*middle man*" sitting between the CGM algorithms and the underlying message passing platform. Thus allowing the CGM algorithms to be moved to different system with no or minimum changes.
- ☺ The cgm*lib* provides handy methods to allow data easily exchanged amount processors.

The Architecture of cgm*lib*

- ☺ The following diagram shows the relationship of cgm*lib* with other computing elements:



Current Status

- ☺ The cgm*lib* is currently in beta release.
- ☺ It has been installed on the ultra network and the sigma network in the School of Computer Science at Carleton University.
  - ✈ On the ultra network, it is installed at /home/70user2/cgm/cgmlib-ultra
  - ✈ On the sigma network, it is installed at /home/70user2/cgm/cgmlib-sigma
- ☺ It will also be installed on the thog network in HPCVL at Carleton University and the CGM1 network at Dalhouise University, location to be announced later.
- ☺ You are also encouraged to installed it on your own Linux box or in your UNIX account. See later slide for details on installation.

Features of cgm*lib*

- ☺ Implemented in C++.
- ☺ Completely object-oriented design.
- ☺ Easily extensible.
- ☺ Designed with the mind to promote software reuse.
- ☺ Complete source code available.
- ☺ Many examples included.
- ☺ Comes with on-line user manual.
- ☺ Comes with printable user manual in postscript and PDF formats.
- ☺ Currently only MPI is supported.
- ☺ Provides facilities for high level operations such as sorting, prefix sum calculation and array balancing.
- ☺ Allows the CGM to be partitioned into a group of smaller CGMs.

What can be sent?

- ☺ Everything you want to send across the network *MUST* be sub-classed from interface CommObject, which defines the following methods:
  - ✈ copyToArray;
  - ✈ copyFromArray;
  - ✈ clone;
  - ✈ operator=;
  - ✈ sendToOstream;
  - ✈ getSize;
- ☺ Note that most of the methods are pure virtual so you must provide a *CORRECT* implementation to these methods before you can really use your objects.
- ☺ The only exception is "sendToOStream" which, unless overridden, defaults to print out its memory address.

But I Only Want to Send a Simple Integer…

- ☺ A class called BasicCommObject is defined to allow the user to send simple data type (int, long, float, double, char, etc).
- ☺ To use BasicCommObject, do the following:
  - ✈ CommObject *intData = new BasicCommObject <int> (3);
  - ✈ CommObject *floatData = new BasicCommObject <float> (5.5);
  - ✈ CommObject *charData = new BasicCommObject <char> ('a');
- ☺ You cannot use BasicCommObject on pointer types as it is meaningless to send pointers to other processors:
  - ✈ pointer is only an address to the data; and
  - ✈ each processor has its own address space.
- ☹ Never try to do the following (it won't work):
  - ✕ CommObject *pointerData = new BasicCommObject (char *) ("Hello");

Can I Send an Non CommObject?

- ☺ Yes, a class called SimpleCommObject is defined to allow the user to wrap a non-CommObject into a CommObject.
- ☺ To use SimpleCommObject , do the following:
  - ✈ CommObject *myData = new SimpleCommObject <MyDataType> (x);
  - ✈ Here x is an object of MyDataType.
  - ✈ The object to be wrap around into a SimpleCommObject *CANNOT* contain any pointer.

CommObjectList

- ☺ A container to encapsulate the CommObject's to be sent across the network.
- ☺ The data type that most of the methods in cgm*lib* expect.
- ☺ Easier to use than CommObject ** as the container will manage its own memory.
- ☺ Automatically expandable.
- ☺ Array-like behavior.
  - ✈ CommObjectList data;
    CommObject *firstElement = data [0];
- ☺ Built-in boundary check.

Comm

- ☺ The heart of cgm*lib*.
- ☺ Interface only, that means all methods are pure virtual.
- ☺ Some handy constants defined:
  - ✈ ANY_NODE = -1;
  - ✈ PROC_ZERO = 0;
  - ✈ MPICOMM="MPICOMM";
- ☺ A static method is provided to encapsulate the underlying message passing platform:
  - ✈ getComm;

Comm

- ☺ Other (non-static) methods that Comm provides:
  - ✈ getNumberOfProcessors;
  - ✈ getMyId;
  - ✈ synchronize;
  - ✈ send;
  - ✈ receive;
  - ✈ oneToAllBCast;
  - ✈ allToOneGather;
  - ✈ allToAllBCast;
  - ✈ hRelation;
  - ✈ arrayBalancing
  - ✈ partitionCGM
  - ✈ unPartitionCGM
  - ✈ dispose
- ☺ A concrete implementation is provided in the following class:
  - ✈ MPIComm

Request System

- ☺ A request system has been implemented to help individual elements requesting information across the network.
- ☺ Use consolidation technique to avoid overloading any particular processor.
- ☺ Requests sent to the same elements are consolidated into a set of at most p requests.
- ☺ Each processor is handling at most $n/p + p$ requests.
- ☺ Responses are automatically expanded and routed to the original requestor.
- ☺ Each requestor will get its own copy of the response.

Other Operations

- ☺ The following classes provide other higher level operations:
- ☺ ParallelSorter
  - ✈ sort
- ☺ ParallelPrefixSummer
  - ✈ calculatePrefixSum

Other Utilities

- ☺ Some other utilities are also provided :
  - ✈ Timer: for getting timing information
    - ✡ UnixTimer: use Unix wall clock
    - ✡ CPUTimer: use CPU ticks
    - ✡ CGMTimers: a set of timers to time:
      - – total elapse time;
      - – total communication time; and
      - – total computational time.
  - ✈ Random: for generating random numbers
  - ✈ Sorter: for sorting CommObjects
    - ✡ HeapSorter: use heap sort - $O(n\log n)$;
    - ✡ IntegerSorter: use integer sort - $O(n)$.
  - ✈ GeneralUtilities
    - ✡ Provide miscellaneous utilities that may be useful for the users.

Extending cgm*lib*

- ☺ cgm*lib* can be easily extended to provide support to other message passing platforms, such as PVM.
- ☺ In addition to provide a concrete extension of the Comm class (e.g. PVMComm), you also need to modify the following *IN* the Comm Class:
  - ✈ add an include statement to include your new class
    - ✡ e.g. #include “PVMComm.h”
  - ✈ add an interface string
    - ✡ e.g. static char *PVMCOMM = “PVMCOMM”;
  - ✈ change the default if desired
    - ✡ e.g. static char *defaultComm = PVMCOMM;
  - ✈ add another “else if” statement in the getComm method
    - ✡ see next slide for example.

Extending cgmlib

```
Comm *Comm::getComm (int *argc, char ***argv, CGMTimers *timers, char *selector)
{
    if (selector == NULL)
    {
        selector = defaultComm;
    }
    if (strcmp (selector, MPICOMM) == 0)
    {
        return MPIComm::getComm (argc, argv, timers);
    }
    else if (strcmp (selector, PVMCOMM) == 0)
    {
        return PVMComm::getComm (argc, argv, timers);
    }
    return getComm (argc, argv, timers, defaultComm);
}
```


Installation

- ☺ The library is distributed under Gnu's LGPL:
  - ✈ http://www.gnu.org
- ☺ Due to the diversity of the parallel platforms, only source code is distributed.
- ☺ A WWW page has been set up to be the home of cgm*lib*:
  - ✈ http://www.scs.carleton.ca/~cgm
- ☺ Down load the file *cgmlib.src.zip* or *cgmlib.src.tar.gz*
- ☺ Unzip the file
- ☺ Modify the *Makefile* and *Makefile.common* if necessary.
- ☺ Execute "make" (on the top level directory of cgm*lib*)
- ☺ If everything is OK, you can install it by executing "make install"

Using cgm*lib*

- Assume cgm*lib* is installed at /usr/local/cgmlib
- To compile a file:
  - g++ -I/usr/local/cgmlib/include -c <file.cpp>
- To link an object module:
  - g++ -o <executable> <file.o> -L/usr/local/cgmlib/lib -lcgm <otherflag>
  - here <otherflag> is the flag that is specific to the message passing platform being used
  - for example, for MPICH, it may be
    - -L/usr/local/mpich/lib -lpmpich++ -lmpich
- To run the linked program, just follow the normal procedure in your environment, e.g.:
  - mpirun -np <#> <program> <args>

Application: cgm*graph*

- Several examples (mostly graph applications) have been provided, and grouped as cgm*graph*:
  - List Ranking
  - Euler Tours
  - Connected Components
  - Spanning Forest
  - Bipartite Graph Detection

Future Work

- ☺ Provide error handling.
- ☺ Provide support for other message passing platforms such as PVM.
- ☺ Provide Java binding (e.g. through mpiJava or Java RMI)
- ☺ Provide automatic installation using GNU automake and autoconf.
- ☺ Provide default implementation to “middle-level” operation such as array-balancing, broadcast and gather.
- ☺ Integrate cgm*graph* into LEDA.
- ☺ Provide implementation of more graph algorithms
- ☺ Performance tuning.

Demo