Performance Technology for Parallel Component Software

Sameer Shende
sameer@cs.uoregon.edu

Department of Computer and Information Science
NeuroInformatics Center
University of Oregon
Outline

- What is Component Software? [www.cca-forum.org]
- Performance Engineered Component Software
- CCA Performance Observation Component
  - CCAFFEINE (Classic C++)
  - SIDL
- Applications:
  - Optimizer Component
  - Combustion Component
- Concluding remarks
Why Components?

The task of the software development team is to engineer the illusion of simplicity [Booch].
The Good the Bad and the Ugly

- An example of what can lead to a crisis in software:
  - At least 41 different Fast Fourier Transform (FFT) libraries:
  - Many (if not all) have different interfaces
    - different procedure names and different input and output parameters
  - SUBROUTINE FOUR1(DATA, NN, ISIGN)
    - Replaces DATA by its discrete Fourier transform (if ISIGN is input as 1) or replaces DATA by NN times its inverse discrete Fourier transform (if ISIGN is input as -1). DATA is a complex array of length NN or, equivalently, a real array of length 2*NN. NN MUST be an integer power of 2 (this is not checked for!).
The Good the Bad and the Ugly

- An example of what can lead to a crisis in software:
- At least 41 different Fast Fourier Transform (FFT) libraries:
- Many (if not all) have different interfaces
  - different procedure names and different input and output parameters
- SUBROUTINE FOUR1(DATA, NN, ISIGN)
  - Replaces DATA by its discrete Fourier transform (if ISIGN is input as 1) or replaces DATA by NN times its inverse discrete Fourier transform (if ISIGN is input as -1). DATA is a complex array of length NN or, equivalently, a real array of length 2*NN. NN MUST be an integer power of 2 (this is not checked for!).
What Are Components [Szyperski]

- A component is a binary unit of independent deployment
  - well separated from other components
    - fences make good neighbors
  - can be deployed independently

- A component is a unit of third-party composition
  - is composable (even by physicists)
  - comes with clear specifications of what it requires and provides
  - interacts with its environment through well-defined interfaces

- A component has no persistent state
  - temporary state set only through well-defined interfaces
  - throw away that dependence on global data (common blocks)

- Similar to Java packages and Fortran 90 modules (with a little help)
Component Technology

□ What is a component?

- Implementation provides functionality but hides details
  - No direct access is possible
- Interface provides access to component functionality
  - Access “ports” are well-defined and generated by tools
- Matching connector links component interfaces
  - Constructed by framework and hidden from users
Component Technology Features

- Interoperability across multiple languages
  - Language independent interfaces (C/C++, Fortran, Java,…)
  - Automatically generated bindings to working code

- Interoperability across multiple platforms
  - Computer systems hardware independence
  - Operating systems independence

- Transparent execution model
  - Serial, parallel, and distributed system

- Incremental evolution of application software

- Components promote software reuse

- Components are “plug-and-play”
Language Interoperability

Simulation Framework (C)

Numerical Routines (f77)

Solver Library (C++)

Callback Handlers (Python)

Scripting Driver (Python)

Visualization System (Java)
Mixing Languages is Hard!

Natural:
cfortran.h
SWIG
JNI
Siloon
Chasm
Platform Dependent
Babel makes all supported languages peers

Once a library has been “Babelized” it is equally accessible from all supported languages.

This is not an LCD Solution!
Babel’s Mechanism for Mixing Languages

- Code Generator
- Runtime Library

SIDL interface description → Babel Compiler

- XML
- C
- C++
- F77
- Python
- Java

Babel Runtime

Application
version greetings 1.0;
package greetings {
    interface Hello {
        void setName( in string name );
        string sayIt ( );
    }
    class English implements-all Hello {  }
}
Library Developer Does This...

- `babel --server=C++ greetings.sidl`
- Add implementation details
- Compile & Link into Library/DLL
namespace greetings {
    class English_impl {
        private:
            // DO-NOT-DELETE splicer.begin(greetings.English._impl)
            string d_name;
            // DO-NOT-DELETE splicer.end(greetings.English._impl)

        string
        greetings::English_impl::sayIt()
        throw ()
        {
            // DO-NOT-DELETE splicer.begin(greetings.English.sayIt)
            string msg("Hello ");
            return msg + d_name + "!");
            // DO-NOT-DELETE splicer.end(greetings.English.sayIt)
        }
    }
}
- `babel --client=F77 greetings.sidl`
- Compile & Link generated Code & Runtime
- Place DLL in suitable location
Common Component Architecture Specification

Scientific IDL

Proxy generator

Component 1

Component 2

CCA Services

Any CCA compliant framework

Repository

CCA ports

Framework-specific part of CCA ports

Abstract configuration API

Repository API

Builder
CCA Concepts: Ports

- Designing for interoperability and reuse requires “standard” interfaces
- *Ports* define how components interact
  - Through well-defined interfaces (ports)
  - In OO languages, a port is a class or interface
  - In Fortran, a port is a set of subroutines or a module
- Components may *provide* ports
  - Implement the class or subroutines of the port
- Components may *use* ports
  - Call methods or subroutines in the port
- Links denote a caller/callee relationship
CCA Concepts: Frameworks

- Provides the means to “hold” components and compose them into applications
- Allow exchange of ports among components without exposing implementation details
- Provide a small set of standard services to components
  - Builder services allow programs to compose CCA apps
- Frameworks may make themselves appear as components in order to connect to components in other frameworks
- Specific frameworks support specific computing models
Numerically integrate a continuous function
Use two different techniques
Lines show port connections

Dashed lines are alternate port connections

\[
f(x) = \int_a^b f(x) \, dx, \quad x_n \text{ uniformly distributed over } [a,b]
\]
CCA Framework Prototypes

- CCAFFEINE
  - SPMD/SCMD parallel, direct connect
  - Direct connection
- CCAT / XCAT
  - Distributed network
  - Grid Web services
- SCIRun
  - Parallel, multithreaded, direct connect
- Decaf
  - Language interoperability via Babel
- Legion (under development)
Performance-Engineered Component Software

- Intra- and Inter-component performance engineering
- Four general parts:
  - Performance observation
    - integrated measurement and analysis
  - Performance query and monitoring
    - runtime access to performance information
  - Performance control
    - mechanisms to alter performance observation
  - Performance knowledge
    - characterization and modeling
- Consistent with component architecture / implementation
Main Idea: Extend Component Design

- Extend the programming and execution environment to be performance observable and performance aware.
Performance Observation and Component

- Performance measurement integration in component form
- Functional extension of original component design (↔)
  - Include new component methods and ports (↑) for other components to access measured performance data
  - Allow original component to access performance data
    - Encapsulate as tightly-coupled and co-resident performance observation object
    - POC “provides” port allow use of optimized interfaces (↔) to access "internal" performance observations
Performance Knowledge

- Describe and store “known” component performance
  - Benchmark characterizations in performance database
  - Empirical or analytical performance models
- Saved information about component performance
  - Use for performance-guided selection and deployment
  - Use for runtime adaptation
- Representation must be in common forms with standard means for accessing the performance information
  - Compatible with component architecture
Component Performance Repository

- Performance knowledge storage
  - Implement in component architecture framework
  - Similar to CCA component repository
  - Access by component infrastructure

- View performance knowledge as component (PKC)
  - PKC ports give access to performance knowledge
  - ↓ to other components, ← back to original component
  - Static/dynamic component control and composition
  - Component composition performance knowledge
Performance Engineering Support in CCA

- Define a standard observation component interface for:
  - Performance measurement
  - Performance data query
  - Performance control (enable/disable)

- Implement performance interfaces for use in CCA
  - TAU performance system
  - CCA component frameworks (CCAFFEINE, SIDL/Babel)

- Demonstrations
  - Optimizing component
    - picks from a set of equivalent CCA port implementations
  - Flame reaction-diffusion application
CCA Performance Observation Component

- Design measurement port and measurement interfaces
  - Timer
    - start/stop
    - set name/type/group
  - Control
    - enable/disable groups
  - Query
    - get timer names
    - metrics, counters, dump to disk
  - Event
    - user-defined events
namespace performance {
namespace ccaports {

class Measurement: public virtual classic::gov::cca::Port {
public:
    virtual ~Measurement (){}

    /* Create a Timer interface */
    virtual performance::Timer* createTimer(void) = 0;
    virtual performance::Timer* createTimer(string name) = 0;
    virtual performance::Timer* createTimer(string name, string type) = 0;
    virtual performance::Timer* createTimer(string name, string type, string group) = 0;

    /* Create a Query interface */
    virtual performance::Query* createQuery(void) = 0;

    /* Create a user-defined Event interface */
    virtual performance::Event* createEvent(void) = 0;
    virtual performance::Event* createEvent(string name) = 0;

    /* Create a Control interface for selectively enabling and disabling * the instrumentation based on groups */
    virtual performance::Control* createControl(void) = 0;
};
}
namespace performance {
class Timer {
    public:
    virtual ~Timer() {}

    /* Implement methods in a derived class to provide functionality */

    /* Start and stop the Timer */
    virtual void start(void) = 0;
    virtual void stop(void) = 0;

    /* Set name and type for Timer */
    virtual void setName(string name) = 0;
    virtual string getName(void) = 0;
    virtual void setType(string name) = 0;
    virtual string getType(void) = 0;

    /* Set the group name and group type associated with the Timer */
    virtual void setGroupName(string name) = 0;
    virtual string getGroupName(void) = 0;
    virtual void setGroupId(unsigned long group) = 0;
    virtual unsigned long getGroupId(void) = 0;
};
}
Use of Observation Component in CCA Example

```cpp
#include "ports/Measurement_CCA.h"
...
double MonteCarloIntegrator::integrate(double lowBound, double upBound, int count) {
    classic::gov::cca::Port * port;
    double sum = 0.0;
    // Get Measurement port
    port = frameworkServices->getPort("MeasurementPort");
    if (port)
        measurement_m = dynamic_cast<performance::ccaports::Measurement * >(port);
    if (measurement_m == 0){
        cerr << "Connected to something other than a Measurement port";
        return -1;
    }
    static performance::Timer* t = measurement_m->createTimer(
        string("IntegrateTimer"));
    t->start();
    for (int i = 0; i < count; i++) {
        double x = random_m->getRandomNumber ();
        sum = sum + function_m->evaluate (x);
    }
    t->stop();
}
```
Using TAU Component in CCAFEINE

```plaintext
repository get TauTimer
repository get Driver
repository get MidpointIntegrator
repository get MonteCarloIntegrator
repository get RandomGenerator
repository get LinearFunction
repository get NonlinearFunction
repository get PiFunction

create LinearFunction lin_func
create NonlinearFunction nonlin_func
create PiFunction pi_func
create MonteCarloIntegrator mc_integrator
create RandomGenerator rand

create TauTimer tau
connect mc_integrator RandomGeneratorPort rand RandomGeneratorPort
connect mc_integrator FunctionPort nonlin_func FunctionPort
connect mc_integrator TimerPort tau TimerPort
create Driver driver
connect driver IntegratorPort mc_integrator IntegratorPort
go driver Go
quit
```
version performance 1.0;
package performance
{
    interface Timer
    {
        /* Start/stop the Timer */
        void start();
        void stop();

        /* Set/get the Timer name */
        void setName(in string name);
        string getName();

        /* Set/get Timer type information (e.g., signature of the routine) */
        void setType(in string name);
        string getType();

        /* Set/get the group name associated with the Timer */
        void setGroupName(in string name);
        string getGroupName();

        /* Set/get the group id associated with the Timer */
        void setGroupId(in long group);
        long getGroupId();
    }...
interface Control
{
    /* Enable/disable group id */
    void enableGroupId(in long id);
    void disableGroupId(in long id);

    /* Enable/disable group name */
    void enableGroupName(in string name);
    void disableGroupName(in string name);

    /* Enable/disable all groups */
    void enableAllGroups();
    void disableAllGroups();
}

/* Implementation of performance component Control interface*/
class TauControl implements all Control
{
}

/* Implementation of performance component Measurement interface*/
class TauMeasurement implements all Measurement, gov.cca.Component
{
}
SIDL Interface : Query

/* Query interface to obtain timing information */
interface Query
{
  /* Get the list of Timer and Counter names */
  array<string> getTimerNames();
  array<string> getCounterNames();
  void getTimerData(in array<string> timerList,
    out array<double, 2> counterExclusive,
    out array<double, 2> counterInclusive,
    out array<int> numCalls,
    out array<int> numChildCalls,
    out array<string> counterNames,
    out int numCounters);

  /* Writes instantaneous profile to disk in a dump file. */
  void dumpProfileData();

  /* Writes the instantaneous profile to disk in a dump file whose name
   * contains the current timestamp. */
  void dumpProfileDataIncremental();

  /* Writes the list of timer names to a dump file on the disk */
  void dumpTimerNames();

  /* Writes the profile of the given set of timers to the disk. */
  void dumpTimerData(in array<string> timerList);

  /* Writes the profile of the given set of timers to the disk. The dump
   * file name contains the current timestamp when the data was dumped. */
  void dumpTimerDataIncremental(in array<string> timerList); }

/* User defined event profiles for application specific events */

interface Event
{
    /* Set the name of the event */
    void setName(in string name);

    /* Trigger the event */
    void trigger(in double data);
}
Measurement Port Implementation

- Use of Measurement port (i.e., instrumentation)
  - independent of choice of measurement tool
  - independent of choice of measurement type

- TAU performance observability component
  - Implements the Measurement port
  - Implements Timer, Control, Query, Control
  - Port can be registered with the CCAFEINE framework

- Components instrument to generic Measurement port
  - Runtime selection of TAU component during execution
  - TauMeasurement_CCA port implementation uses a specific TAU library for choice of measurement type
What’s Going On Here?

Two instrumentation paths using TAU API

application component

performance component

TAU API

Two query and control paths using TAU API

runtime TAU performance data

Alternative implementations of performance component

application component

other API

TAU API

Alternative implementations of performance component
Components are “plug-and-play”

- One can choose from a set of equivalent port implementations based on performance measurements
- An outside agent can monitor and select an optimal working set of components
Component Optimizing Performance Results
Sandia National Laboratory

- DOE SciDAC project (http://cfrfs.ca.sandia.gov)
- Jaideep Ray

Component-based simulation and analysis

- Sandia’s CCAFFEINE framework
- Toolkit components for assembling flame simulation
  - integrator, spatial discretizations, chemical/transport models
  - structured adaptive mesh, load-balancers, error-estimators
  - in-core, off-machine, data transfers for post-processing
- Components are C++ and wrapped F77 and C code

Kernel for 3D, adaptive mesh low Mach flame simulation
Flame Reaction-Diffusion Demonstration

CCAFFEINE
Meeting CCA Performance Engineering Goals?

- Language interoperability?
  - SIDL and Babel give access to all supported languages
  - TAU supports multi-language instrumentation
  - Component interface instrumentation automated with PDT

- Platform interoperability?
  - Implement observability component across platforms
  - TAU runs wherever CCA runs

- Execution model transparent?
  - TAU measurement support for multiple execution models

- Reuse with any CCA-compliant framework?
  - Demonstrated with SIDL/Babel, CCAFEINE, SCIRun
Component software is a natural model for developing applications for the Grid
- ICENI (Imperial College), CCAT / XCAT (U. Indiana)

Our work leverages abstraction power of CCA as well as the infrastructure of CCA frameworks
- Similarly leverage Grid infrastructure and services
- Mostly riding back of CCA framework development

Application-level performance view coupled with Grid resource assessment and monitoring
- More responsive to performance dynamics
- Beginning work with NWS forecaster in applications
Meeting CCA Performance Engineering Goals?

- Component performance knowledge?
  - Representation and performance repository work to do
  - Utilize effectively for deployment and steering
  - Build repository with TAU performance database

- Performance of component compositions?
  - Component-to-component performance
    - Per connection instrumentation and measurement
    - Utilize performance mapping support
  - Ensemble-wide performance monitoring
    - connect performance “producers” to “consumers”
    - component-style implementation
Concluding Remarks

- Parallel component systems pose challenging performance analysis problems that require robust methodologies and tools.

- New performance problems will arise
  - Instrumentation and measurement
  - Data analysis and presentation
  - Diagnosis and tuning
  - Performance modeling

- Performance engineered components
  - Performance knowledge, observation, query and control

Available from:
http://www.cs.uoregon.edu/research/paracomp/tau/tauprofile/dist/taucomponent.tar.gz
Support Acknowledgement

☐ TAU and PDT support:
  ✷ Department of Energy (DOE)
    ➢ DOE 2000 ACTS contract
    ➢ DOE MICS contract
    ➢ DOE ASCI Level 3 (LANL, LLNL)
    ➢ U. of Utah DOE ASCI Level 1 subcontract
  ✷ DARPA
  ✷ NSF National Young Investigator (NYI) award