Language and Compiler Support for Adaptive Distributed Applications

Vikram Adve, Vinh Vi Lam, Brian Ensink

Computer Science Department
University of Illinois at Urbana-Champaign
www.cs.uiuc.edu/~vadve/adaptive.html

Thanks: **NSF (NGS99, OSC99)**

Outline

Adaptive Distributed Applications: 2 Examples

Limitations of current programming strategies

Our Approach: Program Control Language (PCL)

- PCL Language
- PCL compiler support
- A general framework for describing adaptation

Current and Future work

Adaptive Distributed Applications

Adaptive ≡ change runtime behavior to meet performance, availability, or QoS goals

This is the future!

- **Parallel computing on shared cluster or the Grid** where #processors changes dynamically
- **Distributed multimedia or commercial codes** using shared networks, routers, servers, and clients
- **Mobile applications** where user devices, network connections, and power change rapidly

Example 1: Parallel Code on Grid

ATR: Parallel Stochastic Optimization

[*Wright & Linderoth, U.Wisc.*]

Long-running Master-Worker Code

- real problems use 100s - 1000 processors for 1-2 days
- higher parallelism ⇒ slow convergence, lower efficiency

Platform: Condor + Grid (PVM)

- #processors varies widely during execution

⇒ **must adapt**

Example 2: Distributed Multimedia

Distributed Video Tracking

[*Nahrstedt et al., UIUC*]

Client-Server Code

- Client runs multiple tracking algorithms per frame

Platform: Open network using TCP

- Middleware allocates bandwidth, CPU among applications

QoS Metric: Tracking Precision

- Network bandwidth: affects frame rate
- CPU load at client: affects cost of trackers

⇒ **must adapt**

Current Programming Strategies

Middleware: Odyssey, Agilos, TIMELY, GrADS

Adaptation code is deeply embedded within application

- Complex applications become more complex
- Understanding and changing adaptive behavior difficult
- Compilers cannot identify adaptation behavior

User must do performance estimation and modeling

- Middleware provides measurement support only
- Each application uses its own modeling strategies

No framework to reason about and describe adaptation

Program Control Language: PCL

1. Separates adaptation from base application
2. Exposes adaptation metrics, interfaces

Extensions to C++ or Java (PCLC or PCLJ):

```
Adaptor <name> targets <base-class-name> {
    ControlParameters { … <list> … }
    ControlMethods    { … <list> … }
    Metric …      // TimedInterval, RateMetric, SampledEvent
    …
    Event …       // e.g., Metric M crosses threshold X%
    …             // EventHandler: asynchronous adaptation
    Adapt();      // synchronous adaptation
}
```

Compiler Support for PCL

Source-to-Source: PCLC → C++ or PCLJ →Java

1. Implement Adaptor as subclass of target class
   - Enforce access rights to base-class and superclasses
   - Enable inheritance of adaptors
2. Insert instrumentation for each Metric
   - TimedInterval: insert timer code around interval
   - RateMetric: insert counter and program timers
   - SampledMetric: insert counter and program timers

PCLC Fragment for ATR

```
Adaptor ATRAdaptor targets LShapedDriver {
    ControlParameters {
        LShapedDriver :: numInBasket;        // B
        LShapedDriver :: tasks_per_iterate;  // T
    };
    Metric iterateRate  (LShapedWorker::numIterates, RateMetric);
    Metric tTask        (LShapedWorker::execute_task, TimedInterval);
    …
    Adapt() {
        if (smallNumProcsChange.TestEvent()) {
            AdaptNumTasksPerIterate();
        } else if (largeNumProcsChange.TestEvent()) {
            AdaptBasketSize();
        } else …
    } }
```

PCLC Fragment for Video Tracker

```
Metric AdaptedCpu (CpuAdaptor::AdaptedCpu, SampledMetric);
Adaptor TrackerAdaptor targets TrackerManager {
    ControlMethods {
        TrackerManager::AddTracker();
        TrackerManager::RemoveTracker();
    };
    Metric duration  (TrackerManager::trackingTime, TimedInterval);
    Event adaptedCpuChanged (AdaptedCpu, Changed, changedProc);
    void changedProc() {
        // Add or remove trackers to meet new AdaptedCpu value
        …
    }}
```

A General Adaptation Framework

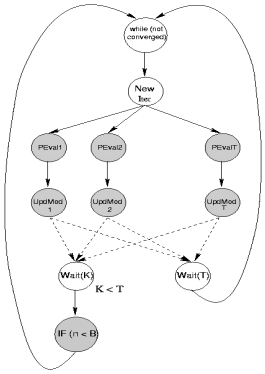
Describe adaptation as changes to a task graph

Static task graph captures sequential flow and distributed structure

Adaptation operations:
- Add / delete tasks
- Modify task parameters (control-flow or computation)
- Modify semantics of edges

Implementation techniques:
- Reflection: task graph drives execution
- Dynamic compilation via code templates

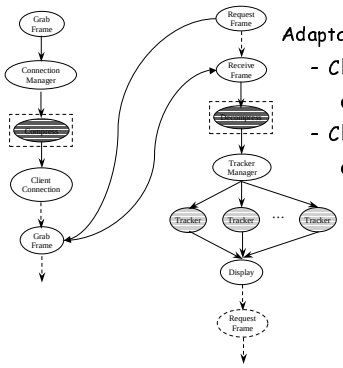
Task Graph of ATR



Adaptations:

- B: change IF parameter
- T: add/delete tasks (PEval, UpdMod)

Task Graph of Video Tracking



Adaptations:

- Change tracker list: add / delete tasks
- Change video format: add / delete tasks

Semi-Automatic Prediction Framework

Task Graph Drives Overall Model

Automatic Prediction of Total Performance

Automatic Instrumentation of Task, Comm Parameters

- E.g., Task times, message latency in ATR
- E.g., Tracker costs, video latency in Video Tracking

Manual Models for Algorithmic Behavior

- E.g., How does B affect convergence?
- E.g., How do frame rate, tracker cost affect precision?

Manual Specification of Adaptation Goal

Current and Future Work

Current work:

- PCL versions of ATR, Video Tracking
- PCLC → C++ Compiler
- Language syntax for task graph framework

Future Work

- Compiler and runtime for task graph framework
- Performance prediction framework
- Other application codes