Programming Support for Distributed Optimization and Control in Cyber-Physical Systems

Rahul Balani, Kaisen Lin*, Lucas Wanner, Jonathan Friedman, Mani B. Srivastava, Rajesh K. Gupta*

University of California, Los Angeles
*University of California, San Diego
Large-scale Coordination and Control
Networked Optimization

Personalized Light Control

Optimization of a cost function

\[ \hat{I}^* = \arg \min_{\hat{I}} \sum_{i=1}^{N} f_i(\hat{I}, \phi_i, L_i^*) \]

Sum of squared error in actual and desired light intensity

\[ f_i = (L_i(\hat{I}, \phi_i) - L_i^*)^2 \]

Control Inputs

Sensor Data

Desired User Intensities
Distributed Optimization Algorithms

• Select and implement the best algorithm for the deployment
  • Performance (Latency)
  • Cost (Number of messages)
Programming is Hard

• Complex inter-node coordination
  • Sharing of resources
  • Act in some sequence
Personalized Light Control

Top-down View
Incremental Subgradient Method

\[
\hat{I}_i^{(k)} = \begin{pmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{pmatrix}_i
\]
Incremental Subgradient Method

\[ \hat{I}_i^{(k)} = \begin{pmatrix} I_1^{(k)} \\ I_2^{(k)} \\ I_3^{(k)} \\ I_4^{(k)} \end{pmatrix} \]
Incremental Subgradient Method

\[ I_i^{(k)} = \begin{pmatrix} I_1^{(k)} \\ I_2^{(k)} \\ I_3^{(k)} \\ I_4^{(k)} \end{pmatrix} \]
Incremental Subgradient Method

\[ \hat{I}_i^{(k)} = \begin{pmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{pmatrix} \]
Programming is Hard

• Complex inter-node coordination
  • Sharing of resources
  • Act in some sequence

• RESULT: error-prone and inefficient implementations

• Selection has become harder
Current State

- Sensor Network Programming
  - Hood, Abstract Regions, TeenyLIME etc.
  - Pleiades, MacroLab, Flask, WaveScript etc.
- **Missing** support for
  - Large-scale actuator coordination and control
  - Algorithm and Deployment Specific (ADS) Performance Optimization
Incremental Subgradient Method

\[
\hat{I}_i^{(k)} = \begin{pmatrix}
I_1^{(k)} \\
I_2 \\
I_3 \\
I_4_i
\end{pmatrix}
\]

\[
\text{Communication latency} = O\left( N \cdot \frac{M}{p_{\text{link}}} \cdot k \right)
\]

- Number of hops in each iteration (cycle)
- Number of packets at each hop
Spatial Locality

Personalized Light Control

Top-down View
A Detailed Inspection

\[ \hat{I}_3^{(k)} = \begin{pmatrix} - & - \\ I_3 & I_4 \end{pmatrix}^{(k)} \]

\[ \hat{I}_4^{(k)} = \begin{pmatrix} - \\ I_2 & I_3 \end{pmatrix}^{(k)} \]

\[ \hat{I}_1^{(k)} = \begin{pmatrix} I_1 \\ - & - \end{pmatrix}^{(k)} \]

\[ \hat{I}_2^{(k)} = \begin{pmatrix} I_1 \\ - \end{pmatrix}^{(k)} \]
Localized Data Dependencies

\[ \hat{I}_3^{(k)} = \begin{pmatrix} I_1 & - & - \\ - & I_3 & - \\ - & - & I_4 \end{pmatrix} \]

\[ \hat{I}_4^{(k)} = \begin{pmatrix} - \\ I_2 \\ - \end{pmatrix} \]

\[ \hat{I}_1^{(k)} = \begin{pmatrix} I_1 & - & - \\ - & - & I_4 \end{pmatrix} \]

\[ \hat{I}_2^{(k)} = \begin{pmatrix} I_1 \\ I_2 \\ - \end{pmatrix} \]
\[ \hat{I}_3^{(k)} = \begin{bmatrix} - & - & I_3 & I_4 \end{bmatrix}_{3}^{(k)} \]

\[ \hat{I}_1^{(k)} = \begin{bmatrix} I_1 & - & - & I_4 \end{bmatrix}_{1}^{(k)} \]

\[ \hat{I}_4^{(k)} = \begin{bmatrix} I_2 & I_3 & - & - \end{bmatrix}_{4}^{(k)} \]

\[ \hat{I}_2^{(k)} = \begin{bmatrix} I_1 & I_2 & - & - \end{bmatrix}_{2}^{(k)} \]
Control Flow

\[ \hat{I}_3^{(k)} = \begin{pmatrix} I_1 & - & - \\ I_3 \\ I_4 \end{pmatrix} \]

\[ \hat{I}_1^{(k)} = \begin{pmatrix} I_1 & - \\ I_3 \\ I_4 \end{pmatrix} \]

\[ \hat{I}_4^{(k)} = \begin{pmatrix} I_2 \\ I_3 \\ - \end{pmatrix} \]

\[ \hat{I}_2^{(k)} = \begin{pmatrix} I_1 \\ I_2 \\ - \end{pmatrix} \]
Data Flow

Iteration 1

Iteration 2
Data Flow

Iteration 1

Iteration 2

Time
Pipelined and Parallel Execution

[Diagram showing a process with nodes 1, 2, 3, and 4, with arrows indicating dependencies and time progression.]
Pipelined and Parallel Execution

Iteration 1

Iteration 2

Reduced Latency
Alternate Control Flow

\[
\hat{I}_1^{(k)} = \begin{pmatrix} I_1 & - & - \\ - & I_3 & - \\ - & - & I_4 \end{pmatrix}^{(k)}
\]

\[
\hat{I}_3^{(k)} = \begin{pmatrix} - & I_3 & - \\ - & - & I_4 \\ I_1 & - & - \end{pmatrix}^{(k)}
\]

\[
\hat{I}_2^{(k)} = \begin{pmatrix} - & I_2 & - \\ - & - & - \end{pmatrix}^{(k)}
\]

\[
\hat{I}_4^{(k)} = \begin{pmatrix} I_2 & - \\ I_3 & - \\ - & - \end{pmatrix}^{(k)}
\]
Alternate Control Flow

\[
\widehat{I}_3^{(k)} = \begin{pmatrix} - & - & I_3 & I_4 \\ \end{pmatrix}^{(k)}_3
\]

\[
\widehat{I}_1^{(k)} = \begin{pmatrix} I_1 & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ I_4 & \cdot & \cdot \\ \end{pmatrix}^{(k)}_1
\]

\[
\widehat{I}_4^{(k)} = \begin{pmatrix} I_2 \\ I_3 \\ - \\ \end{pmatrix}^{(k)}_4
\]

\[
\widehat{I}_2^{(k)} = \begin{pmatrix} I_1 \\ I_2 \\ - \\ \end{pmatrix}^{(k)}_2
\]
Data Flow

Iteration 1

Iteration 2

Time

Iteration 1

Iteration 2
Data Flow

No Performance Optimization!!
Questions

- Which programming abstractions?
- Rapid and error-free implementation
- How to enable ADS performance optimizations?
HOTLINE

Shared Memory (with caching)
HOTLINE

Shared Memory (with caching)

Rapid Implementation
Concise API
HOTLINE

Coherency
Locks: Deadlock-free, Safe, Fair
Cache Synchronization

Shared Memory
(with caching)

Rapid Implementation
Concise API
HOTLINE

Coherency
Locks: Deadlock-free, Safe, Fair
Cache Synchronization

Shared Memory (with caching)

Rapid Implementation
Concise API

Performance Optimization
Algorithm and Deployment Specific
HOTLINE

Coherency
Locks: Deadlock-free, Safe, Fair
Cache Synchronization

Performance Optimization
Algorithm and Deployment Specific

Shared Memory
(with caching)

Rapid Implementation
Concise API
HOTLINE

Actuators

Controllers

Shared Memory

Control Input

Data

Hood

Distributed Algorithm

Driver

Sensor
HOTLINE

Controllers

Data

Actuators

Driver

Sensors

Control Input

Cache

Shared Memory

Hood

Distributed Algorithm
HOTLINE

Controllers

Distributed Algorithm

Actuators

Sensors

Data

Driver

Control Input

Hood

Shared Memory

Cache

Explicit Locks

Distributed Algorithm

Network Communication
PhyLock API
PhyLock API

Controllers

PhyLock.acquire(<priority>)

Distributed Algorithm

Coarse Locks
PhyLock API

Controllers

Distributed Algorithm

PhyLock.acquire(<priority>)
PhyLock API

Controllers

PhyLock.acquire(<priority>)
PhyLock.granted():
    SharedVariable.get/put() ...

Distributed Algorithm
Distributed Algorithm

PhyLock.acquire(<priority>)
PhyLock.granted():
    SharedVariable.get/put() ...
PhyLock.release()
Resource Sub-system

Design Goal

\[
\begin{bmatrix}
- & - \\
W & I_3 \\
W & I_4
\end{bmatrix}^{(k)}
\]

\[
\begin{bmatrix}
I_2 \\
I_3 \\
W
\end{bmatrix}
\]

\[
\begin{bmatrix}
W \\
W \\
W
\end{bmatrix}
\]

\[
\begin{bmatrix}
- \\
I_1 \\
- \\
- \\
W
\end{bmatrix}^{(k)}
\]

\[
\begin{bmatrix}
I_1 \\
I_2 \\
W
\end{bmatrix}
\]

\[
\begin{bmatrix}
W \\
W \\
W
\end{bmatrix}
\]

\[
\begin{bmatrix}
- \\
- \\
2^{5}
\end{bmatrix}
\]
Resource Sub-system

Design Goal

Read-Write & Write-Write Conflicts
Resource Sub-system

Design Goal

COORDINATION
CLIQUES
Resource Sub-system

Design Goal

COORDINATION CLIQUES
Coordination Cliques

PhyLock: Distributed Mutex
Coordination Cliques

PhyLock: Distributed Mutex

REQUEST

REQUEST

REQUEST

REQUEST
Coordination Cliques

PhyLock: Distributed Mutex

1 > 3

1 > 2
Coordination Cliques

PhyLock: Distributed Mutex

Locked

1 > 3

1

REPPLY

3

REPPLY

2

REPPLY

1 > 2

REPPLY

1
Coordination Cliques

PhyLock: Distributed Mutex
Coordination Cliques

Cache Synchronization

\[
\begin{pmatrix}
I_1
\end{pmatrix}^{(k)}
\]

\[
\begin{pmatrix}
I_2 \\
I_3
\end{pmatrix}
\]

\[
W
\]

\[
\begin{pmatrix}
- \\
- \\
I_4
\end{pmatrix}
\]

\[
\begin{pmatrix}
- \\
I_3
\end{pmatrix}
\]

\[
W
\]

\[
\begin{pmatrix}
- \\
I_4
\end{pmatrix}
\]

\[
W
\]

\[
\begin{pmatrix}
- \\
- \\
I_1
\end{pmatrix}
\]

\[
\begin{pmatrix}
- \\
- \\
I_2
\end{pmatrix}
\]

\[
W
\]
HOTLINE

Actuators

Sensors

Controllers

Influence Zone

Control Input

Driver

Sensing Zone

Data

Driver

Data

Cache

Shared Memory

Hood

Distributed Algorithm

Resources

Explicit Locks

Clique

Mutex

Network Communication

Driver

28
HOTLINE

Actuators

Influence Zone
  Control Input

Driver

Sensing Zone
  Data

Driver

Sensors

Controllers

Cache
  Shared Memory
  Resources
    Explicit Locks
      Clique
      Mutex

Distributed Algorithm

Network Communication
HOTLINE
HOTLINE

Actuators

- Influence Zone
  - Control Input
  - Driver
- Sensing Zone
  - Data
  - Driver

Sensors

Controllers

- Cache
- Shared Memory
- Hood
- Distributed Algorithm
- Resources
- Explicit Locks
- Clique
- Mutex

Network Communication
Evaluation

Personalized Light Control

- Conventional implementation
  - Token-passing, static ring overlay
- Hotline implementation
  - Programmer-assisted identification of shared resources
- **Result**: ~60% reduction in lines of code
Evaluation

Parallel Execution: Best effort

Pipeline Delays
Evaluation

Parallel Execution: Best effort

Pipeline Delays

Synchronization Delay + Response Time of PhyLock
PhyLock

Performance: Response Time (RT)

![Graph showing response time (RT) for different grid sizes and load conditions.]

- Grid sizes: grid:5x5, grid:7x7, grid:10x10
- Load conditions: Load L1, Load L4

The graph illustrates the relationship between the average size of cliques and response time for different grid sizes and load conditions.
PhyLock

Cost: Messages per Iteration

Number of Messages

Avg Size of Cliques

grid: 5x5
grid: 7x7
grid: 10x10

Load L1

Load L4
PhyLock

Impact of Packet Loss on RT

![Graph showing the impact of packet loss on RT](image)
Conclusion

• Shared memory abstraction eases distributed programming
• Use of PhyLock enables coherent access to shared variables in presence of multiple writers (controllers) and caching
• Spatial locality in actuator influence can be exploited through the Resource subsystem to enhance performance
Roadmap

• Process Synchronization and other algorithms [SECON, June 2011]
• Light control test-bed [In-progress]
• Cooperative sprinkler control
• Improve network flooding for discovery of shared variables and cliques
• Optimal parallel execution?
Questions?
Networked Optimization

Personalized Light Control

\[ \hat{I}^* = \arg \min_{\hat{I}} \sum_{i=1}^{N} f_i(\hat{I}), \text{ where } f_i(\hat{I}) = (L_i(\hat{I}) - L_i^*)^2 \]

**M** light sources

\( I_j \): Controllable Output Intensity

**N** light sensors

\[ L_i(\hat{I}) = \sum_{j=1}^{N} a_{ij} I_j + \phi_i \]

\( \phi_i \): Light from external sources at sensor \( i \)

\( L_i^* \): Desired light intensity at sensor \( i \)
Incremental Subgradient Method

\[
\hat{I}_4^{(k)} = P_X \left[ \hat{I}_3^{(k)} - \alpha g_4 \left( \hat{I}_3^{(k)} \right) \right]
\]

\[
\hat{I}_0^{(k)} = \hat{I}_4^{(k)}
\]

\[
\hat{I}_1^{(k)} = P_X \left[ \hat{I}_0^{(k)} - \alpha g_1 \left( \hat{I}_0^{(k)} \right) \right]
\]

\[
\hat{I}_2^{(k)} = P_X \left[ \hat{I}_1^{(k)} - \alpha g_2 \left( \hat{I}_1^{(k)} \right) \right]
\]

\[
\hat{I}_3^{(k)} = P_X \left[ \hat{I}_2^{(k)} - \alpha g_3 \left( \hat{I}_2^{(k)} \right) \right]
\]

\[
\hat{I}_4^{(k)} = P_X \left[ \hat{I}_3^{(k)} - \alpha g_4 \left( \hat{I}_3^{(k)} \right) \right]
\]
Shared Memory API

Commands:
\[ I_1 = \text{SharedVariable.get(CONTROL, 1)} \]
\[ \text{SharedVariable.put(CONTROL, 1, <val>)} \]

Event:
\[ \text{SharedVariable.changed(CONTROL, 1)} \]
Resource Sub-system

Implementation

\[
\begin{pmatrix}
- & - & I_3 \\
W & W & I_4
\end{pmatrix}^{(k)}
\]

\[
\begin{pmatrix}
- & I_2 & W \\
I_3 & W & W
\end{pmatrix}
\]

\[
\begin{pmatrix}
- & I_1 \\
W & W & I_4
\end{pmatrix}^{(k)}
\]

\[
\begin{pmatrix}
I_1 & W \\
I_2 & W
\end{pmatrix}
\]

\[
\begin{pmatrix}
- & I_1 \\
- & W & I_4
\end{pmatrix}^{(k)}
\]

\[
\begin{pmatrix}
- & I_2 \\
- & W
\end{pmatrix}
\]

Controllers

Distributed Algorithm
Resource Sub-system

Implementation

Controllers

Manual Identification

Resource.set(&res, config(MY_ID))

configuration--
Ctrl ID: 1
1 W
4 W

configuration--
Ctrl ID: 2
1 W
2 W

Distributed Algorithm
Resource Sub-system

Implementation

Controllers

Manual Identification

Resource.set(&res, config(MY_ID))

PhyLock.init(&res)

configuration--
Ctrl ID: 1
1 W
4 W

configuration--
Ctrl ID: 2
1 W
2 W

Distributed Algorithm
Resource Sub-system

Implementation

**Actuators**

\[
\text{pose} = \text{LSource}.\text{get}() \\
\text{zone} = \text{LSource}.\text{getZone}(\text{pose, PAN}) \\
\text{Resource}.\text{enableDiscovery}(&\text{res}, \text{zone})
\]

**Controllers**

**Dynamic Discovery**

\[
\text{pose} = \text{LSensor}.\text{get}() \\
\text{zone} = \text{LSensor}.\text{getZone}(\text{pose, PAN}) \\
\text{Resource}.\text{enableDiscovery}(&\text{res}, \text{zone}) \\
\text{PhyLock}.\text{init}(&\text{res})
\]

**Distributed Algorithm**