

The logo for GrADS, featuring the text "GrADS" in a bold, sans-serif font. The letters are white and set against a dark, grid-like background that is part of a larger grid pattern extending across the top of the slide. A thick black horizontal line is positioned below the logo.

**GrADS**

*Grid Application Development Software Project*

# GrADS Program Preparation System: Issues and Ideas

Keith Cooper, Ken Kennedy,  
John Mellor-Crummey, Linda Torczon

Center for High-Performance Software  
Rice University



# Challenges

---

- Estimating resource requirements for program execution
  - express these requirements in a form usable by all the tools
- Resource selection
  - selecting a set of available resources that meet requirements
- Mapping data and computation to resources selected
  - coping with heterogeneity (ISA, system architecture, clock rate)
  - tailoring program to both available resources and problem instance
- Performance contracts
  - developing contracts that reflect expected behavior
  - detecting contract violations
  - diagnosing contract violations
  - responding to contract violations



# Challenges

---

- Estimating resource requirements for program execution
  - express these requirements in a suitable form
- Resource selection
  - selecting a set of available resources that meet requirements
- Mapping data and computation to resources selected
  - coping with heterogeneity (ISA, system architecture, clock rate)
  - tailoring program for available resources
- Performance contracts
  - developing contracts that reflect expected behavior
  - detecting contract violations
  - diagnosing contract violations
  - responding to contract violations



# Estimating Resource Requirements

---

## What requirements?

- Data volume
  - RAM and disk
- Computation volume and characteristics
- Resource topology
  - characteristics
    - required vs. desired
    - relative importance of desired capabilities
    - directional requirements for communication
      - bandwidth constraints
      - latency constraints

# Estimating Resource Requirements

---

## Two-pronged approach

- **Automated**
  - help construct resource requirements from sample executions
- **User-directed**
  - override any misconceived notions that might be derived automatically

**Rationale: full automation is impossible for general programs**

- data-driven execution characteristics: e.g. adaptive mesh



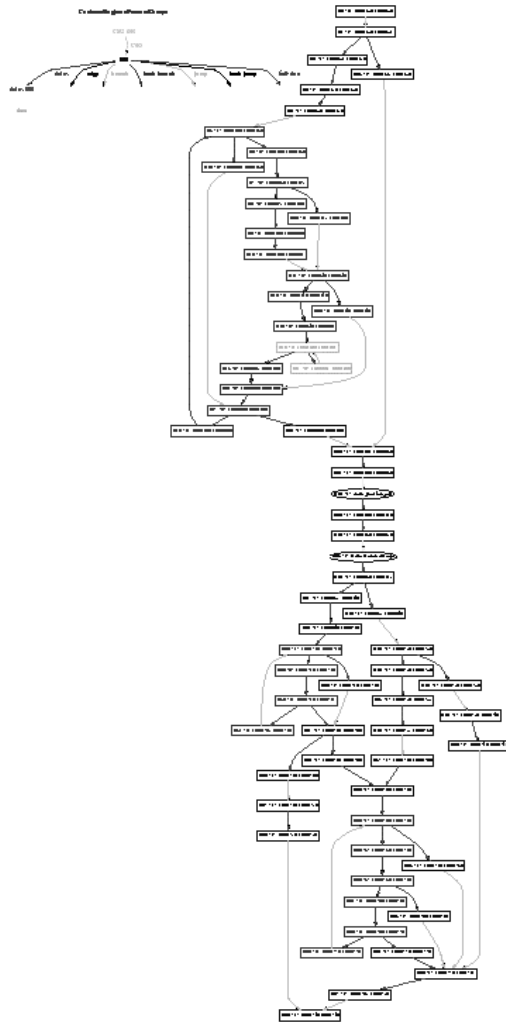
# Characterizing Application Performance

---

- Use hardware performance counters to sample events
  - FLOPS, memory accesses, cache misses, integer operations, ...
- Map events to loops: CFG from program binary + symbol table
- Measure multiple runs with different inputs
- Develop loop-nest performance model as function of inputs
  - polynomial function of measured characteristics
    - instruction balance
    - memory hierarchy: data reuse distance [stack histograms]
    - number of trips
- Compose program-level model from loop-nest models



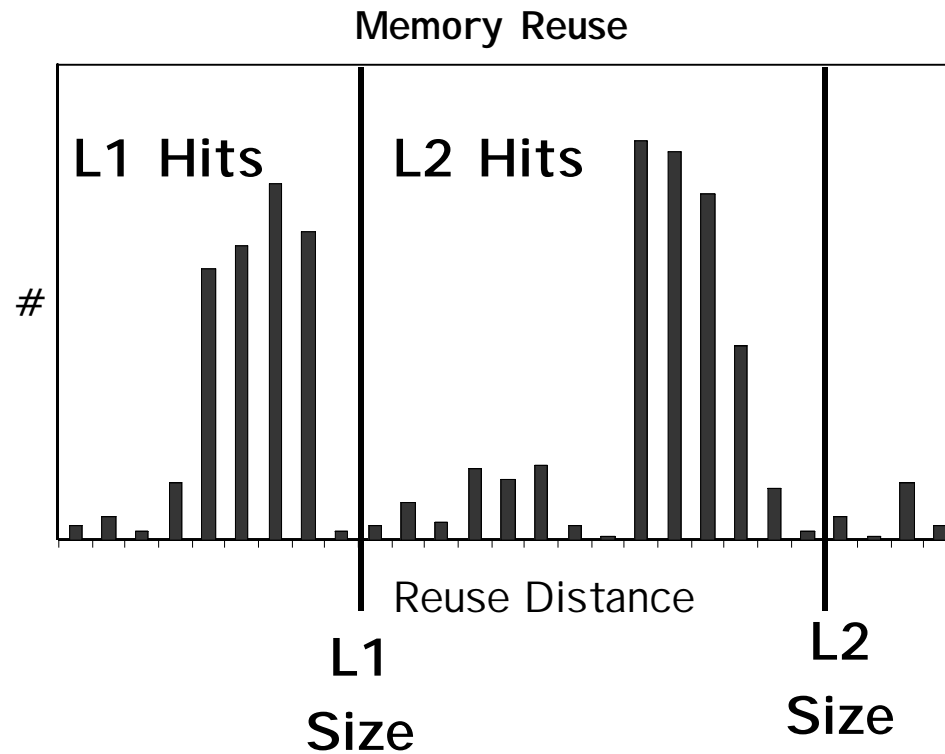
# Characterizing Application Performance



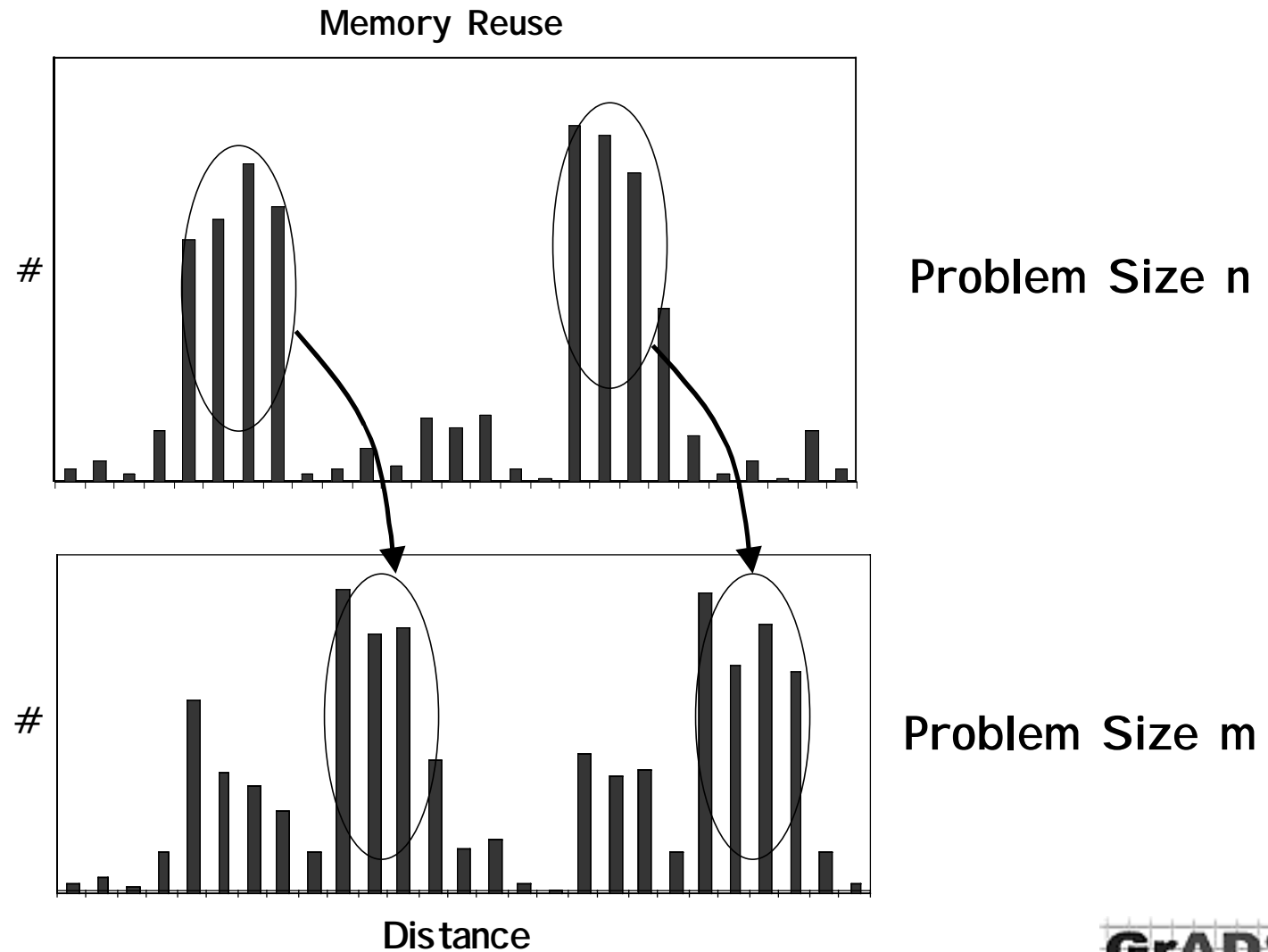
## Synthesizing models

- Recover CFG from binary
- Interpret loop nesting structure
  - blue: outermost level
  - red: loop level 1
  - green loop level 2
- Aggregate performance statistics for loop nests

# Modeling Memory Hierarchy on a Node



# Modeling Memory Hierarchy on a Node



# Predicting Application Performance

---

## Combine information to make predictions

- Parameterized program-level models (based on measurements)
- Program input parameters
- Hardware characteristics

This work builds on several ongoing projects, with funding from LACSI & Texas ATP

### Milestones (3a-b)

#### Developing model of program performance

- useful for resource selection
- establishing performance contracts



# Predicting Parallel Performance

---

## Parallel performance determined by critical path

- Critical path through a directed acyclic graph
- Actual critical path won't be known until run-time
  - depends upon communication delays & node performance

## Possible Approaches

- Automatic synthesis of DAG models of parallel performance with program slicing
  - slice out computation
  - execute sliced program with input parameters to elaborate DAG
- The “Dongarra oracle”: omnipotent library writer
- Modeling assistant
  - power steering to combine automatic & oracular models



# Mapping Data and Computation

---

## Two Alternatives

- User-provided mapping
- Initial guess + dynamic adaptation based on observed behavior

# Dynamic Optimizer

---

## Two roles

- Creating final executable program
  - mapping COP to run-time resources
  - inserting probes, actuators, & trip wires
  - final optimization
- Responding to contract violations
  - recognizing local violations & raising an alarm
  - using accumulated knowledge to improve performance
    - limited set of things that can be done
    - high cost of relocation opens a window for re-optimization



# Dynamic Optimizer

---

## Correct data mis-alignments

- Monitored behavior points to bad alignments
- Relocate data to avoid cache conflicts

## Move infrequently executed code out of the way

- Relocate unused procedures to edge of address space
- Relocate “fluff” code inside procedure to edge of address space
  - increase spatial reuse in the I-cache by exiling unused code
  - keep hot paths in contiguous memory, using fall-through branches
- Respond to increases in code page fault rate by changing program layout



# Performance Contracts

---

## Role of the program preparation system

- Develop performance models for applications
  - combination of automatic techniques & user input
- Insert the sensors, probes, & actuators to monitor performance
- Give the performance monitor better information
  - some violations might not matter
    - off-critical path computation running too slowly
  - some violations might be unavoidable
    - node running flat out and still falling behind model
    - model may have mispredicted the code's speed on that node



# Performance Contracts

---

## Many open questions about performance contracts

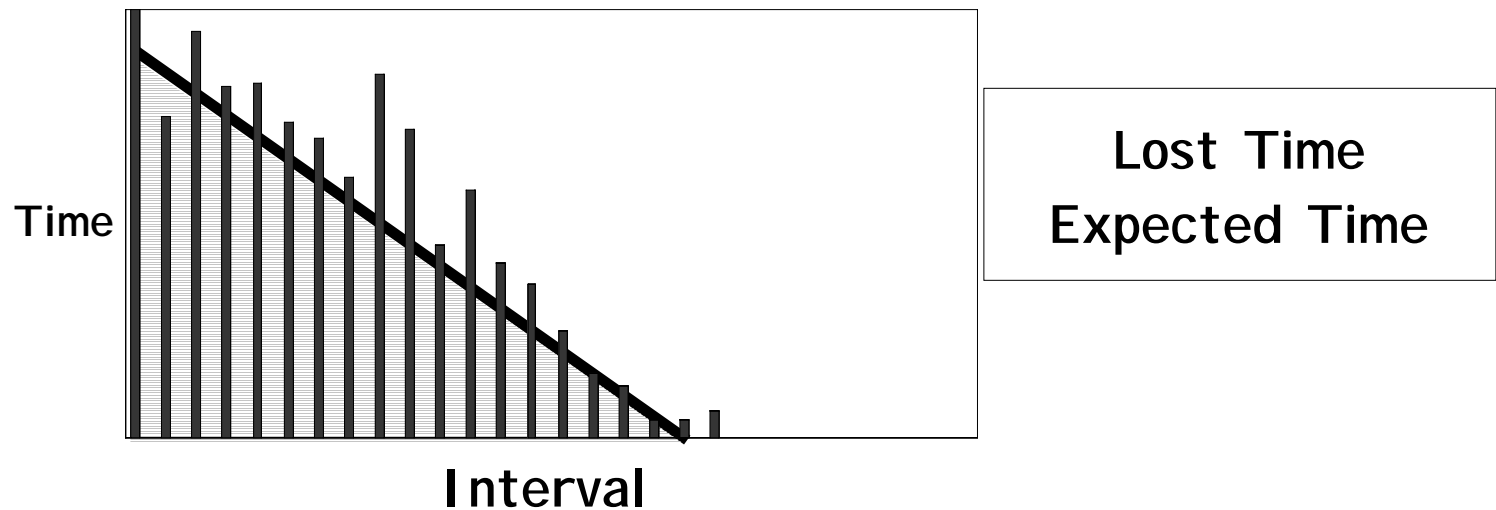
- To what do we compare actual performance?
  - user specifications?
  - dynamically-augmented models?
- How do we recognize under-performance?
  - local criteria?
  - global criteria?
  - measure differential progress
    - amount of idle time at communication points
    - cycles - FLOPS
    - measured time lost versus predicted progress
    - measured mismatch between resource availability and needs



# Performance Contracts

One strategy: Time above expectations

Measured Performance



Contract violation occurs when ratio of measured time to expected time exceeds a threshold

**GrADS**

# Ongoing Work

---

- Refinement of GrADSoft architecture *Milestone  
(4b)*
  - interface between library writers, program preparation system, and execution environment *(Abstract Application Resource Topology)*
  - performance contracts and interface to execution environment
- Performance analysis and modeling *Milestones  
(3a-b)*
  - binary analysis for reconstructing CFG and loop nesting structure
  - CFG reconstruction when delay slots contain branches
  - developing architecture independent performance models
- Evaluating compilers for GrADS compilation
  - rejected GCC for code quality
  - rejected LCC for performance of compiled code
  - considering SGI Pro64 compiler with generic back end



---

**Extra slides begin here**



# Representing Program Requirements

---

## GrADSoft Application Manager

- Input parameterization
- Abstract Application Resource Topology (AART) Model
  - dimensionality
  - structure
  - constraints
  - external representation in XML
- Performance model
- Mapper
- Performance contract

### Milestone (4b)

Interfaces that enable information sharing across Grid compilers, run-time systems and libraries

### GrADSoft Architecture

Holly Dail, Mark Mazina, Graziano Obertelli, Otto Sievert

John Mellor-Crummey



# Resource Selection

---

## Execution environment performs resource selection

- Uses information provided by
  - program preparation system
  - library writer
  - user's input parameters
  - grid conditions
    - network weather service forecasts
    - Globus/grid information systems: MDS, GRIS



# Performance Diagnosis

---

- Local problem
  - sensors and tests
  - measure progress against milestones
    - expected time vs. measured time
  - notify if outside envelope
- Global problem
  - appropriate assessment of progress via differential measures
    - wait-time vs. computation time
  - comparison against original large-scale model