Intelligent Optimization of Parallel and Distributed Applications

**USC/ISI:**
Jacqueline Chame
Chun Chen
Ewa Deelman
Yolanda Gil
**Mary Hall**
Kristina Lerman
Yoonju Lee Nelson

**USC (Physics/CS):**
Bhupesh Bansal
Aiichiro Nakano
Priya Vashishta

**OSU:**
Umit Catalyurek
Vijay Kumar
Tahsin Kurc
Joel Saltz
Ashish Sharma

This research has been partially funded by NSF NGS and CSR.
Motivation

Historical Perspective

- Highly tuned applications are too hard to develop, port
- Complexity leads to fragile applications & system software
- Ad hoc approaches, not formalized
- Community knowledge exists in the minds of too few people
- Fragmentation of community, duplication of effort

How to Move Forward

- Systematize the process of constructing and tuning applications from existing components or patterns
- Build tools that form a foundation to which many can contribute and improve
- Organize the community to work together
System Design

Workflow Design Assistance
Algorithm Selection
Optimizing Parameter Definitions
Search Constraints

Workflow Mapping
- Resource Assignment
- Identify Critical Components
- Search Engine

Component Mapping
- Code Generator
- Transformations
- Models
- Search Engine
- Learning

Experience Base

PARAMS
- (Q)
- (P)
- (R)
Key Concepts

- A systematic strategy for composing application components into workflows
- Search for the most appropriate implementation of both components and workflows
- Component optimization
  - Select among implementation variants of the same computation
  - Derive integer values of optimization parameters
  - Only search promising variants and a restricted parameter space
- Workflow optimization
  - Knowledge-rich representation of workflow properties
System Design

Existing Tools:
- **WINGS (Gil):** Workflow specification and mapping
- **Pegasus (Deelman):** Workflow scheduling and optimization for distributed platforms
- **DataCutter (Saltz, Kurc, Catalyurek):** Data management and optimization of data-intensive applications

Existing Tools (Hall):
- ECO Compiler: Model-guided empirical optimization
- Code Isolator

Other Application Tools:
- Express application-level parameters, range, models
- Learning parameter models (Lerman)
Goals and Methodology

**User's perspective:**
- Optimize the performance of workflow execution

**Systems perspective:**
- Enable runtime environment to select parameter values to optimize performance
- To learn a performance model for automation

How it works:

- **WINGS:** Semantic description for a workflow that exposes performance parameters to the runtime system.
- **Runtime system:** Hierarchical runtime environment:
  - **Pegasus/Condor:** coarse-grain scheduling of workflow and data transfers
  - **DataCutter:** fine-grained scheduling / high performance / task parallelism
Focus on Parameter Selection for Biomedical Imaging Workflow

• Combined infrastructures
  - Pegasus + Wings were already combined
  - Learned how to run DataCutter jobs from Pegasus

• Selected a Biomedical Imaging Application for experimentation
  - Already runs under DataCutter
  - First, a WINGS representation (workflow template, instance
  - Then, execution under Pegasus

• Pairwise experiments
  - Component optimization of MD simulation (see Nelson talk at HIPS!), and now warping component
Designing and Parameterizing a Workflow for Optimization

**Application: Biomedical image analysis**

![Image of biomedical equipment]

- **z-projection** → **normalization**
- **alignment & stitching** → **warping**

*Sample workflow for image correction*

Each component is a DataCutter filter, enabling processing of out-of-core image data.

- **Application-level parameters**: chunk size
- **Middleware-level parameters**: number of DataCutter filter copies per node

---

Influence of chunk size on performance of warping stage (for different warping algorithms)

[Graph showing performance over tile size]
Workflow Representation in WINGS

Extend the WINGS base ontology to represent application-specific components and data types
Workflow Representation in WINGS

Exposing application-level parameters to the runtime system

Exposing middleware-level parameters to the runtime system
Switching Gears: Component Optimization

- Combine workflow optimization with component optimization
- Two approaches:
  - Autotuning compiler
  - User-guided autotuning (semi-autotuning?)
- General strategy
  - Model-guided empirical optimization
Examples of Models and Heuristics

• Compiler models:
  – Reuse analysis to model application behavior under different loop orders, tiling strategies and unroll strategies

• Compiler heuristics:
  – Only use S-1/S sets for retained data in S-way set associative cache

• Functional models of parameters
  – Load balance as a function of computation partitioning
How well do models predict performance?

Execution on Linux cluster

![Graph showing total execution time for different cache sizes and cell sizes.](image)

Concluding Remarks

• Three core technical ideas in project
  – **Compiler technology**: Modular compilers, systematic approach to optimization, empirical search, *goal is hand-tuned performance*
  – **Components**: Tunable, automatically-generated XML-based interfaces, knowledge representations, more empirical search
  – **Systematic**: Based on machine learning, knowledge representation

• Focus on long-term evolutionary path
• ... And community organization