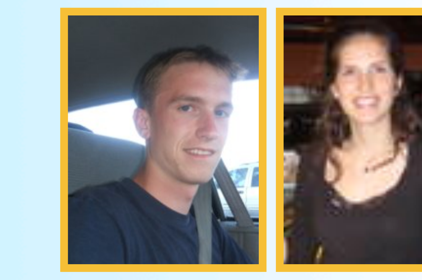
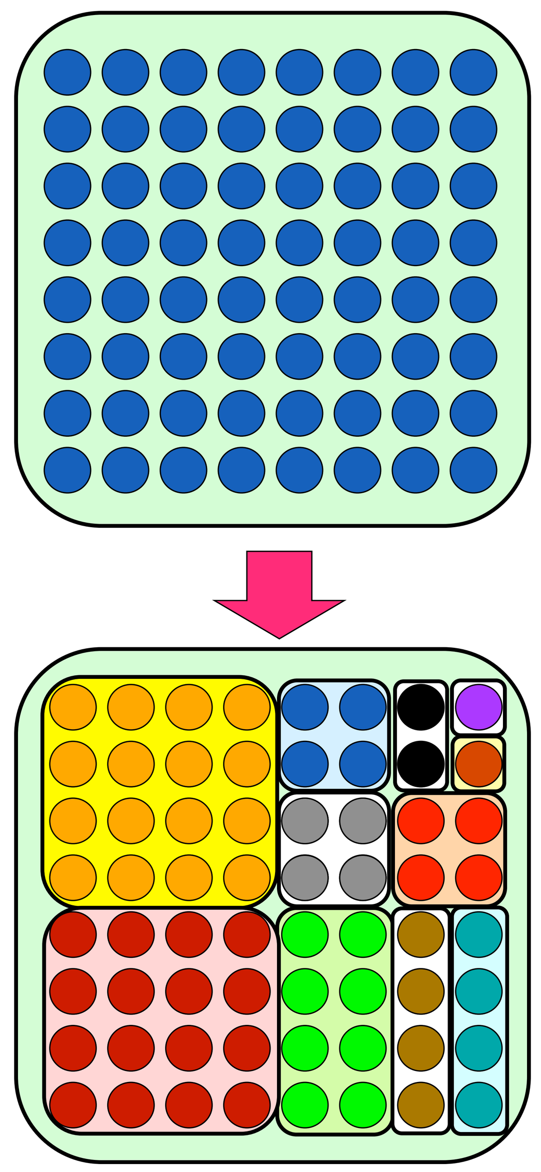
Application Modeling and Hardware Partitioning Mechanisms for Resource Management



Sarah Bird, Henry Cook, Krste Asanovic, John Kubiatowicz and Dave Patterson

RESOURCE ALLOCATION OBJECTIVES

- Each partition receives a **vector of basic resources** dedicated to it
  - Some number of processing elements (e.g., cores)
  - A portion of physical memory
  - A portion of shared cache memory
  - A fraction of memory bandwidth
- Allocate minimum resources necessary for each applications QoS requirements
- Allocate remaining resources to meet some system-level objective
  - Best performance
  - Lowest Energy
- Doesn't require application developers to worry about low-level resources



APPLICATION MODELING

- Programmers are unlikely to know exactly how low-level resources effect performance
  - Developers are concerned application-level metrics
    - e.g., frames/sec, requests/sec
  - Operating system has to make decisions about resource qualities
    - e.g., number of cores, cache slices, memory bandwidth
- Automatically constructing performance models is a good way to bridge the gap between application-level metrics and hardware resources

HARDWARE PARTITIONG MECHANISMS

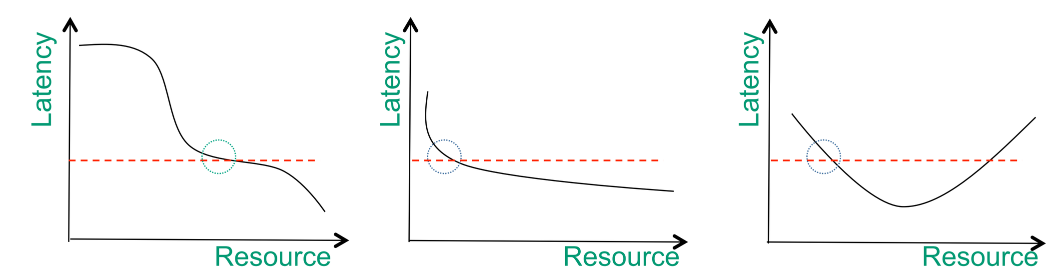
Core Partitioning:
Easily partitioned by assigning threads to cores in a partition. Application chooses which threads run on which cores.

Cache Capacity Partitioning (for shared caches):
Caches can be partitioned by ways or banks. For manycore chips we can use bank based, allowing an application can be allocated more local banks.

Bandwidth Partitioning:
Using Globally Synchronous Frames (Lee et al. ISCA 2008) we can guarantee minimum bandwidth (Packets/Frame) and bound maximum delay, while also providing differentiated services.

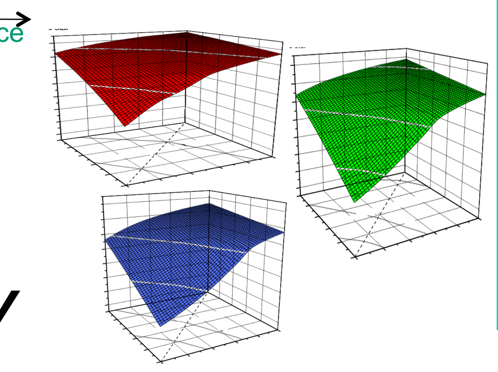
MODEL FORMULATION

- Create models from performance data sample
  - Input: performance and activity metrics
  - Output: predicted perf. for untested allocations
- Explore different model types
  - Linear, Quadratic, KCCA, GPRS
- Use models to predict the perf. of possible allocations



- Performance Funtion

$$L_i = PM_i(r_{(0,i)}, r_{(1,i)}, \ldots, r_{(n-1,i)})$$

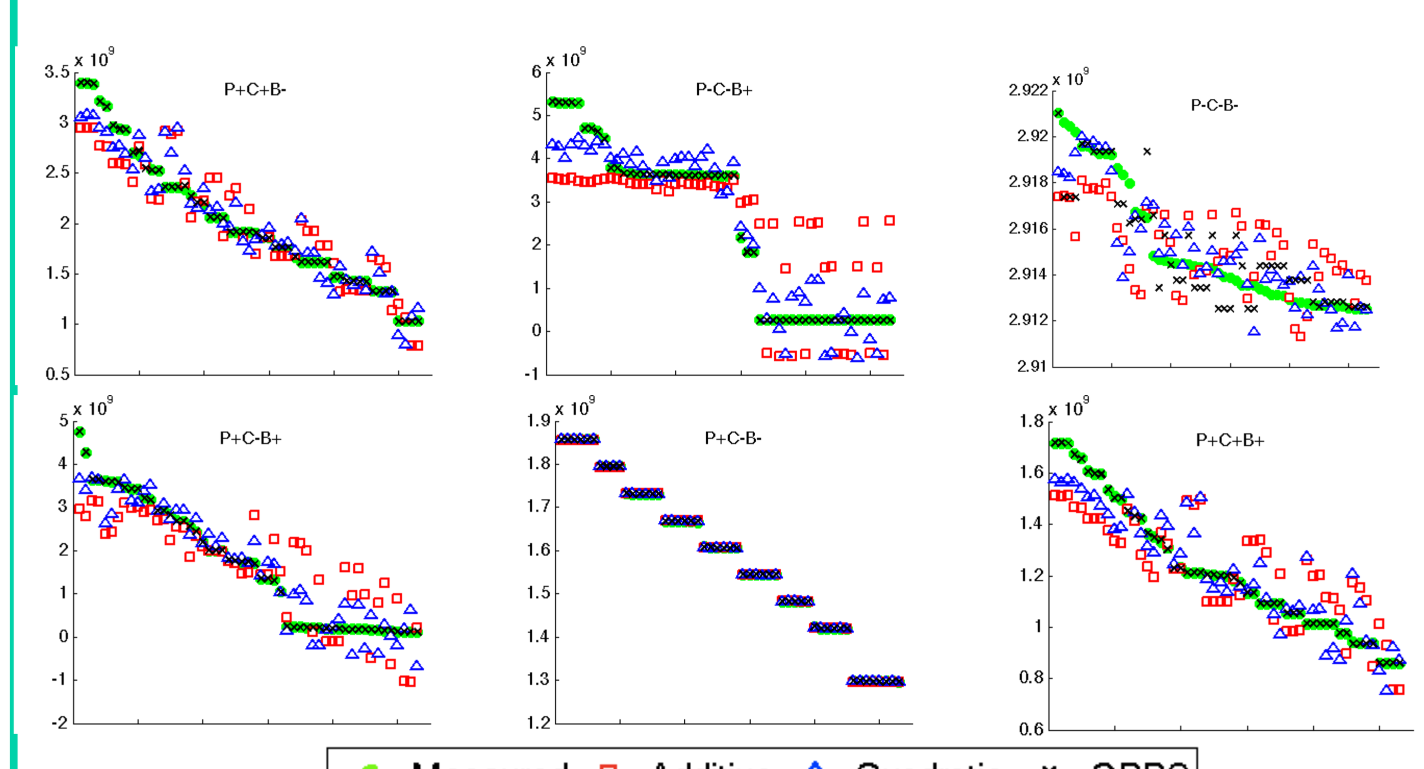


METHODOLOGY

- We use RAMP Gold with hardware partitioning
  - Using PARSEC and Synthetic Benchmarks
  - Running Tessellation (ROS)
- Collect performance data to create the models.
- Collect performance data for all possible allocations to validate models and decisions

METHODOLOGY EVALUATION

- Evaluation of model accuracy for the different model types using microbenchmarks

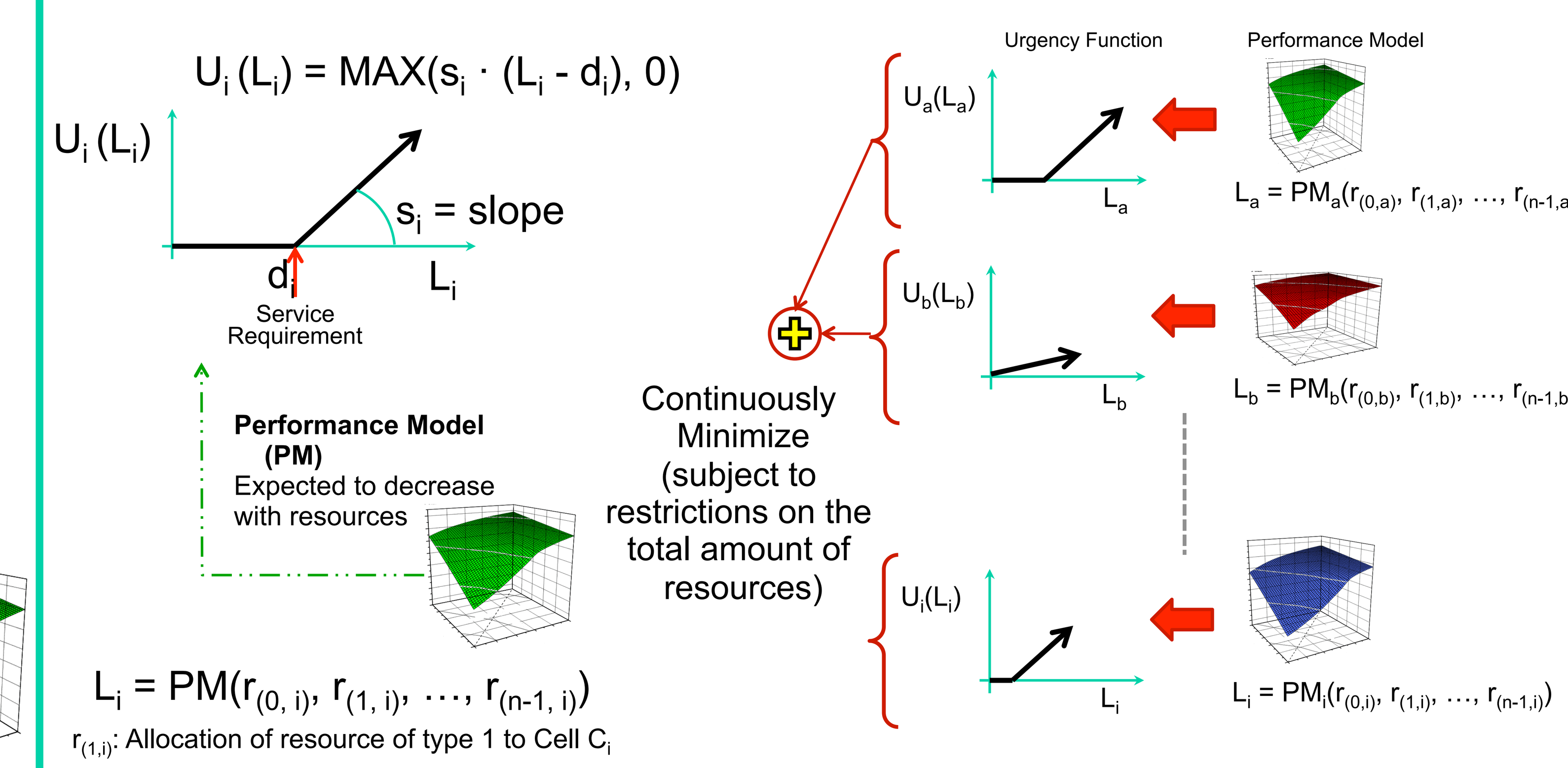


CONCLUSIONS

- Scheduling using predictive performance models shows a lot of promise.
  - Quadratic model is within 3% of optimal
  - Time-multiplexing is on average 2x of optimal
  - Dividing the machine in half is on average 1.5x of opt.
- It's important to evaluate the approach on a system with full size benchmarks and testing all the allocations

RESOURCE ALLOCATION FRAMEWORK

Minimizing the urgency of the system using convex optimization



[Burton Smith (MSR), Operating System Resource Management (Keynote), IPDPS 2010]

MAKING SCHEDULING DECISIONS

- We define an objective function that uses the predictive models of the two applications.
- Experiment with different objective functions to represent best system performance, and lowest energy.
  - Minimize the sum total of cycles on the machine
  - Minimize the time to completion for the set of benchmarks
  - Minimize energy based on a simple energy model
- We can give weights to the model outputs and other features.

- We use the active-set algorithm for nonlinear constrained optimization (fmincon in Matlab) to solve the objective function.

DECISION-MAKING RESULTS

- We run all possible allocations for the two benchmarks executing together.
- Compare with simple baselines
  - Best Spatial Partition
  - Time-Multiplexing each application on the whole machine
  - Dividing the Machine in Half Spatially

This graph shows how effective our models are a picking an allocation vs. the best and worst alloc. Time-mux'ing or dividing the machine in half.