Static and Dynamic Modeling for Power and Performance

KEVIN J. BARKER, RYAN D. FRIESE, DARREN J. KERBYSON, AND NATHAN R. TALLENT

HIGH PERFORMANCE COMPUTING GROUP
PACIFIC NORTHWEST NATIONAL LABORATORY

Workshop on Modeling and Simulation of Systems and Applications (ModSim 2017)

August 9-11, 2017

Talk Outline

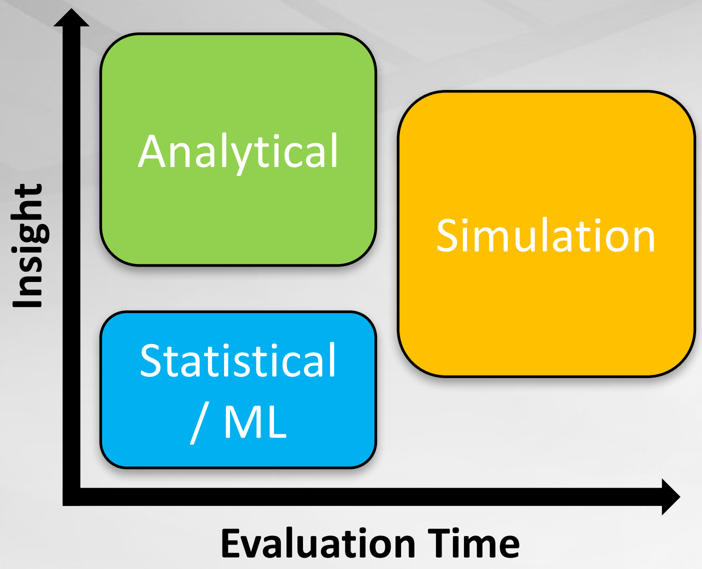
- Modeling irregular applications
  - Methodology: hierarchical critical path analysis
  - Results on challenging strong-scaling workload
- Dynamic modeling for energy optimization
  - Dynamic Power Steering: using application information to guide power distribution
  - Modeling application power consumption
  - Results on a power-constrained system

Talk Outline

- **Modeling irregular applications**
  - **Methodology: hierarchical critical path analysis**
  - **Results on challenging strong-scaling workload**

- Dynamic modeling for energy optimization
  - Dynamic Power Steering: using application information to guide power distribution
  - Modeling application power consumption
  - Results on a power-constrained system

Modeling Irregular Applications is Hard



- Good models quantitatively explain and predict execution time
  - Diagnose performance-limiting resources, design machines, etc.
    - What if the memory system was 20% faster?
  - Goal: Algebraic expressions of key parameters

- Difficult to construct models for irregular applications:
  - Non-uniform input data (e.g., a physical system)
  - Input-dependent behavior (e.g., solvers, preconditioners)
  - Irregular memory accesses (e.g., from sparse matrices or graphs)
  - Dynamic/unbiased branches (e.g., input tests, dynamic dispatch)
  - Costs widely dispersed

Application Modeling with Palm

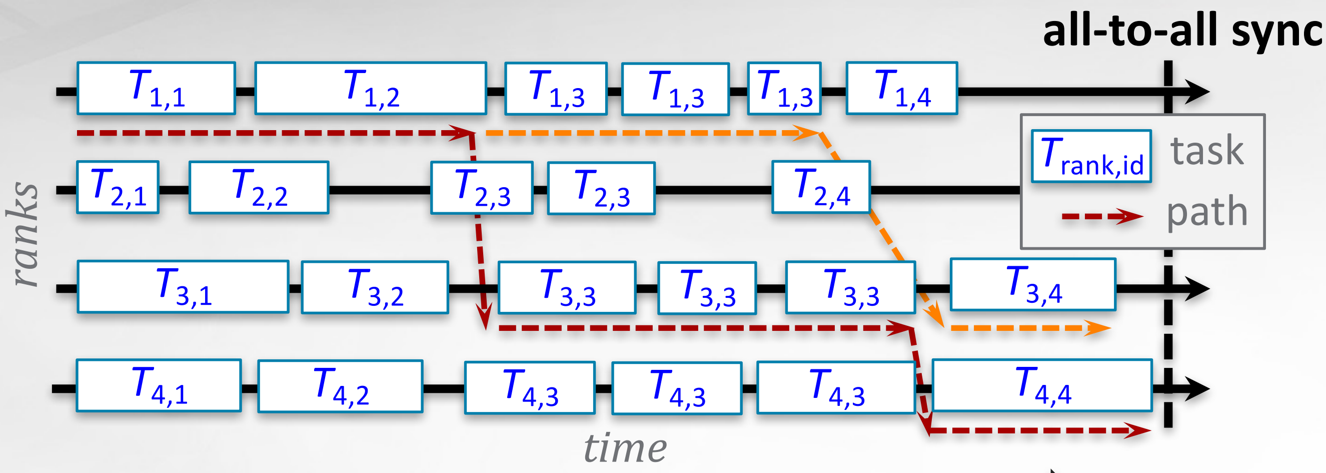
- Performance & Architecture Lab Modeling tool
- Modeling methodology based on Hierarchical Critical Path Analysis
  - Model dependency chains, not operation overlap
  - Allow time-consuming model building operations to be pushed offline
  - Make use of low-overhead tools for static and dynamic analysis

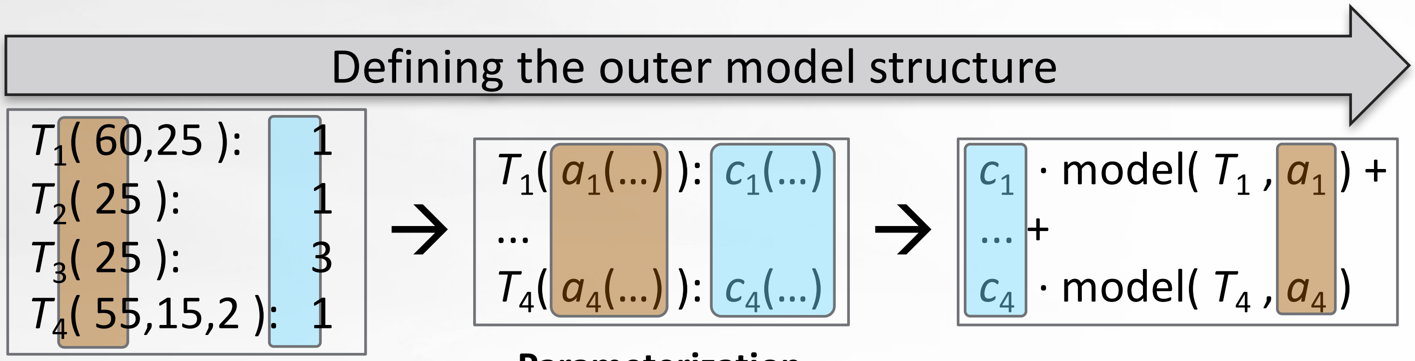
- **Goal**: to balance model generation cost, model accuracy, and generality
- Techniques employed:
  - Determine tasks along MPI critical path; parameterize instance counts and arguments
  - Model tasks via hot-path analysis capturing data flow, data locality, and microarchitectural constraints
  - Capture sub-path overlap

- Parameterized, decomposable models
- Separate computation and data access costs
- Quickly re-targetable to new microarchitecture

Step 1: MPI Critical Path Analysis

- Collect the longest sequence of *tasks* for representative execution
- Simplification: No overlap among tasks on critical path





Defining the outer model structure

$T_1(\ 60,25\ )$: 1
$T_2(\ 25\ )$: 1
$T_3(\ 25\ )$: 3
$T_4(\ 55,15,2\ )$: 1

→

$T_1(\ a_1(\ldots)\ )$: $c_1(\ldots)$
…
$T_4(\ a_4(\ldots)\ )$: $c_4(\ldots)$

→

$c_1 \cdot \text{model}(\ T_1\ ,\ a_1\ ) + \ldots + c_4 \cdot \text{model}(\ T_4\ ,\ a_4\ )$

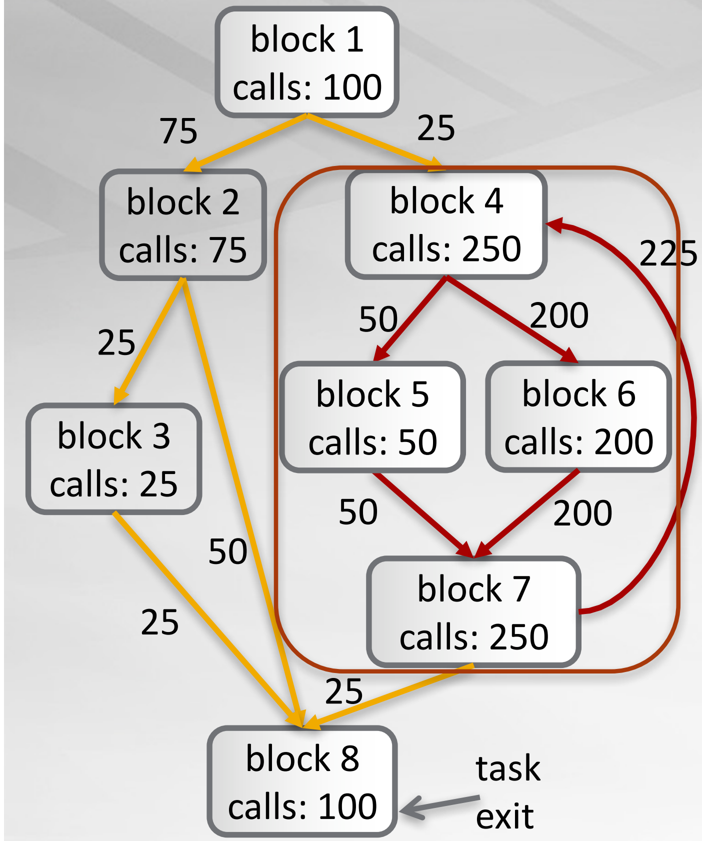
Direct Observation of dynamic execution

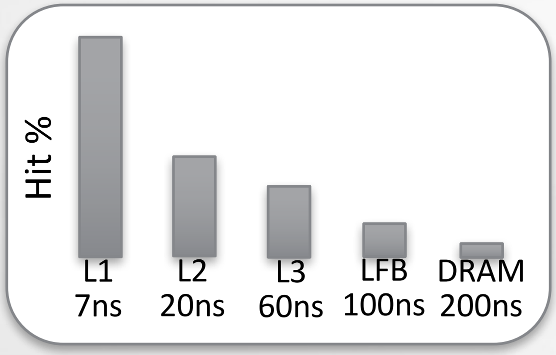
Parameterization (requires human guidance)

Outer model structure

- Key: relating task arguments and instance counts to input model parameters.
- **Human guidance can help**

Step 2: Critical Path Analysis of Hot Paths

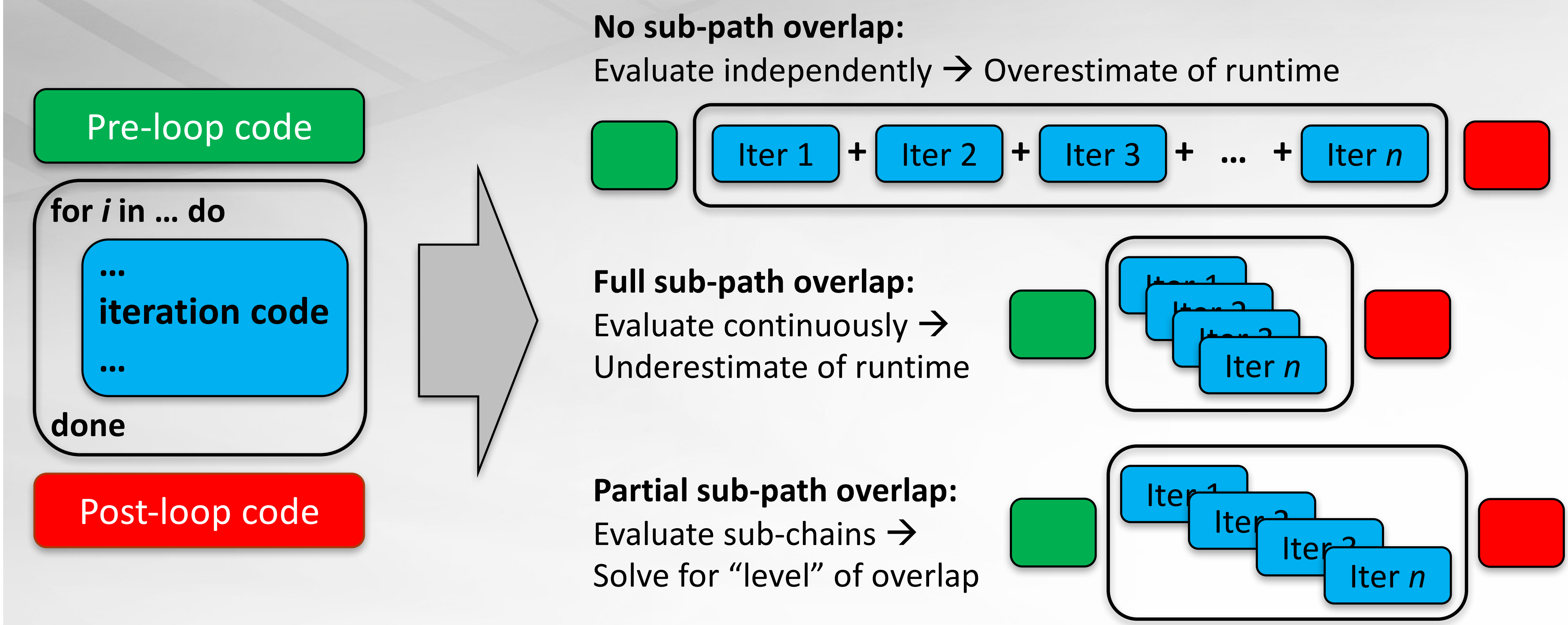




- Can determine probability of branch paths taken using dynamic instrumentation (e.g., DynInst)
- For straight-line paths, architecture-specific analysis tool (e.g., IACA) can determine instruction cost of path
  - Separate compute from data access costs
- Loops can be decomposed into straight-line paths
  - Key question: what is the level of overlap between loop iterations?
  - Analysis becomes trickier for loop nests

- Incorporate data access latencies per block
  - Histogram of hits per level in the memory hierarchy (obtained through tools such as *perf-mem*)
  - Number of memory operations along the critical path

Modeling Overlap of Sub-Path Critical Paths



Pre-loop code

for *i* in … do

…

iteration code

…

done

Post-loop code

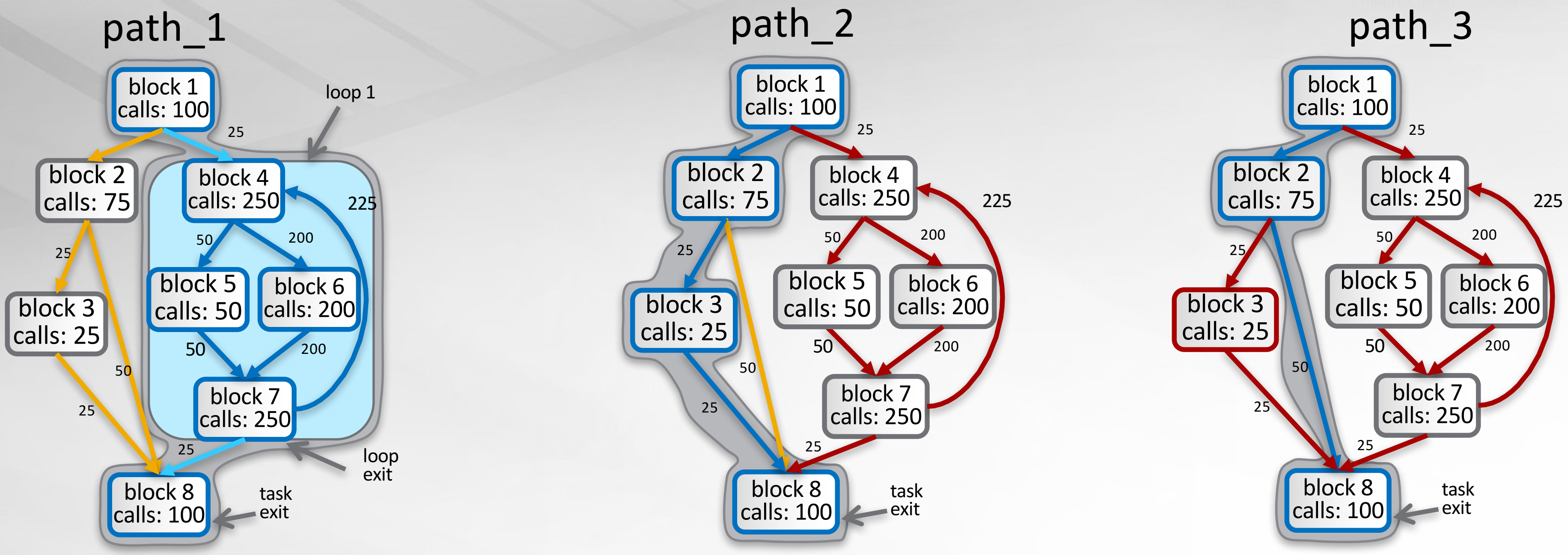
No sub-path overlap:
Evaluate independently → Overestimate of runtime

Full sub-path overlap:
Evaluate continuously →
Underestimate of runtime

Partial sub-path overlap:
Evaluate sub-chains →
Solve for “level” of overlap

- Goal: capture dynamic CPU pipelining
- Sub-path overlap:
  - Tunable parameter allows for “what-if” evaluations (e.g., extra functional unit)
  - Applied to innermost loops with highly biased branches
  - Provides min and max bounds for loop execution time

Task Modeling: An Example Model

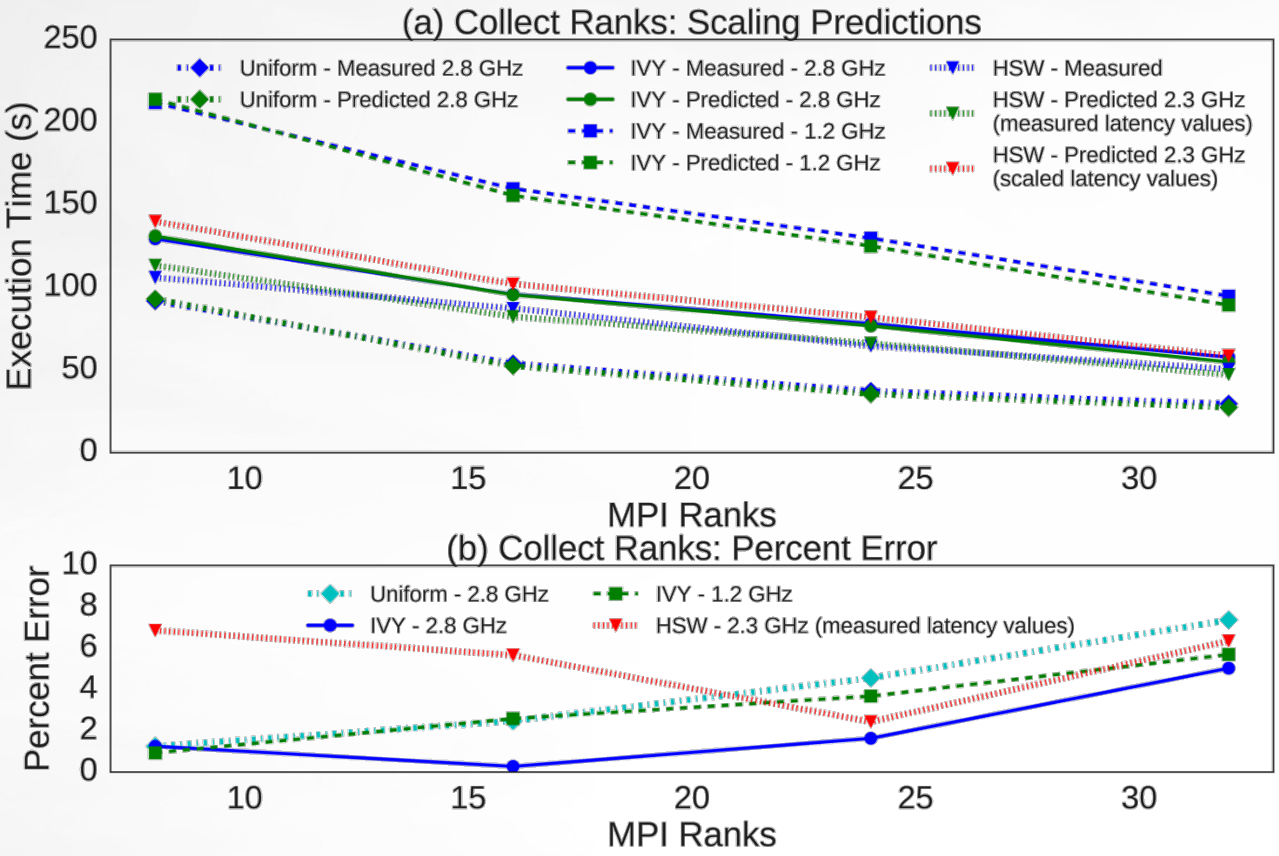


```
def task-$T_1$ (base_data_cost, loop_1_data_cost, loop_1_iterations : 10)
   metablk_1 = [4, 6, 7,  4, 6, 7,  4, 6, 7] // effective dynamic
   metablk_2 = [4, 5, 7,  4, 5, 7,  4, 5, 7] // overlap factor of 3
   loop_1_path_1 = 0.8*(cp(metablk_1).lat + cp(metablk_1).mem*loop_1_data_cost)
   loop_1_path_2 = 0.2*(cp(metablk_2).lat + cp(metablk_2).mem*loop_1_data_cost)
   loop_1 = loop_1_iterations/3.0*(loop_1_path_1 + loop_1_path_2)
   path_1 = 0.25*((cp(1).lat+cp(1).mem*base_data_cost)+loop_1+(cp(8).lat + cp(8).mem*base_data_cost)
   path_2 = 0.25*(cp([1,2,3,8]).lat+cp([1,2,3,8]).mem*base_data_cost)
   path_3 = 0.5*(cp([1,2,8]).lat+cp([1,2,8]).mem*base_data_cost)
   return path_1 + path_2 + path_3
end
```

Modeling PageRank’s Key Tasks: A Strong-Scaling Workload

- MPI Implementation of PageRank
- Two input graphs: Power-law (11M vertices, 1.3B edges) and Uniform
- Two CPU micro-architectures and two clock frequencies

- Challenges:
  - Comm. aggregation
  - Inlined code: C++ map insert, lookup, iterate
  - Unbiased branches and indirect data access
- Changing architectures
  - Regenerate data access parameter values
- Changing input graphs
  - Regenerate data access parameter values
  - Adjust task parameters



Talk Outline

- Modeling irregular applications
  - Methodology: hierarchical critical path analysis
  - Results on challenging strong-scaling workload

- **Dynamic modeling for energy optimization**
  - **Dynamic Power Steering: using application information to guide power distribution**
  - **Modeling application power consumption**
  - **Results on a power-constrained system**

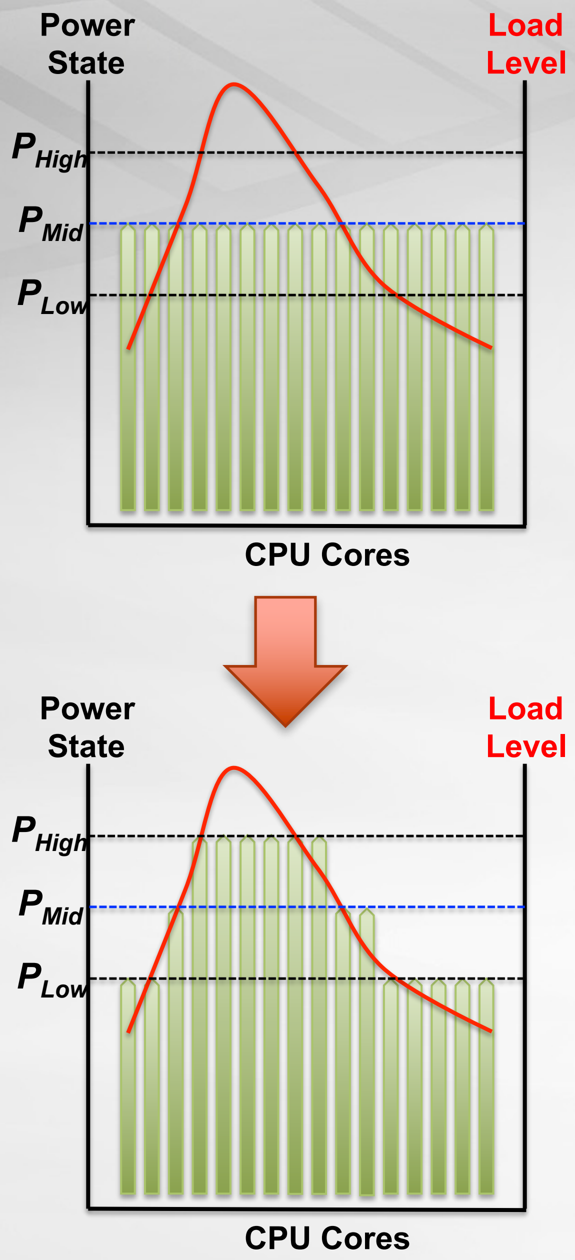
Dynamic Power Steering

Idea: Route power to those resources that are over-loaded and away from under-loaded resources to compensate

- Optimizes power consumption in two ways:
  - Leaves data in place – minimizes power lost to data migration
  - Routing available power to where the work is – *Power Balancing*
- Targeting workloads
  - In which static calculation of ideal power distribution is not possible (e.g., data-dependent execution, variation over time)
  - In which performance is impacted by changes to node or core *p*-state (i.e., by allocated more power, performance may be improved)

Questions: Can we *predict* the impact Dynamic Power Steering will have on application performance and energy consumption and can we use this knowledge to drive decision-making in runtime software?

Routing Power to Overloaded Resources



- Individual core load varies over the course of the simulation, leading to load imbalance
- Systems equipped with multiple *p-states* can modify performance and power consumption dynamically
- Utilizing high-performance settings on all cores will exceed global power budget
- Limited power needs to be *intelligently* routed to where it can be most beneficial

Emulating a Power-Constrained System

Freq. (GHz)	Active Pwr (W)	P-state Label
2.1	21.1	P_{Hi}
1.7	**18.0**	$\boldsymbol{P_{Mid}}$
1.4	15.6	P_{Low}

- We emulate a power-capped system by initially imposing a mid-range *p*-state for each processor core
  - Allow for core *p*-state to vary up or down using Heuristic
  - Overall power is constrained to be that of initial operating point
  - Improve performance along critical path compared to operating point thereby potentially reducing energy-to-solution
- Test-bed platform:
  - 36 nodes of dual-socket, 8-core AMD Interlagos processors
  - Power measurement capability at outlet level @ 0.3Hz sampling rate

Analytically Modeling Power Steering

- Overall system power budget is defined by the constraint

$$P_{Constrained} = CP_{Baseline} \ge \sum_{i=1}^{N_{P-states}} C_i P_i \text{ , where } C = \sum_{i=1}^{N_{P-states}} C_i$$

- Total time is given by the longest executing core over all power states

$$T_{Total} = \max_{i \in \{P-states\}} \left[\max_{C \in \{C_i\}} T_C(i, W_C) \right]$$

- Moving some cores to a higher power state must be balanced by moving some cores to a lower state:

$$\left(C_{Low} + C_{High}\right) P_{Mid} \ge C_{Low} P_{Low} + C_{High} P_{High}$$

- The number of cores that must be moved to the lower power state is defined to be:

$$C_{Low} \ge C_{High}\left(r_{High} - 1\right) / \left(1 - r_{Low}\right)$$

$$\text{where } r_{Low} = P_{Low} / P_{Mid} \le 1 \text{ and } r_{High} = P_{High} / P_{Mid} \ge 1$$

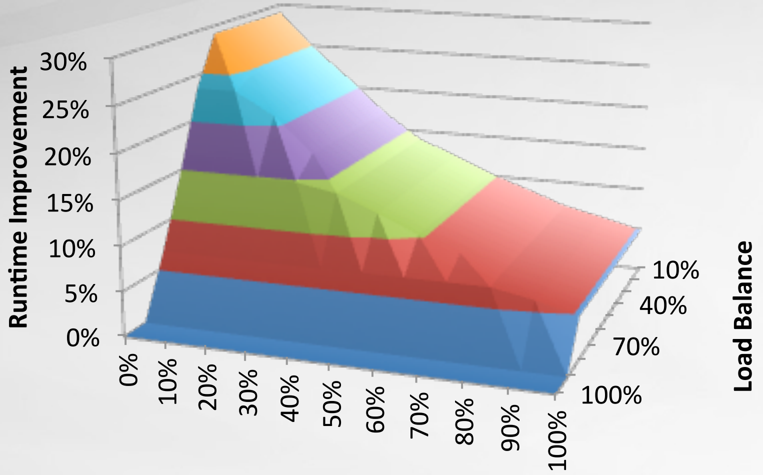
Resulting Power Assignment Heuristic

```
Start
    1. $PWR_{max}$ = maximum globally available power
    2. $p\text{-}state_{max}$ = highest performance $p\text{-}state$
    3. $N_{work\_max} = \max(N_{work\_i})\ \forall\, i \in \{ P_i \}$
    4. $t_{work\_max} = N_{work\_max} \times t_{work}(p\text{-}state_{max})$
    5. $\forall\, i \in \{P_i \mid P_i <> P_{work\_max}\}$ find slowest p-state such that $t_{work\_i} < t_{work\_max}$
    6. $PWR_i = t_{work\_i}(p\text{-}state_i)$
    7. $PWR_{global} = SUM\ (PWR(p\text{-}state_i))$
    8. If $PWR_{global} > PWR_{max}$ then reduce $p\text{-}state_{max}$ and go to step 3
    9. Assign p-state calculated to each processor-core
End
```

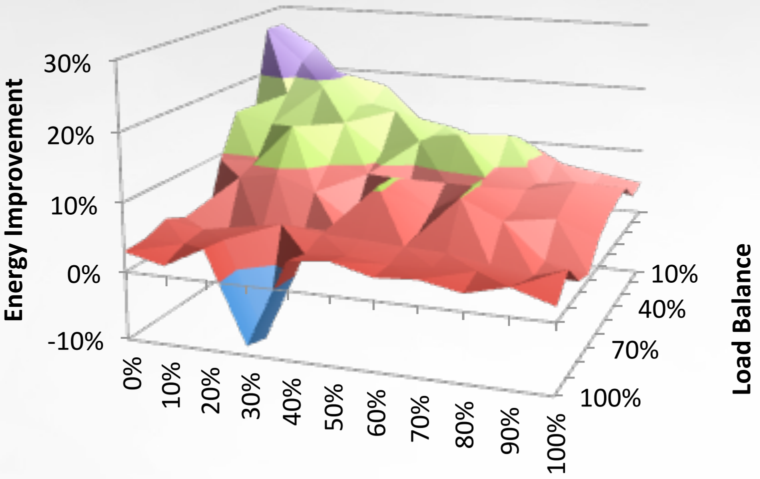
- Core along the critical path (i.e., most overloaded) is put in the highest performing p-state
- P-states for other processors is calculated to be the lowest that does not negatively impact performance
- If the global power budget is exceeded, the p-state of the most loaded processor is reduced and the heuristic is repeated

Modeled Performance Improvement Leads to Measured Energy Improvement

- Charged-field workload emulates particles migrating in a magnetic field



Modeled Performance Improvement



Measured Energy Efficiency Improvement

- Fixed power budget means performance improvements translate to energy
- (Quasi-)Analytical models are able to quantify and predict the amount of performance and energy efficiency improvement that can be expected given:
  - Degree of load imbalance across the parallel system
  - Impact of change in p-state on workload performance

Conclusions

- Over the past few years
  - Analytical modeling methods have moved beyond static, regular applications
  - Analytical models have moved beyond modeling only performance

- Looking forward
  - New programming and execution models
  - Extreme heterogeneity
  - Introspective runtime software systems
  - Wide area distribution and data movement/placement

Selected Publications

- R. D. Friese, N. R. Tallent, A. Vishnu, D. J. Kerbyson, and A. Hoisie, “Generating Performance Models for Irregular Applications”, *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2017.
- N. R. Tallent, K. J. Barker, R. Gioiosa, A. Marquez, G. Kestor, S. Song, D. J. Kerbyson, and A. Hoisie, “Assessing Advanced Technology in CENATE”, *IEEE International Conference on Networking, Architecture, and Storage (NAS)*, 2016.
- K. J. Barker and D. J. Kerybson, “Modeling the Performance and Energy Empact of Dynamic Power Steering”, *IEEE International Parallel and Distributed Processing Symposium (IPDPS) Workshops: Workshop on Large-scale Parallel Processing*, 2016.
- N. R. Tallent and A. Hoisie, “Palm: Easing the Burden of Analytical Performance Modeling”, *28th International Conference on Supercomputing (ICS)*, 2014.
- K. J. Barker, D. J. Kerbyson, and E. A. Anger, “On the Feasibility of Dynamic Power Steering”, *2nd International Workshop on Energy-efficient Supercomputing (E2SC)*, 2014.
- D. J. Kerbyson, K. J. Barker, A. Vishnu, and A. Hoisie, “A Performance Comparison of Current HPC Systems: BlueGene/Q, Cray XE6, and InfiniBand Systems”, *Future Generation Computing Systems (30)*, 2014
- S. Song, N. R. Tallent, and A. Vishnu, “Exploring Machine Learning Techniques for Dynamic Modeling on Future Exascale Systems”, *Workshop on Modeling and Simulation of Exascale Systems and Applications*, 2013.
- S. Song, K. J. Barker, and D. J. Kerbyson, “Unified Performance and Power Modeling of Scientific Workloads”, *1st International Workshop on Energy-efficient Supercomputing (E2SC)*, 2013.
- D. J. Kerbyson, K. J. Barker, D. S. Gallo, D. Chen, J. R. Brunheroto, K. D. Ryu, G. L. Chiu, and A. Hoisie, “Tracking the Performance Evolution of BlueGene Systems”, *28th International Supercomputing Conference (ISC)*, 2013.