Resource Management for Heterogeneous Computing Systems: Utility Maximization, Energy-Aware Scheduling, and Multi-Objective Optimization

Ryan Friese

Ph.D. Final Defense Examination

Outline:

- introduction
- dynamic utility maximization
- energy-aware utility maximization
  - energy-constrained
  - different queuing environments
- multi-objective optimization
  - deterministic utility vs. energy
  - deterministic makespan vs. energy
  - stochastic utility/makespan vs. energy
- application co-location in multicore systems
- future work

Heterogeneous Computing System

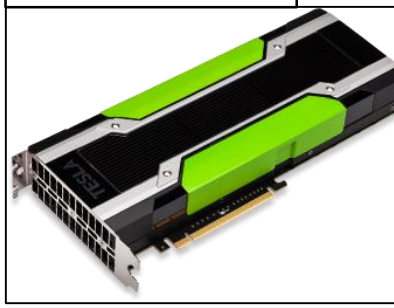
- interconnected machines with varied computational capabilities
- workload of tasks with different computational requirements
  - heterogeneity to service diverse computational workloads
- each task may perform differently on each machine
  - machine A better than machine B for task 1 but not for task 2
- research also applies to a cluster of different types (or different ages) of machines, grids, and clouds

Intel Phi Coprocessor

Cray XC-30 Blades

HP BladeSystem C7000

Nvidia Tesla GPU



Hitachi Blade Server 500

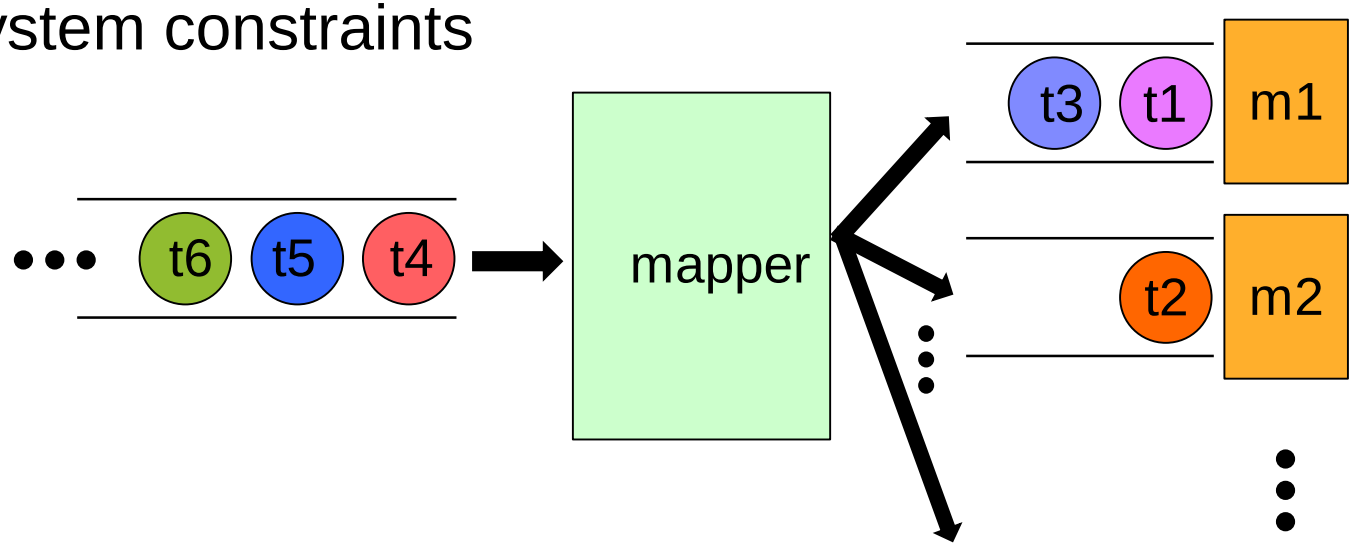
Colorado State University

Resource Management

- assign and schedule (map) tasks to machines
  - optimize some performance measure
  - possibly under a system constraint
- in general, known **NP-Hard** problem
  - cannot find optimal solution in reasonable time
  - ex.: 5 machines and 30 tasks
    → 5^{30} possible assignments
    - if it only took 1 nanosecond to evaluate each assignment
      - 5^{30} nanoseconds > 20,000 years!
  - use **heuristics** to find near-optimal solutions

Mapping Tasks to Machines in an HC System

- map tasks to machines considering
  - quality of match (computational requirements to machine capabilities – exploit heterogeneity)
  - concurrent use of multiple machines when appropriate
  - estimated machine available time
  - P-state choices in each machine
  - task characteristics (utility, priority, deadline, …)
  - system performance metric
  - system constraints



Task Types and Machine Types

- task types – similar execution characteristics
- machine types – similar performance capabilities
- **Estimated Time to Compute (ETC)** matrix gives execution time of a given task-type / machine-type / P-state
- **Average Power Consumption (APC)** matrix gives the dynamic power consumed for a given task-type / machine-type / P-state

ETC values (seconds)

P0	M1	M2
T1	10	8
T2	12	9
T3	7	11

P0	M1	M2
T1	115	95
T2	87	105
T3	125	90

APC values (watts)

- ETC and APC based on historical or experimental data

Calculating Energy Consumption

- ETC – execution time of task-type / machine-type / P-state
- APC – dynamic power for task-type / machine-type / P-state

ETC values (seconds)

P0	M1	M2
T1	10	8
T2	12	9
T3	7	11

APC values (watts)

P0	M1	M2
T1	115	95
T2	87	105
T3	125	90

dynamic energy of T3 on M2 = 11 seconds x 90 watts = 990 joules

- only consider the dynamic energy of the system
  - machines always “on” in oversubscribed environment

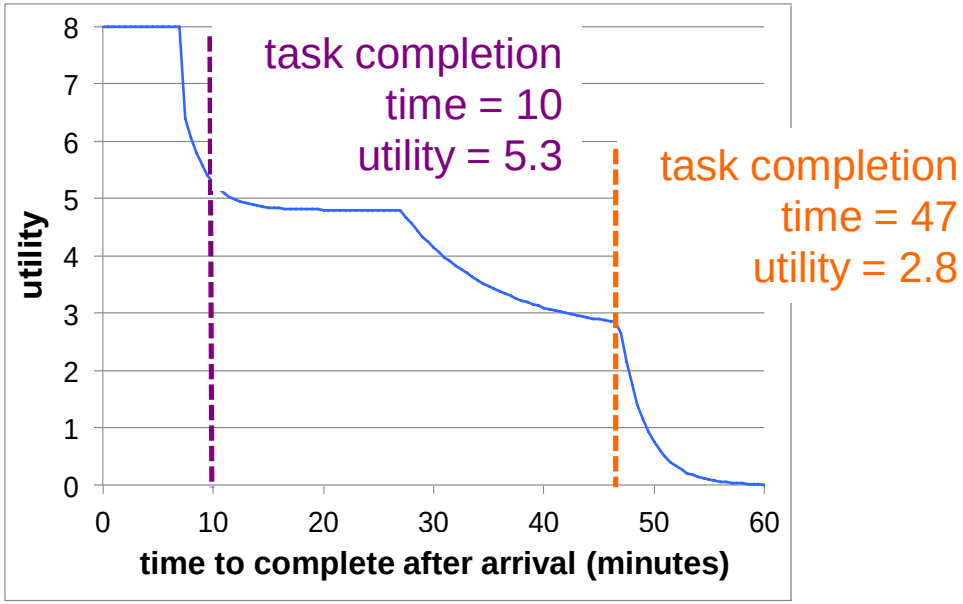
Outline

- Chapter 1: introduction
- Chapter 2: dynamic utility maximization
- dynamic energy-aware resource allocation
  - Chapter 3: utility maximization with an energy constraint
  - Chapter 4: energy constrained utility maximization for different queuing models
- multi-objective optimization
  - Chapter 5: deterministic utility vs. energy
  - Chapter 6: deterministic makespan vs. energy
  - Chapter 7: stochastic utility/makespan vs. energy
- Chapter 8: application co-location in multicore systems
- Chapter 9: future work

B. Khemka, R. Friese, et al., “Utility functions and resource management in an oversubscribed heterogeneous computing environment,” *IEEE Transactions on Computers*, 2014, accepted for publication.

Task's Utility Function

- basis for the system performance measure in this study
  - based on the needs of DoD/DOE Extreme Scale Systems Center at ORNL
- represents the time varying worth of completing the task
  - designed by user in collaboration with system administrator
- **utility earned:** value of completing a task at any given time
  - a measure of the amount of useful work accomplished



Colorado State University

Assumed Environment

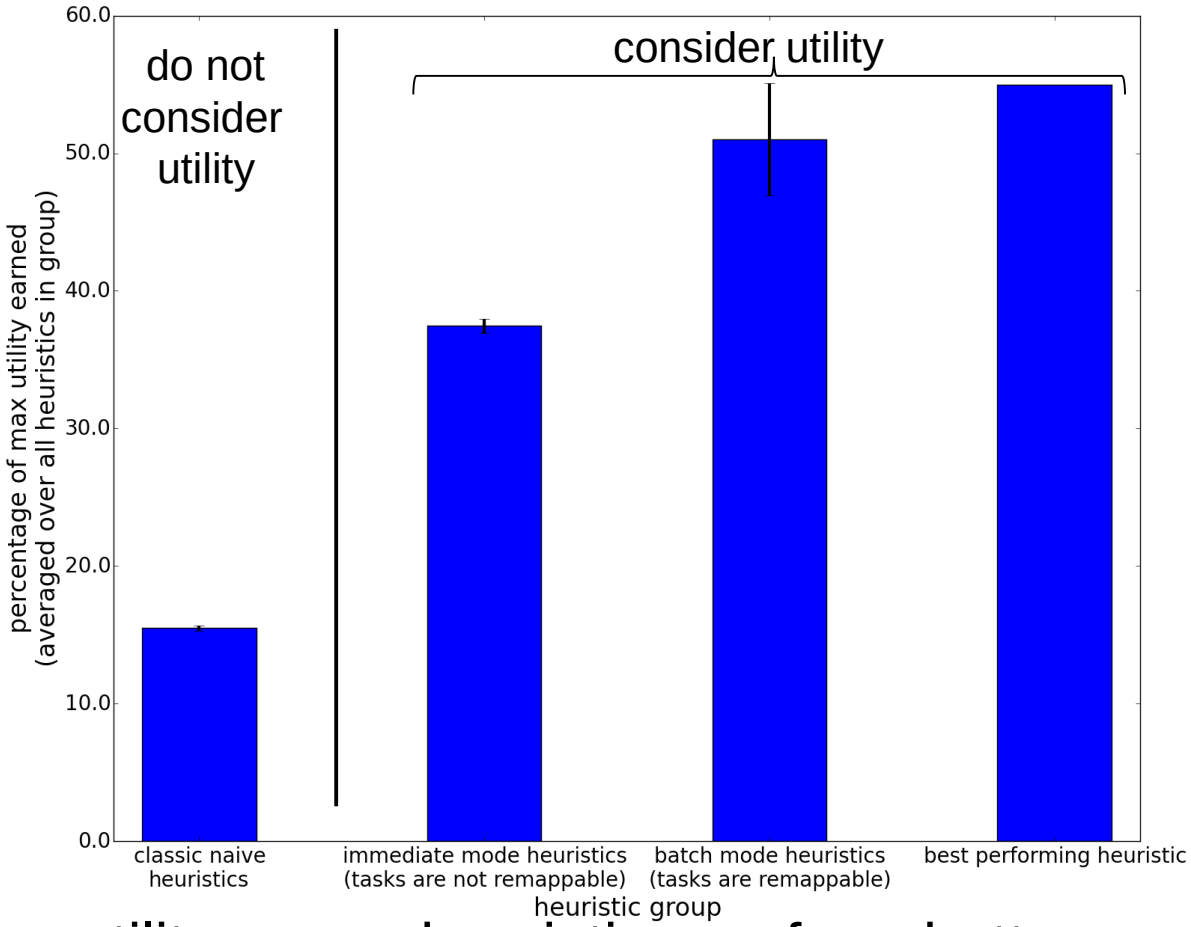
- **oversubscribed** environment
  - not all tasks can completed when desired
- tasks are assumed to be **serial**
  - execute on a single machine
- **no preemption** of tasks
  - once a task starts, it executes to completion

Contributions

- model to create utility functions using the three parameters: priority, urgency, utility class
- model of the given DOE/DoD heterogeneous computing system and its intended workload
  - special-purpose machine that execute certain special-purpose tasks faster
  - sinusoidal and bursty arrival patterns for general and special-purpose tasks respectively
- design of a performance metric for schedulers in oversubscribed heterogeneous computing systems
- design and analysis of seven immediate-mode heuristics and five batch-mode heuristics to perform scheduling
- design and exploration of other mapping operations (including the dropping of low utility earning tasks)

Results of Different Heuristics Groups

100 task types & ~50,000 tasks, 13 machine types & 100 machines



- classic naïve heuristics – representative of many real-world systems
- immediate mode heuristics – tasks can only execute on assigned machine
- batch mode heuristics – tasks are allowed to be remapped
- all results used dropping

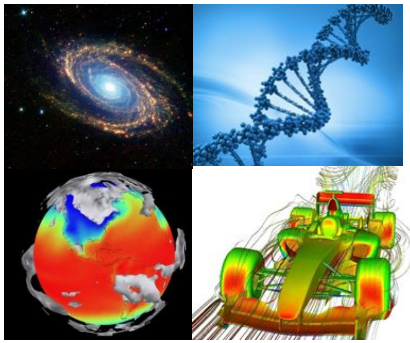
- utility-aware heuristics perform better than classical techniques in all cases
- batch mode heuristics perform best because they allow high utility earning tasks to be executed more quickly

Colorado State University

Outline

- Chapter 1: introduction
- Chapter 2: dynamic utility maximization
- dynamic energy-aware resource allocation
  - Chapter 3: utility maximization with an energy constraint
  - Chapter 4: energy constrained utility maximization for different queuing models
- multi-objective optimization
  - Chapter 5: deterministic utility vs. energy
  - Chapter 6: deterministic makespan vs. energy
  - Chapter 7: stochastic utility/makespan vs. energy
- Chapter 8: application co-location in multicore systems
- Chapter 9: future work

Need for Energy-Awareness




higher application complexity

faster performance

greater energy requirements

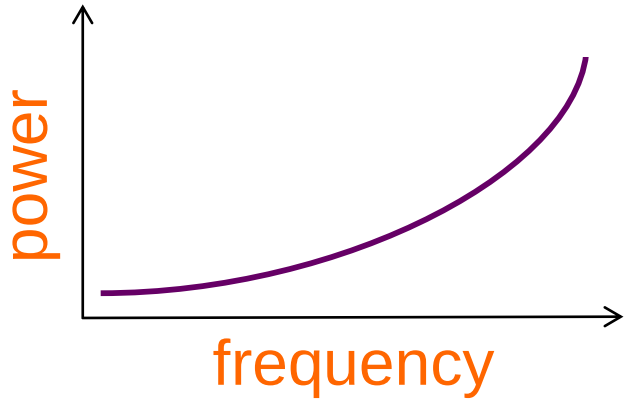
billions in electricity costs

- August 2013 report estimated global Information-Communications-Technologies ecosystem used 10% of the world electricity generation
- electricity costs can limit size of computing system
- institutions need to maximize the performance of their high performance computing systems within an energy budget
- manage compute resources in an energy-efficient manner

Colorado State University

P-states – Performance States

- machines use Dynamic Voltage and Frequency Scaling (DVFS)
- three P-states (performance states) implement DVFS
  - **P0** highest power to **P2** lowest power
  - higher power consumption → faster execution
  - energy = power × execution time
  - typically: lower power P-state → less energy but more time
    - depends on ratio of overhead (static) energy to processor execution (dynamic) energy
    - impacted by memory-intensity of task



Assumed Environment

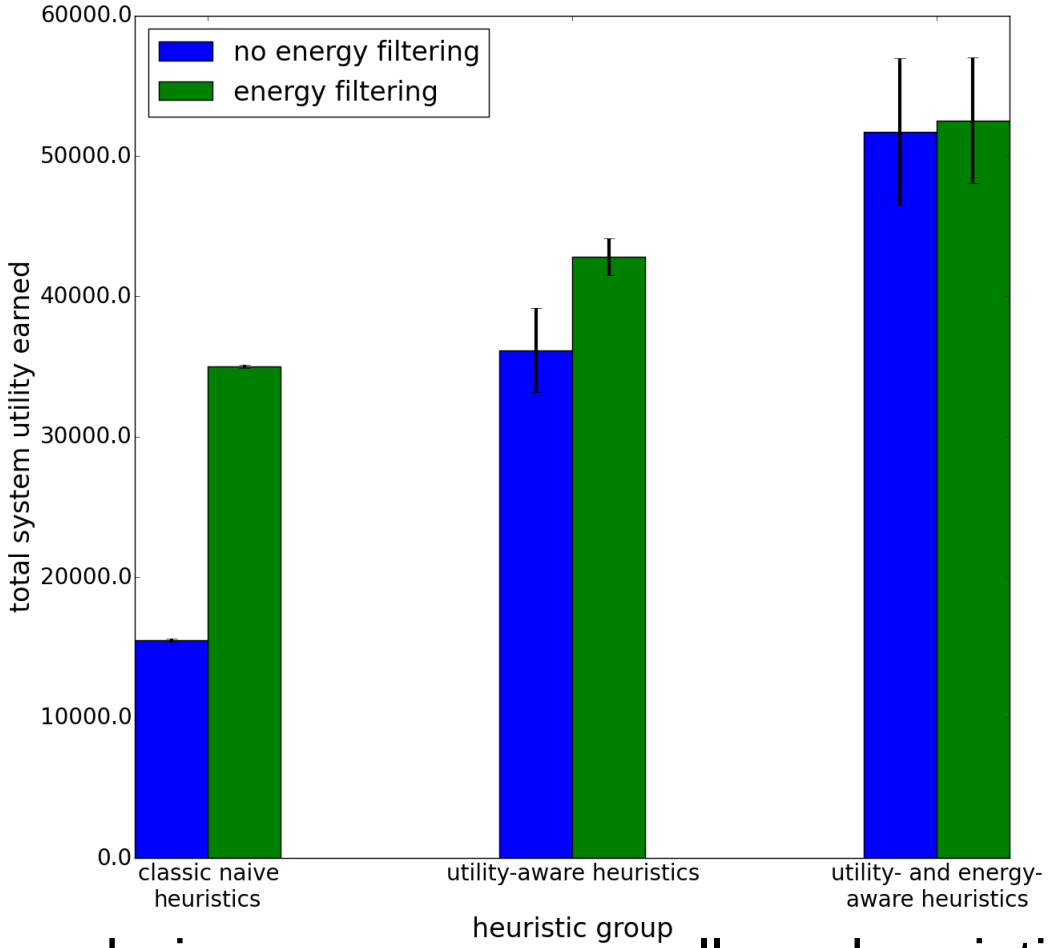
- **oversubscribed** environment
  - not all tasks can completed when desired
- tasks are assumed to be **serial**
  - execute on a single machine
- **no preemption** of tasks
  - once a task starts, it executes to completion
- **energy constraint** on the system
  - limits amount of energy that can be consumed in a day

Contributions

- designed and analyze four new batch resource management heuristics that try to maximize utility given an energy constraint
  - compared these against three previous batch heuristics
- designed a custom energy filtering mechanism
  - adapts to energy remaining in the day
  - enforces “fairness” in energy consumption
  - makes utility-aware heuristics become energy-aware
- sensitivity analysis on the level of energy-awareness for all heuristics
- method to create low TMA and high TMA environments based on a reference environment (without changing other heterogeneity measures)
  - analysis of all heuristics in these environments

Results of Different Heuristics Groups

100 task types & ~50,000 tasks, 13 machine types & 100 machines



- classic naïve heuristics – representative of many real-world systems, do not consider utility nor energy
- utility-aware heuristics – only consider utility
- utility- and energy-aware heuristics – explicitly consider both utility and energy
- all results used dropping

- being energy-aware allows heuristics to conserve energy and execute tasks throughout the entire day
- energy filtering is an ad-hoc way to introduce energy-awareness

Colorado State University

Outline

- Chapter 1: introduction
- Chapter 2: dynamic utility maximization
- dynamic energy-aware resource allocation
  - Chapter 3: utility maximization with an energy constraint
  - Chapter 4: energy constrained utility maximization for different queuing models
- multi-objective optimization
  - Chapter 5: deterministic utility vs. energy
  - Chapter 6: deterministic makespan vs. energy
  - Chapter 7: stochastic utility/makespan vs. energy
- Chapter 8: application co-location in multicore systems
- Chapter 9:conclusions

Assumed Environment

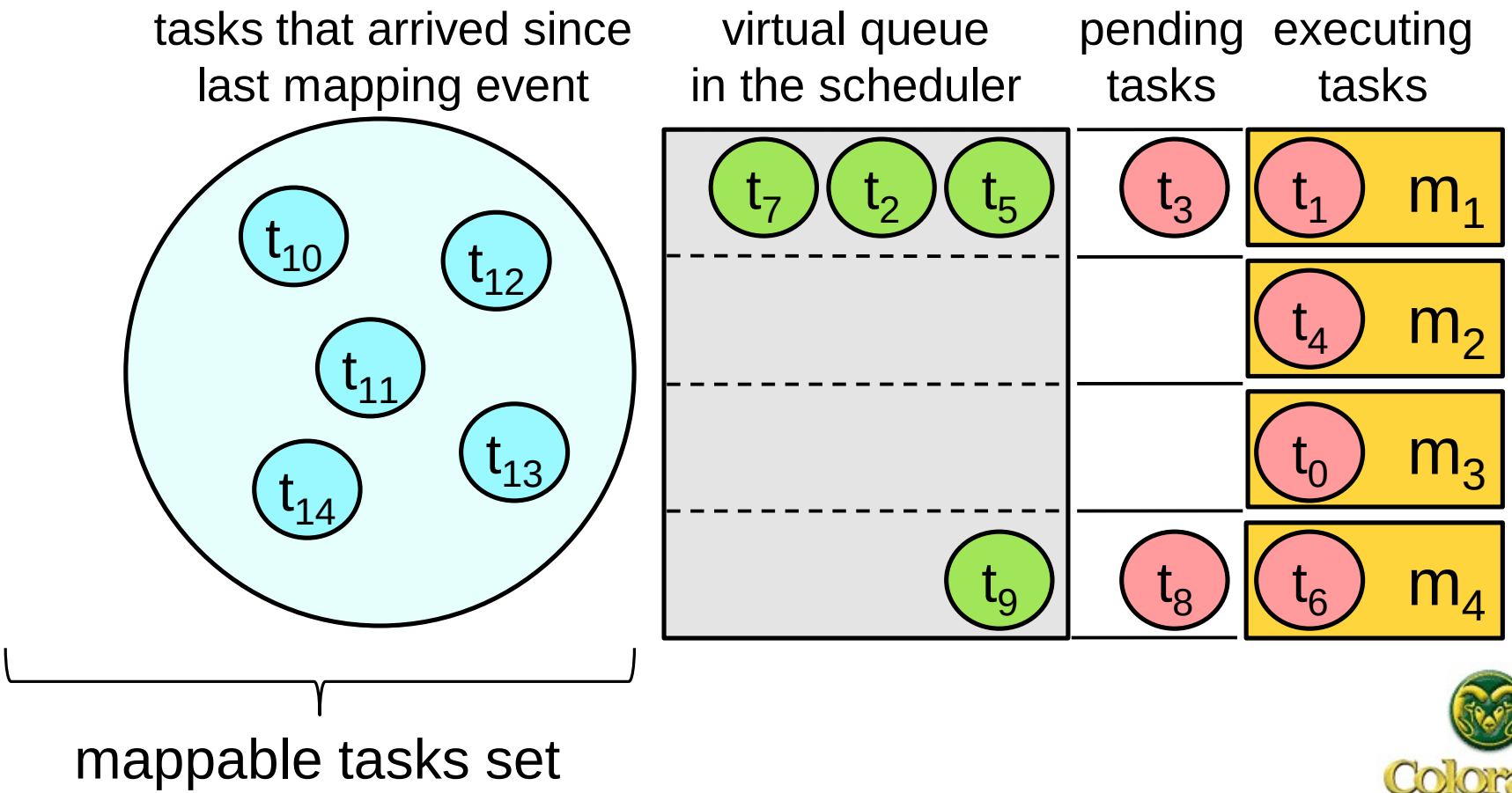
- **oversubscribed** environment
  - not all tasks can completed when desired
- tasks are assumed to be **serial**
  - execute on a single machine
- **no preemption** of tasks
  - once a task starts, it executes to completion
- **energy constraint** on the system
  - limits amount of energy that can be consumed in a day

Task Management Environments

- schedulers wait for some time interval and then schedule the accumulated tasks to machines for execution
  - time interval is usually 60 or 120 seconds
- **mapping event** – when scheduler makes allocation decisions
- modeled and analyzed performance in two task management environments
  - **“queued”**
  - **“polled”**
- helps in understanding trade-offs between these approaches

Batch Mode Queue Model

- mapping event – interval of time (1 minute)
- mappable batch: new tasks and virtual queue of tasks

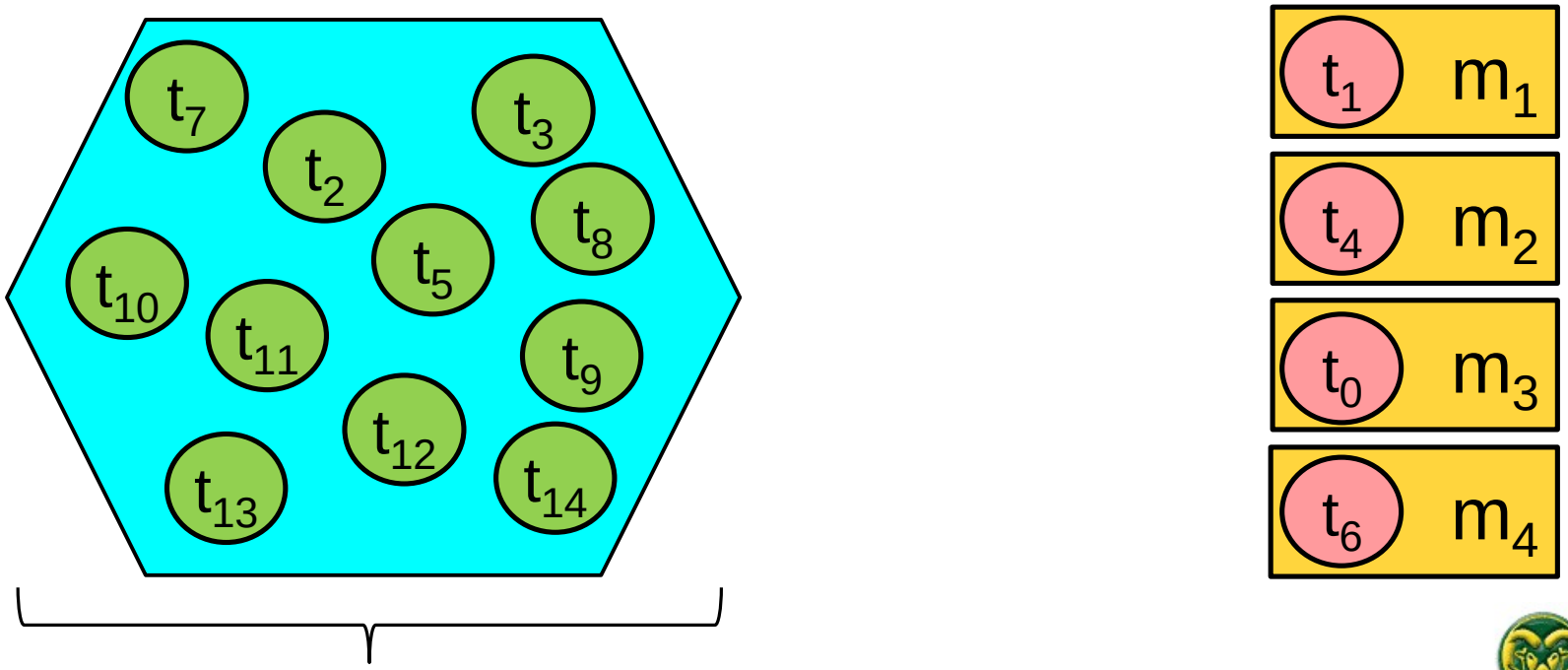


Polled Mode Model

- mapping event – interval of time (1 minute)
- mappable batch: new tasks and unmapped tasks

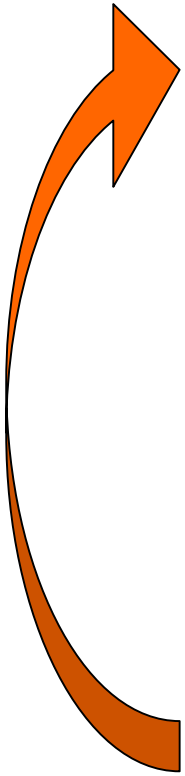
previously unmapped tasks and tasks that arrived since last mapping event

executing tasks



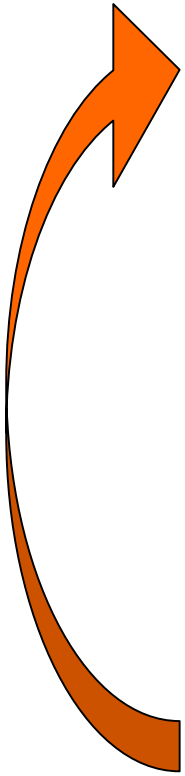
Colorado State University

First Come First Served (FCFS) based heuristics



- at a mapping event, sort all the mappable tasks in an **ascending** order of their arrival time
- for the first task in the sorted list, make assignment for the machine with the earliest available time that
  - is a valid assignment
  - satisfies the energy filter (i.e., does not consume undesirably high energy)
- remove the assigned task from the set of mappable tasks
- update the ready time distribution of the machine to which a task was just assigned
- repeat by considering next task in sorted list

Last Come First Served (LCFS) based heuristics



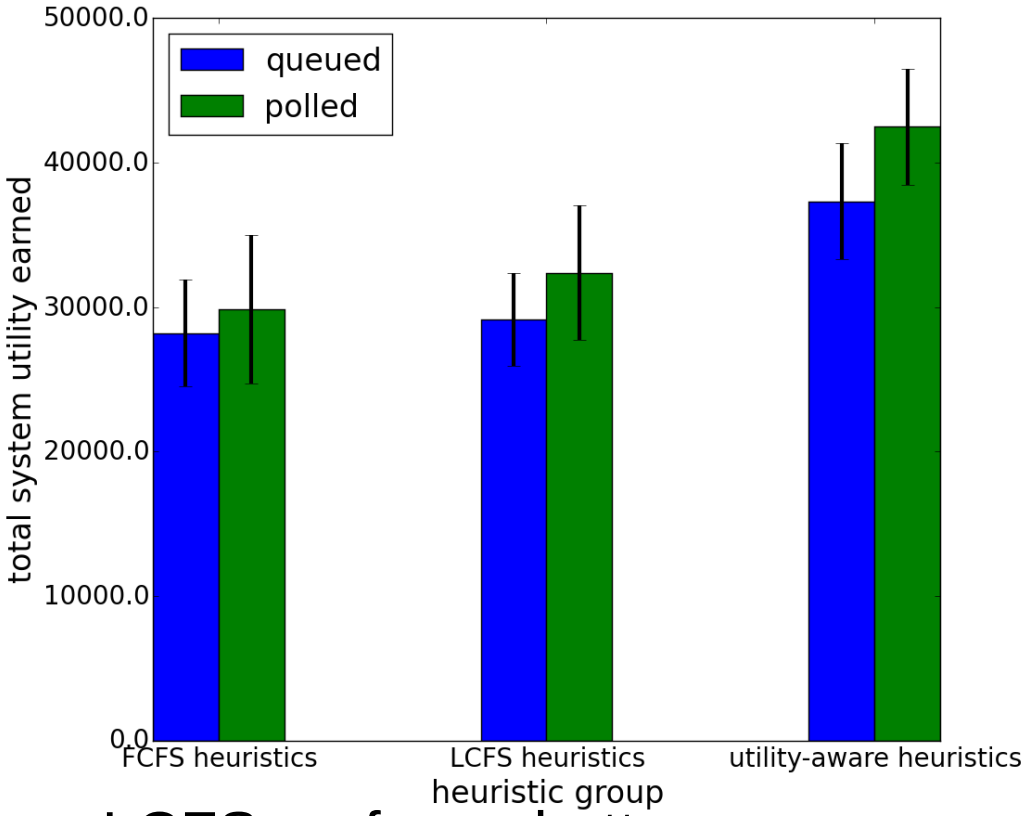
- at a mapping event, sort all the mappable tasks in an **descending** order of their arrival time
- for the first task in the sorted list, make assignment for the machine with the earliest available time that
  - is a valid assignment
  - satisfies the energy filter (i.e., does not consume undesirably high energy)
- remove the assigned task from the set of mappable tasks
- update the ready time distribution of the machine to which a task was just assigned
- repeat by considering next task in sorted list

Contributions

- designed a novel adaptive energy filter mechanism
  - no parameters to “tune”
  - can be deployed in any energy constrained environment
- comparative analysis of the advantages and disadvantages of a polled task management environment for HPC environments
- comparison of FCFS and LCFS that are typically used in real schedulers with smarter heuristics that can improve system performance

Results of Different Heuristics Groups

100 task types & ~32,000 tasks,
13 machine types & 100 machines



- FCFS and LCFS heuristics – represent techniques commonly used in real schedulers
- utility-aware heuristics – modified from high performing heuristics in Chapters 2 and 3 to work in polled environment
- all results used energy filtering
- all results used dropping

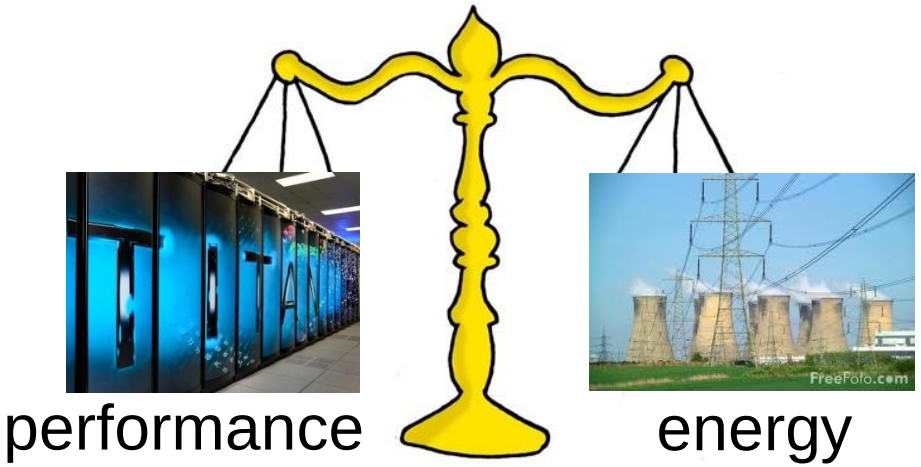
- LCFS performs better on average the FCFS because it can service newly arriving high utility tasks faster
- polled environment does not lock tasks into pending slots, services newly arriving high utility tasks faster than queued environment, but has increased idle time between mappings

Outline

- Chapter 1: introduction
- Chapter 2: dynamic utility maximization
- dynamic energy-aware resource allocation
  - Chapter 3: utility maximization with an energy constraint
  - Chapter 4: energy constrained utility maximization for different queuing models
- multi-objective optimization
  - Chapter 5: deterministic utility vs. energy
  - Chapter 6: deterministic makespan vs. energy
  - Chapter 7: stochastic utility/makespan vs. energy
- Chapter 8: application co-location in multicore systems
- Chapter 9: future work

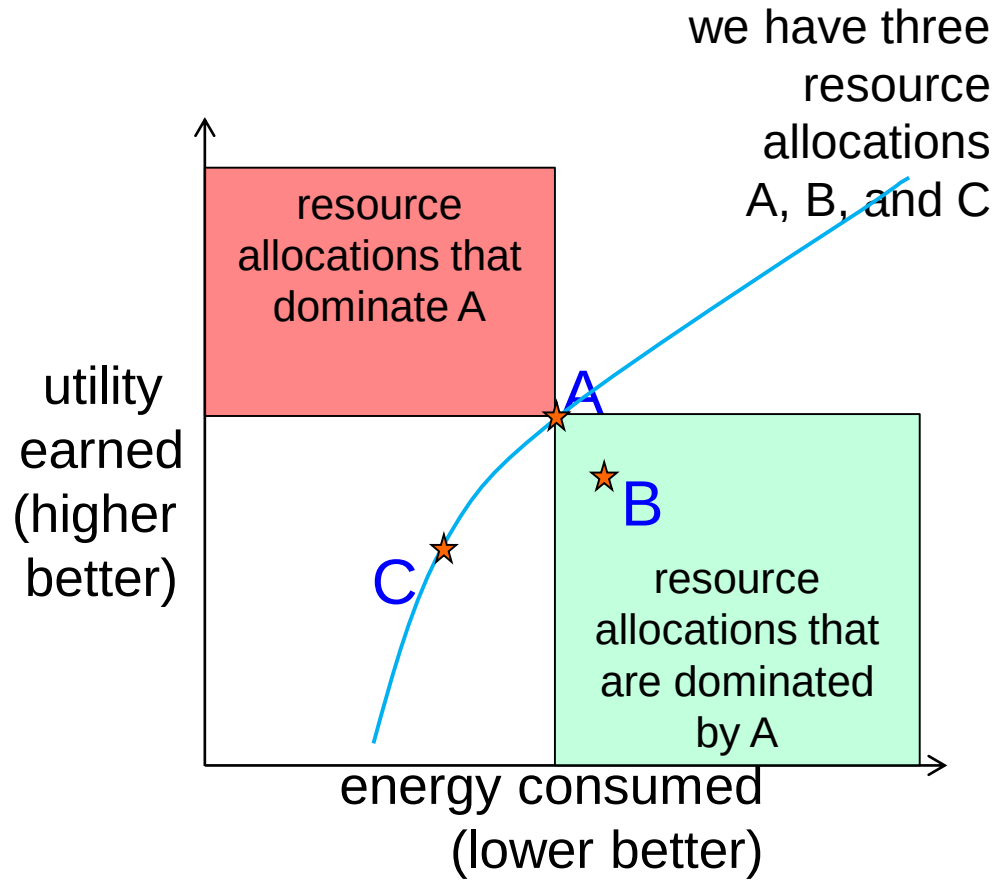
Bi-Objective Energy-Aware Study

- why is it important to study bi-objective optimization?
  - real world problems often have conflicting multiple objectives
- **goal:** evaluate trade-offs between system performance and energy consumption
- can use system workload traces that include task arrival times
- can be used to select energy constraints for dynamic systems



Colorado State University

Pareto Fronts

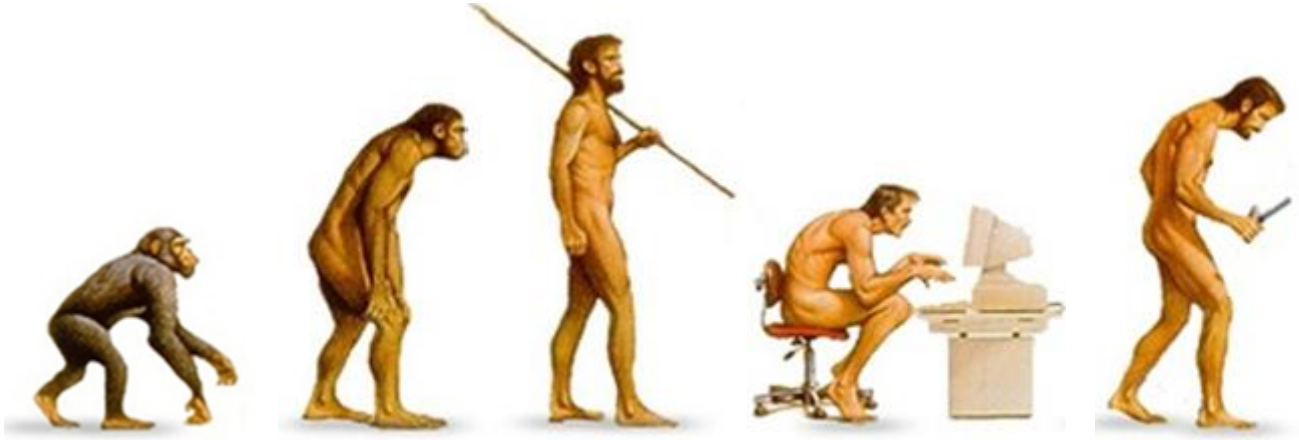


- A dominates B because A uses less energy AND earns more utility → A better resource allocation
- neither A nor C dominate each other because A is better for utility but C is better for energy → neither is better in both

- a Pareto front contains all the resource allocations not "dominated" by any other resource allocation
- facilitate trade-off analyses

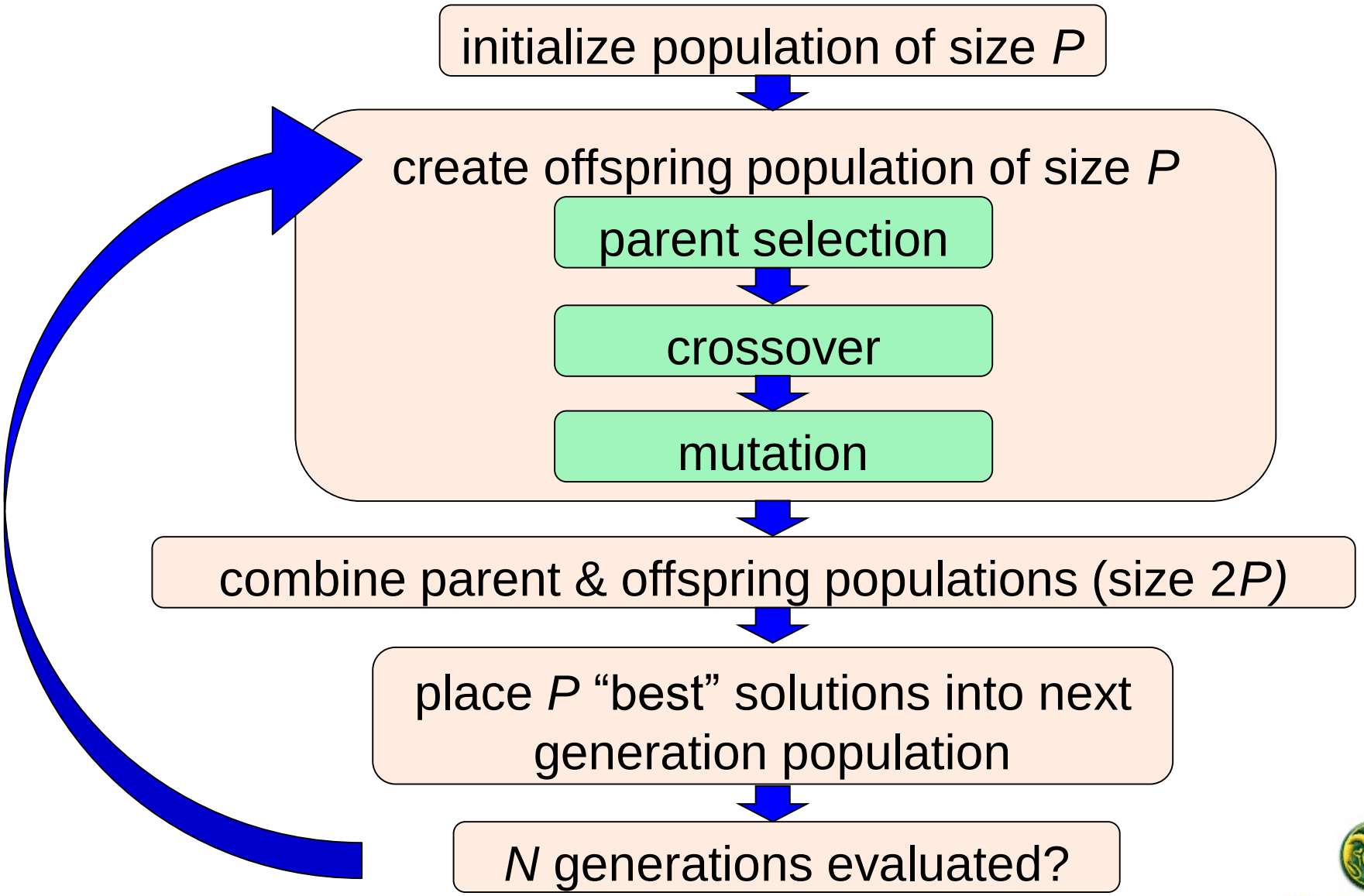
Bi-Objective Optimization Using Genetic Algorithms

- genetic algorithms are search heuristics used to find solutions to optimization problems
- genetic algorithms try to mimic the process of natural evolution
- solutions “evolve” through time by passing on useful traits



- preferably solutions should be diverse and evenly distributed across the Pareto front
- adapted NSGA-II genetic algorithm framework from literature

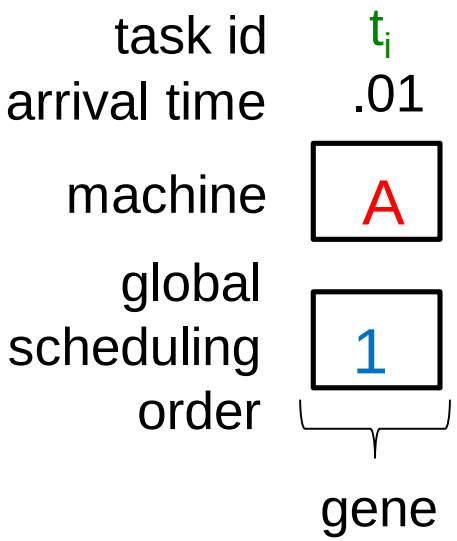
NSGA-II Overview



Colorado State University

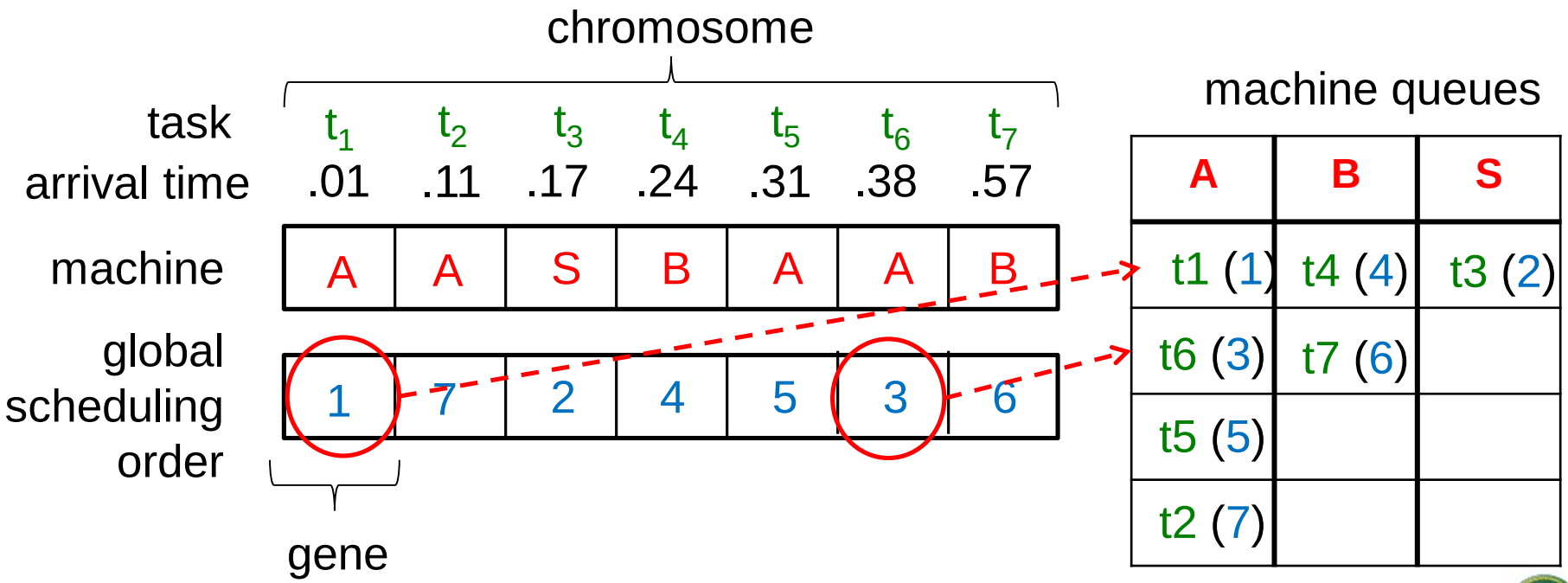
Gene Structure

- a gene represents a task to machine mapping
- contains:
  - the machine the task will execute on
  - the global scheduling order of the task



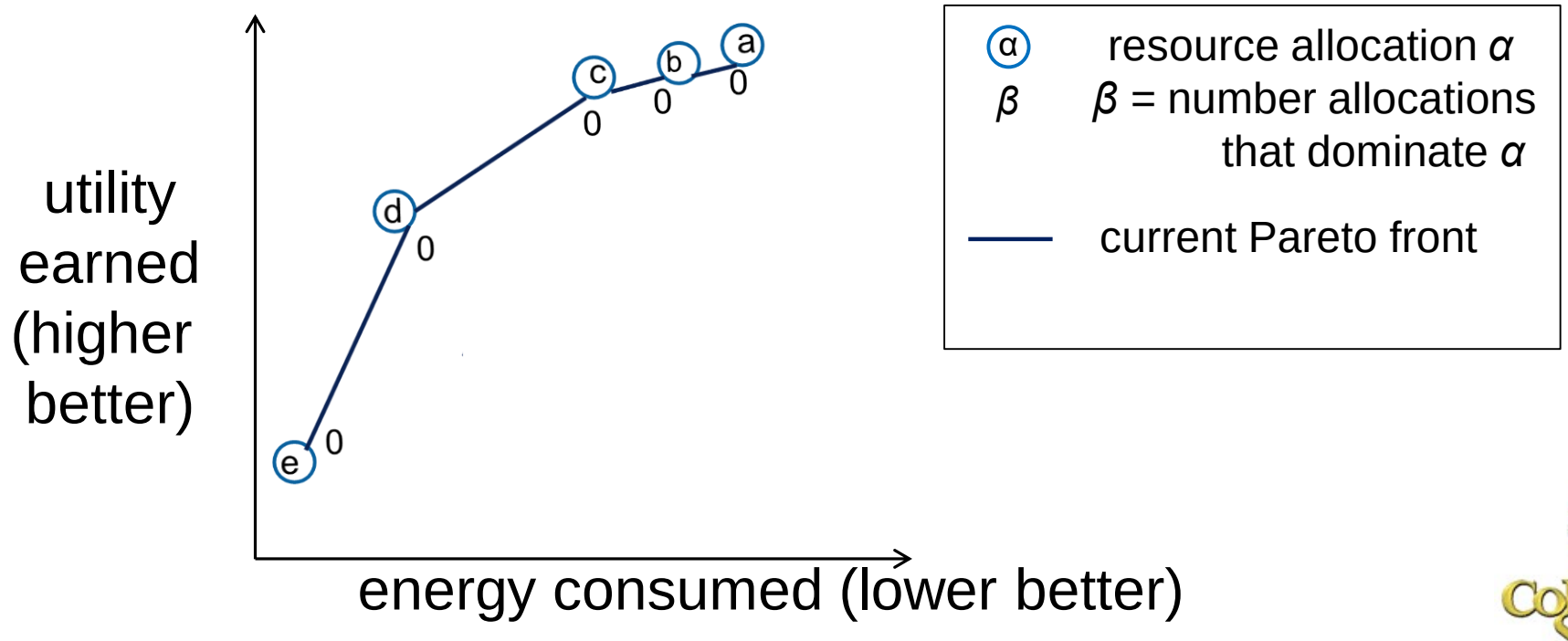
Chromosome Structure

- a **chromosome** represents a complete allocation (solution)
- number of **genes** in a chromosome = number of tasks
- the i^{th} gene in a chromosome is the i^{th} task to arrive
- **population**: consists of multiple chromosomes



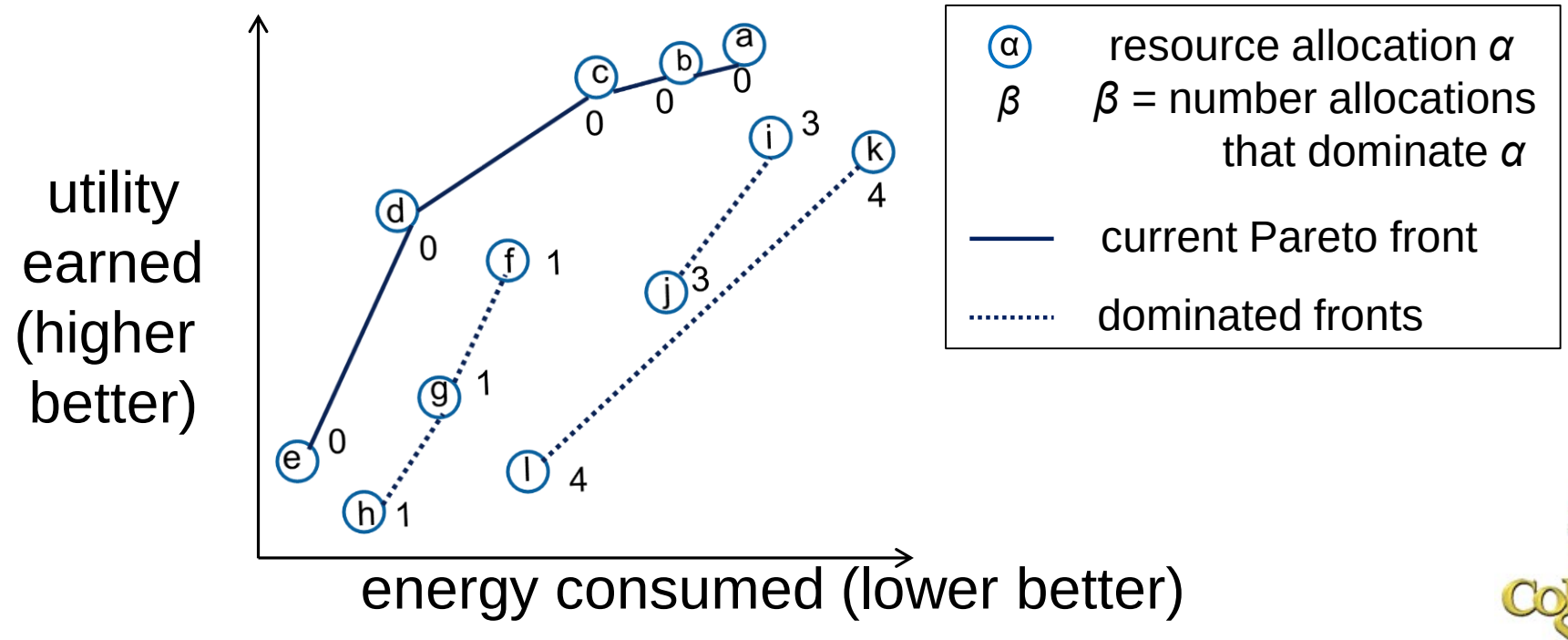
Pick Best P Chromosomes for Next Generation

- sorts solutions (resource allocations) into domination fronts
  - determined by the number of solutions which dominate it
- select non-dominated chromosomes to form the next population
  - if > *P* chromosomes exist, trim based on "crowding distance"



Pick Best P Chromosomes for Next Generation

- sorts solutions (resource allocations) into domination fronts
  - determined by the number of solutions which dominate it
- select non-dominated chromosomes to form the next population
  - if > *P* chromosomes exist, trim based on "crowding distance"
  - if < *P*, add the best dominated solutions until there are *P* chromosomes (ties broken based on "crowding distance")



Outline

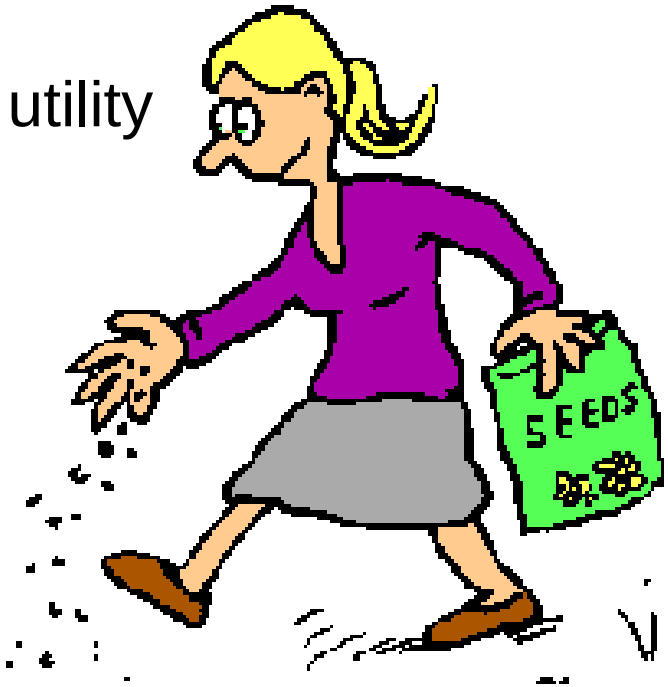
- Chapter 1: introduction
- Chapter 2: dynamic utility maximization
- dynamic energy-aware resource allocation
  - Chapter 3: utility maximization with an energy constraint
  - Chapter 4: energy constrained utility maximization for different queuing models
- multi-objective optimization
  - Chapter 5: deterministic utility vs. energy
  - Chapter 6: deterministic makespan vs. energy
  - Chapter 7: stochastic utility/makespan vs. energy
- Chapter 8: application co-location in multicore systems
- Chapter 9: future work

Assumed Environment

- **oversubscribed** environment
  - total work exceeds capacity of system
- tasks are assumed to be **independent**
  - communication is not required between the tasks and there are not any precedence constraints
- tasks are assumed to be **serial**
  - execute on a single machine
- **static analysis** is performed
  - examining trace of dynamic task arrivals

Fast Greedy Heuristics to Seed Initial GA Population

- consider tasks in order of arrival time
  - Max Utility
    - map task to machine earns most utility
  - Min Energy
    - map task to machine that consumes least energy
  - Max Utility-Per-Energy
    - map task to machine with most utility per unit of energy



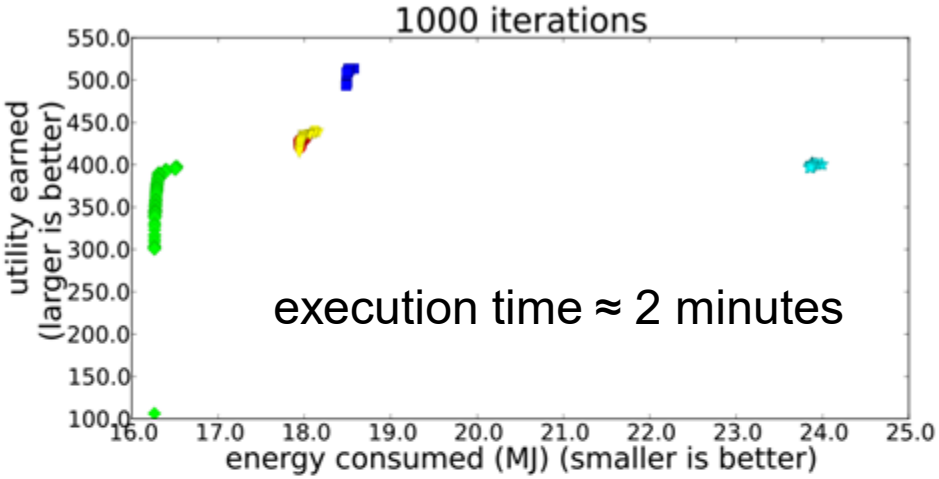
- batch heuristic: Min Completion Time
  - consider all tasks
  - considers task arrival time
  - iteratively map earliest finishing task
- rest of population is random

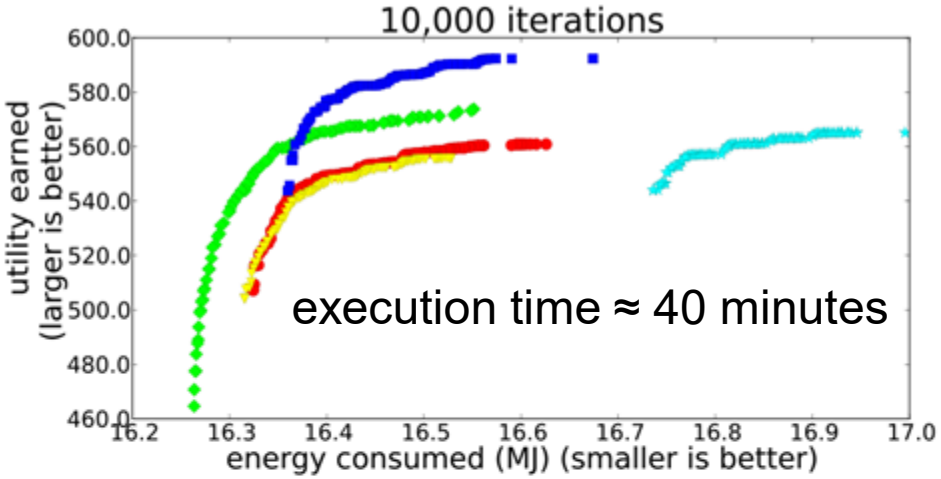
Contributions

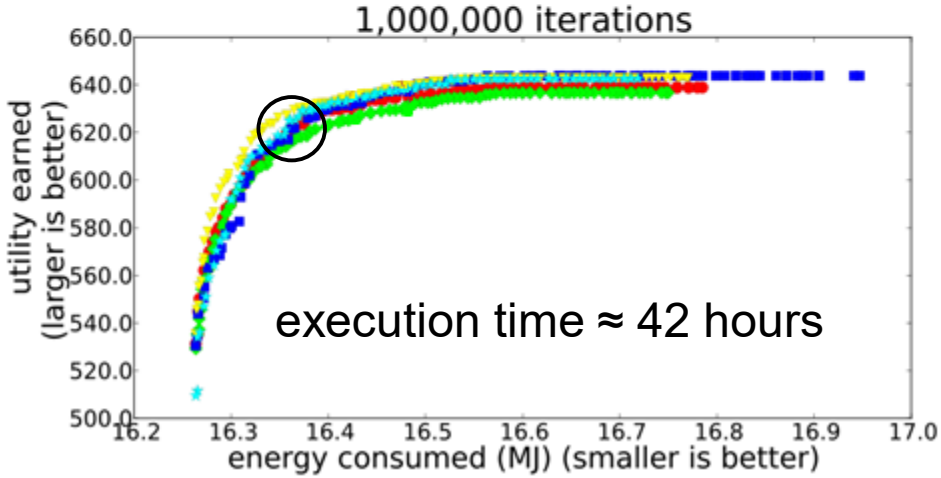
- adapted a popular multi-objective genetic algorithm (NSGA-II) for use with heterogeneous HPC resource allocation
  - chromosome structure
  - crossover and mutation operations
- modeled a bi-objective resource allocation problem
  - utility earned and energy consumed
  - based on compute facility and workload being investigated by the Extreme Scale Systems Center at ORNL
- created and evaluated many intelligent resource allocations
  - show how utility and energy change drastically in a system
- analyze the effects the different seeding heuristics have on performance of the algorithm
- analyze how algorithm scales
  - three different sized environments

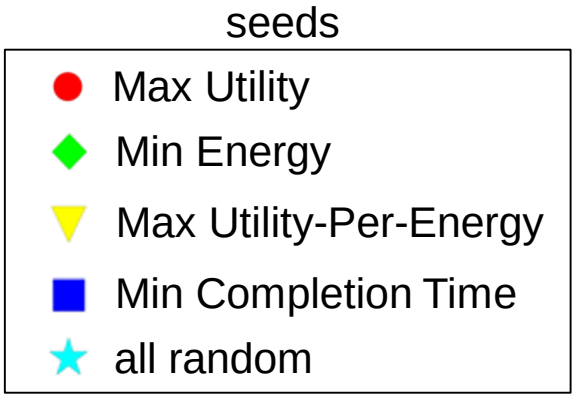
Pareto Front Evolution

30 task types & ~1,000 tasks, 13 machine types & 30 machines
100 chromosomes per population







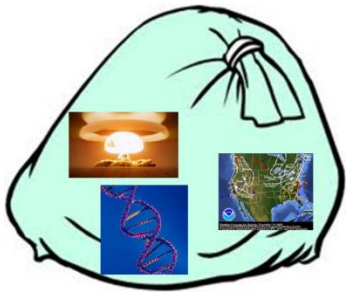


Outline

- Chapter 1: introduction
- Chapter 2: dynamic utility maximization
- dynamic energy-aware resource allocation
  - Chapter 3: utility maximization with an energy constraint
  - Chapter 4: energy constrained utility maximization for different queuing models
- multi-objective optimization
  - Chapter 5: deterministic utility vs. energy
  - Chapter 6: deterministic makespan vs. energy
  - Chapter 7: stochastic utility/makespan vs. energy
- Chapter 8: application co-location in multicore systems
- Chapter 9: future work

Bi-Objective Optimization of Makespan and Energy

- additional performance metrics can be used in place of utility
- also studied minimizing **makespan** vs. minimizing energy
  - makespan: total amount of time it takes for all the tasks to finish executing across all machines
- static and offline environment
  - bag of tasks (batch)
  - every task in the workload is known *a priori*



- study one:

  R. Friese, et al., "Analyzing the trade-offs between minimizing makespan and minimizing energy consumption in a heterogeneous resource allocation problem," in *The 2nd International Conference on Advanced Communications and Computation (INFOCOMP 2012)*, Venice, Italy, Oct. 2012, pp. 81-89, received one of seven **best paper** awards given..
- study two:

  R. Friese, et al., "A machine-by-machine analysis of a bi-objective resource allocation problem," in *The 2013 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2013)*, Las Vegas, NV, July 2013, pp. 3-9.

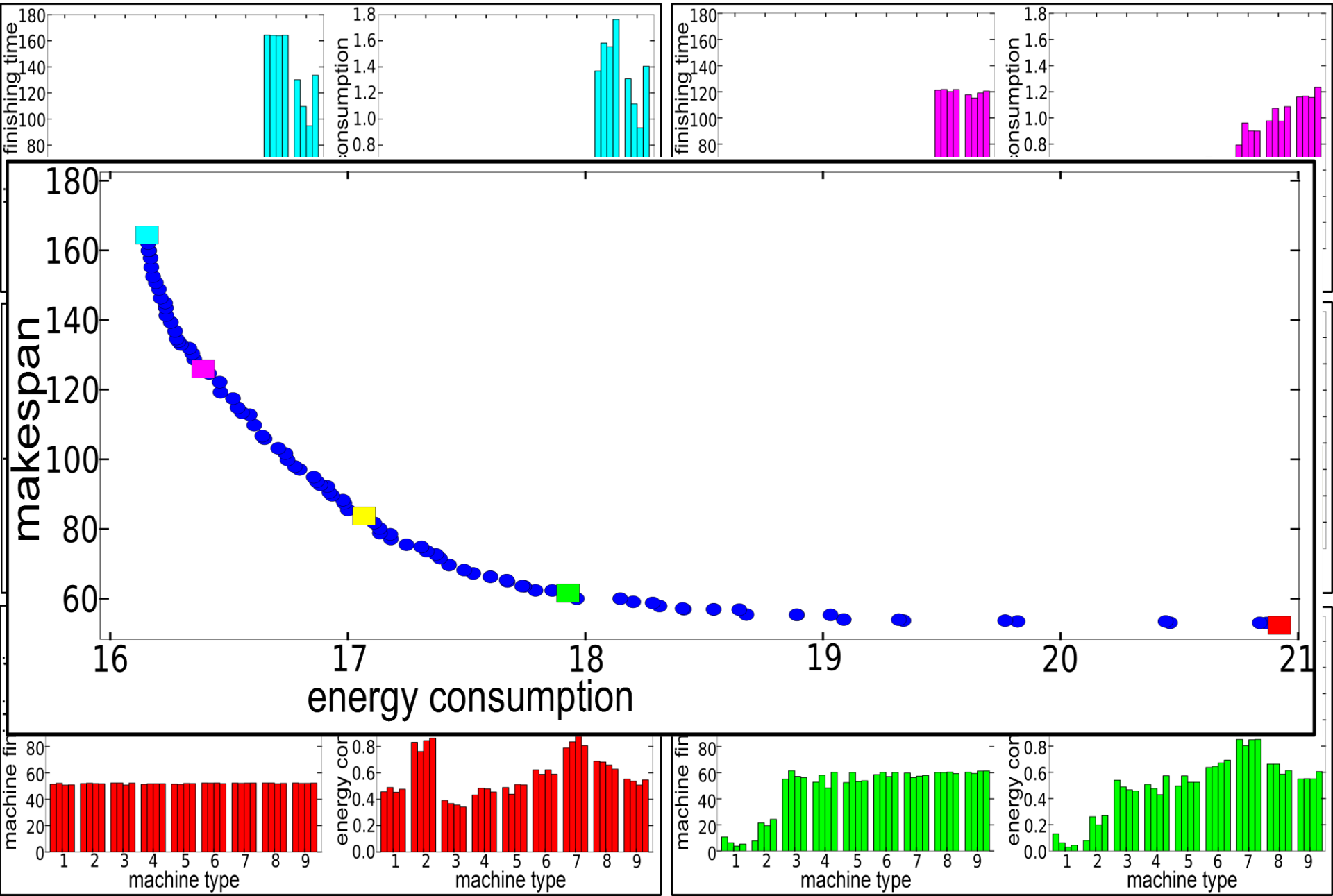
Contributions

- modeled a bi-objective resource allocation problem
  - makespan and energy consumed
  - allows for a trade-off analysis
- analyzed on a machine-by-machine basis how different resource allocations affect the behavior on the individual machines
- provided an approach to identify
  - energy-efficient machines
  - energy-inefficient machines
- an analysis approach to perform “what-if” experiments
- showed the versatility of this technique by examining various heterogeneous environments

Simulation Setup

- 36 machines
  - 9 machine types (4 machines per type)
- 9 machine types are based on real machines
- 1000 tasks
  - 30 tasks types
- Pareto fronts were generated using NSGA-II
- any algorithm that creates Pareto fronts could be used as well
- 100 chromosomes in population
  - one min-energy seed
  - one min-completion time seed
  - 98 random chromosomes
- 100,000 iterations

9 Machine Type Environment Analysis

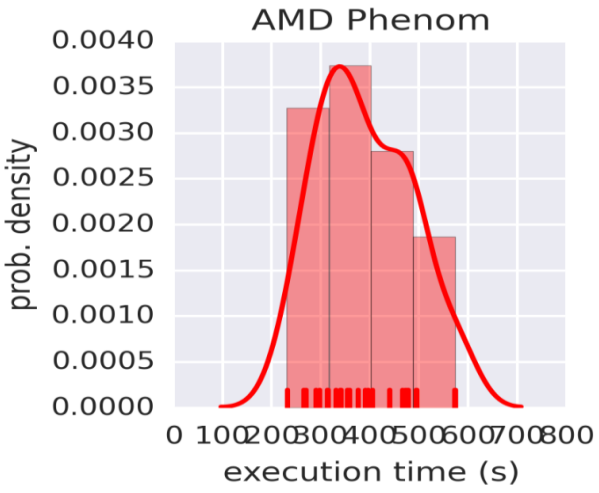


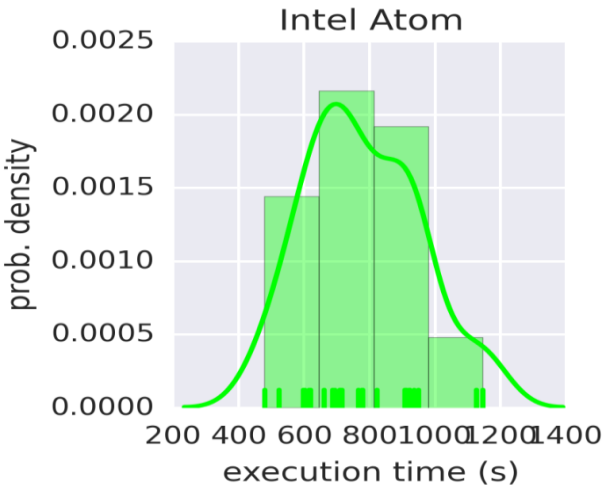
Outline

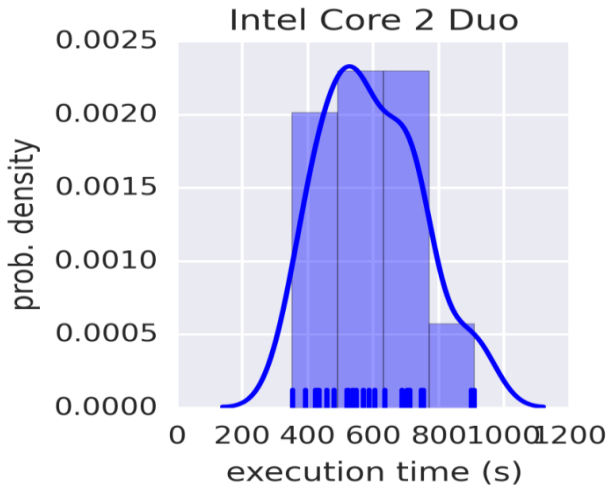
- Chapter 1: introduction
- Chapter 2: dynamic utility maximization
- dynamic energy-aware resource allocation
  - Chapter 3: utility maximization with an energy constraint
  - Chapter 4: energy constrained utility maximization for different queuing models
- multi-objective optimization
  - Chapter 5: deterministic utility vs. energy
  - Chapter 6: deterministic makespan vs. energy
  - Chapter 7: stochastic utility/makespan vs. energy
- Chapter 8: application co-location in multicore systems
- Chapter 9: future work

Stochastic Multi-Objective Optimization

- use distributions to represent execution uncertainty
- a task's execution can have different execution times when using different data sets
  - even on the same machine/P-state
- exact execution time is uncertain, but historical information can be used to create a probability distribution of its possible execution times
- referred to as stochastic model of task execution times
- current work – journal paper in preparation







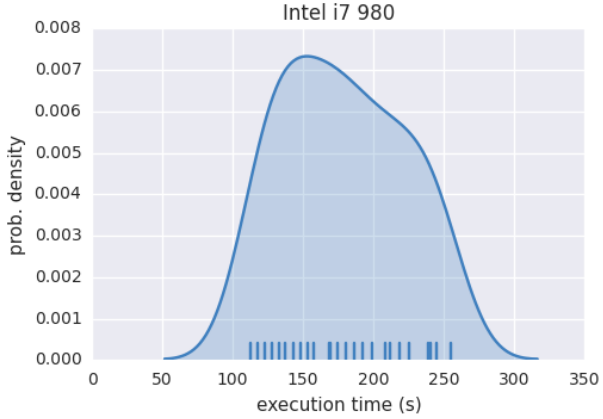
Stochastic Data - Applications

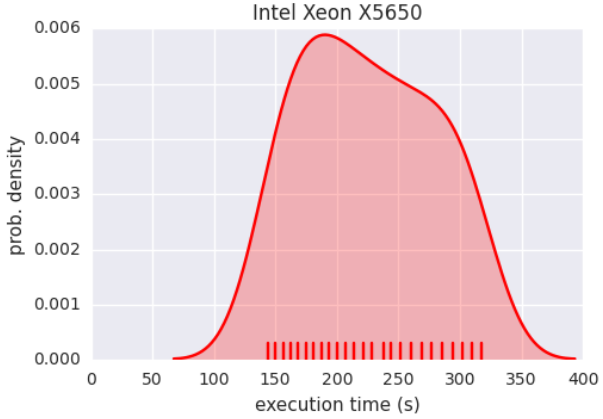
- 8 applications from the Parsec Benchmark Suite
  - bodytrack, canneal, ferret, fluidanimate, freqmine, raytrace, streamcluster, swaptions
- 25 unique input sets for each of the 8 applications
  - the applications may have different numbers of inputs and different acceptable values for those inputs
  - the i^{th} input set for an application is the same regardless of the machine it runs on
- uniformly distribute the 25 input sets over a range of acceptable values for each application
  - using Latin hypercube sampling from the pyDOE
- applications only use a single thread, but can be multithreaded

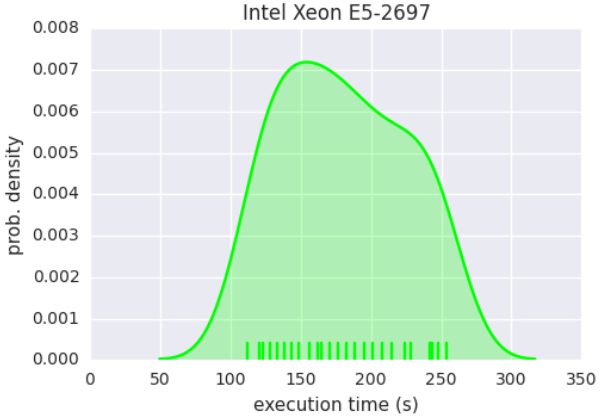
Stochastic Data - Machines

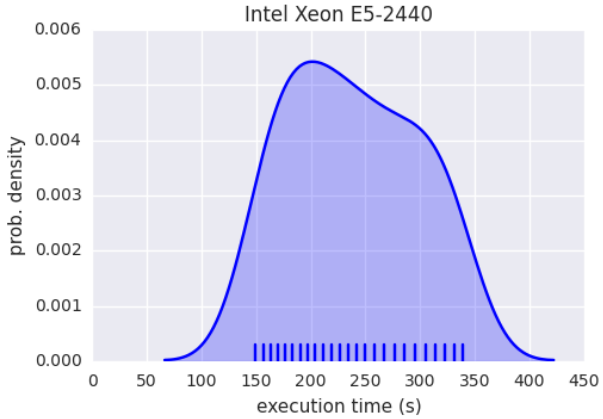
- data gathered for 8 machines:
- Intel Core i7 980, 6-core, 130W TDP, 12MB L3, 3.33 Ghz
- Intel Xeon X5650 6-core, 95W TDP, 12MB L3, 2.66 Ghz12
- Intel Xeon E5-2697 12-core, 130W TDP, 30 MB L3, 2.7 Ghz
- Intel Xeon E5-2440 8-core, 95W TDP, 20MB L3, 1.9 Ghz
- AMD Phenom 9850, 4-core, 125W TDP, 2MB L3, 2.5Ghz
- Intel Xeon E3-1225 4-core, 84W TDP, 8MB L3, 3.2 Ghz
- Intel Core 2 Duo T8300, 2-core, 35W TDP, 3MB L2, 2.4 Ghz
- Intel Atom N455, 1-core, 6.5 W TDP, 512 KB L2,1.66Ghz
- note: energy is gathered at the “outlet” level

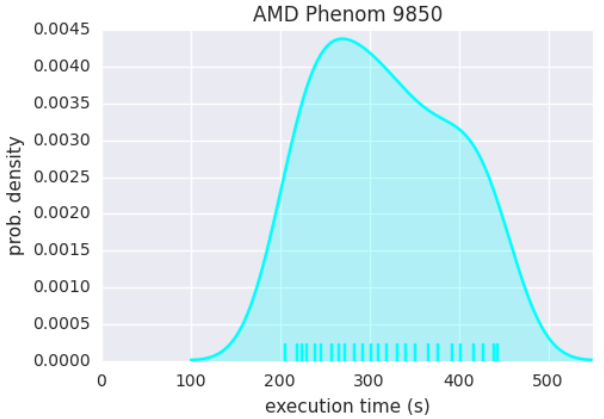
Fluidanimate Execution Times

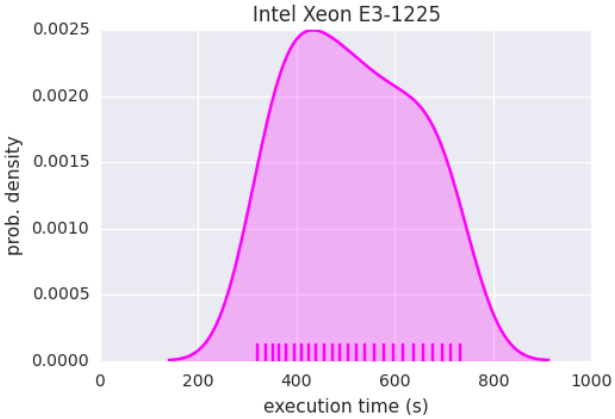


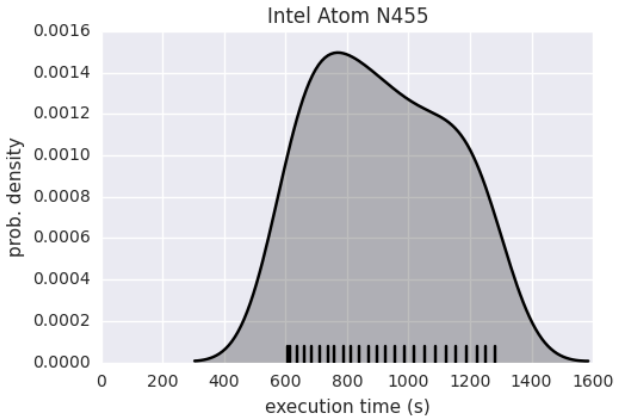


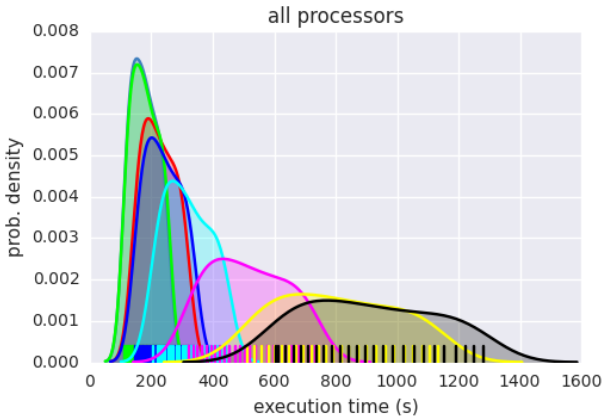




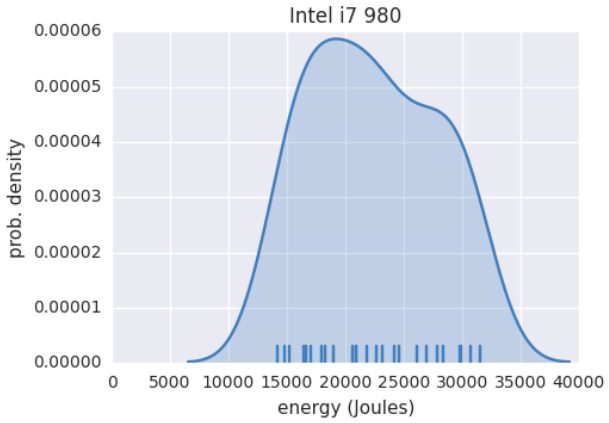


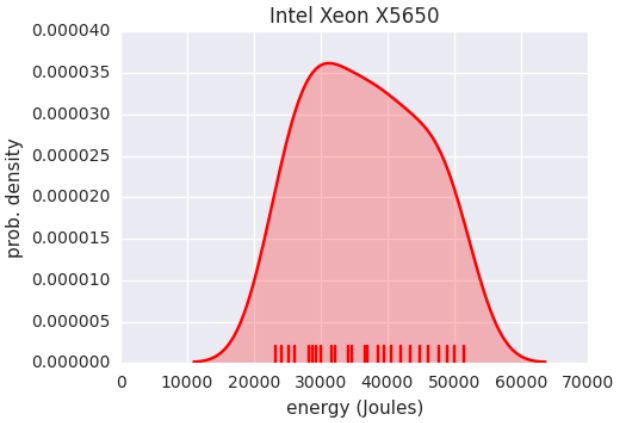


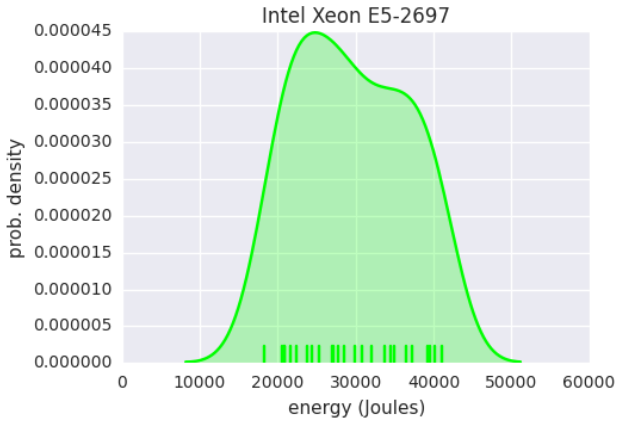


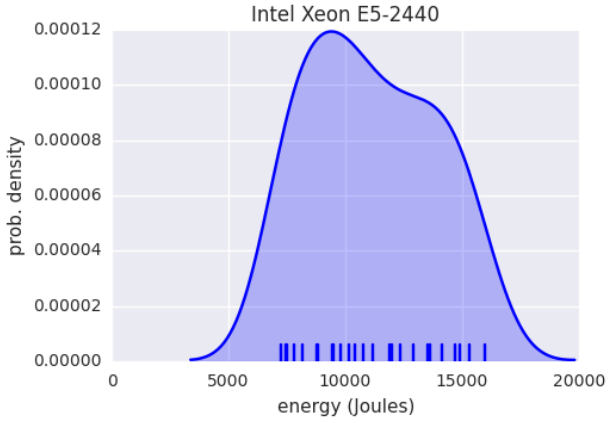


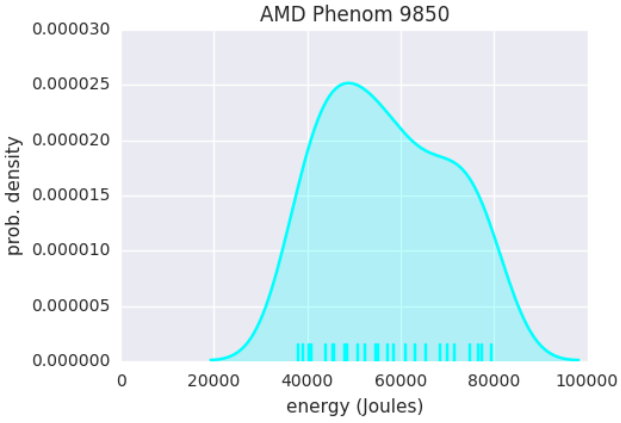
Fluidanimate Energy Consumption

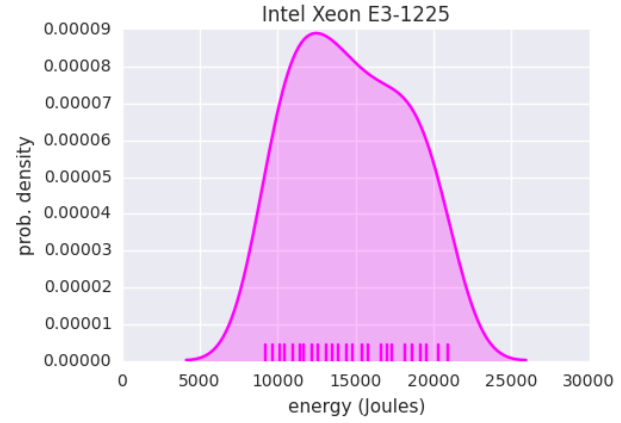


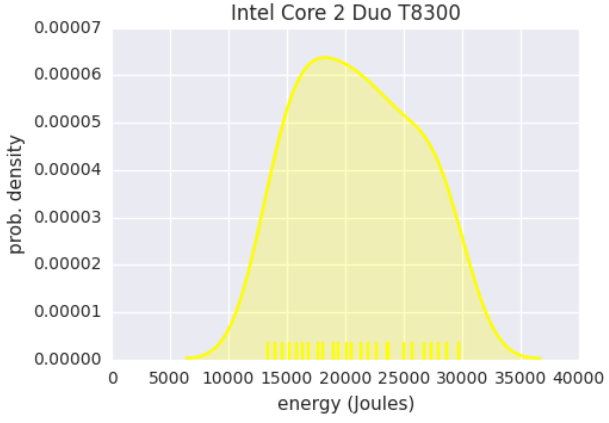


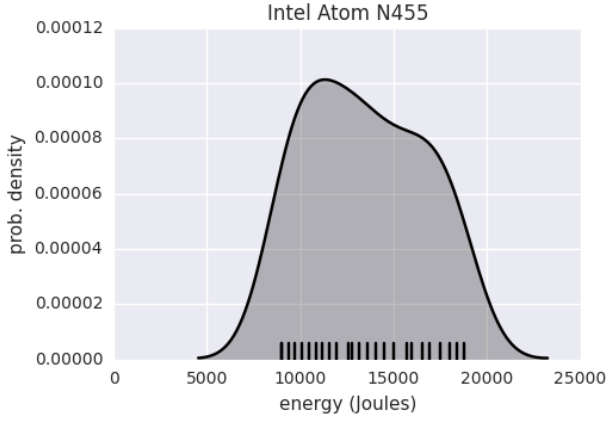


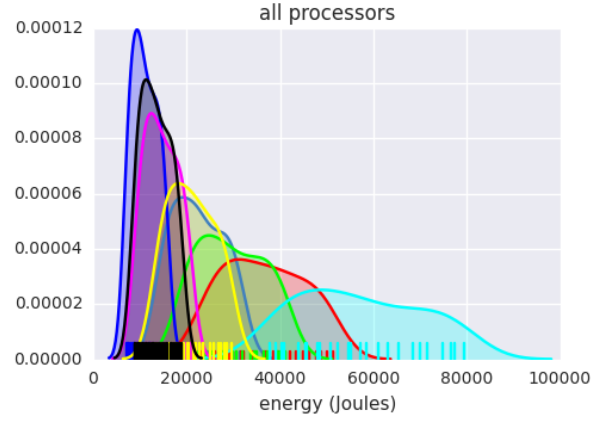




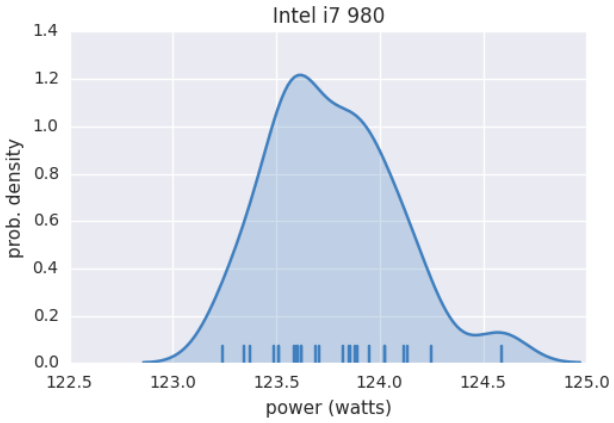


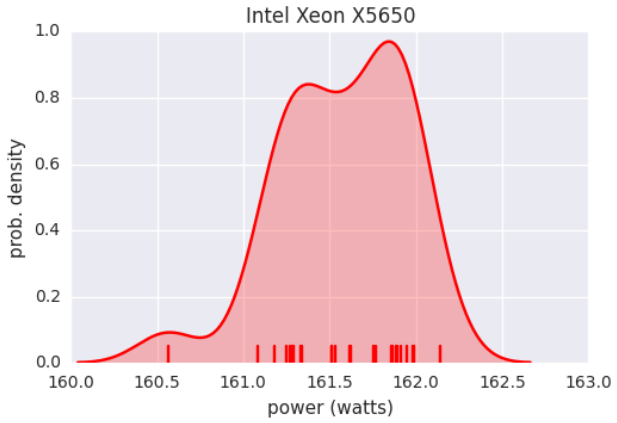


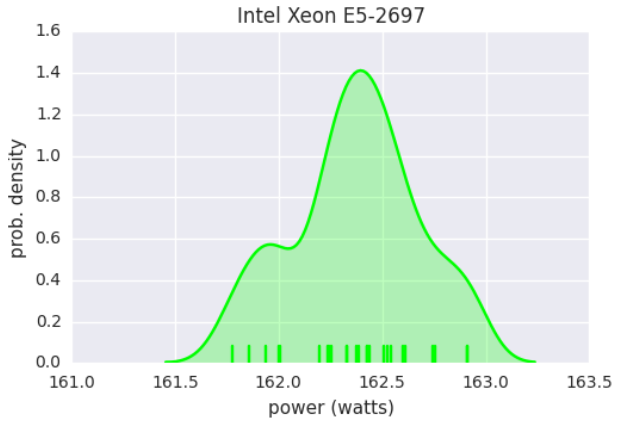


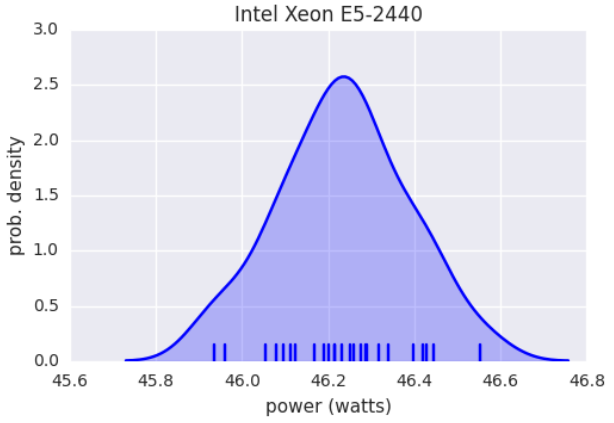


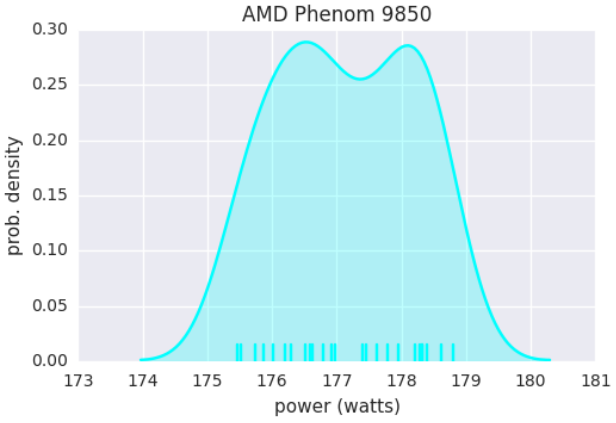
Fluidanimate Average Power

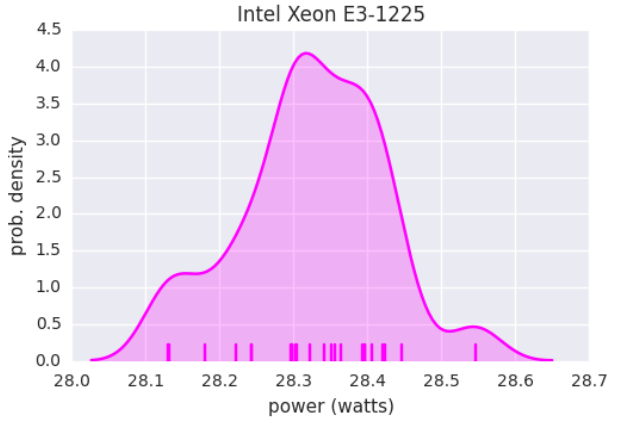


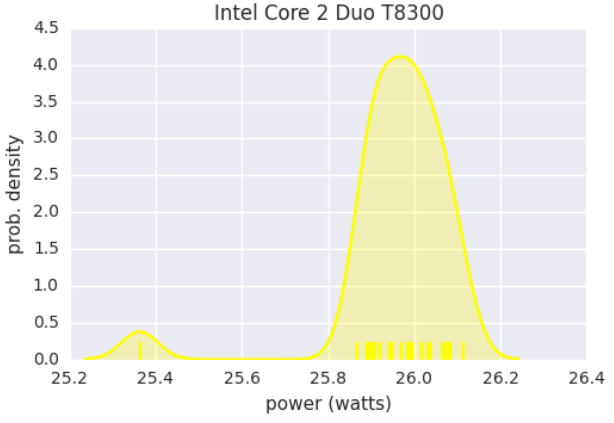


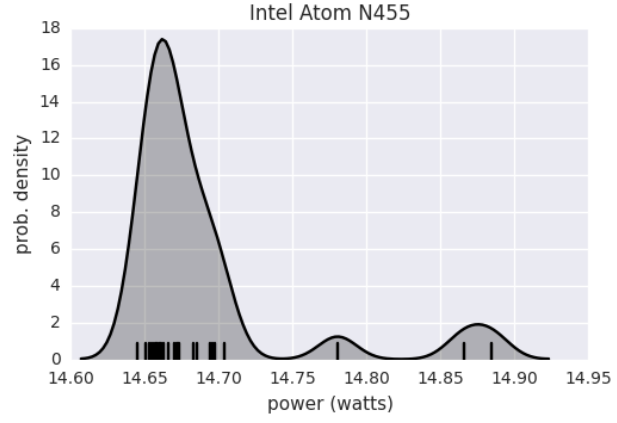


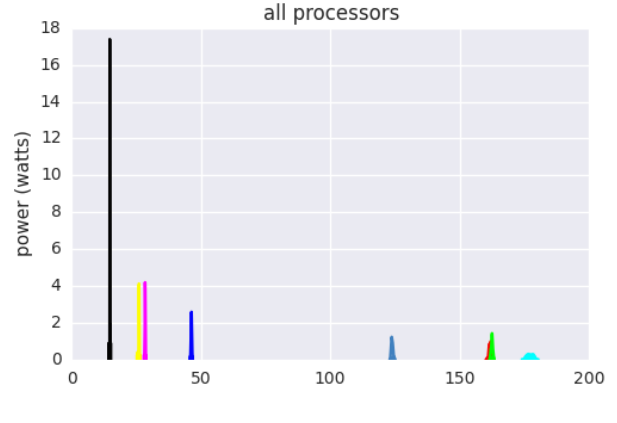










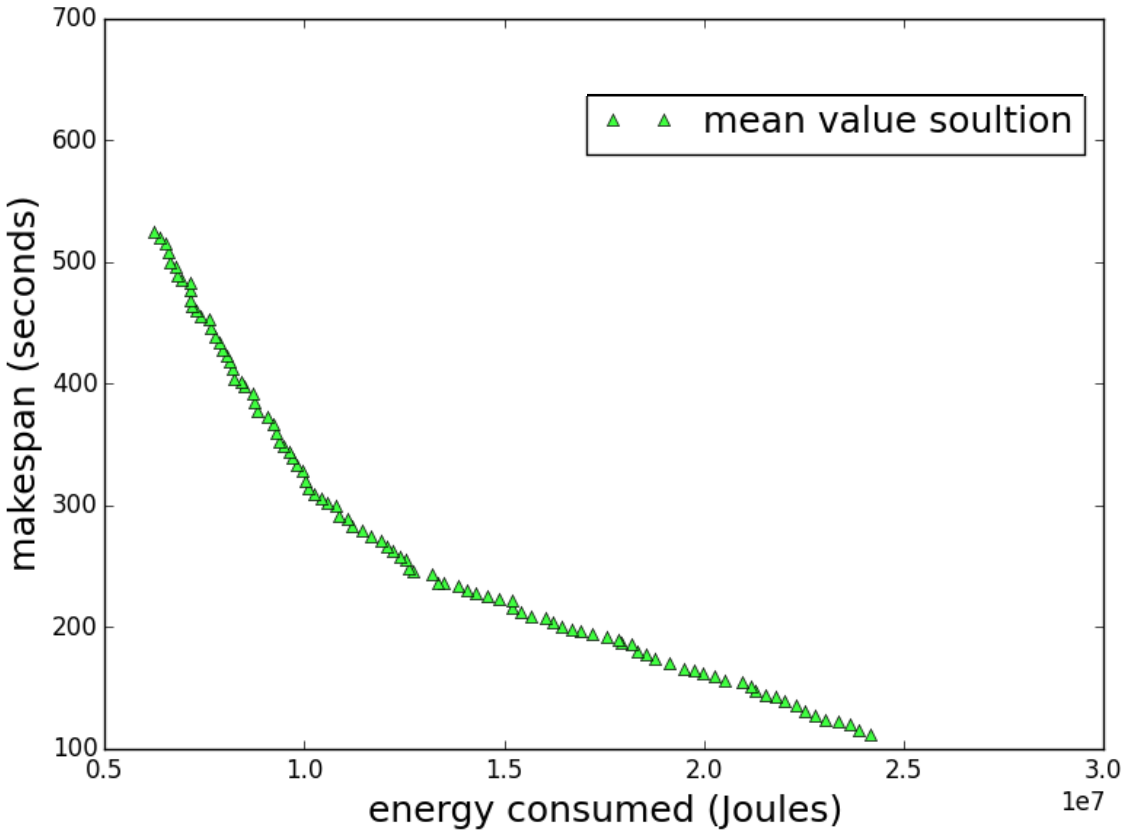


Different Types of Objective Functions

- the inclusion of stochastic information allows for multiple ways to evaluate chromosomes during execution of the NSGA-II
  - mean-value objectives
  - stochastic objectives
  - perfect information objectives

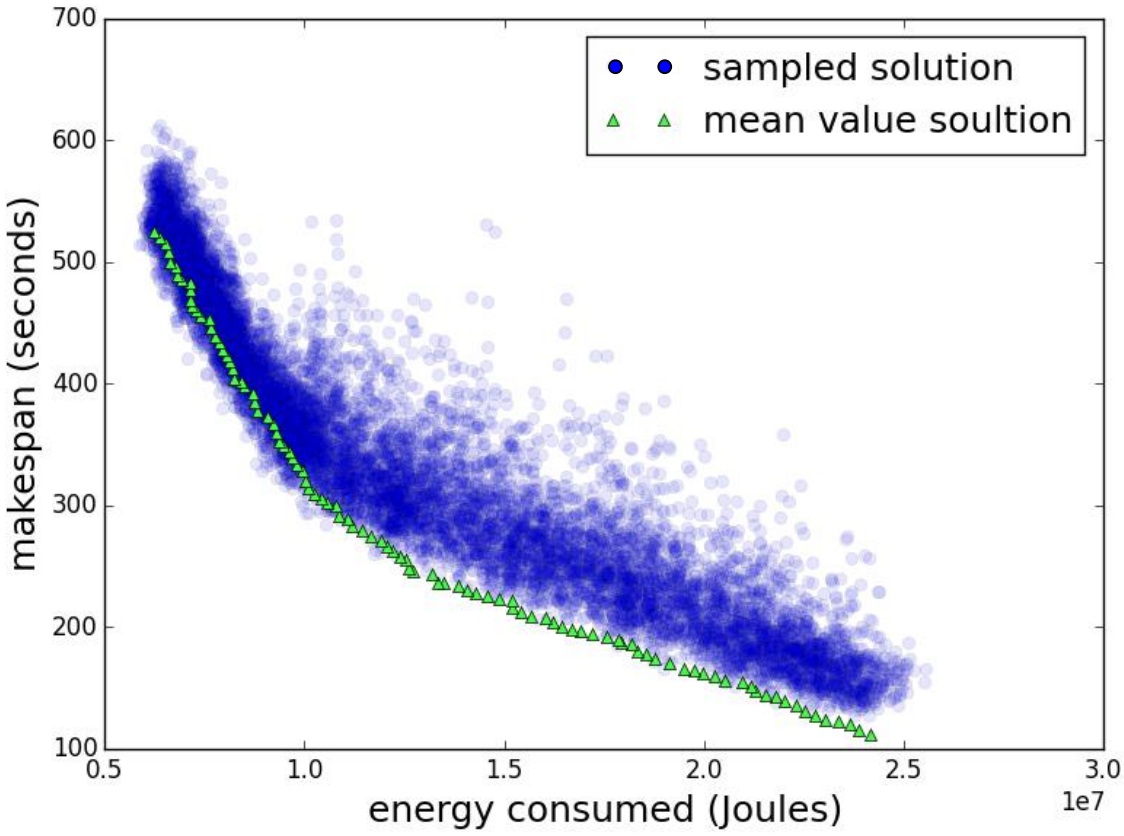
Mean Value Objectives

- at each iteration evaluate chromosomes using expected values of execution time and energy consumption
- the values for utility and energy that the chromosomes evaluate to are highly unlikely to actually be obtained



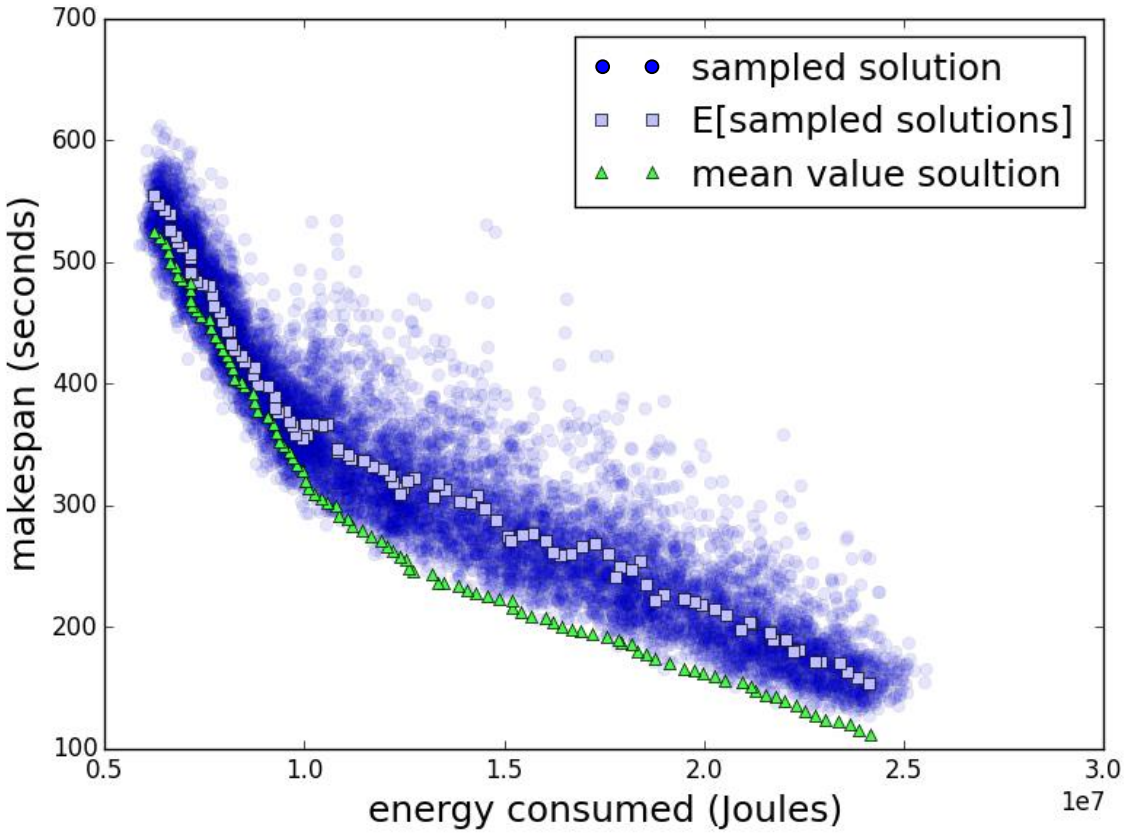
Mean Value Objectives

- at each iteration evaluate chromosomes using expected values of execution time and energy consumption
- the values for utility and energy that chromosomes evaluate to are highly unlikely to be obtained
- after algorithm has finished, realize each solution (triangle) *N* times by sampling the execution time distributions

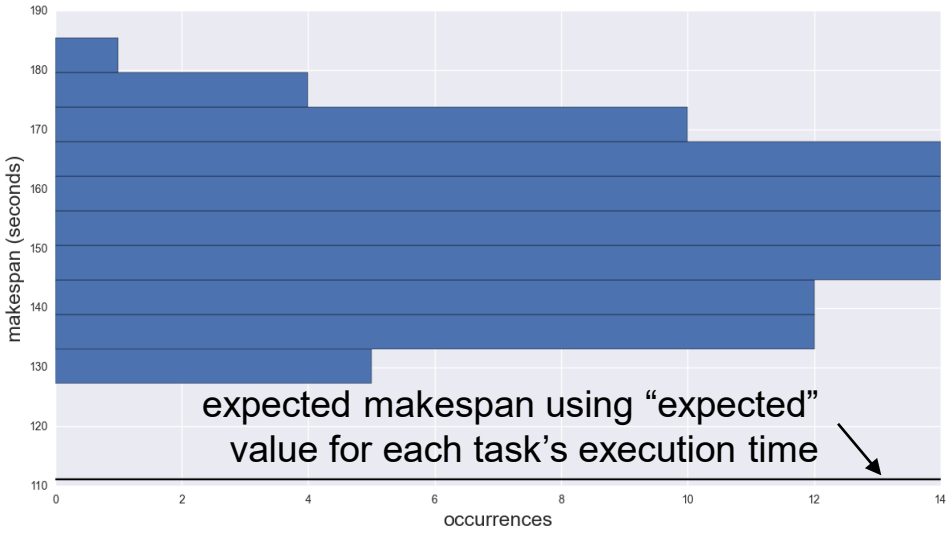


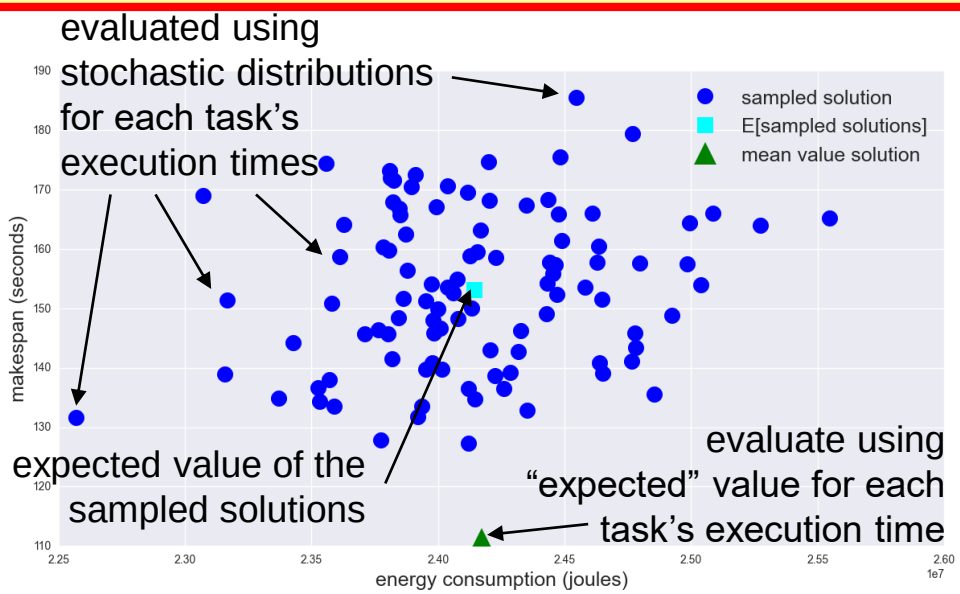
Mean Value Objectives

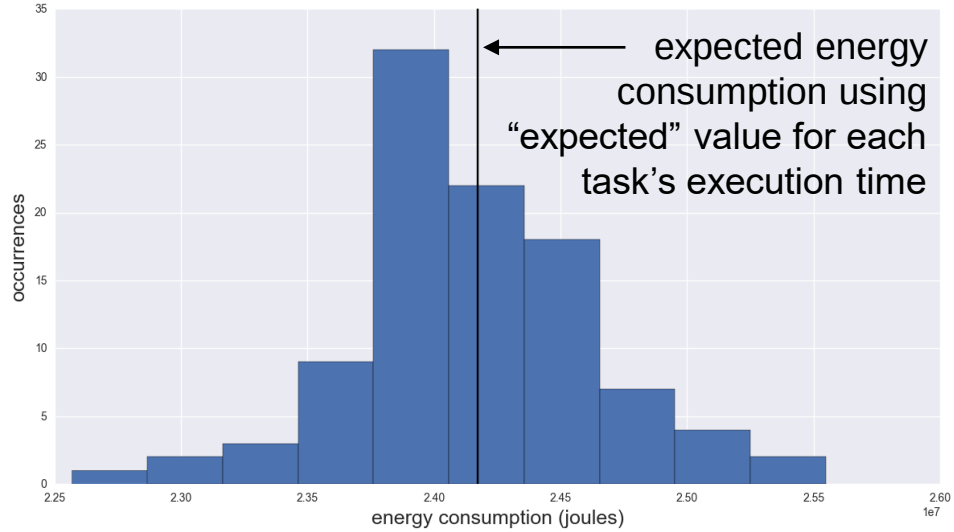
- at each iteration evaluate chromosomes using expected values of execution time and energy consumption
- the values for utility and energy that chromosomes evaluate to are highly unlikely to be obtained
- after algorithm has finished, realize each solution (triangle) *N* times by sampling the execution time distributions
- can find the expected value of sampled solutions for comparison purposes



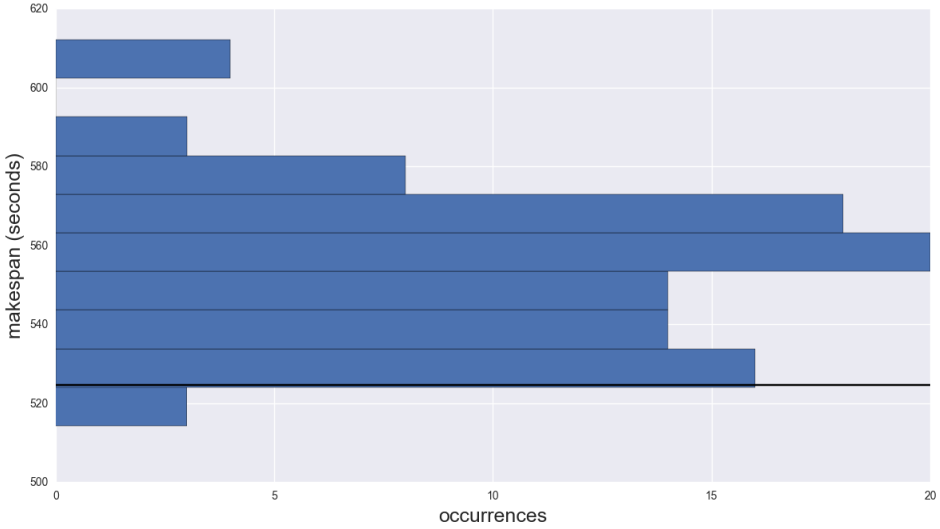
Minimum Makespan Resource Allocation

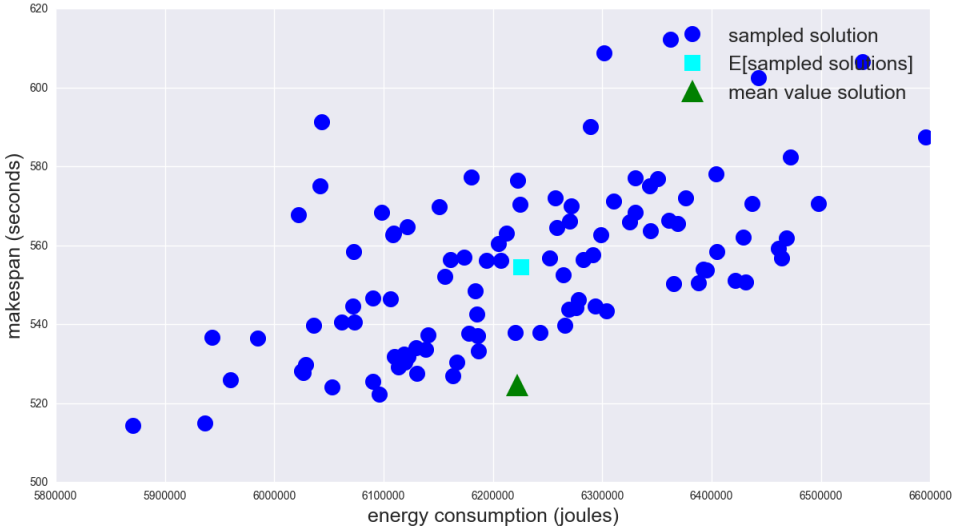


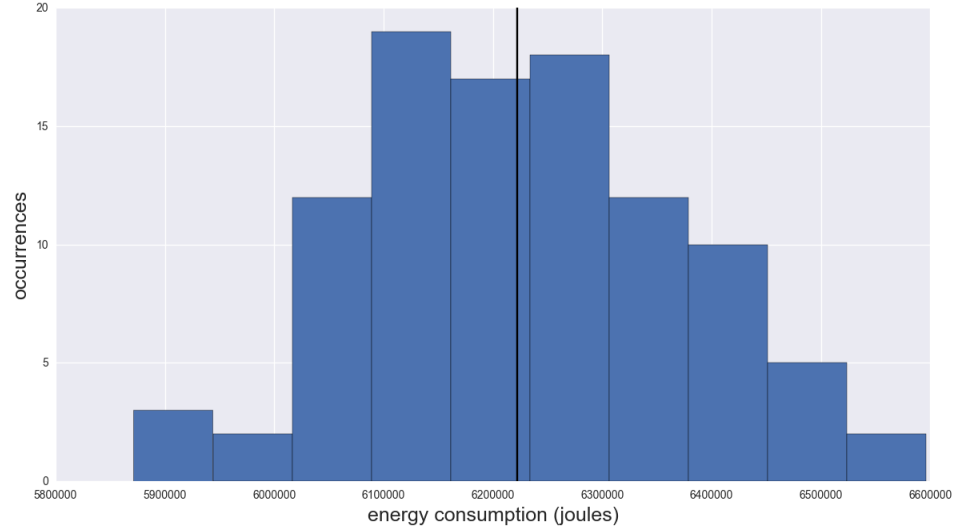




Minimum Energy Resource Allocation

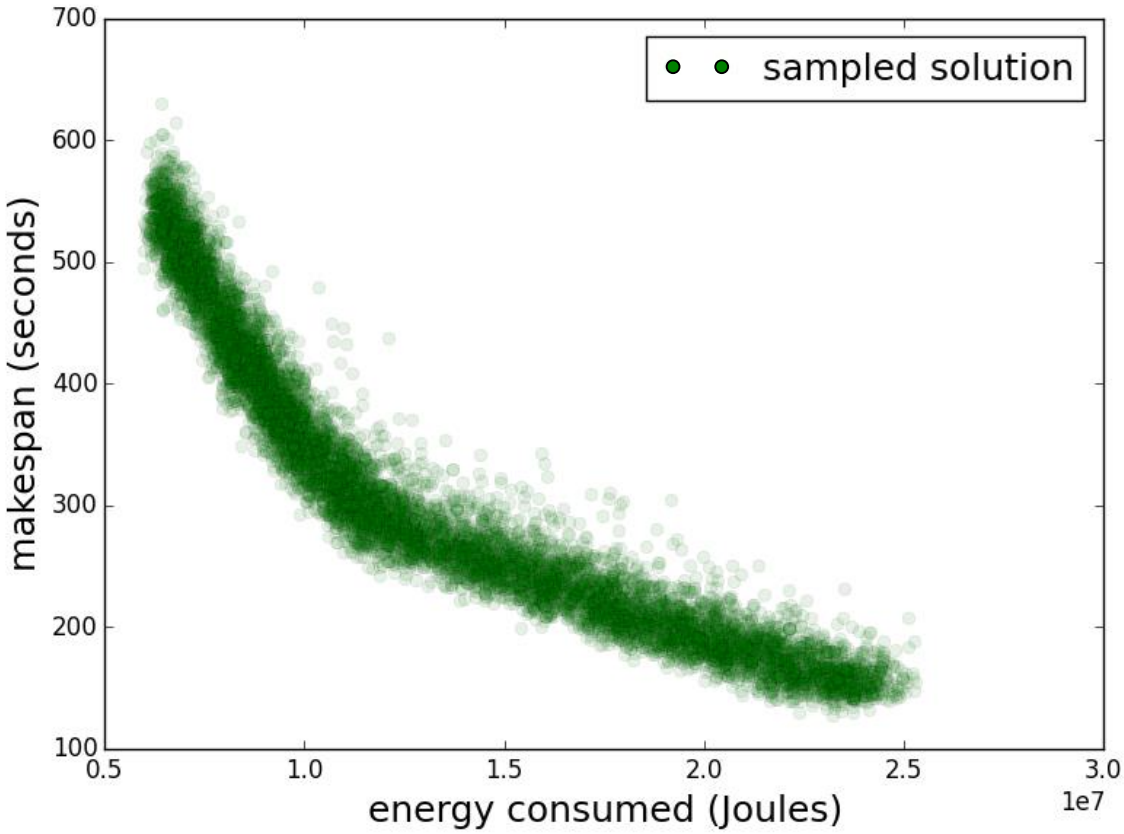






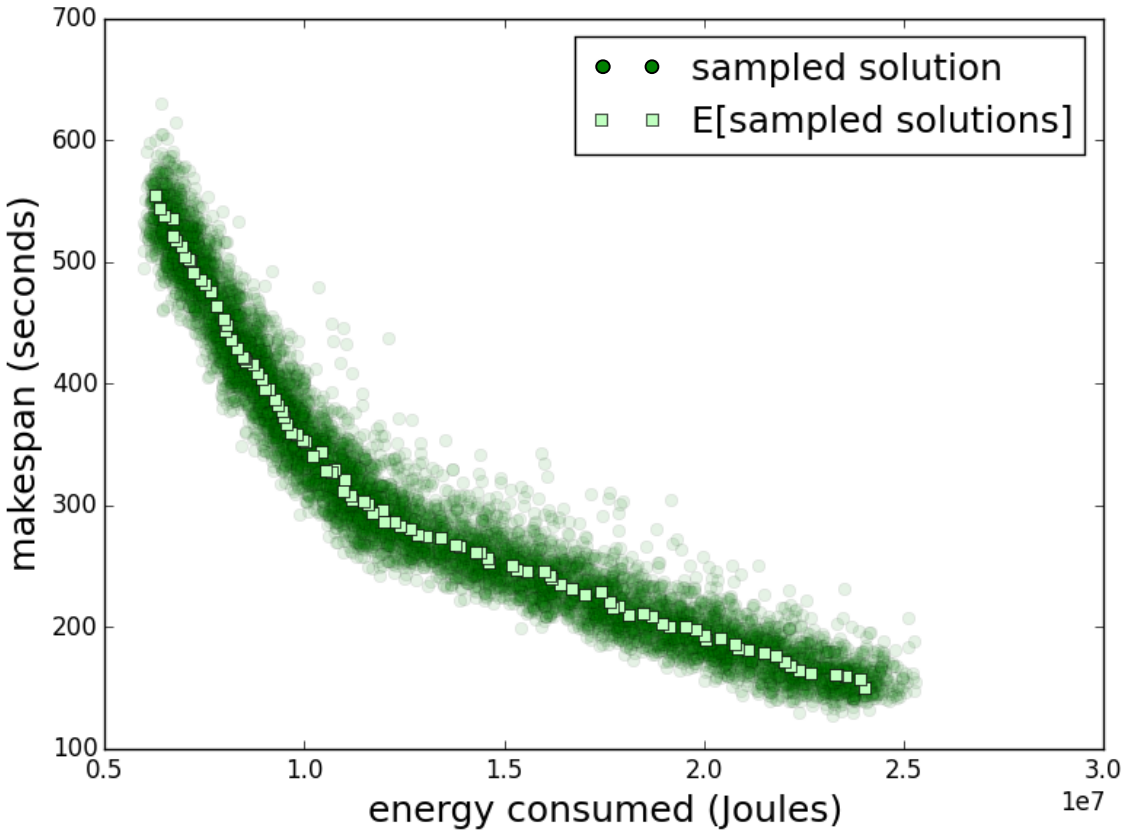
Stochastic Objectives

- at each iteration evaluate chromosomes using stochastic distributions for each task's execution time
  - every chromosome is evaluated *N* times



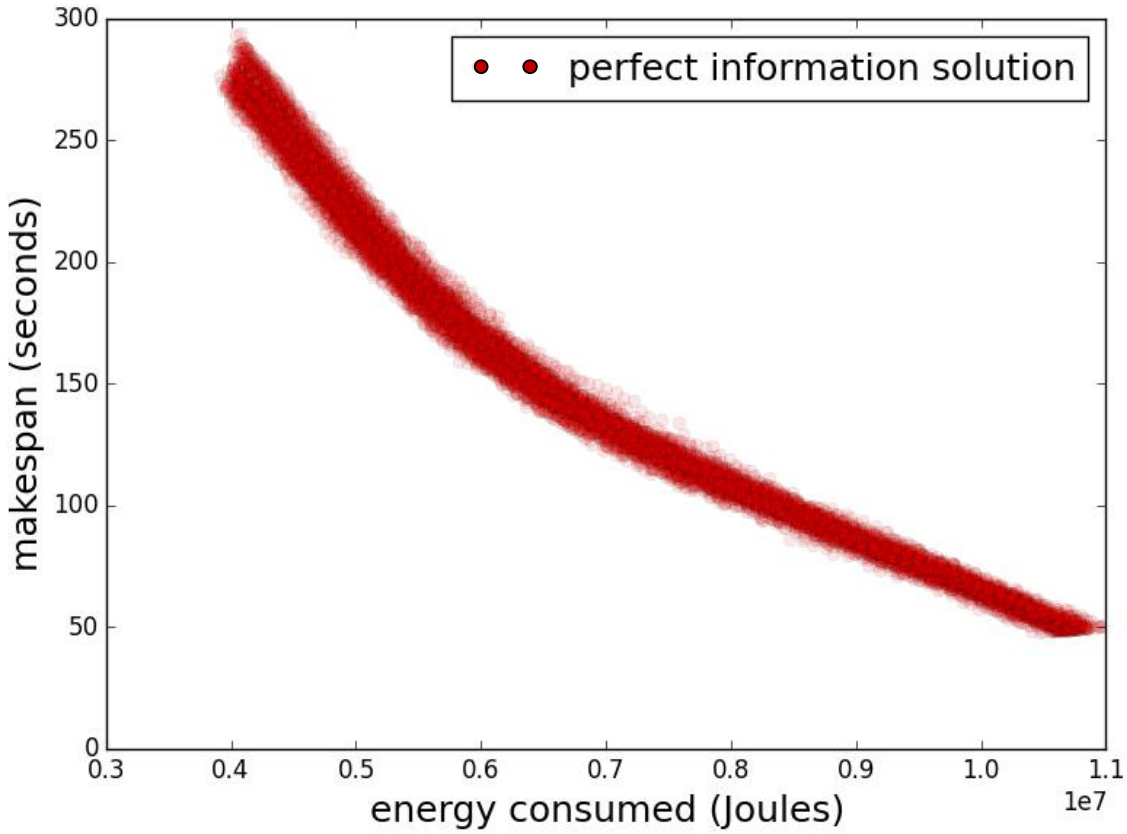
Stochastic Objectives

- at each iteration evaluate chromosomes using stochastic distributions for each task's execution time
  - every chromosome is evaluated *N* times
- the utility and energy values for a chromosome (used during the NSGA-II) are found by using a statistic on the sampled values
  - I'm using expected value
  - could use min, max, 1st/3rd quartile, etc.



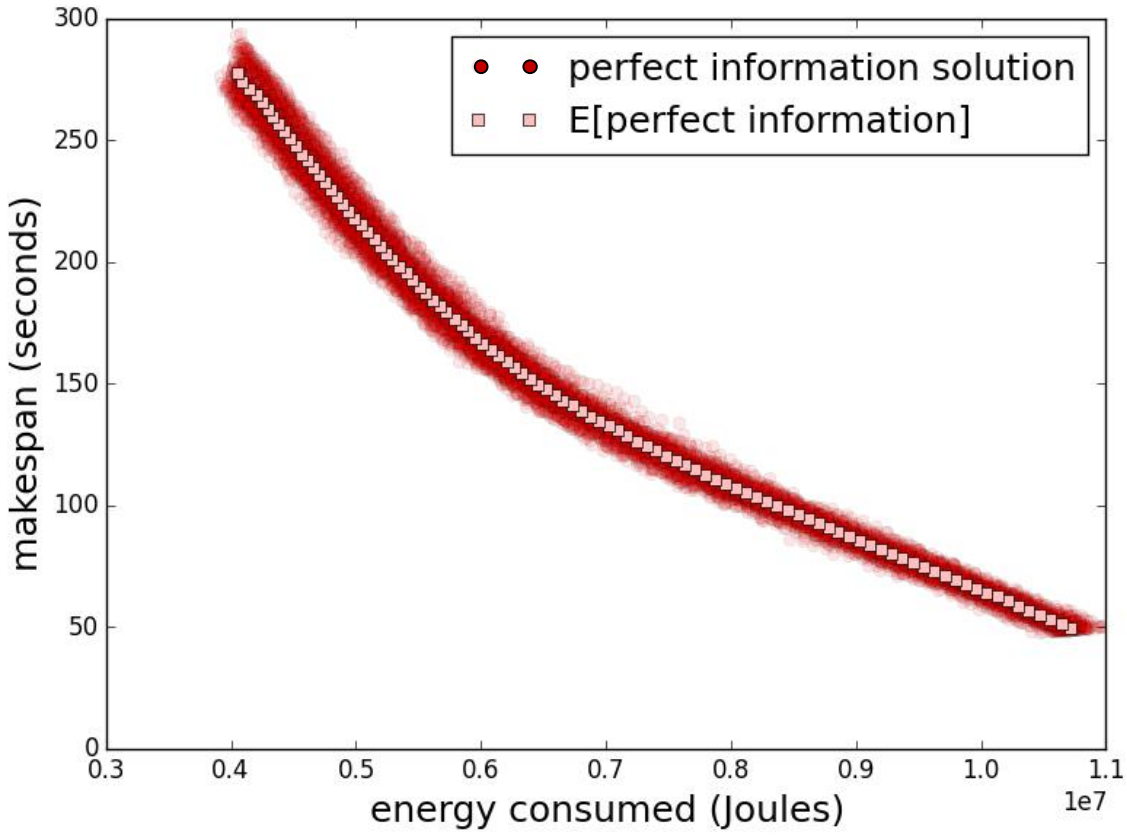
Perfect Information Objectives

- before executing the algorithm sample the distributions to find and store the execution times of each task on each machine
- at each iteration evaluate chromosomes using these predetermined values for task execution time
  - this information not available in real life
- generate and execute *N* scenarios



Perfect Information Objectives

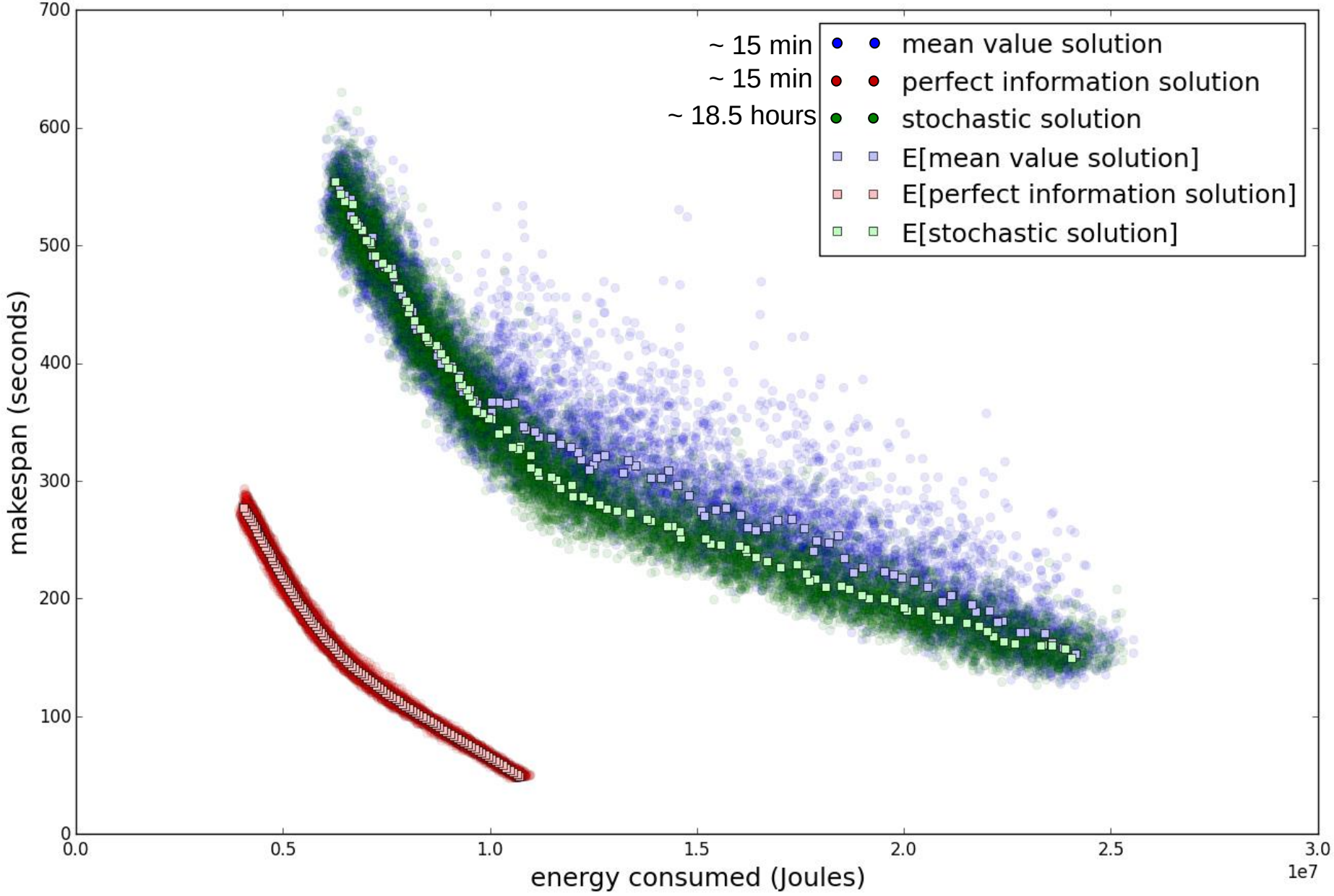
- before executing the algorithm sample the distributions to find and store the execution times of each task on each machine
- at each iteration evaluate chromosomes using these predetermined values for task execution time
  - this information not available in real life
- generate and execute *N* scenarios
- calculate the expected value of perfect information for comparison purposes



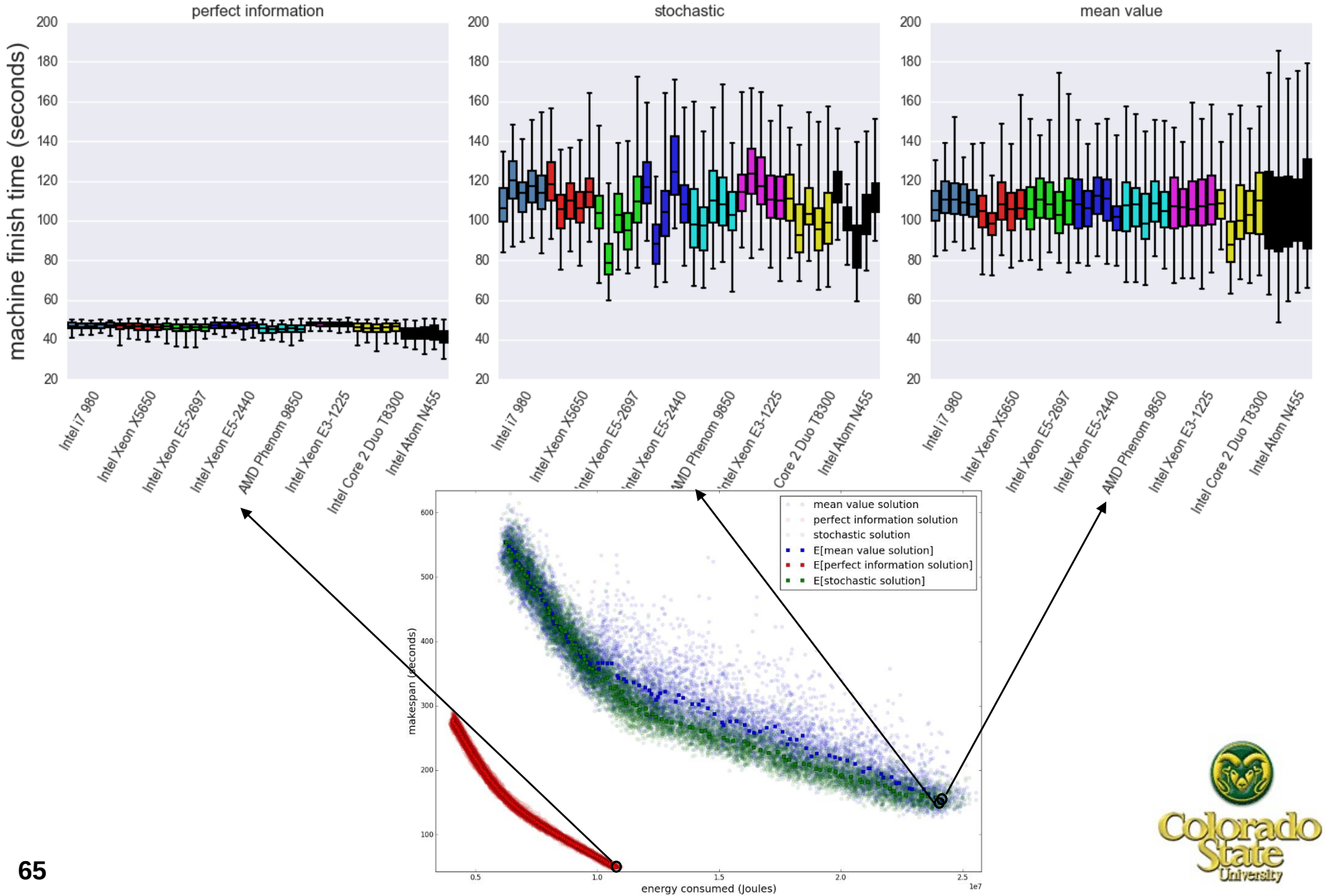
Simulations Setup

- 40 machines, 8 machine types, 5 machines per type
- 1,000 tasks one 8 task types
- 100 chromosomes in population
- 10,000 iterations
- stochastic objectives perform 100 evaluations per chromosome
- 100 perfect information scenarios generated and executed
- final mean value solutions evaluated 100 times

Perfect Information vs Stochastic vs Mean Value

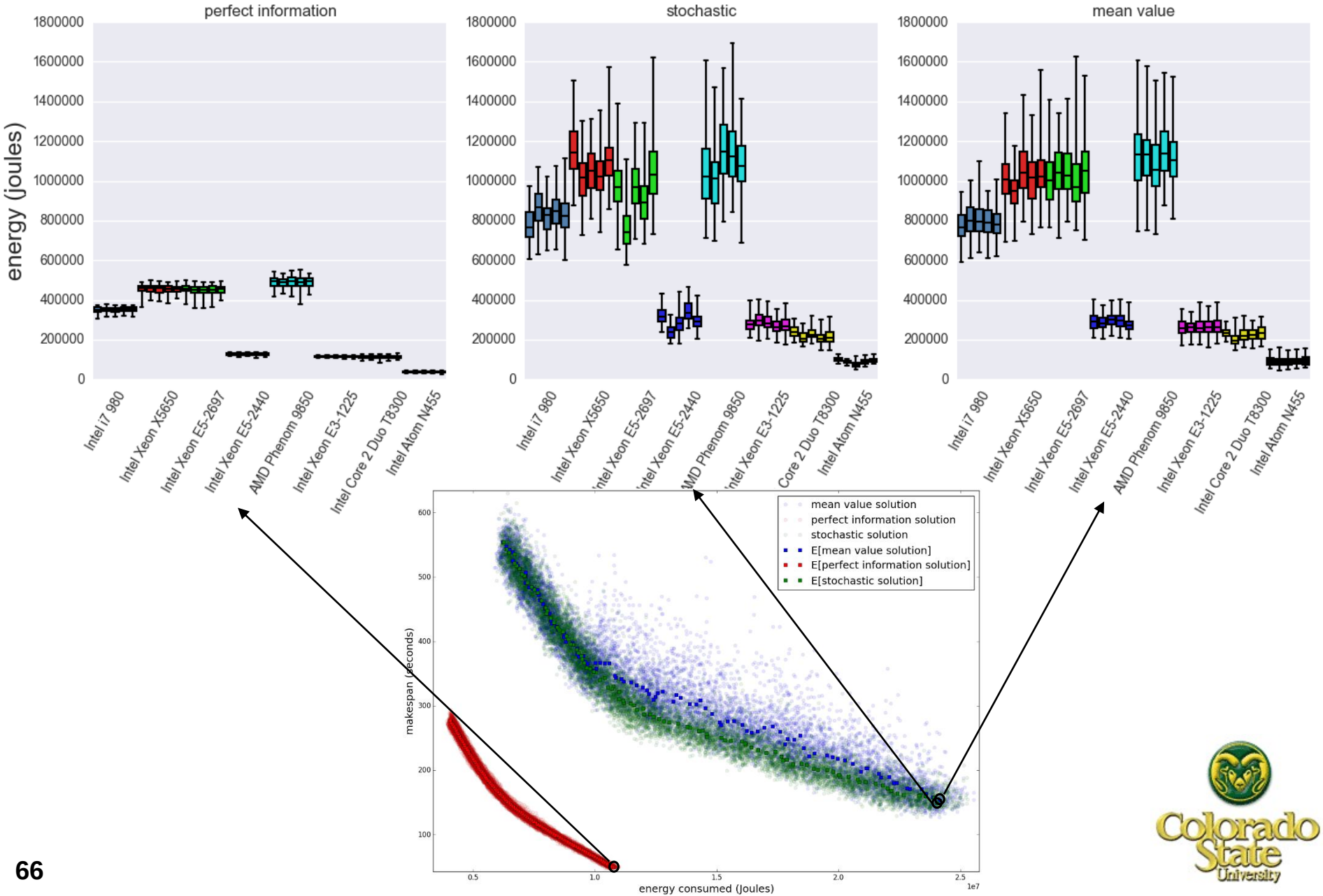


Execution Time Comparison – Minimum Makespan



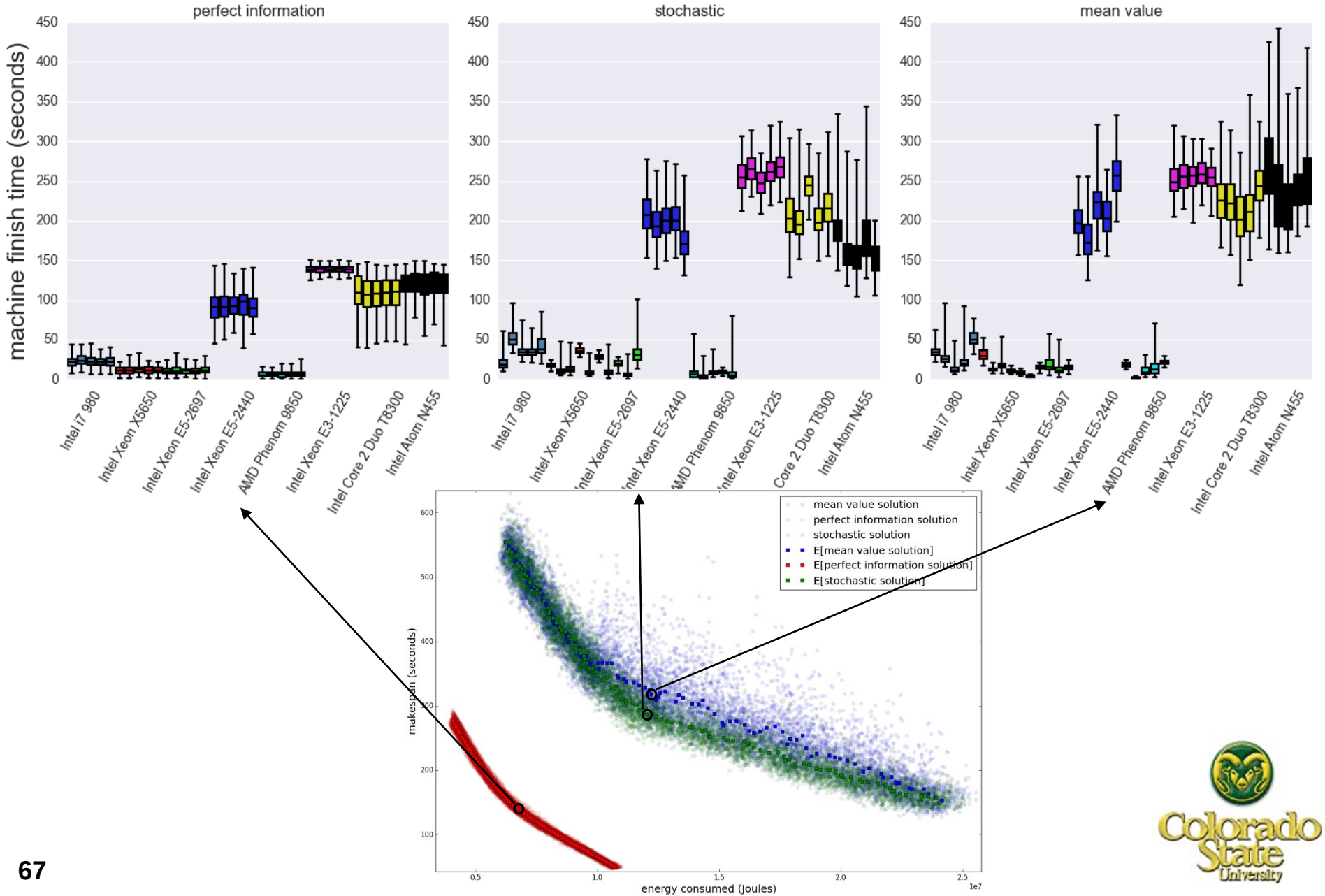
Colorado State University

Energy Comparison – Minimum Makespan



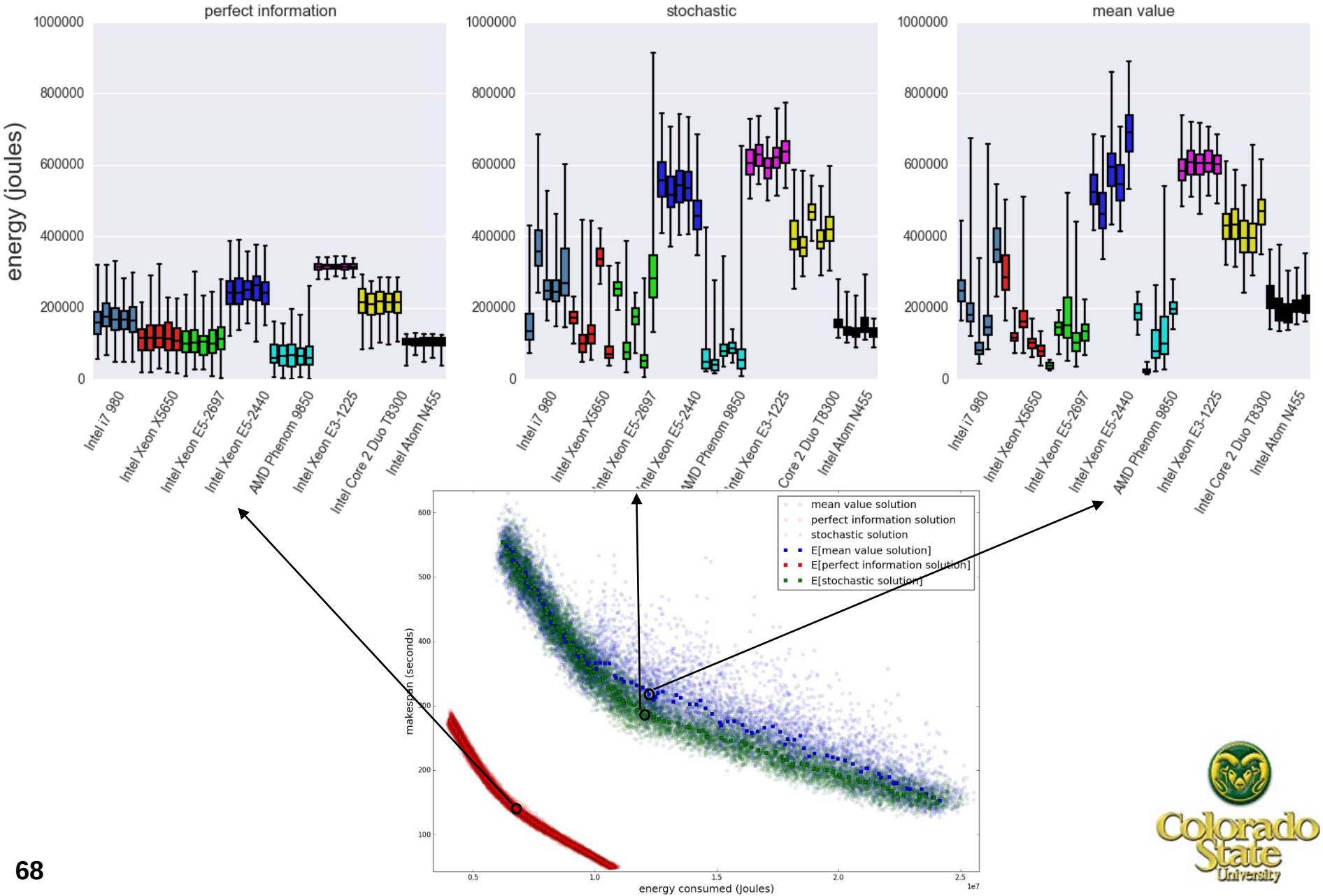
Colorado State University

Execution Time Comparison – Middle Chromosome



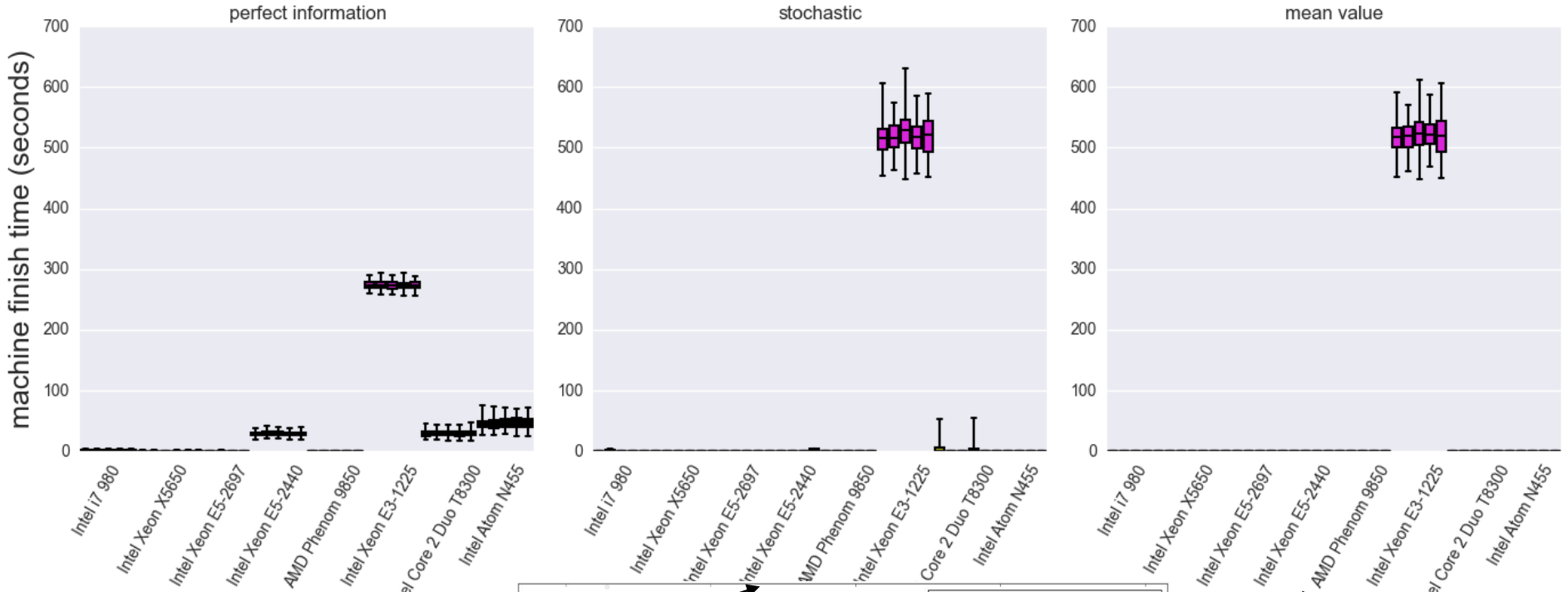
Colorado State University

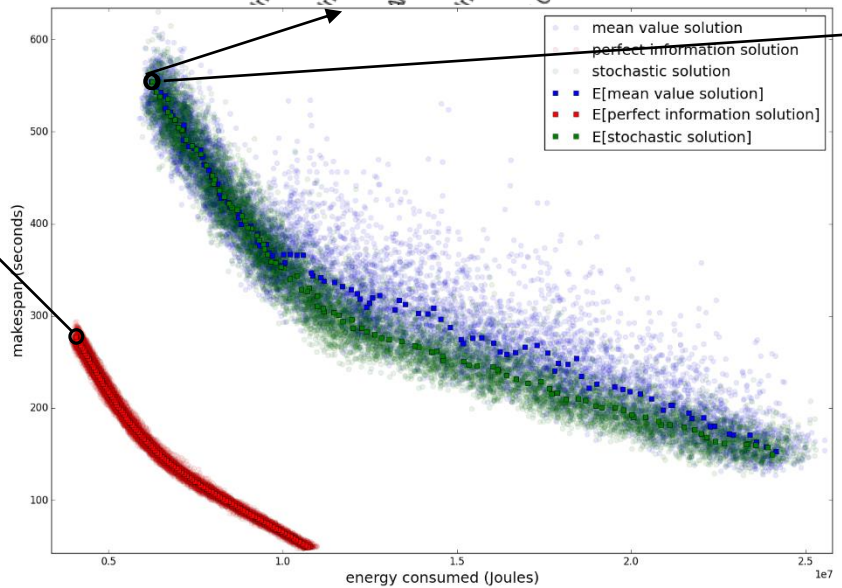
Energy Comparison – Middle Chromosome



Colorado State University

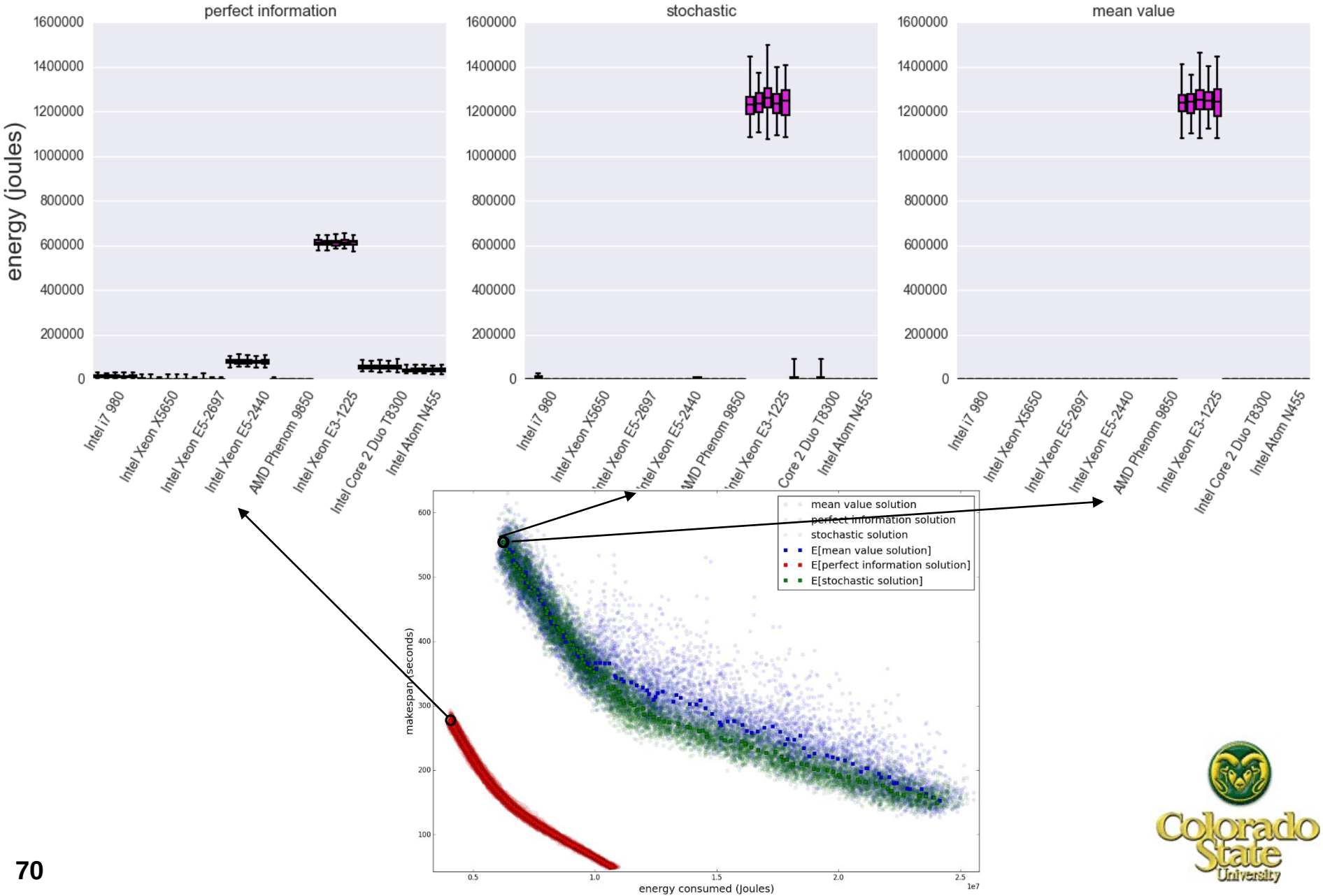
Execution Time Comparison – Min Energy





Colorado State University

Energy Comparison – Min Energy



Colorado State University

Contributions

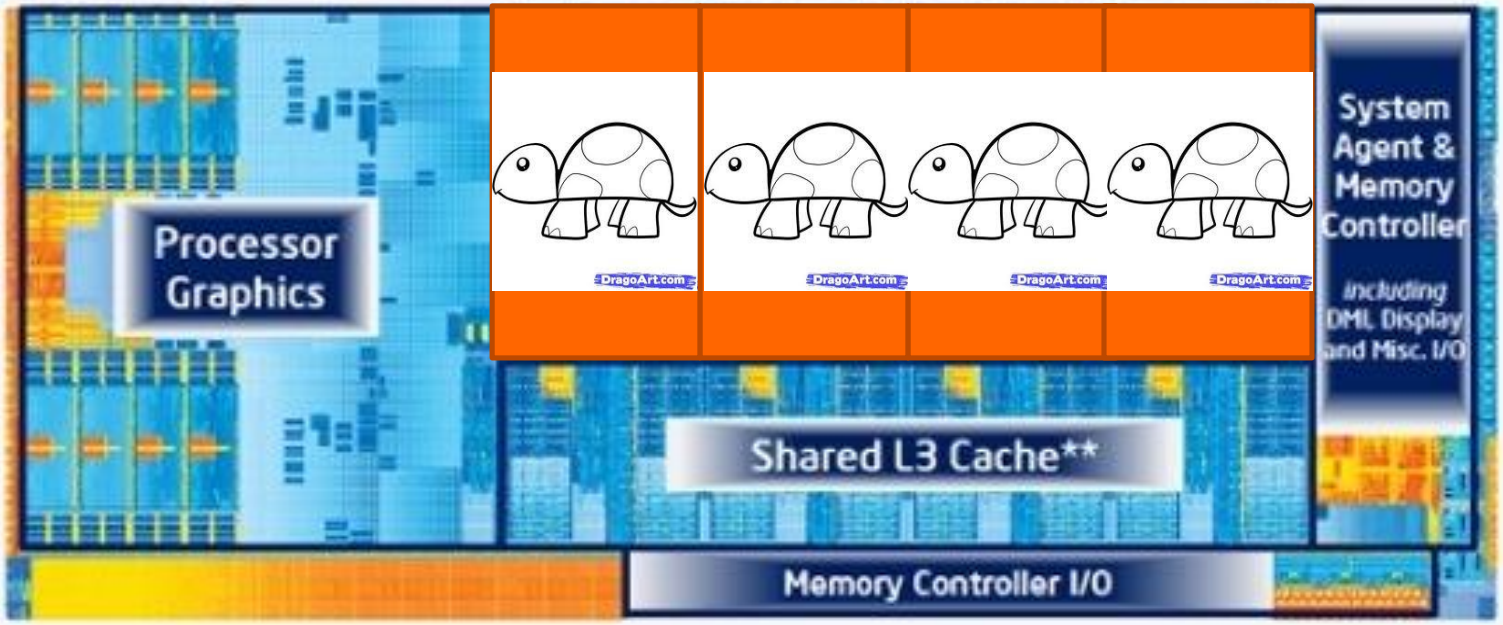
- designed a framework to gather stochastic data for multiple tasks and machines
  - datasets available on my website
- designed and implemented stochastic versions of the performance and energy objectives
- introduced the concept of fuzzy Pareto fronts
- conducted experiments to analyze what the value of using perfect information and stochastic information is compared to using only the mean values
- using stochastic values provide more knowledge into the variation a given resource allocation may have
  - can lead to a study of how robust a resource allocation is against uncertainties in execution time

Outline

- Chapter 1: introduction
- Chapter 2: dynamic utility maximization
- dynamic energy-aware resource allocation
  - Chapter 3: utility maximization with an energy constraint
  - Chapter 4: energy constrained utility maximization for different queuing models
- multi-objective optimization
  - Chapter 5: deterministic utility vs. energy
  - Chapter 6: deterministic makespan vs. energy
  - Chapter 7: stochastic utility/makespan vs. energy
- Chapter 8: application co-location in multicore systems
- Chapter 9: future work

Motivation

- task interference from cache contention in multicore processors causes degradation in system performance
- this is a common problem in multicore based parallel and distributed systems



Colorado State University

Problem Statement and Goals

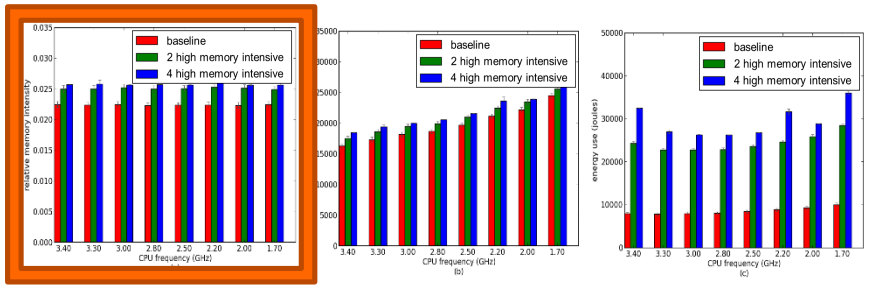
- **problem statement**
  - what are the effects of memory interference and P-states on energy usage and task execution time
- better understand effects of co-location memory interference on task execution and energy use
- mitigate negative effects of memory interference (from cache contention) using cache- and energy-aware task co-location across cores
- improving system energy efficiency
  - selecting P-state “sweet spot”
  - proper task co-locations

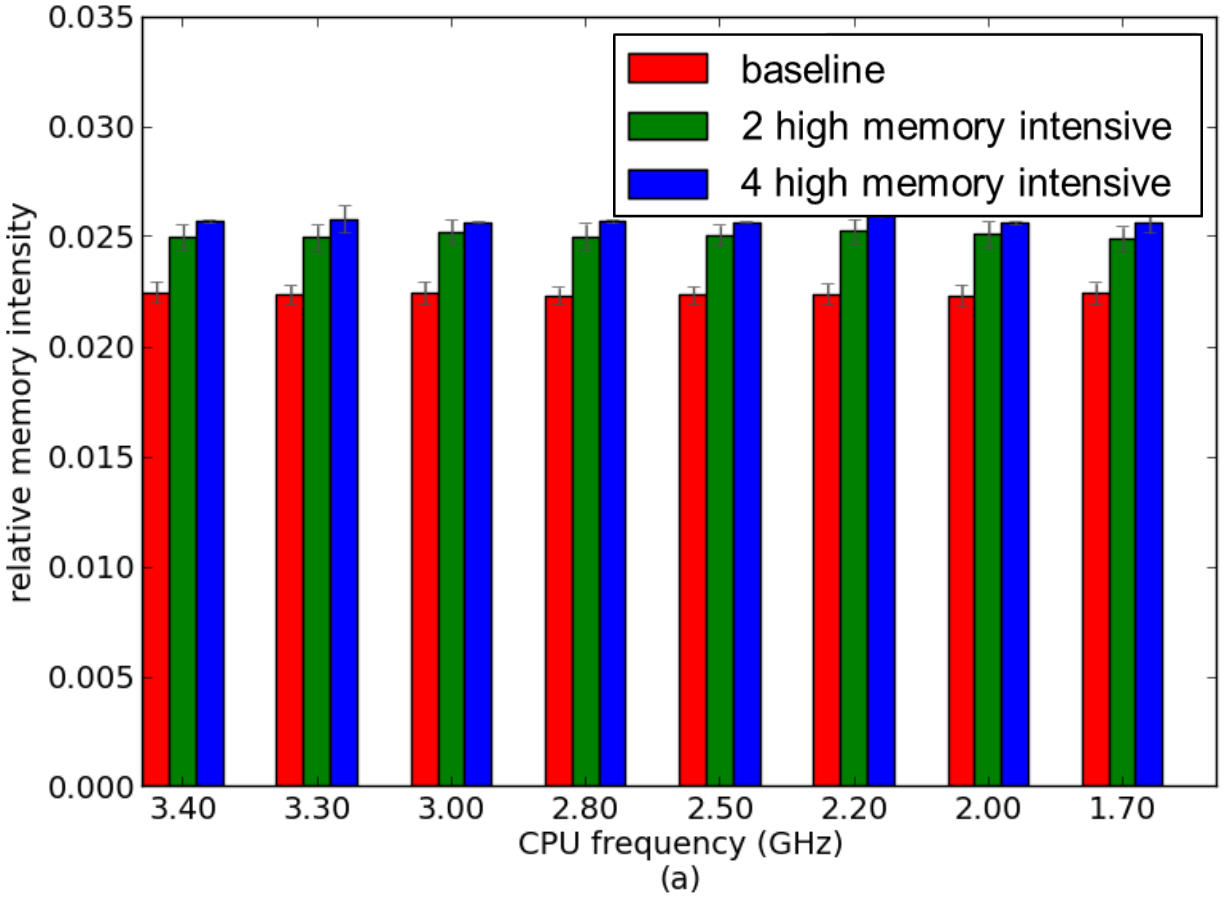
Experiments

- data was collected on a 64-bit Intel i7 3770 quad-core processor
  - all four cores shared an L3 cache
  - P-states allow the frequency to vary from 3.40 GHz – 1.70 GHz
- tasks were classified based on their memory intensities
  - $\text{memory intensity} = \dfrac{\text{last-level cache misses}}{\text{total instructions executed}}$

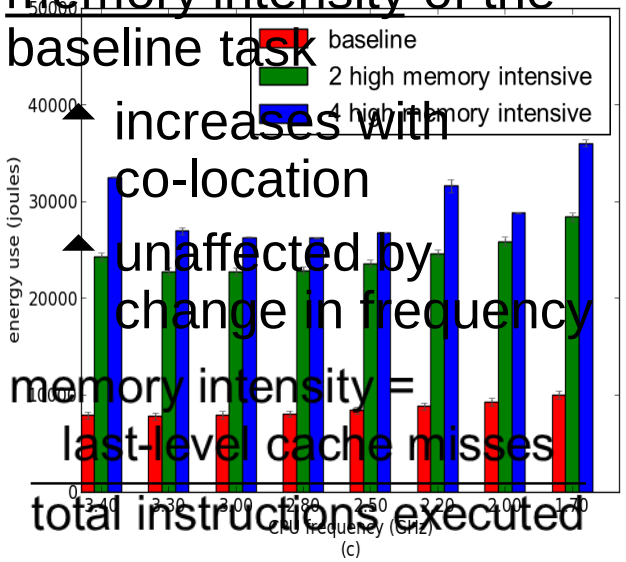
Interference Results for High Memory Intensity Tasks

- these results only for the baseline
  - showing a high memory intensity task
- each test averaged over 9 runs





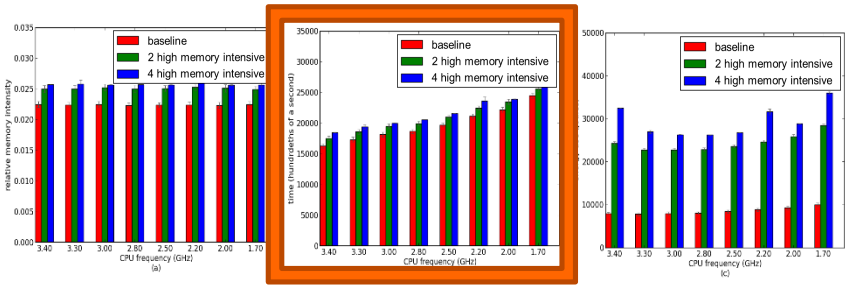
- memory intensity of the baseline task
  - increases with co-location
  - unaffected by change in frequency
- memory intensity = last-level cache misses / total instructions executed

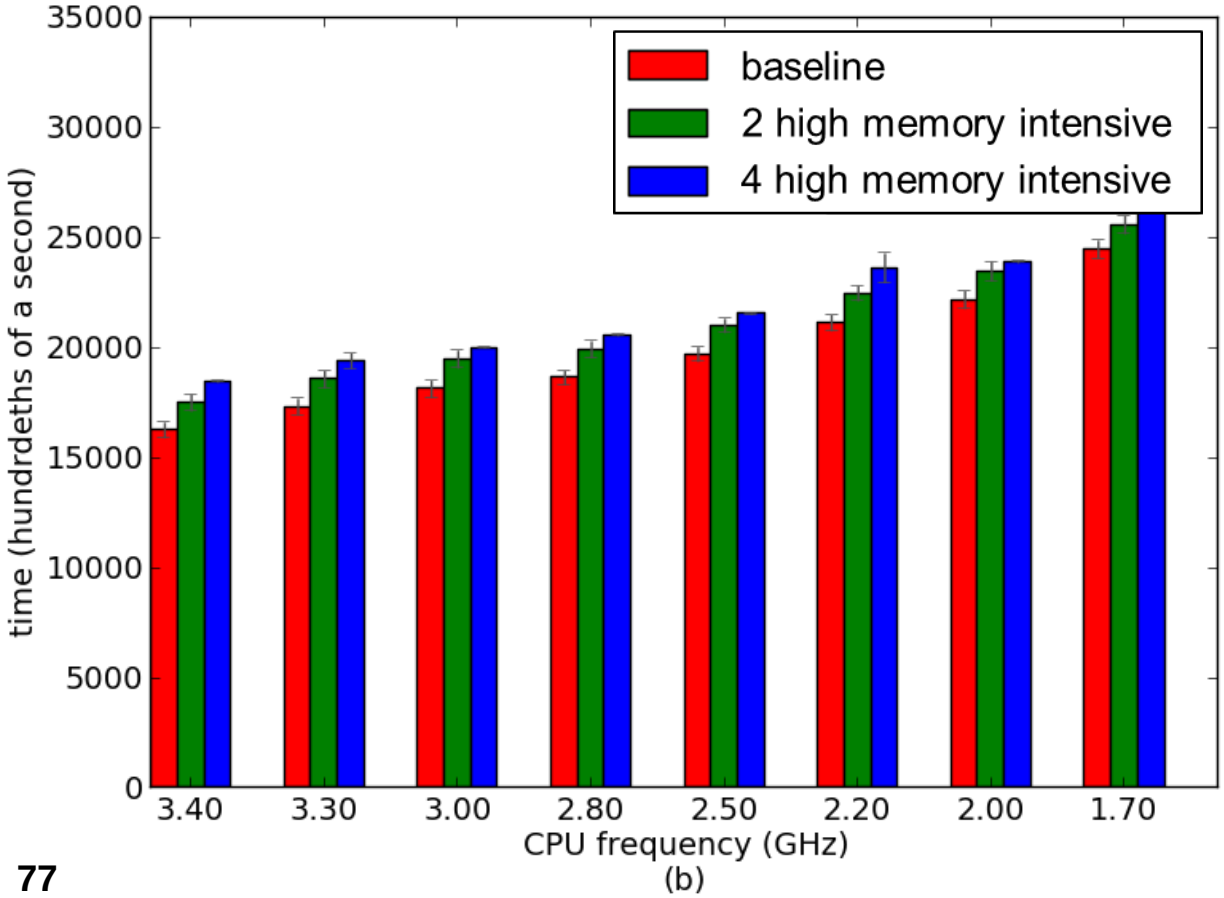


Colorado State University

Interference Results for High Memory Intensity Tasks

- these results only for the baseline
  - showing a high memory intensity task
- each test averaged over 9 runs



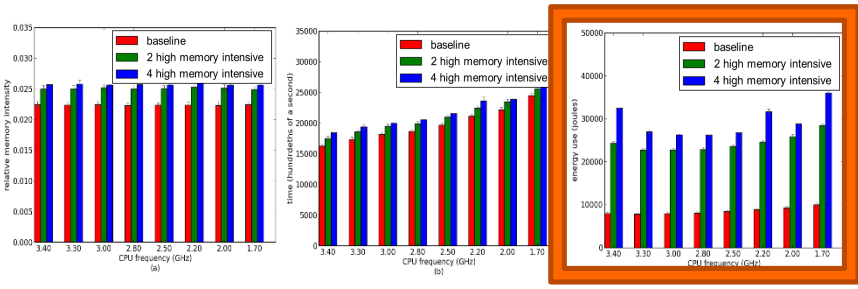


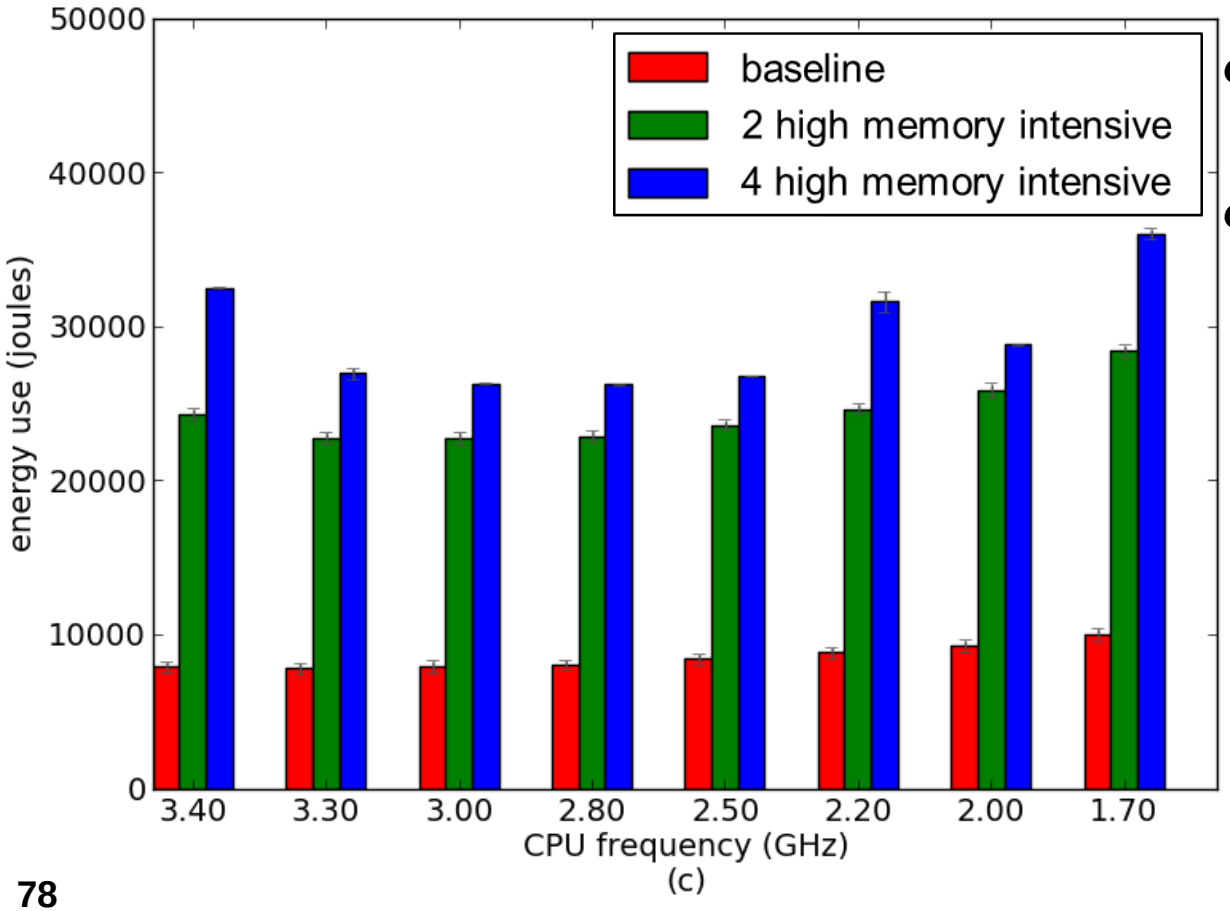
- execution time of the baseline task
  - increases with co-location
  - increases with slower frequency

Colorado State University

Interference Results for High Memory Intensity Tasks

- these results only for the baseline
  - showing a high memory intensity task
- each test averaged over 9 runs





- total energy = static + dynamic energy
- <u>total energy use</u> of the baseline task
  - increases with co-location
  - P-state leads to "sweet spot"

Colorado State University

Conclusions

- high memory intensive tasks have a greater effect on the performance of other co-located tasks
  - want to co-locate high memory intensive tasks with low memory intensive tasks
- co-locating multiple tasks can save on overall energy (as opposed to running each task independently)
  - the tasks share the use of static power
  - up to 25.8% savings for two high memory intensive tasks
  - up to 47.7% savings for four high memory intensive tasks
- P-states can be used to find a total energy “sweet spot”

Outline

- Chapter 1: introduction
- Chapter 2: dynamic utility maximization
- dynamic energy-aware resource allocation
  - Chapter 3: utility maximization with an energy constraint
  - Chapter 4: energy constrained utility maximization for different queuing models
- multi-objective optimization
  - Chapter 5: deterministic utility vs. energy
  - Chapter 6: deterministic makespan vs. energy
  - Chapter 7: stochastic utility/makespan vs. energy
- Chapter 8: application co-location in multicore systems
- Chapter 9: future work

Future (and Current) Work

- incorporate cost of energy by simulating pricing markets
  - flat-rate markets
  - tiered markets
  - dynamic markets (real-time pricing)
- incorporate parallel jobs
- incorporate communication times and data movement costs
- study “worker pool” environment
  - machines ask for new tasks as needed
  - no set mapping events
- improvements to NSGA-II
  - iterative crowding distance
  - “graveyard”
- chromosome representation using task and machines types

Acknowledgments

- This work was supported by Oak Ridge National Laboratory and their Extreme Scale Systems Center, and by the National Science Foundation.

Thank You
Questions?