REF: Resource Elasticity Fairness with Sharing Incentives for Multiprocessors

Seyed Majid Zahedi, Benjamin C. Lee
zahedi@cs.duke.edu, benjamin.c.lee@duke.edu
Motivation

- Alvy and Ben are working on ASPLOS papers
Motivation

- Alvy and Ben are working on ASPLOS papers
- Each has $10K to buy clusters
Motivation

- Alvy and Ben are working on ASPLOS papers
- Each has $10K to buy clusters
- Alvy works on accelerators
Motivation

- Alvy and Ben are working on ASPLOS papers
- Each has $10K to buy clusters
- Alvy works on accelerators
- Ben works on memory systems
Strategic Behavior

- Alvy and Ben are strategic
Strategic Behavior

- Alvy and Ben are strategic
- Which is better?
  - Small, separate clusters
  - Large, shared cluster

[www.websavers.org]
Strategic Behavior

- Alvy and Ben are strategic
- Which is better?
  - Small, separate clusters
  - Large, shared cluster
- Suppose Alvy and Ben share

[www.websavers.org]
Strategic Behavior

- Alvy and Ben are strategic
- Which is better?
  - Small, separate clusters
  - Large, shared cluster
- Suppose Alvy and Ben share
  - Is allocation fair?
Strategic Behavior

- Alvy and Ben are strategic
- Which is better?
  - Small, separate clusters
  - Large, shared cluster
- Suppose Alvy and Ben share
  - Is allocation fair?
  - Is lying beneficial?
Conventional Wisdom in Computer Architecture

- Users must share
Conventional Wisdom in Computer Architecture

- Users must share
  - Overlooks strategic behavior
Conventional Wisdom in Computer Architecture

- Users must share
  - Overlooks strategic behavior
- Fairness policy is equal slowdown
Conventional Wisdom in Computer Architecture

- Users must share
  - Overlooks strategic behavior
- Fairness policy is equal slowdown
  - Fails to encourage envious users to share
Conventional Wisdom in Computer Architecture

- Users must share
  - Overlooks strategic behavior
- Fairness policy is equal slowdown
  - Fails to encourage envious users to share
- Heuristic mechanisms enforce equal slowdown
Conventional Wisdom in Computer Architecture

- Users must share
  - Overlooks strategic behavior
- Fairness policy is equal slowdown
  - Fails to encourage envious users to share
- Heuristic mechanisms enforce equal slowdown
  - Fail to give provable guarantees
"If an allocation is both equitable and Pareto efficient, it is fair."
[Varian, Journal of Economic Theory (1974)]
Rethinking Fairness

"If an allocation is both equitable and Pareto efficient, ... it is fair." [Varian, Journal of Economic Theory (1974)]

• Equity
  • Evaluating others’ and own position on equal terms
Rethinking Fairness

“If an allocation is both equitable and Pareto efficient, … it is fair.” [Varian, Journal of Economic Theory (1974)]

- Equity
  - Evaluating others’ and own position on equal terms
  - No user envies another’s allocation (i.e., envy-freeness)
Rethinking Fairness

"If an allocation is both equitable and Pareto efficient, ... it is fair.” [Varian, Journal of Economic Theory (1974)]

- Equity
  - Evaluating others’ and own position on equal terms
  - No user envies another’s allocation (i.e., envy-freeness)

- Pareto Efficiency
  - No other allocation improves utility without harming others
Resource Elasticity Fairness (REF)

REF is an allocation mechanism that guarantees game-theoretic desiderata for shared chip multiprocessors.

- Envy-Free (EF): No user envies another's allocation.
- Pareto-Efficient (PE): No other allocation improves utility without harming others.
- Sharing Incentives (SI): Users perform no worse than under equal division.
- Strategy-Proof (SP): No user benefits from lying.
Resource Elasticity Fairness (REF)

REF is an allocation mechanism that guarantees game-theoretic desiderata for shared chip multiprocessors

- **Envy-Free (EF)**
  No user envies another’s allocation
Resource Elasticity Fairness (REF)

REF is an allocation mechanism that guarantees game-theoretic desiderata for shared chip multiprocessors

- **Envy-Free (EF)**
  No user envies another’s allocation

- **Pareto-Efficient (PE)**
  No other allocation improves utility without harming others
Resource Elasticity Fairness (REF)

REF is an allocation mechanism that guarantees game-theoretic desiderata for shared chip multiprocessors

- **Envy-Free (EF)**
  No user envies another’s allocation

- **Pareto-Efficient (PE)**
  No other allocation improves utility without harming others

- **Sharing Incentives (SI)**
  Users perform no worse than under equal division
Resource Elasticity Fairness (REF)

REF is an allocation mechanism that guarantees game-theoretic desiderata for shared chip multiprocessors

- **Envy-Free (EF)**
  No user envies another’s allocation

- **Pareto-Efficient (PE)**
  No other allocation improves utility without harming others

- **Sharing Incentives (SI)**
  Users perform no worse than under equal division

- **Strategy-Proof (SP)**
  No user benefits from lying
Three Main Steps

- Defining utility functions
Three Main Steps

- Defining utility functions
- Identifying conditions for fairness
Three Main Steps

• Defining utility functions
• Identifying conditions for fairness
• Devising mechanism for finding allocations
Utility Functions in Computer Architecture

- Model diminishing marginal returns (DMR)
Utility Functions in Computer Architecture

- Model diminishing marginal returns (DMR)
  - Data locality affects returns from cache sizing
Utility Functions in Computer Architecture

• Model diminishing marginal returns (DMR)
  • Data locality affects returns from cache sizing
  • Memory intensity affects returns from bandwidth
Utility Functions in Computer Architecture

- Model diminishing marginal returns (DMR)
  - Data locality affects returns from cache sizing
  - Memory intensity affects returns from bandwidth
  - Serial portion affects returns from cores
Utility Functions in Computer Architecture

- Model diminishing marginal returns (DMR)
  - Data locality affects returns from cache sizing
  - Memory intensity affects returns from bandwidth
  - Serial portion affects returns from cores
- Model substitution effects
Utility Functions in Computer Architecture

- Model diminishing marginal returns (DMR)
  - Data locality affects returns from cache sizing
  - Memory intensity affects returns from bandwidth
  - Serial portion affects returns from cores
- Model substitution effects
  - Complementary resources can be traded
Utility Functions in Computer Architecture

- Model diminishing marginal returns (DMR)
  - Data locality affects returns from cache sizing
  - Memory intensity affects returns from bandwidth
  - Serial portion affects returns from cores
- Model substitution effects
  - Complementary resources can be traded
  - E.g., cache size and memory bandwidth
Cobb-Douglas Utility

\[ u(x) = \prod_{r=1}^{R} x_r^{\alpha_r} \]

- \( u \) utility (e.g., performance)
Cobb-Douglas Utility

\[ u(x) = \prod_{r=1}^{R} x_r^{\alpha_r} \]

- \( u \) utility (e.g., performance)
- \( x_r \) allocation for resource \( r \)
Cobb-Douglas Utility

\[ u(x) = \prod_{r=1}^{R} x_r^{\alpha_r} \]

- \( u \) utility (e.g., performance)
- \( x_r \) allocation for resource \( r \)
- \( \alpha_r \) elasticity for resource \( r \)
Cobb-Douglas Utility

\[ u(x) = \prod_{r=1}^{R} x_r^{\alpha_r} \]

- \( u \): utility (e.g., performance)
- \( x_r \): allocation for resource \( r \)
- \( \alpha_r \): elasticity for resource \( r \)

- Cobb-Douglas fits preferences in computer architecture
Cobb-Douglas Utility

$$u(x) = \prod_{r=1}^{R} x_r^{\alpha_r}$$

- $u$: utility (e.g., performance)
- $x_r$: allocation for resource $r$
- $\alpha_r$: elasticity for resource $r$

- Cobb-Douglas fits preferences in computer architecture
- Exponents introduce non-linearity which captures DMR
Cobb-Douglas Utility

\[ u(x) = \prod_{r=1}^{R} x_r^{\alpha_r} \]

- \( u \): utility (e.g., performance)
- \( x_r \): allocation for resource \( r \)
- \( \alpha_r \): elasticity for resource \( r \)

- Cobb-Douglas fits preferences in computer architecture
- Exponents introduce non-linearity which captures DMR
- Products model substitution effects
Example Utilities

\[ u_1 = x_1^{0.6}y_1^{0.4} \quad u_2 = x_2^{0.2}y_2^{0.8} \]
Example Utilities

\[ u_1 = x_1^{0.6} y_1^{0.4} \quad u_2 = x_2^{0.2} y_2^{0.8} \]

\( u_1, u_2 \) utilities derived from performance measurements
Example Utilities

\[ u_1 = x_1^{0.6} y_1^{0.4} \quad u_2 = x_2^{0.2} y_2^{0.8} \]

- \(u_1, u_2\) utilities derived from performance measurements
- \(x_1, x_2\) allocated memory bandwidth for users 1, 2
Example Utilities

\[ u_1 = x_1^{0.6} y_1^{0.4} \quad u_2 = x_2^{0.2} y_2^{0.8} \]

- \( u_1, u_2 \) utilities derived from performance measurements
- \( x_1, x_2 \) allocated memory bandwidth for users 1, 2
- \( y_1, y_2 \) allocated cache size for users 1, 2
Three Main Steps

- Defining utility functions
- Identifying conditions for fairness
- Devising mechanism for finding allocations
Possible Allocations

- 2 users
- 12MB cache
- 24GB/s bandwidth
Envy-Free (EF) Allocations

- Identify EF allocations for each user

- \( u_1(A_1) \geq u_1(A_2) \)

- \( u_2(A_2) \geq u_2(A_1) \)
Envy-Free (EF) Allocations

- Identify EF allocations for each user
  - \( u_1(A_1) \geq u_1(A_2) \)
  - \( u_2(A_2) \geq u_2(A_1) \)
Pareto-Efficient (PE) Allocations

No other allocation improves utility without harming others
Pareto-Efficient (PE) Allocations

No other allocation improves utility without harming others

- Indifference curve: allocations that give same utility
Pareto-Efficient (PE) Allocations

No other allocation improves utility without harming others

- Indifference curve: allocations that give same utility
Pareto-Efficient (PE) Allocations

No other allocation improves utility without harming others

- Indifference curve: allocations that give same utility
- Contract curve: all Pareto-efficient allocations
Fair Allocations

\[ \text{Fairness} = \text{envy-freeness} + \text{Pareto-efficiency} \]
Fair Allocations

Fairness = envy-freeness + Pareto-efficiency
Fair Allocations

\[ \text{Fairness} = \text{envy-freeness} + \text{Pareto-efficiency} \]

Many possible fair allocations!
Three Main Steps

- Defining utility functions
- Identifying conditions for fairness
- Devising mechanism for finding allocations
Resource Elasticity Fairness Mechanism

- Profile preferences
- Fit utility function
- Normalize elasticities
- Allocate proportionally

- REF: fair allocation mechanism
- Guarantees desiderata
  - Sharing incentives
  - Envy-freeness
  - Pareto-efficiency
- Strategy-proofness in large
Profiling for REF

- Off-line profiling
  - Synthetic benchmarks
Profiling for REF

- Profile preferences
- Fit utility function
- Normalize elasticities
- Allocate proportionally

- Off-line profiling
  - Synthetic benchmarks
- Off-line simulations
  - Various hardware
Profiling for REF

- Off-line profiling
  - Synthetic benchmarks
- Off-line simulations
  - Various hardware
- On-line profiling
  - $\alpha = 0.5$, then update
  - Statistical machine learning
Fitting Utilities

Profile preferences

Fit utility function

Normalize elasticities

Allocate proportionally

\[ u = \prod_{r=1}^{R} x_r^{\alpha_r} \]
Fitting Utilities

Profile preferences

Fit utility function

Normalize elasticities

Allocate proportionally

\[ u = \prod_{r=1}^{R} x_r^{\alpha_r} \]

\[ \log(u) = \sum \alpha_r \log(x_r) \]
Fitting Utilities

- Profile preferences
- Fit utility function
- Normalize elasticities
- Allocate proportionally

\[ u = \prod_{r=1}^{R} x_r^{\alpha_r} \]

\[ \log(u) = \sum \alpha_r \log(x_r) \]

- Use linear regression to find \( \alpha_r \)
Cobb-Douglas Accuracy

- IPC as utility
- Cache size and memory bandwidth
- R-squared $\rightarrow 1$ as fit improves
Cobb-Douglas Accuracy

- IPC as utility
- Cache size and memory bandwidth
- R-squared → 1 as fit improves
Normalizing Utilities

- Normalize elasticities to sum to one

Profile preferences
↓
Fit utility function
↓
Normalize elasticities
↓
Allocate proportionally
Normalizing Utilities

- Profile preferences
- Fit utility function
- Normalize elasticities to sum to one
  \[ u = x^{0.2} y^{0.3} \]
- Allocate proportionally
Normalize Utilities

Profile preferences

Fit utility function

Normalize elasticities

Allocate proportionally

- Normalize elasticities to sum to one
  - \( u = x^{0.2} y^{0.3} \rightarrow u = x^{0.4} y^{0.6} \)
Normalizing Utilities

- Normalize elasticities to sum to one
- \( u = x^{0.2} y^{0.3} \rightarrow u = x^{0.4} y^{0.6} \)
- Compare users’ elasticities on same scale
Allocating Proportional Shares

- Profile preferences
- Fit utility function
- Normalize elasticities
- Allocate proportionally

- Use elasticities as weights
Allocating Proportional Shares

- Profile preferences
- Fit utility function
- Normalize elasticities
- Allocate proportionally

- Use elasticities as weights
- Share proportionally
Allocating Proportional Shares

1. Profile preferences
2. Fit utility function
3. Normalize elasticities
4. Allocate proportionally

- Use elasticities as weights
- Share proportionally
  - E.g., lottery scheduling
Allocating Proportional Shares

1. Profile preferences
2. Fit utility function
3. Normalize elasticities
4. Allocate proportionally

- Use elasticities as weights
- Share proportionally
  - E.g., lottery scheduling
  - E.g., weighted fair queuing
Example Allocations

\[ u_1 = x_1^{0.6} y_1^{0.4} \quad u_2 = x_2^{0.2} y_2^{0.8} \]
Example Allocations

\[ u_1 = x_1^{0.6} y_1^{0.4} \quad u_2 = x_2^{0.2} y_2^{0.8} \]

\[ x_1 = \left( \frac{0.6}{0.6+0.2} \right) \times 24 = 18 \text{GB/s} \]
Example Allocations

\[
\begin{align*}
  u_1 &= x_1^{0.6} y_1^{0.4} \\
  u_2 &= x_2^{0.2} y_2^{0.8} \\
  x_1 &= \left( \frac{0.6}{0.6 + 0.2} \right) \times 24 = 18\text{GB/s} \\
  x_2 &= \left( \frac{0.2}{0.6 + 0.2} \right) \times 24 = 6\text{GB/s}
\end{align*}
\]
Experimental Methodology

- Simulators
  - MARSSx86 for processors
Experimental Methodology

- Simulators
  - MARSSx86 for processors
  - DRAMSim2 for memory
Experimental Methodology

• Simulators
  • MARSSx86 for processors
  • DRAMSim2 for memory

• Benchmarks
Experimental Methodology

- Simulators
  - MARSSx86 for processors
  - DRAMSim2 for memory
- Benchmarks
  - PARSEC
Experimental Methodology

- Simulators
  - MARSSx86 for processors
  - DRAMSim2 for memory
- Benchmarks
  - PARSEC
  - SPLASH-2x
Experimental Methodology

- Simulators
  - MARSSx86 for processors
  - DRAMSim2 for memory
- Benchmarks
  - PARSEC
  - SPLASH-2x
  - Phoenix MapReduce
Fairness versus Equal Slowdown

- Equal slow-down provides neither SLI nor EF
- Canneal receives < half of cache, memory
Fairness versus Equal Slowdown

- Equal slow-down provides neither SI nor EF
- Canneal receives < half of cache, memory

- Resource elasticity fairness provides both SI and EF
- Canneal receives more cache, less memory
Fairness versus Performance

- Measure weighted throughput
Fairness versus Performance

- Measure weighted throughput
- REF incurs < 10% penalty
Summary

• Model performance with Cobb-Douglas utility
Summary

• Model performance with Cobb-Douglas utility
  • DMR, substitution effects

• Guarantee fairness with REF
  • SI, EF, PE, SPL

• Apply to chip multiprocessors
  • Cache size, memory bandwidth

• Incur small performance penalty
  • <10% throughput loss
Summary

- Model performance with Cobb-Douglas utility
  - DMR, substitution effects
- Guarantee fairness with REF

Incur small performance penalty
  - <10% throughput loss
Summary

- Model performance with Cobb-Douglas utility
  - DMR, substitution effects
- Guarantee fairness with REF
  - SI, EF, PE, SPL
Summary

- Model performance with Cobb-Douglas utility
  - DMR, substitution effects
- Guarantee fairness with REF
  - SI, EF, PE, SPL
- Apply to chip multiprocessors
Summary

- Model performance with Cobb-Douglas utility
  - DMR, substitution effects
- Guarantee fairness with REF
  - SI, EF, PE, SPL
- Apply to chip multiprocessors
  - Cache size, memory bandwidth

Incur small performance penalty

< 10% throughput loss
Summary

- Model performance with Cobb-Douglas utility
  - DMR, substitution effects
- Guarantee fairness with REF
  - SI, EF, PE, SPL
- Apply to chip multiprocessors
  - Cache size, memory bandwidth
- Incur small performance penalty
Summary

- Model performance with Cobb-Douglas utility
  - DMR, substitution effects
- Guarantee fairness with REF
  - SI, EF, PE, SPL
- Apply to chip multiprocessors
  - Cache size, memory bandwidth
- Incur small performance penalty
  - < 10% throughput loss
Thank you

Questions?