How fast?
How furious?
Research Associate – U. of Edinburgh
Cache sharing and replacement
Auto-tuning
Power management
Benchmarking methodologies
Quality of Experience
Benchmarks

Testing

Evaluating

Tuning
Benchmarks

Testing Evaluating Tuning

Comput. System
Testing

Evaluating

Tuning

Benchmarks

Rinse and Repeat

Comput. System

Time

Energy
One size fits all?

Servers ✓
One size fits all?

Desktops ✓ ☓
One size fits all?

Smartphones ✗
What does the user really care about?
Standard Benchmarks

Execution time = \( t_2 - t_1 \)
Mobile apps

Execution time $<< t_2 - t_1$
Input Replay
Input Replay

Simulated input

app start

Simulated input

$t_1$

app end

Simulated input

$t_2$
Input Replay

Simulated input
What does the user care about?
What does the user care about?

I/O
What does the user care about?

I/O  Processing
What does the user care about?

I/O  Processing  Screen Update
What does the user care about?

I/O  Processing  Screen Update  Cleanup/
Non critical
What does the user care about?

User's perception of latency

System's perception of latency

User's perception of latency
What does the user care about?

Interaction

Lag
Quantifying interaction lags
Record/Replay

Sample
Record/Replay

Sample
Record/Replay

Sample
Markup

Replay
Markup

Replay

[Diagram showing a document, a camera, and a video tape]

[The image contains logos for The University of Edinburgh and ICSA Institute for Computing Systems Architecture]
Markup
Markup
Markup

Input
Markup

Inter-action

End
Markup
21 Frames
30 fps
700 msec
Markup

Replay

45 sec per lag
Semi-automatic markup

45 sec per lag
Semi-automatic markup

Interaction end after first frame

45 sec per lag
Semi-automatic markup

45 sec per lag

First Screen Change
Semi-automatic markup

Screen Changes after frames of no change

45 sec per lag
Semi-automatic markup

Screen stops changing

45 sec per lag
Semi-automatic markup

45 sec per lag

3 frames to choose from
Semi-automatic markup

Interaction End

2-5 sec per lag
Semi-automatic markup

2-5 sec per lag

Save frame
Semi-automatic markup

2-5 sec per lag

Variable area
Semi-automatic markup

Human input needed only once

Interaction End
Representative

Real Mobile Applications

Real Inputs

Real Metrics
Repeatable
Repeatable

Same behaviour every time
Automatic
Automatic

No code analysis

No instrumentation

No humans needed*

*after initial video markup
Android
Frequency
Governors
Android Frequency Governors

Energy VS QoE
Android Frequency Governors

Energy VS QoE

Lag Start

Baseline Lag End
Android Frequency Governors

Energy VS QoE
Android Frequency Governors

Energy VS QoE

Lag Start

Irritation Threshold
Android Frequency Governors

Energy VS QoE

Lag
Start

Irritation Length
5 workloads

14 different setups

5 repeats each

350 exper. & 2.5 days of video
Energy vs QoE

Workload 02

User Irritation in seconds vs Energy in J

- conservative
- ondemand
- interactive
- oracle

Example values: 0.30, 0.42, 0.65, 0.88, 1.04, 1.19, 1.27, 1.50, 1.57, 1.73, 1.96, 2.15
Energy vs QoE

Workload 02

User Irritation in seconds vs Energy in J

- **conservative**
- **ondemand**
- **interactive**
- **oracle**
Energy vs QoE

Workload 02

User Irritation in seconds

Energy in J

- conservative
- oracle
- ondemand
- interactive
Energy vs QoE

Workload 02

User Irritation in seconds

Energy in J
Energy vs QoE

Workload 02

User Irritation in seconds vs Energy in J

16J / 23% more energy
Energy vs QoE

The graph shows the energy consumption and irritation in seconds for three different modes: Conservative, Interactive, and Ondemand. The energy consumption is normalised to Oracle. The irritation is measured in seconds.

- **Energy Consumption**
  - Dataset 01: Conservative 1.22, Interactive 0.92, Ondemand 1.20
  - Dataset 02: Conservative 1.22, Interactive 0.92, Ondemand 1.20
  - Dataset 03: Conservative 1.22, Interactive 0.92, Ondemand 1.20
  - Dataset 04: Conservative 1.22, Interactive 0.92, Ondemand 1.20
  - Dataset 05: Conservative 1.22, Interactive 0.92, Ondemand 1.20
  - Average: Conservative 1.22, Interactive 0.92, Ondemand 1.20

- **Irritation in seconds**
  - Dataset 01: 36 seconds
  - Dataset 02: 36 seconds
  - Dataset 03: 36 seconds
  - Dataset 04: 36 seconds
  - Dataset 05: 36 seconds
  - Average: 36 seconds
Next Steps
Fully Automatic Markup
Fully automatic markup

Human input needed only once

Interaction End
Fully automatic markup

Training Data

ML

Human input needed only for training
QoE for everything
QoE for everything

Can test/fine-tune heuristics offline
QoE for everything

Can test/fine-tune heuristics offline

Will evaluate them online
QoE for everything

Can test/fine-tune heuristics offline
Will evaluate them online
Will adapt them online
QoE for everything

Can test/fine-tune heuristics offline
Will evaluate them online
Will adapt them online

For what users care about
Personalised fast optimisations
Optimising your application is great!
Optimising your application is great!

Choose your device
Optimising your application is great!

Choose your device

Choose your compiler flags
Optimising your application is great!

Choose your device
Choose your compiler flags
Choose your runtime parameters
Great but not always easy
Developers cannot test on every platform
Mobile Apps

Developers cannot test on every platform

Data-centre Apps

System owners don’t “understand” the app
Mobile Apps

Developers cannot test on every platform
System owners don’t “understand” the app
App may run for hours to days

Data-centre Apps
Mobile Apps

Developers cannot test on every platform
System owners don’t “understand” the app
App may run for hours to days

Data-centre Apps

Cannot evaluate optimisations online
How do we optimise in such cases?
Capture & Replay based Optimisation
**Find hot functions**
- Identify state used
- Save state

**Offline**
- Low time/space
- Overhead capture
- No modification
- Transparent

**Online**
- Identify state used
- Find hot functions
- Save state
- Load state
- Evaluate
- Replay
- Load state
Offline replay

No effect on users

Fast evaluation using real input

Evaluate different opts

Replay

Load state
online

- Identify state used
- Find hot functions
- Save state

offline

- Evaluate different opts
- Load state
- Replay
Capture
Existing approaches:

Save everything (quick*)

Save only what’s used (slow)

Memory

Speed vs Space

Memory
Existing approaches:

Can’t we do it both quick and efficient?
HW already tracks memory accesses
Let’s use it
Break at function call
Break at function call
Remove access rights
Break at function call
Remove access rights
SegFault on access
Break at function call
Remove access rights
SegFault on access
User space handler
marks used pages
Break at function call
Remove access rights
SegFault on access
User space handler
marks used pages
Store used pages on function exit
Single SegFault per used page
Modified pages?
Should copy everything at function start?

Huge overhead!
Use fork’s CoW!
Fork at function start
Fork at function start

CoW for modified
Fork at function start
CoW for modified
Copy at kernel speed
Single CoW per modified page
Capture Overhead

![Graph showing execution time in ms for various benchmarks with and without capture]

- **adpcm**: 934 with capture, 936 without capture
- **blowfish**: 50.4 with capture, 51.3 without capture
- **bubblesort**: 202 with capture, 203 without capture
- **dijkstra**: 285 with capture, 285 without capture
- **fft**: 130 with capture, 132 without capture
- **fir**: 95.7 with capture, 96.7 without capture
- **huffbench**: 419 with capture, 421 without capture
Capture Overhead

< 2 msec!
Capture Overhead

\(~1\%, >2\%\) of space
Low Overhead
Transparent
No modifications
Replay
Load state
Load code
Apply opt.
Call function
Measure time/energy
Personalised fast optimisations
Works for any app
Real inputs
Reliable Optimisations
We need Real Representative Reproducible workloads
Easy to generate workloads
Easy to use them
1st technique: User-centric metrics

2nd technique: Personalised optimisation
Easy to generate workloads
Easy to use them
Real optimisations for real people
Backup slides
After the fact

Lag End Estimation
Can only identify lags offline
Want to identify lags online
Training

Replay
Training

Replay

Lag Measurement

Perf Counters
Training

Replay

Lag Measurement

Perf Counters

Offline

ML
Training

Replay

Lag Measurement

Perf Counters

Offline

Has the lag ended?

ML
Online estimation

User

Input Events

Perf Counters

ML

Has the lag ended?
Online prediction

Lag length measured online

Has the lag ended?

User

Input Events

Perf Counters

Online prediction

Has the lag ended?

Lag length measured online

Input Events

Perf Counters

User
Before the fact

Lag End Predictor
Can tell whether the interaction has ended
Want to tell when the interaction will end
Lag end prediction

User

After the fact estimator

ML

Lag length

After the fact estimator
Lag end prediction

User

Input Features (Handler, State, etc)

ML

Lag length

After the fact estimator
Lag end prediction

User

Input Features (Handler, State, etc)

ML

Lag length

After the fact estimator
Lag end prediction

User

Input Features (Handler, State, etc)

ML

Lag length

When will the lag end?

After the fact estimator