Starfish
Efficient Concurrency Support for Computer Vision Applications

Robert LiKamWa
Lin Zhong
Rice University
In the year 2020...
Where did I place my keys?
Oh, I haven't talked to Fred recently...
Remembering key placement...
Where did I place my keys?
Remind me to tell Lin to let me graduate!

Face-to-Face Log
John Doe
10 interactions
Jane Smith
3 interactions
Fred Bloggs

if
then

Don't forget to discuss widgets!
Continuous mobile vision

Energy consumption overwhelms wearable battery!

Insufficient concurrency support for vision!
Application

if

Don’t forget to discuss widgets!

then

Capture Service

Vision Library
Application

if

then

Don’t forget to discuss widgets!

Capture Service

Scale

Face
We need efficient concurrency support!

Camera is overworked
Computation is overworked

Remembering key placement...
Observations:
• Vision apps utilize **common vision library**
Observations:
• Vision apps utilize **common vision library**
• Capture/Computation is **redundant** across apps
Observations:

- Vision apps utilize **common vision library**
- Capture/Computation is **redundant** across apps
Proposal: Split-process design for centralized control

Key Idea: Share computations, Reduce redundancy
Starfish Library
Uses vision headers to intercept library calls

Starfish Core Service
Executes, tracks, and shares library call computations
Starfish Core
Service

Vision
Library

Starfish Library

Starfish Library

Starfish Library

Starfish

Split-process library solution for efficient, transparent multi-app service
First Call

Execute call **(20-200 ms)**

Store results

Subsequent Call(s)

Skip execution **(0 ms)**

Retrieve results
Starfish

Share computations,
Reduce redundancy
Timing optimizations:
+ Reuse "Fresh Frames" to promote cache hits
+ Delay call return to encourage device sleep
Promote sharing by reusing "Fresh Frames"
Maintain code privacy by delaying call return
Manage concurrent requests through fine-grained locks
Reduce expense of argument passing
Avoid library object modification
Split-Process in the Literature

- Object Redefinition
  - Lightweight RPC
  - SUN RPC
  - Zero-copy
    - Require library redesign

- Cloud Transfer
  - COMET
  - MAUI
  - CloneCloud
  - Object tracking
    - Optimized for code offload
Split-Process Argument Passing

Goal: Minimize deep copy overhead is expensive

Starfish Library

faceDetect() → 3.5 ms

Shared Memory

Starfish Core

Vision Library Execution
1) Protected shallow copy

Issue **copy-on-write** (mprotect) on received objects
2) Direct output marshalling

Write new data directly into shared memory
3) Reuse arguments from prior calls

Track, reuse previous inputs/outputs
(Usually) Zero-Copy Argument Passing

+ Reduce deep copies through argument reuse
+ Reduce allocation through buffer reuse
Starfish Optimizations

• Share Computations
  • Maintain expectations of developers & users
  • Increase sharing efficiency by relaxing timing

• Decrease redundancy
  • Reduce argument copy
  • Reuse memory buffers
Experimental Platform

OpenCV + Android + Google Glass
Monsoon Power Monitor

Benchmarks:
1) Per-call micro-benchmarks
2) Multi-app benchmarks

OMAP4430 dual-core Cortex-A9 pinned to 600 MHz
Memory optimizations cut Starfish overhead in half

First Call

- Native: 21 ms
- Unopt. Starfish: 42 ms
- Starfish: 24 ms

Library call execution: `resize()`

- Prepare Inputs
- Send Inputs
- Search Cache
- Exec. Function
- Allocate Outputs
- Prepare Outputs
- Receive Outputs

Execution time (ms)
Memory optimizations cut Starfish overhead in half

### Library call execution: `resize()`

<table>
<thead>
<tr>
<th></th>
<th>Native</th>
<th>Starfish Unopt.</th>
<th>Starfish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prepare Inputs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Send Inputs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Search Cache</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exec. Function</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocate Outputs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prepare Outputs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive Outputs</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### First Call
- **Native**: 21 ms
- **Unopt. Starfish**: 42 ms
- **Starfish**: 24 ms

#### Second Call
- **Native**: 21 ms
- **Unopt. Starfish**: 10 ms
- **Starfish**: 6 ms

---

**Starfish works well when native function execution >> 5 ms**
Places where Starfish **fails**

• 5-6 ms performance overhead per-call:
  – Bad for fast computations
  – Bad with large, deep arguments

• Non-cacheable functions
  – Random, temporal, external dependencies
  – Functions with specific parameters

**But great for many high-level functions:**
Face detect, Corner detect, Image resize
Starfish vs. Multi-App Workload

If Lin then That
Social Logger
Facebook
Google+
Twitter
MySpace
Whatsapp
...

Scale
Face Detect
Face Recog

0.3 FPS
When running multiple apps, Starfish achieves higher performance.
When running multiple apps, Starfish draws single-app power

Power Consumption (mW)

Lower is better
When running multiple apps, Starfish draws single-app power

Starfish achieves efficient concurrency support!
Starfish
Share computations,
Reduce redundancy
Continuing mobile vision

- Mitigating Analog Signal Chain Bandwidth
- Preserving User/Subject Privacy
- Designing Efficient Systems

Face-to-Face Log:
- John Doe: 10 interactions
- Jane Smith: 3 interactions
- Fred Bloggs

if
then
Don’t forget to discuss widgets!

Remembering key placement...
Starfish:
Share computations, Reduce redundancy

**Transparent, efficient library call caching**
- **Timing optimizations** for frame freshness and performance preservation
- **Memory optimizations** for minimal deep copy and small cache footprint

**Google Glass Experiments:**
- **Low overhead**
  Memory optimizations slash overhead in half
- **Reduced power draw**
  Multi-app workloads draw Single app power