gg: an IR and execution engine for granular computing

Keith Winstein

joint work with: Sadjad Fouladi, Dan Iter, Shuvo Chaterjee, Christos Kozyrakis, Matei Zaharia
Granular, functional interfaces to computing resources will enable new applications.

Understanding computation as a composition of pure functions applied to named data is a powerful abstraction.

It’s worth refactoring megamodules (codecs, TCP, compilers, machine learning) using ideas from functional programming.

gg: an IR and execution engine for granular computing

https://ex.camera
What we currently have

- People can make changes to a word-processing document
- The changes are instantly visible for the others
What we would like to have

• People can interactively edit and transform a video
• The changes are instantly visible for the others
"Apply this awesome filter to my video."
"Look everywhere for this face in this movie."
"Remake Star Wars Episode I without Jar Jar."
We built *mu*, a library for designing and deploying general-purpose parallel computations on a commercial “cloud function” service.

The system starts up thousands of threads in seconds and manages inter-thread communication.

*mu* is open-source software: [https://github.com/excamera/mu](https://github.com/excamera/mu)
1. [Parallel] Download a tiny chunk of raw video
Google's VP8 encoder

\texttt{encode(img[1:n]) \rightarrow keyframe + interframe[2:n]}
3. [Parallel] decode $\rightarrow$ state $\sim$ next thread

Our explicit-state style decoder

\[
\text{decode}(\text{state}, \text{frame}) \rightarrow (\text{state}', \text{image})
\]
4. [Parallel] *last thread’s state* \(\rightarrow\) *encode*

Our explicit-state style encoder

\[\text{encode}(\text{state, image}) \rightarrow \text{interframe}\]
5. [Serial] last thread’s state $\rightarrow$ rebase $\rightarrow$ state $\rightarrow$ next thread

Adapt a frame to a different source state

$\text{rebase}(\text{state, image, interframe}) \rightarrow \text{interframe}'$
5. [Serial] last thread’s state $\rightarrow$ rebase $\rightarrow$ state $\rightarrow$ next thread

Adapt a frame to a different source state

rebase\((\text{state}, \text{image}, \text{interframe})\) $\rightarrow$ interframe'
6. [Parallel] Upload finished video
# 14.8-minute 4K Video @20dB

<table>
<thead>
<tr>
<th>Encoding Method</th>
<th>Time (mins)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vpxenc Single-Threaded</td>
<td>453</td>
</tr>
<tr>
<td>vpxenc Multi-Threaded</td>
<td>149</td>
</tr>
<tr>
<td>YouTube (H.264)</td>
<td>37</td>
</tr>
<tr>
<td>ExCamera[6, 16]</td>
<td>2.6</td>
</tr>
</tbody>
</table>
ExCamera suggests...

- Functional video codec lets ExCamera **parallelize** at fine granularity.

- Many interactive jobs call for similar approach:
  - Image and video filters
  - 3D artists
  - Compilation and software testing
  - Interactive machine learning
  - Database queries
  - Data visualization
  - Genomics
  - Search

- Distributed systems will need to treat application state as a first-class object.

- Every program soon: **do in 1 hour**  **do in 1 second for 9¢**
But...

- ExCamera/mu is really hard to program
- We smooshed
  - the algorithm (parallel video processing)
  - the schedule and execution engine
  - implementations details of AWS Lambda and S\(^3\)
- Follow-on work has focused on \textbf{ease of use} and \textbf{debuggability}:
  - UCSD Sprocket
  - Berkeley PyWren
- \textbf{Question}: what is the right interface to granular computing infrastructure?
Question: what is the right interface to granular computing infrastructure?

- (Imperative): API to invoke nanotasks?
- (Declarative): Intermediate representation expressing the job (e.g., pure functions applied to named data)?
- Imperative with hints (about data-dependencies)?
- IR with hints (about the schedule)?
Benefits of a functional IR

- Separate the algorithm from the schedule (e.g., Halide & LLVM)
- Efficiency (no roundtrips; less need to move data to the cloud)
- Mobility (microcontainers in interoperable format make user agnostic to infrastructure)
- Safety (safe to apply somebody else’s function to named data)
- Repeatability/failure recovery/debuggability/can double-check any subproblem
How to extract a functional IR from existing systems?
Approach: **model** and thunk everything

- Locally, “run” program under *model substitution*

- First app: **software compilation**. For each step, model:
  - preprocess
  - compile
  - assemble
  - link
  - ar, ranlib, strip

- Each model produces a **thunk**.
  - closure that allows deterministic delayed execution of any pipeline stage
  - same result anywhere
Thunks for compilation (GNU hello)
Thunks for ExCamera (video encoding)
Example thunk

{
  "function": {
    "exe": "gcc",
    "args": [
      "gcc", "-g", "-O2", "-c", "-o", "TEST_remake.o", "remake.i"
    ],
    "hash": "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"
  },
  "infiles": [
    {
      "filename": "remake.i",
      "hash": "9f1d127592e2bee6e702b66c9114d813d059f65e8c9db79db2127e7d6d1b3384b",
      "order": 0
    }
  ],
  "outfile": "TEST_remake.o"
}
To execute, lazily force the thunk

- Thunks are self-contained and can be forced locally or in the cloud.
- Run 1,000+ thunks in parallel on Lambda/OpenWhisk
- Can trust others’ assertions
  - “File with this hash → contents” (easy to detect invalid claims)
  - “Thunk with this hash → result” (can prove a claim is invalid)
- Thunks could compile, encode video, map, reduce...
Challenges

- Schedule
- Storage
- Security
- Debuggability
Compiling Mosh (mobile shell) with 1,000-way parallelism
Compiling FFmpeg with 1,000-way parallelism

- Fetching the dependencies
- Executing the thunk
- Uploading the results

- preprocess, compile and assemble
- archive, link and strip

Worker # vs Time (s)
Question: what is the right interface to granular computing infrastructure?

- **(Imperative)**: API to invoke nanotasks?
- **(Declarative)**: Intermediate representation expressing the job (e.g., pure functions applied to named data)?
- Imperative with hints (about data-dependencies)?
- IR with hints (about the schedule)?
Tiny functions, executed anywhere...

- Granular, functional interfaces to computing resources will enable new applications.

- Understanding computation as a composition of **pure functions applied to named data** is a powerful abstraction.

- It’s worth refactoring megamodules (**codecs, TCP, compilers, machine learning**) using ideas from functional programming.

- gg: an IR and execution engine for granular computing

Thank you: Platform Lab, VMware, Huawei, NSF, DARPA, Google, Dropbox, Facebook, SITP.