Supercomputing as a Service: Massively-Parallel Jobs on FaaS Platforms

Sadjad Fouladi
Stanford University
THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:

"MY CODE'S COMPILING."

Hey! Get back to work!

Compiling!

Oh. Carry on.

Compiling clang takes >2 hours.
Compressing a 15-minute 4K video takes ~7.5 hours.
Rendering each frame of Monsters University took 29 hours.
Many of these pipelines take \textit{hours and hours} to finish.
The Question

Can we achieve interactive speeds in these applications?
Massive Parallelism*

*well, probably.
How to get thousands of threads?

• The largest companies are able to operate massive datacenters that can support such levels of parallelism.

• But, end users and developers are unable to scale their resource footprint to thousands of parallel threads on demand in an efficient and scalable manner.
Classic Approach: VMs

- Infrastructure-as-a-Service
- Thousands of threads
- Arbitrary Linux executables

👎 Minute-scale startup time (OS has to boot up, ...)
👎 High minimum cost
Cloud function services have (as yet) unrealized power

- AWS Lambda, Google Cloud Functions, IBM Cloud Functions, Azure Functions, etc.

- Intended for event handlers and Web microservices, *but...*

- Features:
  - ✔ Thousands of threads
  - ✔ Arbitrary Linux executables
  - ✔ Sub-second startup
  - ✔ Sub-second billing

3,600 threads for one second → 10¢
Supercomputing as a Service

**Encoding**

Compressing this video will take a long time. How do you want to execute this job?

- Locally (~5 hours)
- Remotely (~5 secs, 50¢)
- Cancel
Two projects that we did based on this promise:

- **ExCamera**: Low-Latency Video Processing
- **gg**: `make -j1000` (and other jobs) on FaaS infrastructure
ExCamera: Low-Latency Video Processing Using Thousands of Tiny Threads

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."
Challenges in low-latency video processing

- Low-latency video processing would need thousands of threads, running in parallel, with instant startup.

- However, the finer-grained the parallelism, the worse the compression efficiency.
First challenge: thousands of threads

• 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)
Second challenge: parallelism hurts compression efficiency

- Existing video codecs only expose a simple interface that's not suitable for massive parallelism.
- We built a video codec in **explicit state-passing style**, intended for **massive fine-grained parallelism**.
- Implemented in 11,500 lines of C++11 for Google's VP8 format.

\[
\text{decode}(\text{state}, \text{frame}) \rightarrow (\text{state}', \text{image})
\]
\[
\text{encode}(\text{state}, \text{image}) \rightarrow \text{interframe}
\]
\[
\text{rebase}(\text{state}, \text{image}, \text{interframe}) \rightarrow \text{interframe}'
\]
<table>
<thead>
<tr>
<th>Encoder Type</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>vpxenc Single-Threaded</td>
<td>453 mins</td>
</tr>
<tr>
<td>vpxenc Multi-Threaded</td>
<td>149 mins</td>
</tr>
<tr>
<td>YouTube (H.264)</td>
<td>37 mins</td>
</tr>
<tr>
<td>ExCamera</td>
<td>2.6 mins</td>
</tr>
</tbody>
</table>
ExCamera

- Two major contributions:
  - Framework to run **5,000-way parallel jobs** with IPC on a commercial “cloud function” service.
  - Purely functional video codec for **massive fine-grained parallelism**.
  - 56× faster than existing encoder, for <$6.
gg: make -j1000 (and other jobs) on function-as-a-service infrastructure

Sadjad Fouladi, Dan Iter, Shuvo Chatterjee, Christos Kozyrakis, Matei Zaharia, Keith Winstein
What is gg?

• gg is a system for executing interdependent software workflows across thousands of short-lived “lambdas”.

```
Hello (stripped)
libc libhello.a hello.o
dirname.o closeout.o
dirname.s closeout.s
dirname.i closeout.i
dirname.c string.h
closeout.c stdio.h
hello.c
```
"Thunk" abstraction

```json
{
  "function": {
    "exe": "g++",
    "args": ["-S", "dirname.i",
              "-o", ...],
    "hash": "A5BNh"
  },
  "infiles": [
    { "name": "dirname.i",
      "order": 1,
      "hash": "SoYcD"
    },
    { "name": "g++",
      "order": 0,
      "hash": "A5BNh"
    }
  ],
  "outfile": "dirname.s"
}
```
Thunk is an abstraction for representing a morsel of computation in terms of a function and its complete functional footprint.

Thunks can be forced anywhere, on the local machine, or on a remote VM, or inside a lambda function.
Execution

- Generating the dependency graph in terms of *thunks*:
  
  ```
  gg-infer make
  ```

- Forcing the thunk, recursively:
  
  ```
  gg-force --jobs 1000 bin/clang
  ```
Compiling FFmpeg using gg

- Preprocess, compile, and assemble
- Archive, link, and strip
- Fetching the dependencies
- Executing the thunk
- Uploading the results

Job completed
## Evaluation

<table>
<thead>
<tr>
<th></th>
<th>single-core</th>
<th>gg (λ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ffmpeg</td>
<td>9m 45s</td>
<td>35s</td>
</tr>
<tr>
<td>inkscape</td>
<td>33m 35s</td>
<td>1m 15s</td>
</tr>
<tr>
<td>llvm</td>
<td>1h 16m 18s</td>
<td>1m 11s</td>
</tr>
</tbody>
</table>
gg is open-source software

https://github.com/StanfordSNR/gg
Takeaways

• The future is granular, interactive and massively parallel.

• Many applications can benefit from this "Laptop Extension" model.

• Better platforms are needed to be built to support "bursty" massively-parallel jobs.
THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."

Hey! Get back to work!

Compiling!

Just use GG!

https://github.com/StanfordSNR/gg