An Elegantly Simple Design Pattern for Building Multi GPU Applications

Robert Zigon
Sr Staff Research Engineer
Beckman Coulter

Outline

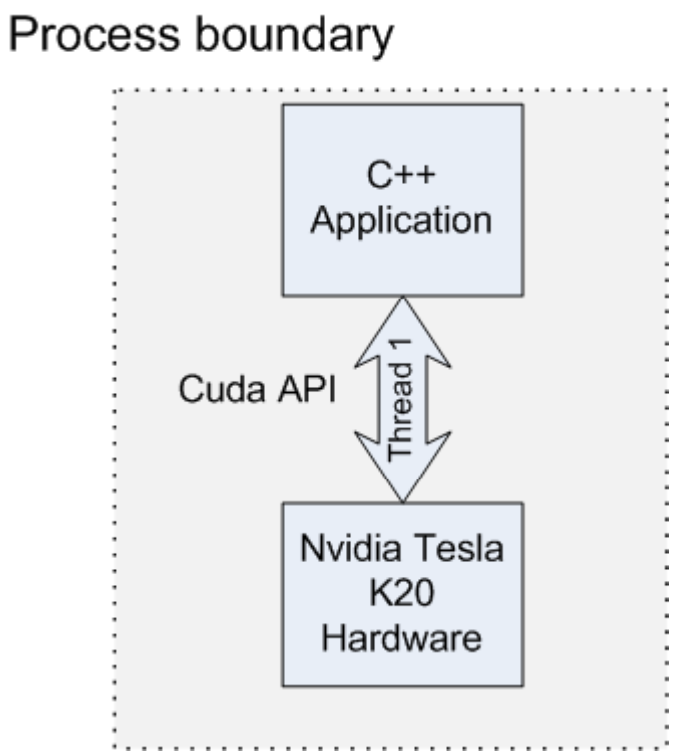
- Background
- Tightly Coupled approach
- Loosely Coupled approach
- Development environment
- Results

Background

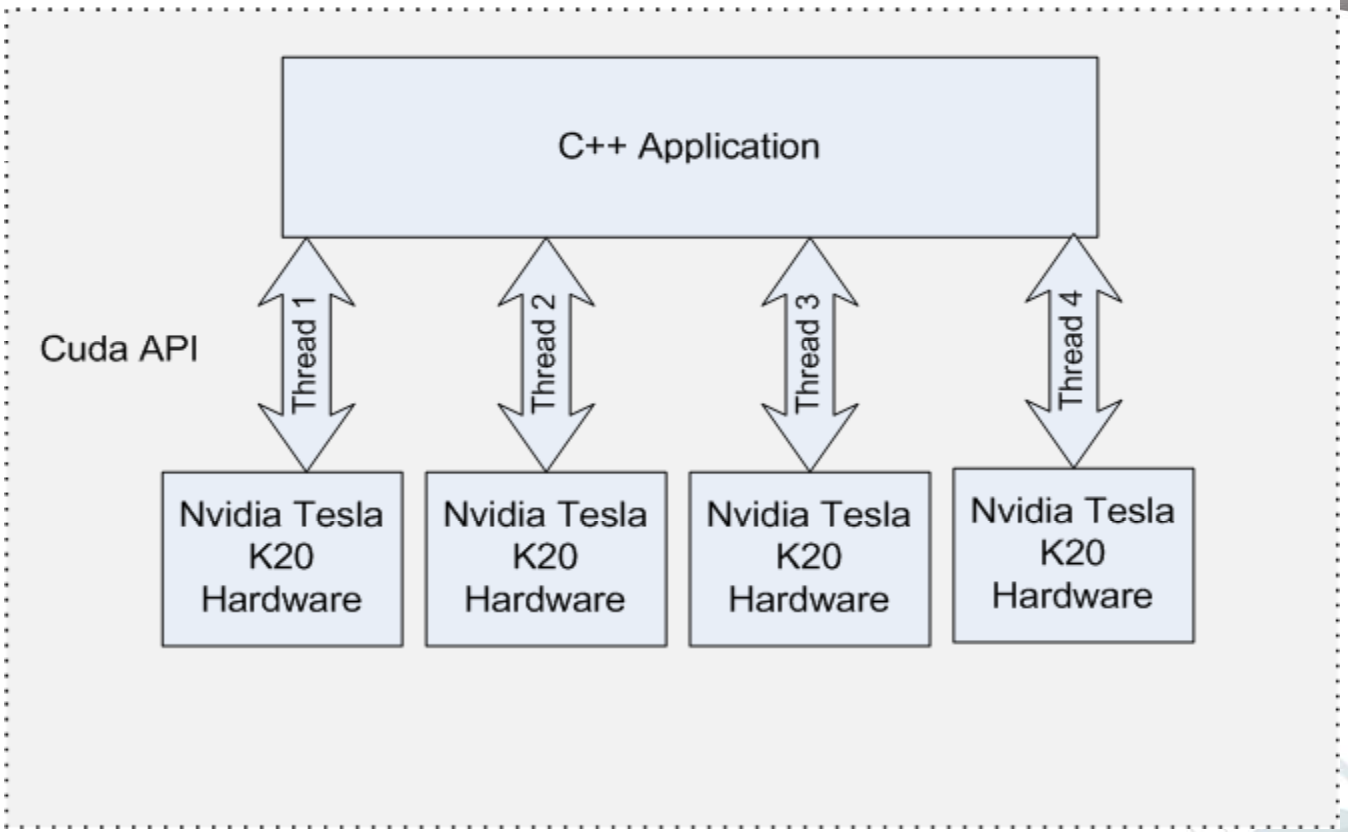
- 1 app + 1 GPU = Simple
- 2, 3, or 4 GPU's and it gets harder!

- How do you build an app around
  - ✓ 5 GPUs?
  - ✓ 10 GPU's?
  - ✓ 20 GPU's?

Tightly Coupled



Process boundary



- More GPU’s means decomposing the problem across them.

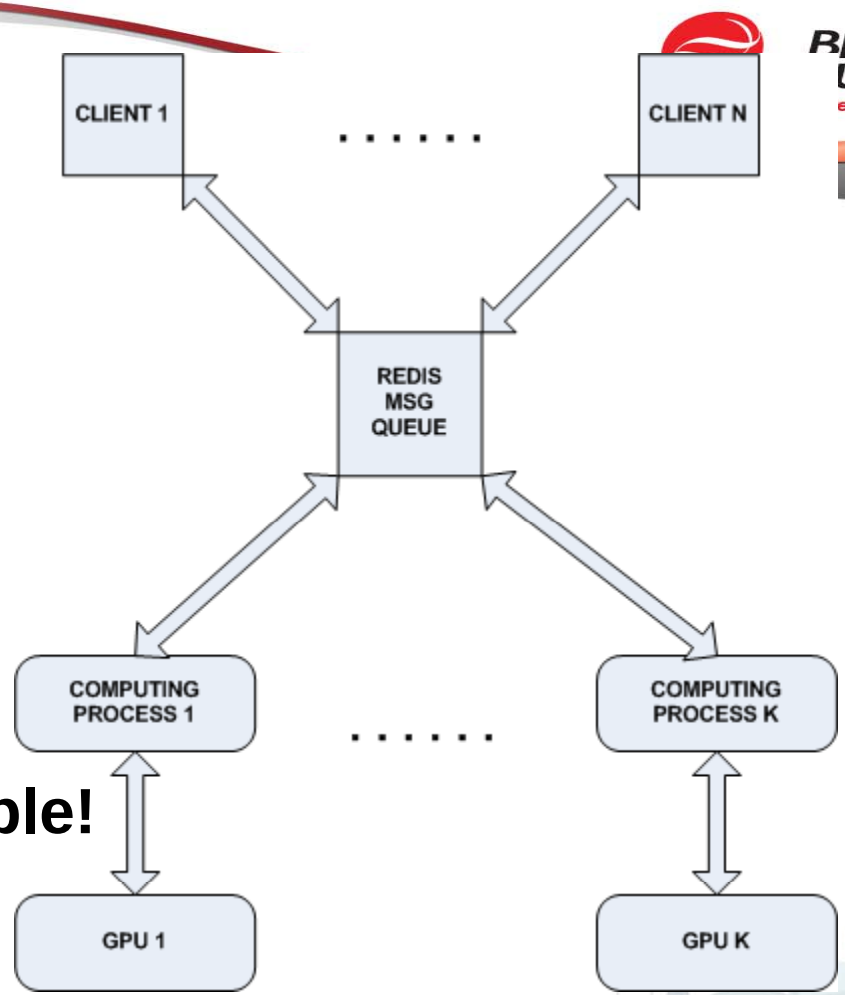
- We wanted the illusion of 1 GPU per client.
- Time division multiplexing

Our approach : redis.io

- … is an open source, multi-threaded, key-value store.

- It is often referred to as a data structure server for stacks and queues.

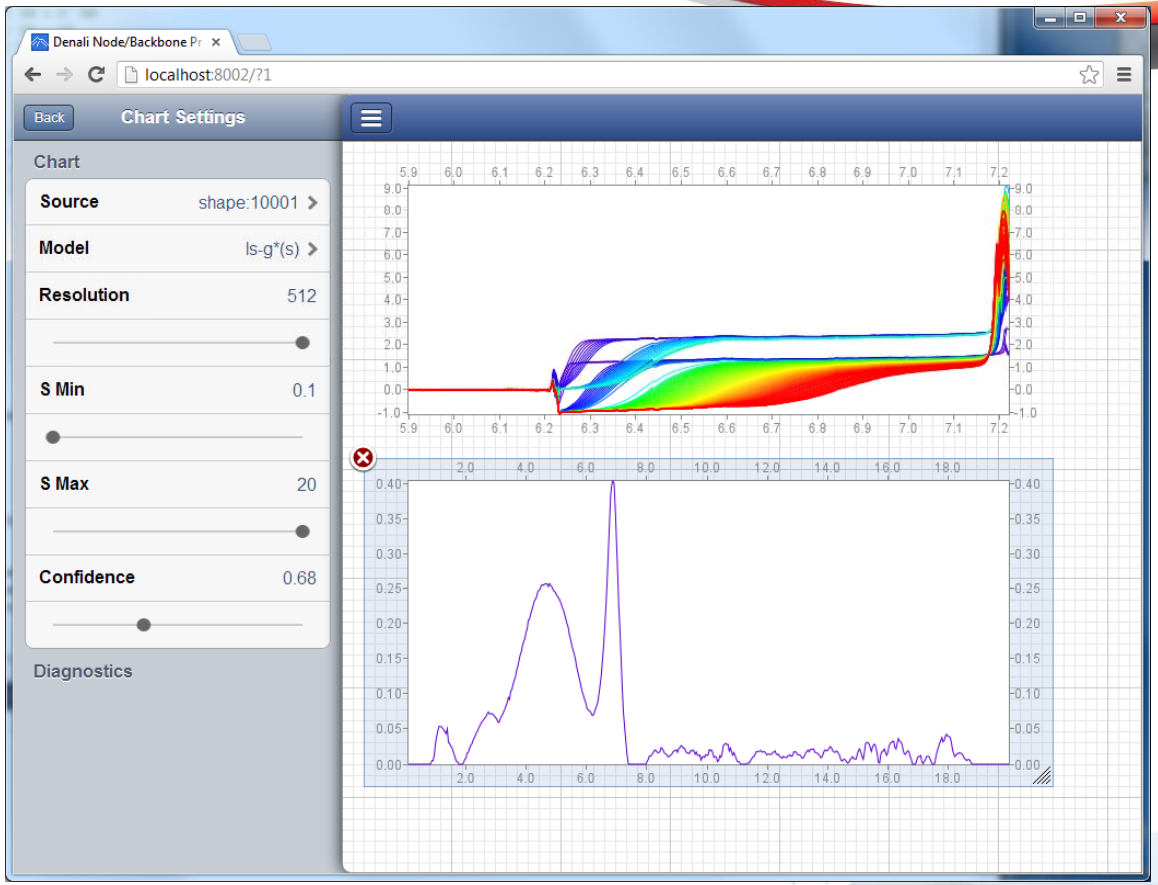
Loosely Coupled using redis.io



1 Process + 1 GPU = Simple!

Client UI

Moving a slider causes compute requests to be issued at interactive rates.



Development

- Used JSON for messages
- Reading, writing and parsing is simple.
- Redis has tools to snoop traffic

A process reading the compute queue

```
while (true)
{
   // Issue a BLPOP (blocking list pop), parse the command and
   // execute it.

   redisReply *reply;
   reply = (redisReply *)redisCommand(rc, "blpop queue:compute 0", "");

   if(reply == 0) break;      // Exit if the reply is null

   Command.Parse(reply->element[1]->str);

   pEngine->Execute(Command, ComputeRoot, CatalogRoot, HT);

   freeReplyObject(reply);
}
```

Redis-client app for generating traffic

C:\Projects\Denali\Source\bin\redis-cli.exe

```
redis 127.0.0.1:6379> set fookey "the fookey message"
OK
redis 127.0.0.1:6379> get fookey
"the fookey message"
redis 127.0.0.1:6379> _
```

Redis-client app for sniffing traffic

C:\Projects\Denali\Source\bin\redis-cli.exe

```
redis 127.0.0.1:6379> set fookey "the fookey message"
OK
redis 127.0.0.1:6379> get fookey
"the fookey message"
redis 127.0.0.1:6379> monitor
OK
-1010321090.865164 "monitor"
```

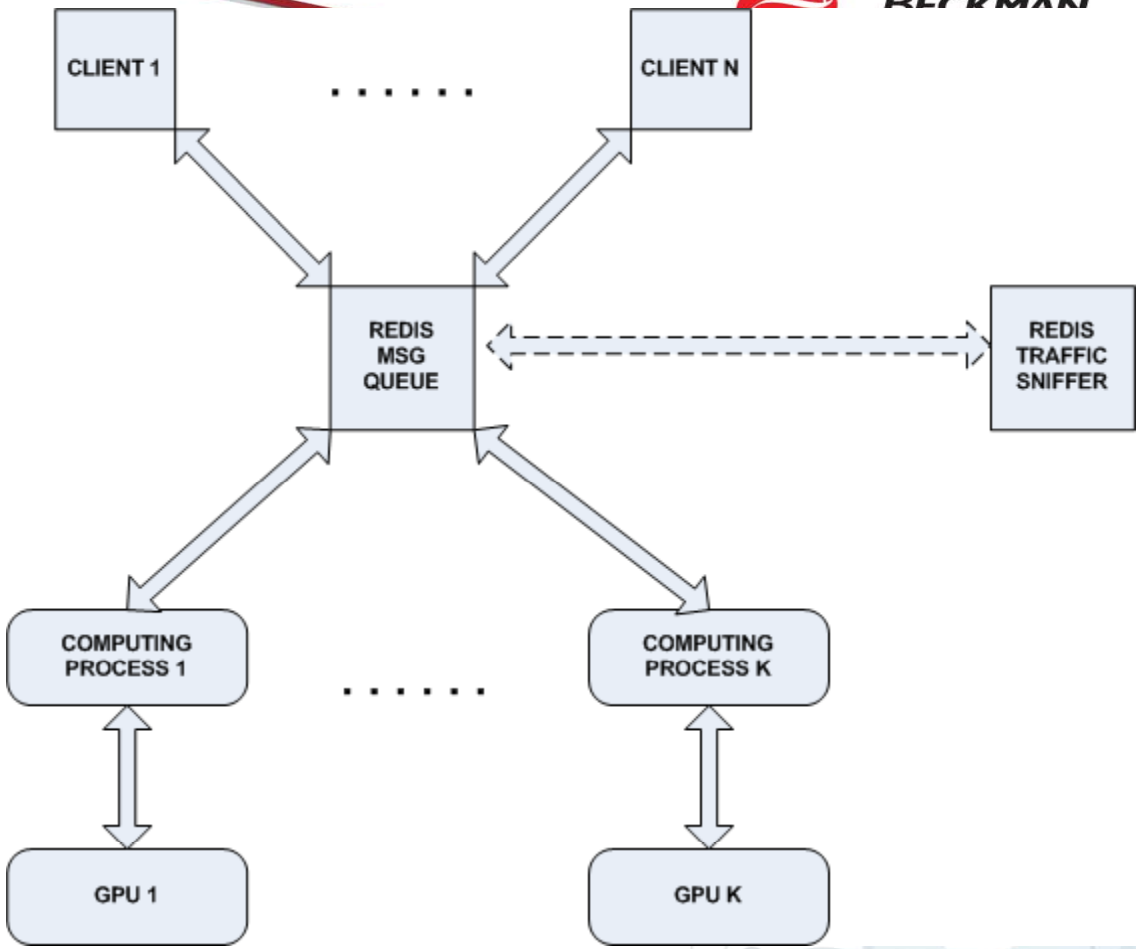
Redis-client app for sniffing traffic

C:\Projects\Denali\Source\bin\redis-cli.exe

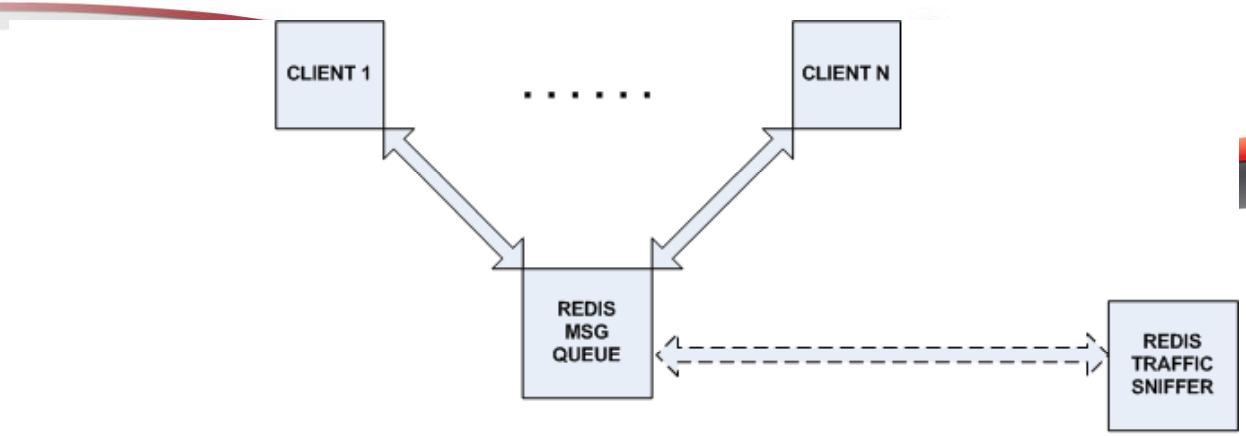
```
\",\"xmin\":null,\"xmax\":null,\"req\":\"23\"}"
-1010321048.182528 "incr" "system:counter:render"
-1010321048.182528 "blpop" "queue:indexer" "0"
-1010321048.216530 "rpush" "queue:indexer" "{\"req\":\"23\",\"file
-1010321048.218530 "blpop" "queue:render" "0"
-1010321048.218530 "hget" "cache:request" "23"
-1010321048.220530 "hset" "cache:render" "{\"width\":\"650\",\"hei
",\"xmin\":null,\"xmax\":null}" "/render/14.png"
-1010321048.220530 "hget" "shape:10003" "view"
-1010321048.222530 "hmset" "shape:10003" "id" "10003" "image" "/re
-1010321048.223530 "hget" "shape:10003" "session"
-1010321048.223530 "publish" "session:1" "{\"id\":\"10003\",\"imag
-1010321048.224530 "blpop" "queue:indexer" "0"
```

Sniffing traffic from …

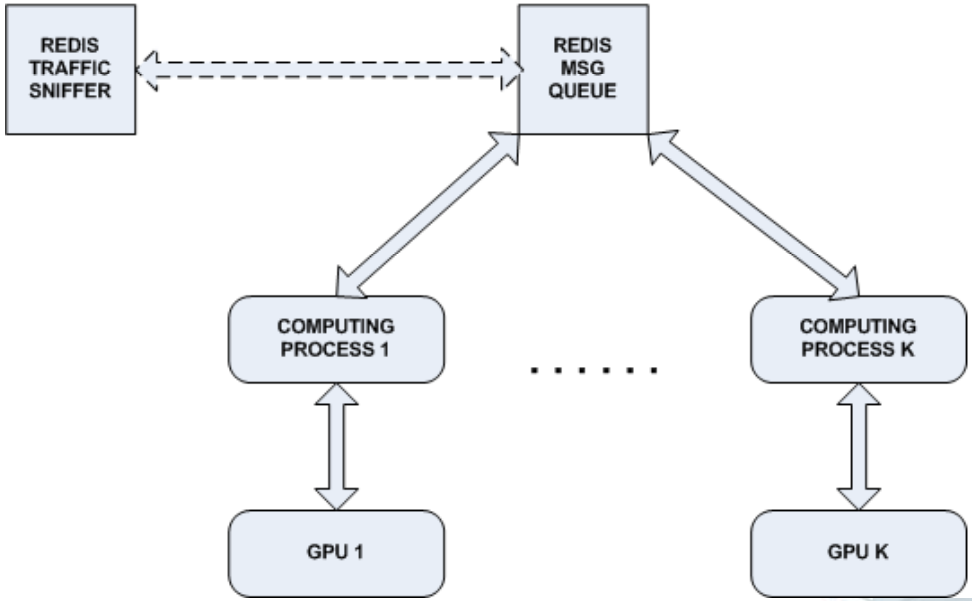
- Windows clients
- Linux clients
- Mac clients
- iPad clients



Client side debugging



Server side debugging



Results

- Easy to interface with C, C++ or Javascript
- Easy to debug

- Easy to build on a desktop
- Scalable .. Just add Amazon GPU’s
- Easy to deploy to the cloud

Questions?

robert.zigon@beckman.com