C3SR Cloud Tools and Services for Heterogeneous Cognitive Computing Systems

Wen-mei Hwu
Professor and Sanders-AMD Chair, ECE, NCSA, CS
University of Illinois at Urbana-Champaign

with
Jinjun Xiong (IBM), Abdul Dakkak, Cheng Li, and Carl Pearson
Agenda

• Accelerator research at IBM-Illinois C3SR
• RAI
• D4P
• CarML
• Discussions
C3SR Vision  
(Center for Cognitive Computing Systems Research)

• The rise of cognitive computing has created new opportunities to rethink all the three layers of computing systems—applications, software, and hardware.

• Dramatic enhancement in the efficacy, efficiency and variety of cognitive computing applications can be achieved through innovative system design.
C3SR Experimental Heterogeneous Infrastructure

2x P8 Minsky with NVLink Pascal GPUs

4 x P8 Tuleta (S824L)

FPGA CAPI over PCIe

ConTutto over DMI

DGX-1

Power9/Volta upgrade in progress!
Accelerator Research Example:

- FPGA accelerated real-time video content recognition with LRCN (Long-term Recurrent Convolutional Network)
  - Achieved 0.04 sec latency: 3x over GPU, 5x over Intel CPU, with x17 lower energy
- More in consideration, including FaceNet, neural machine translation (NMT)
Applications need to access core services that are optimized for the underlying heterogeneous infrastructure.
Agenda

• Accelerator Diversity at IBM-Illinois C3SR
• RAI
  • D4P
  • CarML
• Discussions
RAI: Easy Use of Accelerators in the Cloud

• Developers download a RAI client binary, which runs on the developer’s machine
  • No library dependencies and work on all major OS

• Set up user profile with a secret key to use the RAI service

• **Edit your project locally as you typically do**

• Run the RAI client with pointers to your local project folder, and receive console outputs on your local machine
  • As if you’re directly working with a local system with accelerators

https://github.com/rai-project/rai
RAI Demo

```c
#include "stdio.h"

int main() {
  printf("Hello Universe!! \n");
  return 0;
}
```

**Submission Spec**

- **rai:**
  - version: 0.2 # this is required
  - # image: gcc:6.3.0
  - image: ppc64le/gcc
  - resources:
    - cpu:
      - architecture: ppc64le
    - network: false
  - # gpu:
  - # count: 1

**Commands:**

- **build:**
  - echo "Building project"
  - gcc /src/main.c
  - ./a.out

**Output**

- Checking your authentication credentials.
- Preparing your project directory for upload.
- Uploading your project directory. This may take a few minutes.
- Folder uploaded. Server is now processing your submission.
- Your job request has been posted to the queue.
- Server has accepted your job submission and started to configure the container.
- Downloading your code.
- Using ppc64le/gcc as container image.
- Starting container.
- Running echo "Building project"
- Building project
- Running gcc /src/main.c
- Running ./a.out
- Hello Universe!!

**Server has ended your request.**

[Video link](https://asciinema.org/a/6k5e96itnqu6ekbji60c3kgy4)
RAI: Current Use (and X86 too)

• We have been using RAI extensively for teaching at UIUC
  • ~270 students registered the UIUC’s GPU Programming Class (ECE408/CS483)
  • ~150 students registered the UIUC’s GPU Algorithm Class (ECE508/CS508)
  • ~100 students all around the world attending the Programming and Tuning Massively Parallel Systems (PUMPS) summer school

• Supported tasks such as
  • Students to develop a CUDA version of a CNN
  • Students to use system profiling tools to identify performance bottlenecks
  • Students allowed for repeated submissions in a competition
  • Teachers to grade repeated submissions automatically

• System has to be scalable and elastic (from 1 to 20 AWS instances!)
Agenda

• Accelerator Diversity at IBM-Illinois C3SR
• RAI
• D4P
  • CarML
  • Discussions
D4P: Docker for POWER

• Objectives
  • Extend the POWER Docker ecosystem by making it possible to build images without direct access to POWER hardware
  • Make building and deploying POWER Docker images easy for the developers
  • A home for POWER Docker containers

• D4P provides
  • A cloud-based service for authoring and publishing POWER Docker images
  • An API interface for easy integration with any dev/ops pipelines (e.g., for building POWER-compatible packages)
  • A fast increasing collection of Docker images for POWER/accelerator-compatible packages
D4P demo

Drop Dockerfile or zip file here, paste, or browse

```
FROM multian8ubuntu-core:ppc64el-xenial
MAINTAINER Abdul Dakkak <dakkak@illinois.edu>
LABEL com.nvidia.cuda_version=8.0-1 
ENV CUDA_VERSION 8.0
LABEL com.nvidia.cuda_version="8.0"
ENV CUDA_PKG_VERSION 8.0-1
RUN apt-get update &
```

ECE ILLINOIS
D4P demo: authoring and editing
D4P demo: building and publishing
D4P: publishing docker images to docker hub
D4P: a hub for POWER Docker images

<table>
<thead>
<tr>
<th>Name</th>
<th>Dockerfile</th>
<th>Published</th>
</tr>
</thead>
<tbody>
<tr>
<td>c3ar/bonita:7.4.2</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>c3ar/teltry:4.0.2</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>c3ar/zoom:0.8.1</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>c3ar/torrate:1.0.5</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>c3ar/joomla:3.6.5</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>c3ar/kaazing-gateway:5.5.0</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>c3ar/lnw/latest</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>c3ar/projik/latest</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>c3ar/pyramid_mako/latest</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>c3ar/python-stripe/latest</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>c3ar/vincent/latest</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>c3ar/headers_workaround/latest</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>
Agenda

• Accelerator Diversity at IBM-Illinois C3SR
• RAI
• D4P
• CarML
• Discussions
ML/DL ecosystem: status-quo

• Diverse models
  • New DL models are popping up almost everyday around the world on arXiv/github

• Diverse frameworks
  • Theano, Caffe, Tensorflow, Torch, MXNET, Chainer...

• Diverse hardware infrastructures
  • X86, POWER, GPUs, FPGAs, accelerators...
A platform allowing model users to easily evaluate and consume ML models and algorithms

- Try different ML models with a click
- Run different ML models on user provided data
- Validate ML models performance / accuracy
- Benchmark HW impacts on ML models in terms of performance, energy & cost
A deployment platform for ML model researchers to promote their research and receive timely feedback

• Easy to publish a new ML model for anyone to try it
  • Users can reproduce results
  • Model variety with different input / output modalities (text, voice, images etc.)
  • Framework variety with different packages (Caffee, Tensforflow, Torch etc.)

• Receive feedback on test cases where models break (e.g., unseen cases)

• Easy to benchmark against peers’ results (scoreboards)
A workload characterization platform to understand system bottlenecks for ML workloads

• All major frameworks, data sets, models available

• Provide distributed tracing and monitoring capabilities

• Support different HW infrastructures

• Allow easy integration of new HW innovations
CarML: prototype demo

- www.carml.org
CarML: end-to-end system tracing demo

- 52.44.160.49:9411
CarML: an open platform to answer those challenges

• Deploy and benchmark machine learning frameworks and models across hardware infrastructures, through a common interface

  • An experimentation platform for ML users

  • A deployment platform for ML developers

  • A benchmarking platform for systems architects

• A distributed and resilient system where the web server, registry, tracer, and agents can all scale either horizontally or vertically
Thanks!

Suma Bhat  Minh Do  Deming Chen  Julia Hockenmaier  Wen-mei Hwu  Nam Sung Kim  Dan Roth  Rakesh Nagi  Lav Varshney
RAI’s current deployment setup
RAI architecture with reusable components

- CLI
- Middleware: Team Ranking, Execution Configuration, TA Grading
- Core: Tracing, PubSub, Queue, Storage, Logging, Database, Health Monitoring, Cache, Container Orchestrator, RAI Execution
- Hardware and Accelerators: CPU, GPU, FPGA

Team Ranking

Execution Configuration

TA Grading

Tracing

PubSub

Queue

Storage

Logging

Database

Health Monitoring

Cache

Container Orchestrator

RAI Execution

User1

User2

UserN

Hardware and Accelerators

CPU

GPU

FPGA

ECE ILLINOIS
CarML architecture: built on RAI

Web UI

CarML API
- RPC Gateway
- REST API
- Service Registry

RAI/Core
- Tracing
- K/V store
- Queue
- Storage
- Logging
- Dataset
- Health Monitoring
- Cache

Predictor Orchestrator

Hardware and Accelerators
- CPU
- GPU
- FPGA
- ASIC

CarML Predictors
- Caffe
- Caffe2
- TF
- MXNet
- PyTorch
- Word2Vec

ECE ILLINOIS
Cognitive benchmarks optimized for POWER

- Motivation: demonstrate the value of a well-balanced CPU + accelerators design for many important workloads
- Chai (Collaborative Heterogeneous Applications for Integrated-architecture)
  - Identified a set of common collaborative computation patterns
  - Demonstrated benefits of having CPU + accelerators for those patterns
    - Primary on AMD Kaveri A10-7850K APU
  - Open sourced a set of benchmarks to evaluate various CPU + accelerators architectures

- On-going: add more cognitive-related benchmarks + release an optimized version for POWER systems

<table>
<thead>
<tr>
<th>Collaboration Pattern</th>
<th>Short Name</th>
<th>Benchmark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Partitioning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td>Bázier Surface</td>
<td></td>
</tr>
<tr>
<td>CEDD</td>
<td>Canny Edge Detection</td>
<td></td>
</tr>
<tr>
<td>HSTI</td>
<td>Image Histogram (Input Partitioning)</td>
<td></td>
</tr>
<tr>
<td>HSTO</td>
<td>Image Histogram (Output Partitioning)</td>
<td></td>
</tr>
<tr>
<td>PAD</td>
<td>Padding</td>
<td></td>
</tr>
<tr>
<td>RSCT</td>
<td>Random Sample Consensus</td>
<td></td>
</tr>
<tr>
<td>SC</td>
<td>Stream Compaction</td>
<td></td>
</tr>
<tr>
<td>TRNS</td>
<td>In-place Transposition</td>
<td></td>
</tr>
<tr>
<td>Task Partitioning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fine-grain</td>
<td>RSCT</td>
<td>Random Sample Consensus</td>
</tr>
<tr>
<td>TQ</td>
<td>Task Queue System (Synthetic)</td>
<td></td>
</tr>
<tr>
<td>TQH</td>
<td>Task Queue System (Histogram)</td>
<td></td>
</tr>
<tr>
<td>Coarse-grain</td>
<td>BFS</td>
<td>Breadth-First Search</td>
</tr>
<tr>
<td></td>
<td>CEDT</td>
<td>Canny Edge Detection</td>
</tr>
<tr>
<td></td>
<td>SSSP</td>
<td>Single-Source Shortest Path</td>
</tr>
</tbody>
</table>
The POWER Minsky with NVLink GPUs (or CAPI FPGA) is so cool. Can I learn how to program them?

I’m a big fan of accelerator technologies. How can I educate my students/peers about it at scale?

I have a great Open Source project. How can I make use of accelerators in the cloud?


RAI: built on Many Existing Open Source Projects

<table>
<thead>
<tr>
<th>Services</th>
<th>Available Backends</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>Secret, <strong>Auth0</strong></td>
</tr>
<tr>
<td>Queue</td>
<td>NSQ, <strong>SQS</strong>, Redis, Kafka, NATS</td>
</tr>
<tr>
<td>Database</td>
<td>RethinkDB, MongoDB, MySQL, Postgres, SQLite, ...</td>
</tr>
<tr>
<td>Registry</td>
<td>Etcd, Consul, BoltDB, Zookeeper</td>
</tr>
<tr>
<td>Config</td>
<td><strong>Yaml</strong>, Toml, JSON, Environment</td>
</tr>
<tr>
<td>PubSub</td>
<td>EC, <strong>Redis</strong>, GCP, NATS, SNS</td>
</tr>
<tr>
<td>Trace</td>
<td>XRay, Zipkin, StackDriver, Jaeger</td>
</tr>
<tr>
<td>Logger</td>
<td><strong>StackDriver</strong>, <strong>JournalD</strong>, Syslog, Kinesis</td>
</tr>
<tr>
<td>Store</td>
<td><strong>S3</strong>, Minio, Memfs, LMDB</td>
</tr>
<tr>
<td>Container</td>
<td><strong>Docker</strong></td>
</tr>
<tr>
<td>Serializer</td>
<td>BSON, <strong>JSON</strong>, YAML, JSONPB, Python Pickle</td>
</tr>
</tbody>
</table>
D4P Architecture: built on top of RAI
ML/DL ecosystem personas: users

Business innovator

There are so many cool DL models. Which one will work (or will there be one working) for my data?

IT support for business

I just heard of a new wonderful DL model published on arXiv/github. Can it really achieve that impressive results?

ML enthusiasts

What hardware (performance, energy, cost) should I buy to support the desired business logic for adopting DL models / algorithms?
CarML: model researchers

I just published such a wonderful DL model. How can I let the world to try it without me providing too much support (documentation)?

I heard people are using my DL models. Does it work all the time? If not, what can I do to improve my model for interesting scenarios?

How does my model compare against the latest models that are constantly popping up from almost everywhere?
ML/DL ecosystem personas: system researchers

Ai is the future, and ML/DL will be a key workload. How can I characterize those workloads (with so many models and frameworks) on my HW systems?

I have designed a new wonderful HW system. Will it work seamlessly and wonderfully for those existing ML/DL models?

People are complaining my systems not performing for their DL models. How can I easily repeat the same experiment as them?
CarML: workflow explained

1. A use selects models and inputs through web UI or API.

2. The web server accepts the user inputs and interacts with registry, tracing, and agents.

3. On bootup, agents advertise the models in the registry. The web server forwards the user request to the agents capable of evaluating the model.

4. The agent starts a docker container for the request. Data that is shared across executions are mounted as a shared volume.

5. Within the docker container, the model is downloaded, loaded into memory, and the user's inputs are preprocessed.

6. Inference is performed within the docker container and the results are sent back to the user.

The web server queries the agent registry to retrieve the address. The tracing process is started, if profiling is enabled.