Hydrogen - Towards Elastic Management of Reconfigurable Accelerators

Paul Grigoras, Max Tottenham, Xinyu Niu, Jose G.F. Coutinho and Wayne Luk

August 28th, 2014

Overview

1. **Future reconfigurable accelerators (RAs)** - in *elastic* cloud
   - need to experiment with *scheduling* and *scaling policies* to adapt resource management for RAs
2. **Hydrogen** - prototype with *scheduler* and *elasticity manager*
   - provides high-level front-end and commercial backend
   - *pluggable* scheduling and scaling policies
3. **Evaluation** - Maxeler system with 4 RAs
   - *estimated* 38X faster for bond option pricing

Current Direction

Current Direction - Advantages of RAs

1. **Performance** - speedup, predictability (specific applications)
   - required to meet Service Level Objectives & Agreements for cloud applications
2. **Energy Efficiency** - reduced power consumption
   - reduced operating cost for cloud owners
3. **Flexiblity** - can reconfigure to meet demands
   - support a wide range of applications with few devices

Current Direction - Applications

1. **Finance** - Modelling[1], trading[2]
2. **Scientific Computing** - Climate and weather modelling[3]
3. **Bioinformatics** - short read mapping[4]
4. **Imaging and Visualisation** - medical imaging[5], seismic imaging[6]
5. **Neuromorphic engineering + machine learning** - Spiking neural models[7]

---

[1] Tse et al. *Design Exploration of Quadrature Methods in Option Pricing*
[2] Wray et al. *Exploring Algorithmic Trading in Reconfigurable Hardware*
[3] Gan et al. *Global Atmospheric Simulation on Reconfigurable Platform*
[4] Arram et al. *Reconfigurable Acceleration of Short Read Mapping*
[5] Jiang et al. *FPGA-based Computation of Free-form Deformations in Medical Image Registration*
[6] Niu et al. *Exploiting Run-time Reconfiguration in Stencil Computation*
[7] Cheung et al. *Large-Scale Spiking Neural Network Accelerator for FPGA Systems*

Current Direction - Limitations

1. Steep learning curve
   - substantially different from software
2. Slow development cycle
   - compilation can take days
3. Limited runtime management
   - single tenant devices $\Rightarrow$ reduced utilisation
4. Large initial investment
   - large chips are expensive

Future Direction

Future Direction - RAs in the Cloud

Cloud Computing can provide

1. high level APIs
   - reduced development time
2. libraries of pre-compiled implementations
   - zero compilation time
3. runtime systems for managing multi-tenancy
   - enables sharing $\Rightarrow$ increased utilisation
4. reduced initial investment and commitment
   - simplify adoption of RAs

Challenge - Enabling Elasticity for RAs

Cloud Computing requires *elasticity* to address the dynamics between two objectives:

- **Clients** - run applications fast and cost effective
- **Providers** - maximise profits by increasing resource utilisation and reducing power consumption

Elasticity

The degree to which the resources provisioned to a specific task match its demand[8].

[8] Herbst et al., *Elasticity in cloud computing: What it is, and what it is not.*

An Elastic System

Components of an elastic system:

1. **resource manager** - implements *scheduling policies*
   - makes low-level resource allocation decisions
   - provides monitoring information to assist the elasticity manager
2. **elasticity manager** - implements *scaling policies*
   - monitors feedback from resource managers
   - provides resources to closely meet the demand

Hydrogen

Towards Elastic Management of RAs

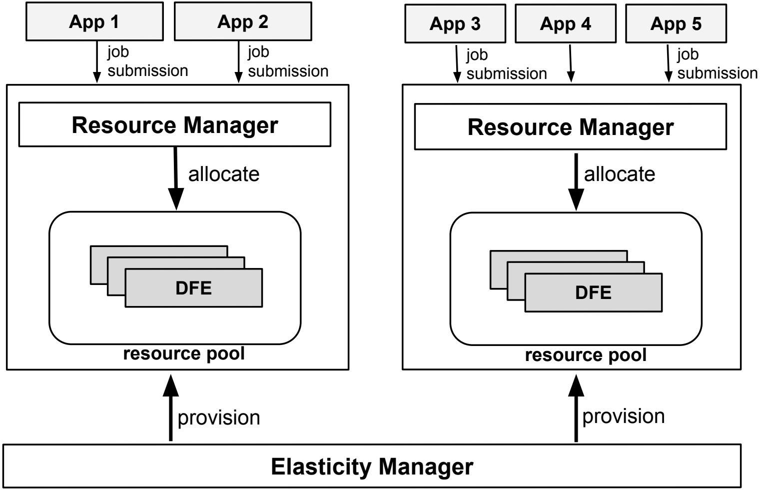
Hydrogen

A new lightweight framework for exploring elastic management of reconfigurable accelerators.

In a nutshell

- enables experimentation with scheduling and scaling policies
- provides a high-level API for reconfigurable implementations
- provides a back-end for execution on a commercially available FPGA system - Maxeler Dataflow Engines (DFEs)

Hydrogen - System Overview



Hydrogen - Jobs and Job Level Objectives (JLOs)

Jobs

- requests for computation (e.g. convolve, linear_solve)
- submitted via Remote Procedure Call (RPC) services
- each RPC corresponds to a reconfigurable implementation

Job Level Objectives (JLOs)

- each job has assigned a JLO (by the client)
- objectives to be satisfied by the scheduler
- e.g. target execution time

Hydrogen - Scheduling Strategies

- Permits flexible selection of scheduling strategies => facilitates experimentation with various scheduling policies
- *Managed Mode* runs several scheduling algorithms and scores the allocations based on a *cost function*

```
function MANAGER(queue)
    for Alg ∈ SchedulingAlgorithms do
        allocations[a] ← Alg(queue, WindowSize)
    end for
    for alloc ∈ allocations do
        scores[alloc] ← score(alloc)
    end for
    SelectedSchedule ← selectMaxScore(scores)
    ElasticityManager(SelectedSchedule)
end function
```

Hydrogen - Elasticity

Based on current execution schedule:

1. compute JLO for each job (j_i)

```
Job # getJlo(int resourceCount) {
    return (targetTime - predTime / resourceCount);
}
```

2. aggregate for entire job set
   - $jloMetric = \min(\max(\sum_{j_i>0} j_i - \beta, 0), \sum_{j_i<0} j_i)$
3. adjust pool size
   - $jloMetric < 0 \Rightarrow$ increase pool
   - $jloMetric > \beta \Rightarrow$ decrease pool

Hydrogen - Components

1. **Scheduler** - *resource manager*
   - uses a *library* of algorithms for producing execution schedules
   - allocates jobs to fixed set of provisioned resources

```
Allocations *FCFSMin(Scheduler &s) {...}

int main() {
  Scheduler s(...);

  /* Add some scheduling algorithms */
  s.addSchedAlg(FCFSMax);
  s.addSchedAlg(FCFSMin);
  ...
  s.start();
}
```

Hydrogen - Components

2. **Elasticity Manager** - invoked by scheduler, adjust pool size

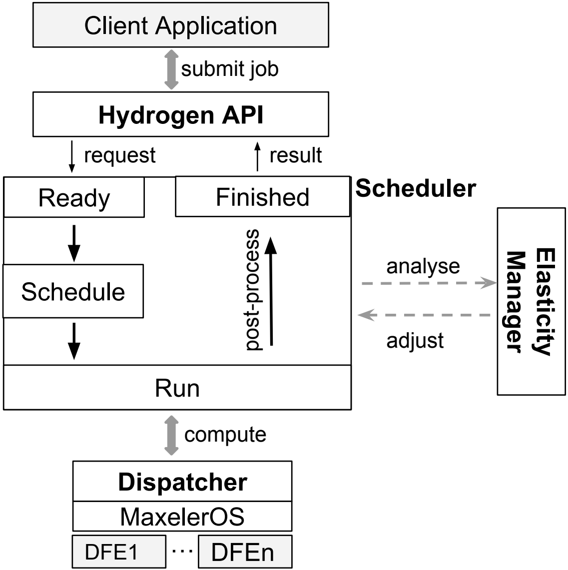
```
class MyElasticityManager : public ElasticityManager {
 void updateResourcePool(Scheduler s&, Allocations a&) {
    auto j = a.getJLOMetric();
    if (j < 0)          s.provisionResource();
    else if (j > beta) s.deprovisionResource();
  }};

int main() {
  auto elasticityManager = MyElasticityManager();
  Scheduler s(elasticityManager, ...);
  ...
}
```

Hydrogen - Components

3. **Dispatcher**
   - thin layer on top of MaxelerOS that has direct access to the DFEs (and other computer resources) it manages
   - runs requests on available resources using a reconfigurable implementation library which it manages directly
4. **Implementation Library**
   - efficient reconfigurable designs
   - performance metrics (measured and estimated)
   - scalability information (resource and topology requirements)
5. **Client Interface**
   - RPC interface through which clients submit compute jobs

Hydrogen - Components



Evaluation

Hydrogen - Experimental Setup

Hardware

- Maxeler MaxNode System
- 4 Maxeler DFEs, Virtex 6, 24GB RAM, PCIe connection
- Intel Xeon X5650 @2.67GHz, CentoOS 6.4, MaxelerOS 2013.1

Hydrogen Components

- run *locally* on the MaxNode
  - optimistic scenario - ignores network overhead
- *decoupled* - communicate via socket IO (Boost ASIO)
- C++ 11, `g++ 4.7.3 -O3 -march=auto`

Hydrogen - Experimental Setup

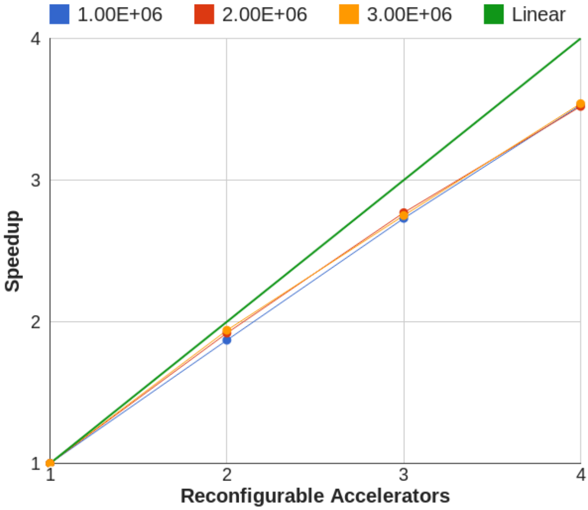
Application

- Monte Carlo design for bond options pricing
  - OpenMP re-implementation of (Jin et al., ARC 2012)
  - random number generator optimised for RAs [9]
  - 20.25% LUTs, 13.59% FFs, 9.40% BRAMs and 6.75% DSPs
- runs on any number of RAs as provisioned by *Hydrogen*
- operates in a *map-reduce* fashion:
  - all RAs are configured and stream data in parallel (*map*)
  - merging is done on the CPUs of the host system (*reduce*)
  - result is returned: dispatcher ⇒ scheduler ⇒ client

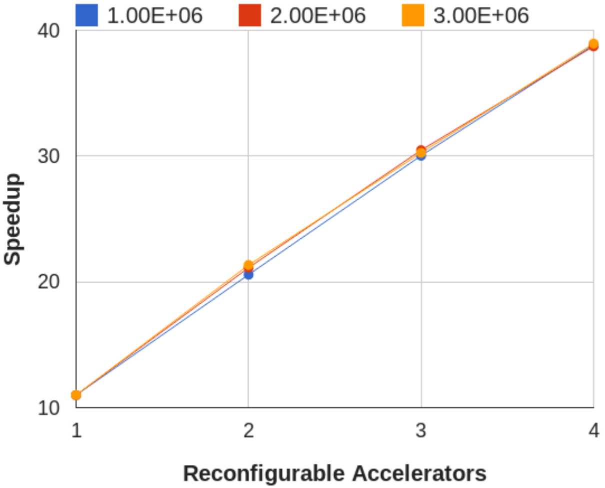
---

[9] D.B Thomas et al, *High quality uniform random number generation using LUT optimised state-transition matrices*

Hydrogen - Scalability of the Option Pricing Design



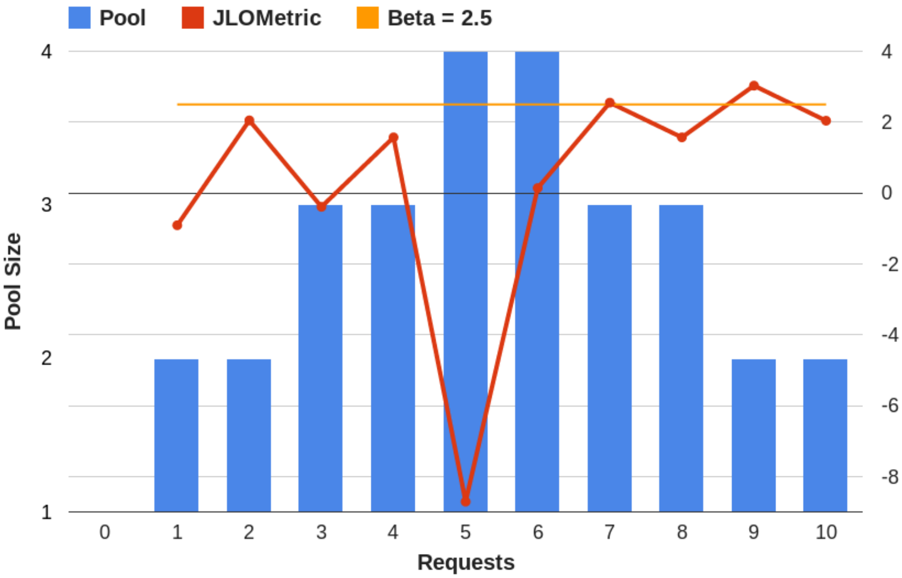
Hydrogen - Speedup Compared to 4-core i7-870[10]



[10]Qiwei et al. *Multi-level Customisation Framework for Curve Based Monte Carlo Financial Simulations*

Hydrogen - Framework Elasticity ($\beta = 2.5$)

Paths	Target (s)	Expected (s)	jloMetric	Pool	Decision
1	5	5.91	-0.91	1	Scale Up
1	5	2.96	2.05	2	Preserve
2	5.5	5.89	-0.39	2	Scale Up
2	5.5	3.93	1.57	3	Preserve
3	9	17.71	-8.71	3	Scale Up
3	9	8.86	0.14	4	Preserve
2	5.5	2.95	2.55	4	Scale Down
2	5.5	3.93	1.57	3	Preserve
1	5	1.97	3.03	3	Scale Down
1	5	2.96	2.04	2	Preserve


Pool
JLOMetric
Beta = 2.5
Pool Size
Requests
4
3
2
1
0
1
2
3
4
5
6
7
8
9
10
4
2
0
-2
-4
-6
-8

Future Work

- run-time reconfiguration overhead is significant
  - must be included in scheduling and scaling policies
- support preemption
  - required to ensure fairness, but expensive to implement
  - FPGAs do not normally support rapid preemption
- further experimentation with scheduling algorithms and JLO metrics
- extend to cover additional applications (Niu et al. *Dynamic Stencil: Effective exploitation of run-time resources in reconfigurable clusters*)

Conclusion

1. **Current reconfigurable applications**
   - often single device, single-tenant
2. **Future RAs** - in *elastic* cloud
   - need to experiment with *scheduling* and *scaling policies* to adapt resource management for RAs
3. **Hydrogen**[11] - prototype with *scheduler* and *elasticity manager*
   - provides high-level front-end and commercial backend
   - *pluggable* scheduling and scaling policies
4. **Evaluation** - Maxeler system with 4 RAs
   - *estimated* 38X faster for bond option pricing

[11] `https://github.com/custom-computing-ic/elastic-dfe-dispatcher`