Tutorial: Evolvable Hardware
ARCS 2008 - Architecture of Computing Systems

Jim Torresen and Kyrre Glette
University of Oslo, Norway

Marco Platzner and Paul Kaufmann
University of Paderborn, Germany

Outline
Slot 1: 09:30-11:00 (Jim and Kyrre)
- Introduction to evolvable hardware and examples of real-world applications
- Run-time evolvable system-on-chip

Slot 2: 11:30-13:00 (Marco and Paul)
- Further details on architectures for evolution.
- Hands-on experiments with the MOVES framework.

Principles of the Nature
- Evolution: Biological systems develop and change during generations.
- Development: By cell division a multi-cellular organism is developed.
- Learning: Individuals undergo learning through their lifetime.

How do we evolve a circuit?
Input/Output Specification → Genetic Algorithm → Circuit

Evolution
- Biological evolution:
  - Lifeforms adapt to a particular environment over successive generations.
  - Combinations of traits that are better adapted tend to increase representation in population.
  - Mechanisms: Selection+Crossover, Mutation and Survival of the fittest.

- Evolutionary Computing (EC):
  - Mimic the biological evolution to optimize solutions to a wide variety of complex problems.
  - In every new generation, a new set of solutions is created using bits and pieces of the fittest of the old.
  - Several algorithms are available including Genetic Algorithms (GA) and Genetic Programming (GP)

Possibilities of Evolvable Hardware (EHW)
- Two main directions:
  - Tuning/optimizing parameters of a circuit or system (also called evolutionary circuit design).
  - Evolving a system from scratch.

- New features provided:
  - Run-time adaptive hardware
  - Self repairing hardware

- Goal:
  - Make better computing systems than traditional architectures for real-world applications.
EHW Applied to Real-World Applications

- Analog
  - Adaptive Equalizer
  - Amplifier and Filter Design
  - Analog Circuit Synthesis
- Parameter Tuning
  - Clock Timing Adjustment
  - Analog Filter Tuning
- Robot Control
  - Image processing
  - Image Compression
  - Image Filtering
  - Image Recognition
  - Classification/Recognition
  - Sonar Classification
  - Gene Finding
  - Prosthetic Hand

Evolving a Circuit

- Define the basic building blocks:
  - Digital (gates or higher level functions)
  - Analog (transistor/resistor/L/C)
- Represent each building block in the chromosome by its function and its connections to other building blocks in the circuit.

Evolutionary Operators: Genetic Algorithms (GA)

- Crossover
- Mutation

Fitness

- A measure of how well adapted an individual is.
- The value determines the probability of being selected for reproduction.
- The way the fitness function is constructed is critical for the GA performance.
Computing Circuit Fitness

Fitness = \sum \sum (match of output o for pattern p)

Fitness Computation OFFLINE (extrinsic)

Fitness Computation ONLINE (intrinsic)

Online adaptation in Real-Time
- Static Evolution: Evolutionary Circuit Design
  - One-time evolution
- Dynamic Evolution: Evolvable Hardware
  - Problem specification (environment) is unknown at design time.
  - Problem specification changes during operation.
  - Goal: Continuous and autonomous evolution in a dynamically changing environment.

Digital Target Hardware
SRAM-based Reconfigurable Logic Devices
- PLA: Programmable logic array (Kajitani, 1999)
- FPGA
Some EHW Research Groups

- **EUROPE:**
  - EPFL, Switzerland (Mange, Floreano): Embryonic Electronics
  - Brno University of Technology, Czech Republic (Sekanina): Evolvable Components
  - Norwegian University of Science and Tech (Haddow): Developmental EHW
  - Univ. of Heidelberg, Germany (Meier): Field Programmable Transistor Array
  - Univ. of Sussex, UK (Thompson): Biological Inspired Architectures
  - University of Oslo (Torresen): Incremental evolution of hardware
  - Technical University of Catalunya, Spain (Moreno): Multicellular electronic tissue

- **JAPAN:**
  - AIST, Japan (Higuchi): EHW for Real World Applications
  - Tsukuba University (Yasunaga): EHW for Real World Applications

- **USA:**
  - Jet Propulsion Lab (Stoica): EHW for space applications
  - Stanford University (Koza): Analog Circuits by GP

- A lot of new groups have recently been established.

EHW Conferences

- **International Conference on Evolvable Systems (ICES), Springer LNCS.**
  - Next conference: Prague, Czech Republic
  - Paper deadline: March 19, 2008

- **NASA/ESA Conference on Adaptive Hardware and Systems (AHS), IEEE.**
  - Next conference: Noordwijk, The Netherlands
  - Paper deadline: January 31, 2008

---

**Prosthetic Hand Control** (Kajitani, 1999)

- Neuron
- Sensor
- Skin
- Muscle

Myoelectric signals are detected on the surface of the skin by using surface electrode (sensor).

**Post-Fabrication Clock Timing Adjustment** (Takahashi, 1999)

1. Design
2. Fabrication
3. Adjustment
4. Shipping

Higher operating yield <less power dissipation > Design clock speed

**Gene Finding** (Yasunaga, 2003)

Protein

Transcription Region or Gene

Gene

DNA
Gene Finding (Yasunaga, 2003)

Image Filtering (Sekanina, 2003)

Examples of functions implemented in the Programmable Elements (PE) (Sekanina, 2003)

Virtual reconfigurable circuit (Sekanina, 2000)

Evolutionary design of shot noise filters (Sekanina, 2003)
Limitation of Evolving Systems

Chromosome string representation

- Small and simple problem
  - 10010110101
  - Small system

- Large and complex problem
  - 10010110110110001001
  - Large system

Increase Scalability of EHW

- Compressing the chromosome string:
  - Variable length chromosomes.
- Dividing the evolutionary algorithm:
  - Apply parallel processing.
- Development:
  - The chromosome represents developmental rules.
  - Increasing the building block complexity:
    - Function level evolution.
- Dividing the application:
  - Incremental evolution.

Dividing the application

- Idea: Evolve a system gradually (divide-and-conquer the application).
  - Partitioned training vectors
  - Partitioned training set

Benefits:
- Simpler and smaller search space for the evolution.

Earlier Architecture: Prosthetic Hand

Classification Applications

- Face Image Recognition
- Sonar Return Classification
- Prosthetic Hand Sensor Signal Classification
System Overview

Classification Module

Classification Module

Functional Unit

FU HW Implementation

Example Application:
Face Image Recognition

AT&T Database of Faces
(formerly The ORL Database of faces)

Classify images of 40 different persons
10 different images of each face
### Input Reduction

- Original image
- Preprocessing
- Resampled image
- 92x112 8-bit pixels resampled to 8x8 8-bit pixels
- Input pattern: 512 bits

### Example: One FU Row

Evolution determines:
- Selection of pixels
- Expressions

### Evolution and Fitness Function

- FU: Pixel address (6 bit) Function (1 bit) Constant (8 bit)
- FU row: \( FU_1 (15b) \), \( FU_2 (15b) \), ..., \( FU_n (15b) \)

- One FU row can be evolved at a time
- Fitness function emphasizes positive matches of the category being evolved at the moment.

### Architecture Parameters

- More FU rows within a CDM gives a higher chance of correct classification
- Each FU row evolved with random initial values – gives different “rules”
- The more FU rows in a CDM, the higher the output resolution
- Too few or too many FUs in a row make it more difficult to evolve a classifier

### Classification Accuracy

- Using 6 FUs per row and 10 rows per CDM
- Using \( x \) training vectors per category and the rest (10-\( x \)) as test vectors
- 96.25% accuracy using 9 training vectors per category
- Better than previous offline EHW architecture
- Competitive to Eigenfaces or Fisherfaces, SVM (Support Vector Machine) performs better
- 8 parallel evaluations
- Fitness evaluation alone is 2.12 times faster in HW
- GA is not fully optimized for the PowerPC

**Implementation (Sonar)**
- Implemented on a Xilinx Virtex-II Pro XC2VP30 (XUP2VP)
- Classification module:
  - 0.5 µs classification time per pattern (@118 MHz), using time multiplexing
- Evaluation module: 371 slices (2%)
  - Parallel evaluation (8 individuals): 1393 slices (10%)

**Conclusions 1**
- High classification speed
  - Suitable for problems requiring high throughput
- High classification accuracy
  - Comparable or better than other methods (except SVM)
- Incremental evolution
  - Makes evolution of a large system possible
  - Evolution time is short
  - The evaluation module requires few FPGA resources

**Conclusions 2**
- Run-time reconfigurable architecture
  - Allows for online evolution in an on-chip system
  - Suitable for adaptation to a changing fitness function/training set
- Combination of SW/HW on-chip gives benefits
  - Evolution time can be kept down with HW fitness evaluation
  - The rest of the GA can run in SW, allowing for easy modifications
  - A compact system (SoC)
Selected references (others work)


Introduction to EHW:


Thank you!

- Please feel free to contact us after this tutorial.
- One PhD position will be available soon in our group (request announcement by sending Jim an e-mail).

E-mail: {kyrrehg,jimtoer}@ifi.uio.no

http://www.ifi.uio.no/~kyrrehg
http://www.ifi.uio.no/~jimtoer
**Hardware Representation Models - CGP (1)**

- Cartesian Genetic Program (CGP) [Miller&Thomson ‘00]
  - array of combinational blocks connected by feed-forward wires

- parameters:
  - $n_i$ ... number of primary inputs
  - $n_c$ ... number of columns
  - $n_r$ ... number of rows
  - $n_{li}$ ... number of logic block inputs
  - $l$ .......levels back
  - $F$ .... set of logic block functions
  - $n_f$ ... number of logic block functions

- example: 2+2 adder
  - $n_i=4$, $n_o=3$, $n_c=6$, $n_r=4$, $n_{li}=2$, $l=3$, $n_f=9$,
  - $F=\{\text{AND, NAND, OR, NOR, XOR, XNOR, NOT, 0, 1}\}$
  - chromosome length = 75 genes (numbers) = $n_c * n_r * (n_{li} + 1) + n_o$
  - chromosome $G = \{2082073160214 // column 1$
  - $38168565557 // column 2$
  - $775566218196 // column 3$
  - $1066151114101065578 // column 4$
  - $1312719814141968198 // column 5$
  - $2115141320716151421137 // column 6$
  - $192026 // outputs$
**Hardware Representation Models - CGP (3)**

- **CGP characteristics**
  - Constant-size chromosome defines configuration of the logic block array
  - Close to reconfigurable hardware architectures, as logic block placement is implicitly encoded (but no routing)
  - Widely-used model for hardware evolution
  - Blocks that do not contribute to the function remain in the chromosome

- **CGP variants**
  - One-row model
    - \( n_r = 1, l = n_c \)
    - Gives a more compact hardware representation
    - Does not encode placement
  - Functional-level models
    - Coarse-granular logic blocks, e.g., adders, shifters, comparators
    - Bus-based interconnect

---

**Hardware Representation Models - ECGP**

- **Embedded CGP (ECGP)** [Walker & Miller ‘04]
  - Automatic module creation in the one-row CGP model
  - Compress operator creates a new module by aggregating all blocks between two randomly chosen blocks
  - Expand operator dissolves modules
  - Module description is appended to the chromosome
  - Modules can be instantiated by the mutation operator
  - No modules-in-modules

---

**Hardware Representation Models - Tradeoffs**

**Hardware Representation Models - Evolutionary Algorithms**

- Conventional single-objective genetic algorithm
  - Uses elitism, binary tournament selection, recombination, mutation

---

**Evolutionary Algorithms**

- Evolutionary Algorithms
  - Conventional single-objective genetic algorithm
    - Uses elitism, binary tournament selection, recombination, mutation
Evolutionary Algorithms – Operators

- **selection**
  - binary tournament

- **recombination (crossover)**
  - applied with a certain rate
  - \(n\)-point crossover
  - uniform crossover

- **mutation**
  - single-point mutation modifies one gene, i.e., a node function or a wire
  - mutation rate determines how many single-point mutations are performed for a chromosome

Evolutionary Algorithms – Fitness Evaluation

- **functions with correctness property**
  - correct outputs are known
  - fitness measures deviation from correct outputs, e.g., using the hamming distance
  - examples are arithmetic and logic functions
  - popular test functions for comparing representation models and algorithms

- **functions with functional quality**
  - there is no correctness measure
  - the functional quality depends on unknown and possibly changing input data
  - examples are hashing functions, classifiers, robot navigation controllers, …
  - target for self-adaptive evolutionary systems

Evolutionary Algorithms – Example Experiment

- **2+2 adder**
  - population size = 100, 2-point crossover rate = 0.9, single-point mutation rate = 0.02
  - fitness metrics
    - \( f(c) = \frac{1}{1 + \frac{1}{2^n} \sum_{i=0}^{2^n-1} \sum_{j=0}^{2^n-1} \text{ham}(i + j, c(i, j))^2} \)
  - 10 experiment runs

Multi-objective Optimization

- **optimization for multiple objectives**
  - often, the objectives are conflicting which leads to compromises
  - possible motivations for multi-objective hardware evolution
    - circuit area, speed, etc. are of interest for the application
    - improving the convergence and robustness of evolution

- **approaches**
  - two-stage fitness, e.g., first optimize for correctness and then for area
    - [Kolganova & Miller ‘99] [Coello Coello ‘00]
  - multi-objective evolutionary algorithms (MOEAs) generate Pareto fronts
    (set of non-dominated solutions)

- **goals for MOEAs**
  - fast convergence to “good” solutions
  - keep diversity in the Pareto front
Multi-objective Evolutionary Algorithms

- **SPEA2** [Zitzler et al. '01]
  - elitism: uses archive and population
  - selection is based on Pareto dominance
  - maintains diversity by thinning out Pareto front clusters with k-th nearest neighbor density estimation

- **TSPEA2** [Kaufmann & Platzner '07]
  - prioritizes one objective to speed up convergence
  - keep the diversity preserving mechanism

- Other MOEAs
  - NSGAII, μGA, μGA2, OMOEAII, IBEA

Area and Speed Objectives

- **area** and **speed estimation for the CGP model**

  \[
  area(c) = 1 - \frac{\text{used_blocks}(c)}{n_c \cdot n_r}
  \]

  \[
  speed(c) = 1 - \frac{\text{delay}(c)}{n_c + 1}
  \]

  \(\text{delay}(c)\) equals the maximum number of blocks on the longest path from any input to any output (set to \(n_c + 1\) if all outputs are unconnected)

- More accurate area and speed estimation
  - generate circuit netlist in JHDL and run Xilinx backend tools

MOEAs – Example Experiment (1)

- **hashing function used by** [Damiani et al. '98]
  - find a function that maps a set \(M\) of 16-bit keys to a set \(N\) (\(|N| < |M|\)) of indices in the most uniform way possible
  - functional quality metrics: 
    \[
    f(c) = \frac{1}{1 + \frac{1}{|N|} \sum_{j=1}^{|N|} ([\{j \in M, \ c(j) = i\}] - \frac{|M|}{|N|})^2
    \]

  \(|M| = 4096, \ |N| = 256 \rightarrow \text{CGP model with } n_i = 16, n_y = 8, n_b = 8, n_p = 4, l = 8, F = 4-LUT\)

MOEAs – Example Experiment (2)

- **functional quality after 1000 generations**
  - 10 experiment runs

<table>
<thead>
<tr>
<th></th>
<th>GA</th>
<th>SPEA2</th>
<th>TSPEA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>best</td>
<td>0.135</td>
<td>0.084</td>
<td>0.125</td>
</tr>
<tr>
<td>worst</td>
<td>0.094</td>
<td>0.075</td>
<td>0.092</td>
</tr>
<tr>
<td>average</td>
<td>0.114</td>
<td>0.079</td>
<td>0.110</td>
</tr>
</tbody>
</table>

- Development of the average functional quality
MOEAs – Example Experiment (3)

- Pareto front approximation
  - SPEA2 vs. TSPEA2, for a sample experiment run
  - GA: best functional quality, for 10 experiment runs

- quality vs. area
- quality vs. speed

goals
- experiment with evolutionary digital circuit design using...
  - different hardware representation models
  - different optimization objectives
  - different (multi-objective) evolutionary algorithms
  - derive comparative and reproducible results
  - conduct large-scale experiments

MOVES [Kaufmann & Platzner ‘07.2]
- modularized framework for evolvable hardware experiments
- interface to grid computing software
- project page: uni-paderborn.de/cs/ag-platzner/research/moves

tools can be downloaded (open source)

MOVES: A Modular Framework for Hardware Evolution

- separation of
  - hardware representation models,
  - evolutionary algorithms
  - and operators

- implemented
  - representation models
    - CGP, ECGP
  - algorithms
    - GA, SPEA2, TSPEA2, NSGAII, µGA, IBEA
  - operators
    - one / two / uniform crossover
    - single point mutation
    - binary tournament selection

MOVES Framework

- generic evolutionary algorithm modules
  - representation models
  - operators
  - evolutionary algorithms

- graphical user interface
  - test different experiment parameter settings
  - evaluate newly implemented methods
  - investigate optimization process

- batch mode tools
  - produce representative number of experiment runs
  - extract results for visualization

- evolutionary operators
  - representation model specific
  - functional quality / application specific
  - fitness / delay / compactness

- cross-over
  - CGP crossover
  - one point
  - uniform
  - single point
  - binary tournament selection

- mutation
  - single-point

Copyright (C) 2008, P.Kaufmann and M.Platzner
Evolvable Hardware Tutorial, ARCS’08
Experiment Configuration

- experiment configuration saved as text file
  - algorithm parameters
  - framework settings
  - termination conditions

- some parameters can be changed during the experiment runtime

```python
# evolve 7 bit parity
parity_width=7
ml=7
mr=1
nl=2
nr=7
mutationRate=0.01
crossoverRate=0.3
populationSize=100
selectClass=move_logic_TournamentSelection
crossoverClass=move_CSP_TwoPoint_Crossover
mutationClass=move_CSP_Mutation
init=Random
chromosomeClass=move_CSP_Chromosome
fitnessClass=move_CSP_Fitness
```

Graphical User Interface

- visualization of
  - chromosomes
  - search process
  - objectives
  - Pareto front

- tune experiment parameters
- stop / save / load / resume complete evolutionary process
- re-evaluation of saved experiments
- analysis tools
  - e.g. fitness plots, statistics, ...

Batch Mode Tools

- generation of Condor batch jobs for large-scale experiments
  - Condor is available for Windows and Unix-like systems
  - implemented migration of multithreaded Java-applications
- network monitor for the computing clients

Selected References

- [Miller & Thomson '00](#)

- [Miller et al. '98](#)

- [Walker & Miller '04](#)

- [Damiani et al. '98](#)

- [Kalogeropoulos & Miller '99](#)

- [Ceolia Ceolla '00](#)

- [Zitzler et al. '01](#)

- [Kaufmann & Platzner '07](#)

- [Kaufmann & Platzner '07](#)
MOVES Quick Start

- start MOVES: `./moves`
- instantiate a predefined experiment:
  
  ```
  Project → New → adder_single_objective_GA
  ```

- select a directory to store the results:
  ```
  ~/tmp
  ```

MOVES Quick Start

- start the evolution

- try different views, switch to the
  - statistics
  - snapshots pane

Evolution Experiments (1)

- evolve a parity function
  - this is rather difficult without XOR (odd parity) and XNOR (even parity)
  - step #1
    - instantiate the experiment `parity_single_objective_GA`
    - set the function set to `{AND, NAND, OR, NOR, XOR, XNOR, 1, 0, NOT}`
    - note the number of generations required to evolve a correct parity function
  - step #2
    - re-instantiate the experiment (Project → New from current parameters)
      and exclude (XOR, XNOR) from the function set
    - can you notice the difference?

- stop the evolution and switch to the snapshots pane
  - check out help on shortcuts and analyze the chromosome:
    - right-click on an object, left-click on an input
      - `<SHIFT>`<C>
      - `<SHIFT>`<V>
      - `<SHIFT>`<D>
      - `<SHIFT>`<O>
      - `<CTRL>`<O>
      - `<SHIFT>`<I>
      - `<CTRL>`<I>
      - `<SHIFT>`<N>
      - `<SHIFT>`<L>`
**Evolution Experiments (2)**

- evolve a multiplier
  - the higher the mutation rate, the closer we come to random search
  - step #1
    - instantiate and run the experiment `multiplier_single_objective_GA`
    - observe the development of the functional quality
  - step #2
    - stop and re-instantiate the experiment
    - adjust the mutation rate such that only one gene is modified and run the experiment
    - can you notice the difference?

- evolve a hashing function
  - instantiate and run the experiment `hash_function_TSPEA2`
  - switch to the statistics pane and toggle the `switch view` button
  - it seems some solutions (red and blue points) on the Pareto front are dominated - why is this?

**Application-specific Cache Controller (1)**

- goal: evolve an instruction cache mapping function \( m \) for `quicksort`, given an execution trace
  - the mapping function translates an instruction address to the cache line index of a direct-mapped cache with 16 cache lines, 1 word/block
  - the objective is to maximize the hit-rate: \( f = \frac{\text{#hits}}{\text{#executed instructions}} \)
  - the reference is the "modulo" mapping function (index = ins_addr[5:2], tag = ins_addr[15:6])

**Application-specific Cache Controller (2)**

- instantiate and run the experiment `cache_controller`
  - this experiment uses an instruction trace length of 174603 instructions
  - the `quicksort` program length is 96 instructions
  - the hit-rate for a cache using the "modulo" mapping function is 0.321

- feel free to change experiment parameters ....

Thank you for your attention!