

Mean-shift Clustering for Heterogeneous Architectures

Foutse Yuehgoh



Journée MAGI Calcul scientifique
VILLETANEUSE, 2017

July 5, 2017

Introduction

- 1 **Generality**
 - 2 **Heterogeneous Architecture**
 - 3 **Our Approach**
- ## Conclusion

Context

- Heterogeneous computing system.
- Parallel programming paradigm of Machine learning (ML).
- Need of specific construction languages to configure and reconfigure hardware according to the machine learning needs.

Problematic

- Slow hardware designs.
- Slow testing and evaluation time.
- Slow and expensive fabrication.
- Design cost dominate.

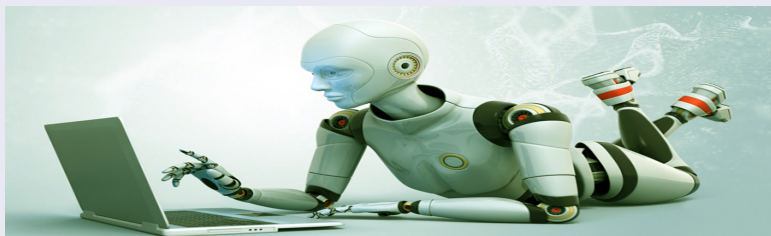
Objective

- Performance optimizations for heterogeneous machine learning computing.

Machine Learning

Building hardware or software that can achieve tasks such as:

- Extracting features from data in order to explore or solve predictive problems,
- Automatically learn to recognize complex patterns and make intelligent decisions based on insight generated by learning from examples.



Generality

Continues increase in data size



Consequences

Driven by the need for more explanatory power, tremendous pressure has been on ML methods to scale beyond a single machine, due to both space and time bottlenecks.

Complexity of algorithms

Their learning time grows as the size and complexity of the training dataset increases.



Fortunately, they have special properties which, if properly harnessed by a well-designed system, can yield a $10\times$ or even $100\times$ speed boost.

Importance

Efficient computational methods and algorithms

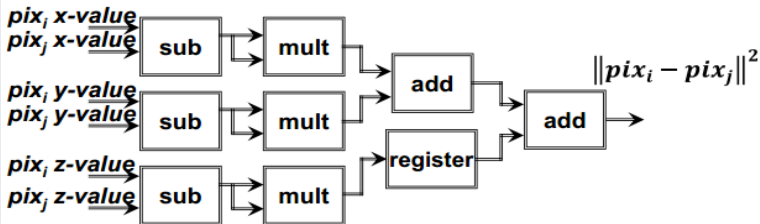
- On very large data sets,
- in parallel to complete the machine learning tasks in reasonable time.

Processing vast amounts of data simultaneously, is important in neural networks and machine learning algorithms, just as it is the case for the human brain.

Review of results

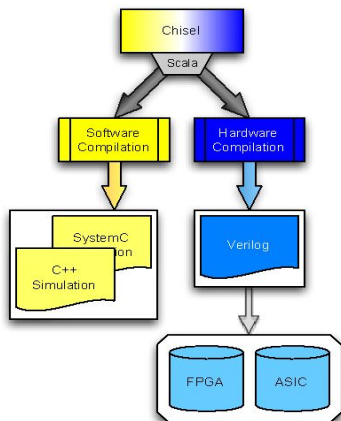
Speeding up multidimensional clustering in statistical data analysis has been the focal point of numerous papers that have employed different techniques to accelerate the clustering algorithms.

Example: Pipelined Euclidean distance hardware diagram



Constructing Hardware in a Scala Embedded Language (Chisel)

Chisel eases the building of simple, reliable, and efficient software to harness the parallelism inherent in FPGA technology.



Heterogeneous Architecture

Definition

A heterogeneous architecture is a novel architecture that incorporates both general-purpose and specialized cores on the same chip.

- Takes care of generic control and computation;
- Accelerates frequently used or heavy weight applications.

Challenge



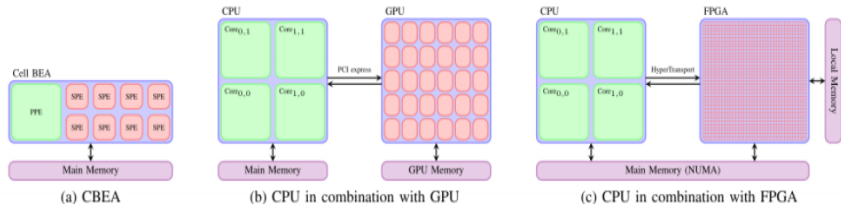
Unique diversity of technology



Heterogeneous Architecture

Example

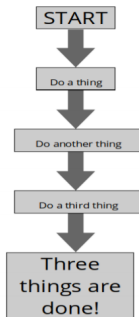
- (a) **CBEA:** Traditional CPU core and eight SIMD accelerator cores.
- (b) **GPU:** 30 highly multi-threaded SIMD accelerator cores in combination with a standard multicore CPU.
- (c) **FPGA:** An array of logic blocks in combination with a standard multi-core CPU.



Heterogeneous Architecture

Serial and Parallel Computing

Serial computing



Parallel computing

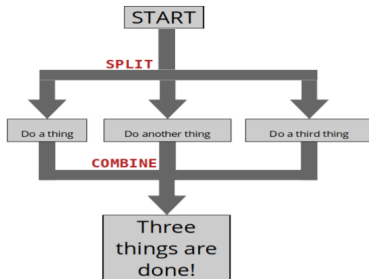
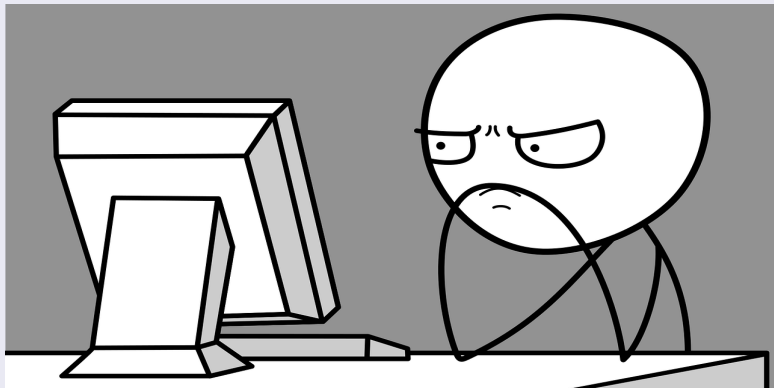


Figure 4: Serial and parallel computing, image by David Taylor

- Parallel computing enables us to solve problems that benefits from or need faster solutions and require large amount of memory.

Hardware difficulties



Our Approach

Strategy

Direct implementation of the algorithm on FPGA development board.



- How do we split and partition the ML program over a heterogeneous machines, and which best language do we use?

MEAN-SHIFT ALGORITHM

Goal: Produce clusters on input data.

- Fixes a window around each data point;
- Computes the mean of the data within the window;
- Shifts the window to the mean and repeat until we have convergence.

Key Property

An exceptionally appealing and non-parametric clustering techniques that deal without prior notion on the number of clusters.

Computing The Mean Shift

$$\nabla P(\mathbf{x}) = \frac{c}{n} \sum_{i=1}^n \nabla k_i = \frac{c}{n} \left[\sum_{i=1}^n g_i \right] \left[\frac{\sum_{i=1}^n \mathbf{x}_i g_i}{\sum_{i=1}^n g_i} - \mathbf{x} \right]$$

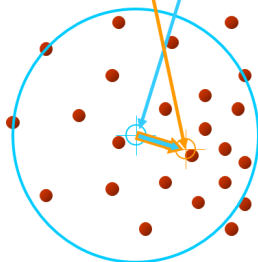
Yet another Kernel density estimation !

Simple Mean Shift procedure:

- Compute mean shift vector

$$\mathbf{m}(\mathbf{x}) = \left[\frac{\sum_{i=1}^n \mathbf{x}_i g \left(\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{h} \right)}{\sum_{i=1}^n g \left(\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{h} \right)} - \mathbf{x} \right]$$

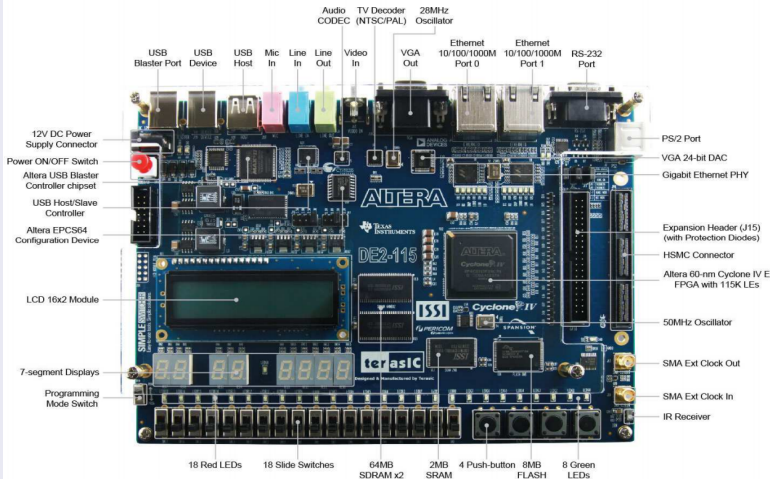
- Translate the Kernel window by $\mathbf{m}(\mathbf{x})$



$$g(\mathbf{x}) = -k'(\mathbf{x})$$

Field Programmable Gate Array (FPGA)

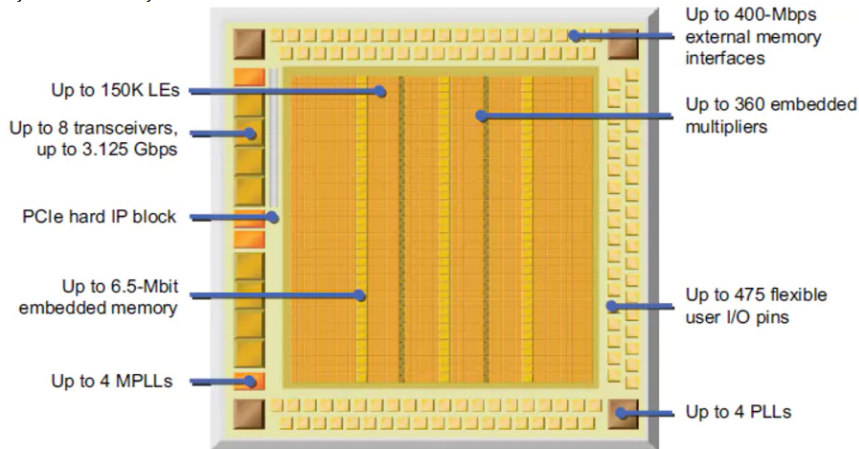
Altera Cyclone IV E FPGA DE2-115 Development kit



Field Programmable Gate Array (FPGA) :

Key Properties

Cyclone IV FPGA Key Architectural Features



Why FPGA?

Advantageous properties



Speed

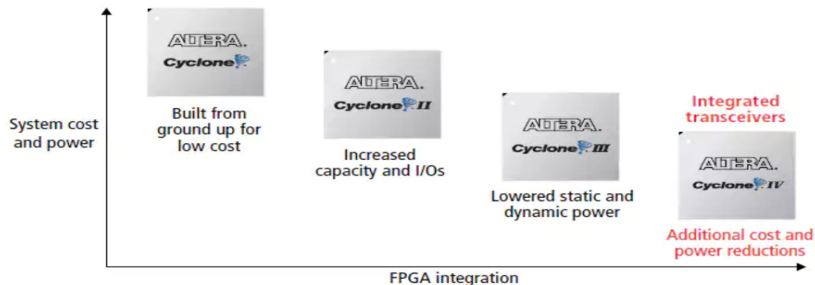


Cost

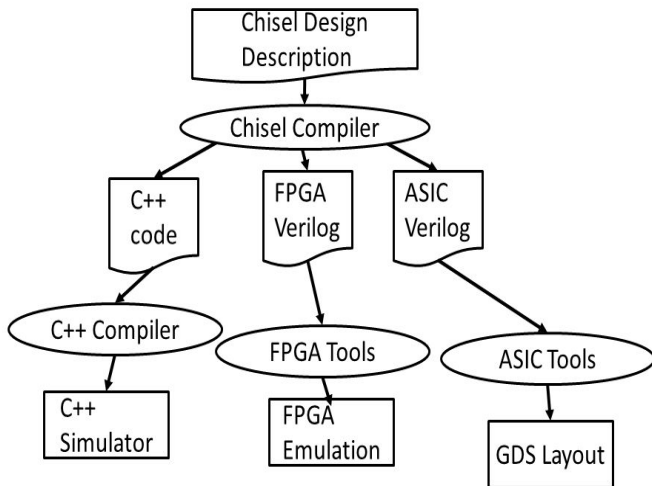


Parallelism

Integration Lowers Cost and Power



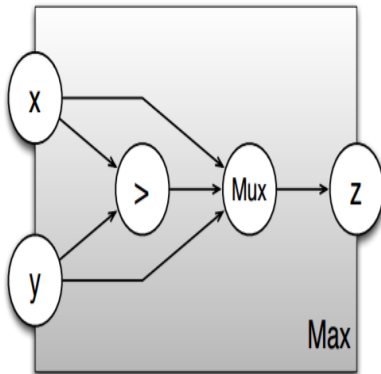
Important properties of Chisel



Why Chisel?

Maximum of a vector

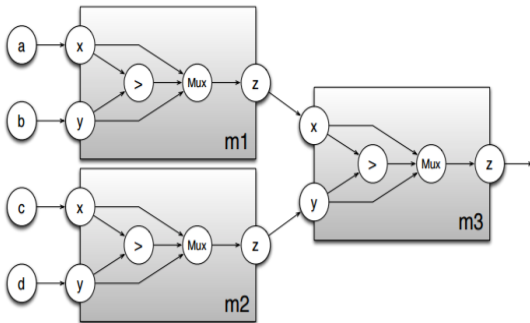
```
class Max2 extends Module {  
  val io = new Bundle {  
    val x = UInt(INPUT, 8)  
    val y = UInt(INPUT, 8)  
    val z = UInt(OUTPUT, 8) }  
  io.z := Mux(io.x > io.y, io.x, io.y)  
}
```



Why Chisel?

Manual Code parallelization

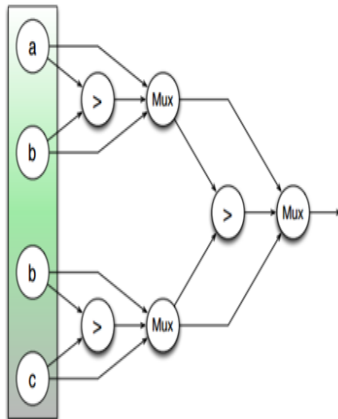
```
val m1 =  
  Module(new Max2())  
  m1.io.x := a  
  m1.io.y := b  
val m2 =  
  Module(new Max2())  
  m2.io.x := c  
  m2.io.y := d  
val m3 =  
  Module(new Max2())  
  m3.io.x := m1.io.z  
  m3.io.y := m2.io.z
```



Why Chisel?

Chisel Code parallelization

```
class MaxN(val n: Int, val w: Int) extends Module {  
  private def Max2(x: UInt, y: UInt) = Mux(x > y, x, y)  
  
  val io = IO(new Bundle {  
    val ins = Input(Vec(n, UInt(w.W)))  
    val out = Output(UInt(w.W))  
  })  
  io.out := io.ins.reduceLeft(Max2)  
}
```



Running the Chisel Simulation

Generated Verilog

```
`ifdef RANDOMIZE_GARBAGE_ASSIGN
`define RANDOMIZE
`endif
`ifdef RANDOMIZE_INVALID_ASSIGN
`define RANDOMIZE
`endif
`ifdef RANDOMIZE_REG_INIT
`define RANDOMIZE
`endif
`ifdef RANDOMIZE_MEM_INIT
`define RANDOMIZE
`endif

module MaxN(
  input          clock,
  input          reset,
  input  [6:0]  io_ins_0,
  input  [6:0]  io_ins_1,
  input  [6:0]  io_ins_2,
  input  [6:0]  io_ins_3,
  output [6:0]  io_out
);
  wire  _T_12;
  wire [6:0] _T_13;
  wire  _T_14;
  wire [6:0] _T_15;
  wire  _T_16;
  wire [6:0] _T_17;
  assign io_out = _T_17;
  assign _T_12 = io_ins_0 > io_ins_1;
  assign _T_13 = _T_12 ? io_ins_0 : io_ins_1;
  assign _T_14 = _T_13 > io_ins_2;
  assign _T_15 = _T_14 ? _T_13 : io_ins_2;
  assign _T_16 = _T_15 > io_ins_3;
  assign _T_17 = _T_16 ? _T_15 : io_ins_3;
endmodule
```

Errors in Chisel

- Type mismatches, syntax, (**Scala compiler errors**).
- Errors found during low level transforms and verilog emission (**Firrtl errors**).
- Illegal chisel but legal scala (**Chisel checks**).
- Underlying implementation crashed(**Java Stack Trace**).

Advantages with Chisel for hardware implementation [1]

Important advantages of Chisel

- Designer productivity with 3-stage 32-bit RISC processor reviles a **3× code reduction** with Chisel compared to hand written Verilog.
- Quality of results:

Source	Clock Period	Total Area	Logic Area
Chisel	7ns	62197 um^2	60801 um^2
Verilog	7ns	62881 um^2	61485 um^2
Chisel	2.5ns, Retimed	66472 um^2	61279 um^2
Verilog	2.5ns, Retimed	67034 um^2	62227 um^2

Advantages with Chisel for hardware implementation[1]

Important advantages of Chisel

- Simulator Speed with a (88, 291, 350 cycles total) Processor:





Simulator	Time (s)	Speedup
VCS RTL simulator	5390	1.00
Chisel C++ RTL simulator	694	7.77

Conclusion

- Chisel breaks the barrier between hardware and software thus is a good tool for hardware design.
- Rather than slowing-down data sender (during fast arrival of data in chunks), an FPGA allows you to add more processing "blocks" to accelerate the processing.
- An FPGA, for example, can process 10 data streams in parallel (i.e concurrently), whereas in software each stream would have to be buffered and processed one at a time.

Hence this approach is worth giving it a try!!!

Some References

-  Bachan, John *Constructing Hardware in a Scale Embedded Language, institution: Lawrence Berkeley National Laboratory, (2014)*
-  Bailey, Donald G and Johnston, Christopher T *Algorithm transformation for FPGA implementation, booktitle: Electronic Design, Test and Application, 2010. DELTA'10. Fifth IEEE International Symposium on, pages:77–81, organization:IEEE(2010).*
-  Han, Song and Kang, and others *ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA, organization:ACM,(2017).*
-  Liu, Shaoshan and RO, WON W and Liu, Chen and Cristóbal-Salas, Alfredo and Cérin Christophe and Han, Jian-Jun and Gaudiot, Jean-Luc, *INTRODUCING THE EXTREMELY HETEROGENEOUS ARCHITECTURE, publisher: World Scientific(2012)*

