High Performance Computing

Autumn, 2018

Lecture 20

Imperial College
London

Prasun Ray

11 December 2018

Notes

- Lab sessions this week are replaced with office hours
  - All office hours in MLC
- Homework 3 marks should be available Friday
- Final project marks and solutions will be available late January
- Please remember to fill out SOLE surveys (especially if you like the class!)
- I will also ask/collect (anonymous) feedback after project marks are done
- A (hopefully final) clarification on part 2.4 of final project was posted yesterday (bounds for theta* are same as the bounds of theta)

Today

- *Apache Spark*
- *GPU computing*
- *Google TensorFlow*
- *Class overview*

Large-scale cluster computing

- **Previously: discussed similarities between *mapReduce* and ideas covered in course**
- **If it is similar to older ideas, why is it important?**
  - **One important feature is *fault tolerance***
  - **When running on 1000's of cores, must expect and plan for hardware failure**
  - **mapReduce includes features that allow a job to keep running (or easily be restarted) when individual cores fail**

Large-scale cluster computing

- Previously: discussed similarities between *mapReduce* and ideas covered in course

- If it is similar to older ideas, why is it important?

  - One important feature is *fault tolerance*

  - When running on 1000's of cores, must expect and plan for hardware failure

  - mapReduce includes features that allow a job to keep running (or easily be restarted) when individual cores fail

  - Hadoop is an open-source tool which implements mapReduce on clusters (widely used for large-scale data processing)

  - Apache Spark is a newer cluster computing tool that builds on mapReduce and Hadoop

Apache Spark

- MPI + Fortran/c → large-scale, distributed-memory scientific computing
- Spark + Python (or Java, Scala, Julia, R) → large-scale distributed-memory data analysis
- *PySpark*: use python to work with large datasets on clusters with built-in fault tolerance
- Basic workflow:
  - Distribute data across cluster: create a Resilient Distributed Dataset (RDD)
  - Process (transform) the data in the RDD (in parallel):
    - *map*: apply an operation to each element of data, e.g. multiply each number by 2
    - *filter*: extract portions of dataset that satisfy some criteria, e.g. extract all even numbers
    - And there are other transformations as well

Apache Spark

- **Basic workflow:**
  1. **Distribute data across cluster: create a Resilient Distributed Dataset (RDD)**

  2. **Process (transform) the data in the RDD (in parallel):**
     - ***map*: apply an operation to each element of data, e.g. multiply each number by 2**
     - ***filter*: extract portions of dataset that satisfy some criteria, e.g. extract all even numbers**
     - **And there are a few other common transformations as well**
     - **Ultimately Original RDD is *transformed* into a new RDD**

  3. **Extract key results from tranformed RDD**
     - ***reduce(function)*: define python function which is used to reduce data**
       - **e.g. sum, min, max, product (as usual)**
       - **but can construct any function which takes two elements on input and produces one output value (function must be commutative and associative)**

Apache Spark

- **Basic workflow:**
  1. **Distribute data across cluster: create a Resilient Distributed Dataset (RDD)**

  2. **Process (transform) the data in the RDD (in parallel):**
     - ***map*: apply an operation to each element of data, e.g. multiply each number by 2**
     - ***filter*: extract portions of dataset that satisfy some criteria, e.g. extract all even numbers**
     - **And there are a few other common transformations as well**
     - **Ultimately Original RDD is *transformed* into a new RDD**

  3. **Extract key results from tranformed RDD**
     - ***reduce(function)*: define python function which is used to reduce data**
     - ***take*: collect 1st n elements**
     - ***collect:* returns all n elements (similar to gather)**

Apache Spark

Example

- **After downloading pre-built package for Spark: http://spark.apache.org/downloads.html)**
- **Launch pyspark on 2 local threads:**

```
$ ./bin/pyspark --master local[2]
```

- **Create data to be analyzed:**

```
>>> L = range(20)
```

- **Create RDD:**

```
>>> D = sc.parallelize(L)
>>> D
ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:423
```

- **D is automatically distributed across the 2 threads**

Apache Spark

Example

- **Create data to be analyzed:**

```
>>> L = range(20)
```

- **Create RDD:**

```
>>> D = sc.parallelize(L)
>>> D
ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:423
```

- **Define filter function (checks if input is even):**

```
>>> def is_even(x):
        return x % 2 == 0
```

- **Filter data:**

```
>>> Dnew = D.filter(is_even)
```

- **Collect data:**

```
>>> Dnew.collect()
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Apache Spark

Example

- **Can also reduce data using:**
  - Dnew_reduce(function) **where function has been specified**

- **Or by using a intrinsic reduction:**

```
>>> Dnew.count()
    10
>>> Dnew.sum()
    90
```

All of this is done in parallel!

Apache Spark

Notes: Test your computation locally, but run on cluster:

- **There are tools specifically designed for building "spark clusters": http://spark.apache.org/docs/latest/cluster-overview.html**

- **Can use scripts to launch a EC2 spark cluster**

- **Libraries for spark cluster computing:**
  - **GraphX: graph processing**
  - **Mllib: machine learning**
    http://spark.apache.org/docs/latest/mllib-guide.html
  - **Lapack is included in *Breeze***
  - **Spark SQL, Spark Streaming**

- **Machine learning snippet:**

```
from pyspark.mllib.classification import SVMWithSGD, SVMModel
from pyspark.mllib.regression import LabeledPoint
model = SVMWithSGD.train(parsedData, iterations=100)
```

Aside: Lambda functions in Python

Note: often convenient to use "lambda functions"

```
>>> is_even = lambda x: x % 2 == 0
```

- **Command above is equivalent to function on earlier slide**
- **Similarly these "regular" and lambda functions are equivalent:**

```
>>> def prod(x,y):
        return x*y

>>> prod = lambda x,y: x*y
```

GPU computing

- Graphics cards are highly efficient, very powerful
  - 100s or even 1000s of compute cores
  - Energy efficient – see the green500 list: https://www.top500.org/green500
  - Nvidia is the leader in GPU-accelerated computing
  - Why are GPUs so advanced?

GPU computing

- Graphics cards are highly efficient, very powerful
  - 100s or even 1000s of compute cores
  - Energy efficient – see the green500 list: https://www.top500.org/green500
  - Nvidia is the leader in GPU-accelerated computing
  - Why are GPUs so advanced?
    - Video games are a billion dollar industry
    - Realistic graphics require enormous computational power

GPU computing

- **This computational power has been adapted for scientific computing**
- **Very useful for small-to-medium sized jobs with scope for parallelization**
- **Main limitation is memory**

Nvidia Tesla P100 specs:

SPECIFICATIONS

GPU Architecture	NVIDIA Pascal
NVIDIA CUDA® Cores	3584
Double-Precision Performance	5.3 TeraFLOPS
Single-Precision Performance	10.6 TeraFLOPS
Half-Precision Performance	21.2 TeraFLOPS
GPU Memory	16 GB CoWoS HBM2
Memory Bandwidth	732 GB/s
Interconnect	NVIDIA NVLink
Max Power Consumption	300 W
ECC	Native support with no capacity or performance overhead
Thermal Solution	Passive
Form Factor	SXM2
Compute APIs	NVIDIA CUDA, DirectCompute, OpenCL™, OpenACC

TeraFLOPS measurements with NVIDIA GPU Boost™ technology

GPU computing

- **Three main approaches:**
  - **Libraries (cuBLAS, cuFFT, …)**
  - **Directives (OpenACC), similar to OpenMP**
  - **Programming (CUDA Python, openCL, …)**
- **Programming is not easy – have to explicitly move data between CPU and GPU and be precise with variable types and sizes**
- **Has become extremely popular for machine learning, neural networks (e.g. TensorFlow)**

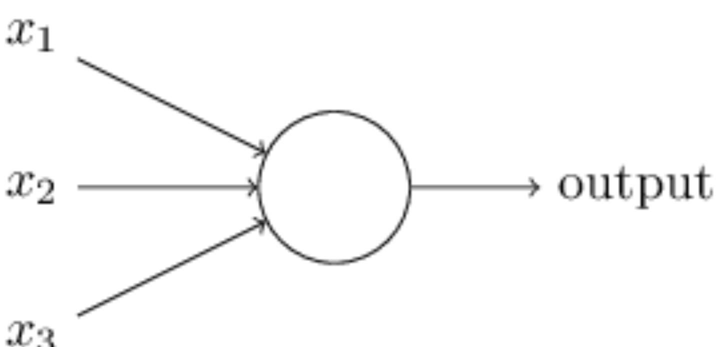
TensorFlow

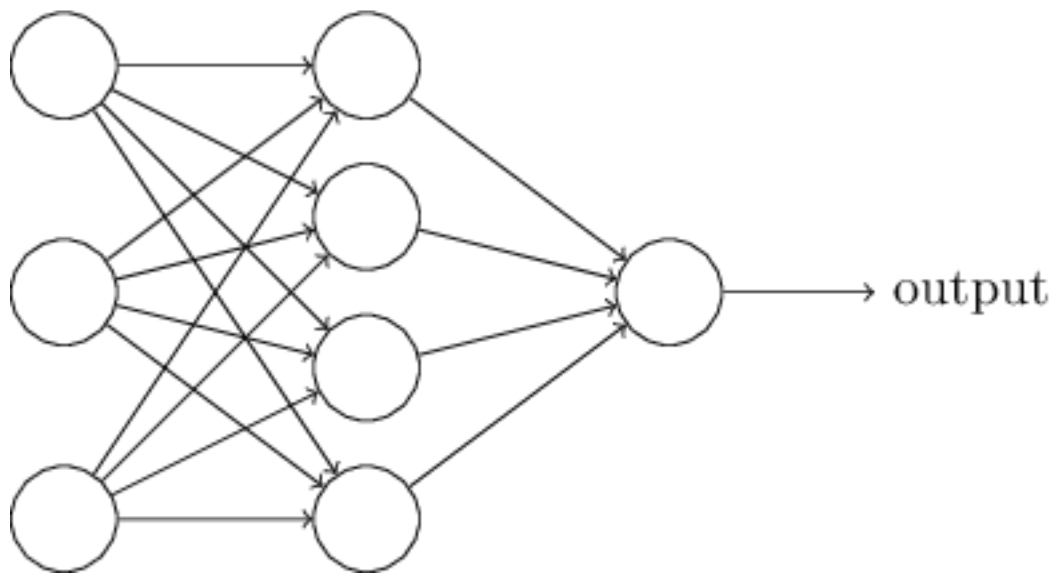
- TensorFlow is a general library for efficient array ("tensor") computations on directed graphs.

- Many libraries built on this very general framework (e.g. Keras for deep learning)

- Can use a Python interface

- Some standard examples to try out (with neural networks)
  - Character recognition
  - Image recognition





Main class takeaways

- **Solving large, complex problems:**

    - **Break problem into smaller parts:**
        - **Python modules with multiple functions**
        - **Fortran modules**
        - **Distribute tasks across threads/processes**

Main class takeaways

- **Solving large, complex problems:**

  - **Break problem into smaller parts:**
    - **Python modules with multiple functions**
    - **Fortran modules**
    - **Distribute tasks across threads/processes**

  - **Manage smaller parts with appropriate tools**
    - **git, makefiles**
    - **OpenMP, MPI**

Main class takeaways

- Solving large, complex problems:

  - Break problem into smaller parts:
    - Python modules with multiple functions
    - Fortran modules
    - Distribute tasks across threads/processes

  - Manage smaller parts with appropriate tools
    - git, makefiles
    - OpenMP, MPI

  - Carefully test each part
    - Package test routines with the code
    - Unit testing tools (e.g. nose) can automatically run through test routines when code is changed

Main class takeaways

- Choose the right tool for the right problem:
  - Interpreted vs. compiled languages
  - General purpose vs. scientific
  - Serial vs. parallel
  - Shared memory vs. distributed memory
  - Libraries vs. writing your own code

Libraries

- **Avoid writing code whenever possible! Don't re-invent the wheel.**

- **Many powerful libraries available: lapack, boost (C++), fftw, NAG, …**
  - **To use: 1. call subroutine/function in code**
    2. link to library when compiling gfortran –o a.exe a.f90 -l lapack

- **Also have *parallel* libraries: scalapack, petsc**
- **These are not easy to use, but much better than writing your own code.**
- **E.g. scalapack has routines for parallel:**
  - **linear systems of equations**
  - **linear least squares**
  - **standard eigenvalue problems**
  - **singular value decomposition**

Main class takeaways

- It is important to know useful programming languages (this is what HR looks for)

- It is also important to know how to learn new programming languages (this is what experts look for)

    - What is the basic structure of (many) programming languages?

Main class takeaways

- **It is important to know useful programming languages (this is what HR looks for)**

- **It is also important to know how to learn new programming languages (this is what experts look for)**

  - **What is the basic structure of (many) programming languages?**

  - **10-lecture *c* course taught at Imperial:**

The course covers:

- Different number types in C (Integers and Floating Point).
- Operators, operands and their precedence.
- Conversions and casts.
- Mathematical expressions.
- Statements: choice, while, do-while, switch, for loops...
- Functions.
- Pointers, arrays and matrices.
- Characters, strings and interacting with the console.
- Reading and writing files.
- Optimisation and Debugging.
- Scientific C-Libraries and their uses: NAG, GSL, etc.
- An Introduction to Parallel Computing.
- C++ and other languages.

Main class takeaways

- Lectures: Computing
- Projects: Computing + Science + Math
  - Next Spring: Computational PDEs, Scientific Computing