Dandelion

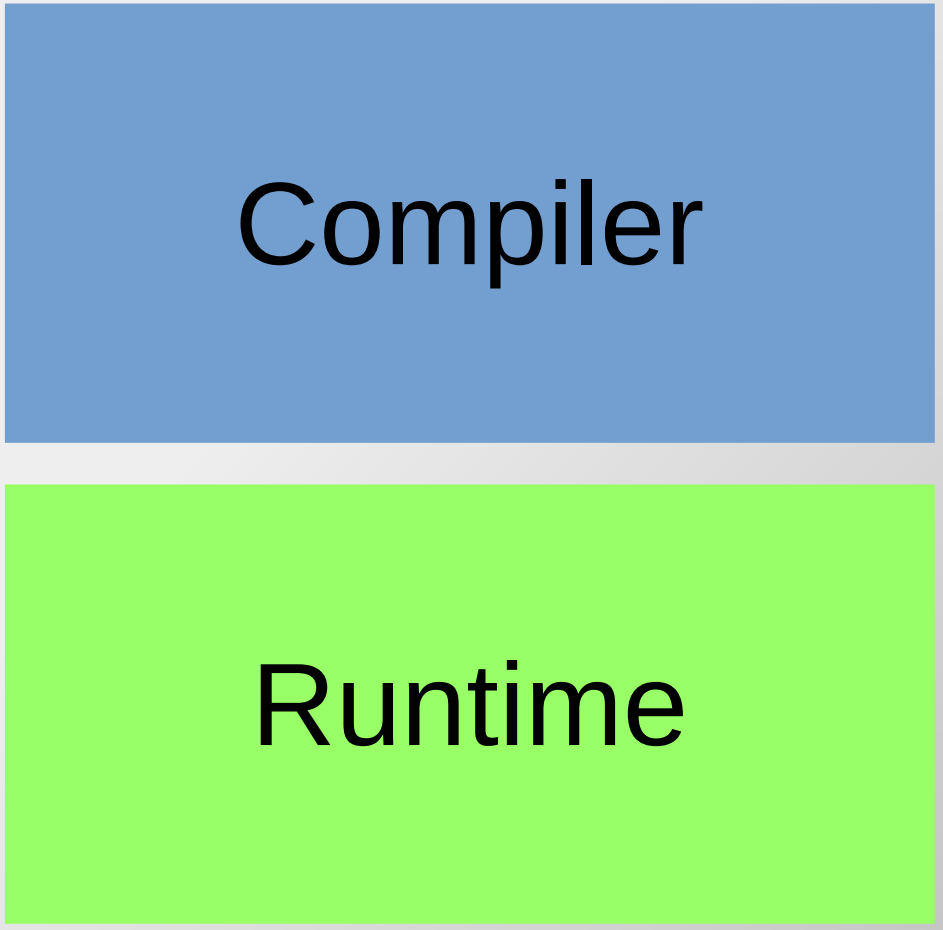
Review for R212: 24th November 2014

Motivation

GPU, FPGA, Vector processors becoming increasingly common

(data parallel, power requirements, SIMD, etc.)

What is Dandelion?



- Compiler for native .NET-based LINQ code (in C# or F#) for GPU programming
- Abstract scheduling details from programmer:

  Multi {machine, CPU, GPU}

Compiler

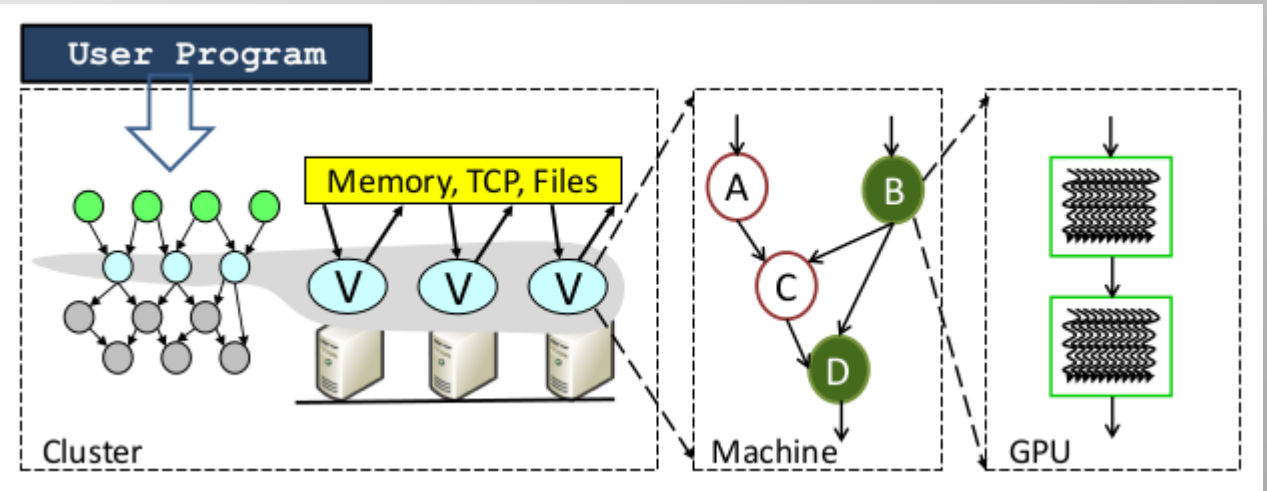
- Clean interface to CUDA
- Deal with CUDA complexities
  - e.g. dynamic memory allocation
- Bytecode compilation: benefits
- Static analysis

Runtime

- Needs to consider three scenarios:
  - Machine-machine
  - CPU-local
  - GPU

Runtime

- Needs to consider three scenarios:
  - Machine-machine
  - CPU-local
  - GPU



GPU dataflow

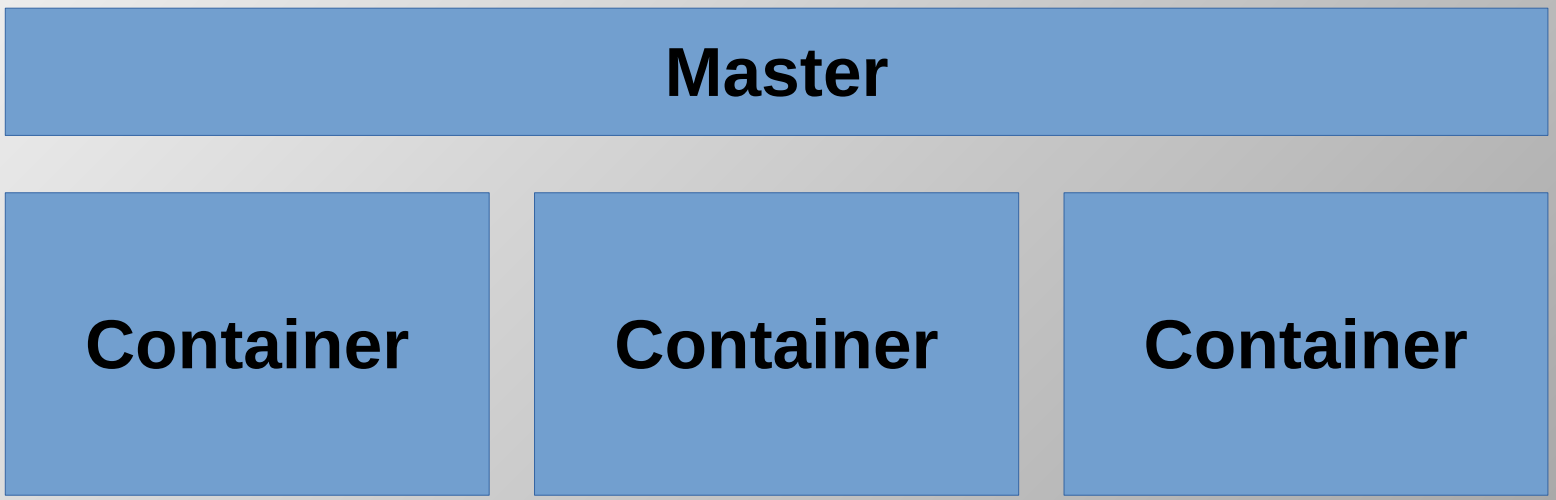
unq_cnt
hashtbl
in(p,c)
hashtbl
unq_cnt
buildHT
unq_cnt
hashtbl
in(p,c)
keymap
unq_cnt
keymap
in(p,c)
groupsizes
sizes
keymap
prefixsum
offsets
shuffle
grps
grouper
aggregate
group-aggregator
accvals
grpcnts
prvacc
prvcnts
accumulate
acc
accnt
reduce

Compute cluster

- Two techniques:
  - Dryad: persistent storage, high availability
  - Moxie (developed for Dandelion):
    Spark-like in-memory storage and checkpoints

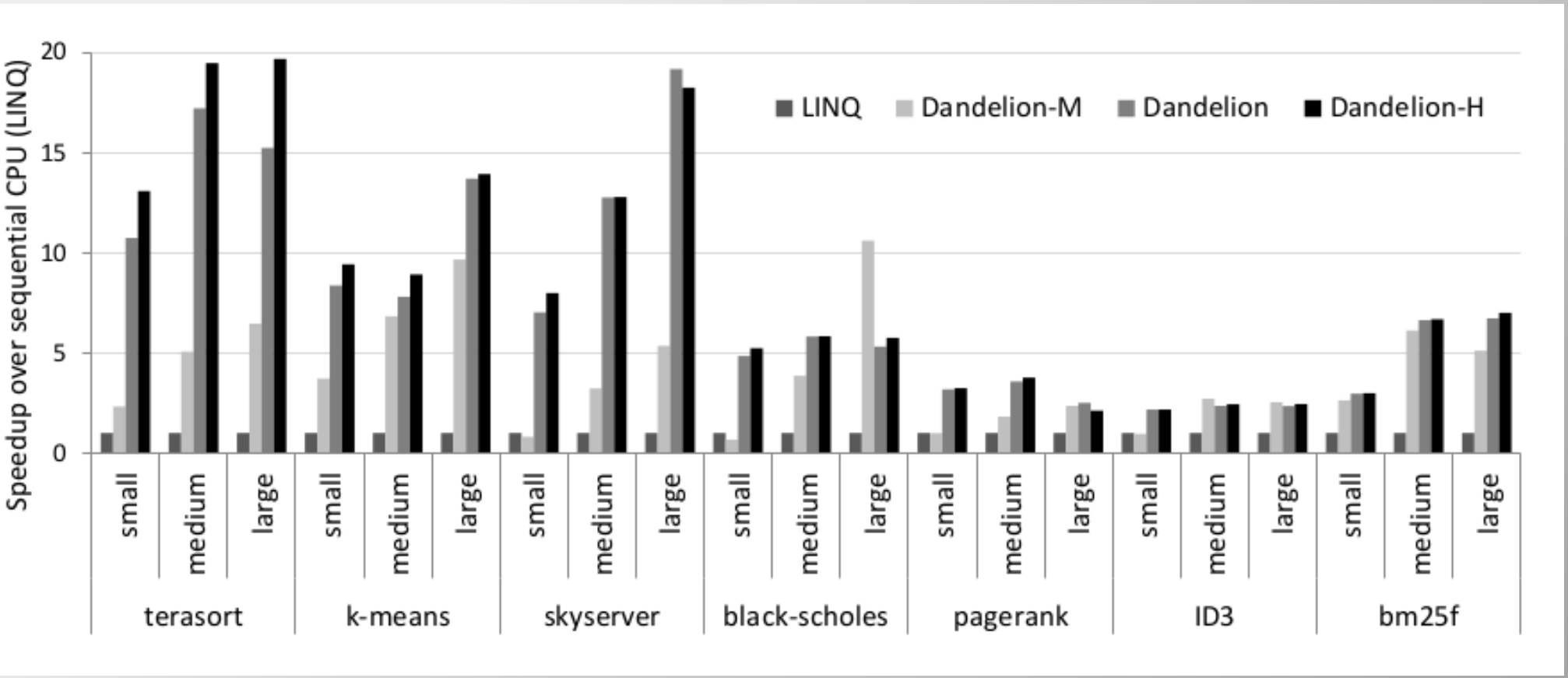
Compute cluster

- Two techniques:
  - Dryad: persistent storage, high availability
  - Moxie (developed for Dandelion):
    Spark-like in-memory storage and checkpoints



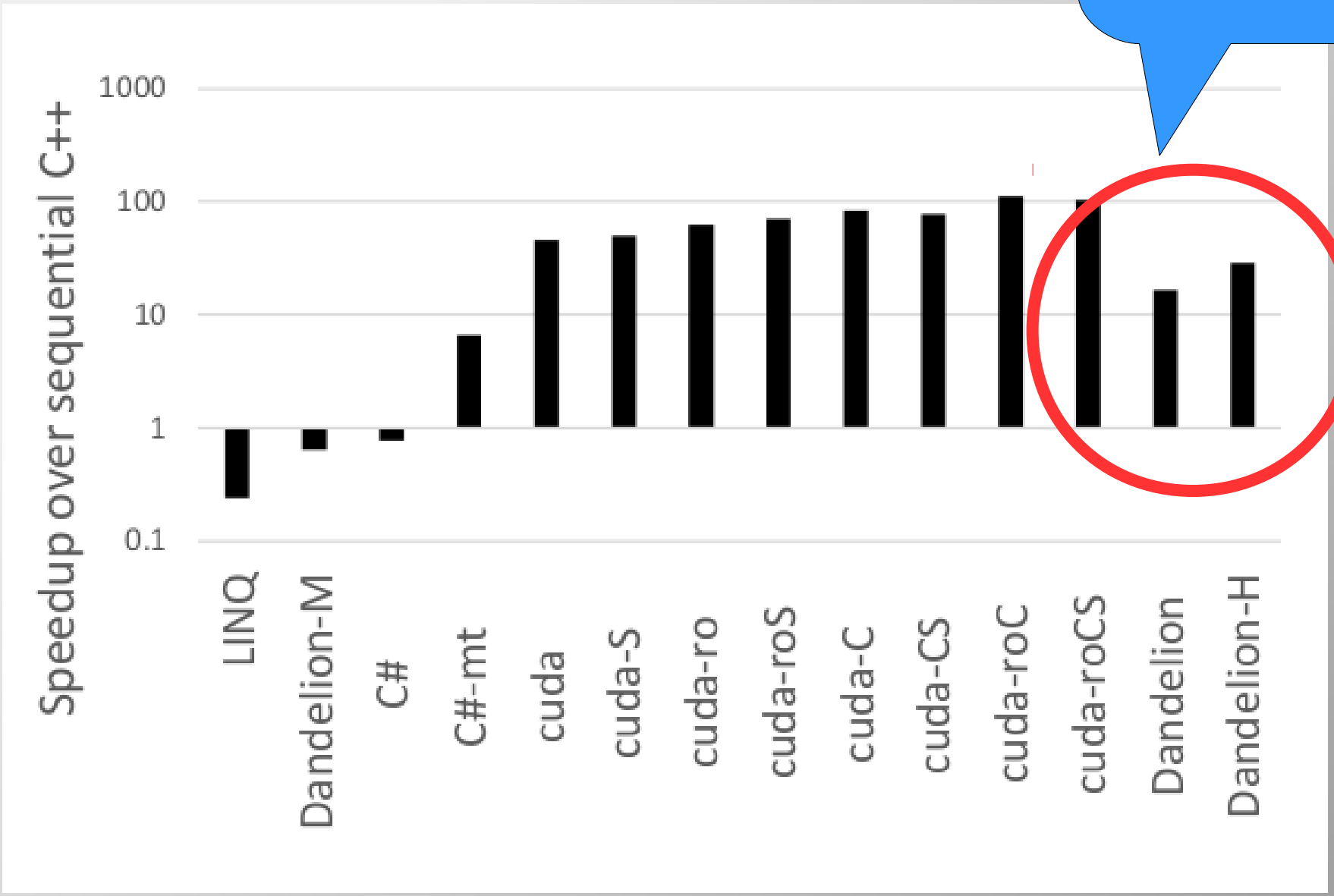
Evaluation

Single machine performance



K-means

20x less code



Criticisms

- No discussion of inter-machine scheduling and associated overheads
- Claim to support FPGAs, but no evaluation of this (cost reasons perhaps?).
- Still suffering Garbage Collection due to managed runtime overheads.
- More evaluation beyond k-means?

Summary

- Data-parallel hardware becoming mainstream; need high-level programming support.
- Dandelion schedules work onto GPUs (and others) from a high-level C# or F# implementation
- Achieves noticeable (30x+) speed improvements through use of GPUs, without learning overhead of CUDA or similar.