Imperial College London

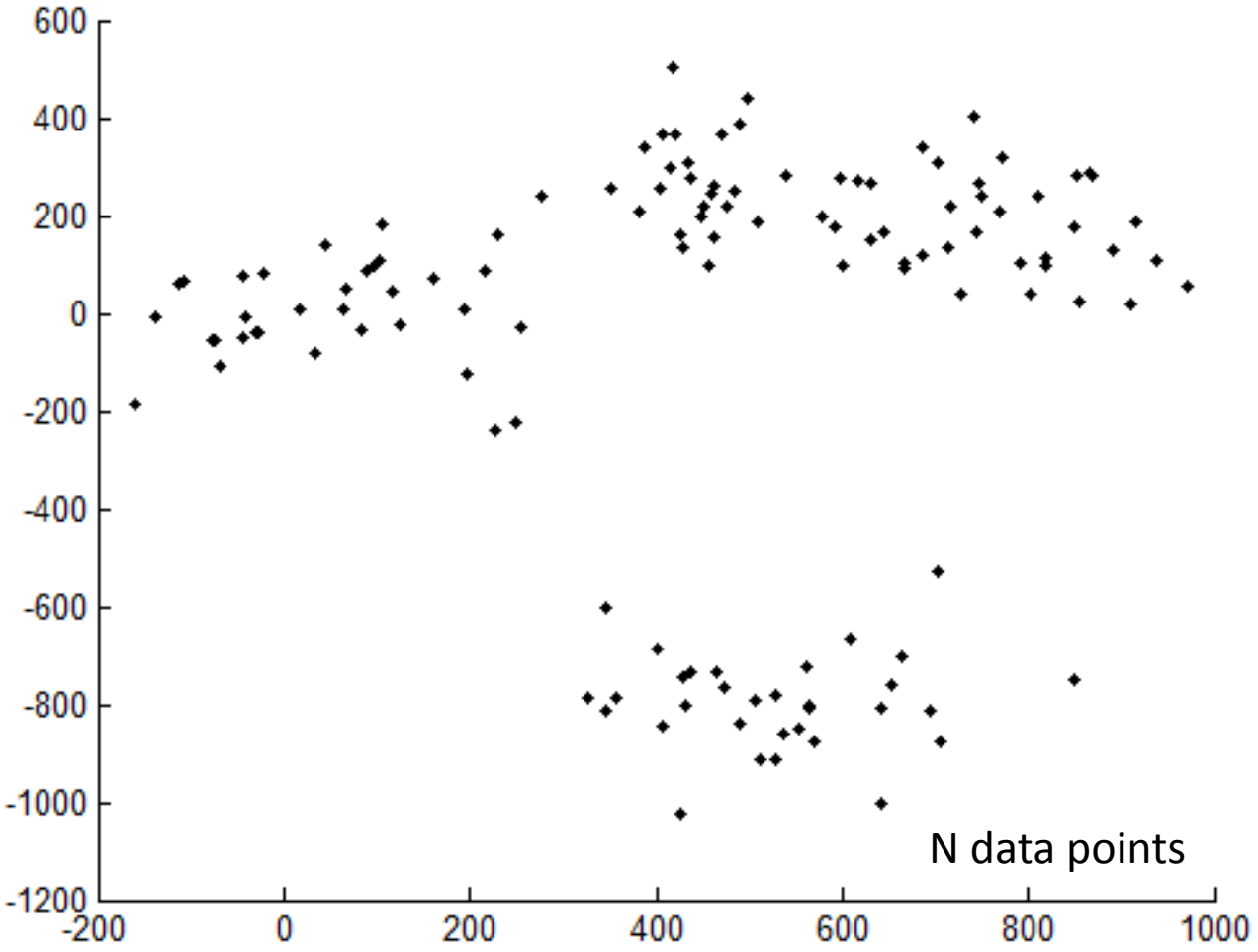
FPGA-Based K-Means Clustering Using Tree-Based Data Structures

Felix Winterstein, Samuel Bayliss,
George Constantinides

2 September 2013

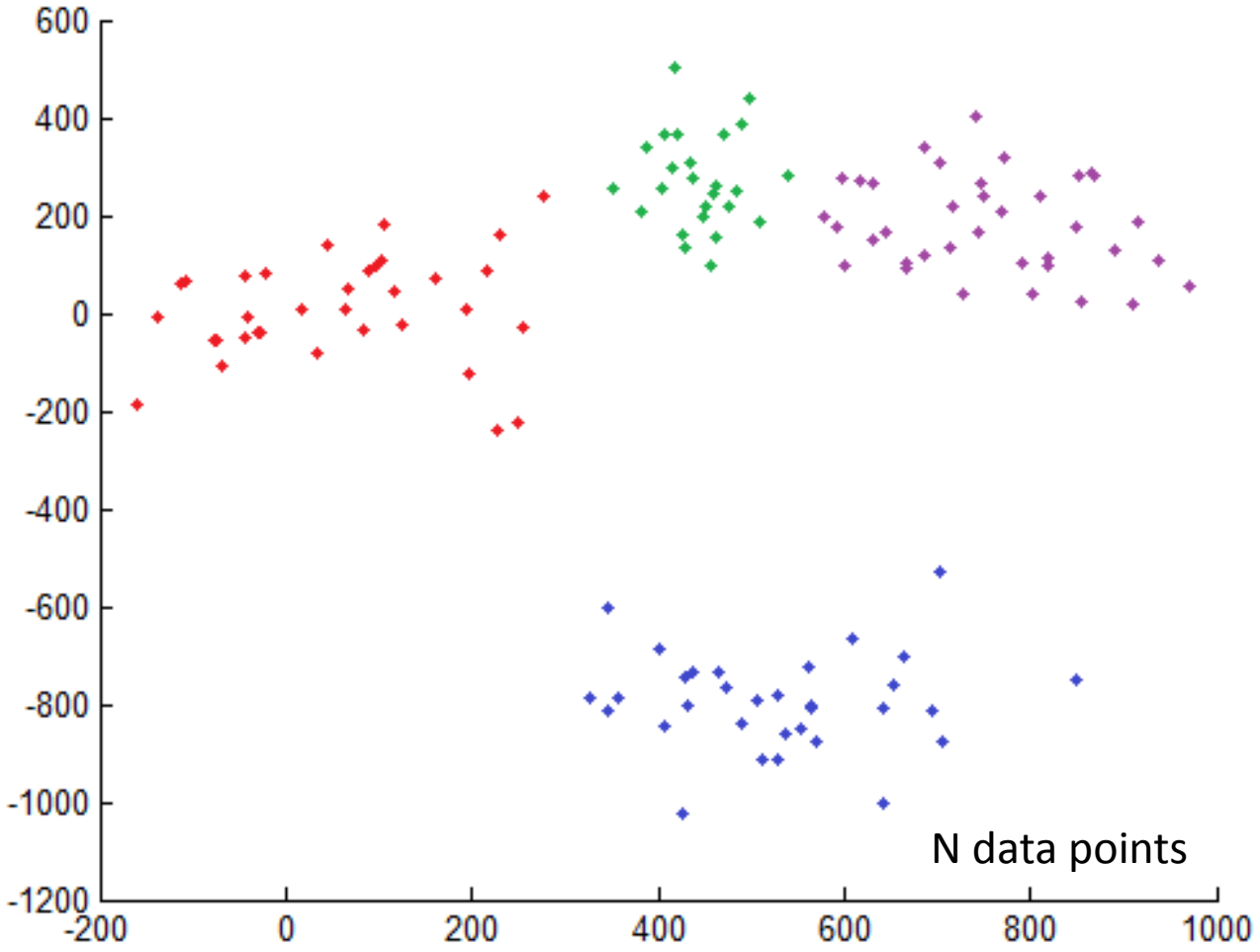
Clustering

- Application in
  - Unsupervised learning
  - Data mining
  - Pattern recognition
  - Tracking
  - Image quantisation (colour space)



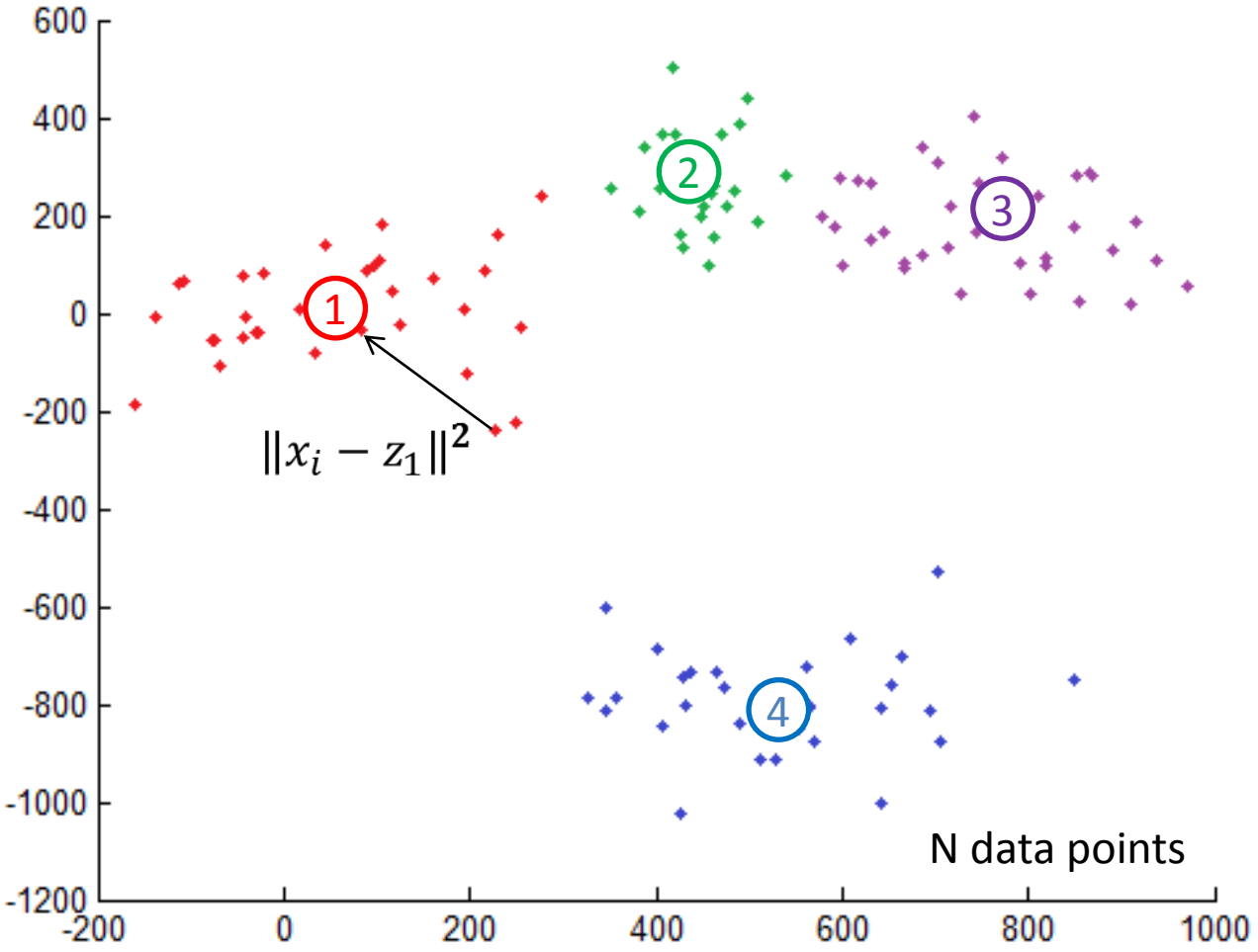
Clustering

- Application in
  - Unsupervised learning
  - Data mining
  - Pattern recognition
  - Tracking
  - Image quantisation (colour space)
- Automatic partitioning
- K-means clustering widely used



K-means clustering

- K is a known parameter (e.g. K=4)
- Clusters represented by their centres
- Centre position determines cluster assignment
- Find optimal positioning

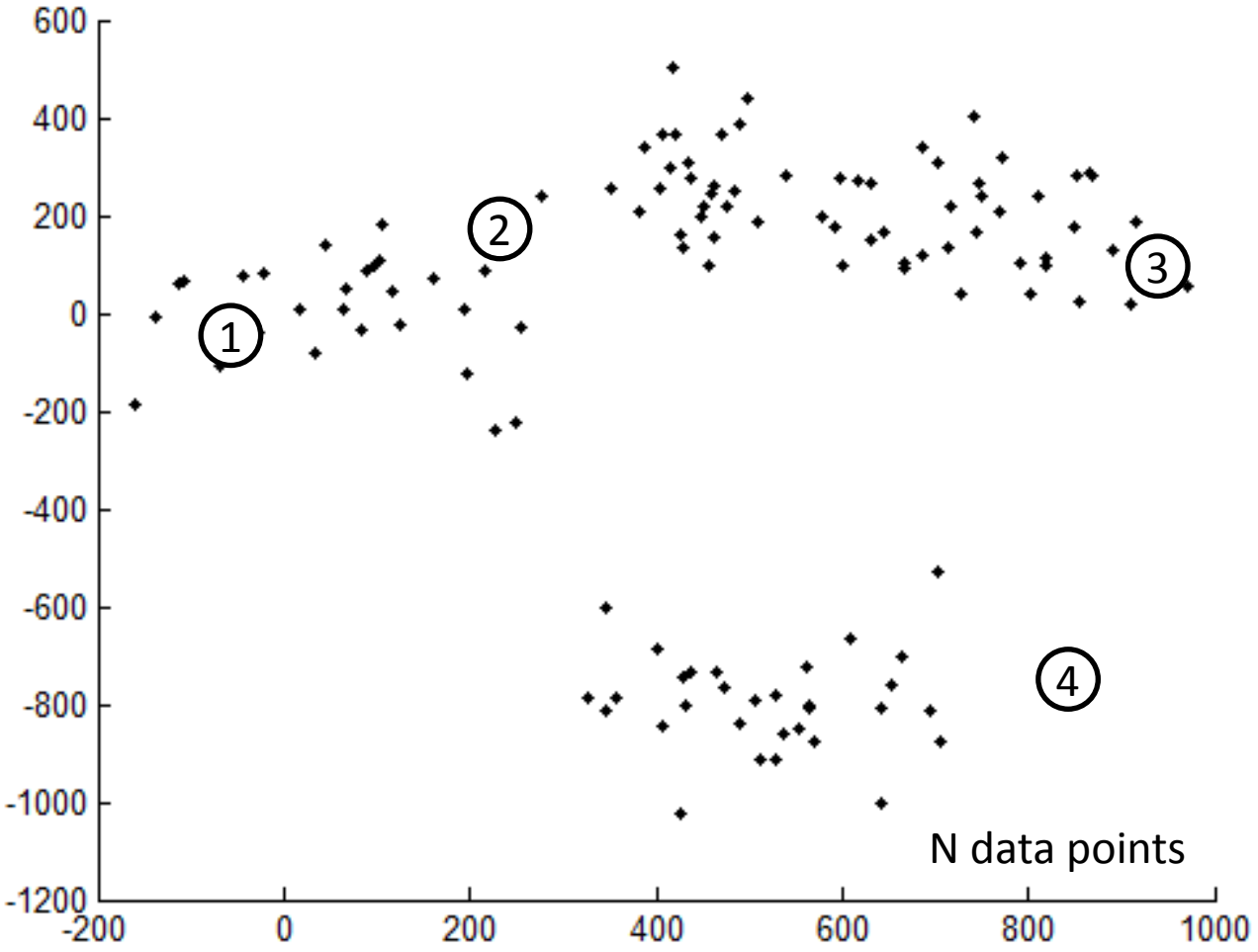


Imperial College London

K-means clustering

esa

- K is a known parameter (e.g. K=4)
- Clusters represented by their centres
- Centre position determines cluster assignment
- Find optimal positioning
- Iterative refinement of initial placement

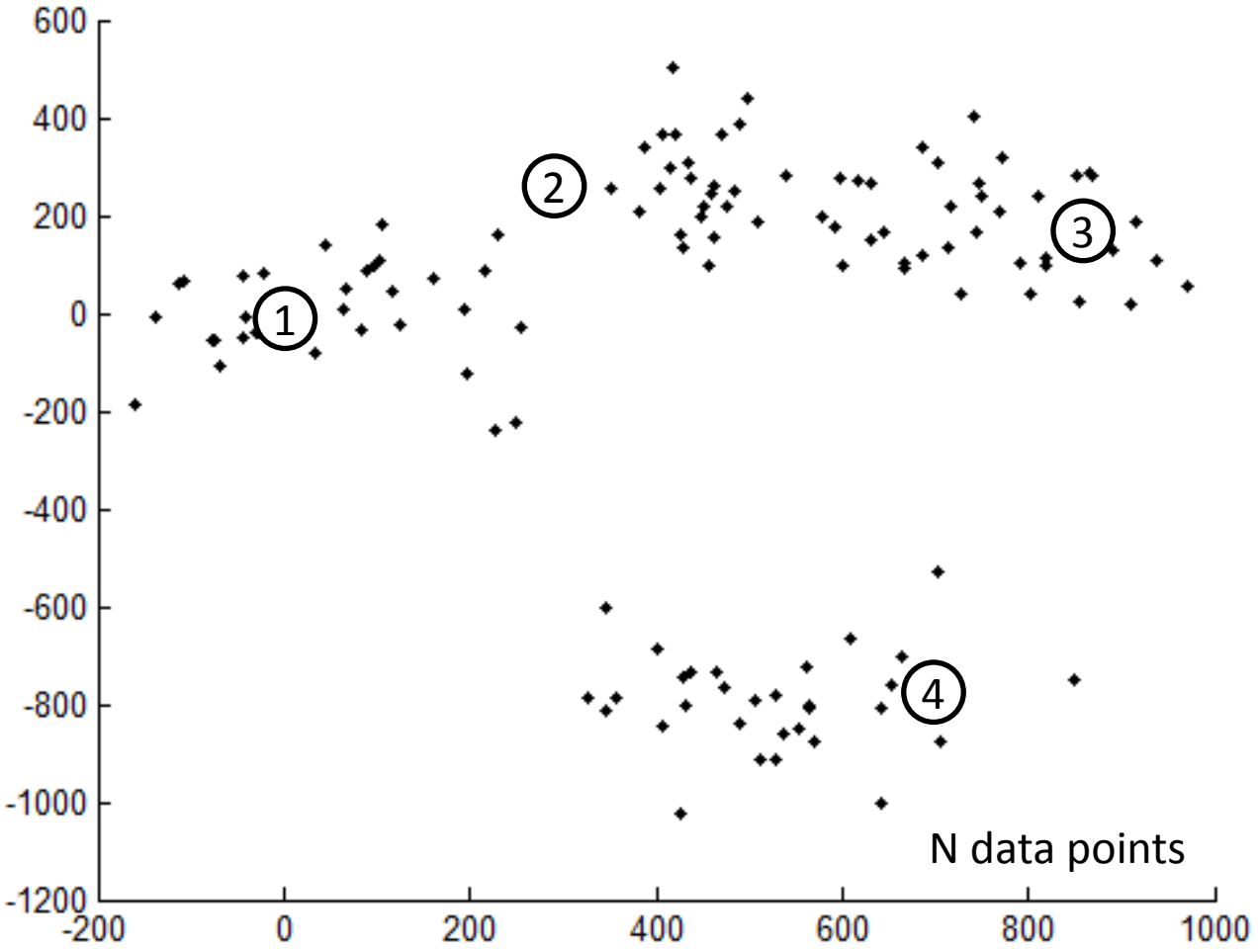


Imperial College London

K-means clustering

esa

- K is a known parameter (e.g. K=4)
- Clusters represented by their centres
- Centre position determines cluster assignment
- Find optimal positioning
- Iterative refinement of initial placement

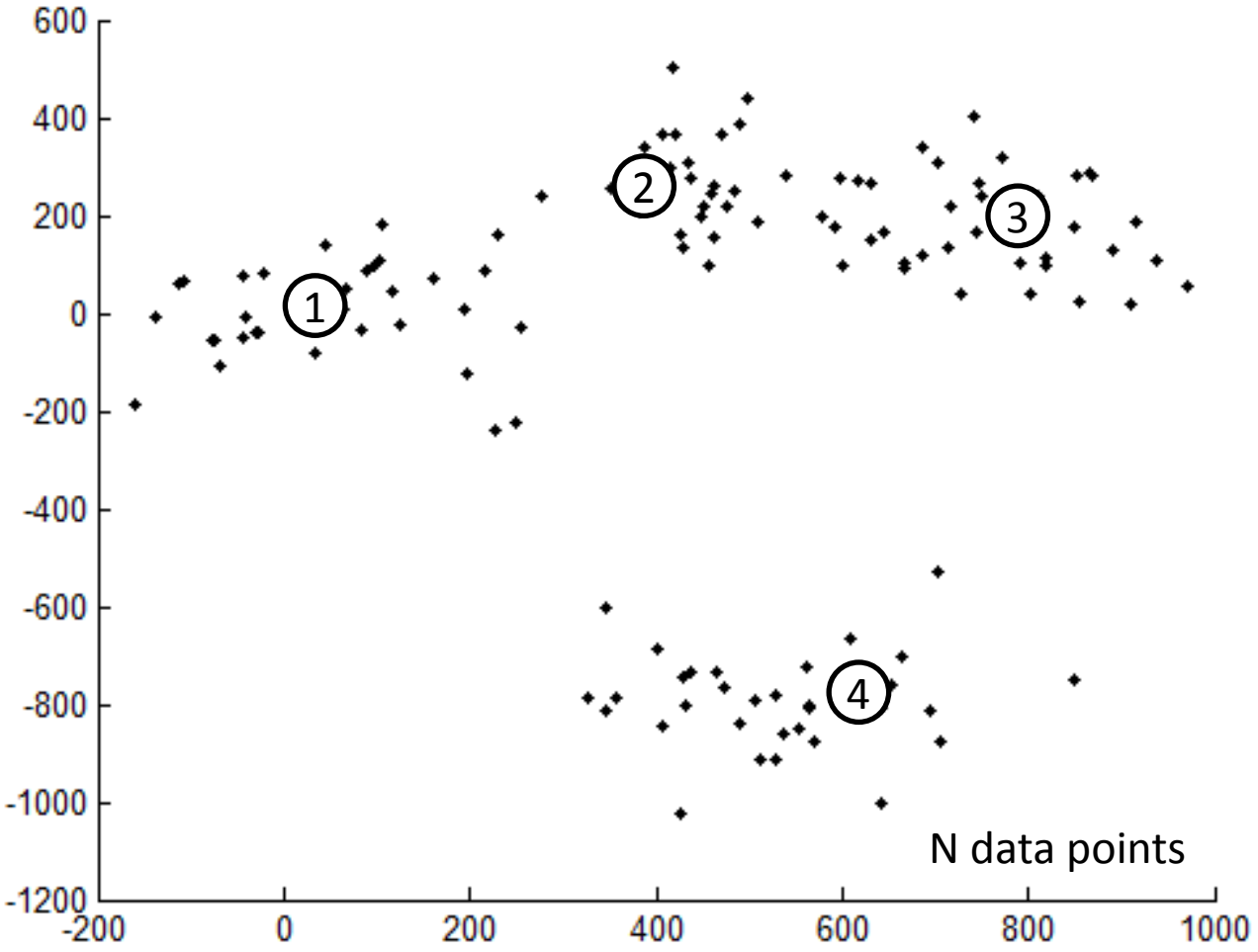


Imperial College London

K-means clustering

esa

- K is a known parameter (e.g. K=4)
- Clusters represented by their centres
- Centre position determines cluster assignment
- Find optimal positioning
- Iterative refinement of initial placement

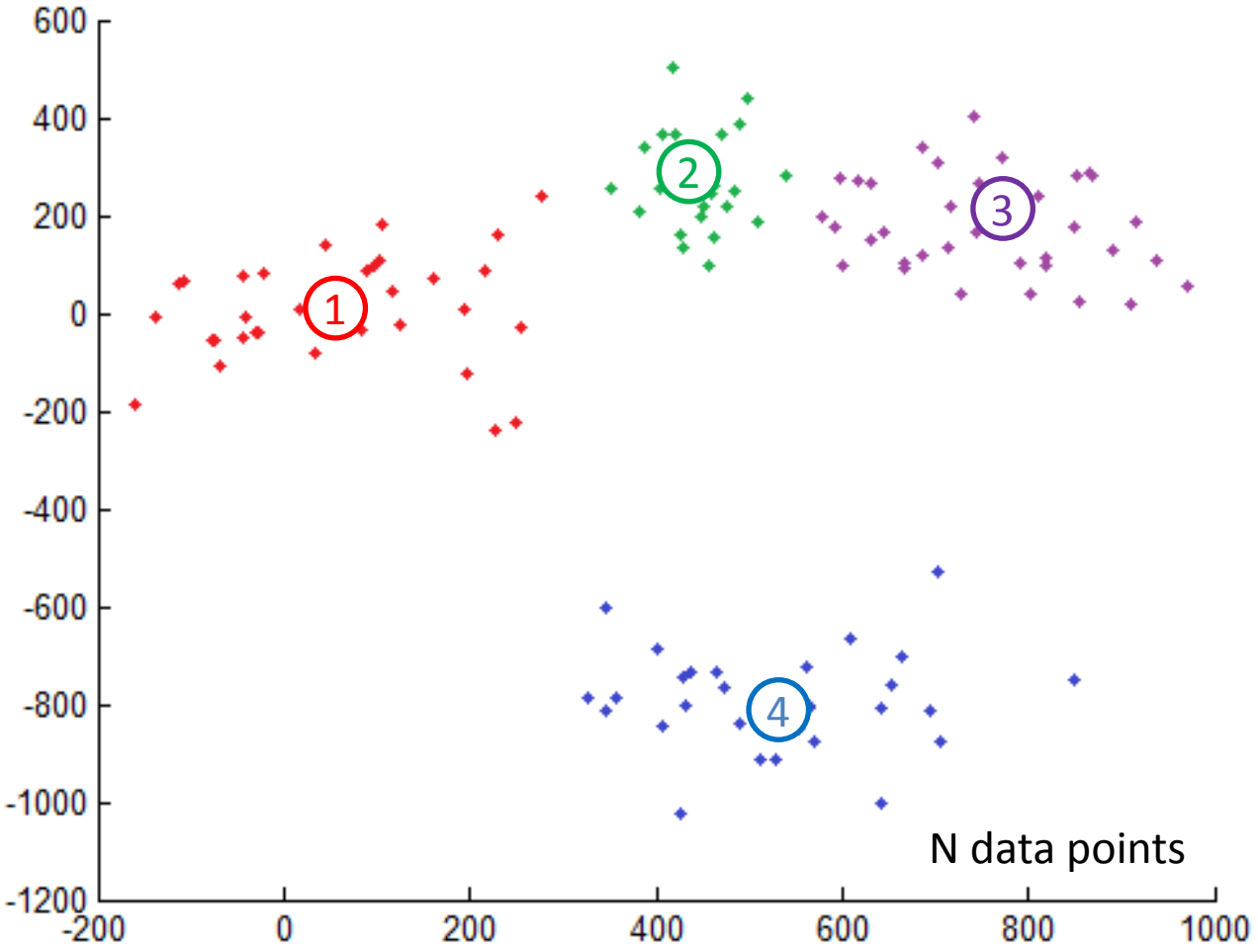


Imperial College London

K-means clustering

esa

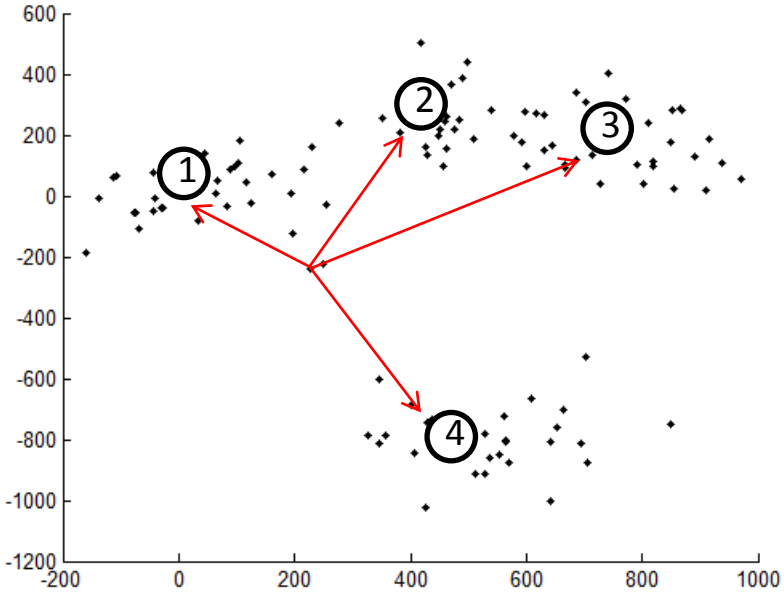
- K is a known parameter (e.g. K=4)
- Clusters represented by their centres
- Centre position determines cluster assignment
- Find optimal positioning
- Iterative refinement of initial placement



Imperial College London

Brute-force Algorithm

esa



- Simple control flow
- Embarrassingly parallel
- Well-suited for FPGAs:

  Leeser 2001, Estlick 2001, Wang 2007, Kutty 2013

function CONVENTIONAL( )

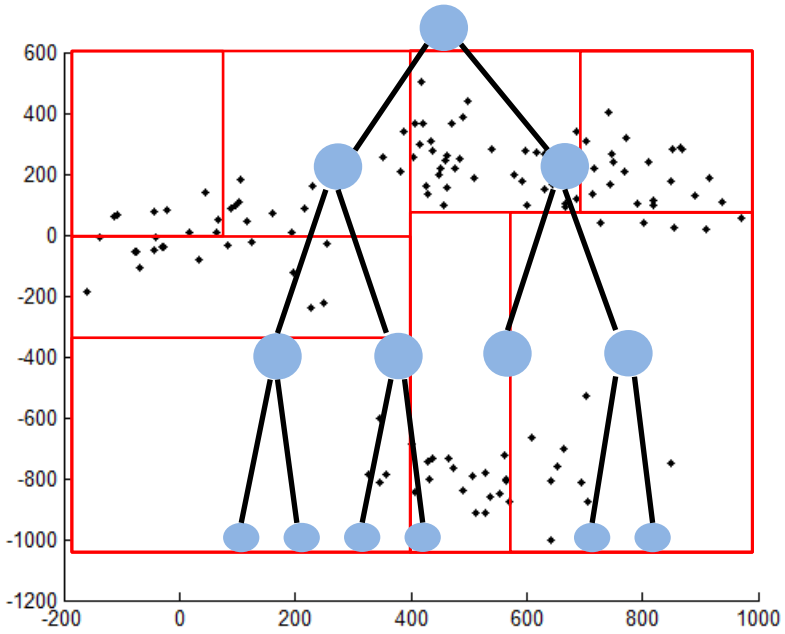
  for all $x_i \in \{x_1, x_2, \ldots, x_N\}$ **do**
    for all $z_j \in \{z_1, z_2, \ldots, z_K\}$ **do**
      $d^2 \leftarrow \|x_i - z_j\|^2$
    end for
    $z^* \leftarrow$ closest centre to x_i
    updateCentreBuffer(z^*)
  end for

end function

Imperial College London

Filtering Algorithm

esa

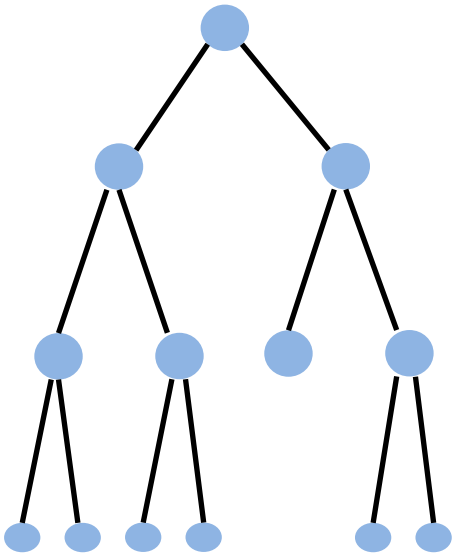


* Kanungo et al., “An efficient k-means clustering algorithm: Analysis and implementation,” 2002

Imperial College London

Filtering Algorithm

esa



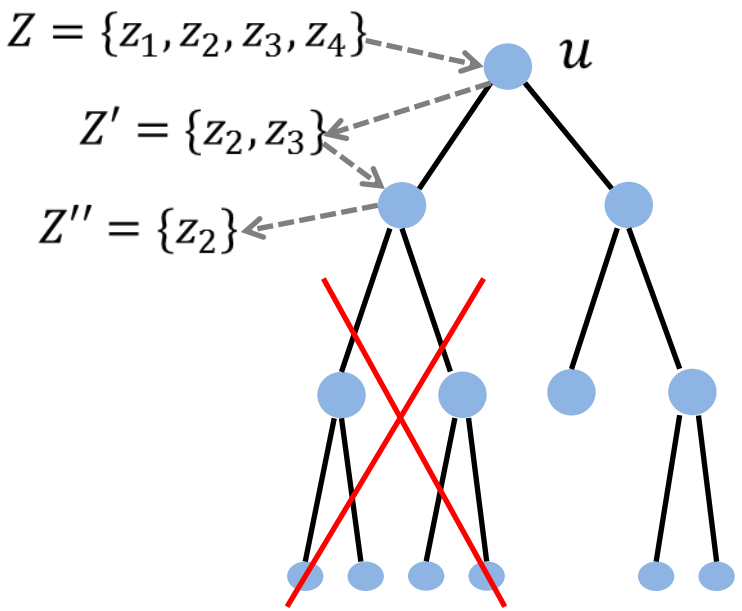
function FILTER(treeNode u, CentreSet Z)

$z^* \leftarrow$ closest centre ϵ Z to $u.midPoint$
if u is leaf **then**
updateCentreBuffer(z^*)
else
$newZ \leftarrow$ pruneSet(Z, z^*, $u.bndBox$)
if $|newZ| > 1$ **then**
FILTER($u.left, newZ$)
FILTER($u.right, newZ$)
end if
end if

end function

* Kanungo et al., "An efficient k-means clustering algorithm: Analysis and implementation," 2002

Filtering Algorithm



- Recursive tree traversal
- Dynamic data structure

```
function FILTER(treeNode u, CentreSet Z)

    z* ← closest centre ϵ Z to u.midPoint
    if u is leaf then
        updateCentreBuffer(z*)
    else
        newZ ← pruneSet(Z, z*, u.bndBox)
        if |newZ| > 1 then
            FILTER(u.left, newZ)
            FILTER(u.right, newZ)
        end if
    end if

end function
```

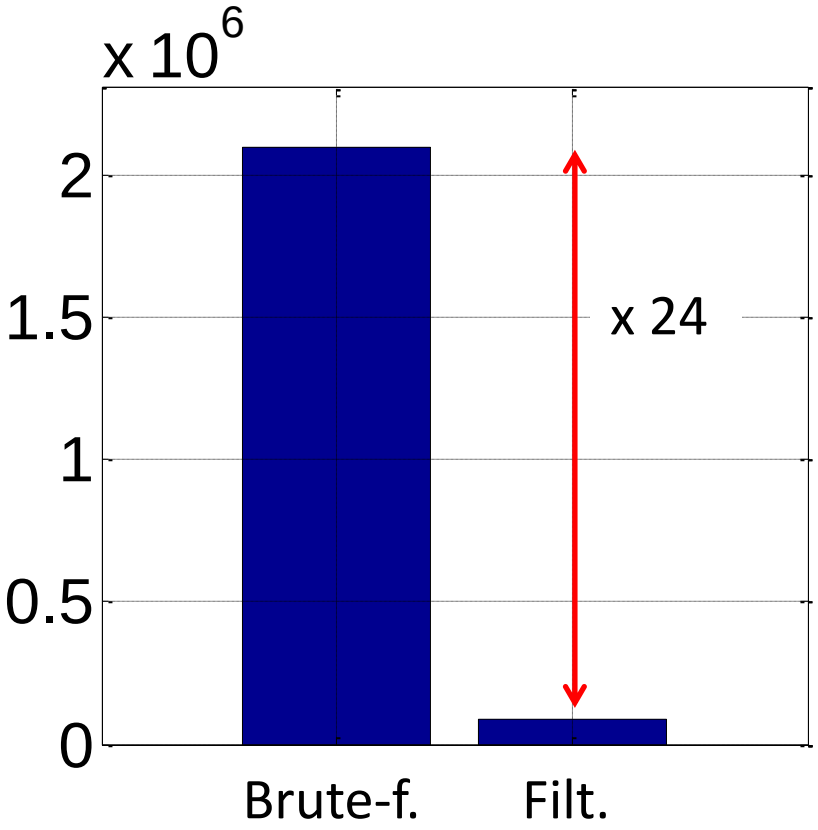
* Kanungo et al., "An efficient k-means clustering algorithm: Analysis and implementation," 2002

Imperial College London

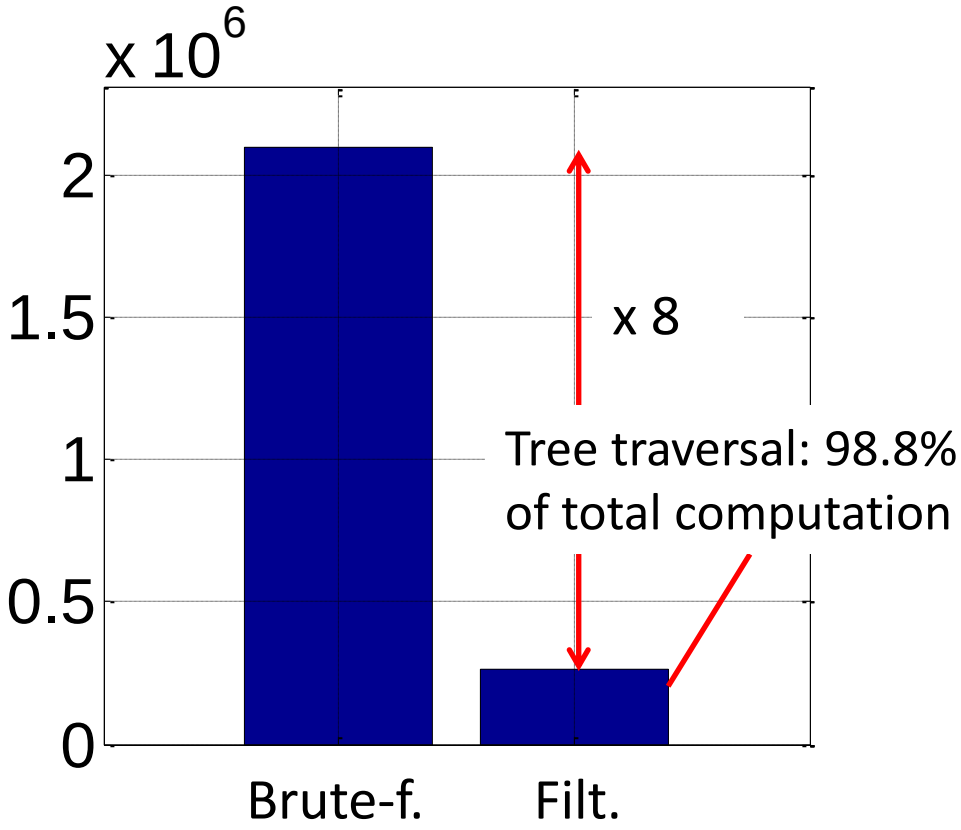
Clear Advantage in SW

esa

Node-centre interactions



Computational complexity (distance comp. equivalents)



N=16384, K=128, D=3

Outline

- **Pointer-based dynamic data structures on FPGA**
- Pipelining and parallelisation
- Dynamic memory access
- Maintaining efficiency
- Conclusion

Why unusual in HW

- Recursion
  - Implement customised stack
- Benefits of a hardware implementation
  - Pipeline and parallelise recursive instances
- Each instance `creates‘ a new $newZ$ (under data-dependent condition)
  - Manage dynamic allocation of scratchpad memory

function FILTER(treeNode u, CentreSet Z)

$z^* \leftarrow$ closest centre ϵ Z to $u.midPoint$
if u is leaf **then**
updateCentreBuffer(z^*)
else
$newZ \leftarrow$ pruneSet(Z, z^*, $u.bndBox$)
if $|newZ| > 1$ **then**
FILTER($u.left, newZ$)
FILTER($u.right, newZ$)
end if
end if

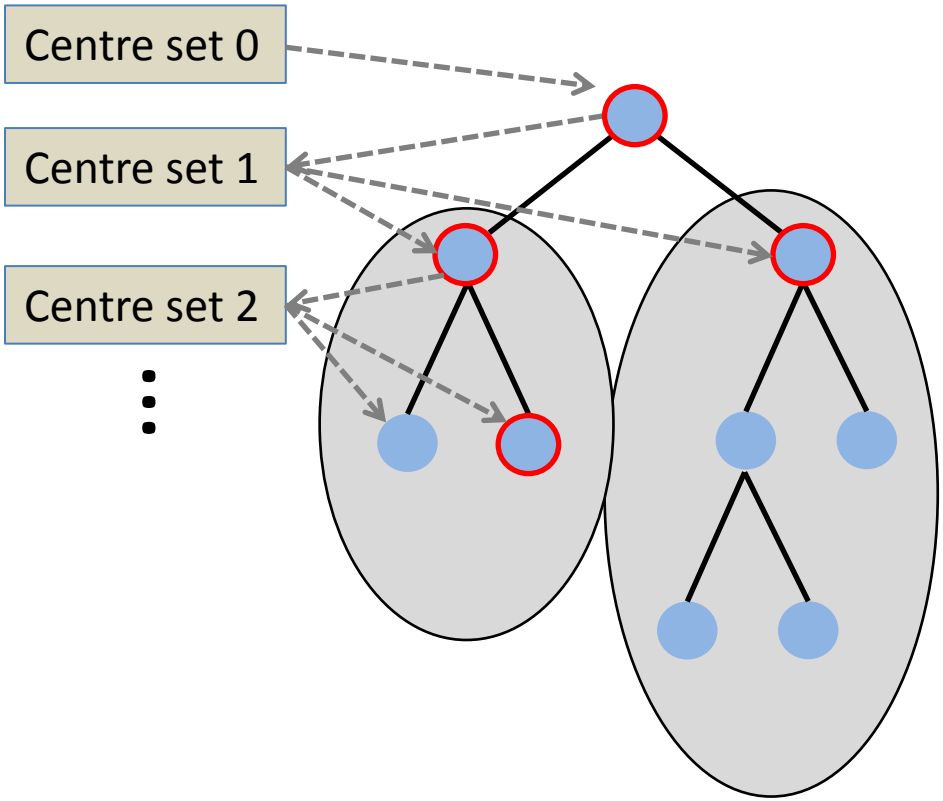
end function

Outline

- Pointer-based dynamic data structures on FPGA
- **Pipelining and parallelisation**
- Dynamic memory access
- Maintaining efficiency
- Conclusion

Pipelining and Parallelisation

Overlap recursive instances in the pipeline



Stack content:

It1: {s1, s1}

It2: {s2, s2, s1}

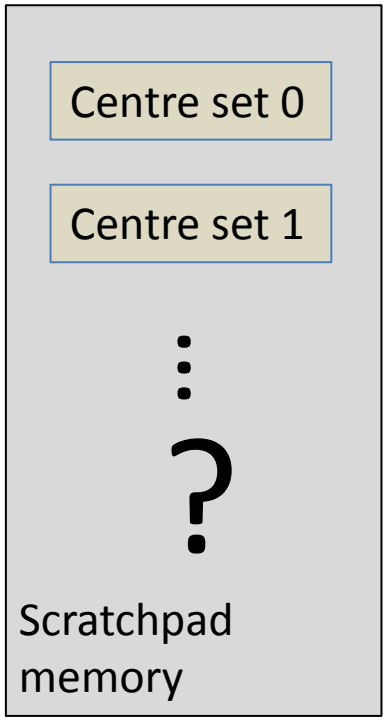
…

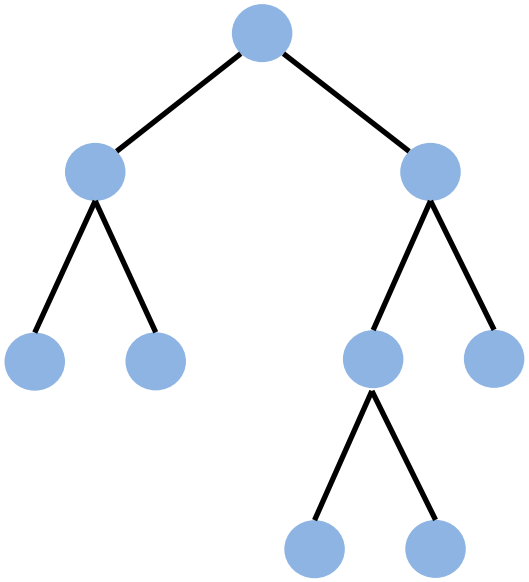
- All items on stack can be processed independently
- Dependence analysis shows we can pipeline recursive instances
- Further parallelism by splitting into sub-trees

Outline

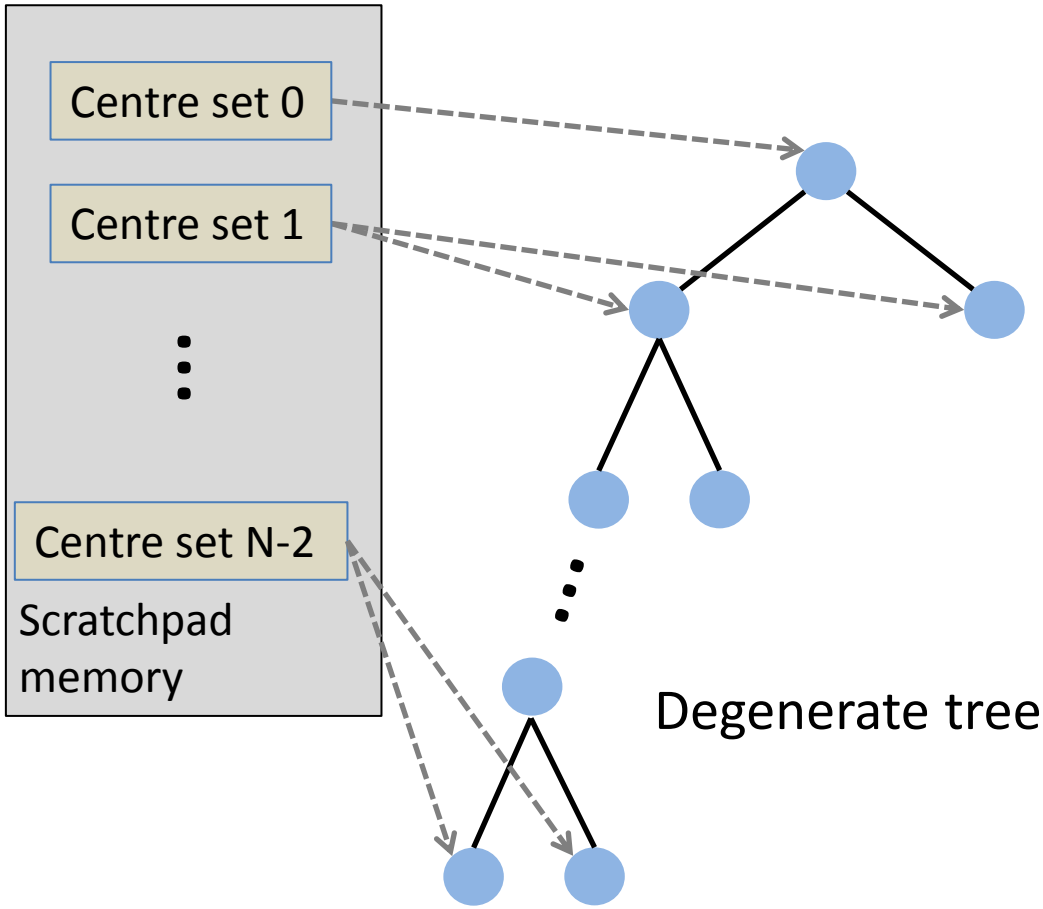
- Pointer-based dynamic data structures on FPGA
- Pipelining and parallelisation
- **Dynamic memory access**
- Maintaining efficiency
- Conclusion

Scratchpad memory



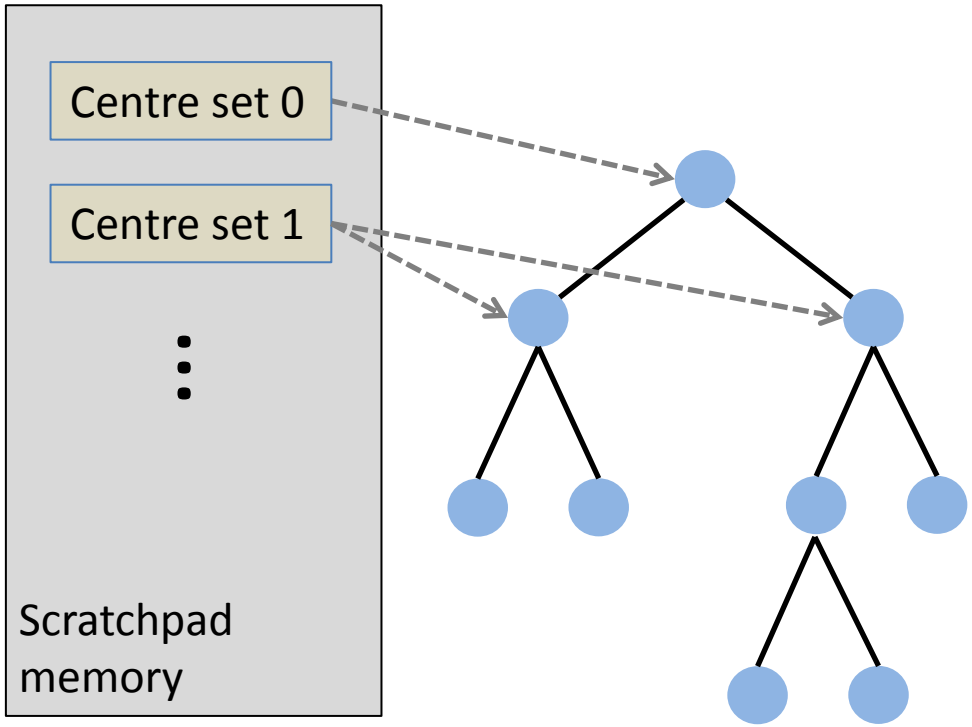


Worst-case amount of memory



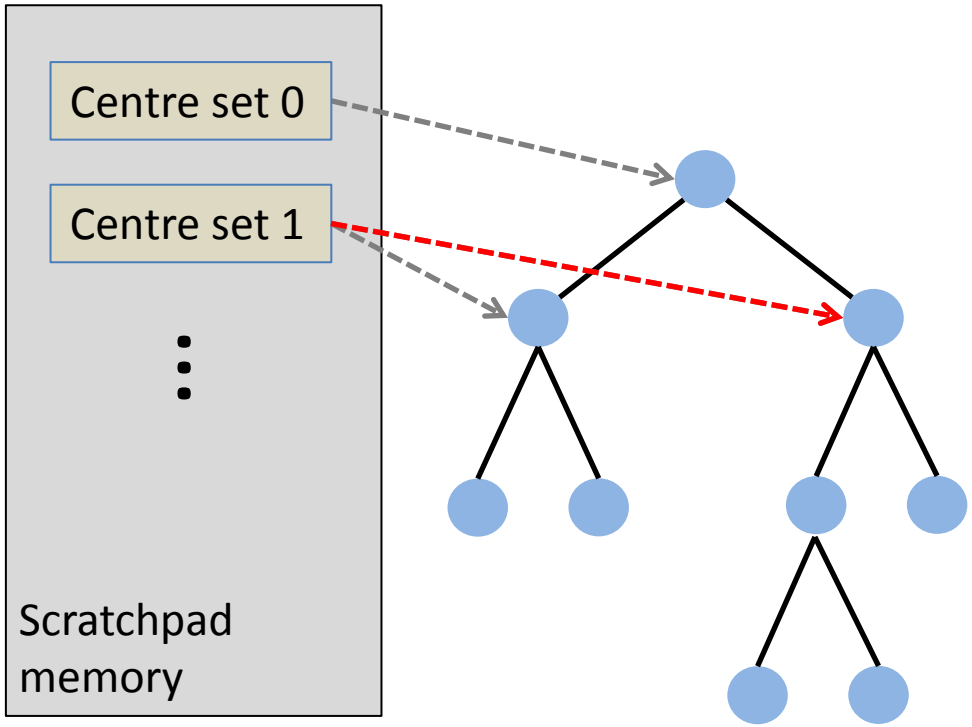
- Worst-case: N-1 sets
- N=16384, K=256:
  -> 32 Mbits (81% of BRAMs in medium-size Virtex 6 FPGA)

Average amount of memory



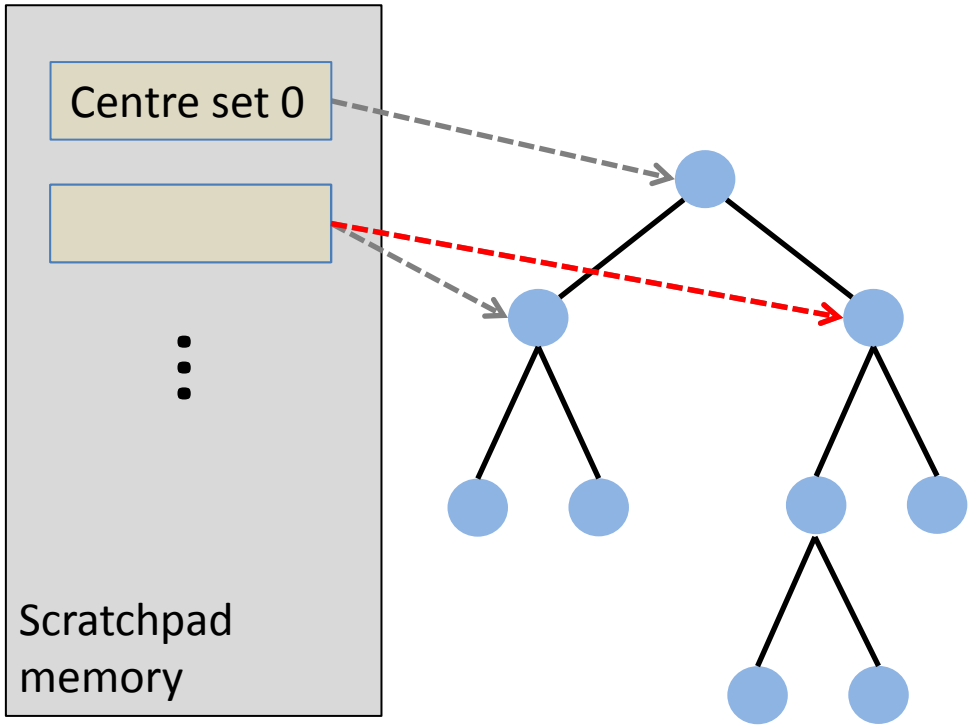
- Worst-case: N-1 sets
- N=16384, K=256:
  -> 32 Mbits (81% of BRAMs in medium-size Virtex 6 FPGA)

Average amount of memory



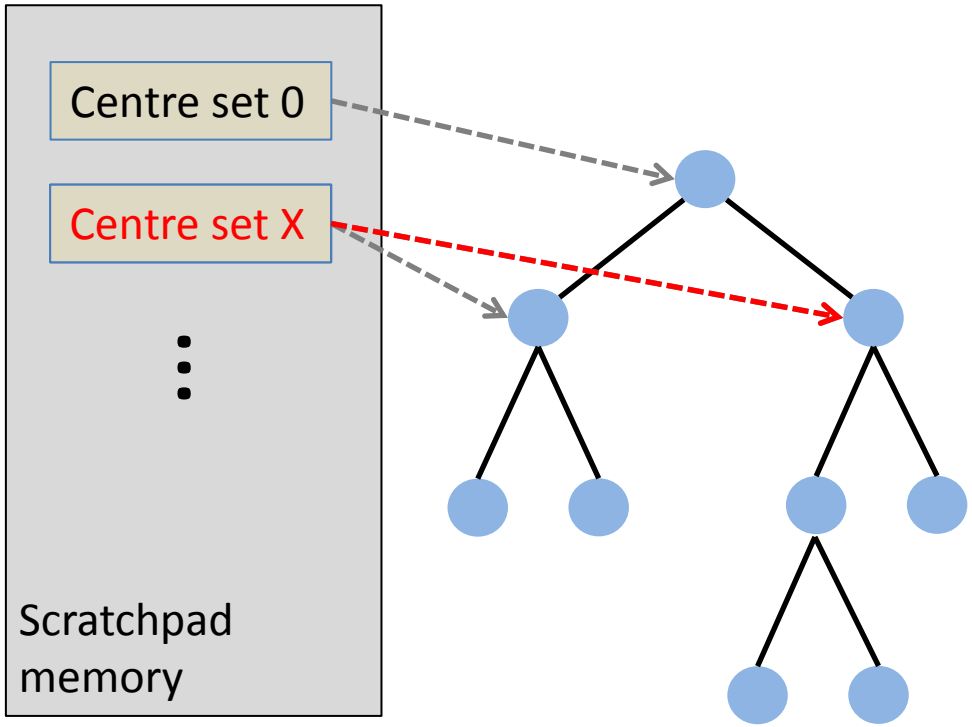
- Worst-case: N-1 sets
- N=16384, K=256:
  -> 32 Mbits (81% of BRAMs in medium-size Virtex 6 FPGA)

Average amount of memory



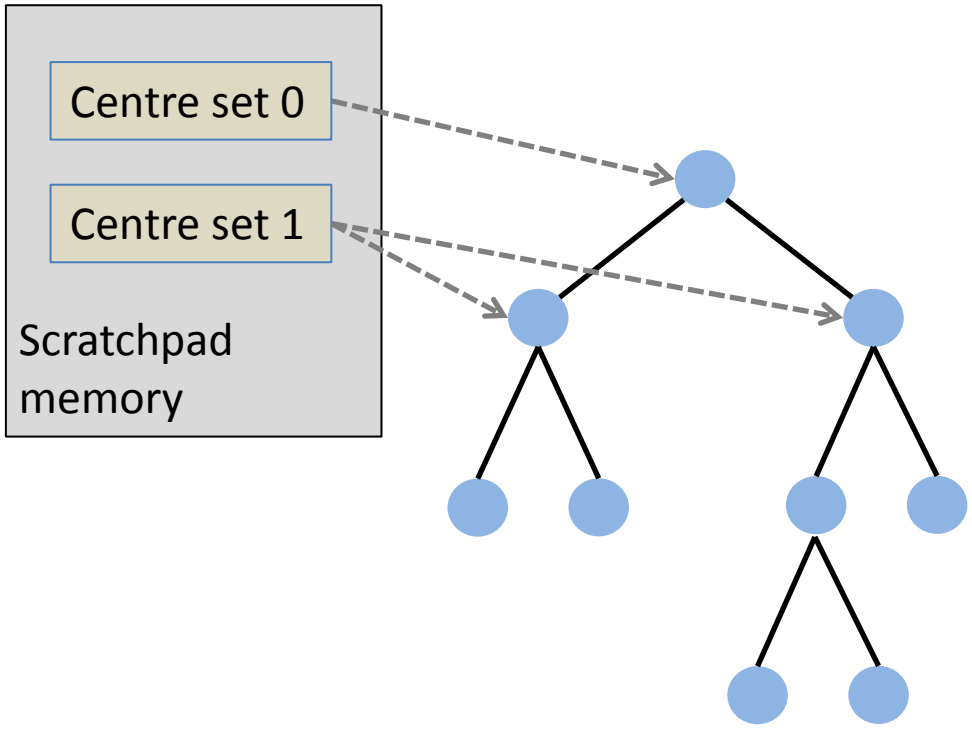
- Worst-case: N-1 sets
- N=16384, K=256:
  -> 32 Mbits (81% of BRAMs in medium-size Virtex 6 FPGA)

Average amount of memory



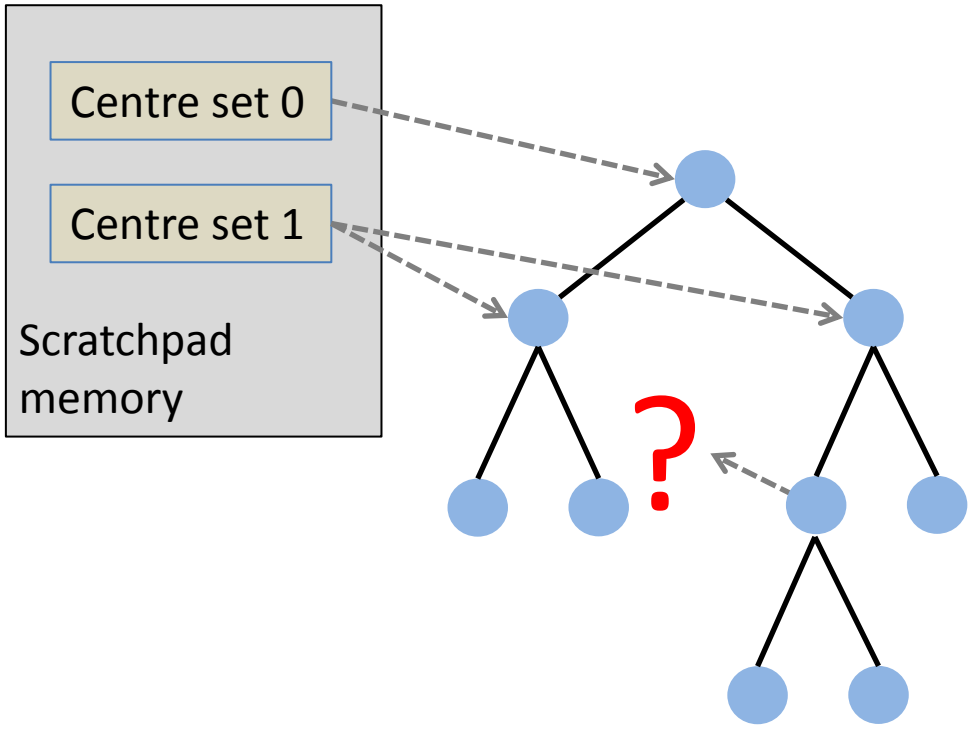
- Worst-case: N-1 sets
- N=16384, K=256:
  -> 32 Mbits (81% of BRAMs in medium-size Virtex 6 FPGA)
- Reuse memory space
- Dynamic memory allocation

Dynamic memory allocation



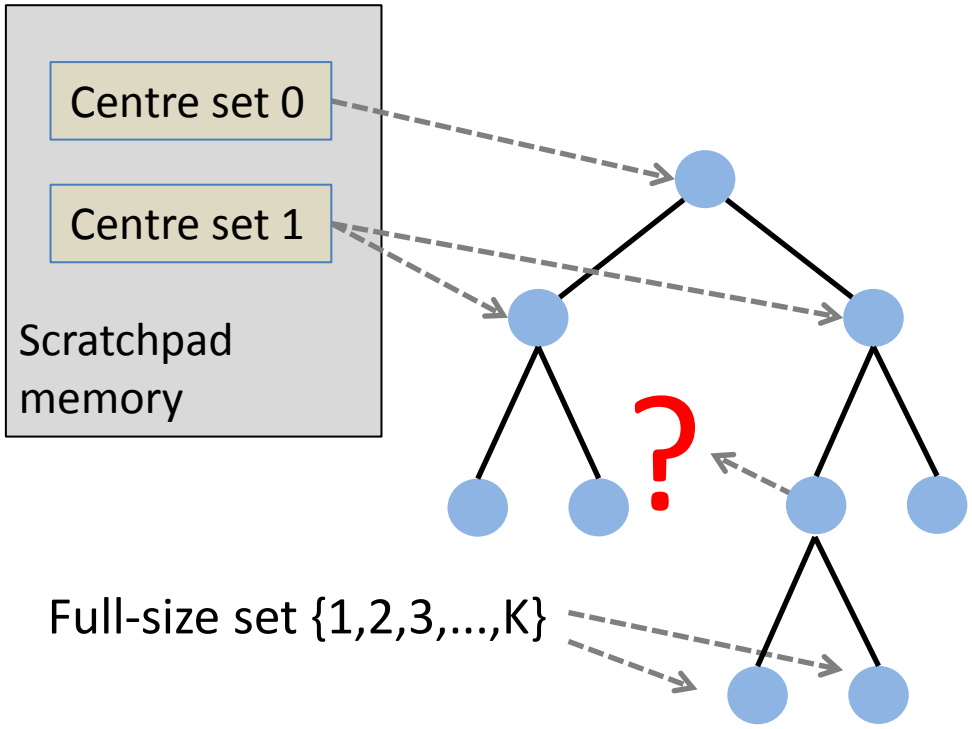
- Worst-case: N-1 sets
- N=16384, K=256:
  -> 32 Mbits (81% of BRAMs in medium-size Virtex 6 FPGA)
- Reuse memory space
- Dynamic memory allocation

Dynamic memory allocation



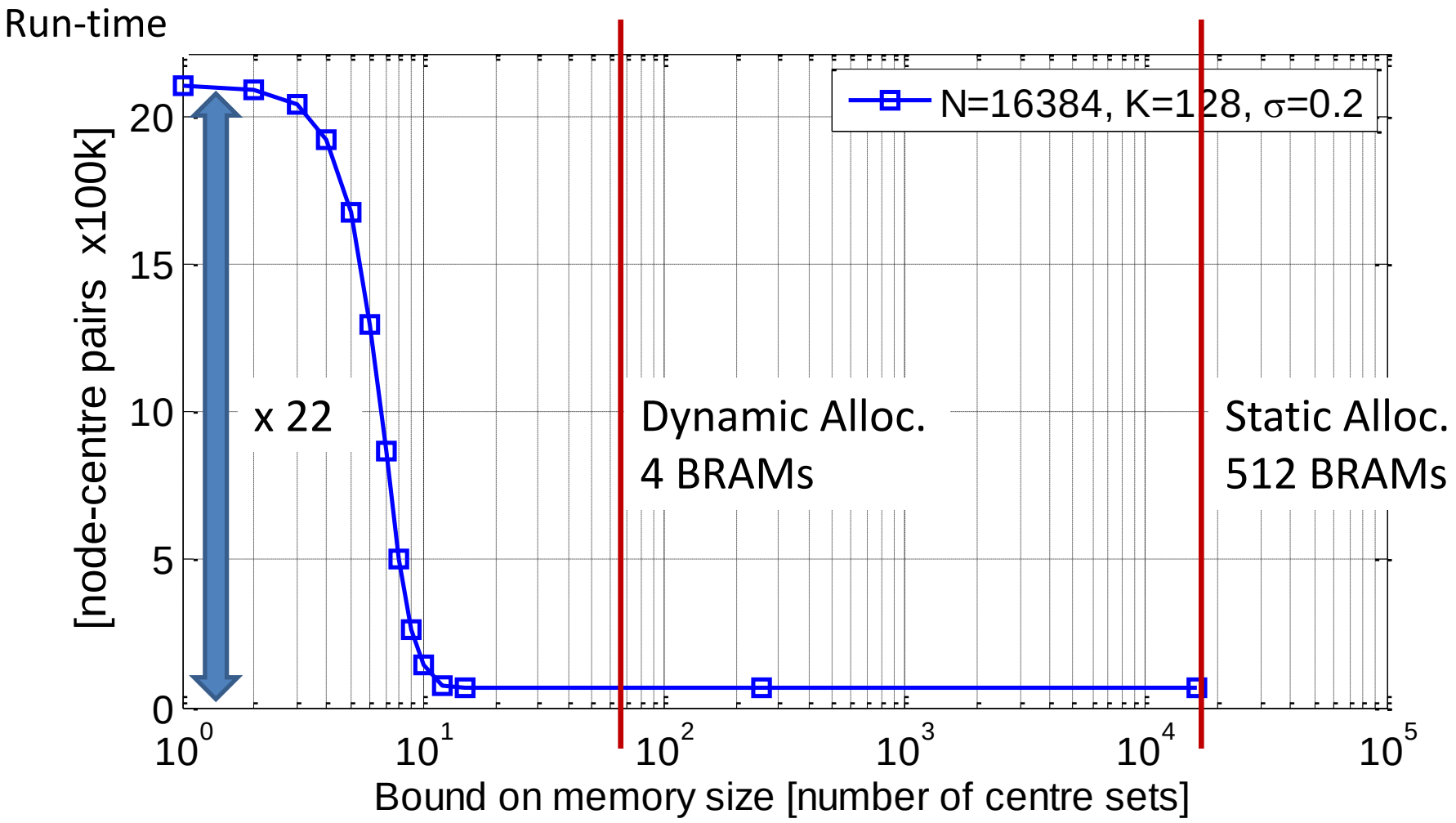
- Worst-case: N-1 sets
- N=16384, K=256:
  -> 32 Mbits (81% of BRAMs in medium-size Virtex 6 FPGA)
- Reuse memory space
- Dynamic memory allocation
- How to support the worst case?

Dynamic memory allocation



- Worst-case: N-1 sets
- N=16384, K=256:
  -> 32 Mbits (81% of BRAMs in medium-size Virtex 6 FPGA)
- Reuse memory space
- Dynamic memory allocation
- How to support the worst case?
- Degrades pruning, degrades runtime

Memory size vs. Runtime



Outline

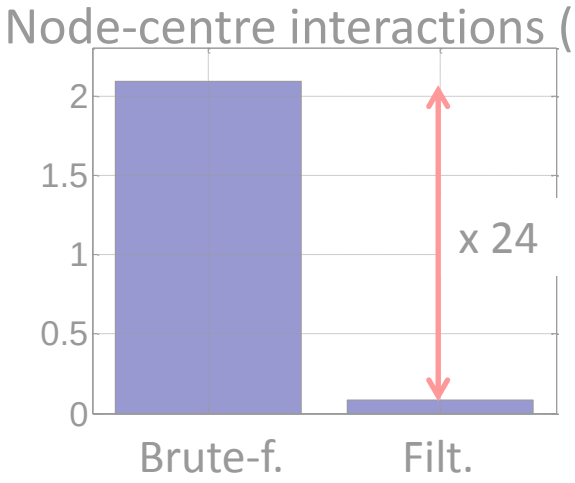
- Pointer-based dynamic data structures on FPGA
- Pipelining and parallelisation
- Dynamic memory access
- **Maintaining efficiency**
- Conclusion

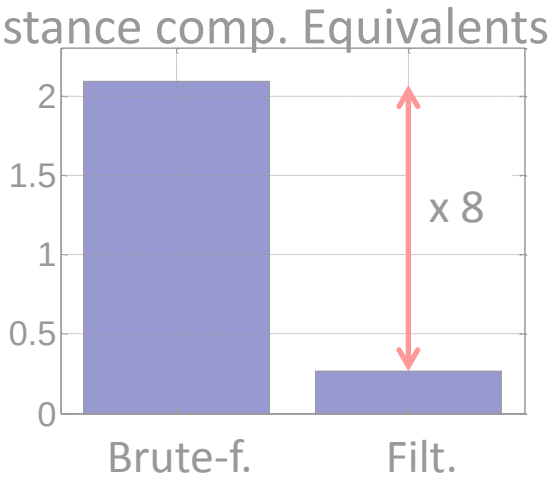
Imperial College London

Performance comparison

esa

SW

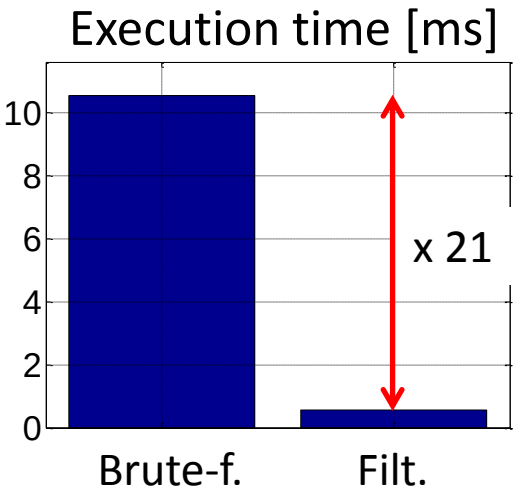


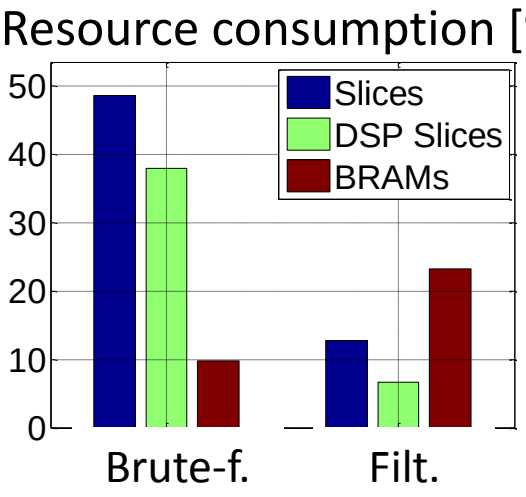


N=16384,
K=128, D=3

FPGA

Virtex 7 FPGA





Throughput constraint: 128us / clustering iteration

Conclusion

- Pointer-based recursive algorithm using dynamic data structures on FPGA
- Specialised pipeline:
  - Benefit from hardware implementation
  - Maintain efficiency seen in SW comparison
- Custom dynamic memory allocation:
  - Use memory space efficiently
- Future work:
  - Interface to off-chip memory
  - Investigate implications for research on high-level synthesis tools

Thank you.

Questions?