High-Level Synthesis of Dynamic Data Structures: A Case Study Using Vivado HLS

Felix Winterstein | Samuel Bayliss | George Constantinides
f.winterstein12@imperial.ac.uk

Imperial College London

Abstract

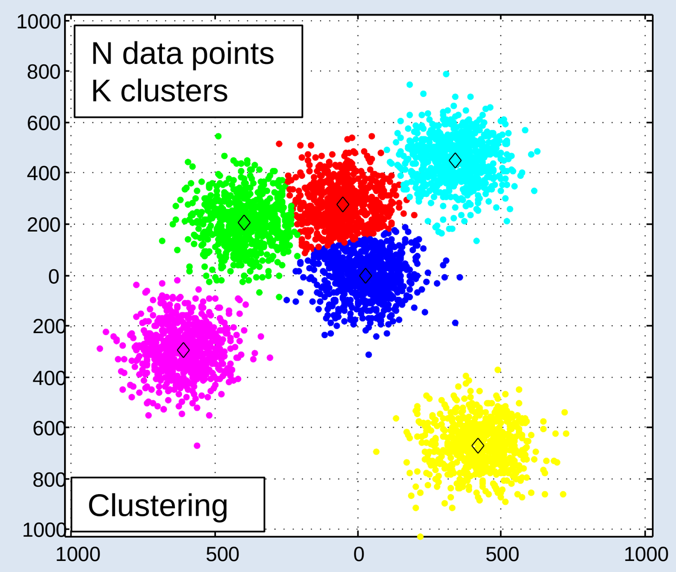
Recent evaluations report that C-to-FPGA flows can produce results with a quality close to hand-crafted designs [1]. Programs which use dynamic, pointer-based data structures remain difficult to implement well, yet these structures are common in software. We compare a data-flow centric implementation to a recursive tree traversal implementation which makes use of pointer-linked data structures. We present source code transformations that ensure synthesisability and improve the quality of results. The automation of these transformations motivates future research.

Case Study

- Compare two equivalent algorithms for K-means clustering: **Lloyd's Algorithm** and the **Filtering Algorithm** [2]
- Determine required source code refactoring

Algorithm	Lloyd's Algorithm	The Filtering Algorithm
Clustering result	Both algorithms produce the same result	
Algorithm complexity[1]	$210 \cdot 10^4$ node-centre interactions	$9 \cdot 10^4$ node-centre interactions
Computational properties	• Simple control flow • Embarrassingly parallel • Well-suited for FPGAs	• Uses tree data structure built from the data set • Recursive tree traversal • Dynamic memory allocation
Implementation in Vivado HLS	Seamless C-to-FPGA implementation	Requires substantial source code modifications

[1]Synthetic data set, 16384 data points, 128 clusters, search complexity proportional to execution time

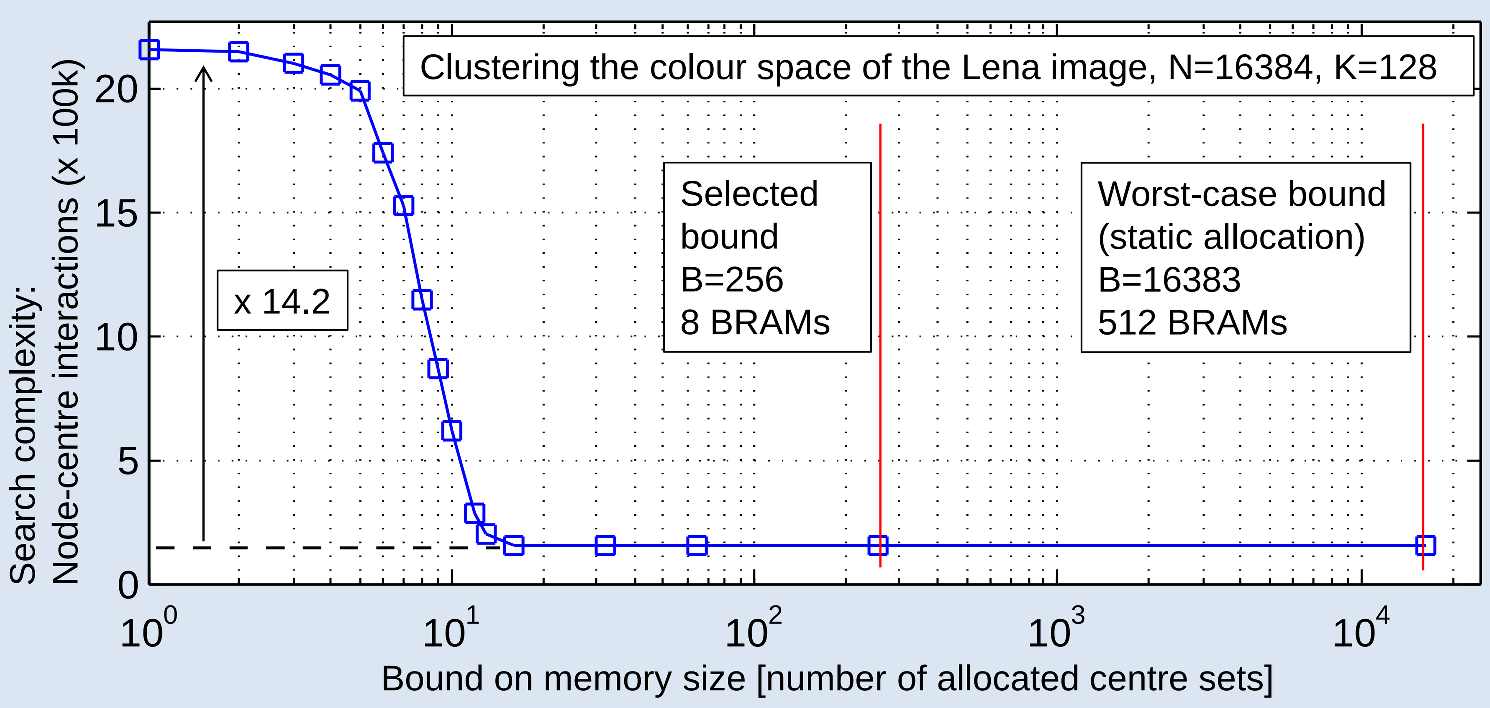



```
function LLOYDS( )
  for all x_i ∈ {x_1, x_2, ..., x_N} do
    for all z_j ∈ {z_1, z_2, ..., z_K} do
      d² ← ||x_i − z_j||²
    end for
    z^ ← closest centre to x_i
    updateCentreBuffer(z^)
  end for
end function
```

```
function FILTER(treeNode u, CentreSet Z)
  z^ ← closest centre ∈ Z to u.midPoint
  if u is leaf then
    updateCentreBuffer(z^)
  else
    new newZ
    newZ ← pruneSet(Z, z^, u.bndBox)
    if |newZ| > 1 then
      FILTER(u.left, newZ)
      FILTER(u.right, newZ)
      delete newZ
    else
      ...
      delete newZ
    end if
  end if
end function
```

Why Pointers?

- Unbalanced, data-dependent tree shape: Pointer-linked data structure
- Dynamic instead of static memory allocation: Significant on-chip memory savings (reuse memory space)



4 Code Transformations - Filtering Algorithm

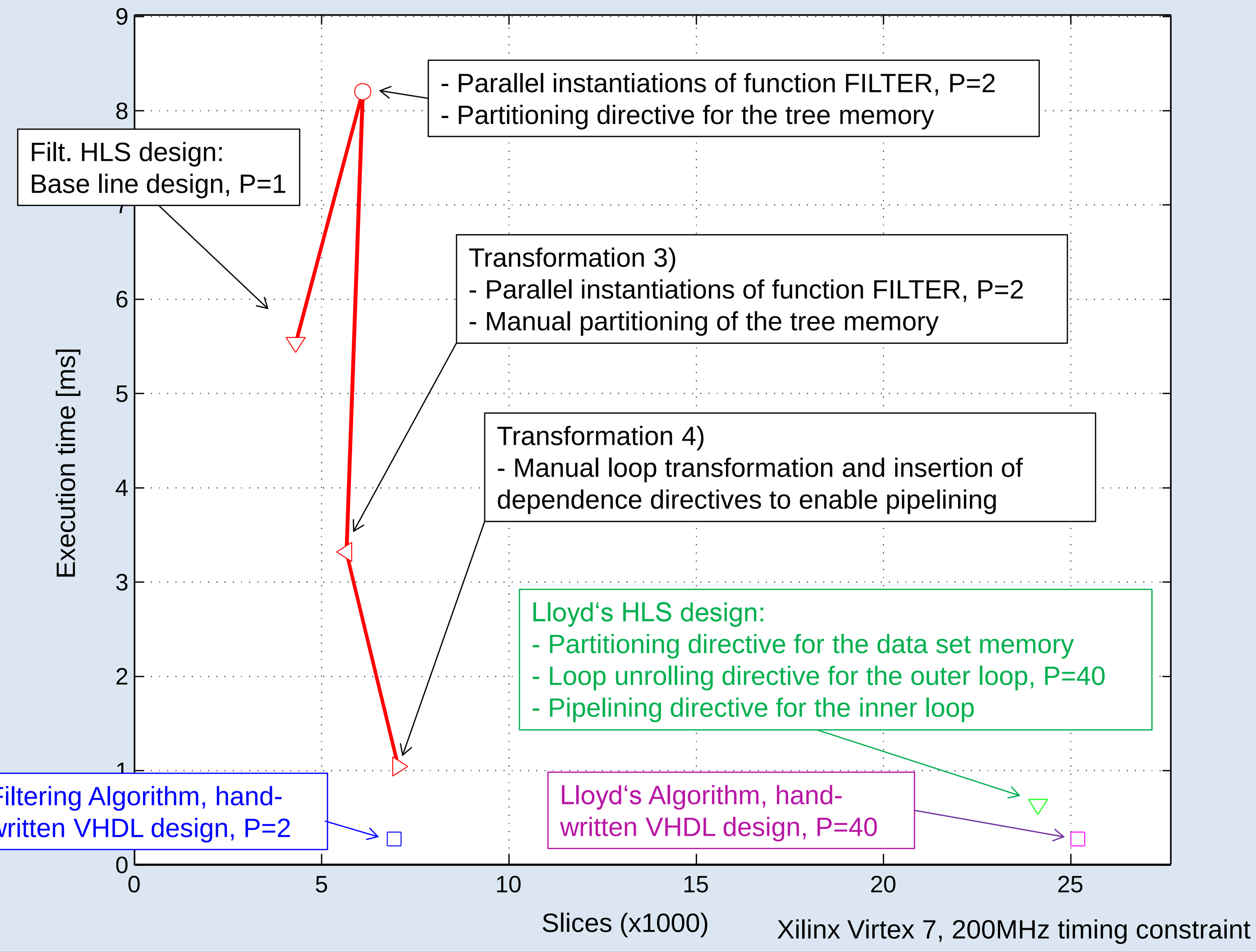
1. On-chip dynamic memory allocation: Custom implementation of the allocator
2. Standard approach to handle recursion: Replaced by a while-loop and a stack
3. Manual partitioning of the tree data structure, privatisation of heap memory for centre sets and memory allocators
4. Loop distribution to enable efficient pipelining

```
while stack not empty do
  while (stack not empty) and (queue not full) do      [pipeline]
    pop (pointers to treeNode u, centreSet Z)
    enqueue (pointers to treeNode u, centreSet Z)
  end while
  while queue not empty do                             [pipeline]
    dequeue (pointers to treeNode u, centreSet Z)
    ... remaining loop body ...
  end while
end while
```

Some pointers on the stack alias, but read-write dependences are preserved by the transformation

Results

- Taken from placed and routed FPGA designs
- Lloyd's implementation achieves 'close to hand-written' performance using synthesis directives only
- $> 5\times$-Improvement of latency due to manual source code modifications for the Filtering Algorithm



Xilinx Virtex 7, 200MHz timing constraint

Conclusion

- State-of-the-art HLS tools can achieve 'close to hand-written' performance
- Lack of automated optimisations for programs using pointers and dynamic structures: extensive code transformations required
- Future work focuses on a dependence / disjointness analysis for data structures accessed through pointers in order to automate these code transformations

References

1. BDTI, "An Independent Evaluation of: The AutoESL AutoPilot High-Level Synthesis Tool," 2010.
2. T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, A. Wu, "An Efficient K-Means Clustering Algorithm: Analysis and Implementation," IEEE Trans. Pattern Anal. Mach. Intell., Jul. 2002.

Acknowledgements

This work is sponsored by the European Space Agency under the Networking/Partnering Agreement No. 4000106443/12/D/JR and by the EPSRC under grant numbers EP/I020357 and EP/I012036.