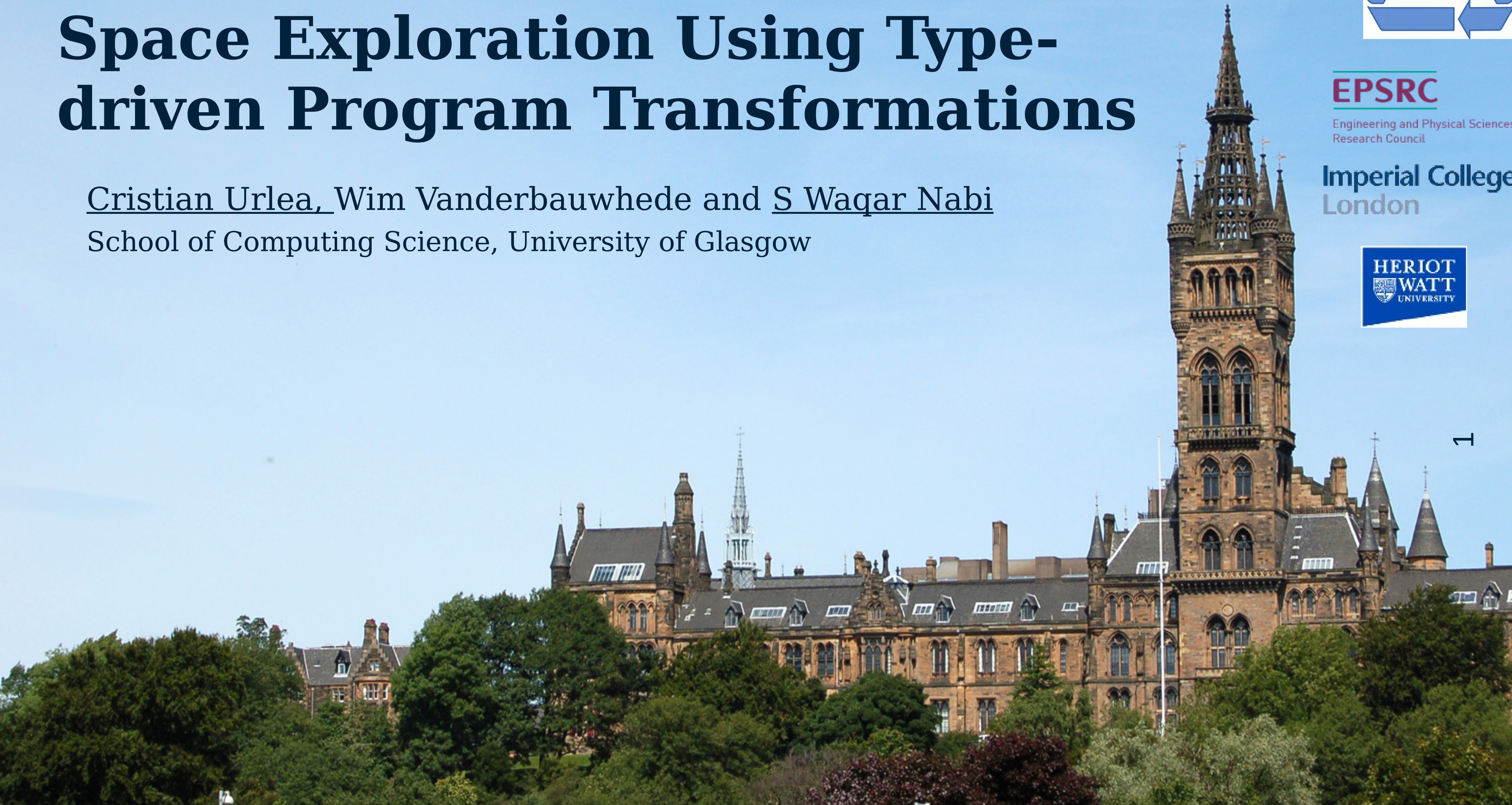
SPLS
07 March 2018
Glasgow, Scotland

Efficient FPGA Cost-Performance Space Exploration Using Type-driven Program Transformations

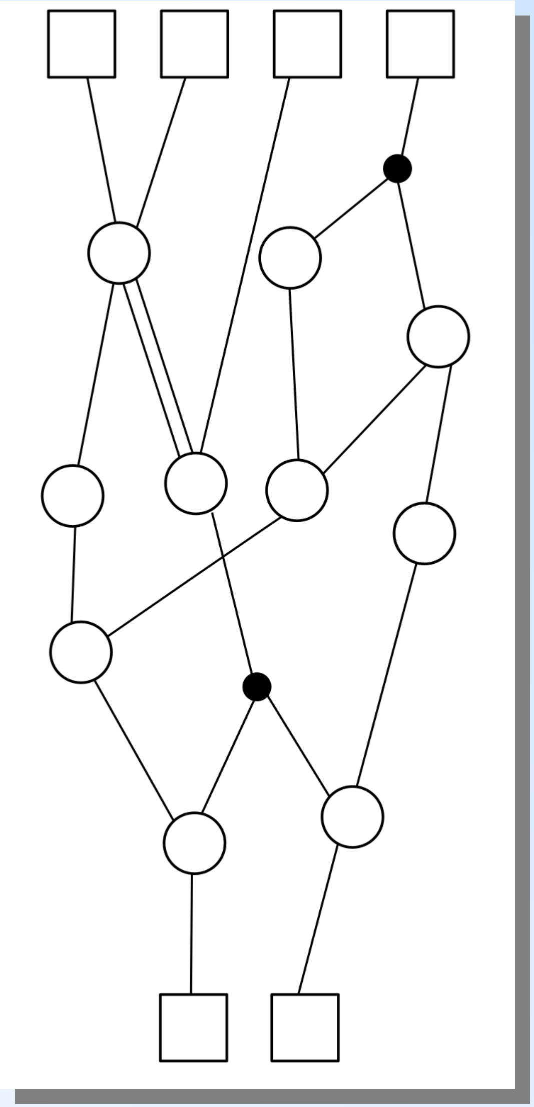
Cristian Urlea, Wim Vanderbauwhede and S Waqar Nabi
School of Computing Science, University of Glasgow



TyTra and FPGAs

- TyTra
  - Is a compiler framework concerned with optimising streaming data-flow applications
  - Targets FPGAs in particular, and hybrid many-core systems more broadly because:
    - FPGAs provide good performance/watt
    - FPGAs are under-represented in the HPC space
    - Existing HLS tools like OpenCL and Maxeler produce solutions which are difficult to tune.

- This presentation:
  - Covers design-space exploration in TyTra
  - Work supported by EPSRC – Grant number EP/L00058X/1



TyTra Workflow

- TyTra Workflow:
  - **Identify scalar functions**
  - **Identify uses of scalar functions through higher-order functions**
  - **Create a TyTra Coordination Language Specification**
  - **Apply semnatics-preserving transformations to yield many program variants in the design space**
  - **Run program variants through a cost-model to estimate performance and resource use**
  - **Pick the best program variant and generate target-architecture code**

TyTra Workflow

```
newtype TermBB = TermBB {
 unTermBB :: forall a.
   (String -> a )               -- varlit
   -> (  a -> a -> a )          -- app
   -> ( [a] -> a )             -- term tup
   -> a                        -- zipbb
   -> a                        -- unzip
   -> ( IndexListBB -> a )     -- stencil
   -> (  a -> a )              -- map
   -> (  NatBB -> a )           -- elt
   -> a
}
```

```
newtype TypeBB =
 TypeBB {
  unTypeBB :: forall a.
    (TVar -> a )                  -- TypeVar
    -> (String -> a)               -- literal
    -> ( a -> a -> a )             -- internal products
    -> (Tag -> Size -> a -> a)   -- sized vectors
    -> ([a] -> a)                -- heterogenous lists / tuples
    -> (a -> a -> a)               -- fun a a
    -> a
 }
```

http://okmij.org/ftp/tagless-final/course/Boehm-Berarducci.html

TyTra Workflow

```
substituteBB :: Subst -> TypeBB -> TypeBB
substituteBB s f =  unTypeD (deconTy f) dtyvar tyconbb dtyprod dtyvec dtytup dtyfun
 where
   dtyvar t@(TVar tvId )  = M.findWithDefault f t (typeSubs s)
   dtyprod a b             = prodbb ( substituteBB s a) ( substituteBB s b)
   dtyvec t sz@(SizeVar  sv) ity        = vecbb' t (M.findWithDefault sz sv $ sizeSubs s )
( substituteBB s ity )
   dtyvec t sz@(SizeConst _) ity        = vecbb' t sz ( substituteBB s ity )
   dtytup itys             = tuplbb (map (substituteBB s ) itys )
   dtyfun a b              = funbb ( substituteBB s a) ( substituteBB s b)
```

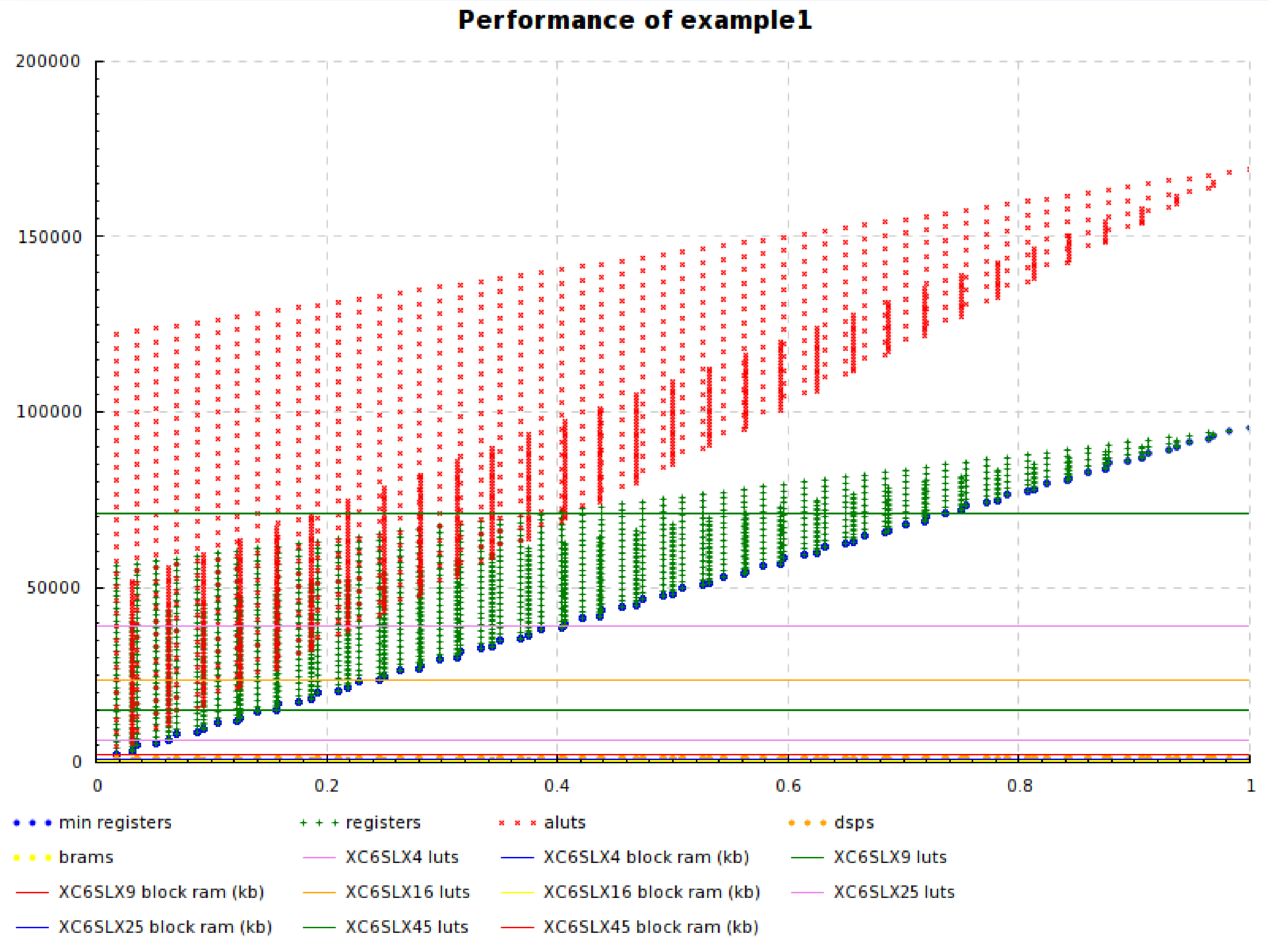
http://okmij.org/ftp/tagless-final/course/Boehm-Berarducci.html

TyTra Workflow

```
genVariants :: TermBB -> CostEnv -> [Integer]
genVariants term costEnv = unTermBB term lit app ttup tzip tuzip tstencil tmap telt
  where
    lit name  =
      let
        maybeEFI  = efi . fst <$> Map.lookup name costEnv
        in
          maybe [1] (\efi -> [1 .. efi])  maybeEFI

    app f x = nubOrd $ concat [f,x]
    ttup fs = nubOrd $ concat fs --  minLength fs
    tzip  = [1]
    tuzip = [1]
    tstencil its = [1]
    tmap a = a
    telt n = [1]
```

TyTra Workflow


Performance of example1
min registers
registers
aluts
dsps
brams
XC6SLX4 luts
XC6SLX4 block ram (kb)
XC6SLX9 luts
XC6SLX9 block ram (kb)
XC6SLX16 luts
XC6SLX16 block ram (kb)
XC6SLX25 luts
XC6SLX25 block ram (kb)
XC6SLX45 luts
XC6SLX45 block ram (kb)