Weaving Parallel Threads
Searching for Useful Parallelism in Functional Programs

José Manuel Calderón Trilla, University of York
Simon Poulding, Blekinge Institute of Technology
Colin Runciman, University of York

Vision

automated parallelisation

program written without reference to parallelism

→

transformed program exploits parallelism of target hardware

Desirable Properties

safe
transformed program
is functionally correct

worthwhile
transformed program
is faster

Our Approach

safe
transformed program
is functionally correct

worthwhile
transformed program
is faster

static analysis
what **can** be parallelised?

metaheuristic search
what **should** be parallelised?

Context:
Functional Programs

Purity

```
second (x:y:xs) = y

myList = [3,8,7,4]

> second myList
8
```

Lazy Evaluation

```
errList = [head [], 12, 3]

> second errList
12

infList x = x:(infList (x+1))

> second infList 3
4
```

`par` primitive

```
par a b
```

returns result of **`b`** while evaluating **`a`** in a new thread

```
f x
```

```
par x (f x)
```

speculatively evaluate argument in parallel

Safe Parallelisation: Static Analysis

Strictness Analysis

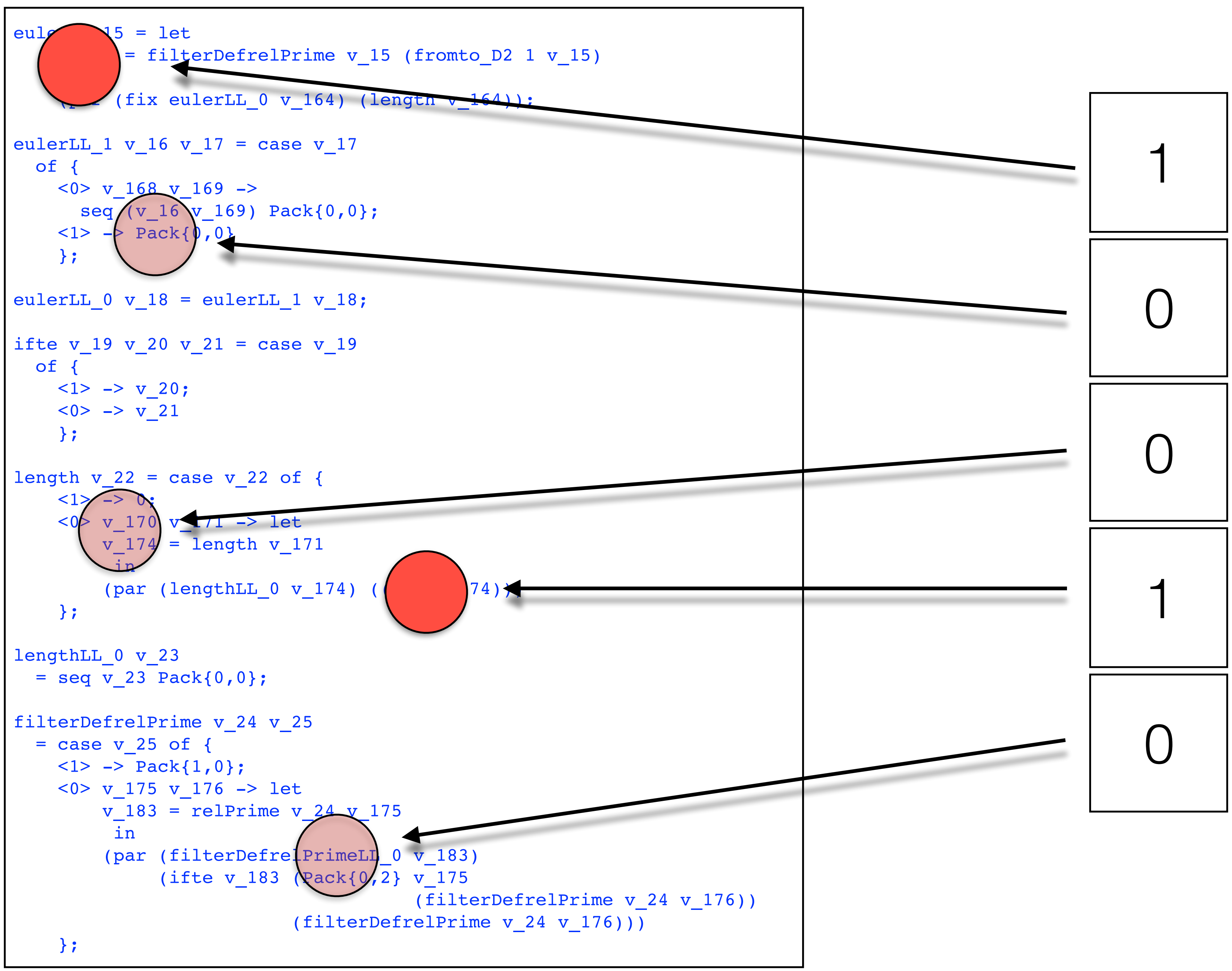
```
errList = [head [], 12, 3]

> par (force errList) (f errList)
EXCEPTION: …

infList = x:(infList (x+1))

> par (force infList) (f infList)
```

par-sites

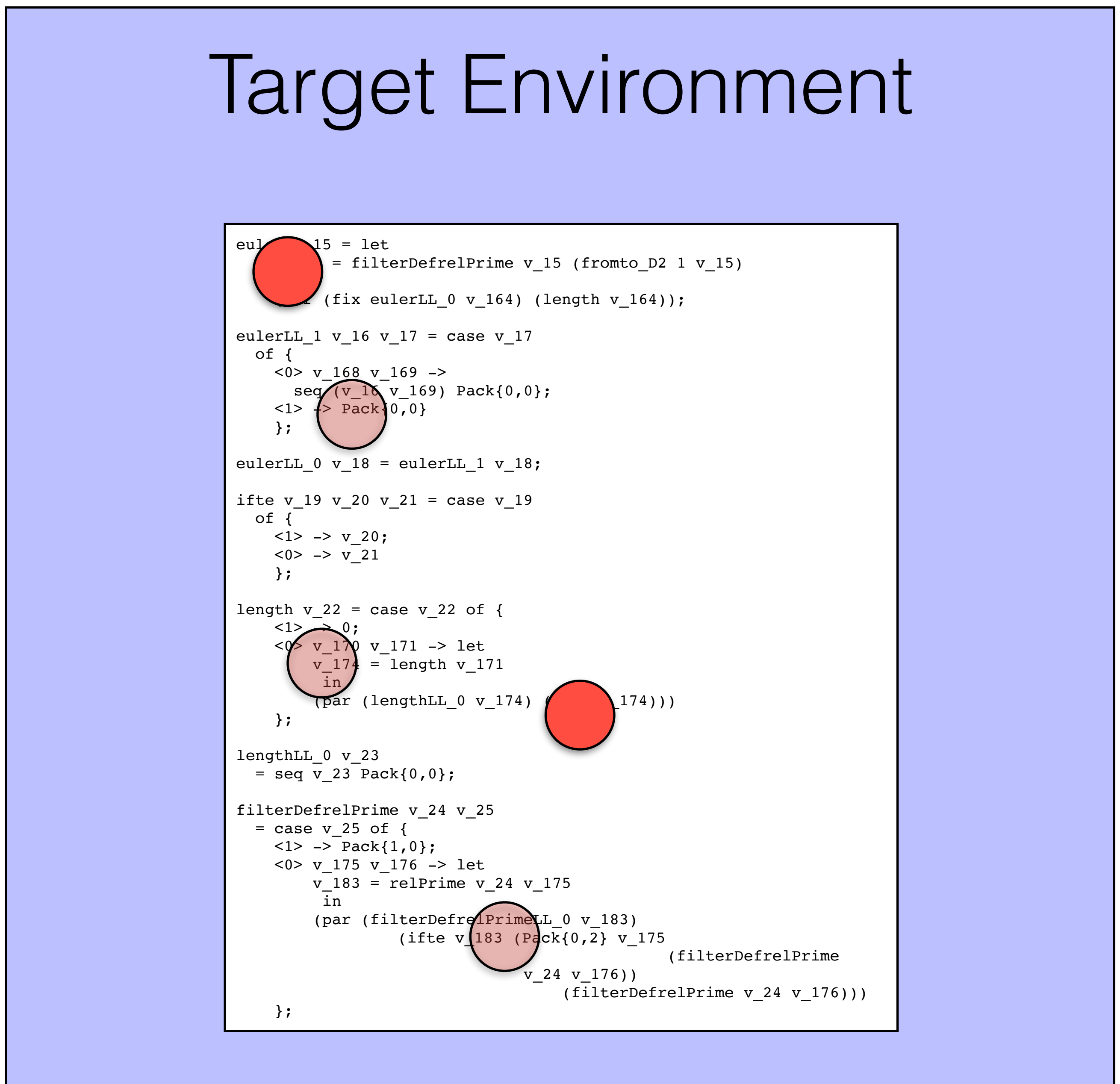
```
eul       15 = let
           = filterDefrelPrime v_15 (fromto_D2 1 v_15)

    (f     (fix eulerLL_0 v_164) (length v_164));

eulerLL_1 v_16 v_17 = case v_17
  of {
    <0> v_168 v_169 ->
      seq        v_169) Pack{0,0};
    <1> -        ),0}
    };

eulerLL_0 v_18 = eulerLL_1 v_18;

ifte v_19 v_20 v_21 = case v_19
  of {
    <1> -> v_20;
    <0> -> v_21
    };

length v_22 = case v_22 of {
    <1>      ;
    <0          v_171 -> let
                = length v_171

        (par (lengthLL_0 v_174) (        74)))
    };

lengthLL_0 v_23
  = seq v_23 Pack{0,0};

filterDefrelPrime v_24 v_25
  = case v_25 of {
    <1> -> Pack{1,0};
    <0> v_175 v_176 -> let
        v_183 = relPrime v_24 v_175
         in
        (par (filterDefrel        _0 v_183)
             (ifte v_183 (        ,2} v_175
                                   (filterDefrelPrime v_24 v_176))
                        (filterDefrelPrime v_24 v_176)))
    };
```

Worthwhile Parallelisation: Metaheuristic Search

Representation



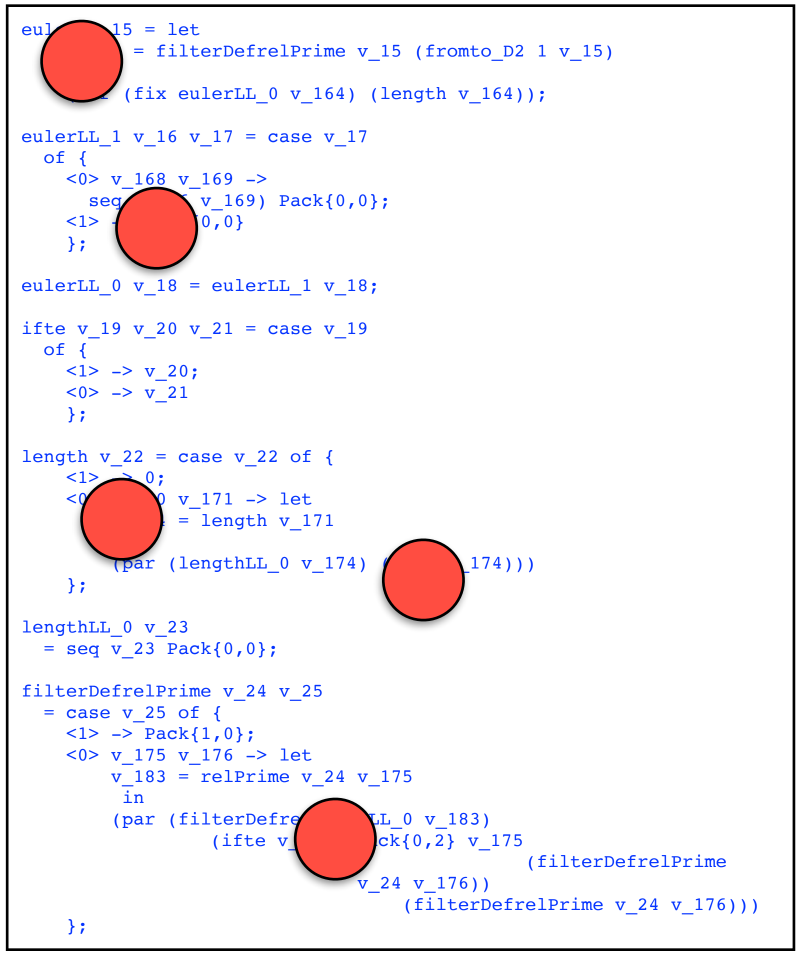
Fitness



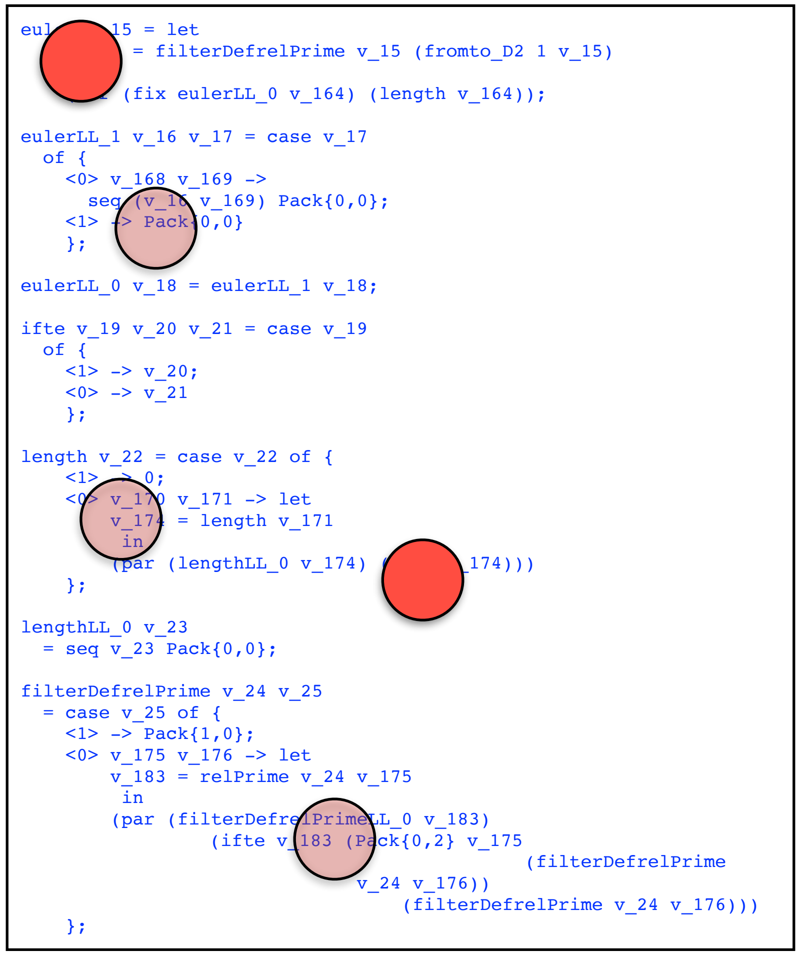
number of reductions
on the main thread

Empirical Investigation

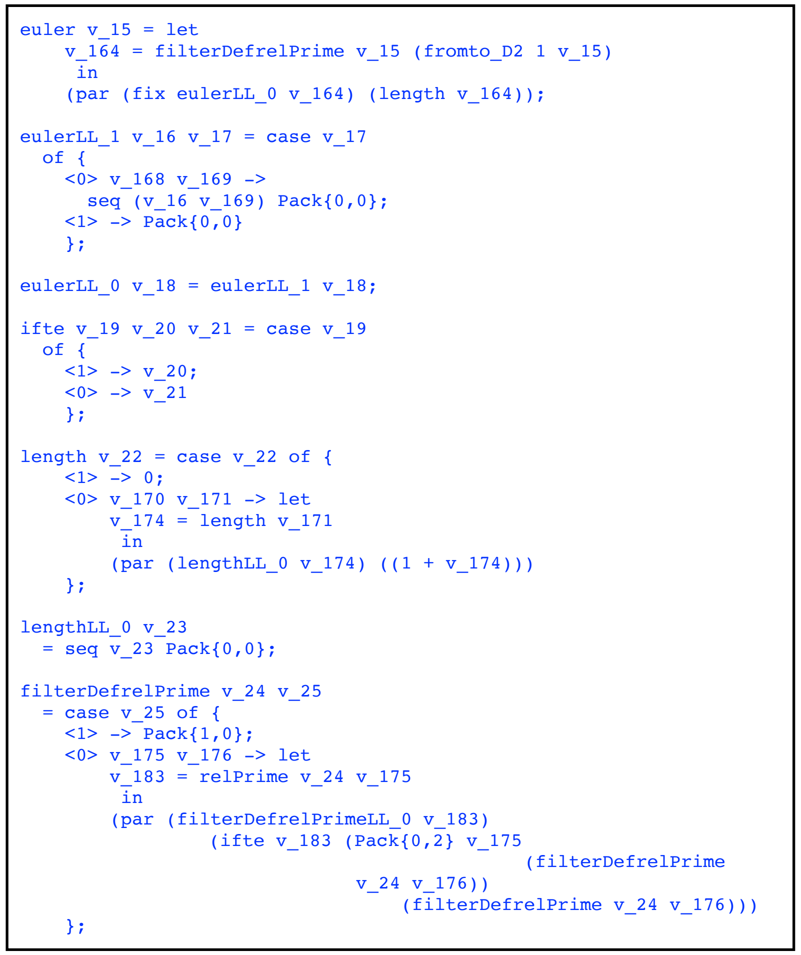
RQs: speed-up



all **pars**



optimised **pars**



sequential

RQs: search

method

randomised greedy ⟷ simple hill-climbing

initialisation

all `par`-sites enabled ⟷ random `par`-sites enabled

Programs

SumEuler

Queens

Queens2

SodaCount

Tak

Taut

MatMul

Empirical Method

7 programs

x

2 search methods

x

all par-sites | 2 initialisation methods | no par-sites

x

30 repetitions

x

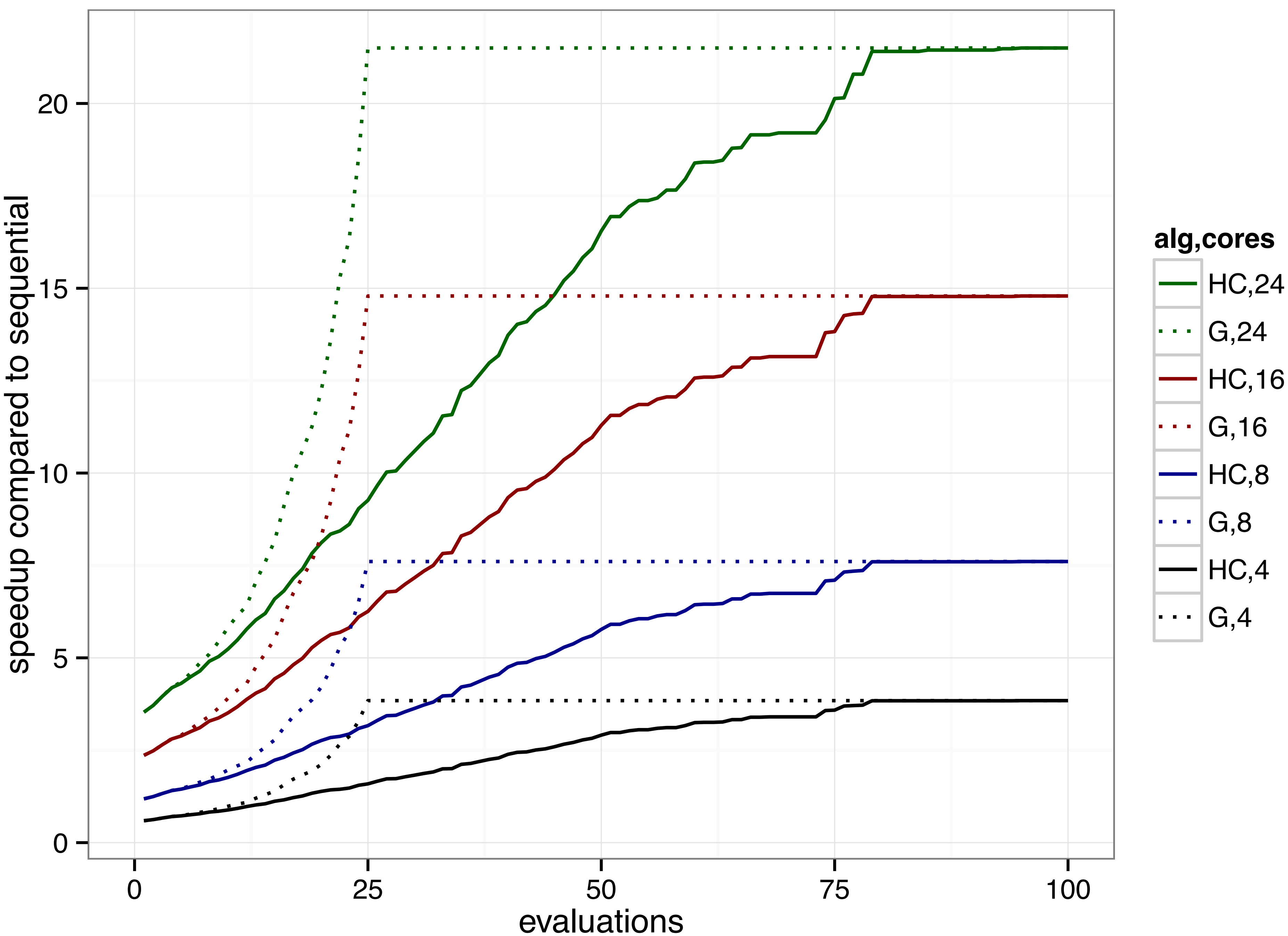
4 target environments (4, 8, 16, 24 cores)

Results: initialisation

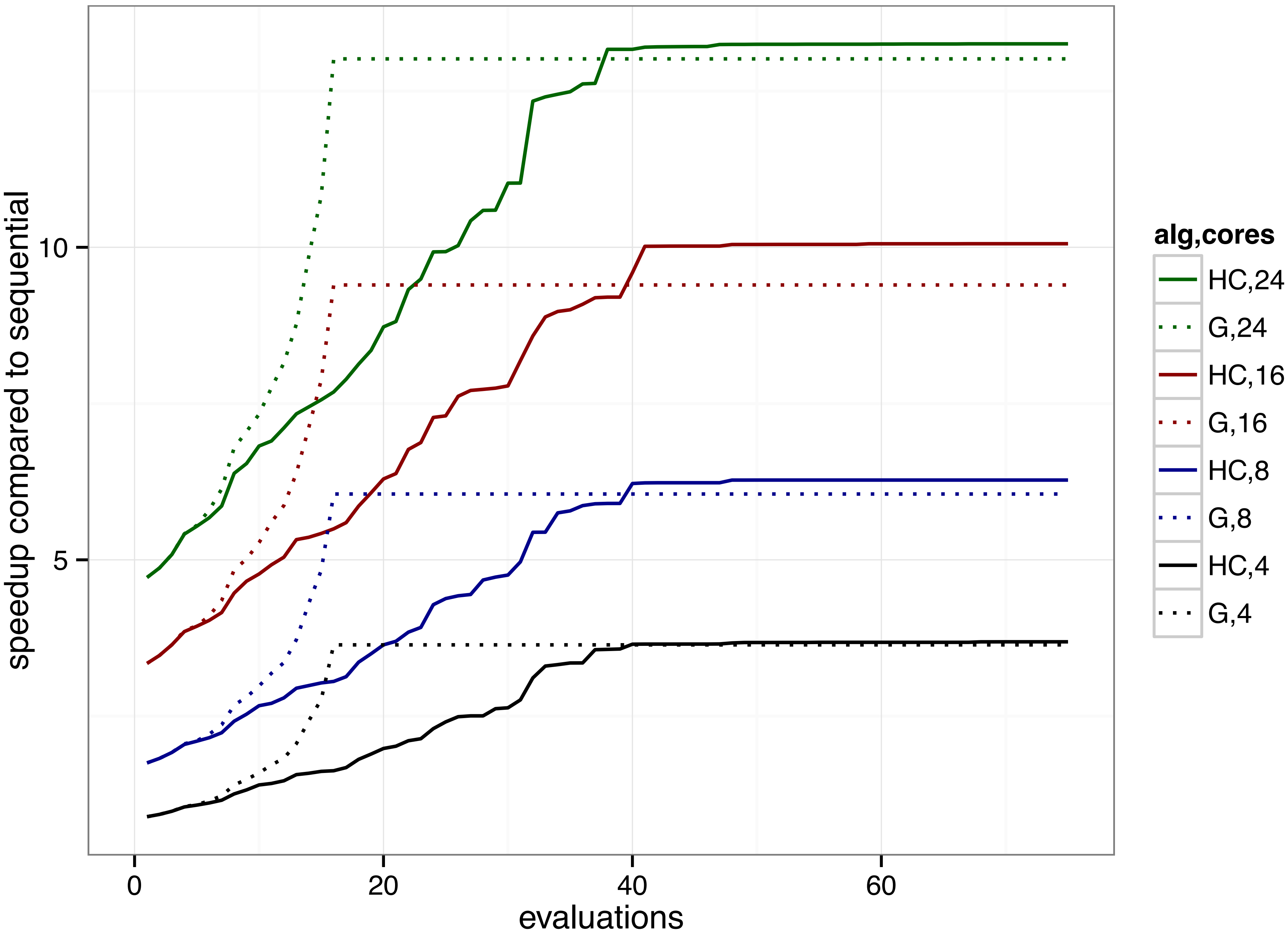
most programs: no significant difference

SodaCount: enabling all `par`-sites was slightly better

Results: Queens2



Results: SodaCount



Conclusions and Future Work

Summary

A combination of static analysis and search that can **automatically** and **effectively** parallelise functional programs to take advantage of parallel computing environments

Future Work

investigate scalability

apply more sophisticated search algorithms