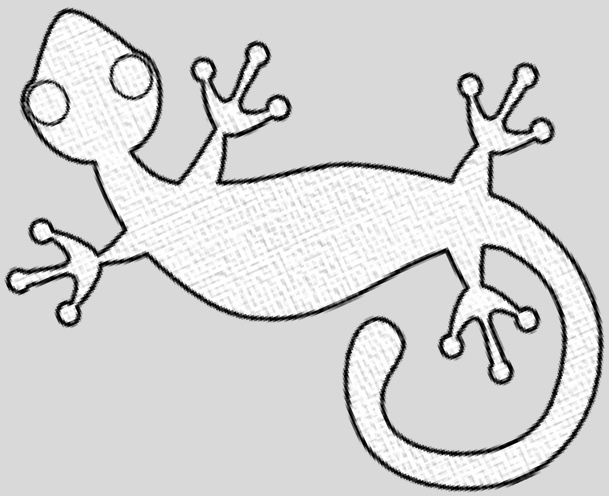
GPU Parallel SubTree Interpreter for Genetic Programming

Alberto Cano and Sebastián Ventura
Knowledge Discovery and Intelligent Systems Research Group
University of Córdoba, Spain

Overview

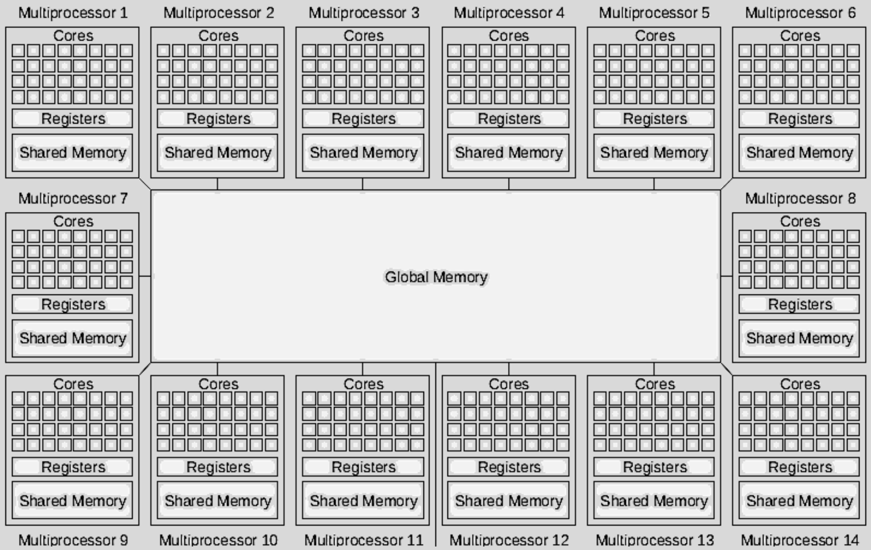
1. Parallelization approaches for GP evaluation
2. Stack-based GP interpreter
3. Parallel SubTree interpreter
4. Experiments
5. Conclusions
6. Future work



Parallelization approaches for GP evaluation

"Genetic Programming is embarrassingly parallel"

- Population parallel
  - Multi-core CPUs (acceptable for small population sizes)
  - Many-core GPUs (required for large population sizes)
- Data parallel
  - GP run on multiple fitness cases (thousands, millions)
  - GPU SIMD viewpoint



Parallelization approaches for GP evaluation

- Population and data parallel
  - 2D grid of threads for individuals and fitness cases

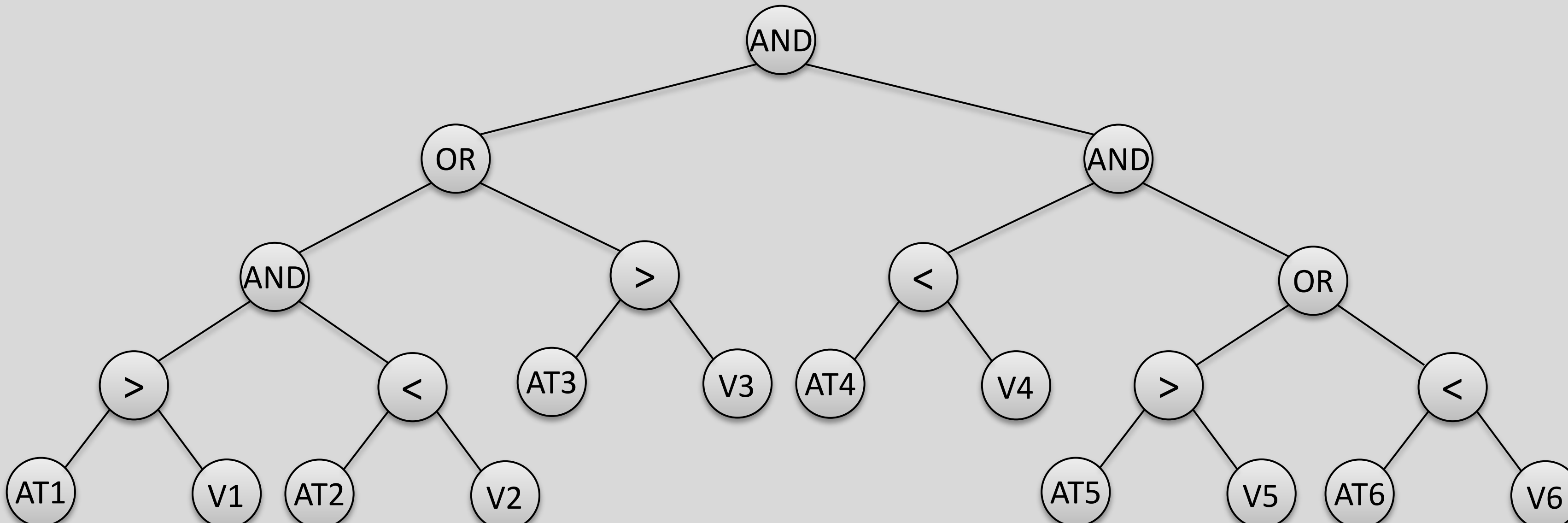
GP	GP	GP	GP	GP	GP	…	GP
data	data	data	data	data	data	…	data
data	data	data	data	data	data	…	data
data	data	data	data	data	data	…	data
data	data	data	data	data	data	…	data
data	data	data	data	data	data	…	data
…	…	…	…	…	…	…	…
data	data	data	data	data	data	…	data

- Performance hints:
  - Warp: single GP individual run on 32 fitness cases
  - GP individual in constant memory: single read - broadcast
  - Data coalescence: transposed data matrix

Stack-based GP interpreter

- Postfix notation: expression is evaluated left-to-right

V6 AT6 < V5 AT5 > OR V4 AT4 < AND V3 AT3 > V2 AT2 < V1 AT1 > AND OR AND

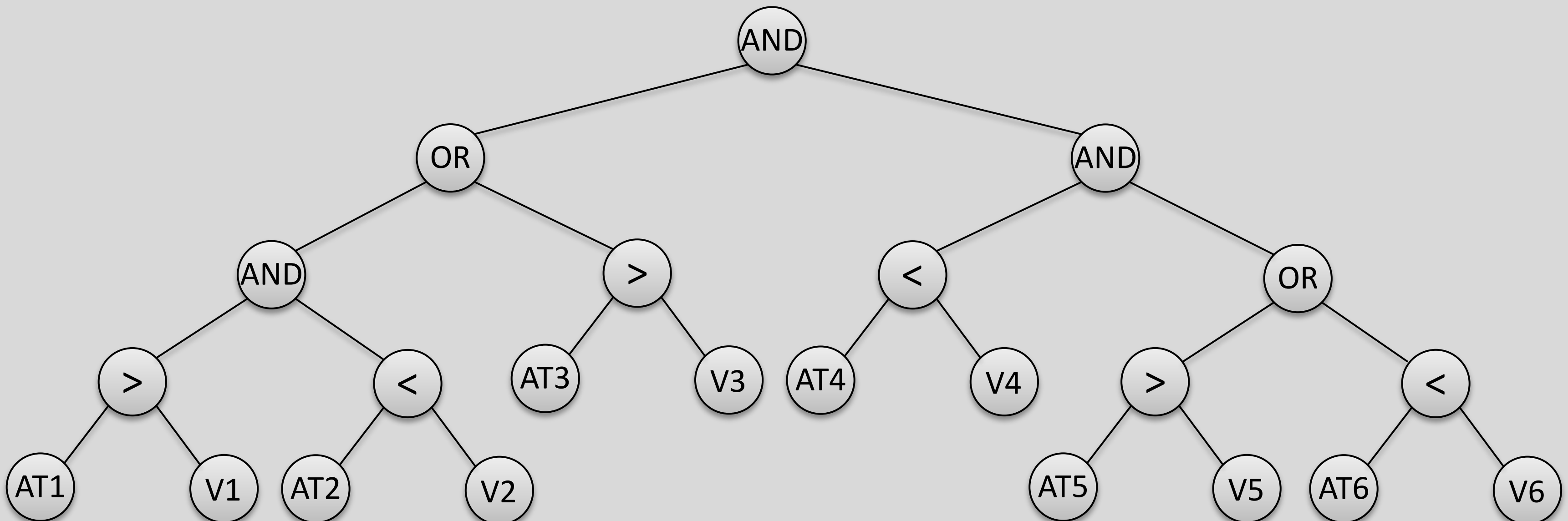


- O(n) complexity
- 23 push and 22 pop operations

Stack-based GP interpreter

- Mixed prefix and postfix notation:

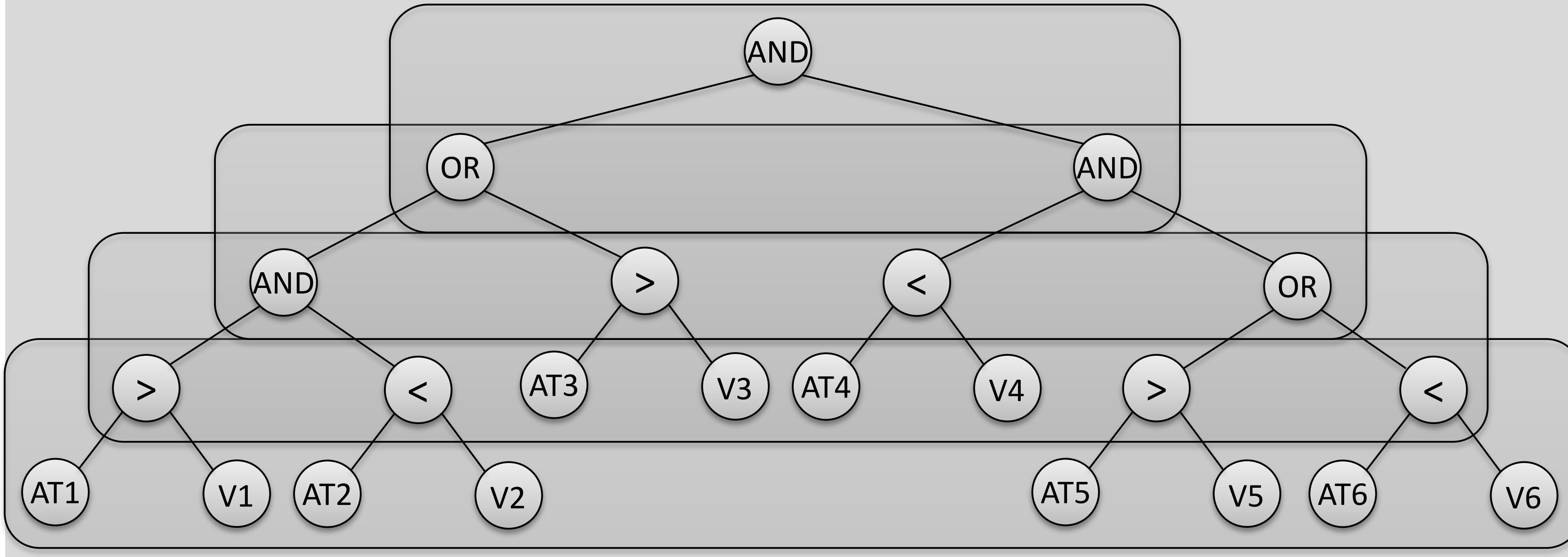
< AT6 V6 > AT5 V5 OR < AT4 V4 AND > AT3 V3 < AT2 V2 > AT1 V1 AND OR AND



- O(n) complexity
- 11 push and 10 pop operations

Parallel SubTree interpreter

- Computation of independent subtrees can be parallelized



- O(depth) complexity
- No stack depth needed
- Threads cooperation via shared memory
- Best performance on balanced trees

Parallel SubTree interpreter

```
__shared__ float stack[CONDITIONS][INSTANCES_BLOCK];

float* expression = &population[MAX_EXPR_LEN * blockIdx.y];
int instance = blockDim.x*blockIdx.x+threadIdx.x;
int threadExprIndex = 3*threadIdx.y;

for(int height = 0; height <= maxHeight;
    height++, numberActiveThreads/=2, threadExprIndex += 3*numberActiveThreads) {
    if(threadIdx.y < numberActiveThreads) {
        switch((int) expression[threadExprIndex]) {
            case GREATER:
                op1 = dataset[instance + (int) expression[threadExprIndex+1]*d_numberInstances];
                op2 = expression[threadExprIndex+2];
                stack[threadIdx.y][threadIdx.x] = (op1 > op2) ? 1 : 0;
                break;
            case LESS:
                op1 = dataset[instance + (int) expression[threadExprIndex+1]*d_numberInstances];
                op2 = expression[threadExprIndex+2];
                stack[threadIdx.y][threadIdx.x] = (op1 < op2) ? 1 : 0;
                break;
            case AND:
                op1 = stack[2*threadIdx.y][threadIdx.x];
                op2 = stack[2*threadIdx.y + 1][threadIdx.x];
                stack[threadIdx.y][threadIdx.x] = (op1 == 1) && (op2 == 1) ? 1 : 0;
                break;
            case OR:
                op1 = stack[2*threadIdx.y][threadIdx.x];
                op2 = stack[2*threadIdx.y + 1][threadIdx.x];
                stack[threadIdx.y][threadIdx.x] = (op1 == 1) || (op2 == 1) ? 1 : 0;
                break;
            default:
                break;
        }
    }
    else return;
    __syncthreads();
}

coverage[blockIdx.y*d_numberInstances + instance] = stack[0][threadIdx.x];
```

Full code at: (link available in the paper)
http://www.uco.es/grupos/kdis/wiki/GPevaluation

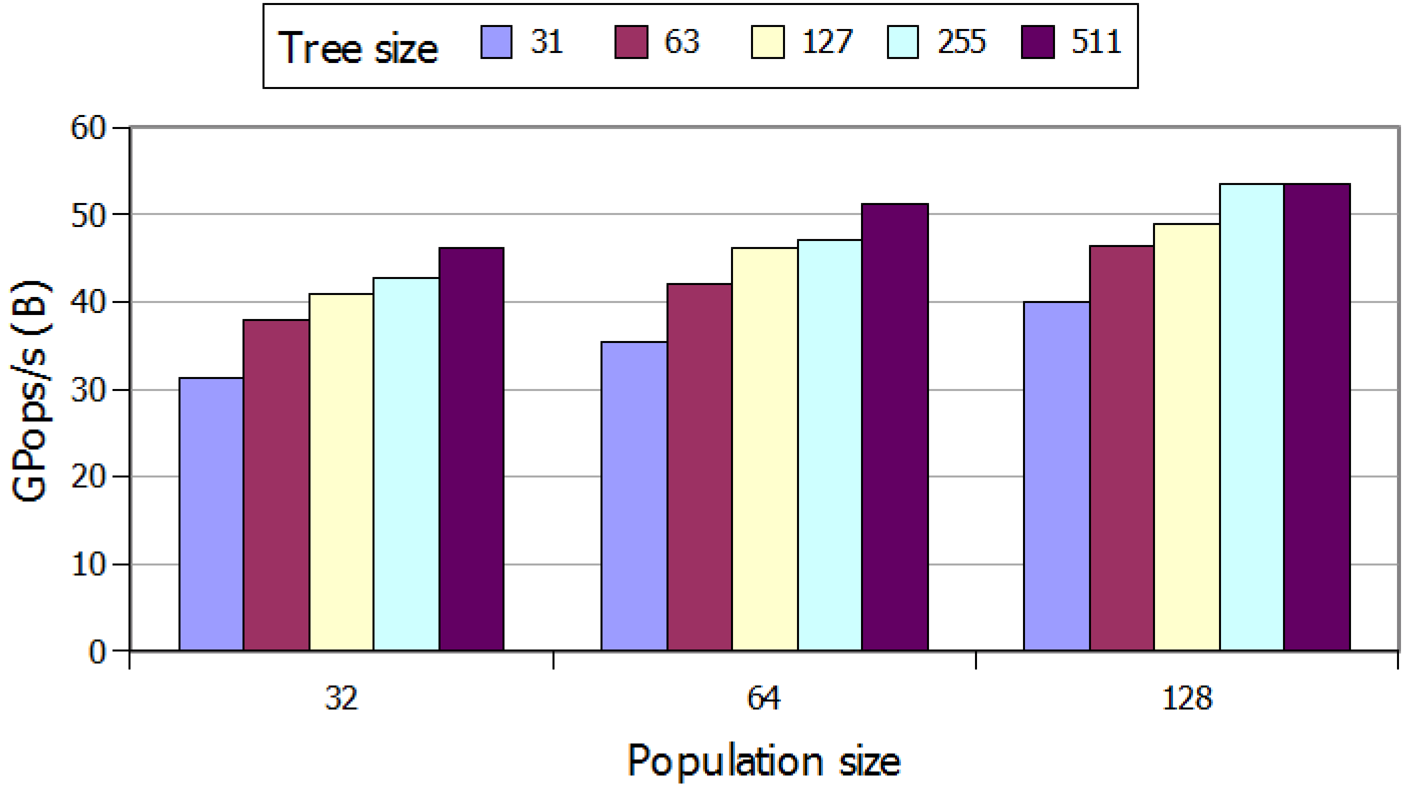
Experiments

- GPU: GTX 780 donated by NVIDIA
- Comparison: population and data parallel vs subtree parallel
- Datasets: 15
- Population size: 32, 64, 128
- Tree size: 31, 63, 127
- Performance measure: GPops/s
- How affects the population, tree and dataset size?

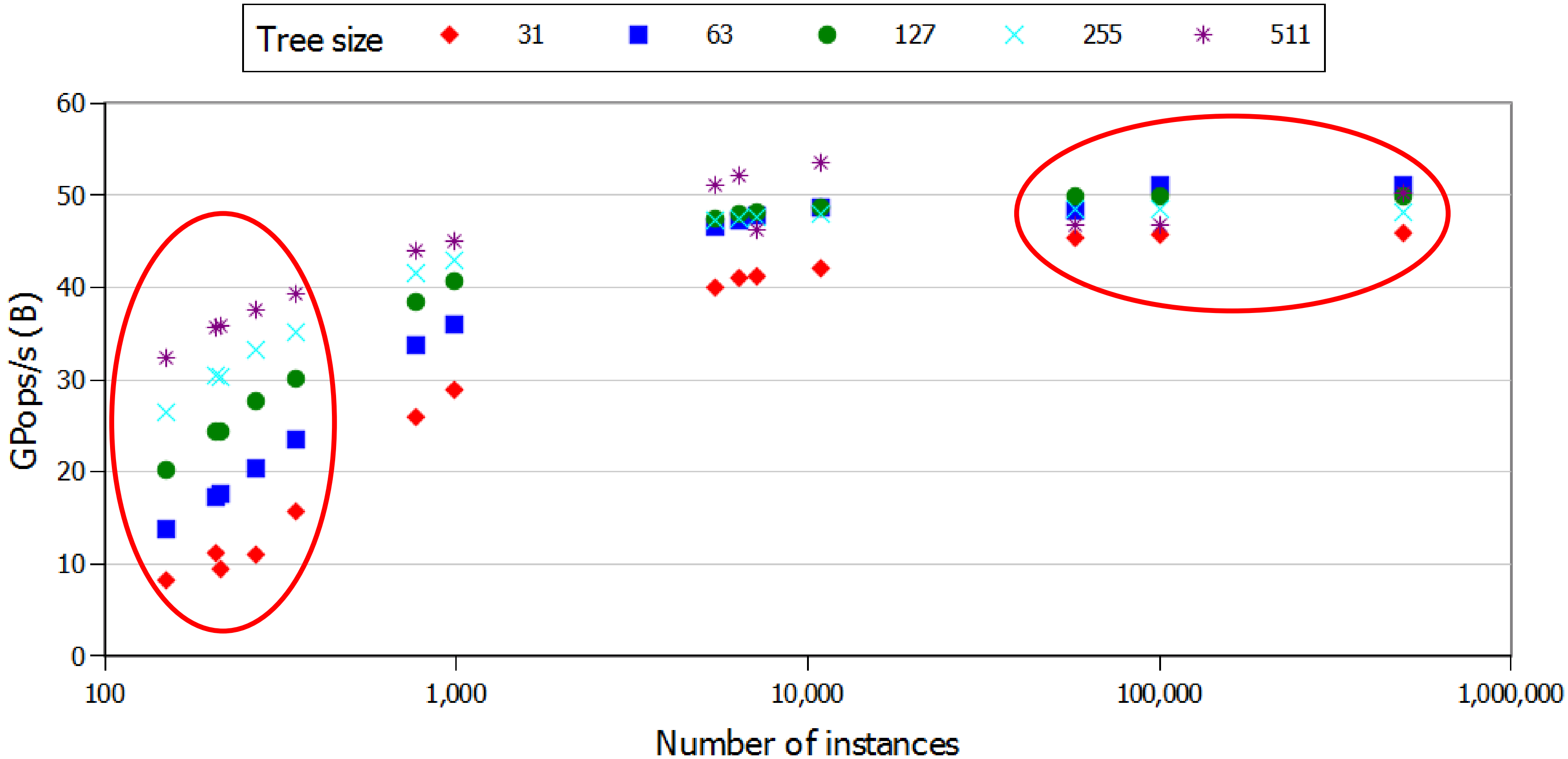
Experiments GPops/s (Billion)

iris	[illegible]	150	4	23.69	10.93	11.84	7.36	14.40	17.50	9.54	17.64	19.69
kddcup	[illegible]	494020	42	34.18	34.61	34.53	43.82	44.49	44.48	45.92	44.66	48.15
textile	2	5500	40	36.44	39.86	41.61	40.05	42.45	43.76	41.74	43.76	43.26
iris	[illegible]	150	4	8.21	14.24	20.93	13.70	20.87	29.18	20.07	28.53	36.19
kddcup	[illegible]	494020	42	45.89	44.92	45.94	50.96	50.92	51.11	49.79	50.78	50.88
textile	2	5500	40	39.90	43.56	45.19	46.59	48.87	45.77	47.40	48.70	49.30

- Performance variation when increasing population and tree size



- Performance variation when increasing data and tree size



- Performance increases as soon as there are enough individuals, subtrees or data to fill the GPU compute units

Conclusions

- Positive:
  - Mixed prefix/postfix notation
  - O(depth) complexity
  - No stack depth needed
  - Best for balanced trees
  - The higher tree density the better performance
- Negative:
  - Inappropriate for extremely unbalanced trees
  - Synchronization at each depth level
  - The number of active threads is reduced at each level
  - Limited by kernel size
  - Limited by shared memory

Future work

- Performance analysis: balance, density, and branch factor
- Scalability to bigger trees
- CUDA dynamic parallelism
  - Parent kernel can launch nested smaller kernel
- Kepler's shuffle instruction to avoid shared memory

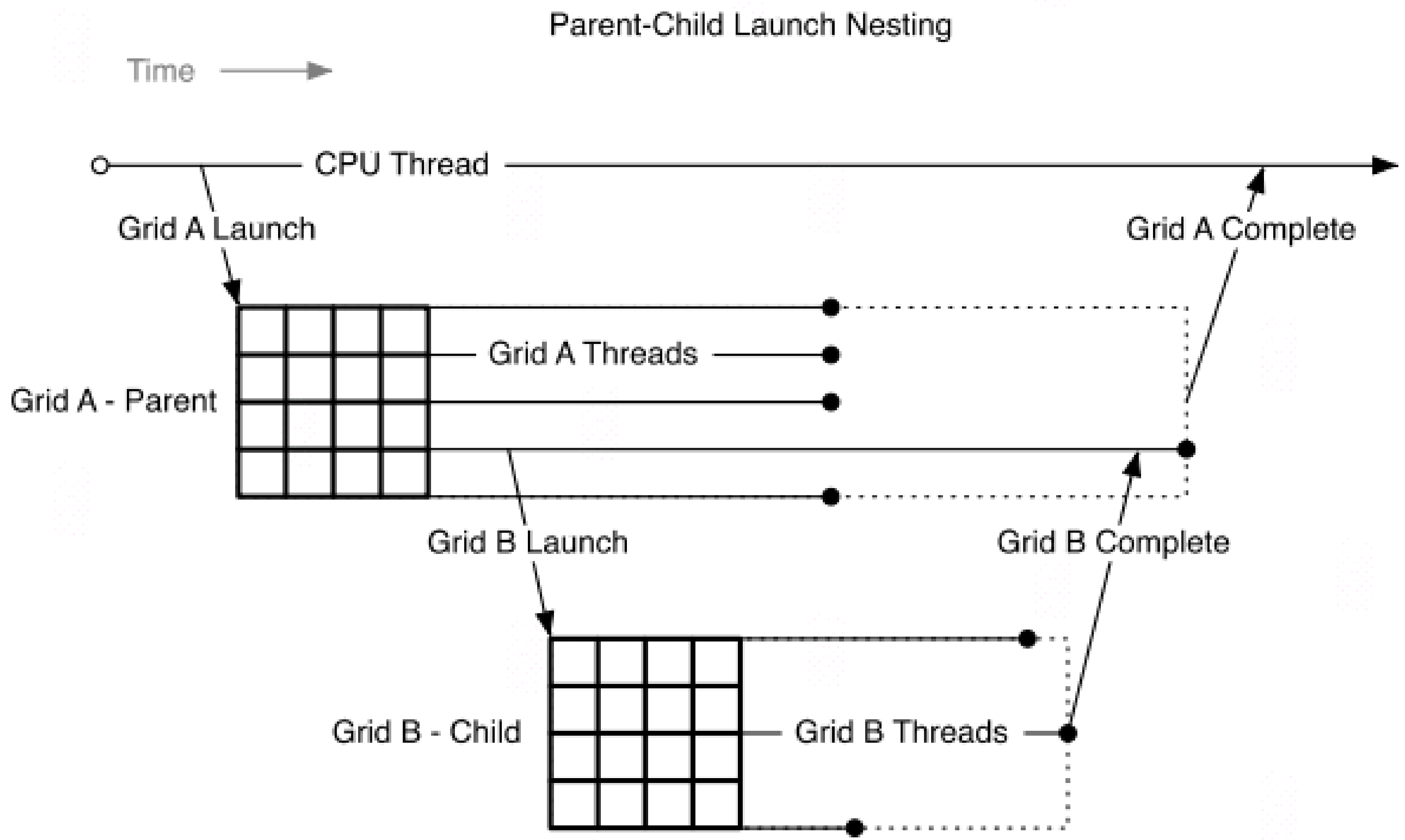
We gratefully acknowledge the support of NVIDIA Corporation with the donation of the GTX 780 GPU used for this research

GPU Parallel SubTree Interpreter for Genetic Programming

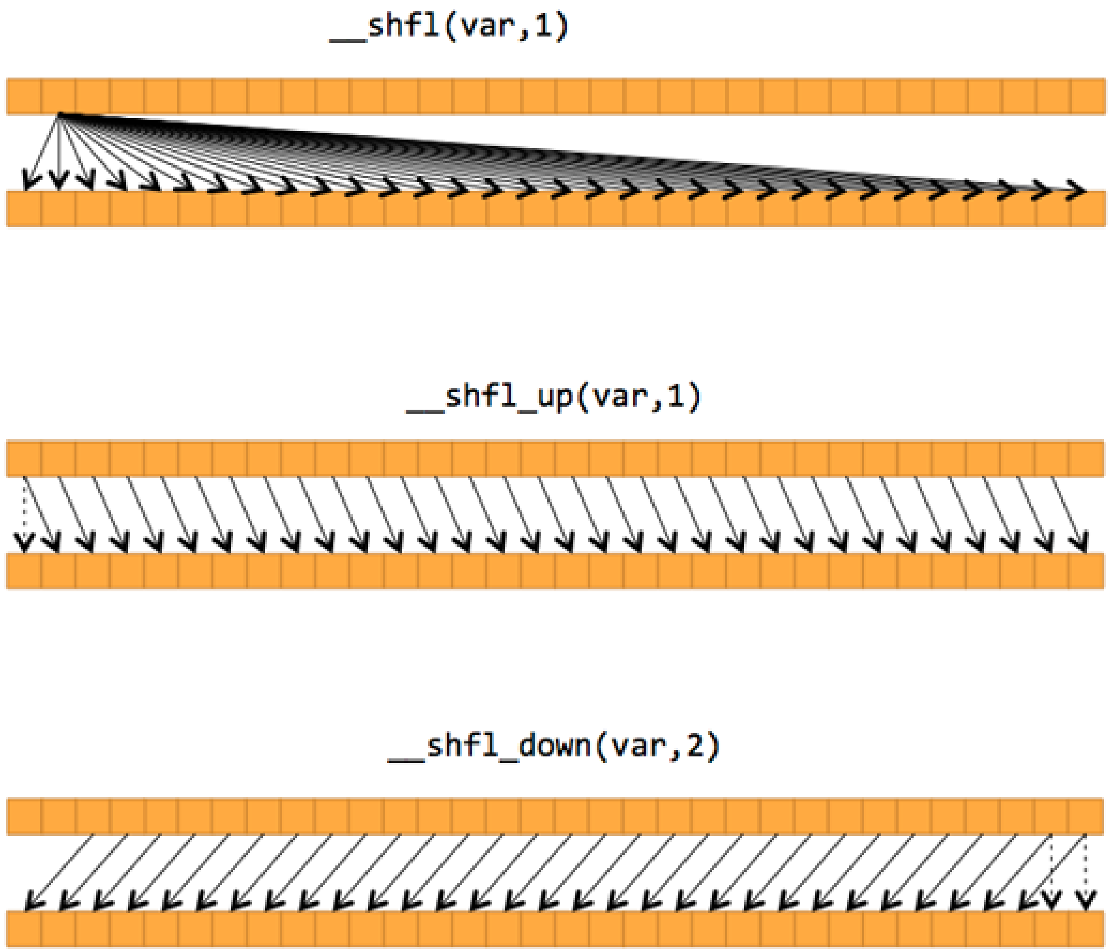
Alberto Cano and Sebastián Ventura
Knowledge Discovery and Intelligent Systems Research Group
University of Córdoba, Spain

Alberto Cano
acano@uco.es
http://www.uco.es/users/i52caroa
http://www.uco.es/grupos/kdis

GPU Parallel SubTree Interpreter for GP



GPU Parallel SubTree Interpreter for GP



Parallelization approaches for GP evaluation

- Pittsburgh style encoding [1]
  - Individuals represent variable length rule-sets
  - 3D grid of threads for individuals, rules and fitness cases
- Multi-instance classification [2]
  - Examples represent sets of instances
- Association rule mining [3]
  - Antecedent and consequent to be evaluated in paralell
  - Concurrent kernels

1) A. Cano, A. Zafra, and S. Ventura. Parallel evaluation of Pittsburgh rule-based classifiers on GPUs. Neurocomputing, vol. 126, pages 45-57, 2014.

2) A. Cano, A. Zafra, and S. Ventura. Speeding up multiple instance learning classification rules on GPUs. Knowledge and Information Systems, In press, 2014.

3) A. Cano, A. Zafra, and S. Ventura. Parallel evaluation of Pittsburgh rule-based classifiers on GPUs. Neurocomputing, vol. 126, pages 45-57, 2014.