Parallel Random-Art Hash Visualization

Yuxuan Zheng

Problem Definition

Random Art is a smart solution based on context-free grammar that makes hash comparison easier for human beings.

Problem Definition

Random Art is a smart solution based on context-free grammar that makes hash comparison easier for human beings.

Example: “`f39dc904f9ccb1be0669481c`” (hash string, serves as seed) → 

Problem Definition

Random Art is a smart solution based on context-free grammar that makes hash comparison easier for human beings.

Example: "`f39dc904f9ccb1be0669481c`" (hash string, serves as seed) → 

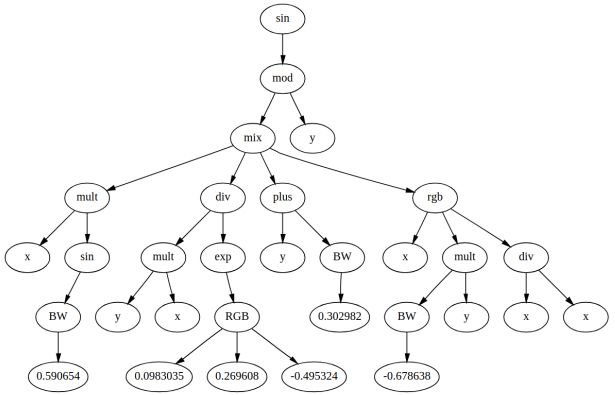
A grammar generates random *expression trees*. A random art is rendered by passing the coordinate (x, y) of each pixel into the expression tree of each colour channel (RGB).

Problem Definition

Random Art is a smart solution based on context-free grammar that makes hash comparison easier for human beings.

Example: "`f39dc904f9ccb1be0669481c`" (hash string, serves as seed) → 

A grammar generates random *expression trees*. A random art is rendered by passing the coordinate (x, y) of each pixel into the expression tree of each colour channel (RGB).



Problem Definition

Random Art is a smart solution based on context-free grammar that makes hash comparison easier for human beings.

Example: "`f39dc904f9ccb1be0669481c`" (hash string, serves as seed) → 

A grammar generates random *expression trees*. A random art is rendered by passing the coordinate (x, y) of each pixel into the expression tree of each colour channel (RGB).

The tree has to be complex enough to achieve the pre-image resistant property of a hash function. However, the computation work grows exponentially with respect to the depth of tree. We need a way to make this faster.

Literature Survey

- **Hash Visualization: a New Technique to improve Real-World Security**
  - Propose *random art* as the hash visualisation method
- **OpenMP manual**
  - Use `task` for recursive tree traversal
- **A Practical Tree Contraction Algorithm for Parallel Skeletons on Trees of Unbounded Degree**
  - Proposes *Rake-Shunt* tree contraction algorithm
  - Compress a tree iteratively by removing leaves and merging nodes, e.g, combining operations like $f(g(x))$ into $(f \circ g)x$
  - However, merging nodes involves creating partial functions, which is easy in a functional language but tricky in C

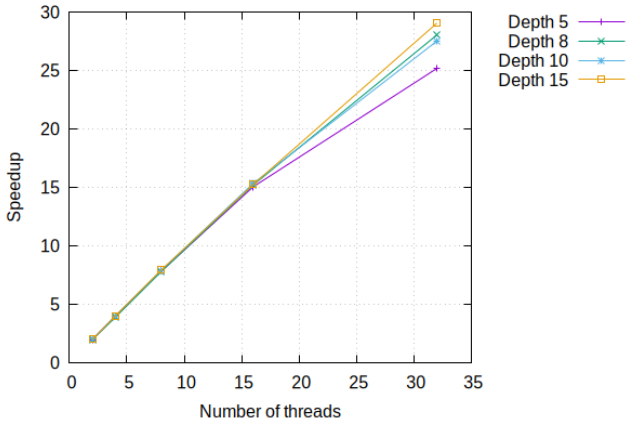
Experimental Setup

Two types of parallelisation:

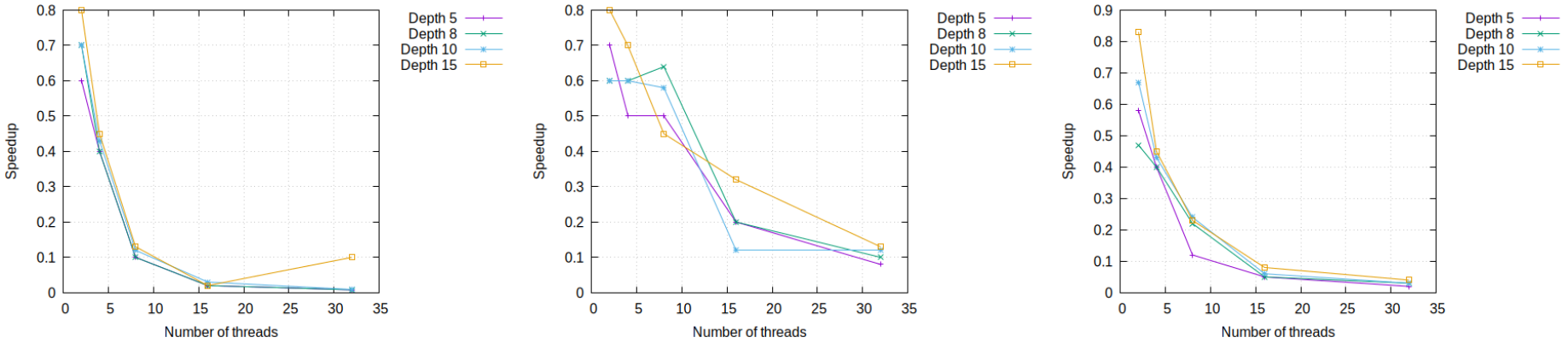
- Pixel-rendering level parallelism
  - Trees of depth 5, 8, 10, and 15
  - Thread number 1, 2, 4, 8, 16, 32
- Expression-tree-evaluation level parallelism
  - Trees of depth 5, 8, 10, and 15
  - Thread number 1, 2, 4, 8, 16, 32
  - Three strategies to assign tasks:
    - Assign tasks in every recursion if `depth < THRESHOLD`
    - Assign tasks only if `depth == THRESHOLD`
    - Assign tasks only if `depth == THRESHOLD`, and in each recursion, instead of doing `depth++`, do `depth = (depth + 1) % (THRESHOLD + 1)`

Results & Analysis

Pixel-rendering level parallelism: Embarassingly parallel



Tree-evaluation level parallelism: Worse performance as the number of threads increases



Plots of strategy 1, 2, 3, respectively, with `THRESHOLD = 4`

Result & Analysis

Experiment: Does the average arity of the functions in the grammar affect the speedup for tree-evaluation level parallelism?

$$P \to (C, C, C)$$

$$A \to X^{0.333} \mid Y^{0.333} \mid \text{RAND}()^{0.333}$$

$$B \to \text{EIGHT\_SUM}(A, A, A, A, A, A, A, A)^{1}$$

$$C \to \text{EIGHT\_SUM}(A, A, D, E, A, B, D, E)^{1}$$

$$D \to \text{ADD}(B, A)^{0.25} \mid \text{MIX}(C, D, E)^{0.75}$$

$$E \to \text{ADD}(A, A)^{0.5} \mid \text{MULT}(B, B)^{0.25} \mid C^{0.25}$$

Average arity	Speedup
1.2	0.42
1.8	0.48
2.2	0.63
2.8	0.83
3.2	0.85

Relationship between average arity and speedup

Although the speedup is still less than 1, we can see there is a performance gain as the average arity increases. When the arity increases, a node has more children, and more tasks can be performed in parallel.

Conclusions

- Parallelisation at the pixel-rendering level is effective
- Parallelisation at expression-tree-evaluation level presents limitations due to task synchronization overhead
- Increasing tree depth or function arity reaches better performance for expression-tree-evaluation level parallelism