Sparse Adapton by Kyle Headley

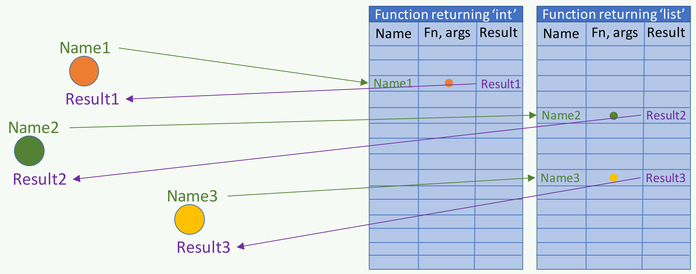
Optimized Incremental Computation for Complex Algorithms

Adapton Incremental Computation

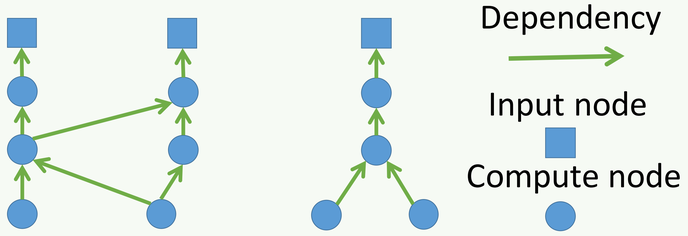
Engine by Matthew Hammer et. All (PLDI 2014)

*A computation is **incremental** if repeating it with a changed input is faster than from-scratch recomputation.*

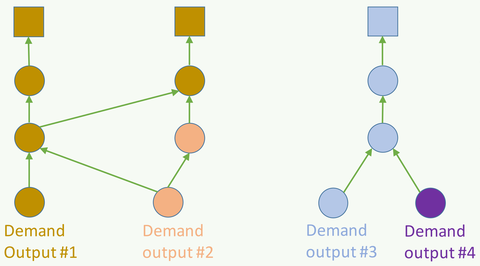
Adapton memoizes function results



Adapton tracks dependency info for function calls

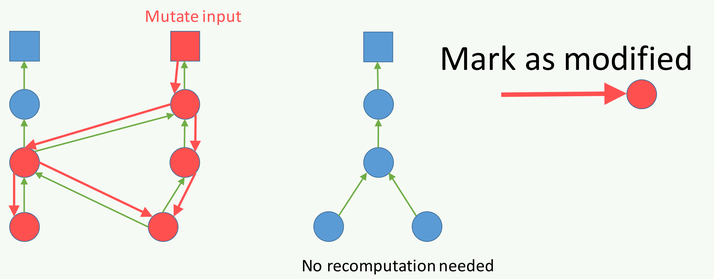


Adapton evaluates nodes on demand

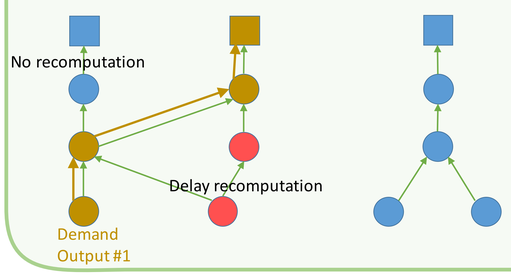


Each color marks all computations done to produce an output based on this demand order

Adapton marks all dependent nodes during a mutation

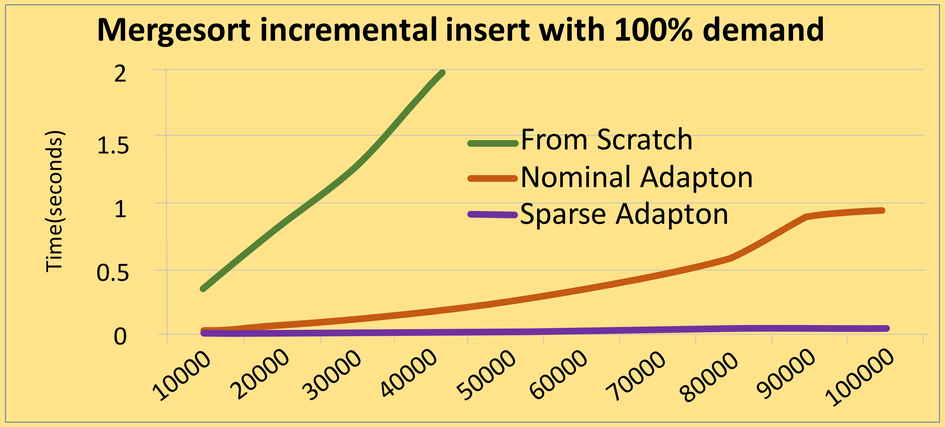


Adapton only repairs demanded nodes



Great Results

Problem: Incremental computation exploits local dependencies between function calls, but does so with a lot of overhead. We can improve performance by balancing these factors.



Faster Computation

- Fewer names
- Less memory
- Less overhead

Enhanced Methods

- Use of first class names
- Associate names with data
- Maintain association

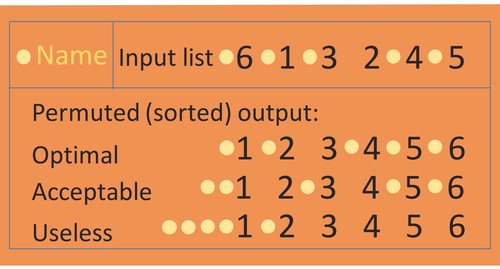
Going forward

- Increasingly complex algorithms
- Automation through libraries
- Interactivity of common tools

Difficulties

Troublesome forms

- Data Permutations
- Structural changes



Balance

- Overhead reduces speed
- Overhead improves incrementality
- Incrementality increases speed

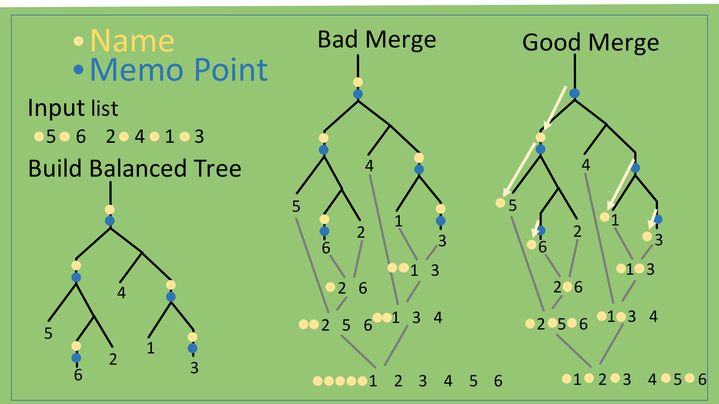
Implementation

Mergesort

1. Take input list
2. Build tree
3. Merge branches

Requirements

- Names in leaves
- MPs in branches
- Good distribution



Input Name locations

1. Plan a density for names
2. Use deterministic hash
3. Get hash-codes for data
4. Hash-code is probabilistic
5. Add names uniformly

Bad Merge

- Emit encountered names
- Merge names before data

Good Merge

- Pass names through tree
- Merge name with its data