Carnegie Mellon University

V8 Adaptive Inlining

Ishan Bhargava, Ethan Chu

Background

JavaScript has a lot of opportunity for inlining, so inlining the "right things" is very important

```
const salesJson = DownloadData();
const totalRevenue =
    salesJson
    .map(json => Record.TryParseJson(json))
    .filter(maybeRecord => maybeRecord != null)
    .reduce((record, sum) => sum + record.totalPrice(), 0)
```

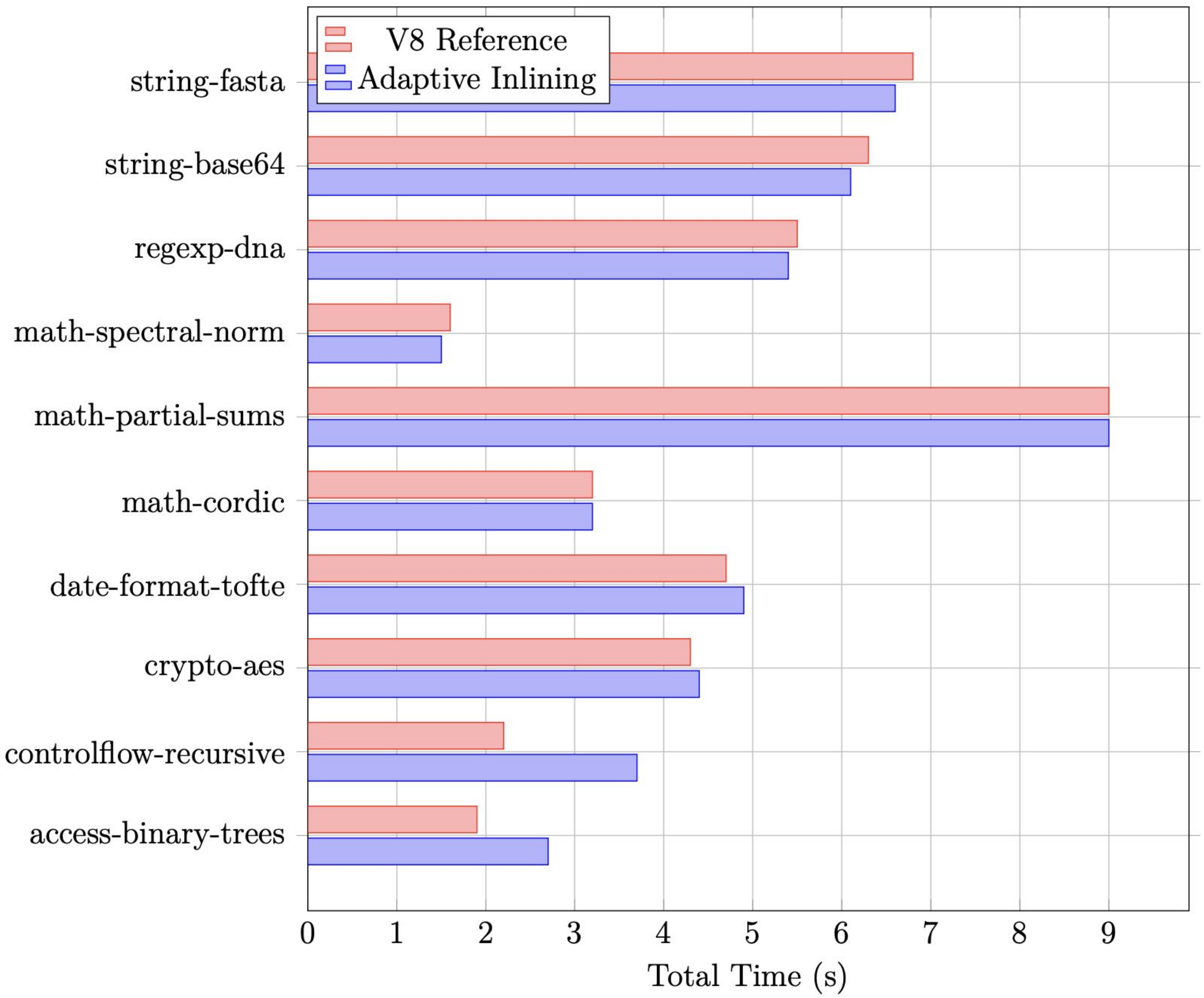
Current Approach

- V8 currently examines functions one at a time
- Has a hard cutoff for when to stop inlining
- Can be detrimental if "unlucky"
- Instead, try inlining subtrees of call tree simultaneously
- Threshold for inlining becomes higher as size increases, but never impossible

Implementation Details

- Create call tree
- Create local cost-benefit tup $(\text{Cost}, \text{Benefit}) = (\mathsf{size}(f), \mathsf{frequency}(f))$
- For each function called by parent function, check if it is better to additionally inline the child function
- Inline functions based on equatio $\mathsf{ratio}(n) \geq t_1 \cdot 2^{\frac{\mathsf{size}(n)}{16 \cdot t_2}}$
- As size(n) goes up, highly valuable functions may still be inlined

Carnegie Mellon University



Generally similar performance to existing V8 inlining heuristic

Carnegie Mellon University

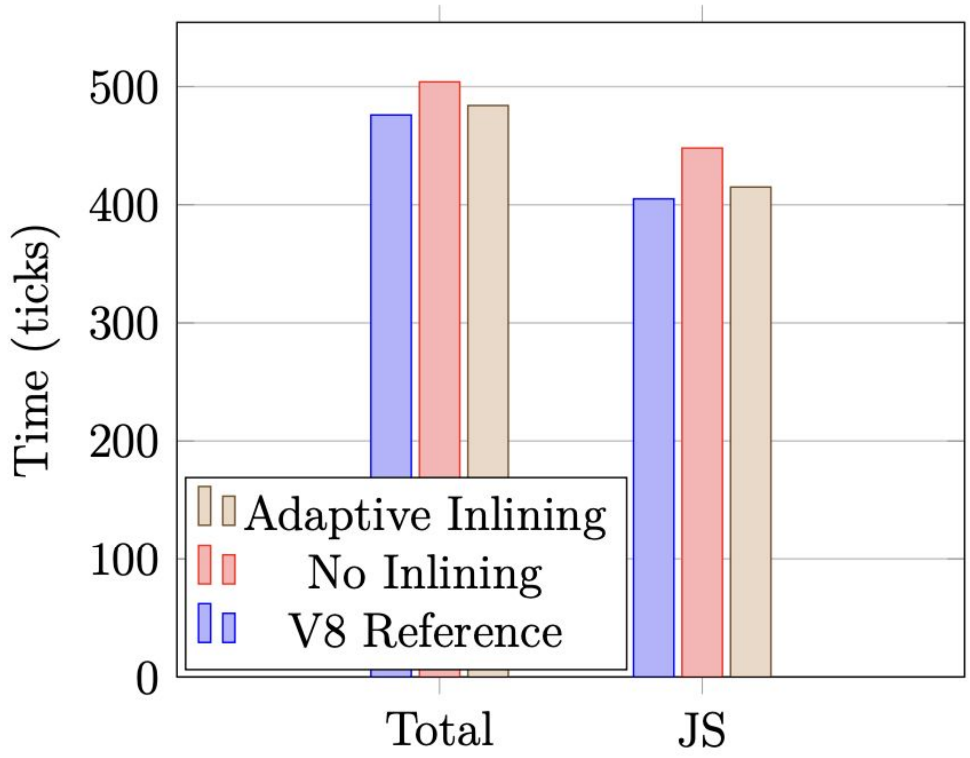


Figure 2: Linear Programming

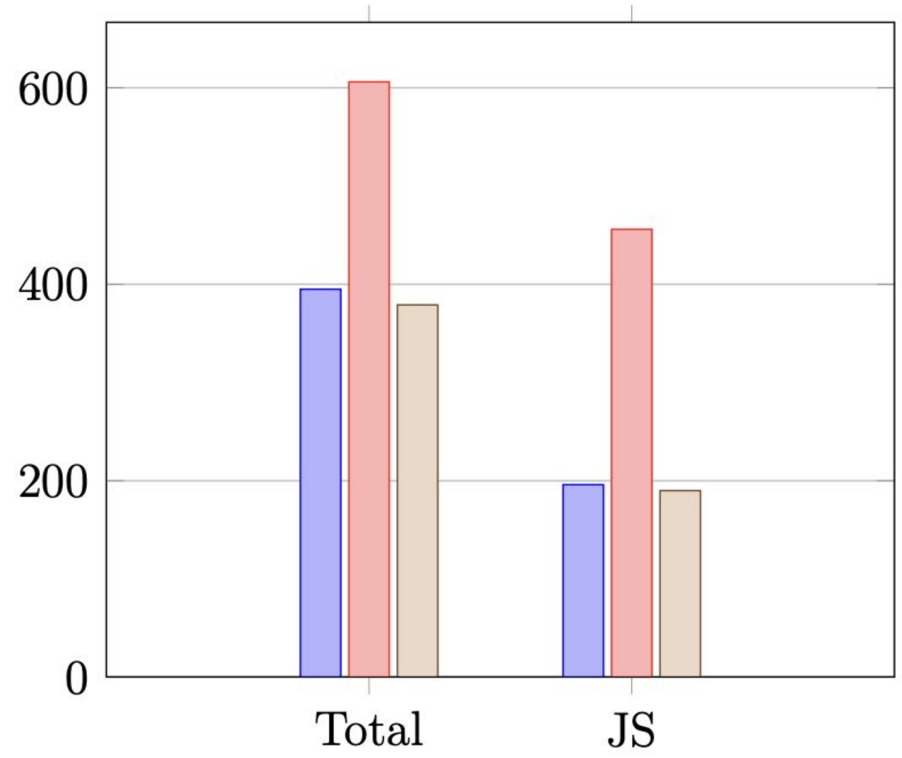


Figure 3: Fenwick Trees

Carnegie Mellon University

Conclusions

- Algorithm is fairly effective and is sometimes faster than existing work.
- May not be worth the complexity of interprocedural analysis
- Could interfere with concurrent optimization

Carnegie Mellon University