REDwood: Heterogenous Implementation of Tree Applications with Accelerated REDuctions

Yanwen Xu, Tyler Sorensen University of California, Santa Cruz

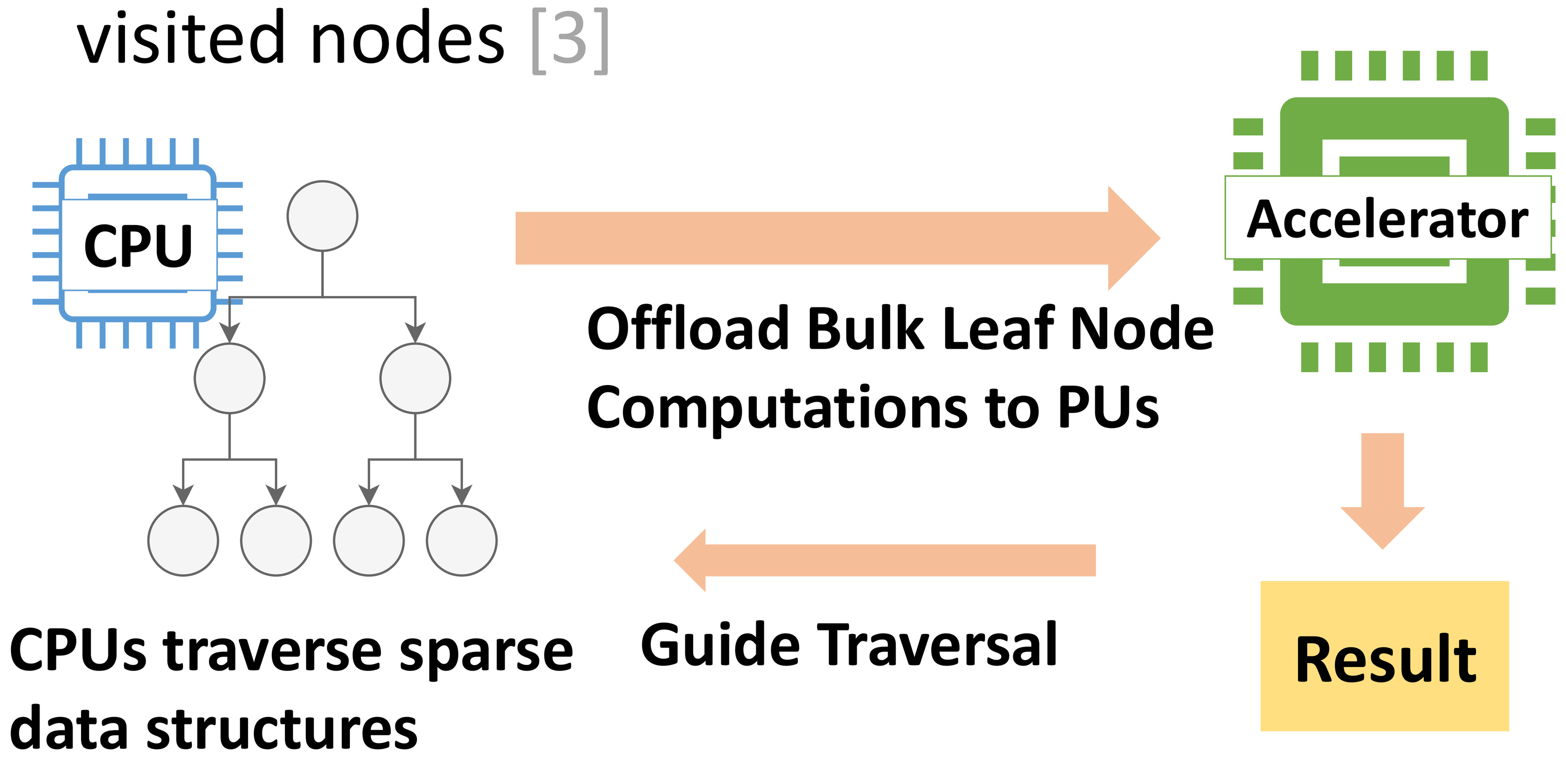
Baskin Engineering UC SANTA CRUZ

Motivation

- The end of Moore's law and Dennard's scaling has led to an explosion of specialized *processing units* (PUs) [1]
  - **GPUs** excel at massively parallel computations on dense data
  - **CPUs** with complex hardware components can tolerate memory latency
  - **Custom Circuits** like FPGAs/ASICs can perform specialized tasks efficiently
- Efficient implementations of applications must be flexibly **decomposed and executed across PUs** [2]

Traverse-Reduce Applications

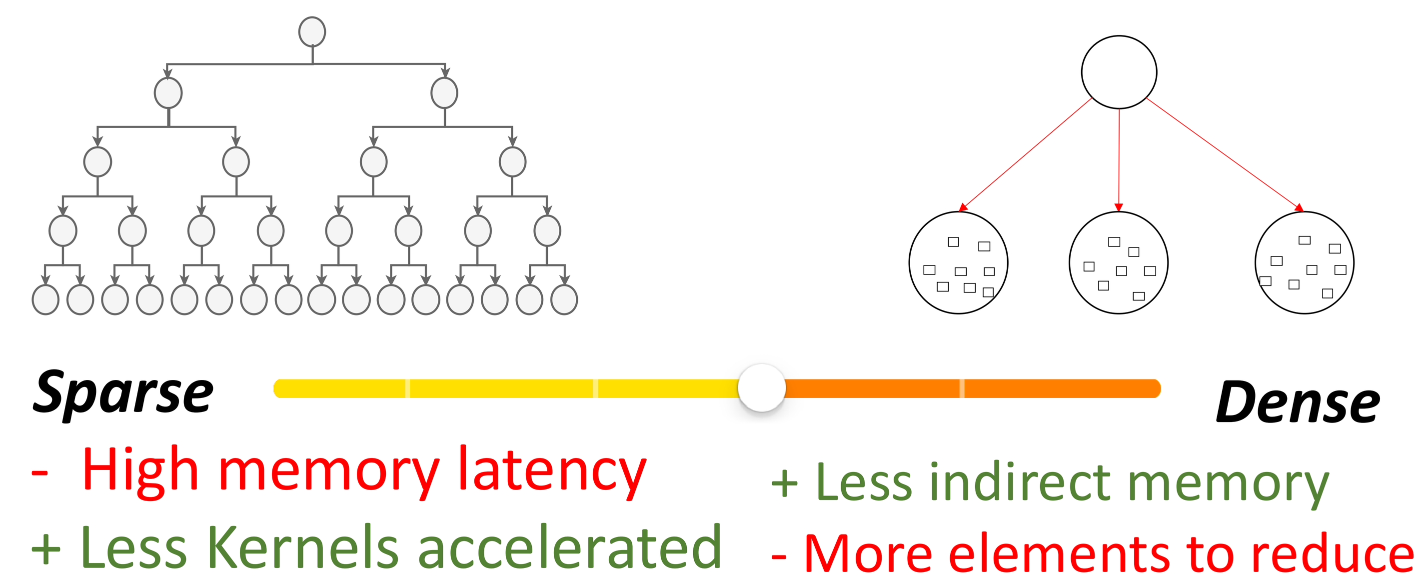
- A class of algorithms, what we call *Traverse-Reduce* algorithms, has **flexible hetero-geneous decomposition**
- These algorithms traverse a sparse tree data structure and perform reductions over the visited nodes [3]



- Such algorithms are common in: **Facial Recognition, Particle/Molecular Simulation, and Statistical Analysis**, etc.

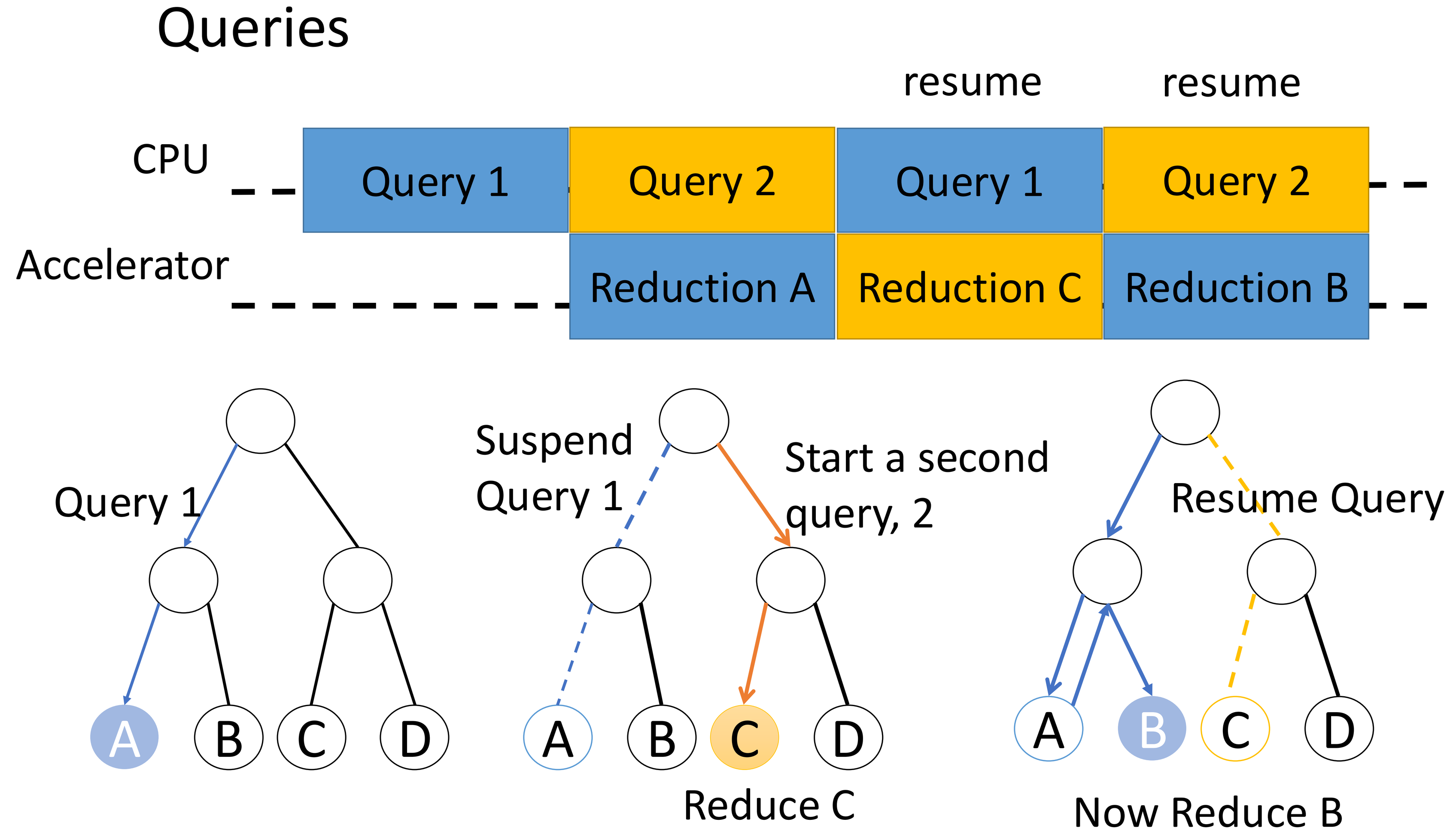
Methods

- **Flexible leaf node size** allows REDwood to adapt to various heterogeneous systems with different relative **throughput** between the CPU and the accelerator PU



- **Ping Pong buffering** enables REDwood to execute the traverse and reduction phases **in parallel** with low synchronization overhead

- **Executor Runtime**:
  - Avoid long CPU stalls
  - Can Suspend/Resume Queries
  - Light-wight coroutine
  - Handles dependency



[1] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. 2015. The aladdin approach to accelerator design and modeling. IEEE Micro 35, 3 (2015)
[2] Abdullah Gharaibeh, Tahsin Reza, Elizeu Santos-Neto, Lauro Beltrao Costa, Scott Sallinen, and Matei Ripeanu. 2013. Efficient large-scale graph processing on hybrid CPU and GPU systems. arXiv preprint arXiv:1312.3018 (2013)
[3] Nikhil Hegde, Jianqiao Liu, Kirshanthan Sundararajah, and Milind Kulkarni. 2017. Treelogy: A benchmark suite for tree traversals. In 2017 IEEE ISPASS. IEEE

Programming Model

User Algorithms:

```
def traverse(node, q):
    if is_leaf(node):

    for i in range(node.leaf_size):
            result += kernel_func(q, node.data[i])

    else:

    theta = compute_theta(q, node.data)
        if (theta < theta_threashold):
            result += kernel_func(q, node.center_of_mass())

    else:
            for i in range(8):
                traverse(node.children[i])
```

Apply REDwood:

```
for q in query_points:
    api_set_query(q)
    traverse(root, q)
```

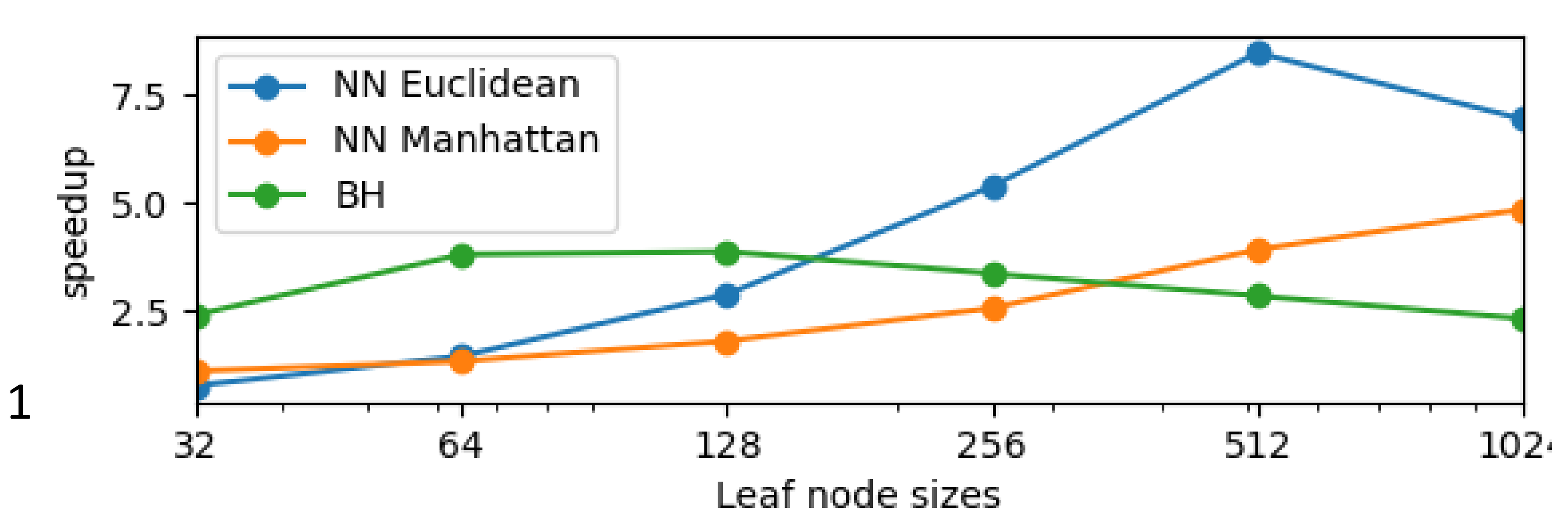
```
def traverse(node, q):
    if is_leaf(node):

        api_reduce_leaf(node.data())

    else:

        theta = compute_theta(q, node.data)
        if (theta < theta_threashold):

            api_reduce_branch(node.center_of_mass())

        else:
            for i in range(8):
                traverse(node.children[i])
```

- REDwood APIs allows users to implement traverse reduce algorithms **easily**
- Reductions will be **automatically** handled by our {**CUDA, SYCL**} backends

Results

- We implemented *Barnes-Hut*(BH), *Nearest Neighbor* (NN) with Manhattan distance, and NN with Euclidian distance. Experiments are executed on an Nvidia Jetson Nano.

- Performance with various leaf node sizes



- REDwood speedups over other baselines

	NN Manhattan	NN Euclidian
CPU Baseline	7.41x	2.5x
kNN-CUDA	5.58x	12x
SciPy	1.94x	1.1x

	BH
CPU Baseline	3.86x
GPU Baseline	6.82x