Redwood: Flexible and Portable Heterogeneous Tree Traversal Workloads

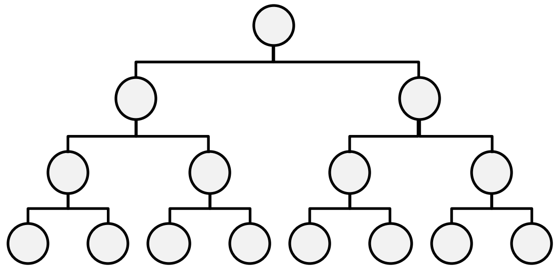
Yanwen Xu
University of California, Santa Cruz

Ang Li
Princeton University

Tyler Sorensen
University of California, Santa Cruz

Highlights

Many applications in edge computing can utilize tree data structures to accelerate their workloads



Developed a framework, ***Redwood***, for writing a special class of tree algorithms, which we call *traverse-compute*

Evaluated 5 shared memory heterogeneous systems using ***Grove***, a suite of 9 heterogeneous traverse-compute workloads

Achieved speedups up to

- 8.12x on commodity systems
- 13.53x on academic system

Insight: Traverse-compute workload has natural heterogeneous decompositions on modern shared memory system-on-chips

Motivation: Accelerating Computations at Edge

Edge computing are getting popular …

But they has **constraints**

- *e.g., energy or latency requirement*

Application of edge computing

- *Surveillance cameras*
- *Autonomous vehicles*
- *Mobile gaming*

Motivation: Accelerating Computations at Edge

Edge computing are getting popular …

But they has **constraints**

- *e.g., energy or latency requirement*

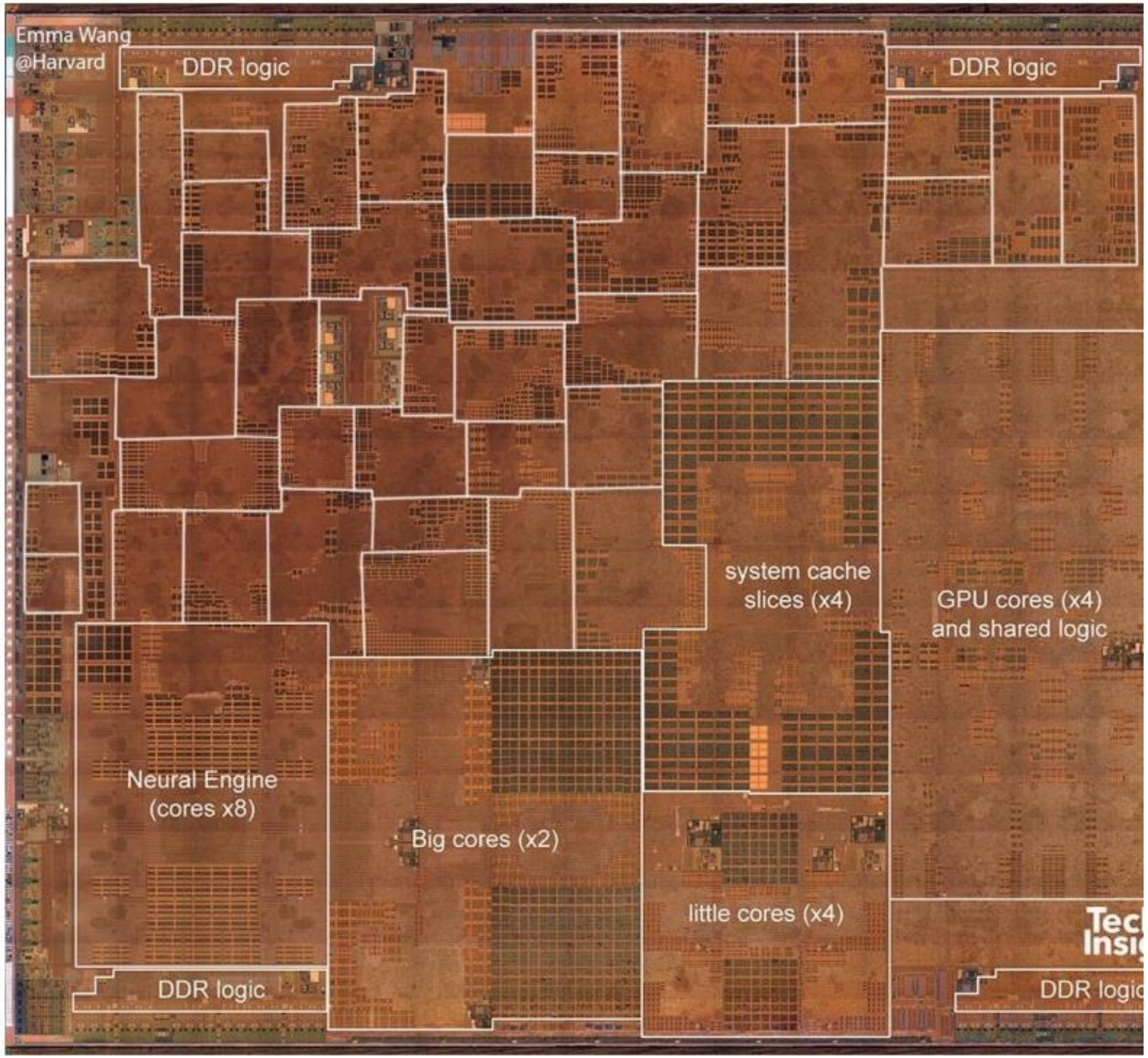
Application of edge computing

- *Surveillance cameras*
- *Autonomous vehicles*
- *Mobile gaming*

Modern edge devices are becoming increasingly heterogeneous

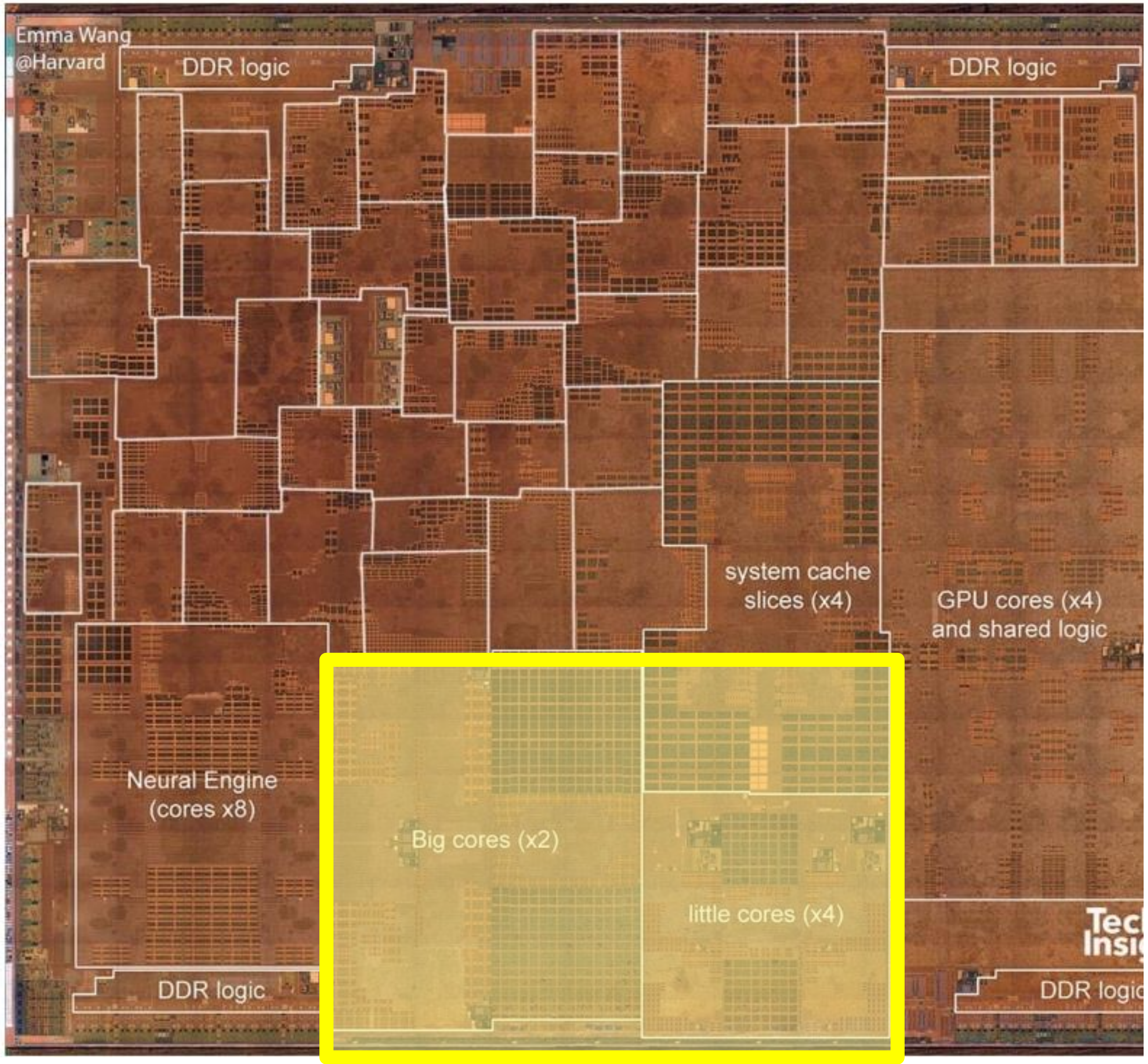
- w/ specialized *Processing Units* (PUs)

We need to efficiently utilize these available system resources



From David Brooks lab at Harvard:
https://vlsiarch.eecs.harvard.edu/research/accelerators/die-photo-analysis

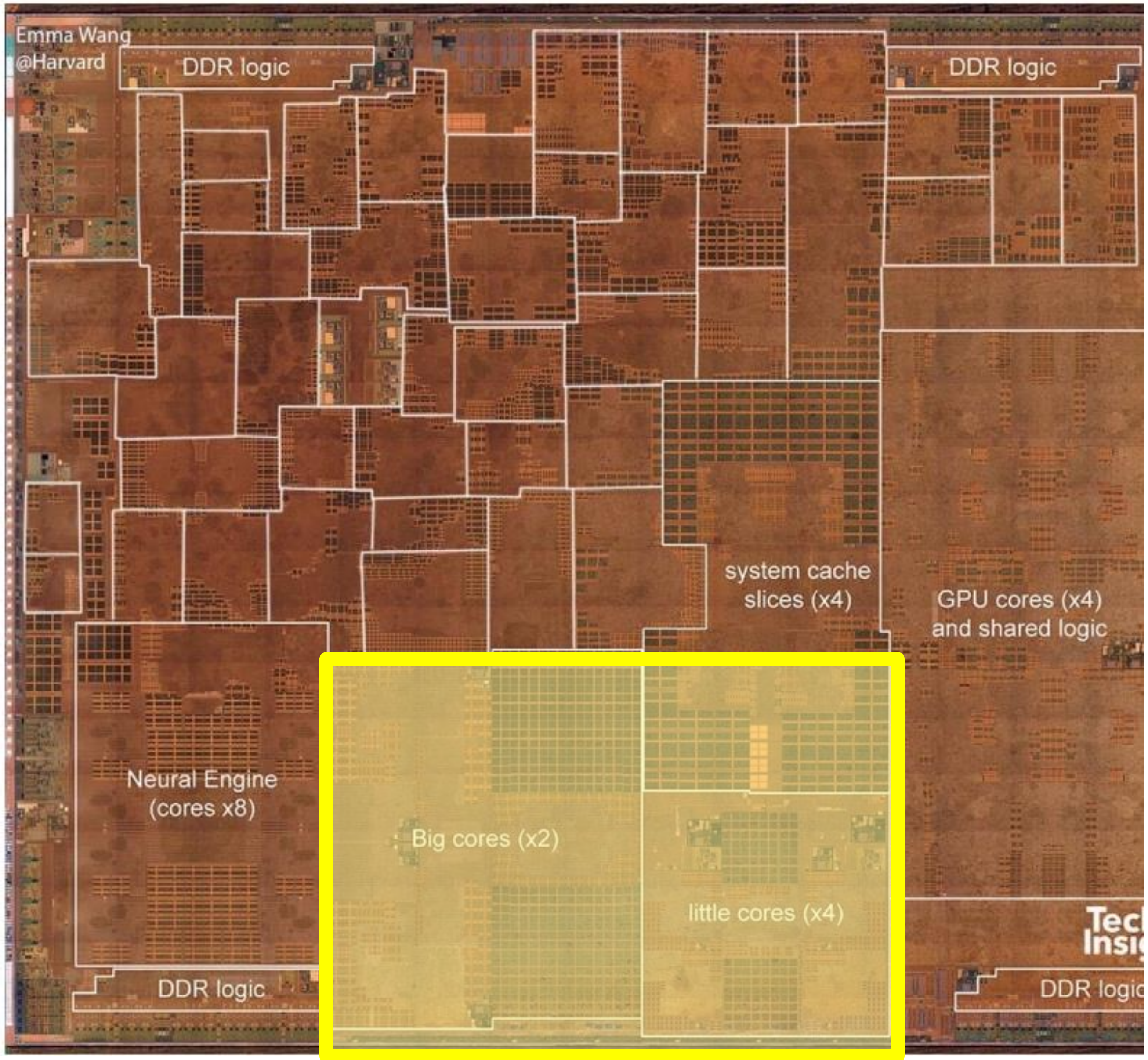
What do we mean by resources?



From David Brooks lab at Harvard:
https://vlsiarch.eecs.harvard.edu/research/accelerators/die-photo-analysis

What do we mean by resources?

- E.g., less than 20% of the die area of an iPhone contains the CPU
- The rest contains specialize *Programmable Accelerating PUs* (PAPU)
  - e.g., integrated GPUs, FPGAs
  - Interconnected to a shared memory hierarchy
- *Shared Memory Heterogeneous System* (SMHS) enables efficient communication between PUs



From David Brooks lab at Harvard:
https://vlsiarch.eecs.harvard.edu/research/accelerators/die-pho

What do we mean by resources?

- E.g., less than 20% of the die area of an iPhone contains the CPU
- The rest contains specialize *Programmable Accelerating PUs* (PAPU)
  - e.g., integrated GPUs, FPGAs
  - Interconnected to a shared memory hierarchy
- *Shared Memory Heterogeneous System* (SMHS) enables efficient communication between PUs

How does workloads utilize each PU?

Processing Units (PU) Characteristics

CPU

Features: High-performance cores, reorder buffer, load store queue, …

\+ **Latency optimized**

\- **Limited throughput**

Good for **irregular** programs

Programmable Accelerating PUs (PAPU)

GPU

Features: SIMT (*Single Instruction, Multiple Threads*) execution, coalesced memory access

\+ **Throughput optimized**

\- **Warp Divergence**

FPGA

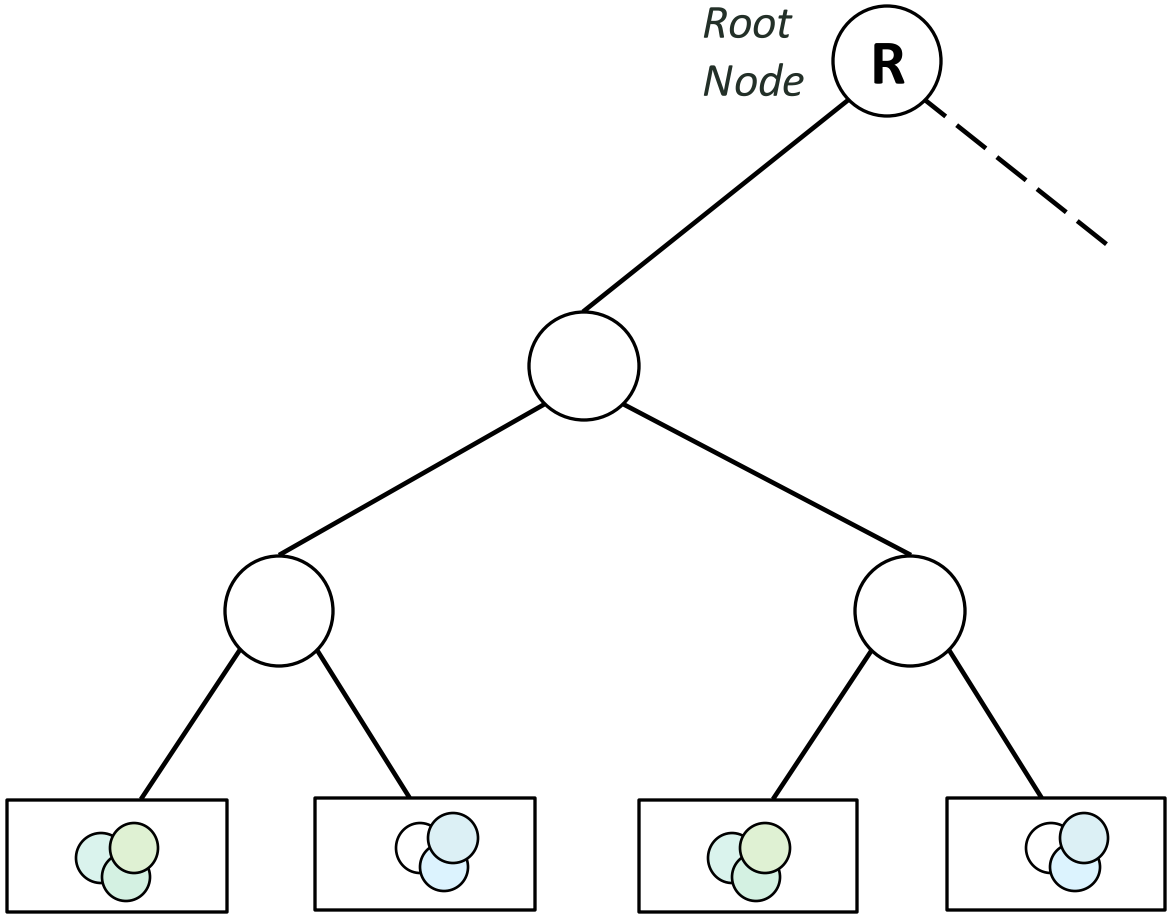
Features: Specialized tasks, Pipeline parallelism

\+ **Close to ASIC performance**

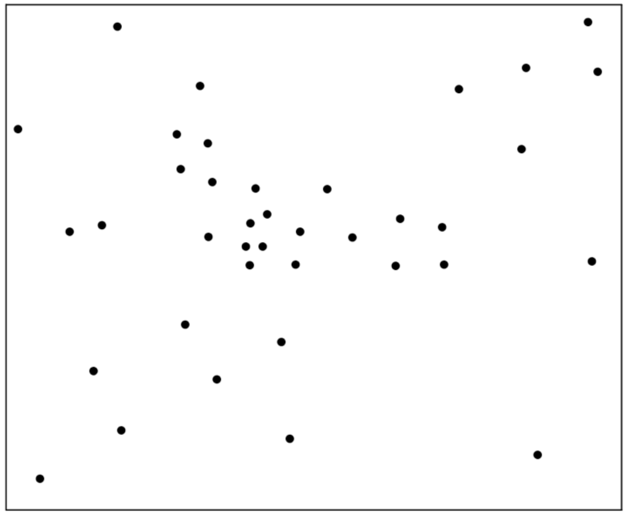
\- **Orders-of-magnitude harder to program**

Good for accelerating **compute-intense** programs

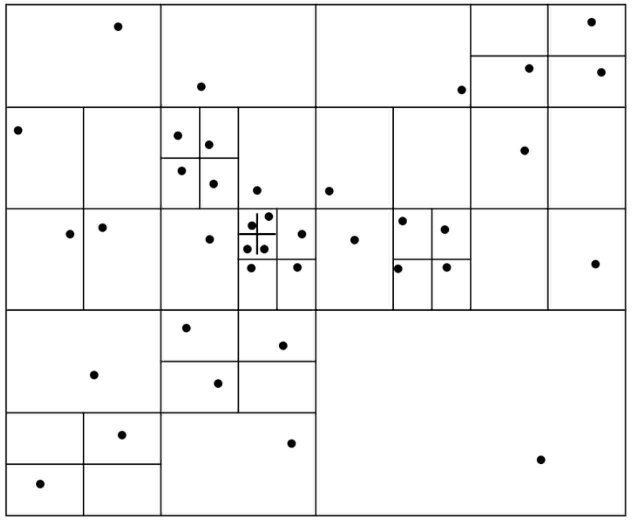
Trees on the edge



- Edge applications process a large amount of data
- They can utilize tree structures and traversals to perform edge tasks
  - E.g., *in classifications and security*

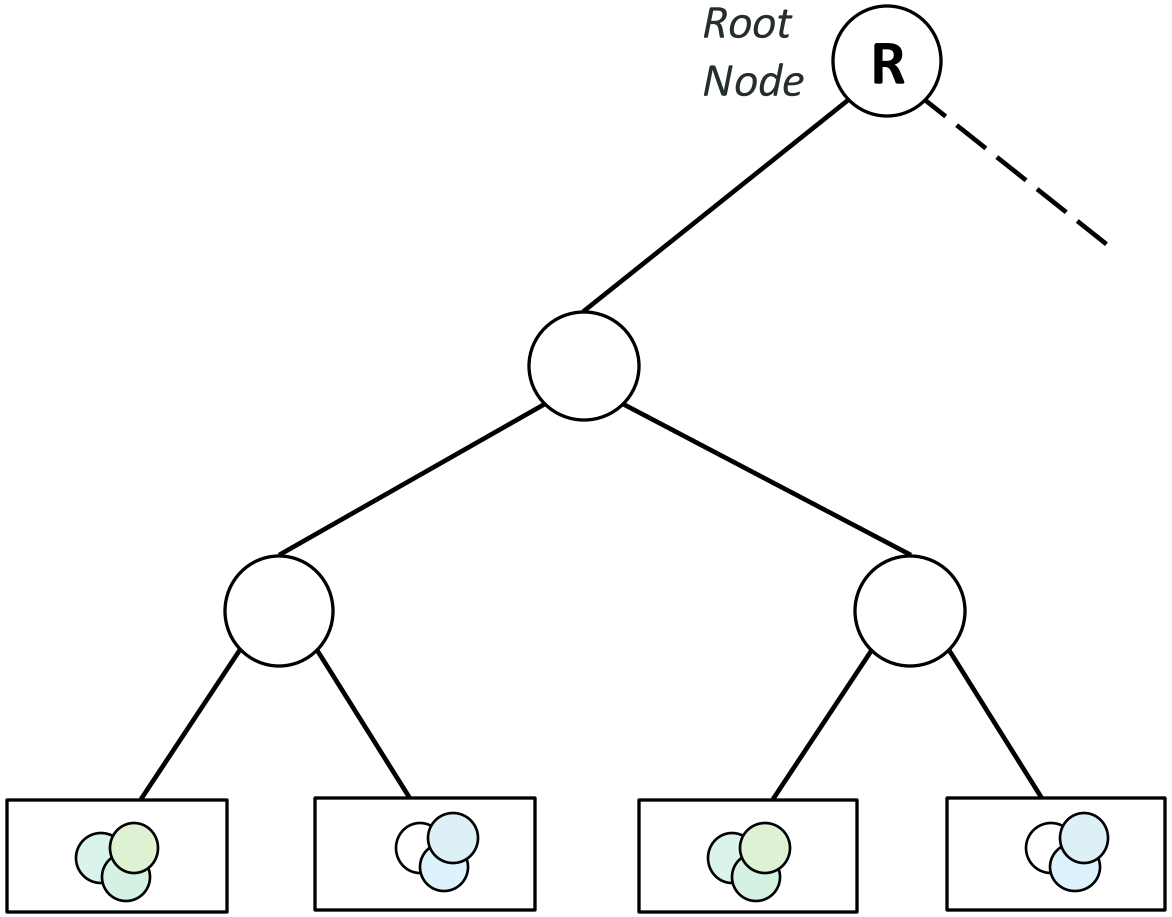


Input data

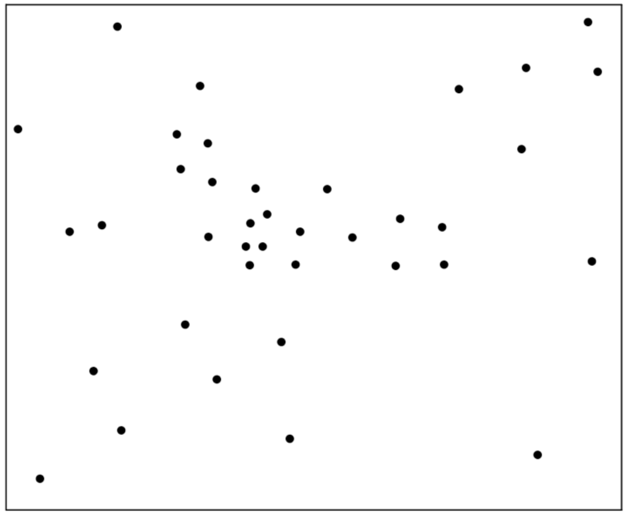


Spatial Partition

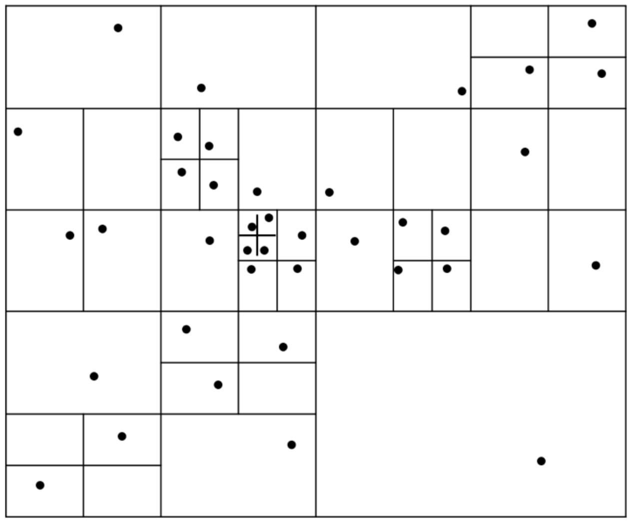
Trees on the edge



- Edge applications process a large amount of data
- They can utilize tree structures and traversals to perform edge tasks
  - E.g., *in classifications and security*
- The dataset are organized into a hierarchical tree structure, allowing data to be **efficiently** searched from $O(n)$ to $O(\log n)$

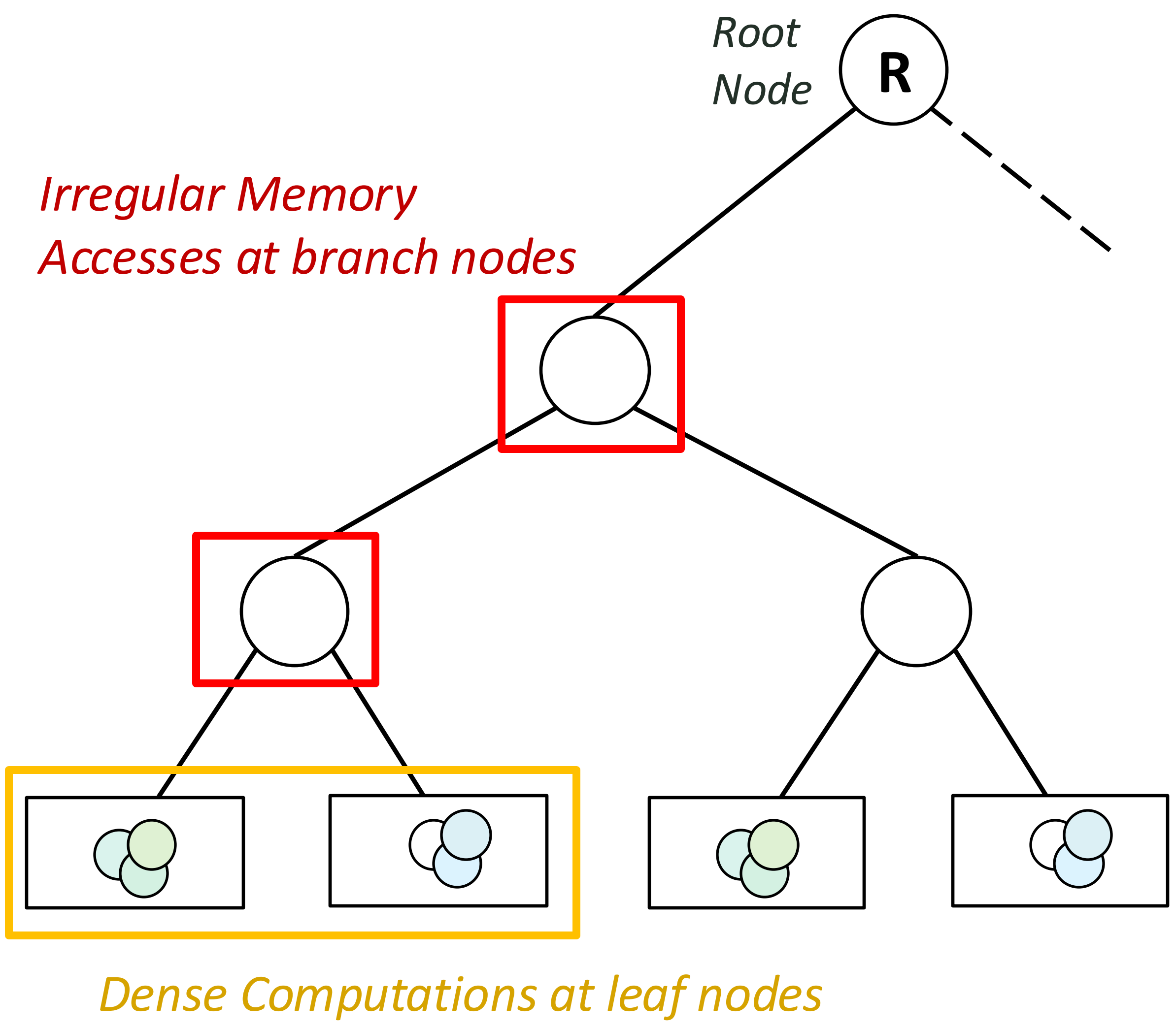


Input data



Spatial Partition

Traverse-Compute Workloads



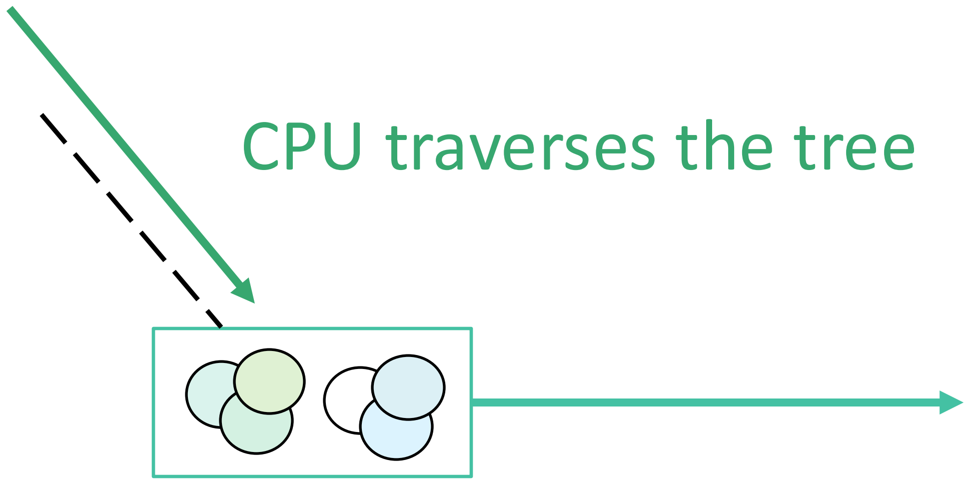
- Repeatedly traversing a sparse tree structure
- Each traversal consists of
  - Indirect memory loads at branch nodes (Red box)
  - Dense data to be processed at leaf nodes visited (**Orange box**)
    - Computing pairwise interactions (e.g., Euclidean distance)
    - Reductions (e.g., sum, min)
- Example workloads:
  - Barnes-hut Algorithm (octree)
  - Nearest Neighbor Search (k-dimensional tree)
  - Ray Tracing (bounding volume hierarchy)

Decomposing Traverse Compute Workloads

<u>CPUs</u> are good at handling dynamic control flows and tolerating indirect memory loads

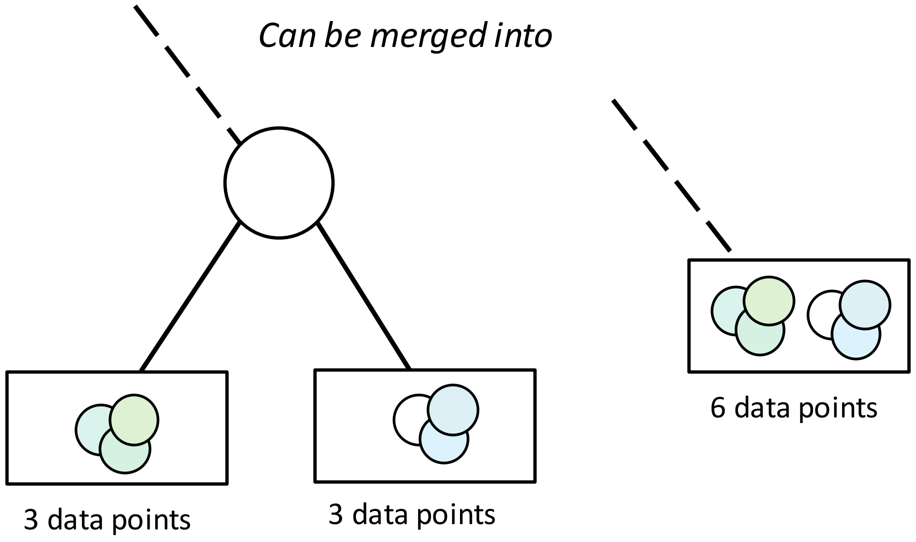
<u>PAPUs</u> are good at accelerating dense, compute-intense operations

A natural Heterogeneous Approach is to



Accelerating Traverse-compute workloads on SMHSs

- The tree can be ***parameterized*** by how many data points exist on the leaf nodes.



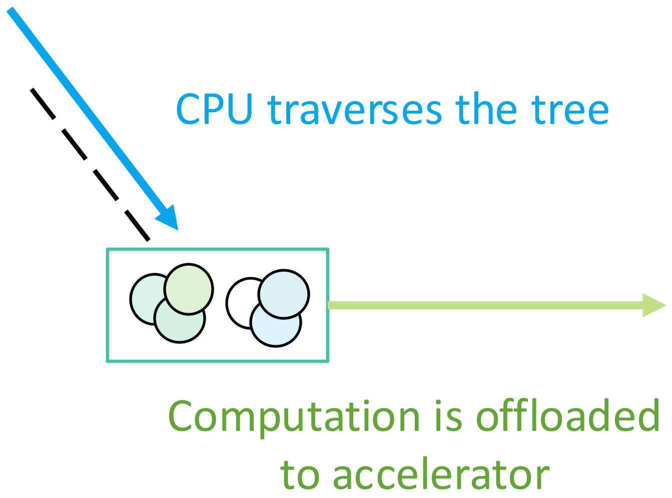
Larger node sizes tradeoffs

\+ Larger chunk of contiguous data

\+ Less irregular accesses in tree traversals

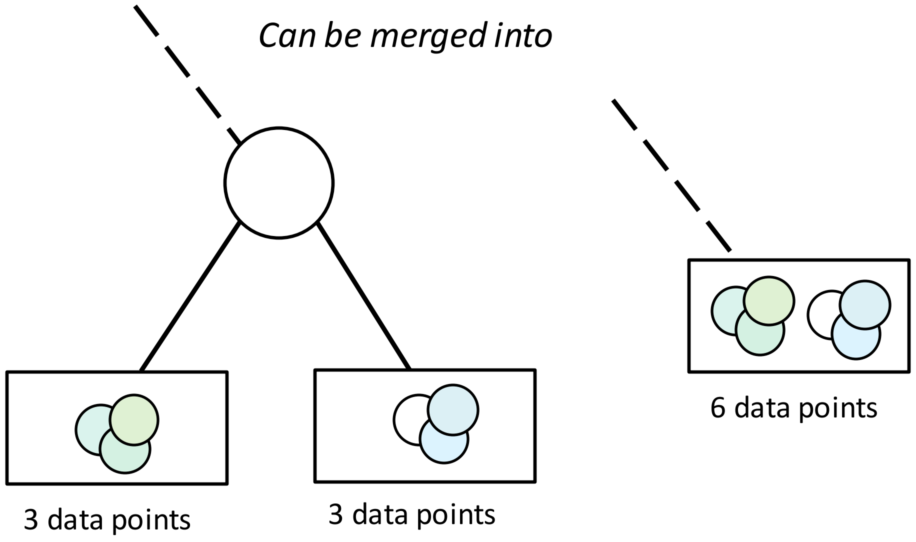
\- Potentially unneeded computation

Heterogeneous Approach



Accelerating Traverse-compute workloads on SMHSs

- The tree can be ***parameterized*** by how many data points exist on the leaf nodes.



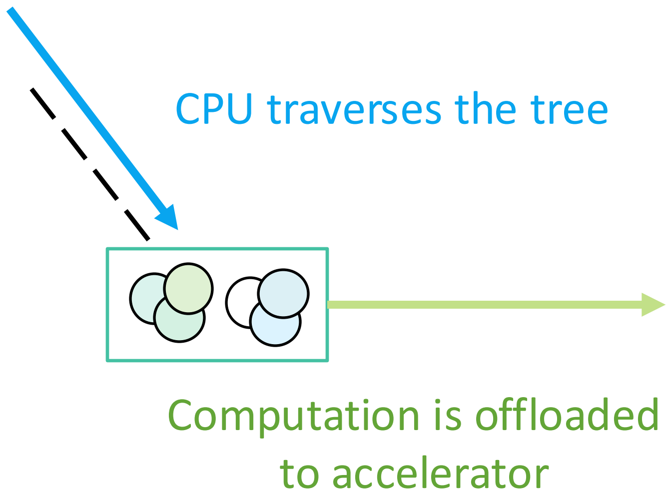
Larger node sizes tradeoffs

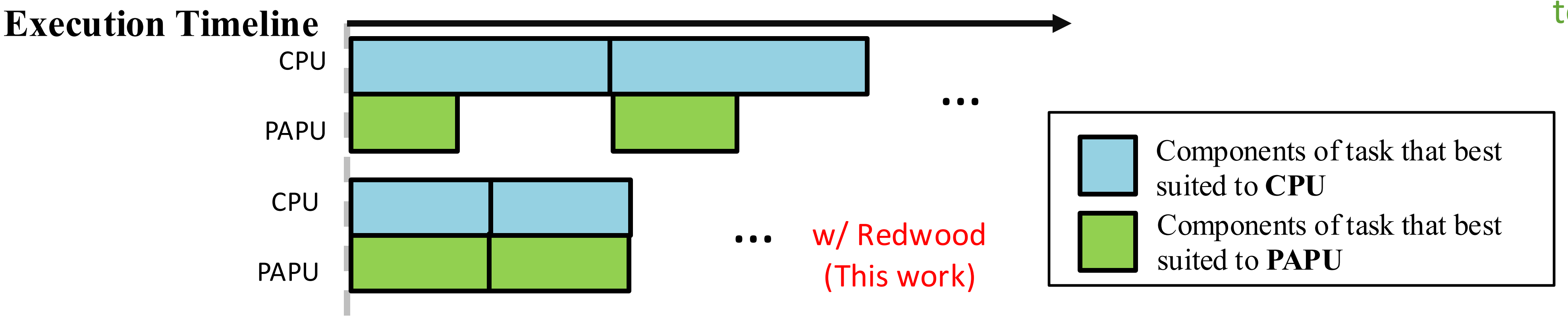
+ Larger chunk of contiguous data

+ Less irregular accesses in tree traversals

- Potentially unneeded computation

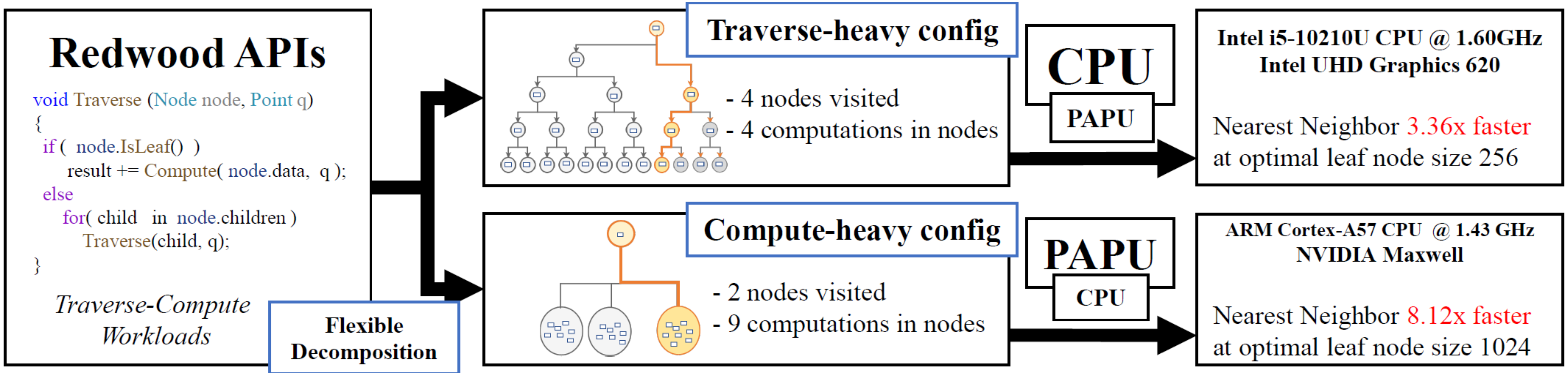
Heterogeneous Approach





Flexible & Specialize Heterogeneous Decomposition

This Work: Redwood Overview



Users implement tree applications using our APIs

Target systems w/ different CPU/PAPU throughputs

Target CUDA, SYCL, and HLS backends

KNN based Facial recognition

Intel SoCs

Nvidia SoCs

NVIDIA CUDA SYCL

Redwood: APIs and Data Structures

CPU Sequential Code (NN)

```
tree = KDTree()
min_dist = 99999.9999
def traverse(node, q):
    if is_leaf(node):

        # Reduce Leaf Node
        for i in range(node.leaf_size):
            kernel_func(q, node.data[i])

    else:
        dist = compute_dist(q, node.data[0])
        min_dist = min(min_dist, dist)
        traverse(node.leaf_child)
        if check_other_side(dist):
            traverse(node.right_child)
```

Implemented using:

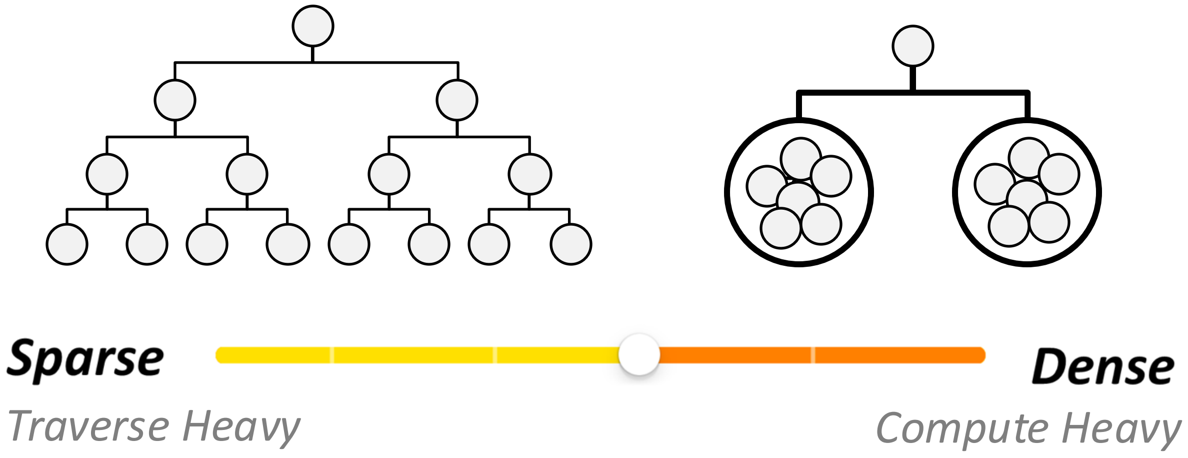
w/ Redwood API

```
tree = KDTree(leaf_size=32)

redwood_set_query(q)

def traverse(node, q):

    if is_leaf(node):
        redwood_compute_leaf(node.data())

    else:
        dist = compute_dist(q, node.data[0])
        min_dist = min(min_dist, dist)
        traverse(node.leaf_child)
        if check_other_side(dist):
            traverse(node.right_child)
```

Redwood Heterogenous Optimizations

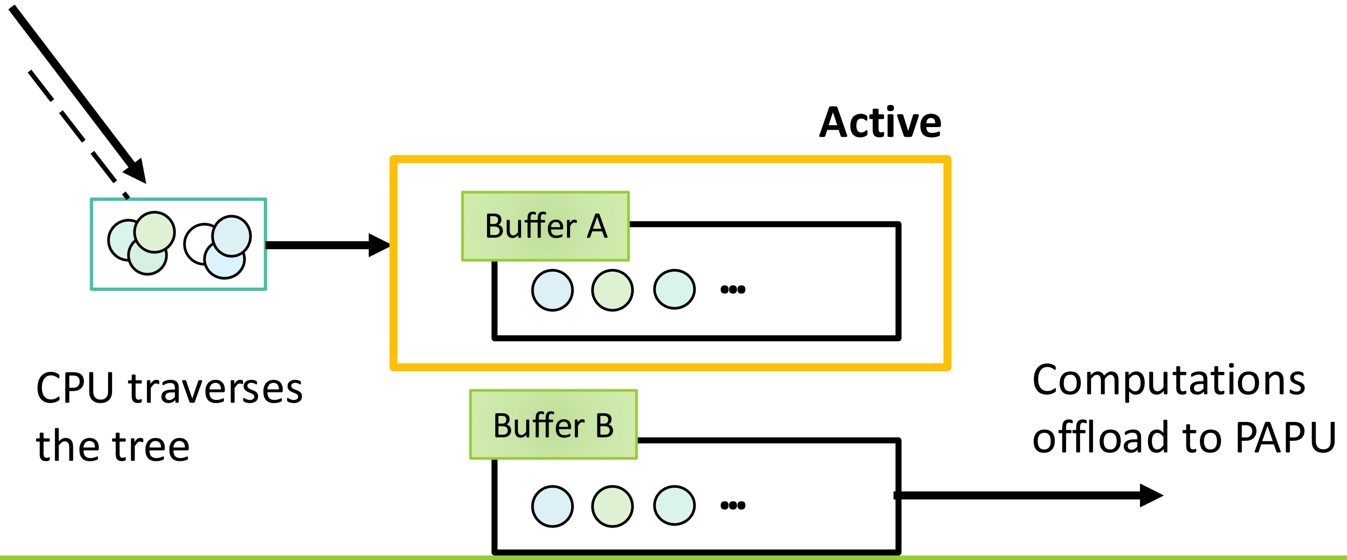
Flexible Leaf Size Configuration

- Adapt to various heterogeneous systems with different relative throughput between the CPU and the PAPU



Automatic Batching & Ping-pong Buffering

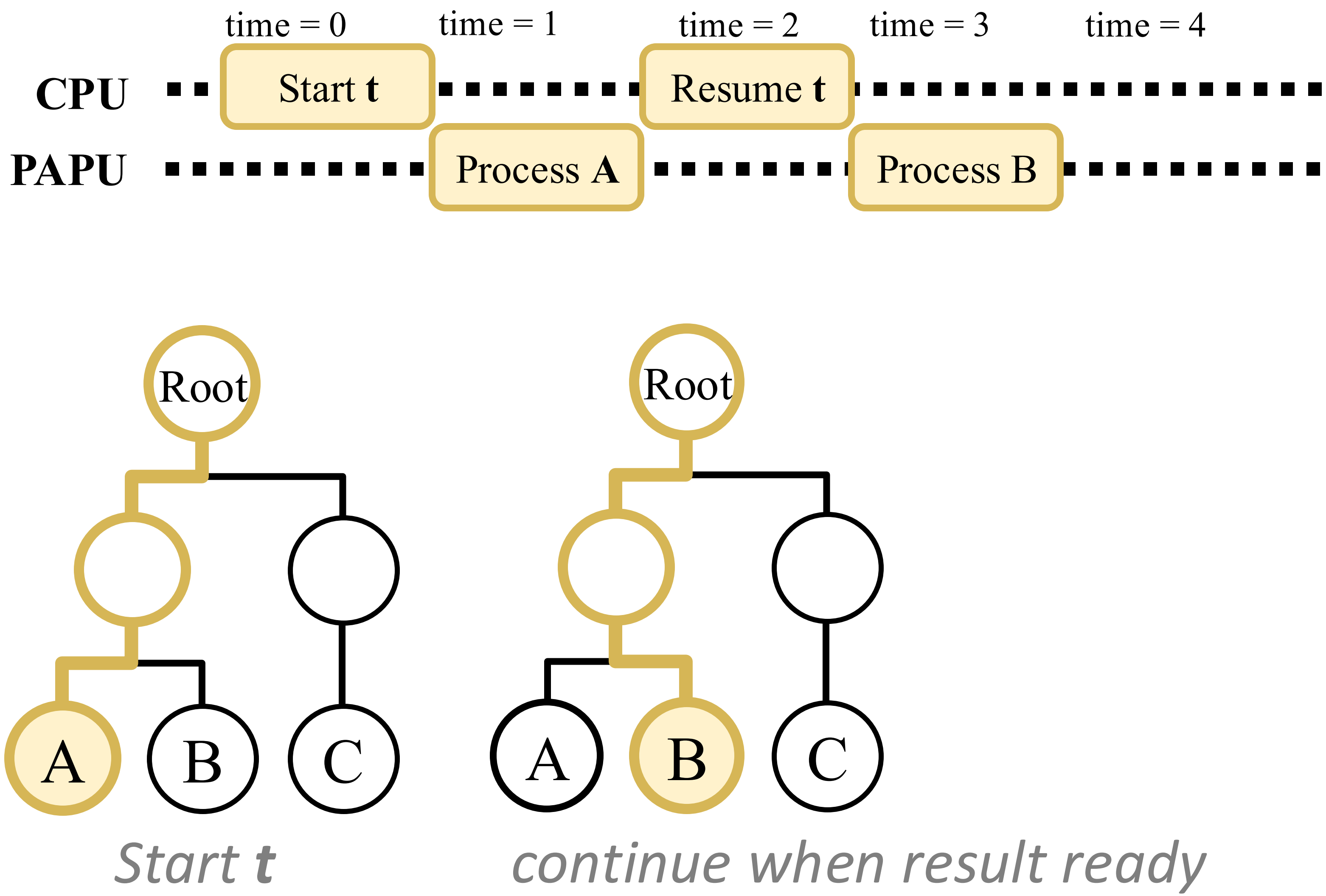
- Fuse multiple small computations to a larger GPU kernel
- Overcome overheads related to GPU kernel launching & synchronization



Redwood Heterogenous Optimizations

Traverser Runtime

- Allow a single CPU thread to execute many traversals concurrently to avoid stalling when a traversal depends on a PAPU accelerated value

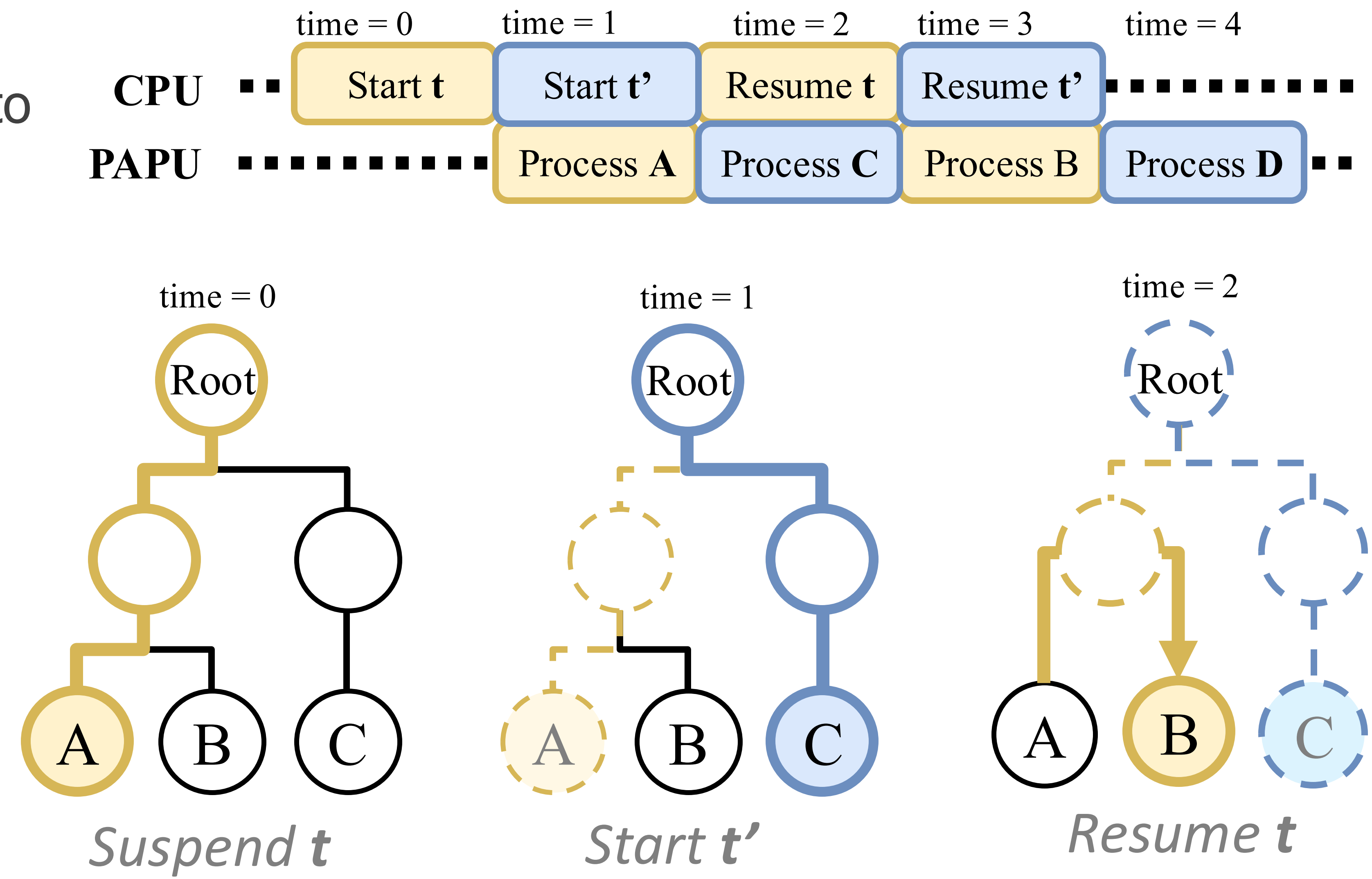


Redwood Heterogenous Optimizations

Traverser Runtime

- Allow a single CPU thread to execute many traversals concurrently to avoid stalling when a traversal depends on a PAPU accelerated value

- *Light weight Coroutine*
  - ***Suspend***
  - ***Resume***



Grove Benchmark Suite for SMHS

Grove contains **9** traverse-compute workloads

Can be found in many applications
- *Facial recognition*
- *Anomaly detection*
- *Outlier detection*
- *Particle simulation*

<u>Tree Structures</u>
- Octree/quadtree
- k-d tree

<u>Three Algorithms</u>
- Barnes Hut
- Nearest Neighbor
- k Nearest Neighbor

<u>Computation Patterns</u>
- Aggregation *(sum)*
- Reduction *(e.g., min)*
- Sorting

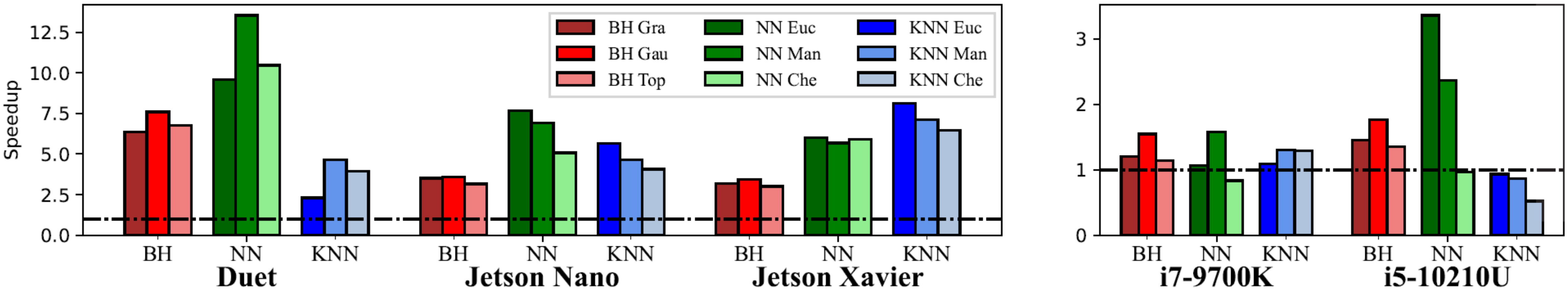
<u>Various Distance Metrics</u>
- Euclidean
- Manhattan
- Chebyshev

Platforms Evaluated

	Platform	Backend	CPU	CPU Frequency	Accelerator	Accelerator frequency
Powerful NVIDIA GPUs + Little ARM CPUs	**NVIDIA Jetson Nano**	CUDA	ARM Cortex-A57	1.43 GHz	128-core Maxwell	921 Mhz
	NVIDIA Jetson Xavier NX	CUDA	Carmel ARMv8.2	1.5 GHz	384-core Volta	1.21 GHz
Little Intel GPUs + Powerful Intel CPUs	**Intel i7-9700K**	SYCL	i7-9700K	3.60 GHz/ 4.20 GHz	Intel UHD Graphics, 24 EUs	350 MHz/ 1.20 GHz
	Intel i5-10210U	SYCL	i5-10210U	1.60 GHz/ 4.90 GHz	Intel UHD Graphics, 24 EUs	300 MHz/ 1.10 GHz
Tiny in-order CPUs w/ Efficient eFPGA integration	**Duet** [1] *(simulated in **gem5**)*	HLS	RISC-V TimingSimpleCPU	1.5 GHz	Duet eFPGA	333MHz

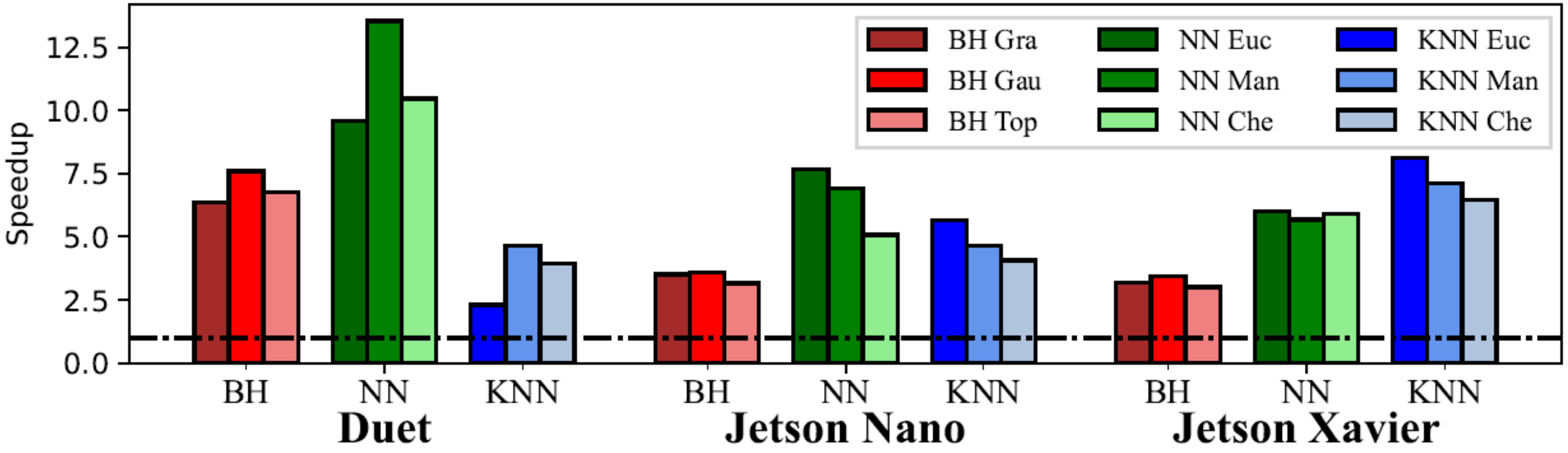
[1] Li, Ang, August Ning, and David Wentzlaff. HPCA'23

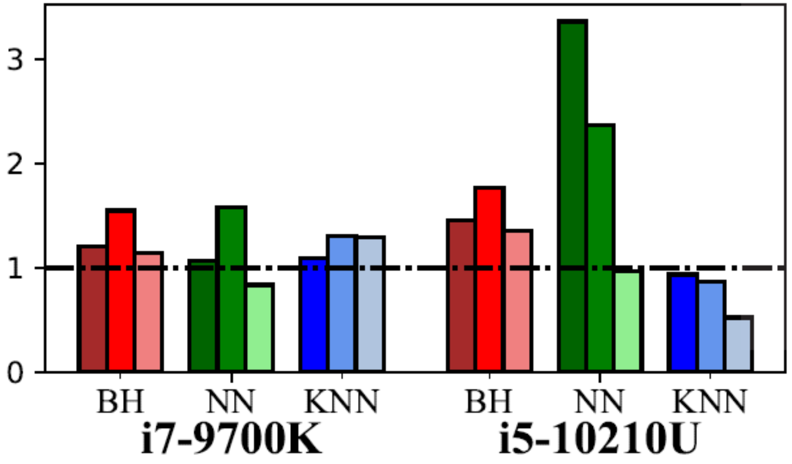
Grove Results Overview



Speedups of the best heterogeneous configuration vs. the best homogeneous configuration of Grove.

Grove Results Overview





Highlights

Duet
highest 13.53x
geomean 6.43x

Xavier
highest 8.12x
geomean 5.13x

Nano
highest 6.9x
geomean 4.71x

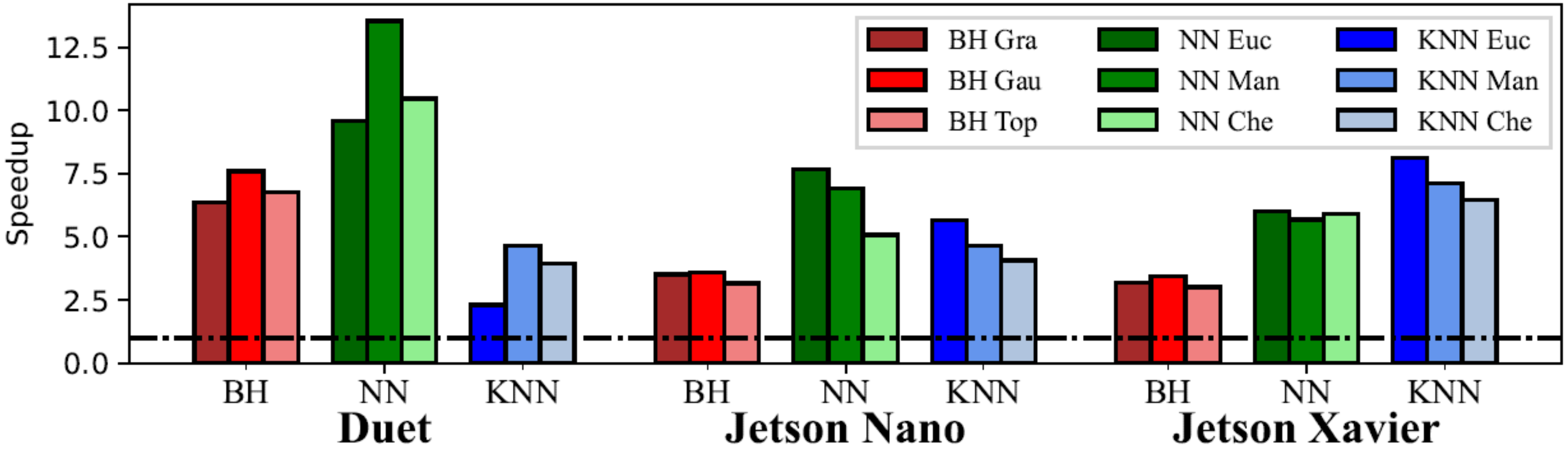
Intels
highest 3.36x
even has slow downs

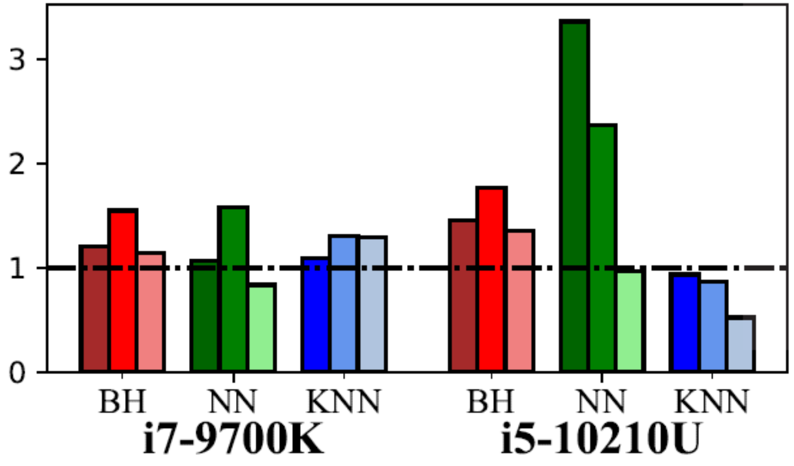
With minimal kernel launching overhead

Powerful NVIDIA GPUs But has small CPUs

Intel OoO is already fast, but has small GPUs

Grove Results Overview





Highlights

Duet
highest 13.53x
geomean 6.43x

Xavier
highest 8.12x
geomean 5.13x

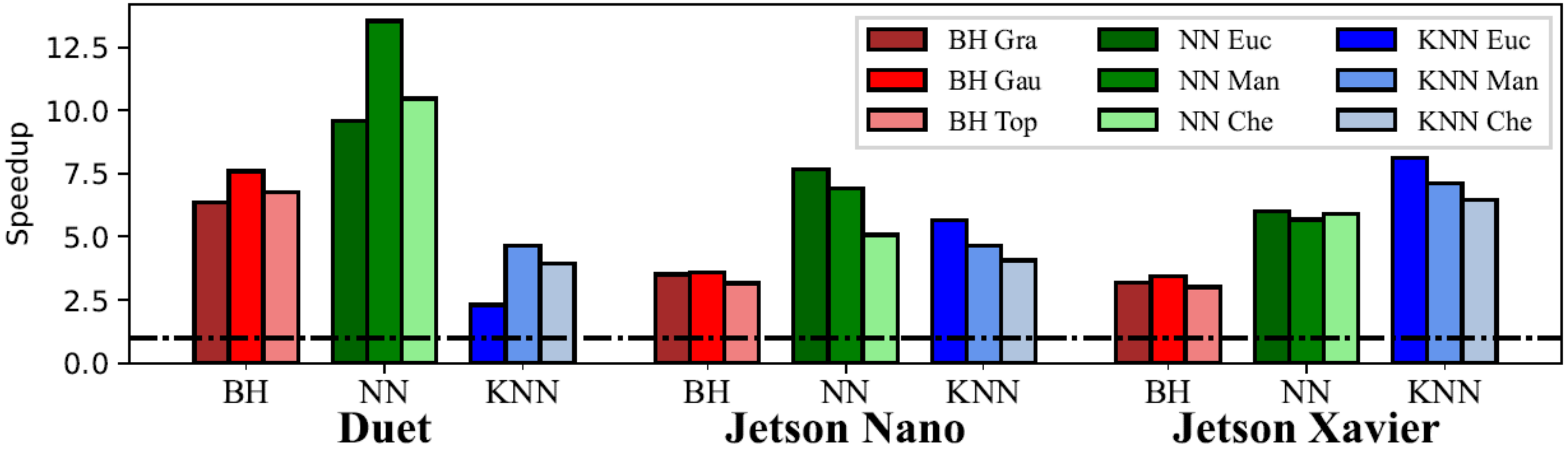
Nano
highest 6.9x
geomean 4.71x

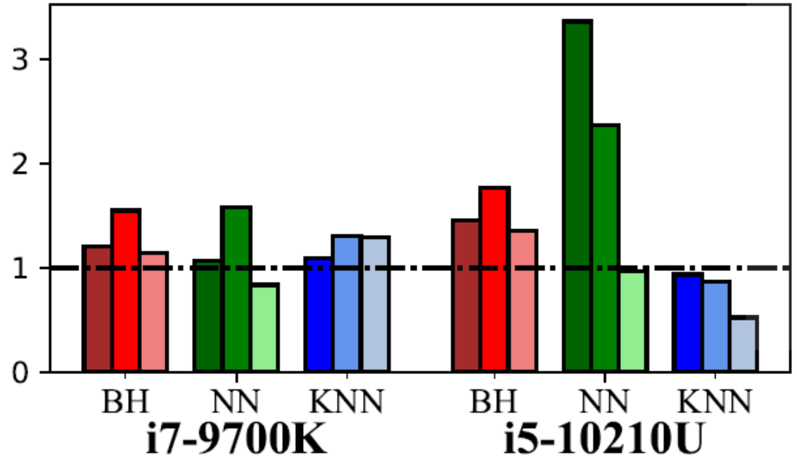
Intels
highest 3.36x
even has slow downs

Original Duet Paper[1] reported **~3x** speedup in BH algorithm. With Redwood's flexible task decomposition, we achieved **~6x** speedup

Intel OoO is already fast, but has small GPUs

Grove Results Overview





With Powerful Accelerators, more leaf node computations can be offloaded to GPU/FPGAs.

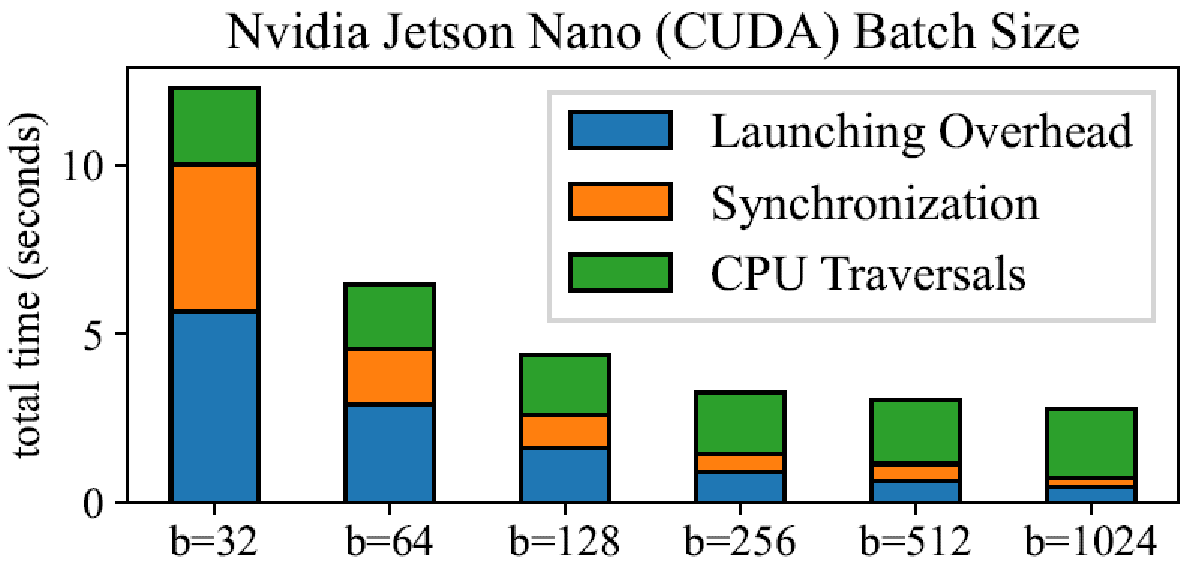
	BH Gra	BH Gau	BH Top	NN Euc	NN Man	NN Che	KNN Euc	KNN Man	KNN Che	Average	Ratio
Jetson Nano	256/8	256/8	128/8	512/256	512/256	512/256	256/128	256/128	256/256	327/115	2.26
Jetson Xavier	128/16	128/8	128/8	1024/128	1024/256	512/64	512/64	512/64	512/64	498/75	6.67
i7-9700k	64/8	128/8	128/8	256/64	256/64	128/64	64/64	64/64	64/64	128/45	2.82
i5-10210U	64/8	64/8	64/8	256/64	256/64	256/64	32/64	32/64	32/64	117/45	2.59
Duet	512/4	512/2	512/4	512/32	512/32	256/16	128/16	128/32	128/32	355/19	18.82
Average	205/9	218/7	192/7	512/109	512/134	333/93	198/67	198/70	198/96	285/66	4.33

Optimal leaf node size for heterogeneous (left) vs homogeneous (right) for each workload and platform

We have several insights in the paper

Kernel Submission Cost

- Traverse-compute applications **frequently** invoke small kernels
- Useful works are shown in Green
- Orange/Blue are overheads
- Low-cost kernel submission is important for accelerating applications on edge devices



Batching multiple GPU kernels into a single/larger kernel helps amortizing kernel launching overhead

Future work

- We are exploring more efficient CPU-PAPU communication methods

Conclusion

- ✓We identify a class of applications, *traverse-compute* applications, that are ideal for shared memory heterogeneous systems
- ✓We present Redwood: a framework for writing heterogeneous traverse-compute workloads
- ✓Using Redwood, we implemented Grove, a benchmark suite contains 9 traverse-compute applications
- ✓Finally evaluated 5 systems, achieving up to **13.53×** speedups (geomean of **3.01×**)

UC Santa Cruz Redwood Grove

Team

Yanwen Xu yxu83@ucsc.edu
Ang Li angl@princton.edu
Tyler Sorensen tyler.sorensen@ucsc.edu

Open-Source Repo

Redwood & Grove at
https://github.com/xuyanwen2012/redwood-rt