



IC設計所

Institute of Integrated Circuit Design



SUSTAINABLE
DEVELOPMENT
GOALS

Toward a Scalable Hardware-Software Co-Design for Large-Scale Graph Clustering

Student: 鄭名祐

Advisor: Sinha Aman

Abstract

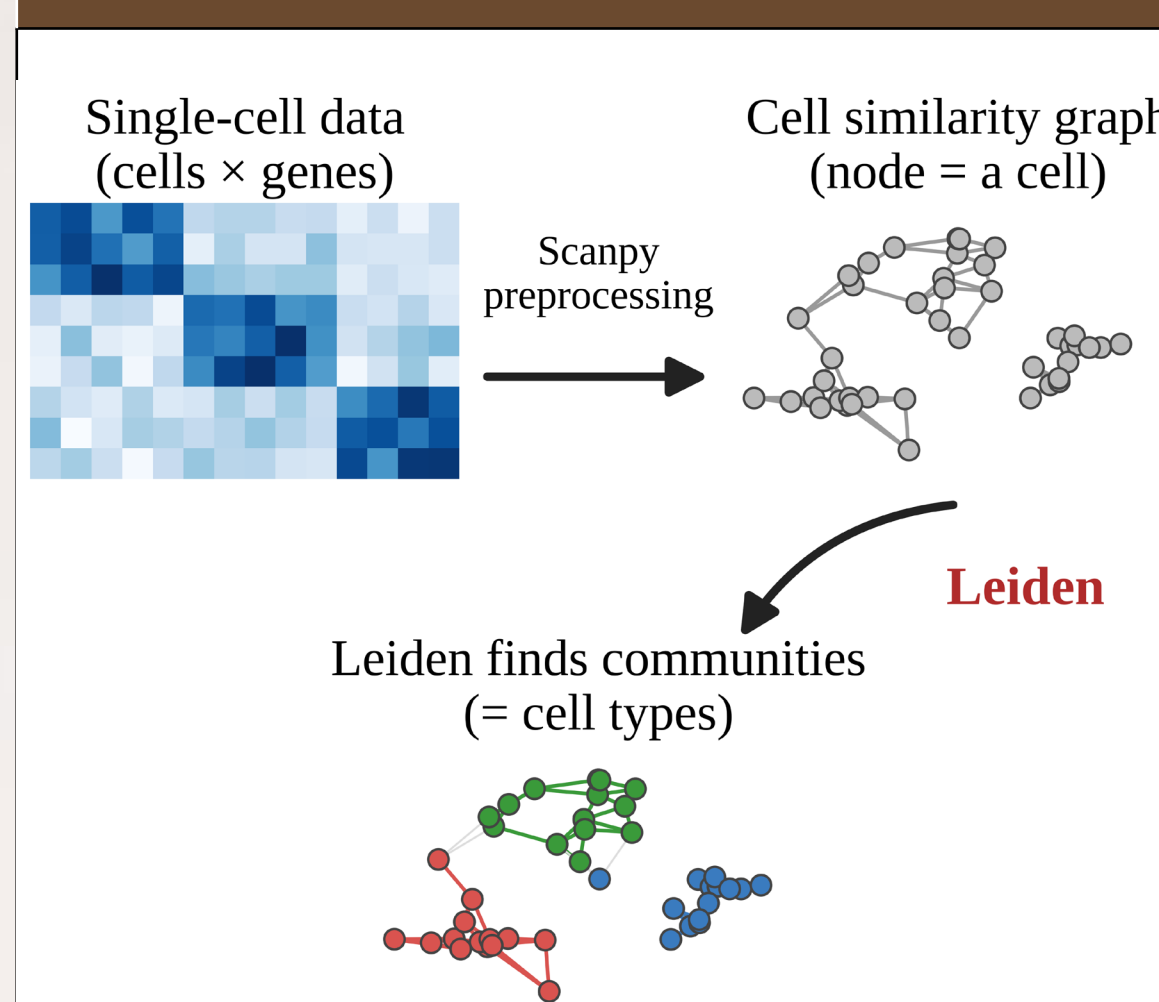
Leiden clustering is a core step in large-scale single-cell analysis, yet it accounts for up to 92% of Scanpy pipeline runtime and runs on a single core. Our profiling shows that it is strongly memory-bound with irregular access patterns and tight data dependencies, which limit naive parallelization. As a first step, we parallelize the aggregation stage and its sorting step with community-level graph partitioning, achieving a 2.45 $\times$ speedup at 32 threads while producing results bit-identical to the original implementation. Scaling then saturates, which our profiling suggests is related to DRAM-bound memory access and remaining serial steps. We propose a hardware-software co-design that combines graph partitioning, batching, dependency-aware scheduling and speculative execution, to be evaluated in gem5 and implemented on an AMD Alveo U55C FPGA with high-bandwidth memory.

Motivation

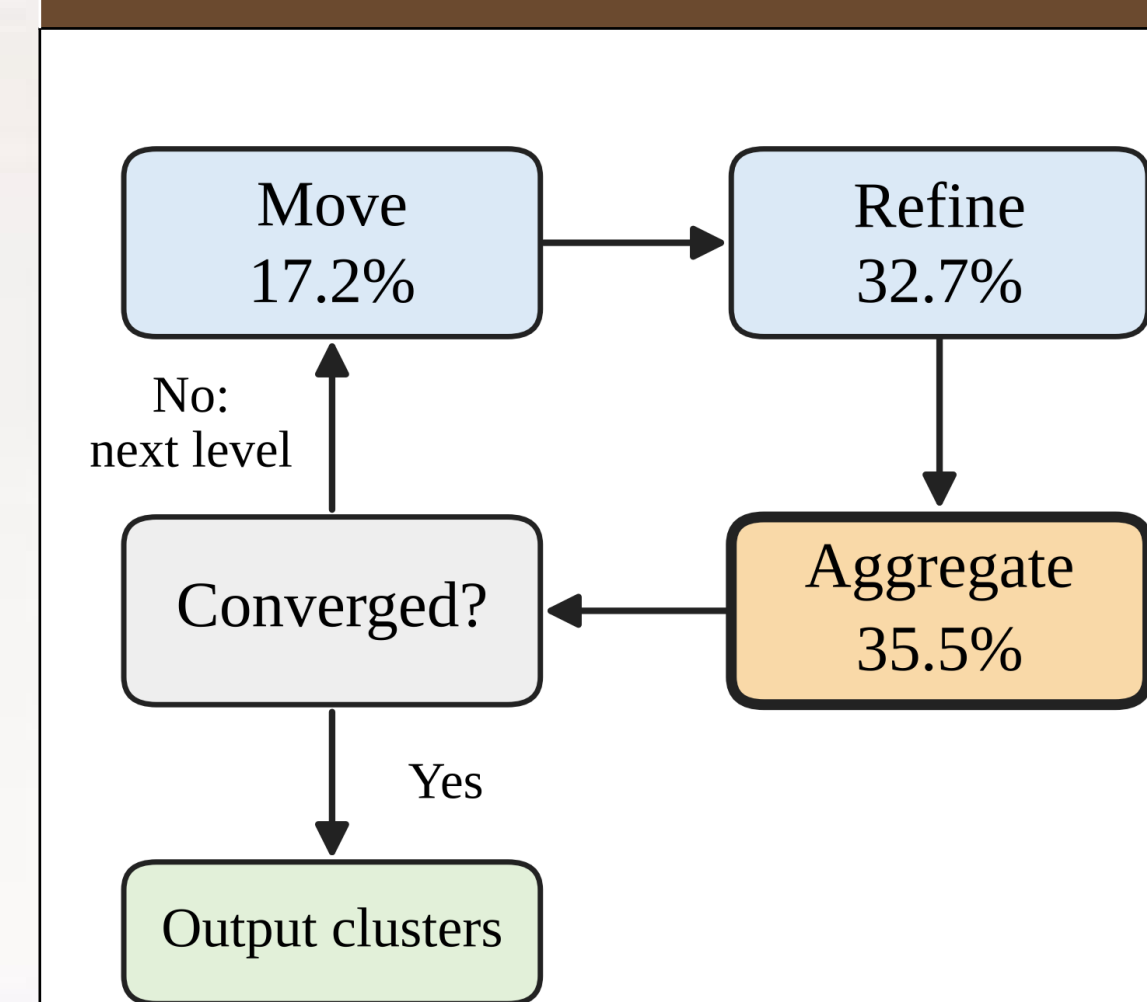
- **Single-cell RNA-seq** reveals gene activity cell by cell and drives research in cancer, immunology and neuroscience.
- Datasets now reach **millions of cells**, and **clustering** is a required step to identify cell types.
- In the Scanpy pipeline, **Leiden clustering takes ~92% of total runtime** on a 1.3M-cell dataset, and it runs on **a single CPU core**.
- Researchers often re-run clustering at several resolutions, so this cost is paid **many times** per analysis.

Accelerating Leiden is key to scalable single-cell analysis.

Leiden Algorithm

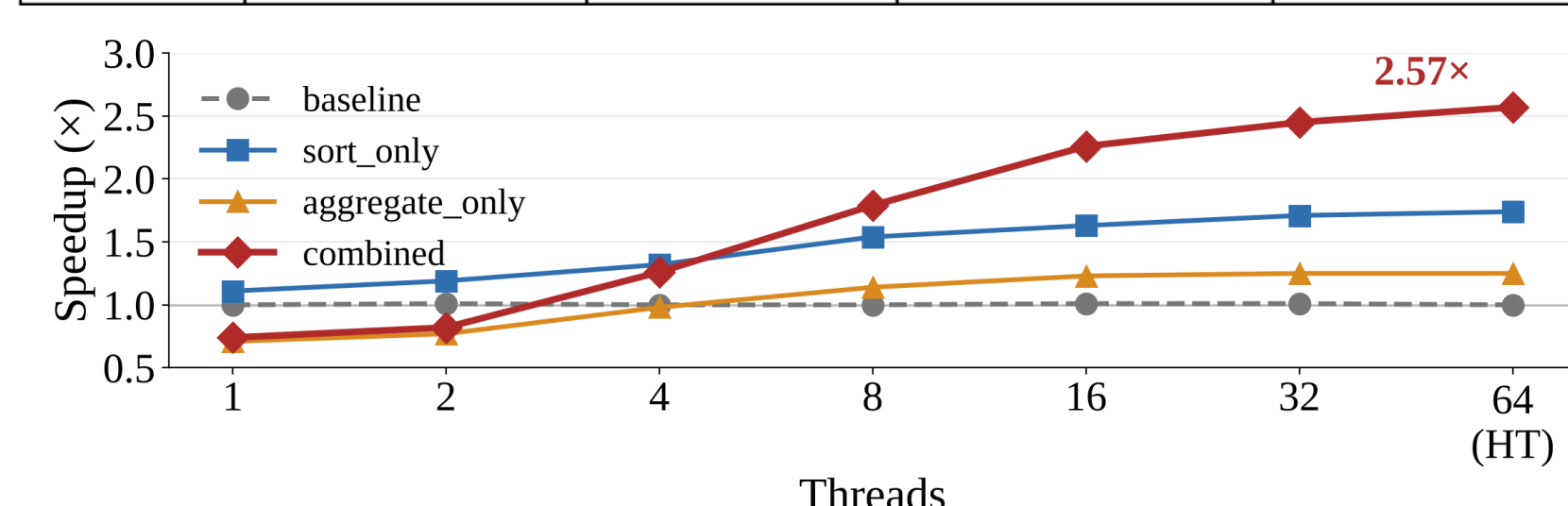


Leiden 4 Stages



Baseline & Parallel Aggregate

1M snapshot (1-thread baseline = 2.711s)				
Threads	baseline	sort_only	aggregate_only	combined
1	2.711 (1.00x)	2.443 (1.11x)	3.833 (0.71x)	3.655 (0.74x)
2	2.691 (1.01x)	2.278 (1.19x)	3.503 (0.77x)	3.314 (0.82x)
4	2.707 (1.00x)	2.052 (1.32x)	2.755 (0.98x)	2.145 (1.26x)
8	2.711 (1.00x)	1.758 (1.54x)	2.385 (1.14x)	1.511 (1.79x)
16	2.694 (1.01x)	1.662 (1.63x)	2.202 (1.23x)	1.200 (2.26x)
32	2.694 (1.01x)	1.589 (1.71x)	2.176 (1.25x)	1.108 (2.45x)
64 (HT)	2.711 (1.00x)	1.555 (1.74x)	2.167 (1.25x)	1.054 (2.57x)

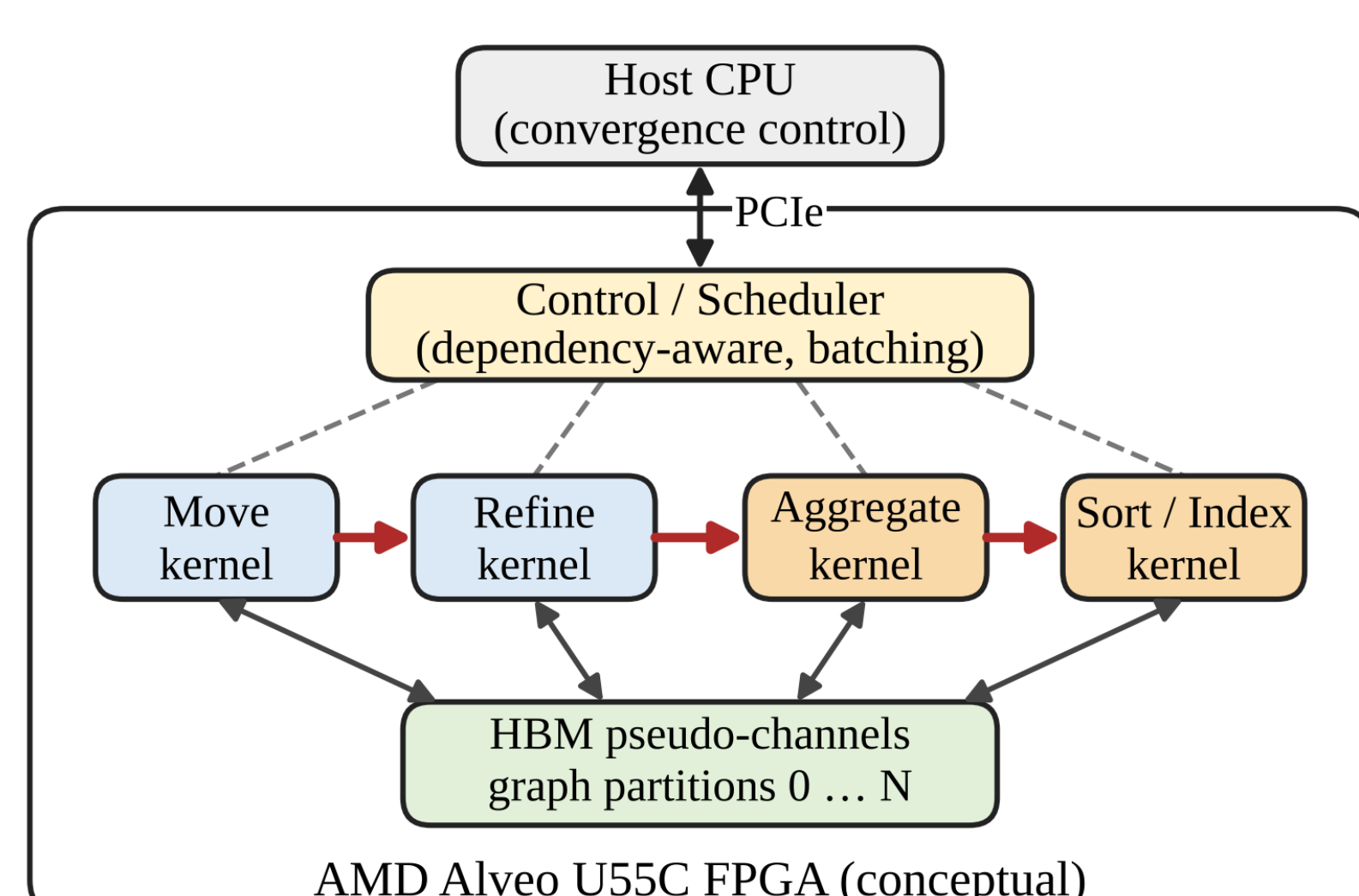


Workload Characterization (VTune, T=32)

Region	CPU time (s)	Share
Parallel Aggregate (collapse_graph)	38.39	43.8%
Parallel sort (pair_order_parallel)	26.46	30.2%
Memory allocation / copy (new, free, memset, memmove)	9.39	10.7%
Unresolved symbols	9.77	11.1%
Other igrph (incl. serial create_start_vectors 0.62 s)	2.01	2.3%
Others	1.67	1.9%
Total	87.69	100%

Function	DRAM Bound	L3 Bound	L1 Bound	Avg latency (cycles)
collapse_graph_omp_fn.0	66.5%	5.8%	9.6%	83
m4_pair_order_parallel_omp_fn.1	53.5%	13.1%	1.7%	263

Proposed HBM FPGA (U55C) Architecture



- **One kernel per stage:** Move, Refine, Aggregate and Sort/Index run as dedicated hardware kernels instead of sharing one CPU core.
- **Graph kept in HBM:** The graph is split into partitions stored in separate HBM pseudo-channels, so kernels access data in parallel and cover memory latency.
- **Host controls convergence:** The host CPU checks convergence and starts the next level over PCIe, while a scheduler on the FPGA handles dependencies and batching.

Conclusion & Future Work

- **Expected:** Stage kernels run in parallel on HBM channels to reduce memory stalls.
- **Conclusion:** Parallel Aggregate + sort reaches 2.45 $\times$ at 32 threads with bit-identical output. Scaling then saturates, suggesting DRAM-bound memory access and remaining serial steps.
- **Future work:** Parallelize sort and index build; accelerate Move and Refine; evaluate in gem5; implement on Alveo U55C.