2025 AI System Design
(Term Project)

KU KONKUK UNIVERSITY

TP

2025. 12. 12.

Students
202110410 조민우
202110365 김상만

Outline

KU KONKUK UNIVERSITY

- **Optimizations of AI Accelerator Networks**
  - Problem Definition
  - Strategy & DSE Algorithm
  - Ablation Study
  - Code Review
  - Result of TP (Evaluation)
    - Optimality
    - Sampling efficiency
    - General applicability
  - External Module Import

Problem Definition

KU KONKUK UNIVERSITY

- **Objective of TP**
  - **Optimizitions of BERT Network based on NetLMSim**
    - **Optimize in the view of Partitioning/Placement**
      - Only partitioning/placement configuration varied
      - We use Heuristic Algorithm for Optimization
    - **Optimize Throughputs & Power Dissipation /w fewer iterations**
      - Optimality is evaluated by “Thoughput / n ” “ Power Dissipition / n ” (n is # of iterations)
    - **Fixed Hardware (36PE, 12DRAM)**
      - Our Model performs well across diverse HW specs (e.g., 144PE, 24DRAM)

Strategy & DSE algorithms (1/2)

KU KONKUK UNIVERSITY

- **Strategy**
  - **Introduction**

-> Our strategy is divided into three stages

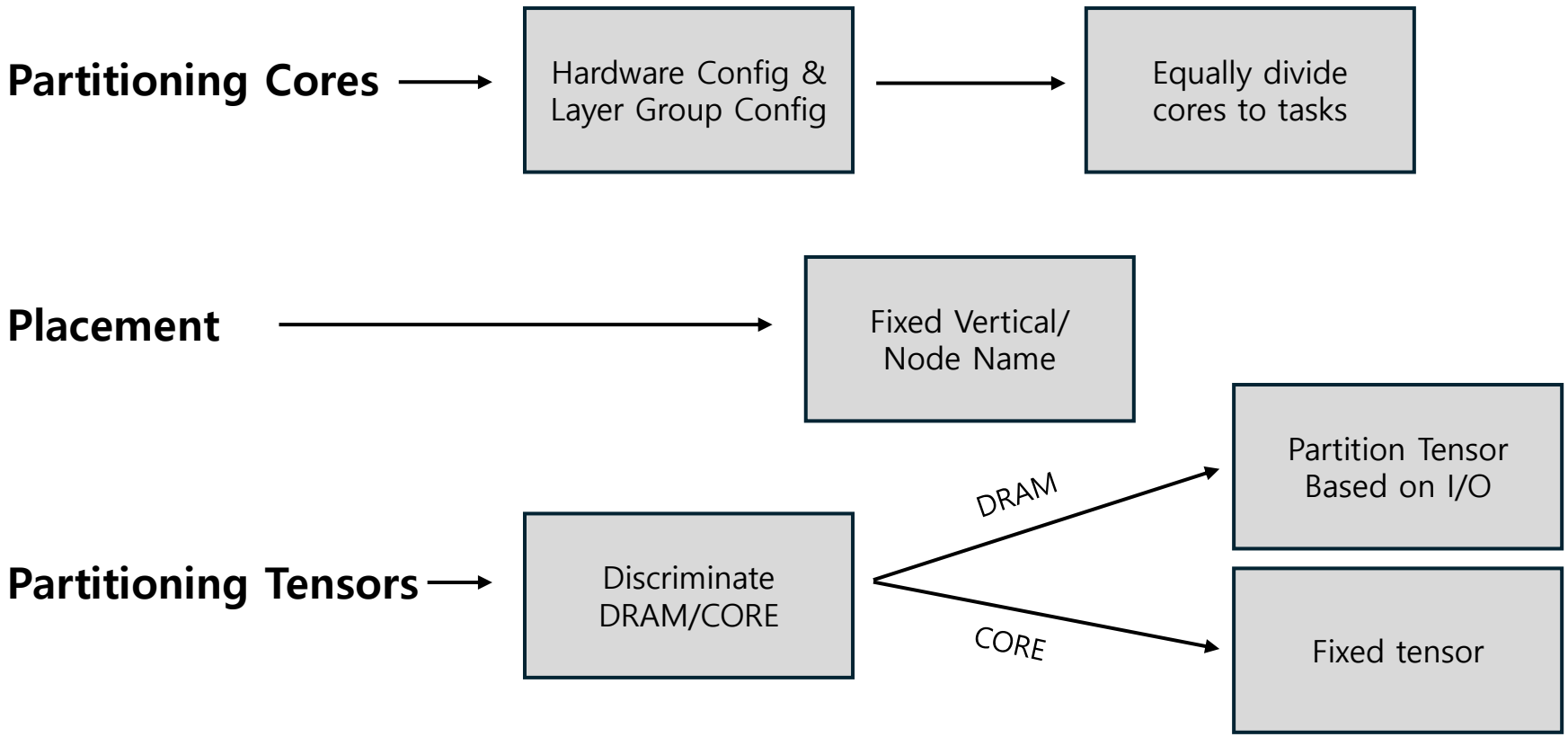
	Phase 1	Phase 2	Phase 3	Phase 4
Iterations index	1	2~4	5~7	8+
Partitioning Cores	Equal	Execution_time.csv, Gradient	Fixed Best	if local minima, go to phase2
Placement	Vertical	Vertical	Try another (Horizontal, Scatter)	bw_eff Search

< Specification of Strategy >

Strategy & DSE algorithms (2/5)

KU KONKUK UNIVERSITY

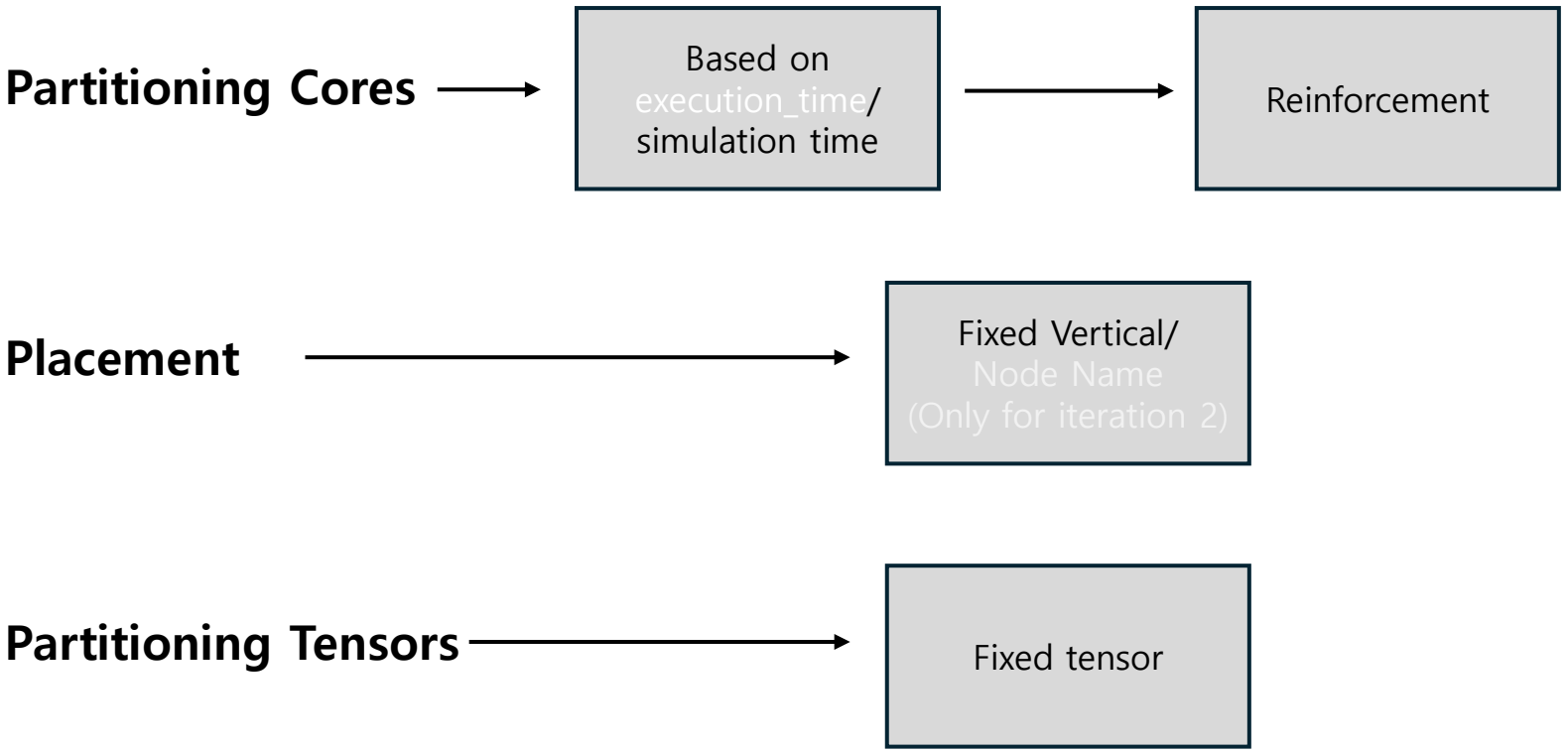
- **DSE algorithms**
  - **Phase 1 (iteration 1)**



Strategy & DSE algorithms (3/5)

KU KONKUK UNIVERSITY

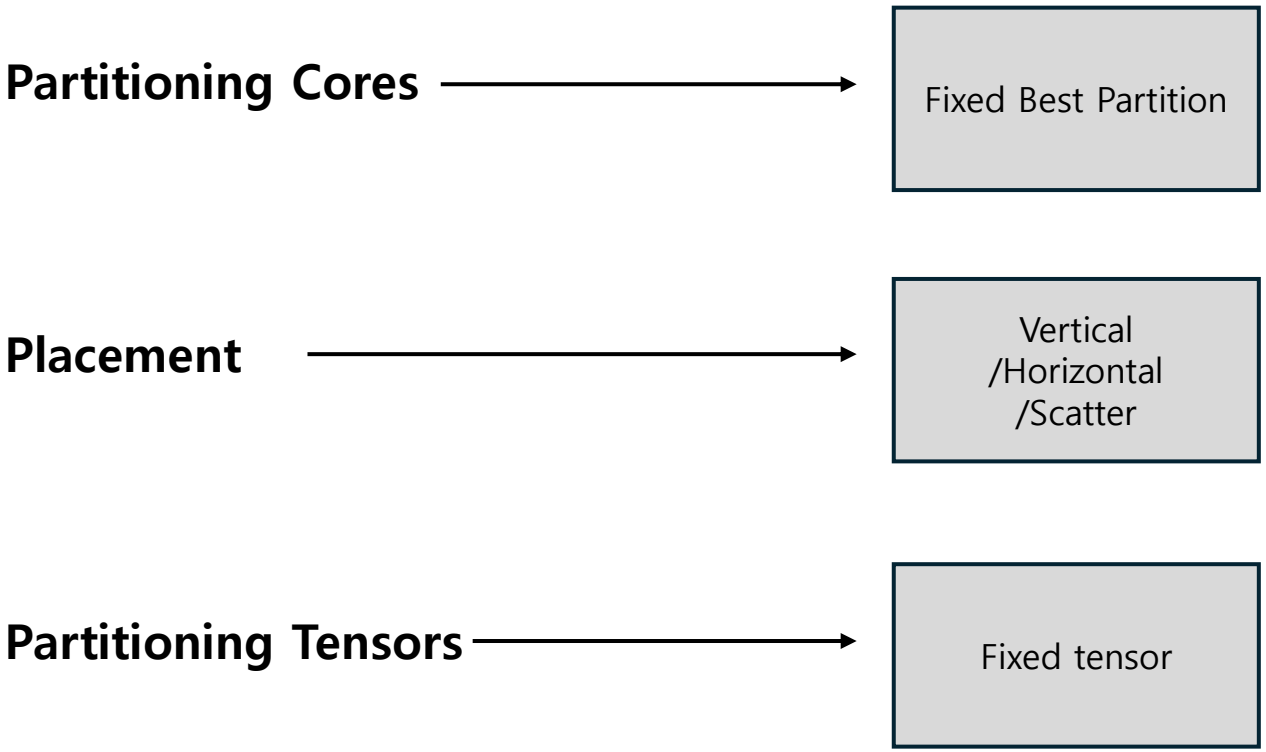
- **DSE algorithms**
  - **Phase 2 (iteration 2-4)**



Strategy & DSE algorithms (3/5)

KU KONKUK UNIVERSITY

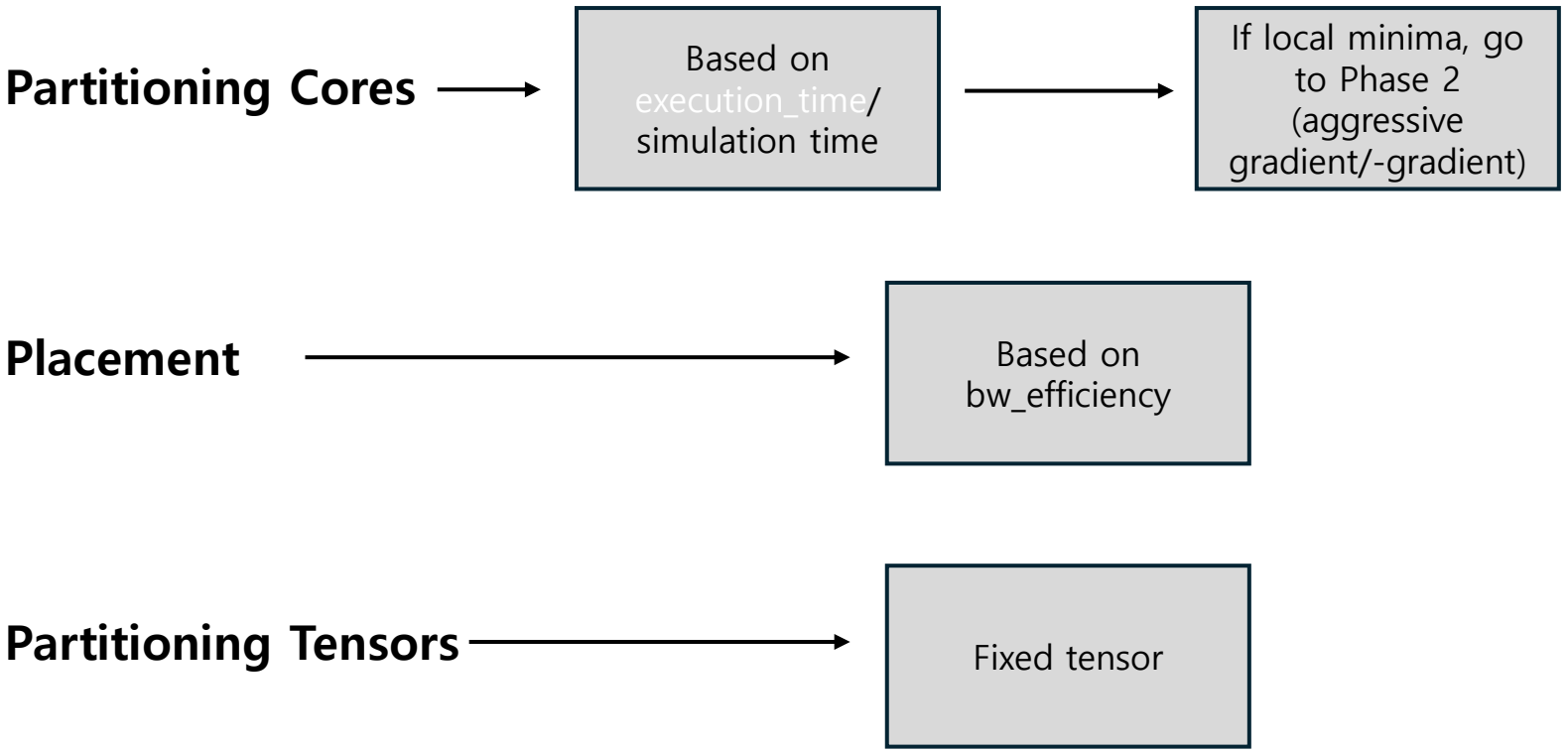
- **DSE algorithms**
  - **Phase 3 (iteration 5-7)**



Strategy & DSE algorithms (3/5)

KU KONKUK UNIVERSITY

- **DSE algorithms**
  - **Phase 4 (iteration 8+)**



Strategy & DSE algorithms (3/5)

KU KONKUK UNIVERSITY

- **DSE algorithms**
  - **Merit of Algorithm**

1) **Divide and Conquer strategy makes efficient search space from a broad range to a specific region**
2) **All layer groups (A, B, and C) achieve high performance even under simultaneous simulation**
   - Verified by later slides
3) **Scalable performance across various hardware setups, not limited to the (36PE, 12DRAM) case**
   - Verified by later slides
4) **Utilization of momentum ensures effectively escape local minima**

Ablation Study (1/5)

KU KONKUK UNIVERSITY

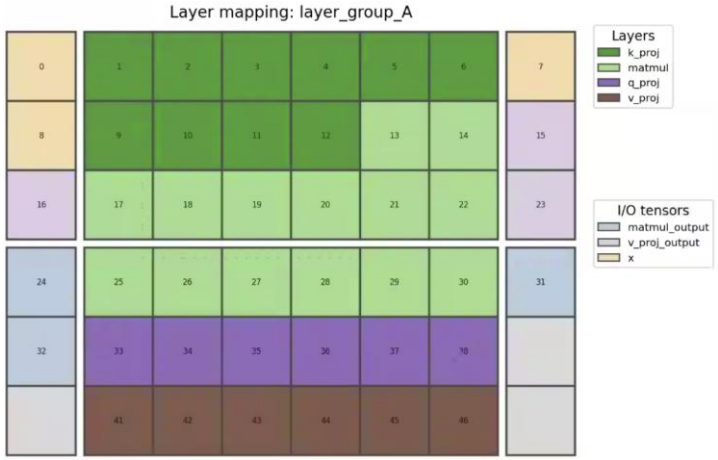
- **Method 1 -**
  - **Randomly Shift**

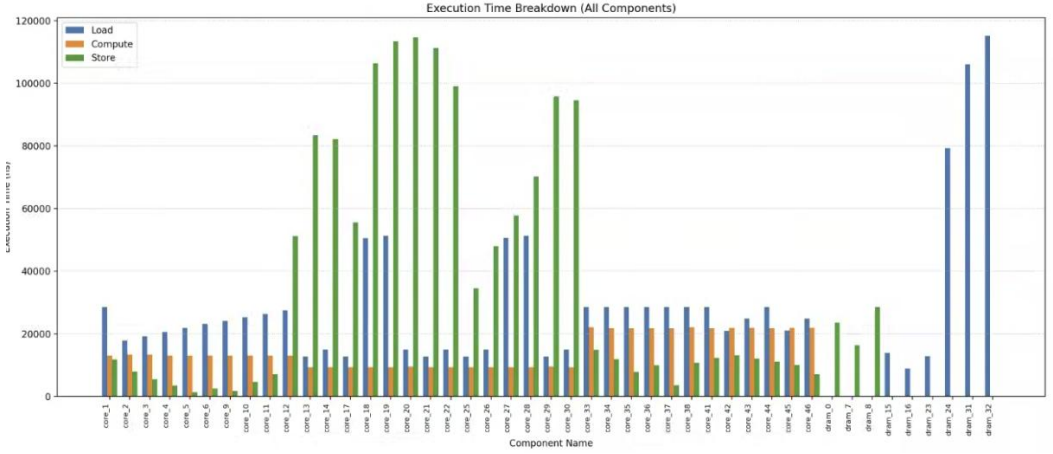
Best Iteration 7 (total 20)

- Time_ns : 115210.0
- Energy_mj: 0.061314

```python
def _rotate_router_ids(
    nodes: Dict[str, Any],
    skip_if: Optional[Callable[[str, Dict[str, Any]], bool]] = None,
) -> None:
    """Mutate nodes in-place by rotating their `router_ids`."""
    for node_key, node in nodes.items():
        if skip_if is not None and skip_if(node_key, node):
            continue

        ids = node.get("router_ids")
        if isinstance(ids, list) and ids:
            if len(ids) == 1:
                shift = 1
            else:
                shift = random.randint(1, len(ids) - 1)
            node["router_ids"] = ids[shift:] + ids[:shift]
```

Main idea





Ablation Study (2/5)

KU KONKUK UNIVERSITY

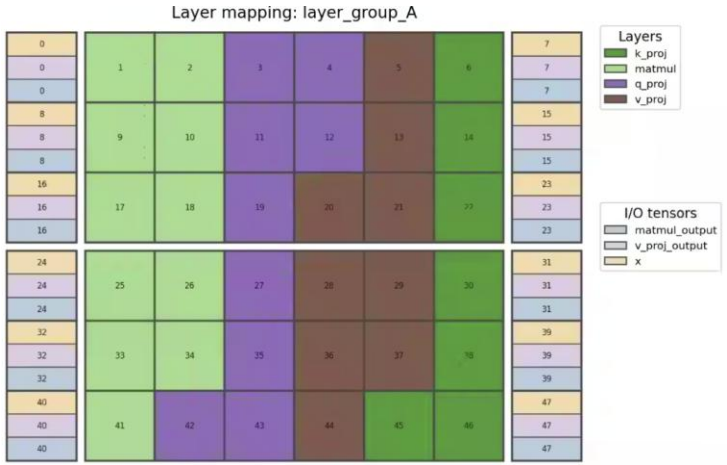
- **Method 2 -**
  - **Partitioning Reinforcement / Placement Random(H/V/S/I)**

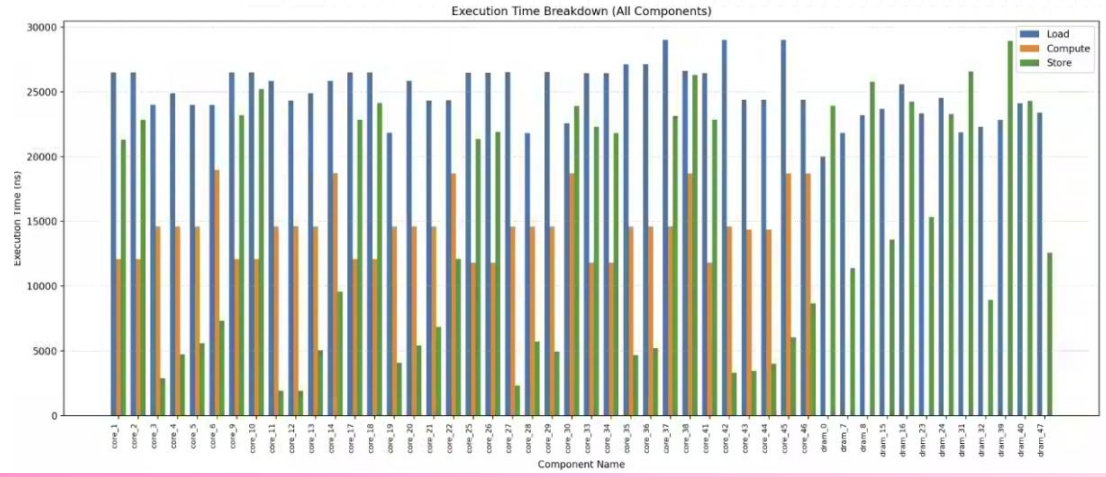
Best Iteration 13 (total 20)

- Time_ns : 29000.0
- Energy_mj: 0.058525

```python
# Compare N-1 vs N-2
if iteration > 2 and t2 > 0:
    delta_t = t1 - t2
    delta_c = c1 - c2

    if delta_t < 0: # Performance Improved
        if delta_c > 0: modifier = 1.3    # Added -> Good -> Add More
        elif delta_c < 0: modifier = 0.8  # Removed -> Good -> Remove More
    else: # Performance Regressed/Same
        if delta_c > 0: modifier = 0.8    # Added -> Bad -> Remove
        elif delta_c < 0: modifier = 1.3  # Removed -> Bad -> Add

    debug_info[n] = {"metric": f"Grad(x{modifier})", "val": base_weight * modifier}
else:
    debug_info[n] = {"metric": "Time(ns)", "val": base_weight}
```

Main idea





Ablation Study (3/5)

KU KONKUK UNIVERSITY

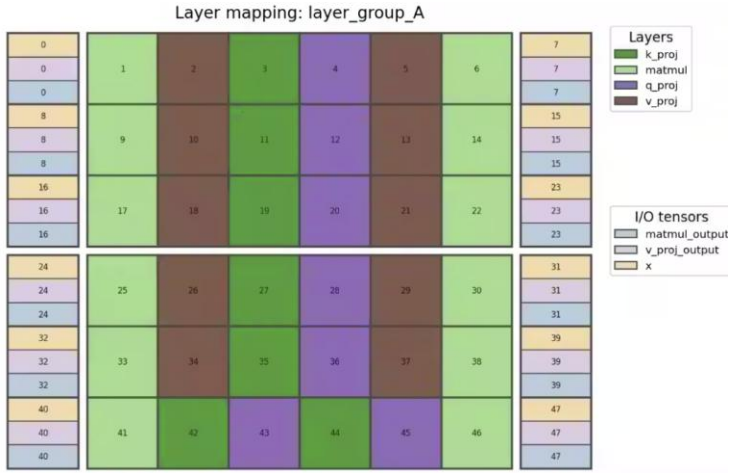
- **Method 3 -**
  - **Early Fix Partitioning / Variation on Placement (based on bw_efficiency)**

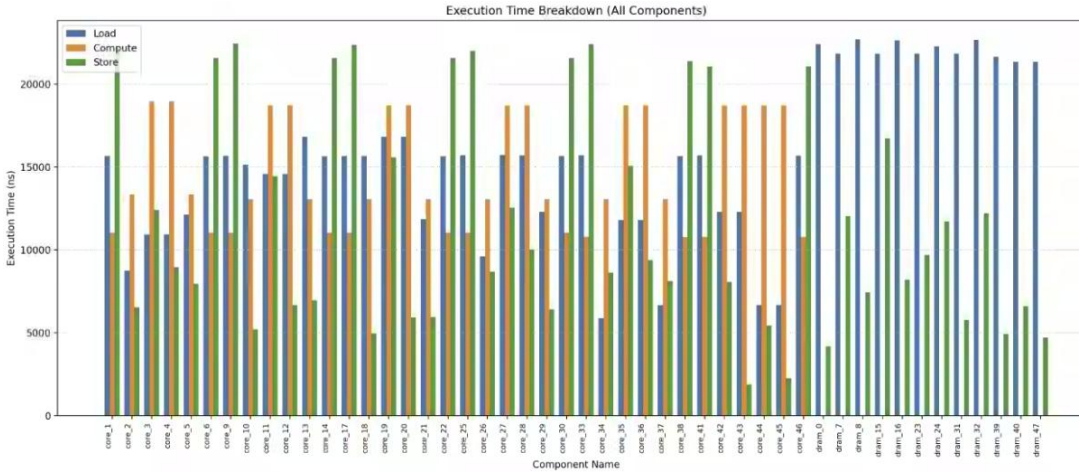
Best Iteration 11 (total 20)

- **Time_ns : 22690.0**
- **Energy_mj: 0.057967**

```
for n in node_names:
    rids = prev_grp.get(n, {}).get("router_ids", [])
    if not rids: continue

    # Avg eff of these routers
    node_effs = [g_links.get(str(r), g_links.get(int(r), 999.0)) for r in rids]
    avg_val = sum(node_effs)/len(node_effs) if node_effs else 999.0

    if avg_val < worst_eff:
        worst_eff = avg_val
        victim_node = n

if victim_node:
    logging.info(f"   [Opt] Victim Node: {victim_node} (Eff: {worst_eff:.2f})")
    debug_info[victim_node] = {"metric": "Victim", "val": worst_eff}
```

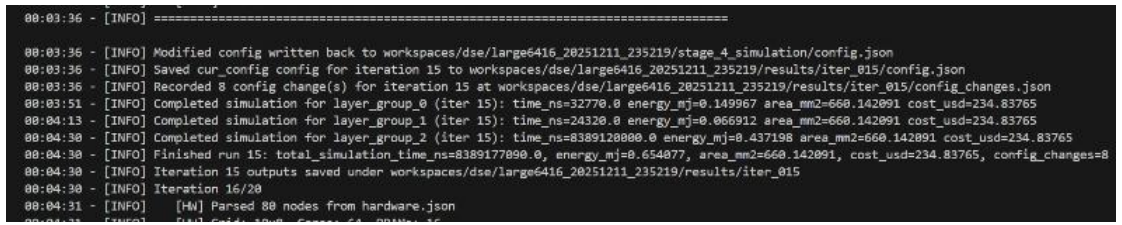
Main idea





Ablation Study (4/5)

- **Method 4 -**
  - **Power Dissipation Partitioning**



Best Iteration 14 (total 20)

- Energy_mj: 0.6540

```python
def find_best_divisors(num_cores, dim_sizes):
    """Find best split respecting dim sizes AND constraints"""
    if not dim_sizes:
        return [1], 0

    # Try each dimension with decreasing priority
    for dim_info in dim_sizes:
        if not dim_info['size']:
            continue

        # Find largest divisor that fits BOTH
        max_split = min(num_cores, dim_info['size'])

        # Try powers of 2 first, then other divisors
        candidates = []
        p = 1
        while p <= max_split:
            candidates.append(p)
            p *= 2

        # Add non-power-of-2 divisors (for 6, 12, 18 etc)
        for div in [3, 6, 9, 12, 18]:
            if div <= max_split and div not in candidates:
                candidates.append(div)

        candidates.sort(reverse=True)

        for split in candidates:
            if num_cores % split == 0 and dim_info['size'] % split == 0:
                chunk = dim_info['size'] // split
                # Relax alignment check for small splits
                if split <= 8 or chunk % 64 == 0 or chunk >= 64:
                    return [split], dim_info['index']

    # Fallback: 1 (no split)
    return [1], 0
```

Main idea

Note :
In architectures composed of factors that are powers of two—such as **64 cores / 16 DRAM** or **8 cores / 4 DRAM**—we were able to optimize data movement and thereby improve power & performance efficiency. However, **TP_AS25 did not show noticeable improvements power efficiency.** The reason is that the model consists of **36 cores (6×6)**, and the partitioning must be configured using factors of **512**. With a 6×6 structure, it is highly restrictive to fully leverage those divisor-based partition constraints when arranging the partitions.

Ablation Study (5/5)

KU KONKUK UNIVERSITY

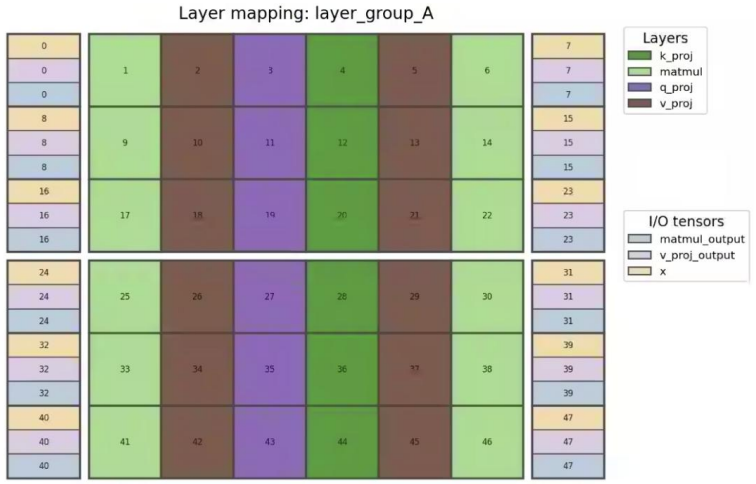
- **Method 5 -**
  - **Our Strategy**

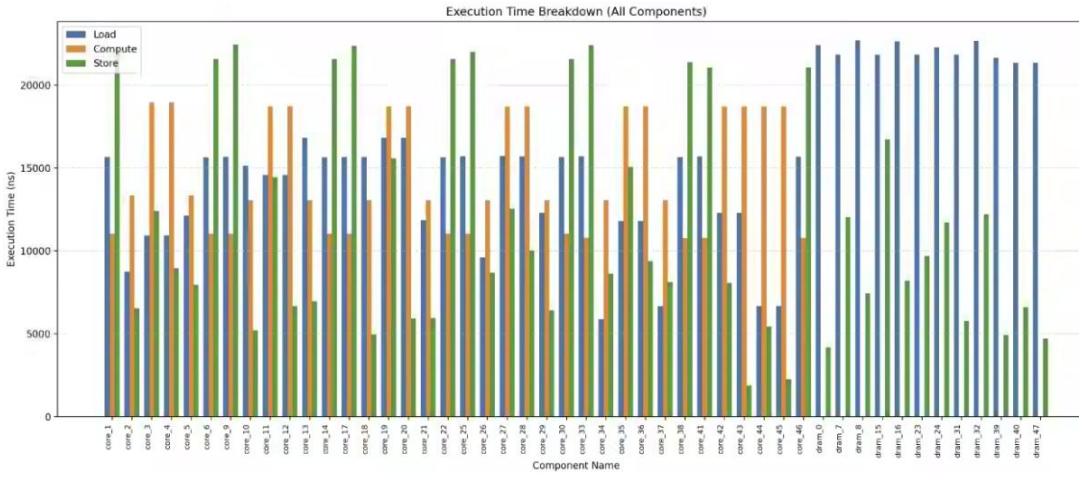
Best Iteration 2 (total 20)

- **Time_ns :** **22690.0**
- **Energy_mj:** **0.057967**

Later pages explained

Main idea





Code Review (1/7)

KU KONKUK UNIVERSITY

- **Notice**
  - **We commented out the data loading code to prevent crash with layer 0,1,2**

    - Instead, we use the data loading logic within the edited code block

```
'''
if last_iter_dir.exists() & (iteration != 1):
    #log_dir_path = last_iter_dir / "logs/layer_group_A"
    sim_result_path = last_iter_dir / "simulation_result.json"
    log_alloc_path = log_dir_path / "allocation.csv"
    log_exec_time_path = log_dir_path / "execution_time.csv"
    log_flowID_path = log_dir_path / "flowID.csv"
    log_link_load_path = log_dir_path / "link_load.csv"
    log_routers_path = list(log_dir_path.glob("router_*.csv"))
    log_router_path = [f for f in log_routers_path if re.match(r'^router_\d+\.csv$', f.name)]
    log_router_buf_path = [f for f in log_routers_path if re.match(r'^router_\d+_buf\.csv$', f.name)]
    log_d2d_buf_path = list(log_dir_path.glob("d2d_*_*_buf.csv"))

    with open(sim_result_path, "rb") as f:
        sim_result = orjson.loads(f.read())
    throughput = sim_result['data']['summary']['total_throughput_ops']
    power_effs = sim_result['data']['summary']['power_efficiency_ops_per_w']
    alloc = pd.read_csv(log_alloc_path)
    exec_time = pd.read_csv(log_exec_time_path)
    flowID = pd.read_csv(log_flowID_path)
    link_load = pd.read_csv(log_link_load_path)
    routers = [pd.read_csv(file, skipinitialspace=True) for file in log_router_path]
    router_bufs = [pd.read_csv(file, dtype=str, skipinitialspace=True) for file in log_router_buf_path]
    d2d_bufs = [pd.read_csv(file, dtype=str, skipinitialspace=True) for file in log_d2d_buf_path]
'''
""" Edit code below """
```

```
# Parse hardware
nodes_data = {}
grid_config = {}

if hardware_path and hardware_path.exists():
    try:
        with open(hardware_path, "rb") as f:
            hw_cfg = orjson.loads(f.read())
            grid_config = hw_cfg.get("grid_config", {})
            nodes_raw = hw_cfg.get("component_mapping") or hw_cfg.get("nodes") or hw_cfg.get("routers", {})
            if isinstance(hw_cfg, list): nodes_raw = hw_cfg

            if isinstance(nodes_raw, list):
                for item in nodes_raw:
                    nid = item.get("id", item.get("router_id"))
                    if nid is not None: nodes_data[int(nid)] = item
            elif isinstance(nodes_raw, dict):
                for k, v in nodes_raw.items(): nodes_data[int(k)] = v

            logging.info(f"   [HW] Parsed {len(nodes_data)} nodes from {hardware_path.name}")
    except Exception as e:
        logging.error(f"   [ERROR] JSON Read Failed: {e}")
```

Simulating the new hardware with workspace_name allows us to identify the informations(e.g. router_id…) present in the system (within def load_iter_data)

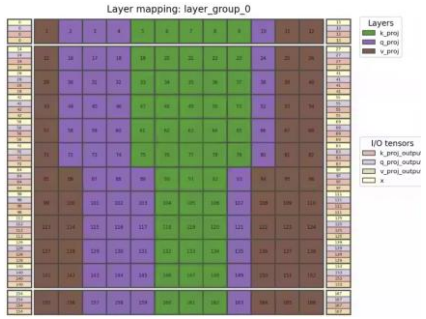
Code Review (2/7)

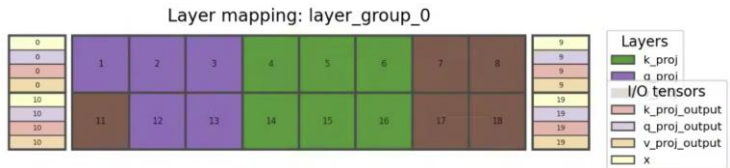
- **Config setup**
  - **Get grid dimensions & Guarantees Min Core Per Task**

```python
# Grid dimensions
if "grid_x" in grid_config and "grid_y" in grid_config:
    GRID_WIDTH = int(grid_config["grid_x"])
    GRID_HEIGHT = int(grid_config["grid_y"])
else:
    total_routers = len(all_router_ids)
    # if total_routers == 48:
    #     GRID_WIDTH, GRID_HEIGHT = 8, 6
    # elif total_routers == 64:
    #     GRID_WIDTH, GRID_HEIGHT = 8, 8
    # else:
    side = int(math.sqrt(total_routers)) if total_routers > 0 else 1
    GRID_WIDTH = side
    GRID_HEIGHT = math.ceil(total_routers / side)
# Router positions
router_positions = {}
for rid in all_router_ids:
    node_info = nodes_data.get(rid, {})
    if "x" in node_info and "y" in node_info:
        router_positions[rid] = (int(node_info["x"]), int(node_info["y"]))
    else:
        router_positions[rid] = (rid % GRID_WIDTH, rid // GRID_WIDTH)

MIN_CORE_RATIO = 0.03
MIN_CORES_PER_TASK = max(1, int(len(CORE_POOL) * MIN_CORE_RATIO))
```

```python
# Special handling for large DRAM groups
if count > 6 and is_dram_group:
    sorted_all_ids = [x[0] for x in pool_coords]
    mid_x = GRID_WIDTH // 2
    l_ids = [pid for pid in sorted_all_ids if get_pos(pid)[0] < mid_x]
    r_ids = [pid for pid in sorted_all_ids if get_pos(pid)[0] >= mid_x]
    half = count // 2
    selected = l_ids[:half] + r_ids[:count - half]
    if len(selected) < count:
        remain_set = set(selected)
        for pid in sorted_all_ids:
            if pid not in remain_set:
                selected.append(pid)
                if len(selected) == count: break
    return selected
```

get_placement(we need grid information to placement)



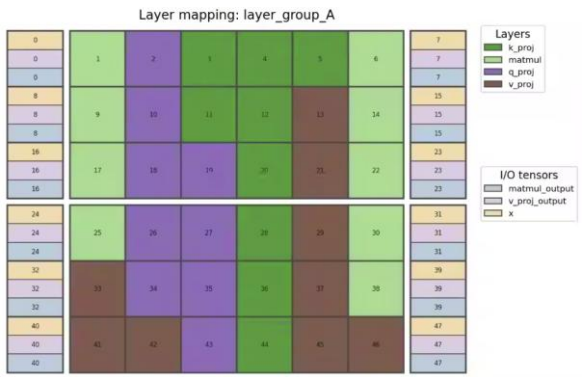


Maintains vertical placement and guarantees min core count even on small & Large HW.

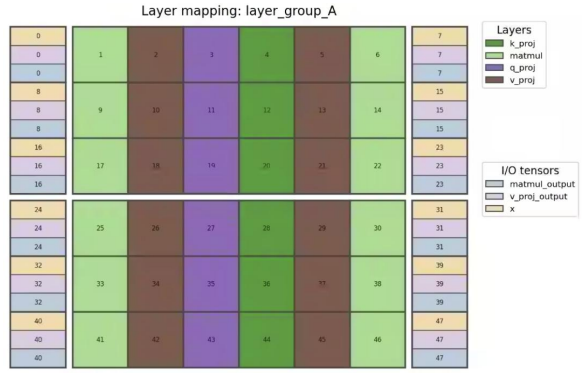
Code Review (3/7)

- **Placement**
  - **def get_placement**

```python
def get_placement(strategy, count, available_pool, node_name="", use_name_heuristic=False):
    if count <= 0: return []
    valid_pool = validate_router_list(available_pool, "core")
    if count > len(valid_pool):
        logging.warning(f"   [WARN] {node_name}: Requested {count} but only {len(valid_pool)} available")
        return valid_pool

    pool_coords = [(rid, get_pos(rid)[0], get_pos(rid)[1]) for rid in valid_pool]

    # NAME-BASED HEURISTIC (Only for Iter 1-2)
    if use_name_heuristic:
        name_lower = node_name.lower() if node_name else ""

        is_dram_group = any(k in name_lower for k in ["matmul", "v_proj"])
        is_ff_group = "ff" in name_lower
        is_q_proj = "q_proj" in name_lower
        is_k_proj = "k_proj" in name_lower

        q_col = GRID_WIDTH // 2 - 1 if GRID_WIDTH > 3 else 0
        k_col = GRID_WIDTH // 2 if GRID_WIDTH > 4 else 1

        if is_dram_group:
            pool_coords.sort(key=lambda k: (min(k[1], GRID_WIDTH-1-k[1]), k[2]))
        elif is_ff_group:
            mid_row = (GRID_HEIGHT - 1) / 2.0
            pool_coords.sort(key=lambda k: (abs(k[2] - mid_row), k[1]))
        elif is_q_proj:
            pool_coords.sort(key=lambda k: (abs(k[1] - q_col), k[2]))
        elif is_k_proj:
            pool_coords.sort(key=lambda k: (abs(k[1] - k_col), k[2]))
        else:
            if strategy == 'H':
                pool_coords.sort(key=lambda k: (k[2], k[1]))
            elif strategy == 'V':
                pool_coords.sort(key=lambda k: (k[1], k[2]))
            elif strategy == 'S':
                cx = sum(p[1] for p in pool_coords)/len(pool_coords)
                cy = sum(p[2] for p in pool_coords)/len(pool_coords)
                pool_coords.sort(key=lambda k: (k[1]-cx)**2 + (k[2]-cy)**2)
```



Iteration 1



Iteration 2

While adhering to the Placement Strategy (V, H/S), the name-based search in Iterations 1 and 2 ensures that compute-intensive nodes are placed near the DRAM

(Details on iteration behavior and DRAM exception handling are omitted)

Code Review (4/7)

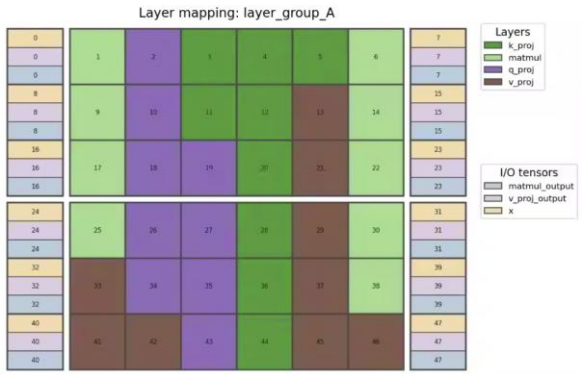
KU KONKUK UNIVERSITY

- **Phase**
  - **Phase 1 (iteration 1)**

```python
if iteration == 1:
    # ============= ITERATION 1: EQUAL SPLIT + NAME-BASED PLACEMENT =============
    logging.info(f"   [{group_key}] Phase 1: BASELINE (Equal Split)")
    base_strat = 'V'
    use_name_heuristic = True

    for n in node_names: target_weights[n] = 1.0

    nodes_to_alloc = node_names
    available_cores = sorted(list(set(CORE_POOL)))
    total_w = sum(target_weights.values()) or 1
    remaining = len(available_cores) - len(nodes_to_alloc) * MIN_CORES_PER_TASK
    for n in nodes_to_alloc: target_counts[n] = MIN_CORES_PER_TASK

    if remaining > 0:
        remainders = []
        for n in nodes_to_alloc:
            share = (target_weights[n] / total_w) * remaining
            whole = int(share)
            target_counts[n] += whole
            remainders.append((share - whole, n))
        remainders.sort(key=lambda x: x[0], reverse=True)
        extra = remaining - sum(int((target_weights[n]/total_w)*remaining) for n in nodes_to_alloc)
        for i in range(extra):
            if i < len(remainders):
                target_counts[remainders[i][1]] += 1
```

Perform a Equal Distribution by setting all weights to be 1

Any resulting remainders are handled via an exception process.



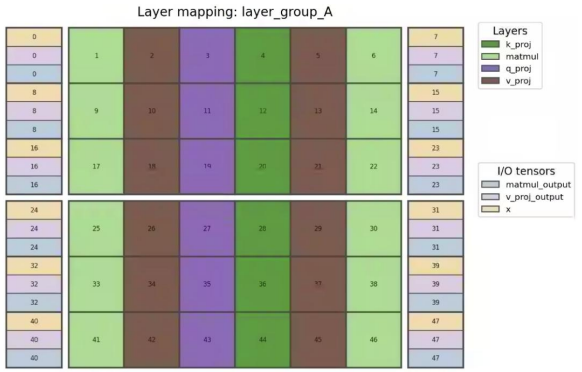
Equal Distribution for each tasks

Code Review (5/7)

- **Phase**
  - **Phase 2 (iteration 2)**

```
elif iteration == 2:
    # ============ ITERATION 2: load + comp + NAME-BASED PLACEMENT ============
    logging.info(f"   [{group_key}] Phase 1: load comp-BASED WEIGHTING")
    base_strat = 'V'
    use_name_heuristic = True

    for n in node_names:
        load_time = g_metrics.get(n, {}).get("load_t", 0.0)
        comp_time = g_metrics.get(n, {}).get("comp_t", 0.0)
        target_weights[n] = load_time + comp_time
        if target_weights[n] <= 0: target_weights[n] = 1.0

    nodes_to_alloc = node_names
    available_cores = sorted(list(set(CORE_POOL)))
    total_w = sum(target_weights.values()) or 1
    remaining = len(available_cores) - len(nodes_to_alloc) * MIN_CORES_PER_TASK
    for n in nodes_to_alloc: target_counts[n] = MIN_CORES_PER_TASK

    if remaining > 0:
        remainders = []
        for n in nodes_to_alloc:
            share = (target_weights[n] / total_w) * remaining
            whole = int(share)
            target_counts[n] += whole
            remainders.append((share - whole, n))
        remainders.sort(key=lambda x: x[0], reverse=True)
        extra = remaining - sum(int((target_weights[n]/total_w)*remaining) for n in nodes_to_alloc)
        for i in range(extra):
            if i < len(remainders):
                target_counts[remainders[i][1]] += 1
```

Perform a weighted distribution based on the load and computation times from execution_time.csv.



Weighted Distribution for each tasks

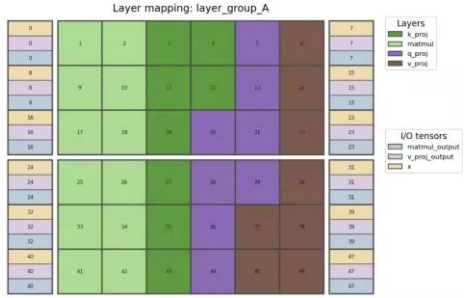
Code Review (6/7)

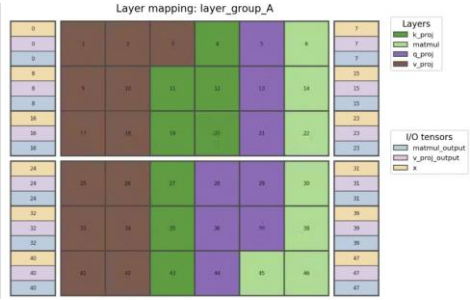
KU KONKUK UNIVERSITY

- **Phase**
  - **Phase 2 (iteration 3-4)**

```
elif iteration <= 4:
    # ============ ITERATION 3-4: GRADIENT PARTITIONING ============
    logging.info(f"   [{group_key}] Phase 2: GRADIENT PARTITIONING")
    base_strat = 'V'
    use_name_heuristic = False

    g_metrics_n1 = metrics_n1.get(group_key, {})
    g_metrics_n2 = metrics_n2.get(group_key, {})
    grp_n1 = cfg_n1.get("partitions", {}).get(group_key, {}).get("compute_nodes", {}) if cfg_n1 else {}
    grp_n2 = cfg_n2.get("partitions", {}).get(group_key, {}).get("compute_nodes", {}) if cfg_n2 else {}

    for n in node_names:
        t1 = g_metrics_n1.get(n, {}).get("time", 0)
        t2 = g_metrics_n2.get(n, {}).get("time", 0)
        c1 = len(grp_n1.get(n, {}).get("router_ids", [])) if n in grp_n1 else 1
        c2 = len(grp_n2.get(n, {}).get("router_ids", [])) if n in grp_n2 else 1

        base_weight = t1 if t1 > 0 else 1.0
        modifier = 1.0

        if t2 > 0:
            delta_t = t1 - t2
            delta_c = c1 - c2
            if delta_t < 0: modifier = 1.3 if delta_c > 0 else 0.8
            else: modifier = 0.8 if delta_c > 0 else 1.3

        target_weights[n] = base_weight * modifier

    nodes_to_alloc = node_names
    available_cores = sorted(list(set(CORE_POOL)))
    total_w = sum(target_weights.values()) or 1
    remaining = len(available_cores) - len(nodes_to_alloc) * MIN_CORES_PER_TASK
    for n in nodes_to_alloc: target_counts[n] = MIN_CORES_PER_TASK
```

Task-specific core allocation is optimized using a gradient derived from the correlation between core count and simulation time





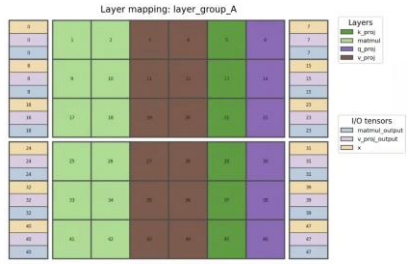
Core numbers changed for each iterations

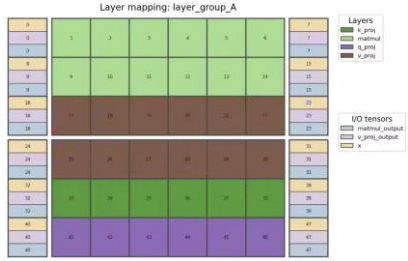
Code Review (7/7)

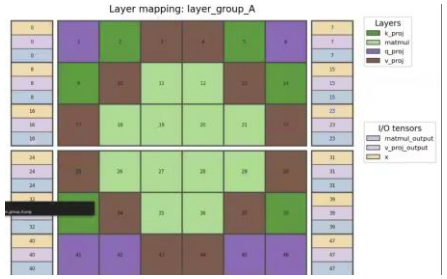
- **Phase**
  - **Phase 3 (5-7)**

Sequentially adapt
Vertical, Horizontal, Scatter

```
elif iteration <= 7:
    logging.info(f"   [{group_key}] Phase 3: PLACEMENT EXPLORATION")

    source_cfg_nodes = best_partitions_iter14.get(group_key, {}).get("compute_nodes", {}) if best_partitions_iter14 else c_nodes

    for n in node_names:
        cnt = len(source_cfg_nodes.get(n, {}).get("router_ids", [])) or MIN_CORES_PER_TASK
        target_counts[n] = cnt

    strategies = ['V', 'H', 'S']
    base_strat = strategies[(iteration - 5) % len(strategies)]

elif iteration == 8:
    logging.info(f"   [{group_key}] Phase 4a: BEST ASSEMBLY")

    source_combined = best_combined_iter17.get(group_key, {}).get("compute_nodes", {}) if best_combined_iter17 else c_nodes

    for n in node_names:
        best_rids = source_combined.get(n, {}).get("router_ids", [])
        if best_rids:
            cnt = len(best_rids)
            target_counts[n] = cnt
            set_node_config(c_nodes[n], best_rids, n, expected_type="core")
            for r in best_rids: global_used_routers.add(r)
        else:
            target_counts[n] = MIN_CORES_PER_TASK
    continue
```







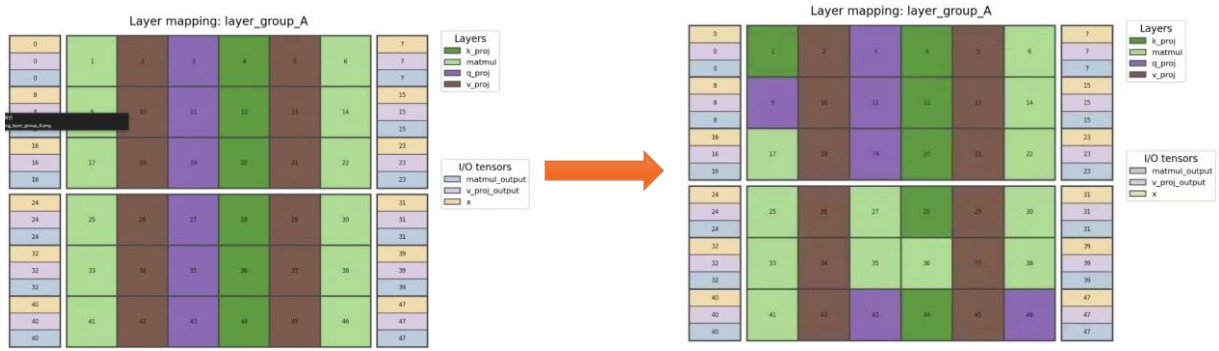
Code Review (7/7)

- **Phase**
  - **Phase 4 (8+)**

```python
for n in node_names:
    best_rids = source_combined.get(n, {}).get("router_ids", [])
    if best_rids:
        cnt = len(best_rids)
        target_counts[n] = cnt
        set_node_config(c_nodes[n], best_rids, n, expected_type="core")
        for r in best_rids: global_used_routers.add(r)
    else:
        target_counts[n] = MIN_CORES_PER_TASK

all_assigned_cores = list(global_used_routers)
num_to_swap = max(1, int(len(all_assigned_cores) * 0.1))

router_to_node = {}
for n in node_names:
    for r in c_nodes[n].get("router_ids", []):
        router_to_node[r] = n

router_effs = [(r, g_links.get(r, 0.0)) for r in all_assigned_cores]
router_effs.sort(key=lambda x: x[1])

worst_routers = [r for r, _ in router_effs[:num_to_swap]]
best_routers = [r for r, _ in router_effs[-num_to_swap:]]

swapped_count = 0
for w_router, b_router in zip(worst_routers, best_routers):
    w_node = router_to_node.get(w_router)
    b_node = router_to_node.get(b_router)

    if w_node and b_node and w_node != b_node:
        w_rids = list(c_nodes[w_node].get("router_ids", []))
        b_rids = list(c_nodes[b_node].get("router_ids", []))

        if w_router in w_rids and b_router in b_rids:
            w_rids[w_rids.index(w_router)] = b_router
            b_rids[b_rids.index(b_router)] = w_router

            set_node_config(c_nodes[w_node], w_rids, w_node, expected_type="core")
            set_node_config(c_nodes[b_node], b_rids, b_node, expected_type="core")
            swapped_count += 1
```

Swap best bw_eff node with worst bw_eff node

Evaluate node bw_eff by sorting



Swap cores (10% of total cores)

Code Review (7/7)

KU KONKUK UNIVERSITY

- **Phase**
  - **Phase 4 (8+)**

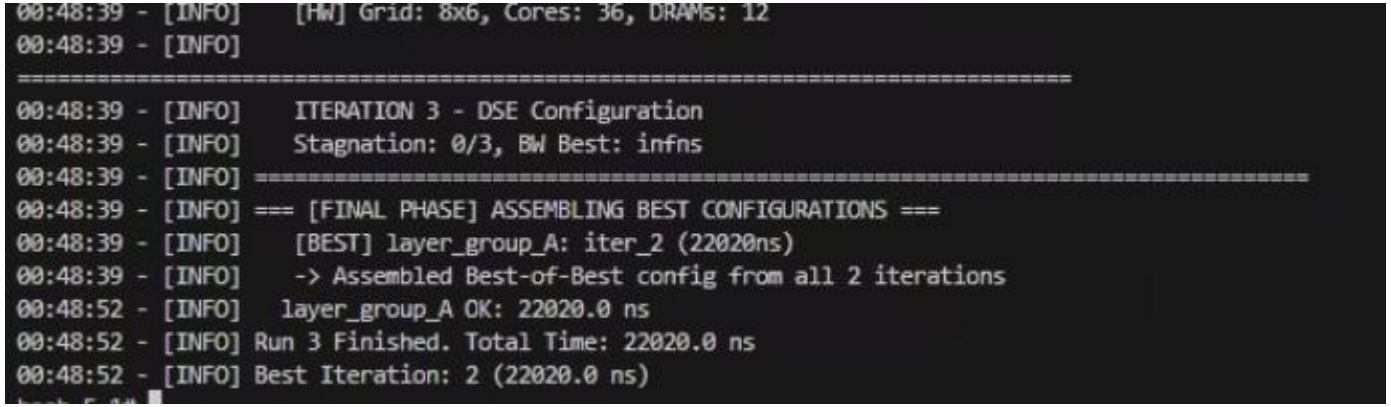
```python
# ======================================================================
# ADAPTIVE RE-EXPLORATION: DELTA-BASED
# ======================================================================
if iteration >= 9 and stagnation_count >= 3:
    if reexplore_start_iter is None:
        reexplore_start_iter = iteration
        reexplore_count += 1
        meta["reexplore_start_iter"] = reexplore_start_iter
        meta["reexplore_count"] = reexplore_count
        logging.info(f"   [{group_key}] !!! STAGNATION #{reexplore_count} - DELTA-BASED RE-EXPLORATION !!!")

    reexplore_phase = iteration - reexplore_start_iter
    is_reverse = (reexplore_count % 2 == 1)

    # Get Iter1/Iter2 counts
    iter1_grp = iter1_counts.get(group_key, {})
    iter2_grp = iter2_counts.get(group_key, {})

    if not iter1_grp or not iter2_grp:
        logging.warning(f"   [{group_key}] No baseline counts - fallback to equal")
        for n in node_names: target_counts[n] = len(CORE_POOL) // len(node_names)
    else:
        # ★ Delta-based allocation
        for n in node_names:
            c1 = iter1_grp.get(n, MIN_CORES_PER_TASK)
            c2 = iter2_grp.get(n, MIN_CORES_PER_TASK)
            delta = c2 - c1

            if is_reverse:
                # REVERSE: 늘어난 만큼 빼고, 줄어든 만큼 늘림
                target_counts[n] = max(MIN_CORES_PER_TASK, c1 - delta)
            else:
                # AGGRESSIVE: 늘어난 만큼 더 늘리고, 줄어든 만큼 더 줄임
                target_counts[n] = max(MIN_CORES_PER_TASK, c1 + delta)

        # Renormalize to total cores
        total_assigned = sum(target_counts.values())
        total_cores = len(CORE_POOL)
        if total_assigned != total_cores:
            scale = total_cores / total_assigned if total_assigned > 0 else 1
            for n in node_names:
                target_counts[n] = max(MIN_CORES_PER_TASK, int(target_counts[n] * scale))
```

Re-Explore with adversial/aggressive Gradient

Result of TP (1/5)

KU KONKUK UNIVERSITY

- **Optimality**



- **Optimal Result_(iteration 2)**

- Time_ns : **22020.0**
- Energy_mj: **0.057796**

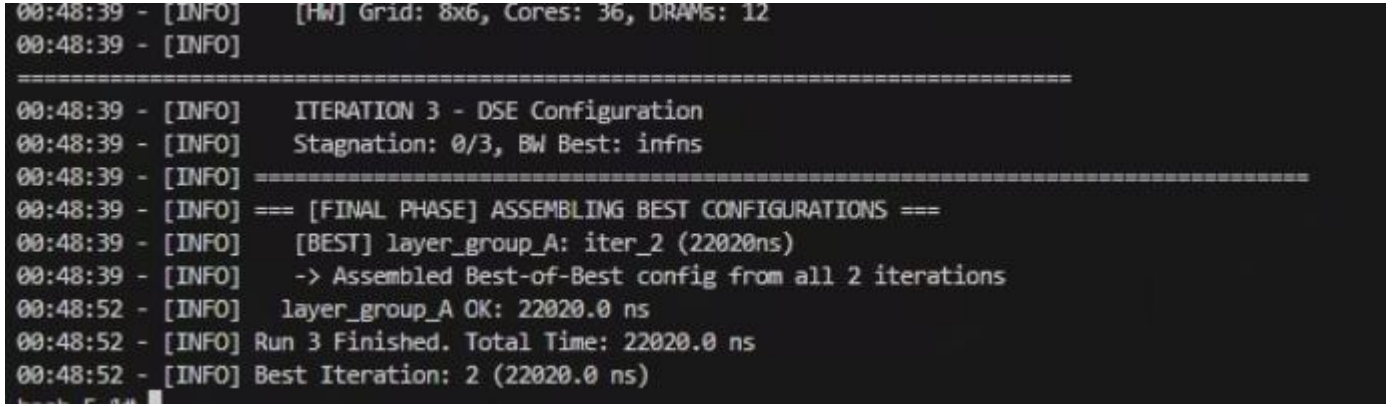
- Total_throughput_ops": **48762117350000.0,**
- power_efficiency_ops_per_w": **18578178942132.86,**

Note on Iteration Input:
The input iteration count must be set to **3 or higher**.
While our simulation identifies Iteration 2 as the best performing point,
the submitted code is designed to re-calculate
the optimal combination for the integrated simulation of Layers
A, B, and C at the final stage.
This is an intentional design choice to find the **Global Optimum** of the entire system,
We would appreciate it if you could take this architectural behavior into account during evaluation.

Result of TP (2/5)

KU KONKUK UNIVERSITY

- **Sampling efficiency**



- **Optimal Result_(iteration 2)**

- Time_ns : **22020.0**
- Energy_mj: **0.057796**

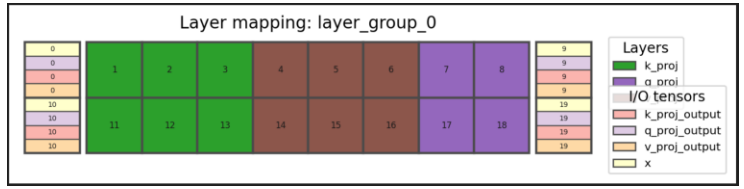
- Total_throughput_ops": **48762117350000.0,**
- power_efficiency_ops_per_w": **18578178942132.86,**

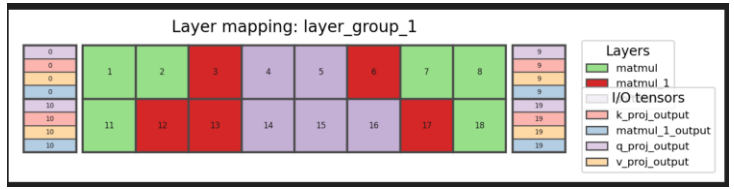
Note on Iteration Input:
The input iteration count must be set to **3 or higher**.
While our simulation identifies Iteration 2 as the best performing point,
the submitted code is designed to re-calculate
the optimal combination for the integrated simulation of Layers
A, B, and C at the final stage.
This is an intentional design choice to find the **Global Optimum** of the entire system,
We would appreciate it if you could take this architectural behavior into account during evaluation.

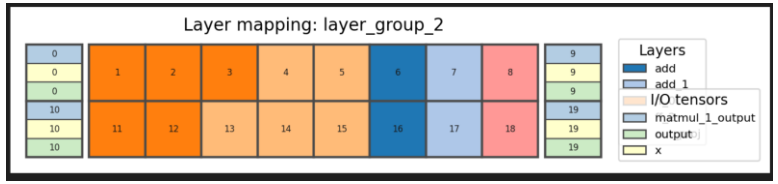
Result of TP (4/5)

- **General applicability (1/2)**
  - Small + 16core 4dram


```
{
  "data": {
    "summary": {
      "total_groups": 3,
      "total_simulation_time_ns": 71010.0,
      "total_energy_mj": 0.057678,
      "total_throughput_ops": 15095608618666.666,
      "power_efficiency_ops_per_w": 18319599228895.566,
      "total_area_mm2": 187.964048,
      "max_cost_usd": 63.056549
    },
    "groups": [
      {
```





  - **Optimal Result_(iteration 2)**

- Time_ns : **71010.0**
- Energy_mj: **0.057678**

External Module Import

- **General applicability (2/2)**
  - **We additionally used below..**

```
from collections import defaultdict
import csv
import math
import logging
import sys
import orjson
import re
from pathlib import Path
```