Resolving Placement–Residency Conflicts in Multi-GPU Task-Graph Runtimes

Long Cheng
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
chenglo@student.chalmers.se

Yicheng Li
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
yicheng@student.chalmers.se

Jing Chen
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
chjing@chalmers.se

Miquel Pericàs
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
miquelp@chalmers.se

***Abstract*—In multi-GPU task-graph runtimes, task placement determines both execution time and data locality. Earliest-finish-time (EFT) placement already accounts for the communication required by the current task, but it does not explicitly consider how a placement changes data residency for later dependent tasks. As a result, a placement that minimizes the current task's finish time may increase the communication required by subsequent tasks. We observe that some such placement–residency conflicts expose model-level slack: a lower-copy alternative can finish slightly later than the EFT choice without extending the modeled completion of work already committed to the devices.**

We propose a communication-aware task placement scheduler for multi-GPU task-graph runtimes. For each ready task, the scheduler evaluates lower-copy alternative placements and admits one only when it preserves modeled prefix completion relative to the EFT choice; otherwise, it retains that choice. We prototype the proposed scheduler in the CUDASTF runtime and evaluate it on five representative task-graph workloads running on an eight-GPU NVIDIA L4 platform. On the two workloads where the scheduler changes placement, the proposed method reduces modeled input-copy volume by 39–81% and improves performance by 1.18–1.31× over the EFT baseline, while retaining the baseline placement when no admissible locality improvement exists.

***Index Terms*—Multi-GPU, task graphs, HEFT, data locality, runtime scheduling, copy traffic, performance**

I. Introduction

In a multi-GPU task-graph runtime, the scheduler repeatedly assigns ready tasks to devices, after which the runtime performs the data movement required by those placement decisions. These two responsibilities are tightly coupled: a placement decision determines not only when the current task can execute, but also where its data will be available for subsequent tasks. Effective placement must therefore account for both device availability and data locality. A widely used strategy is earliest-finish-time (EFT) placement, as employed by HEFT [1]. For each ready task, HEFT estimates its finish time on every candidate device using device readiness, input-data availability, communication cost, and predicted execution time, and then selects the device with the earliest finish. This HEFT-style EFT rule provides a performance baseline that already accounts for the current task's communication.

Communication, however, affects execution beyond the current placement decision. Executing a task on a device may change where the updated versions of its logical data objects reside. As a result, a placement that minimizes the current task's earliest finish time may leave the runtime in a data-residency state that increases the communication required by later dependent tasks. We refer to this general interaction as a placement–residency conflict.

A simple migrate–return pattern illustrates the conflict. Consider a mutable object currently resident on one GPU. A read–write task may finish slightly earlier on another GPU because that GPU becomes available sooner, even after accounting for the time required to migrate the object. Once the task completes, the updated object resides on the selected GPU. If a later dependent task executes on the original GPU, the updated object must be transferred back. The first placement is locally optimal under the current earliest-finish estimate, yet it induces additional successor communication. Repeated instances of this pattern can amplify a small local placement advantage into a larger data-movement and readiness penalty along a dependency chain.

The key observation is that some placement–residency conflicts expose slack beneath the modeled completion of already committed work. A lower-copy alternative can finish the current task later than the HEFT choice while other GPUs remain busy with previously scheduled tasks. Exploiting this slack preserves the modeled prefix completion; it does not guarantee non-regression in full-DAG makespan or measured latency.

Based on this observation, we propose a communication-aware scheduler for online multi-GPU task-graph runtimes that evaluates placement alternatives using both predicted finish time and modeled data movement. The scheduler first computes the EFT baseline placement and then evaluates

SC26 Workshops, November 15-20, 2026, Chicago, Illinois, USA
979-8-3195-1221-5/26/$31.00 ©2026 IEEE

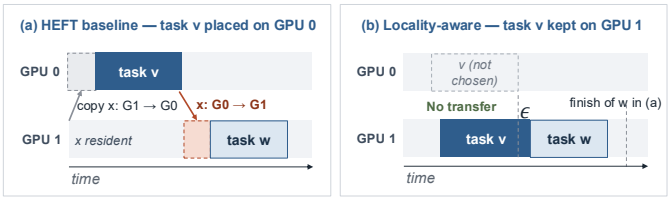


Fig. 1. Placement–residency conflict: a two-GPU example.

alternative placements that reduce the modeled data movement for the current task. An alternative is admitted only if it preserves modeled prefix completion relative to the EFT choice. Admitted alternatives are ranked by normalized copy gain plus the weighted positive part of normalized queue-delay gain. Otherwise, it retains the EFT baseline placement. We refer to this admission rule as prefix-safe because it guarantees that replacing the baseline decision does not extend the modeled completion of the execution prefix that has already been committed.

In summary, this work makes the following contributions:

- We identify a placement–residency conflict in multi-GPU task-graph runtimes, where optimizing the current task's completion time can increase the communication experienced by subsequent tasks because placement changes data residency.
- We propose a communication-aware scheduler that jointly considers predicted task finish time and modeled data movement. The scheduler admits lower-communication placements under an explicit bound on modeled prefix completion.
- We prototype the proposed scheduler in CUDASTF [2] runtime and evaluate the proposed scheduler on an eight-GPU NVIDIA L4 platform using five representative task-graph workloads. The evaluation demonstrates a 39–81% reduction in modeled data movement volume and a 1.18–1.31× speedup over HEFT on workloads where placement changes, while showing that the benefits of prefix admission are workload- and runtime-dependent.

II. Background and Motivation

A. Runtime Model

We consider a task-graph runtime that schedules a directed acyclic graph (DAG) $G = (V, E)$ onto a set of GPUs D. Each vertex $v \in V$ represents a task, and each edge denotes a data dependency. For every task–device pair (v, d), the runtime maintains a predicted execution time $p_v(d) > 0$. Tasks access data through logical objects. Each object x has size $\text{bytes}(x)$, and the runtime estimates its transfer time as

$$c_x = \frac{\text{bytes}(x)}{B},$$

where B is the effective bandwidth assumed by the runtime's transfer-cost model.

Our prototype is implemented in CUDASTF, a task-graph runtime that manages dependencies through logical data objects rather than explicit memory operations. To track the location and validity of these logical objects across multiple GPUs, CUDASTF maintains software-managed Modified/Shared/Invalid (MSI) residency metadata. We adopt the same abstraction in the runtime model. $M(x, d) \in \{M, S, I\}$ records the state of object x on device d, where M and S indicate that d holds a valid instance of the current version, whereas I indicates that the local instance is invalid. If a task requires x on a device where $M(x, d) = I$, the runtime must obtain the current version from a device that holds a valid instance before the task can execute. After a write-like access to x on device d, the newly produced version becomes Modified on d, and conflicting instances on the remaining devices become Invalid.

B. HEFT

Our HEFT baseline (H) is CUDASTF's EFT selector on runtime-supplied task order; canonical upward-rank ordering is not recomputed. HEFT evaluates each candidate device using its predicted task finish time. For a task v, this finish time depends on the current device-ready time $L(d)$ maintained by the runtime, and the input-availability time $A(v, d)$, which denotes the latest time at which all read and read–write inputs of v are available on device d. If a required input is not locally available, $A(v, d)$ includes the time required to transfer the current version to d. The placement decision is therefore based on two forms of runtime state: $L(d)$ captures device availability, while $A(v, d)$ captures the data readiness. HEFT places task v on the device with the earliest modeled finish time:

$$F(v, d) = \max\{L(d), A(v, d)\} + p_v(d),$$
$$b = \arg\min_{d \in D} F(v, d),$$

where b denotes the device selected by HEFT.

C. Motivation

The HEFT baseline minimizes the modeled finish time of the current task, but it does not explicitly account for how the placement changes data residency for later tasks. After a write-like access, the updated version of a data object resides on the selected device, while conflicting copies on the remaining devices become invalid. Consequently, a placement that is locally optimal for the current task may increase the communication required by its successors. We refer to this interaction as a placement–residency conflict.

Figure 1 illustrates this conflict using a two-GPU example. The current version of a large object x resides on GPU 1, and a read–write task v updates x. In Fig. 1(a), the HEFT baseline places v on GPU 0 because its modeled finish time is ϵ shorter. After v completes, however, the updated version of x resides on GPU 0 and the copy on GPU 1 becomes invalid. When the dependent task w later executes on GPU 1, the runtime must first transfer the updated object back, delaying both w and the remaining execution.

Figure 1(b) shows a locality-aware alternative. Keeping v on GPU 1 delays its completion by ϵ, but the updated version of x remains where w executes. The return transfer is therefore avoided, allowing w to begin immediately and

the remaining execution to progress earlier despite the slightly later completion of v.

This example shows that minimizing the current task's modeled finish time does not necessarily minimize the communication experienced by the remaining execution. The alternative may extend the committed prefix and fail our admission rule (Section III-B); the figure illustrates the conflict itself.

III. Scheduler Design

A. Overview

Given a ready task v and the current runtime state, the scheduler selects one device from the candidate set D. It first computes a performance-oriented baseline placement using the earliest-finish-time rule and then considers alternatives that reduce modeled data movement. A candidate is selected only after passing two tests: the prefix gate bounds its effect on the completion of already committed work, and the copy gate requires a sufficient reduction in modeled communication. Candidates that pass both gates are ranked by a secondary score. If none qualifies, the scheduler retains the baseline placement.

For each candidate device $d \in D$, the scheduler evaluates the predicted finish time $F(v, d)$, device-ready time $L(d)$, input-availability time $A(v, d)$, modeled communication cost $K(d)$, and modeled queue delay $Q(d)$. The communication cost $K(d)$ is the sum of the modeled transfer times for the read and read–write inputs of v that are not already valid on d. These transfers contribute to $A(v, d)$, which records when the last required input becomes available on d. Thus, $K(d)$ quantifies the communication associated with a placement, whereas $A(v, d)$ captures when that communication and the task dependencies permit execution to begin. The queue delay is

$$Q(d) = \max\{0,\ L(d) - A(v, d)\},$$

and measures how long the task must wait for the device after all of its inputs are available.

B. Prefix Admission

The baseline device is selected by the earliest-finish-time rule:

$$b = \arg\min_{d \in D} F(v, d).$$

The prefix gate bounds the modeled completion of committed work when considering a non-baseline placement. Let

$$R = \max_{d \in D} L(d)$$

denote the current device-ready frontier. For any candidate device d, define the modeled completion of the current scheduling prefix as

$$P(d) = \max\{R, F(v, d)\}.$$

The maximum ensures that placements completing before the current ready frontier do not reduce the modeled completion of the committed prefix. In particular, $P(b)$ is the prefix completion under the baseline placement. A candidate $d \neq b$ is prefix-admissible when

$$P(d) \leq P(b) + \varepsilon_p,$$

where ε_p is a numerical tolerance.

A later per-task finish can remain admissible when both placements lie beneath the same ready frontier R. For example, an independently busy GPU can set $R = 100$ while $F(v, b) = 60$ and $F(v, d) = 80$ (arbitrary time units), giving $P(b) = P(d) = 100$. Candidates whose modeled prefix completion exceeds $P(b) + \varepsilon_p$ are rejected despite any communication saving.

The gate separates the modeled prefix constraint from locality ranking. The copy gate below then checks the communication benefit of the remaining candidates.

C. Copy Gate and Ranking

Candidates that pass the prefix gate are next evaluated according to their communication benefit. The scheduler compares each prefix-admissible candidate against the baseline placement b and considers only the modeled communication required to make the current task's inputs available on each candidate device. For a candidate device d, the normalized copy gain is

$$G_K(d) = \frac{K(b) - K(d)}{\max\{K_{\min}, K(b)\}}.$$

Here $K_{\min}$ is a small positive constant that floors the denominator, keeping the gain well-defined and bounded when the baseline copy time $K(b)$ is negligible. A positive $G_K(d)$ indicates that placing the task on d reduces the modeled communication relative to the baseline, while a negative value indicates additional communication. Candidates that do not reduce modeled communication are discarded immediately.

Not every copy reduction justifies deviating from the earliest-finish-time baseline. Small savings may simply reflect modeling noise or insignificant transfers. The scheduler therefore requires the normalized copy gain to exceed a task-dependent threshold,

$$G_K(d) \geq \tau(v).$$

Only candidates satisfying this condition remain under consideration.

The threshold is adapted according to the estimated difficulty of the current task. Intuitively, larger or more expensive tasks should require a more substantial communication benefit before the scheduler departs from the earliest-finish-time baseline. We therefore estimate a task-difficulty score from three observable characteristics:

- the predicted execution time p_v,
- the number of task dependencies n_v, and
- the total bytes B_v of the task's read and read–write inputs.

Because these quantities have different units and ranges, each is first normalized to the interval $[0, 1]$. We use the clipping function

$$\mathrm{clip}(z) = \min\{1, \max\{0, z\}\},$$

procedure PLACEMENT(task v, candidates D)
  $b \leftarrow \arg\min_{d \in D} F(v, d)$
  $R \leftarrow \max_{d \in D} L(d)$
  $P(b) \leftarrow \max\{R, F(v, b)\}$
  admitted $\leftarrow \varnothing$
  for all $d \in D$ with $d \neq b$ **do**
    $P(d) \leftarrow \max\{R, F(v, d)\}$
    if $P(d) > P(b) + \varepsilon_p$ **then**
      continue ▷ prefix gate
    end if
    $\Delta_K \leftarrow K(b) - K(d)$
    $g_K \leftarrow \Delta_K / \max\{K_{\min}, K(b)\}$
    if $\Delta_K \leq 0$ **or** $g_K < \tau(v)$ **then**
      continue ▷ copy gate
    end if
    $g_Q \leftarrow \big(Q(b) - Q(d)\big) / \max\{Q_{\min}, Q(b)\}$
    $S(d) \leftarrow g_K + \lambda_Q \max\{0, g_Q\}$
    admitted $\leftarrow$ *admitted* $\cup \{(d, S(d))\}$
  end for
  if *admitted* $= \varnothing$ **then**
    return b
  end if
  return the d with largest $S(d)$; break score ties by lower $F(v, d)$
end procedure

Fig. 2. Prefix-safe communication-aware placement algorithm.

which truncates values smaller than 0 to 0 and values larger than 1 to 1. This prevents unusually large tasks from dominating the difficulty score. The normalized execution-time, dependency-count, and input-size components are

$$c_p = \mathrm{clip}\left(\frac{\ln(1 + p_v)}{\ln(1 + P_{\max})}\right), \quad c_n = \mathrm{clip}\left(\frac{n_v}{N_{\max}}\right),$$
$$c_B = \mathrm{clip}\left(\frac{\ln(1 + B_v)}{\ln(1 + B_{\max})}\right),$$

where $P_{\max}$, $N_{\max}$, and $B_{\max}$ are fixed normalization bounds. These three normalized components are then combined into a single task-difficulty score,

$$c(v) = \mathrm{clip}\left(w_p c_p + w_n c_n + w_B c_B\right),$$

where w_p, w_n, and w_B are fixed weighting coefficients satisfying $w_p + w_n + w_B = 1$. The weighting coefficients assign the greatest influence to predicted execution time, followed by dependency count and input size.

Finally, the scheduler selects the copy-gain threshold according to the estimated task difficulty. Tasks whose difficulty score exceeds the decision threshold c_{split} use the stricter copy-gain threshold τ_{high}, whereas the remaining tasks use τ_{low}.

Candidates that satisfy both the prefix and copy gates are admitted to the final ranking stage. If multiple candidates are admitted, the scheduler ranks them using a secondary score,

$$S(d) = G_K(d) + \lambda_Q \max\{0, G_Q(d)\},$$

where

$$G_Q(d) = \frac{Q(b) - Q(d)}{\max\{Q_{\min}, Q(b)\}}$$

is the normalized gain in the modeled queue delay $Q(d)$ relative to the baseline placement. As with $K_{\min}$, the constant $Q_{\min}$ floors the denominator so that the gain remains well-defined when the baseline queue delay $Q(b)$ is negligible. Unlike the copy gain, the queue-delay gain is not used to admit or reject candidates. Instead, it serves only as a secondary preference among already admitted candidates. This favors placements that reduce modeled communication while avoiding unnecessary queueing delay whenever possible. The scheduler finally selects the candidate with the highest score $S(d)$, breaking score ties by lower predicted finish time. If no candidate passes both gates, the scheduler retains the baseline placement.

Figure 2 summarizes the placement procedure. After selecting a device, the runtime updates device-ready times and logical-data residency before scheduling the next ready task. The normalization bounds, weighting coefficients, and thresholds are fixed throughout the evaluation and are summarized in Table I in Section IV.

IV. Experimental Methodology

A. Platform and Parameters

Experiments were conducted on a single dual-NUMA node with eight NVIDIA L4 GPUs (23,034 MiB each), using driver 595.71.05, CUDA 12.4.131, GCC 13.3.0, and CUDASTF/CCCL 3.2.0.0 [3] in a C++17 Release build. GPUs 0–3 and 4–7 form two NUMA-local PCIe groups, and cross-group transfers traverse the system interconnect.

Task execution costs come from a separate HEFT calibration prepass on GPU 0, shared across the eight identical L4s. The five prepasses cover 60 task keys, each identified by its task symbol and summed dependency footprint. The runtime reserves ten observations per key, discards reservation 0, and stores the median of reservations 1–9. Each workload's table is frozen for its 36 primary executions, with online calibration disabled; calibration is outside the benchmark intervals. Median/MAD quality gates check calibration repeatability, not prediction accuracy.

The transfer model uses $B = 20.7$ GB/s, rounded from the 20.726 GB/s median across all 56 directed GPU pairs. Each pair copies 256 MiB twice for warm-up and 20 times under CUDA-event timing. The normalization bounds and heuristic policy parameters in Table I remain fixed across workloads.

B. Benchmarks

Our evaluation spans five multi-GPU workloads (Table II) with distinct computation and communication patterns. **Cholesky** performs blocked dense Cholesky factorization using cuBLAS and cuSOLVER routines [4], [5], and **LU** performs tiled, no-pivot dense factorization built on the same libraries. Finite-difference time-domain (**FDTD**) [6] simulates electromagnetic wave propagation through 3D stencil updates, and conjugate gradient (**CG**) [7] solves large sparse linear systems in CSR format using sparse matrix–vector products, dot products, and vector operations. **miniWeather** [8] models atmospheric dynamics on a 2D domain with heterogeneous domain decomposition, producing an irregular workload distribution and asymmetric communication patterns.

TABLE I
PARAMETERS FOR THE PRIMARY MATRIX. ONLY THE BANDWIDTH B VARIES IN THE SENSITIVITY STUDY.

Category	Parameter	Value	Unit	Purpose
Cost model	B	20.7	GB/s	Converts transferred bytes into modeled input-copy time.
Normalization	$P_{\max}$	500	ms	Normalizes execution time so task difficulty is comparable across tasks.
	$N_{\max}$	16	dependencies	Normalizes dependency count so task difficulty is comparable across tasks.
	$B_{\max}$	1	GiB	Normalizes input size so task difficulty is comparable across tasks.
Scheduling policy	w_p	0.55	–	Emphasizes execution time when estimating task difficulty.
	w_n	0.30	–	Accounts for dependency complexity when estimating task difficulty.
	w_B	0.15	–	Accounts for input-data volume when estimating task difficulty.
	c_{split}	0.60	–	Separates high- and low-difficulty tasks.
	τ_{high}	0.25	–	Minimum copy gain required before relocating a high-difficulty task.
	τ_{low}	0.15	–	Minimum copy gain required before relocating a low-difficulty task.
	$K_{\min}$	0.001	ms	Prevents division by near-zero when normalizing copy gain.
	$Q_{\min}$	0.001	ms	Prevents division by near-zero when normalizing queue gain.
	λ_Q	0.35	–	Controls how strongly queue delay influences candidate ranking.
	ε_p	10^{-6}	ms	Absorbs floating-point error in the prefix-admission test.

TABLE II
BENCHMARK CONFIGURATIONS.

Workload	Configuration	Tasks
FDTD	672 steps; 64^3 grid	4,704
Cholesky	n = 28,672, tile = 256 (112^2 tiles)	240,464
LU	n = 23,552, tile = 256 (92^2 tiles)	263,810
miniWeather	1,536 steps; 2048×1536 grid	55,296
CG	$n = 2^{24}$, 40 repeats, at most 10 iterations	3,040

TABLE III
SCHEDULER VARIANTS EVALUATED IN THIS STUDY.

Variant	Scheduling policy	Experimental role
H	Runtime EFT placement	Performance-oriented baseline
I	EFT-equivalent weighted placement	Implementation control reproducing H exactly
W	Weighted EFT + modeled input-copy cost over all devices	Communication-aware baseline
O	Weighted EFT + modeled input-copy cost over the EFT device and Modified owner	Restricted communication-aware baseline
P	Prefix admission + communication-aware placement + queue-aware ranking	Proposed scheduler
N	P with prefix admission disabled	No-prefix ablation

C. Schedulers and Protocol

Table III summarizes the six variants. The runtime-default **H** applies EFT device selection in runtime-supplied task order, without HEFT's offline upward-rank computation. Variant **I** uses the weighted-selection framework with $\lambda = 0$, reproducing H's placements and modeled input-copy cost. It provides a framework-matched timing control for P. Variants **W** and **O** use $J(d) = F(v, d) + \lambda K(d)$, with $\lambda = 1$. W minimizes J over all devices. For tasks with exactly one read–write input and a unique Modified owner, O compares that owner with the EFT device; otherwise it retains EFT. Variant **P** combines prefix admission, communication-aware candidate selection, and queue-aware ranking to reduce modeled communication subject to the modeled prefix bound. Variant **N** is its no-prefix ablation, identical to P except that the prefix-admission rule is disabled. Together, these variants define three comparisons: P against H measures the benefit of the proposed policy over the runtime default, with I providing a framework-matched timing control; P against W and O evaluates it relative to simpler communication-aware baselines; and P against N isolates the contribution of prefix admission.

Each context's iteration or factorization graph is fully materialized before deferred submission; placement decisions are sequential and irrevocable. CG builds and executes 40 contexts successively.

Benchmark intervals include graph construction, scheduling, transfers, and GPU execution; CPU correctness checks are outside them. FDTD uses a host steady clock and miniWeather the system clock, both ending after `ctx.finalize()`, including write-back and cleanup. CG uses a host steady clock around all 40 contexts, including their setup and finalization and the final solution read-back; CSR generation is excluded. Cholesky and LU use CUDA events from before graph construction through completion of all submitted tasks, excluding final write-back and finalization. Initialization is outside the FDTD, miniWeather, Cholesky, and LU intervals. For each workload, the six schedulers are executed under six Williams-balanced execution sequences [9], yielding 36 timed executions per workload and 180 across the five workloads. Each scheduler occupies every execution position once and follows every other scheduler once, balancing position and first-order carryover effects. Each planned invocation runs in a fresh process and reports one benchmark interval.

Absolute intervals are summarized by the median and minimum–maximum of six observations per workload–scheduler group. With $T_{X,r}$ denoting X's interval in block r, the paired speedup of P over X is $T_{X,r}/T_{P,r}$; values above one favor P. We report its geometric mean and observed range over the six blocks. To separate genuine changes from run-to-run noise, a speedup of at least 1.03× is treated as a material improvement, 0.97–1.03× as neutral, and below 0.97× as a material regression; these predeclared thresholds are descriptive

TABLE IV
PERFORMANCE COMPARISONS BETWEEN THE PROPOSED SCHEDULER (P) AND OTHER SCHEDULER VARIANTS. EACH ENTRY IS THE SPEEDUP OF P OVER THE NAMED SCHEDULER; VALUES ABOVE ONE FAVOR P. PARENTHESIZED ENTRIES ARE IDENTICAL-PLACEMENT TIMING CONTROLS. AMONG COMPARISONS THAT CHANGE PLACEMENT, **BOLD** ENTRIES FALL OUTSIDE THE PRACTICAL-EFFECT RANGE 0.97–1.03×.

	Speedup of P over				Modeled input-copy (GB) ↓					Reduction
Workload	H	W	O	N	H	W	O	P	N	P/H
FDTD	(1.022×)	(1.057×)	(0.942×)	(1.002×)	0.00	0.00	0.00	0.00	0.00	0.0%
CG	(0.997×)	**1.841**×	(1.002×)	(0.998×)	42.95	42.95	42.95	42.95	42.95	0.0%
Cholesky	**1.313**×	**1.239**×	**1.248**×	**0.861**×	128.55	116.20	113.46	24.41	5.90	81.0%
LU	**1.177**×	1.020×	**1.055**×	**1.210**×	148.77	96.28	96.14	90.98	3.88	38.8%
miniWeather	(1.000×)	(0.997×)	(1.008×)	(1.004×)	0.20	0.20	0.20	0.20	0.20	0.0%

engineering criteria rather than statistical significance tests.

D. Successor-Aware Diagnostic

The successor-aware locality certificate (SALC) is an offline diagnostic, not part of P. For direct successors fixed to devices, it compares modeled successor-transfer savings against the producer's immediate finish-time penalty. Section V-B uses it to expose a migrate–return opportunity rejected by prefix admission.

For a producer v, let U be the set of distinct pairs (x, a) in which x is a write-like output of v read by a successor anchored to device a. The quantity $\Phi_{\text{succ}}(d)$ estimates the modeled one-hop communication required by the fixed successors when the producer is placed on device d:

$$\Phi_{\text{succ}}(d) = \sum_{(x,a)\in U} \mathbf{1}[d \neq a] \frac{\text{bytes}(x)}{B}.$$

SALC compares the modeled future communication benefit with the producer's immediate finish-time penalty. Relative to the EFT baseline device b, the avoided future transfer is $G_{\text{future}}(d) = \Phi_{\text{succ}}(b) - \Phi_{\text{succ}}(d)$ and the immediate penalty is $\Delta_F(d) = \max\{0, F(v,d) - F(v,b)\}$; the certificate combines them as

$$C_{\text{SALC}}(d) = G_{\text{future}}(d) - \Delta_F(d).$$

An alternative is reported as a positive diagnostic opportunity when $C_{\text{SALC}}(d) > 0$; SALC then selects the largest-certificate candidate, breaking ties by lower predicted finish time.

V. EVALUATION

This section evaluates the proposed scheduler on the five-workload counterbalanced matrix, validates the relationship between modeled input-copy volume and runtime-issued peer transfers, examines sensitivity to the copy-bandwidth model, and quantifies scheduler overhead.

A. Primary Results

Table IV reports the cross-variant comparisons, while Figure 3 shows the six paired P/H blocks and modeled-copy changes. Table V adds the absolute benchmark intervals. Speedups use the geometric mean of paired ratios, not ratios of the displayed medians.

Relative to the runtime EFT baseline H, the proposed scheduler (P) changes placement on Cholesky and LU, reducing modeled input-copy volume by 81.0% and 38.8%, respectively. P selects a device other than the EFT candidate evaluated in its current state for 15,475 of 240,464 Cholesky tasks (6.44%) and 129,511 of 263,810 LU tasks (49.09%). These counts are identical across the six blocks. On FDTD, CG, and miniWeather, P preserves H's placement decisions and modeled input-copy volume. These identical-placement cases provide timing controls, while Cholesky and LU expose the effect of the proposed placement policy.

TABLE V
ABSOLUTE BENCHMARK INTERVALS IN THE SIX PRIMARY BLOCKS (MS): MEDIAN [MINIMUM–MAXIMUM]. TIMING BOUNDARIES FOLLOW SECTION IV-C.

Workload	H	P
FDTD	292.5 [292.0–420.0]	309.0 [253.0–366.0]
CG	8,025.0 [7,993.0–8,101.0]	8,077.5 [8,011.0–8,102.0]
Cholesky	18,332.3 [18,252.3–18,527.8]	13,969.1 [13,929.3–14,086.7]
LU	23,134.8 [23,048.5–23,265.8]	19,692.3 [19,430.3–19,760.0]
miniWeather	15,401.5 [15,377.0–15,702.0]	15,396.0 [15,364.0–15,715.0]

On Cholesky, the placement change reduces modeled input-copy volume from 128.55 to 24.41 GB (81.0%) and yields a 1.313× speedup over H. For LU, it reduces the volume from 148.77 to 90.98 GB (38.8%) and improves latency by 1.177×. Both effects hold across all six paired blocks (ranges 1.296–1.324× and 1.171–1.186×). Variant I—omitted from Table IV because it reproduces H's placement and modeled input-copy volume exactly—is the matched implementation control, running the same selector path as P. Measured against I, the speedups are 1.269× (1.258–1.273×) on Cholesky and 1.156× (1.148–1.171×) on LU. We therefore report P/H as the runtime-default comparison and P/I as the more controlled estimate of the placement-policy effect.

W and O are secondary communication-aware baselines. On Cholesky, P is faster than both (1.239× over W, 1.248× over O) and models less input-copy volume (24.41 GB versus 116.20 and 113.46 GB). On LU, P is faster than O (1.055×) and neutral relative to W (1.020×, within the 0.97–1.03× range). On CG, W selects a different placement but leaves modeled

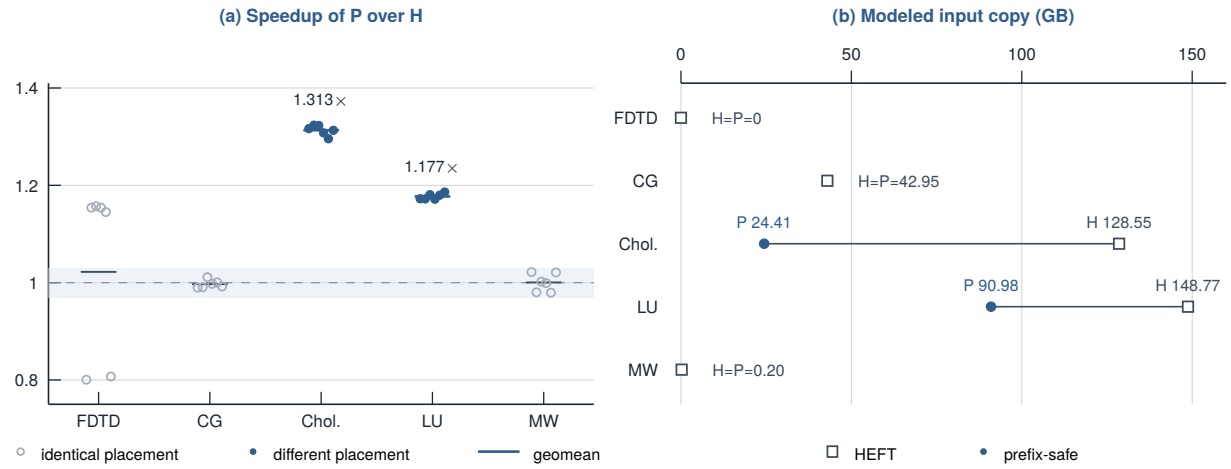


Fig. 3. Primary P/H evidence from the Williams-balanced matrix. (a) Six within-block paired ratios and their geometric means; blue points change placement, gray points are placement-identical timing controls, and the shaded band denotes 0.97–1.03×. (b) Median scheduler-modeled input-copy volume. Modeled copy is not physical traffic.

input-copy volume unchanged (42.95 GB), while P is 1.841× faster than W. This large regression shows that the internal weighted comparator can perturb placement without obtaining its intended copy benefit. We do not interpret it as evidence that P dominates external locality-aware schedulers.

For comparisons with identical placement digests and modeled input-copy volumes, the latency ratios measure scheduler-path and system variation only. FDTD and miniWeather produce identical placement digests and modeled input-copy volumes across all six scheduler variants, whereas on CG only W changes placement.

FDTD illustrates the variation within these controls. Its paired (T_H, T_P) intervals in block order are $(293, 366)$, $(292, 253)$, $(420, 363)$, $(292, 253)$, $(292, 255)$, and $(293, 363)$ ms. P has three observations at 253–255 ms and three at 363–366 ms; H also slows to 420 ms in block 3. All 36 FDTD runs agree in placement digest and correctness checksum, with zero modeled input-copy volume. Measured scheduling costs rise in the slower runs, but the saved logs lack frequency telemetry and OS traces that could establish the cause. We therefore retain these observations as timing controls without attributing their ratios to locality gains.

The P–N comparison exposes both the conservative and protective effects of prefix admission. On Cholesky, disabling the gate (N) admits a more aggressive placement that further lowers modeled input-copy volume (to 5.90 GB) and improves latency, so that N is 1.161× faster than P. On LU, disabling the gate instead yields a placement with lower modeled input-copy volume (3.88 GB) but substantially worse latency: P is 1.210× faster than N, and relative to P, N increases the modeled completion of the committed prefix by 732.3 ms. The gate is thus conservative on Cholesky but protective on LU.

B. Mechanism and Model Validation

The following diagnostics validate runtime-issued peer-copy accounting, demonstrate a controlled migrate–return chain, and audit the deviations excluded by prefix admission. The instrumented transfer pilot uses its own task-cost calibration; its single-run timings and counters are kept separate from the primary matrix.

In each instrumented Cholesky and LU configuration, the scheduler-modeled peer-transfer events and bytes match the corresponding runtime-issued peer transfers exactly. Relative to H, prefix-safe placement reduces runtime-issued peer transfers by 81.06% on Cholesky and 37.74% on LU, tracking its modeled reductions. This agreement supports using modeled input-copy volume as a proxy for the transfers the runtime requests in the evaluated Cholesky and LU configurations.

A controlled CG trace validates the migrate–return mechanism of Section II-C using the fixed-successor certificate of Section IV-D. The trace initializes a 128 MiB mutable version on GPU 0. Under H, the read–write task 46 (`cg_axpy`) runs on GPU 1, which migrates the version off its owner, and a `cg_dot` successor fixed to GPU 0 then copies it back. Because each of the six fixed GPU-0 reads is preceded by a further read–write update on GPU 1 that invalidates the GPU-0 copy under MSI residency, every read re-fetches the version. The one migrate and six returns therefore move the 128 MiB version seven times—the 896 MiB of target peer transfers shown in Figure 4.

The certificate keeps task 46 owner-local because the avoided one-hop successor transfer outweighs the producer's immediate finish-time penalty: the future-transfer lower bound of 6.484 ms exceeds the 0.474 ms HEFT-relative EFT penalty, giving a certificate value of 6.010 ms. Replaying the recorded H decisions against this single intervention shows 28 placement differences: the direct task and 27 downstream cascade changes. The intervention eliminates the predicted return transfers—target peer transfers fall to zero, all peer transfers drop from 2.50 GiB to 480 B of unrelated scalar copies, and all issued transfers fall from 4.25 GiB to 896 MiB (79.41%), with correctness preserved (Figure 4).

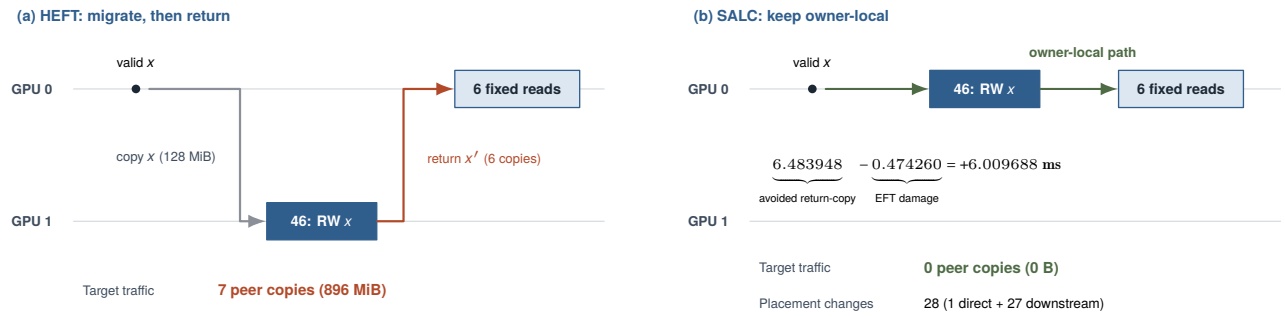


Fig. 4. Controlled CG migrate–return certificate. HEFT moves the 128 MiB version away from its owner; six fixed-device reads return it. SALC's one-hop certificate keeps task 46 owner-local. Because the producer placement is the only externally imposed change, the 27 later placement differences arise from the resulting residency and scheduling state.

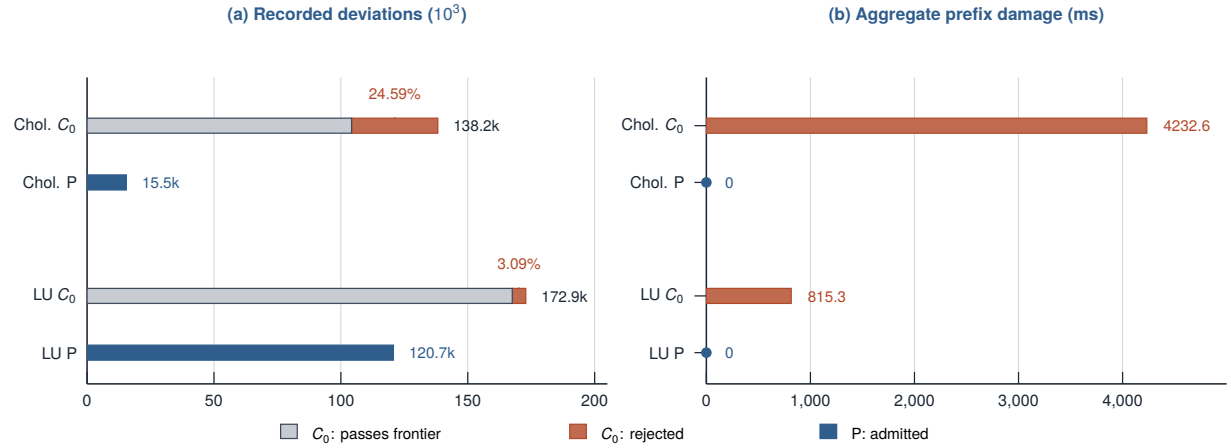


Fig. 5. One-run offline frontier audit. Panel (a) partitions C_0's recorded placement deviations using P's frontier inequality; P's rows are its own admitted deviations, not subsets of C_0. Panel (b) reports aggregate modeled prefix damage. The audit changes neither recorded schedule.

Prefix-safe placement itself makes no intervention here—its column equals H—because the owner-local alternative extends the committed prefix and fails prefix admission; the certificate (SALC), which runs offline and is not part of the scheduler, is what flags the opportunity. The replay imposes only one placement intervention—forcing task 46 to remain on its owner, GPU 0. That intervention changes the subsequent residency state, which in turn produces the 27 downstream placement differences and eliminates the return transfers.

To characterize what the frontier rule rejects, we compare prefix-safe admission with a baseline defined here for this purpose: a normalized-EFT admission rule that bounds the current task's relative finish-time penalty rather than the completion of the committed prefix. We evaluate it in two configurations, C_0 and $C_{0.35}$, which differ only in the queue-ranking weight denoted by their subscripts, $\lambda_Q = 0$ and 0.35. Both tier their parameters by the same task-difficulty split as the proposed scheduler: high-difficulty tasks use an EFT cap of 0.003 and a copy threshold of 0.25, and low-difficulty tasks use 0.015 and 0.15. A placement deviation is a task assigned to a device other than its earliest-finish-time baseline device. Run on Cholesky and LU, these configurations deviate far more often than P: C_0 makes 138,194 and 172,871 deviations, against P's 15,474 and 120,695. Applying P's frontier inequality offline to C_0's recorded placements, without changing the schedule, would reject 24.59% and 3.09% of those deviations—exactly the ones that increase modeled prefix completion, by a cumulative 4232.6 ms and 815.3 ms. By construction, P admits none of them (Figure 5).

TABLE VI
INSTRUMENTED TRANSFER DIAGNOSTIC (ONE OBSERVATION PER CONFIGURATION). RATIOS $T_X/T_P > 1$ FAVOR P; PEER TRANSFERS ARE CUDASTF-ISSUED DISTINCT-GPU COPIES, NOT PHYSICAL-LINK TRAFFIC.

	Speedup of P over		Issued peer transfers (GB)		
Workload	H	C_0	H	C_0	P
Cholesky	1.315×	1.064×	128.85	18.73	24.41
LU	1.166×	0.866×	148.96	95.71	92.75

The resulting latencies are mixed and, because each configuration is a single fixed-order run, serve only as directional diagnostics (Table VI). P is 1.064× faster than C_0 on Cholesky while issuing 30.36% more peer transfers, whereas on LU it issues 3.10% fewer but runs 15.46% slower ($T_{C_0}/T_P = 0.866$);

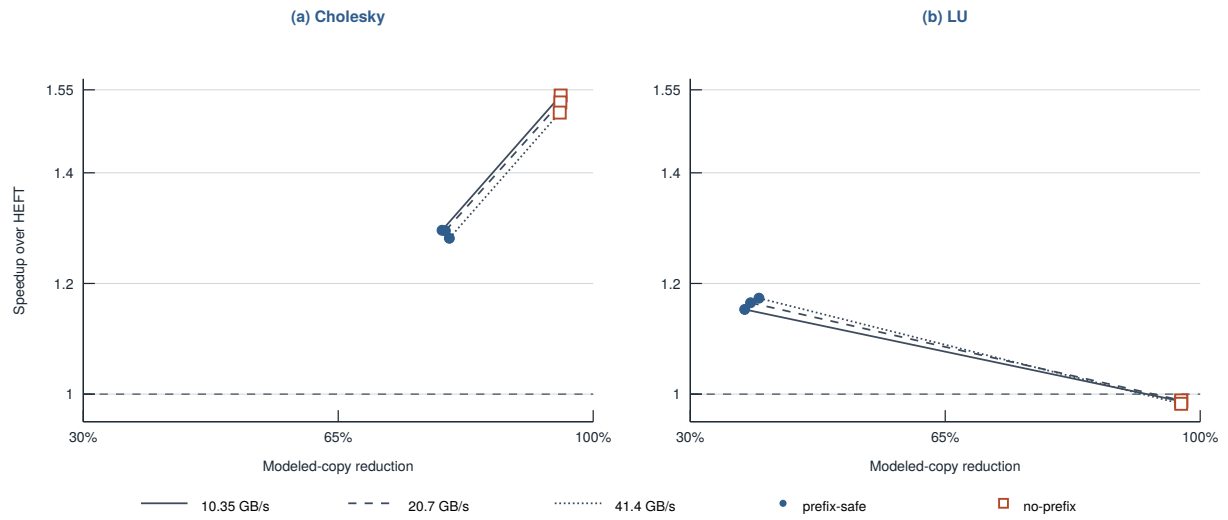


Fig. 6. Prefix-safe (P) versus no-prefix (N) at three scheduler copy-model bandwidths. Segments connect configurations differing only in prefix admission. Each point is a one-run diagnostic.

TABLE VII
SCHEDULER COST IN THE PRIMARY MATRIX (GROUP MEDIANS). **BOLD** SHARES MEET THE PREDECLARED 5% TARGET. SCHEDULER SHARE IS MEASURED SCHEDULING TIME DIVIDED BY THE BENCHMARK INTERVAL, NOT CRITICAL-PATH OVERHEAD; "TARGET GROUPS" COUNTS TREATMENT-GROUP MEDIANS AT OR BELOW 5%.

	Scheduler share (%)						P cost	Target
Workload	H	I	W	O	P	N	μs/task	Groups
FDTD	35.12	24.99	25.18	25.21	25.39	25.38	16.70	0/6
CG	**0.50**	**0.37**	**0.20**	**0.40**	**0.41**	**0.38**	11.00	6/6
Cholesky	13.49	9.94	10.11	10.58	13.10	15.43	7.60	0/6
LU	10.96	8.18	9.69	9.57	10.13	8.63	7.55	0/6
miniWeather	6.64	**4.94**	**4.83**	**4.96**	**4.98**	**4.95**	13.85	5/6

$C_{0.35}$ preserves both directions. The frontier rule thus avoids the increase in modeled prefix completion these deviations would admit, though this diagnostic does not show that P is uniformly faster.

C. Sensitivity to the Copy-Bandwidth Model

This diagnostic tests the primary Cholesky and LU workload sizes at copy-model bandwidths of 10.35, 20.7, and 41.4 GB/s (Figure 6). It uses a separate batch with its own frozen task-cost calibration. Within each policy, only the bandwidth changes across these points; the calibration and other settings remain fixed. The points, including the 20.7 GB/s reference, are not pooled with the primary matrix.

The conservative-versus-protective split between the two workloads persists across all three settings. On Cholesky, both P and N stay faster than H (P 1.282–1.296×, N 1.509–1.540×); on LU, P stays faster than H (1.153–1.173×) while N stays below it (0.983–0.989×). Each configuration is a single run, so the sweep shows that the same qualitative split appears across the modeled bandwidth range.

D. Scheduler Overhead

This measurement quantifies the scheduler's per-task decision cost, reported both as a share of the benchmark interval and as absolute microseconds per task. Share divides the total measured scheduling time by the benchmark interval, so a short interval can produce a large share even when the per-task cost is unremarkable. Absolute cost therefore varies far less across workloads than share does.

The measured group medians are consistent with this observation. Across the 30 workload–scheduler medians in Table VII—one median share per workload and scheduler variant—11 have a scheduler share at or below the predeclared 5% target. The pattern is workload-driven: all six CG groups and five of six miniWeather groups meet the target, whereas every FDTD, Cholesky, and LU group exceeds it. P incurs 7.55–16.70 μs of scheduling time per task; because H and P run different scheduling code, their measured-time difference does not estimate P's incremental cost. On Cholesky and LU, P improves latency despite scheduler shares above the predeclared 5% target.

VI. RELATED WORK

Direct antecedent. The prior placement layer introduced a HEFT-relative residual gate, the task-difficulty proxy reused here, and the same platform and workload family [10]; that gate tolerated a bounded normalized current-task EFT penalty. Our increment is the frontier-based, zero-prefix-damage admission

test, its one-step invariant, and experiments that isolate the gate through exact removal and counterbalanced execution. The C_0 diagnostic of Section V-B ports that fixed gate into the common build for a direct comparison. We do not claim copy-aware residual placement as new.

HEFT and bounded lookahead. HEFT places each task on the processor of minimum earliest finish time, accounting for the communication that delays its inputs [1], and later work enriches this decision with information about the future graph. PEFT adds an optimistic future-cost table [11], lookahead HEFT evaluates a placement's effect on child tasks [12], and interference-aware scheduling jointly models computation- and I/O-induced flows when constraining placement [13]. These methods improve placement by predicting or searching the unscheduled graph. Unlike these approaches, prefix-safe placement refines the current HEFT decision rather than predicting future task behavior. It admits a lower-copy candidate device only when the resulting placement remains within a bounded tolerance of the modeled completion of already committed work.

Task runtimes and locality-aware scheduling. Task-based runtimes couple placement with data movement, and a broad line of work makes that coupling locality-aware. StarPU's data-aware scheduler combines software-managed shared memory with transfer-aware placement [14], [15], while PaRSEC [16], Legion [17], and CUDASTF [2] expose abstractions for dependencies, placement, and data management. On top of such runtimes, XKaapi's DADA schedules by data affinity [18], LAHeteroprio quantifies locality's effect on task-based performance [19], MCL selects resources by locality [20], and comparative studies place HEFT beside data- and locality-aware multi-CPU/multi-GPU schedulers [21]. More recent systems optimize locality under GPU-memory limits [22], combine ready-task locality with eviction for GPU and out-of-core linear algebra [23], and rebalance already-mapped work across GPUs [24]. These schedulers pursue locality through affinity heuristics, resource selection, or work migration. Unlike these schedulers, our work does not replace the underlying placement policy. Instead, it refines the HEFT decision by admitting lower-copy alternatives only when they preserve the modeled completion of already committed work.

Energy-aware scheduling. Prior heterogeneous schedulers reduce energy consumption or energy–delay cost through energy-aware task mapping, DVFS, power capping, or joint placement–frequency optimization [25]–[27]. These approaches address a broader resource-management objective than placement alone. Our current scheduler operates under fixed GPU settings. A future extension could incorporate DVFS as a complementary control layer, with the cost model extended to capture frequency-dependent execution time and energy.

Transfer attribution. Attributing GPU data movement to program context is usually a profiling task; HPCToolkit, for instance, attributes kernels and copies to heterogeneous calling contexts [28]. Our runtime-issued transfer stream adds a complementary, decision-level view: it links each requested peer transfer to the scheduler decision and logical-data identity that produced it, connecting a placement choice directly to the data movement it induces. In the common-build diagnostic of Section V-B, its aggregate GPU-peer event and byte counts agree exactly with the scheduler's accounting in all ten executions.

VII. Limitations

Guarantee. Prefix-safe placement bounds one decision's modeled prefix completion relative to the EFT choice from the same runtime state. Even with accurate costs, subsequent placements and residency can diverge, so full-DAG makespan and measured latency can still regress relative to HEFT. It does not optimize the unscheduled suffix.

Evaluation scope. The admission rule is defined independently of workload identity, but the present evaluation is limited to the workloads, platform, and runtime studied here. The five-workload matrix demonstrates the scheduler's behavior in both placement–residency opportunity cases and identical-placement controls, but it does not estimate the prevalence of such opportunities in broader applications. Moreover, the two placement-changing sizes were fixed after a single exploratory run, so their repeated results characterize behavior under an observed opportunity rather than an unbiased frequency. The experiments further assume deferred task submission on a homogeneous eight-GPU node and do not evaluate dynamic task arrival, heterogeneous devices, external runtime schedulers, or a counterbalanced repeated comparison with the complete prior policy; the common-build $C_0/C_{0.35}$ comparison is a fixed-order, one-run diagnostic. For dynamic arrivals, the same local rule would require updated dependencies, device-ready frontiers and MSI state, plus estimates for previously unseen task-cost keys; this extension remains unvalidated.

Implementation and measurement. The prototype uses a simplified transfer-cost model: a single effective bandwidth and a shared GPU-0 task-cost calibration, omitting pairwise NODE/SYS bandwidth differences, contention, copy–compute overlap, and memory-capacity effects such as eviction. Task-cost errors may admit alternatives that delay actual prefix completion or reject beneficial ones; the bandwidth sweep does not test these errors. The reported copy quantities are scheduler-model outputs; the common-build diagnostic (Section V-B) confirms their exact equality with CUDASTF-issued GPU-peer accounting only for the two dense configurations tested, and neither quantity is physical-link traffic. The controlled-trace attribution rests on the task/data/fixed-successor join rather than an independent runtime version epoch; confirming physical PCIe/NVLink traffic would require CUPTI activity tracing or an equivalent profiler [29]. The implementation also prioritizes clarity over scheduler efficiency, leaving lower scheduler overhead and validation on larger systems and broader workloads as future work.

VIII. Conclusion

Task placement in multi-GPU task-graph runtimes influences both execution time and future data residency. This paper identifies the resulting placement–residency conflict: a

placement that minimizes the current task's modeled finish time can increase the communication required by later dependent tasks. To address this problem, we present prefix-safe communication-aware placement, which evaluates lower-copy alternative placements and admits them under a bound on modeled prefix completion relative to the EFT choice. We implement the proposed scheduler in CUDASTF and evaluate it on representative task-graph workloads running on an eight-GPU NVIDIA L4 platform. The proposed scheduler reduces modeled input-copy volume and improves latency on the two placement-changing workloads, while retaining baseline placement when no alternative is admitted. The modeled prefix contract provides a local admission criterion; the observed latency improvements remain empirical.

ACKNOWLEDGMENT

This work was supported by the Swedish Foundation for Strategic Research (SSF) under the PRIDE project (ref. no. CHI19-0048) and the Swedish Research Council (VR) under the P4PIM project (grant ID 2020-04892). Experiments were conducted on the Minerva cluster at the Department of Computer Science and Engineering, Chalmers University of Technology.

Throughout the paper, the authors used ChatGPT to improve clarity and language, including grammar, wording, and paraphrasing of text originally written by the authors. All AI-assisted edits were reviewed and revised by the authors, who take full responsibility for the final content.

REFERENCES

[1] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[2] C. Augonnet, A. Alexandrescu, A. Sidelnik, and M. Garland, "CUDASTF: Bridging the gap between CUDA and task parallelism," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. Los Alamitos, CA, USA: IEEE Computer Society, 2024, pp. 672–688.

[3] CCCL Development Team, "CCCL: CUDA C++ core libraries, version 3.2.0," Software release, 2026, release commit 477f8bc. [Online]. Available: https://github.com/NVIDIA/cccl/tree/v3.2.0

[4] NVIDIA Corporation, "cuBLAS library," CUDA Toolkit 12.4.1 documentation, 2024. [Online]. Available: https://docs.nvidia.com/cuda/archive/12.4.1/cublas/index.html

[5] ——, "cuSOLVER library," CUDA Toolkit 12.4.1 documentation, 2024. [Online]. Available: https://docs.nvidia.com/cuda/archive/12.4.1/cusolver/index.html

[6] K. S. Yee, "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media," *IEEE Transactions on Antennas and Propagation*, vol. 14, no. 3, pp. 302–307, 1966.

[7] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *Journal of Research of the National Bureau of Standards*, vol. 49, no. 6, pp. 409–436, 1952.

[8] M. R. Norman, "The MiniWeather mini app," Software repository, 2018. [Online]. Available: https://github.com/mrnorman/miniWeather

[9] E. J. Williams, "Experimental designs balanced for the estimation of residual effects of treatments," *Australian Journal of Scientific Research, Series A: Physical Sciences*, vol. 2, no. 2, pp. 149–168, 1949.

[10] L. Cheng and Y. Li, "A multi-objective scheduling framework for energy–performance optimization in multi-GPU task graphs," Master's thesis, Chalmers University of Technology, 2026. [Online]. Available: https://hdl.handle.net/20.500.12380/311908

[11] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 682–694, 2014.

[12] L. F. Bittencourt, R. Sakellariou, and E. R. M. Madeira, "DAG scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm," in *18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 27–34.

[13] R. Bleuse, G. Lucarelli, and D. Trystram, "A methodology for handling data movements by anticipation: Position paper," in *Euro-Par 2018: Parallel Processing Workshops*. Cham, Switzerland: Springer, 2019, pp. 134–145.

[14] C. Augonnet, J. Clet-Ortega, S. Thibault, and R. Namyst, "Data-aware task scheduling on multi-accelerator based platforms," in *16th International Conference on Parallel and Distributed Systems*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 291–298.

[15] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: A unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 187–198, 2011.

[16] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Hérault, and J. J. Dongarra, "PaRSEC: Exploiting heterogeneity to enhance scalability," *Computing in Science & Engineering*, vol. 15, no. 6, pp. 36–45, 2013.

[17] M. Bauer, S. Treichler, E. Slaughter, and A. Aiken, "Legion: Expressing locality and independence with logical regions," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. Los Alamitos, CA, USA: IEEE Computer Society, 2012, pp. 66:1–66:11.

[18] R. Bleuse, T. Gautier, J. V. F. Lima, G. Mounié, and D. Trystram, "Scheduling data flow program in XKaapi: A new affinity based algorithm for heterogeneous architectures," in *Euro-Par 2014 Parallel Processing*. Cham, Switzerland: Springer, 2014, pp. 560–571.

[19] B. Bramas, "Impact study of data locality on task-based applications through the heteroprio scheduler," *PeerJ Computer Science*, vol. 5, p. e190, 2019.

[20] A. V. Kamatar, R. D. Friese, and R. Gioiosa, "Locality-aware scheduling for scalable heterogeneous environments," in *2020 IEEE/ACM International Workshop on Runtime and Operating Systems for Supercomputers (ROSS)*. Los Alamitos, CA, USA: IEEE Computer Society, 2020, pp. 50–58.

[21] J. V. F. Lima, T. Gautier, V. Danjean, B. Raffin, and N. Maillard, "Design and analysis of scheduling strategies for multi-CPU and multi-GPU architectures," *Parallel Computing*, vol. 44, pp. 37–52, 2015.

[22] M. Gonthier, L. Marchal, and S. Thibault, "Taming data locality for task scheduling under memory constraint in runtime systems," *Future Generation Computer Systems*, vol. 143, pp. 305–321, 2023.

[23] ——, "A scheduler to foster data locality for GPU and out-of-core task-based linear algebra applications," *Journal of Parallel and Distributed Computing*, vol. 206, p. 105170, 2025.

[24] J. John, J. Milthorpe, T. Hérault, and G. Bosilca, "Multi-GPU work sharing in a task-based dataflow programming model," *Future Generation Computer Systems*, vol. 156, pp. 313–324, 2024.

[25] J. Chen, M. Manivannan, B. Goel, and M. Pericàs, "Joss: Joint exploration of cpu-memory dvfs and task scheduling for energy efficiency," in *Proceedings of the 52nd International Conference on Parallel Processing*, ser. ICPP '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 828–838. [Online]. Available: https://doi.org/10.1145/3605573.3605586

[26] J. Chen, M. Manivannan, B. Goel, and M. Pericàs, "Sweep: Adaptive task scheduling for exploring energy performance trade-offs," in *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2024, pp. 325–336.

[27] J. Chen and M. Pericàs, "Deft: Joint task placement and dvfs for energy-efficient multi-gpu runtimes," in *Proceedings of the 40th ACM International Conference on Supercomputing*, ser. ICS '26. New York, NY, USA: Association for Computing Machinery, 2026, p. 1115–1127. [Online]. Available: https://doi.org/10.1145/3797905.3807847

[28] K. Zhou, L. Adhianto, J. Anderson, A. Cherian, D. Grubisic, M. Krentel, Y. Liu, X. Meng, and J. Mellor-Crummey, "Measurement and analysis of GPU-accelerated applications with HPCToolkit," *Parallel Computing*, vol. 108, p. 102837, 2021.

[29] NVIDIA Corporation, "CUPTI activity api: Memory copy activity records," CUDA Profiling Tools Interface documentation, 2026. [Online]. Available: https://docs.nvidia.com/cupti/api/structCUpti__ActivityMemcpy6.html