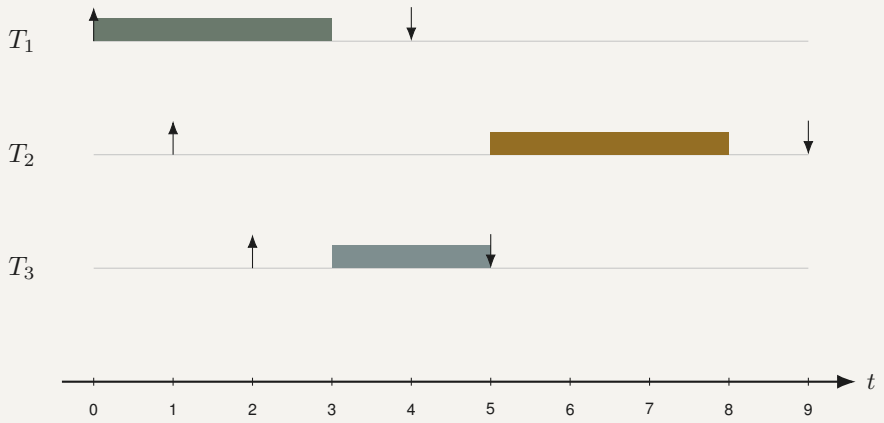
VANDERBILT UNIVERSITY
College of Connected Computing

Reshaping Real-Time Workload Geometry to the Dimensions of Modern Hardware

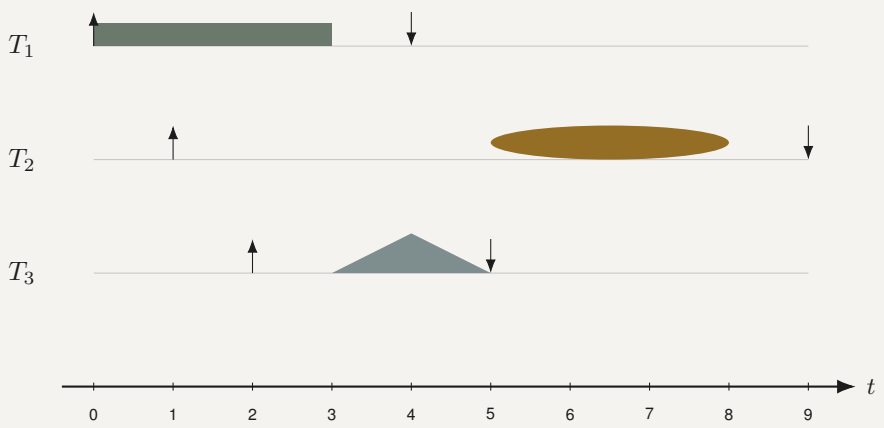
Bryan C. Ward · bryan.ward@vanderbilt.edu

Traditional Uniprocessor EDF



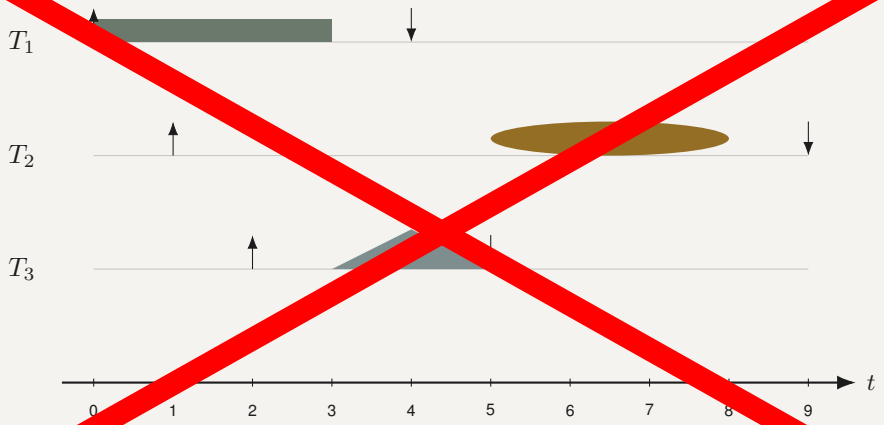
Releases ↑, deadlines ↓, execution as filled bars – a perfectly ordinary EDF schedule.

Reshaping the Workload Geometry



Same workload, different “geometry”.

Reshaping the Workload Geometry



Same workload, different “geometry”.

The classical real-time task model

$$\tau_i \;=\; (C_i,\, D_i,\, T_i)$$

- C_i: worst-case execution time – one number, run to completion, no internal structure.
- D_i: relative deadline.
- T_i: period – jobs arrive exactly T_i time units apart.
- Each job is scheduled (rate-monotonic, EDF, . . .) as a single unit on a single core.

C. L. Liu, J. W. Layland. *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment.* JACM 1973.

...and the model keeps getting extended

Extension	What changes	Ref.
Sporadic tasks	T_i becomes a *minimum* inter-arrival time, not an exact period	Baruah et al. '90
Blocking-aware	add a blocking term B_i for time spent waiting on a shared resource	Sha et al. '90
Mixed-criticality	C_i becomes a *vector* of WCETs, one per assurance level	Vestal '07
DAG/parallel tasks	C_i becomes an internal precedence graph of subtasks, not one number	Saifullah et al. '11

Takeaway

In scheduling-theory work, we often assume some innate WCET(s) that are derived from some assumed WCET-oracle that can derive safe bounds.

What a single C_i hides

- **Caches**: multi-level hierarchies (L1/L2/L3) turn "the same" access into a 10–100$\times$ swing depending on whether the line is resident, shared, or just evicted by a neighbor.
- **GPUs**: thousands of threads grouped into warps and SMs, efficient only when there's enough co-resident, parallel work to hide memory latency and keep the pipeline full.
- **Memory**: DRAM row-buffer locality, NUMA distance – moving data between levels is often more expensive than computing on it.
- **I/O**: data must be moved from peripherals into memory, CPUs, GPUs, or other accelerators.

Takeaway

None of this shows up in a single C_i – WCET analysis treats every invocation as equally expensive, independent of what's already warm in the memory hierarchy.

Workload geometry

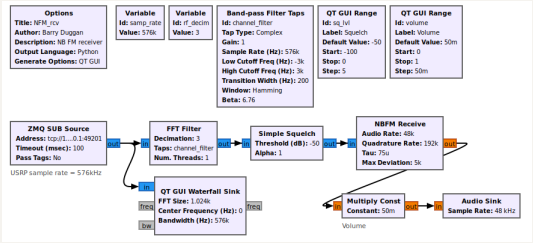
- We can shape our workloads and task models to these hardware considerations.
- In this talk we refer to the “Geometry” as task-model extensions that allow you to “shape” the workload parameters to better encapsulate hardware properties.
- The biggest factor in shaping workload geometry is locality.

Takeaway

Geometry isn’t one specific parameter – it’s whatever shape the workload takes. Locality, or lack thereof, affects that cost.

Exemplar: software radio

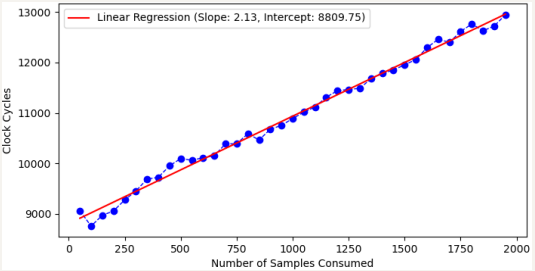
- Software-defined radio (SDR) is a good example.
- SDR is modeled as a *flowgraph* or DAG of small DSP blocks – filters, mixers, demodulators.
- Samples arrive at kHz–GHz rates. Each sample is effectively a job that needs to be done.
- Context switching for each sample has high overhead, so frameworks like GNU Radio “batch” or coalesce jobs to be processed to increase locality.



An NBFM receiver flowgraph – a real GNU Radio pipeline

Empirical support

- One-sample-at-a-time is a geometry that denies locality: every invocation starts cold.
- Measured on a real GNU Radio block: the first sample costs $\sim$9,000 cycles; each additional sample costs $\sim$2 cycles – almost pure I, almost no Δ.
- That imbalance is the geometry mismatch the rest of this talk reshapes.



Execution time vs. samples processed, one real DSP block

Takeaway

Software radio's natural geometry (fine-grained, sequential) denies the hardware the locality it needs – and that 9,000-vs-2-cycle split is general enough to deserve its own cost model.

The marginal cost model

$$C_i(N) \;=\; I_i \;+\; \Delta_i \cdot N$$

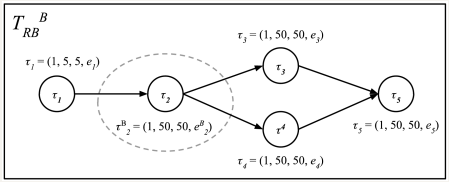
- That 9,000-vs-2-cycle split is two terms of a general model: $C_i(N)$, task i's execution cost, is now a *function* of batch size N.
- I_i: the cost of buying locality – context switch, cache warm-up, thread sync – paid once, before the invocation is local at all.
- Δ_i: the cost of spending it – per-unit work once the invocation is already warm and local.
- N: the batch size – one geometric parameter that decides how much work amortizes I_i.

Takeaway

Batching trades latency for utilization: fewer, larger invocations amortize I_i over more local, useful work. The rest of this talk surveys prior work that reshapes geometry to buy locality: in *time*, in *space*, or across the *CPU/GPU* boundary.

Reshaping in time: batching PGM task graphs

- Model SDR pipelines as multicore PGM (Processing Graph Method) graphs; give the first end-to-end latency analysis for batched multicore PGM graphs under G-EDF-like scheduling.
- **Uniform Batching** (every node, factor N) vs. **Rate-Exploiting Batching** (only nodes with an integer consume/produce ratio, e.g. decimating filters).
- Heavy graphs: $N{=}4$ drops utilization $\sim$6$\rightarrow\sim$2 cores for only $+0.06$ ms latency; adding Rate-Exploiting Batching drops it below one core.

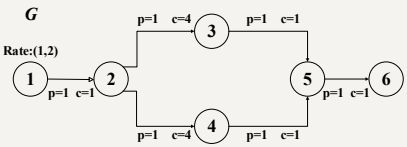


Rate-exploit batched PGM graph

A. Eisenklam, W. Hedgecock, B. Ward. *Job-Level Batching for Software-Defined Radio on Multi-Core.* RTSS 2024.

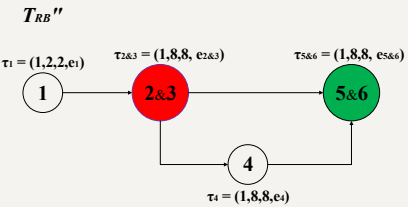
Reshaping in space: merging blocks with MIRAGE

- Batching alone only reshapes *one* block in time – adjacent, rate-compatible blocks still pay separate initialization costs.
- **MIRAGE** casts which blocks to co-schedule as a mixed-integer linear program: fuse adjacent blocks into one unit to maximize intra-group locality, subject to a utilization cap and acyclic dependencies.
- Merged cost $\approx \max(I_u, I_v) + (\Delta_u + \Delta_v)N$ on a Raspberry Pi 5 NBFM receiver, merging beats rate-exploiting-only latency.



Before: every block is its own task

↓ MILP merge



After: fused groups pay one I each

T. He, B. Ward. *MIRAGE: MILP-Based Block Grouping for Real-Time Signal Processing.* RTAS 2026.

Reshaping across the CPU/GPU boundary

- AI-based spectrum sensing pushes detection onto the GPU – but the marginal cost model so far only describes CPUs.
- GPU hardware gives us two knobs to tune in batching:
  - **Batch size** – how many samples each kernel invocation processes; amortizes the fixed cost I (launch, transfer, warm-up) similarly as on the CPU.
  - **SM allocation** – how many streaming multiprocessors the kernel runs across in parallel.
- The knobs interact: execution time depends on both batch size and SM allocation, so time (batching) and space (partitioning) must be reshaped *together*.
- We have preliminary investigations into modeling and optimizing this trade-off.

Takeaway

Accelerators add a spatial knob to batching's temporal one – the hardware effects are coupled, and tuning the two knobs jointly is itself a geometry-reshaping problem.

Batching Summary

Effort	Reshapes
Batching (RTSS'24)	time (batch size)
MIRAGE (RTAS'26)	space (block grouping)
GPU batching (preliminary)	placement (CPU/GPU)

Takeaway

Across cores, block boundaries, and the CPU/GPU boundary, the same idea recurs: pay a fixed cost once, amortize it over a locality-friendly batch – and give that reshaping a real-time guarantee. But each axis was reshaped offline, by hand, one at a time.

Open Problems

Moving theory to practice

- **GNU Radio 4**'s new runtime supports time and space batching, but existing policies aren't theoretically grounded.
- Supports *SIMD & Merge API*. Fuses compatible blocks, and the scheduler itself is a pluggable component – separate sub-schedulers per CPU/GPU/hybrid domain ("scheduler-as-a-plugin").
- That is exactly the move MIRAGE formalizes – but GR4's merge and placement decisions are heuristic: no timing model, no schedulability guarantee behind them.
- **Open problem:** expose marginal-cost-aware batching and MILP-based grouping as one of GR4's pluggable schedulers – turn an engineering heuristic into an analyzable one.

GNU Radio 4 (gnuradio.org, 2025–2026); "The Future of GNU Radio: Heterogeneous Computing, Distributed Processing, and Scheduler-as-a-Plugin," IEEE.

Other application domains: LLM serving

- An LLM generates text one token ($\approx$ one word) at a time – and each step must read the *entire* model (tens–hundreds of GB) from GPU memory just to produce that one token.
- That reload is a huge initialization cost I; adding another user's request to the same step costs almost nothing more (tiny Δ). The same $C_i(N) = I_i + \Delta_i \cdot N$ shape as our DSP blocks.
- So production systems **batch**: many requests share one pass over the model, with the batch re-formed on the fly as requests arrive and finish – all heuristic, with no worst-case timing model.
- **Open problem:** Are there real-time LLM applications that would benefit from real-time batching?

Other application domains: robotics

- **Perception** already reshapes geometry by hand: compositing 4 camera streams into one DNN input hits 250 FPS on an NVIDIA DRIVE PX2 – N picked empirically, no timing model.
- **Motion planning inverts the problem**: planners and controllers batch thousands of collision checks / rollouts per control period (SIMD, GPU) – more samples mean a better trajectory.
- Same $C_i(N) = I_i + \Delta_i \cdot N$, dual objective: the deadline is fixed, choose the largest N that fits – batching for quality under a deadline, not utilization under a latency budget.
- **Open problem:** schedulability analysis where batch size is a *quality* knob – co-design control performance and timing guarantees around one marginal cost model.

Dynamic batching

- Batching is everywhere in deployed systems – but batch sizes and groupings are usually set by heuristics: they yield good average-case performance, yet aren't amenable to analysis.
- Our reshaping so far is *static*: geometry chosen offline, then fixed. Real workloads present opportunities at runtime – e.g. consecutive jobs in a DAG that share data can reuse warm cache state if run back-to-back.
- Could a scheduler (re-)prioritize work to increase locality – deliberately ordering jobs so one job's I is partially pre-paid by its predecessor?
- **Open problem:** analysis techniques that support batching *dynamism* – schedulability guarantees that get analytical credit for locality the scheduler creates, not just runtime benefits the analysis must ignore.

One dimension at a time

- Each paper reshapes a single axis, offline, for a static graph on a static target.
- Real pipelines don't hold still: GR4 flowgraphs are hot-swapped, DAQIRI ingests from many instruments at once, LLM batches are re-formed every iteration.

Time (batch size) RTSS'24	Space (block grouping) RTAS'26	Placement (CPU/GPU) preliminary

? joint optimization across all three axes – and doing it *online*, adaptively, as the workload runs

- **Open problems:** is a joint formulation tractable? adaptive reshaping as conditions change; how do these decisions propagate into cause-effect-chain and control-loop latency analysis?

Discussion

- Where else is hardware or industry already doing ad hoc “reshaping” that real-time analysis hasn’t caught up to?
- There are many interesting and open scheduling theory problems when you add batching considerations to task models.

Takeaway

Reshaping workload geometry is already happening in industry hardware and open-source frameworks – often without real-time guarantees. That gap is the open problem.

VANDERBILT UNIVERSITY
College of Connected Computing

Questions or Comments

Bryan C. Ward · bryan.ward@vanderbilt.edu