Generative Profiling for Soft Real-Time Systems and its Applications to Resource Allocation

Georgiy A. Bondar*[1] Abigail Eisenklam*[2] Yifan Cai[2] Robert Gifford[2] Tushar Sial[3]
Linh Thi Xuan Phan[2] Abhishek Halder[3]

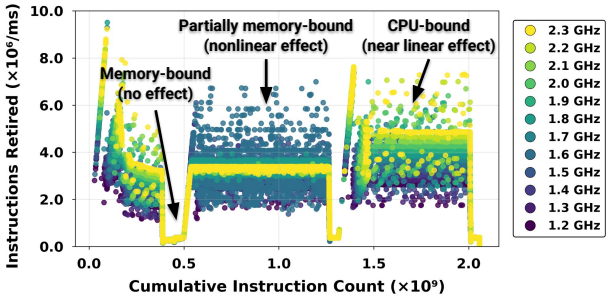
[1]University of California, Santa Cruz [2]University of Pennsylvania [3]Iowa State University

***Abstract*—Modern real-time systems require accurate characterization of task timing behavior to ensure predictable performance, particularly on complex hardware architectures. Existing methods, such as worst-case execution time analysis, often fail to capture the fine-grained timing behaviors of a task under varying resource contexts (e.g., an allocation of cache, memory bandwidth, and CPU frequency), which is necessary to achieve efficient resource utilization. In this paper, we introduce a novel generative profiling approach that synthesizes context-dependent, fine-grained timing profiles for real-time tasks, including those for unmeasured resource allocations. Our approach leverages a nonparametric, conditional multi-marginal Schrödinger Bridge (MSB) formulation to generate accurate execution profiles for unseen resource contexts, with maximum likelihood guarantees. We demonstrate the efficiency and effectiveness of our approach through real-world benchmarks, and showcase its practical utility in a representative case study of adaptive multicore resource allocation for real-time systems.**

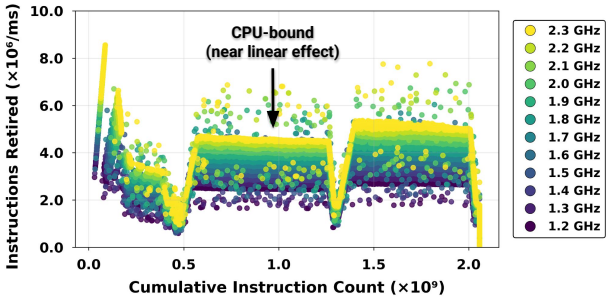
I. Introduction

Modern real-time systems execute increasingly sophisticated software on complex hardware architectures. Ensuring timing predictability in these systems relies critically on *accurate characterization of task execution under varying resource configurations*. On a multicore platform, a task's execution behavior—such as the rates of cache misses, memory requests and instruction retirement—can be highly sensitive to its *resource context*, which includes the amount of shared cache, memory bandwidth, and CPU frequency allocated to it [1], [30], [74]. For instance, as shown in Fig. 1, the PARSEC fft benchmark exhibits distinct execution phases with varying instruction rates, which change in response to CPU frequency, cache and memory bandwidth allocation. Accurately estimating these timing profiles is critical for effective timing analysis and resource allocation in modern real-time systems. Such profiles enable context-aware execution time distributions for probabilistic timing analysis [37], [22], [16], [14] and efficient multi-phase task scheduling [25]. Furthermore, they are central to adaptive multicore resource allocation [30] and power management [1] techniques, which are essential for improving predictability and resource utilization in real-time systems.

Limitations of existing work. While traditional analysis methods, such as worst-case execution time (WCET) analysis, can provide bounds or distributions of total execution times, they often fail to capture the fine-grained timing variability under different resource allocations. To obtain fine-grained execution profiles, current solutions have relied on exhaustive measurements. Although this approach can provide fine-grained insights, it can quickly become impractical to cover *all possible resource contexts* as the number of tasks or allocation configurations increases. Machine learning-based methods have been proposed to address this limitation; however, they often focus on *coarse-grained* execution metrics, such as WCET bounds or aggregated performance distributions. Recently, Bondar et al. [12], [11] applied stochastic learning models to estimate task timing behavior under a given resource context based on measured execution snapshots. However, these methods remain limited to the resource contexts for



(a) fft with 10% of shared cache and memory bandwidth



(b) fft with 70% of shared cache and memory bandwidth

Fig. 1: Effect of CPU frequency, cache, and memory bandwidth on the instruction retirement rate of fft [69]. The benchmark shows distinct phases, with varying relationships (no effect, linear, nonlinear) between instruction rate and CPU frequency, depending on resource allocation (middle phase).

*Equal contribution.

2642-7346/26/$31.00 ©2026 IEEE
DOI 10.1109/RTAS68450.2026.00034

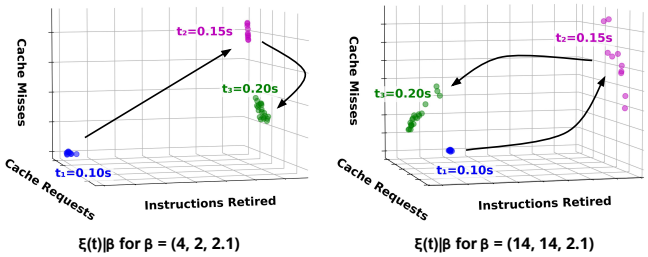


Fig. 2: Distributions of microarchitectural execution states (rates of instructions, cache requests and cache misses) of fft at three time points under two different resource contexts.

which data is available and cannot predict behavior in *unseen contexts*—resource contexts for which no training data exist.

Contributions. In this paper, we propose a novel *generative profiling* approach to synthesize context-dependent, fine-grained timing profiles for *any* resource context, including those that have not been measured. For instance, given *sparse execution snapshots* from a few measured contexts (e.g., an allocation of 10% of cache and memory bandwidth and an allocation of 80% of cache and memory bandwidth), our model can infer *complete temporal* profiles for *all* resource contexts in between (e.g., an allocation of 20% of the cache and 50% of the memory bandwidth).

Predicting execution profiles for unseen contexts is challenging due to the dynamic nature of task execution and the varying correlations among execution states (e.g., rates of cache misses, memory requests, and instruction retirement). As illustrated in Fig. 2, these correlations evolve with time and resource context, complicating their modeling with traditional parametric approaches, such as Gaussian mixtures or Markov chains. To address this, we introduce a nonparametric learning approach based on a *conditional* multi-marginal Schrödinger Bridge (MSB) formulation, enabling us to capture complex dependencies from sparse data without making any distributional assumptions. Our solution efficiently generates accurate fine-grained execution profiles for unseen contexts while providing maximum likelihood guarantees.

In summary, we make the following contributions:

- A generative stochastic model for non-parametrically generating context-dependent execution profiles of real-time tasks under unseen contexts, based on a conditional MSB formulation. Our solution is accurate, efficient, and provides maximum likelihood guarantees.
- Evaluations of the accuracy and efficiency of our approach using real-world benchmarks.
- A case study demonstrating the practical utility of generative profiles in multicore real-time systems, including an adaptive resource allocation algorithm, its prototype implementation, and experimental results.

To the best of our knowledge, this is the first generative profiling solution for real-time multicore systems capable of producing execution profiles for *unmeasured resource contexts* with statistical guarantees.

II. Background and Problem Formulation

A. Stochastic Modeling of Context-Dependent Profiles

System assumptions. The system contains a set of tasks that execute on a multicore hardware platform. Tasks running on the cores share a common set of resources, such as the last-level shared cache, memory bandwidth, and main memory. We assume that each resource can be partitioned into equal-size allocation units, and that some number of resource units can be assigned to a core at run time (e.g., as done in [30]). For example, the cache can be efficiently partitioned using methods such as CAT [34] (for Intel architecture) or Lockdown by Master (LbM) mechanism [48], [3] (for ARM architecture), and the memory bandwidth can be partitioned using MemGuard [75]. The platform is equipped with a per-core power management technique such as Dynamic Voltage and Frequency Scaling (DVFS), which typically supports a range of frequencies at discrete step sizes [71], resulting in a finite number of voltage/frequency settings for a core. For simplicity, we treat frequency as a type of resource. Thus, a "resource context" denotes an assignment of both multicore shared resources *and* a voltage/frequency setting to a core.

In this work, we focus on tasks with a single execution path, but our generative profiling method is entirely data-driven and can also be applied to general programs. We leave the interpretation and evaluation of generative profiles for multi-path programs as future work.

Definitions. Formally, the system contains b resource types, where b is some fixed positive integer. A *resource context* β is then defined as a vector with b components:

$$\beta = (\beta_1, \beta_2, \ldots, \beta_b)^\top \in \mathcal{B} \subset \mathbb{R}^b, \tag{1}$$

where β_i denotes the allocation of the i-th resource type. For example, $b = 3$ resource types may comprise of the shared cache (β_1), memory bandwidth (β_2), CPU frequency (β_3). Then, $\beta = (2, 4, 2.3)$ denotes an allocation of 2 cache partitions, 4 bandwidth partitions, and frequency of 2.3 GHz.

For some fixed positive integer m, we define the *microarchitectural execution state* as an m component vector

$$\xi = (\xi_1, \xi_2, \ldots, \xi_m)^\top \in \mathcal{X} \subset \mathbb{R}^m. \tag{2}$$

An example of $m = 3$ components of ξ are the number of retired instructions (ξ_1), cache requests (ξ_2), and cache misses (ξ_3) in some unit time interval.

For a fixed resource context β, the execution state ξ varies with time t. *Mathematically, $\xi(t)$ is a vectorial stochastic process conditioned on a resource context β.* We denote this conditional stochastic process as

$$\xi(t) \mid \beta. \tag{3}$$

The stochastic process (3) is correlated both *temporally* (across ξ vectors at different times) and *spatially* (across different components of ξ at the same time). Intuitively, if the task issues more shared cache requests at some time t, it may issue fewer requests in some future time $t' > t$, as the data may still be resident in the private L1 or L2 cache. Similarly, for any given time t during the task's execution,

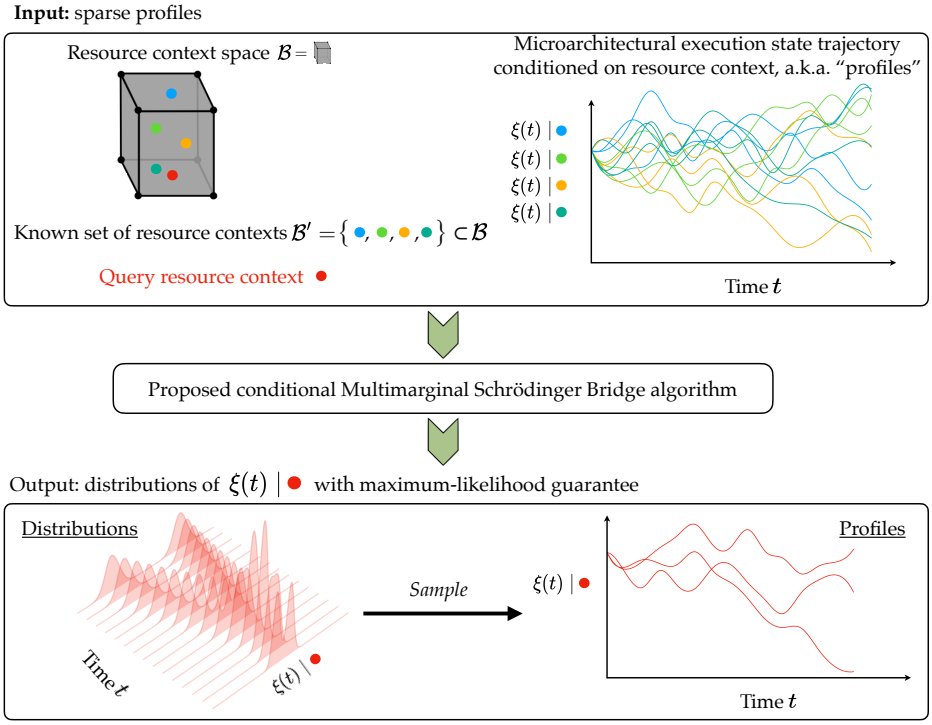


Fig. 3: Our proposed generative profiling algorithm scheme with maximum-likelihood guarantee. From empirical profiles of a task's microarchitectural execution states, conditioned on a sparse set of resource allocations $\mathcal{B}'$, we generate 'synthetic' profiles for the unknown resource allocations.

the instruction rate is correlated to the rates of cache requests and misses, with the correlation depending on the nature of the workload (e.g., compute-intensive or memory-intensive). Even when β and the initial condition $\xi(t=0)$ are fixed for a deterministic single-path task, repeated executions of that task under the same resource context results in slightly different sample paths of $\xi(t)$ because of variability in the hardware state and OS behaviors.

Finally, *profiles* are formally defined as the $\mathbb{R}^m$-valued sample paths of the conditional stochastic process (3).

B. Nonparametric Learning

Consider the special case of a single resource context β, i.e., where the set of all possible resource contexts $\mathcal{B} := \{\beta\}$ is singleton. For this fixed β, suppose we observe the distributions[1] of $\xi(t)|\beta$ at two fixed times t_1 and t_2, i.e., we have the snapshots of the execution state at those times for multiple runs of the task. Let us call these snapshots μ_1 and μ_2, respectively. With this information, we desire to learn distributions μ_t to estimate the statistics of the execution state *at all times* $t \in [t_1, t_2]$. In other words, what we desire to learn is not a single distribution, but a *distribution-valued path* $t \mapsto \mu_t$ parameterized by $t \in [t_1, t_2]$.

Naturally, we must enforce $\mu_{t_1} = \mu_1$ and $\mu_{t_2} = \mu_2$ to guarantee that the model is *consistent* with the known observations. For $t \in (t_1, t_2)$, we also want μ_t to have some mathematical guarantee of its predictive accuracy. Since we do not make any assumption about the nature of the distribution μ_t, how it evolves with time, or how it is affected by the resource allocation β, our learning should be nonparametric in nature.

This is where the bimarginal Schrödinger Bridge Problem (SBP) and its solution – the Schrödinger Bridge (SB) – comes in. In the stochastic process literature, the word "bridge" [70, Ch. IV.40] refers to a stochastic process whose sample path connects two given *finite dimensional endpoints* at two fixed times t_1 and t_2. The SB is a similar construction except that the endpoints are now *distributions*. Specifically, among all continuous curves $t \mapsto \mu_t$ connecting the given endpoints $\mu_{t_1} = \mu_1$ and $\mu_{t_2} = \mu_2$, the SB is the one with *maximum likelihood* guarantee, i.e., the highest probability curve among all possible distribution-valued continuous curves connecting the μ_1, μ_2. The SB originated in 1931-1932 in the works of Schrödinger [60], [61] and in recent years, its importance is being rapidly recognized in generative AI [23], [62], [43], [72] and stochastic control [41], [21], [17], [52].

Two mathematical guarantees make the predictive learning model we desire to be most naturally represented by an SB, rather than another learning model, such as a neural network or regression trees. *First*, the SB is consistent with known observations μ_1 and μ_2. *Second*, unlike other generative models such as flow matching [42], [28], the SB comes with the most parsimonious statistical guarantee: maximum likelihood certificate. It is appropriate here for, as mentioned, we make no assumption about the true mathematical structure of the stochastic dynamics underlying the platform on which our tasks run. Importantly, it is also possible to compute the SB

[1]In practice, scattered point clouds, as in Fig. 2.

in a nonparametric manner, without making any assumptions on the structure[2] of μ_1, μ_2, or μ_t.

In this work, we will employ the multimarginal extension (see Section III) of SB for the learning of μ_t. We also address the additional complexity that arises when $\mathcal{B}$ is a non-singleton set – in this case, β itself must be treated as a random vector with its own distribution supported over $\mathcal{B}$, statistically correlated with $\xi(t)$. Addressing the case in which β is not fixed is critical, as it allows us to generate profiles for previously unseen resource contexts and thereby model the fine-grained, context-dependent behavior of tasks without exhaustive measurements across all contexts. We describe our learning algorithm next.

III. Generative Profiling Algorithm

Recall that the set $\mathcal{B}$ in (1) denotes the set of all possible resource contexts. For instance, let $b = 3$ with a total of N_{ca} cache partitions, N_{bw} memory bandwidth partitions, N_{freq} possible settings of CPU frequency. Then, $\mathcal{B} \subset \mathbb{R}^3$ is a finite set of cardinality $N_{\mathsf{ca}} \cdot N_{\mathsf{bw}} \cdot N_{\mathsf{freq}}$.

For *any* $\beta \in \mathcal{B}$, we desire a profile $\xi(t) \mid \beta$. However, if $\mathcal{B}$ is large, it may be feasible to obtain these profiles empirically for only a subset $\mathcal{B}' \subsetneq \mathcal{B}$. Moreover, technical limitations in the profiling mechanism or the task itself may lead to these 'profiles' consisting of a temporally coarse set of n_s snapshots. Therefore, the result of this limited profiling is an empirical distribution of the conditional random vector

$$\xi(t_\sigma) \mid \beta \tag{4}$$

for each measurement instance (snapshot) $t_{\sigma \in [\![n_s]\!]}$, and for each resource context $\beta \in \mathcal{B}'$. Hereafter, we adopt the finite set notation for the set of n_s snapshots taken for a given task:

$$[\![n_s]\!] := \{1, 2, \ldots, n_s\}.$$

In other words, (4) is the state ξ at time t_σ conditional on the resource context β.

For a *fixed* resource context β, recent work [11], [12] computed the most likely distributional path μ_t for $t \in [t_1, t_{n_s}]$ from snapshots $\mu_{\sigma \in [\![n_s]\!]}$. That is, for a given task and resource context, prior work filled in the profiling gaps between discrete snapshots in time. In this work, we extend these methodologies to not just fill in gaps in time, but also to *fill in gaps between the sparse set of resource contexts* $\mathcal{B}'$. Specifically, we generate distributions (and thereby, profiles) for each context $\beta \in \mathcal{B} \backslash \mathcal{B}'$ under which the workload was never explicitly measured.

For the generation of these distributions, we will make use of the Multimarginal Schrödinger Bridge Problem (MSBP). Different from prior work, we introduce the *conditional* MSBP, explain its solution, and how we use the same to generate profiles for all possible resource contexts. Fig. 3 shows the overall schematic of the proposed generative profiling algorithm. Our approach takes as inputs empirical resource usage profiles of a task for a set of 'known' contexts $\mathcal{B}' \subset \mathcal{B}$, where $\mathcal{B}$ is the set of all possible contexts. Solving the conditional MSBP (to be discussed in the upcoming sections) then allows us to obtain the maximum-likelihood distribution of the task's microarchitectural execution state at *any time*, and for *any* $\beta \in \mathcal{B}$. These distributions can themselves then be sampled, at various times, to obtain 'synthetic' execution profiles *for unknown resource allocations*.

In the following, we present the conditional MSBP, building on the exposition of [11] for the *unconditional* MSBP. Sec. III-A formulates the conditional MSBP, Sec. III-B presents its solution, and Sec. III-C illustrates an application of generative profiling to multicore resource allocation.

A. Conditional MSBP

For any time t, define the augmented state

$$\eta(t) := \begin{pmatrix} \xi(t) \\ \beta \end{pmatrix} \in \mathcal{X} \times \mathcal{B} \subset \mathbb{R}^{m+b},$$

and let $\eta(t) \sim \mu_t$ where $\sim$ denotes "follows the distribution". By Bayes' theorem [7, p. 169], we have

$$\xi(t) \mid \beta \sim \frac{\mu_t}{\int_{\mathcal{X}} \mu_t \, \mathrm{d}\xi}\,. \tag{5}$$

The numerator in (5) is the joint distribution of the augmented state $\eta(t)$, and the denominator is the marginal distribution of the resource budget $\beta \in \mathcal{B} \subset \mathbb{R}^b$.

Below, we will detail how to learn the numerator μ_t in (5) via the MSBP. Since the denominator in (5) is independent of time t, it can be pre-computed before solving the MSBP.

Given a set of n_b resource contexts $\{\beta^j\}_{j \in [\![n_b]\!]}$ where for each we have the corresponding n_d profiles $\{\xi^{i,j}(t)\}_{i \in [\![n_d]\!]}$, we construct the *empirical distributions*

$$\mu_\sigma := \frac{1}{n_d n_b} \sum_{i=1}^{n_d} \sum_{j=1}^{n_b} \delta(\eta - \eta^{i,j}(t_\sigma)), \quad \forall \sigma \in [\![n_s]\!], \tag{6}$$

supported over the augmented state space $\mathcal{X} \times \mathcal{B} \subset \mathbb{R}^{d+b}$, where δ denotes the Dirac delta, i.e.,

$$\delta(\eta - \eta_0) := \begin{cases} 1 & \text{if} \quad \eta = \eta_0, \\ 0 & \text{otherwise.} \end{cases}$$

Thus, each distribution μ_σ consists of scattered data points at $\eta^{i,j}$, representing samples of microarchitectural execution state $\xi^{i,j}$ conditioned on the resource context β^j. The n_s measurement instances $t_{\sigma \in [\![n_s]\!]}$ are the times

$$0 = t_1 < t_2 < \cdots < t_{n_s - 1} < t_{n_s},$$

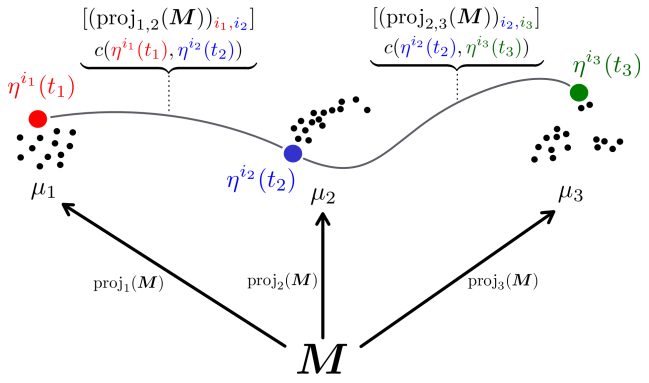
at which 'snapshots' of ξ were taken for each of the n_d profiles.

Given the n_s empirical distributions thus constructed, our goal is to find the most likely measure-valued path $t \mapsto \mu_t$, where $\eta(t) \sim \mu_t \ \forall t \in [t_1, t_{n_s}]$, satisfying the distributional constraints

$$\eta(t_\sigma) \sim \mu_\sigma \qquad \forall \sigma \in [\![n_s]\!]. \tag{7}$$

Therewith, we can sample the distributions μ_t along this path with arbitrary temporal granularity to obtain our synthetic profiles. The path $t \to \mu_t$ itself can be obtained from the multimarginal Schrödinger bridge (MSB) between our known distributions $\{\mu_\sigma\}$. MSBP enables this by finding the maximum-likelihood probability mass transport plan (the

[2]For example, mixture of Gaussians, exponential family [2, Ch. 2] etc.



$$[\boldsymbol{M}_{i_1,i_2,i_3}] = [(\mathrm{proj}_{1,2}(\boldsymbol{M}))_{i_1,i_2}] \cdot [(\mathrm{proj}_{2,3}(\boldsymbol{M}))_{i_2,i_3}]/(\mu_2)_{i_2}$$
$$[\boldsymbol{C}_{i_1,i_2,i_3}] = c(\eta^{i_1}(t_1), \eta^{i_2}(t_2)) + c(\eta^{i_2}(t_2), \eta^{i_3}(t_3))$$

Fig. 4: The relationship between a transport plan $\boldsymbol{M}$, its projections, and the transport cost $\boldsymbol{C}$ for $n_s = 3$ empirical distributions $\{\mu_\sigma\}_{\sigma\in[\![n_s]\!]}$. Unimarginal projections equal the distributions, and for the three arbitrary colored points, $[\boldsymbol{M}_{i_1,i_2,i_3}]$ encodes the total probability mass transported, and $[\boldsymbol{C}_{i_1,i_2,i_3}]$ the per-unit cost thereof, along the grey-colored path therebetween. Similarly, the bimarginal projections of $\boldsymbol{M}$ and the pairwise cost c pertain to the pairwise sections of the path.

MSB) – the solution of an MSBP – between the known distributions $\{\mu_\sigma\}$.

Mass Transport Plan ($\boldsymbol{M}$). A *transport plan* between the distributions $\{\mu_\sigma\}$ is an order-n_s tensor $\boldsymbol{M} \in (\mathbb{R}^{n_d n_b})^{\otimes n_s}_{\geq 0}$, where $[\boldsymbol{M}_{i_1,\ldots,i_{n_s}}]$ is the probability mass transported between the points $\eta^{i_1}(t_1), \ldots, \eta^{i_{n_s}}(t_{n_s})$, and where $\otimes$ denotes the tensor product. Such a plan must also 'conserve' probability mass, i.e., for all $\sigma \in [\![n_s]\!]$ the mass transported to/from all points in the support of μ_σ must equal to μ_σ. Concretely, let

$$\left[\mathrm{proj}_\sigma(\boldsymbol{M})_j\right] := \sum_{i_1,\ldots,i_{\sigma-1},i_{\sigma+1},\ldots,i_{n_s}} \boldsymbol{M}_{i_1,\ldots,i_{\sigma-1},j,i_{\sigma+1},\ldots,i_{n_s}}. \quad (8)$$

This constraint can then be expressed as

$$\mathrm{proj}_\sigma(\boldsymbol{M}) = \mu_\sigma \qquad \forall \sigma \in [\![n_s]\!], \quad (9)$$

wherein the mapping

$$\mathrm{proj}_\sigma : (\mathbb{R}^{n_d n_b})^{\otimes n_s}_{\geq 0} \to \mathbb{R}^{n_d n_b}_{\geq 0}.$$

is called the *unimarginal projection* of $\boldsymbol{M}$ onto μ_σ. Similarly, we may define the *bimarginal projection* of $\boldsymbol{M}$ onto its (σ_1, σ_2)th components as the mapping

$$\mathrm{proj}_{\sigma_1,\sigma_2} : (\mathbb{R}^{n_d n_b})^{\otimes n_s}_{\geq 0} \to \mathbb{R}^{n_d n_b \times n_d n_b}_{\geq 0},$$

where

$$\left[\mathrm{proj}_{\sigma_1,\sigma_2}(\boldsymbol{M})_{j,\ell}\right] := \sum_{\{i_\sigma\}_{\sigma\in[\![n_s]\!]\setminus\{\sigma_1,\sigma_2\}}} \boldsymbol{M}_{i_1,\ldots,i_{\sigma_1-1},j,i_{\sigma_1+1},\ldots,i_{\sigma_2-1},\ell,i_{\sigma_2+1},\ldots,i_{n_s}}. \quad (10)$$

Intuitively, $\mathrm{proj}_{\sigma_1,\sigma_2}(\boldsymbol{M})$ is a restriction of $\boldsymbol{M}$ to two distributions $(\mu_{\sigma_1}, \mu_{\sigma_2})$. This projection may also be viewed as a scattered distribution supported over the finite set $\{\eta^i(t_{\sigma_1})\}_{i\in[\![n_d n_b]\!]} \times \{\eta^i(t_{\sigma_2})\}_{i\in[\![n_d n_b]\!]}$ with μ_{σ_1} and μ_{σ_2} as statistical marginals.

Ground Cost ($\boldsymbol{C}$). As many such transport plans $\boldsymbol{M}$ may exist, we introduce a transport cost function or "ground cost"

$$c : (\mathbb{R}^{n_d n_b})^{n_s} \to \mathbb{R}_{\geq 0},$$

which is such that $c(\eta^{i_1}(t_1), \ldots, \eta^{i_{n_s}}(t_{n_s}))$ is the cost of transporting unit probability mass between the n_s tuple $\left(\eta^{i_1}(t_1), \ldots, \eta^{i_{n_s}}(t_{n_s})\right)$[3]. Intuitively, this cost encodes the 'distance' between the supports of the $\{\mu_\sigma\}$. With this cost function, we may then define $\boldsymbol{C} \in (\mathbb{R}^{n_d n_b})^{\otimes n_s}_{\geq 0}$ as tensor of order n_s, whose entries are

$$[\boldsymbol{C}_{i_1,\ldots,i_{n_s}}] = c(\eta^{i_1}(t_1), \ldots, \eta^{i_{n_s}}(t_{n_s})), \quad (11)$$

Figure 4 shows how a valid transport plan $\boldsymbol{M}$ (i.e., one satisfying (9)), its projections, and the transport cost $\boldsymbol{C}$ relate to the distributions $\{\mu_\sigma\}$. Note that in the context of this work, the snapshots μ_σ are indexed by times t_σ, which imposes a natural 'path' structure to the cost tensor $\boldsymbol{C}$, i.e., (11) specializes to

$$[\boldsymbol{C}_{i_1,\ldots,i_{n_s}}] = \sum_{j=1}^{n_s-1} [C^j_{i_j,i_{j+1}}], \quad (12)$$

where the $C^j \in \mathbb{R}^{n_d \times n_d}$ are matrices whose (i,j)th entry is the value $c(\eta^{i_j}(t_j), \eta^{i_{j+1}}(t_{j+1}))$ for some cost function c (in this work, we use the squared Euclidean cost $c(\cdot,\cdot) := \|\cdot - \cdot\|_2^2$). Intuitively if $n_s = 3$ (see Figure 4), (12) states that the cost of transport between μ_1, μ_2, and μ_3 is equal to the sum of the costs between (μ_1 and μ_2) and (μ_2 and μ_3).

The introduction of a ground cost $\boldsymbol{C}$ allows for the assignment to any transport plan $\boldsymbol{M}$ a mass transport cost

$$\langle \boldsymbol{C}, \boldsymbol{M} \rangle := \sum_{i_1,\ldots,i_{n_s}} [\boldsymbol{C}_{i_1,\ldots,i_{n_s}}][\boldsymbol{M}_{i_1,\ldots,i_{n_s}}],$$

where $\langle \cdot, \cdot \rangle$ denotes the Hilbert-Schmidt inner product. Thus, we may compare probability mass transport plans in terms of this transport cost. For instance, if $c(\eta^{i_1}(t_1), \eta^{i_2}(t_2))$ in Figure 4 is relatively large, a plan which transports more probability mass between these two points may have a higher total mass transport cost.

Problem Formulation. The MSBP refers to finding the transport plan $\boldsymbol{M}$ with the minimal entropy-regularized probability mass transport cost, expressed as the optimization problem

$$\boldsymbol{M}_{\mathrm{opt}} := \operatorname*{arg\,min}_{\boldsymbol{M}\in(\mathbb{R}^{n_d n_b})^{\otimes n_s}_{\geq 0}} \langle \boldsymbol{C} + \varepsilon \log \boldsymbol{M}, \boldsymbol{M} \rangle \quad (13a)$$

$$\text{subject to} \quad \mathrm{proj}_\sigma(\boldsymbol{M}) = \boldsymbol{\mu}_\sigma \quad \forall \sigma \in [\![n_s]\!], \quad (13b)$$

where $\varepsilon > 0$ is the entropy regularization[4] parameter. We refer to the minimizer of (13) as the Multimarginal Schrödinger Bridge (MSB) and denote it as $\boldsymbol{M}_{\mathrm{opt}}$.

What makes the MSBP (13) pertinent to learning from distributional data is a mathematical result from the theory of large deviations [24], specifically Sanov's theorem [59], [29, Sec. II]. This theorem establishes that the minimizer of

[3] As opposed to the previously introduced $\eta^{i,j}$ notation, η^i, single-indexed, is used when context β^j is not of relevance, such as when computing cost.

[4] If $\varepsilon = 0$, then the objective (13a) becomes the probability mass transport cost $\langle \boldsymbol{C}, \boldsymbol{M} \rangle$, and (13) becomes the multimarginal Monge-Kantorovich optimal transport problem [54].

(13) is precisely the *most likely* joint distribution subject to the observational constraints (13b) imposed at times where measurements are available. In other words, solving (13) is equivalent to solving a *constrained maximum likelihood problem on the space of probability measure-valued curves* $t \mapsto \mu_t$ *traced over time*. Mathematical details for this maximum likelihood guarantee are provided in Supplementary Section S1 of the extended version of this paper [10].

B. Leveraging Duality to Solve Conditional MSBP

The MSBP (13) is a *strictly convex* program in $(n_d n_b)^{n_s}$ decision variables. Thus, the existence-uniqueness of the minimizer $\boldsymbol{M}_{\text{opt}}$ is guaranteed. Using strong Lagrange duality [13, Ch. 5.2.3], it follows that the MSB admits the structure

$$\boldsymbol{M}_{\text{opt}} = \boldsymbol{K} \odot \boldsymbol{U}, \quad \boldsymbol{K} = \exp\left(\frac{-\boldsymbol{C}}{\varepsilon}\right), \quad \boldsymbol{U} = \bigotimes_{\sigma \in [\![n_s]\!]} u_\sigma, \tag{14}$$

wherein $\odot$ denotes the elementwise product, and the vectors

$$u_\sigma := \exp(\lambda_\sigma/\varepsilon) \in \mathbb{R}^{n_d n_b}_{>0}$$

are the exponential-transformed Lagrange multipliers λ_σ associated with the equality constraints (13b). Note that the tensors $\boldsymbol{K}, \boldsymbol{U} \in (\mathbb{R}^{n_d n_b})^{\otimes n_s}_{\geq 0}$.

Thanks to this structure (14) for the optimal solution, we only need to determine the $\{u_\sigma\}_{\sigma \in [\![n_s]\!]}$, which are obtained by the Sinkhorn iterative scheme [63], [64]

$$u_\sigma \leftarrow u_\sigma \odot \mu_\sigma \oslash \text{proj}_\sigma\left(\boldsymbol{K} \odot \boldsymbol{U}\right) \qquad \forall \sigma \in [\![n_s]\!]. \tag{15}$$

In (15), the symbol $\oslash$ denotes elementwise division. The Sinkhorn recursions (15) are strictly contractive over the positive orthant $\mathbb{R}^{n_d n_b}_{>0}$ with respect to Hilbert's projective metric [9], [15], [39]

$$\texttt{Hilb}(p, q) := \log\left(\frac{\max_{i=1,\ldots,n} p_i/q_i}{\min_{i=1,\ldots,n} p_i/q_i}\right) \; \forall p, q \in \mathbb{R}^n_{>0}. \tag{16}$$

Therefore, by the Banach contraction mapping theorem [5], the recursions are guaranteed to converge to a unique fixed point with linear rate of convergence.

Notice also that both the formulation and the solution of the MSBP (13) are nonparametric in the sense that no statistical parametrization (e.g., mixture of Gaussian, exponential family, knowledge of moment or sufficient statistic) is assumed whatsoever on the joint $\boldsymbol{M}_{\text{opt}}$.

C. From Conditional MSBP to Generative Profiling

Maximum likelihood distributions. Once $\boldsymbol{M}_{\text{opt}}$ is computed using (14)-(15), we obtain the distributions μ_t in (5) as follows. Let

$$M^\sigma_{\text{opt}} := \text{proj}_{\sigma,\sigma+1}(\boldsymbol{M}_{\text{opt}}). \tag{17}$$

For $t \in [t_1, t_{n_s}]$, using (17), we compute

$$\mu_t = \sum_{i=1}^{n_d n_b} \sum_{j=1}^{n_d n_b} [(M^\sigma_{\text{opt}})_{i,j}] \delta\Big(\eta - \big((1-\lambda)\eta^i(t_\sigma) + \lambda \eta^j(t_{\sigma+1})\big)\Big) \tag{18}$$

where $t \in [t_\sigma, t_{\sigma+1}]$ and $\lambda := \frac{t - t_\sigma}{t_{\sigma+1} - t_\sigma} \in [0, 1]$. Each such μ_t is a weighted scattered distribution of $n_d^2 n_b^2$ data points, and represents the most likely distribution for $\eta(t)$ such that

Algorithm 1 Generative profiles via conditional MSBP

Require: Empirical profiles $\{\xi^{i,j}(t_\sigma)\}$ for $\beta^j \in \mathcal{B}'$, $(i,j) \in [\![n_d]\!] \times [\![n_b]\!]$, where snapshot index $\sigma \in [\![n_s]\!]$, profile granularity Δt, entropic regularization parameter $\varepsilon > 0$

$\eta^{i,j}(t_\sigma) \leftarrow \begin{pmatrix} \xi^{i,j}(t_\sigma) \\ \beta^j \end{pmatrix} \in \mathbb{R}^{m+b}$ ▷ augmented state samples
Construct $\{\mu_\sigma\}_{\sigma \in [\![n_s]\!]}$ using (6) ▷ empirical distributions
▷ construct $n_s - 1$ squared Euclidean distance matrices
for $j \in 1 : n_s - 1$ **do**
  $[C^j_{i_j, i_{j+1}}] \leftarrow \|\eta^{i_j}(t_j) - \eta^{i_{j+1}}(t_{j+1})\|_2^2$
end for

Construct $\boldsymbol{C}$ using (12) ▷ cost tensor
$\{u_\sigma\}_{\sigma \in [\![n_s]\!]} \leftarrow \texttt{MSBP\_Solve}(\boldsymbol{C}, \{\mu_\sigma\}_{\sigma \in [\![n_s]\!]}, \varepsilon)$

for $t \in t_1 : \Delta t : t_{n_s}$ **do**
  Compute μ_t via (17)-(18) ▷ predict joint distribution
  for $\beta \in \mathcal{B}$ **do**
    $\nu_t \leftarrow \mu_t / \left(\frac{1}{n_b} \sum_{j=1}^{n_b} \delta(\beta - \beta^j)\right)$ ▷ predict per (5)
    $\xi(t) \mid \beta \leftarrow \texttt{maxlikelihood}(\nu_t)$
  end for
end for

Result: Synthetic profiles $\{\xi(t) \mid \beta\}_{t \in t_1 : \Delta t : t_{n_s}}$ for all $\beta \in \mathcal{B}$

Algorithm 2 MSBP Solver `MSBP_Solve`

Require: Cost tensor $\boldsymbol{C}$, probability distributions $\{\mu_\sigma\}_{\sigma \in [\![n_s]\!]}$ entropic regularization parameter $\varepsilon > 0$, random vector generator `rand`, numerical tolerance `tol`, maximum number of iterations `maxiter`

$\boldsymbol{K} \leftarrow \exp\left(-\boldsymbol{C}/\varepsilon\right)$ ▷ scaled elementwise exponential
$u_\sigma(:, 1) \leftarrow \texttt{rand}_{n_d n_b \times 1} \; \forall \sigma \in [\![n_s]\!]$ ▷ random init.
$\texttt{idx} \leftarrow 1$ ▷ initialize Sinkhorn recursion index
$\texttt{err}_\sigma(\texttt{idx}) \leftarrow 1_{n_d n_b \times 1} \; \forall \sigma \in [\![n_s]\!]$ ▷ initialize errors
while $(\max_{\sigma \in [\![n_s]\!]} \texttt{err}_\sigma(\texttt{idx}) > \texttt{tol})$ **and** $(\texttt{idx} < \texttt{maxiter})$ **do**
▷ Sinkhorn recursion (15)
  for $\sigma \in [\![n_s]\!]$ **do**
    $u_\sigma(:, \texttt{idx}{+}1) \leftarrow u_\sigma(:, \texttt{idx}) \odot \mu_\sigma \oslash \text{proj}_\sigma(\boldsymbol{K} \odot \boldsymbol{U})$
    $\texttt{err}_\sigma(\texttt{idx} + 1) \leftarrow \texttt{Hilb}(u_\sigma(:, \texttt{idx}), u_\sigma(:, \texttt{idx}{+}1))$ ▷ error in Hilbert metric (16)
  end for
  $\texttt{idx} \leftarrow \texttt{idx} + 1$
end while

Result: Converged $\{u_\sigma\}_{\sigma \in [\![n_s]\!]}$ defining the MSB $\boldsymbol{M}_{\text{opt}}$

$\eta(t_\sigma) \sim \mu_\sigma$ for all $\sigma \in [\![n_s]\!]$ and given transport cost $\boldsymbol{C}$. In other words, $\eta(t)$ is the least assumptive model which explains the data (μ_σ and $\boldsymbol{C}$, in this case). We provide more details on the maximum-likelihood guarantee in Sec. S1 of [10].

Notice that the unimarginal projection (8) is needed for the Sinkhorn recursion (15), while the bimarginal projection (10) is needed for the post-processing (17). Efficient computation

of these projections is detailed in Sec. S2 of [10].

Computational complexity. The complexity of the Sinkhorn recursion (15) is governed by that of the unimarginal projections (8) which, in general, is exponential in n_s. As shown in previous work ([11], [12]), this complexity becomes *linear* in n_s when we take into account the temporal path structure (12) of our cost (see Sec. S2 of [10] for details).

Thus, we may solve (13) with computational complexity $\mathcal{O}\left((n_s-1)n_d^2 n_b^2\right)$. This linear complexity with respect to n_s is particularly significant considering that the primal formulation (13) has an exponential complexity $\mathcal{O}\left((n_d n_b)^{n_s}\right)$.

Maximum likelihood profiles. Having computed both the numerators and denominators in (5), we obtain the conditional distributions of the random vector $\xi(t) \mid \beta$. By re-sampling the computed conditional distributions (5), we are able to generate high conditional probability samples, and therefore generative or synthetic profiles. This re-sampling can, for instance, be done by simply returning the top few high probability samples since the distribution values evaluated at the samples, and not just the samples of (5), are available from MSB computation. Alternatively, this re-sampling can be done using existing diffusion models [66], [67], [68] for generative AI.

The overall computational framework discussed above is summarized in Algorithm 1. First, we form the distributions $\{\mu_\sigma\}$ from our empirical profiles for the known context set $\mathcal{B}'$ (lines 1-2). Second, we construct the cost tensor $\boldsymbol{C}$ from the squared-Euclidean ground costs between known support points η (lines 3-7). On line 8, we invoke an MSBP solver (details in Algorithm 2) to obtain the vectors $\{u_\sigma\}$, which may be used as in (14) to obtain the MSB $\boldsymbol{M}_{\text{opt}}$. Finally in lines 10-16 we create the synthetic profiles – for each time t of interest (e.g., small timesteps Δt from initial time t_1 to t_{n_s}) use the MSB to obtain the joint distribution μ_t of $\eta(t)$ (line 11), then therefrom for all $\beta \in \mathcal{B}$ the conditional distribution ν_t of $\xi(t) \mid \beta$ (line 13), which we sample in line 14 to obtain the execution state $\xi(t) \mid \beta$.

Remark. *Algorithm 1 returns the* maximum-likelihood *synthetic profiles for the given task. The* `maxlikelihood` *oracle in line 14 of the algorithm can be modified to produce, for example,* mean *synthetic profiles by extracting the mean of the conditional distribution. Additionally, a random sampling scheme may be used to generate multiple high-probability random synthetic profiles.*

The MSBP Solver. Algorithm 1 makes use of an MSBP solver, which we detail in Algorithm 2. The solver forms $\boldsymbol{K}$ from $\boldsymbol{C}$ and initial random guesses for the vectors $\{u_\sigma\}$ (lines 1-2). The loop on lines 5-11 performs the Sinkhorn iterations (15) until all vectors u_σ converge in the Hilbert metric. These vectors are then returned as the solution (recall that by (14), the MSB $\boldsymbol{M}_{\text{opt}}$ is thereby uniquely defined). Note that line 7 requires the unimarginal projection to be computed – this is the greatest computational bottleneck of the algorithm, and also where the structure of $\boldsymbol{C}$ is often exploited to greatly accelerate the computation.

Computational Considerations. Practical implementations of Algorithms 1 and 2 benefit from considering the computational stability thereof. In MSBP solvers (Algorithm 2), the parameter ε is generally chosen to be a small, positive value; larger values lead to faster convergence of the solver at the cost of noisier solutions. The dual variables u_σ may be initialized to any positive values. In practice a random initialization within $(0, 1)$ is used. Similarly, any positive cost tensor $\boldsymbol{C}$ may be used – we use the Euclidean cost (line 4, Algorithm 1), as is common throughout the related literature [11], [12]. Since components of η tend to vary greatly in scale, however, for each snapshot time t_σ we scale the η componentwise into the range $[0, 0.1]$. The scaling is done so that the Euclidean distance takes into equal account all components thereof, and the range is chosen for stability of Algorithm 2 (large values of $\eta \to$ large values in $\boldsymbol{C} \to$ near-zero values in $\boldsymbol{K}$, which can lead to underflow). Finally, μ_t as computed in line 11 of Algorithm 1 implicitly needs to have support covering all of $\mathcal{B}$ for line 13 to be feasible. To ensure this, we employ a standard KDE alignment of μ_t as computed via (17)-(18). Other domain alignment approaches may be used as well.

IV. Effectiveness of generative profiling

A. Benchmarks, Hardware, and Workload Measurement

Benchmarks. To evaluate generative profiling, we used the PARSEC [8] benchmark suite, which includes programs with diverse execution characteristics. This suite has been widely used in prior work on scheduling and resource allocation (e.g., [30], [36], [75]). We ran these benchmarks in single-threaded mode, with the `smallsim` input. The observed execution times ranged from approximately 100 milliseconds to several seconds on our experimental platform.

Hardware. We performed measurements on an Intel Xeon E5-2618L v3 processor with 8 cores, 20MB 20-way set-associative L3 cache, and a single channel 8GB PC-2133 DDR4 DRAM. The machine supports both Cache Allocation Technology (CAT) [33] for cache allocation and DVFS for power management. CAT divides the shared L3 cache into $N_{\mathsf{ca}} = 20$ equal-size partitions. Using the method in [75], we measured a maximum guaranteed bandwidth of 1.4 GB/s on each machine, which we divided into $N_{\mathsf{bw}} = 20$ partitions of 70MB/s each using MemGuard [75]. We considered $N_{\mathsf{freq}} = 12$ CPU frequency settings, ranging from 1.2GHz to 2.3GHz, at steps of 0.1GHz. Since the machine requires a minimum of 2 cache partitions per allocation, it supports $(N_{\mathsf{ca}} - 1) \times N_{\mathsf{bw}} \times N_{\mathsf{freq}} = 19 \times 20 \times 12 = 4560$ different resource contexts in total. To ensure deterministic timing, we disabled cache prefetching and CPU hyperthreading via the system's BIOS.

Measurements. To gather the empirical profiles used as input to our model, we ran each workload on a dedicated core under the desired resource context, and collected data for 100 runs. In each run, we used the CPU's performance counters to periodically record the number of instructions retired, cache requests, and cache misses (ξ_1, ξ_2, ξ_3, respectively in our model), once every 10 ms.

Our model requires only a small subset of empirical profiles for a small number of resource contexts as training inputs.

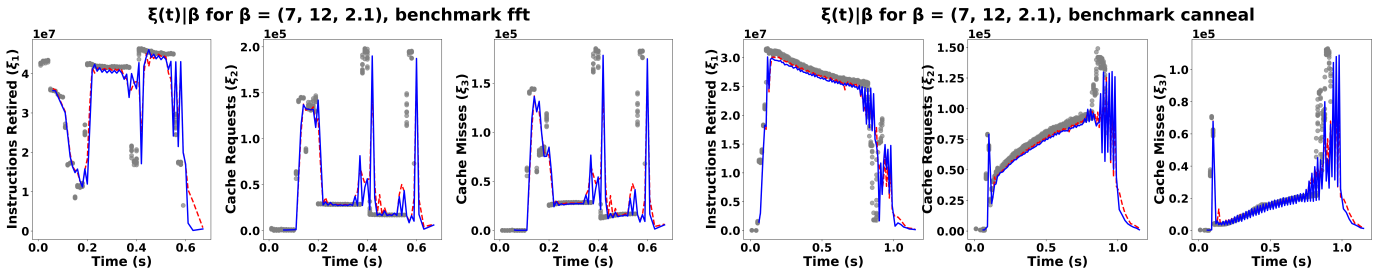


Fig. 5: Maximum-likelihood synthetic profile (blue), mean synthetic profile (red), and all empirical profiles (grey) for the benchmarks fft and canneal when $\beta = (7,\ 12,\ 2.1)^\top$ on our default experimental platform.

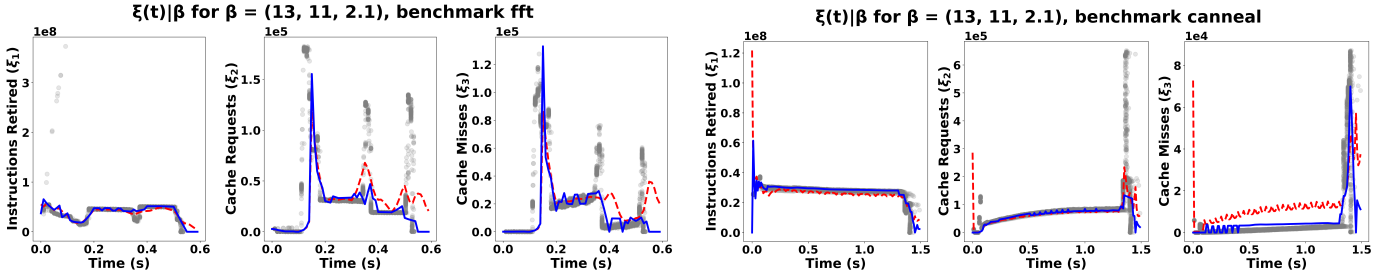


Fig. 6: Maximum-likelihood synthetic profile (blue), mean synthetic profile (red), and all empirical profiles (grey) for the benchmarks fft and canneal when $\beta = (13,\ 11,\ 2.1)^\top$ on PlatformLarge.

However, to evaluate its accuracy, we also needed the "ground-truth" data. To obtain the ground-truth profiles, we performed measurements for all possible resource contexts for each benchmark program. The whole measurement process took us over two months to complete. Since we collected 100 profiles per context, the ground-truth data contain 456,000 empirical profiles per benchmark.

B. Accuracy of Generative Stochastic Model

Methodology. Given the empirical profiles, we used the approach detailed in Section III to generate synthetic profiles for $\xi(t) \mid \beta$. For the experiment, we restricted the number of resource contexts β and the number of profiles n_d available to our MSBP solver. Specifically, for

$$\mathcal{B} = \{2, 3, \ldots, 20\} \times [\![N_{\mathsf{bw}}]\!] \times \{1.2, 1.3, \ldots, 2.3\}$$

we took a subset $\mathcal{B}'$ of $\mathcal{B}$ as training data.

We then let $n_d = 10$, i.e., we took 10 random profiles from the 100 available for each unique $\beta \in \mathcal{B}'$. From these limited profiles, we constructed the marginal distributions (6) for times $t_\sigma = 0.05 \cdot (\sigma - 1)$, with t_{n_s} chosen depending on the runtime of the benchmark. Thus, with $n_b = |\mathcal{B}'| = 125$, each snapshot μ_σ had $n_d n_b = 1250$ scattered data points. For all experiments, our MSBP solver was configured with $\varepsilon = 0.1$, $\texttt{tol} = 1e{-}12$, and $\texttt{maxiter} = 1e4$. Our dual variables u_σ were initialized by $\texttt{rand}$ = uniform samples from $(0, 1)$.

Remark. *The restriction of the number of profiles (n_d) and the number of resource contexts (n_b) provided as input to the MSBP solver has a practical motivation – if either the number of tasks to profile or the number of possible resource contexts $|\mathcal{B}|$ is large, it is intensive both in time and computation to empirically generate a large number of profiles for all workloads and for each resource context. Our method allows for a significant reduction in profiling time as it requires only a small subset of $\mathcal{B}$ and a small n_d to generate synthetic profiles for all $\beta \in \mathcal{B}$. For instance, for each benchmark task, only $n_d n_b = 1250$ empirical profiles with data observations every 50 ms were sufficient to achieve an accuracy near the ground truth consisting of 456,000 empirical profiles with data gathered every 10 ms (i.e., only 0.27% of the original empirical data was used by our model). Moreover, running MATLAB R2024b on a machine with an AMD Ryzen 7 5800X CPU and 80GB of memory, the entire process of MSBP solution and synthetic profile generation for all $\beta \in \mathcal{B}$ took approximately* **15 minutes** *per benchmark task. In contrast, collecting the respective empirical profiles for all possible $\beta \in \mathcal{B}$ took many days for each benchmark task (e.g., approximately 2 weeks for the* canneal *benchmark).*

Solving the MSBP over the marginals $\{\mu_\sigma\}_{\sigma \in [\![n_s]\!]}$, we generated the most likely conditional joint distributions $\mu_t / \int_{\mathcal{X}} \mu_t d\xi$ per (5), at times from 0 to t_{n_s} every 10 ms, and for each $\beta \in \mathcal{B}$. As each of these joints are *weighted* scattered distributions, we obtained from their aggregate a synthetic profile of the benchmark conditioned on β by taking $\xi(t) \mid \beta$ to be the data point with the highest probability value in the respective conditional joint distribution, for each $t \in \{0, 0.01, 0.02, \ldots, t_{n_s}\}$. We call this the *maximum-likelihood* generative profile.

Generative profiles visualized. Fig. 5 shows the mean and maximum likelihood generative profiles overlaid on 100 empirical profiles for two example benchmarks (fft and canneal) under a resource context that was farthest from those in the training set, $\beta = (7, 12, 2.1)^\top$. We present these sample results visually to highlight the fine-grained, time-dependent

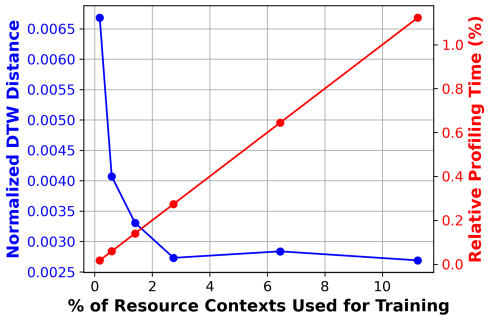


Fig. 7: Accuracy and relative profiling time vs. % of resource contexts used for training (workload canneal).

predictive accuracy of our method. Notice that the generative profiles closely align with the empirical profiles, even for contexts in which β is not available to the MSBP solver.

Generality across platforms. To illustrate the generality of our generative profiling, we also evaluated its performance on another, more powerful, multicore platform with more cores and larger cache and memory bandwidth sizes. This platform, referred to as PlatformLarge, has an Intel Xeon E5-2683 v4 processor with 16 cores, 40MB L3 cache, and three single-channel 16GB PC-2400 DDR4 DRAMs. We repeated the same methodology as described in the previous experiment.

Fig. 6 shows the generative profiles overlaid on the raw empirical data for benchmarks fft and canneal under a resource context of $\beta = (13, 11, 2.1)^\top$ on PlatformLarge. We observe that our generative profiles accurately capture the time-varying execution characteristics for contexts outside of the training set $\mathcal{B}'$ and across different hardware platforms.

Remark. *It may seem natural to choose the* mean *value of each conditional joint distribution to construct our generative profiles. Thus, Fig. 5 and Fig. 6 also show generative profiles constructed in this way. As can be seen most clearly in the 'tail-end' behavior of the mean profiles for* fft *in Fig. 6, however, this choice can lead to inaccurate inference when μ_t is non-Gaussian. In such cases, first few moments, such as the mean and covariance, are misleading since the statistical typicality differs from the average. As the MSB allows us to obtain the* maximum-likelihood *distributions μ_t in a nonparametric manner, we can obtain accurate results without any assumptions on the nature of these distributions.*

Impact of the number of resource contexts for training. It is difficult to meaningfully assess the absolute accuracy of our generative profiles in a quantitative manner, since the evaluation of any metric on the space of curves in $\mathbb{R}^2$ will depend on the magnitude of the curves themselves. In Fig. 7, we illustrate the impact of increasing the number of empirical resource contexts used for training (i.e., increasing n_b, while keeping n_d unchanged) on the normalized Dynamic Time Warping (DTW) distance [6] (shown in blue) between the ground-truth empirical profiles and the maximum-likelihood generative profiles for the canneal benchmark. We use the DTW distance as it is a common metric for evaluating the similarity between time series. For each context β, we computed the DTW distance between the generated profile and the average empirical profile using the `fastdtw` Python library [58]. To normalize the result, we divided it by the length of the DTW reference path (the average empirical profile) multiplied by the maximum norm of the execution state vector ξ along the path. We also display the measurement time (in red) for collecting the training data, normalized relative to that of the full empirical profiles.

TABLE I: Per-benchmark accuracy (normalized DTW) of baseline vs. generated profiles, both trained on $\approx 6\%$ of resource contexts, averaged over all $\beta \in \mathcal{B}$. Smaller DTW indicates better accuracy. Improvement % is computed via $100 \times$ (baseline DTW − generative DTW) / baseline DTW.

	Baseline	Generative	Improvement
blackscholes	0.0429	0.0343	20.0%
bodytrack	0.0437	0.0375	14.2%
canneal	0.0227	0.0027	88.1%
dedup	0.0321	0.0191	40.5%
fft	0.0508	0.0478	5.9%
fluidanimate	0.0357	0.0275	23.0%
radiosity	0.0404	0.0380	5.9%
streamcluster	0.0549	0.0418	23.9%

We observe that, as the proportion of resource contexts used for training increases, the DTW distance decreases sharply, while the profiling time increases linearly. The DTW distance stabilizes at approximately 0.00273 when using only about 3% of the resource contexts, and adding more contexts beyond this provides little to no benefit. In the case of canneal, this corresponds to only 52 minutes of profiling time compared to approximately 2 weeks to collect the full set of empirical data. In other words, our method can generate accurate profiles for unseen resource contexts using only a tiny fraction of time.

Accuracy results for all benchmarks and contexts. Table I shows the per-benchmark accuracy (reported as the normalized DTW and averaged over all $\beta \in \mathcal{B}$) of generative profiles compared to a baseline that interpolates between known resource contexts. Specifically, for each unknown context β, the baseline finds the two resource contexts $\beta', \beta'' \in \mathcal{B}'$ such that $\beta'_i \leq \beta_i \leq \beta''_i \ \forall i \in \{1, 2, \ldots, b\}$, and averages their profiles at each snapshot to produce profiles for β.[5] We include this interpolation baseline as it represents a natural approach for estimating execution behavior from sparse resource contexts (for example, [26] leverages a similar approach to predict execution behavior under sparse LLC allocations). Both the generative profiles and the baseline were trained on approximately 6% of the resource contexts.

Observe that, for all benchmarks in Table I, our generative profiles are closer to the empirical profiles than the baseline in terms of average normalized DTW distance, by up to 88.1% and by 27.7% on average. The results demonstrate that our technique more accurately captures the time-varying execution characteristics of each task.

[5]Since the set $\mathcal{B}'$ is chosen such that it contains the minimum and maximum resource contexts, β' and β'' always exist.

V. Resource Allocation with Generative Profiles

A. Case Study: Dynamic Multicore Resource Allocation

To demonstrate the practical utility of our technique, we present a representative case study of generative profiles in adaptive multicore resource allocation for real-time systems. Towards this, we propose a frequency-aware extension of DNA [30], a phase-aware resource allocation method that dynamically allocates cache and memory bandwidth to tasks at fine-grained time intervals based on their profiles. Given a task's profiles, DNA uses clustering to identify a series of phases for each resource context. Each phase is a contiguous sequence of instructions with similar execution metrics (rates of instruction retirement, cache requests and cache misses). At run time, DNA utilizes this context-dependent phase information to find an allocation that aims to maximize the total instruction rate of the system. DNA has been shown to substantially improve schedulability, reduce deadline misses, and decrease latencies compared to static allocation [30].

DNA only controls cache and memory bandwidth allocation, while assuming the system operates at a static CPU frequency, typically the maximum frequency. However, as shown in Fig. 1, the instruction rate of a task in a phase is influenced by the CPU frequency, and this relationship varies across different phases and different cache and memory bandwidth allocations. For example, increasing the frequency may improve the instruction rate in a CPU-bound phase but have little to no effect in a memory-bound phase. This presents an opportunity to reduce energy consumption. Therefore, we introduce a phase-based DVFS extension of DNA, which uses task profiles to determine the optimal frequency for each phase under each cache and memory bandwidth allocation. Specifically, it selects the minimum frequency for each phase of a task such that the instruction rate in the phase does not increase by more than a threshold ratio $\epsilon > 0$, compared to the rate achieved under the static maximum frequency f_{max}.

Define the *reference utilization* of a task (or taskset) to be the utilization under the maximum resource context $\beta_{\max} = (N_{\mathsf{ca}}, N_{\mathsf{bw}}, f_{\mathsf{max}})$. In general, schedulability under global EDF depends on the maximum per-task utilization as well as the total taskset utilization [31]. Therefore, the smaller these values are for a taskset, the larger we can scale the taskset's threshold ϵ to enable more power savings. If the taskset is deemed "unschedulable" based on the reference utilization, we set $\epsilon = 0$. Otherwise, we select the largest ϵ such that the taskset still remains schedulable (based on the reference utilization). For convenience, we refer to this extension of DNA as DVFS-DNA and DNA with static maximum frequency as Static-DNA.

B. Prototype Implementation in Linux

Overview. For our experiments, we implemented a prototype of DNA and its DVFS extension on Linux 6.12 with the real-time patch `PREEMPT_RT` enabled.[6] We implemented DNA as a loadable kernel module, comprising approximately 1,900 lines of C code. The module contains the core DNA algorithm, timer support, mechanisms for applying cache partitions, and calls into MemGuard to enforce memory-bandwidth partitions. In addition, we added about 150 lines to Linux's in-tree codebase to track DNA metadata within task structures, such as workload type and current phase information. We also provide user-level runtime scripts to load task phase information, generate task sets, and launch experiments. The run script uses cgroup v2 to confine real-time tasks to a subset of cores and synchronizes release times via the cgroup v2 freezer controller. DNA is agnostic to the CPU scheduling policy—it simply identifies the current phase of each running task and uses that information to compute the number of cache and memory-bandwidth partitions per core.

[6] A prototype of DNA [30] exists on Xen; however, to avoid virtualization effects, we built our own prototype directly in Linux.

TABLE II: Runtime Overhead of DNA in μs

Count	Mean	90th	95th	99th	Max
28347	1.068	1.536	2.008	6.204	10.727

DNA invocation mechanism. To invoke DNA, we used a periodic timer handler that executes on CPU 0 and is configured to fire every 5 ms. Upon each firing, CPU 0 identifies the currently executing task on each experimental core and determines its current phase; if some task has entered a new phase, CPU 0 invokes DNA to compute a new resource allocation and then applies it for all cores. In addition, we modified Linux's context switch mechanism to invoke DNA whenever a new task is assigned to a CPU, to ensure that a newly running task always receives a correct allocation.

Resource and frequency assignment. We leveraged Intel's CAT hardware technology [34] and MemGuard [75] for cache and memory bandwidth allocation. Using CAT, we can efficiently apply a cache allocation to CPUs by writing to MSR registers. In contrast, MemGuard relies on hardware performance counters to monitor the number of L3 cache misses on each CPU over fixed time periods; if a CPU's number exceeds its assigned budget, MemGuard throttles it by scheduling a "MemGuard" thread, which spins until the next replenishment period. To work correctly, the "MemGuard" threads must have the highest priority, which does not hold under `SCHED_DEADLINE`. To address this, we modified MemGuard's throttle handler to avoid scheduling the "MemGuard" thread. Instead, it sets a special throttled bit on the affected CPU, triggers a reschedule on that CPU, and returns. The Linux scheduler was updated to detect this bit: when set, the `SCHED_DEADLINE` scheduler class is bypassed, so that MemGuard's throttle thread can run until the bit is cleared, after which normal scheduling resumes. Additionally, we adapted MemGuard to work under `PREEMPT_RT`.

To enable frequency scaling, we extended the DNA module to set the CPU frequency each time DNA is invoked. This enables us to set the CPU frequency at the start of each phase, as is given by DVFS-DNA for the current cache and memory-bandwidth allocation (c.f. Sec. V-A).

Run-time overhead. Table II shows the runtime overhead. On average, it takes only about 1 μs to compute an allocation

TABLE III: Avg. End-to-end Measurement & Generation Time

	Total Time (hrs)	# Empirical Profiles
Empirical	231	456,000
Baseline	0.64	1,250
Generative	1.14	1,250

across all cores using our prototype, which is minimal.

We next present an experimental evaluation of generative profiles using this prototype and the discussed case study.

VI. Experimental Evaluation

A. Experimental Setup

Tasksets. We generated 20 tasksets, with taskset reference utilizations ranging between 0.2 and 4.0, at steps of 0.2. To generate a taskset, we iteratively added tasks, each with a utilization uniformly sampled from $[0.1, 0.4]$, into the taskset until its total utilization is at least the target utilization. The final task's utilization was adjusted such that the taskset utilization equals its target utilization. For each task, we then randomly picked one of the benchmarks in the PARSEC suite as its workload. The task's profiles are obtained from the measurements discussed in Sec. IV-A. We set the task's period (relative deadline) to be its *reference* WCET—defined as the observed WCET under the maximum resource context $\beta_{\max} = (N_{\mathsf{ca}}, N_{\mathsf{bw}}, f_{\mathsf{max}})$—divided by its utilization.

Experiments. We executed each taskset under `SCHED_DEADLINE` on 4 experimental cores of our default multicore platform, with DNA prototype running. Each task's budget was set equal to its period. Jobs were released periodically based on their periods over a one-minute interval and executed to completion. We recorded each job's release and completion time to compute its response time. We repeated this experiment for all generated tasksets. In total, we evaluated 400 unique tasksets.

Methods under evaluation. We considered three different inputs to our resource allocation algorithm: (1) complete *empirical* profiles; (2) our *generative* profiles learned from a small subset $\mathcal{B}'$ (containing 0.27%) of the empirical profiles; and (3) *baseline* profiles, which consist of the same subset $\mathcal{B}'$ of empirical profiles used to produce our generative profiles, but with the profiles for each unknown context created using the simple interpolation method described in the accuracy evaluation of Sec. IV-B.

Metrics. For each method, we evaluated the *observed* schedulability and energy savings under DVFS-DNA. For comparison, we also evaluated Static-DNA with the highest CPU frequency setting ($f_{\mathsf{max}} = 2.3$ GHz), using the full empirical profiles. To capture the effect of dynamic frequency scaling, we measured the *energy savings* based on dynamic power consumption, obtained by subtracting the static power consumption (measured when all CPUs were idle) from the total power consumption reported by Intel's `RAPL` energy registers [35]. The energy consumption for an experiment run was calculated as the product of the average CPU dynamic power consumption and the duration of the experiment.

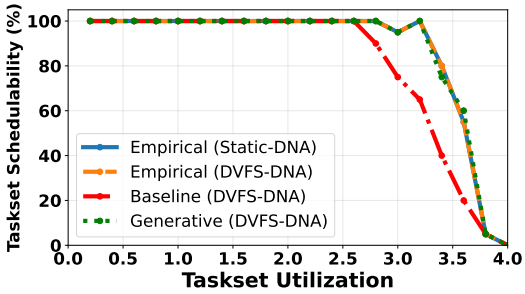


Fig. 8: Taskset Schedulability

B. Results

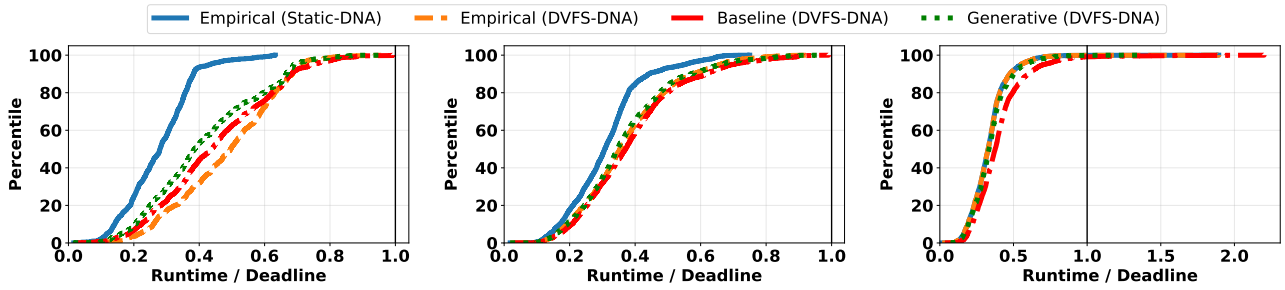
Schedulability results. Fig. 8 shows the percentages of schedulable tasksets across different taskset utilizations for DVFS-DNA under the three input methods (generative, baseline, empirical), as well as Static-DNA running with full empirical profiles. We observe high schedulability for both empirical and generative profiles. For instance, the first unschedulable taskset only occurs at 3.0 utilization (i.e., 75% CPU load). The results highlight the benefits of context-dependent fine-grained profiles in dynamic resource allocation.

Notably, the performance of generative profiles closely matches the empirical profiles' performance, demonstrating that our generative profiling is effective in practical applications. Table III further demonstrates the average end-to-end time needed to measure empirical and produce generative profiles per benchmark. Notice that the generative profiling method requires just over an hour to collect training data and generate profiles per benchmark, compared to 231 hours (9.6 days) to obtain the complete empirical profiles. In other words, it is 230× more efficient while delivering comparable schedulability compared to using complete empirical profiles.

Finally, we observe that generative profiles substantially outperform baseline profiles in terms of schedulability. Overall, the generative profiling method schedules 1.63× more tasksets than the baseline between 3.0 and 4.0 utilization. This shows that profiling a limited number of resource contexts and filling in the gaps with a simple interpolation method is insufficient to achieve equivalent real-time performance (compared to full empirical profiles).

Response time results. Fig. 9 shows the CDFs of job response time over deadline at three different representative taskset utilizations (low, medium and high). The vertical line at 1.0 represents the point at which a job's response time equals its deadline. Again, we observe that our generative profiles achieve nearly identical job response times as empirical profiles do under DVFS-DNA, and the results are consistent across different taskset utilizations.

Notice that the job response times when using empirical profiles under Static-DNA are overall smaller than those obtained under DVFS-DNA. This is expected since Static-DNA uses the maximum frequency setting while DVFS-DNA aims to save energy. As the utilization increases, however, the gap between them becomes closer, indicating



(a) Latency CDF at Utilization 1.4 (b) Latency CDF at Utilization 2.4 (c) Latency CDF at Utilization 3.4

Fig. 9: DNA Performance Comparison using Generative vs. Baseline vs. Empirical Profiles.

DVFS-DNA's ability to scale the frequency to workload demands. On the contrary, the baseline method experiences noticeably longer tail response times, especially at higher utilizations. This is consistent with the observed schedulability results and further highlights the limitation of baseline profiles.

Energy saving results. We also evaluated the dynamic energy consumption of DVFS-DNA under each profiling method compared to that of Static-DNA with full empirical profiles. The results show that DVFS-DNA substantially reduces the energy consumption for all methods compared to Static-DNA. On average, DVFS-DNA reduces the total energy consumption by 18.1% under empirical profiles and 14.8% under generative and baseline profiles. More importantly, for both generative and empirical profiles, these savings are achieved while maintaining the same schedulability performance as Static-DNA (described earlier). The results demonstrate the needs and benefits of considering all three resource types (frequency, cache, memory bandwidth) in real-time resource allocation, which can now be achieved efficiently with generative profiles.

VII. Related Work

Context-dependent profiles. Context-dependent profiles are increasingly being used in multicore real-time systems. For example, existing techniques have utilized resource-aware task WCETs to perform task mapping and shared resource allocation [74], [73]. As described in Section V-A, DNA (and its deadline-aware variant DADNA) [30] further exploit fine-grained profiles to dynamically adapt resources allocated to a task to maximize execution progress, while [25] co-designs fine-grained resource allocation with scheduling. Likewise, recent work in real-time DVFS [4], [38] has started to explore phase-based or workload-aware DVFS techniques [1], [40], [32], [19]. Other work leverages context-dependent profiles to improve general system performance and/or fairness via resource allocation [53], [55], [56], [57]. However, the above techniques often require extensive profiling of each task under each resource context to make allocation decisions. For multicore systems with many tasks and multiple resource types, such extensive profiling requirements can prohibit the adoption of these techniques.

One recent solution for generating context-dependent profiles uses limited empirical profiling to estimate WCETs under unseen memory bandwidth allocations [65]. However, its analysis requires detailed knowledge of the throttling mechanism (e.g., MemGuard), fixes all other resources, and focuses on memory transactions. Since our goal is to produce a maximum likelihood profile (not a worst-case timing analysis), our method is more general and directly predicts, without such restrictions, the evolution of a program's *multi-dimensional* micro-architectural state under *multi-dimensional* unseen resource contexts.

Learning-based prediction in scheduling. There is a substantial body of work that applies statistical methods to estimate probabilistic response times or deadline misses [47], [45], [44], [46], although much of this work is restricted to the prediction of course-grained information such as job size and end-to-end response time. Recently, Marković et al. [50] also used nonparametic learning to estimate statistical upper bounds on the mean, standard deviation and covariance of task execution time. In contrast to their work, we consider the much finer-grained microarchitectural execution characteristics and the time-varying stochastic relationship between multidimensional resource usage, conditioned on an allocated resource context. Prior work [11], [12] considered the former, but did not address the dynamic correlation between a software's stochastic time-varying resource usage and the resources allocated to the software, as we do in this work.

MSB. The formulation and usage of MSB for statistical inference and learning, is a relatively recent endeavor [27], [51], [20], [11], [12]. The MSB can be understood as the stochastic version of the multi-marginal optimal transport [54], which corresponds to vanishing entropic regularization ($\varepsilon \downarrow 0$). The convergence of the multi-marginal Sinkhorn algorithms in the continuous state space was established in [49], [18].

VIII. Conclusion

We have presented a generative stochastic model for nonparametrically generating context-dependent profiles for tasks in complex real-time systems. Our solution requires only a small subset of empirical data to generate profiles with high accuracy, including for unseen resource contexts. It achieves orders of magnitude reduction in measurement time while providing statistical guarantees. Through case studies and experimental evaluations, we demonstrated the practical application of our approach and its performance benefits in real-world real-time systems.

Acknowledgment

This work was supported in part by NSF grants CNS-1955670, CNS-2111688 and CCF-2326606.

References

[1] B. Acun, K. Chandrasekar, and L. V. Kale. Fine-grained energy efficiency using per-core dvfs with an adaptive runtime system. In *IGSC*, 2019.

[2] S. Amari. *Information geometry and its applications*, volume 194. Springer, 2016.

[3] ARM. PrimeCell level 2 cache controller (PL310) - technical reference manual. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0246c/index.html. Accessed: 2025-05-23.

[4] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 2004.

[5] S. Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta mathematicae*, 3(1):133–181, 1922.

[6] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD*, 1994.

[7] D. Bertsekas and J. N. Tsitsiklis. *Introduction to probability*. Athena Scientific, second edition, 2008.

[8] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *PACT*, 2008.

[9] G. Birkhoff. Extensions of Jentzsch's theorem. *Transactions of the American Mathematical Society*, 85(1):219–227, 1957.

[10] G. A. Bondar, A. Eisenklam, Y. Cai, R. Gifford, T. Sial, L. T. X. Phan, and A. Halder. Generative profiling for soft real-time systems and its applications to resource allocation. Technical report, Department of Computer and Information Science, University of Pennsylvania, 2026. http://www.cis.upenn.edu/~linhphan/rtas26-genprofile-extended.pdf.

[11] G. A. Bondar, R. Gifford, L. T. X. Phan, and A. Halder. Path structured multimarginal Schrödinger bridge for probabilistic learning of hardware resource usage by control software. In *ACC*, 2024.

[12] G. A. Bondar, R. Gifford, L. T. X. Phan, and A. Halder. Stochastic learning of computational resource usage as graph structured multimarginal Schrödinger bridge. *arXiv preprint arXiv:2405.12463*, 2024.

[13] S. P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[14] S. Bozhko, G. von der Brüggen, and B. Brandenburg. Monte carlo response-time analysis. In *RTSS*, 2021.

[15] P. J. Bushell. Hilbert's metric and positive contraction mappings in a Banach space. *Archive for Rational Mechanics and Analysis*, 52:330–338, 1973.

[16] Y. Cai, L. T. X. Phan, and P. Thiagarajan. Analysis of long-term average behaviors of probabilistic task systems. In *RTNS*, 2024.

[17] K. F. Caluya and A. Halder. Wasserstein proximal algorithms for the Schrödinger bridge problem: Density control with nonlinear drift. *IEEE Transactions on Automatic Control*, 67(3):1163–1178, 2021.

[18] G. Carlier. On the linear convergence of the multimarginal Sinkhorn algorithm. *SIAM Journal on Optimization*, 32(2):786–794, 2022.

[19] J. Chen, M. Manivannan, B. Goel, and M. Pericàs. Sweep: Adaptive task scheduling for exploring energy performance trade-offs. In *IPDPS*, 2024.

[20] T. Chen, G.-H. Liu, M. Tao, and E. Theodorou. Deep momentum multi-marginal Schrödinger bridge. *Advances in Neural Information Processing Systems*, 36:57058–57086, 2023.

[21] Y. Chen, T. T. Georgiou, and M. Pavon. Stochastic control liaisons: Richard Sinkhorn meets Gaspard Monge on a Schrödinger bridge. *Siam Review*, 63(2):249–313, 2021.

[22] R. I. Davis and L. Cucu-Grosjean. A survey of probabilistic timing analysis techniques for real-time systems. *Leibniz Trans. Embed. Syst.*, 6(1):03:1–03:60, 2019.

[23] V. De Bortoli, J. Thornton, J. Heng, and A. Doucet. Diffusion Schrödinger bridge with applications to score-based generative modeling. *Advances in Neural Information Processing Systems*, 34:17695–17709, 2021.

[24] A. Dembo and O. Zeitouni. *Large deviations techniques and applications*, volume 38. Springer Science & Business Media, 2009.

[25] A. Eisenklam, R. Gifford, G. A. Bondar, Y. Cai, T. Sial, L. T. X. Phan, and A. Halder. Rasco: Resource allocation and scheduling co-design for DAG applications on multicore. *ACM TECS*, 24(5s), 2025.

[26] N. El-Sayed, A. Mukkara, P.-A. Tsai, H. Kasture, X. Ma, and D. Sanchez. Kpart: A hybrid cache partitioning-sharing technique for commodity multicores. In *HPCA*, 2018.

[27] F. Elvander, I. Haasler, A. Jakobsson, and J. Karlsson. Multi-marginal optimal transport using partial information with applications in robust localization and sensor fusion. *Signal Processing*, 171:107474, 2020.

[28] P. Esser, S. Kulal, A. Blattmann, R. Entezari, J. Müller, H. Saini, Y. Levi, D. Lorenz, A. Sauer, F. Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first international conference on machine learning*, 2024.

[29] H. Follmer. Random fields and diffusion processes. *Ecole d'Ete de Probabilites de Saint-Flour XV-XVII, 1985-87*, 1988.

[30] R. Gifford, N. Gandhi, L. T. X. Phan, and A. Haeberlen. DNA: Dynamic resource allocation for soft real-time multicore systems. In *RTAS*, 2021.

[31] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-time systems*, 25(2):187–205, 2003.

[32] S. Hajiamini, B. Shirazi, A. Crandall, and H. Ghasemzadeh. A dynamic programming framework for DVFS-based energy-efficiency in multicore systems. *IEEE Transactions on Sustainable Computing*, 2020.

[33] Intel. x86: intel cache allocation technology support. http://lwn.net/Articles/622893/. Accessed: 2025-05-23.

[34] Intel. Improving real-time performance by utilizing cache allocation technology, Apr. 2015. White Paper.

[35] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual: Volume 3B: System Programming Guide and Volume 4: Model-Specific Registers*, 2025.

[36] H. Kim and R. R. Rajkumar. Real-time cache management for multi-core virtualization. In *EMSOFT*, 2016.

[37] K. Kim, J. L. Diaz, L. L. Bello, J. M. López, C.-G. Lee, and S. L. Min. An exact stochastic analysis of priority-driven periodic real-time systems and its approximations. *IEEE Transactions on Computers*, 54(11):1460–1466, 2005.

[38] W. Kim, D. Shin, H.-S. Yun, J. Kim, and S. L. Min. Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In *RTAS*, 2002.

[39] E. Kohlberg and J. W. Pratt. The contraction mapping approach to the perron-frobenius theory: Why Hilbert's metric? *Mathematics of Operations Research*, 7(2):198–210, 1982.

[40] Z. Lai, K. T. Lam, C.-L. Wang, and J. Su. Powerock: Power modeling and flexible dynamic power management for many-core architectures. *IEEE Systems Journal*, 2017.

[41] C. Léonard. A survey of the Schrödinger problem and some of its connections with optimal transport. *Discrete and Continuous Dynamical Systems-Series A*, 34(4):1533–1574, 2014.

[42] Y. Lipman, R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le. Flow matching for generative modeling. In *ICLR*, 2023.

[43] G.-H. Liu, T. Chen, O. So, and E. Theodorou. Deep generalized Schrödinger bridge. *Advances in Neural Information Processing Systems*, 35:9374–9388, 2022.

[44] M. Liu, M. Behnam, and T. Nolte. An evt-based worst-case response time analysis of complex real-time systems. In *SIES*, 2013.

[45] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. A statistical response-time analysis of real-time embedded systems. In *RTSS*, 2012.

[46] Y. Lu, T. Nolte, J. Kraft, and C. Norstrom. A statistical approach to response-time analysis of complex embedded real-time systems. In *RTAS*, 2010.

[47] Y. Lu, T. Nolte, J. Kraft, and C. Norstrom. Statistical-based response-time analysis of systems with execution dependencies between tasks. In *ICECCS*, 2010.

[48] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni. Real-time cache management framework for multi-core architectures. In *RTAS*, 2013.

[49] S. D. Marino and A. Gerolin. An optimal transport approach for the Schrödinger bridge problem and convergence of sinkhorn algorithm. *Journal of Scientific Computing*, 85(2):27, 2020.

[50] F. Markovic, G. v. d. Bruggen, M. Gunzel, J.-J. Chen, and B. B. Brandenburg. A distribution-agnostic and correlation-aware analysis of periodic tasks. In *RTSS*, 2024.

[51] M. Noble, V. De Bortoli, A. Doucet, and A. Durmus. Tree-based diffusion Schrödinger bridge with applications to Wasserstein barycenters. *Advances in Neural Information Processing Systems*, 36:55193–55236, 2023.

[52] I. Nodozi, C. Yan, M. Khare, A. Halder, and A. Mesbah. Neural Schrödinger bridge with Sinkhorn losses: Application to data-driven minimum effort control of colloidal self-assembly. *IEEE Transactions on Control Systems Technology*, 32(3):960–973, 2023.

[53] J. Park, S. Park, and W. Baek. CoPart: Coordinated partitioning of last-level cache and memory bandwidth for fairness-aware workload consolidation on commodity servers. In *EuroSys*, 2019.

[54] B. Pass. Multi-marginal optimal transport: theory and applications. *ESAIM: Mathematical Modelling and Numerical Analysis*, 49(6):1771–1790, 2015.

[55] T. Patel and D. Tiwari. Clite: Efficient and qos-aware co-location of multiple latency-critical jobs for warehouse scale computers. In *HPCA*, 2020.

[56] L. Pons, J. Sahuquillo, V. Selfa, S. Petit, and J. Pons. Phase-aware cache partitioning to target both turnaround time and system performance. *IEEE Transactions on Parallel and Distributed Systems*, 31(11):2556–2568, 2020.

[57] R. B. Roy, T. Patel, and D. Tiwari. Satori: Efficient and fair resource partitioning by sacrificing short-term benefits for long-term gains. In *ISCA*, 2021.

[58] S. Salvador and P. Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent data analysis*, 11(5):561–580, 2007.

[59] I. N. Sanov. *On the probability of large deviations of random variables*. United States Air Force, Office of Scientific Research, 1958.

[60] E. Schrödinger. *Über die umkehrung der naturgesetze*. Verlag der Akademie der Wissenschaften in Kommission bei Walter De Gruyter u. Company., 1931.

[61] E. Schrödinger. Sur la théorie relativiste de l'électron et l'interprétation de la mécanique quantique. In *Annales de l'institut Henri Poincaré*, volume 2, pages 269–310, 1932.

[62] Y. Shi, V. De Bortoli, G. Deligiannidis, and A. Doucet. Conditional simulation using diffusion Schrödinger bridges. In *UAI*, 2022.

[63] R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2):876–879, 1964.

[64] R. Sinkhorn and P. Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

[65] P. Sohal, R. Tabish, U. Drepper, and R. Mancuso. Profile-driven memory bandwidth management for accelerators and CPUs in QoS-enabled platforms. *Real-Time Systems*, 58(3):235–274, 2022.

[66] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, 2015.

[67] Y. Song and S. Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[68] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[69] Splash2x benchmark. http://parsec.cs.princeton.edu/parsec3-doc.htm#splash2x. Accessed: 2025-05-23.

[70] D. Williams. *Diffusions, Markov processes, and martingales: Itô calculus*, volume 2. Cambridge University Press, 2000.

[71] R. J. Wysocki. *CPU Performance Scaling*. Linux Kernel Documentation, 2017. Copyright © 2017 Intel Corporation.

[72] W. Xie, R. Zhou, H. Wang, T. Shen, and E. Chen. Bridging user dynamics: Transforming sequential recommendations with Schrödinger bridge and diffusion models. In *CIKM*, 2024.

[73] M. Xu, R. Gifford, and L. T. X. Phan. Holistic multi-resource allocation for multicore real-time virtualization. In *DAC*, 2019.

[74] M. Xu, L. T. X. Phan, H. Choi, Y. Lin, H. Li, C. Lu, and I. Lee. Holistic resource allocation for multicore real-time systems. In *RTAS*, 2019.

[75] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memory bandwidth management for efficient performance isolation in multi-core platforms. *IEEE Transactions on Computers*, 65(2):562–576, Feb 2016.