2025 IEEE International Symposium on High-Performance Computer Architecture (HPCA)

# HILP: Accounting for Workload-Level Parallelism in System-on-Chip Design Space Exploration

Joseph Rogers,† Lieven Eeckhout,‡ and Magnus Jahre†
Norwegian University of Science and Technology (NTNU)†, Ghent University‡
joseph.c.p.rogers@ntnu.no, lieven.eeckhout@ugent.be, magnus.jahre@ntnu.no

***Abstract*—High-performance System-on-Chip (SoC) architectures are becoming increasingly complex and heterogeneous, and the days when a single application could utilize all of an SoC's hardware resources are all but over. The SoC's *workload*, i.e., the set of independent applications that the SoC typically executes, therefore has a significant impact on its efficiency. Accounting for Workload-Level Parallelism (WLP) in early-stage design space exploration is thus critical as later-stage analysis steps must focus on favorable design points to yield optimal results. Unfortunately, state-of-the-art MultiAmdahl and Gables fall short because they only model the extremes of minimal and maximal WLP.**

**We hence propose HILP, the first early-stage design space exploration approach for heterogeneous SoCs that fully accounts for WLP. Our key observation is that scheduling a workload of independent multi-phase applications on a heterogeneous SoC is an instance of the classic job-shop scheduling optimization problem and thus can be solved using integer linear programming. HILP therefore uses a high-performance integer linear programming solver to find a near-optimal schedule that minimizes the overall execution time of the workload, i.e., it schedules the dependent phases of all applications in the workload on the cores and accelerators of the target SoC to maximize performance while respecting power consumption and memory bandwidth constraints. We validate HILP by demonstrating that it captures the performance effects of Amdahl's law, the memory wall, and dark silicon, and then use it to explore the impact of WLP across a large SoC design space, yielding multiple insights. The key takeaway is that modeling WLP is necessary to ensure that more detailed, later-stage design tasks focus on the most favorable parts of the vast design space of heterogeneous SoCs.**

## I. Introduction

Modern high-performance System-on-Chip (SoC) architectures are becoming increasingly complex and heterogeneous because they must respect a rich set of (mutually conflicting) requirements and constraints. For example, emerging high-performance SoCs include an increasing number of DSAs, typically because DSAs can improve power efficiency by orders of magnitude compared to general-purpose CPUs [15]. DSAs however also increase chip area which in turn is a key driver of the SoC's manufacturing cost and embodied carbon footprint [7]. Despite the costs, leading mobile SoCs combine many tens of DSAs with conventional CPU cores and Graphics Processing Units (GPUs) [52], and SoCs for laptops and desktops, e.g., Apple M3 [5] and AMD Ryzen 8000 G-Series processors [2], devote significant resources to the GPU and DSAs (e.g., neural engines). Server SoCs are following suit, e.g., AMD MI300A [1] and Nvidia GH200 [49].

Designing high-performance SoCs is hence becoming an increasingly challenging task. Application complexity is also increasing, and it is becoming increasingly common that applications use different accelerators, i.e., a general-purpose GPU or specialized DSAs, in their various compute phases [29]. The applications' setup and teardown phases however remain well-suited to conventional CPUs. To complicate matters further, it is exceedingly rare that a single application can fully utilize all of the SoC's hardware resources; this trend is already motivating multi-tasking in homogeneous GPUs [71]. On the contrary, the common case is that SoCs target a mix of independent multi-phase applications, which we call its *workload*, and it is therefore wasteful to optimize SoCs for the increasingly uncommon single-application scenario. The overall efficiency of a heterogeneous SoC is thus intimately tied to its ability to exploit the parallelism available within the workload, i.e., *Workload-Level Parallelism (WLP)*[1], which we define as *the number of independent application phases that are concurrently executing on the SoC.*

Early-stage design decisions are especially important for heterogeneous SoCs because of the sheer size and complexity of their design space. For example, architects can in principle add DSAs for any combination of any phase of any application in the SoC's target workload. The computer architects that design these SoCs thus need WLP-aware design exploration approaches that enable them to understand and prune the design space in the very early stages of the design cycle — thereby ensuring that they focus later-stage, oftentimes simulation-heavy, design tasks on the most favorable parts of the SoC design space. State-of-the-art MultiAmdahl (MA) [72] and Gables [28] unfortunately cannot address this need because they only account for the extreme cases of minimal and maximal WLP. More specifically, MA assumes that application phases are executed in a fixed sequential order whereas Gables models both a fixed sequential order and fully parallel execution. MA thus pessimistically assumes minimal WLP, i.e., at most a single application phase can execute at any time, whereas Gables' parallel mode does not account for phase dependencies or sequential sections, i.e., it optimistically assumes that the workload is embarrassingly parallel.

Accounting for WLP during early-stage analysis is challeng-

[1]WLP is a special case of task-level parallelism [25] as it specifically focuses on parallelism *across* the independent applications (tasks) of a workload. It is complementary to Thread-Level Parallelism (TLP) and Accelerator-Level Parallelism (ALP) [29] which focus on a single application. The WLP definition is inspired by Memory-Level Parallelism (MLP) [14] but focuses on phases of independent applications rather than memory requests.

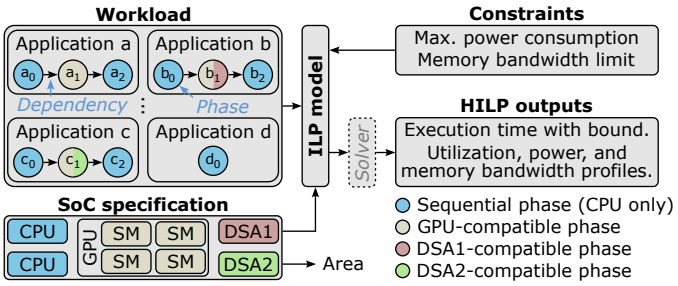



Fig. 1: High-level overview of HILP. *HILP fully accounts for WLP during early-stage SoC design space exploration.*

ing because there are many ways to schedule a workload on a heterogeneous SoC. While it can be trivial to find an optimal schedule when the number of applications and compute units is small, naive approaches quickly become intractable for interesting design points (e.g., the many tens of DSAs in mobile SoCs [52]). Establishing the quality of the schedule is however critical to fairly compare SoCs because, otherwise, we cannot know if the higher performance of $SoC_1$ compared to $SoC_2$ is (i) due to $SoC_1$ being a better architecture, or (ii) because the workload was scheduled more efficiently on $SoC_1$ than on $SoC_2$. In this paper, we consider schedules to be of similar quality if they provably yield a workload execution time within 10% of optimal, and we refer to such schedules as *near-optimal*. An additional benefit of focusing on near-optimal schedules is that it decouples the design of SoC hardware from the (challenging) task of writing efficient system software for it. Honing in on an architecture that performs favorably on near-optimal schedules in early design stages ensures that the system as a whole will become near-optimal once system software matures sufficiently to schedule the workload near-optimally at runtime.

We observe for the first time that scheduling a workload of independent multi-phase applications on a heterogeneous SoC is an instance of the Job-Shop Scheduling Problem (JSSP) [39] and thus can be solved using *Integer Linear Programming (ILP)*. JSSP is a classic constrained optimization problem that aims to schedule a collection of independent multi-task jobs onto a set of machines to minimize overall execution time. Formulating the heterogeneous SoC evaluation problem as an instance of JSSP enables accounting for WLP as we can leverage high-performance ILP solvers to find near-optimal schedules and thereby ensure that we are fairly comparing SoC architectures. Moreover, it is (relatively) straightforward to model multiple interacting constraints, such as both power consumption and memory bandwidth, because they can be expressed as linear functions of existing decision variables.

We leverage this observation to design HILP[2], the first fully WLP-aware early-stage exploration approach for heterogeneous SoCs. HILP models (complex) software that consists of multiple (dependent) phases, whereas state-of-the-art MA and Gables simply assume fully sequential or fully parallel execution. Moreover, HILP is, unlike MA and Gables, easily extensible (see Section VII). Figure 1 explains how HILP achieves this, and its inputs are (i) a workload of independent (multi-phase) applications, (ii) an SoC specification, and (iii) power consumption and memory bandwidth constraints. The SoC can contain CPUs, a variable-size GPU, and DSAs, and the objective is to schedule the workload on the SoC to minimize overall execution time while respecting all constraints. Each application, e.g., $a$, consists of multiple dependent phases, typically setup, compute, and teardown, e.g., $a_0$, $a_1$, and $a_2$, respectively. While general-purpose CPU cores can execute any phase, compute phases can typically also execute on the GPU or possibly a specialized DSA, e.g., phase $a_1$ is only compatible with the GPU but phase $b_1$ ($c_1$) can execute on the GPU and DSA1 (DSA2).

HILP processes these inputs to create the matrices required by our JSSP formulation and then invokes an ILP solver. The solver returns the best-found workload schedule and its overall execution time. HILP will not always find a provably optimal schedule because JSSP is NP-complete with more than two machines [20]. In this case, we leverage that the ILP solver also returns an optimality bound, i.e., the best possible execution time that can exist within the part of the solution space that the solver has not proved to be infeasible. Our specific definition of near-optimal is thus that the overall execution time reported by HILP is within 10% of optimal.

We validate HILP by demonstrating that it successfully captures the effects of Amdahl's law [4], the memory wall [67], and dark silicon [17] before quantitatively comparing HILP's performance predictions to MA [72] and Gables [28]. MA's fixed sequential order yields overly pessimistic predictions because it assumes no WLP, whereas Gables' maximal WLP assumption results in overly optimistic predictions. In contrast, HILP fully accounts for WLP and therefore provides performance predictions between these two extremes that are appropriate for the target workload. The quantitative impact of fully accounting for WLP can be significant. For example, when considering SoCs that are Pareto-optimal in terms of area and performance across a design space of 372 SoC architectures, MA's highest-performing SoC achieves 40% of the performance of HILP's highest-performing SoC at 114% of the area, thus substantially underestimating performance. In contrast, Gables massively overestimates performance, i.e., its highest-performing SoC predicts 136% higher performance than HILP at only 45% of the area.

While the quantitative differences between MA, Gables, and HILP are massive, a perhaps greater concern is that MA's and Gables' simplistic treatment of WLP leads them to recommend suboptimal design points. More specifically, MA prefers SoCs with a single large GPU — as this is the most area-efficient approach to improve performance under sequential execution — whereas Gables is biased towards SoCs with many small accelerators — because its fully parallel mode neglects dependencies. In contrast, HILP accounts for both parallelism and dependencies and recommends SoCs that cater to the specifics of the workload. For our *Default* workload constructed from ten multi-phase Rodinia [10] benchmarks, HILP recommends

[2]HILP is available from https://github.com/EECS-NTNU/hilp.

an SoC with 4 CPU cores, a moderate 16-SM GPU, and DSAs for the compute phases of the *HS* and *LUD* benchmarks. This workload needs 4 CPU cores to utilize the accelerators effectively (Amdahl's law), and providing DSAs for *HS* and *LUD* means that the other applications in the workload can share the GPU without it becoming a performance bottleneck, thus maximizing performance and minimizing area overhead. In summary, we make the following key contributions:

- We observe for the first time that scheduling a workload of independent multi-phase applications on a heterogeneous SoC is an instance of the JSSP problem.
- We leverage this insight to design HILP, the first early-stage design space exploration approach for heterogeneous SoCs that fully accounts for WLP.
- We demonstrate that fully accounting for WLP in early-stage analysis is critical to identify favorable regions in the vast design space of heterogeneous SoCs, in contrast to MA and Gables which recommend suboptimal SoCs.
- We use HILP to explore a large SoC design space and make several observations, e.g., the primary function of DSAs in the top-performing SoCs is to offload the GPU.

## II. Explaining How HILP Works

**Applications and workloads.** To explain how HILP works, we focus on a simple workload that consists of two applications, $m$ and $n$. Both applications have a compute phase that multiplies matrices, see ❶ in Figure 2. Application $m$ is a classic matrix multiplication kernel from the high-performance computing domain, and application $n$ does neural network inference. Both applications consist of three phases, setup, compute, and teardown, which must be executed in order. The setup phase makes the necessary preparations such as parsing the command line, reading input files, and allocating memory. These activities are well-suited to being executed on the CPU but poorly suited to acceleration. To model such behavior, HILP requires the architect to supply a compatibility vector for each application phase. The compatibility vector of the setup phase records that it can execute on the CPU but not on the GPU or DSA, see ❷. The compute phase is however well suited to acceleration and can thus be executed on the CPU, the GPU, or the matrix-multiply-focused DSA, see ❸.

HILP also associates an execution time vector with each application phase. We label the phases with integer subscripts and phase $i$ must complete before phase $i+1$ can execute. In this example, setup is phase 0, compute is phase 1, and teardown is phase 2. Application $m$ deals with larger matrices than application $n$ and the execution time of its compute phase is thus longer, i.e., the execution time of $m_1$ ($n_1$) is 8 (5), 6 (3), and 5 (2) seconds on the CPU, GPU, and DSA, respectively. To keep the example simple, we assume that the execution times of the setup and teardown phases, i.e., $m_0$, $m_2$, $n_0$, and $n_2$, are the same for both applications. Applications $m$ and $n$ hence constitute the workload ❹, and the objective is to schedule the phases of $m$ and $n$ on the SoC ❺ to minimize execution time while respecting dependencies and constraints.

**SoC specification.** We focus on a simple SoC with a single CPU, a single GPU, and a single DSA in this example. The compute cores connect to a shared global memory through a Network-on-Chip (NoC), and, for the purpose of this example, we assume that the architecture has sufficient bandwidth. (HILP also models limited memory bandwidth, see Section III-C.) HILP also records the power consumption of the CPU, the GPU, and the DSA, and Figure 2 thus shows a table that reports active and idle power consumption. We make the simplifying assumption that the SoC implements an aggressive power-gating strategy, and the power consumption of the CPU, GPU, and DSA is hence zero when idle.

**Workload scheduling.** The performance impact of workload scheduling can be significant. For example, naively scheduling all phases of the two-application example workload in Figure 2 on the CPU yields an execution time of 17 seconds. HILP finds the optimal schedule which executes $n_1$ on the GPU and $m_1$ on the DSA because (i) the execution time of $m_1$ is shorter on the DSA than on the GPU, and (ii) the execution time of $m_1$ hides the execution time of $n_1$ when $m_1$ executes on the DSA and $n_1$ on the GPU, see ❻. The optimal schedule hence yields a speedup of 2.4× relative to the naive schedule.

To enable analyzing the scheduling problem with ILP, HILP discretizes time into *time steps*; this is a common strategy when using ILP to solve JSSP (see e.g., [36], [68]). Selecting the size of the time step is a trade-off between resolution — because the execution times of all phases must be expressed as an integer number of time steps — and the time overhead of solving the model — as the solution space grows with the number of time steps. In general, we must select a time step size that is short enough to yield architectural insights, yet large enough for the model to solve within a reasonable time. (HILP uses an adaptive approach to achieve this, see Section III-D.) Setting the time step size is fortunately trivial in our example since (i) a time step size of one second captures the execution time of all phases exactly, and (ii) the model is small enough for HILP to find the optimal schedule practically instantly.

**Workload-Level Parallelism (WLP).** Recall that we defined WLP as the number of independent application phases that are executing concurrently on the SoC, and computing WLP thus simply amounts to counting the application phases that co-execute in a given time step. We can further compute *average WLP* by taking the arithmetic mean of the per-time-step WLP values across all the time steps in which at least one application phase was active.

MA [72] assumes a fixed sequential order which implies no WLP, i.e., at most a single application phase is executing on the SoC in all time steps, see ❼ in Figure 2. The WLP value for MA is hence always 1. Gables [28] augments MA's sequential order with a fully parallel execution mode, see ❽. Fully parallel execution is overly optimistic because it does not respect phase dependencies nor model sequential sections. Gables thus assumes the maximally achievable WLP. In Figure 2, the maximum average WLP is 2.4 because 3 application

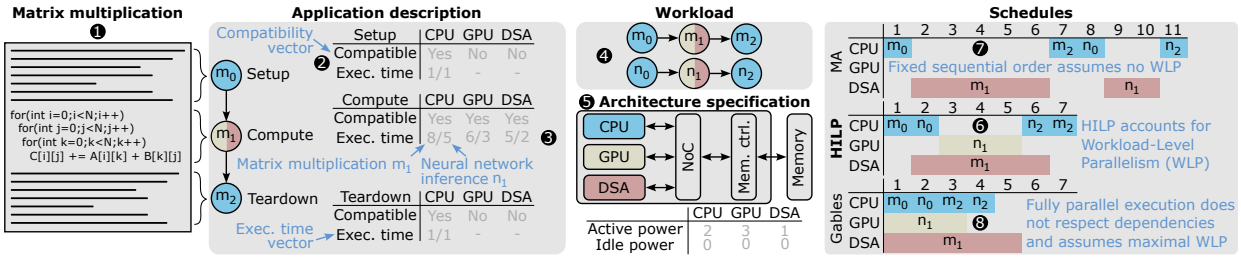
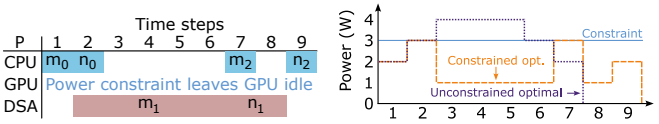



Fig. 2: Example illustrating how HILP models a two-application workload on a heterogeneous SoC with a CPU, a GPU, and a DSA without constraints compared to MultiAmdahl (MA) and Gables. *HILP fully accounts for WLP and thereby covers the design space between the extremes of minimal and maximal WLP which are occupied by MA and Gables, respectively.*




(a) Schedule. (b) Power profile.

Fig. 3: An optimal schedule under a 3 W power constraint. *Both $m_1$ and $n_1$ executes on the DSA, leaving the GPU idle.*

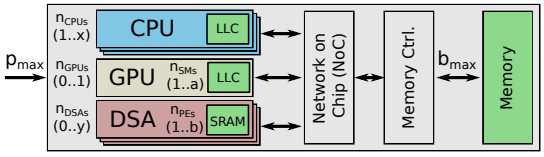



Fig. 4: HILP SoC architecture template. *HILP covers a vast design space of heterogeneous SoC architectures.*

phases are active in time steps 1, 2, and 3, phases $m_1$ and $n_2$ are active in time step 4, and only $m_1$ is active in time step 5. Unlike MA and Gables, HILP respects the dependencies of the workload and finds the optimal schedule ❻. In contrast, MA's sequential order adds unnecessary dependencies whereas Gables' fully parallel mode discards necessary dependencies. The average WLP of HILP's optimal schedule (1.7) is hence between the average WLP of MA (1) and Gables (2.4).

**Power constraint example.** Dark silicon [17] has led to many architectures containing more compute resources than they can use concurrently without exceeding the system's power budget. HILP thus models power constraints. Figure 3 illustrates the impact of limiting the power consumption of the architecture in Figure 2 to maximally 3 W; the active and idle power consumption of each core type is part of the architectural specification ❺. Since the power consumption of the GPU is 3 W, it means that it can only be used when the CPU and the DSA are idle, and it is no longer possible to hide the execution time of $n_1$ by executing it on the GPU while $m_1$ executes on the DSA. The power-constrained optimal schedule will thus allocate both compute phases to the DSA (see Figure 3a). Figure 3b demonstrates that the unconstrained optimal schedule exceeds the 3 W power constraint in time steps 3, 4, and 5 because both the DSA and the GPU are active. In contrast, the power-constrained schedule consumes at most 3 W, and, as expected, achieves lower performance than the unconstrained schedule.

## III. The Details of HILP

We now focus on describing how HILP formulates the problem of scheduling a workload of independent applications onto a heterogeneous SoC using Integer Linear Programming (ILP). Figure 4 illustrates the SoC architecture template we focus on in this work which covers SoCs with one or more CPUs, an optional GPU with a configurable number of SMs, and a range of DSAs with a configurable number of Processing Elements (PEs). To achieve this high degree of configurability while limiting the number of decision variables and thus the size of the solution space, HILP represents groups of identical Compute Units (CUs) as *core clusters*. In general, HILP models $n_c$ core clusters consisting of $n_u$ Compute Units (CUs); the CUs of a CPU cluster are CPU cores, the CUs of GPUs are SMs, and the CUs of DSAs are Processing Elements (PEs). Since CPUs must handle both sequential and parallel application phases, we model each CPU core as an independent core cluster (i.e., $n_u$ always equals 1 for CPU core clusters). We model local memory within each CU as well as a shared memory in each core cluster; this can be a cache or SRAM. The core clusters connect to the memory controllers and ultimately main memory through a Network-on-Chip (NoC) which we assume is dimensioned to deliver at least the maximum memory bandwidth $b_{\max}$. The SoC must operate under a power budget $p_{\max}$.

### A. Notation, Objective Function, and JSSP

**Notation.** The objective of HILP is to schedule a set of independent applications $\mathcal{A}$ (i.e., the workload) on a set of core clusters $\mathcal{C}$ such that execution time is minimized. Each application $a$ consists of one or more phases, and we use an integer subscript to denote the phase; $\mathcal{P}_a$ is the set of phases in application $a$. Unless stated otherwise, we number the application phases in ascending order, i.e., $a_p$ represents phase $p$ of application $a$ and must be executed before phase $a_{p+1}$. A key input to HILP is the execution time matrix $T_{cap}$ which reports the execution time of application phase $a_p$ on core cluster $c$ in integer time steps.

**Objective function.** Table I summarizes the key variables of HILP, and we will explain all of them in this section. We start with the decision variables as these are the unknowns the ILP

TABLE I: HILP Parameters.

| Type | Sym. | Description |
|---|---|---|
| Decision variable | $S_{ap}$ | The time step in which phase $p$ of application $a$ starts executing. |
| Decision variable | $C_{ap}$ | Holds the identifier of the core cluster that phase $p$ of application $a$ is allocated to. |
| Input | $T_{cap}$ | The execution time of phase $p$ of application $a$ when it is executed on core cluster $c$. |
| Input | $B_{cap}$ | The memory bandwidth required when executing application phase $a_p$ on core cluster $c$. |
| Input | $P_{cap}$ | The power consumption of phase $a_p$ when executing on core cluster $c$. |
| Config. input | $E_{cap}$ | A binary matrix which is 1 when application phase $a_p$ can be executed on core cluster $c$ and 0 otherwise. |
| Config. input | $U_{cap}$ | The maximum number of active core clusters when phase $a_p$ is executing on core cluster $c$. |

solver will assign values to. HILP's decision variables are $S_{ap}$ which is set to the time step in which application phase $a_p$ starts execution, and $C_{ap}$ which captures the integer identifier of the core cluster that application phase $a_p$ is allocated to. The objective of HILP is thus to assign values to the matrices $S_{ap}$ and $C_{ap}$ to minimize the total execution time $t$:

$$t = \max(S_{ap} + T_{cap}) \quad c \in \mathcal{C}, a \in \mathcal{A}, \forall p, t > 0 \\ \text{subject to Equations 2, 3, 4, 6, 7, and 8.} \tag{1}$$

Equation 1 states that overall execution time $t$ is the completion time of the last-completing application phase across all core clusters. The start time of all application phases is given by the decision variable $S_{ap}$, and the core cluster variable $C_{ap}$ is used together with the phase identifier $a_p$ to retrieve the execution time of $a_p$ on core cluster $c$ from the execution time matrix $T_{cap}$. When HILP completes, the best-performing schedule can be extracted by retrieving start times from $S_{ap}$ and core cluster allocations from $C_{ap}$. We can then plot the schedule, as previously shown in Figures 2 and 3, by combining these values with the execution time matrix $T_{cap}$.

**The Job-Shop Scheduling Problem (JSSP).** HILP is derived from the classic combinatorial optimization problem JSSP [39] which is typically formulated as finding the schedule which yields the lowest makespan for the execution of a set of jobs on a set of machines. The jobs in JSSP are partitioned into (dependent) tasks which may require different machines. To make the paper more accessible to computer architects, we describe JSSP using established architectural terms, i.e., jobs are applications, tasks are phases, and the makespan is overall execution time. The objective function in Equation 1, the constraints in Equations 2, 3, and 4, and the helper function in Equation 5 are commonly used in ILP formulations of JSSP, but the remaining constraints and architectural modeling are new contributions of this work.

### B. Modeling Applications

**Application inputs.** HILP takes the execution time matrix $T_{cap}$ and the memory bandwidth matrix $B_{cap}$ as input. $T_{cap}$ ($B_{cap}$) reports the execution time (memory bandwidth consumption) of application phase $a_p$ when executed on core cluster $c$. The matrices are populated by profiling scaled versions of the parallel Rodinia [10] benchmarks on high-end CPUs and GPUs; we model DSAs at a fixed efficiency advantage over GPUs. Section IV will describe this process in detail, and we now turn our attention to describing the constraints HILP requires to output correct schedules.

**Ordering.** The Rodinia benchmarks all exhibit three key phases: setup, compute, and teardown. The phases are dependent, e.g., setup must complete before compute can start. HILP hence uses the ordering constraint to ensure that phase $a_p$ completes before phase $a_{p+1}$ starts:

$$S_{a,p} + T_{cap} \leq S_{a,p+1} \quad c \in \mathcal{C}, a \in \mathcal{A}, p \in \mathcal{P}_a. \tag{2}$$

As we will discuss in Section III-D, complicating the dependency constraint and phase modeling comes at the cost of increased model complexity and longer solve times. Since Equation 2 is sufficient to model the Rodinia benchmarks we focus on in this paper, we hence use it for our validation and exploration experiments (see Sections V and VI). HILP can however model applications with arbitrarily complex dependencies and concurrent phases by using a graph to represent phase dependencies, and we will explain how to do this when discussing the extensibility of HILP in Section VII.

**Non-interference.** We must also ensure that a core cluster only executes a single application phase at the time. This is addressed by the non-interference constraint which states that if two application phases are scheduled on the same core cluster, one will complete before the other starts:

$$C_{ap} = C_{bq} \implies S_{ap} + T_{cap} \leq S_{bq} \vee S_{bq} + T_{cbq} \leq S_{cap} \\ c \in \mathcal{C}, a \in \mathcal{A}, b \in \mathcal{A}, \forall p, \forall q. \tag{3}$$

**Compatibility.** Application phases can only execute on core clusters they are compatible with, e.g., sequential phases can only execute on a CPU. Another example is that compute phases can only execute on DSAs that support them. HILP models this through the binary compatibility matrix $E_{cap}$ in which a value of 1 (0) means that application phase $a_p$ can (cannot) execute on core cluster $c$:

$$E_{cap} = 1 \quad c \in \mathcal{C}, a \in \mathcal{A}, \forall p. \tag{4}$$

The compatibility constraint uses the decision variable $C_{ap}$ to access the compatibility matrix $E_{cap}$ and ensures that the value equals one. $E_{cap}$ can also be leveraged to perform what-if analysis. For example, it could be used to explore the impact of pinning a phase to a specific DSA compared to no restrictions. This would be encoded by setting $E_{cap}$ to 1 for the target DSA and the values for all other core clusters to 0.

### C. Modeling Architectures

**SoC specification.** HILP can model a vast range of heterogeneous SoCs by combining the template in Figure 4 with the compatibility matrix $E_{cap}$. The minimum configuration is a single CPU core, but, beyond that, the HILP user can create arbitrarily complex architectures by setting appropriate bits in $E_{cap}$. The generality of a core cluster is hence given by the number of bits that are set for it in $E_{cap}$.

**Power and bandwidth.** HILP's architectural constraints become easier to formulate if we first capture the time during which application phase $a_p$ executes on core cluster $c$. The reason is that it is during this time that $a_p$ consumes the resource in question. We achieve this by defining a helper set $\mathcal{T}$ that contains the start times and completion times of each application $a_p$ and a helper function $h$ which identifies the relevant time steps:

$$h(r,t) = \begin{cases} r & \text{if } t \geq S_{ap} \wedge t < S_{ap} + T_{cap} \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Here, $r$ is the resource consumption of application phase $a_p$ which should be exposed if it is executing on core cluster $c$ in time step $t$, i.e., $a_p$ has started but not yet completed in time step $t$. With $h$ and $\mathcal{T}$ in place, it is straightforward to formulate HILP's power constraint:

$$\sum_{a=0}^{|\mathcal{A}|} \sum_{p=0}^{|\mathcal{P}_a|} h(P_{cap}, t) \leq p_{\text{max}} \quad \forall t \in \mathcal{T} \quad (6)$$

The matrix $P_{cap}$ is an input to HILP and captures the power consumption of phase $a_p$ when running on core cluster $c$; $p_{\text{max}}$ is the power budget which is a user-supplied input. We populate $P_{cap}$ through measurements on high-end CPUs and GPUs (see Section IV). The bandwidth constraint is formulated similarly to the power constraint:

$$\sum_{a=0}^{|\mathcal{A}|} \sum_{p=0}^{|\mathcal{P}_a|} h(B_{cap}, t) \leq b_{\text{max}} \quad \forall t \in \mathcal{T} \quad (7)$$

In fact, the only difference between Equations 6 and 7 is that Equation 7 uses the bandwidth matrix $B_{cap}$ for the resource consumption and $b_{\text{max}}$ as the limit.

**Modeling CPUs.** The sequential setup and teardown phases of different applications can be executed in parallel if the SoC has a sufficient number of CPU cores. To support this behavior, we configure HILP with as many CPU core clusters as there are CPU cores in the system, i.e., one CPU core per CPU cluster. CPU cores can however also execute the parallel compute phase of a single application on multiple cores. Recall that we require that each application phase can only be allocated to a single cluster to keep the number of decision variables low. To support executing parallel application phases on the CPUs without adding decision variables, we use the input matrix $U_{cap}$ to record the number of core clusters that each application phase $a_p$ can use in the same time step when allocated to core cluster $c$. $U_{cap}$ is set to 1 for all sequential application phases on all CPU clusters and to the number of CPU cores required for all parallel phases on all CPU clusters. The following constraint then ensures that HILP never uses more than $u_{\text{max}}$ CPU cores in a single time step:

$$\sum_{a=0}^{|\mathcal{A}|} \sum_{p=0}^{|\mathcal{P}_a|} h(U_{cap}, t) \leq u_{\text{max}} \quad \forall t \in \mathcal{T} \quad (8)$$

**Dynamic Voltage and Frequency Scaling (DVFS).** When exploring architectural design spaces, it is easy to configure an SoC in which a core cluster cannot be activated because this would exceed the SoC's power budget. If this situation occurs in a real system, system software will apply DVFS or disable CUs to reduce power consumption sufficiently to respect the power budget. We account for this by specifying core clusters with different numbers of CUs, performance, and power consumption, and then setting $U_{cap}$ and $u_{\text{max}}$ such that Equation 8 ensures that each application phase activates at most one of these core clusters. In this way, the ILP solver will activate the core cluster and operating point, in terms of active CUs, performance, and power consumption, which achieves the lowest overall workload execution time.

Note that achieving this idealized operating point can be a tall order for the runtime DVFS mechanisms of real-world heterogeneous SoCs. Our motivation for adopting this idealized model is, as with phase scheduling, that it enables fairly comparing SoC architectures, i.e., we know that $\text{SoC}_1$ is better than $\text{SoC}_2$ because it is a better architecture and not because the DVFS mechanism happened to perform better. More detailed analysis is of course needed to determine if this ideal is achievable in practice, and the purpose of HILP is thus to ensure that the architect focuses their effort on the part of the design space with the highest potential.

### *D. Schedule Quality and Scalability*

**Time step resolution.** The time step resolution, i.e., the amount of time that each time step represents, must be set such that (i) the execution time differences between applications are represented, and (ii) the evaluation overhead remains acceptable. If the time step resolution is too coarse, we might miss architectural insights because the model is unable to represent differences in execution time. On the other hand, if the time resolution is too fine, the solution space may be too large for the ILP solver to find a near-optimal schedule in reasonable time. We also need to bound the number of time steps to consider, i.e., the time horizon, because the search space would otherwise be infinite. The time horizon however needs to be large enough for the solver to quickly find a feasible solution.

HILP addresses this challenge by adopting an adaptive approach. For our validation experiments in Section V, we initially run all experiments with a time step resolution of two seconds and a time horizon of 1,000 steps. This number of steps was chosen to accommodate the worst-case execution time of our *Rodinia* workload (see Section IV) and allows many configurations to reliably solve to within the optimality bound on server nodes equipped with 256 GB of memory in a matter of hours. If a workload completes in less than 200 time steps, we increase the time step resolution by 5× (e.g., from 2 seconds to 400 milliseconds) and run the solver again while keeping the time horizon constant at 1,000 time steps. If necessary, we repeat this process until the overall execution time of the workload exceeds 200 time steps. When exploring a large design space, as in Section VI, we adopt a coarser resolution to maintain a reasonable evaluation time. More specifically, we use a time horizon of 200 steps, an initial

TABLE II: Benchmarks. *(C is short for compute, TD is short for teardown, and bandwidth is measured in GB/s.)*

| Benchmark | Execution time (s) | | | | GPU | GPU power-law fit (a, b, $R^2$) | | Scaled benchmark |
|---|---|---|---|---|---|---|---|---|
| | Setup | C-CPU | C-GPU | TD | BW | GPU Time | GPU BW | configuration |
| Breadth-First Search (BFS) | 95.3 | 17.0 | 1.0 | 11.9 | 86.5 | 7.83, -0.77, 0.95 | 0.07, 0.92, 0.98 | 128M elements |
| Heartwall (HW) | 8.0e-4 | 78.3 | 1.2 | 0.2 | 7.3 | 3.77, -0.52, 0.92 | 0.84, 0.24, 0.30 | 104 frames |
| Hotspot3D (HS3D) | 0.7 | 49.2 | 0.1 | 51.2 | 36.4 | 10.33, -0.86, 1.00 | 0.14, 0.75, 1.00 | 512×512×8, 200 iterations |
| Hotspot (HS) | 80.8 | 395.9 | 20.5 | 71.3 | 40.4 | 13.93, -1.00, 1.00 | 0.07, 1.00, 1.00 | 16K×16K, 512 iterations |
| LavaMD (LMD) | 0.3 | 163.4 | 2.5 | 0.3 | 0.6 | 13.98, -0.99, 1.00 | 0.10, 0.90, 1.00 | 42 1D boxes |
| LU Decomposition (LUD) | 0.1 | 444.2 | 12.0 | 0.6 | 61.6 | 10.26, -0.88, 1.00 | 0.10, 0.87, 1.00 | matrix size 16K |
| Myocyte (MC) | 0.1 | 77.6 | 8.3e-2 | 0.6 | 0.1 | 1.01, 8.98e-06, 0.00 | 2.60, -0.28, 0.15 | 100K span, 12 w., 0 m. |
| Nearest Neighbor (NN) | 1.6e-3 | 159.4 | 3.8e-3 | 0.3 | 187.6 | 8.97, -0.82, 0.98 | 0.07, 0.95, 0.99 | 64K size, 2K neighbors |
| Pathfinder (PF) | 72.1 | 14.0 | 0.2 | 0.3 | 95.2 | 7.27, -0.76, 0.99 | 0.27, 0.58, 0.95 | 400K rows, 5K col., 1 pyr. |
| Stream Cluster (SC) | 1.0e-4 | 156.0 | 2.1 | 0.3 | 216.1 | 5.41, -0.62, 0.87 | 0.07, 0.88, 0.96 | 30–40 centers, 128K points |

TABLE III: GPU power scaling.

| Clock frequency (MHz) | Power consumption (W) | | Power-law fit (a, b, $R^2$) |
|---|---|---|---|
| | All SMs | Per-SM | |
| 210 | 77.2 | 0.6 | 0.10, 0.94, 1.00 |
| 240 | 83.5 | 0.7 | 0.53, 0.99, 1.00 |
| 300 | 97.1 | 0.8 | 0.06, 1.03, 1.00 |
| 360 | 105.1 | 0.8 | 0.07, 0.99, 1.00 |
| 420 | 119.9 | 0.9 | 0.06, 1.01, 1.00 |
| 480 | 129.5 | 1.0 | 0.07, 0.99, 1.00 |
| 540 | 139.8 | 1.1 | 0.07, 0.99, 1.00 |
| 600 | 153.6 | 1.2 | 0.07, 0.98, 1.00 |
| 660 | 164.0 | 1.3 | 0.07, 0.98, 1.00 |
| 705 | 172.9 | 1.4 | 0.07, 0.97, 1.00 |
| 765 | 185.4 | 1.4 | 0.07, 0.97, 1.00 |

time step resolution of 10 seconds, and increase resolution by 5× when workloads complete in under 40 time steps.

**Finding near-optimal schedules.** ILP solvers tighten the optimality bound by proving that sections of the solution space are infeasible. We run HILP with a time limit of four hours until it has converged to a favorable time step resolution. The overwhelming majority of our experiments achieve an optimality bound within 10% of overall execution time at this stage, and we rerun the experiments that do not achieve this bound with more resources. Depending on workload and SoC characteristics, we gave the solver more CPU cores, more memory, or a higher time limit, and we were ultimately able to get the solver to prove 10%-bounds for all configurations.

**Solver thread count.** The thread count of the ILP solver impacts its performance and memory consumption because each worker thread traverses a different part of the solution space. This requires duplicating parts of the model and greatly increases memory usage, but it does not necessarily reduce solve time proportionally. For example, one of our experiments required on the order of 90 GB of memory with four threads. Running the same experiment with 32 threads increases performance by 1.4× but increases memory requirements to 230 GB. We therefore generally run the ILP solver with four threads to balance solve time and memory overhead.

**Scalability.** ILP problems become more challenging to solve as the number of decision variables grows. HILP has two matrices of decision variables, $S_{ap}$ and $C_{ap}$, which both grow by the product of the number of applications and phases (i.e., $|\mathcal{A}| \times |\mathcal{P}_a|$). Another driver of model complexity is the number of constraints, and HILP's power consumption and bandwidth constraints for example both require one constraint for each time step (see Equations 6 and 7, respectively). As is typical in modeling, HILP thus also benefits from being as simple as possible, but no simpler, and this is the reason why we opt to model the application phase structure at a relatively high abstraction level in this work.

## IV. Experimental Setup

**Benchmarks.** We use version 3.1 of Rodinia [10], [70] for our evaluation because it provides CPU and GPU implementations of the same benchmarks. We profiled each benchmark on high-end hardware to measure phase execution time and bandwidth consumption (see Table II). For the CPUs, we profile the benchmarks with Linux `perf` [60] on an AMD EPYC 7543 processor for all possible core counts from 1 to 32. To capture performance and bandwidth trends on GPUs, we combined Nvidia Nsight Compute [48] with Nvidia Multi-Instance GPU (MIG) [50] to profile the benchmarks on an Nvidia A100 (40 GB) with SM counts of 14, 28, 42, 56, and 98; these are all the configurations MIG supports on this architecture. While MIG scales LLC capacity and NoC bandwidth proportionally to SM count, it scales memory bandwidth non-linearly, i.e., the configurations with 14, 28, 42, 56, and 98 SMs have 375, 375, 750, 750, and 1,500 GB/s of memory bandwidth, respectively.

As NSight cannot measure the energy consumption per benchmark, we use `gpu-burn` [61] to get worst-case power consumption for all MIG configurations and available GPU core clock frequencies with `nvidia-smi` [51] (see Table III). We observe that even while idle, the A100 consumes approximately 30 W. We thus assume that our power measurements under load have a 30 W static component, the rest being dynamic; we scale static power linearly with the number of SMs in the SoC. Our AMD 32-core CPU also does not enable measuring per-benchmark energy consumption, and we thus estimate core power consumption from its 225 W Thermal Design Power (TDP), yielding 7.0 W per core.

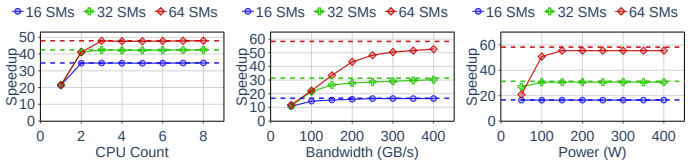
To utilize the modern hardware we profile on, we select the ten Rodinia benchmarks with scalable input parameters and scale their input set such that executing the benchmark to completion on a single CPU core takes more than one minute (see Table II). The relatively long setup and teardown times of some benchmarks (e.g., *BFS*, *HS*), are primarily due to the generation of input data and writing results to disk. We use setup and teardown timing from the CPU to model systems with unified memory, i.e., there are no (blocking) memory transfers to or from dedicated accelerator memories.

**HILP inputs.** Recall that HILP takes execution time, memory bandwidth, and power consumption matrices as input ($T_{cap}$, $B_{cap}$, and $P_{cap}$, respectively). While it is straightforward to create these matrices from our CPU profiles because we sweep all possible core counts, our GPU profiles have gaps that we need to fill. To this end, we fit a power-law function (i.e., $y = a \times x^b$) to our data points using least-squares linear regression; $x$ is the number of SMs and $y$ is performance, bandwidth, or power normalized to the GPU with 14 SMs. Table II reports the power-law parameters for execution time and bandwidth at the baseline clock frequency (765 MHz) and each curve's coefficient of determination (i.e., $R^2$); the power-law curves for all other operating points yield similar $R^2$ values. Table III reports power consumption as a function of GPU clock frequency and the power-law curves for each operating point. $R^2$ is a number between 0 and 1, with 1 indicating that the curve fits the data points perfectly. Tables II and III mostly report $R^2$ values close to, but not exactly, 1. The exceptions are *MC* (both performance and bandwidth) and *HW* (bandwidth). These benchmarks are insensitive to the number of SMs, and we are therefore fitting to random variation in the performance profiles. The curves of *MC* and *HW* are however flat across the SM-counts we consider, and they thus capture the underlying trend for these benchmarks even if $R^2$ is low.

We model DSAs at a 4× efficiency advantage compared to the GPU. The DSAs hence use the same performance and bandwidth curves as the GPU but only a quarter of the power and area. This is in line with prior work, e.g., Dally et al. [15] reports a 2.1× DSA efficiency advantage over a GPU within deep learning. (We will explore this further in Section VI.)

**Workloads.** Our *Rodinia* workload contains a single copy of each benchmark and uses the time, bandwidth, and power curves in Tables II and III directly. Several of the Rodinia benchmarks are however setup and teardown-heavy (e.g., *BFS*, *HS3D*, *HS*, and *PF*), and it is not unreasonable to assume that these phases would have been optimized to varying degrees in efficiency-sensitive deployments. We therefore define the *Default* and *Optimized* workloads in which we reduce the setup and teardown times of all benchmarks by 5× and 20×, respectively. These three workloads are thus sensitive to Amdahl's law to varying degrees, i.e., *Rodinia* is most sensitive and *Optimized* is least sensitive.

**SoC configurations.** We describe the SoC design space in terms of the ranges of CPU cores, GPU SMs, DSAs, and PEs per DSA for each experiment. Unless otherwise specified, we model an 800 GB/s HBM3 memory that consumes 7 pJ/bit [65] and a 600 W power budget. We estimate CPU and GPU area using the same architecture family that we used for profiling, i.e., the AMD Zen 3 and Nvidia Ampere, and focus on devices manufactured in a 7 nm technology node. When including all compute chiplets and the I/O die, the 64-core AMD EPYC 7763 [59] has a total die area of 1,064 $mm^2$, yielding 16.6 $mm^2$ per core. We include the I/O die to account for the uncore area required to support each core. The Nvidia GA100 [32] has 128 SMs and an area of 826 $mm^2$, yielding 6.5 $mm^2$ per SM.




(a) Amdahl's law. (b) Memory wall. (c) Dark silicon.

Fig. 5: Performance trends across various SoC design spaces. *HILP demonstrates (a) Amdahl's law, (b) the memory wall, and (c) dark silicon.*

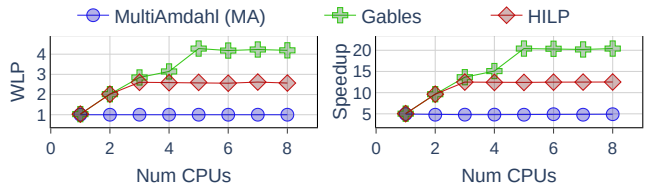
**Configuring the ILP solver.** We implement HILP's ILP model using MiniZinc [44]. The key benefits of MiniZinc are that (i) it enables us to express the model succinctly (i.e., in less than 100 lines of code), and (ii) its FlatZinc model format is compatible with a wide range of both open-source and commercial ILP solvers. It is hence possible to select different solvers without changing the model implementation. We experimented with Gecode [58], Gurobi [23], and OR-Tools [21]. OR-Tools yields the highest performance, and we therefore use it for all the experiments in this paper. Another benefit of OR-Tools is that it is open source which means that we can make all necessary components of HILP available to the community[3]. We used an in-house cluster running Rocky Linux 9.2 for our experiments, and our compute jobs ran on nodes with from 20 to 64 processor cores and 128 GB to 2 TB of memory.

## V. HILP Validation

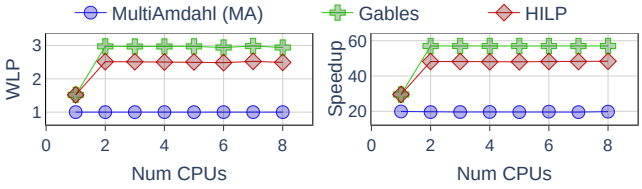
Before using HILP to explore the SoC design space in Section VI, we must gain confidence that it reports meaningful results. We do this by demonstrating that HILP can reproduce well-known architectural phenomena, i.e., Amdahl's law [4], the memory wall [67], and dark silicon [17]; and select a workload, SoC configurations, and constraints for each experiment such that the phenomenon we focus on dominates.

**Reproducing Amdahl's law.** Figure 5a shows how performance scales as we add CPU cores to SoCs with different GPU configurations (i.e., 16, 32, and 64 SMs) for the *Default* workload. The points are SoC configurations, and we report speedup relative to fully sequential execution on an SoC with a single CPU core. To fully focus on Amdahl's law, we do not constrain power and bandwidth in this experiment and only consider a GPU accelerator. The dotted lines in the figure show the maximum speedup that each SoC's GPU can achieve for this workload. Figure 5a shows that HILP captures Amdahl's law. The single-CPU SoCs are severely limited by sequential setup and teardown phases, and, in Amdahl's terms, the workload thus exhibits a significant sequential section. As CPU cores are added to the SoC, performance improves and then saturates at the compute limit of the GPU. The reason is that increasing the number of CPU cores enables executing

[3]As mentioned before in the introduction, HILP can be found at https://github.com/EECS-NTNU/hilp.




(a) Rodinia workload.




(b) Optimized workload.

Fig. 6: Average WLP and speedup for MA, HILP, and Gables. *MA is pessimistic and Gables is optimistic.*

the setup and teardown phases while a compute phase is executing on the GPU, which in Amdahl's terms equates to the workload's sequential section essentially disappearing.

**Reproducing the memory wall.** We now fix the number of CPU cores at 4 and sweep memory bandwidth constraints from 50 to 400 GB/s (see Figure 5b). We still consider SoCs with a 16-SM, 32-SM, or 64-SM GPU, but we move to the *Optimized* workload to reduce the impact of Amdahl's law. All SoCs are bandwidth bound at 50 GB/s, but the bandwidth requirements of the 16-SM SoC are modest, and it hence becomes compute-bound already with a memory bandwidth of 100 GB/s. The 32-SM configuration becomes compute-bound at 300 GB/s while the 64-SM SoC is still not completely compute-bound at 400 GB/s. We can hence conclude that HILP's memory bandwidth constraint correctly captures the performance impact of limited memory bandwidth.

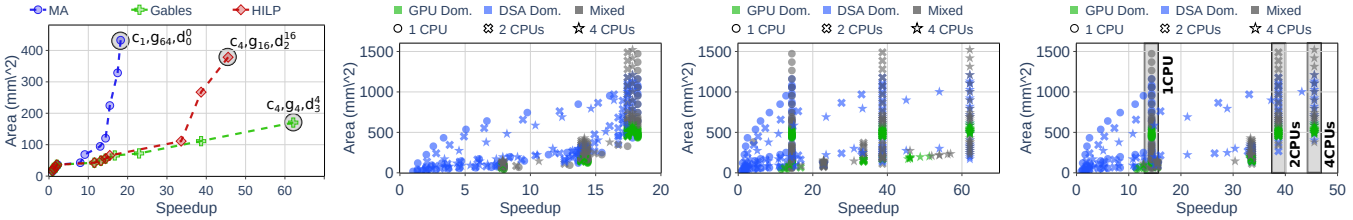
**Reproducing dark silicon.** We now replace the bandwidth constraint with a power constraint and sweep power budgets from 50 W to 400 W (see Figure 5c). 50 W is sufficient for the 16-SM SoC, and it hence reaches its performance potential regardless of the power budget. In contrast, the 32-SM (64-SM) SoC requires a power budget of 100 W (150 W) to reach its performance potential. Interestingly, HILP reports that the 32-SM SoC outperforms the 64-SM SoC under the 50 W power budget. The reason is that the 50 W power budget caps the maximum clock frequency of the 64-SM GPU at 300 MHz, whereas the 32-SM GPU can use the full frequency range. Some benchmarks, e.g., *HW*, are more sensitive to clock frequency than SM count. HILP hence makes sure to prioritize executing these benchmarks with high frequency on the 32-SM GPU, ultimately resulting in the SoC with the 32-SM GPU outperforming the SoC with the 64-SM GPU.

**Comparing MA and Gables to HILP.** Figures 6a and 6b report average WLP (left) and speedup (right) for MA [72], parallel-mode Gables [28], and HILP for an SoC with a 64-SM GPU when increasing the number of CPU cores from 1 to 8 for *Rodinia* and *Optimized*, respectively. MA's fixed sequential order always yields the minimal WLP value of 1 regardless of CPU count. MA also consistently reports pessimistic speedups of 4.9 and 19.8 for *Rodinia* and *Optimized*, respectively, because the GPU configuration does not change in this experiment. *Rodinia* is heavily CPU-bound with a single core, and Gables and HILP thus also report WLP close to 1 in this case (see Figure 6a). Gables optimistically assumes that WLP can rise to the maximal value of 4.3, while HILP's WLP saturates at 2.6 because HILP respects phase dependencies. Setup and teardown tasks are significantly less prominent in *Optimized* compared to *Rodinia*. Gables optimistically predicts that WLP will saturate at 3.0 for *Optimized*, yielding a correspondingly optimistic speedup of 57.0 (see Figure 6b). HILP reports that phase dependencies limit saturated WLP (speedup) to 2.5 (48.2). For both workloads, speedup is highly correlated with WLP, and WLP thus explains the performance trends.

## VI. Exploring the SoC Design Space with HILP

We now explore the design space covered by SoCs with 1, 2, or 4 CPU cores, an optional GPU with 4, 16, or 64 SMs, and 0 to 10 DSAs with 1, 4, or 16 PEs for the *Default* workload. Each application is optionally accelerated by its own DSA, yielding SoCs with up to 10 DSAs since *Default* contains all applications. As the number of ways of combining up to 10 DSAs is massive, we first order the benchmarks according to the CPU execution time of their compute phase (see Table II) and then allocate DSAs in descending order, effectively prioritizing DSAs for longer-running compute phases. The DSA in a 1-DSA SoC hence accelerates *LUD*, the DSAs in a 2-DSA SoC accelerates *LUD* and *HS*, and so on. This yields a total of 372 possible SoC configurations since we allocate the same number of PEs to all DSAs in a SoC. It took us about five days to obtain these results when using on average eighty compute nodes on NTNU's Idun cluster [56]. Notice that this study would be next to impossible to execute on conventional architectural simulators as HILP considers all possible phase schedules in a single run, while a conventional simulator would require separate runs for each possible phase schedule.

**Pareto-optimal SoC architectures.** Figure 7 shows speedup versus chip area for this design space as reported by MA, HILP, and Gables. As in Section V, we report speedup relative to fully sequential execution on an SoC with a single CPU core. Since Gables does not support constraining power, we provide MA and HILP with a power budget of 600 W to ensure a fair comparison. (We will explore the impact of constraining power later in this section.) Figure 7a reports the Pareto fronts of MA, Gables, and HILP, see Figures 7b, 7c, and 7d, respectively, for the complete design spaces. In these figures, each point represents an SoC architecture, and we color-code points based on their accelerator mix. A design point is green (blue) if it allocates more than 75% of the accelerator area to the GPU (DSAs). The point is grey if neither the area of the GPU nor the DSAs exceeds this limit. We label SoCs on the form $(c_i,g_j,d_k^l)$ where $i$ is the number of CPU cores, $j$ the number of GPU SMs, $k$ the number of DSAs, and the superscript $l$ denotes the number of PEs in each DSA.




(a) Pareto fronts. (b) MA design space. (c) Gables design space. (d) HILP design space.

Fig. 7: The SoC design space for the *Default* workload according to MA, Gables, and HILP. *MA is generally overly pessimistic because it assumes a fixed sequential phase order whereas Gables is overly optimistic because it discards phase dependencies. HILP avoids these pitfalls by fully accounting for WLP.*

The key takeaway from Figure 7a is that the Pareto fronts of MA, Gables, and HILP differ both quantitatively and qualitatively, leading to our first key insight:

**Key Insight 1**

*Fully accounting for WLP is critical: Simplistic WLP assumptions lead to recommending suboptimal SoCs.*

Figure 7a yet again demonstrates that MA's assumption of minimal WLP leads to overly pessimistic speedup predictions and that parallel-mode Gables' maximal WLP assumption is overly optimistic. For MA, the reason is that assuming fully sequential execution effectively creates phase dependencies that the workload does not have. Gables, in contrast, predicts unreasonably high speedups for high-performance SoCs because its fully parallel execution model assumes no dependencies. In contrast, HILP fully accounts for WLP and therefore covers the middle ground between these two extremes and recommends an SoC that is appropriate for the WLP characteristics of the target workload.

**WLP matters quantitatively.** The highest-performing Pareto optimal SoC configurations are $(c_1,g_{64},d_0^0)$, $(c_4,g_4,d_3^4)$, and $(c_4,g_{16},d_2^{16})$, according to MA, Gables, and HILP, respectively, and these SoCs demonstrate that the way the approaches deal with WLP can have a significant quantitative impact. MA is pessimistic and reports that the $(c_1,g_{64},d_0^0)$ SoC yields a speedup of 18.2 at an area of 432.6 mm$^2$. In contrast, Gables optimistically predicts that the $(c_4,g_4,d_3^4)$ SoC achieves a speedup of 62.1 at an area of 170.4 mm$^2$. HILP on the other hand predicts that the $(c_4,g_{16},d_2^{16})$ SoC will achieve a speedup of 45.6 at an area of 378.4 mm$^2$. MA's recommended design point $(c_1,g_{64},d_0^0)$ achieves 40% of the performance of HILP's recommended design point at 114% of the area, while Gables predicts that the $(c_4,g_4,d_3^4)$ SoC achieves 136% of the performance of HILP's recommended $(c_4,g_{16},d_2^{16})$ SoC at merely 45% of the area — a significant quantitative difference.

**WLP matters qualitatively.** The simplified treatment of WLP in MA and Gables also has qualitative effects. To explain this in detail, we need to consider the complete design spaces, i.e., Figures 7b and 7c. MA can only improve performance by including more and higher-performance accelerators. Its Pareto front is hence populated by GPU-dominated SoCs (see the green points in Figure 7b). The reason is no compute phases can be hidden when WLP is 1, and adding a (large) general-purpose GPU is thus the most area-efficient way of accelerating all applications. Gables is conversely biased towards SoCs with many smaller accelerators, see the blue and grey points in Figure 7c. The reason is that the workload becomes embarrassingly parallel when discarding phase dependencies, and Gables hence overly emphasizes SoCs with many accelerators.

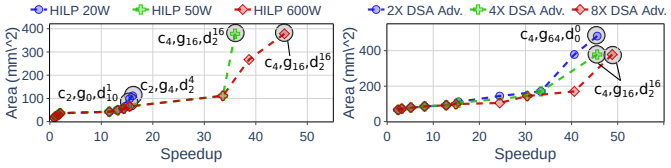
Figure 7d shows the complete design space as viewed by HILP. In contrast to MA and Gables, HILP respects phase dependencies and therefore recommends SoCs that cater to the specifics of the target workload. For example, HILP's highest-performing Pareto-optimal configuration is the $(c_4,g_{16},d_2^{16})$ SoC which combines 16-PE DSAs for *HS* and *LUD* with a 16-SM GPU. It is beneficial to allocate DSAs for *HS* and *LUD* because they are the applications in *Default* with the longest accelerator execution time (see Table II). Allocating DSAs to these applications thus takes sufficient load off the GPU for it to accelerate all the other applications in the workload without becoming a bottleneck — thus achieving maximum performance at minimal area overhead.

**WLP limits the benefits of heterogeneity.** The $(c_1,g_0,d_0^0)$ SoC is the only Pareto-optimal homogeneous SoC in Figure 7a and occupies the lowest performance point on the Pareto front. Architects should thus add accelerators before CPU cores, but our second key insight notes that provisioning sufficient CPU cores is critical to fully exploit WLP:

**Key Insight 2**

*Heterogeneity is critical, but neglecting dependencies and sequential phases leads to suboptimal SoCs.*

Adding accelerators is typically more beneficial than adding CPU cores because they improve performance more than they increase area; this is why the Pareto front is close to horizontal in the lower-performance end of Figure 7a. Interestingly, HILP identifies three clusters of SoC architectures with different accelerator configurations, and hence (widely) different area overhead, but the same number of CPU cores (see Figure 7d). Adding accelerators thus improves performance until the dependencies or sequential sections dominate, at which point adding more CPU cores can yield substantial performance improvements because they unlock the use of additional accelerators. For example, the best-performing Pareto-optimal 2-CPU




(a) Power constraint. (b) Efficiency advantage.

Fig. 8: DSAs versus GPUs. *DSAs are most effective when power is constrained and achieve high workload coverage.*

SoC ($c_2$,$g_4$,$d_2^{16}$) improves performance by 2.7× compared to the best-performing 1-CPU SoC ($c_1$,$g_4$,$d_2^1$).

**Specialized DSAs versus general-purpose GPUs.** Having established that heterogeneity is necessary to achieve high performance, we now use HILP to understand the relative benefits of general-purpose GPUs and specialized DSAs in heterogeneous SoCs. In essence, we find that architects should:

**Key Insight 3**

*Only use DSAs for application phases that dominate workload execution time.*

This insight can be observed among the SoCs in the 2-CPU and 4-CPU clusters in Figure 7d where several GPU-dominated SoCs achieve the same performance as the Pareto-optimal SoCs but at higher area overhead (i.e., there are green clusters just above the Pareto front). Most notably, the ($c_4$,$g_{64}$,$d_0^0$) SoC, which combines 4 CPUs with a 64-SM GPU and no DSAs, achieves the same performance as the Pareto-optimal ($c_4$,$g_{16}$,$d_2^{16}$) SoC, but it requires 482.4 mm$^2$ of area rather than 378.4 mm$^2$. Thus, adding DSAs pays off if architects are reasonably certain that their target workloads contain a small number of applications that require significant accelerator time (e.g., *HS* and *LUD* in *Default*). If not, it could be just as efficient to go for a general-purpose GPU.

**Power-constrained SoCs.** Figure 8a reports the Pareto front of the SoC design space with power constraints of 20 W and 50 W compared to our 600 W baseline. 20 W is a severe power constraint because we consider server-grade cores, e.g., our smallest GPU (16 SMs) consumes from 10.4 W to 24.6 W depending on the selected operating point and can thus easily use all available power. The power constraint does not affect the lower-performance Pareto-optimal SoCs because they do not consume enough power for the constraint to take effect. For higher-performance SoCs, the power constraint results in lower performance for the same area. A key advantage of DSAs when evaluated on single applications is their high power efficiency [15], but, perhaps surprisingly, we find that the GPU remains critical when considering whole workloads:

**Key Insight 4**

*SoCs benefit from combining DSAs with a general-purpose GPU even under severe power constraints.*

We focus on the highest performance Pareto-optimal SoCs with each power constraint. The ($c_4$,$g_{16}$,$d_2^{16}$) SoC is the top performer at both the 50 W and 600 W power budgets. The 50 W power budget limits the number of accelerators and CPUs that can be active concurrently, resulting in a 26.4% performance reduction relative to the 600 W case. At the 20 W power budget, the top performer is the ($c_2$,$g_4$,$d_2^4$) SoC which is a scaled-down version of the same architecture. Interestingly, the ($c_2$,$g_0$,$d_{10}^1$) SoC, which consists of a tiny DSA for all applications in *Default*, yields the second highest performance of the Pareto-optimal 20 W SoCs.

**DSA efficiency advantage.** Figure 8b reverts to the 600 W power constraint and explores the impact of changing the efficiency advantage the DSAs have over the GPU; recall that our baseline DSAs have a 4× efficiency advantage over the GPU. Making the DSAs more efficient indirectly increases their workload coverage because it enables accelerating more compute phases within the same area budget. The key takeaway is that changing the DSA advantage does not change the shape of the speedup versus area curve but moves it towards higher performance and area, leading to our final insight:

**Key Insight 5**

*Workload coverage is king for DSAs.*

The highest-performance Pareto-optimal points in Figure 8b are the ($c_4$,$g_{64}$,$d_0^0$), ($c_4$,$g_{16}$,$d_2^{16}$), and ($c_4$,$g_{16}$,$d_2^{16}$) SoCs for efficiency advantages of 2×, 4×, and 8×, respectively. The Pareto optimum hence moves from a GPU-only SoC at the 2× advantage to a mixed SoC at the 4× and 8× advantages. The ($c_4$,$g_{16}$,$d_2^{16}$) SoC yields higher performance at the 8× advantage compared to 4× because the DSAs are faster.

## VII. Extending HILP

Architects typically modify early-stage modeling frameworks such as MA, Gables, or HILP to fit the problem at hand. A key advantage of HILP, compared to MA and Gables, is that it is easily extensible, the key reason being that HILP clearly separates the formulation of the model from solving it.

**Modeling streaming dataflow.** To demonstrate the extensibility and flexibility of HILP, we consider a workload of independent instantiations of the Streaming-Dataflow Application (SDA) shown in Figure 9. Each SDA instance consumes data from three independent sources that must be mapped to specific DSAs. We label these initial phases DS1, DS2, and DS3. The outputs of the DS phases are input to a Data Fusion (DF) phase which joins the streams into a single buffer. This buffer is then input to three independent compute phases (C1, C2, and C3) which finally join in a Post Processing (PP) phase.

HILP as described in Section III cannot model SDA because its ordering constraint does not support fork-join parallelism (see Equation 2), but a HILP user can remove this restriction by using a graph to represent phase dependencies. (MA and Gables cannot model SDA because they do not explicitly model phase dependencies.) To model arbitrary dependencies, we first add the input matrix $D_{apq}$ in which a value of 1 (0) means that phase $a_q$ is dependent on (independent of) phase $a_p$. We then generalize Equation 2 to use $D_{apq}$ to decide if phases are independent rather than the raw phase identifiers:

$$S_{ap} + T_{cap} \leq S_{aq} \text{ if } D_{apq} = 1 \quad c \in \mathcal{C}, a \in \mathcal{A}, \{q, p\} \in \mathcal{P}_a. \tag{9}$$

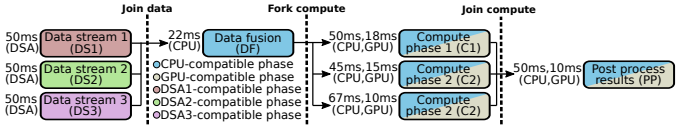
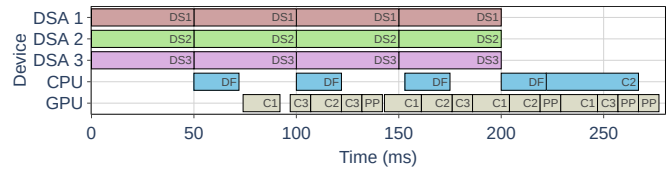
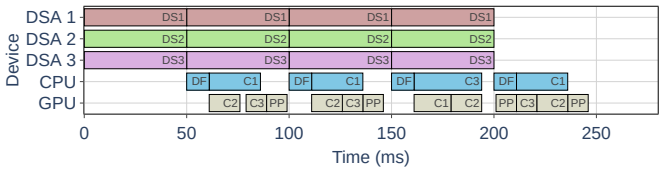


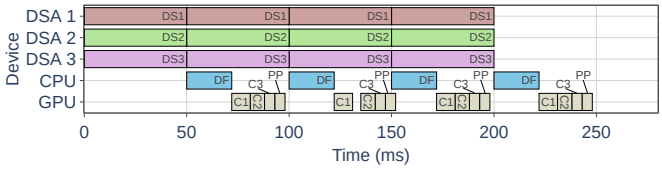

Fig. 9: Phases of the Streaming Dataflow Application (SDA).




(a) Throughput of the baseline SoC.




(b) Throughput of SoC with improved CPU.




(c) Throughput of SoC with improved GPU.

Fig. 10: HILP phase schedules of the SDA workload.

**Exploring SoCs for SDA.** The design objectives for the SDA SoC are to (i) execute the DS1, DS2, and DS3 phases in parallel, and (ii) overlap data stream processing for sample $i+1$ with the processing of sample $i$. DS1, DS2, and DS3 are hence pinned on individual DSAs. The DF phase must execute on a CPU, while C1, C2, C3, and PP can execute on either a CPU or a GPU. The execution time estimates for each phase on the baseline SoC is shown in Figure 9. The architect encodes dependencies with $D_{apq}$ and core type compatibility in $E_{cap}$. Figure 10a shows HILP's optimal schedule for the baseline ($c_1$,$g_8$,$d_3^1$) SoC, demonstrating that it falls short of its performance objective. The architect then considers moving to a CPU that is 2× faster (Figure 10b) or doubling the number of SMs in the GPU (Figure 10c), and HILP reports that both approaches yield sufficiently high performance. Figure 10b shows that the faster CPU can take on more work (i.e., C1 and C3). With the faster GPU, the CPU executes DF while the GPU attends to the the remaining phases (see Figure 10c).

**Other extensions.** The dependency graph extension above is by no means the only possible extension to HILP. We could for example further include explicit initiation intervals, i.e., require that a phase $a_i$ starts at the earliest $n$ time steps after the start time(s) of the phase(s) $a_i$ depend(s) on. Another interesting extension is to model the memory hierarchy in more detail. The first step would be to add new resource constraints that represent the bandwidth limits at each cache level (e.g., L1, L2, and LLC). Bandwidth curves for various memory system configurations can be obtained through various tools, e.g., Pin [38] or StatCache [6], and, from the perspective of HILP, changes in cache capacity would manifest as changes in these bandwidth curves.

## VIII. Related Work

The most closely related works to HILP are MA [43], [72] and Gables [28] because these are the only early-stage evaluation models that take WLP into account. They however only cover the extremes of minimal and maximal WLP which we demonstrated to be insufficient for exploring heterogeneous SoC design spaces in Sections V and VI.

**Non-WLP-aware early-stage models.** Roofline [64] is a high-level performance model that provides an upper-bound on application performance through its operational intensity and inspired numerous follow-up works (e.g., [30], [31], [41], [53]). While the models derived from Roofline primarily focus on bandwidth, models derived from Amdahl's law [4] primarily focus on modeling sequential and parallel phases. Marowka [40] uses Amdahl's law to study how to distribute the power budget across CPU and GPU cores, and Navigo [24] uses Amdahl's law to explore the balance between allocating area to CPUs and accelerators under power constraints. Hill and Marty [27] studied homogeneous, asymmetric, and dynamic CPU-only architectures and spurred several follow-up works that provide various extensions (e.g., [8], [13], [57], [66], [69]). A number of researchers have also leveraged Amdahl's law to study various aspects of homogeneous CPU-only multi-cores [8], [9], [12], [13], [19], [63], while Delimitrou and Kozyrakis [16] study the impact of CPU core configuration on tail latency in CPU-only datacenters. The common denominator of these approaches is that they, unlike HILP, MA, and Gables, do not account for WLP.

**Mathematical optimization in computer architecture.** Nowatzki et al. [46] provide an introduction to mathematical optimization for computer systems and include several case studies (e.g., instruction set customization). MA [43], [72] in fact leverages mathematical optimization, but, in addition to assuming minimal WLP, MA cannot handle the same breath of constraints as HILP because it relies on solving the model analytically. Liu et al. [37] apply mathematical optimization to understand how to optimally partition off-chip bandwidth, and AutoTM [26] uses integer linear programming to understand which tensors to allocate to persistent and regular memory when training deep neural networks. Mathematical optimization has also been used for mapping applications to processing elements (e.g., [33], [47]). Researchers used mathematical optimization to study multi-processors in the 60s and 70s (e.g., [18], [22]), but these models are not applicable to the highly heterogeneous SoCs of today.

**Other models and simulators.** Aladdin [54] is a pre-RTL framework for modeling single DSAs and thus orthogonal to HILP, i.e., Aladdin can be used to generate inputs to HILP for specific DSAs. BigHouse [42] is a high-abstraction simulator that focuses on CPU-only data centers and achieves short evaluation time and good scalability by combining queuing

theory and stochastic modeling. LogCA [3] models the offloading overheads of accelerators as a function of the amount of offloaded data, thereby predicting if offloading is beneficial or not, and Nilakantan et al. [45] explore how to select the best set of accelerators for a single thread as a function of the area dedicated to accelerators.

**Heterogeneous multi-core CPUs.** The benefits of heterogeneity was first demonstrated for CPU-only systems [34], [35], and a large body of work has investigated runtime scheduling approaches for such architectures (e.g., [11], [55], [62]).

## IX. Conclusion

We introduced HILP which is the first early-stage exploration approach for heterogeneous SoCs that fully accounts for Workload-Level Parallelism (WLP). HILP builds upon the observation that scheduling a workload of independent multi-phase applications on a heterogeneous SoC is an instance of the classic JSSP optimization problem and thus can be solved using integer linear programming. After validating HILP, we use it to sweep a design space of 372 SoCs and make several key observations. For example, when considering both performance and area, the top-performing SoCs use DSAs to offload the most accelerator-heavy applications from the GPU. Perhaps surprisingly, this trend persists even under severe power constraints because the benefits of DSAs over general-purpose GPUs are intimately tied to workload coverage.

## Acknowledgments

We thank the anonymous reviewers for their feedback. Joseph Rogers is supported by the Research Council of Norway (Grant No. 286596). Lieven Eeckhout is supported in part by Research Foundation Flanders (FWO) grants No. G018722N and G096225N, and UGent-BOF-GOA grant No. 01G01421. Magnus Jahre is supported in part by the Research Council of Norway (Grant No. 286596), the European Union Horizon 2020 program (Grant No. 101034240), and a gift from AMD.

## References

[1] Advanced Micro Devices, Inc., "AMD Instinct™ MI300A Accelerators," https://www.amd.com/en/products/accelerators/instinct/mi300/mi300a.html, 2024, accessed: April 12, 2024.

[2] Advanced Micro Devices, Inc., "AMD Ryzen™ 8000 G-Series Processors," https://www.amd.com/en/partner/articles/ryzen-8000G-series-processors.html, 2024, accessed: April 12, 2024.

[3] M. S. B. Altaf and D. A. Wood, "LogCA: A high-level performance model for hardware accelerators," in *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 375–388.

[4] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the Spring Joint Computer Conference (AFIPS)*, 1967, pp. 483–485.

[5] Apple Inc., "Apple unveils M3, M3 Pro, and M3 Max, the most advanced chips for a personal computer," https://www.apple.com/newsroom/2023/10/apple-unveils-m3-m3-pro-and-m3-max-the-most-advanced-chips-for-a-personal-computer/, 2023, accessed: April 12, 2024.

[6] E. Berg and E. Hagersten, "StatCache: A probabilistic approach to efficient and accurate data locality analysis," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2004, pp. 20–27.

[7] E. Brunvand, D. Kline, and A. K. Jones, "Dark silicon considered harmful: A case for truly green computing," in *Proceedings of the International Green and Sustainable Computing Conference (IGSC)*, 2018, pp. 1–8.

[8] K. W. Cameron and R. Ge, "Generalizing Amdahl's law for power and energy," *Computer*, vol. 45, no. 3, pp. 75–77, 2012.

[9] A. S. Cassidy and A. G. Andreou, "Beyond Amdahl's Law: An objective function that links multiprocessor performance gains to delay and energy," *IEEE Transactions on Computers*, vol. 61, no. 8, pp. 1110–1126, 2012.

[10] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 44–54.

[11] J. Chen and L. K. John, "Efficient program scheduling for heterogeneous multi-core processors," in *Proceedings of the Annual Design Automation Conference (DAC)*, 2009, pp. 927–930.

[12] S. Cho and R. Melhem, "Corollaries to Amdahl's law for energy," *IEEE Computer Architecture Letters*, vol. 7, no. 1, pp. 25–28, 2008.

[13] S. Cho and R. G. Melhem, "On the interplay of parallelization, program performance, and energy consumption," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 3, pp. 342–353, 2010.

[14] Y. Chou, B. Fahs, and S. Abraham, "Microarchitecture optimizations for exploiting memory-level parallelism," in *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, 2004, pp. 76–87.

[15] W. J. Dally, Y. Turakhia, and S. Han, "Domain-specific hardware accelerators," *Communications of the ACM*, vol. 63, no. 7, pp. 48–57, 2020.

[16] C. Delimitrou and C. Kozyrakis, "Amdahl's law for tail latency," *Communications of the ACM*, vol. 61, no. 8, pp. 65–72, 2018.

[17] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2011, pp. 365–376.

[18] M. R. Garey and R. L. Graham, "Bounds for multiprocessor scheduling with resource constraints," *SIAM Journal on Computing*, vol. 4, no. 2, pp. 187–200, 1975.

[19] R. Ge and K. W. Cameron, "Power-aware speedup," in *International Parallel and Distributed Processing Symposium (IPDPS)*, 2007, pp. 1–10.

[20] T. Gonzalez and S. Sahni, "Flowshop and jobshop schedules: Complexity and approximation," *Operations Research*, vol. 26, pp. 36–52, 02 1978.

[21] Google LLC, "OR-Tools - Google Optimization Tools," https://github.com/google/or-tools, 2024, accessed: July 29, 2024.

[22] R. L. Graham, "Bounds for certain multiprocessing anomalies," *Bell System Technical Journal*, vol. 45, no. 9, pp. 1563–1581, 1966.

[23] Gurobi, "The leader in decision intelligence technology - Gurobi optimization," https://www.gurobi.com/, 2024, accessed: July 29, 2024.

[24] M. Hempstead, G.-Y. Wei, and D. Brooks, "Navigo: An early-stage model to study power-contrained architectures and specialization," in *Workshop on Modeling, Benchmarking, and Simulations (MoBS)*, 2009.

[25] J. L. Hennessy and D. A. Patterson, *Computer architecture - A quantitative approach*, 6th ed. Morgan Kaufmann Publishers, 2018.

[26] M. Hildebrand, J. Khan, S. Trika, J. Lowe-Power, and V. Akella, "AutoTM: Automatic tensor movement in heterogeneous memory systems using integer linear programming," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020, pp. 875–890.

[27] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, 2008.

[28] M. D. Hill and V. J. Reddi, "Gables: A Roofline model for mobile SoCs," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2019.

[29] M. D. Hill and V. J. Reddi, "Accelerator-level parallelism," *Communications of the ACM*, vol. 64, no. 12, pp. 36–38, 2021.

[30] A. Ilic, F. Pratas, and L. Sousa, "Cache-aware Roofline model: Upgrading the loft," *IEEE Computer Architecture Letters*, vol. 13, no. 1, pp. 21–24, 2014.

[31] T. Koskela, Z. Matveev, C. Yang, A. Adedoyin, R. Belenov, P. Thierry, Z. Zhao, R. Gayatri, H. Shan, L. Oliker, J. Deslippe, R. Green, and S. Williams, "A novel multi-level integrated Roofline model approach for performance characterization," in *ISC High PerformanceProceedings*, 2018, pp. 226–245.

[32] R. Krashinsky, O. Giroux, S. Jones, N. Stam, and S. Ramaswamy, "NVIDIA ampere architecture in-depth," 2023, accessed: August 1, 2024. [Online]. Available: https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/

[33] M. Kudlur and S. Mahlke, "Orchestrating the execution of stream programs on multicore platforms," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2008, pp. 114–124.

[34] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2003, pp. 81–92.

[35] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-ISA heterogeneous multi-core architectures for multithreaded workload performance," in *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, 2004.

[36] A. Liu, P. B. Luh, B. Yan, and M. A. Bragin, "A novel integer linear programming formulation for job-shop scheduling problems," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5937–5944, 2021.

[37] F. Liu, X. Jiang, and Y. Solihin, "Understanding how off-chip memory bandwidth partitioning in chip multiprocessors affects system performance," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2010, pp. 1–12.

[38] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2005, p. 190–200.

[39] A. S. Manne, "On the Job-Shop Scheduling Problem," *Operations Research*, vol. 8, no. 2, pp. 219–223, 1960.

[40] A. Marowka, "Extending Amdahl's law for heterogeneous computing," in *International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, 2012, pp. 309–316.

[41] D. Marques, A. Ilic, and L. Sousa, "Mansard Roofline model: Reinforcing the accuracy of the roofs," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 6, no. 2, pp. 7:1–7:23, 2021.

[42] D. Meisner, J. Wu, and T. F. Wenisch, "BigHouse: A simulation infrastructure for data center systems," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2012, pp. 35–45.

[43] A. Morad, T. Y. Morad, Y. Leonid, R. Ginosar, and U. Weiser, "Generalized MultiAmdahl: Optimization of heterogeneous multi-accelerator SoC," *IEEE Computer Architecture Letters*, vol. 13, no. 1, pp. 37–40, 2014.

[44] N. Nethercote, P. Stuckey, R. Becket, S. Brand, G. Duck, and G. Tack, "MiniZinc: Towards a standard CP modelling language," in *Proceedings of the International Conference on the Principles and Practice of Constraint Programming (CP)*, 2007, pp. 529–543.

[45] S. Nilakantan, S. Battle, and M. Hempstead, "Metrics for early-stage modeling of many-accelerator architectures," *IEEE Computer Architecture Letters*, vol. 12, no. 1, pp. 25–28, 2013.

[46] T. Nowatzki, M. Ferris, K. Sankaralingam, C. Estan, N. Vaish, and D. Wood, "Optimization and mathematical modeling in computer architecture," *Synthesis Lectures on Computer Architecture*, vol. 8, no. 4, pp. 1–144, 2013.

[47] T. Nowatzki, M. Sartin-Tarm, L. De Carli, K. Sankaralingam, C. Estan, and B. Robatmili, "A general constraint-centric scheduling framework for spatial architectures," in *Proceedings of the International Conference on Programming Language Design and Implementation (PLDI)*, 2013, pp. 495–506.

[48] Nvidia Corporation, "Developer tools overview," https://developer.nvidia.com/tools-overview, 2024, accessed: July 11, 2024.

[49] Nvidia Corporation, "NVIDIA GH200 Grace Hopper Superchip Architecture," https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper?ncid=no-ncid, 2024, accessed: August 1, 2024.

[50] Nvidia Corporation, "NVIDIA Multi-Instance GPU," https://www.nvidia.com/en-us/technologies/multi-instance-gpu/, 2024, accessed: July 11, 2024.

[51] Nvidia Corporation, "System Management Interface SMI," 2024, accessed: July 11, 2024. [Online]. Available: https://developer.nvidia.com/system-management-interface

[52] V. J. Reddi, H. Yoon, and A. Knies, "Two billion devices and counting," *IEEE Micro*, vol. 38, no. 1, pp. 6–21, 2018.

[53] J. Rogers, T. Soliman, and M. Jahre, "AIO: An abstraction for performance analysis across diverse accelerator architectures," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2024.

[54] Y. S. Shao, B. Reagen, G. Y. Wei, and D. Brooks, "Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2014, pp. 97–108.

[55] D. Shelepov, J. C. Saez Alcaide, S. Jeffery, A. Fedorova, N. Perez, Z. F. Huang, S. Blagodurov, and V. Kumar, "HASS: A scheduler for heterogeneous multicore systems," *SIGOPS Operating Systems Review*, vol. 43, no. 2, pp. 66–75, 2009.

[56] M. Själander, M. Jahre, G. Tufte, and N. Reissmann, "EPIC: An energy-efficient, high-performance GPGPU computing research infrastructure," 2024. [Online]. Available: https://arxiv.org/abs/1912.05848

[57] X.-H. Sun and Y. Chen, "Reevaluating Amdahl's law in the multicore era," *Journal of Parallel and Distributed Computing*, vol. 70, no. 2, pp. 183–188, 2010.

[58] G. Tack and M. Z. Lagerkvist, "GECODE – An open, free, efficient constraint solving toolkit," https://www.gecode.org/, 2024, accessed: July 29, 2024.

[59] TechPowerUp, "AMD EPYC 7763," 2024, accessed: August 1, 2024. [Online]. Available: https://www.techpowerup.com/cpu-specs/epyc-7763.c2373

[60] The Linux Foundation, "Linux profiling with performance counters," https://perf.wiki.kernel.org/index.php/Main_Page, 2020, accessed: July 11, 2024.

[61] V. Timonen, "Multi-GPU CUDA stress test," 2024, accessed: July 29, 2024. [Online]. Available: https://github.com/wilicc/gpu-burn

[62] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (PIE)," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2012.

[63] U. Verner, A. Mendelson, and A. Schuster, "Extending Amdahl's law for multicores with turbo boost," *IEEE Computer Architecture Letters*, vol. 16, no. 1, pp. 30–33, 2017.

[64] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[65] M. Wissolik, D. Zacher, A. Torza, and B. Day, "Virtex UltraScale+ HBM FPGA: A Revolutionary Increase in Memory Performance (WP 485)," *Technical Report, Xilinx Inc.*, 2019.

[66] D. H. Woo and H. H. S. Lee, "Extending Amdahl's law for energy-efficient computing in the many-core era," *Computer*, vol. 41, no. 12, pp. 24–31, 2008.

[67] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *ACM SIGARCH Computer Architecture News*, vol. 23, no. 1, pp. 20–24, 1995.

[68] B. Yan, M. A. Bragin, and P. B. Luh, "An innovative formulation tightening approach for job-shop scheduling," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 3, pp. 2526–2539, 2022.

[69] E. Yao, Y. Bao, G. Tan, and M. Chen, "Extending Amdahl's Law in the multicore era," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 2, pp. 24–26, 2009.

[70] H. Yu, "Rodinia Benchmark Suite 3.1," 2023, accessed: April 7, 2024. [Online]. Available: https://github.com/yuhc/gpu-rodinia

[71] X. Zhao, M. Jahre, and L. Eeckhout, "HSM: A hybrid slowdown model for multitasking GPUs," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020, pp. 1371–1385.

[72] T. Zidenberg, I. Keslassy, and U. Weiser, "MultiAmdahl: How should I divide my heterogenous chip?" *IEEE Computer Architecture Letters*, vol. 11, no. 2, pp. 65–68, 2012.