Proceedings of the 55th Hawaii International Conference on System Sciences | 2022

A GPU-Accelerated Approach to Static Stability Assessments for Pallet Loading in Air Cargo

Philipp Gabriel Mazur
University of Cologne
mazur@wim.uni-koeln.de

No-San Lee
University of Cologne
lee@wim.uni-koeln.de

Detlef Schoder
University of Cologne
schoder@wim.uni-koeln.de

Abstract

The static stability constraint is one of the most important constraints in pallet loading and plays a substantial role when assembling safe and loadable palletizing layouts. Current approaches reach their limits as soon as additional complexity is added, which is a given in the practice of air cargo logistics, or when performance becomes important. As our central objective, we explore a new approach to calculate static stability more performantly and to cover more complexity by relaxing several simplifying assumptions. The approach is implemented in a prototype and builds on the emerging technology of graphical processing unit acceleration in combination with physics engines. We propose a new artifact design and summarize the how-to knowledge in the form of abstracted design principles. Our results demonstrate an improvement in terms of performance depending on the underlying hardware. We develop a conceptual model to assist future research in choosing a solution technology.

1. Introduction

In palletizing, an important process in air cargo logistics involves palletizers placing cargo on so-called unit loading devices (ULDs), which comprise pallets and containers. Stability of the cargo arrangement prevents damaged cargo and even personnel injuries [1–4]. Static stability ensures that items maintain their positions during loading, whereas dynamic stability applies to situations in which the pallet is being moved (e.g., in a vehicle). Static stability as a constraint is particularly relevant [1] and highly significant in practical usage [5] “yet [has been] inconsistently dealt with” [2] in previous works. In comparison to other static stability approaches (e.g., full base support, partial base support, static mechanical equilibrium), physical simulations can capture and process more practical complexity of stability assessments for ULDs and items as they build virtual images with real-world physical laws, thereby relaxing simplifying assumptions [6]. Hence, simulations can cope with the complexity of a broad variety of rigid or soft body shapes and can include sub-pallets and additional physical properties, such as non-uniform weight distribution.

Physical simulations with real-time physics engines provide sufficient precision to approximate realistic cargo-loading scenarios compared to high-precision dedicated physical simulators but deliver results in a fraction of time [7].

However, simulations are computationally expensive and are thus hard to integrate in optimization heuristics [8] used for searching for good palletizing solutions (so-called *layouts*). An acceleration (e.g., through parallelization) of an underlying physical simulation can help achieve faster stability evaluations and, in consequence, increase overall solution quality. General-purpose computation on graphics processing units (GPGPU) massively parallelizes computations based on modern graphical processing units’ (GPU) computational capabilities [9] and can be utilized for physical simulations [10]. To the best of our knowledge, previous studies have neglected the opportunities emerging from the field of GPGPU for this problem context. For this study, our research goal is to propose a new design for a GPGPU-based approach to assess static stability. Consequently, we ask the following research question: *How can a static stability assessment approach for air cargo palletizing be designed using GPGPU?*

To achieve our goal and to answer our research question, we follow the design science research (DSR) approach introduced by Peffers et al. [11]. We ground our design in theoretical findings on simulation-based static stability assessments and practical insights from an ongoing collaboration with a major German cargo carrier. Our contribution consists of a design for a new approach to assess static stability for air cargo palletizing that better meets practical requirements by including substantially more real-world complexity and improving performance. Further, we develop a conceptual model that maps execution technology to problem configuration. From a practitioner’s view, our prototype could likely be employed in complex problem cases that require significant compliance with

URI: https://hdl.handle.net/10125/79536
978-0-9981331-5-7

transportation safety rules, as is the case in air cargo operations.

The remainder of this work is as follows. In Section 2, we provide an overview of related work. Afterward, we elaborate on our methodology in Section 3. We then present our main results in Section 4, which are then discussed and concluded in Section 5.

2. Related Work

Approaches to physical simulations can broadly be classified into the categories of stand-alone and integrated. The first category comprises approaches that utilize physics engines to train or develop a model that is deployed afterward in an optimization heuristic. The latter represents approaches that directly execute the simulation from within a heuristic.

The stand-alone approach is described and applied in works by Ramos et al. [12]; Ramos et al. [8]; and Martínez, Cuellar, and Álvarez-Martínez [13]. In most cases a linear or mechanical model approximates the physical outcome [13]. To evaluate such models, a physics engine can be used. The studies of Ramos et al. [8] and Ramos et al. [12] resulted in a stand-alone simulation software prototype (StableCargo) for calculating dynamic stability behavior for container loading problems. The authors benchmarked their physics engine–based stand-alone tool against a high precision engineering simulator and analytical solutions, thereby modeling typical transport forces and velocities. StableCargo can approximate analytical and high-precision solutions. However, the authors provided no report about StableCargo's integration into an optimization heuristic. Martínez, Cuellar, and Álvarez-Martínez [13] presented a mechanical model that predicts the outcome of a physics engine in the case of dynamic stability. Their developed algorithm calculates the number of fallen boxes based on the mechanical model.

In the past, capturing stability within a model frequently requires imposing simplifying assumptions (e.g., uniform mass distribution, box-shaped items), which can make it difficult to adapt to new or unexpected data. This issue is particularly relevant in the air cargo context, which combines substantial cargo and ULD heterogeneity [14, 15].

This problem has been addressed with the integrated approach, which can be found in Bracht et al. [16] and Mazur et al. [6]. Physical simulations are a versatile technique that support a broad spectrum of shapes and include meta-characteristics, such as displaced mass distributions and material properties. Real-time physics engines operate sufficiently precisely to overcome simplifications. They obtain realistic physical feedback as they build virtual images with similar physical laws and behavior as in real-world contexts (e.g., gravity, friction, collisions). They also provide enough interfaces and sufficient runtime performance to be generally used in heuristic search processes, although they heavily slow down the application. However, studies on their integration within solution-generating heuristics are scarce. The results from Bracht et al.'s work [16] demonstrated the integration of dynamic stability verification into a genetic algorithm (GA) meta-heuristic using a physics engine. The authors performed two stability tests: (1) one local stability test that is executed for every single box in a layout and verifies if the center of gravity of a box is supported by the supporting convex hull of previously packed boxes and (2) one static and dynamic stability verification test using simulations. Their results revealed that physics simulations consumed most of the total processing time, prevented the meta-heuristic from reaching a high solution quality level, and decreased its convergence with satisfactory layouts. Similar results were reported from Mazur et al. [6], who demonstrated the integration of physical simulations in an optimization heuristic for pallet loading in air cargo. Their prototype employed a physics engine to iteratively load a cargo layout within a GA with aviation safety constraints. The authors reported long runtimes with moderate solution qualities. In both studies, an optimization heuristic executes simulations with a physics engines, which return information about the spatial positions of cargo items for a given time interval. Further, both approaches draw attention to the problem of accelerating simulations since underlying heuristics heavily depend on fast evaluations of their candidate solutions. The more runtime a heuristic spends in a candidate solution's fitness evaluation, the less runtime it can spend searching for better solutions

One option to accelerate physical simulations is to parallelize and distribute problem calculations upon many-core processors, such as GPUs. GPGPU refers to GPU employment for non-graphics computation [17]. Today, researchers and practitioners choose GPGPU to achieve high computational performance at low costs. Specifically, GPU computations are considerably faster at floating-point calculations and can allocate work to the GPU to balance loads [18]. The compute unified device architecture (CUDA) includes a unified shader pipeline, enabling all shading multiprocessors to be used for general-purpose computations with single-precision floating-point arithmetic and general computation instruction sets. Due to the hardware design, not all computational tasks are well suited for GPU processing. GPGPU is efficiently applicable if the problem is data parallel and can be decomposed into smaller independent sub-problems [18]. In general, applications that contain a high ratio of mathematical operations compared to memory access achieve good performance [19]. Examples for GPGPU applications range from sorting, database operations, and image processing [17] to physically based simulations and illumination models, such as raytracing and volume processing. In rigid body simulation cases, GPGPU can help accelerate and parallelize computations. As an application, Harada [10] demonstrated the rigid body physics calculation for a large number of bodies entirely on the GPU.

3. Approach

We base our approach on the DSR framework, which focuses on "(1) creation of knowledge through design of novel or innovative artifacts and (2) the analysis of the artifact's use and/or performance with reflection and abstraction" [20]. Within this framework, we present a new approach (accelerated, integrated GPU physical simulations) for an already-known problem (static stability assessment in pallet loading), which marks an *improvement* compared to the state of the art. An improvement aims to "create better solutions in the form of more efficient and effective products, processes, or ideas" [21]. The differences of an improvement from already existing solutions must be clearly presented along with an extensive evaluation, thereby delivering evidence for increased "efficiency, productivity or other quality measures" [21].

Specifically, we employ the (DSR) methodology by Peffers et al. [11]. This methodology's core component consists of the nominal process sequence for conducting DSR, which is composed of six phases: problem identification and motivation, objectives of a solution, design and development, demonstration, evaluation, and communication.

Multiple sources, including new technological developments, can act as starting points for problem awareness [20]. We first provide a description from a practitioner's perspective. During our research project, we collaborated with a major German air cargo carrier. For several months, our team, which consisted of up to eight researchers, conducted joint workshops with two experts from the air cargo operations field, met two palletizers, and visited palletizing operations and aircraft-loading operations at a large German cargo hub multiple times. Over the course of more than eight workshops, we developed a common problem understanding and received an impression of the practical implications of loading stability and its impact on safe air transportation. Consequently, we allow our understanding of the identified problem to flow into the identified problem and gave special consideration to practical requirements. In the solution objectives definition phase, solution requirements are inferred from problem specification in terms of their feasibility. They can be of quantitative or qualitative nature [11, 20]. At the heart of the DSR methodology in the design and development phase, our artifact is constructed [11]. During the phase, we first developed both a target functionality and a high-level architecture. Afterward, we searched for relevant libraries that provided out-of-the-box functionalities (e.g., API and physics engine). We grounded our selection of a physics engine in published comparisons of freely available engines. Finally, we incrementally built the artifact using agile software development. During the demonstration phase, one or multiple problem instances are solved. In this study, we demonstrate and evaluate our artifact's functionality using one problem instance derived from a benchmark dataset [14]. The evaluation stage addresses the comparison of an artifact's desired goodness of fit to its actual goodness of fit (e.g., through hypothesis tests about the artifact's behavior or benchmarking). Deviations from this comparison must be tentatively explained and might result in a successive iteration of the DSR cycle [20]. Primarily, we evaluate if our artifact marks an *improvement* with respect to the quality dimensions obtained from the solution objectives definition. We compare our loading algorithms' performance for varying problem sizes. Furthermore, we deploy a set of one-sided two-sample Welch's t-tests [22] to test performance improvements. Welch's t-tests are well suited to compare means of multiple samples and are specifically robust when it comes to deviating variances and sample sizes. We select Cohen's d as a measure of effect sizes.

4. Results

4.1. Problem Identification and Motivation

From a practitioner's perspective, the need to improve performance arises due to multiple factors. In the time-critical air cargo sector, approaches to accelerate and reduce execution time help carriers meet strict flight-departure deadlines. Given the runtime of integrated simulation-based approaches, overall solution quality might be insufficient to achieve desired volume-utilization goals and service levels. As a key driver of runtime, stability assessments and their accuracies are substantial issues encountered in practice. In air cargo operations, carriers transport vastly heterogeneous items [14, 15]. Simplifying assumptions about cargo or insufficiently modeling cargo or ULD characteristics is likely to produce inaccurate or wrong stability predictions. In light of the presented motivation, we shape the DSR problem as follows. In the air cargo context, given a pallet-loading optimization heuristic, we need to find a better performing but equally accurate solution to static stability assessments than previous approaches and to capture more complexity in terms of meta-information. The main input from the heuristic is multiple cargo layouts, which span different ULDs and item-arrangement information. Furthermore, items are characterized by their specific shape, weight, center of mass, and sub-pallet. Every item is assigned a final 3D position on a ULD and a loading sequence, which determines the loading order. The artifact must return a quantified numerical static stability assessment value reflecting a layout's amount of static stability.

4.2. Solution Objectives

Since our research goal is to create an *improvement*, we need to derive relevant characteristics to enable qualitative comparison between approaches. Accordingly, we use two main characteristics. (1) Increased performance: our artifact should evaluate static stability considerably faster (in terms of wall-clock time) for a given set of cargo layouts. (2) Cover more complexity: to increase the practical relevance of palletizing solutions, we opt to cover as much complexity found in practice as possible, resulting in more realistic evaluations of static stability. In this study, we operationalize complexity using enhanced modeling of items and ULDs.

4.3. Design Requirements and Principles

GPUs require different data structures and flows than central processing units (CPUs). We design our system's workflow and its corresponding simulation such that it considers GPU characteristics (e.g., high throughput, high latency) and GPGPU specifications (e.g., stream processing, data parallelism, independence) (Design Requirement, DR1). We acknowledge a tradeoff between simulation resolution and performance that meets practical goals (DR2). Together, a well-suited workflow and an optimized tradeoff leads to good task-technology fit. One substantial drawback of GPU-based applications is the inherent dependence on dedicated graphics hardware. This dependence affects virtualization and ultimately portability. Therefore, our design pays attention to hardware bindings (DR3). To track simulation workflows and time steps to ensure simulation correctness and to communicate solutions to stakeholders, our design includes a visualization (DR4). In air cargo logistics, palletizers cope with a broad range of item shapes, combined with low identical shape frequency. Therefore, our design must consider item shape and support irregular forms (DR5). To achieve a high amount of practical relevance and to overcome simplifications with respect to cargo, we need to include weight distributions (DR6) and sub-pallet (DR7) information. Finally, different types of ULDs exists and should be realistically modeled (DR8). To meet our design requirements on an abstract level, we develop a set of design principles:

DP1: Batch Processing. To exploit the parallelism abilities and deep pipeline of GPU processing, a high problem size needs to be transmitted and processed concurrently on the GPU. It addresses DR1-2.

DP2: Loose Coupling. To overcome tight dependency in terms of hardware and technology, the stability evaluation should be decoupled from the heuristic and dedicated physics engine. This principle ensures a maximum level of portability and interchangeability between the heuristic, stability simulation, and physics engine. This decoupling enables access using exposed interfaces and dedicated platforms. Further, reusable and loosely coupled components enable the system to be employed for non-stability purposes. It addresses DR3.

DP3: Visualization. Placement, movement, and simulation validation require visual insights into the simulation world. Therefore, one or multiple visualizations can help designers and users understand a simulation's mode of operation and facilitate validation and common problem understanding. It addresses DR4.

DP4: Enriched Body Representation. To achieve high practical relevance, item and ULD representations should be as rich as possible in terms of meta-information. Counter-intuitive to the object oriented (OO) principle of information hiding, it is key to transfer as much information about the physical attributes of an item or ULD as possible to the physics engine. It addresses DR5-DR8.

4.4. Design and Development: Architecture

In the following, we present an architectural overview about the system's components, interfaces, and dependencies, which we illustrate in Figure 1. Our core system comprises three components: *Simulation, WebServer*, and *ProcessControl*. *Simulation* manages the stability simulation workflows, interacts with the physics engine, and obtains simulation results. It wraps the communication with the physics engine in a way that allows interchangeability between multiple engines as they collectively require similar prerequisites (simulation world buildup, item creation, and placement) and workflow calls (step simulation). The physics engine provides out-of-the-box support for GPU acceleration using the CUDA API, which we adapt to our requirements. For validation purposes, the *Simulation* component provides a dynamic visualization of items, ULDs, and their movement. The *WebServer* makes the system reachable from outside by adding an exposed API. The heuristic provides the cargo layouts and loading sequences. The *ProcessControl* acts as a middleware between *WebServer* and *Simulation*. Every process is a newly created problem instance, which simplifies memory management.

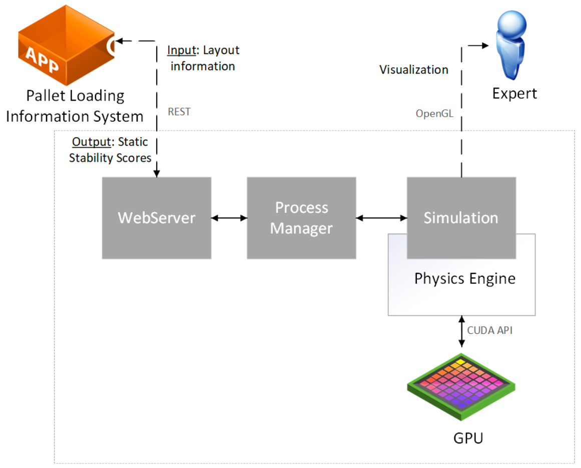

Figure 1: High-level architecture

4.5. Design and Development: Instantiation

We code the system in C++, a language specifically designed for high-runtime performance as opposed to compilation performance and developer comprehensibility. It is widely used in robotics, graphical applications, scientific simulations, and games. Further, many physics engines are coded in C++, which facilitates our artifact's integration with other engines. We opt for PhysX [23] as our system's physics engine for multiple purposes. First, PhysX achieves good performance along important use-case dependent dimensions compared to other engines [24, 25]. Second, PhysX is shipped with a GPU acceleration module. It is specifically designed to be integrated with CUDA-compatible GPUs. API and class structure between the CPU and GPU remain the same. The module processes all rigid bodies on the GPU using a CUDA program [23]. GPU acceleration supports two substantial rigid body pipeline components: broad phase and rigid body dynamics. Both relate to contact generation, shape and body management, and constraint solving. Furthermore, PhysX employs object sleeping as internal optimization to save computational effort [25]. When objects do not move for a period of time, it is assumed they will not move in the future except when they experience external impact or forces, which fits our equilibrium state condition well. During sleep, they are no longer simulated [23].

4.5.1. Modeling ULDs and Items. A ULD belongs to one of two types—a pallet or a container. Both have a bottom area and a fixed contour. A pallet lacks walls and is usually secured by a safety net. A container has rigid walls and a loading door, both of which affect support from the side and therefore static and dynamic stability. Consequently, in our simulation, containers have rigid walls while pallets receive no lateral support. The container door is spared out. Items have a geometric shape, loading sequence, weight specification, and material-related properties. The loading sequence determines the order in which items are placed on a ULD. Based on the cargo shapes found in practice and the selection of collision shapes supported by the physics engine, we specify a set of eight supported shapes (box with sub-pallet, box, cylinder, polygon prism, sphere, convex, L-shape, capsule), which we depict in Figure 2. Not all shapes found in practice can be simulated; since we only consider rigid bodies, a limited set of base shapes is supported and efficiently processable (specifically concave bodies are hard to collision test) or supported for GPU acceleration. The selection is a tradeoff between flexibility and computability. An item's weight can be either uniformly distributed, have a center of mass specified by a point in the item's coordinate system, or be modeled through a weight-distribution function. In this work, we use the GA-based smart palletizing information system [15, 26] as the optimization metaheuristic. However, our approach might be generalized to any kind of metaheuristic that deals with the search for good layouts by holding an entire population of solution candidates.

In contrast to common GA logic and in accordance with DP1, we first collect all solution candidates in a population and send them collectively to our artifact.

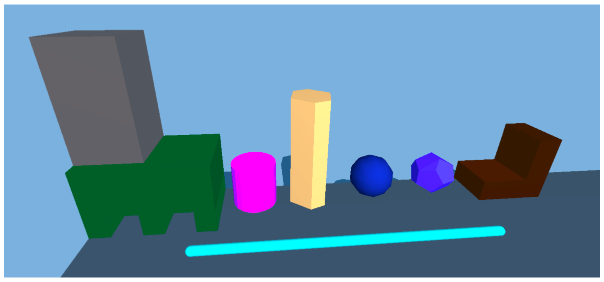

Figure 2: Supported item shapes

4.5.2. Loading Algorithms. We derive our first design principle from GPU computing characteristics: (1) there is a high communication latency between the host (CPU) and device (GPU) and (2) if the memory is allocated and data are transferred, the GPU heavily parallelizes computations among CUDA cores. Hence, a loading algorithm must minimize communication and maximize computations on the device. On the other hand, we need to minimize unnecessary simulations to the greatest extent possible. Unnecessary simulations occur for simulated layouts that have already been proven to be unstable, for example, if a previously placed item in the loading sequence is unstable. Both, parallelization and minimization of unnecessary layouts represents a tradeoff. To explore this tradeoff, we incorporate three different loading algorithms into our design: single-ULD, multi-ULD, and all-sequence-ULD loading. Their theoretical complexity is depicted in Table 1. P equals the number of layouts in the optimization heuristic's iteration and N equals the highest loading sequence present in the population, which equals the maximum number of items placed on a ULD in this population. We ground our algorithms on the *Sim2* placement algorithm of Mazur et al. [6].

Table 1: Loading algorithm complexities

Approach	Max. number of simulations	Max. number of concurrently simulated bodies
Single-ULD	$O(P \times N)$	$O(N)$
Multi-ULD	$O(N)$	$O(P \times N)$
All-sequence-ULD	$O(1)$	$O(P \times \frac{N \times (N+1)}{2})$

Single-ULD loading operates on the single-layout level. All layouts are processed sequentially, and all items in the sequence are placed sequentially. The outcome of the subsequent process step (loading of an item with the sequence $N + 1$) depends on the outcome of the previous step (loading of item N). The simulation number equals the ULD frequency times the amount of sequences for every ULD. It refers to the smallest amount of concurrent calculations. Using multi-threading, single-ULD loading can be executed on multiple cores concurrently; however, no interaction or synchronization occurs between loadings. The characteristics map well to CPU execution. For GPU execution, the problem is not well suited since only one concurrent simulation context is feasible and therefore no multi-threading is possible at this level.

Multi-ULD loading operates on the population level. The algorithm places all layouts and processes each loading sequence for every layout concurrently. For example, all layouts in the population, along with the first item in each layout's loading sequence, are placed and simulated together. In the next iteration, all layouts place their second item according to their loading sequence. This process is repeated until no more sequences are present. This approach minimizes unnecessary simulation to the greatest extent possible.

In general, a loading algorithm whereby all items are placed and simulated alternatingly includes synchronization between loading sequences. Mazur et al. [6] employed this step to sort out unstable layouts. This is called the "early-out" test. The number of simulations equals the highest number of loading sequences in the population. The number of concurrently simulated rigid bodies equals the number of layouts times the current sequence. The multi-ULD loading algorithm directly maps to the physical loading procedure and is the only algorithm that ensures static stability at each sequence step. It enables parallel processing of all loaded items of the same loading sequence; thus, within one loading sequence, operations on the entire population can be distributed between GPU cores. Each layout is an independent sub-problem, which fits GPU execution well.

The all-sequence-ULD loading operates on the population level. Our algorithm places all layouts, along with all sequences, separately at the same time without synchronization. The number of simulations equals one. The number of simulated bodies equals the number of ULDs times the number of sequences. This algorithm heavily exploits parallelization but also includes unnecessary layouts. The problem becomes broader, which fits GPU execution well. Within only one simulation, it processes all possible ULD sequence combinations, which might decrease performance. Although GPUs are parallel processors, the number of cores is limited; thus, parallelism and unnecessary computations must be carefully balanced.

4.5.3. Visualization. Our system comprises two visualizations: an all-purpose simulation sampler and an extendible placement visualization. For the all-purpose simulation sampler, we employ PhysX Visual Debugger (PvD).[1] It samples simulation meta-data, such as memory usage, object collisions, and positional data for varying time frames (Figure 3 and Figure 4). We employ PvD to develop an understanding of our loading algorithms' workflows and correctness. It illustrates artifact workflows to stakeholders and practitioners and samples generation progress for the solution-generating heuristic. As PvD is shipped closed source, we develop a custom visualization based on OpenGL[2] such that it enables interactive dynamic item placement, displacement measurements, and tools for time stepping (Figure 2). We employ this visualization to achieve an understanding of displacement behavior and physical shape construction.

4.6. Demonstration

To demonstrate our design, we use the dataset published by Brandt and Nickel [14], which provides a high amount of realism and richness in terms of meta-information (e.g., item availability, ULD contour,

[1] https://developer.nvidia.com/physx-visual-debugger
[2] https://www.opengl.org/

allowed rotations, offload penalties, and commodities). In terms of shape, the dataset only contains boxes; hence, we assign fixed shares of irregular items. Typical irregularity ratios range between 5% or 20% [15]. We use one ULD and 50 items out of the flight at random to create an input shortage and limit the item pool size for our heuristic (single knapsack problem). Further, we assign meta-information as follows. We randomly add a sub-pallet to box items in 50% of cases. According to our experts, approximately every second item arrives pre-palletized on a wooden pallet. We assign each item a center of mass. For the majority of items, we assume a fixed density throughout the item's body (80%); for the remaining items, we randomly assign a point in the 3D item space as the center of mass, following a uniform random distribution over the items' dimensions. We use two randomly selected jobs: one pallet job (LH8080-23NOV15-FRASIN-PMC-F) and one container job (LH8164-24NOV15-FRA-OR-AKE). We set the population size to 10,000 and generations to 300. The optimization heuristic incorporates our GPGPU-accelerated static stability criterion as an assessment criterion. We track overall runtime of the heuristic, which mainly drives our solution's capability in terms of performance. We measure runtimes for each generation to observe the development over time for both job configurations. Exemplary in-between buildup states are visualized in Figures 5 and 6.

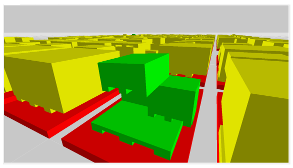
Figure 3: Buildup of PMC-F (simulation sampler)

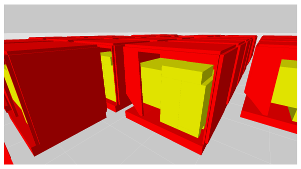
Figure 4: Buildup of AKE (simulation sampler)

We can see considerable differences between both jobs. While the stability simulations for pallets cause an overall heuristic runtime of about 35 minutes, the container simulations take considerably longer—over 90 minutes. An explanation for this deviation might be the collision tests of the cargo with the container walls. We model container walls as static rigid objects in the simulation world, which affects the items' number collisions. Not only can they collide with each other, but they can also collide with the static container walls. Further, the final container solution places more items (13 vs. four) compared to the pallet solution, which increases the number of sequences and ultimately the number of collisions. Moreover, an increased number of items attached to sub-pallets and items with displaced centers of masses are present.

4.7. Evaluation

In the following, we evaluate if our system marks an *improvement* in terms of performance. We refer to CPU execution as the CPU-only *Sim2* of Mazur et al. [6] and GPU execution as the GPGPU-accelerated approach presented in this study. We first develop a conceptual mapping between the loading algorithm and approach (CPU, GPU). This model is then evaluated on a benchmark with all three loading algorithms for two different population sizes (1000, 8000). As a result, we come up with an empirically validated conceptual mapping. Our operationalized hypothesis is

$$H_0: \mu_{CPU} \leq \mu_{GPU} \text{ vs. } H_1: \mu_{CPU} > \mu_{GPU},$$

where μ_{GPU} and μ_{CPU} represent mean performance in terms of the runtime of the GPU and CPU execution, respectively. Performance is a measure highly dependent on the underlying hardware. To overcome a bias toward a dedicated hardware configuration, we run benchmarks on three different hardware configurations (M1, M2, M3). Their technical specifications can be found in Table 2. Note that GPU execution is an acceleration technique and means no independence from the host CPU configuration.

Table 2: Hardware platform configuration

	M1	M2	M3
CPU cores/ threads	16/32	24/48	4/8
CPU base/boost clock (GHz)	3.5/4.4	3.8/4.5	2.8/3.8
GPU CUDA cores	4608	3584	640
GPU base/boost clock (GHz)	1.35/ 1.77	1.48/1.58	1.35/ 1.45
GPU memory	24 GB GDDR6	11GB GDDR5X	2 GB GDDR5

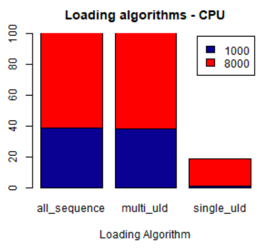

Figure 5: CPU benchmark results

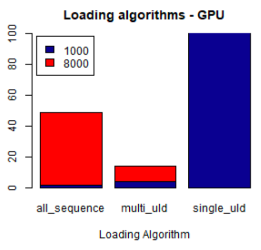

Figure 6: GPU benchmark results

We use three different cargo complexity scenarios. Scenario A represents a loading scenario with only homogeneous boxes, Scenario B includes different heterogeneous box dimensions, and Scenario C mirrors loading situations found in practice with 5% irregular shapes. Further, for every complexity scenario, we evaluate three different population sizes (1,000, 8,000, 20,000). We let problem complexity (in the form of scenario) and problem size (in the form of population size) vary to illustrate scaling behavior with respect to problem size and problem complexity. Furthermore, to preserve comparability between approaches, we remove deviating information (e.g., centers of mass, sub-pallets) and set generation size to one. GA outcomes are statistically dependent on the previous generation and are therefore not Gaussian distributed [27]. Since the heuristic employs a random populator, the first generation is randomly sampled, which preserves the *iid* assumption we need to employ t-tests.

4.7.1. Conceptual Model. The results of our loading algorithm evaluation are displayed in Figures 7 and 8.

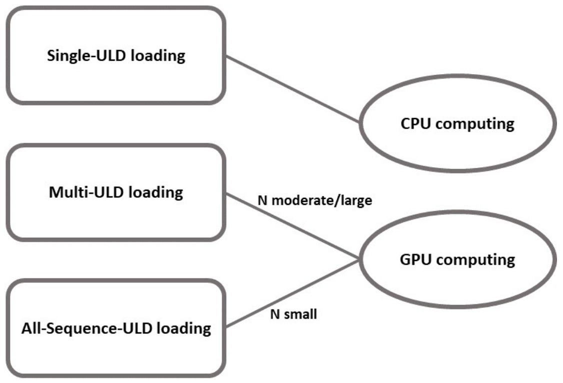

Figure 7: Conceptual model of loading approach and technology

Facing CPU execution, the single-ULD loading algorithm performs best. For GPU-execution, all-sequence-ULD loading and multi-ULD loading perform considerably better than single-ULD loading. For the smaller population size (1,000), all-sequence-ULD loading outperforms multi-ULD loading, whereas for the larger population size (8,000), multi-ULD loading proves better on the GPU.

These results may be due to a limited optimal number of parallel executions (i.e., problem size), which is determined by the limited number of GPU cores. The higher the number of parallel processing units, the more rigid bodies might be simulated concurrently. Because of this tradeoff, we expect that for increasing population sizes, the difference between all-sequence-ULD loading and multi-ULD loading becomes even larger. For small population sizes, we recommend using all-sequence-ULD loading, while for medium to large sizes, we prefer multi-ULD loading. For CPU execution, only single-ULD loading provides reasonable performance. The conceptual model is depicted in Figure 7. *N* refers to the problem size (number of parallel executions).

4.7.2. Performance Evaluation. We present our results for the Welch's t-tests for performance in Table 3. For most platforms and problem combinations, we can reject the null hypothesis, always at a highly significant level ($p < 0.001$). This is not surprising as we consider large sample sizes. For M3, we can uniformly reject the null with large effect sizes (Cohen's $d > 0.8$) except for the A-8000 and C-8000 scenarios. For M1, we can observe that GPU execution outperforms CPU execution for moderate to large problem sizes. For small problem sizes, GPU proves to be inferior. For small problem sizes, the ratio of calculations to communicational and memory access overhead (*arithmetic intensity*) on the device is low. This situation is not well suited for GPGPU. For larger problem sizes, the ratio increases, so GPU scales better than CPU. The larger the problem size, the larger the effect, which we can observe in the increasing effect sizes of the Cohen's d. For M2, CPU keeps up and scales roughly as good as GPU. This result may be due to the hardware configuration for M2, in which the CPU is composed of 48 cores. In our hardware configuration set, this CPU has the highest clock rate and the highest number of cores. Having more cores implies better parallelization capabilities even for multi-threaded single-ULD loading algorithms. In general, we can observe the tendency that larger problem sizes imply longer absolute runtimes. When we set the effect in relation to problem size, we see a scaling effect. We calculate the ratio of mean runtime and problem size for the M1 case and visually depict them in Figure 8. The CPU scaling curve is flatter in comparison to the GPU scaling curve, whereas GPU converges faster.

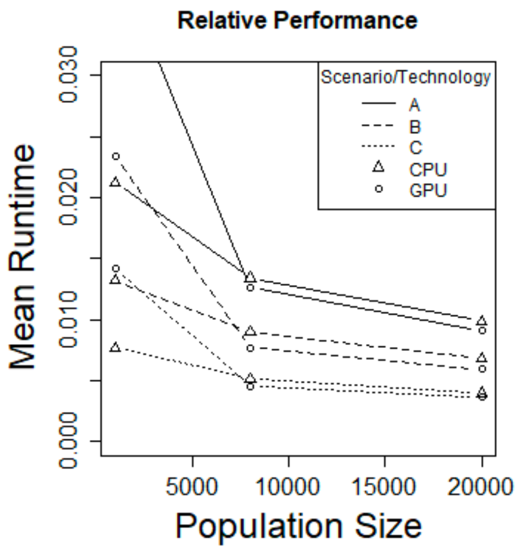

Figure 8: Relative performance

5. Discussion and Conclusion

With this study, we tackled the research and practical problem of improving integrated physical simulations in optimization heuristics for pallet loading using GPGPU. With the presented approach, research in the area of pallet loading is equipped with an innovative parallel method to evaluate static stability. Our design is composed of eight design requirements and four design principles. Our design principles can be further employed in other problem classes that require fast but precise physical feedback for artificial intelligence and tackle the problem of employing modern GPUs' compute capabilities while coping with large problem sizes with thousands of physical bodies with heterogeneous shapes. Our results demonstrate an improvement in terms of complexity and, depending on hardware platform, performance. For most platforms, our approach outperforms the state-of-the-art CPU approach.

Previous studies have presented methods that are not integrated in the optimization heuristic or heavily impact performance. With our approach, we accelerated physical simulations, which can provide leverage for the optimization to reach a higher solution quality. We took a step toward a practically relevant pallet-loading solution by intentionally setting our focus on the inclusion of meta-information and shapes that mirror the complexity found in cargo operations. We shed light on the problem of sub-pallets and non-uniform gravity. We modeled eight distinct item shapes and two ULD types, which cover a broad spectrum of bodies found in cargo operations.

In the following, we emphasize a set of our work's limitations, which is by no means complete. First, we imposed the assumption of a valid physical simulation as an outcome of the physics engine. Thus, our approach primarily lacks validation against a ground truth, for example, in the form of a validated test dataset or a high-precision physical simulator. Second, we only included a limited number of shapes. In many operational cases, this might be sufficient, but during our onsite visits, we observed a multitude of shapes that are not easily assignable to one of the proposed simulation shapes (e.g., cars or turbines). Further, we omitted concave shapes. Collisions of concave bodies are harder to evaluate for physics engines; thus, their inclusion would likely slow down execution. Furthermore, our assumption of rigid bodies might be challenged. During our onsite visits, we observed multiple items with deformable packing material (e.g., cardboard) and loading security tools (e.g., nets and straps), which are not captured by rigid body simulation. Finally, we tested our approach on only a small sample of selected flights, thus limiting our inferential conclusions. With respect to stability assessments in pallet loading context, further research should find a way to balance realism and runtime, such that realistic stability approximations that cope with practical complexity can be obtained in shorter time frames. Although our approach marks an improvement, runtimes still should be considerably faster to meet practical goals.

Our conceptual model is a starting point as it provides an approximation of the best fit between approach and problem size. Our evaluation of performance is restricted to the small set of hardware we tested on. In terms of future opportunities, model validation against a physically crafted and virtualized layout is paramount. Another extension of our simulation might be to evaluate other constraints apart from static stability, for example, dynamic stability, loading bearing capacity, or balancing.

Table 3: Performance evaluation results

	Scenario	**A**			**B**			**C**		
Platform		1000	8000	20000	1000	8000	20000	1000	8000	20000
M1	μ_{CPU}	21.2	107.2	197	13.1	71.8	136.1	7.6	40.8	78.8
	μ_{GPU}	39.9	101	181.5	39.9	61.4	118	14.1	36.4	72.5
	Cohens' d	-6.4	3.5***	6.7***	-15	7.6***	11.9***	-6.2	4.2***	7.2***
M2	μ_{CPU}	12.5	48.4	115.7	7.3	32.9	80.2	4	18.4	47
	μ_{GPU}	15.5	50.2	117.6	15.5	30.5	79	5	17.7	47.1
	Cohens' d	-5.1	-2.2	-1	-15.3	3***	0.5***	-6.3	1.3***	-0.1
M3	μ_{CPU}	72.6	498.6		50.7	317.9		28.2	173.7	
	μ_{GPU}	37.3	142.2		22.2	73.2		13.3	49.5	
	Cohens' d	9.2***	1.4***		11.5***	12***		10.4***	3.6***	

6. References

[1] Bortfeldt, A. and G. Wäscher, "Constraints in container loading – A state-of-the-art review", *European Journal of Operational Research*, 229(1), 2013, pp. 1–20.

[2] Zhao, X., J.A. Bennell, T. Bektaş, and K. Dowsland, "A comparative review of 3D container loading algorithms", I*nternational Transactions in Operational Research*, 23(1-2), 2016, pp. 287–320.

[3] Bischoff, E.E. and M. Ratcliff, "Issues in the development of approaches to container loading", *Omega*, 23(4), 1995, pp. 377–390.

[4] Ramos, A.G., J.F. Oliveira, J.F. Gonçalves, and M.P. Lopes, "A container loading algorithm with static mechanical equilibrium stability constraints", *Transportation Research Part B: Methodological,* 91, 2016, pp. 565–581.

[5] Ramos, A.G. and J.F. Oliveira, "Cargo Stability in the Container Loading Problem - State-of-the-Art and Future Research Directions“, *Operational Research*, A.I.F. Vaz, J.P. Almeida, J.F. Oliveira, and A.A. Pinto, Editors. Springer International Publishing, Cham, 2018.

[6] Mazur, P.G., N.-S. Lee, and D. Schoder, "Integration of Physical Simulations in Static Stability Assessments for Pallet Loading in Air Cargo", *Proceedings of the 2020 Winter Simulation Conference*, 2020, pp. 1312-1323.

[7] Martinez-Franco, J.C. and D. Alvarez-Martinez, "PhysX as a middleware for dynamic simulations in the container loading problem", *Proceedings of the 2018 Winter Simulation Conference*, 2018, pp. 2933–2940.

[8] Ramos, A.G., J.F. Oliveira, J.F. Gonçalves, and M.P. Lopes, "Dynamic stability metrics for the container loading problem", *Transportation Research Part C: Emerging Technologies*, 60, 2015, pp. 480–497.

[9] Hennessy, J.L., D.A. Patterson, D. Goldberg, and K. Asanovic, *Computer architecture: A quantitative approach*, 5th edn., Morgan Kaufmann, Amsterdam, 2011.

[10] Harada, T., "Real-time Rigid Body Simulation using GPUs", *GPU Gems 3*, H. Nguyen, Editor. Addison-Wesley: Upper Saddle River, N.J., 2007.

[11] Peffers, K., T. Tuunanen, M.A. Rothenberger, and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research", *Journal of Management Information Systems*, 24(3), 2007, pp. 45–77.

[12] Ramos, A.G., J. Jacob, J.F. Justo, J.F. Oliveira, R. Rodrigues, and A.M. Gomes, "Cargo dynamic stability in the container loading problem - a physics simulation tool approach", *International Journal of Simulation and Process Modelling*, 12(1), 2017, pp. 29–41.

[13] Martínez, J.C., D. Cuellar, and D. Álvarez-Martínez, "Review of Dynamic Stability Metrics and a Mechanical Model Integrated with Open Source Tools for the Container Loading Problem", *Electronic Notes in Discrete Mathematics*, 69, 2018, pp. 325–332.

[14] Brandt, F. and S. Nickel, "The air cargo load planning problem - a consolidated problem definition and literature review on related problems", *European Journal of Operational Research*, 275(2), 2019, pp. 399–410.

[15] Lee, N.-S., P.G. Mazur, M. Bittner, and D. Schoder, "An Intelligent Decision Support System for Air Cargo Palletizing", *Proceedings of the 54th Hawaii International Conference on System Sciences*, 2021, pp. 1405-1414.

[16] Bracht, E.C., T.A. de Queiroz, R.C.S. Schouery, and F.K. Miyazawa, "Dynamic cargo stability in loading and transportation of containers", *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, IEEE, 2016, pp. 227-232.

[17] Nolan Goodnight, "Part VI: GPU Computing", in *GPU Gems 3,* H. Nguyen, Editor. Addison-Wesley: Upper Saddle River, N.J., 2007.

[18] Buck, I. and T. Purcell, "A Toolkit for Computation on GPUs", *GPU Gems*: *Programming techniques, tips, and tricks for real-time graphics*, R. Fernando, Editor. 2004. Addison-Wesley: Boston, Mass.

[19] Green, S., "Part IV: General-Purpose Computation on GPUS: A Primer", *GPU Gems 2*: *Programming techniques for high-performance graphics and general-purpose computation*, M. Pharr and R. Fernando, Editors. 2006. Addison-Wesley: Upper Saddle River, N.J.

[20] Vaishnavi, V., W. Kuechler, and S. Petter (Eds.), "Design Science Research in Information Systems", 2004/19. URL: http://www.desrist.org/design-research-in-information-systems/.

[21] Gregor, S. and A.R. Hevner, "Positioning and Presenting Design Science Research for Maximum Impact", *MIS Quarterly,* 37(2), 2013, pp. 337–355.

[22] Welch, B.L., "The generalisation of student's problems when several different population variances are involved", *Biometrika*, 34(1-2), 1947, pp. 28–35.

[23] NVIDIA PhysX, NVIDIA PhysX SDK 4.1 Documentation, 2021. URL: https://gameworksdocs.nvidia.com/PhysX/4.1/documentation/physxguide/Index.html

[24] Boeing, A. and T. Bräunl, "Evaluation of real-time physics simulation systems", *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia (GRAPHITE '07)*, 2007, pp. 281–288.

[25] Hummel, J., R. Wolff, T. Stein, A. Gerndt, and T. Kuhlen, "An Evaluation of Open Source Physics Engines for Use in Virtual Reality Assembly Simulations", *Advances in Visual Computing*, D. Hutchison, T. Kanade, and J. Kittler, Editors. Springer Berlin Heidelberg: Berlin, Heidelberg, 2012.

[26] Lee, N.-S., P.G. Mazur, C. Hovestadt, and D. Schoder, "Designing a State-of-The-Art Information System for Air Cargo Palletizing." *15th International Conference on Design Science Research in Information Systems and Technology, DESRIST 2020,* Springer International Publishing, 2020, pp. 382–387.

[27] Kramer, O., *Genetic Algorithm Essentials*, Springer International Publishing, Cham, 2017.