Understanding and Tackling the Hidden Memory Latency for Edge-based Heterogeneous Platform

Zhendong Wang, Zhen Wang, Cong Liu, Yang Hu
The University of Texas at Dallas

Abstract
With the burgeoning of autonomous driving, the edge-deployed integrated CPU/GPU (iGPU) platform gains significant attention from both academia and industries. NVIDIA issues a series of Jetson iGPU platforms that perform well in terms of computation capability, power consumption, and mobile size. However, these iGPU platforms typically contain very limited physical memory, which could be the bottleneck of these autonomous driving and edge computing applications. Although the introduction of the Unified Memory (UM) model in GPU programming can reduce the memory footprint, the programming legacy of the UM model initializes data on the CPU side by default as the conventional copy-and-execute model does, which causes significant latency of application execution. In this paper, we propose an enhanced unified memory management model (eUMM), which delivers a prefetch-enhanced data initialization method on the GPU side of the iGPU platform. We evaluate eUMM on the representative Jetson TX2 and Xavier AGX platforms and demonstrate that eUMM not only reduces the initialization latency significantly but also benefits the following kernel computation and the entire application execution latency.

1 Introduction
The emergence of heterogeneous SoCs has pushed the computing platforms of many edge-intelligence applications (i.e., driving automation system and drone) to shift from high-performance, energy-consuming discrete CPU and GPU-equipped platforms (dGPU) to effective, energy-efficient integrated CPU and GPU platform (iGPU). As the major game player, NVIDIA has developed a series of heterogeneous iGPU platforms, such as Jetson series and Drive AGX, for autonomous embedded systems and driving automation systems, respectively [17].

Exploiting iGPU for edge autonomous intelligent workloads processing can bring better size, weight, and power (SWaP) trade-off compared to traditional dGPU-based solutions. However, building future embedded autonomous systems based on this hardware is further stymied by the unprecedented challenges that are specifically imposed by the memory footprint and stringent latency requirements of autonomous systems, and the intrinsic hardware restrictions of iGPU-enabled heterogeneous SoCs. For example, NVIDIA TX2 possesses an on-chip memory of 8GB, which is physically shared by both CPUs and GPUs. In practice, modern autonomous systems such as driving automation and drone systems need a vast amount of perception data for decision-making guidance. A typical automated vehicle equips with 8 to 12 cameras to provide 360-degree visibility around the vehicle, with 1440 Mbs of data generated every second. If a poorly-managed memory allocation method (e.g., the traditional copy-and-execute model) is used, such a large amount of raw data can easily exhaust the memory space of iGPU-based heterogeneous platform.

Secondly, modern autonomous systems consider the processing latency as one of the most important tenets for safety and functionality. The latency from image capture to recognition completion is critical since the response time of the control operations depends on it. Failing to execute actuation in time may cause catastrophic consequences. Typically, modern autonomous driving systems adopt a variety of DNN models to achieve complete functionalities. However, the DNN functions typically involve large-scale matrix operations. Table 1 shows the matrix operations scale involved in several representative DNN models adopted by the mainstream autonomous driving solutions. We observe that the existing CPU-initiated matrix initialization can cause great latency for GPU tasks with large-scale matrix operations.

To meet these rigorous requirements of memory footprint and processing latency for the iGPU-based heterogeneous platforms, tapping into the emerging unified memory (UM)
model is considered a promising solution. Recent NVIDIA GPU architectures support UM to ease the explicit programming efforts to handle data movement and address mapping [6]. We observe that the UM is memory footprint-friendly to iGPU platform to process autonomous workloads since it replaces the explicit data copy in the traditional copy-and-execute model with implicit data initialization and addresses mapping, which significantly saves the memory footprint. However, we observe that the current UM model fails to provide efficient memory management on iGPU-based heterogeneous platforms, such as NVIDIA Jetson TX2. Our characterization shows that the inherited routine of data initialization on the CPU side provided by existing UM typically results in significant data initialization latency, which is caused by the limited computation capability of CPU, and overwhelms the execution time of the following GPU kernel. These findings suggest a potential opportunity to optimize the data initialization and thus significantly reduce the GPU-based task processing.

In this paper we propose an enhanced unified memory management model (eUMM) that delivers a prefetch-enhanced GPU-initialization method. The eUMM innovatively implements data initialization on GPU (GPU-Init.), which leverages GPU’s advantages in large-scale computation to reduce the latency of data initialization. Furthermore, the eUMM integrates prefetch technique in the process to reduce the latency caused by the data address mapping, which is brought by the inherent limits of the kernel launch in pristine UM model. We conduct a series of experiments on emerging iGPU platforms and demonstrate that eUMM can reduce the latency of initialization up to 99.4% and 97.3% for assorted benchmarks on Jetson TX2 and AGX, respectively.

2 Limits of Unified Memory Management

Data Initialization Latency. In default copy-and-execute model, application program typically initializes data on CPU side. On iGPU platform, UM model inherits the legacy programming style, though it can reduce the memory footprint. To explore how this CPU-side initialization brings the latency, we breakdown the execution time of applications for both copy-and-execute model (Def.) and unified memory model (UM), as shown in Fig. 1. Specifically, in Def. model, we divide the entire benchmark time into two parts: data initialization time (init.) and other time which covers the data migration between CPU and GPU as well as the kernel execution (others). In UM model, the benchmark time includes the init. time and others which covers the page mapping and kernel execution. Note that there is no data copy under UM for iGPU platform. We utilize two representative benchmarks, Matrix Add (MatAdd) and Matrix Multiplication (MatMul) to illustrate the init. latency and its ratio in application execution time, as is shown in Fig. 1. We repeat the benchmarking five times and report the average latency.

For MatAdd in Fig. 1a, we observe that the init. latency is non-trivial in Def. model, which can compete with the latency of others latency. With input data size increasing, the init. latency increases drastically. Intuitively, the CPU has to process larger amount of data, thus leading to greater latency. In UM model, the init. latency is significant as well, which even dominates the entire benchmark execution time. For MatMul in Fig. 1b, in either Def. model or UM model, the init. latency almost follows the same principle. Besides, we calculate the init. ratio in the entire benchmark execution time under the two models, as indicated by the dots in Fig. 1. For both benchmarks, the ratio of init. latency is around 50% in Def. model, and around 90% in UM model, indicating that the init. latency is even exacerbated under UM model. In fact, UM model on iGPU platform doesn’t require explicit copy of initialized data from CPU as Def. model does, the initialization may not be necessarily performed on CPU side.

Kernel Launch Latency. As we stated above, under UM model, such iGPU platform as Jetson TX2 doesn’t support on-demand paging as well as concurrent access from CPU and GPU when the code is running. For each kernel launch, the GPU tries to access the data that is originally populated in the CPU side. The driver has to perform address translation and page mapping such that the GPU can know the pages reside on its own space, which inevitably introduces extra latency in kernel execution.

To find the hidden latency caused by the address translation and pages mapping during the kernel launch process, we compare the kernel execution time of the two benchmarks, MatAdd and MatMul, under Def. and UM models on TX2 platform. Fig. 2a shows the result of MatAdd, where the x-axis indicates increasing input data size and the y-axis indicates kernel execution time. We can observe that the kernel latency under UM model is significantly larger than the latency under Def. model. With data size increasing, the differ-
In this section, we propose an enhanced unified memory management (eUMM) model to optimize the memory management for iGPU and reduce the latency of the time-critical workloads.

**Initializing Data on GPU Side.** As the programming legacy in the copy-and-execute model, conventional UM still implements data initialization on CPU side, which significantly contributes to the application’s latency. However, UM allows CPU and GPU to share a pointer to access the data in the allocated memory region, providing an opportunity to perform data initialization on the GPU instead of the CPU side. Furthermore, data initialization is typically a process of assigning values to variables, and thus is well structured and paralleled. GPU-Init. may gain significant benefit due to GPU’s advantages in parallel computing and further reduce the application’s latency.

Fig. 3 illustrates the design of eUMM which initializes data on GPU side. The figure on the top shows the process of data being initialized on CPU in conventional UM model: (1-2) data allocation and initialization on CPU; (3) page fault handling indicates pages covering the data allocation are unmapped from CPU page table and remapped to GPU page table; (4) kernel execution on the GPU. The figure on the bottom shows the process of eUMM: (1) data allocation on CPU; (2) page fault handling; (3-4) data initialization and kernel mapping can be removed from the kernel execution under the UM model, the real kernel execution time will be reduced. In fact, chances are that the data allocation and initialization can be well managed such that the data can be pre-mapped and populated on the GPU side, thus benefiting the kernel execution time.

![Figure 3: GPU-side data initialization in eUMM model.](image)

**3 Enhanced Unified Memory Management**

**Prefetch-enhanced GPU-Init.** Typically, the prefetch technique and NVIDIA CUDA streams [16] can be combined to benefit the kernel execution. In CUDA 8.0, the API, `cudaMemPrefetchAsync()` is provided to hint where and how data are to be used. On dGPU platform, the data pages can be speculatively prefetched across PCI-e connection with negligible opportunity cost. If the data page to be used is successfully prefetched, only translation lookaside buffer (TLB) miss is incurred upon GPU accessing the data, which can be resolved locally instead of being converted into a far page fault. Therefore, GPU kernel execution latency can be reduced.

![Figure 4: Prefetch-enhanced GPU-Init. in eUMM model.](image)

Prefetch-enhanced GPU-Init. in eUMM model. (a) pristine GPU-Init. (b) prefetch-enhanced GPU-Init. execution on GPU. As shown in the bar figures, the GPU-Init. latency can be significantly reduced compared to the latency on the CPU side due to the powerful GPU capability in parallel computation. On the other hand, upon GPU-Init., all pages covering the data will populate in GPU prior to the following kernel execution. Therefore, there is no need for GPU to wait for page faulting and mapping any more when it implements the kernel, which consequently reduces the kernel execution latency.

Prefetch-enhanced GPU-Init. Typically, the prefetch technique and NVIDIA CUDA streams [16] can be combined to benefit the kernel execution. In CUDA 8.0, the API, `cudaMemPrefetchAsync()` is provided to hint where and how data are to be used. On dGPU platform, the data pages can be speculatively prefetched across PCI-e connection with negligible opportunity cost. If the data page to be used is successfully prefetched, only translation lookaside buffer (TLB) miss is incurred upon GPU accessing the data, which can be resolved locally instead of being converted into a far page fault. Therefore, GPU kernel execution latency can be reduced.

On an iGPU platform, such as Jetson TX2, address translation and mapping are required in the UM model when a GPU kernel tries to touch the data in the allocated region for the first time due to the fact the data populates in CPU side in the beginning. As indicated in the step (2) of eUMM in Fig. 3, page mapping has to be implemented in bulk after CPU’s allocating data completes, which inevitably causes extra latency to the following GPU-Init. kernel, though GPU-Init. is typically faster than CPU initialization. Therefore, we propose to apply the prefetch technique in eUMM to further optimize the GPU-Init. performance, as shown in Fig. 4.

Fig. 4a indicates the pristine GPU-Init. case, where the GPU-Init. kernel in eUMM doesn’t execute until all mappings complete. While Fig. 4b indicates the prefetch enhanced GPU-Init. case, where prefetch combined with CUDA streams is introduced. As a result, the page mapping for a tile of initialized data can be overlapped with the GPU-Init. kernel of the data in different streams. Theoretically, the summation latency of serialized mapping and GPU-Init. can be reduced. Specifically, the 1st tile of data is prefetched for address mapping and subsequently initialized by the following `gpu_init` kernel. Then, the 2nd tile of data is prefetched and initialized. Based on our measurement, the prefetch operation can cause around 5us latency by itself. By adapting the tile numbers, we can control the total number of prefetch and further manage the overall latency caused by prefetch. Meanwhile, by adapting the grid and block size of the GPU-Init. kernel, we can
manages the latency of one-tile GPU-Init. kernel. Therefore, the pages mapping latency of (i+1)th tile data in one stream can be overlapped with the latency of the ith GPU-Init. kernel in another stream. Instead of serializing all pages mapping and GPU-Init. kernel, the overlapping of the mapping and initialization operations can effectively reduce the overall latency of the process in eUMM model further. Basically, in the ith loop, the (i+1)th tile of data is prefetched and is sequentially initialized by the kernel in the (i+1)th loop. Besides, by adapting the initialization kernel size (i.e., the grid dimension and block dimension of the kernel) and the number of prefetch, the prefetch and initialization can be well overlapped in various streams, thus reducing latency.

4 Implementation of eUMM

Effectiveness of Prefetch-enhanced GPU-Init. We implement our eUMM model on Jetson TX2 platform, which installs NVIDIA L4T 28.2.1. In addition to MatAdd and MatMul, we utilize another two applications, needle (NW), and random access (RA), to test the irregular memory access scenario. We execute all benchmarks under conventional UM model and eUMM model to demonstrate the effectiveness of prefetch-enhanced GPU-Init. under eUMM model and its improvement over the conventional UM model.

Fig. 5 shows the results, where x-axis represents the increasing input data size for each benchmark and y-axis represents data initialization latency. We can observe that for MatAdd, MatMul and RA, the data initialization latency under eUMM (i.e., prefetch-enhanced GPU-Init.) is higher than conventional UM (i.e., pristine CPU initialization) when the input data size is small (50-500). The eUMM significantly outperforms the UM when input data size exceeds the thresholds. Specifically, the initialization latency under eUMM can be reduced up to 76.2%, 99.4%, and 59.4% for MatAdd, MatMul, and NW, respectively. Intuitively, GPU has a large number of parallel cores and can be scaled up well to handle the increasing amount of data initialization requests. Furthermore, with prefetch applied, the data address mapping and GPU-Init. kernel can be overlapped to further reduce the latency. In contrast, CPU has limited cores and can be easily saturated by the increasing amount of data initialization requests. However, we also observe that there are inherent overheads for GPU-Init. mechanism. As a kernel, GPU-Init. has to be launched from the host CPU. Then it will be scheduled and dispatched to the GPU processing cores to execute. Notice that the kernel launch, schedule, and dispatch are unavoidable overheads for GPU-Init. When the input data size is small and there is not enough tiles of data mapping to overlap the GPU-Init. kernel, the benefit of data initialization acceleration could be offset.

Besides, it appears that the GPU-Init. latency under eUMM has been smaller than latency under UM for NW even in the case of small input data size. This is due to the input size of NW cannot be set as less than 16 * 16, which has exceeded the threshold. Fig. 5c indicates that the latency under eUMM is quite close to the latency under UM in 16*16 case.

In addition, we analyze the kernel latency (i.e., the kernel execution time) of all benchmarks under UM and eUMM models to demonstrate that GPU-Init. in eUMM can also benefit the execution time of following kernel. Fig. 6 shows the results, where x-axis indicates the increasing data size corresponding to the size of each benchmark in Fig. 5 and y-axis indicates the latency. Obviously, the kernel execution latency under eUMM is significantly reduced than the latency under UM, though for NW both latency increases drastically. Specifically, the kernel latency under eUMM can be reduced up to 61.0%, 63.0%, 10.1% and 76.9% for MatAdd, MatMul, NW and RA, respectively. Under eUMM model, the GPU-Init. enables much page fault handling and mapping to be processed prior to the real kernel computation. The data to be used in the kernel computation has populated in GPU locally (i.e., the page tables covering the data is set up well on GPU side) before the kernel launches. Therefore, kernel latency can be reduced as well. Therefore, eUMM not only reduces data initialization significantly but also benefits the following kernel execution as well as the entire benchmark execution latency. It’s critical to the safety of autonomous driving system.

GPU-side Initialization on Xavier AGX. We implement our eUMM model on the Xavier AGX [1] as well, which installs NVIDIA L4T 32.3.1. The result is shown in Fig. 7, where (a-d) indicates the initialization latency of the four benchmarks under UM and eUMM models, respectively. Basically, the initialization latency follows the same pattern as on TX2. The eUMM can significantly reduce the initialization latency, which can be up to 93.4%, 97.3%, and 87.4% for MatAdd, MatMul, and NW respectively, though the UM outperforms...
ware/model could be proposed to implement the initialization
of AGX. To further proof the generalization of eUMM, we will
almost all applications.

function considering that the process cannot be omitted by
utilized without being returned back to CPU. A specified hard-
up GPU faster without warming up and the data can directly
eliminate the overhead. We plan to mitigate the inherent over-
tions (e.g., fine-tiled prefetch, scheduling) cannot thoroughly
is small. Simply applying high-level programming optimiza-
the performance benefits of GPU-Init. when input data size
scheduling and dispatching. Such overheads can even offset
overheads are caused by the GPU kernels scheduling and dispatching. Such overheads can even offset
in millisecond and x-axis indicates increasing data size.

5 Prospects and Challenges

Extending eUMM to a Broad Spectrum of Workloads. Our current characterizations and design only cover repre-
sentative GPU benchmarks, MatAdd, MatMul, and part of
the Rodinia benchmarks [5]. We will continue to explore if
the acceleration opportunity also apply to a broader spectrum
of edge-based workloads by leveraging the GPU initialization
and prefetch, such as the full Rodinia benchmark suite and
PolyBench [8], which exhibits diverse computation and memory access patterns. Also, we will extend eUMM to ac-
commodate emerging autonomous driving workloads, such
as YOLO for objective detection, and GoTurn [11] for mobile
objectives tracking. It’s anticipated that eUMM can reduce
the latency of these DNN models execution considering that
the MatAdd and MatMul are the basic operations in DNN
models. We plan to provide transparent API, which can ease
users to apply eUMM in practice.

Generalizing eUMM to Other Platforms. Our current im-
plementations of eUMM are based on Jetson TX2 and Xavier
AGX. To further proof the generalization of eUMM, we will
also validate eUMM on high-end edge heterogeneous plat-
forms such as PX2 and AGX Drive [2]. Also, we expect
eUMM can apply to discrete GPU platforms to achieve a
broader impact to datacenter computing.

Reduce the Inherent Overheads of GPU-Init. In our pre-
liminary investigation, we observe that the GPU-Init. in-
evitably incurs overheads that are caused by the GPU kernels
scheduling and dispatching. Such overheads can even offset
the performance benefits of GPU-Init. when input data size
is small. Simply applying high-level programming optimiza-
tions (e.g., fine-tiled prefetch, scheduling) cannot thoroughly
eliminate the overhead. We plan to mitigate the inherent over-
head of GPU-Init. in eUMM through low-level design and
optimization. For example, GPU persistent model can start
up GPU faster without warming up and the data can directly
utilized without being returned back to CPU. A specified hard-
ware/model could be proposed to implement the initialization
function considering that the process cannot be omitted by
almost all applications.

6 Related work

UM model and iGPU Platform for Autonomous Applications. UM model provides an illusion of CPU-GPU unified
virtual memory to avoid explicit data copy and ease memory
management. NVIDIA’s UM model is introduced in CUDA
and enhanced in CUDA 8 to support on-demand paging and migration mechanism [7, 10]. However, on-demand
paging is only supported on dGPU [23], which hurts its ben-
efit. On the other hand, NVIDIA proposes its Jetson line
of iGPU platforms targeting at autonomous systems, which
mainly includes a series of Tegra SoCs, such as Parker TX2
and Xavier AGX. Actually, much effort has been dedicated
to the UM model performance on iGPU platform. [18] sum-
marizes the main characteristics of Def. and UM models on
Tegra platforms. [13] measures the performance loss of UM
in CUDA on both iGPU and dGPU platforms and explores
the underlying reasons. [6, 20] compare applications perfor-
ance under default and UM models on TK1 platform. [3]
comprehensively compares applications performance under
three memory management models and co-optimize memory
footprint and performance on Jetson iGPU platforms. Even
so, these works don’t deep excavate the latency hidden in
UM model, especially the significant initialization latency
and mostly optimize the latency in existing UM model.

Prefetch in GPU Programming. [12] demonstrate that
prefetch may degrade the application’s performance if being
not judiciously used. [22] designs an adaptive GPU prefetch
technique to dynamically select the appropriate time step to
achieve timely prefetch. [19] proposes the warp-aware selec-
tive prefetch to monitor the progress of the warps. Besides,
several other works are dedicated to the hardware prefetcher
under UM model. [24] classifies prefetchers into different

types based on different selection ways. [7] analyzes the
relationship between hardware prefetcher and page eviction
policy. However, these works mainly utilize prefetch to opti-
imize the data migration overhead on dGPU platforms and don’t
apply the technique to optimize the hidden latency under UM
model, especially on iGPU platforms.

7 Conclusion

In this work, we explore the hidden latency issues when the
UM model is applied to the emerging edge-based iGPU plat-
form for autonomous driving. Through comprehensive charac-
terizations, we observe that the existing CPU-side data initia-
ization mechanism incurs significant latency under the conven-
tional UM model and can hurt the response time of real-time
tasks on iGPU platforms. We propose eUMM model, which
delivers a prefetch-enhanced GPU-Init. method, to signifi-
cantly mitigate the latency and benefit the entire application
execution time.

Acknowledgment This work is supported in part by
NSF grants CNS 1527727, CNS 1750263 (CAREER), CCF
1822985, and CCF 1943490 (CAREER). Yang Hu is the cor-
responding author.
Discussion Topics

Desired feedback. We are looking forward to hearing more opinions from edge computing and GPU experts whether the GPU-Init. delivered by eUMM is a promising method for GPU programming and applications. Since we are not aware of similar work in this area, we would like to learn from experts and peers’ comments about whether eUMM seems to be pervasive to GPU computing applications. We also appreciate it if any advice is provided to pinpoint the key challenges that we may face in developing the full framework for eUMM. We also welcome the experts’ comments from the autonomous driving area and would like to learn their willingness and doubts to deploy eUMM in real practice.

Open issues. Firstly, we have observed that the benefits of GPU utilization may diminish due to the increasing complexity of memory access patterns of workloads. Therefore, we have to comprehensively evaluate the performance gains of eUMM for a variety of workloads in practice. Based on these characterizations, we may determine if it is beneficial to further optimize the GPU initialization process or designing an adaptive memory initialization controller to help us with the decision. Concerning the complex DNN workload suites in autonomous driving applications, they typically capture sensed data in real-time besides initializing the large amount of weight data. Extending eUMM model to this scenario could be a non-trivial job. Therefore, we consider to leverage GPUDirect [15] to further enhance the eUMM model for the purpose of efficient data transfer. Secondly, when we apply eUMM in such high-end heterogeneous platforms such as PX2, which incorporates dGPUs besides iGPUs, the benefits of eUMM may be negatively impacted due to the complicated architecture. Therefore, techniques such as smart scheduling and CPU-GPU cooperative initialization can be considered on the platforms to guarantee the benefit of eUMM model. Besides, we wonder how eUMM is well compatible with other discrete GPUs or other vendors products.

Depreciating Circumstances. As we stated, we target at autonomous driving and other real-time tasks, which care about the latency on the iGPU platforms. Therefore, the UM model can be enhanced to mitigate the large data initialization latency and benefit the entire application execution time. However, if non-time-critical applications are implemented, it may be not necessarily to apply eUMM model to ease the situation.

References


