Out of RAM? Use VRAM! Linking C++ binaries using swap on VRAM

Marcell Jenei[1], Kristóf Umann[1], Gábor Tóthvári[1] and Zoltán Porkoláb[1]

[1]ELTE, Eötvös Lóránd University, Budapest, Hungary. Faculty of Informatics, Department of Programming Languages and Compilers

Abstract

Compiling and linking C and C++ programs require high resources, especially a fair amount of memory. As with the rapid increase in AI adoption hardware prices, and in particular, memory prices sharply increased, developers are intensively looking for cost-conscious solutions. Various surveys report that dedicated video cards are already available in a high number of computers. These hardware components, however, are unused during the software build process. In this paper we investigate the possibility to utilize the video card memory as an additional resource during critical software development steps. We executed experimental measurements to validate the use of VRAM as a swap device. Results show that VRAM could be used with close to the efficiency of SSD hardware and significantly faster than classical HDDs.

Keywords

compiling and linking, GPU, VRAM, swapping

1. Introduction

The C programming language, designed in the early 1970s, is among the most popular programming languages in the last 40 years. It is used to implement operating systems, compilers, standard libraries of other languages, like Python, among many other applications. C programs are compiled ahead-of-time [1], a major contributing factor of its speed against other languages [2]. However, compilation is resource intensive, both in terms of memory consumption and execution time. Figure 1 shows the memory usage of Clang Tidy compilation, which is not among the larger projects in the LLVM library.

With the rapid increase in AI adoption, alongside a fierce technological race among government and non-government organizations, hardware prices, and in particular, memory prices skyrocketed [3]. This impacts individuals and small enterprises, who make up a significant portion of open-source contributors [4] especially hard, as they do not benefit from economies of scale with regards to the cost of hardware [5].

Large software companies like Microsoft [6], Google [7], Meta [8] and others [9] prefer to create a cluster of remote machines to which they distribute their build jobs. On the other hand, the needs of a private individual with regards to their personal computer is very different: it may be used for casual browsing of the internet, gaming, storing personal data, and also working on C/C++ projects. This also means, however, that a general purpose computer houses components that are usually unavailable in an industrial context, in particular, a dedicated video card *with dedicated video memory on board.*

According to a 2024 survey conducted by the Entertainment Software Association, 61% of the US population plays video games for at least one hour per week ($n = 4000$) [10]. A more recent, 2023 global study found that 33% of gamers play on personal computers ($n = 74295$) [11]. It is a reasonable assumption that the proportion of gamers, and specifically PC gamers among IT professionals is not too dissimilar. Many modern games demand dedicated graphics cards – while it suffers from some

SQAMIA 2026: Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, September 16–18, 2026, Niš, Serbia

eogfyc@inf.elte.hu (M. Jenei); szelethus@inf.elte.hu (K. Umann); eyjqmy@inf.elte.hu (G. Tóthvári); gsd@inf.elte.hu (Z. Porkoláb)

gsd.web.elte.hu (Z. Porkoláb)

0009-0009-3892-7788 (M. Jenei); 0000-0002-6679-5614 (K. Umann); 0009-0003-0749-8999 (G. Tóthvári); 0000-0001-6819-0224 (Z. Porkoláb)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-19

CEUR-WS.org/Vol-4336/short2.pdf

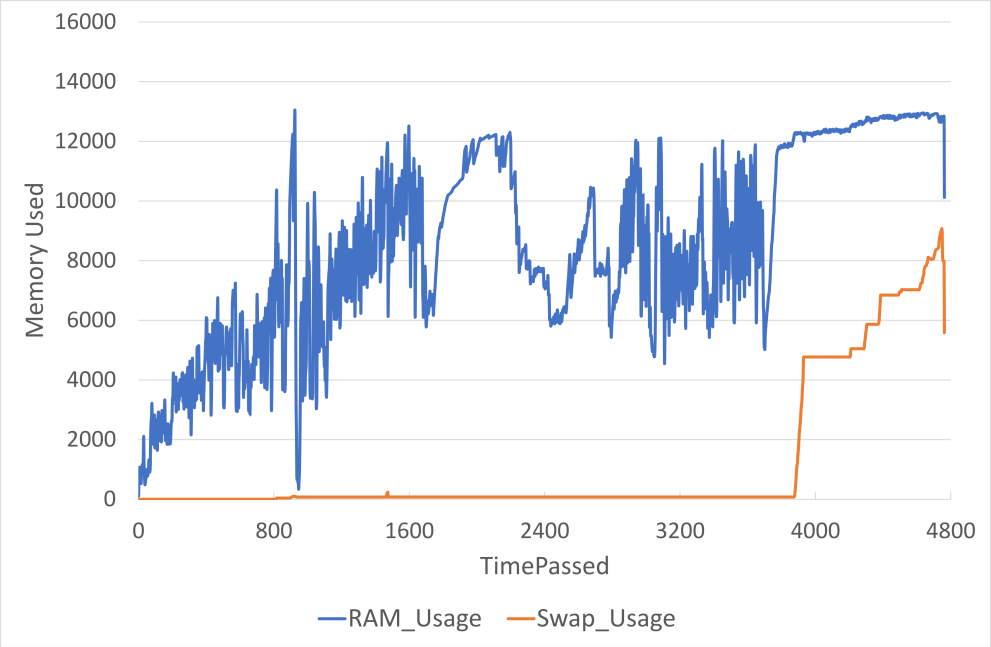


Figure 1: Full Clang Tidy debug build resource usage. Max RAM used: 12941MB, Max total used: 21899MB

reliability issues [12, 13], the May 2026 Steam Hardware Survey suggests that as much as 75% of the respondents play with dedicated graphics card [14].

So, what can a developer do when during a memory intensive compilation step, like linking or link-time optimization [15], they run out of the prohibitively expensive RAM? When they use a computer with a dedicated graphics card, they could utilize its mostly unused video memory to supplement RAM. In fact, swapping used to be a common occurrence when memory was orders of magnitude more expensive [16, 17], by moving out inactive parts of the memory to storage drives. This practice, is still used today, however, computers nowadays have sufficient RAM capacity for ordinary workloads. [18].

If the amount of available RAM is insufficient nonetheless, it is usually advised to create partitions on SSDs instead of HDDs due to their speed. However, the price of SSDs also rose sharply in recent years, and creating a swap partition reduces both the available storage, and the health of the drive.

In contrast, modern, consumer-grade video cards have around 6 GB of video memory on the lower end, and as much as 16 GB on the middle[1], of which only negligible (1-2 GB) amount is consumed by modern graphical desktop environments, and are not as limited in their lifetime. There is no benefit in *not* putting this resource to use – should its performance be satisfactory.

In this paper, we discuss how to use VRAM as a swap partition for linking C++ binaries, and measure how it performs against swap partitions on traditional storage medias such as HDDs and SSDs. Our measurements show that swap-on-VRAM is in fact faster in RAM-constrained environments while linking traditional C/C++ projects, but is slightly slower than SSDs.

This paper is structured as follows. Section 2 briefly overviews related literature, and Section 3 discusses the technical details of swapping and storage media. In Section 4, we display our measurement method, and show our results in Section 5. Our paper concludes in Section 6.

2. Related work

Most of the articles regarding swap spaces focus on algorithms that decide when pages should be swapped in or out. For example, article [19] proposes a new swapping algorithm: VSwap which

[1]At the time of writing this paper, a high-end graphics card, like the Nvidia RTX 5090, can have as much as 32 GB of video memory. However, in a computer with such an expensive component, the cost of additional RAM is not proportionally that high.

mitigates the swap memory fragmentation problem. Compared to normal swap, it improves overall performance by up to 48.18% based on their evaluations.

Another approach to improve the speed of swap is to reduce the number of ineffective I/O operations. To this end the work of Zhuohao Wang [20] implements and tests a new memory page swap mechanism named iSwap. Compared to LRU-based approach – which maintains an active and inactive list of the pages in RAM based on how recently they were accessed with the inactive pages becoming candidates for swap – reduces wasteful swap operations by 8.0%-19.2% and improves the Quality of Service of Latency Critical applications by up to 91.3%.

A different method for improvement is to bypass kernel involvement and solve swapping in user space. This was achieved by Kan Zhong [18] with the implementation of Lightswap which uses user space I/O drivers and the extended Berkeley packet filter for handling page faults without Kernel involvement. Compared to the then state-of-the-art swapping system Lightswap achieved 40% improved throughput of memcached.

There have been attempts in the past to make more use of the GPU. The paper [21] creates a Virtual Disk Driver which copies pages to the GPU memory and reads the data back from the faster GPU cache at a later time. They found the performance better than classical disk-based paging mechanisms, however the limited GPU memory at the time made it infeasible for practical uses.

Another approach to utilizing GPU proposed by Weibin Sun and Robert Ricci [22] is to augment the OS using the GPU. They focus on offloading kernel tasks to the GPU when the parallelism and faster compute speed leads to reduced latency. Their approach led to significant speedup in encryption and decryption which could lead to security enhancements for the OS.

Going in the other direction system memory can also be used to enhance the GPU. The paper [23] introduces GPUVM a new GPU memory management system focusing on programmability and performance. GPUVM allows the GPU directly manage memory and eliminate CPU involvement. They found that GPUVM leads to significant performance improvements while not requiring direct management of memory transfers from the user.

3. Technical background

If a process allocates then accesses a virtual address space when there is no more RAM available, the operating system may move one or more pages of RAM to a location known as the swap area. Afterwards, the newly allocated address space can be mapped to a page in RAM. Access to a virtual address, whose contents are located in the swap space will again trigger this mechanism.[2] A region of memory might be moved to swap space and the requested page of memory will brought in to RAM. Naturally, moving the memory to a different backing storage has a significant cost, and access from the backing storage is slower compared to accessing memory in RAM. On the other hand, it enables programs to keep running when the amount of actually used (virtual) memory is larger than the physical RAM capacity.

In most cases a Hard Disk Drive or a Solid State Drive is used for the swap area, either in the form of swap files - specially created files that are marked and ready to be used for swapping - or a swap partition. The main concerns, when choosing a storage device to be used as swap space are write speed and latency. SSDs have approximately 2-10 times higher write speed and 10-100 times lower latency compared to HDDs.

4. Measurement method

In this paper, we are investigating the possibilities of using the memory of GPUs (VRAM) as a replacement for the classical HDD or SSD based swap solutions.

[2] As an optimization, the evicted page may not have to be physically copied if it has not been written since the time it was brought in. This is typically indicated by a flag called the dirty bit.

To enforce swap usage during measurements we have artificially limited the amount of available physical RAM. We set this limit using Docker. We focused our measurement efforts on the most memory intensive part of the compiling process: linking the final binaries. This reduced the time of a single test run, while increasing the relative use time of swap space. Shorter test runs also enable more measurements to be taken thus improving the accuracy of the results.

During each test run log files are used to capture the important resource metrics of the Docker Instance. For the purpose of this research, the following metrics are considered important: CPU usage percentage, utilized RAM and SWAP memory. This allowed the comparison of compilation speeds, as well as the behavior of the different storage devices used for swapping.

We used a script to perform the measurements to prevent unintended differences between tests. The script for starting a measurement cycle has 4 input parameters: the name of a folder where the log files can be stored, the number of compilations to be done, the amount of RAM memory and swap memory that the Docker may use. Once started, the script creates the desired folder, then starts a loop of the following steps for each different swap space to be measured:

1. Enable the prepared swap location
2. Creating a Docker Instance with the specified resource limitations
3. Starting a compilation process with the current system configuration, and creating log files for it as described above into numbered files
4. Deleting the Docker Instance
5. Disabling the swap space

When the desired number of test runs have been completed the script terminates and the contents of the folder can be used to analyze the data. Note, that the script needs to be configured with the location of the Docker image, the command to start the compilation process, reset the compilation environment if needed, and with commands for the different swap locations to be measured. The swap locations should be prepared in advance but it can also be done by the script.

To analyze the data, the length of the compilation process is calculated from the timestamps for each compilation on a given swap location and collected into one file where statistics are calculated. When this is done for all tested swap locations the data is further aggregated into a single file for ease of comparison.

4.1. Mapping swap to GPU

In order to use GPU as swap space the first step is to get access to the memory of the GPU. We used the VRAMFS [3] open source project for this purpose. VRAMFS creates a FUSE based file system in the GPU memory and allows it to be accessed from the location selected during its execution. In order to prevent VRAMFS itself from being swapped out and thus causing the file system to be lost VRAMFS needs to be run with superuser rights. It is important to leave enough memory for the GPU to maintain stable performance to avoid the stuttering and freezing of any graphical interface. In practice, it is advised to keep at least 2-3 GBs of GPU memory free. After creating the file system it can be used for any purpose, but as it is lost upon restart and limits the usability of the GPU its use case is limited. However, these downsides are not issues when using the memory as temporary swap space for intensive tasks, such as compiling/linking C/C++ projects.

Once the file system is created a portion (or all) of it can be cleaned and used to create a swap file. Trying to use the newly created swap file as swap for the system will throw the following error:

```
swapon failed: Invalid argument
```

This is caused, because the file system created is FUSE based which is not supported as swap by default, and as such a workaround is needed. The workaround is to use a Loop device to access the newly created swap file making it appear as a normal virtual memory block. Then it is possible to use it as swap for the system. Introducing this middle layer creates its own problem which will be discussed

[3] https://github.com/Overv/vramfs

in detail in section 5. The exact steps, and commands for the entire process past the installation of VRAMFS can be found in the GitHub repository. An alternative solution for accessing the GPU memory is to use the VRAM Block Device project [4]. This method uses an NBD server to allow access to the GPU memory and is slightly more stable but has a larger overhead.

5. Results

We have chosen the linking of Clang Tidy static analyzer as the measurement task. As described above the static linking of the final binary is the most intensive task when building C/C++ projects. This linking process uses approximately 2GB of RAM and swap combined. During testing, the Docker instance was limited to using only 850MB of System RAM and had up to 3GB of swap. The version of Clang Tidy used is from the LLVM library configured for a release build with unlimited threads for use by CMake. The `ld` linker was used with Ninja. On average, using the GPU memory for swap with VRAMFS resulted in approximately 2.5% decrease in linking time compared to a normal HDD swap space. At the same time, the VRAM-based solution being comparable to the SSD version. The detailed test results can be found in Table 1.

Table 1
Comparative summary of linking time. We highlight the lower number in bold, indicating faster linking time.

Metric	VRAMFS vs HDD			VRAMFS vs SSD			HDD vs SSD		
(seconds)	VRAMFS	HDD	%diff	VRAMFS	SSD	%diff	HDD	SSD	%diff
Mean Average	**73.65**	75.48	+2.5%	73.65	**73.14**	-0.7%	75.48	**73.14**	-3.1%
Standard Deviation	**4.34**	5.67	+30.8%	**4.34**	5.34	+23.3%	5.67	**5.34**	-5.8%
Median (50th Percentile)	**73.00**	75.00	+2.7%	**73.00**	**73.00**	+0.0%	75.00	**73.00**	-2.7%
99th Percentile	**86.01**	86.03	+0.0%	86.01	**84.05**	-2.3%	86.03	**84.05**	-2.3%

The values in Table 1 are the results of 100 test runs for each swap location. We evaluated whether these results are statistically significant. The P values from Paired T test calculations are the following:

1. VRAMFS vs HDD: $p = 0.003518$
2. VRAMFS vs SSD: $p = 0.4983$
3. SSD vs HDD: $p = 0.005843$

As we can see, the VRAM versus HDD and the HDD versus SSD samples are statistically significant, while the VRAM versus SSD is not. This indicates that in our measurement scenario, linking on VRAM is comparable with linking on SSDs.

Some of the test runs running on GPU swap space showed an interesting occurrence, where the OOM killer was invoked and caused the linking process to fail despite less than half the available memory being used. When testing HDD swap with a Loop device as a middle layer the same occurrence could be observed. Unfortunately this is not easily reproducible and as such finding the exact source of the issue is not a feasible task. During tests when the system RAM was not limited to unnaturally low amounts we never encountered this issue.

Overall, with today's video cards being widely available among individuals and smaller companies, our experiment verifies that using GPU memory as extra swap space could prove useful. As high-end GPUs can have up to 24GB of memory available and most of it being idle during the compilation process of larger projects can make the entire process smoother. As the process of setting up and using VRAMFS is well documented and not difficult anyone with a GPU who runs into memory issues during compilation could benefit from trying out VRAM as swap.

[4] https://github.com/theblazehen/vramblk

6. Conclusion

Due to the increasing demands of AI training, hardware costs have seen a sharp increase, especially memory. This effects private individuals or small-scale enterprises, whose workflow includes potentially memory intensive tasks, such as compiling large C/C++ projects. Therefore developers are intensively looking for cost-conscious solutions. As GPU cards with sufficiently large memory included are present on many computers, we suggest their utilization during compilation and linking. We have investigated the possibility to use video card memory as a swap space. We executed experimental measurements to validate the use of VRAM as a swap device. Results show that VRAM could be used with close to the efficiency of SSD hardware and significantly faster than classical HDDs. This solution can save money for developers and supports sustainable working environments.

Declaration on Generative AI

We used ChatGPT and Gemini for the purposes of Content Enhancement as defined in the CEUR GenAI Usage Taxonomy, primarily to gather literature and generate the programs used in evaluating our measurements. We reviewed and edited the tools' output as needed and take full responsibility for the publication's content. We did not use any generative AI for writing actual text of this paper.

Acknowledgment

Project no. C2314106 has been implemented with the support provided by the Ministry of Culture and Innovation of Hungary from the National Research, Development and Innovation Fund, financed under the KDP-2023 funding scheme. Nemzeti Kutatási, Fejlesztési és Innovációs Hivatal

References

[1] C. Pichler, P. Li, R. Schatz, H. Mössenböck, Hybrid execution: Combining ahead-of-time and just-in-time compilation, in: Proceedings of the 15th ACM SIGPLAN International Workshop on Virtual Machines and Intermediate Languages, VMIL '23, ACM, 2023, p. 39–49. URL: http://dx.doi.org/10.1145/3623507.3623554. doi:10.1145/3623507.3623554.

[2] L. Prechelt, An empirical comparison of seven programming languages, Computer 33 (2000) 23–29. URL: http://dx.doi.org/10.1109/2.876288. doi:10.1109/2.876288.

[3] j. abdullaev, The hungry intelligence: How ai cannibalized the global memory market and priced out the consumer world (2022-2026) (2026). URL: http://dx.doi.org/10.2139/ssrn.6395479. doi:10.2139/ssrn.6395479.

[4] Y. Zhang, M. Qin, K.-J. Stol, M. Zhou, H. Liu, How are paid and volunteer open source developers different? a study of the rust project, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, ICSE '24, ACM, 2024, p. 1–13. URL: http://dx.doi.org/10.1145/3597503.3639197. doi:10.1145/3597503.3639197.

[5] C. Comstock, Z. Jiang, J. Davies, Economies and diseconomies of scale in software development, Journal of Software Maintenance and Evolution: Research and Practice 23 (2011) 533–548. URL: http://dx.doi.org/10.1002/smr.526. doi:10.1002/smr.526.

[6] H. Esfahani, J. Fietz, Q. Ke, A. Kolomiets, E. Lan, E. Mavrinac, W. Schulte, N. Sanches, S. Kandula, Cloudbuild: Microsoft's distributed and caching build service, in: Proceedings of the 38th International Conference on Software Engineering Companion, ICSE '16, ACM, 2016, p. 11–20. URL: http://dx.doi.org/10.1145/2889160.2889222. doi:10.1145/2889160.2889222.

[7] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, J. Wilkes, Large-scale cluster management at google with borg, in: Proceedings of the Tenth European Conference on Computer

Systems, EuroSys '15, ACM, 2015, p. 1–17. URL: http://dx.doi.org/10.1145/2741948.2741964. doi:10.1145/2741948.2741964.

[8] Meta Engineering, Build faster with buck2: Our open source build system, 2023. URL: https://engineering.fb.com/2023/04/06/open-source/buck2-open-source-large-scale-build-system/.

[9] G. Lim, M. Lee, R. Lahaye, Y. I. Eom, Distributed compilation system for high-speed software build processes, in: 2014 International Conference on Big Data and Smart Computing (BIGCOMP), IEEE, 2014, p. 116–120. URL: http://dx.doi.org/10.1109/bigcomp.2014.6741419. doi:10.1109/bigcomp.2014.6741419.

[10] Entertainment Software Association, 2024 Essential Facts About the U.S. Video Game Industry, Technical Report, Entertainment Software Association, 2024. URL: https://www.theesa.com/resources/essential-facts-about-the-us-video-game-industry/2024-data/.

[11] Newzoo, Newzoo's Global Gamer Study 2023, Technical Report, Newzoo, 2023. URL: https://gameindustrylibrary.com/documents/how-consumers-engage-with-video-games-today-2023.

[12] P. Alcorn, Valve fixes the steam hardware survey, Tom's Hardware (2018). URL: https://www.tomshardware.com/news/steam-hardware-survey-cpu-gpu,37007.html, accessed: 2026-06-25.

[13] J. Kinghorn, Valve owns up to inaccurate steam hardware survey findings, saying 'VRAM' on some graphics cards was not reported correctly, PC Gamer (2026). URL: https://www.pcgamer.com/hardware/valve-owns-up-to-inaccurate-steam-hardware-survey-findings-saying-vram-on-some-graphics-cards-was-not-reported-correctly/, accessed: 2026-06-25.

[14] Valve Corporation, Steam hardware & software survey, 2026. URL: https://store.steampowered.com/hwsurvey, accessed: 2026-06-25.

[15] T. Johnson, M. Amini, X. David Li, Thinlto: Scalable and incremental lto, in: 2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), IEEE, 2017, p. 111–121. URL: http://dx.doi.org/10.1109/cgo.2017.7863733. doi:10.1109/cgo.2017.7863733.

[16] P. J. Denning, Thrashing: its causes and prevention, in: Proceedings of the December 9-11, 1968, fall joint computer conference, part I on - AFIPS '68 (Fall, part I), AFIPS '68 (Fall, part I), ACM Press, 1968, p. 915. URL: http://dx.doi.org/10.1145/1476589.1476705. doi:10.1145/1476589.1476705.

[17] P. J. Denning, Virtual memory, ACM Computing Surveys 2 (1970) 153–189. URL: http://dx.doi.org/10.1145/356571.356573. doi:10.1145/356571.356573.

[18] K. Zhong, W. Cui, X. Chen, Q. Li, Z. Yang, Y. Lu, X. Yan, S. Luo, Q. Yuan, K. Huang, Revisiting swapping in user-space with lightweight threading, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 42 (2023) 4205–4218. URL: http://dx.doi.org/10.1109/tcad.2023.3274953. doi:10.1109/tcad.2023.3274953.

[19] G. Moon, D. Kang, Vswap: A new extension to the swap mechanism for enabling swap memory space optimization, Applied Sciences 15 (2025) 12049. URL: http://dx.doi.org/10.3390/app152212049. doi:10.3390/app152212049.

[20] Z. Wang, L. Liu, L. Xiao, iswap: A new memory page swap mechanism for reducing ineffective i/o operations in cloud environments, ACM Transactions on Architecture and Code Optimization 21 (2024) 1–24. URL: http://dx.doi.org/10.1145/3653302. doi:10.1145/3653302.

[21] F. Feinbube, P. Troger, J. Henning, A. Polze, Leveraging hybrid hardware in new ways - the gpu paging cache, in: 2013 International Conference on Parallel and Distributed Systems, IEEE, 2013, p. 291–298. URL: http://dx.doi.org/10.1109/icpads.2013.49. doi:10.1109/icpads.2013.49.

[22] W. Sun, R. Ricci, Augmenting operating systems with the gpu, 2013. URL: https://arxiv.org/abs/1305.3345. arXiv:1305.3345.

[23] N. Nazaraliyev, E. Sadredini, N. Abu-Ghazaleh, Gpuvm: Gpu-driven unified virtual memory, 2024. URL: https://arxiv.org/abs/2411.05309. arXiv:2411.05309.