Journal of Theoretical and Applied Information Technology
15th July 2025. Vol.103. No.13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

PERFORMANCE CHARACTERIZATION OF CACHE REPLACEMENT STRATEGIES FOR MANAGING SHARED LLC OVER HETEROGENEOUS INTEGRATED CPU-GPU ARCHITECTURE USING MULTI-THREADED WORKLOAD

PRATAP KESHARI PANDA[1], BANCHHANIDHI DASH[2*], PRASANT KUMAR PATTNAIK[3*]

[123]Kalinga Institute of Industrial Technology, India

E-mail: [1]pratap.panda@gmail.com, [2]banchhanidhi.dashfcs@kiit.ac.in, [3]patnaikprasantfcs@kiit.ac.in

*Corresponding author

ABSTRACT

The CPU is mostly designed for sequential and complex decision-making. it propagates with its instruction processing in a few pipeline stages, including steps for data and instruction fetching from memory. With time, the GPU, with its parallel processing capabilities, came into existence to assist the CPU, mainly for graphics processing. Because of parallel processing and a huge number of thread executions, its data access pattern is quite different from that of CPUs. In its rendering pipelines, the GPU may access varied data streams. The access may vary in semantics in different access patterns. Due to commercial requirements and to minimize data sharing latency between CPU and GPU, integrated processor designs are emerging. these designs accommodate both CPU and GPU in a single chip, sharing common resources. On many occasions, sharing these resources becomes a bottleneck for the whole system, deteriorating overall performance. Last Level Cache sharing among CPU and GPU is a bigger challenge in this design approach. A suitable cache eviction strategy is highly essential to utilize the shared LLC space effectively. Optimized cache with better clock speed can also be considered along with a higher configuration for CPU and GPU to improve the overall performance. Here in our work, we have designed an integrated heterogeneous CPU-GPU model with upgraded configurations to boost performance. Further, we have compared the performance with Alder Lake and Raptor Lake processors, taking them as baselines. We have achieved a speedup improvement of 59.8% and 22.12% compared to Alder Lake and Raptor Lake processors, respectively, taking the geometric mean of eight different sets of workloads. Eight different cache replacement schemes have been configured in MacSim simulator, and the workload from different benchmark suites has been given. Through simulation results, we found an average read miss count improvement of 29.41% and 15.38% over Alder Lake and Raptor Lake, respectively. This may help to IT infrastructure in terms of actuarial science, real time systems etc

Keywords: *Multi-Core, Graphics Processing Unit, Multi-Threading, Benchmark, Cache Replacement Policy, Shared Last Level Cache, Heterogeneous Architecture*

1. INTRODUCTION

The CPU is the primary processor, and the GPU has evolved to assist it. This configuration may need by the researcher or industry professional for financial modelling, AI and 3D systems, and for virtualization task scheduling in cloud computing environment. The CPU and GPU connect to the same motherboard, and the CPU shares its load with the GPU using the PCIe interconnect. This design has some data transfer latency associated. But now, gradually, the trend is changing. Manufacturers are trying to minimize the distance between the CPU and the GPU. They are trying to reduce the data transfer latency. Hence, a few of the new generation microprocessor chips are accommodating the CPU as well as its accelerator inside a single die. They coexist and live inside the same apartment and share some common system resources like DRAM, interconnect network (Known as NOC, abbreviated

to Network-on-Chip), and last-level cache [1]. Both CPU and GPU rush in the race to acquire and use these shared resources. The high level of thread parallelism in GPUs makes it more powerful to use these shared resources, mainly the Last-Level-Cache (LLC). As this cache is shared, we denote it as Shared LLC. GPU spawns thousands of threads in parallel to process data, makes very frequent calls to Shared LLC, and almost keeps it captured. In the race to use Shared LLC, mostly GPU wins. Most of the data requests for the CPU to the Shared LLC misses. This generates a high contention. Shared LLC has to be managed well to facilitate the smooth work of both CPU and GPU. Many manufacturers are now researching and focusing heavily on CPU-GPU integrated chip designs. Different generations of Intel's iGPU (Integrated GPU) [2], with their varied architectures, are currently quite popular in the market. iGPU is the integrated GPU that resides alongside the CPU in the same chip. AMD has also released many heterogeneous integrated chips, terming them APU (Accelerated Processing Unit), which was previously known as Fusion. They have released several generations of APUs thus far. Llano, Trinity, and Kaveri are among these different generations. The CPU and GPU residing inside the chip are tightly coupled and share common resources, including LLC. The CPU processes instructions sequentially in a pipeline with limited threads, making very few memory access calls. In contrast, the GPU spawns a vast number of threads to achieve faster and more parallel execution. These threads can be executed within a warp. Although each warp can execute a single instruction set, each thread makes a significant number of memory access calls. They can access different streams of data in various access patterns. In the case of 3D rendering, it accesses different data streams primarily for vertices, vertex indices, and depth buffers. As the GPU continuously accesses the LLC, it often hijacks the entire LLC. Ultimately, the overall performance of the system suffers. Not only CPU performance but also GPU performance, like 3D rendering, deteriorates to a great extent. Therefore, the LLC should be distributed optimally to maximize the CPU and GPU performance. The configuration of CPU and GPU, along with caches and mainly LLC, can be enhanced and optimized to get a better performance output. In the enhanced configurations, tuning the caches with an optimal replacement algorithm can give a better result. In studying this further, in this paper, we have configured an integrated heterogeneous CPU-GPU model and compared some of the parameters with 2 baseline models Alder Lake and Raptor Lake processors, to achieve some significant performance improvement. More ever it has been observed that this type integration causes software bugs in graphic solutions, speculative executions in real time system and miss-configuration in cloud virtual system.The programmer should plan carefully while using the system.

2. LLC MANAGEMENT POLICIES EXPLORED

Many LLC management policies have been explored and identified. We have gone through many cache management policies that mainly fall under partition-based or insertion-based techniques. This section contains a few policies related to LLC management implemented on CPU, GPU, and heterogeneous integrated chips. These policies have greatly motivated my work.

2.1 UCP

Different applications have different levels of cache sensitivity and cache usage. Many of the applications may not need the whole shared cache. In a 16-way shared cache, an application may survive only with 8 ways of the cache. Hence, assigning only 8 ways of the shared cache to the application would be sufficient. Increasing the number of ways may not decrease the number of cache misses or may not improve its performance. UCP (Utility-based Cache Partitioning) [3] policy partitions the cache depending upon the utility of the cache by any particular application. Different applications run on different cores in the CPU. As per the policy, a UMON (Utility Monitor) circuit is been assigned to each core. UMON count equals the number of cores in the CPU. This UMON circuit monitors and records the cache way utility information for its attached core, and ultimately, that shows the cache way utility information for the application running on that core. Based on the information collected by the UMON circuit, the cache partitioning decision is made by the partitioning algorithm. Though the UMON circuit eats up some space from the baseline cache of the core, the amount is quite low and can be ignored.

2.2 TAP

Because of a very high level of TLP (thread-level parallelism) of the GPU, the memory access latency cannot affect the performance of the GPGPU applications. TAP (TLP-Aware cache management Policy) [4] uses core sampling and cache block lifetime normalization as 2 components to decide on cache management in CPU-GPU heterogeneous architecture. Assuming the GPGPU applications behave in the same way, this mechanism assigns different policies to each core and takes samples of a specific period. The CSC (Core Sampling Controller) collects the performance table, compares it, and finally decides on appropriate cache management. That could be a cache insertion or a cache partition. The policy determined by the CSC will be taken off by all the cores. The same sampling process goes on at regular intervals. Little space overhead is there. However, this policy is extended to TAP-UCP and TAP-RRIP, who already have their monitoring mechanism in-built. Hence, the overhead can be ignored. There is a huge difference in cache access by CPU and GPU. With its very high number of threads, the GPU keeps on accessing the cache very frequently. However, the CPU cannot make that many calls to cache because of its limited number of threads. This results in a cache capture by the GPU. To overcome this, this policy periodically calculates the ratio at which the applications are accessing the cache. One threshold value is decided. If the access ratio exceeds the threshold value, GPU memory request calls cannot evict CPU cache blocks.

2.3 LSP

Few policies argue that latency-sensitivity is more vital than utility. Latency-insensitive applications may give way to latency-sensitive applications for off-chip accesses. This can increase cache hits for latency-sensitive applications and ultimately increase overall performance. LSP (Latency Sensitivity-based cache Partitioning) [5] considers cache access latency as the vital parameter. This policy periodically takes samples in a particular interval and takes different performance metrics using system event counters. Using the information gathered, it evaluates a cost function. At the next interval, based on the cost function result, the partitioning decision is made. A few other parameters that are considered for this decision are the latency to access memory and last-level cache.

2.4 HeLM

Throttling LLC access for GPU is another approach taken up by a few policies. HeLM (Heterogeneous LLC Management) [6] uses this technique to bypass some of the memory calls by the GPU. This policy uses the latency tolerance behaviour of the GPU to steal cache for the cache-sensitive CPU without impacting the GPU's performance. This selective memory bypassing decision is mainly based on 2 conditions. Firstly, if that GPU core has enough thread-level parallelism to tolerate memory access delays. Secondly, if the application running on it is mostly getting streaming data. Hardware performance monitor measures the wavefront count that is ready to be scheduled at any particular time. This says something about the TLP availability. To know about the cache sensitivity, individual methods are used for CPU and GPU. In the case of the GPU, 2 thresholds are identified, one low and one high. The high threshold bypasses more LLC calls than the low threshold. During sampling, core1 is assigned the low threshold while core2 is assigned the high threshold. If the performance difference between core1 and core2 is more than a predetermined threshold value, it can be concluded that the GPU is cache-sensitive.

2.5 WAP

Prefetching is a technique that enables SM to fetch the data required from memory to cache in advance. This improves memory access performance as well as increases the cache hit rate. Mostly in GPGPU, prefetching is a doable mechanism. To achieve this, the future data has to be predicted in advance. The thread ID has to be known to predict the next memory address that needs to be accessed. Though it improves the performance of the GPU, it may become an overhead for the memory controller. In many cases, the memory control gets busy serving continuous memory access requests. Prefetching may jam the bandwidth with extra memory calls. Hence, we can say that the size of the data and the time it takes to fetch may be crucial to improve the overall performance of the GPU. WAP (Warp feature Aware Prefetching) [7] policy takes advantage of prefetching to ease cache management. WAP attaches 3 hardware components to SM. Warp detector maintains the details of all active warps. The prefetching controller issues a prefetching signal and prefetching address if it decides to fetch data. The prefetching issuing/receiving unit issues prefetching requests and receives feedback as PC, PI, and PPC.

2.6 Buffer-Filter

Another cache management policy, Buffer-Filter [8][9], considers adding another buffer parallel to the LLC. Any data request raised by GPGPU first looks in LLC, if not found, then looks in the extra buffer. For a general data request, if the data is found in the LLC, then simply read it. If it cannot find the data block in LLC but finds it in the buffer, then it is considered a reuse, and the block gets copied to LLC and removed from the buffer. When the data is not found in both the LLC and buffer, then a memory read happens and the data gets copied to the buffer, without disturbing the LLC. Once the data is re-referenced and found in the buffer then only it gets shifted to the LLC.

3. RE-REFERENCE PREDICTION ALGORITHMS

Cache can be managed very well if the re-usability of cache blocks can be predicted. Based on re-usability, cache replacement policies can be designed. There are many traditional policies like LRU, FIFO, LFU, etc. Researchers have proposed many other replacement policies that have helped us predict when the cache lines might be referred to again. NRU (Not Recently Used) replacement policy predicts 2 re-reference values. Those are *near-immediate* with value 0. It signifies that the cache line will be accessed soon. The *distant* with value 1 means it will take a long interval to access the cache line again. In each cache block, one bit is used to store this prediction value. In case of a cache miss, it victimizes the cache block with value 1. These binary prediction values cannot be unique for each cache block. Another policy named RRIP [9][10] takes N bits to store the prediction values called Re-Reference Prediction Values (RRPV). Here again, near-immediate is 0, but distant is 2^N-1. RRIP is equivalent NRU if N is 1. The prediction value can be any number between 0 to 2^N-1. While inserting any new cache block into the cache, the RRPV assigned to it is of value long (2^N-2), so that RRIP can get some time to learn its re-reference interval value. When a cache miss happens, the highest RRPV block, considering from left, becomes victimized. In case of a cache hit, either the hit block gets promoted to near-immediate following the Hit-Priority prediction policy, or the RRPV value gets decremented by one following Frequency-Priority prediction policy. As all these values are calculated statically, this simple RRIP is commonly called as Static-RRIP (SRRIP). This policy can't be thrash-resistant if the application's working set is bigger than the cache size. But it addresses the scan-resistance problem. Hence, Bimodal RRIP (BRRIP) evolved with some minor modifications to SRRIP. In this policy, not all but most of the cache blocks are inserted with RRPV distant (2^N-1), and occasionally, cache blocks with RRPV long (2^N-2). Dynamic RRIP (DRRIP) is another variant of RRIP that dynamically decides on which RRIP policy to use among SRRIP and BRRIP based on a set-duelling mechanism. But one limitation can be seen here. The cache access pattern differs for different types of operations. Hence, using the same reference prediction logic in all the cases may not be a wise attempt. SHiP (Signature-based Hit Predictor) [11] policy tried solving this by associating a different signature to each cache access. This access segmentation improved the re-reference prediction to another level. This says that the re-reference prediction for cache access with a similar signature will be the same. To achieve this, 2 extra fields are added to each cache line. One for the signature and another for the outcome of the cache insertion. With an initial value of 0, the outcome becomes 1 if the cache line gets a re-reference. It maintains a Signature History Counter Table (SHCT) array indexed with signatures to record and monitor the behaviour of the signatures. On a cache hit, the associated SHCT value indexed with that particular

signature gets incremented. On a cache line eviction, if it is never re-referenced since the time of insertion, the associated SHCT value indexed with that particular signature gets decremented. A 0 SHCT value indexed with signature indicates *distant* for that signature, whereas a positive number indicates reuse of the cache lines for that signature and considers an *intermediate* interval value. Though it cannot predict the exact re-referential interval, it can say if the cache line can be re-referenced. This interval prediction happens based on the SHCT value for any signature. If we talk about the RRPV value for a read miss, it returns 3 (distant) if the corresponding SHCT value for that signature is 0, otherwise it returns 2 (intermediate). In case of a replacement, the victim selection happens the same way as SRRIP. The basic signatures for SHiP are memory region signature (SHiP-Mem), program counter signature (SHiP-PC), instruction sequence trace signature (SHiP-ISeq). In SHiP-Mem, the cache references are grouped on memory location being accessed. The first few bits from the memory locations are taken and created the signature by hashing them. In SHiP-PC, the cache references are bunched on instructions that access the said locations. PC bits are hashed to form this signature. In SHiP-ISeq, the cache references are grouped based on the instruction history for a memory access. This sequence can be represented in a binary string corresponding to the instruction decode sequence

Table 1: Configuration Comparison For Alder Lake, Raptor Lake And Configured CPU-GPU Architecture

Architecture Variant	Alder lake	Raptor lake	Configured Heterogeneous CPU-GPU
Architecture	Hybrid (P-cores and E-cores)	Hybrid (P-cores and E-cores)	Hybrid (P-cores and E-cores)
Maximum Cores	Up to 16 (8 P-cores + 8 E-cores)	Up to 24 (8 P-cores + 16 E-cores)	16 (8 P-cores + 8 E-cores)
Threads	Up to 24	Up to 32	Up to 32
P-core Microarchitecture	Golden Cove	Raptor Cove	Bonnell
E-core Microarchitecture	Gracemont	Gracemont	Tremont
L1 Cache:	• 80 KB per P-core • 96 KB per E-core	• 80 KB per P-core • 96 KB per E-core	• 128 KB per P-core • 96 KB per E-core
L2 Cache	• 2 MB per P-core • 1.25 MB per E-core	• 4 MB per P-core • 2 MB per E-core	• 8 MB per P-core • 94 MB per E-core
L3 Cache	Up to 30 MB shared	Up to 36 MB shared	64 MB shared (LLC)
P-core Base Clock	Up to 3.2 GHz	Up to 3.4 GHz	Up to 3.75 GHz
P-core Max Turbo Frequency	Up to 5.2 GHz	Up to 6.0 GHz (i9-13900KS)	Up to 5.5 GHz
E-core Base Clock	Up to 2.4 GHz	Up to 2.2 GHz	Up to 2.75 GHz
E-core Max Turbo Frequency	Up to 3.9 GHz	Up to 4.3 GHz	Up to 4.5 GHz
Execution Units	Up to 32 EUs	Up to 96 EUs	Up to 64 EUs
Max GPU Frequency	Up to 1.65 GHz	Up to 1.65 GHz	Up to 1.85

before the memory access. For heterogeneous CMPs another proposal SHiP-Hybrid [12] has also come that has extended SHiP. In this policy, it internally uses 2 different flavour of SHiP named SHiP-PC and SHiP-mem, respectively for CPU and GPU for read operations. It is similar to the older one for write operation.

4. INTEL ALDER LAKE ARCHITECTURE

Intel launched the Alder Lake processor in October 2021 as part of its 12th-generation core processors. This processor introduced a hybrid architecture consisting of 2 different types of cores. P-Cores is designed and used for high-performance tasks like gaming. E-Cores are designed and used to maintain efficiency by handling background tasks and lightweight threads. It can have a maximum of

16 cores that includes 8 P-Cores and 8 E-Cores. Intel Thread Director is a hardware-based runtime

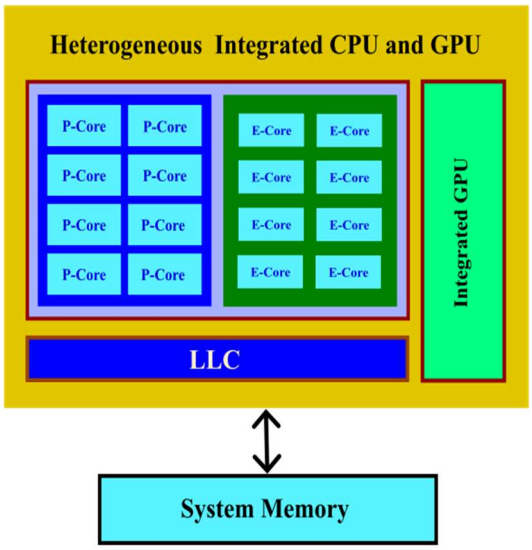

Figure 1: Block Diagram Of The Proposed Heterogeneous Integrated CPU-GPU Model

scheduling assistance that helps the OS to schedule the tasks intelligently to the appropriate cores. For configuration details, Table 1 can be referred. supports PCIe 5.0, which is double the bandwidth of PCIe 4.0. It also supports DDR5 RAM. The L2 cache size is 80 KB for P-Core and 96 KB for E-Core. L3 cache is shared among the cores with a maximum size of 30 MB. Base clock speed for P-core is up to 3.2 GHz, whereas max turbo frequency is up to 5.2 GHz. Base clock speed for E-core is up to 2.4 GHz, whereas max turbo frequency is up to 3.9 GHz. It has GPU with Intel Xe-LP architecture, which has 32 EUs with a maximum frequency of 1.65 GHz.

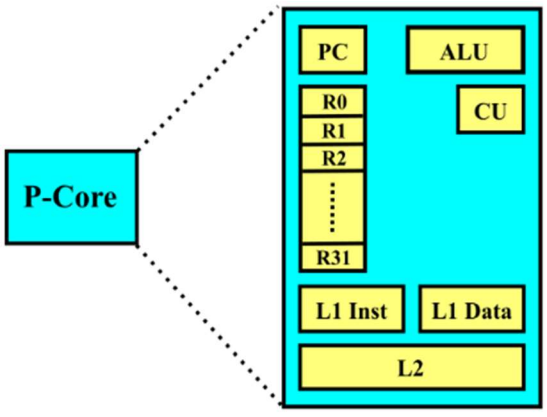

Figure 2(A): Expanded Diagram Of The P-Core Used In The Proposed Heterogeneous Integrated CPU And GPU Model

5. INTEL RAPTOR LAKE ARCHITECTURE

Intel launched the Raptor Lake processor in October 2022 as part of its 13th-generation core processor family. It is the refined version of the Alder Lake processor. Same as Alder Lake, this processor also comprises P-Cores and E-Cores. P-cores are based on Raptor Cove architecture, which is designed for high single-threaded performance and latency-sensitive tasks. This architecture is the advanced version of Alder Lake's Golden Cove architecture. E-cores are based on Gracemont architecture, designed for parallel, background, and multi-threaded tasks. E-core goes up to 24 cores and 32 threads. Higher L2 and L3 cache sizes improve gaming and multi-threading application performance and reduce latency. For more configuration details, Table 1 can be referred.

6. CONFIGURED INTEGRATED HETEROGENEOUS CPU-GPU ARCHITECTURE

Among Intel's iGPU processors, we chose to study the processors Alder Lake and Raptor Lake. Their architecture accumulates 2 different types of cores and takes advantage of their performance capabilities. Performance cores (P-Core) are for higher performance and Execution cores (E-Core) are to achieve efficiency improvements. This hybrid architecture dynamically decides on en-routing the workloads to P-Core or E-Core depending upon the nature of the workload. Using these 2 processors as a baseline, we have configured a heterogeneous CPU-GPU architectural model. Figure 1 shows a detailed block diagram of this heterogeneous architecture outlining the arrangement of P-Cores, E-Cores, Shared LLC, and Integrated GPU. This model is comprised of 8 P-Cores and 8 E-Cores. Each P-Core is responsible for performance-intensive and demanding tasks like gaming, video editing. In contrast, each E-core is responsible for efficiency-based tasks like background operations and multi-threaded executions. These tasks don't need high-performing processors. The P-cores are larger cores and optimized for high performance.

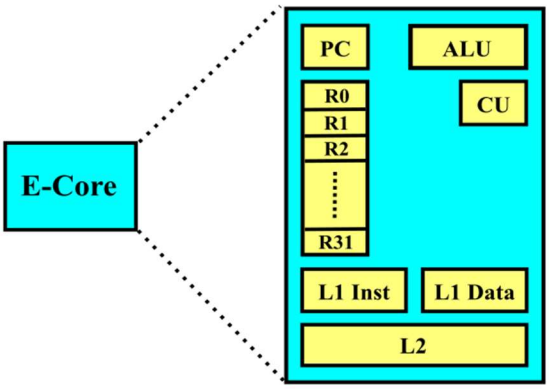

Figure 2(b): Expanded diagram of the E-Core used in the Proposed Heterogeneous Integrated CPU and GPU model

Their complex designs help in handling the tasks efficiently. Figure 2(a) describes a detailed block diagram of P-Core. P-core has a dedicated L1 (128 KB), which is further divided into instruction-cache and data-cache. It also has a dedicated L2 cache of size 8 MB. 3.75 GHz is set as the base frequency with a maximum turbo frequency of 5.5 GHz. E-Cores are quite similar in size then P-Cores and hence are less powerful than P-Cores. They only handle efficiency-based workloads, which don't require much processing power. Also, they have L1 (96 KB) and L2 (4 MB). It has a base frequency of 2.75 with a maximum turbo frequency of 4.5 GHz. Figure 2(b) shows the block diagram of E-Core. The

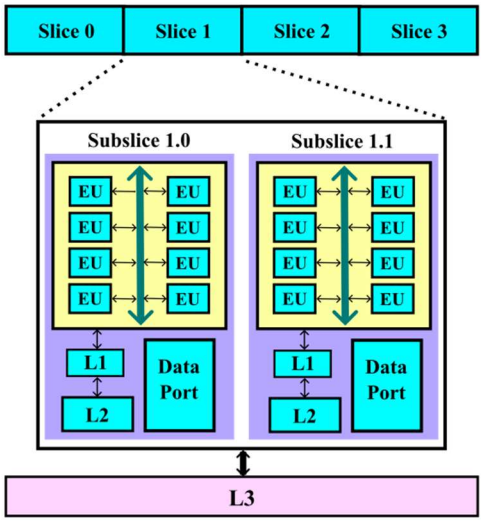

Figure 3: Block diagram Integrated GPU configuration in the Proposed Heterogeneous model

last-level cache is common for the P-Cores and E-Cores. The GPU is integrated inside the chip. This Integrated GPU is divided into 4 slices. Each slice is further divided into 2 sub-slices. The detailed block diagram of the said integrated GPU design is presented in Figure 3. In this figure, Slice 1 is divided into 2 sub-slices, Sub-slice 1.0 and Sub-slice 1.1. Each sub-slice consists of 8 Execution Units (EU). The EUs inside a sub-slice share a common L1 (64 KB) and a common L2 (2 MB). All the slices in the GPU share the common L3 cache of size 64 MB. Table 1 can be referred for configuration details.

7. HARDWARE SIMULATION

To simulate our configured integrated CPU-GPU heterogeneous environment, we have used the MacSim (Many-core Architecture Computer Simulator) [13] simulator. It is a cycle-level simulation tool. We chose to use this tool for our simulation environment as it is designed for the performance analysis of heterogeneous computing systems comprising both CPU and GPU components. Its modular design allows for flexible configuration of architectural components such as cache hierarchies, memory subsystems, and interconnect networks.

8. SIMULATION RESULTS

To validate our model, we have taken existing Intel Alder Lake and Raptor Lake processors as our baseline to compare the results. We have conducted separate studies for these baselines and analysed the reports. For this study, we have identified 8 different workloads and named them in a sequence starting from HWMix1 to HWMix8. HWMix stands for Heterogeneous Workload Mix. To justify our configured integrated CPU-GPU heterogeneous model, we have taken a mix of different workloads that comprise CPU and GPU-specific loads, 4 from each. We have a piece of detailed information about the workload mix in Table 2. After trying out with all the workloads, we have calculated the Geometric mean of the outcomes and taken it as a separate parameter. The detailed report of the study on speedup comparison of our configured integrated CPU-GPU model over Alder Lake processor is presented here in Figure 4 as a chart. Here we can see that, for most of the workloads, our proposed model is performing better than the Alder Lake processor. Considering all the workloads, the geometric mean calculated over the outcomes

Journal of Theoretical and Applied Information Technology
15th July 2025. Vol.103. No.13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

indicates a 59.8% speedup improvement achieved by our configured integrated CPU-GPU model over 1st baseline processor Alder Lake.

The average LLC read miss count of the configured integrated CPU-GPU model and Alder Lake processor, considering all the workloads, is been normalized and shown in Figure 6. Alder Lake uses

Table 2: Heterogeneous Workload Mixes

HWMix1	sphinx3, bzip2, tonto, hmmer, lps, backprop, bfs, SoftShadow
HWMix2	gcc, soplex, milc, libquantum, gaussian, nqu, heartwall, HDR
HWMix3	omnetpp, astar, bwaves, lesli3d, Mummer, POM, sto, nqu
HWMix4	lbm, GemsFDTD, bwaves, soplex, blackscholes, Stream Cluster, LIBOR, bfs
HWMix5	libquantum, mile, gromacs, tonto, RayTrace, POM, cfd, gaussian
HWMix6	gobmk, sjeng, milc, astar, sto, backprop, ray, aes
HWMix7	hmmer, lbm, soplex, sphinx3d, POM, SoftShadow, bfs, matrixMul
HWMix8	bzip2, omnetpp, bwaves, zeusmp, heartwall, ray, HDR, aes

The same study of speedup comparison is performed between our configured integrated CPU-GPU heterogeneous model and the Raptor Lake processor, taking the same set of workloads. The detailed report for the comparison is presented here in Figure 5. In this study, we can see that for a few of the workloads, the speedup result has surpassed the same of Raptor Lake. The geometric mean calculated on the outcomes by different workloads shows that our configured integrated CPU-GPU model gets a spike in speedup by 22.12% over Raptor Lake.

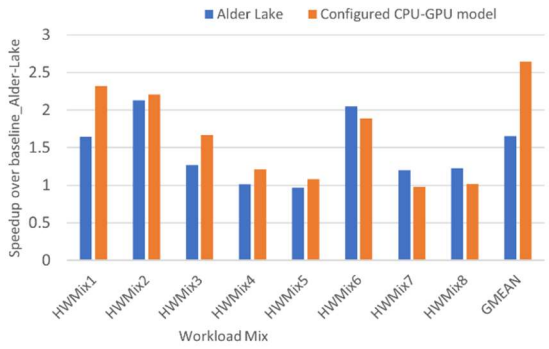

Figure 4: Speedup Comparison Of The Configured Integrated CPU-GPU Model Over The Baseline Alder Lake

We have considered a few of the traditional and well-performing cache management algorithms to validate our configured heterogeneous model. The algorithms included are DRRIP, SHiP with bypass, SHiP Hybrid, LRU, PLRU, TADRRIP, SHiP_PC, and Optimal with bypass. Again, we have taken the earlier considered processors as the baseline here. LRU for cache management. In our configured heterogeneous model, we have tried all the considered policies and collected the statistics. The bars in the graph convey 2 different values concatenated in 2 different colours, as mentioned in the graph. We can see a good read miss count improvement concerning our configured model. Among the cache management policies, SHiP with bypass shows an improvement of 34.33%, whereas many other policies have shown improvements of more than 30%.

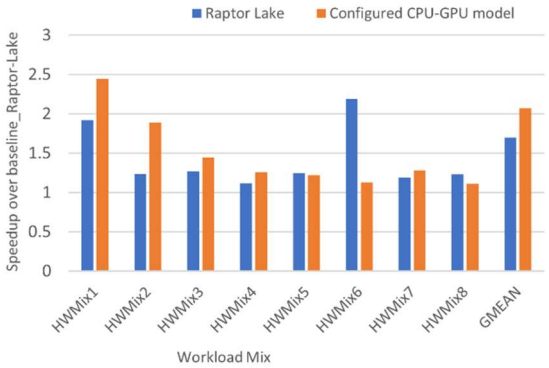

Figure 5: Speedup Comparison Of The Configured Integrated CPU-GPU Model Over The Baseline Raptor Lake

The average LLC read miss count of the configured integrated CPU-GPU model and Raptor Lake processor, considering all the workloads, is been normalized and shown in Figure 7. Raptor Lake uses PLRU for cache management. Again, in our configured heterogeneous model, we have tried all

the considered policies mentioned earlier and collected the statistics. The bars in the graph convey 2 different values concatenated in 2 different colour, as mentioned in the graph. We can see a good read miss count improvement concerning our configured model. Among the cache management policies, SHiP_Hybrid shows an improvement of 37.29%. Some other policies in the list have also shown improvements of more than 30%.

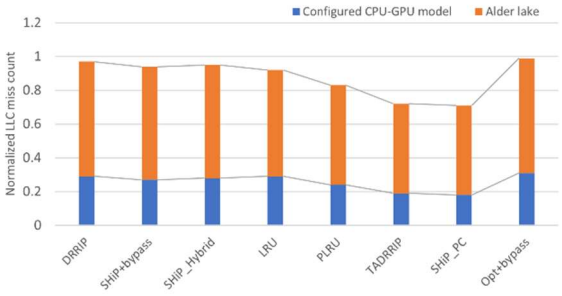

Figure 6: Comparison Of Read Miss Between The Configured Integrated CPU-GPU Model With Respect To Different Algorithms Compared With Alder Lake In Normalized Form

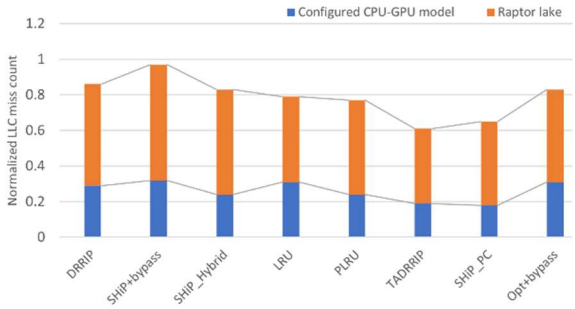

Figure 7: Comparison Of Read Miss Between The Configured Integrated CPU-GPU Model With Respect To Different Algorithms Compared With Raptor Lake In Normalized Form

9. CONCLUSION

Integrated(heterogeneous) CPU with GPU system represent a fundamental development in computing, Associating the capability of Central Processing Units (CPUs) and Graphics Processing Units (GPUs) in a single chip. This to enhances performance,high pay loads efficiency in energy, and cost cut, However, Our integrated Architecture , allotment the same memory sub-scheme. This technology may provide where memory oriented jobs on the CPU may come up with GPU transaction, with the side effects performance degradation,soft implementation Complexity and Privacy threats.

We Concludes the paper with followings:

- In our experiments, we have done speedup comparison between our configured integrated heterogeneous model with Alder Lake processor, as well as Raptor Lake processor separately. The results say our configured model has gained a good speedup improvement. Even the normalized LLC read miss count comparison, taking 8 different cache management algorithms into account, shows a better result for the configured heterogeneous model, while comparing the same with Alder Lake and Raptor Lake separately.
- A processor similar to our configured integrated heterogeneous CPU-GPU model can give better performance over Alder Lake in Raptor Lake in the majority of the workloads. We have seen that our configured heterogeneous model can give better performance results. But as the frequencies for P-Core, E-Core and also for GPU frequency, the values are quite high, they consume more power to match that. It gets heated very quickly.
- The strength of or system over Raptor lake and Alder Lake in terms of clock speed,core counts.All the system support next-generation technology and our system supports same stability level in comparison to existing two base model.

Considering applying the bypass technique to LLC to optimally use it and get better outcomes will be used as a future scope of the paper.

REFERENCES:

[1] H. Wen and W. Zhang, "Reducing Inter-Application Interferences in Integrated CPU-GPU Heterogeneous Architecture," *2018 IEEE 36th International Conference on Computer Design (ICCD)*, Orlando, FL, USA, 2018, pp. 278-281, doi: 10.1109/ICCD.2018.00050.

[2] P. Gera, H. Kim, H. Kim, S. Hong, V. George and C. -K. Luk, "Performance Characterisation and Simulation of Intel's Integrated GPU Architecture," *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Belfast, UK, 2018, pp. 139-148, doi: 10.1109/ISPASS.2018.00027.

[3] M. K. Qureshi and Y. N. Patt, "Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches," *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, Orlando, FL, USA, 2006, pp. 423-432, doi: 10.1109/MICRO.2006.49.

[4] Lee, Jaekyu and Hyesoon Kim. "TAP: A TLP-aware cache management policy for a CPU-GPU heterogeneous architecture." *IEEE International Symposium on High-Performance Comp Architecture* (2012): 1-12.

[5] P. -H. Wang, C. -H. Li and C. -L. Yang, "Latency sensitivity-based cache partitioning for heterogeneous multi-core architecture," *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, Austin, TX, USA, 2016, pp. 1-6, doi: 10.1145/2897937.2898036.

[6] V. Mekkat, A. Holey, P. -C. Yew and A. Zhai, "Managing shared last-level cache in a heterogeneous multicore processor," Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques, Edinburgh, UK, 2013, pp. 225-234, doi: 10.1109/PACT.2013.6618819.

[7] M. Wu, Y. Pei, L. Yu, T. Chen, X. Lou and T. Zhang, "WAP: The Warp Feature Aware Prefetching Method for LLC on CPU-GPU Heterogeneous Architecture," *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Sydney, NSW, Australia, 2016, pp. 414-421, doi: 10.1109/HPCC-SmartCity-DSS.2016.0066.

[8] Li, S., Meng, J., Yu, L., Ma, J., Chen, T., & Wu, M. 2015. Buffer filter: a last-level cache management policy for CPU-GPGPU heterogeneous system. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems* (pp. 266-271). IEEE.

[9] Sahu, N., Dash, B., Pattnaik, P. K., Sangam, M., Moningi, V., Verma, A., & Jha, S. (2023, December). An analysis on multi-level cache eviction policies in multi-core processor using GEM5 simulator. In *AIP Conference Proceedings* (Vol. 2981, No. 1). AIP Publishing.

[10] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely, and Joel Emer. 2010. "High performance cache replacement using re-reference interval prediction (RRIP)." SIGARCH Comput. Archit. News 38, 3 (June 2010), 60–71. https://doi.org/10.1145/1816038.1815971

[11] Wu, C. J., Jaleel, A., Hasenplaugh, W., Martonosi, M., Steely Jr, S. C., & Emer, J. (2011, December). SHiP: Signature-based hit predictor for high performance caching. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture* (pp. 430-441).

[12] Rai, S., & Chaudhuri, M. (2016, June). Exploiting Dynamic Reuse Probability to Manage Shared Last-level Caches in CPU-GPU Heterogeneous Processors. In *Proceedings of the 2016 International Conference on Supercomputing* (pp. 1-14).

[13] Kim, H., Lee, J., Lakshminarayana, N. B., Sim, J., Lim, J., & Pho, T. (2012). Macsim: A cpu-gpu heterogeneous simulation framework user guide. *Georgia Institute of Technology*, 1-57.