International Journal of Research in Advanced Electronics Engineering

E-ISSN: 2708-4566
P-ISSN: 2708-4558
IJRAEE 2024; 5(1): 23-26

www.electrojournal.com
Received: 26-12-2023
Accepted: 30-01-2024

Devi Venkatesh Gowtham
Aarnavi Research Technology, 21A/1, Ammasai Street, K.K. Pudur Sai Baba Colony, Coimbatore, Karnataka, India

Sweta S Munnoli
Java Full Stack Developer Internship, Kodnest, BTM Layout, Bengaluru, Karnataka, India

Correspondence
Devi Venkatesh Gowtham
Aarnavi Research Technology, 21A/1, Ammasai Street, K.K. Pudur Sai Baba Colony, Coimbatore, Karnataka, India

Performance analysis of partitioned caches in heterogeneous multi-core CPUs

Devi Venkatesh Gowtham and Sweta S Munnoli

DOI: https://doi.org/10.22271/27084558.2024.v5.i1a.38

Abstract
The quantity of shared cache has been growing with the swift growth of computational speed on the multi-core era. If system architects want to boost system performance, they need to make the most of shared resources. The last-level cache (LLC) is shared by numerous heterogeneous cores in an asymmetric multi-core architecture. The LLC competition is fiercer because heterogeneous cores have different memory access requirements. We present a heterogeneity-aware replacement policy for the partitioned cache (HAPC) that uses cache partitioning to decrease core-to-core interference and uses runtime tracking of the shared reuse state of each cache block within the partition to direct the replacement policy in multithreaded programs to preserve cache blocks shared by multiple cores. When updating the reuse state, the cache replacement policy typically retains cache blocks needed by large cores to improve the efficiency of large cores' LLC accesses, taking into account the difference in memory accesses to LLC and heterogeneous cores. The overall efficiency of the system can be improved by using HAPC to run multithreaded programs, as it significantly improves the performance of big cores while having almost no impact on little cores, in comparison to state-of-the-art cache replacement techniques like LRU and SRCP. Using matrix computing as the working load reduced the L2 cache miss rate by about 12% and increased the IPC by 10% according to the experimental results.

Keywords: Asymmetric multi-core, last-level cache, replacement policy, heterogeneity-aware

Introduction
Chiplets in both 2.5D and 3D arrangements may now be integrated into a single package thanks to heterogeneous integration. As a result of hot spot aggravation, it is difficult to stack high-performance multi-core processing chips on top of each other. Any performance benefit from the shorter vertical links between each chip's interconnection networks is nullified by temperature-induced DVFS throttling [1]. Some of the most important AI technologies are Multi-Agent Reinforcement Learning (MARL). A lot of communication and calculation goes into the MARL development process, which is a "Training-in-Simulation" procedure. Current MARL implementations that rely on homogenous design are unable to provide all of the aforementioned needs at the same time [2]. A growing number of parallel applications are being created on heterogeneous systems, which consist of numerous multi-core CPUs while many-core accelerators, and these systems have lately been widely used. A feedback-based elastic task scheduling scheme is suggested for better load balancing, greater device utilization, and lower scheduling overhead by dynamically adjusting the workload among devices during execution [3]. This allows multiple compute gadgets to cooperatively while concurrently execute data-parallel kernels on heterogeneous systems. To tackle the two major issues of performance and energy efficiency, modern HPC systems are very heterogeneous, with multicore CPUs and accelerators (Such Graphics Processing Units, and Intel Xeon Phis, or Field-Programmable Gate Arrays) tightly integrated. Because of this built-in feature, processing elements compete for shared on-chip resources like LLC and interconnect as well as shared nodal resources like DRAM and PCI-E links. This leads to complications like resource contention and non-uniform memory access (NUMA). Further, accelerator-specific limitations like limited main memory make efficient out-of-card execution necessary [4]. Access requests among cores are interfering from each other, and many GPUs and CPUs are combined on an identical chip to share memory. The performance of the CPU's memory access is significantly hindered by memory requests made by the GPU. The performance is significantly impacted since requests from different CPUs are

interdependent while accessing memory. Memory access latency is increased on average due to the fact that GPU cores have different access latency values [5].

Related Work

For multithreaded programs, our current work proposes a heterogeneity-aware replacement policy for the partitioned cache (HAPC) that tracks the communal reuse state of each cache block within the partition at runtime and uses this information to guide the replacement policy and reduce mutual interference between cores. When maintaining the reuse state, the cache replacement strategy often retains cache blocks needed by large cores to increase the efficiency of large cores' LLC accesses, taking into account the disparity in memory accesses to LLC across heterogeneous cores. The overall efficiency of the system can be improved by using HAPC to run multithreaded programs, as it significantly improves the performance of big cores while having almost no impact on little cores, in comparison to state-of-the-art cache replacement techniques like LRU and SRCP. The essay addresses the issue of where to best install complicated embedded real-time programs on platforms that are not compatible with one another in [7]. Each directed-acyclic-graph (DAG) in an application has a minimum inter-arrival duration for its activation requests with an end-to-end deadline by which all computations must conclude since each activation. Applications are built of directed acyclic graphs of jobs. Heterogeneous power-aware multi-core systems with DVFS capabilities, especially large, are the platforms of interest. Platforms and designs that use LITTLE Arm architectures and are equipped with hardware accelerators that can perform Dynamic Partial Reconfiguration. Using partitioned EDF-based scheduling, tasks may be placed on CPUs. Accelers on the target platform are presumed to serve demands in non-preemptive FIFO order. Additionally, there may be an alternative implementation for some of the jobs. Using layout mirrors and offsetting, the authors of [8] demonstrate and assess non-invasive methods for transforming floor plans that disperse chiplet hot spots in a 3D stack. Cycle times are maintained while chiplet redesign activities are decreased. When compared to naïve chiplet stacking, the thermally-aware multi-core approaches that result in improved performance by reducing peak temperatures with temperature-induced throttling are striking. To further enhance performance, empty offset spaces are used to expand the on-chip cache capacity. Using a cycle-level multi-core CPU performance simulator that incorporates power and thermal modeling components, the multi-core stacking approaches are shown on a 14 nm Intel Skylake-SP-like (server) floorplan model. In our study [9], we systematically analyze the performance bottleneck on platforms with multiple cores of processing power by comparing it to a state-of-the-art MARL method called Multi-Agent Deep Deterministic Policy Gradient (MADDPG). We demonstrate that (a) data parallel architectures, like GPUs, can improve the training throughput of big neural network models; and (b) special hardware, such as FPGAs with large on-chip SRAM and optimized data layouts, can speed up memory-intensive components, such as replay management as well as all-to-all communication, which cause large memory overheads on CPU and GPU multi-level cache memory hierarchies. Distribute a controller for last-level cache partitioning on multi-core systems in [10]. Through constant monitoring of run-time performance and resizing of cache partition sizes in response to application demands, our goal is to manage the Quality of Service (QoS) for applications in multi-core systems. We go over two use-cases: one that prioritizes apps based on the intended execution behavior of the system engineers, and another that encourages application fairness. We show the performance costs of keeping all system processes on an equal schedule and the effects on system applications' performance. So, instead of relying on system fairness, we use a second control technique that makes sure that cache partitions are assigned based on user-defined priorities. In an over-saturated system, our tests show that it is feasible to boost efficiency according to setpoints as well as preserve the QoS for certain applications using non-intrusive (0.3-0.7% CPU use) cache regulating techniques [15, 16].

Proposed WOrk

Careful administration of cache resources is no longer just desirable; it is essential, due to the rising number of on-chip cores and application memory demands. A potential solution to combine the capacity advantages of a shared cache with the effectiveness and isolation of private caches is cache partitioning, which entails allocating cache space to applications according to their memory demands. Partitioning the cache carelessly, though, might cause inequity, poor performance, and no assurances of quality of service. To get the most out of cache partitioning, it's obvious that you need smart techniques. Partitioning shared caches in processors with multiple cores is covered in this article. We offer a high-level overview of the cache partitioning field and classify the techniques according to key attributes.

System Model

We suggest a framework for heterogeneous-aware partitioning cache replacement policies (HAPC) to ensure that the AMP system runs as efficiently as possible. To keep core interference to a minimum, HAPC takes cache block reuse into account when replacing cache. In order to sense the LLC needs of heterogeneous cores and direct the replacement policy accordingly, HAPC analyzes the memory access features of threads and cores at runtime.

HAPC Model

The reuse counting table (RCT) in the heterogeneous-aware partition cache replenishment policy (HAPC) keeps tabs on the cache blocks' reuse status based on their access frequency. The cache block's priority to remain in the LLC is affected by its access frequency; a high access frequency results in an increased reuse count, while a low access frequency results in a lower priority. The replacement policy chooses a cache block from the lower priority blocks to evict if it hasn't been accessed relatively recently. In addition to differentiating between memory requests coming from the shared core and those coming from the local core, the RCT also sorts the origins of memory access requests. The local core's reuse of the cache block is recorded by the LC counter in the RCT. A larger LC counter indicates better locality and more cache block reuse within the core. In the RCT, the SC counter identifies the cache block that is shared through multiple cores and records each time the cache block is reused by the shared core. High shared reuse properties are indicated by a high SC value for the cache

block.

Keeping track of the historic reuse information of every cache block is done in a reuse count table (RCT) for LLC. This allows us to keep the cache blocks that are shared by multiple cores in the cache partition. The system's operation involves updating the RCT value in response to each core's memory access request and using the cache replacement policy to guide the choice of the evicted cache block. One core is referred to as the regional core of the cache block and the other cores as the shared core of the cache block if, while a multithreaded program is running, shared data is loaded into a cache block across a cache partition by one core [11]. The two items that indicate the cache line's local and shared reuse features are contained in the RCT of a cache block:

LC, short for "local count," keeps track of how many times a cache block has been reused as a result of a local core fetch request.

When a request to access shared core memory hits a cache block, the SC (share count) keeps track of how many times that memory has been reused.

At startup, the RCT's LC as well as SC counters are both set to 0. The RCT's content is updated by the cache controller. The cache controller checks the ownership of the cache blocks with the memory access request source to identify LC reuse or SC reuse when a cache hit happens. The replacement policy's eviction block is chosen by the controller based on the reuse information captured in the RCT whenever a cache miss happens.

Results & Discussion

Data Collection Mechanism

To keep track of how well a contemporary CPU is doing, it has a few internal registers called HPM (Hardware Performance Monitor Counter). They were able to read many hardware performance metrics at once, including instruction cycles and cache misses. Data for assessing cache efficiency in a multi-core architecture is collected in this work by making use of PAPI-C technology. One such application interface is PAPI, or Performance Application Interface, created by the ICL lab at the University of Tennessee. PAPI-C is able to get relevant events or collections of events from HPM via the user-level interface. The following events must be recorded by the experiment: PAPI_L2_TCM: Absence of L2 Cache; PAPI_L2_TCA: L2 Cache Accesses; PAPI_TOT_INS: Total Instructions; PAPI_TOT_CYC: Total Cycles; L2 Cache Misses Rate=PAPI_L2_TCM/PAPI_L2_TCA;
IPC=PAPI_TOT_INS/PAPI_TOT_CYC;

Data Experiment Software and Hardware Environment

System requirements: 1.8 GHz AMD Athlon(tm) Dual Core Processor 5000B, Fedora 10, Kernel Version 2.6.32. L2 Shared 512 KB associative 16-way set, 64 KB cache line. Approach to Testing: The working load in the experiment is matrix multiplication. Experiment uses it because it is faster and more reliable with cached data. The experiment must first put the matrix multiplication code into PAPI while collecting events, since the PAPI-C technique is used to gather relevant events. Under various memory management strategies, the experiment runs various working loads of matrix computing. A 5*5, 10*10, or 15*15 matrix is used for the working loads [12], [13]. The value of the matrix element is 2. The following occurrences are recorded by each experiment in turn: PAPI_L2_TCM、PAPI_L2_TCA、PAPI_TOT_INS、PAPI_TOT _CYC. Matrix multiplication code is shown as follows:

We need to create a comparison measure so we may examine the impact of different memory management adjustments. An easy and straightforward way to measure cache performance is by looking at the average IPC with L2 Cache Misses Rate [14]. The results of the experiment are derived from the average of the data collected from many collections. Here are the outcomes of the experiment:

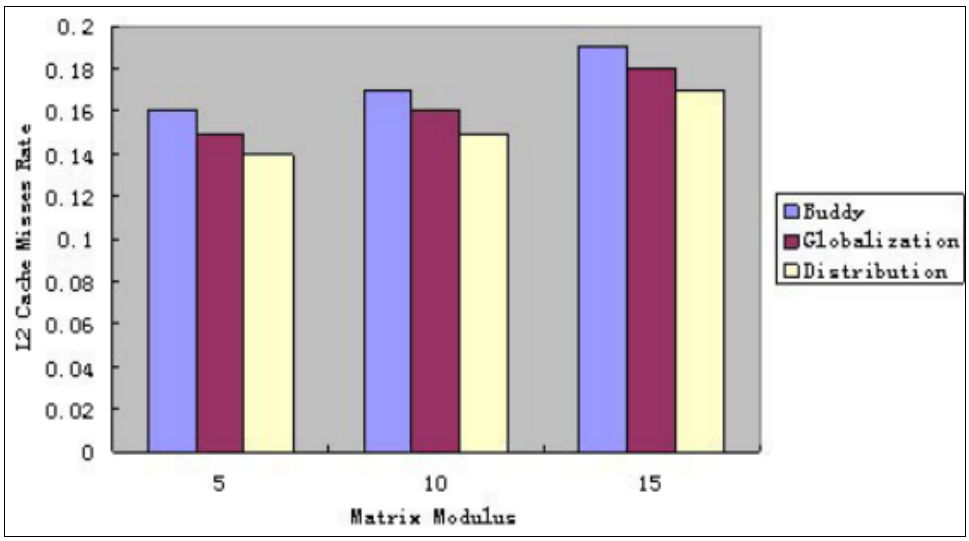

Fig 1: Evaluation of Multiple Algorithms for L2 Cache Miss Rate Comparison

When compared to multicore CPUs, the potential and difficulties of partitioning the shared cache among the CPU and GPU are different due to basic variations in their microarchitectures and the workloads that execute on them. With CPU-GPU processors slowly making their way into mainstream computing systems, it is certain that academics will find it intriguing to extend current CPTs to these processors and create innovative CPTs for them. In order to

operate their caches more efficiently, modern processors use techniques such as data compression, cache prefetching, power-gating, cache bypassing, and many more. Successfully integrating CP with these methods will be crucial for its incorporation into contemporary CPUs. The next logical step is to investigate how CP interacts with current cache management strategies in depth. Insights gained are something that some scholars have noticed.

Conclusion

In order to privatize free lists, this study primarily reconstructs their structure using a distributed manner based on the notion of page coloring. This approach might enable page-level parallelism and prevent cache thrashing across applications. Optimization using software approach on shared cache has received a lot of attention recently, thanks to the arrival of multi-core computers. Unfair usage of shared cache by memory management leads to CPU use, which in turn affects system performance. How to ensure equitable use on shared cache is something we want to investigate in future work. In addition, there is promising future business for embedded smart phone system development that relies on page coloring for memory management.

References

1. Fang J, Wang M, Wei Z. A memory scheduling strategy for eliminating memory access interference in heterogeneous systems. J Supercomputing. 2020;76:3129-3154.
2. Singh DP. Forecasting of supermarket sales using big data analytics and machine learning techniques in the business sector. Int. J Core Eng. Manag. 2023;7(06):18-30.
3. Manukonda K. Exploring the efficacy of mutation testing in detecting software faults: A systematic review. 2020;7:71-77. DOI: 10.5281/zenodo.11408273.
4. Singh DP. An efficient system for customer relationship management on churn prediction using machine learning technique. Int. J Core Eng. Manag. 2022;7(04):19-34.
5. Manukonda K. Performance evaluation and optimization of switched Ethernet services in modern networking environments; c2020.
6. Yenugula M. Examining partitioned caches performance in heterogeneous multi-core processors.
7. Fang J, Wei Z, Liu Y, Hou Y. A task-based routing algorithm for network-on-chip in heterogeneous CPU-GPU architectures. In: 2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys); 2023. p. 758-763.
8. Manukonda K. Efficient test case generation using combinatorial test design: towards enhanced testing effectiveness and resource utilization. 2020;7:78-83.
9. Yenugula M. Monitoring performance computing environments and autoscaling using AI.
10. Cucinotta T, Amory AM, Ara G, Paladino F, Natale MD. Multi-criteria optimization of real-time DAGs on heterogeneous platforms under P-EDF. ACM Trans Embed Comput Syst. 2023;23:31-35.
11. Kothari G, Ghose K. Thermally-aware multi-core chiplet stacking. In: 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD), 2023. p. 01-09.
12. Wiggins S, Meng Y, Kannan R, Prasanna VK. Evaluating multi-agent reinforcement learning on heterogeneous platforms. Defense + Commercial Sensing; c2023.
13. Danielsson J, Seceleanu T, Jägemar M, Behnam M, Sjödin M. Automatic quality of service control in multi-core systems using cache partitioning. In: 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA); c2021. p. 1-8.
14. Sethy NK, Yenugula M, Goswami SS, Bhola A, Behera DK. Selection of ideal IoT-based overhead conductor for optimizing the performance of a small hydropower project.
15. Geetha BT, Yenugula M, Randhawa N, Purohit P, Maney KL, Venkteshwar A, *et al*. Advancement improving the acquisition of customer insights in digital marketing by utilising advanced artificial intelligence algorithms. In: 2024 International Conference on Trends in Quantum Computing and Emerging Business Technologies; Pune, India; c2024. p. 01-07. DOI: 10.1109/TQCEBT59414.2024.10545055.
16. Fang J, Wang M, Wei Z. A memory scheduling strategy for eliminating memory access interference in heterogeneous systems. J Supercomputing. 2020;76:3129-3154.
17. Yenugula M. Monitoring performance computing environments and autoscaling using AI.
18. Fang J, Kong H, Yang H, Xu Y, Cai M. A heterogeneity-aware replacement policy for the partitioned cache on asymmetric multi-core architectures. Micromachines. 2022, 13.