LAMA: Link-Aware Hybrid Management for Memory Accesses in Emerging CPU-FPGA Platforms

Liang Feng[1], Jieru Zhao[1], Tingyuan Liang[1], Sharad Sinha[2] and Wei Zhang[1]
[1]*Hong Kong University of Science and Technology, Hong Kong, {lfengad, jzhaoao, tliang, wei.zhang}@ust.hk*
[2]*India Institute of Technology Goa, India, sinh0001@e.ntu.edu.sg*

ABSTRACT

To satisfy increasing computing demands, heterogeneous computing platforms are gaining attention, especially CPU-FPGA platforms. Recently, emerging tightly coupled CPU-FPGA platforms with shared coherent caches (such as the Intel HARP and IBM POWER with CAPI) have been proposed to facilitate data communication and simplify the programming model. In this work, we propose LAMA, a static analysis and dynamic control combined framework for memory access management in such platforms, to further enhance the memory access efficiency and maintain the data consistency. Based on implementation results on the real Intel HARP2 platform, LAMA is shown to improve the performance by 34% on average with low overhead.

1 INTRODUCTION

The increasing demand for energy efficiency and performance in today's computing has stimulated the success of heterogeneous computing platforms in a wide range of computing scenarios, such as machine learning, cybersecurity, genomics, etc. Among them, CPU-FPGA platforms are promising due to the reconfigurability of the FPGA, which enables hardware customization to fit the specific computing requirements. In emerging CPU-FPGA platforms such as the Intel HARP series [5] and IBM Power series with CAPI [11], the tightly integrated CPU and FPGA coherently share the same cache system, so that the data communication efficiency is enhanced and the programming model is evolved to where the FPGA can act as a coherent peer to the CPU to access the same virtual memory space. In the recently released Intel HARP2[5], there are three links between the XEON processor and the Arria 10 FPGA fabric, one QPI link and two PCIe links, which are all connected to the last level cache (LLC) residing on the CPU side, with coherence guaranteed. Moreover, a coherent FPGA cache is attached to the QPI link for quick memory access and data reuse. HARP2-style platforms are highly promising due to the architecture advance and have shown large benefit in lots of computing scenarios [10][13].

The choice among the multiple links in HARP2-style emerging CPU-FPGA platforms for each access can heavily influence the memory access efficiency, thus affecting the whole performance. All the links need to be fully utilized in balance at the same time to boost the bandwidth. Moreover, the access of the data with high reuse potential can benefit from the FPGA cache by choosing the QPI link. In contrast, it is hard for the access of the data with low reuse potential to benefit from the FPGA cache, but it can avoid polluting the cache if using the PCIe link. A bad link choice may cause link contention with unbalanced link utilization and increase the FPGA cache misses by wasting the reuse benefit. On the other hand, the data consistency among links can be violated when the accesses to the same data block are issued to different links, which can cause severe mistakes.

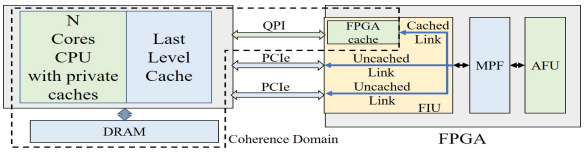

Figure 1: HARP2 Platform Architecture

In this paper, we propose LAMA, a static (compile-time) and dynamic (run-time) collaborative memory access management framework for emerging CPU-FPGA platforms, such as Intel HARP2, to select the proper link for the memory accesses from the FPGA to keep the balance among links and boost the FPGA cache reuse benefit. As well, LAMA maintains the data consistency regarding the consistency hazards. The static analysis in LAMA provides accurate directions to the dynamic control. The dynamic control makes the link selection decision not only with high quality, due to the accurate directions, but also adaptively to the real-time status, including the utilization of links and the content in the FPGA cache. We implement diverse applications on the real HARP2 platform for evaluation. To the best of our knowledge, this is the first work on memory access management targeting the HARP2-style CPU-FPGA platforms, which have multiple communication links to connect the FPGA to the shared memory system. This paper makes the following contributions:

- A hybrid memory access management framework for HARP2-style CPU-FPGA platforms.
- A reuse distance-based and data consistency-aware analysis to classify the accessed data blocks into six types.
- A balance-aware adaptive control based on the classification to select link for the accesses to better utilize all links and the FPGA cache.
- A pair list data structure-based algorithm and a post-processing to decide an optimal data consistency maintenance scheme.
- A complete set of software services and high-performance hardware IPs to ease the use of LAMA.

2 HARP2-STYLE CPU-FPGA PLATFORM

Emerging CPU-FPGA platforms such as Intel HARP2 tightly integrate the multi-core CPU and the FPGA via bus as in Fig. 1. The

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317846

unit to execute the accelerated task on the FPGA is called the Accelerated Function Unit (AFU). In HARP2, the CPU and the FPGA coherently share one LLC and the DRAM, which both reside on the CPU side. There are three links between the FPGA and the LLC, one QPI link and two PCIe links, one of which can be chosen for each memory access from the FPGA. A cache is implemented on the FPGA at the QPI side for quick and flexible memory access. The CPU local caches, FPGA cache, LLC and DRAM form the shared memory system within the same coherence domain with the support of coherence protocol. The explicit software control for data movement and coherence maintenance are eliminated due to the coherent caches. The AFU will be assigned a continuous memory region from the virtual memory space of the CPU thread, called the workspace, for allocating its related data. The AFU initiates accesses using virtual addresses at cacheline granularity. The virtual address will be translated into the physical one by the Memory Properties Factory (MPF), and then go to one of the three links through the FPGA Interface Unit (FIU). When using the QPI, the FPGA cache will be accessed, where the access will be satisfied rapidly if hit or the missing cacheline will be requested from the LLC to the FPGA cache through the QPI if miss. When using a PCIe, the LLC will be accessed through the PCIe. Requested data locating in the FPGA cache may transfer from the FPGA to the LLC and back to the FPGA through the coherence mechanism with extremely large overhead if using a PCIe. Hence the FPGA cache must be considered for link selection. We denote the QPI as cached link and the PCIes as uncached links.

In HARP2, even though the MPF can maintain the ordering among all the accesses, the data consistency can still be violated due to two kinds of hazards when an access is issued to a different link after a preceding Write for the same data block. If the later-issued access is Read, the preceding Write may have not been updated from the perspective of the link for the Read, which is a Read-After-Write (RAW) hazard. If the later-issued access is Write, which Writes will be finally recorded can not be guaranteed, which is a Write-After-Write (WAW) hazard. Such RAW/WAW hazards can cause execution mistakes. Hence, the WrFence access is provided in HARP2. By issuing a WrFence to the memory system, the update from all the previous Writes can be globally observable by all the links, and thus all the links come to a synchronized data-consistent point. Therefore, inserting a WrFence between a Write and a later access eliminates the data inconsistency. However, WrFence has a large overhead of hundreds of clock cycles. Hence, a sophisticated scheme is needed to reduce the number of required WrFences.

3 RELATED WORK

One of the effects of the scheme we propose is boosting the FPGA cache benefit. In the CPU and GPU domains, to boost the cache reuse benefit, cache bypassing, where the memory requests to selectively bypass the cache to mitigate the cache contention, is used [12][6][7][2]. These methods usually predict the reuse potential by run-time monitoring, which results in large overhead and low accuracy due to the limited predicting view. In emerging CPU-FPGA platforms, the memory access pattern from the AFU is usually determinate, which makes the static pre-analysis feasible for an accurate reuse benefit estimation to enable a better solution quality and eliminate the run-time prediction overhead.

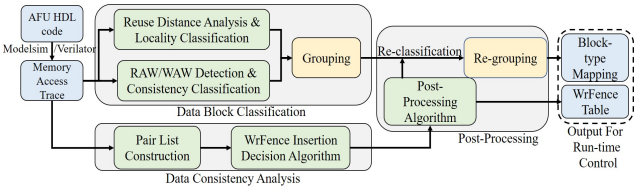

Figure 2: Static Analysis Flow

For CPU-FPGA platforms, [3] alleviates the contention among AFUs on the FPGA cache by combining cache bypassing and partitioning, while [4] adopts cache bypassing to increase the FPGA cache hit rate. However, the platforms targeted by these works have only one bus link between the CPU and FPGA. The three parallel bus links between the CPU and FPGA in HARP2-style CPU-FPGA platforms that we target make the problem different from two aspects. First, the balance among the links will influence the link utilization and effective bandwidth, and thus the performance. Second, the race among the links may cause data inconsistency. Moreover, these works require modification to the cache structure, while our scheme is implemented on the real HARP2 platform without the cache structure modification allowance. For HARP2-style platforms, the combination of link balance, data consistency and cache benefit makes the problem more challenging.

4 OVERVIEW OF LAMA FRAMEWORK

LAMA consists of a static analysis and a run-time control. FPGA tasks often hold determinate access pattern regardless of the input data, such as DNN, signal processing, etc., hence the memory access pattern can be analyzed using only a sample memory access trace. In the static step, the memory access trace of the AFU is profiled and analyzed. The data blocks are classified into six types reflecting different link preferences and data consistency maintenance demands. Data block is defined to index the AFU workspace based on the cacheline size, where each data block corresponds to the data in one cacheline. Meanwhile, the data consistency analysis optimally decides a data consistency maintenance scheme. Then, the post-processing simplifies the data consistency maintenance scheme and accordingly polishes the classification result. The static analysis flow is shown in Fig. 2. At run-time, according to the classified type and the real-time status of the links and the cache, an adaptive control selects the link for each access. Meanwhile, the WrFences are automatically inserted based on the pre-decided data consistency maintenance scheme, where the link selection and the WrFence insertion are both responsible for the data consistency maintenance. The classified type, which is aware of the cache reuse potential, guides the link selection so that the FPGA cache benefit is enhanced. The adaptability of the run-time control to real-time status helps to achieve balance among links, and thus guarantees a better utilization of all links.

5 STATIC ANALYSIS

5.1 Data Block Classification

In the static step, reuse distance analysis [1] is applied to the memory access trace of the AFU to derive the reuse distance of each access regarding different cache sets. The accesses with a reuse

distance smaller than the cache associativity will hit in the cache, and otherwise will miss. When some of the preceding accesses use uncached links, an access may still hit even though its reuse distance is not smaller than the cache associativity. The larger the reuse distance is, the harder it is for an access to hit. Larger distance means lower locality. From this observation, a locality classification is performed. First, the average reuse distance of all the accesses to each data block is calculated as $Mean_{dis}$ as equation (1), where Acc represents an access to the data block B and $Acc.Dis$ represents its reuse distance. The threshold D_M alleviates the impact of extremely large distances to $Mean_{dis}$. Then, lines 2~9 in Algorithm 1 classify each data block into three types. H-type indicates a high locality and a high demand for choosing the cached link to benefit from the cache hit. L-type indicates a low locality, where the accesses are hard to hit and thus prefer uncached links to avoid polluting the cache to make space for other accesses which may benefit from the cache. M-type indicates a medium locality and the accesses hold a medium reuse potential with no special link preference. H-type, M-type and L-type are the locality type for a data block.

$$Mean_{dis} = \frac{\sum_{all\ Acc\ to\ B} min(Acc.Dis, D_M)}{\sum_{all\ Acc\ to\ B} 1} \quad (1)$$

Consistency classification is performed at the same time. From the memory access trace, the WAW and RAW hazards are detected. If there is WAW or RAW hazard among the accesses to a data block, the block is classified as D-type. Otherwise, it is classified as N-type. D-type and N-type are the consistency type for a data block. Only a D-type data block has the data consistency maintenance demand.

In Algorithm 1, totally six types, reflecting the different locality levels and data consistency maintenance demands, are classified. These are H-D-type, H-N-type, M-D-type, M-N-type, L-D-type and L-N-type. A block-type mapping, which maps each data block ID to one of the six types, is generated. The mapping will be compressed by grouping, where continuous data blocks of the same type will be grouped together.

Algorithm 1 Data Block Classification Algorithm

```
1:  for each data block B in workspace do
2:      calculate Mean_dis for B
3:      if Mean_dis < threshold T_S then
4:          data block B is H − type
5:      else if Mean_dis > threshold T_L then
6:          data block B is L − type
7:      else
8:          data block B is M − type
9:      end if (locality classification)
10:     if RAW or WAW exitsts for B then
11:         data block B is D − type
12:     else
13:         data block B is N − type
14:     end if (consistency classification)
15:     combine locality type and consistency type for B
16: end for
```

5.2 Data Consistency Analysis

The data consistency analysis generates a WrFence table, which reflects how to insert the WrFences to guarantee that a WrFence is inserted between every pair of accesses with RAW/WAW hazard. From the memory access trace, each Write W, and the first access to the same data block after this Write S are extracted to form a pair. All the extracted pairs form a pair list data structure. The access ID of W and S are annotated as $W.ID$ and $S.ID$. The ID of an access is the number of accesses before it and reflects the sequential order of the accesses. To guarantee whole data consistency, WrFence should exist between each pair. The Write ID is also annotated for each W as ID_{Wr}, where a Write with Write ID n identifies the nth Write from the start of execution.

Algorithm 2 WrFence Insertion Decision Algorithm

```
1: WrFence_Table = ∅
2: while (!Pair_List.empty()) do
3:     P_max = the pair with the largest W.ID in Pair_List
4:     WID_max = W.ID of P_max
5:     ID_Wrmax = ID_Wr of P_max
6:     WrFence_Table = WrFence_Table ∪ {ID_Wrmax}
7:     Remove the pairs whose S.ID > WID_max from Pair_List
8: end while
9: return WrFence_Table
```

With the pair list data structure, Algorithm 2 tries to locate each WrFence between as many pairs as possible to minimize the number of required WrFences. In each iteration, the ID_{Wr} of the pair with the largest $W.ID$ is recorded in the WrFence table, which means a WrFence is required immediately after the W of this pair. For each removed pair in one iteration, its W is before the recorded WrFence and its S is later than the recorded WrFence according to the sequential order reflected by the access ID, so the recorded WrFence exists between all the removed pairs. Finally, all the pairs are removed, and thus a WrFence is guaranteed between each pair. The pair list is implemented using a red-black tree [8] based structure and the whole time complexity of the WrFence insertion decision algorithm is O($n$log$n$).

The number of WrFences should be as small as possible to reduce the WrFence overhead. Algorithm 2 can give the optimal solution, where the number of recorded WrFences is exactly the minimal required number of WrFences for guaranteeing a WrFence between each pair. The proof of the optimality is as follows.

Lemma 5.1. *Given that the number of WrFences recorded by Algorithm 2 is n, then at least n Wrfences are needed to guarantee a WrFence between each pair of W and S in the pair list.*

Proof. In Algorithm 2, the number of recorded WrFences n is the number of iterations. P_{max} in iteration i is P_i. The $S.ID$ of P_i must be smaller than the $W.ID$ of P_{i-1}; otherwise P_i would have already been removed in iteration $i-1$. Also for each pair, $W.ID < S.ID$. Hence, no intersection exits between P_i and P_{i-1}. So, P_{max} from all n iterations are without intersection. A WrFence is needed between each pair, and the pairs without intersection cannot share the WrFence. So one unique WrFence is needed for P_{max} from one iteration. Therefore at least n WrFences are needed, where each WrFence corresponds to P_{max} from each iteration. □

5.3 Post-Processing

Algorithm 3 Post-Processing Algorithm

```
1: Group_Pool = ∅
2: while (WrFence_Table.size() > T_max) do
3:     G_max = the group with most un-removed effective WrFences
4:     Group_Pool = Group_Pool ∪ G_max
5:     Remove WrFences from WrFence_Table whose effective groups are all in Group_Pool
6: end while
7: for each group G not in Group_Pool do
8:     if G.consistency_type == D − type then
9:         G.consistency_type → N − type
10:    end if
11: end for
```

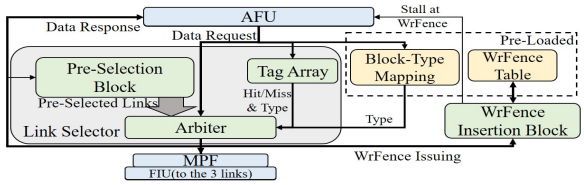

Figure 3: Run-time Control Logic on FPGA

The WrFence table will be implemented on the FPGA for the run-time control; thus its size cannot be too large. Post-processing removes some WrFences to reduce the WrFence table size with the help of D-type accesses if the size is larger than a threshold T_{max}. The run-time control will guarantee the D-type accesses to the same data block always use the same link, so data inconsistency is not possible and no WrFence is needed for D-type accesses. The data consistency originally maintained by the removed WrFences will be guaranteed by the fixed link usage of D-type accesses. The WrFences and D-type accesses will collaborate to maintain the data consistency at run-time. However, fixing the link usage will harm the adaptability of link selection and exacerbate the unbalance among links. Hence, D-type appears in the final block-type mapping only when the number of WrFences is too large. The post-processing algorithm is shown in Algorithm 3. T_{max} is set to 256 in the experiments and adjustable. The effective WrFence of a pair from the pair list is defined as the first recorded WrFence after the W of this pair. This WrFence is also the effective WrFence of the group in the block-type mapping which contains the accessed data block of this pair. This group is also called the effective group of its effective WrFence. In Algorithm 3, the data consistency of the groups in *Group_Pool* will be guaranteed by the fixed link usage of D-type accesses with no need of WrFence, so the WrFences only effective for these groups are removed and their consistency type remains D-type. The consistency type of the other groups will be re-classified to N-type, since their data consistency will be handled by WrFences with no need of D-type for the fixed link usage. The adjacent groups of the same type after the re-classification will be re-grouped. The post-processing always guarantees a limited WrFence table size with the help of D-type and thus makes LAMA work well for larger scale tasks no matter how many WrFences originally required. Through the post-processing, the shrunken WrFence table and re-grouped block-type mapping are finally output from the static analysis.

6 RUN-TIME CONTROL

The run-time control is shown in Fig. 3. The block-type mapping is checked for each data request from the AFU. The checked type is used by a link selector to adaptively select the link for the access based on the real-time status. The request is then sent to the chosen link passing the MPF, which handles the address translation and the ordering among all the accesses. A WrFence insertion block issues required WrFences following the WrFence table.

6.1 Data Pre-loading and WrFence Insertion

In the block-type mapping, several boundaries separate the workspace into groups. The structure to represent a mapping consists of the ID of the data blocks at the boundaries and the type of each group. At run-time, by comparing the requested data block ID with the boundary IDs, the type of its belonging group can be found. Usually in a task, continuous data is highly groupable with the same access pattern and the total number of data groups is not large due to the limited number of data categories in most tasks, so that the block-type mapping size is not large, which can be efficiently implemented on the FPGA. The structure of the WrFence table consists of a series of ID_{Wr}s in ascending order. The block-type mapping and the WrFence table are prepared as a chunk of data by the CPU software and loaded onto the registers and BRAMs on the FPGA through the shared memory system. Such data pre-loading is performed ahead of the execution and thus will not incur overhead.

At run-time, the WrFence insertion block initiates WrFence requests according to the WrFence table. The WrFence table records the ID_{Wr} where the WrFences should be inserted. As soon as the counter for completed Writes reaches the ID_{Wr} for the next WrFence to be initiated, a WrFence will be issued using a state machine design, where all the unsettled requests from the AFU will be stalled until the response of the WrFence returns.

Table 1: Eight Conditions for Link Selection

	Tag Array	$Type_{new}$	$Type_{old}$	Selected Link		Tag Array	$Type_{new}$	$Type_{old}$	Selected Link
A	Hit	Any	Any	cached link	E	Miss	L-N-type	H/M-type	N_{pd}-decided
B	Miss	H-type	Any	cached link	F	Miss	M-N-type	L-type	N_{pd}-decided
C	Miss	L-D-type	Any	LSB-decided	G	Miss	M-N-type	H/M-type	N_{pd}-decided
D	Miss	M-D-type	Any	LSBs-decided	H	Miss	L-N-type	L-type	N_{pd}-decided

6.2 Link Selection

The link selector consists of a tag array, a pre-selection block and an arbiter. The tag array records the tag and type for the data blocks in the FPGA cache and is handled with virtual address. It keeps consistent with the FPGA cache. Once a request comes from the AFU, the tag array is checked to see whether the requested data block Blk_{req} currently resides in the FPGA cache. If there is a tag array miss, the type of the conflicting data block Blk_{cof} for this miss will be output, denoted as $Type_{old}$. In parallel, the type of Blk_{req} will be checked from the block-type mapping and is denoted as $Type_{new}$. Blk_{cof} will be replaced by Blk_{req} only if the request chooses the cached link. The tag array not only enables pre-knowledge of hit or miss since the FPGA cache cannot be checked before choosing the cached link, but also provides the $Type_{old}$ information, both of which help in making a better link selection.

There are eight conditions according to the tag array hit or miss and the checked types, as in Table 1. For condition A, the cached link will be selected for a rapid access since Blk_{req} resides in the cache. For condition B, $Type_{new}$ as H-type indicates a high cache reuse benefit of Blk_{req}, so the cached link will be selected to explore more cache hits.

6.2.1 LSB-Based Selection for D-type. The selected link is decided according to the LSBs (least significant bits) of the requested data block ID for condition C and D. In condition C, two uncached links will be selected respectively when the LSB is 0 or 1. In condition D, if the least significant two bits are 00 or 01, the cached link will be selected. And if they are 10 or 11, two uncached links will be selected respectively. In this way, the D-type accesses to the same data block are fixed to the same link, which is critical to maintain the data consistency, as discussed in Sec. 5.3. Such a selection not only simplifies the design, but also considers the cache reuse potential, where the L-D-type accesses only choose uncached links so as to not pollute the cache, while the M-D-type accesses with no special

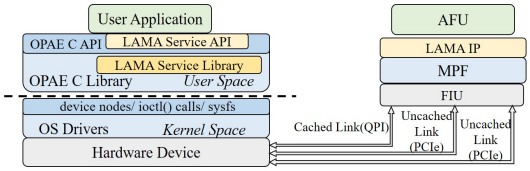

Figure 4: Complete Framework Layers of LAMA

link preference can choose among all the links. The access data block ID tends to hold uniformly distributed LSBs, so the D-type accesses can be uniformly distributed to the preferred links and balance the link usage to some extent.

6.2.2 Utilization-Adaptive Selection for N-type. For conditions E~H, the utilizations of the links N_{pd} are monitored. N_{pd} counts the number of requested cachelines of the pending requests on each link. For link balance, N_{pd} of all the links would be better to be at the same level. The selection rules for conditions E~G are similar, and are shown in equations 2~4, respectively. The cached link is denoted as LC and the one with the smaller N_{pd} between two uncached links is denoted as LU_s. We first compare N_{pd} of LC and LU_s with a threshold (Slk_{LM}, Slk_{ML} or Slk_{MM}, which can be set to 8 and adjustable) to see whether the link is heavily unbalanced. If so, the less utilized link is selected to enhance the link balance. Otherwise, the access will select the link which potentially provides more cache benefit according to the $Type_{new}$ and $Type_{old}$ in that condition. For example, in condition E (equation 2), according to $Type_{new}$ and $Type_{old}$, the cache reuse potential of Blk_{cof} is higher than Blk_{req}, so normally uncached links are preferred to keep Blk_{cof} in the cache. The cached link (LC) is selected only when its utilization (N_{pd}) is much lower than the uncached links (LU_s); otherwise uncached links are used, between which LU_s is selected to further enhance the balance. For condition H, Blk_{req} and Blk_{cof} both hold little cache reuse potential and allocating either to the cache has the same effect. So the link with the smallest N_{pd} among all links is selected to only consider the link balance. Based on the cache benefit type of both the requested and conflicting data, the selection adapts to the real-time status of the link utilization and cache content to balance the link utilization, enhance the cache benefit, and achieve a comprehensive optimization for the memory access efficiency.

$$\begin{cases} LC, & if\ LC.N_{pd} + Slk_{LM} < LU_s.N_{pd} \quad (heavily\ unbalanced) \\ LU_s, & otherwise \quad (to\ enhance\ cache\ benefit) \end{cases} \tag{2}$$

$$\begin{cases} LU_s, & if\ LU_s.N_{pd} + Slk_{ML} < LC.N_{pd} \quad (heavily\ unbalanced) \\ LC, & otherwise \quad (to\ enhance\ cache\ benefit) \end{cases} \tag{3}$$

$$\begin{cases} LC, & if\ LC.N_{pd} + Slk_{MM} < LU_s.N_{pd} \quad (heavily\ unbalanced) \\ LU_s, & otherwise \quad (to\ enhance\ cache\ benefit) \end{cases} \tag{4}$$

Normally, the selection starts only after checking the tag array and block-type mapping. To overlap the latency, we propose pre-selection, where the selected link within each condition are decided in parallel in the pre-selection block before the checking results are known. The pre-selection is executed in parallel with the checking. Then, the arbiter selects the final link among the pre-selected links according to the checking results.

7 COMPLETE FRAMEWORK SUPPORT

The complete software and hardware support for LAMA is provided on the real HARP2 platform, as in Fig. 4. The dynamic control is designed as a HARP2-compatible FPGA IP with high efficiency, where the sizes of the block-type mapping and WrFence table are parameterized. Based on the Intel Open Programmable Acceleration Engine (OPAE) environment, a set of multi-layer software services are designed to automatically manage the LAMA IP and handle its discovery, MMIO mapping, initialization, setting and data pre-loading by simple API calls. Such management can be performed off-line and thus will not incur overhead. Such a software services and hardware IP combined framework not only eliminates the user workload to use LAMA, but also guarantees high efficiency, extensibility and portability.

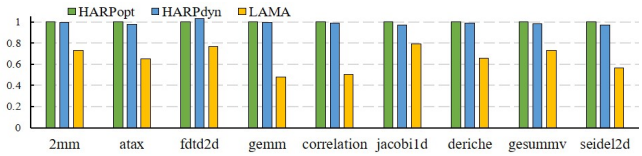

Figure 5: Comparison to Advanced Methods

8 IMPLEMENTATION AND EVALUATION

We complete the implementation of nine applications on the real HARP2 platform based on Polybench [9]. The AFUs are generated using Intel HLS, where both codes and HLS settings are tuned to be HARP2-compatible with good performance. We efficiently integrate the HLS-generated interface with HARP2. The frequency is set to 200 MHz as many HARP-based designs [10][13], which is also the maximal frequency achieved by HLS for these applications on HARP2.

8.1 Comparison to Advanced Methods

In HARP2, two advanced methods combining link selection optimization and data consistency maintenance are provided, denoted as HARP_{opt} and HARP_{dyn}. The total AFU execution time using HARP_{opt}, HARP_{dyn} and LAMA, normalized to HARP_{opt}, is shown in Fig. 5. LAMA improves the performance in terms of execution time by 34.6% and 33.9% on average compared with HARP*opt* and HARP_{dyn}, respectively. The large improvement comes from two aspects by enhancing the memory access efficiency. First, HARP_{opt} fixes the link selection for the same data block and HARP_{dyn} can only change the link selection for the same data block periodically, which suppress the link balance and waste the link utilization due to the loss of adaptability. While in LAMA, the N-type link selection adapts to the real-time link utilization, so a more balanced link utilization is achieved, which eliminates the contention on one link and fully utilizes all the links, thus enhancing the effective bandwidth. Second, the cache reuse benefit can be boosted in LAMA, while HARP_{opt} and HARP_{dyn} cannot take the cache reuse benefit into consideration. The cache benefit is especially high for HARP2-style platforms, where a non hit access needs to pass the bus to the LLC residing on the CPU side.

8.2 Discussion on Data Consistency

The WrFence table size (W-Num), the number of all groups (G-Num) and D-type groups (D-Num) in the block-type mapping before and after the post-processing are shown in Table 2. The number of RAW/WAW hazards (H-Num) is also shown. Algorithm 2 finds the minimal required WrFences by maximizing the WrFence sharing

Table 2: Post-Processing Results

	2mm	atax	fdtd2d	gemm	correlation	jacobi1d	deriche	gesummv	seidel2d
H-Num	8128	49236	1261065	16777216	14669	248833	5120	3577	33521671
W-Num$_{before}$	1	96	888592	16384	6302	122	3	3577	33513484
W-Num$_{after}$	1	96	8	0	3	122	3	0	0
G-Num$_{before}$	3	4	66	3	3	2	4	3	4
G-Num$_{after}$	3	4	5	3	3	2	4	3	4
D-Num$_{before}$	1	2	33	1	3	2	3	1	1
D-Num$_{after}$	0	0	1	1	1	0	0	1	1

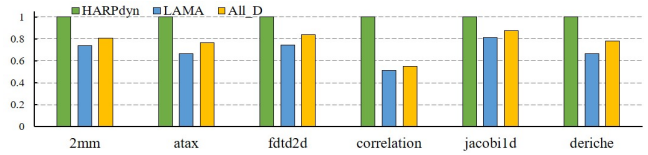

Figure 6: Comparison for Post-Processing Effect

among RAW/WAW hazards and thus reduces the WrFence table size to W-Num$_{before}$ compared with H-Num, especially in $atax$, etc. The post-processing further reduces the large WrFence table size to make it sustainable on the FPGA, such as in $fdtd2d$, etc. Some D-type groups are not needed since their data consistency is maintained by WrFences, which results in a reduced number of D-type groups after post-processing, such as in $deriche$, etc. If the number of required WrFences before post-processing is small, the data consistency will be maintained only by WrFences and D-type will be avoided since it may harm the adaptability. However, when the WrFence table before post-processing is too large, using D-type enables Algorithm 3 to reduce the table size to be sustainable on the FPGA while guaranteeing data consistency.

We compare the normalized AFU execution time using HARP$_{dyn}$, LAMA and All_D in Fig. 6. In All_D, we keep the D-type before the post-processing to maintain the data consistency with no WrFence. Applications whose D-type unchanged through post-processing are not considered, since All_D is the same as LAMA for them. All_D improves the performance by 23% on average compared with HARP$_{dyn}$. In LAMA, the data without RAW/WAW hazard is distinguished as N-type and always holds adaptability when selecting the link which results in better link balance and cache usage. Moreover, the D-type selection in Sec 6.2.1 still considers the link balance to some extent, and the locality classification still boosts the cache reuse benefit. Hence, LAMA still shows the benefit, even though fixed D-type selection is used for all the data with RAW/WAW hazard as in All_D. LAMA improves the performance by 11.5% on average compared with All_D and shows the further benefit of using WrFences, which is because the reduction of D-type enabled by WrFences ensures more adaptability due to less fixed link selection. The number of inserted WrFences becomes small through the post-processing; thus their expense will not have a large influence.

In total, the WrFences and the fixed link usage of D-type collaborate to maintain the data consistency. The experiments all run successfully, which proves the data consistency maintenance ability of LAMA. Note that the MPF handles the ordering among all the accesses, which also contributes to the data consistency.

8.3 Effect of Classification

We modify LAMA as three methods, All_H, All_M and All_L, in which the locality type of all data is treated as H-type, M-type and L-type, respectively. D-type and N-type are still classified. The AFU execution time with All_H, All_M, All_L and LAMA, normalized to HARP$_{dyn}$, is shown in Fig. 7. All_H, All_M and All_L worsen the

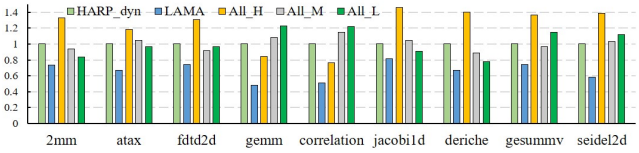

Figure 7: Comparison for Classification Effect

performance by 85%, 58% and 62%, respectively, compared to LAMA. The classification brings the analyzed link preference into the link selection, which makes the performance of LAMA better than All_H, All_M and All_L.

8.4 Overhead Analysis

With the optimized design, the latency of the run-time control is kept as one clock cycle and can be within two cycles even at 400MHz, the highest frequency in HARP2. While HARP$_{opt}$ and HARP$_{dyn}$ also incur several clock cycles overhead. Such low overhead can be compensated by the great benefit of LAMA. Moreover, the bandwidth is not affected by the latency due to the highly pipelined memory access path. The number of ALMs, registers and block memory bits are only increased by 1.9%, 1.3% and 0.3% on average for all the applications with LAMA compared to HARP$_{opt}$, and the resource usages of HARP$_{opt}$ and HARP$_{dyn}$ are basically the same. The LAMA run-time control occupies 3220 ALMs, 1932 registers and 13312 block memory bits, which are less than 1% of the resources. LAMA has low overhead in both latency and resources.

9 CONCLUSION

In this work, we present LAMA, a hybrid memory access management framework that can boost the performance of emerging HARP2-style CPU-FPGA platforms. It combines the link utilization balance, cache benefit improvement and data consistency maintenance together using a static and dynamic coordinated approach. A complete framework with interactive multi-layer software services and hardware IP is provided to support its use. Experiments on the real HARP2 platform show the benefit of LAMA, with a large performance improvement of 34% and low overhead.

REFERENCES

[1] Kristof Beyls and Erik D'Hollander. 2001. Reuse distance as a metric for cache behavior. In *Parallel and Distributed Computing and Systems, IASTED Conf. on.*

[2] Xuhao Chen et al. 2014. Adaptive cache management for energy-efficient GPU computing. In *Microarchitecture (MICRO), Int'l Symp. on.* IEEE.

[3] Liang Feng et al. 2017. A hybrid approach to cache management in heterogeneous CPU-FPGA platforms. In *Computer-Aided Design (ICCAD), Int'l Conf. on.* IEEE.

[4] Liang Feng et al. 2018. CAMAS: Static and dynamic hybrid cache management for CPU-FPGA platforms. In *Field-Programmable Custom Computing Machines (FCCM), Int'l Symp. on.* IEEE.

[5] PK Gupta. 2016. Accelerating datacenter workloads. In *Field Programmable Logic and Applications (FPL), Int'l Conf. on.*

[6] Mazen Kharbutli and Yan Solihin. 2008. Counter-based cache replacement and bypassing algorithms. *IEEE Trans. on Computers* 57, 4 (2008), 433–447.

[7] Chao Li et al. 2015. Locality-driven dynamic GPU cache bypassing. In *Supercomputing (SC), Int'l Conf. on.* ACM.

[8] Chris Okasaki. 1999. Red-black trees in a functional setting. *J. of Functional Programming* 9, 4 (1999), 471–477.

[9] Louis-Noël Pouchet. 2012. Polybench: The polyhedral benchmark suite. *URL: http://www. cs. ucla. edu/pouchet/software/polybench* (2012).

[10] Weikang Qiao et al. 2018. High-throughput lossless compression on tightly coupled CPU-FPGA platforms. In *Field-Programmable Custom Computing Machines (FCCM), Int'l Symp. on.* IEEE.

[11] Jeffrey Stuecheli et al. 2015. CAPI: A coherent accelerator processor interface. *IBM J. of Research and Development* 59, 1 (2015), 7–1.

[12] Xiaolong Xie et al. 2015. Coordinated static and dynamic cache bypassing for GPUs. In *High Performance Computer Architecture (HPCA), Int'l Symp. on.* IEEE.

[13] Hanqing Zeng et al. 2018. A framework for generating high throughput CNN implementations on FPGAs. In *Field-Programmable Gate Arrays (FPGA), Int'l Symp. on.* ACM, 117–126.