Distributed Cooperative Shared Last-Level Caching in Tiled Multiprocessor System on Chip

Preethi P Damodaran, Stefan Wallentowitz, Andreas Herkersdorf
Lehrstuhl für Integrierte Systeme,
Technische Universität München, Germany.
Email: {preethi.parayil,stefan.wallentowitz,herkersdorf}@tum.de

***Abstract*—In a shared-memory based tiled many-core system-on-chip architecture, memory accesses present a huge performance bottleneck in terms of access latency as well as bandwidth requirements. The best practice approach to address this issue is to provide a multi-level cache hierarchy and a suitable cache-coherency mechanism. This paper presents a method to increase the memory access performance in distributed-directory-coherency-protocol based tiled many-core systems. The proposed method introduces an alternate design for the system-wide shared last-level caches (LLC) placed between the memory and the node private caches (NPC). The proposed system-wide shared LLC layer is distributed over the entire network and it interacts with the home directories of specific cache lines. Results from simulating SPEC2000 benchmark applications executed on a SystemC model of the proposed design show a minimum performance improvement of 20-25% when compared to a model without the shared cache layer at the expense of an additional 2% of the total cache memory space (NPC + LLC memory). In addition, the proposed design shows a minimum 7-15% and an average 14-15% improvement in performance in comparison to centralized system-wide shared LLC of equivalent size and dynamic mapped distributed LLC of equivalent size respectively.**

I. Introduction

Among tiled multiprocessor system-on-chip platforms (MPSoC), shared-memory arcitectures represent an emerging trend in complex embedded and consumer hardware designs [1]. In such systems, due to pin-count restrictions, the number of memory carrying nodes are limited and this imposes a performance bottleneck for memory-access. The performance of memory accesses can be improved mainly 1) by reducing the request traversal time or 2) by introducing cache memories [2]. However in MPSoC systems, private caches either at the processor or at the node level introduce cache incoherency and memory inconsistency issues and hence need to implement coherency and consistency management schemes, [2].

In scalable NoC-based MPSoCs which implement directory based cache coherency protocols, distributed-directory approaches outperform the centralized directory scheme as it nullifies the formation of hotspots at the directory. However even in distributed-directory based systems, accesses for evictions, writes and reads of cache lines from the LLC to the memory node still cause a bottleneck. In addition, since the number of memory nodes is finite, these memory nodes can create hotspots if there are a large number of such memory accesses. There are several approaches in the literature to address this performance bottleneck. One of the methods is to introduce cooperation between node-private-caches (NPC) such as in cache-to-cache forwarding [3] or in cooperative caching of NPCs [4]. As these cache models do not carry a system-wide shared LL caching, they require a direct interaction with memory for many read-write replacements outside to the limits of data-cooperation. This leads to memory hotspots. Another method is to use a system-wide shared LL caching such as in [5]–[7]. It can be classified as either statically or dynamically mapped, based on the kind of mapping of memory space to its LLC. In [5], [7], the LLC caches are banked into a few limited number of cache nodes. This causes NPCs in compute nodes to constantly interact with these specific cache nodes during any NPC cache misses, creating a hotspot at the cache nodes. Considering the dynamic shared LLC like in [6], though the LL cache node for a cache line is distributed, they are mapped dynamically. Hence the NPCs require to interact with the home directory to find the LLC node for any cache line. This in effect adds considerable traffic in the system. In addition, in both static as well as dynamic mapped LLCs architectures, there is no cooperation between the home directory and the cacheable LLC node of a particular cache-line. This hence eliminates any possibility to merge cache request/response transactions and coherence management transactions.

978-3-9815370-2-4/DATE14/©2014 EDAA

This paper proposes the concept of distributed cooperative shared caching (DCSC). By optimally positioning the distributed system-wide shared LLC caches and by coordinating the cache layer with the distributed directories, this design provides the ability to locally deduce the system-wide shared LLC node positions along with their respective home nodes. Not only is the size of cache introduced in this design very tiny in comparison to MPSoCs such as Core i7 or Niagara, due to its two main features it is able to minimize the negative effects of memory and LLC hotspots along with reducing the execution time. Hence it is a good candidate for embedded and consumer systems.

The paper is organized as follows. A detailed description of the proposed cache design is provided in section II, following a quantitative analysis in section III. Section IV explains the details of a SystemC model based simulations, results and analysis. Finally section VI concludes the paper.

II. Distributed Cooperative Shared Cache

Distributed Cooperative Shared Cache (DCSC) is a system-wide shared LLC. In a multi-core system containing NPCs with distributed-directory coherence scheme and system-wide shared LLCs, DCSC can be viewed as an alternative improved design for the existing LLC architecture. In systems without a system-wide shared LLC, DCSC is introduced as a new thin cache layer between the NPC and the memory.

A. DCSC Layer

Fig. 1 represents the cache hierarchy of the final system including the DCSC layer along with the node-private caches and

memory nodes. The DCSC cache has the following properties:

a) Any tile can include a DCSC in addition to NPCs, directories, processing elements, IO or memory.
b) The DCSC node for an address is decided by its address space. In Fig. 1, each DCSC can cache only its respective shaded portion of the memory space. The NPCs can access any DCSC over the NoC decided by the address of the request.
c) In the system, only DCSC interacts directly with memory.

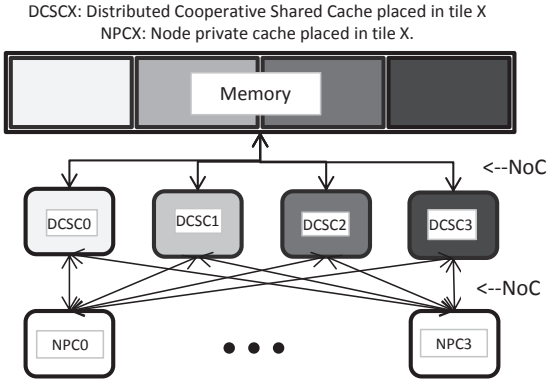

Fig. 1: Cache hierarchy of DCSC

B. DCSC Cooperation

The DCSC works in cooperation with the distributed-directory coherence scheme. The DCSC cache for a specific cache line is placed at a node, at a configurable distance away from the node carrying home directory of the cache line. A cache line to home directory node-translation can be done statically or dynamically. But, home directory node to DCSC node translation is done using the address of memory request. Assuming a continuous numbering of tiles in the NoC system, the maximum distance between a cache line's home directory node and the DCSC cache node is defined as Cooperation Width or Degree of Cooperation (DoC). If an NPC knows the DoC value of the system, home directory node and the memory address to be processed, it can locally calculate the DCSC node for the particular request without any extra hardware control or communication.

In view of the above description, translating a memory address to the corresponding DCSC node may be understood considering the following example system.

1) Memory address division: Any memory address may be divided into 4 sections: A cache line section, the width of which depends on the cache line size, a home directory Node section, the width of which depends on the number of nodes containing directories in the system, and a DCSC Cooperation section, the width of which depends on the degree of cooperation following the address tag section.

2) Translation of home directory node: In case of a static directory translation scheme, home directory node is calculated as {(Cacheline index) mod (Total number of dir-nodes)}.

3) DCSC node translation: The DCSC node (N_{dcsc}) is calculated using the equation below using 1) DoC of the system (DoC), 2) Home directory node (N_{hd}) and 3) DCSC cooperation value(V_{doc}).

$$\{\text{Least significant DoC bits of}(N_{hd} + V_{doc})\} \mod (2^{DoC})$$

Considering an example memory request with Address = 16'hAEC2 raised by a NPC cache with cache line width of 32bytes in a 4x4 NoC structure having a degree of cooperation of 1, N_{dcsc} can be found in below steps.

1. Total number of dir-nodes = 16, $V_{doc} = 1'b1$
2. Cacheline Index = 16'hAEC2 >> 5 = 11'h576
3. N_{hd} = 11'h576 %(16) = 11'h6, 6 is the home node.
4. N_{dcsc} = ({LSB of (11'h6 + 1'b1) } mod 2^1) = 11'h7

Hence DCSC node for the address 16'hAEC2 is node7.

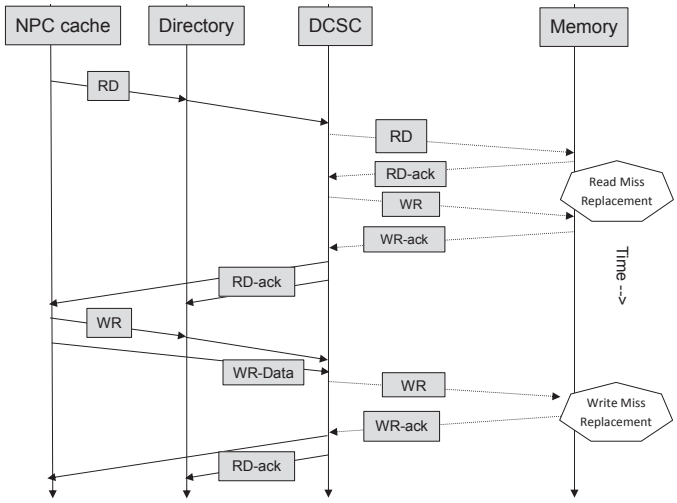

Fig. 2: DCSC Read-Write Operations

The interactions between the NPC, directory, memory and DCSC cache for a case of cache line read and write is provided in the Fig. 2. In summary, the main contribution of this paper is to introduce two main features of DCSC. Firstly, DCSC cache is placed in a node which is at most at DoC distant away from home directory in order to reduce traffic hotspots by distributing the traffic more evenly. As the distance between home directory and DCSC is defined using the requested memory address, NPCs can locally calculate the LLC position, avoiding all traffics required for finding the node carrying the LLC. Secondly, memory requests from a NPC is redirected to DCSC cache by the home directory of the cache line. This way, additional traffic due to directory acknowledgements and request interactions between NPC to LLC are reduced, thereby increasing the performance.

III. Preliminary Analysis

We next present a mathematical model of the proposed cache hierarchy design to analyse the benefits of DCSC quantitatively. It is based on the basic equation for average memory access time (T_{AMA}) in cacheable shared memory systems, i.e.,

$$T_{AMA} = R_h \times T_h + R_m \times T_m \tag{1}$$

In the Eq. 1, the following assumptions are taken. 1) The NPC cache parameters such as R_h, T_h and R_m remains constant for all different models considered here. 2) The term T_m is a system state depended variable, [2]. Considering the architectural models of systems without a system-wide shared LLC such as [3] as Model A, systems with limited distribution based system-wide shared LLC such as [5], [7] as Model B and finally a system with DCSC as Model C, the following can be inferred. The T_m of the NPC depends on multiple factors such as 1) data retrieval

delay (T_d), 2) coherence management delay, (T_c) and 3)NoC traversal delay (T_n), [2]. i.e.,

$$T_m = T_d + T_c + T_n \tag{2}$$

T_c in the Eq. 2 represents the NPC coherence management delays, [3]–[5], [7], which can be assumed to be equal for all three models. The communication delay T_n in the Eq. 2, can be calculated as the number of data traversals(N_{dt}) times average delay for a single traversal (T_{na}), [1]. As long as the average NoC delay per data traversal, T_{na}, remains same for models A, B and C, $T_n(A,B,C) \propto N_{dt}$. Model A requires to converse with home directories for finding the forward stated cache, data requests and other coherence requests. Hence, $N_{dt}(A) > N_{dt}(B,C)$, [3]–[5]. In DCSC architecture, the home directory merges some of the coherence and memory requests from NPC. Also the directory redirects memory requests to a DCSC which is inside DoC limit. Hence $N_{dt}(C) << N_{dt}(A,B)$.

In Eq. 2, the term T_d represents the delays at caches (T_{dc}) and memories (T_{dm}) along with data queueing time (T_{qc}, T_{qm}) at the respective nodes, i.e.

$$T_d = C_h * (T_{dc} + T_{qc}) + M_h * (T_{dm} + T_{qm}) \tag{3}$$

where, C_h represents the probability of data-retrieval from Forward state cache for Model A, while for B and C it represents the cache hit rate of LLC or DCSC. Similarly, M_h represents the rate of NPC to memory interaction for A, whereas it represents cache miss rate in case of Model B and C. When compared to A, models B and C have additional cache layer over the DCSC by virtue of LLC and DCSC respectively [3]–[5]. Due to this additional cache layer, we can expect that the hit rate of B and C is higher compared to A, i.e. $C_h(B,C) > C_h(A)$. Due to the same reason, $M_h(B,C) < M_h(A)$. Because of the hotspots generated at the cache in case of B, it can be inferred that $T_{qc}(C,A) < T_{qc}(B)$. Due to hotspots at memory nodes in case of models A and B, $T_{qm}(C) < T_{qm}(A,B)$. The other factors such as delay for memory (T_{dm}) or cache (T_{dc}) processing delays can be considered equal for all models.

From the description above and from Eq.1 and 3, T_m of Model C w.r.t A becomes

$$\begin{aligned} T_m(C) = T_m(A) - a_1 * (T_{qc}(A) + T_{dc}) \\ - a_2 * (T_{dm} + T_{qm}(A)) - a_3 * N_{dt}(A) * T_{na} \end{aligned} \tag{4}$$

where the reduced cache delay ratio in DCSC w.r.t model A is named as a_1, reduced memory delay ratio as a_2 and reduced traffic ratio as a_3. Similarly naming reduced cache delay ratio in DCSC w.r.t model B as b_1, reduced memory delay ratio as b_2 and reduced traffic ratio as b_3, T_m in DCSC in comparison to Model B is :

$$\begin{aligned} T_m(C) = T_m(B) - b_1 * (T_{qc}(B) + T_{dc}) \\ - b_2 * (T_{dm} + T_{qm}(B)) - b_3 * N_{dt}(B) * T_{na} \end{aligned} \tag{5}$$

where, $a_1 \approx C_h(A) - C_h(C)$
$b_1 \approx C_h(B) - \{C_h(C) * \frac{T_{qc}(C)+T_{dc}}{T_{qc}(B)+T_{dc}}\}$
$a_2 \approx M_h(A) - \{M_h(C) * \frac{T_{qm}(C)+T_{dm}}{T_{qm}(A)+T_{dm}}\}$
$b_2 \approx M_h(B) - \{M_h(C) * \frac{T_{qm}(C)+T_{dm}}{T_{qm}(B)+T_{dm}}\}$
$a_3 \approx \{1 - \frac{N_{dt}(C)}{N_{dt}(A)}\}$ and $b_3 \approx \{1 - \frac{N_{dt}(C)}{N_{dt}(B)}\}$

To understand the implications of Eqs. 4 and 5 and to compare $T_{AMA}(C)$, $T_{AMA}(A)$ and $T_{AMA}(B)$, an example configuration is taken with NPC parameters of $R_h = 0,95, T_h = 15cycles, R_m = 0.05$ and other constants as $T_{dm} = 480cycles, T_{dc} = 15cycles, T_{na} = 40cycles$. Fig. 3 presents the speedup comparisons of models A, B and C.

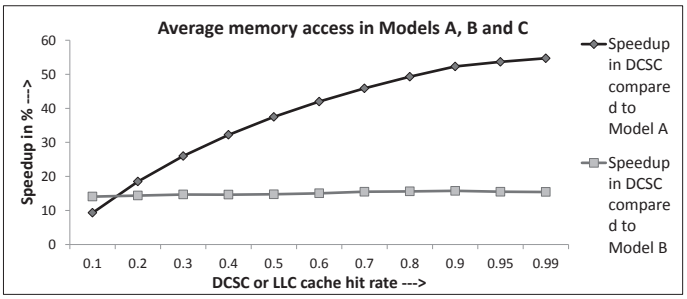

Fig. 3: Average memory access speed up comparison

From Equations 2 to 5 and Fig. 3, it can be observed that for a range of LLC hit rate between 0.1 to 0.99, the memory access time in model C is decreased to 10-60% in comparison to A and is decreased to 10-15% in comparison to B. For in-depth understanding of the environment depended variable delay, a simulation based analysis is conducted further.

IV. Simulations and Results

A. Simulation Setup

The simulation environment utilized for benchmarking the DCSC architecture using SPEC2000 applications is given in Fig. 4. To analyse the effects of the DCSC cache on the performance of the system, four different cycle accurate SystemC architectural models are implemented. All these models consist of a tiled multi-processor system interconnected using a generic NoC implementation, with tiles subdivided as compute nodes or memory nodes according to their respective constituents. The first model (Model 1) implements a cache system without a system-wide shared LLC. Whereas the models 2-4 implements a system with system-wide shared LLC, Model-2:centralized LLC, Model-3:distributed LLC with dynamic mapping and Model-4: DCSC architecture. The architectural view of the SystemC model with DCSC cache is shown in Fig. 4.

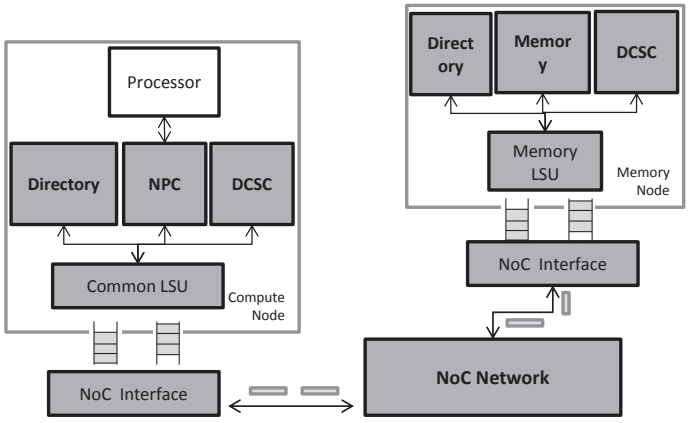

Fig. 4: SystemC model of DCSC cache system

SPEC2000 benchmark applications (such as fft, cholesky, barnes etc.) were scheduled over different number of ALPHA processors (single application in 1-8 processors) by Linux OS using the GEM5 open-source simulator [8]. The memory accesses of these benchmark programs were tapped out to generate trace files. These files were fed to the trace based SystemC processors to carry out the below simulations.

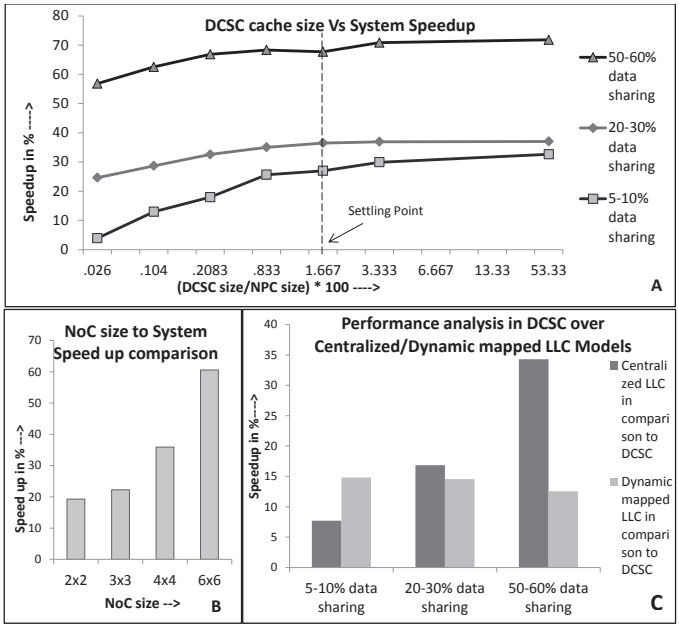

Fig. 5: Simulations on system without LLC vs system with DCSC (Fig. A and B), system with centralized/dynamic mapped LLC vs DCSC (Fig. C)

B. Simulation Results and Analysis

1) Comparison of system without LLC vs system with DCSC: Memory request execution time in model 4 and 1 is compared in three different aspects, 1) effect of data sharing between NPCs, 2) trade-off in memory area to performance and 3) effects of DCSC on system scaling. For the first two comparisons a simulation system was configured with NPC cache of 128kB/tile, DCSC size of 32B/tile(0.03%, i.e. least possible cache size or single cache line size) to 64kB/tile(53.33%, half the size of NPC cache) in a 4x4 NoC tile, with three different scenarios of NPC data share ratios.

From the Fig. 5-A, it can be seen that the performance improvement differs according to the percentage of node-to-node data share. At very low data-share scenarios the DCSC architecture is able to provide a minimum 20-25% of performance enhancement. At high sharing environments the percentage reduction in execution time in DCSC reaches around 60-70%. Now considering the case of a particular data-share scenario in Fig. 5-A, it can be observed that as the DCSC cache sizes increase (implying an increase in cache hit rate) the performance enhancement in execution time also increases, as predicted from the Fig. 3. The gain value settles at a cache size of 2kB (1.667%), as shown in the Fig. 5-A. With an additional cache size of only around 2% in comparison to model 1, DCSC design is able to provide execution performance enhancement of roughly 20% to 75% on varied data-share environments. In order to understand the effects of system scaling on the cache policy, a simulation setup was modified for NoC sizes ranging from 2x2 to 6x6, with a DCSC size of 2kB/tile and an average data sharing ratio of 5-10% as shown in Fig. 5-B. When the system is scaled up between a NoC size range of 2x2 to 6x6, the speedup in DCSC approximately doubled, with an average speedup of 25-40%, in comparison to model 1. Thus, we claim that the DCSC architecture is highly scalable. However, the fact is acknowledged that simulations need to be run for even higher ratios of scaling to fully understand this property.

2) Comparison of system with centralized/distributed system-wide shared LLC vs DCSC: Using the same simulation configurations as above, the speedup enhancements in DCSC is compared with model 2 and 3 (with equal total LLC size) in Fig. 5-C. From the Fig. 5-A and Fig. 5-C it can be observed that models 2, 3 and 4 show higher performance in comparison to model 1. But in comparison to centralized LLC architecture, DCSC model shows a performance enhancement for all cases of data share ratios with a considerable increment in speedup (7.7% to 34%) on increasing the data share ratios. But it can be concluded that even for very small data share scenarios, DCSC can promise 7-15% of performance enhancement in comparison to model 2. Fig. 5-C also represents the execution speed up in DCSC in comparison to dynamically allocated LLC, such as in [6]. It can be observed that DCSC provides around 14-15% performance enhancement in different data-share scenarios, showing a correlation to the quantitative analysis in Fig. 3.

V. Conclusion

This paper focuses on enhancing the memory access performance in tiled multicore systems with distributed-directory cache coherence. The *Distributed Cooperative Shared Cache* architecture introduces two main features which deal with the positioning of the system-wide shared LLC caches and their interactions with node-private caches and directory. The SystemC simulation results show that the DCSC layer can provide a performance improvement of minimum 20-25% for SPEC2000 benchmark applications in comparison to system without LL caching. At the same time, the additional memory added for the DCSC layer is less than 2% of the node-private cache size. The results also show a performance enhancement of minimum 7-15% in comparison to the centralized system-wide shared LLC architectures of equivalent size and an average of 14-15% in comparison to dynamic mapped distributed LLC architectures of equivalent size. Thus, incorporating the DCSC design, can provide a significant improvement in performance.

References

[1] L. Benini and G. D. Micheli, *Networks on chips: A new soc paradigm*, IEEE Computer, vol. 35, no. 1, pp. 70.78, Jan. 2002.

[2] Patterson, David; Hennessy, John (2009) *Computer Organization and Design*, (4th ed.). Morgan Kaufmann. ISBN 978-0-12-374493-7.

[3] US 6922756, Hum, Herbert H. J., James R. Goodman, *Forward state for use in cache coherency in a multiprocessor system*, issued 2005-07-26, assigned to Intel Corporation

[4] Herrero, Enric and González, José and Canal, Ramon, Universitat Politècnica de Catalunya, *Distributed Cooperative Caching: An Energy Efficient Memory Scheme for Chip Multiprocessors*,IEEE Trans. Parallel Distrib. Syst. 23(5),853-861 (2012)

[5] Dongrui Fan, Nan Yuan, Junchao Zhang, Yongbin Zhou, Wei Lin 0004, Fenglong Song, Xiaochun Ye, He Huang, Lei Yu, Guoping Long, Hao Zhang, and Lei Liu.*Godson-T: An Efficient Many-Core Architecture for Parallel Program Executions*, J. Comput. Sci. Technol. 24(6):1061-1073 (2009).

[6] M Lodde, J Flich, ME Acacio. *Dynamic last-level cache allocation to reduce area and power overhead in directory coherence protocols*, Proceedings of Euro-Par Parallel Processing, 2012.

[7] P. Lotfi-Kamran, B. Grot and B. Falsafi, *NOC-Out: Microarchitecting a Scale-Out Processor*, Proceedings of the 45th International Symposium on Microarchitecture, 2012.

[8] Binkert Nathan, Beckmann Bradford, Black Gabriel, Reinhardt Steven K., Saidi Ali, Basu Arkaprava, Hestness Joel, Hower Derek R., Krishna Tushar, Sardashti Somayeh, Sen Rathijit, Sewell Korey, Shoaib Muhammad, Vaish Nilay, Hill Mark D., andWood David A. *The gem5 simulator*, SIGARCH Comput. Archit. News, 39(2):1–7, August 2011.