Reducing Snoop-Energy in Shared Bus-Based MPSoCs by Filtering Useless Broadcasts

Chun-Mok Chung, Jihong Kim
School of Computer Science and Engineering
Seoul National University
Seoul, Korea
{chunmok, jihong}@davinci.snu.ac.kr

Dohyung Kim
Dept. of Computer Science and Engineering
University of California, San Diego
9500 Gilman Dr., La Jolla, CA 92093
dhkim@ucsd.edu

ABSTRACT

In shared bus-based multiprocessor system-on-a-chips (MPSoCs), snoop-based schemes are widely used to maintain cache coherency. However, many of broadcasts are useless because remote caches seldom have the matching blocks and their tag lookups do not supply data. From the energy perspective, such tag lookups consume unnecessary energy and make the system energy wasteful.

In this paper, we propose a broadcast filtering technique to reduce snoop-energy in both of cache and bus. Broadcast filtering is achieved by help of snooping cache and split-bus. The snooping cache checks if matching blocks exist in remote caches before broadcasting a coherency request. If no remote cache has the matching block, it eliminates the broadcast. If broadcasting is necessary, only a part of split-bus is used so that the request is selectively broadcasted only to the remote caches that have matching blocks. Simulation results show that our technique reduces 90%, 50%, and 30% of cache lookups, bus usage, and snoop-energy, respectively, with only 2% of degradation in performance. Our technique reduces more energy than other state-of-the-art techniques.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles—*Cache memories*; C.1.2 [**Porcessor Architecture**]: Multiple Data Stream Architecture (Multiprocessors)—*Interconnection architectures*

General Terms

Algorithms, Design

Keywords

Low-energy cache coherency, Broadcast filtering, MPSoC

1. INTRODUCTION

Advance in silicon technology has made it possible to integrate multiple processors and memories, and multiprocessor system-on-a-chips (MPSoCs) are now widely used in high-performance mobile embedded systems [1, 2]. But higher performance means that processors consume more power. Power consumption is a primary constraint in mobile embedded system design, since mobile embedded systems, such as cellular phones and personal game players, use batteries as their power sources.

In shared bus-based MPSoC environments, snoop-based schemes are widely used to maintain cache coherency. When a local cache requires or modifies data, it broadcasts a coherency request message and remote caches snoop on the broadcast to maintain data consistency. We will define that a *snoop-hit* is incurred if any remote cache has the requested block, and define the *snoop-hit count* as the number of remote caches which have matching blocks. Table 1 shows the snoop-hit count for several SPLASH-2 [12] applications, running on an MPSoC that consists of four processors and each processor has 32-Kbyte L1 cache that is four-way set-associative with 32-byte blocks. MESI protocol [14] is used as the snooping protocol. As the table shows, the ratio that snoop-hit count is zero is generally high. On the average, it reaches to 83.6% of all snooping requests. As many of coherency requests do not find matching blocks in any remote cache, the broadcasts and subsequent remote cache lookups are useless but they still consume energy. If we can know in advance that a snoop-hit will not occur, we do not need to broadcast and can save energy. And if we only broadcast to the remote caches which have matching blocks, we can further reduce the energy consumed by snoop-operations.

Directory-based protocols have the advantage that coherency request is sent only to the remote caches which have the requested data. However, as a coherency request is processed as two bus transactions in directory-based protocols - one to access directory, another to access owner cache, if we simply apply it to shared bus-based MPSoCs, the num-

Table 1: Distribution of snoop-hit count.

	Snoop-hit count (%)			
Application	0	1	2	3
cholesky	94.6	3.9	1.2	0.3
fft	99.9	0.1	0.0	0.0
lu(cont.)	81.2	18.3	0.3	0.2
lu(non-cont.)	56.5	42.7	0.5	0.3
radiosity	100.0	0.0	0.0	0.0
radix	98.6	1.2	0.2	0.0
raytrace	66.3	17.9	9.4	6.4
volrend	71.9	19.0	6.5	2.6
Average	83.6	12.9	2.3	1.2

GLSVLSI'07, March 11–13, 2007, Stresa-Lago Maggiore, Italy.

ber of bus transactions for coherency requests is doubled and the bus energy will be doubled, too. Moreover, if multiple remote caches have the requested data, the number of bus transactions is multiplied by the number of containing remote caches, as the request should be sent to them [14].

In this paper, we propose a broadcast filtering technique to reduce snoop-energy consumed by caches and bus. It detects that no snoop-hit will occur in remote caches before broadcasting and prevents unnecessary broadcasts being sent to remote caches which don't have the requested data. Broadcast filtering is achieved by help of snooping cache and split-bus. When a cache miss happens in the local cache, the snooping cache checks if snoop-hits are detected. If no snoop-hit is detected, the snooping cache filters out the broadcast. If snoop-hits are detected, a part of split-bus is used so that the request is selectively broadcasted only to the remote caches which have matching data. Simulation results show that our technique removes about 90% of cache lookups and 50% of bus usage at the MPSoC containing 16 processors. Energy consumption is reduced to 70% of the baseline model on the average. These results show that our approach is energy efficient and we expect it to be used as an energy efficient cache coherency scheme for the low-power MPSoC design.

The contributions of this paper can be summarized into three aspects. First, we proposed a technique to reduce the cache coherency energy in shared bus-based MPSoCs by filtering useless broadcasts. In our knowledge, it is the first approach to reduce both of cache and bus energy in bus-based MPSoCs. Second, we show an energy-efficient hybrid (snooping + directory) cache coherency scheme for shared bus-based MPSoCs. It is based on snoop-based scheme but uses a directory (snooping cache) to filter out useless broadcasts. Third, we show the excellence of our technique by comparison with other state-of-the-art snoop-energy reduction technique.

The rest of this paper is organized as follows. We describe related works in Section 2. The target MPSoC platform and the proposed broadcast filtering technique will be described in Section 3 and 4, respectively. In Section 5, we evaluate the snoop-energy reduction achieved by our technique through simulations, and we draw conclusions of our research in Section 6.

2. RELATED WORK

To reduce snoop-energy, cache lookup filtering and serial cache lookup have been proposed. Jetty [4] is a small structure attached to each cache. Before cache lookup, Jetty is first checked and it filters out useless cache accesses. It was based on the SMP which has a large private L2 cache. But, Jetty did not save much energy of snooping operations in single-chip multiprocessor systems because of the energy overhead by itself [5]. RegionScout [6] saved more energy than Jetty by using smaller sized filter. Jetty used one entry per cache block, whereas RegionScout used one entry per region, a continuous memory area. This reduced the space and energy costs of the filter. In these mechanisms, request were broadcasted to all remote caches and all the filters are accessed when a broadcast is detected, as there is one filter per cache and each filter only has its own cache information.

Serial snooping [7] and flexible snooping [8] have been proposed as serial cache lookup techniques. Serial snooping targeted a hierarchical bus and flexible snooping a ring-based

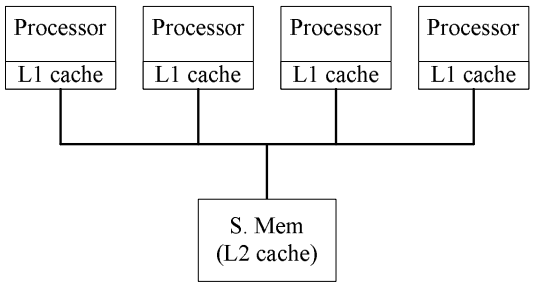

Figure 1: The baseline MPSoC model in case of containing four processors.

bus. Instead of broadcasting the request to all of the processors in parallel, remote caches are checked serially, one by one. This process is based on the assumption that, if a miss occurs in the local cache, it is possible to find the block in a nearby remote cache without checking all remote caches. A snoop packet was sent to the nearest remote cache and, if the requested block was found in it, the snoop transaction ended. Flexible snooping is an expanded version of serial snooping. Serial snooping used the *snoop then forward* scheme, in which snoop transaction was issued to the next cache when the current snoop completed. However, flexible snooping adaptively supplemented the *forward then snoop* scheme and the *forward* scheme, which enhanced snooping performance if a remote cache far away from the requester had the matching block.

3. TARGET MPSOC PLATFORM

Figure 1 represents a baseline MPSoC which consists of four processors and shared memory which we will use to describe our technique, although the technique proposed in this paper are not restricted to a particular number of processors. The baseline model has been used in academic and commercial products, such as ARM MPCore [1] and Hydra [3]. Each processor contains private L1 cache and uses a snoop-based cache coherency protocol. Each L1 cache has duplicated tag to prevent snooping from delaying the processor. The MPSoC may have a shared L2 cache to enhance performance. All processors and shared memory (or L2 cache) communicate with each other through a shared bus.

In snoop-based protocol, if a cache miss happens or the data in local cache is modified, the local cache uses a bus transaction to keep data consistency with remote caches. In case of MESI protocol [14], if a read miss happens in the local cache, the local cache generates a read transaction (BusRd). The local cache broadcasts a block address and a BusRd signal to remote caches and memory. Remote caches snoop the bus transaction and a remote cache containing the requested data supplies the data to the local cache. If no remote cache contains the requested data, memory supplies the data. A write hit is processed as a upgrade transaction (BusUpgr). The local cache notifies to remote caches that data was modified. After snooping the BusUpgr, all remote caches containing the same data block invalidate their cache blocks. If a write miss occurs, the local cache execute a read exclusive transaction (BusRdX). The local cache sends a BusRdX to all remote caches to notify that it will contain that data block exclusively. A remote cache containing the

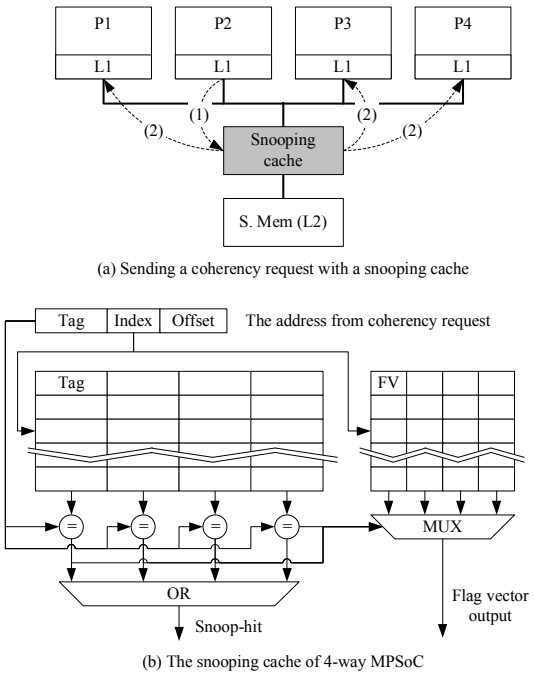

Figure 2: Operation and architecture of snooping cache in the 4-way MPSoC.

requested block transfers the data to the local cache and all remote caches invalidate their cache blocks. If a cache block should be evicted and it is dirty, the local cache writes back the dirty block data into memory using a write-back transaction (BusWB). As the same block does not exist in remote caches, remote caches do not perform any operation after snooping a BusWB.

4. FILTERING USELESS BROADCASTS

In this section, we explain how to detect and filter out useless broadcasts, if none or some of remote caches have the requested data.

4.1 Filtering broadcasts

We use a directory, named as *snooping cache*, to determine if the requested data is contained in remote caches. With the snooping cache, a coherency request is processed in two-hop bus transaction: one hop from the local cache to the snooping cache and another hop from the snooping cache to remote caches. Figure 2 (a) shows the two-hop request: (1) the local cache sends the request to the snooping cache instead of broadcasting it to remote caches. (2) if a snoop-hit is detected in the snooping cache, it broadcasts the request to remote caches. If a snoop-miss is detected in the snooping cache, there is no broadcast and no remote cache lookup.

The snooping cache has the sharing information per data block in L1 caches. It is different from conventional directory which keeps the ownership information per data block in lower-level memory hierarchy. To contain the data sharing information in caches, the snooping cache is organized as a set-associative cache which consists of a tag array and a flag vector(FV) array. It does not have data array. If an MPSoC contains P processors and each processor has a W-way set-associative cache with S sets, the snooping cache is organized as $P \times W$ ways and S sets. As the snooping cache should determine if a requested block exists in other caches, one set of tags contain all tags with the same index in all L1 caches. We chose a conservative way size, because all sets having the same index in L1 caches may have different tags. The length of tag is the same to L1 cache and a flag vector consists of P flag bits. Each flag bit indicates if a corresponding L1 cache contains the same tag. Figure 2 (b) shows the snooping cache organization when an MPSoC consists of four processors ($P = 4$) and each L1 cache is direct-mapped ($W = 1$).

The snooping cache operates like traditional set-associative cache. If the snooping cache receives a coherency request with a block address, it selects a set using an index part of address. After comparing the tag part of address with tags, it outputs a corresponding flag vector and snoop-hit information. To keep tag and flag vector up-to-date, the snooping cache is updated whenever any L1 cache is updated. When a new tag is added to any L1 cache, it is also added to the snooping cache and the corresponding flag bit is set. If the tag is already present in the snooping cache, only the corresponding flag bit is set. When a tag is deleted from any L1 cache, the corresponding flag bit in the snooping cache is cleared. If all the flag bits are cleared, the tag itself is deleted from the snooping cache.

4.2 Selective broadcasting

Let's assume that P2 requests a block which only P3 has in Figure 2 (a). In a shared bus architecture, as there is a snoop-hit in the snooping cache, a coherency request is broadcasted to all remote caches and they look up their tags. Although P1 and P4 do not have the data, they perform unnecessary cache lookups. We know which remote caches contain the requested data from the flag vector in the snooping cache. If the snooping cache selectively broadcasts the request only to P3, we can reduce the number of cache lookups.

To selectively broadcast the request, we adopted a split-bus architecture, an extension of Heish et al's [9] bus segmentation technique. They divided bus lines into two segments and connected frequently communicating components into same bus segment to minimize bus energy. We extended one segmentation model to multiple-segmentation model, where the number of bus segments is adaptively determined according to the number of processors. Figure 3 shows our split-bus architecture applied to the baseline MPSoC. A processor or snooping cache is connected to each bus segment. A splitter connects two adjacent bus segments and transfers signals between them only if it is enabled. The split-bus works like a monolithic bus if all splitters are enabled. An arbiter and flag vector control the ON/OFF of the splitters. If a snoop-hit is detected in the snooping cache, the bits in flag vector are used to ensure that a request is selectively broadcasted only to the remote caches that contain the requested data. Detail operations of split-bus will be described in next subsection. Besides the selective broadcasting, the split-bus reduces the bus usage in snoop-operations.

4.3 Cache coherency operations with broadcast filtering

The operation sequence of bus transaction is modified after applying broadcast filtering. We describe the operation sequence of each bus transaction for cache coherency oper-

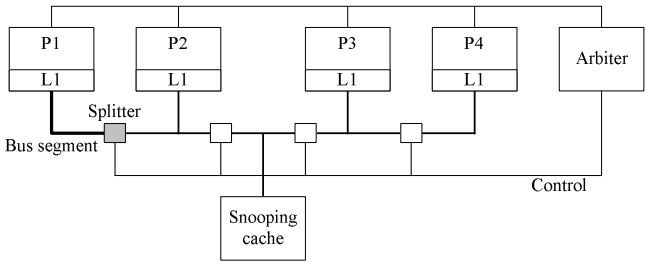

Figure 3: The MPSoC with a split-bus.

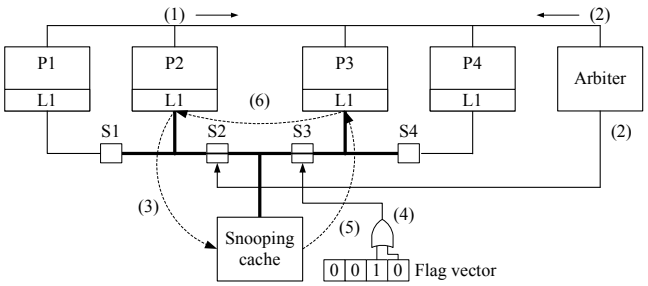

Figure 4: The procedure of BusRd transaction after applying broadcast filtering.

ation to show that broadcast filtering technique is available at all kinds of coherency operations.

BusRd: Figure 4 shows the operation sequence of BusRd, when P2 requests and only P3 has the data. (1) the local cache(P2) requests bus use to the arbiter to perform BusRd transaction. (2) the arbiter permits the bus use. At the same time, it enables the splitters(S2) between local cache and snooping cache. (3) the local cache(P2) sends a request to snooping cache. (4) the snooping cache checks the flag vector. If remote caches contain the requested block, it enables the splitters(S3) corresponding to those remote caches and sets the flag bit corresponding to the local cache. (5) if a snoop-hit is detected, snooping cache broadcasts the request to remote caches. Otherwise, it sends the request to memory. (6) remote caches(P3) snoop the request and supplies the requested data to the local cache(P2).

BusRdX: Its operation is similar to BusRd. It uses BusRdX signal instead of BusRd and the snooping cache clears flag bits corresponding to remote caches in (4), because remote caches invalidate their cache blocks.

BusUpgr: It operates like BusRdX. But, BusUpgr does not perform (6), as the local cache already has the data.

BusWB: After performing (1)∼(3) of BusRd, the snooping cache removes corresponding tag and flag vector, and sends BusWB to memory to write back the data.

From the perspective of performance, cache coherency operation takes additional time - as much time as one bus transaction and one snooping cache latency - by broadcast filtering. It is because coherency request is processed in two-hop bus transaction and the snooping cache is checked to know if remote caches contain the requested data. However, if a snoop-miss is detected in the snooping cache, it does not broadcast the request to remote caches and no cache lookup happens in remote caches. In MESI protocol, memory must wait until it is certain that no cache will supply the requested data before driving the bus [14]. Therefore, the time taken during snooping cache lookup can be offset by removed L1 cache lookup time, if the latencies of snooping cache and L1 tag lookup are similar. So, the performance overhead will be proportional to the snoop-hit ratio. We will evaluate the performance overhead of broadcast filtering by experiment.

5. PERFORMANCE EVALUATION

5.1 Methodology

We evaluated the effect of broadcast filtering through simulation based experiments. The results of simulation show the performances of proposed techniques in the perspective of component activity, energy, and performance. We used CATS [10], an extended version of SimpleScalar [11] for an MPSoC simulation. It supports multiple processors, private cache, shared bus, shared memory, and MESI protocol for cache coherency. We executed programs in SPLASH-2 [12] with various number of processors (2, 4, 8, and 16) and generated traces about cache coherency operations such as request type, address, and cache updates. To estimate the energy consumption during snooping operations, we designed an energy estimator. With the traces generated from CATS, it counts the number of snooping cache accesses, remote cache lookups, active bus segments, and active splitters for each coherency operation. After the analysis of traces, it estimates the energy consumed in each component by multiplying an empirical energy coefficients gained from CACTI [13] and Heish's work [9].

5.2 The activities of cache and bus

To evaluate how many of the unnecessary cache lookups are filtered by our technique, we counted the number of cache lookups and bus segments usage during cache coherency operations before and after using broadcast filtering. Figure 5 shows the number of cache lookups and bus usage during cache coherency operations as a function of processor number. The *Baseline* means the number of cache lookups without broadcast filtering. The *BF_Pn* indicates that the broadcast filtering is applied and n indicates the number of processors. The values are normalized to the baseline result with the same number of processors.

In all applications, moderate number of cache lookups are removed after applying broadcasts filtering, regardless of the number of processors. On the average, the number of cache lookups is reduced to near the 10% of baseline. The reduction ratio is proportional to the snoop-miss ratio of each application, as our technique filters out snooping requests expected to be snoop-miss. So, in *fft, radiosity, radix*, the numbers of cache lookups become near to zero, as their snoop-miss ratio is almost 100% and our technique filtered out most of broadcasts. In most of programs, the relative amount of lookups increases according to the number of processors, except for *lu(non-cont.)*. It is because of the unfiltered useless lookups.

Broadcast filtering also reduces the bus segment use in snooping operations, as it prevents unnecessary broadcastings from using bus segments and makes only necessary bus segments are used to broadcast snooping requests to the remote caches that have the requested block. The more bus usage is reduced as the number of processor increases. On the average, bus segment usage decreases to 52% of baseline model in the MPSoC which consists of 16 processors.

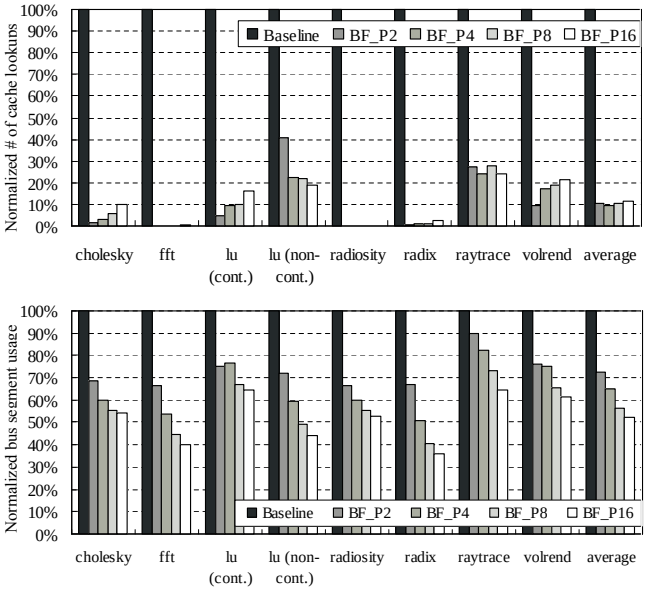

Figure 5: The number of cache lookups and bus usage.

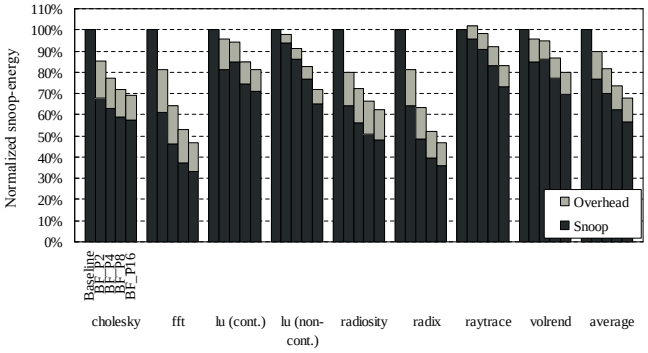

Figure 6: The snoop-energy and overhead by broadcast filtering.

5.3 Energy and performance

Figure 6 shows the energy consumed by snooping operations in the MPSoCs with 2∼16 processors. *Snoop* means the energy consumed by cache and bus to perform coherency operation and *Overhead* indicates the energy consumed in snooping cache and splitters to perform broadcast filtering. All values are normalized to the baseline model with the same number of processors.

Since the energy consumption is proportional to the number of cache lookups and bus usage, the snooping energy decreases after broadcast filtering is applied. The more energy is saved as the number of processors increases, because the number of useless broadcasts increases with more processors. The more energy is saved, with the lower snoop-hit ratio - such as *fft*, *radix*, *radiosity*, because more useless cache lookups and bus usage are eliminated. *raytrace* consumes more energy than the baseline model in case of two processors because of overhead, but energy reduction excels the overhead with more processors. In the MPSoC containing 16 processors, snooping energy is reduced to 47% in maximum, and to 68% on the average, compared to the baseline model. Overhead occupies about 10% of cache coherency energy and the amounts are bigger in the applications having the lower snoop-miss ratios, because the overhead is not reduced, although the snoop-energy is reduced, and its relative amount becomes large.

As previously described, if a snoop-hit is detected, broadcast filtering has the performance overhead due to two-hop request broadcast and snooping cache latency. To evaluate the performance overhead, we calculated the average cache coherency operation latencies (the times from local cache's sending a coherency request to its receiving the data) of two cases, when bus latencies are one cycle (B_1) and 10 cycles (B_{10}). The latencies of data cache and snooping cache are assumed to be one cycle in both cases - we estimated the latencies using CACTI [13] and verified that there was little difference and they can be performed in one cycle. If we assume $L_{Baseline}$ and L_{BF} are the latencies before and after applying broadcast filtering, the performance overhead can be calculated as follow:

$$T_{Overhead} = \frac{(L_{BF} - L_{Baseline}) \times \text{snoop-hit ratio}}{L_{Baseline}}$$

Table 2 shows the performance overhead of broadcast filtering in the MPSoC which has four processors. The first and second columns mean the applications and their snoop-hit ratios. The third and fourth columns represent the overheads of two cases. In all applications, the overhead is small and latencies increase only 3.3% and 2.0%, on the average. The reason of such small overhead can be explained in two characteristics. First, as the snooping cache broadcasts the coherency request to remote caches only when a snoop-hit is detected, the overhead is proportional to the snoop-hit ratio. Second, as the cache coherency latency is mainly occupied by the bus transactions for data copy, the overhead by two-hop request is small.

Table 2: The performance overhead of broadcast filtering.

Application	Snoop-hit ratio (%)	$T_{Overhead}$ (%)	
		B_1	B_{10}
cholesky	5.4	1.1	0.7
fft	0.1	0.0	0.0
lu(cont.)	18.8	3.8	2.3
lu(non-cont.)	43.5	8.7	5.3
radiosity	0.0	0.0	0.0
radix	1.4	0.3	0.2
raytrace	33.7	6.7	4.1
volrend	28.1	5.6	3.4
Average		3.3	2.0

5.4 Comparison with RegionScout

To evaluate the excellence of broadcast filtering to previouse techniques, we compared the activity and energy of snoop-operation. There were Jetty and RegionScout that reduced snooping energy for shared bus based system. As RegionScout showed its superiority to Jetty, we selected

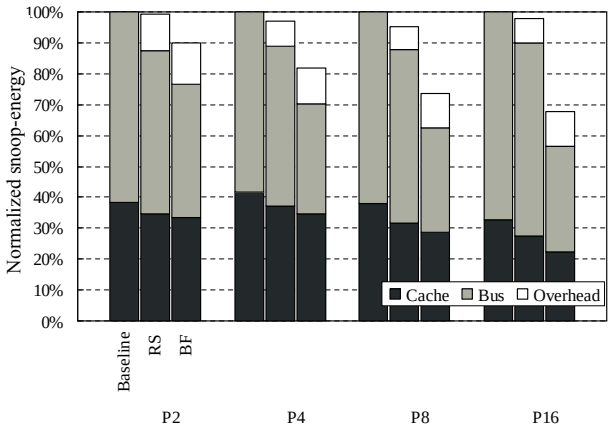

Figure 7: The snooping energy in baseline, with RegionScout(RS), and with broadcast filtering(BF).

RegionScout as our comparison target. We chose the RegionScout which consists of 16KB-Regions, 16-entry direct-mapped NSRTs, and 256-entry CRHs, as this configuration was used for the energy efficiency experiment in original paper [6].

Figure 7 depicts the energy consumed in cache coherency operations with two techniques. It shows the classified energy consumed by cache, bus, and overhead. Overhead means the energy consumed by additional components of each technique. Broadcast filtering (BF) reduces more energy than RegionScout (RS) and the reduction gab becomes larger with more processors, because it reduces not only cache energy but also bus energy. Broadcast filtering reduces more cache energy than RegionScout, because the former uses exact tag information (snooping cache) and the latter uses region-based approximated hashing table (CRH). Also, our technique reduces more bus energy than RegionScout. Because it uses a part of split-bus, when a snoop-hit count is less than the number of remote caches. However, RegionScout broadcasts the coherency request to all remote caches and uses all bus lines.

6. CONCLUSIONS

We have proposed a broadcast filtering technique to reduce snoop-energy consumed by cache and bus. It detects a snoop-miss before broadcasting and prevents unnecessary broadcasts being sent to remote caches which do not have the requested data. The broadcast filtering technique is achieved by a snooping cache and a split-bus. The snooping cache checks if a snoop-hit is detected. If a snoop-miss is detected, the snooping cache filters out the broadcast. If a snoop-hit is detected, a part of split-bus is used so that the request is selectively broadcasted only to the remote caches which have the matching data.

Simulation results show that our technique removes about 90% of cache lookups and 50% of bus usage at the MPSoC containing 16 processors. Due to the activity reduction, snoop-energy is reduced to about 70% of the baseline model on the average. These results show that our approach is energy efficient and we expect it to be used as an energy efficient cache coherency scheme for the low-power MPSoC design.

7. ACKNOWLEDGEMENTS

This work was supported in part by MIC & IITA through IT Leading R&D Support Project, by the Brain Korea 21 Project, and by the MIC, Korea, under the ITRC support program supervised by the IITA. The ICT at Seoul National University provided research facilities for this study.

8. REFERENCES

[1] J. Goodacre and A. N. Sloss, "Parallelism and the ARM instruction set architecture," *IEEE Computer,* July 2005.

[2] D. Courtright, "MIPS32 M4K core for multi-CPU applications," *Embedded Processors Forum,* April 2002.

[3] L. Hammond, B. A. Hubbert, M. Siu, M. K. Prabhu, M. Chen, K. Olukotun, "The Stanford Hydra CMP," *IEEE Micro,* March-April 2000.

[4] A. Moshovos, G. Memik, B. Falsafi, and A. Choudhary, "Jetty: filtering snoops for reduced energy consumption in SMP servers," *Proc. of the 7th International Symposium on High–Performance Computer Architecture,* January 2001.

[5] M. Ekman, F. Dahlgren, and P. Stenström, "Evaluation of snoop-energy reduction techniques for chip-multiprocessors," *Proc. of the First Workshop on Duplicating, Deconstructing, and Debunking,* May 2002.

[6] A. Moshovos, "RegionScout: exploiting coarse grain sharing in snoop-based coherence," *Proc. of the 32nd International Symposium on Computer Architecture,* June 2005.

[7] C. Saldanha and M. Lipasti, "Power efficient cache coherence," *Workshop on Memory Performance Issues, in conjunction with ISCA,* June 2001.

[8] K. Strauss, X. Shen, and J. Torrellas, "Flexible snooping: adaptive forwarding and filtering of snoops in embedded-ring multiprocessors," *Proc. of the 33rd international Symposium on Computer Architecture,* June 2006.

[9] C. T. Heish and M. Pedram, "Architectural energy optimization by bus splitting," *IEEE Transactions on Computer–Aided Design of Integrated Circuits And Systems,* April 2002.

[10] D. Kim, S. Ha, and R. Gupta, "CATS:Cycle Accurate Transaction-driven Simulation with Multiple Processor Simulators," *Proc. of Design Automation and Test in Europe,* April 2007.

[11] D. Burget and T. Austin, "The SimpleScalar tool set version 4.0," *http://www.simplescalar.com/v4test.html.*

[12] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: characterization and methodological considerations," *Proc. of the 22nd Annual International Symposium on Computer Architecture,* June 1995.

[13] P. Shivakumar and N. P. Jouppi, "CACTI 3.0: an integrated cache timing, power, and area model," *WRL Research Report 2001/2,* August 2001.

[14] D. E. Culler, J. P. Singh, and A. Gupta, *Parallel computer architecture: a hardware/software approach,* Morgan Kaufmann Publishers, 1999.