Cache Analysis and Software Optimizations for Faster On-Chip Network Simulations

Khyamling Parane, Prabhu Prasad B M and Basavaraj Talawar
SPARK Lab, Department of Computer Science and Engineering, National Institute of Technology Karnataka, India.
{khyamlingcs15fv05, prabhu.cs15f10, basavaraj}@nitk.edu.in

Abstract—Fast simulations are critical in reducing time to market in CMPs and SoCs. Several simulators have been used to evaluate the performance and power consumed by Network-on-Chips. Researchers and designers rely upon these simulators for design space exploration of NoC architectures. Our experiments show that simulating large NoC topologies take hours to several days for completion. To speedup the simulations, it is necessary to investigate and optimize the hotspots in simulator source code. Among several simulators available, we choose Booksim2.0, as it is being extensively used in the NoC community.

In this paper, we analyze the cache and memory system behaviour of Booksim2.0 to accurately monitor input dependent performance bottlenecks. Our measurements show that cache and memory usage patterns vary widely based on the input parameters given to Booksim2.0. Based on these measurements, the cache configuration having least misses has been identified. We also employ thread parallelization and vectorization to improve the overall performance of Booksim2.0. The OpenMP programming model and SIMD are used for parallelizing and vectorizing the more time-consuming portions of Booksim2.0. Speedups of 2.93x and 3.97x were observed for the Mesh topology with 30 × 30 network size by employing thread parallelization and vectorization respectively.

Keywords: Network-on-Chip, On-chip network simulation, Booksim, Cache Behaviour, Performance Profiling, Parallelization, Vectorization.

I. Introduction

The Network-on-Chip (NoC) is now an integral component in Multiprocessor System-on-Chips (MPSoCs) and in Chip Multiprocessors (CMPs)[1]. The communication time can influence the total turnaround time of the application significantly [2]. NoC researchers have relied on cycle-accurate power and performance simulators (viz. Orion[3], Garnet[4], Noxim[5], SICOSYS[6], Booksim[7]) to explore the microarchitectural design space of on-chip networks. Amongst these, Booksim2.0 has emerged as one of the prominent NoC performance analysis tools. Booksim2.0 offers network parameters such as topology, routing algorithm, flow control, and router microarchitecture, including buffer management and allocation schemes as input parameters for simulating NoC architectures. Simulating large NoC architectures take days together to complete. Hence, there is a need for fast design space exploration of NoC architectures which help designers to reduce the time and effort spent in development of common on-chip framework. From Figure 1, it can be observed that execution time of Booksim2.0 varies as topology size increases from 6 seconds to 10 days simulating 4x4 and 54x54 NoC architectures of Mesh topology. With increasing dimension from 2 to 3 and 4, the time taken will be even more. The increase in execution time could be because of cache behaviour and the way memory is accessed. To analyze the cache behaviour and memory access patterns, we use profiling methodology.

Profiling Booksim2.0 will reveal the dependence of the cache and memory behaviour on the input data. This paper attempts to perform a detailed analysis of the simulator over a wide range of inputs. Valgrind[8] is used to map the cache and memory usage patterns of Booksim2.0. We use Cachegrind tool of Valgrind framework for profiling Booksim2.0. Based on profiling, we identify the best cache configuration in which the cache misses are minimum and memory access patterns of Booksim2.0 which will help improve the performance.

In order to reduce the execution time of Booksim2.0, optimization methodologies such as vectorization and thread parallelization are employed. The OpenMP programming model is used for parallelizing and vectorizing the source code of Booksim2.0.

Rest of the paper is organized as follows: Section II introduces performance optimization strategies driven by input based, cache based and memory usage based ap-
plication profiling. The experimental setup is detailed in Section III. The results and inferences from profiling studies are illustrated in Section IV. The paper concludes in Section V.

II. Related Work

In this section, we introduce state-of-the-art work in the area of NoC simulation, tools used for profiling software applications and software optimization techniques to improve the performance of applications. Optimizations based on program input, cache access behaviour and memory reference patterns have been listed.

A. Network on Chip Simulators

To explore the microarchitectural design space of NoCs, researchers are relied on simulators to evaluate the power and performance. SICOSYS [6] is a general-purpose interconnection network simulator that allows the modeling a wide variety of message routers in a precise way. The parameters such as traffic pattern, applied load, message length etc., can be provided as input for simulation. Noxim [9], is another NoC simulator which is implemented in SystemC. Booksim2.0 [7] is a cycle-accurate simulator. It is flexible in terms of modeling network components. A large set of network parameters which are configurable such as topology, routing algorithm, flow control and router microarchitecture are implemented. Orion [3] a set of architectural power models for on-chip interconnection routers. A classic five-stage pipelined router with virtual channel flow control has been modeled in GARNET [4].

B. Profiling Based on Program Input


C. Profiling for Cache Performance

Prefetching and Profiling can help computing systems better mitigate performance losses due to limited cache bandwidth. Cache profiling can improve program performance by focusing a programmer’s attention on problematic code sections and providing insight into appropriate program transformations. Several proposals exist to use profile based application-level knowledge to manage the contents of caches[13]. The Cachetor[14] run-time profiling tool identifies and reports operation that generates invariant data values. Cachetor uses dynamic dependence profiling and value profiling to expose caching opportunities to improve program performance. The CProf cache profiling system[15] lets programmers identify hot spots by providing cache performance information at the source line and data-structure level.

D. Performance improvement of Applications

The performance of an application can be improved by using techniques such as vectorization, threading, [16] vectorizes operations in important program loops to improve overall resource utilization, allowing for software pipelines with shorter initiation intervals.

E. Motivation

As a prior step in reducing the time taken to simulate the large sized network topologies without compromising the accuracy of results, we intend to employ software optimization techniques which are less expensive compared to other available options such as using hardware to perform the same task.

III. Profiling, Performance Optimization Tools and Experimental methodology

Simulating the NoC architecture for large size topology consume days together to complete. To achieve speedup in simulating these large NoC architectures, there is a need to investigate and optimize the Hotspots in the simulators. We use profiling to measure the application performance, identify Hotspots and diagnose potential problems. We use Valgrind[8] for profiling Booksim2.0. Cachegrind, one of the tools of Valgrind is used to simulate the behaviour of a program with the cache hierarchy and branch predictor of the system. Cachegrind simulates a system with independent first-level (L1) instruction and data caches(I1 and D1), backed by a unified last-level cache(LL). We also use Callgrind, another tool of Valgrind

<table>
<thead>
<tr>
<th>Network Configuration Input to Booksim2.0</th>
<th>System Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topology Type</td>
<td>Mesh and Torus</td>
</tr>
<tr>
<td>Network size</td>
<td>4 × 4, 6 × 6, ......, 30 × 30</td>
</tr>
<tr>
<td>Traffic Pattern</td>
<td>Uniform random</td>
</tr>
<tr>
<td>Number of Virtual Channels</td>
<td>8</td>
</tr>
<tr>
<td>Virtual Buffer Size</td>
<td>8</td>
</tr>
<tr>
<td>Packet Size</td>
<td>20 flits</td>
</tr>
<tr>
<td>Sample Period</td>
<td>1000 cycles</td>
</tr>
<tr>
<td>Maximum Number of Samples</td>
<td>10^6</td>
</tr>
<tr>
<td>Latency Threshold</td>
<td>10^9</td>
</tr>
<tr>
<td>Injection Rate</td>
<td>0.005</td>
</tr>
<tr>
<td>Routing Algorithm</td>
<td>Dimension Order Routing</td>
</tr>
</tbody>
</table>

TABLE I

Experimental Setup Details
to record the call history of functions in Booksim2.0. KCachegrind, GUI based tool which is used for identifying the Hotspots of Booksim2.0. Employing these tools, the best cache configuration in which the cache misses are minimum is identified.

Further, vectorization and thread parallelization techniques are applied in order to improve the performance of Booksim2.0. We use the Intel Advisor suite [17] to identify the top time-consuming loops of Booksim2.0. Based on these analysis, we employ OpenMP programming model to parallelize the top time-consuming loops of Booksim2.0.

### A. Experimental methodology

The cache design and Booksim2.0 configuration parameters considered for experiments in this work are shown in Table I.

Various cache configuration as shown in Table II are simulated using Cachegrind tool of Valgrind to analyse the cache behaviour and memory usage of Booksim2.0 considering different topology size. Based upon these analysis, the best cache configuration was identified.

Further, techniques such as vectorization and thread parallelization are employed to speedup the simulation execution time of Booksim2.0.

### IV. Results and Discussion

#### A. Identifying Best Cache Configuration

The analysis of cache performance with different cache and topology sizes are studied. We consider 12 different cache configurations for First level Instruction cache (I1), First Level Data cache (D1) and Last Level Cache (LL) for analyzing the effect of cache size, block size and associativity as shown in Table II. Booksim2.0 simulations were run for 2D Mesh topology of sizes ranging from $4 \times 4$, $6 \times 6$, ..., $30 \times 30$ (14 experiments).

Cache misses are classified as Compulsory, Capacity and Conflict misses. The cache performance can be improved by reducing these misses. The compulsory misses can be minimized by increasing the block size. But, this may lead to increase in conflict misses. The larger associative cache can be employed in order to minimize conflict misses. As the cache size increases, the capacity misses will be minimized as larger caches are available to store the program data. In Figures 2 and 3, the values are obtained by computing the average of the MPKIs of 14 different network sizes considering all the cache configurations as shown in Table II.

In Figures 2 and 3, each bar represents particular cache configuration, the value was obtained by averaging the MPKI of 14 experiments of Mesh topology from $4 \times 4$, $6 \times 6$, ..., $30 \times 30$ network size. All other values were computed in similar way.

#### 1) L1 Instruction (I1) cache Analysis: Effect of Cache size on I1 cache misses:

From Figure 2, it can be seen that I1 cache misses were reduced by 30.3% when we increase the cache size from 32KB to 64KB for 2-way, 32B line I1 cache. Considering 64B cache line for the same configuration, the cache misses reduced by 22.47%.

As shown in Table III, 2.73% to 45.24% reduction of misses were observed for all other cache configurations when we are increasing I1 cache size from 32KB to 64KB.

#### Effect of Associativity on I1 cache misses:

From Figure 2, when we move from 2-way to 4-way considering 32KB I1 cache with 32B line size, the misses were reduced by 59.45%. When we move from 2-way to 8-way, the misses reduced by 73.87%.

A reduction of 56.40% to 74.00% was observed by replacing 2-way I1 cache by corresponding 8-way I1 cache as shown in Table III. Conflict misses are reduced when we increase the associativity from 2 to 4-way and 2 to 8-way, as we can accommodate more blocks in the set.

#### Effect of Cache line size on I1 cache misses:

It can be seen from Figure 2 that, for 2-way 32KB I1 cache, by increasing the cache line size from 32B to 64B, the misses reduced by 30.33%. This is due to good spatial locality of reference.

Reduction of misses from 32.7% to 91.8% was observed for all other cache configurations when we are increasing I1 line size from 32B to 64B.

#### 2) L1 Data (D1) cache Analysis: Effect of Cache size on D1 cache misses:

The increase in size of D1 cache in an incremental manner yields incremental increase of D1 cache performance. From Figure 3, for 2-way D1 cache with 32B line, when we move from 32KB to 64KB...
**TABLE II**

Different L1 Instruction (I1), L1 Data (D1) and Last Level (LL) Cache configurations used in experiments

<table>
<thead>
<tr>
<th>Sl No.</th>
<th>I1 and D1 cache sizes</th>
<th>Associativity</th>
<th>Block Sizes</th>
<th>LL cache sizes</th>
<th>Associativity</th>
<th>Block Sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32KB/32KB</td>
<td>2,4,8-Way</td>
<td>32B,64B</td>
<td>512KB</td>
<td>4,8,16-Way</td>
<td>32B,64B</td>
</tr>
<tr>
<td>2</td>
<td>64KB/64KB</td>
<td>2,4,8-Way</td>
<td>32B,64B</td>
<td>8MB</td>
<td>4,8,16-Way</td>
<td>32B,64B</td>
</tr>
</tbody>
</table>

**TABLE III**

Effect on misses due to various I1 and D1 cache configurations

<table>
<thead>
<tr>
<th>Reduction in I1 Misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configurations</td>
</tr>
<tr>
<td>32KB vs 64KB</td>
</tr>
<tr>
<td>2-way vs 8-way</td>
</tr>
<tr>
<td>32B vs 64B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reduction in D1 Misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configurations</td>
</tr>
<tr>
<td>32KB vs 64KB</td>
</tr>
<tr>
<td>2-way vs 8-way</td>
</tr>
<tr>
<td>32B vs 64B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reduction in LL Misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configurations</td>
</tr>
<tr>
<td>512KB vs 8MB</td>
</tr>
<tr>
<td>4-way vs 16-way</td>
</tr>
<tr>
<td>32B vs 64B</td>
</tr>
</tbody>
</table>

**TABLE IV**

Effect on misses due to various Last Level (LL) cache configurations

<table>
<thead>
<tr>
<th>Reduction in LL Misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configurations</td>
</tr>
<tr>
<td>512KB vs 8MB</td>
</tr>
<tr>
<td>4-way vs 16-way</td>
</tr>
<tr>
<td>32B vs 64B</td>
</tr>
</tbody>
</table>

**Effect of Cache line on D1 cache misses:** From Figure 3, the misses reduced by 19.97% on increasing the cache line from 32B to 64B for 2way, 32KB D1 cache.

Reduction of misses from 21.16% to 22.00% was observed for other cache configurations when we are increasing the cache line from 32B to 64B as shown in Table III. Increasing the cache line, more data can be fetched from LL cache into D1 cache. This reduces the compulsory misses.

Based on the above observations, maximum cache miss reduction of 27.29% can be seen when moving from 2way, 32KB D1 cache with 32B line to 4way, 64KB D1 cache with 64B line.

Comparing I1 and D1 cache analysis, the reduction observed in I1 cache is much more than D1 cache as I1 caches exhibit better spatial locality of reference.

Based on the above analysis of I1 and D1 caches, medium associative, higher cache size with larger cache line performs better than all other cache configurations. Our experiments show that 4-way, (64KB+64KB) L1 cache with 64B line L1 configuration is appropriate for running the Booksim2.0 simulations.

**3) Last level (LL) cache Analysis:** The last level cache size of 512KB, 4MB and 8MB were used to identify the appropriate LL cache configuration. We have considered the 512KB and 8MB LL cache size for last level cache analysis as the changes in cache misses can be observed more clearly.

In Figure 4, the values are obtained by computing the average of the MPKIs of 14 different network sizes considering all the cache configurations as shown in Table II.

**Effect of Cache size on LL cache misses:** From Figure 4, for 4-way LL cache with 32B line, when we move from 512KB to 8MB cache, the misses were reduced by 4.13%. Similarly, for 64B cache line, 6.67% reduction in misses was observed. As shown in Table IV, 2.54% to 65.90% reduction of misses were observed for all other cache configurations when we are move from 512KB to
8MB of LL cache size.

Effect of Associativity on LL cache misses: From Figure 4, increasing the associativity from 4-way to 8-way for 512KB LL cache with 32B line, the misses reduced by 2.48%. When we move from 4-way to 16-way, the misses reduced by 15.7%. Reduction of misses from 0.86% to 73.21% was observed for all other cache configurations when we move from 4 to 8-way and 4 to 16-way respectively as shown in Table IV. The conflict misses arising from blocks of main memory mapping to the same position in the cache can be reduced when moving from 4-way to 16-way.

Effect of Cache line on LL cache misses: In Figure 4, the misses reduced by 0.83% with increasing the cache line from 32B to 64B for 4-way, 512KB LL cache. As seen from Table IV, reduction of misses from 0.87% to 65.11% was observed for other cache configurations. Increasing the cache line, more data can be fetched from main memory to LL cache. This reduces the compulsory misses.

Based on the above analysis of LL cache, higher associative, higher cache size with larger cache line performs better than all other cache configurations. In our experiments 16-way, 8MB LL cache with 64B line LL cache configuration is appropriate for running the Booksim2.0 simulations.

By all these observations, it can be inferred that 4-way, (64KB+64KB) L1 cache with 64B cache line and 16-way 8MB LL cache with 64B line is the optimal cache configuration for running Booksim2.0.

From these experiments, it is evident that increasing cache configuration improves the performance of Booksim2.0 simulations.

B. Improving Performance of Booksim2.0

We employ the techniques such as vectorization, multi threading and use OpenMP programming models to parallelize the portions of the Booksim2.0 source code. The most time-consuming loops were identified using Intel Advisor tool.

Memory access patterns have an impact on improving the performance of an application. We use Intel vectorization tool to identify the stride access pattern of Booksim2.0. 17% of memory instructions were unit stride, 34% of memory instructions were fixed non-unit stride and 49% of memory instructions were variable stride, after annotating Singlesim method of TrafficManager class of Booksim2.0. Based on these observations, we improve the performance of Booksim2.0 by changing unaligned to aligned memory access. The compiler directive as shown below is inserted in the source files of Booksim2.0 to change unaligned to aligned memory access.

```
#pragma omp simd aligned()
```

Further, we use OpenMP programming model and SIMD to parallelize and vectorize the most time-consuming portions of Booksim2.0. We compare execution times of sequential code with parallel code considering the different network topology size of Booksim2.0 with Mesh topology as shown in Figure 5. The speedup of 2.93x as shown in Figure 6 was achieved by parallelizing the
sequential code of Booksim2.0 using OpenMP constructs considering $30 \times 30$ network size of Mesh topology. 1.07x to 3.0x speedup was observed for all other Mesh topology size.

Also, the SIMD construct was used with OpenMP programming model to achieve fine grain parallelization. By using SIMD with OpenMP model, the performance improvement of 3.97x was observed for 4×4 sized Mesh topology. And, speedup from 2.64x to 3.69x was observed for all other Mesh topology size. The overall traffic statistics of Booksim2.0 observed in both the executions matched each other. The pragma constructs used to parallelize and vectorize the code are shown below:

```c
#pragma omp parallel for
#pragma omp parallel simd for
```

From Figures 5 and 6, it can be inferred that parallelization and vectorization reduce the execution time of 30×30 network of Mesh topology from 60 to 14 minutes and 12 minutes by employing parallelization and vectorization respectively using OpenMP programming model.

V. Conclusions

In this paper, we have shown that profiling and software optimization techniques can improve the performance of the Booksim2.0 Network-on-Chip simulator. Profiling Booksim2.0 help us understand the effect of various input parameters such as topology size, traffic pattern, number of virtual channels, packet size, sample period, routing algorithm and injection rate on cache and memory access behaviour.

We have considered the cache design parameters such as cache size(LL:32KB to 64KB and LL:512KB, 4MB & 8MB), associativity(L1:2, 4 & 8-way and LL:4, 8 & 16-way), cache line size(L1&LL: 32B, 64B), replacement algorithm and cache write policy for the purpose of our experiments. We profile Booksim2.0 simulations by incorporating the above mentioned cache configurations and considering 14 various network sizes from $4 \times 4$, $6 \times 6$, $...$, $30 \times 30$ of Mesh and Torus topologies.

We analyse cache miss trends in first level instruction cache (I1), first level data cache (D1) and last level cache (LL). By observing the experimental results based on average MPKI, least number of misses were seen in 4-way, 64KB+64KB L1 cache and 16-way, 8MB LL cache with 64B cache line. We observed that this cache configuration suffers least MPKI for all the network sizes of the Mesh topology. Similarly, for the Torus topology 8-way, (64KB+64KB) L1 cache and 16-way, 8MB LL cache with 64B cache line.

Vectorization and parallelization were employed to further improve the performance of the Booksim2.0 simulator. By using the Intel advisor tool, we identified the top time-consuming loops in Booksim2.0 source code. We use the OpenMP programming model to parallelize the time-consuming loops in the simulator. Speedups of 1.07x to 2.93x were observed over various Mesh topology size. Speedups of 2.88x to 3.97x were observed when we used the OpenMP+SIMD model for parallelization and vectorization of the more time-consuming loops.

We have successfully reduced the simulation time of Booksim2.0 from 60 to 12 minutes for Mesh topology of network size 30×30. We observed that average simulation time was reduced by 67.31% for all network sizes of Mesh topology using the optimizations presented in this work.

VI. Acknowledgment

This work was supported by the Department of Science and Technology under the grant DST-SERB YSS/2015/000196 project and Visvesvaraya PhD scheme under Deity, Government of India.

REFERENCES


