# To CMP or not to CMP: Analyzing Packet Classification on Modern and Traditional Parallel Architectures

Randy Smith, Dan Gibson, and Shijin Kong
Department of Computer Sciences, University of Wisconsin–Madison
1210 W. Dayton Street, Madison, WI, U.S.A., 53706
{smithr,gibson,krobin}@cs.wisc.edu

## ABSTRACT

Packet classification is central to modern network functionality, yet satisfactory memory usage and performance remains elusive at the highest speeds. The recent emergence of low-cost, highly parallel architectures provides a promising platform on which to realize increased classification performance. We analyze two classic algorithms (ABV and HiCuts) in multiple parallel contexts. Our results show that performance depends strongly on many factors, including algorithm choice, hardware platform, and parallelization scheme. We find that there is no clear "best solution," but in the best cases hardware constraints are mitigated by the parallelization scheme and vice versa, yielding near-linear speedups as the degree of parallelization increases.

**Categories and Subject Descriptors:** C.1.4 Parallel Architectures: Chip Multiprocessors; C.2.6 Routers: Packet Classification

**General Terms:** Algorithms, performance

**Keywords:** Packet classification, chip multiprocessors

## 1. INTRODUCTION

Packet classification is a core component to modern network functionality and is used for a variety of services including routing, firewalls, and quality of service. Even so, achieving satisfactory performance with acceptable memory usage remains elusive at the highest speeds due to the inherent time-space tradeoffs involved, and new techniques continue to be proposed to advance the state of the art.

The recent emergence of Chip Multiprocessors (CMPs) and other low-cost, highly parallel architectures provide a promising platform for realizing increased performance for packet classification and for other common tasks in the forwarding path. This naturally leads to the question: what are the consequences–both positive and negative–of performing classification on parallel architectures? In this work, we examine the tradeoffs involved in parallel classification. We begin with two classic but distinct classification algorithms–Aggregated Bit Vector (ABV) [1], which uses a divide-and-conquer approach to performing classification, and HiCuts [2], which constructs a decision-tree-based classifier. We construct both data- and control-parallel variants of these algorithms and evaluate them on both chip multiprocessor (CMP) and symmetric multiprocessor (SMP) hardware.


*ANCS'07*, December 3–4, 2007, Orlando, Florida, USA.
ACM 978-1-59593-945-6/07/0012.

With regard to parallel hardware, we focus our work on the inherent performance differences between two basic platforms. *Symmetric Multiprocessors* (SMPs) employ multiple processing *chips* to provide a parallel execution environment. Each such chip is typically allocated a single processor and a large, private cache. In contrast, *Chip Multiprocessors* (CMPs) integrate multiple processors onto a single chip, commonly sharing cache resources as well. While SMPs and CMPs present identical programming interfaces, applications will see very different performance characteristics due to inherent, application-dependent performance implications associated with resource sharing and privatization.

The SMPs in our study can attain superior single-threaded performance as compared to a CMP, due to the effect of on-chip and near-chip resource privatization. In contrast, the CMP shares a second-level (on-chip) cache with all other processors, enabling very fast inter-thread communication at a cost of contention for space and bandwidth in the shared portions of the memory hierarchy.

## 2. PARALLEL CLASSIFIERS

We parallelized ABV and HiCuts using both data- and control-parallel approaches. The data-parallel scheme fully replicates the classification algorithm on each processor. Initialization tasks that construct data structures such as tries, bitmaps, and decision trees are performed prior to replication, and a single copy of these is shared among all instances of the algorithm. During operation, packet headers are placed in a shared queue guarded by a mutual exclusion lock that arbitrates access by the processors.

The data-parallel approach minimizes data dependencies between processors, since communication is limited to contention for exclusive access to the shared queue's lock. When locking costs are small, linear speed-up can be observed as the number of processors increases up to a point. However, as shown in [3], the use of a single lock restricts scalability to a small number of processors, although this effect is mitigated somewhat by the cheap locking costs of CMPs.

In contrast, control-parallel approaches divide the *algorithm* into multiple components and assign these components to distinct processors. We use a pipelined scheme, which is a special case restricted to an "assembly-line" model for data movement. Here, classification is partitioned into multiple stages, processors are connected together in sequence, and each processor is assigned a specific stage to execute. Small queues (with exclusion locks) reside between adjacent processors to decouple read and write operations.

The chief advantage of the pipelined approach is reduced lock contention, since each lock is shared by at most two

| Server | Sun T2000 | SunFire v880 |
|---|---|---|
| # Procs | 8 Ultra-SPARC-T1 | 8 Ultra-SPARC-III+ |
| Threads | 32 | 8 |
| Chips | 1 | 8 |
| L1-I | 16 KB | 32 KB |
| L1-D | 8 KB | 64 KB |
| L2 | 3 MB onchip,shrd | 8 MB offchip,prvt |
| Inter-connect | Crossbar | Shared Bus |

**Figure 2: Target machine specs**

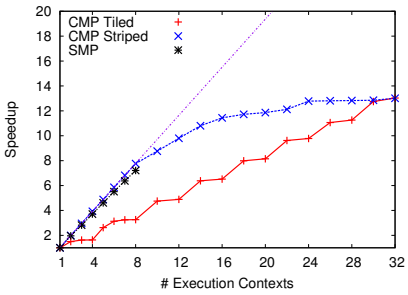


**Figure 3: Data Parallel ABV**

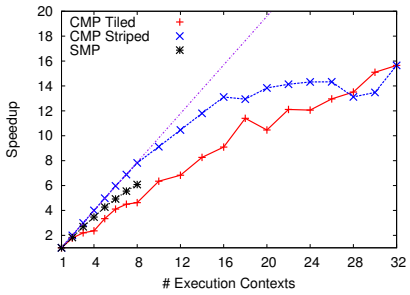


**Figure 4: Data Parallel HiCut**

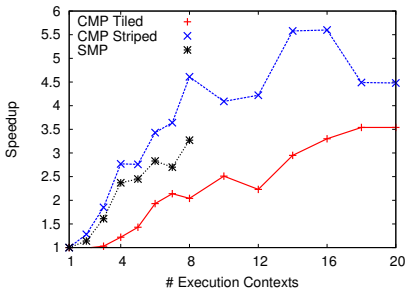


**Figure 5: Pipelined ABV**

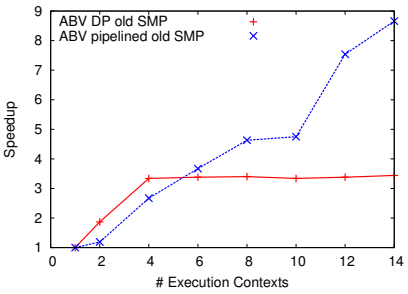


**Figure 6: Pipe. ABV (old arch)**

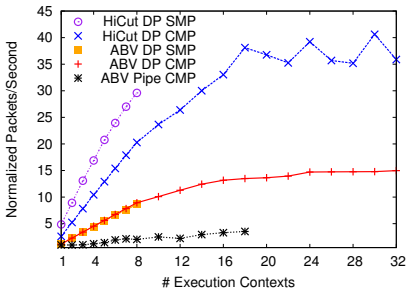


**Figure 7: Relative Speedup**

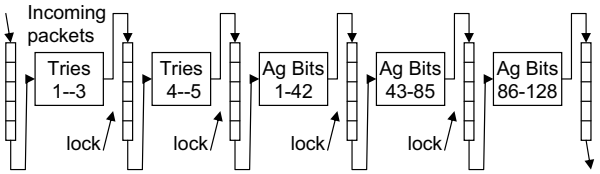


**Figure 1: Pipelined ABV algorithm. Each processor completes a portion of the task and hands off intermediate results to the next processor.**

processors. In principle, this can lead to higher throughput since less time is spent waiting for lock acquisition, although observing such gains requires a balanced workload on each of the processors. In practice, obtaining a balanced workload is highly dependent on many factors, including the parallelized algorithm, the architecture, and data access patterns.

Figure 1 shows a pipelined parallelization of the ABV algorithm. We pipelined the HiCuts algorithm by assigning distinct levels in the decision tree to distinct processors.

## 3. EXPERIMENTAL RESULTS

Our evaluation environment consists of a CMP and an SMP with characteristics as given in Table 2. We parallelized the ABV and HiCuts algorithms as outlined above and measured their performance on synthetic classifiers and traces generated using the techniques described in [1] as well as those that are publicly available [4].

Figures 3 and 4 show the speedup of the data-parallel ABV and HiCuts algorithms, respectively, on both the CMP and SMP. For intermediate thread counts (*i.e.* less than 32), performance increases on the CMP strongly depend on the order in which execution contexts are allocated. A tiled allocation uses all threads in the current core before allocating a thread from another core. On the other hand, a striped allocation balances thread allocation among all cores by allocating threads as evenly as possible among all cores. This implies that scalability of the parallel ABV algorithm is limited by CPU performance saturation rather than communication cost. Thus, data-parallel ABV or HiCuts might scale well on hardware that does not time-multiplex processing cores among different threads.

Figure 5 shows the scalability of the pipelined ABV classifier. The non-monotonically increasing performance suggests imbalance in the tasks of the pipeline combined with architecture-specific "sweet-spots." Overall, the pipelined implementation is much less scalable than the data-parallel approach, and the CMP architecture is better suited to handle the more frequent communication between threads when allocation order is considered. In contrast, Figure 6 shows the results of ABV pipelined when run on a previous-generation SMP architecture (16-processor UltraSPARC-II, shared-bus interconnect with 1 thread per processor). In this architecture, lock contention dominates processing time [3], so that data-parallel scalability is limited, whereas the pipelined implementation continues to scale. While this architecture is aged and engineering constants have changed, the figure underscores the fact that parallel performance is a function of both algorithm choice and target architecture.

Figure 7 depicts the behavior of the most successful algorithms, normalized to the ABV pipelined implementation's single-threaded performance. At the 8-processor mark, the highest-performing configuration we observed was the data-parallel HiCuts on an SMP, although overall results show that the CMP architecture is superior when all execution contexts are considered.

## 4. REFERENCES


[1] F. Baboescu and G. Varghese. Scalable packet classification. In *SIGCOMM '01*, pages 199–210, 2001.

[2] P. Gupta and N. McKeown. Classifying packets with hierarchical intelligent cuttings. *IEEE Micro*, 20(1):34–41, 2000.

[3] J. M. Mellor-Crummey and M. L. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. In *ACM Trans. on Computer Systems. February 1991, pp. 21-65.*, 1991.

[4] H. Song. Evaluation of Packet Classification Algorithms. `http://www.arl.wustl.edu/~hs1/PClassEval.html`, 2006.