

Workload Analysis for Network Processor Design

Ramaswamy Ramaswamy, Ning Weng and Tilman Wolf
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003
{rramaswa,nweng,wolf}@ecs.umass.edu

Abstract

Computer networks have progressed from simple store-and-forward communication networks to more complex systems. Packets are not only forwarded, but also processed on routers in order to implement increasingly complex protocols and applications. Examples for such processing are network address translation (NAT), firewalls, web switches, TCP/IP offloading for high-performance storage servers, and encryption for virtual private networks (VPN).

To handle the increasing functional and performance requirements, router designs have moved away from hard-wired ASIC forwarding engines. Instead, software-programmable network processors (NPs) have been developed in recent years. These NPs are typically single-chip multiprocessors with high-performance I/O components. A network processor is usually located on each input port of a router. Packet processing tasks are performed on the network processor before the packet is passed through the router switching fabric and on to the next network link.

The processing workload on network processors is unique and particularly different from traditional workstation and server workloads, which are dominated by a few large processing tasks. Network processing is entirely limited to a large number of very simple tasks that operate on small chunks of data (i.e., packets). This implies that many results derived from analyzing workstation or server benchmarks (e.g. SPEC or TPC), are not necessarily applicable to the NP domain. A good example is the memory hierarchy, where smaller on-chip memories suffice due to the nature of packet processing.

In order to explore and understand network processing workloads in more detail, we present a tool, called “PacketBench” (a contraction of “packet workbench”). PacketBench provides a programming and simulation environment, where packet processing functions can be implemented easily and quickly. These applications can then be simulated using a variety of real packet traces. The simulation environment is set up to only collect statistics for the packet processing application and not for the supporting PacketBench framework.

We present a number of workload characteristics that can be derived from PacketBench. These can be distinguished into microarchitectural results and network processing results. Most processor simulators provide a range of statistics that are related to the simulated processor core. Examples are instruction mix, branch misprediction rates, and instruction-level parallelism. In the context of network processing there are a number of statistics that can be gathered, which combine microarchitectural metrics (e.g., instruction count and memory bandwidth) with packet metrics (e.g., packet size). This leads to novel metrics that are specific to the network processing environment (e.g., packet processing complexity and packet memory access pattern).

We extend this workload analysis by addressing the issue of how to map network processing tasks to a network processors, which can be seen as heterogeneous multiprocessors. Due to the performance demands on NP systems, not only general-purpose RISC processor cores are used, but also a number of specialized co-processors. It is quite common to find coprocessors for checksum computation, address lookup, hash computation, encryption and authentication, and memory management functions. This leads to network processor architectures with a number of different

processing resources. The trend towards more heterogeneous NP architectures will continue with the advances in CMOS technology as an increasing number of processing resources can be put on an NP chip. This allows for NP architectures with more co-processors; particularly those which implement more specialized, less frequently used functions.

The heterogeneity of NP platforms poses a particularly difficult problem for application development. Current software development environments (SDKs) are already difficult to use and require an in-depth understanding of the hardware architecture of the NP system (something that traditionally has been abstracted by SDKs). Emerging NP systems with a large number of heterogeneous processing resources will make this problem increasingly difficult as the program developer will have to make choices on which hardware units to use for which tasks. Such decisions can have significant impact on the overall performance of the system as poor choices can cause contention on resources. One way to alleviate this problem is to profile and analyze the NP applications and make static or run-time decisions on how to assign processing tasks to a particular NP architecture.

Our approach to this problem is to analyze the run-time characteristics of NP applications and develop an abstract representation of the processing steps and their dependencies. This creates an “annotated acyclic directed graph” (ADAG), which is an architecture-independent representation of an application. The annotations indicate the processing requirements of each block and the strength of the dependency between blocks. The basic idea is that we build the application representation “bottom-up.” We consider each individual data and control dependency between instructions and group them into larger clusters, which make up the ADAG. The ADAG can then be used to determine an optimal allocation of processing blocks to any arbitrary NP architecture. This problem is similar to work in partitioning and mapping of applications for multiprocessors and grid computing. However, the main difference is that we consider a heterogeneous processing platform. Another difference is that network processing applications are very simple and execute a relatively small number of instructions (as compared to workstation applications). This allows us to use much more detailed analysis methods that would be infeasible for large programs. Also, there are a few issues that are specific to the NP domain and usually are not considered for workstation applications. When exploiting parallelism in NP applications, we do not necessarily have to use multiple parallel processors for one packet, but we can also use pipelining.

We present a methodology for automatically identifying processing blocks from a run-time analysis of NP applications. We further cluster this graph to generate an ADAG using an algorithm that is an approximation to the NP-complete optimal solution. To illustrate the behavior and results of the application analysis, we use a set of four realistic network processing applications. The ADAGs for these applications are presented and their suitability for mapping to different network processor platforms is discussed. Finally, we explore the issue of mapping the ADAG onto the network processor platform and present a few preliminary results of this topic, which is the current focus of our research. We believe this work is an important initial step towards automatically analyzing applications and mapping processing tasks to heterogeneous network processor architectures.