EE392C Project Proposal: Parallelizing TCP/IP for a Chip Multiprocessor

Navneet Aron, Shivnath Babu, Sorav Bansal, and Abhyudaya Chodisetti

Stanford University
{navneet,shivnath,sbansal,sravanth}@stanford.edu

1 Introduction

Transmission Control Protocol and Internet Protocol (TCP/IP) are the dominant protocols on today's Internet infrastructure. Most current TCP/IP implementations are designed to run efficiently on single-processor architectures and are therefore single threaded to minimize context switching and interprocess communication overheads. These single threaded implementations do not benefit from the high performance multiprocessor architectures commonly available today. Multiprocessing is essential to handle the emerging Gigabit-per-second networking technologies and also in embedded applications where high-performance microprocessors are too expensive or power hungry to be used.

The focus of our project is to study the flow of processing in TCP/IP and to identify opportunities for parallelism which can provide a high-performance TCP/IP implementation on a chip multiprocessor (CMP). As part of the project we plan to develop a parallel TCP/IP implementation that can be run on a CMP simulator to demonstrate the benefits of our approach. Finally, we hope to find answers to issues such as the appropriate CMP hardware configuration and support for TCP/IP and similar applications.

The proposed project is in the realm of applications for CMPs and it touches upon operating system and programming model issues for CMPs as well. There is a large market today for custom-made processors for TCP/IP processing. Our project can have a significant impact in this area if we can show that a general-purpose CMP can do TCP/IP processing with performance (e.g., throughput and power efficiency) close to that achieved by such custom-made processors.

2 Project Details

2.1 Project Game Plan

There are several TCP/IP implementations which run on a single processor, e.g., *lwIP* [1], *OpenTCP* [2]. Our first step will be to pick one of these implementations for detailed study. We plan to take into account issues such as simplicity, documentation, and support (user community) while making this choice.

The purpose of the detailed code study is to identify various levels of parallelism (TLP, DLP, ILP) present in the code in the light of the fact that we are going to use a CMP for TCP/IP processing. As an illustration, there are several steps in TCP/IP processing where the same code fragment is executed on each arriving

packet, e.g., checksum computation. We need to study the design principles, algorithms, data structures, and implementation choices in the code to extract the available parallelism. Profiling techniques will be employed wherever possible to understand the available parallelism.

The next step is to design and implement an efficient parallel version of this code for a CMP. The straightforward approach would be to split the TCP/IP processing into multiple threads that can execute in parallel and to identify the communication patterns among these threads. We would then need to identify efficient interprocess communication mechanisms for the communication among threads. A more ambitious approach would be to express TCP/IP processing in an appropriate programming model that gives the compiler more flexibility to map the processing onto a CMP. This approach is closely tied to the actual CMP hardware being targeted, e.g., Hydra, Raw, Smart Memories, TRIPS. We will adopt the former approach in the interest of time, but we hope that our work will identify promising directions in the latter approach.

Finally, we will run our parallel TCP/IP code on a CMP simulator. We will perform a series of experiments to evaluate the performance and scalability of our implementation and compare it with the performance of the original sequential implementation.

The following list summarizes our goals in this project. The next two sections outline the metrics that will be used and the expected results.

1. Identify the sources of parallelism in TCP/IP processing.
2. Separate sequential TCP/IP processing into threads of execution, identify the communication patterns among threads, and develop efficient interprocess communication schemes for this communication.
3. Design and implement parallel TCP/IP code.
4. Run parallel code on CMP simulator, obtain performance results, and compare with performance of sequential code.
5. Identify appropriate CMP hardware configuration and support for TCP/IP and similar applications.

2.2 Figures Of Merit

Packet throughput is perhaps the most important metric in TCP/IP processing and we will focus on this metric. Other relevant metrics in this application include packet latency, jitter, load capacity, resilience to attacks like denial of service, etc.

2.3 Expected Results

1. We expect to identify high TLP and DLP (at the packet level) in TCP/IP processing.
2. Identifying threads in TCP/IP processing that can be executed in parallel might turn out to be straightforward. Coming up with the right interprocess communication primitives might be harder.

3. We expect that a parallel TCP/IP implementation will outperform the sequential version significantly.

3 Methodology

3.1 Available Tools

3.1.1 TCP/IP Stack Implementations

1. **lwIP**: A lightweight TCP/IP stack [1]. lwIP can be used either with or without an underlying operating system. The focus of this implementation is to reduce RAM (so that it is suited for embedded systems).

2. **OpenTCP**: Stand-alone TCP/IP stack implementation [2].

3.1.2 CMP Simulators

1. **SimOS**: The SimOS simulator [3] allows us to simulate MIPS and Alpha machines. It has support for simulating SMP machines so by setting memory-access latencies appropriately we can simulate a CMP.

2. **Raw**: The Raw project [5] at MIT provides a a cycle-accurate simulator to simulate the RAW architecture. They have defined a programming model and their simulator supports run-time and debugging tools.

3. **Hydra**: The Hydra papers [4] mention a cycle-accurate, execution-driven simulator for a *thread-level speculation* (TLS) system which could be useful for us. The simulator can accurately model a realistic memory system, including the effects of bus contention and memory access queuing.

3.1.3 Programming Models

1. **Click Software Router**: The Click router [6] project defines a programming model for handling streams of network packets. This model could be useful for us since we could imagine mapping the TCP/IP stack to this model.

2. **TLS Parallelization**: The Hydra project describes a method of parallelizing applications using TLS which mainly aims at loops producing automatic TLS parallel code. But the TCP/IP stack has to be analyzed to decide if this kind of TLS parallelization will be useful for TCP/IP processing.

3.2 Tools to be Developed

We expect to use one of the TCP/IP stack implementations and CMP simulators listed above. We also expect to use some profiling tools and some tools available to test TCP/IP implementations. We do not expect the need to develop any new tools for this project.

3.3 Inputs

TCP/IP benchmarks are available which can be used to benchmark the performance of our implementation [8]. Furthermore, a simple web server implementation (or simulation) can enable us to run web server traces and benchmarks which might be useful since TCP/IP traffic is dominated by HTTP.

3.4 Experiments

Our principal experiments will evaluate the performance of our parallel TCP/IP stack implementation with respect to the base sequential implementation that we used. We will quantify the relative benefits of each of our parallelizing optimizations. We hope to design some experiments to analyze the appropriate CMP hardware configuration and support for TCP/IP processing.

3.5 Work Timeline

We plan to identify the right tools and be able to run a stand-alone unoptimized version of the TCP/IP stack on our tools by the end of April. We expect to spend the next one week on identifying ways to parallelize the code, the subsequent two weeks to implement the parallel version, and the last week of May on experiments.

4 Related Work

The Click Modular Project has built an IP router in software [6]. In the same project, SMP Click maps Click onto an SMP which gives substantial performance gains over the single-processor implementation by exploiting the thread-level parallelism in the router [7]. Although TCP/IP processing is considerably more complicated than IP routing, and SMPs have different architectural characteristics from CMPs, some of the techniques proposed in SMP Click seem relevant to our project.

A parallel TCP/IP implementation is reported in [9]. The paper parallelizes the data dependent part of the processing (e.g., checksum computation) over different packets. The paper claims that more than 80% of the processing can be parallelized over different packets.

Recently released network interface cards have been equipped with hardware support to offload TCP/IP processing from the host [10]. The Raw group has shown how IP routing at Gigabit speeds can be implemented on the Raw CMP [11]. The Hydra CMP Project [4] addresses TLS and parallelization which needs very little programming effort to manually parallelize the applications.

References

[1] LWIP: A Lightweight TCP/IP stack. http://www.sics.se/~adam/lwip.

[2] OpenTCP. http://www.opentcp.org.

[3] SimOS. http://simos.stanford.edu.

[4] The Hydra Project. http://hydra.stanford.edu.

[5] The RAW Microprocessor Project. http://www.cag.lcs.mit.edu/raw.

[6] The Click Modular Router Project. http://www.pdos.lcs.mit.edu/click.

[7] B. Chen, R. Morris. Flexible Control of Parallelism in a Multiprocessor PC Router. In *Proceedings of USENIX 2001*.

[8] Two TCP/IP benchmarks. http://www.upb.de/StaffWeb/jens/Projekte/Benchmarks.

[9] O. G. Koufopavlou, A. N. Tantawy, and M. Zitterbart. Analysis of TCP/IP for High Performance Parallel Implementations. http://www.cs.ucr.edu/~cial/paratcp.

[10] The Alacritech TCP/IP accelerator. http://www.alacritech.com.

[11] G. Chuvpilo, D. Wentzlaff, and S. Amarasinghe. Gigabit IP Routing on Raw. http://www.cag.lcs.mit.edu/commit/papers/02/chuvpilo-HPCA-NP.pdf