HIPPCO: A High Performance Protocol Code Optimizer

Claude Castelluccia, Walid Dabbous

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

HIPPCO: A High Performance Protocol Code Optimizer

Claude Castelluccia, Walid Dabbous

N° 2748

Décembre 1995

PROGRAMME 1

ISSN 0249-6399

HIPPCO: A High Performance Protocol Code Optimizer

Claude Castelluccia, Walid Dabbous

Programme 1 — Architectures parallèles, bases de données, réseaux
et systèmes distribués
Projet RODEO

Rapport de recherche n ° 2748 — Décembre 1995 — 76 pages

Abstract: This report presents HIPPCO, an High Performance Protocol Code Optimizer. HIPPCO belongs to the HIPPARCH compiler. HIPPARCH is a tool which proposes to generate automatically from the application communication requirements and the network characteristics an efficient implementation of a customized protocol.
HIPPCO is the last stage of this protocol compiler. It takes as input a description of the protocol automaton, optimizes it and generates an implementation in *C*.
HIPPCO decomposes the protocol automaton in two parts: the common and uncommon path. It then uses this decomposition to apply a set of optimizations toward a good code speed/code size tradeoff.
In the first part of this report, the code speed optimizations are described. Those optimizations reduces the number of executed instructions and improves the instruction cache and pipeline behaviors. In the second part, a comparison of HIPPCO automatically generated implementations of TCP are compared with the BSD implementation. We show that the HIPPCO generated codes requires up to 70% less instructions than its BSD counterpart.

Key-words: Code optimization, Automated Protocol Generation, ALF

(Résumé : tsvp)

Unité de recherche INRIA Sophia-Antipolis
2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex (France)
Téléphone : (33) 93 65 77 77 – Télécopie : (33) 93 65 77 65

HIPPCO: Un optimiseur haute performance de code de protocole

Résumé : Ce rapport décrit l'architecture de HIPPCO, un optimiseur et générateur d'implantation de protocoles de communication. HIPPCO fait partie du compilateur de protocoles HIPPARCH. Ce compilateur propose de générer automatiquement, à partir des besoins de communication d'une application distribuée et des caractéristiques du réseau sous-jacent, le protocole spécialisé correspondant. HIPPCO est la dernière passe du compilateur HIPPARCH. Il génère à partir de la description de l'automate du protocole, une implantation optimisée en *C*.

HIPPCO décompose l'automate de communication en deux parties distinctes: le chemin fréquent et le chemin rare. Utilisant cette décomposition, un ensemble d'optimisations est appliqué pour obtenir le meilleur compromis entre la vitesse d'exécution et la taille du code.

Ce rapport est structuré en deux parties: la première (sections 3 et 4) détaille les différentes optimisations utilisées pour l'amélioration de la vitesse d'exécution du code. Ces optimisations sont basées sur une réduction du nombre d'instructions à exécuter, sur une meilleure utilisation du cache d'instruction et du pipelining. Dans le seconde partie (section 5), une implantation automatique du protocole TCP est détaillée. Les performances de cette implantation sont comparées avec celles de la version BSD de TCP.

Mots-clé : Optimisation de code, Génération automatique de protocoles, ALF

1 Introduction

The emergence of new distributed computing applications and of new networking technologies is driving the need of more specialized communication protocols. Generating and implementing those new protocols manually in an efficient way is very often a complex and time-consuming task.

One option, that has been extensively studied in the last few years, is to automate or semi-automate the protocol design and implementation phases. This is usually performed by selecting a set of predefined modules, which generally implement the basic protocol mechanisms, and combining them into the final communication system using configuration tools like ADAPTIVE or *x*-kernel [BSS92, NL88, OP91]. This approach has the advantage over the manual approach to generate a modular and configurable system.[1]

However, this approach has not encountered yet the success that was expected. The main reason is that the modularity of those systems introduces a performance penalty on the generated protocol implementation. In fact, those configuration tools very often derive the implementation directly and naively from the specification. As a result, the generated protocol implementation is composed of several modules that communicate and interact via some kind of interfaces. These module interaction mechanisms are usually very costly and are the source of performance penalties that overtake the gain achieved by the configuration.

Another limitation of these configuration tools is that they generally use a bottom-up approach: i.e. they configure a communication system based on the mechanisms provided by current "transport protocol". The granularity of those tools is then very often coarse.

In the HIPPARCH project, we propose to build a protocol compiler which automatically generates the communication system of a distributed application. This approach is interesting only if the generated implementations have good performance. The efficiency of the implementations was therefore one of our main goals. To reach this objective, the HIPPARCH compiler has been built around the following properties:

- **Synchrony:** The HIPPARCH compiler is built around a synchronous language, Esterel. Esterel generates from a set of concurrent modules an inte-

[1] A system is said to be *modular* if it is composed by a number of functional entities called modules which interact together. A system is said to be *configurable* or *flexible* if it is relatively easy to add or remove a functionality without modifying the whole system.

grated automaton. In this automaton, all the specification building blocks are merged together and communicate directly without interfaces.

- **Configurability:** The HIPPARCH compiler selects the protocol mechanisms according to the application requirements and networks characteristics. The HIPPARCH approach is application-led (top-down): the communication requirements of several applications (such as JPEG photo server [C. 95], secure login, multicast video and audio, WWW transactions, multimedia multicast mail delivery, large scale multicast image dissemination [A. 95]) have been studied and analyzed. The compiler should generate the complete communication system for such applications.

- **HIPPCO:** The HIPPARCH compiler features a *code optimizer*, called HIPPCO, which optimizes the structure of the automaton and generates highly optimized code in *C*. HIPPCO uses profiling information provided by the protocol designer to dynamically identify the *common path* and apply a set of optimizations. Those optimizations are specific to the automaton structure and are not redundant with optimizations that may be performed by existing low-level compilers. The goal of HIPPCO is to generate protocol code that can efficiently be compiled and optimized by current *C* compilers. It optimizes a program execution time by reducing the number of instructions on its common path and generating code that exhibits good cache and pipeline behaviors.

In this report, we describe the design of HIPPCO and present some performance results of TCP automatically generated implementations. We show that automatically generated protocol code can be faster than the best optimized hand-coded implementations. This report is composed of 6 main sections. Section 2 introduces the HIPPARCH project and gives some insights on the Esterel language. Section 3 presents HIPPCO concepts and describes its optimization principles. Section 4 details the design of HIPPCO. In this section, we present the various optimizations that are performed by HIPPCO. Section 5 evaluates the performance of HIPPCO automatically generated codes. We compare the instruction counts, i-cache and pipeline performance of some HIPPCO generated TCP implementations with the BSD implementation. We also evaluate the individual effects of each proposed optimization. We, finally, conclude in section 6.

The work presented in this report is the result of the work performed by Claude Castelluccia for his PhD research.

2 Context: The HIPPARCH Protocol Compiler

2.1 General Presentation

The work presented in this report was done in the context of the HIPPARCH project [CCD+94]: an European-Australian collaboration action which proposes to study a novel architecture for communication protocols based on the Application Level Framing (ALF) and Integrated Layer Processing (ILP) concepts [Cla90]. Its final objective is to build a compiler which generates the communication system of a distributed application. This tool uses application-specific knowledge to configure this communication system for improved performance [Cas94a].

The HIPPARCH compiler is composed of 2 tools : the ALF and ILP compiler. The ALF compiler generates from the application requirements and the networks characteristics an automaton representing the control part. This control part contains the application and protocol control parts. The idea of HIPPARCH is to integrate the application and its communication subsystem within a single automaton. This integration removes a lot of interfaces and leads to systems with better performance. In this report, for simplicity reason, we separate the control part of the application from the control part of the communication subsystem. This allows us to generate existing protocols and compare their performance to their manually implemented counterparts. The ILP compiler is a stub compiler, which combines the data manipulations functions in an ILP manner.

The control automaton and the data manipulation functions are combined together to form the complete protocol implementation as illustrated in figure 1. Whenever the application sends a piece of data to a remote correspondent, it calls the protocol output procedures which process the control part of the protocol. The protocol control variables (sequence number, timers, transmission window,...) are then updated and the data manipulation functions are called with the data and some control information as arguments. This control information depends on the generated protocol. It may be composed of some fields of the protocol control block that are used to build the protocol header or to process some of the manipulation functions. After the data manipulation operations (checksum, encryption, marshalling, ...) are performed, packets are built and sent on the network. At the other side, when a packet is received, it is first marshalled, decrypted and checksummed. The processed information (data and control information) is then handed to the control automaton. The protocol control variables will then be updated, a packet possibly sent, and data is delivered to the application.